

UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS - CAMPUS BAURU
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

GABRIEL JUNQUEIRA DO VAL

**IMPLEMENTAÇÃO DE UM SISTEMA DE CAPTURA DE
MOVIMENTOS DE SINAIS DE LIBRAS PARA ANIMAÇÃO DE
AVATARES 3D**

BAURU
Novembro/2024

GABRIEL JUNQUEIRA DO VAL

**IMPLEMENTAÇÃO DE UM SISTEMA DE CAPTURA DE
MOVIMENTOS DE SINAIS DE LIBRAS PARA ANIMAÇÃO DE
AVATARES 3D**

Trabalho de Conclusão de Curso do Curso
de Ciência da Computação da Universidade
Estadual Paulista “Júlio de Mesquita Filho”,
Faculdade de Ciências, Campus Bauru.
Orientador: Prof. Assoc. Antonio Carlos
Sementille

BAURU
Novembro/2024

V135i

Val, Gabriel Junqueira do

Implementação de um sistema de captura de movimentos de sinais de LIBRAS para animação de avatares 3D / Gabriel Junqueira do Val.

-- Bauru, 2024

76 p. : il., fotos

Trabalho de conclusão de curso (Bacharelado - Ciência da Computação) - Universidade Estadual Paulista (UNESP), Faculdade de Ciências, Bauru

Orientador: Antonio Carlos Sementille

1. LIBRAS. 2. MediaPipe. 3. Animação 3D. 4. Avatares Virtuais,. 5. Unity3D. I. Título.

Gabriel Junqueira do Val

IMPLEMENTAÇÃO DE UM SISTEMA DE CAPTURA DE MOVIMENTOS DE SINAIS DE LIBRAS PARA ANIMAÇÃO DE AVATARES 3D

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.

Banca Examinadora

Prof. Assoc. Antonio Carlos Sementille

Orientador

Universidade Estadual Paulista "Júlio de Mesquita Filho"

Faculdade de Ciências

Departamento de Ciência da Computação

Profa. Dra. Simone das Graças Domingues Prado

Universidade Estadual Paulista "Júlio de Mesquita Filho"

Faculdade de Ciências

Departamento de Ciência da Computação

Prof. Assoc. João Eduardo Machado Perea Martins

Universidade Estadual Paulista "Júlio de Mesquita Filho"

Faculdade de Ciências

Departamento de Ciência da Computação

Bauru, 14 de Novembro de 2024.

Este trabalho é dedicado à minha família que me incentivou a vir até aqui.

Agradecimentos

Agradeço a Deus, à minha família e a todos os meus amigos que me acompanharam nessa jornada. Pai e mãe, obrigado sobretudo a vocês, que são a razão da minha vida. Apesar de todos os problemas e desafios, nunca deixaram de me amar e apoiar. A cada dia longe de vocês, aprendi a valorizar ainda mais a importância de tê-los juntos e ao meu lado. Em especial, gostaria de agradecer aos meus queridos padrinhos, Luís Alfredo e Maria Cecília, e à minha tia Dirce, que se fizeram presentes em minha vida durante todos os momentos. Vocês incentivaram esta jornada e hoje repousam ao lado de Deus. Agradeço também a minha avó, todos os meus tios e tias, meus irmãos e meus primos.

Quero agradecer a todos que ajudaram na minha formação como humano e cidadão, e me incentivaram a crescer. Obrigado aos meus amigos, que aqui fiz e aqueles que vieram comigo. Devo muito por todo o apoio, carinho, atenção e por enfrentarem todos os desafios ao meu lado até aqui, e por estarem prontos para os que virão. Agradeço especialmente ao professor Sementille, o melhor orientador que poderia ter, que entendeu meus problemas e medos e sempre ofereceu suporte durante toda a graduação. Juntamente, obrigado Davi, sua dedicação e generosidade fizeram toda a diferença neste trabalho.

"People won't know how you feel unless you tell them."
- Frieren.

Resumo

Este trabalho propõe um sistema que captura e redireciona os movimentos da Língua Brasileira de Sinais (LIBRAS) para animar avatares 3D, usando ferramentas de inteligência artificial e visão computacional. A captura dos gestos é realizada com o MediaPipe, que identifica e rastreia pontos-chave do corpo em vídeos de sinais de LIBRAS. Esses dados, uma vez extraídos, são processados pelo motor Unity3D, que anima avatares com base nas informações de movimento obtidas. O sistema implementa um *pipeline* que envolve: o pré-processamento dos vídeos com os sinais isolados de Libras obtidos do *dataset* V-LIBRASIL, a extração dos posicionamentos das juntas da parte superior do corpo e mãos (com o uso da biblioteca MediaPipe Pose), a transposição dos movimentos capturados para um esqueleto virtual, bem como a renderização final deste esqueleto, por meio da biblioteca Unity3D. O esqueleto virtual gerado, pode, em trabalhos futuros, ter sua topologia adaptada para ser utilizado na animação de avatares humanóides 3D.

Palavras-chave: LIBRAS, MediaPipe, Animação 3D, Avatares Virtuais, Unity3D.

Abstract

This work proposes a system that captures and redirects Brazilian Sign Language (LIBRAS) movements to animate 3D avatars using artificial intelligence and computer vision tools. Gesture capture is performed with MediaPipe, which identifies and tracks body keypoints in videos of LIBRAS signs. Once extracted, these data are processed by the Unity3D engine, which animates avatars based on the captured motion data. The system implements a pipeline that includes preprocessing videos of isolated LIBRAS signs from the V-LIBRASIL dataset, extracting joint positions of the upper body and hands (using the MediaPipe Pose library), transferring the captured movements to a virtual skeleton, and rendering this skeleton with the Unity3D library. The generated virtual skeleton can, in future works, have its topology adapted for animating humanoid 3D avatars.

Keywords: LIBRAS, MediaPipe, 3D Animation, Virtual Avatars, Unity3D.

Lista de figuras

Figura 1 – Exemplos de sinais que utilizam a mesma configuração de mão.	18
Figura 2 – Exemplos de pontos ou local de articulação.	19
Figura 3 – Exemplos de sinais com movimentos ou sem movimentos.	19
Figura 4 – Exemplos de sinais com orientação e direção.	20
Figura 5 – Exemplos de sinais com expressões faciais e corporais.	20
Figura 6 – Sinais do <i>database</i> LIBRAS-HC-RGBDS.	21
Figura 7 – Exemplo de sinal do V-LIBRASIL.	22
Figura 8 – Funcionamento do <i>pipeline</i> do BlazePose.	24
Figura 9 – <i>Landmarks</i> extraídos do MediaPipe Pose.	25
Figura 10 – Continuum de Virtualidade.	25
Figura 11 – Exemplo de hierarquia no Unity3D.	27
Figura 12 – Exemplo de grafo de cena do Unity3D.	28
Figura 13 – Exemplo de uso da janela do <i>Inspector</i>	29
Figura 14 – Processo de <i>pooling</i>	30
Figura 15 – Arquitetura de correção de profundidade.	31
Figura 16 – Estrutura do Sistema.	33
Figura 17 – Fluxograma do programa da subseção 3.2.1.	34
Figura 18 – Estrutura do arquivo JSON.	36
Figura 19 – Fluxograma do programa da subseção 3.2.2.	37
Figura 20 – Grafo de cena do ambiente desenvolvido.	39
Figura 21 – Variáveis na interface do protótipo.	42
Figura 22 – Experimento 1.	43
Figura 23 – Experimento 2.	44
Figura 24 – Experimento 3.	45
Figura 25 – Experimento 4.	45
Figura 26 – Experimento 5.	46
Figura 27 – Experimento 6.	47
Figura 28 – Experimento 7.	47
Figura 29 – Experimento 8.	48
Figura 30 – Experimento 9.	49
Figura 31 – Experimento 10.	50
Figura 32 – Experimento 11.	51
Figura 33 – Experimento 12.	52

Lista de quadros

Quadro 1 – Comparação entre datasets de LIBRAS.	23
Quadro 2 – Comparação entre trabalhos correlatos.	31

Lista de abreviaturas e siglas

2D	Duas Dimensões
3D	Três Dimensões
AM	Aprendizado de Máquina
ASL	<i>American Sign Language</i>
AVI	<i>Audio Video Interleave</i>
CSL	<i>Chinese Sign Language</i>
CNN	<i>Convolutional Neural Network</i>
HD	<i>High Definition</i>
IA	Inteligência Artificial
IBGE	Instituto Brasileiro de Geografia e Estatística
JSON	<i>JavaScript Object Notation</i>
KNN	<i>K-Nearest Neighbors</i>
LIBRAS	Língua Brasileira de Sinais
MP4	<i>Moving Picture Experts Group 4</i>
OMS	Organização Mundial da Saúde
RF	<i>Random Forest</i>
RGB	<i>Red, Green and Blue</i>
RGBD	<i>Red, Green, Blue and Depth</i>
SLERP	<i>Spherical Linear Interpolation</i>

Sumário

1	INTRODUÇÃO	14
1.1	Problemática	15
1.2	Justificativa	16
1.3	Objetivos	16
1.3.1	Objetivo Geral	16
1.3.2	Objetivos Específicos	17
1.4	Organização da Monografia	17
2	FUNDAMENTAÇÃO TEÓRICA	18
2.1	Fundamentos da Língua Brasileira de Sinais	18
2.1.1	<i>Datasets</i> de Sinais Isolados de Libras	21
2.2	Biblioteca MediaPipe	23
2.2.1	MediaPipe Pose	24
2.3	Realidade Virtual	25
2.4	Avatares 3D	26
2.5	Motor de Jogos Unity3D	27
2.6	Trabalhos Correlatos	29
3	METODOLOGIA	32
3.1	Método Proposto	32
3.2	Implementação do Protótipo	33
3.2.1	Etapas de Extração do Esqueleto e Geração do Arquivo JSON	34
3.2.2	Etapas de Redirecionamento do Movimento e Geração do Avatar	37
4	EXPERIMENTOS	41
4.1	Ambiente Experimental	41
4.2	Experimentos	41
4.2.1	<i>Dataset</i> Utilizado	41
4.2.2	Interface do Protótipo	42
4.2.3	Experimentos	42
4.2.3.1	Experimento 1 - Abacaxi	43
4.2.3.2	Experimento 2 - Anteriormente	44
4.2.3.3	Experimento 3 - Bode	44
4.2.3.4	Experimento 4 - Chuva	45
4.2.3.5	Experimento 5 - Documento	46
4.2.3.6	Experimento 6 - Encerrar	46

4.2.3.7	Experimento 7 - Nascer do Sol	47
4.2.3.8	Experimento 8 - Parabéns	48
4.2.3.9	Experimento 9 - Picante	48
4.2.3.10	Experimento 10 - Príncipe	49
4.2.3.11	Experimento 11 - Queda de Energia	50
4.2.3.12	Experimento 12 - Vomitar	51
4.2.4	Considerações Finais	52
5	CONCLUSÕES	53
5.1	Trabalhos Futuros	53
	REFERÊNCIAS	54
	APÊNDICES	58
	APÊNDICE A – CÓDIGO DE PROCESSAMENTO DO MEDIAPIPE E GERAÇÃO DOS ARQUIVOS JSON	59
	APÊNDICE B – CÓDIGO DE ANIMAÇÃO DO ESQUELETO NO UNITY3D	67

1 Introdução

Comunicação é um processo essencial para a vida em sociedade. Sua origem, de acordo com Schramm (1960), vem do latim *communis*, em português, comum. Ainda de acordo com o autor, a comunicação entre indivíduos pode ocorrer de diversas maneiras. Além da forma oral, os gestos também fazem parte da comunicabilidade entre sujeitos, podendo incluir movimentos com as mãos, expressões faciais e até corporais (CRISTIANO, 2017).

Aqueles que possuem algum tipo de deficiência auditiva ou oral têm, em geral, o seu primeiro contato com a comunicação a partir da língua de sinais do local onde vivem.

De acordo com dados da Organização Mundial de Saúde (OMS), cerca de 5% da população global sofre com algum tipo de deficiência auditiva. A perda auditiva pode gerar diversos problemas sociais, tais como, o isolamento social, dificuldade de acesso a informação, educação, mercado de trabalho e serviços públicos de maneira efetiva (World Health Organization, 2023).

No Brasil, tem-se a Língua Brasileira de Sinais (LIBRAS), que é reconhecida como meio legal de comunicação e expressão pela lei Nº10.436, de 24 de abril de 2002 (BRASIL, 2002). A LIBRAS é uma língua própria, não uma interpretação ou tradução do português brasileiro para gestos ou mímicas (CRISTIANO, 2017).

Segundo Capovilla et al. (2017) e Cristiano (2018), existem mais de 14 mil sinais diferentes dentro dessa linguagem, que possui estrutura gramatical própria, com regras de sintaxe e semântica distintas das linguagens orais, caracterizada pela combinação de gestos, movimentos corporais e expressões faciais.

Além disso, segundo Dias et al. (2014), atualmente, no Brasil, existem mais de 10 milhões de pessoas com alto grau de perda auditiva. No que tange à educação, menos da metade da população surda completa o ensino fundamental. Uma pesquisa realizada pelo Instituto Brasileiro de Geografia e Estatística (IBGE) mostra que, considerando a população brasileira, mais da metade dos trabalhadores que possuíam algum tipo de deficiência estavam no ramo informal (GOMES, 2023).

No Brasil, o decreto Nº 5.626, de 22 de dezembro de 2005 estabelece que é obrigatório que professores dos mais diversos níveis de ensino estejam aptos a utilizar LIBRAS (BRASIL, 2005). Esta determinação, porém, não é respeitada, visto que uma grande parcela da população, incluindo os professores, não dispõe dos conhecimentos necessários para se comunicar em LIBRAS (PERES et al., 2006). Na

prática, a dependência de intérpretes é uma realidade para que os alunos surdos possam acompanhar o conteúdo ensinado em sala de aula, ou no contexto profissional, para que os colaboradores possam participar de reuniões e outras atividades em que a comunicação tenha papel crucial.

No entanto, considerando-se o contexto do uso da tecnologia, diversas ferramentas, tais como aplicativos de tradução e recursos de acessibilidade em plataformas digitais têm facilitado a comunicação entre surdos e ouvintes. Particularmente os avanços nos campos da Inteligência Artificial, Aprendizado de Máquina, Aprendizado Profundo e Visão Computacional, têm viabilizado a criação de novos métodos, como a captura de movimentos de sinais para o reconhecimento e a tradução automática da LIBRAS, o que poderia amenizar a dependência de intérpretes e aumentar a inclusão social das pessoas surdas.

1.1 Problemática

A inclusão de pessoas com deficiência auditiva e oral ainda é pequena no âmbito global e como foi descrito pela World Health Organization (2023). A integração destas pessoas como parte válida da comunidade é essencial para que possam se desenvolver socialmente e não sejam marginalizados como frequentemente acontece (GOMES, 2023). Para que esta integração ocorra, novas tecnologias devem ser desenvolvidas e aperfeiçoadas.

O reconhecimento e a tradução automática da LIBRAS apresenta grandes desafios, tais como a disponibilidade de bases de sinais robustas para o treinamento dos modelos de Aprendizado Profundo, bem como, métodos para o rastreamento de características das mãos, corpo e expressões faciais de sequências de vídeo das bases de sinais. Este rastreamento é necessário, não somente para a extração das características para fins de reconhecimento e classificação dos sinais, mas, também, para a estimação da pose do corpo humano visando a animação de avatares 3D em aplicações de tradução para LIBRAS.

No contexto das ferramentas que viabilizam o diálogo entre surdos e ouvintes utilizando LIBRAS, destaca-se o software V-LIBRAS. Ele é uma plataforma de software, disponibilizada pelo governo brasileiro, que realiza a tradução da língua portuguesa para LIBRAS, por meio do uso de avatares virtuais (SILVA et al., 2022). Porém, segundo seus autores, ele possui diversas limitações, tais como a transposição de parte do léxico da língua de sinais para a estrutura morfossintática do português e a movimentação robótica e não fluida dos avatares. Um instrumento que utiliza avatares virtuais deve ter movimentos que pareçam orgânicos, pois o cérebro humano é capaz de identificar, com facilidade, a diferença entre um movimento artificial e um natural (WOLFE et al.,

2022). Cabe ressaltar que o uso de avatares virtuais, em detrimento do uso de imagens reais, traz a vantagem do anonimato, bem como faz com que os usuários se sintam mais confortáveis no uso da ferramenta (LUNA et al., 2023).

1.2 Justificativa

Diante da problemática exposta na seção 1.1, justifica-se o projeto e a implementação de um sistema capaz de realizar a animação de avatares para realização de sinais de LIBRAS, cujos movimentos de mãos, corpo e cabeça, sejam mais fluidos e naturais. Tais avatares animados poderiam, então, ser utilizados como parte de um futuro sistema de tradução da língua portuguesa para LIBRAS. Para conseguir-se a animação dos sinais com alta fidelidade e naturalidade, foi proposto, neste projeto, que os movimentos fossem extraídos diretamente de sequências de vídeos de sinais, provenientes de *datasets* selecionados, e que fossem redirecionados apropriadamente para o esqueleto virtual dos avatares.

Para a estimação dos pontos de referência (*landmarks*)¹ do esqueleto humano a partir de uma sequência de vídeo, utilizou-se a biblioteca MediaPipe da Google, especificamente a Pose (MediaPipe Pose, 2024). Esta biblioteca de código aberto fornece um conjunto de ferramentas e modelos pré-treinados. Ela foi concebida para facilitar o desenvolvimento de aplicações que envolvam a análise de mídia em tempo real, oferecendo blocos de construção modulares e um *pipeline*² de processamento de dados eficiente.

Cabe ressaltar que, neste projeto, foram utilizados apenas os *landmarks* necessários para a determinação do posicionamento de cabeça, ou seja, não foram realizados os mapeamentos de expressões faciais.

1.3 Objetivos

1.3.1 Objetivo Geral

O objetivo principal deste projeto consiste no desenvolvimento de um sistema capaz de animar avatares virtuais 3D de alta fidelidade, a partir de sequências de vídeos de sinais da LIBRAS. O redirecionamento dos movimentos, extraídos da sequência de vídeos dos sinais, se concentrarão na parte superior do corpo, mãos e cabeça. Os *landmarks* extraídos das expressões faciais não serão tratados, estando, portanto, fora

¹ *Landmarks* do corpo, também conhecidos como pontos de referência ou informações do esqueleto, são pontos significativos do corpo humano, tais como juntas das mãos e do corpo, extremidades e estruturas faciais.

² *Pipeline* é uma sequência estruturada de etapas ou processos que transformam ou processam dados de maneira ordenada.

do escopo do projeto. A animação dos avatares poderá ser utilizada, em trabalhos futuros, como parte de um futuro sistema de Realidade Virtual ou Realidade Aumentada para a tradução da LIBRAS para língua portuguesa.

1.3.2 Objetivos Específicos

- Construir um sistema capaz de redirecionar os movimentos de sinais de LIBRAS para avatares humanóides 3D a partir de sequências de vídeos;
- Identificar as fontes de sinais de LIBRAS (*datasets*) mais apropriados para o projeto;
- Coletar os vídeos de sinais isolados;
- Realizar experimentos para validação do sistema implementado, a partir de um conjunto selecionado de sinais dinâmicos, obtidos do *dataset* de sinais de LIBRAS.

1.4 Organização da Monografia

O presente trabalho está estruturado da seguinte forma: no Capítulo 2 tem-se a fundamentação teórica, onde explicado, em detalhes, os conceitos mais importantes para que esse trabalho pudesse se concretizar, como os fundamentos da LIBRAS e estimação dos pontos de referência. Já o Capítulo 3 apresenta os processos utilizados no projeto, apresentando o método proposto para seu desenvolvimento e também a implementação do protótipo. O Capítulo 4, por sua vez, expõe os experimentos realizados, bem como os resultados obtidos. Finalmente, o Capítulo 5, apresenta as conclusões e trabalhos futuros.

2 Fundamentação Teórica

Neste capítulo são apresentados os conceitos considerados mais importantes para este trabalho, tais como: fundamentos da Língua Brasileira de Sinais e principais *datasets* disponíveis, estimação dos pontos de referência (*landmarks*) do esqueleto humano a partir de uma sequência de vídeo e com a Biblioteca MediaPipe, ambientes de Realidade Virtual, avatares humanoides e o motor de jogos Unity3D e, finalmente, os trabalhos correlatos encontrados na literatura.

2.1 Fundamentos da Língua Brasileira de Sinais

Como foi apresentado no Capítulo 1, a Língua Brasileira de Sinais é uma forma válida de comunicação em todo o território nacional. Sendo, portanto, uma língua típica que é constituída por estrutura gramatical própria, além de sintaxe e semânticas que diferem das linguagens orais (CRISTIANO, 2017). Ainda de acordo com o autor, é possível distinguir o canal de comunicação por meio do uso exclusivo da recepção de forma visual e transmissão que utiliza do espaço físico.

Conforme descrito por Capovilla et al. (2017) e Felipe e Monteiro (2007), existem o total de 5 parâmetros normativos da LIBRAS que são utilizados para a formação dos seus mais de 14 mil sinais. São eles:

1. **Configuração de mão:** tem como resultado a posição dos dedos, sendo feito pela mão dominante, ou pelas duas mãos. Existem 64 configurações distintas. Alguns exemplos podem ser observados na Figura 1.

Figura 1 – Exemplos de sinais que utilizam a mesma configuração de mão.

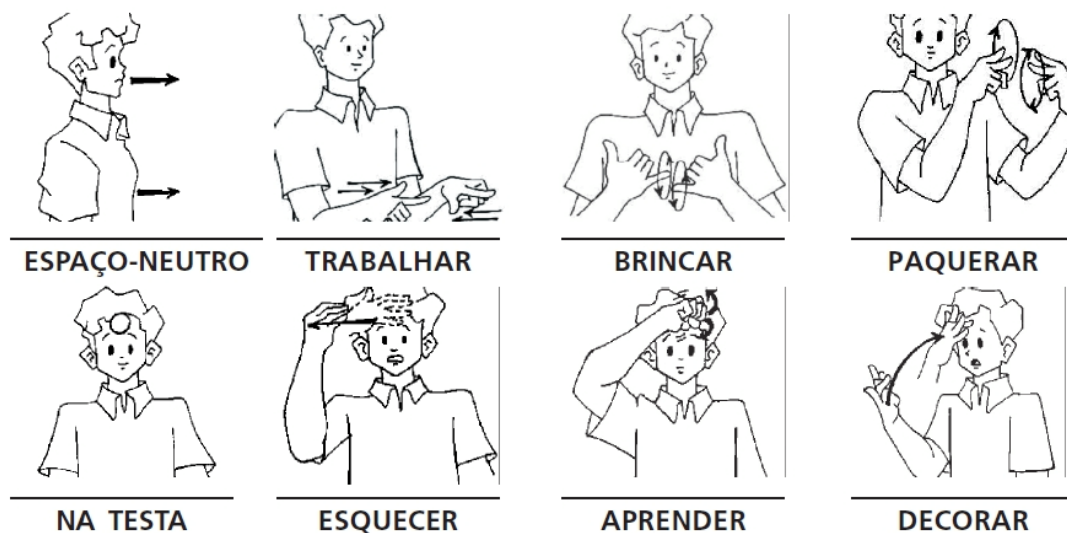


Fonte: Felipe e Monteiro (2007).

2. **Ponto ou local de articulação:** indica o local onde o movimento pode ser realizado, delimitando pelos braços do emissor e pode tocar em alguma parte

do corpo ou não. O tamanho do sinal pode ser comparado a intensidade da voz. Alguns exemplos podem ser observados na Figura 2.

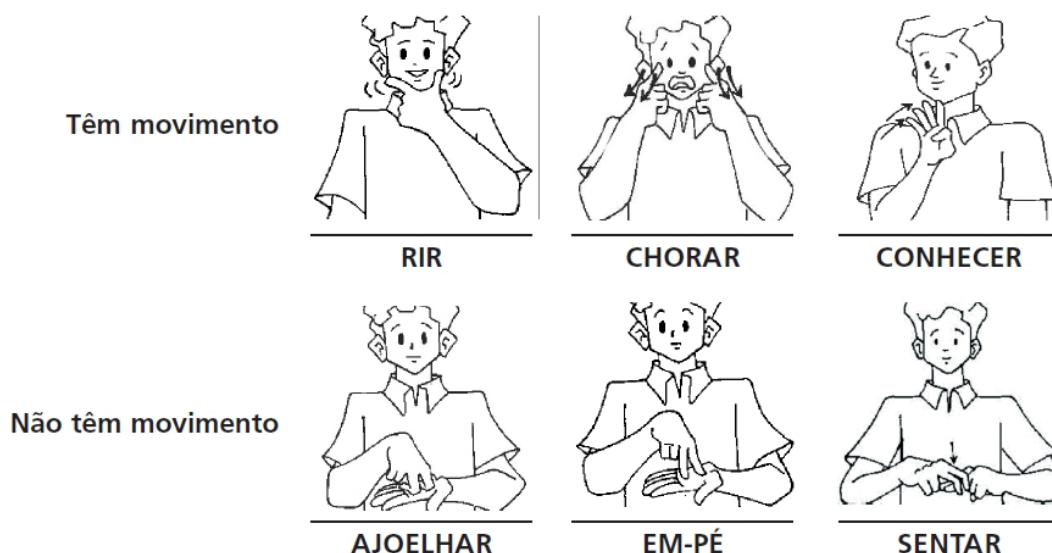
Figura 2 – Exemplos de pontos ou local de articulação.



Fonte: Felipe e Monteiro (2007).

3. **Movimento:** refere-se ao modo com que as mão se movem. Alguns exemplos podem ser observados na Figura 3.

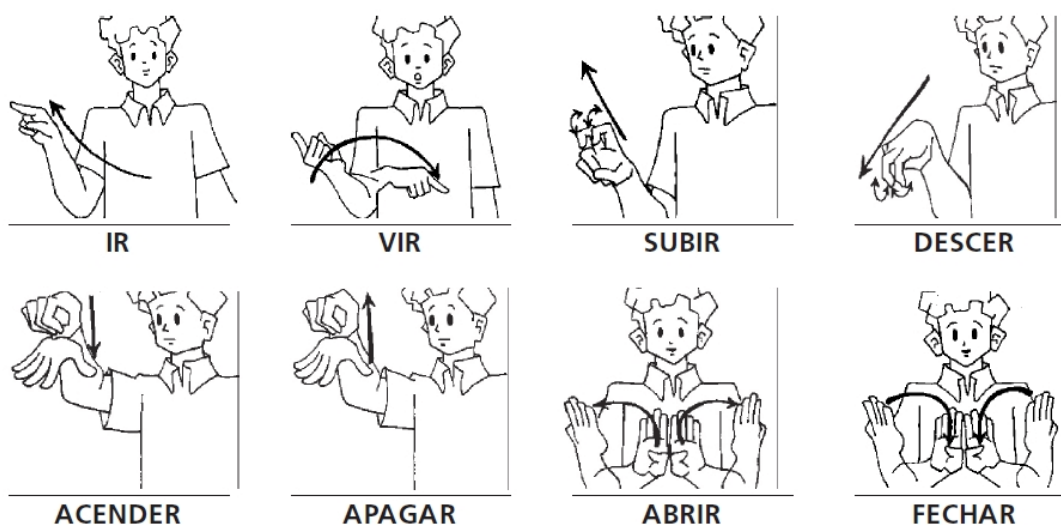
Figura 3 – Exemplos de sinais com movimentos ou sem movimentos.



Fonte: Felipe e Monteiro (2007).

4. **Orientação/direcionalidade:** refere-se ao plano em direção ao qual a a palma da mão é orientado, possibilitando alterar significativamente o significado dos sinais. Alguns exemplos podem ser observados na Figura 4.

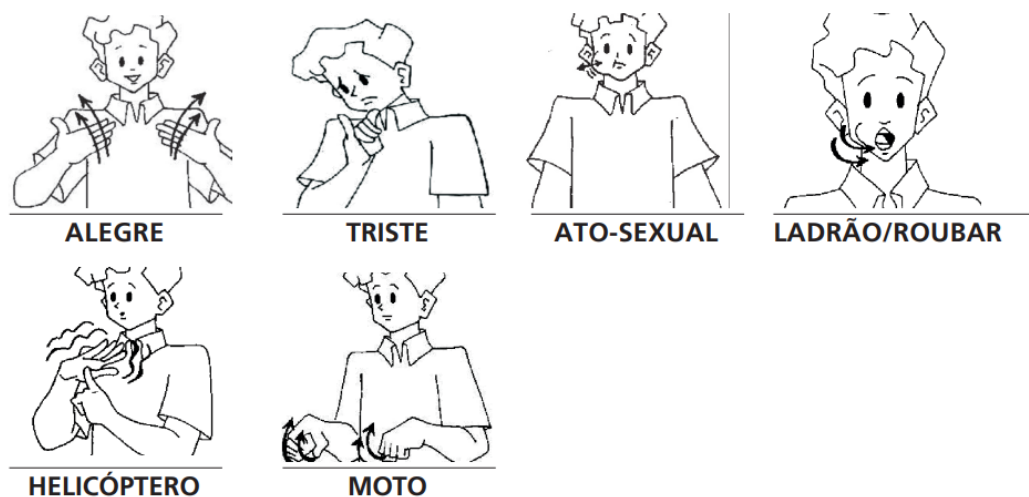
Figura 4 – Exemplos de sinais com orientação e direção.



Fonte: Felipe e Monteiro (2007).

5. **Expressão facial/corporal:** são componentes não manuais, usados em complementos com gestos e sinais para exemplificar uma mensagem e trazer sentimentalismo. Alguns exemplos podem ser observados na Figura 5.

Figura 5 – Exemplos de sinais com expressões faciais e corporais.



Fonte: Felipe e Monteiro (2007).

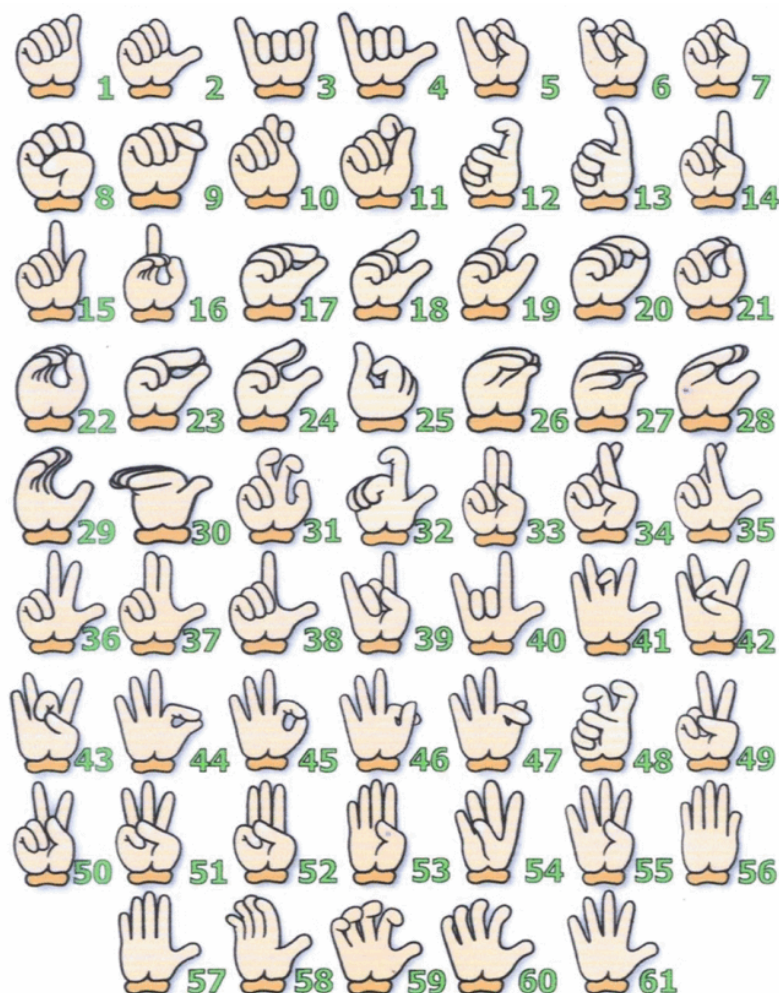
Os fundamentos apresentados anteriormente são essenciais para a comunicação, e provam que a LIBRAS não é apenas uma tradução baseada em gestos da língua portuguesa (CRISTIANO, 2017), mas sim, uma língua própria, complexa e sofisticada, permitindo pensamentos, expressão de sentimentos e a plena comunicação entre partes.

2.1.1 *Datasets* de Sinais Isolados de Libras

As bases de dados que contenham sinais de LIBRAS ainda são escassas, principalmente quando comparadas com bases de dados de línguas de sinais de outros países, como a Língua de Sinais Americana (ASL) ou Língua de Sinais Chinesa (CSL).

Dentre as bases nacionais, uma das primeiras foi a LIBRAS-HC-RGBDS, criada por Porfirio et al. (2013), focou na construção de de uma base centrada nas configurações de mãos, como aquelas exemplificadas pela Figura 1. Em seu desenvolvimento foi utilizado um dispositivo Kinect V1, gravando vídeos RGBD de resolução 640x480 *pixels*, para 61 diferentes classes, com o auxílio de 5 intérpretes. Além do vídeo dos sinais, Porfirio et al. (2013) também criou um esqueleto do intérprete e um modelo 3D com base em seus dados extraídos. Seus sinais podem ser observados na Figura 6.

Figura 6 – Sinais do *database* LIBRAS-HC-RGBDS.



Fonte: Porfirio et al. (2013).

Outro *dataset* que se destaca é o CEFET LIBRAS, apresentado por Gameiro et al. (2020). Essa base possui 547 sequências de vídeos RGB de resolução 640x480

pixels, com a ocultação do rosto de seus 20 intérpretes, permitindo que apenas sinais de mãos sejam rastreados, impossibilitando, por exemplo, o rastreamento de sinais faciais, como aqueles exemplificados na Figura 5. Em relação à seus resultados, o autor implementou a extração de características baseadas em resíduos e depois na classificação usando métodos KNN, onde obteve 53,90% de acurácia e RF, com 65,81% de acurácia.

Além dela, uma outra base de dados é a LIBRAS-UFOP, em que foram catalogados 56 sinais, divididos pelas semelhanças fonológicas entre si, sendo eles: configuração de mão, pontos de articulação ou movimentos (CERNA et al., 2021). Ela também foi gravada utilizando um dispositivo Kinect V1, com câmera RGBD gerando vídeos com resolução de 640x480 *pixels* e com o auxílio de 5 intérpretes. Seus resultados chegaram em até 61,25% de acurácia em testes, utilizando diferentes interpelações de uma CNN.

Já a base de dados do V-LIBRASIL, publicada por Rodrigues (2021), representa uma grande base de sinais quando comparadas às outras, totalizando 1364 palavras e expressões e cada uma sendo apresentada diferentes por 3 diferentes intérpretes, de tal modo a cobrir algumas das variações regionais de cada palavra. Essa base é disponibilizada gratuitamente e em alta qualidade, possuindo uma resolução *FullHD*, de 1920x1080 *pixels*, além do uso de um fundo verde. Na Figura 7 é possível observar um exemplo de sinal deste *database*.

Figura 7 – Exemplo de sinal do V-LIBRASIL.



Fonte: Rodrigues (2021).

Mais uma base de dados é a MINDS-Libras, publicada por Rezende (2021), ela possui 20 sinais de LIBRAS que foram gravados 5 diferentes vezes por cada um de seus 12 intérpretes, criando um total de 1200 amostras. Para que isso fosse possível, uma

câmera profissional foi utilizada, seguida de um Kinect V2 e com auxílio de um fundo verde, criando então vídeos RGB de 1920x1080 *pixels* e vídeos RGBD de 640x480 *pixels* respectivamente. Por último, um *dataset* criado por Sarmento e Ponti (2023) foi consultado. Sua finalidade era de juntar uma grande quantidade de outras bases de dados de alta qualidade e centralizá-las, com isso, bases citadas anteriormente foram agrupadas aqui, como o V-LIBRASIL por exemplo. Totalizando 2098 sinais, com 6017 amostras de expressões de LIBRAS.

Tendo em mente os *datasets* de LIBRAS, foi desenvolvida a Quadro 1 para facilitar a comparação entre os diversos trabalhos.

Quadro 1 – Comparação entre datasets de LIBRAS.

Autor do Dataset	Quantidade de Sinais	Quantidade de Intérpretes	Quantidade de Amostras	Resolução do Vídeo
Porfirio et al. (2013)	61	5	610	640x480
Gameiro et al. (2020)	24	20	547	640x480
Cerna et al. (2021)	56	5	3040	640x480
Rodrigues (2021)	1364	3	4089	1920x1080
Rezende (2021)	20	12	1200	1920x1080 e 640x480
Sarmento e Ponti (2023)	2098	-	6017	1920x1080 e 640x480

Fonte: Elaborada pelo autor.

2.2 Biblioteca MediaPipe

A biblioteca MediaPipe, do Google, é uma solução de *Libraries*¹ que auxiliam na rápida aplicação de Inteligência Artificial (IA) e Aprendizado de Máquina (AM) (Google AI Edge, 2024). Conforme o autor, essas soluções podem ser aplicadas para diversas finalidades, sendo seu principal foco o tratamento de imagens e vídeos, em tempo real ou não, possibilitando, assim, o rastreamento de atividades físicas e a interpretação de língua de sinais.

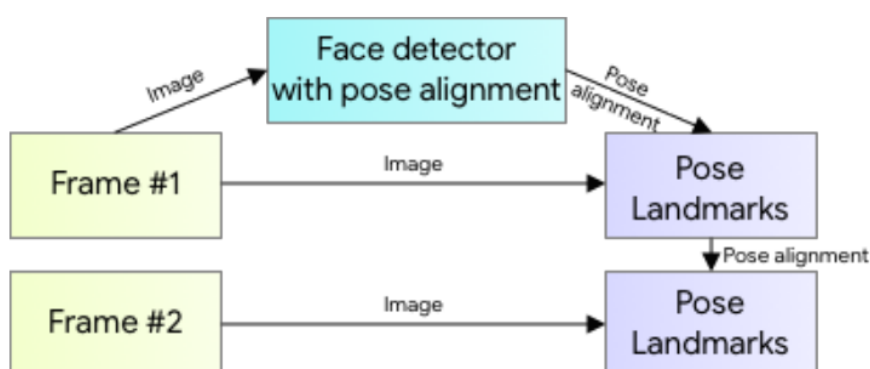
Para realizar o mapeamento dos *landmarks* de um corpo humano, a IA empregada é uma Rede Neural Convolucional (CNN) disponibilizada na biblioteca BlazePose, descrita por Bazarevsky et al. (2020). A biblioteca BlazePose identifica 33 pontos-chave no corpo humano usando uma abordagem híbrida de mapas de calor (ou *heatmaps*) e regressão para prever as coordenadas dos *landmarks*. Durante a etapa de treinamento, são gerados *heatmaps* para cada articulação; contudo, durante a inferência, esses mapas de calor são descartados, e a rede realiza a regressão direta das coordenadas.

O processo de inferência segue uma abordagem de “detecção e rastreamento”, onde um detector de corpo leve é seguido por uma rede rastreadora de pose, capaz

¹ *Libraries*, ou bibliotecas, são coleções de funções e dados auxiliares para o desenvolvimento de *software* de forma modular.

de prever a presença de uma pessoa no quadro e ajustar a região de interesse conforme necessário (BAZAREVSKY et al., 2020). Essa arquitetura “mais leve” possibilita que o BlazePose mantenha uma alta precisão sem sacrificar a velocidade de processamento, sendo capaz de superar métodos tradicionais em termos de desempenho computacional, especialmente em dispositivos móveis. É possível ver mais detalhes do funcionamento de seu *pipeline* na Figura 8.

Figura 8 – Funcionamento do *pipeline* do BlazePose.



Fonte: Bazarevsky et al. (2020).

Além disso, o BlazePose foi projetado para executar em tempo real, com a capacidade de operar em dispositivos móveis. Isso o torna ideal para casos de uso em monitoramento de atividades físicas, reconhecimento de língua de sinais e aplicações em realidade aumentada. Diferente de soluções anteriores, como o OpenPose, que utilizam apenas *heatmaps*, o BlazePose emprega uma abordagem mista de detecção por mapas de calor e regressão de coordenadas.

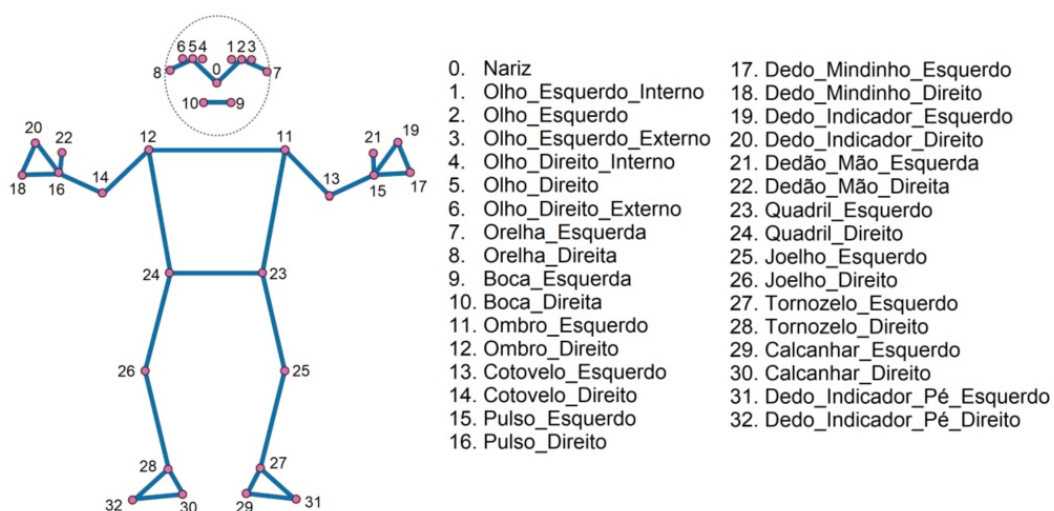
2.2.1 MediaPipe Pose

O MediaPipe Pose, ou também *Pose Landmark Detection*, permite que o corpo humano seja rastreado a partir de pontos de referência, seu *pipeline* é baseado totalmente no BlazePose. Além disso, essa biblioteca permite a análise de postura e a categorização de movimentos (MediaPipe Pose, 2024). Para que isso seja possível, Google AI Edge (2023) explica que é necessário utilizar um *pipeline* de AM de duas etapas, que utiliza um detector e um rastreador. Primeiramente o detector localiza a pessoa que será rastreada no quadro desejado. A partir disso, o rastreador prevê os pontos de interesse, *landmarks*, e os segmenta dentro de uma região de interesse.

Como resultado, 33 pontos serão rastreados e armazenados. Em casos de vídeos, esse *pipeline* só é aplicado no primeiro quadro ou quando o corpo não é mais

identificado na cena. Os pontos que são identificados por padrão podem ser vistos na Figura 9.

Figura 9 – *Landmarks* extraídos do MediaPipe Pose.

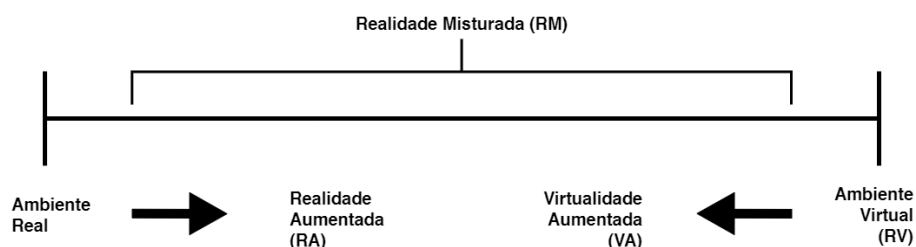


Fonte: Adaptada de Google AI Edge (2023)

2.3 Realidade Virtual

O conceito de Realidade Virtual (RV) origina-se do “continuum de virtualidade”, exposto na Figura 10 e apresentado inicialmente por Milgram et al. (1995). Neste conceito, é possível determinar as diferenças entre os ambientes reais e virtuais. Nota-se que o ambiente virtual, localizado na extremidade direita, está em oposição ao ambiente real. Esse ambiente pode ser definido como um ecossistema em que os participantes e observadores estão completamente imersos em um mundo sintético, o qual pode ou não imitar ou se parecer com o mundo real em que vivemos, conforme descrito pelos autores. Já na extremidade esquerda da Figura 10, é encontrado o mundo real, composto exclusivamente por objetos concretos.

Figura 10 – Continuum de Virtualidade.



Fonte: Adaptada de Milgram et al. (1995).

Ainda embasando este conceito, Zheng, Chan e Gibson (1998) apresentaram que a RV deve obedecer à três regras fundamentais: a fluidez de resposta para o usuário deve ser imediata, os gráficos devem ser renderizados em tempo real, e o sistema deve promover a sensação de imersão. Os autores também mencionam a RV em *desktops*, na qual um teclado e *mouse* são utilizados para o controle do ambiente, enquanto um monitor é empregado para a visualização.

Sistemas como esses, no entanto, falham em proporcionar uma completa sensação de imersão, pois seus usuários não estão inteiramente absorvidos pela RV. A tecnologia que não consegue “enganar” nossos cérebros ao oferecer um ambiente virtual incompleto (ARNALDI; GUITTON; MOREAU, 2018). Isso pode ser resolvido através de interfaces de interação com o ambiente que sejam realistas, incluindo sons, imagens e um retorno imediato das ações. Embora a aplicação dessa tecnologia em uma forma de interação total seja desafiadora, ela pode ser encontrada em diversas áreas e continua em constante desenvolvimento.

2.4 Avatares 3D

O termo avatar é uma palavra originária do Hinduísmo, como uma adaptação da antiga escrita sânscrita. Nela, o avatar é a personificação de um espírito ou divindade que permite a interação com humanos e alterar sua perspectiva para experienciar a vida na Terra (NOWAK; FOX, 2018). Já sua popularização para as mídias do mundo digital se deu pela história de ficção científica *Snow Crash* de Stephenson (1992), onde ele usava o termo para descrever representações no mundo virtual.

Avatar é aceito hoje como a representação de um indivíduo dentro do mundo virtual, Nowak e Fox (2018) dizem ainda que o avatar serve como um símbolo ou uma entidade que um usuário pode escolher para se auto inserir neste universo. Também é necessário que estejam hábeis a interagir, mesmo que de forma não verbal, como gestos, por exemplo. Nem sempre sua forma necessita ser fiel ao mundo real, podendo ser acompanhada de traços exagerados ou impossíveis de serem obtidos.

Apesar disso, os avatares para o contexto de RV devem ser capazes de replicar os movimentos de um indivíduo, permitindo um alto nível de interação e personalização (WANG et al., 2024). Isso se apresenta, ainda como dito pelo autor, como a principal diferença entre os avatares para os personagens 3D. Personagens 3D têm comportamentos e características programadas que devem ser seguidas, sem a possibilidade de interação que reflita qualquer ação, todo comportamento possível para um personagem 3D deve ser predefinido.

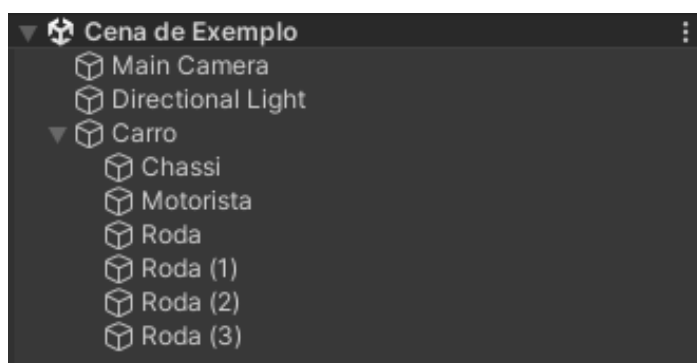
2.5 Motor de Jogos Unity3D

A Unity3D é uma plataforma de desenvolvimento versátil para criação de jogos e aplicações em ambientes 2D e 3D (Unity Technologies, 2022). Com uma ampla gama de recursos, ela permite a personalização tanto do ambiente de desenvolvimento quanto das próprias aplicações. A plataforma também conta com uma loja de *assets*², que possibilita a incorporação de criações de outros desenvolvedores, ampliando as opções de customização e aceleração do desenvolvimento dos projetos. Além disso, conta com total integração para importação de objetos provenientes de *softwares* de modelagem, como o Blender.

A Unity utiliza principalmente a linguagem de programação C#, mas também permite a integração de outras linguagens conforme as necessidades de cada usuário na plataforma. Seu desenvolvimento é multiplataforma, o que facilita a criação de projetos para diferentes sistemas operacionais, abrangendo desde dispositivos *mobile* até sistemas de *desktop* e consoles.

Esse motor de jogos oferece uma estrutura hierárquica robusta para organizar objetos no ambiente de desenvolvimento, facilitando o gerenciamento de elementos dentro de uma cena. Na Unity, cada objeto inserido na cena é chamado de *GameObject*³, podendo ser agrupado em uma hierarquia que reflete as relações de dependência entre eles. Essa estrutura permite que objetos “pai” contenham objetos “filhos”, herdando transformações de posição, rotação e escala que se aplicam aos objetos abaixo. Um exemplo dessa estrutura hierárquica pode ser visto na Figura 11, em que foi criado um carro, com chassi, motorista e rodas.

Figura 11 – Exemplo de hierarquia no Unity3D.



Fonte: Elaborada pelo autor.

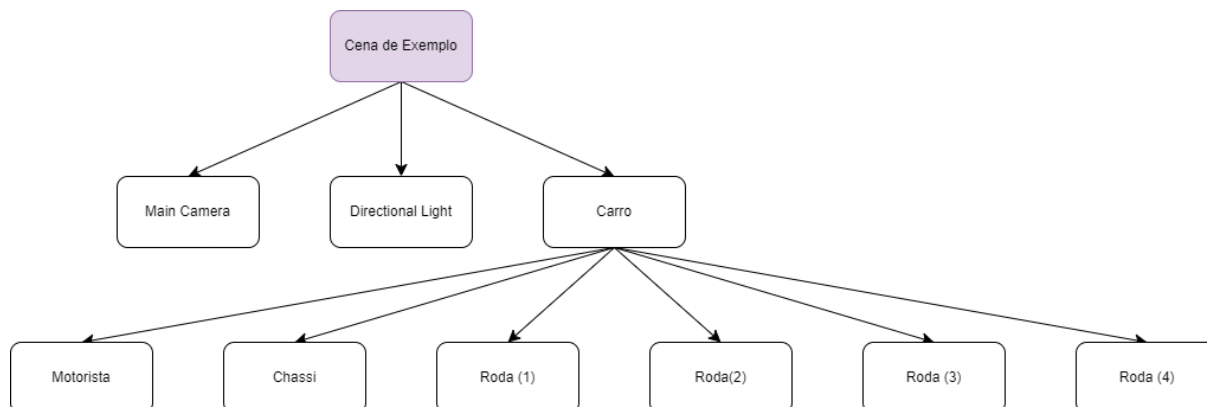
Junto da hierarquia, é recomendado criar um grafo de cena, que permite a

² *Assets* no Unity3D são recursos digitais, como modelos 3D, texturas, sons e scripts, que podem ser usados para construir cenas e funcionalidades em um projeto.

³ Um *GameObject* na Unity3D é a unidade básica de um jogo, representando qualquer elemento na cena, como personagens, objetos interativos e luzes, ao qual componentes podem ser adicionados.

visualização da disposição dos *GameObjects* presentes. Na Figura 12 é possível observar o grafo da estrutura hierárquica da Figura 11.

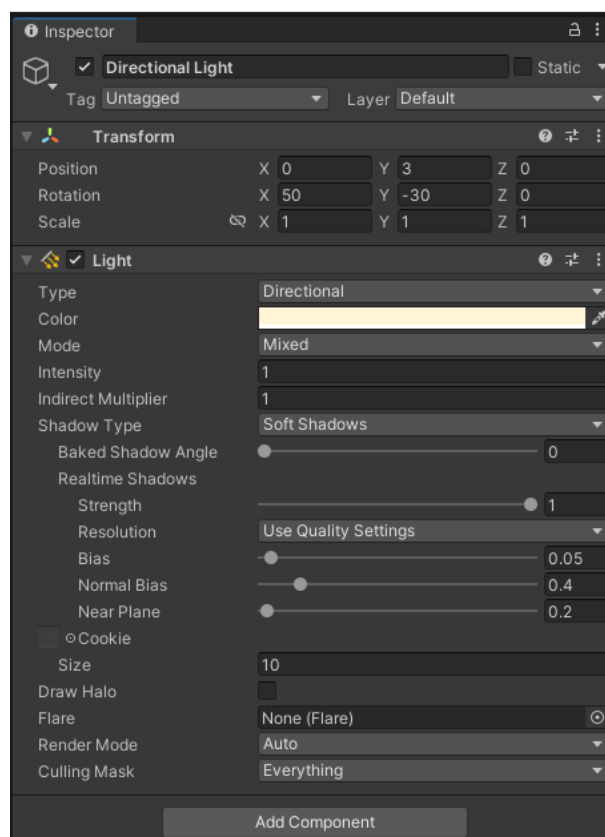
Figura 12 – Exemplo de grafo de cena do Unity3D.



Fonte: Elaborada pelo autor.

O *Inspector* na Unity3D é outra funcionalidade extremamente importante. Ele permite visualizar e editar as propriedades de um *GameObject* ou qualquer outro elemento selecionado na cena. A partir dele é possível ajustar o componente *transform*, responsável por definir o posicionamento, rotação e escala de um objeto no ambiente 3D, baseando-se no sistema de coordenadas do Unity3D. Além disso, o *Inspector* possibilita a configuração de componentes adicionais como *scripts*, que controlam o comportamento do objeto a partir de código, geralmente C#, componentes de física, que simulam interações e colisões, materiais, que definem a aparência visual e outros componentes associados, como pode ser visto na Figura 13 para o *GameObject Directional Light*, que funciona como a iluminação desse ambiente virtual.

Figura 13 – Exemplo de uso da janela do *Inspector*.



Fonte: Elaborada pelo autor.

2.6 Trabalhos Correlatos

Nesta seção tem-se o levantamento dos principais trabalhos do estado da arte associados ao tema.

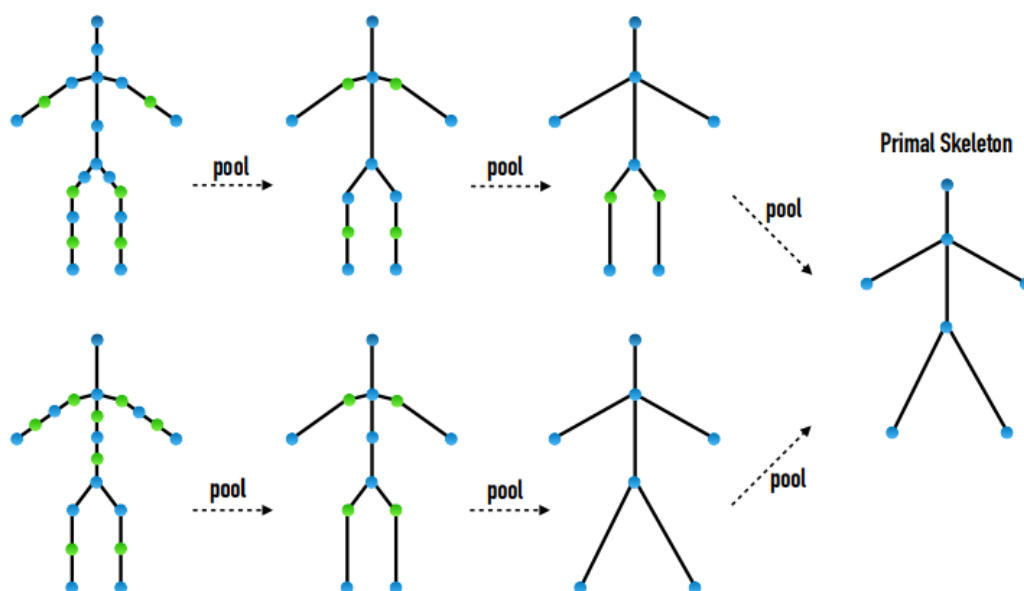
O trabalho apresentado por Aberman et al. (2020) descreve a criação de uma estrutura de uma rede neural para redirecionamento de movimentos capturados em diferentes esqueletos. Chamada *Skeleton-Aware Network*, essa rede neural permite o mapeamento entre esqueletos de diferentes estruturas, desde que topologicamente equivalentes.

Isso só se torna possível porque a hierarquia das juntas dos esqueletos é levada em consideração. Ela utiliza dois componentes, uma representação de movimento profundo e operadores diferenciáveis de convolução e *pooling*⁴, o processo descrito pode ser visto na Figura 14. Com isso, os movimentos são codificados e decodificados de forma eficiente. Seus resultados indicam que o *framework* desenvolvido não apenas

⁴ *Pooling* é uma operação de simplificação topológica por combinação, neste contexto, seriam as juntas dos esqueletos.

melhora a transferência de movimento entre diferentes esqueletos, mas também oferece estabilidade em animações.

Figura 14 – Processo de *pooling*.



Fonte: Aberman et al. (2020).

Já o artigo de Song et al. (2023) apresentou sobre um novo método para o redirecionamento de movimento para avatares com diferentes esqueletos e topologias. Nele, um sistema desenvolveu um esqueleto *proxy*⁵ unificado que serve como uma estrutura para mapear o esqueleto de entrada e finalizar com os avatares animados.

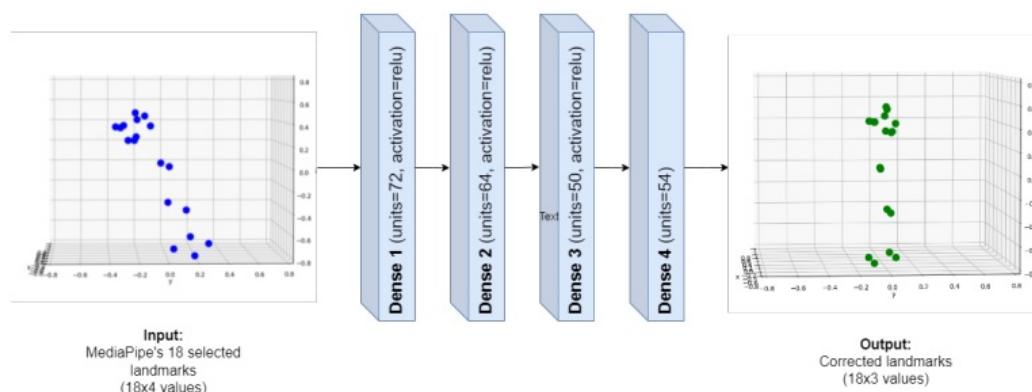
Em seu trabalho, foram apresentados 3 componentes em sua *pipeline*, a captura de movimento usando o BlazePose, a criação do esqueleto *proxy* com base no mapeamento inicial da captura e uma rede para o redirecionamento do avatar final. Para esses avatares, foram utilizados os disponibilizados pela ferramenta Mixamo, onde diversos avatares estão disponíveis para uso gratuito.

Por último, foi analisado o recente trabalho de Poullos et al. (2024), que propôs um método para transferir movimentos de humanos para avatares 3D usando apenas uma câmera RGB. O processo é dividido em três partes principais: inicialmente, a estimativa de pose 3D é realizada usando o algoritmo de detecção de *landmarks* do MediaPipe, que mapeia as coordenadas corporais em um espaço tridimensional. Em seguida, ocorre a etapa de correção de imprecisões, onde problemas como a falta de precisão na profundidade e a ausência de dados de rotação longitudinal dos ossos são tratados.

⁵ *Proxy* é um intermediário entre o produto final e o produto inicial, no caso do artigo, um esqueleto intermediário entre o que foi processado e aquilo que será animado.

Para isso, foram implementadas duas redes neurais profundas: a “*Landmarks Correction Network*”, que corrige as coordenadas 3D das juntas, e a “*Bones Rotation Information Enrichment Network*”, que enriquece o modelo com informações adicionais de rotação dos ossos dos membros. Na Figura 15 é possível observar a arquitetura criada pelo autor para a correção da profundidade.

Figura 15 – Arquitetura de correção de profundidade.



Fonte: Poulios et al. (2024).

Por fim, a transferência dos movimentos ocorre para o avatar no Unity3D. Nessa fase, são utilizados vetores direcionais e *quaternions* para definir as orientações dos ossos do avatar, evitando problemas de deformação e inconsistências no movimento. Esse método mostrou resultados promissores, com uma redução significativa no erro médio por junta e na maioria das imprecisões originais do MediaPipe, como o erro médio de posição das juntas, que caiu de 7,4 cm para 3,9 cm.

Para facilitar a compreensão entre os trabalhos, foi desenvolvida a Quadro 2 para comparação.

Quadro 2 – Comparação entre trabalhos correlatos.

Autor do Trabalho	Método Desenvolvido	Tecnologia Utilizada	Resultados
Aberman et al. (2020)	Redes neurais cientes do esqueleto	<i>Pooling</i> e convoluções	Alta precisão de movimentos
Song et al. (2023)	<i>Proxy</i> para esqueletos	BlazePose e redes de mapeamento	Redução de ruídos e estabilidade
Poulios et al. (2024)	Correção em tempo real	MediaPipe e redes de correção	Redução de imperfeições

Fonte: Elaborada pelo autor.

3 Metodologia

Neste capítulo, será apresentado o método proposto para o desenvolvimento deste projeto, detalhando também o processo de desenvolvimento do *pipeline* do protótipo.

3.1 Método Proposto

O *pipeline* do sistema proposto é mostrado na Figura 16.

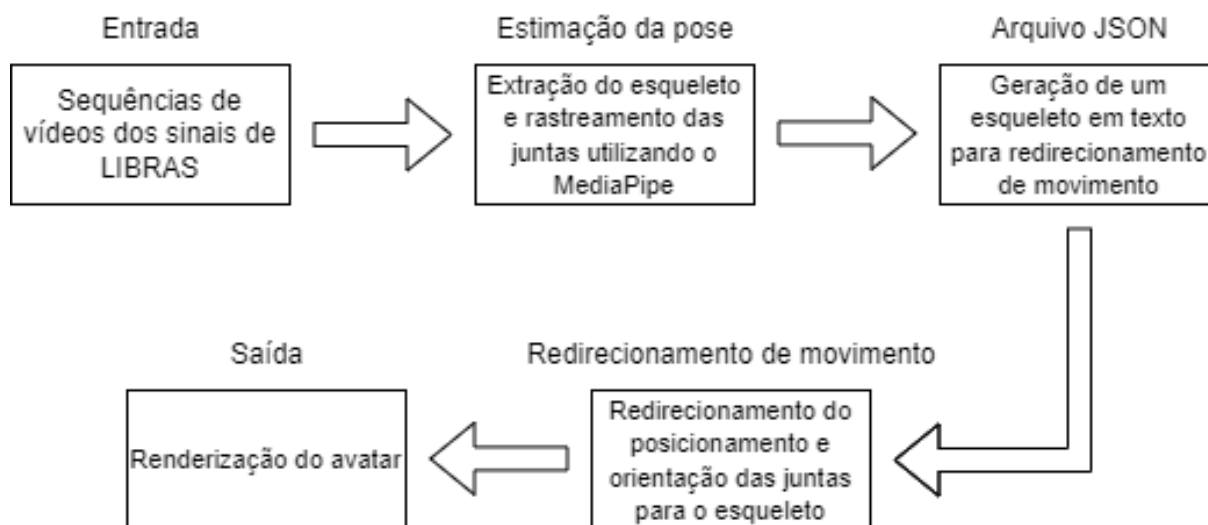
O primeiro bloco indica a entrada do sistema, a qual consiste da sequência de vídeo de determinado sinal da LIBRAS obtido.

O segundo bloco é responsável pela estimação da pose do corpo humano encontrado em cada quadro, realizando a detecção dos *landmarks*, a extração do esqueleto, o rastreamento e conexão das juntas. É utilizado o algoritmo do MediaPipe para realizar esta parte do processo, conforme proposto em Bazarevsky et al. (2020), Amrutha, Prabu e Paulose (2021) e Poullos et al. (2024).

A terceira etapa, representada pelo terceiro bloco, armazena os movimentos do esqueleto obtido da etapa anterior, para um arquivo de movimentos relativos ao sinal de LIBRAS.

O quarto bloco, por sua vez, faz o redirecionamento dos pontos do esqueleto para o esqueleto virtual do avatar no ambiente de Realidade Virtual, mapeando os pontos chaves (*keypoints*) cinemáticos para a animação do avatar. O último bloco ou etapa, realizará a renderização do avatar. Como ambiente de geração e animação do avatar, utilizou-se os serviços do motor de jogos Unity3D.

Figura 16 – Estrutura do Sistema.



Fonte: Elaborada pelo autor.

Como parte da metodologia de desenvolvimento, testou-se o protótipo do sistema implementado selecionando-se alguns sinais dinâmicos de LIBRAS, a partir do *dataset* V-LIBRASIL.

O sistema, inicialmente, faz o redirecionamento dos movimentos da parte superior do corpo, mãos e cabeça, porém, os *landmarks* obtidos do rastreamento das expressões faciais não são considerados.

3.2 Implementação do Protótipo

Para o início do desenvolvimento, selecionou-se uma base de dados apropriada e inseri-la em um programa capaz de processá-la com a biblioteca MediaPipe Pose. Para isso, foi criado um programa em Python com o objetivo de extrair os *landmarks* dos vídeos de sinais em LIBRAS, permitindo o reconhecimento e o mapeamento dos movimentos. A subseção 3.2.1 fornece uma explicação detalhada sobre o processo de desenvolvimento realizado para isso.

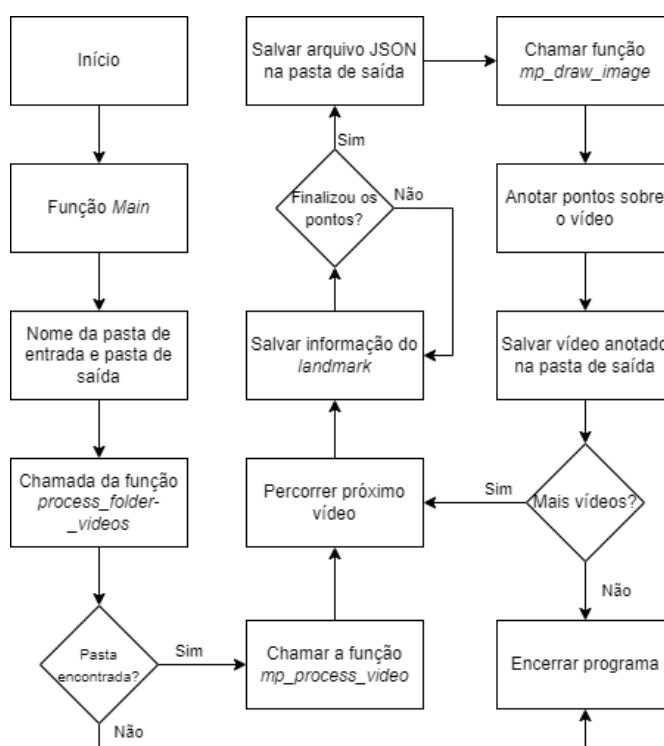
Após essa etapa, tornou-se essencial desenvolver o ambiente dentro do motor de jogos Unity3D para que os testes pudessem ser realizados e o esqueleto fosse renderizado adequadamente. A subseção 3.2.2 ilustra o desenvolvimento dessa parte do projeto.

A íntegra dos códigos desenvolvidos está disponível nos Apêndices A e B.

3.2.1 Etapas de Extração do Esqueleto e Geração do Arquivo JSON

O fluxograma da Figura 17 ilustra o funcionamento geral do código relativo às duas primeiras etapas do método. Ao final da execução do programa, obtém-se uma pasta contendo os vídeos processados com os *landmarks* anotados, um arquivo JSON com as informações detalhadas dos *landmarks* extraídos de cada quadro e o vídeo original permanece disponível.

Figura 17 – Fluxograma do programa da subseção 3.2.1.



Fonte: Elaborada pelo autor.

Para a implementação das duas primeiras etapas do método (vide Figura 16), é essencial realizar as primeiras importações no arquivo de código Python com as bibliotecas que serão utilizadas. A íntegra do código implementado nestas duas primeiras etapas está no Apêndice A. Entre as mais importantes estão *cv2*, *mediapipe* e *json*, que desempenham os seguintes papéis: a *cv2*, derivada da OpenCV, é responsável pelo carregamento e processamento de vídeos, fornecendo funcionalidades para ler, manipular e salvar quadros. A *mediapipe* realiza o rastreamento dos *landmarks* corporais, facilitando a detecção e análise de poses em cada quadro do vídeo e *json* serializa os dados extraídos, salvando as informações de forma estruturada em um arquivo de texto do tipo JSON.

Outras importações também auxiliam em tarefas específicas. Por exemplo, *numpy* permite manipular e copiar dados de imagem. Enquanto as funções *land-*

mark_pb2 e *timestamp_pb2* são usadas para estruturar os pontos e os tempos de cada quadro, garantindo precisão na detecção de poses. A biblioteca *os* tem como objetivo facilitar a manipulação de arquivos por meio do código também.

Em seguida, é criada a função *mp_draw_image*, responsável por desenhar os *landmarks* nos quadros processados pelo MediaPipe. Essa função recebe um quadro de imagem em formato RGB e os dados de *landmarks* gerados pela análise de pose. Utilizando a biblioteca *drawing_utils* do MediaPipe, ela percorre cada ponto-chave detectado e aplica uma sobreposição visual no quadro, conectando os pontos para representar a pose completa.

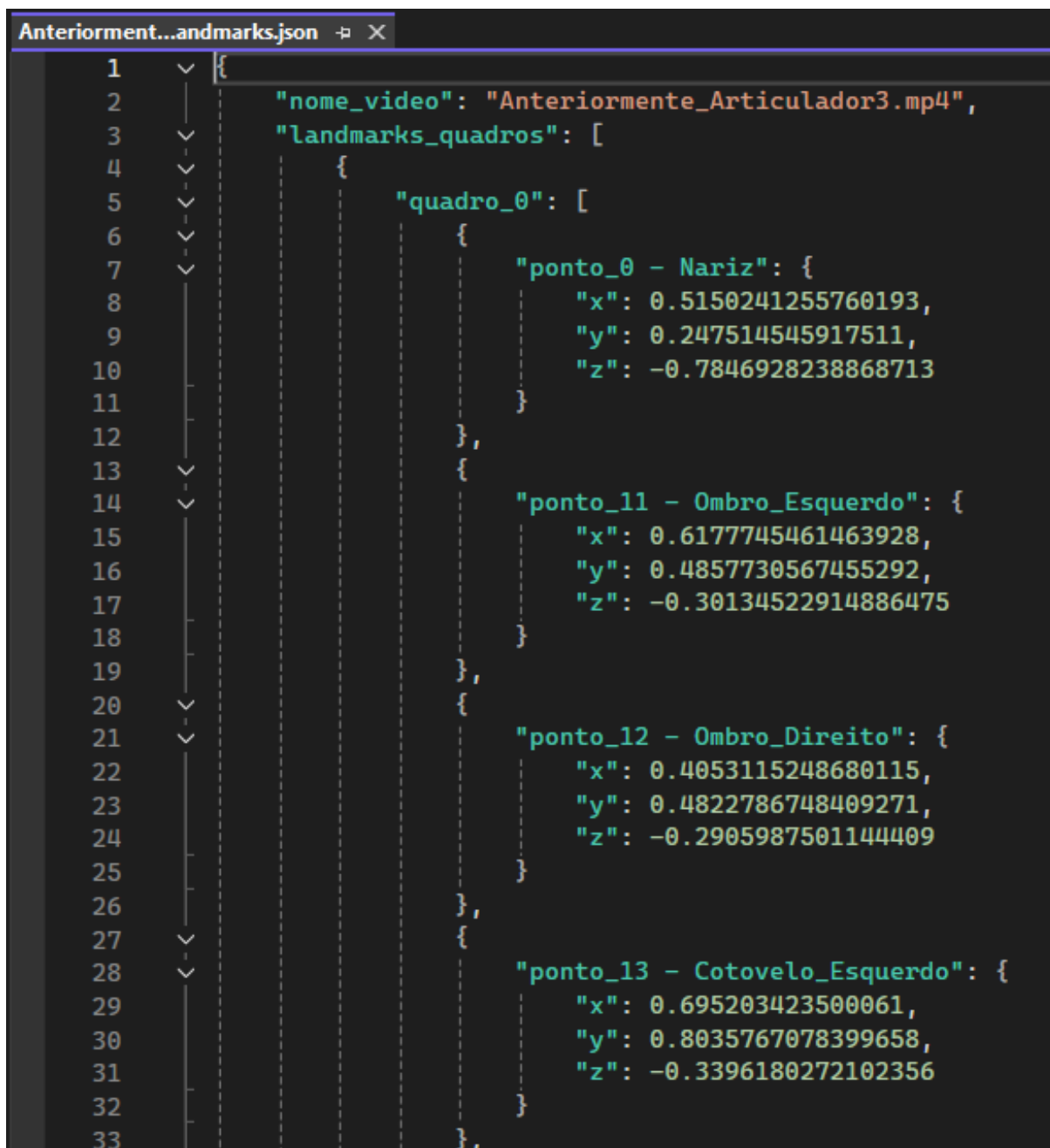
A função *mp_process_video* realiza o processamento detalhado de cada vídeo, permitindo a escolha entre diferentes modelos MediaPipe (*Lite*, *Full*, *Heavy*), ajustando assim o equilíbrio entre precisão e desempenho conforme necessário. Após definir o modelo, a função carrega o vídeo quadro a quadro usando *cv2*, extraindo os *landmarks* em cada quadro através do modelo MediaPipe configurado para rastreamento de pose.

Essa função anterior também chama *mp_draw_image* para aplicar as anotações visuais diretamente nos quadros processados. Além de processar os *landmarks* em cada quadro, *mp_process_video* permite que pontos específicos sejam ignorados, configurando uma lista que exclui *landmarks* irrelevantes. Esses ajustes são especialmente úteis para cenários em que o vídeo captura principalmente a parte superior do corpo, como nos casos de interpretação de sinais em LIBRAS. O processo inclui, por fim, a gravação dos quadros anotados em um novo vídeo e a exportação das informações dos *landmarks* em formato JSON, organizados por quadro para facilitar futuras análises ou aplicações. A estrutura desse arquivo JSON é ilustrado na Figura 18.

Nessa estrutura, após iniciar o arquivo JSON, o nome do vídeo em processamento é registrado, facilitando a identificação de cada conjunto de dados. Em seguida, é criado um vetor contendo todos os quadros do vídeo, sendo que cada quadro inclui um vetor próprio com os *landmarks* capturados. Dentro de cada quadro, os *landmarks* são impressos com suas identificações numéricas, nomes descritivos e coordenadas (x, y, z) no plano do MediaPipe.

A função *process_folder_videos* permite o processamento em lote de vídeos, automatizando a análise de múltiplos arquivos em vez de processá-los individualmente. Ela percorre uma pasta especificada, identificando arquivos de vídeo válidos como *mp4* ou *avi*, e, para cada um, chama *mp_process_video* para realizar a extração e anotação dos *landmarks*. Após o processamento, os dados dos *landmarks* são salvos em arquivos JSON separados, juntamente de seus vídeos processados, organizados no diretório de saída.

Figura 18 – Estrutura do arquivo JSON.



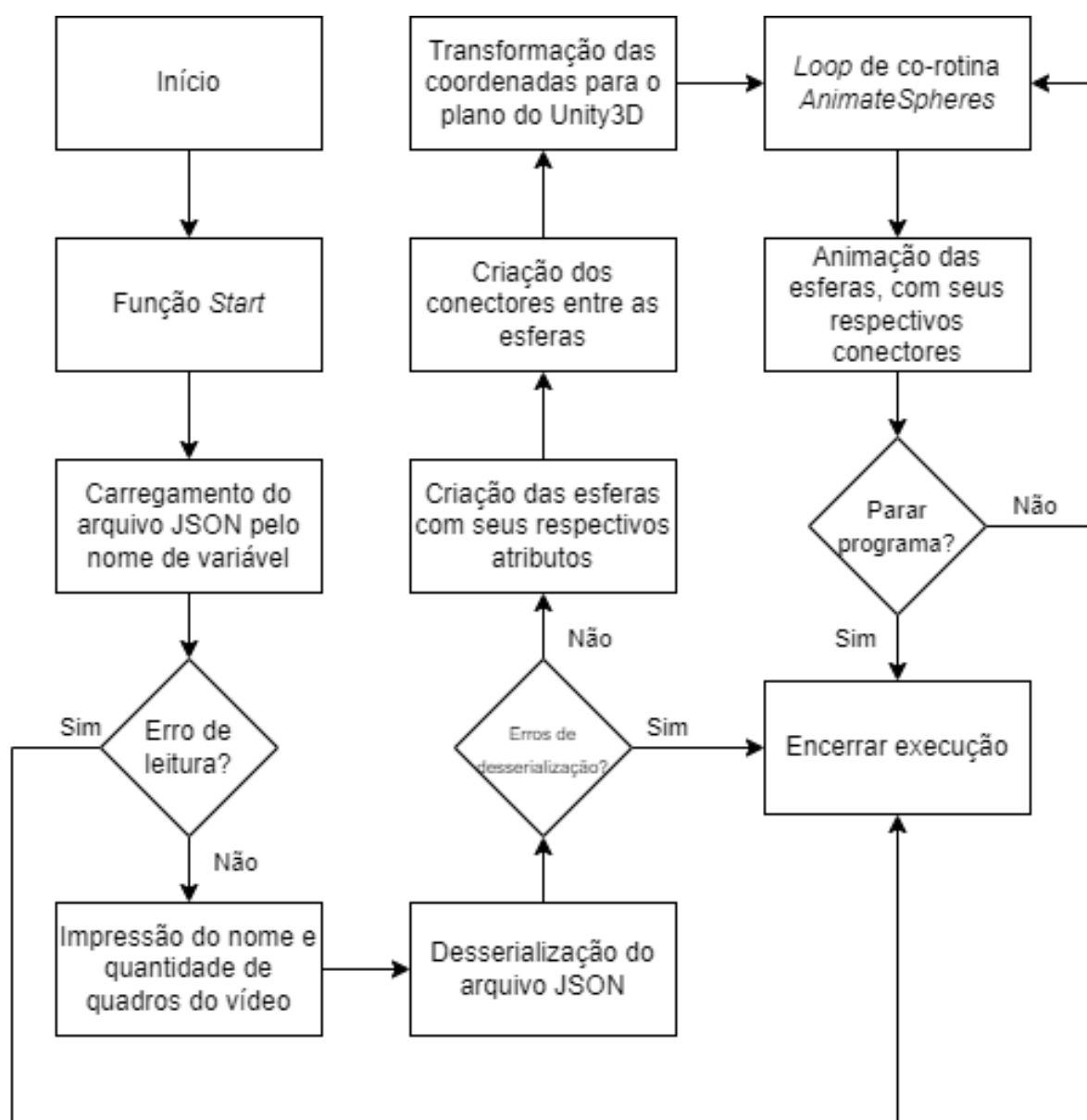
```
1  {
2    "nome_video": "Anteriormente_Articulador3.mp4",
3    "landmarks_quadros": [
4      {
5        "quadro_0": [
6          {
7            "ponto_0 - Nariz": {
8              "x": 0.5150241255760193,
9              "y": 0.247514545917511,
10             "z": -0.7846928238868713
11           }
12         ],
13         {
14           "ponto_11 - Ombro_Esquerdo": {
15             "x": 0.6177745461463928,
16             "y": 0.4857730567455292,
17             "z": -0.30134522914886475
18           }
19         },
20         {
21           "ponto_12 - Ombro_Direito": {
22             "x": 0.4053115248680115,
23             "y": 0.4822786748409271,
24             "z": -0.2905987501144409
25           }
26         },
27         {
28           "ponto_13 - Cotovelo_Esquerdo": {
29             "x": 0.695203423500061,
30             "y": 0.8035767078399658,
31             "z": -0.3396180272102356
32           }
33         }
34       ]
35     }
36   }
```

Fonte: Elaborada pelo autor.

3.2.2 Etapas de Redirecionamento do Movimento e Geração do Avatar

Para a facilitar a visualização, foi desenvolvido um fluxograma que apresenta o funcionamento do programa apresentado no Apêndice B. Ele pode ser visto na Figura 19.

Figura 19 – Fluxograma do programa da subseção 3.2.2.



Fonte: Elaborada pelo autor.

O ambiente no Unity3D foi o responsável tanto pela renderização do avatar quanto pela desserialização dos dados do arquivo JSON de *landmarks* de cada vídeo. Primeiro, com o Unity3D aberto, criou-se um *GameObject* vazio que servirá como controlador principal, onde será anexado o *script* responsável por gerenciar o programa.

O Apêndice B apresenta o código implementado para esta parte do método. A biblioteca System e suas extensões permitem interações com o sistema do computador, facilitando operações gerais, enquanto a UnityEngine e suas variações fornecem funções específicas do Unity, como renderização e controle de objetos. Finalmente, *Newtonsoft.Json* é utilizada para desserializar os dados do arquivo JSON, convertendo-os em objetos manipuláveis no Unity3D.

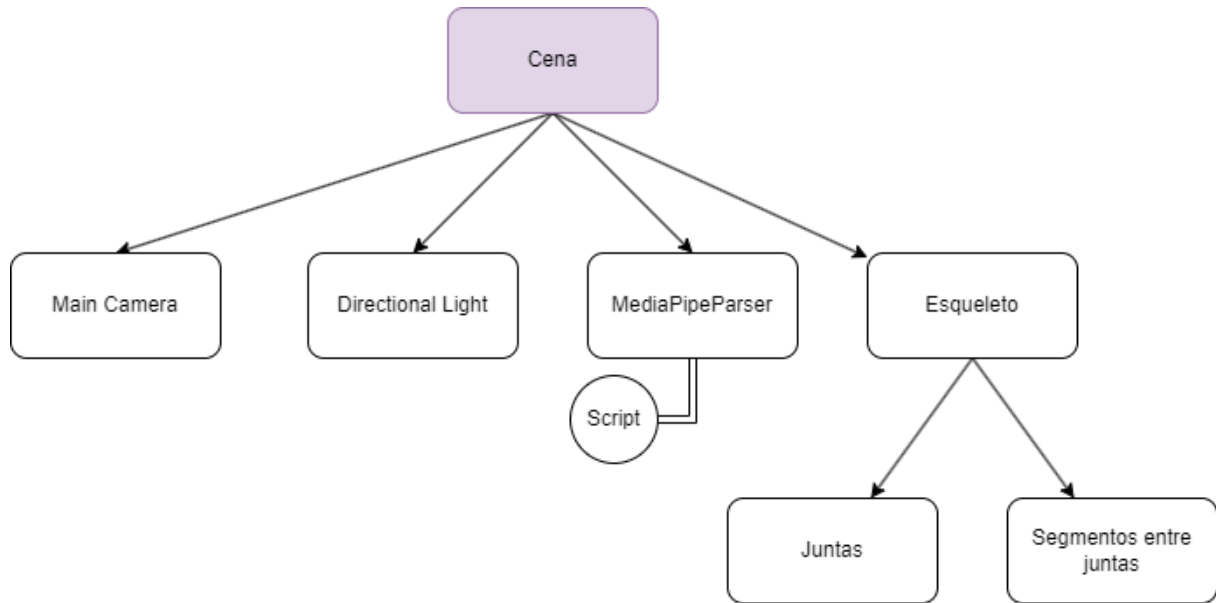
Em seguida, foram desenvolvidas as estruturas e variáveis fundamentais para o programa. Foram criadas classes auxiliares para facilitar a desserialização do arquivo JSON. A classe *RootObject* permite extrair o nome do vídeo processado e armazena uma lista de pontos referentes a cada *frame*. Já a classe *Landmark* contém as coordenadas tridimensionais de cada *landmark*, permitindo que as informações de posição sejam acessíveis e manipuladas.

São declaradas as variáveis e os dicionários que manipulam o esqueleto do avatar no ambiente virtual. Entre as variáveis, destacam-se *avatarScaleFactor*, que ajusta a escala de distância entre os pontos do avatar para corresponder às dimensões desejadas, e *sinalDesejado*, que define o nome do vídeo processado para cada sinal específico.

A lógica do algoritmo começa com a função *Start*, essa função é executada toda vez que se dá início no processo de execução do programa, onde o arquivo JSON é carregado e desserializado para possibilitar a leitura dos dados necessários. Nesta etapa, o nome do vídeo e a quantidade de *frames* são exibidos, fornecendo uma visão geral dos dados processados. Em seguida, dois *GameObjects* vazios são criados para estruturar a hierarquia do esqueleto do avatar: um deles armazena as esferas que representam os *landmarks*, e o outro, os cilindros que conectam os *landmarks*, formando a estrutura visual do esqueleto.

Inicia-se a criação das esferas que representarão cada *landmark*. A função percorre todos os pontos de um *frame*, nomeando cada esfera com base no dicionário *pointNames*, ajustando sua escala conforme *sphereScaleMap* e aplicando cores definidas em *sphereColorMap*. Em seguida, as esferas são inseridas na hierarquia de forma estruturada. O grafo de cena do ambiente que foi desenvolvido nessa seção pode ser visto na Figura 20.

Figura 20 – Grafo de cena do ambiente desenvolvido.



Fonte: Elaborada pelo autor.

Após isso, são inicializados os cilindros que conectam as esferas. Utilizando o dicionário de conexões, cada cilindro recebe um nome que identifica as duas esferas conectadas no ambiente virtual. Além disso, a lógica hierárquica é aplicada junto de sua cor.

Um novo *loop* é criado, agora para iniciar a leitura das posições no primeiro *frame* do arquivo JSON. Nessa parte, também foi onde foi necessário fazer uma correção em relação às coordenadas do MediaPipe quando passadas para o ambiente do Unity3D. Essas correções se resumiram em inverter o eixo y e multiplicar o eixo z para que a profundidade fosse corrigida.

Foram calculadas, também, as mudanças necessárias de coordenadas para que o avatar fosse renderizado no centro do plano de coordenadas do ambiente virtual do motor de jogos.

A co-rotina de animação será executada em todo *frame*, e com isso, fará com que o esqueleto do avatar “ganhe” movimento. Para isso ela muda a posição das esferas por cada *frame* em que elas estão registradas. Após atualizar a posição das esferas, ele precisa atualizar a posição também dos cilindros que fazem suas conexões. Para isso, ele utiliza o cálculo da distância entre duas esferas, aplicando metade do tamanho da menor esfera como valor de seu raio. A fórmula matemática para chegar nessa solução é apresentada na Equação 3.1, onde d representa a distância.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \quad (3.1)$$

Também é importante calcular a rotação dos cilindros, para entender melhor os sinais. Para isso foi calculado com base na função do Unity3D para cálculo de *quaternions*¹, utilizando seu ponto final, subtraído de seu ponto inicial.

¹ *Quaternions* são números complexos de quatro dimensões usados na matemática e na computação gráfica para representar rotações tridimensionais de forma mais eficiente.

4 Experimentos

O capítulo 4 tem como objetivo expor o ambiente em que os experimentos foram realizados, além de apresentar detalhes de cada etapa. Descrevendo assim, o *dataset*, os experimentos em si, a interface utilizada, bem como as considerações finais.

4.1 Ambiente Experimental

Os componentes de *software* e *hardware* necessários para a execução deste projeto são os seguintes:

1) *Software*

- Sistema Operacional: Windows 10 64bit (*desktop*);
- Ambiente de Desenvolvimento: Microsoft Visual Studio; Motor de Jogos Unity3D;
- Biblioteca de recursos: Unity 3D Store;
- *Framework* de Aprendizado de Máquina: Google MediaPipe Pose;
- Biblioteca de visão computacional: OpenCV.

2) *Hardware*

- Notebook pessoal (Acer Aspire A315-23G); Processador: AMD Ryzen 7-3700U; GPU: AMD Radeon RX Vega 10; Memória RAM: 12 GBytes DDR4; Armazenamento: 512 GBytes SSD.

4.2 Experimentos

Os experimentos foram realizados todos dentro do ambiente do Unity3D, já com o *dataset* processado e os vídeos e arquivos JSON inseridos no projeto.

4.2.1 *Dataset* Utilizado

Para os testes, utilizou-se o *dataset* V-LIBRASIL de Rodrigues (2021), escolhido por sua vasta quantidade de sinais em LIBRAS e pela presença de múltiplos intérpretes para cada sinal, proporcionando visões variadas e diferentes formas de execução de cada movimento. Outro aspecto importante para sua escolha foi o acesso gratuito e aberto por meio de um repositório na *internet*. Além disso, os vídeos de alta resolução permitem uma maior precisão nas inferências da IA MediaPipe Pose,

4.2.2 Interface do Protótipo

Como citado no Capítulo 3, o ambiente de desenvolvimento utilizado para os testes foi configurado no Unity3D. Primeiramente, organizou-se o ambiente de forma a garantir fácil acesso e manipulação de *scripts*, *GameObjects* e *Prefabs*. Além disso, criou-se um *GameObject* vazio chamado MediaPipeParser, que contém o *script* MediaPipeJSONParser como seu componente principal.

Este *script* pode ser consultado integralmente no Apêndice B. Graças à sua modularidade, ele permite a fácil modificação de variáveis, como os *prefabs* utilizados para as esferas e cilindros. A janela do *inspector* também facilita o ajuste da escala de distância entre as esferas que representam os *landmarks*. Além disso, é possível alterar rapidamente o sinal desejado, bastando inserir o nome do arquivo JSON correspondente ao sinal de LIBRAS desejado, como pode ser visto na Figura 21.

Figura 21 – Variáveis na interface do protótipo.



Fonte: Elaborada pelo autor.

Uma vez que o *dataset* completo for processado, ele pode ser totalmente integrado ao projeto Unity3D. Isso permite o acesso a todos os sinais de LIBRAS, abrangendo os diferentes trabalhos apresentados na subseção 2.1.1.

4.2.3 Experimentos

Para os experimentos, foram selecionados 12 sinais diferentes de LIBRAS do *dataset* V-LIBRASIL. Na escolha dos sinais, priorizou-se aqueles com gestos que enfatizassem o movimento dos braços, minimizando a necessidade de configurações detalhadas dos dedos e também sinais que não dependessem de expressões faciais. Essa seleção foi feita considerando que o MediaPipe Pose possui limitações no rastreamento das mãos, capturando apenas os pontos do pulso, dedão, dedo indicador e dedo mindinho em cada mão.

Também foram desconsiderados os movimentos e expressões faciais nos experimentos, pois os pontos de rastreamento do rosto, exceto o do nariz, não foram

incluídos na renderização. O ponto do nariz foi mantido como uma referência central para o rosto, facilitando a orientação do avatar no ambiente virtual.

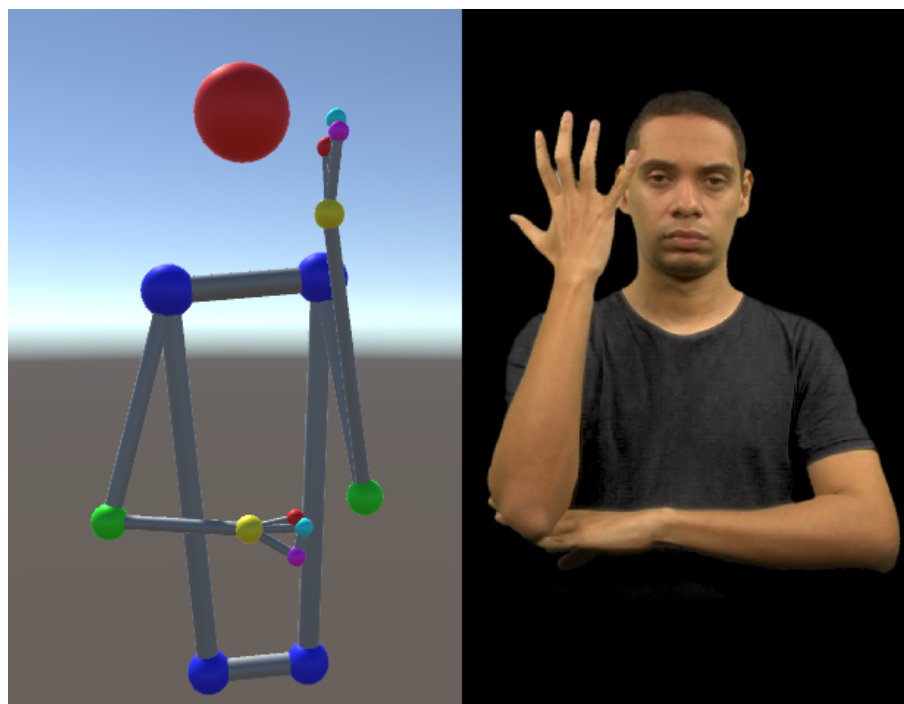
Em todos os experimentos, o esqueleto irá começar de uma posição neutra e realizar o sinal, voltando em seguida para a posição neutra. O movimento é executado em *loop*, até que a execução do programa se encerre.

Além disso, observou-se que o esqueleto do avatar está invertido em relação ao intérprete original. Essa inversão ocorre porque foi necessário inverter eixo “y” das coordenadas para que os dados do MediaPipe fossem compatíveis com o sistema de coordenadas do Unity3D, como descrito na subseção 3.2.2.

4.2.3.1 Experimento 1 - Abacaxi

Para o primeiro experimento, o sinal que representa a palavra “Abacaxi” foi escolhido. Esse movimento começa com uma configuração de mão neutra, onde a mão dominante se posiciona próxima ao rosto. A mão realiza um movimento de rotação, sugerindo a textura da casca do abacaxi, enquanto a palma está orientada para dentro, voltada em direção ao rosto. Na Figura 22 é possível observar o esqueleto renderizado pelo programa ao lado do intérprete.

Figura 22 – Experimento 1.

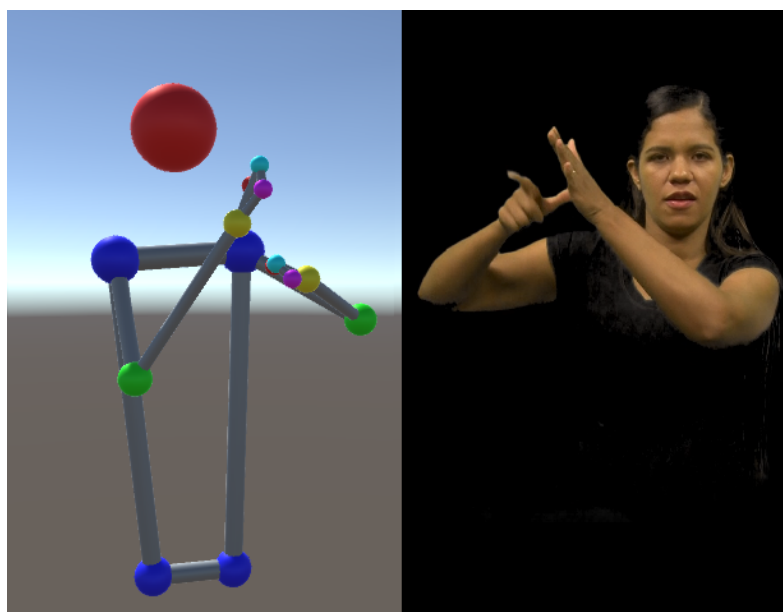


Fonte: Elaborada pelo autor.

4.2.3.2 Experimento 2 - Anteriormente

Para o segundo experimento, o sinal “Anteriormente” foi escolhido. Esse movimento começa com uma configuração de mão neutra, posicionada próximo ao corpo. O ponto de articulação é o pulso da mão dominante, e a mão faz um movimento leve para trás com o dedo indicador apontado enquanto o dedão está encostado na outra mão, que se mantém aberta. O esqueleto virtual, gerado pelo programa, é exibido na Figura 23 junto ao intérprete.

Figura 23 – Experimento 2.

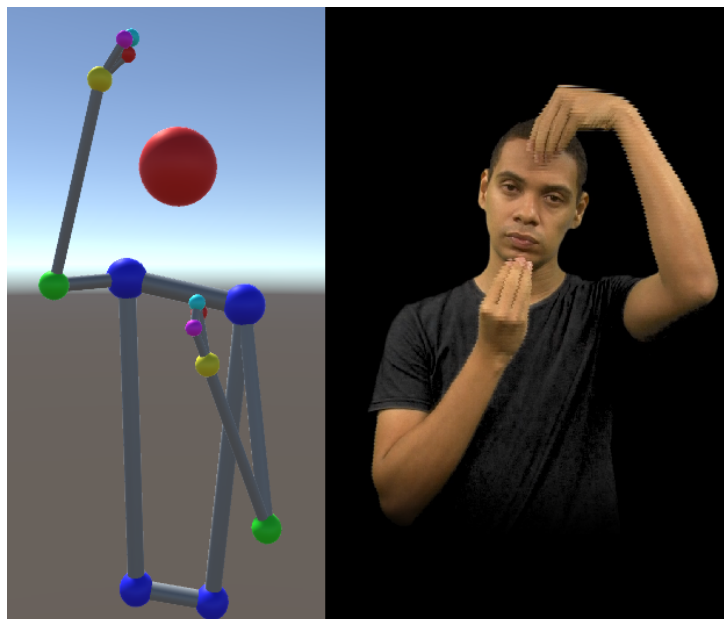


Fonte: Elaborada pelo autor.

4.2.3.3 Experimento 3 - Bode

No terceiro experimento, o sinal “Bode” é representado. O movimento inicia com a mão posicionada ao lado do rosto. O ponto de articulação é na testa e no queixo, com um gesto que simula as características de um bode. A palma da mão está voltada para o rosto. Conforme mostrado na Figura 24, o esqueleto é exibido ao lado do intérprete.

Figura 24 – Experimento 3.

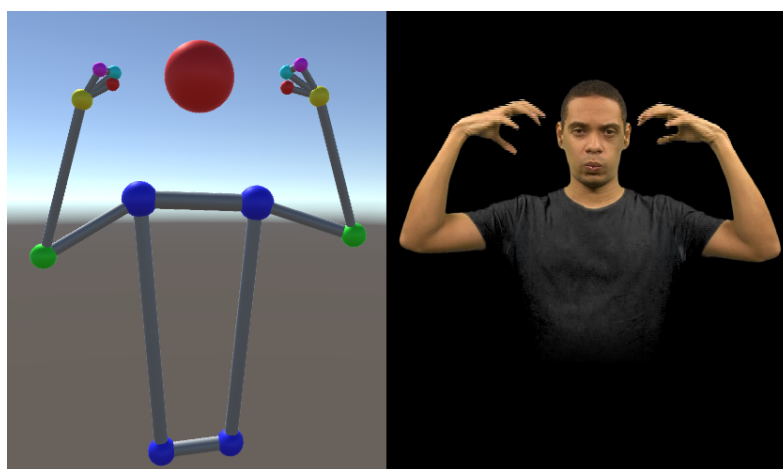


Fonte: Elaborada pelo autor.

4.2.3.4 Experimento 4 - Chuva

Para o quarto experimento, o sinal “Chuva” é executado com as mãos abertas, posicionadas ao lado da cabeça. O ponto de articulação são os pulsos, e o movimento envolve uma descida alternada das mãos, imitando o cair da chuva. As palmas estão voltadas para baixo, representando gotas caindo. Na Figura 25, o esqueleto renderizado está posicionado ao lado do intérprete.

Figura 25 – Experimento 4.

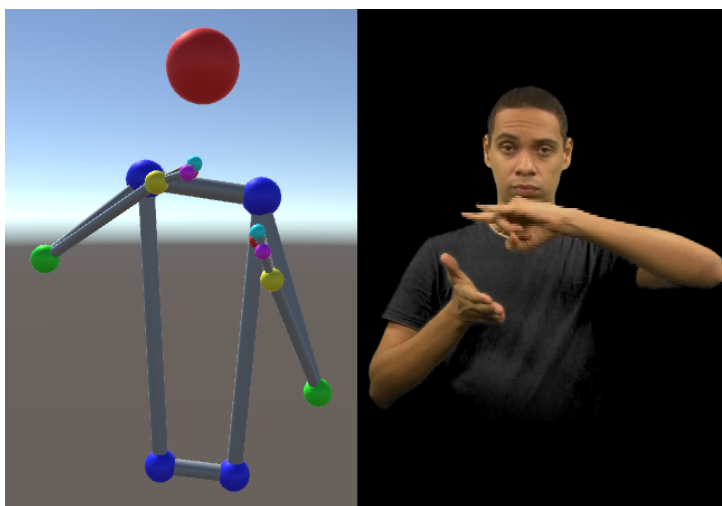


Fonte: Elaborada pelo autor.

4.2.3.5 Experimento 5 - Documento

No quinto experimento, o sinal “Documento” usa uma configuração em que sua mão dominante fica aberta. O ponto de articulação é o cotovelo do outro braço, com a mão fazendo um sinal de em que o dedo indicador e o dedo mindinho apontam para a palma da outra mão. Na Figura 26 é possível observar o esqueleto renderizado pelo programa ao lado do intérprete.

Figura 26 – Experimento 5.

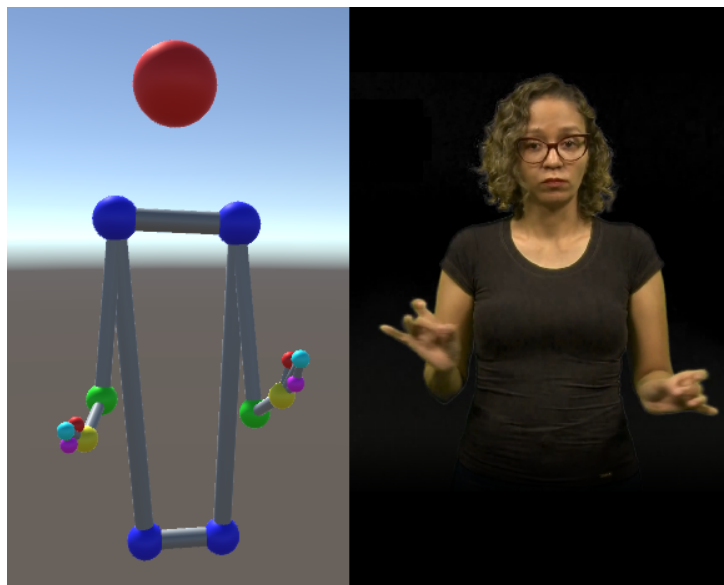


Fonte: Elaborada pelo autor.

4.2.3.6 Experimento 6 - Encerrar

Para o sinal “Encerrar”, o movimento começa com ambas as mãos em formato de pinça e abrindo rapidamente, enquanto se cruzam. O ponto de articulação está nos cotovelos. A Figura 27 demonstra o esqueleto animado pelo programa posicionado ao lado do intérprete.

Figura 27 – Experimento 6.

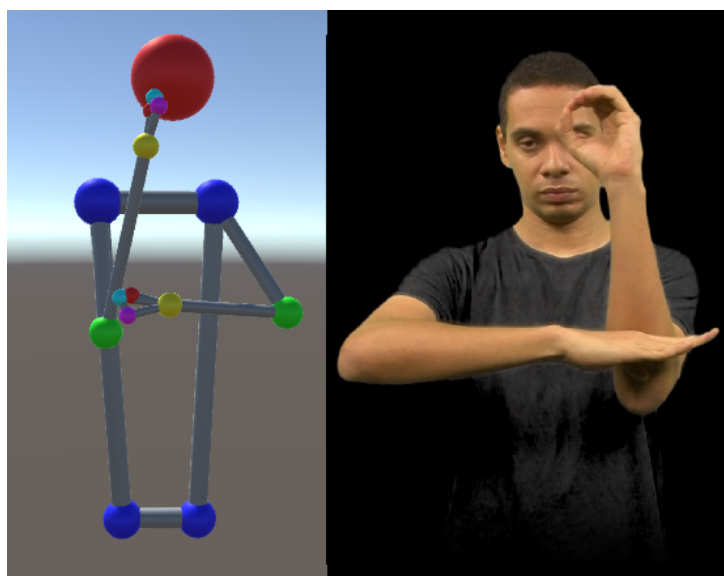


Fonte: Elaborada pelo autor.

4.2.3.7 Experimento 7 - Nascer do Sol

O sinal “Nascer do Sol” utiliza uma mão fazendo o mesmo sinal da letra “O”. O ponto de articulação é o cotovelo, e a mão sobe, simulando o movimento do sol surgindo. O outro braço permanece reto, com a outra mão um pouco acima do cotovelo que é articulado, representando o horizonte. A Figura 28 ilustra o esqueleto gerado pelo programa ao lado do intérprete.

Figura 28 – Experimento 7.

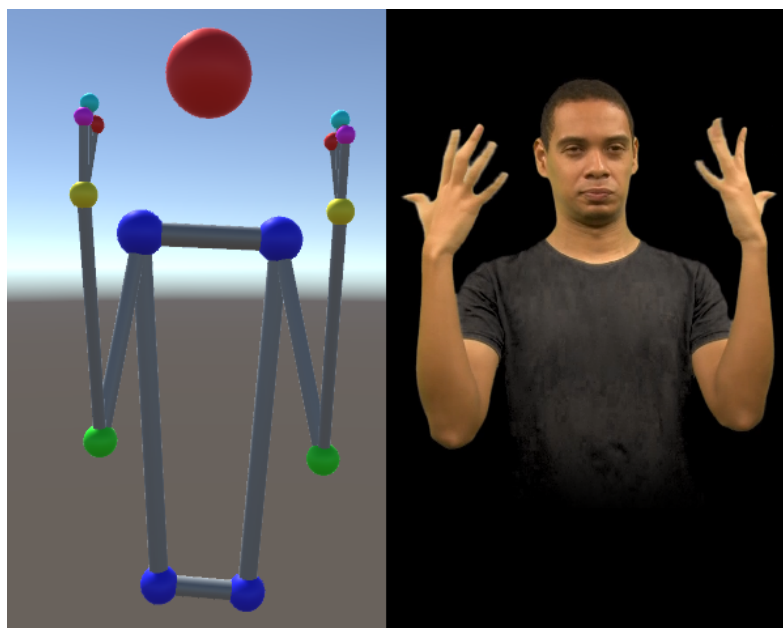


Fonte: Elaborada pelo autor.

4.2.3.8 Experimento 8 - Parabéns

Para o sinal “Parabéns”, o movimento é realizado com as mãos próximas à cabeça. Elas estão com as palmas abertas. O ponto de articulação são os pulsos, que irão rotacionar as mãos com as palmas abertas enquanto o movimento for executado. Na Figura 29, o esqueleto renderizado está posicionado ao lado do intérprete.

Figura 29 – Experimento 8.

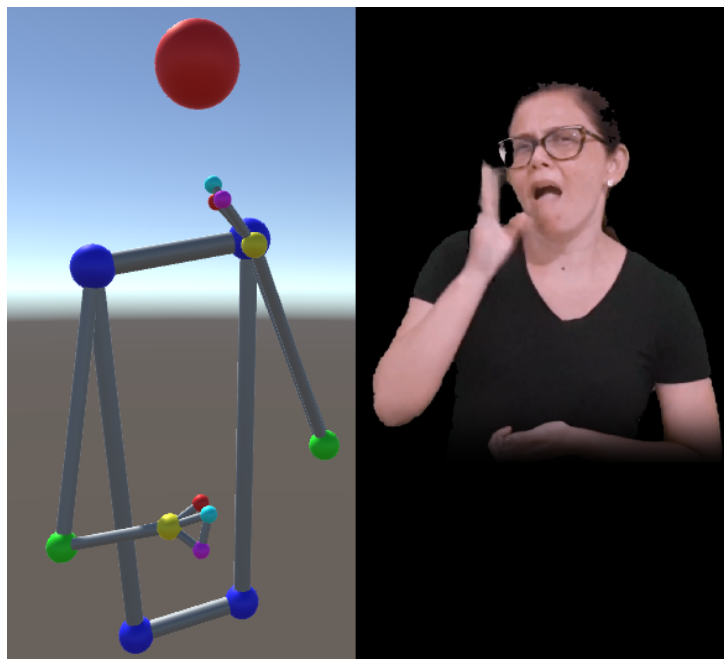


Fonte: Elaborada pelo autor.

4.2.3.9 Experimento 9 - Picante

No sinal “Picante”, o movimento envolve a mão próxima à boca com gesto de afastamento, com a palma da mão dominante aberta. O ponto de articulação é o pulso. Na Figura 30, visualiza-se o esqueleto processado pelo sistema junto ao intérprete.

Figura 30 – Experimento 9.

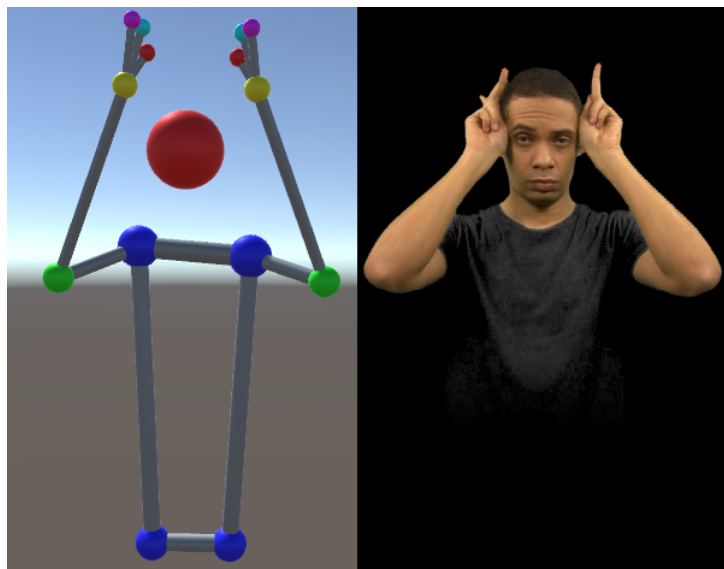


Fonte: Elaborada pelo autor.

4.2.3.10 Experimento 10 - Príncipe

Para o sinal “Príncipe”, o movimento começa com a mão ao lado da cabeça, onde ambas as mãos apontam o dedo indicador para cima, representando uma coroa. A mão dominante então desce e fecha com um formato de pinça no queixo, representando uma barba. Tanto os cotovelos como os pulsos são utilizados neste sinal. Na Figura 31 é possível observar o esqueleto renderizado pelo programa ao lado do intérprete.

Figura 31 – Experimento 10.

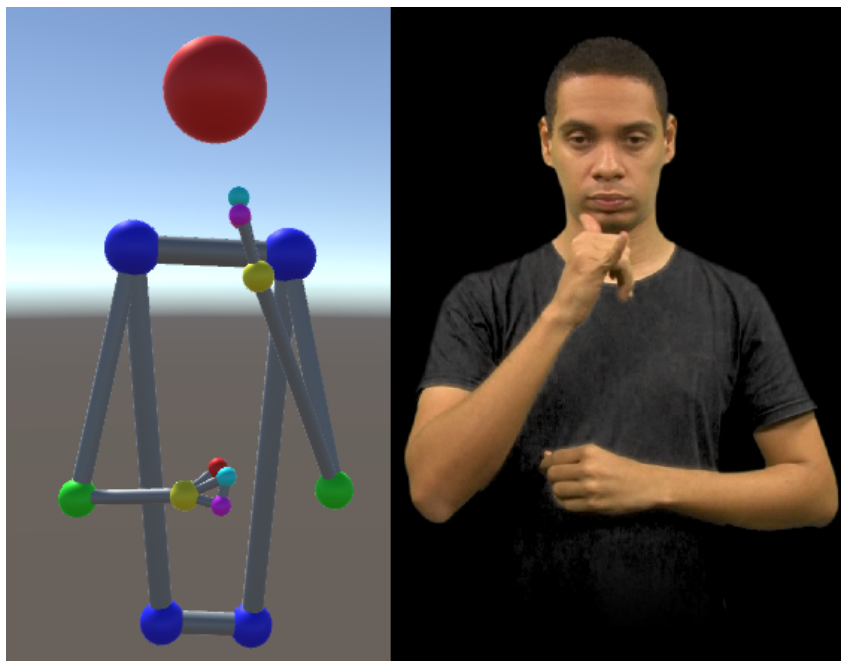


Fonte: Elaborada pelo autor.

4.2.3.11 Experimento 11 - Queda de Energia

O sinal “Queda de Energia” começa com a mão se movendo em direção à cabeça com as palmas abertas, e rapidamente se fecham. Depois fazem um sinal com o dedão e o dedo mindinho abertos, apenas na mão dominante, se movendo. Ambos os pulsos e cotovelos são utilizados. A Figura 32 ilustra o esqueleto gerado pelo programa ao lado do intérprete.

Figura 32 – Experimento 11.

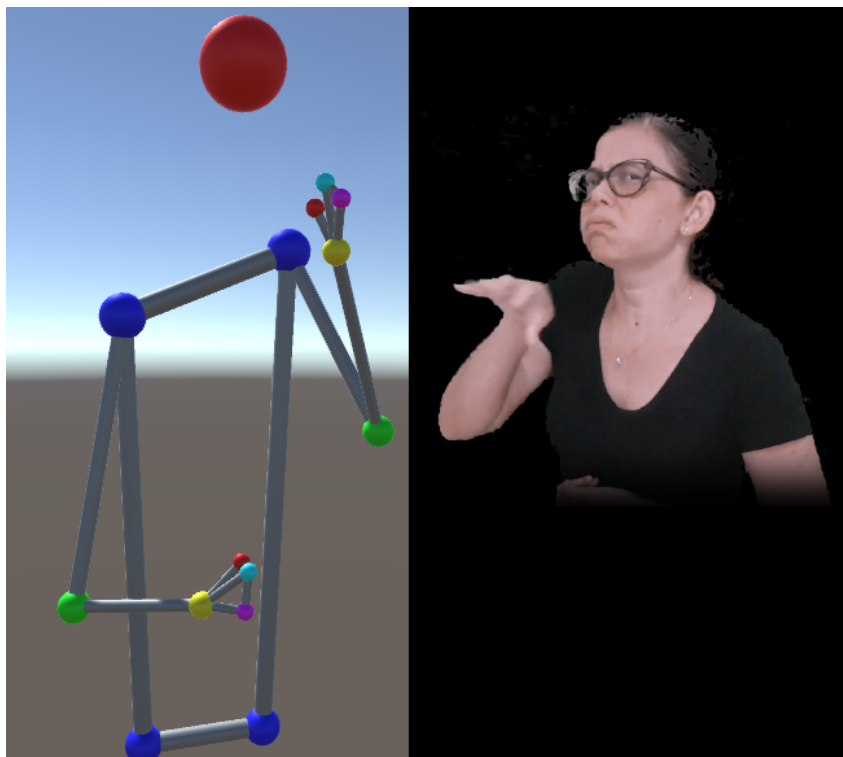


Fonte: Elaborada pelo autor.

4.2.3.12 Experimento 12 - Vomitar

Para o sinal “Vomitar”, a mão dominante completamente fechada é posicionada próxima à boca, com um movimento para frente ela se abre. Neste sinal, o pulso e o cotovelo são utilizados. Na Figura 33, observa-se o esqueleto renderizado ao lado do intérprete.

Figura 33 – Experimento 12.



Fonte: Elaborada pelo autor.

4.2.4 Considerações Finais

Como observado nos experimentos da subseção 4.2.3, o rastreamento de *landmarks* com MediaPipe representa uma maneira eficaz de transportar a interpretação de LIBRAS para o ambiente virtual. No entanto, os dados brutos coletados pelo MediaPipe Pose introduzem uma quantidade significativa de *jitter*, que corresponde a variações indesejadas na posição dos *landmarks*. Esse efeito gera instabilidade e reduz a fluidez das animações, o que compromete a fluidez do avatar durante a interpretação dos sinais.

Para resolver o problema de *jitter* no movimento dos *landmarks*, a aplicação do método SLERP seria ideal na aplicação de um trabalho futuro. Esse método utiliza uma interpolação esférica entre duas rotações, gerando uma transição suave em um arco ao invés de em linha reta. Isso permite uma transição contínua e natural entre as posições dos *landmarks*, suavizando os movimentos e proporcionando uma fluidez superior nas animações do avatar.

5 Conclusões

Este trabalho tinha como objetivo a criação de um sistema capaz de animar avatares virtuais 3D de alta fidelidade a partir de sequências de vídeos de sinais de LIBRAS. O projeto só se tornou possível graças à disponibilidade de tecnologias capazes de fazerem o rastreamento dos sinais de LIBRAS a partir de vídeos, como o MediaPipe e do *dataset* que está disponível de maneira gratuita.

Optou-se por não rastrear os *landmarks* do rosto em um primeiro momento, para concentrar nos movimentos dos braços e mãos. Apesar de terem sido utilizados apenas 12 sinais de LIBRAS para teste do protótipo, potencialmente o método desenvolvido poderia ser utilizado para a maioria dos sinais da linguagem. Foi desenvolvido um ambiente virtual capaz de animar o esqueleto de um avatar virtual 3D para qualquer *dataset* completo, como por exemplo, o V-LIBRASIL. Também foi desenvolvido um algoritmo capaz de processar os vídeos dos *datasets* e deixá-los aptos para a renderização no projeto do Unity3D.

5.1 Trabalhos Futuros

Considerando a continuidade desse trabalho, alguns aspectos e ideias que podem ser levados em consideração são:

- Aplicar o *slerp* para suavizar transições nas animações, reduzindo tremores e melhorando a fluidez do movimento.
- Desenvolver uma topologia aprimorada para o esqueleto do avatar, conforme sugerido por Song et al. (2023), visando uma maior fidelidade dos movimentos.
- Testar modelos avançados do MediaPipe, como o Holistic, que captura corpo, rosto e mãos, para uma animação mais completa e realista.
- Testar e integrar novos *datasets* para ampliar a aplicação e análise de diferentes sinais em LIBRAS.
- Aplicação de uma rede neural para a estimação de dados de profundidade com maior precisão, conforme apresentado por Poullos et al. (2024).
- Realização de uma avaliação de inferência, utilizando métricas mais objetivas e um *ground truth*.

Referências

ABERMAN, K.; LI, P.; LISCHINSKI, D.; SORKINE-HORNUNG, O.; COHEN-OR, D.; CHEN, B. Skeleton-aware networks for deep motion retargeting. *ACM Trans. Graph.*, Association for Computing Machinery, New York, NY, USA, v. 39, n. 4, ago. 2020. ISSN 0730-0301. Disponível em: <https://doi.org/10.1145/3386569.3392462>. Acesso em: 26 out. 2024.

AMRUTHA, K.; PRABU, P.; PAULOSE, J. Human body pose estimation and applications. In: *2021 Innovations in Power and Advanced Computing Technologies (i-PACT)*. [s.n.], 2021. p. 1–6. Disponível em: <https://ieeexplore.ieee.org/document/9696513>. Acesso em: 23 mar. 2024.

ARNALDI, B.; GUITTON, P.; MOREAU, G. *Virtual Reality and Augmented Reality: Myths and Realities*. 1. ed. [S.l.]: Wiley-IEEE Press, 2018. ISBN 1-78630-105-9.

BAZAREVSKY, V.; GRISHCHENKO, I.; RAVEENDRAN, K.; ZHU, T.; ZHANG, F.; GRUNDMANN, M. *BlazePose: On-device Real-time Body Pose tracking*. 2020. Disponível em: <https://arxiv.org/abs/2006.10204>. Acesso em: 23 mar. 2024.

BRASIL. *Lei nº 10.436, de 24 de abril de 2002*. Dispõe sobre a Língua Brasileira de Sinais - Libras e dá outras providências. Brasília, DF, 2002. Disponível em: https://www.planalto.gov.br/ccivil_03/leis/2002/l10436.htm. Acesso em: 16 mar. 2024.

BRASIL. *Decreto nº 5.626, de 22 de dezembro de 2005*. Regulamenta a Lei nº 10.436, de 24 de abril de 2002, que dispõe sobre a Língua Brasileira de Sinais - Libras, e o art. 18 da Lei nº 10.098, de 19 de dezembro de 2000. Brasília, DF, 2005. Disponível em: https://www.planalto.gov.br/ccivil_03/_ato2004-2006/2005/decreto/d5626.htm. Acesso em: 16 mar. 2024.

CAPOVILLA, F. C.; RAPHAEL, W. D.; MARTINS, A. C.; TEMOTEO, J. G. *Dicionário da Língua de Sinais do Brasil: A Libras em suas Mãos*. 1. ed. [S.l.]: EDUSP, 2017. ISBN 978-85-314-1645-3.

CERNA, L. R.; CARDENAS, E. E.; MIRANDA, D. G.; MENOTTI, D.; CAMARA-CHAVEZ, G. A multimodal libras-ufop brazilian sign language dataset of minimal pairs using a microsoft kinect sensor. *Expert Systems with Applications*, v. 167, p. 114179, 2021. ISSN 0957-4174. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0957417420309143>. Acesso em: 23 out. 2024.

CRISTIANO, A. *"O que é Libras?"*. 2017. Disponível em: <https://www.libras.com.br/o-que-e-libras>. Acesso em: 16 mar. 2024.

CRISTIANO, A. *"Os Cinco Parâmetros da Libras"*. 2018. Disponível em: <https://www.libras.com.br/os-cinco-parametros-da-libras>. Acesso em: 16 mar. 2024.

DIAS, L.; BRAZ, R. M.; DELOU, C.; WINAGRASKI, E.; CARVALHO, H.; CASTRO, H. Deafness and the educational rights: A brief review through a brazilian perspective. *Creative Education*, v. 05, p. 491–500, 01 2014. Disponível em:

<https://www.scirp.org/journal/paperinformation?paperid=45393>. Acesso em: 24 mar. 2024.

FELIPE, T. A.; MONTEIRO, M. *Libras em Contexto: Curso Básico : Livro do Professor*. 6. ed. [S.l.]: WalPrint, 2007.

GAMEIRO, P. V.; PASSOS, W. L.; ARAUJO, G. M.; LIMA, A. A. de; GOIS, J. N.; CORBO, A. R. A brazilian sign language video database for automatic recognition. In: *2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE)*. [s.n.], 2020. p. 1–6. Disponível em: <https://ieeexplore.ieee.org/document/9307017>. Acesso em: 23 out. 2024.

GOMES, I. *Pessoas com deficiência têm menor acesso à educação, ao trabalho e à renda*. 2023. Disponível em: <https://agenciadenoticias.ibge.gov.br/agencia-noticias/2012-agencia-de-noticias/noticias/37317-pessoas-com-deficiencia-tem-menor-acesso-a-educacao-ao-trabalho-e-a-renda>. Acesso em: 16 mar. 2024.

Google AI Edge. *MediaPipe Pose*. 2023. Disponível em: <https://github.com/google-ai-edge/mediapipe/blob/master/docs/solutions/pose.md>. Acesso em: 22 out. 2024.

Google AI Edge. *MediaPipe Solutions Guide*. 2024. Disponível em: <https://ai.google.dev/edge/mediapipe/solutions/guide>. Acesso em: 22 out. 2024.

LUNA, A.; WALLER, J.; KUSHALNAGAR, R.; VOGLER, C. Evaluation of anonymized sign language videos filtered using mediapipe. *The Journal On Technology and Persons with Disabilities*, 2023. ISSN 2330-4219. Disponível em: <https://scholarworks.calstate.edu/concern/publications/765378195>. Acesso em: 16 mar. 2024.

MediaPipe Pose. *Pose landmark detection guide*. 2024. Disponível em: https://developers.google.com/mediapipe/solutions/vision/pose_landmarker#:~:text=The%20MediaPipe%20Pose%20Landmarker%20task,with%20single%20images%20or%20video. Acesso em: 23 mar. 2024.

MILGRAM, P.; TAKEMURA, H.; UTSUMI, A.; KISHINO, F. Augmented reality: a class of displays on the reality-virtuality continuum. *Telemanipulator and Telepresence Technologies*, SPIE, 1995. Disponível em: <https://www.spiedigitallibrary.org/conference-proceedings-of-spie/2351/1/Augmented-reality--a-class-of-displays-on-the-reality/10.1117/12.197321.short>. Acesso em: 22 out. 2024.

NOWAK, K. L.; FOX, J. Avatars and computer-mediated communication: a review of the definitions, uses, and effects of digital representations. *Review of Communication Research*, v. 6, p. 30–53, 2018. ISSN 2255-4165. Disponível em: <https://nbn-resolving.org/urn:nbn:de:0168-ssolar-55777-7>. Acesso em: 25 out. 2024.

PERES, S. M.; FLORES, F. C.; VERONEZ, D.; OLGUIN, C. J. M. Libras signals recognition: a study with learning vector quantization and bit signature. In: *2006 Ninth Brazilian Symposium on Neural Networks (SBRN'06)*. [s.n.], 2006. p. 119–124. Disponível em: <https://ieeexplore.ieee.org/document/4026821>. Acesso em: 23 mar. 2024.

PORFIRIO, A. J.; WIGGERS, K. L.; OLIVEIRA, L. E.; WEINGAERTNER, D. Libras sign language hand configuration recognition based on 3d meshes. In: *2013 IEEE International Conference on Systems, Man, and Cybernetics*. [s.n.], 2013. p. 1588–1593. Disponível em: <https://ieeexplore.ieee.org/document/6722027>. Acesso em: 23 out. 2024.

POULIOS, I.; PISTOLA, T.; SYMEONIDIS, S.; DIPLARIS, S.; IOANNIDIS, K.; VROCHIDIS, S.; KOMPATSIARIS, I. Enhanced real-time motion transfer to 3d avatars using rgb-based human 3d pose estimation. In: *Proceedings of the 2024 ACM International Conference on Interactive Media Experiences Workshops*. New York, NY, USA: Association for Computing Machinery, 2024. (IMXw '24), p. 88–99. ISBN 9798400717949. Disponível em: <https://doi.org/10.1145/3672406.3672427>. Acesso em: 26 out. 2024.

REZENDE, T. M. *Reconhecimento automático de sinais da Libras : desenvolvimento da base de dados MINDS-Libras e modelos de redes convolucionais*. Tese (Tese de Doutorado) — Universidade Federal de Minas Gerais, jul. 2021. Disponível em: <http://hdl.handle.net/1843/39785>. Acesso em: 24 out. 2024.

RODRIGUES, A. J. *V-LIBRASIL : uma base de dados com sinais na língua brasileira de sinais (Libras)*. Tese (Dissertação de Mestrado) — Universidade Federal de Pernambuco, ago. 2021. Disponível em: <https://repositorio.ufpe.br/handle/123456789/43491>. Acesso em: 23 mar. 2024.

SARMENTO, A. H. de A.; PONTI, M. A. A cross-dataset study on the brazilian sign language translation. In: *2023 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*. [s.n.], 2023. p. 2808–2812. Disponível em: <https://ieeexplore.ieee.org/document/10350372>. Acesso em: 24 out. 2024.

SCHRAMM, W. *The Process and Effects of Mass Communication*. 4. ed. [S.l.]: University of Illinois Press, 1960. ISBN 978-0252001970.

SILVA, A. L. d. C.; Sá, T. M. d.; DINIZ, R. S.; FERREIRA, S. B. L.; SIQUEIRA, S. W. M.; BOURGUIGNON, S. C. Prescriptive and Semantic Analysis of an Automatic Sign Language Translation: Cases on VLibras Avatar Translation Using Video Interviews and Textual Interactions With a Chatbot. *Interacting with Computers*, v. 35, n. 2, p. 231–246, 10 2022. ISSN 1873-7951. Disponível em: <https://doi.org/10.1093/iwc/iwac020>. Acesso em: 17 mar. 2024.

SONG, W.; ZHANG, X.; GAO, Y.; LUO, Y.; WANG, H.; WANG, X.; HOU, X. Upsr: a unified proxy skeleton retargeting method for heterogeneous avatar animation. In: *2023 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*. [s.n.], 2023. p. 867–868. Disponível em: <https://ieeexplore.ieee.org/document/10108598>. Acesso em: 23 mar. 2024.

STEPHENSON, N. *Snow Crash*. 1. ed. Nova York: Bantam Books, 1992. ISBN 978-0553088533.

Unity Technologies. *Unity - Manual: Unity User Manual 2022.3 (LTS)*. 2022. Disponível em: <https://docs.unity3d.com/2022.3/Documentation/Manual/index.html>. Acesso em: 25 out. 2024.

WANG, R.; CAO, Y.; HAN, K.; WONG, K.-Y. K. A survey on 3d human avatar modeling – from reconstruction to generation. 2024. Disponível em: <https://arxiv.org/abs/2406.04253>. Acesso em: 25 out. 2024.

WOLFE, R.; MCDONALD, J. C.; HANKE, T.; EBLING, S.; LANDUYT, D. V.; PICRON, F.; KRAUSNEKER, V.; EFTHIMIOU, E.; FOTINEA, E.; BRAFFORT, A. Sign language avatars: A question of representation. *Information*, v. 13, n. 4, 2022. ISSN 2078-2489. Disponível em: <https://www.mdpi.com/2078-2489/13/4/206>. Acesso em: 17 mar. 2024.

World Health Organization. *Deafness and hearing loss*. 2023. Disponível em: <https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss>. Acesso em: 16 mar. 2024.

ZHENG, J. M.; CHAN, K. W.; GIBSON, I. Virtual reality. *IEEE Potentials*, IEEE, v. 17, n. 2, p. 20–23, 1998. Disponível em: <https://ieeexplore.ieee.org/abstract/document/666641/authors#authors>. Acesso em: 22 out. 2024.

Apêndices

APÊNDICE A – Código de processamento do MediaPipe e geração dos arquivos JSON

```
1 import os
2 import numpy as np
3 import json
4 import cv2 # OpenCV para processamento dos vídeos
5
6 # Ver doc. MediaPipe Holistic (OLD): https://ai.google.dev/edge/mediapipe/solutions/vision/pose\_landmarker/python
7 import mediapipe as mp
8 from mediapipe.tasks import python
9 from mediapipe.tasks.python import vision
10
11 # Para desenhar a saída em um quadro: https://github.com/google-ai-edge/mediapipe-samples/blob/main/examples/pose\_landmarker/python/%5BMediaPipe\_Python\_Tasks%5D\_Pose\_Landmarker.ipynb
12 from mediapipe import solutions
13 from mediapipe.framework.formats import landmark_pb2
14 from google.protobuf import timestamp_pb2
15
16 def mp_draw_image(rgb_image, landmarks_result):
17     """Função para desenhar a saída do MediaPipe em um quadro.
18
19     Args:
20         rgb_image : TYPE
21             Quadro da imagem no formato de RGB (usar OpenCV2). \n
22         landmarks_result : TYPE
23             Resultado dos pontos-chave após processamento do MediaPipe Pose. \n
24
25     Returns:
26         annotated_image : TYPE
27             Quadro contendo os pontos-chave do MediaPipe anotados. \n
28     """
29     pose_landmarks_list = landmarks_result.pose_landmarks
30     annotated_image = np.copy(rgb_image)
31
32     # Percorrendo cada ponto-chave calculado
33     for idx in range(len(pose_landmarks_list)):
```

```

34     pose_landmarks = pose_landmarks_list[idx]
35
36     # Desenhando os pontos sobre o quadro
37     pose_landmarks_proto = landmark_pb2.NormalizedLandmarkList() #
type: ignore
38     pose_landmarks_proto.landmark.extend(
39         [
40             landmark_pb2.NormalizedLandmark( # type: ignore
41                 x=landmark.x, y=landmark.y, z=landmark.z
42             )
43             for landmark in pose_landmarks
44         ]
45     )
46     solutions.drawing_utils.draw_landmarks( # type: ignore
47         annotated_image,
48         pose_landmarks_proto,
49         solutions.pose.POSE_CONNECTIONS, # type: ignore
50         solutions.drawing_styles.get_default_pose_landmarks_style(),
51         # type: ignore
52     )
53     return annotated_image
54
55
56 def mp_process_video(video_path: str, video_name: str, output_path: str)
:
57     """Função para Aplicar o MediaPipe Holistic em UM vídeo.
58
59     Args:
60         video_path : str
61             Caminho do vídeo (com extensão) para processamento (ex: C:/
video.mp4). \n
62         video_name : str
63             Nome do arquivo de vídeo para criação dos vídeos de pós-
processamento (ex: video.mp4). \n
64         output_path : str
65             Pasta de saída para criação dos vídeos de pós-processamento
(ex: C:/). \n
66
67     Returns:
68         video_landmarks : list ['Quadro1': [landmark0, landmark1, ...],
...]\n
69             Lista contendo os pontos rastreados, por quadro, de um vídeo
. \n
70         """
71
72     # Definindo o modelo a ser utilizado

```

```

73     # actual_dir = os.path.dirname(__file__)
74     holistic_models_path = os.path.join('.', 'holistic_models') # Pasta
    contendo os modelos do MediaPipe
75     holistic_models = {
76         'Lite': os.path.join(holistic_models_path, "pose_landmarker_lite
    .task"),
77         'Full': os.path.join(holistic_models_path, "pose_landmarker_full
    .task"),
78         'Heavy': os.path.join(holistic_models_path, "
    pose_landmarker_heavy.task"),
79     }
80     model = holistic_models.get('Full') # Selecionar um modelo para o
    Holistic
81
82     # Criando a tarefa do MediaPipe (configuração do grafo de
    funcionamento)
83     BaseOptions = mp.tasks.BaseOptions
84     PoseLandmarker = mp.tasks.vision.PoseLandmarker
85     PoseLandmarkerOptions = mp.tasks.vision.PoseLandmarkerOptions
86     VisionRunningMode = mp.tasks.vision.RunningMode
87
88     # Criando a tarefa de rastreamento de pose no MODO DE VÍDEO
89     # Ver parâmetros em: https://ai.google.dev/edge/mediapipe/solutions/
    vision/pose\_landmarker/python#configuration\_options
90     options = PoseLandmarkerOptions(
91         base_options=BaseOptions(model_asset_path=model),
92         running_mode=VisionRunningMode.VIDEO,
93         num_poses=1,
94         min_pose_detection_confidence=0.5,
95         min_pose_presence_confidence=0.5,
96         min_tracking_confidence=0.5,
97         output_segmentation_masks=False,
98     )
99     # Inicializando o grafo de funcionamento do MediaPipe Holistic
100     with PoseLandmarker.create_from_options(options) as landmarker:
101
102         # Carregando o vídeo com o OpenCV
103         video = cv2.VideoCapture(video_path)
104         frame_width = video.get(cv2.CAP_PROP_FRAME_WIDTH) # Largura do
    Vídeo
105         frame_height = video.get(cv2.CAP_PROP_FRAME_HEIGHT) # Altura do
    Vídeo
106
107         print("Video Resolution: ", frame_width, " x ", frame_height)
108         print("Video FPS: ", video.get(cv2.CAP_PROP_FPS))
109         print("Video Frame Count:", video.get(cv2.CAP_PROP_FRAME_COUNT))
110

```

```

111     #! Criação de um vídeo de saída, com o resultado de
processamento
112     video_output_test = cv2.VideoWriter(
113         filename=os.path.join(output_path, f'{video_name}_output.mp4
'),
114         fourcc=cv2.VideoWriter_fourcc(*"mp4v"), # type: ignore
115         # fourcc=cv2.VideoWriter_fourcc(*"MP4V"),
116         fps=video.get(cv2.CAP_PROP_FPS),
117         frameSize=(int(frame_width), int(frame_height)),
118     )
119
120     # Percorrendo os quadros do vídeo
121     frames_landmarks = []
122     frame_count = 0
123     while video.isOpened():
124         sucess, frame = video.read()
125         if not sucess:
126             print("Ignorando quadro vazio")
127             break # 'break' para vídeo e 'continue' para live-vídeo
128
129         # Convertendo o quadro obtido para um objeto válido do
MediaPipe (BGR para RGB)
130         frame_converted = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
131         mp_image = mp.Image(
132             image_format=mp.ImageFormat.SRGB,
133             data=frame_converted
134         )
135
136         # Aplicando a tarefa de rastreamento do Holistic/Pose
Estimator
137         frame_timestamp = video.get(cv2.CAP_PROP_POS_MSEC) # Tempo
em segundos
138         # print('Processando: ', mp_image, " ", frame_timestamp)
139         pose_landmarker_result = landmarker.detect_for_video(
140             mp_image,
141             mp.Timestamp.from_seconds(frame_timestamp).microseconds
142         ) # Tempo em Microsegundos
143
144         # Mapeamento dos pontos do MediaPipe Pose
145         point_names = {
146             0: "Nariz",
147             1: "Olho_Esquerdo_Interno",
148             2: "Olho_Esquerdo",
149             3: "Olho_Esquerdo_Externo",
150             4: "Olho_Direito_Interno",
151             5: "Olho_Direito",

```

```

152         6: "Olho_Direito_Externo",
153         7: "Orelha_Esquerda",
154         8: "Orelha_Direita",
155         9: "Boca_Esquerda",
156         10: "Boca_Direita",
157         11: "Ombro_Esquerdo",
158         12: "Ombro_Direito",
159         13: "Cotovelo_Esquerdo",
160         14: "Cotovelo_Direito",
161         15: "Pulso_Esquerdo",
162         16: "Pulso_Direito",
163         17: "Dedo_Mindinho_Esquerdo",
164         18: "Dedo_Mindinho_Direito",
165         19: "Dedo_Indicador_Esquerdo",
166         20: "Dedo_Indicador_Direito",
167         21: "Dedao_Mao_Esquerda",
168         22: "Dedao_Mao_Direita",
169         23: "Quadril_Esquerdo",
170         24: "Quadril_Direito",
171         25: "Joelho_Esquerdo",
172         26: "Joelho_Direito",
173         27: "Tornozelo_Esquerdo",
174         28: "Tornozelo_Direito",
175         29: "Calcanhar_Esquerdo",
176         30: "Calcanhar_Direito",
177         31: "Dedo_Indicador_Pe_Esquerdo",
178         32: "Dedo_Indicador_Pe_Direito",
179     }
180
181     # Lista de pontos para serem ignorados
182     # Neste caso, os pontos ignorados são aqueles que não serão
183     considerados para um primeiro momento.
184     ignored_points = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 25, 26, 27,
185                      28, 29, 30, 31, 32}
186
187     # Percorrendo cada ponto-chave encontrado no quadro para
188     salvar em uma lista
189     formatted_pose_landmarks = []
190     for idx in range(len(pose_landmarker_result.pose_landmarks)):
191         pose_landmarks = pose_landmarker_result.pose_landmarks[
192             idx]
193
194     # Salvando os pontos, formatadamente, para a lista de saída
195     for num, landmark in enumerate(pose_landmarks):
196         if num in ignored_points:

```



```

193         continue
194
195         point_name = point_names.get(num, "Desconhecido")
196         formatted_pose_landmarks.append(
197             {
198                 f"ponto_{num} - {point_name}": {
199                     "x": landmark.x,
200                     "y": landmark.y,
201                     "z": landmark.z,
202                 }
203             }
204         )
205
206         # Salvando os pontos, formatadamente, para a lista de saída
207         frames_landmarks.append({f"quadro_{frame_count}":
formatted_pose_landmarks})
208         frame_count = frame_count + 1
209
210         #! Criando o quadro para vídeo de saída (BGR)
211         annotated_frame = np.full(
212             (int(frame_height), int(frame_width), 3), 0, dtype=np.
uint8
213         ) # Apenas os pontos com fundo personalizável
214         # annotated_frame = frame.copy() # Colocar os pontos em cima
do quadro do vídeo original
215         annotated_frame = mp_draw_image(annotated_frame,
pose_landmarker_result)
216         video_output_test.write(annotated_frame)
217
218         # Fechando os streams dos vídeos de entrada e saída
219         video.release()
220         video_output_test.release()
221
222         return frames_landmarks
223
224
225 def process_folder_videos(input_folder: str, output_folder: str):
226     """Processar os vídeos das subpastas de uma pasta, sendo essas
227     subpastas com o nome da classe dos vídeos (ex: para uso com V-
LIBRASIL).
228
229     Args:
230         input_folder : str
231             Caminho da pasta que contém as subpastas para processamento.
232         \n
233         output_folder : str
234             Caminho da pasta para saída dos arquivos JSON do

```

```

processamento. \n
234
235     Notes:
236         Os arquivos JSON de saída são organizados por classe
237
238     """
239     # Percorrer todas as subpastas (classes) na pasta de entrada
240     #for class_name in os.listdir(input_folder):
241     #    class_path = os.path.join(input_folder, class_name)
242 #
243     # Para cada classe, processar os vídeos internos
244 #    if os.path.isdir(class_path):
245         # Diretório para saída do processo, POR CLASSE
246         #class_output_folder = os.path.join(output_folder, class_name)
247     os.makedirs(output_folder, exist_ok=True)
248
249     # Percorrer cada vídeo e processar com MediaPipe
250     # json_output_list = []
251     for filename in os.listdir(input_folder):
252         if filename.endswith('.mp4') or filename.endswith('.avi'):
253             # Configurando o caminho do vídeo, com extensão
254             video_path = os.path.join(input_folder, filename)
255
256             # Procesando com MediaPipe Holistic
257             print(f"Processando... -> '{filename}'")
258             video_landmarks = mp_process_video(
259                 video_path, os.path.splitext(filename)[0], output_folder
260             )
261             print(f"Processado! -> '{filename}'")
262
263             # Salvando os resultados do vídeo para um dicionário
264             #formatado
265             json_output_list = {
266                 "nome_video": f"{filename}",
267                 "classe": f"{class_name}",
268                 "landmarks_quadros": video_landmarks,
269             }
270
271             # Salvando as variáveis para saída no JSON
272             output_path = os.path.join(
273                 output_folder,
274                 f"{filename}_landmarks.json",
275             )
276
277             # Salvar os landmarks POR CLASSE no arquivo JSON
278             with open(output_path, "w") as outfile:

```

```

279         json.dump(json_output_list, outfile, indent=4)
280
281
282 def main():
283     """Função principal do programa
284     """
285
286     # Caminhos para a pasta contendo os vídeos de entrada e saída para o
    JSON
287     input_folder = "./videos UFPE (V-LIBRASIL)/data"  #! Cada classe de
    um dataset teria que ser atribuída nesta variável
288     output_folder = "./saida_processamento"  #! O(s) JSON(s) de saída
    ficam nas subpastas (classes) desta variável
289
290     # Processar todos os vídeos da pasta de entrada e salvar o JSON
    resultante na pasta de saída
291     process_folder_videos(input_folder=input_folder, output_folder=
    output_folder)
292
293
294 if __name__ == '__main__':
295     """Ponto de entrada do programa
296     """
297
298     #Chama a função principal
299     main()

```

APÊNDICE B – Código de animação do esqueleto no Unity3D

```
1 using System;
2 using System.Collections;
3 using System.Collections.Generic;
4 using UnityEditor;
5 using UnityEngine;
6 using System.IO;
7 using System.Linq;
8 using Newtonsoft.Json;
9
10 // Estrutura auxiliar para armazenar os dados do JSON
11 [System.Serializable]
12 public class RootObject
13 {
14     public string nome_video { get; set; }
15     public List<Dictionary<string, List<Dictionary<string, Landmark>>>>
        landmarks_quadros { get; set; }
16 }
17
18 // Estrutura com as coordenadas das esferas
19 [System.Serializable]
20 public class Landmark
21 {
22     public float x { get; set; }
23     public float y { get; set; }
24     public float z { get; set; }
25 }
26
27 public class MediaPipeJSONParser : MonoBehaviour
28 {
29     // string file_path = "D:/Downloads/Ambiente---TCC/Assets/Resources/
        JSON/abacaxi_articulador1.mp4_landmarks.json";
30
31     public GameObject spherePrefab; // Prefab da esferas que representam
        os landmarks
32     public GameObject cylinderPrefab; // Prefab dos cilindros que
        representam as conexoes entre cada esfera
33     private Dictionary<string, List<Dictionary<string, Vector3>>>
        framesLandmarks; // Dicionario para armazenar os landmarks de cada
        quadro
34     private List<GameObject> spheres; // Lista de esferas que
```

```

representarao os landmarks
35     private List<GameObject> cylinders; // Lista de cilindros que
conectarao os landmarks
36     private int currentFrame = 0; // Quadro atual da animacao
37     public float avatarScaleFactor; // Escala de distancia dos pontos do
avatar desejado
38     public string sinalDesejado; // Nome do video e articulador desejado
39
40     // Dicionario para armazenar o nome de cada landmark, representados
pelas esferas
41     private Dictionary<int, string> pointNames = new Dictionary<int,
string>()
42     {
43         { 0, "Nariz" },
44         { 1, "Ombro_Esquerdo" },
45         { 2, "Ombro_Direito" },
46         { 3, "Cotovelo_Esquerdo" },
47         { 4, "Cotovelo_Direito" },
48         { 5, "Pulso_Esquerdo" },
49         { 6, "Pulso_Direito" },
50         { 7, "Dedo_Mindinho_Esquerdo" },
51         { 8, "Dedo_Mindinho_Direito" },
52         { 9, "Dedo_Indicador_Esquerdo" },
53         { 10, "Dedo_Indicador_Direito" },
54         { 11, "Dedao_Esquerdo" },
55         { 12, "Dedao_Direito" },
56         { 13, "Quadril_Esquerdo" },
57         { 14, "Quadril_Direito" }
58     };
59
60     // Dicionario para armazenar os tamanhos de cada esfera que
representa os landmarks
61     private Dictionary<string, Vector3> sphereScaleMap = new Dictionary<
string, Vector3>()
62     {
63         { "Nariz", new Vector3(0.5f, 0.5f, 0.5f) },
64         { "Ombro_Esquerdo", new Vector3(0.3f, 0.3f, 0.3f) },
65         { "Ombro_Direito", new Vector3(0.3f, 0.3f, 0.3f) },
66         { "Cotovelo_Esquerdo", new Vector3(0.2f, 0.2f, 0.2f) },
67         { "Cotovelo_Direito", new Vector3(0.2f, 0.2f, 0.2f) },
68         { "Pulso_Esquerdo", new Vector3(0.15f, 0.15f, 0.15f) },
69         { "Pulso_Direito", new Vector3(0.15f, 0.15f, 0.15f) },
70         { "Dedo_Mindinho_Esquerdo", new Vector3(0.1f, 0.1f, 0.1f) },
71         { "Dedo_Mindinho_Direito", new Vector3(0.1f, 0.1f, 0.1f) },
72         { "Dedo_Indicador_Esquerdo", new Vector3(0.1f, 0.1f, 0.1f) },
73         { "Dedo_Indicador_Direito", new Vector3(0.1f, 0.1f, 0.1f) },
74         { "Dedao_Esquerdo", new Vector3(0.1f, 0.1f, 0.1f) },

```

```

75     { "Dedao_Direito", new Vector3(0.1f, 0.1f, 0.1f) },
76     { "Quadril_Esquerdo", new Vector3(0.25f, 0.25f, 0.25f) },
77     { "Quadril_Direito", new Vector3(0.25f, 0.25f, 0.25f) }
78 };
79
80 // Lista das conexoes entre as esferas que representam os landmarks
81 private List<(int, int)> connections = new List<(int, int)>
82 {
83     (1, 2), // Ombro_Esquerdo -> Ombro_Direito
84     (1, 13), // Ombro_Esquerdo -> Quadril_Esquerdo
85     (1, 3), // Ombro_Esquerdo -> Cotovelo_Esquerdo
86     (2, 4), // Ombro_Direito -> Cotovelo_Direito
87     (2, 14), // Ombro_Direito -> Quadril_Direito
88     (3, 5), // Cotovelo_Esquerdo -> Pulso_Esquerdo
89     (4, 6), // Cotovelo_Direito -> Pulso_Direito
90     (5, 7), // Pulso_Esquerdo -> Dedo_Mindinho_Esquerdo
91     (5, 9), // Pulso_Esquerdo -> Dedo_Indicador_Esquerdo
92     (5, 11), // Pulso_Esquerdo -> Dedao_Esquerda
93     (7, 9), // Dedo_Mindinho_Esquerdo -> Dedo_Indicador_Esquerdo
94     (6, 8), // Pulso_Direito -> Dedo_Mindinho_Direito
95     (6, 10), // Pulso_Direito -> Dedo_Indicador_Direito
96     (6, 12), // Pulso_Direito -> Dedao_Direita
97     (8, 10), // Dedo_Mindinho_Direito -> Dedo_Indicador_Direito
98     (13, 14) // Quadril_Esquerdo -> Quadril_Direito
99 };
100
101 // Dicionario para armazenar as cores de cada esfera que representam
102 // os landmarks
103 private Dictionary<string, Color> sphereColorMap = new Dictionary<
104 string, Color>()
105 {
106     { "Nariz", Color.red },
107     { "Ombro_Esquerdo", Color.blue },
108     { "Ombro_Direito", Color.blue },
109     { "Cotovelo_Esquerdo", Color.green },
110     { "Cotovelo_Direito", Color.green },
111     { "Pulso_Esquerdo", Color.yellow },
112     { "Pulso_Direito", Color.yellow },
113     { "Dedo_Mindinho_Esquerdo", Color.magenta },
114     { "Dedo_Mindinho_Direito", Color.magenta },
115     { "Dedo_Indicador_Esquerdo", Color.cyan },
116     { "Dedo_Indicador_Direito", Color.cyan },
117     { "Dedao_Esquerdo", Color.red },
118     { "Dedao_Direito", Color.red },
119     { "Quadril_Esquerdo", Color.blue },
120     { "Quadril_Direito", Color.blue }
121 };

```

```

120
121     // Funcao Start
122     void Start() {
123
124         // Carregar o JSON, utilizando a variavel com o nome e
articulador do sinal
125         string jsonPath = Application.dataPath + "/Resources/JSON/" +
sinalDesejado + ".mp4_landmarks.json";
126         string jsonString = File.ReadAllText(jsonPath);
127         Debug.Log("Loaded JSON: " + sinalDesejado);
128
129         // Deserializacao do arquivo JSON
130         var data = JsonConvert.DeserializeObject<RootObject>(jsonString)
;
131
132         // Impressao do nome do video, juntamente da quantidade de
quadros do mesmo
133         Debug.Log("Nome do vídeo: " + data.nome_video);
134         Debug.Log("Quantidade de quadros: " + data.landmarks_quadros.
Count);
135
136         // Criacao de um GameObject para manter as esferas que
representam os landmarks, e criacao do mesmo para os cilindros que
representam os conectores
137         GameObject parentObject = new GameObject("EsqueletoAvatar");
138         GameObject cylinderParentObject = new GameObject("Conectores");
139
140         // Preparar as esferas para os landmarks e os cilindros para os
conectores
141         spheres = new List<GameObject>();
142         cylinders = new List<GameObject>();
143
144         // Variaveis declaradas para manter salvas as posicoes dos
ombros e do nariz (FUNCAO NAO IMPLEMENTADA - CRIACAO DE PESCOCO E
CABECA)
145         Vector3 ombroEsquerdoPosition = Vector3.zero;
146         Vector3 ombroDireitoPosition = Vector3.zero;
147         Vector3 nosePosition = Vector3.zero;
148
149         // Dicionario para definir as relacoes de hierarquia entre as
esferas, iniciando nos ombros
150         Dictionary<string, GameObject> parentMap = new Dictionary<string
, GameObject>();
151
152         // Criacao das esferas
153         int index = 0; // Index auxiliar para indexacao das esferas
154         foreach (var landmark in data.landmarks_quadros[0]["quadro_0"])

```

```

155     // Loop para percorrer todos os landmarks dentre de um quadro
156     {
157         // Instanciar as esferas utilizando o prefab escolhido
158         GameObject sphere = Instantiate(spherePrefab);
159
160         // Localizacao das esferas baseadas no codigo do dicionario
161         if (pointNames.ContainsKey(index))
162         {
163             // Aplicacao de nome e insercao das esferas no ambiente
164             virtual
165             string sphereName = pointNames[index];
166             sphere.name = sphereName;
167             spheres.Add(sphere);
168
169             // Aplicar escala e cor para as esferas, baseados nos
170             seus devidos dicionarios
171             if (sphereScaleMap.ContainsKey(sphereName))
172             {
173                 sphere.transform.localScale = sphereScaleMap[
174                 sphereName];
175                 sphere.GetComponent<Renderer>().material.color =
176                 sphereColorMap[sphereName];
177             }
178
179             // Buscar posicao dos ombros e tambem do nariz (FUNCAO
180             NAO IMPLEMENTADA - CRIACAO DE PESCOCO E CABECA)
181             if (sphereName == "Ombro_Esquerdo")
182             {
183                 ombroEsquerdoPosition = sphere.transform.position;
184             }
185             else if (sphereName == "Ombro_Direito")
186             {
187                 ombroDireitoPosition = sphere.transform.position;
188             }
189             else if (sphereName == "Nariz")
190             {
191                 nosePosition = sphere.transform.position;
192             }
193
194             // Criacao de um GameObject vazio para criacao da
195             hierarquia sem afetar a escala
196             GameObject dummyParent = new GameObject(sphereName + "
197             _Holder"); // Nome do GameObject vazio
198             dummyParent.transform.SetParent(parentObject.transform);
199             // Inserir o GameObject dentro da hierarquia
200
201             //Codigo para que quando adicionar a esfera para dentro

```



```

do GameObject vazio nao altere a escala
193     sphere.transform.SetParent(dummyParent.transform, false)
;
194
195     // Estabelecimento da logica de hierarquia
196     if (sphereName.Contains("Ombro")) // Ombros sao o
primeiro nivel
197     {
198         parentMap[sphereName] = dummyParent; // Armazenar as
esferas dentro do GameObject pai vazio, de mesmo nome
199     }
200     else if (sphereName.Contains("Cotovelo") || sphereName.
Contains("Quadril"))
201     {
202         // Colocar cotovelos e quadris como filhos de ombro
203         if (sphereName.Contains("Esquerdo"))
204             dummyParent.transform.SetParent(parentMap["
Ombro_Esquerdo"].transform, false); // Filhos do ombro esquerdo
205         else if (sphereName.Contains("Direito"))
206             dummyParent.transform.SetParent(parentMap["
Ombro_Direito"].transform, false); // Filhos do ombro direito
207
208         parentMap[sphereName] = dummyParent; // Armazenar
estes como pontos pais, para que possam receber mais esferas na
hierarquia
209     }
210     else if (sphereName.Contains("Pulso"))
211     {
212         // Colocar pulsos como filhos do cotovelo
213         if (sphereName.Contains("Esquerdo"))
214             dummyParent.transform.SetParent(parentMap["
Cotovelo_Esquerdo"].transform, false); // Filhos do cotovelo esquerdo
215         else if (sphereName.Contains("Direito"))
216             dummyParent.transform.SetParent(parentMap["
Cotovelo_Direito"].transform, false); // Filhos do cotovelo direito
217
218         parentMap[sphereName] = dummyParent; // Armazenar
estes como pontos pais, para que possam receber mais esferas na
hierarquian
219     }
220     else if (sphereName.Contains("Dedao") || sphereName.
Contains("Dedo"))
221     {
222         // Colocar dedos como filhos do pulso
223         if (sphereName.Contains("Esquerdo"))
224             dummyParent.transform.SetParent(parentMap["
Pulso_Esquerdo"].transform, false); // Filhos do pulso esquerdo

```

```

225         else if (sphereName.Contains("Direito"))
226             dummyParent.transform.SetParent(parentMap["
Pulso_Direito"].transform, false); // Filhos do pulso direito
227     }
228 }
229
230     index++;
231 }
232
233     // Instanciacao dos cilindros para cada conexao especifica,
baseado no dicionario de conexoes
234     foreach (var connection in connections)
235     {
236         GameObject cylinder = Instantiate(cylinderPrefab);
237
238         // Pegar os indices em que duas esferas estao conectadas
239         int sphereIndex1 = connection.Item1;
240         int sphereIndex2 = connection.Item2;
241
242         // Pegar os nomes das duas esferas que estao conectadas
243         string sphereName1 = spheres[sphereIndex1].name;
244         string sphereName2 = spheres[sphereIndex2].name;
245
246         // Nomear os cilindros de acordo com as esferas que ele
conecta
247         cylinder.name = $"Conector: {sphereName1} -> {sphereName2}";
248
249         // Selecionar a cor dos cilindros
250         cylinder.GetComponent<Renderer>().material.color = Color.
gray;
251
252         // Adicionar o cilindro ao GameObject pai
253         cylinder.transform.SetParent(cylinderParentObject.transform)
;
254
255         // Adicionar os cilindros conectores no ambiente virtual
256         cylinders.Add(cylinder);
257     }
258
259     // Armazenar os dados de cada quadro
260     framesLandmarks = new Dictionary<string, List<Dictionary<string,
Vector3>>>();
261
262     // Variaveis utilizadas para salvar os valores maximos e minimos
para calcular o centro do "esqueleto" criado
263     Vector3 minValues = new Vector3(float.MaxValue, float.MaxValue,
float.MaxValue);

```

```

264     Vector3 maxValues = new Vector3(float.MinValue, float.MinValue,
float.MinValue);
265
266     // Inicio do loop de leitura da posicao dos landmarks no
primeiro quadro do arquivo JSON
267     foreach (var quadro in data.landmarks_quadros)
268     {
269         foreach (var frame in quadro)
270         {
271             // Dicionario de cada landmark em um certo quadro do
video
272             List<Dictionary<string, Vector3>> landmarksList = new
List<Dictionary<string, Vector3>>();
273
274             foreach (var landmark in frame.Value)
275             {
276                 // Dicionario dos valores salvos das coordenadas de
cada um dos landmarks
277                 Dictionary<string, Vector3> landmarkData = new
Dictionary<string, Vector3>();
278
279                 foreach (var point in landmark)
280                 {
281                     /* TRANSFORMACAO DAS COORDENADAS DO MEDIAPIPE
PARA O UNITY3D
282
283                     Para a transformacao dos dados do
MediaPipe, foi necessario fazer algumas mudancas devido ao diferente
sistemas de coordenadas
284
285                     - Inverter o eixo Y
- Multiplicar o eixo Z para reposicionar
e melhorar a profundidade
286
- Multiplicar todos os valores por uma
escala de distância
287
288                     */
289                     Vector3 pos = new Vector3((point.Value.x) * 1.0f
* avatarScaleFactor, 1.0f - (point.Value.y) * 1.0f *
avatarScaleFactor, -(point.Value.z) * 0.23f * avatarScaleFactor);
290                     landmarkData.Add(point.Key, pos);
291
292                     // Salvar os valores maximos e minimos de x, y e
z
293
294                     minValues = Vector3.Min(minValues, pos);
maxValues = Vector3.Max(maxValues, pos);
295                 }
296                 // Adicionar os landmarks na lista

```

```

297         landmarksList.Add(landmarkData);
298     }
299     // Armazenar os dados de cada quadro na lista de
landmarks
300     framesLandmarks.Add(frame.Key, landmarksList);
301 }
302 }
303 // Calcular o centro do "esqueleto" do avatar
304 Vector3 bodyCenter = (minValues + maxValues) / 2;
305
306 // Aplicar os valores para centralizar todos os landmarks com o
plano de coordenadas do unity
307 foreach (var quadro in framesLandmarks.Keys.ToList())
308 {
309     for (int i = 0; i < framesLandmarks[quadro].Count; i++)
310     {
311         var landmarksList = framesLandmarks[quadro][i];
312         foreach (var point in landmarksList.Keys.ToList())
313         {
314             // Recalcular cada posicao com base no centro do "
esqueleto"
315             landmarksList[point] -= bodyCenter;
316         }
317     }
318 }
319 // Iniciar a corotina de animacao das esferas
320 StartCoroutine(AnimateSpheres());
321 }
322
323 // INICIO DO CODIGO PARA ANIMACAO DAS ESFERAS DENTRO DO AMBIENTE
VIRTUAL
324
325 // Atualizar as posições das esferas em cada quadro
326 IEnumerator AnimateSpheres()
327 {
328     while (true)
329     {
330         // Atualizar a posição de cada esfera durante cada frame
331         for (int i = 0; i < spheres.Count; i++)
332         {
333             // Pegar a posicao correta de cada esfera dentro de um
quadro
334             var currentLandmark = framesLandmarks["quadro_" +
currentFrame][i];
335             foreach (var point in currentLandmark)
336             {
337                 // Inserir a esfera na sua posicao correta salva

```

```

338         spheres[i].transform.position = new Vector3(point.
Value.x, point.Value.y, point.Value.z);
339     }
340 }
341
342     // Atualizar a posição e escala dos cilindros com base nas
conexões específicas
343     for (int i = 0; i < connections.Count; i++)
344     {
345         var (indexA, indexB) = connections[i]; // Buscar na
lista as duas esferas conectadas
346         Vector3 start = spheres[indexA].transform.position; //
Iniciar um vetor na primeira esfera
347         Vector3 end = spheres[indexB].transform.position; //
Iniciar um vetor na segunda esfera
348
349         // Colocar um cilindro no meio das duas esferas
selecionadas por esta conexao
350         cylinders[i].transform.position = (start + end) / 2;
351
352         // Selecionar o comprimento do cilindro com base na
distancia entre as duas esferas
353         float distance = Vector3.Distance(start, end);
354         cylinders[i].transform.localScale = new Vector3(
355             Mathf.Min(spheres[indexA].transform.localScale.x *
0.5f, spheres[indexB].transform.localScale.x * 0.5f),
356             distance / 2, //Comprimento
357             Mathf.Min(spheres[indexA].transform.localScale.z *
0.5f, spheres[indexB].transform.localScale.z * 0.5f)
358         ); // Diametro baseado em metade do tamanho da esfera
menor
359
360         // Selecionar a rotacao do cilindro de forma que a
primeira esfera esteja conectada com a segunda
361         cylinders[i].transform.rotation = Quaternion.
FromToRotation(Vector3.up, end - start);
362     }
363
364     // Aguardar o próximo quadro
365     yield return new WaitForSeconds(0.1f); // Ajuste a
velocidade da animação
366
367     // Avançar para o próximo quadro
368     currentFrame = (currentFrame + 1) % framesLandmarks.Count;
369 }
370 }
371 }

```