

Trabalho de Conclusão de Curso

Implementação de um par de luvas de baixo custo como um dispositivo de entrada para sistemas de realidade virtual

de Pedro Henrique Silva de Almeida

orientado por
Prof. Dr., UFAL Ícaro Bezerra Queiroz de Araújo

Universidade Federal de Alagoas
Instituto de Computação
Maceió, Alagoas
11 de Junho de 2025

UNIVERSIDADE FEDERAL DE ALAGOAS
Instituto de Computação

IMPLEMENTAÇÃO DE UM PAR DE LUVAS DE BAIXO CUSTO COMO UM DISPOSITIVO DE ENTRADA PARA SISTEMAS DE REALIDADE VIRTUAL

Trabalho de Conclusão de Curso submetido ao Instituto de Computação da Universidade Federal de Alagoas como requisito parcial para a obtenção do grau de Engenheiro de Computação.

Pedro Henrique Silva de Almeida

Orientador: Prof. Dr., UFAL Ícaro Bezerra Queiroz de Araújo

Banca Avaliadora:

Tiago Alves de Almeida Prof. Dr., UFAL
Andressa Martins Oliveira Msc.

Maceió, Alagoas
11 de Junho de 2025

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico
Bibliotecária: Lívia Silva dos Santos – CRB-4 – 1670

A447i Almeida, Pedro Henrique de.
Implementação de um par de luvas de baixo custo como um dispositivo de entrada para sistemas de realidade virtual / Pedro Henrique de Almeida .– 2025.
71 f. : il. color.

Orientador: Ícaro Bezerra Queiroz de Araújo.
Monografia (Trabalho de Conclusão de Curso em Engenharia de Computação) – Universidade Federal de Alagoas. Instituto de Computação. Maceió, 2025.

Bibliografia: f. 43- 46.
Apêndice: f. 47-55
Anexo: f. 56 -71

1. Realidade virtual - Luvas. 2. Sistemas embarcados. 3. Tecnologia – Sensores flexíveis ópticos. 3. Sensores flexíveis elétrico – Luvas. I. Título.

CDU: 004.896

UNIVERSIDADE FEDERAL DE ALAGOAS
Instituto de Computação

IMPLEMENTAÇÃO DE UM PAR DE LUVAS DE BAIXO CUSTO COMO UM DISPOSITIVO DE ENTRADA PARA SISTEMAS DE REALIDADE VIRTUAL

Trabalho de Conclusão de Curso submetido ao Instituto de Computação da Universidade Federal de Alagoas como requisito parcial para a obtenção do grau de Engenheiro de Computação.

Aprovado em 11 de Junho de 2025:

gov.br Documento assinado digitalmente
ICARO BEZERRA QUEIROZ DE ARAUJO
Data: 16/06/2025 16:20:58-0300
Verifique em https://validar.iti.gov.br

Ícaro Bezerra Queiroz de Araújo,
Prof. Dr., UFAL, Orientador

gov.br Documento assinado digitalmente
TIAGO ALVES DE ALMEIDA
Data: 16/06/2025 16:27:49-0300
Verifique em https://validar.iti.gov.br

Tiago Alves de Almeida,
Prof. Dr., UFAL

gov.br Documento assinado digitalmente
ANDRESSA MARTINS OLIVEIRA
Data: 17/06/2025 10:35:05-0300
Verifique em https://validar.iti.gov.br

Andressa Martins Oliveira,
Msc.

Dedicatória

Aos meus pais, por me criarem bem e me ajudarem a chegar até aqui;
À minha namorada, por acreditar e me apoiar durante todo o processo de escrita do trabalho.

Agradecimentos

Ao professor Ícaro, por ter sido o primeiro professor a acreditar no meu potencial e por toda sua confiança e paciência;
Ao Grupo AI-SPARC, por me guiar e auxiliar no meu crescimento enquanto engenheiro e pesquisador;
Aos meus amigos Thiago Laurentino e Rangel por toda ajuda ao longo da execução do projeto;
Aos meus amigos Manoel, Sátiro, Lívia, Tavares, Wykthor, Wyvian, Ycaro, Natalia e a todos os outros que não lembrei de citar mas me tranquilizaram ou me apoiaram de alguma forma, por tornar a jornada da graduação mais leve.

Resumo

O uso das mãos como dispositivos de entrada para computadores é um tópico debatido há décadas, mas mudanças no mercado em meados dos anos 2000 reduziram o interesse por implementações deste tipo de ferramenta. Com o crescente investimento em aplicações de Realidade Virtual, a necessidade de luvas como meio de interação e *feedback* ao usuário faz-se necessário em aplicacionais educacionais e até mesmo de uso geral. Este trabalho detalha o processo de confecção de uma luva para RV (Realidade Virtual) de baixo custo, com *feedback* háptico, integrando-a com *softwares* bem difundidos no mercado (Blender e Unity) e expandindo a literatura acerca da produção de dispositivos deste tipo.

***Palavras-chave*: *Luvas de Realidade Virtual*; *Sistemas Embarcados*; *Feedback Háptico*.**

Abstract

The utilization of hands as input devices for computers has been a topic of discussion for decades; however, market shifts in the early 2000s diminished interest in implementing such tools. With the growing investment in Virtual Reality applications, the necessity for gloves as a means of user interaction and feedback becomes evident in educational applications and even general use. This work details the process of crafting a low-cost VR glove with haptic feedback, integrating it with widely adopted software on the market, and expanding the literature on the development of such devices.

***Keywords*: *Virtual Reality Gloves*; *Embedded Systems*; *Haptic Feedback*.**

Lista de Figuras

1.1 Exemplos de HMDs criados durante a década de 60. A figura (a) apresenta o primeiro HMD não-interativo e sem *motion tracking*, enquanto a figura (b) Apresenta o primeiro HMD para RV conectado a um computador com motion tracking. Obtido de (1) 13
1.2 Exemplos de aplicações do VIDEOPLACE. 14
1.3 Modelos desenvolvidos por Tom Furness. Na figura (a), o *display* é acoplado em um visor disposto na frente de um dos olhos, forçando o cérebro a misturar o mundo real com a imagem virtual. Em (b), a imagem é projetada na viseira do capacete. 14
1.4 Biblioteca MediaPipe em uso. Obtida de (2) 15

2.1 HMDs classificados como 3DOF. 19
2.2 Meta Quest 2. Fonte: Autor 19
2.3 Placa com o módulo IMU BNO08x acoplado no centro. O conjunto do acelerômetro, magnetômetro e giroscópio, além de um processador Cortex M0, são integrados em um componente que mede $3.8mm \times 5.2mm$. A versão exibida na imagem é adaptada para trabalhar com conectores STEMMA, uma versão sem solda para protocos de transmissão que usam 4 pinos. 20
2.4 Capturas de tela do sistema *Virtual Vietnam*. 20
2.5 Sayre Glove. Obtida de (3) 21
2.6 Nintendo Power Glove. Obtida de (4) 22
2.7 CyberGlove III. Obtida de (5) 22
2.8 Luvas vendidas pelas empresas SenseGlove e Manus. 23
2.9 A luva HaptX G1 possui conexões para um dispositivo chamado Airpack, uma mochila responsável por prover *feedback* háptico e táctil com o uso de ar, que podem ser vistas na figura (a). Na figura (b), vemos o funcionamento deste dispositvo que ativou alguns atuadores microfluídicos voltados para os dedos do usuário. 24
2.10 25

2.11 A bHaptics TactGlove DK2 possui 6 atuadores linear ressonante para *feedback* ao usuário, autonomia de bateria de 3.5h, comunicação Bluetooth Low Energy (BLE) e pesa de 110g a 116g a depender do tamanho escolhido. 25

3.1 LucidVR Gloves. 27
3.2 Diagramas informativos sobre potênciometros. A figura (a) apresenta seus componentes e (b) ilustra o comportamento de um potencriômetro linear. Obtidas de (6) 28
3.3 Potenciômetro do tipo Green. Fonte: Autor. 28
3.4 Servomotor MG90S: Fonte: Autor. 29
3.5 Placa ESP32-WROOM. Obtida de (7) 29
3.6 Componentes para montagem do módulo háptico. Fonte: Autor. 31
3.7 Diagrama de conexões elétricas para as LucidVR Gloves usando a ESP32. Adaptado de (8). 34

4.1 Luva completamente construída e vestida por um usuário. A construção com os módulos hápticos pode ser vista à esquerda, quanto a construção sem os módulos pode ser vista à direita. 36
4.2 Programa feito na linguagem Python para decodificar dados produzidos pela LucidVr Glove e encaminhá-los via conexão Socket. Fonte: Autor 37
4.3 Diagrama de arquitetura do *software* baseados em entidades e conexões das mesmas. Fonte: Autor. 38
4.4 Capturas de tela do *software* Blender com o cenário montado para aplicação. A figura (a) foi feita sem a execução do cliente socket, enquanto as figuras (b), (c) e (d) foram feitas após inicialização do mesmo. 39
4.5 Kit completo contendo par de luvas para uso. Contém dois cabos micro-usb e dois suportes para controles meta quest. 40

Lista de Tabelas

3.1 Parâmetros do sistema térmico analisado 30
3.2 Configurações da impressora 3D 31
3.3 Tabela de configurações 35

Lista de Símbolos

Ω	Ohms, medida de resistência elétrica.
baud	Taxa de transmissão, considerando bits por segundo.
U$	Dólar americano.
mAh	Miliampère-hora, capacidade energética de uma bateria.
mm	Milimetros.
ms	Milissegundos.
R$	Real Brasileiro.
¥	Yene, moeda japonesa.

Lista de Abreviaturas

API	*Application Programming Interface*.
CAD	*Computer Aided Design*, Design assistido por computador.
DoF	*Degrees of Freedom*, graus de liberdade.
HMD	*Head Mounted Display*, Display montado em uma cabeça.
IA	Inteligência Artificial.
IC	Instituto de Computação.
ML	*Machine Learning*.
RV	Realidade Virtual.
UFAL	Universidade Federal de Alagoas.

Sumário

1 Introdução **13**

1.1 Objetivos . . . 16

1.2 Estrutura do texto . . . 16

2 Fundamentação teórica **18**

2.1 Classificação de dispositivos baseadas em graus de liberdade . . . 18

2.1.1 Dispositivos 3DOF . . . 19

2.1.2 Dispositivos 6DOF . . . 19

2.2 Luvas para Realidade Virtual e suas tecnologias . . . 20

2.2.1 Luvas desenvolvidas com sensores flexíveis ópticos . . . 21

2.2.2 Luva desenvolvida com sensor flexível elétrico . . . 22

2.3 Dispositivos modernos para Realidade Virtual . . . 23

3 Metodologia **26**

3.1 LucidVR Gloves . . . 26

3.2 Componentes utilizados . . . 27

3.2.1 Potênciômetro . . . 27

3.2.2 Servomotor . . . 28

3.2.3 ESP32 . . . 29

3.2.4 Listagem final de componentes . . . 29

3.3 Processo de montagem . . . 30

3.3.1 Impressão 3D . . . 30

3.3.2 Módulo háptico . . . 31

3.3.3 Base rígida . . . 32

3.3.4 Guias . . . 33

3.4 Conexão dos fios . . . 33

3.5 Configuração do *firmware* e *softwares* compatíveis . . . 33

4 Resultados e discussões **36**

5 Conclusão **41**

Bibliografia **43**

Apêndice **47**

A Códigos utilizados no projeto **48**

B Instruções de uso **55**

Anexo **56**

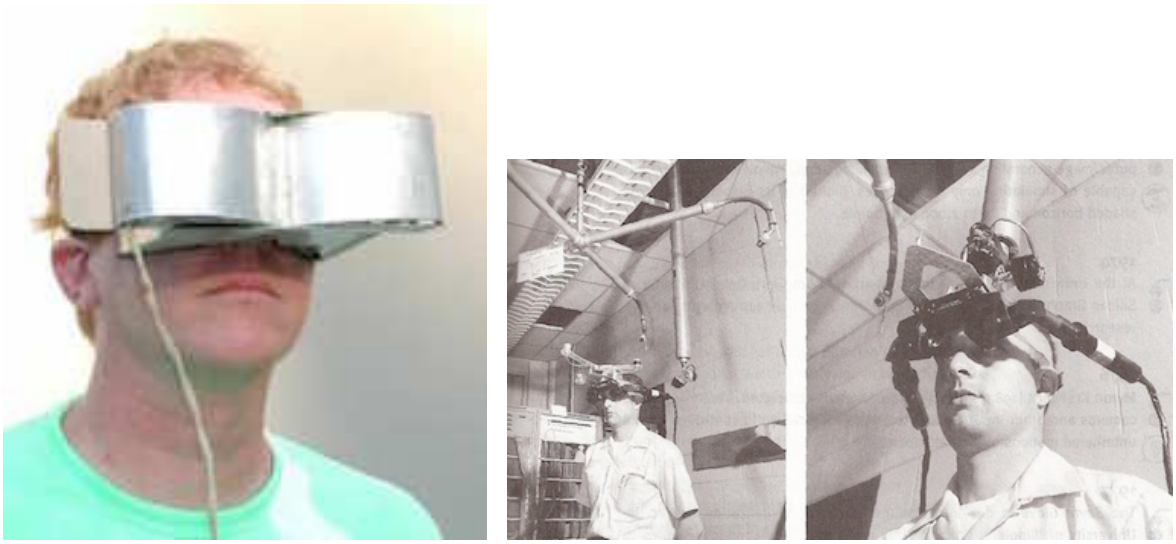
A Códigos do *firmware* para ESP32-WROOM **57**

B Instruções de uso **71**

Capítulo 1

Introdução

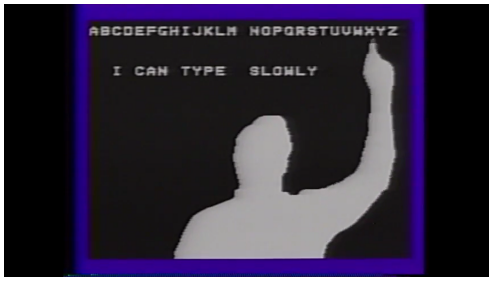
A Realidade Virtual (RV) é definida pelo uso de modelagens e simulações computacionais que permite ao usuário interagir com um ambiente artificial tridimensional usando apenas a visão e/ou outros sentidos. As primeiras implementações desta tecnologia consistiam em *Head Mounted Displays* (HMDs), dispositivos que possuem telas dedicadas para cada olho com alguma forma de visualização tridimensional, ou disposições de monitores visando maior imersão dos usuários. Alguns exemplos de implementações desses visores são o Sensorama, feito pelo cinegrafista Morton Helig durante a década de 50 e patenteado durante os anos 60, e a Sword of Damocles. (1).



(a) Telesphere mask. (b) Sword of Damocles.

Figura 1.1: Exemplos de HMDs criados durante a década de 60. A figura (a) apresenta o primeiro HMD não-interativo e sem *motion tracking*, enquanto a figura (b) Apresenta o primeiro HMD para RV conectado a um computador com motion tracking. Obtido de (1)

A tecnologia considerada como primeiro sistema de RV interativo foi criada pelo artista Myron Krueger e usava uma mistura de cameras, telas, projeções de luz e computação gráfica para determinar a posição do usuário e permitir que o mesmo interagisse com o ambiente.



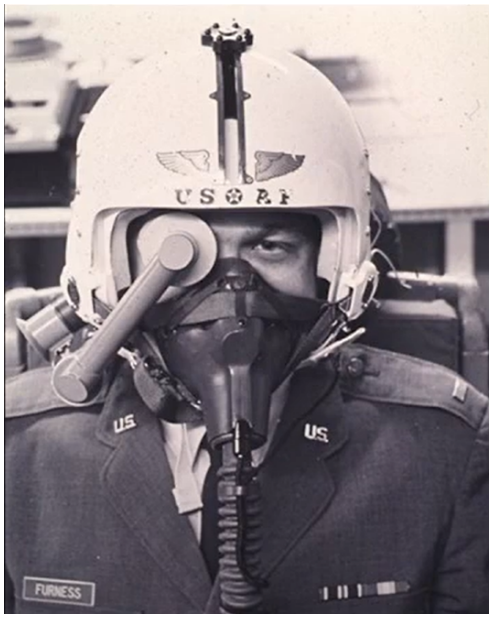
(a) Entrada de texto. Obtida de (9)



(b) Entrada dedesenho com os dedos. Obtida de (10)

Figura 1.2: Exemplos de aplicações do VIDEOPLACE.

Outras aplicações iniciais de Realidade Virtual foram voltadas a simuladores aéreos usando displays, HMDs e controles similares ao manche de um avião visando trazer uma maior imersão aos usuários com finalidade de treiná-los. Vale ressaltar que o aspecto educacional se manteve desde o ínicio, pois um parcela considerável do mercado investe em RV para treinamentos, mas o uso das mãos como dispositivos de interação não foi uma abordagem escolhida inicialmente, tendo em vista que todas as invenções anteriores focavam no uso da cabeça ou de outros controles como interface de entrada.



(a) *Helmet Mounted Display*. Obtida de (11)

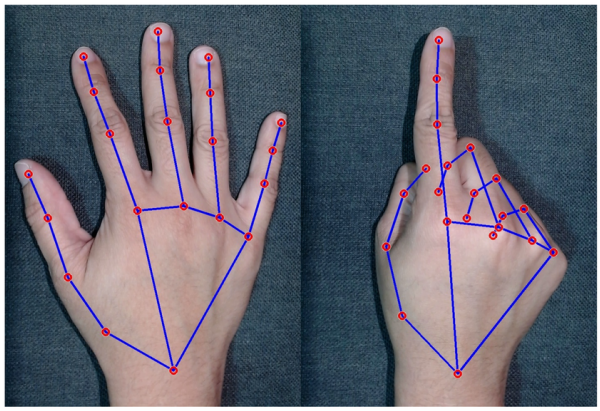


(b) *Visor Projected Display*.Obtida de (11)

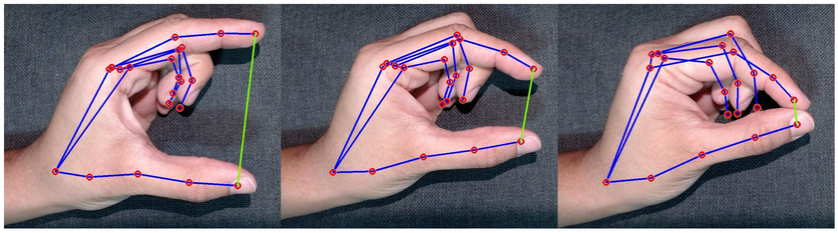
Figura 1.3: Modelos desenvolvidos por Tom Furness. Na figura (a), o *display* é acoplado em um visor disposto na frente de um dos olhos, forçando o cérebro a misturar o mundo real com a imagem virtual. Em (b), a imagem é projetada na viseira do capacete.

Uma outra forma de modelar mãos como dispositivos de entrada em implementações de baixo custo seria com o uso de *webcams* ou câmeras dedicadas, junto de bibliotecas de processamento de imagem. Um exemplo seria a biblioteca MediaPipe, um projeto de código aberto

provido pela Google que oferece um conjunto de bibliotecas e ferramentas para aplicação de tecnologias Inteligência Artificial (IA) e *Machine Learning* (ML) voltadas para diversos usos. No campo de visão computacional esta biblioteca possui funcionalidades de *Hand Tracking*, que detectam posição, orientação e até mesmo gestos específicos que um usuário realiza.



(a) Gesto estático.



(b) Gesto dinâmico.

Figura 1.4: Biblioteca MediaPipe em uso. Obtida de (2)

Apesar do baixo custo o uso dessa tecnologia está sujeito a limitações, pois ao utilizar apenas uma câmera perde-se noção de profundidade e a implementação de soluções multi-câmeras tem seu custo computacional maior pois deve-se mapear o mesmo objeto em diferentes ângulos e fazer a equivalência de pontos. Independente da quantidade de câmeras utilizadas, este tipo de solução sofre interferências causadas por artefatos de imagem e por fenômenos físicos, como oclusão.

A implementação de Luvas como dispositivos de entrada possui aplicações em áreas como educação (12), saúde, tanto em contextos educacionais (13) quanto em contextos práticos (14) e também em teleoperação de dispositivos não populados (15), entre outras. Apesar do mercado de Realidade Virtual (RV) moderno ter sua ascenção em 2014, o ínicio deste tipo de tecnologia começou há mais de quatro décadas. A primeira implementação de luvas para RV foi a luva Sayre, em 1982. Desde então, Algumas outras implementações de tecnologias para *hand tracking* foram disponibilizadas no mercado, mas o usuário comum não tinha acesso pois este tipo de produto sempre foi muito caro.

Atualmente, devido ao estado da arte desta área de pesquisa, os dispositivos denominados *Head Mounted Devices* (HMDs) possuem funcionalidades de *Hand Tracking* com o auxílio de

suas câmeras, fator que proporciona um grau maior de interatividade e imersão ao usuário. Mas além das problemáticas citadas anteriormente existem limitações no escopo de desenvolvimento de aplicações, pois as políticas de desenvolvimento não disponibilizam acesso à essas camêras para terceiros em seus sistemas nativos.

Existem luvas voltadas para aplicações de Realidade Virtual, mas os problemas que existem desde a luva Sayre ainda persistem: Os preços são muito altos e muitas destas novas implementações são voltadas ao escopo empresarial, sendo vendidas apenas para setores de manufatura que podem realizar treinamentos com o auxílio deste tipo de tecnologia.

O presente trabalho visa detalhar o funcionamento e a implementação já existente de uma luva para RV de baixo custo e a sua utilização em aplicativos de computador que são executados em um *headset* RV, termo definido como *Personal Computer Virtual Reality* (PCVR).

No contexto de trabalhos científicos, existem publicações relacionadas à produção de dispositivos com funcionamento análogo, a exemplo de Freire (16), que apresenta uma luva para interação com estruturas moleculares, e do modelo proposto por Shigapov (17), que não possui aplicação específica mas ilustra o processo de criação de uma luva digital de baixo custo. Cada um dos trabalhos citados possui vantagens e limitações, pois o dispositivo construído em Freire (16) baseia-se no uso de tecidos condutivos e um dispositivo externo para mapeamento de posição enquanto Shigapov (17) indica o uso de sensores flexíveis e giroscópios, de modo que o aumento de precisão dos dois modelos é condicionado a um aumento de custo dos seus materiais usados e nenhum dos artigos institui feedback ao usuário, fatores a serem considerados no processo de escolha de um dispositivo a ser reproduzido.

1.1 Objetivos

O presente trabalho tem como objetivos:

- Documentar a construção de uma luva RV de baixo custo, baseado em um projeto *open source* já existente.
- Identificar pontos de melhora no projeto, visando simplificar a reprodutibilidade do mesmo sem implicações significativas no custo.
- Propor alternativas ao uso convencional do projeto, que depende de *softwares* externos para o funcionamento, e montar um protótipo de código aberto.

1.2 Estrutura do texto

No capítulo 2 são apresentados alguns conceitos chave a serem referenciados ao longo do trabalho, além de elaborar a sua questão central. A fundamentação teórica contém um modelo de classificação de dispositvos RV com base nos sensores usados, seguido de uma linha do

tempo referente ao desenvolvimento de luvas para RV baseada em seus princípios de funcionamento, encerrando com uma disposição do estado da arte nesta área.

O capítulo 3 descreve o projeto *open source* no qual este trabalho se baseia e seus princípios de funcionamento, expõe os materiais utilizados e lista suas respectivas quantidades utilizadas e preços. Em seguida, todo o seu processo de confecção é detalhado, visando facilitar a reprodução deste modelo.

O capítulo 4 expõe o produto resultante do capítulo anterior e ilustra seus usos, abordando o *software* fornecido pelo autor original e a plataforma compatível com o mesmo, seguindo para uma implementação em python que mescla comunicação serial e udp.

O capítulo 5, por fim, traz um apanhado geral dos resultados e conclui com indicações de passos futuros.

Capítulo 2

Fundamentação teórica

Este capítulo aborda os fundamentos necessários para o desenvolvimento de luvas como dispositivos de entrada em Realidade Virtual (RV), começando pela classificação dos sistemas com base em seus graus de liberdade (DoF), que determinam os movimentos detectáveis pelos dispositivos. Em seguida, é traçada uma linha histórica da evolução das luvas para RV, desde os primeiros protótipos ópticos até modelos comerciais. Por fim, discute-se o estado atual do mercado, evidenciando a escassez de soluções acessíveis para o público geral, o alto custo dos modelos comerciais com *feedback* háptico e o surgimento de alternativas promissoras em fase de prototipagem por meio de financiamento coletivo.

2.1 Classificação de dispositivos baseadas em graus de liberdade

O conceito de grau de liberdade ou *Degrees of freedom* (DoF) aplicado à RV consiste na quantidade de maneiras que um corpo pode se mover e interagir com um espaço tridimensional (18). Desta forma, tem-se dois tipos de movimentos aplicados aos eixos de largura, altura e profundidade, sendo eles a rotação e a translação. Desta forma, tem-se seis movimentos:

- Mover-se para esquerda ou direita;
- Mover-se para cima ou para baixo;
- Mover-se para frente ou para trás
- Rotacionar ao redor do eixo x (rolamento ou *roll*);
- Rotacionar ao redor do eixo y (inclinação ou *pitch*);
- Rotacionar ao redor do eixo z (guinada ou *yaw*) (19).

Partindo deste princípio, existem duas classificações para dispositivos RV, descritas a seguir.



(a) Google Cardboard.



(b) Oculus go.

Figura 2.1: HMDs classificados como 3DOF.

2.1.1 Dispositivos 3DOF

Os três graus de liberdade destacados são referentes ao movimento de rotação, de modo que HMDs mais simples entram neste tipo de categoria. Com o uso de giroscópios, é possível detectar a inclinação da cabeça do usuário, fazendo com que dispositivos nesta configuração sejam ideais para visualizar vídeos 360 ou experiências de interação simples (20). Alguns exemplos deste tipo de tecnologia são o Google Cardboard e o Oculus Go, dois headsets de Realidade Virtual que foram descontinuados.

2.1.2 Dispositivos 6DOF

Dispositivos deste tipo realizam todos os movimentos citados anteriormente, ou seja, um *headset* RV com seis graus de liberdade detecta tanto as rotações de cabeça quanto o movimento do corpo em um ambiente (21). Um famoso exemplo é o Meta Quest, que desde sua segunda versão possui a funcionalidade de delimitar um ambiente de movimentação sem obstáculos no qual o usuário pode andar e mantém sua localização relativa durante todo o seu uso.



Figura 2.2: Meta Quest 2. Fonte: Autor

No contexto da criação de dispositivos RV, existem sensores vendidos com nove graus de liberdade, chamados de *Inertial Measurement Unity* (IMU). Uma IMU é composta de acelerômetros, giroscópios e magnetômetros, cada um com três graus de liberdade. Apesar desses

sensores serem mais precisos, Os equipamentos que usam este tipo de tecnologia ainda são considerados como 6DOF, pois é o máximo de movimentos a serem feitos em um espaço tridimensional. A linha de IMUs BNO0x se tornou famosa no campo devido à precisão dos dados enviados, além de *features* adicionais como envio de vetores de rotação otimizados e relatórios de classificação de estabilidade, passos, tremores e atividade(22).

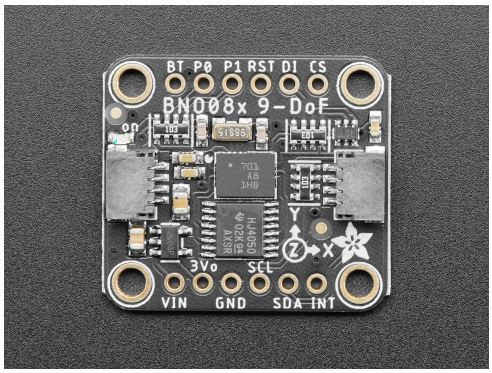


Figura 2.3: Placa com o módulo IMU BNO08x acoplado no centro. O conjunto do acelerômetro, magnetômetro e giroscópio, além de um processador Cortex M0, são integrados em um componente que mede $3.8mm \times 5.2mm$. A versão exibida na imagem é adaptada para trabalhar com conectores STEMMA, uma versão sem solda para protocos de transmissão que usam 4 pinos.

2.2 Luvas para Realidade Virtual e suas tecnologias

O uso de luvas como interface de interação entre usuário e aplicações RV foi conceituado há cinco décadas (23) e chegou no seu ápice com as Power Gloves. Durante a década de 90 uma série de fracassos comerciais envolvendo dispostivos RV de empresas como Atari, Sega e até mesmo a própria nintendo(4) desmotivou o investimento nessa área como meio de entretenimento. Apesar de projetos em outras áreas continuarem, como um sistema de terapia em RV chamado *Virtual Vietnam* (24), A RV entrou em declínio até 2010, quando Palmer Lucky criou um Kit de RV que se tornou o Oculus Rift (1).



(a)



(b)

Figura 2.4: Capturas de tela do sistema *Virtual Vietnam*.

Há cerca de uma década novos modelos de luvas surgiram no mercado com foco de venda para empresas de grande porte e instituições educacionais, mas a criação de modelos por parte de usuários nunca deixou de existir. Esta seção compila achados dos principais modelos e as tecnologias usadas em cada um, visando compreender o processo de manufatura para dispositivos similares.

2.2.1 Luvas desenvolvidas com sensores flexíveis ópticos



Figura 2.5: Sayre Glove. Obtida de (3)

Desenvolvida em 1977 pelo Eletronic Visualization Laboratory (EVL)(3), as luvas usam um tubo flexível com uma fonte de luz em uma extremidade e um sensor fotoelétrico na extremidade oposta. A quantidade de luz recebida pelo sensor indicava flexão dos dedos. Como se tratava de um protótipo, um produto final nunca chegou ao mercado, mas a técnica de sensoriamento gerou uma patente em 1982 e foi utilizada em projetos posteriores, que chegaram ao mercado.

A combinação dos sensores desenvolvidos por Thomas G. Zimmerman incorporados com o trabalho de Jaron Lanier, autor responsável por popularizar o termo 'Realidade Virtual', resultou na criação de duas novas luvas: Z-Glove e DateGlove (25). Ao contrário da versão anterior, esses dispositivos contavam com monitoramento de posição e orientação, além de *feedback* táctil fornecido através de vibração. A implementação dos sistemas de orientação e posição usou transdutores de ultrasom para a Z-Glove e um sistema de campos magnéticos com baixa frequência para a DateGlove. A pesquisa foi conduzida através da empresa VPL Research, considerada a primeira empresa a vender luvas RV e *Head Mounted Devices*(HMDs) (1).

Além dos dispositivos físicos, a VPL também desenvolveu aplicativos de computador que reconheciam os sinais enviados pelas luvas e exibia animações equivalentes em modelos tridimensionais, além de mapear entradas de movimento do mundo real à comandos executados por um usuário de computador com o *mouse*.

Após a publicação desta pesquisa, surge a primeira luva disponível para usuários comuns: A Nintendo Power Glove.



Figura 2.6: Nintendo Power Glove. Obtida de (4)

Apesar desta luva não ter sido amplamente difundida a todos os consumidores, pois tinha apenas dois jogos compatíveis, é um marco considerável e inspirou construções de luvas por usuários independentes ao extraírem os sensores da power glove (26).

Outro experimento com contribuições advindas da DateGlove foi o projeto *Virtual Interface Environment Workstation* (VIEW), consistindo em um simulador de treino em RV para astronautas. O experimento começou apenas com um HMD e teve a luva integrada ao seu sistema, possibilitando que usuários interagissem com cenários produzidos por computador.

2.2.2 Luva desenvolvida com sensor flexível elétrico



Figura 2.7: CyberGlove III. Obtida de (5)

Lançada em 1990, a CyberGlove é uma luva instrumentada que focou em obter ângulos das juntas através de sensores flexíveis e exibir esses dados numa renderização em tempo real. O produto ainda é vendido no mercado até hoje, direcionado à usuários que precisem realizar captura de movimentos. Esta luva não mede posição ou orientação das mãos, mas fornece os dados com *timestamps*, representações numéricas ou textuais que indicam pontos específicos no tempo, para sincronização das entradas de outros dispositivos, como câmeras e roupas especializadas para *motion capture*(mocap) (5). O sensor empregado nesta luva funciona alterando o valor da resistência conforme o usuário dobra os dedos, atribuindo um valor de resistência para cada ângulo de dobra independente do sentido.

2.3 Dispositivos modernos para Realidade Virtual

Parte dos HMDs modernos possuem câmeras integradas e as utilizam com diversas finalidades, a exemplo de mapeamento de ambientes (27), experiências em Realidade Aumentada (RA) e *hand tracking*. Algumas vantagens de usar este tipo de tecnologia são: Redução de custos, pois não é necessário usar um controle específico; Simplificação de interfaces de interação, pois as funcionalidades são implementadas em *Software Development Kits* (SDKs), que são coleções de ferramentas, bibliotecas e documentação que auxiliam e orientam desenvolvedores na construção de aplicações com uma determinada tecnologia, e segurança, pois ao embarcar funcionalidades em seus SDKs é possível limitar o acesso à câmera para aplicações verificadas e limitar o acesso aos dados da câmera por terceiros. Por outro lado, O sensoriamento por imagens é propenso a erros em ambientes com iluminação muito baixa ou muito intensa e também pode ser prejudicado devido ao fenômeno de oclusão, limitando os movimentos do usuário ao campo de visão das câmeras.

Apesar de uma busca crescente por parte de entusiastas da RV, não existem muitas luvas no mercado para consumidores comuns a um preço acessível. Os modelos que estão à venda são voltados a empresas e instituições educacionais que visam investir em treinamentos com o auxílio da RV, a exemplo da SenseGlove Nova 2 (28), que possui *feedback* de contato ativo na palma das mãos e vibrotáctil para os dedos, além de *finger tracking* baseado em sensores e algortimos de processamento de imagem, ao passo que a mesma também pode ser integrada com *tracking* externo para HMDs que não permitem o uso de câmeras, vendida a 5999 euros[1].



(a) SenseGlove Nova 2.



(b) Algumas opções de luvas RV vendidas pela Manus. A empresa vende 7 modelos distintos de luvas, um conjunto *trackers* de posição focado em *full body tracking* e acessórios complementares para uso destes produtos.

Figura 2.8: Luvas vendidas pelas empresas SenseGlove e Manus.

Outras empresas focam em diversificar seus produtos e suas aplicações, a exemplo da Manus (29), que comercializa uma gama de luvas e dispositivos vestíveis para *body tracking* com propósitos genéricos, como *motion capture* (mocap), aplicações em pesquisas e treinamentos ou simulações na área da robótica. O outro ponto de destaque é que os itens vendidos são compatíveis com o Robot Operating System (ROS), além das principais *Game Engines* como Unity

[1] https://www.senseglove.com/product/nova-2/

e Unreal. Outra empresa que possui um modelo de mercado parecido é a HaptX, mas o seu diferencial é o tipo de atuador de *feedback* empregado, já que seus produtos focam em pontos infláveis distribuídos ao longo das suas luvas, de modo a simular toque realista e interação com diferentes texturas ao longo da mão, como é possível ver no modelo G1 ([30]). Não é possível fazer afirmações concretas quanto ao preço das luvas de ambas empresas já que compras destes produtos só são concluídas mediante orçamento.

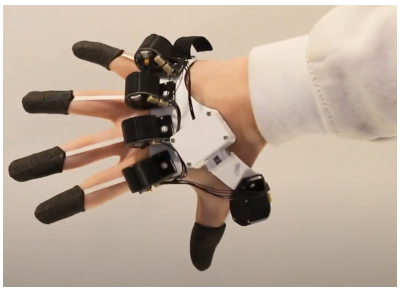


(a) (b)

Figura 2.9: A luva HaptX G1 possui conexões para um dispositivo chamado Airpack, uma mochila responsável por prover *feedback* háptico e táctil com o uso de ar, que podem ser vistas na figura (a). Na figura (b), vemos o funcionamento deste dispositvo que ativou alguns atuadores microfluídicos voltados para os dedos do usuário.

Por outro lado existem protótipos sendo desenvolvidos através de financiamento coletivo, se mostrando mais acessíveis em termos de preço. Alguns exemplos de destaque são a Bifrost Pulse ([31]), que traz *feedback* háptico realista e *finger tracking* em um único produto por U299 mas ainda está em estágio de prototipagem e não possui atualizações há mais de um ano, a luva UDCAP([32]), que não possui *feedback* mas apresenta tracking completo com uso de IMUs e sensores elásticos proprietários ao custo de ¥5130, que equivale a R$$\approx 204,38$, e por fim a Contact Glove([33]), que se mostra um pouco mais cara do que as outras campanhas citadas, mas ainda está bem abaixo do valor de mercado, vendendo opções com ou sem *feedback* háptico por ¥94400 e ¥65000 respectivamente.

Dentre os modelos pesquisados, o mais próximo dos requisitos propostos por este trabalho é desenvolvido pela empresa bHaptics ([34]). Foram considerados aspectos como facilidade de uso, peso, compatibilidade com PCVR, existência de algum *feedback* ao usuário e preço mais baixo ou igual em comparação a produtos similares. A TactGlove cumpre todos os requisitos citados mas custo ainda é considerado alto, considerando que o mesmo é equivalente a metade do valor de um óculos RV Quest 3, vendido pela Meta ([35]).



(a) Bifrost Pulse.



(b) Udexreal Udcap



(c) Contact Gloves

Figura 2.10



Figura 2.11: A bHaptics TactGlove DK2 possui 6 atuadores linear ressonante para *feedback* ao usuário, autonomia de bateria de $3.5h$, comunicação Bluetooth Low Energy (BLE) e pesa de $110g$ a $116g$ a depender do tamanho escolhido.

Capítulo 3

Metodologia

O estudo do uso de luvas como dispositivos de entrada abrange uma gama diversa de áreas, como saúde, robótica e educação. Aplicações que testam os efeitos destes dispositivos normalmente usam produtos já estabelecidos no mercado, que possuem um custo elevado e podem se tornar um fator de impedimento da reprodutibilidade de experimentos. Portanto, a produção de projetos *open source* geralmente são considerados disruptivos na comunidade de Realidade Virtual. Isto ocorre também pelo fato das tecnologias associadas a este mercado serem encapsuladas em produtos finais, fornecendo acesso limitado ou nulo a *Software Development Kits* (SDKs). O projeto escolhido para ser representado destacou-se pelo seu baixo custo e facilidade de montagem. Outros modelos existem e foram publicados na internet, mas eles exigem outras habilidades mais complexas, como costura, e os sensores utilizados são um pouco mais caros.

3.1 LucidVR Gloves

A luva LucidVR foi desenvolvida por Lucas de Bonet em 2021 ([36]) e foi um marco para a comunidade devido ao seu baixo custo de produção, sua integração com diversos HMDs e sua compatibilidade com diversos jogos de RV vendidos através da plataforma Steam. Esta última característica é possível pois os *drivers* escritos para estas luvas são compatíveis com a SteamVR, uma ferramenta para reprodução de conteúdos em RV compatível com os mais famosos HMDs vendidos no mercado: Meta Quest, HTC Vive, Windos Mixed Reality, entre outros.

Cada luva é composta de 5 servomotores, 5 potenciômetros, 1 ESP32-WROOM, 5 molas retráteis e 5 fios de nylon. Os componentes eletrônicos são acoplados a peças feitas com uma impressora 3D, onde o conjunto de uma mola, um fio, um potênciometro e um servomotor se transforma em um módulo háptico para cada dedo. Esses módulos podem ser dispostos com o auxílio de uma base rígida, também impressa em 3D, ou de elásticos, fornecendo uma alocação mais flexível para o usuário.

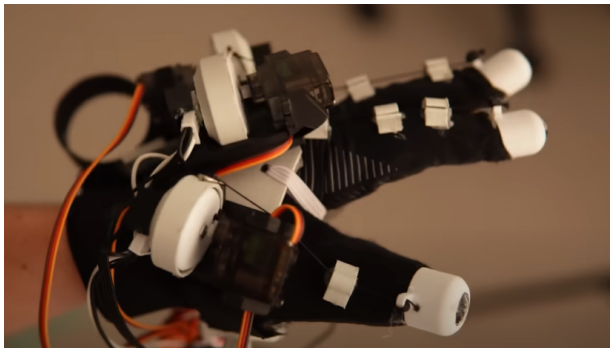


Figura 3.1: LucidVR Gloves.

O cálculo de posição de cada módulo se baseia no princípio utilizado por dispositivos com cabos retráteis: Ao puxar o fio, a mola se contrai e o potênciomentro ligado ao fio e à mola gira, obtendo uma rotação em graus. Por outro lado, caso o fio seja solto a mola retorna à posição inicial, desfazendo o movimento feito pelo potênciometro. Para mapear o funcionamento do módulo ao movimento dos dedos é necessário que o fio esteja fixado nas extremidades da luva e próximo às falanges, pois o driver associa a mudança de ângulo do potenciômetro ao movimento de abrir ou fechar a mão completamente.

Os servomotores servem como *feedback* háptico para o usuário, onde uma força emitida pelos mesmos faz o restante do módulo travar o seu movimento, simulando a sensação de segurar um objeto com as mãos e trazendo mais realismo às experiências RV. Os fios acoplados aos potênciometros e aos servomotores são conectados à ESP32 que fica responsável por ler a posição dos potênciometros e enviar sinais elétricos aos servomotores, caso seja necessário. É possível conectar a ESP ao computador com o auxílio de cabos ou via Bluetooth com o auxílio de uma bateria para manter o sistema ligado, mas visando redução de custos a implementação feita neste trabalho não usou baterias. A seguir, cada dispositivo eletrônico terá uma explicação mais detalhada do seu funcionamento.

3.2 Componentes utilizados

3.2.1 Potênciômetro

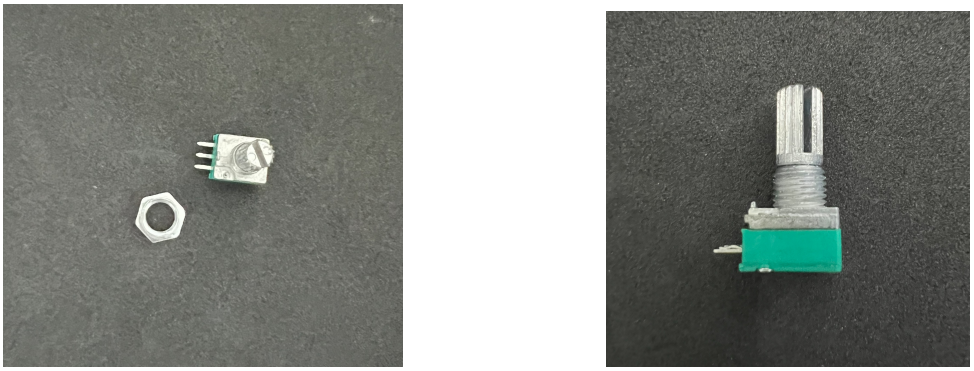
Potenciômetros ou *encoders* são componentes passivos formados por um elemento resistivo, normalmente chamado de trilha, um cursor móvel e 3 pinos, associados a entrada, aterramento e sinal. Estes terminais são controlados pelo cursor, normalmente associado a um eixo rotativo. O movimento cursor permite alteração total da resistência do componente para um determinado valor, implicando na mudança do valor de tensão que passa sobre o mesmo. O termo linear, quando usado para classificar este tipo de componente, indica a escala usada para associar a posição do cursor ao valor de resistência, possibilitando mapear os valores de tensão às posições de um *encoder* usando esta mesma escala.



(a) (b)

Figura 3.2: Diagramas informativos sobre potênciometros. A figura (a) apresenta seus componentes e (b) ilustra o comportamento de um potencriômetro linear. Obtidas de (6)

Para o projeto foi usado um potenciômetro linear rotativo de 10 $k\Omega$ que atua como um divisor de tensão, diminuindo a tensão enviada pela ESP32 e obtendo uma leitura da nova tensão resultante. Durante o processo de calibração, que será detalhado em uma seção futura, registra-se os valores de tensão para a mão aberta e mão fechada, mapeando assim o movimento da mão de forma acurada. Com o auxílio de uma impressora 3D, foi impressa uma peça com encaixe para acoplar uma porca na qual o encoder será rosqueado. Esta peça também possui um encaixe para a extremidade mola retrátil, que será enrolada e terá sua outra extermidade encaixada no cursor.



(a) Vista do topo. (b) Vista lateral.

Figura 3.3: Potenciômetro do tipo Green. Fonte: Autor.

3.2.2 Servomotor

O servomotor é o elemento responsável pelo *feedback* háptico da Luva. Normalmente, controles aplicados a contextos de jogos e simuladores utilizam *buzzers* como forma de simular interações com o usuário, mas no contexto de uma aplicação para RV faz-se necessária uma resposta mais sofisticada. O motor escolhido foi o MG90S, que fornece até $19,6\ N$ de força e possui um Ângulo de rotação de 180º.

O servomotor é acoplado ao módulo háptico com o auxílio de 3 parafusos, onde dois são usados para fixação e o terceiro é parafusado ao exterior da bobina que contém o fio de nylon e

já está acoplada ao *encoder*. No contexto definido, o motor exerce um torque na direção inversa ao movimento de fechamento das mãos, resultando na ilusão de segurar algum objeto.

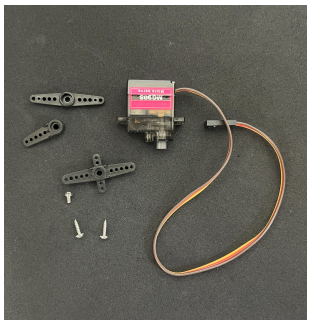


Figura 3.4: Servomotor MG90S: Fonte: Autor.

3.2.3 ESP32

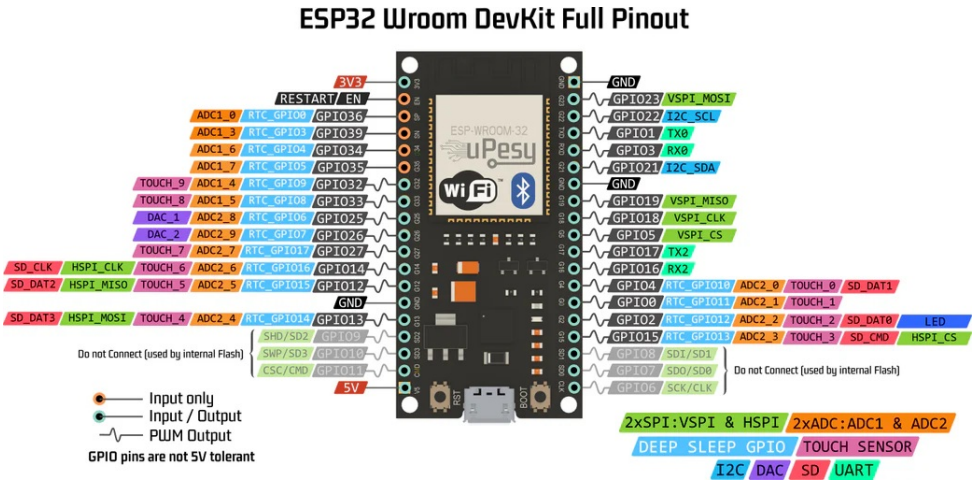


Figura 3.5: Placa ESP32-WROOM. Obtida de ([7])

A ESP32 pode ser definida como uma série de microcontroladores de baixo custo e baixo consumo de energia. Seu uso em projetos de eletrônica é extremamente recomendado devido a vasta gama de modelos e a grande quantidade de interfaces para periféricos que o mesmo possui, como conexão UART, PWM, Ethernet, Barramento CAN, entre outros.

O modelo usado no projeto se chama ESP32-WROOM, uma placa que contém uma *Micro Controller Unit* (MCU) e uma antena que possibilita conexões Wi-Fi, Bluetooth e *Bluetooth Low Energy* (BLE). O *firmware* inserido na placa foi escrito através da linguagem C e é possível carregar os arquivos na placa através do programa *Arduino IDE*.

3.2.4 Listagem final de componentes

A tabela a seguir lista os componentes usados, sua quantidade, e o valor unitário e valor total pago, desconsiderando a adição de impostos. As compras foram feitas em janeiro de 2025

através da plataforma Aliexpress.

Tabela 3.1: Parâmetros do sistema térmico analisado

Descrição	Quantidade	Valor unitário	Valor total
ESP32-WROOM	2 unidades	R$18,77	R$37,54
Crachá retrátil	10 unidades	R$2,639	R$26,39
Servomotor MG90S 180º	10 unidades	R$8,198	R$81,98
Potenciômetro green 10K Ohm	10 unidades	R$1,758	R$17,58
Kit de terminais p/ crimpagem	1 unidade	R$17,52	R$17,52
Kit fios 26AWG-120 pcs	1 unidade	R$8,59	R$8,59
Par de luvas de algodão	1 unidade	R$5,00	R$5,00
TOTAL	**R$194,60**		

Considerando o dólar como a moeda padrão para comparativo de preços neste mercado e assumindo a cotação de 1 $USD \approx 5,7$ *BRL*, o custo de material das luvas sem os impostos é de aproximadamente 33,26 dólares.

3.3 Processo de montagem

Esta seção guia a confecção de um par de luvas para Realidade Aumentada de baixo custo tomando o vídeo tutorial das Lucid Gloves como base. Para construir esse projeto, é necessário ter conhecimentos básicos em eletrônica visando facilitar a compreensão dos esquemáticos de conexão dos componentes às placas ESP32. Saber usar softwares de *Computer aided design* (CAD) é desejável, pois grande parte dos componentes serão impressos e alguns deles devem ser feitos sob medida. É possível seguir com esse projeto sem este conhecimento, aumentando os custos de produção com a contratação de serviços de impressão 3D.

3.3.1 Impressão 3D

Antes de começar o processo de montagem, é necessário definir qual tipo de potenciômetro que será usado. É possível fazê-lo usando potênciometros do tipo green ou do tipo WL, mas o primeiro é recomendado por ser mais compacto e resistente. Em seguida, é necesário imprimir os arquivos listados abaixo:

- 10 Tampas para os dedos (*finger caps*) em tamanhos variados;
- 10 Seguradores de potenciômetros compatíveis com o modelo escolhido;
- 10 Tensionadores compatíveis com o modelo de potênciometro escolhido;
- 10 Carretéis (*spool*);
- 10 Coberturas de carretel (*spool cover*);

- 18 Nós guia (*Guide nodes*);
- 2 Bases rígidas de montagem (*rigid mount*), uma para cada mão;
- 2 Bases de montagem para o controle usado em conjunto com a luva.

As configurações recomendadas para a impressora 3D são mostradas na tabela abaixo. Não é necessário usar suportes, mas abas (*brims*) são aceitas em peças que precisam de sustentação.

Tabela 3.2: Configurações da impressora 3D

Configuração	**Valor**
Nozzle diameter	0.4
Infill	10-20%
Layer height	0.2mm
Shell thickness	0.8mm (2 *outlines*)
Retraction settings	5mm

As peças foram construídas de modo a ser possível dispor quantas unidades a impressora for capaz de produzir, com base no seu tamanho, mas o ideal seria imprimir cada uma delas de forma isolada, visando maior qualidade estrutural. A peça denominada endcap ou ”tampa”dos dedos deve ter seu tamanho ajustado, e a recomendação do autor é fazer esse projeto através de tentativa e erro. É possível usar instrumentos de precisão, como paquímetros ou fitas métricas, para ajustar o arquivo ao tamanho ideal e reduzir a quantidade de impressões indesejadas.

3.3.2 Módulo háptico

Este módulo é composto das seguintes peças: Um carretel, um tensionador, uma mola plana espiral, um fio de nylon, um potenciômetro, uma proteção para o carretel, um servomotor e uma peça denominada de segurador, que será a base do módulo háptico. O carretel possui um encaixe para a mola e outro para o contato deslizante do potenciômetro. A proteção do carretel possui um furo, permitindo a passagem do fio de nylon.

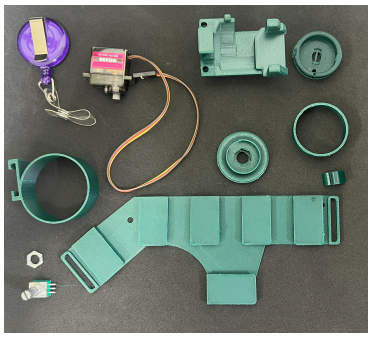


Figura 3.6: Componentes para montagem do módulo háptico. Fonte: Autor.

Inicialmente, é necessário acoplar o potênciomentro ao tensionador. Para isso, usa-se uma porca, acoplando-a no lado interno do tensionador. Em seguida, basta rosquear o potenciômetro

pelo lado externo até que o mesmo esteja bem fixado. Em seguida, extraímos a mola e o fio do crachá retrátil. Como o mesmo não possui parafusos, será necessário quebrá-lo. Agora é necessário fixar a mola no encaixe lateral, passá-la pela abertura ao lado deste encaixe e enrolar a mesma, de modo que ela se mantenha na parte interna do tensionador. A extremidade restante deve ser acoplada à cavidade do potenciômetro.

Com o tensionador totalmente montado, monta-se o carretel. Este processo é feito usando o fio extraído do crachá. É necessário cortar uma de suas pontas, que está presa a um plástico. A outra extremidade já contém um nó, que será usado para impedir que o fio escape do carretal. Com o auxílio de um clipe, é necessário passar o fio pelo carretel de modo que o mesmo fique com o nó na parte externa. Agora basta encaixar o carretel no tensionador e enrolar o fio, checando se o mesmo não está no sentido contrário de compressão da mola. Não é necessário enrolar o fio totalmente, pois o mesmo ainda deve ser preso às guias que serão coladas à luva.

Para finalizar a montagem inicial do módulo háptico, é necessário passar a capa do carretel pelo fio, atentando-se ao lado que possui o encaixe. Este lado deve ser encaixado no tensionador. Se este último processo for feito corretamente, deve-se ouvir um barulho de clique. Em seguida, basta acoplar o tensionador ao segurador através do encaixe modelo para a base do potenciômetro. A base também possui um espaço para acoplar o servomotor e cavidades para que o mesmo seja parafusado, mas este procedimento será feito após a calibração.

3.3.3 Base rígida

É necessário que esta luva tenha um componente responsável por acoplar os 5 módulos hápticos à mão do usuário. Uma opção oferecida pelo autor é usar elásticos, o que reduz o custo das impressões 3D, mas deixa a estrutura geral mais frágil e propensa ao desgaste dos mesmos ao longo tempo. O uso de uma base rígida resolve esta problemática, mas é necessário checar o tipo de filamento utilizado, pois a base será dobrada ao redor da mão e materiais como PLA podem quebrar, por se tratar de um objeto com baixa espessura. Os materiais usados para construir a base rígida são: Uma folha de espuma EVA, tiras de eslástico, tiras de velcro e as peças impressas correspondentes.

A espuma EVA é usada para fornecer mais uma camada maleável entre a mão do usuário e a estrutura rígida que forma a luva. Para isso, é necessário cortá-la no mesmo formato de cada uma das bases e colá-la usando cola quente. A luva possui 6 encaixes deslizantes, sendo 5 deles verticais. Estes encaixes são destinados aos módulos hápticos, enquanto o encaixe horizontal é destinado para acoplar suportes aos controles ou *trackers* de posição. Antes de prosseguir com o encaixe dos módulos é necessário cortar as tiras de velcro e de elástico, que serão responsáveis por impedir que a base se mova ou desgrude da luva sem proporcionar desconforto ao usuário.

O tamanho ideal é de aproximadamente 30cm para a tira de velcro e 15cm para a tira de elástico. A base possui 2 furos em sua parte frontal, onde o elástico deve ser passado com um auxílio de uma chave de fenda. Em seguida, um nó deve ser feito para manter o elástico fixo. A

tira de velcro é colocada através das abas laterais, mantendo o lado mais macio voltado para a espuma, evitando desconfortos ao usuário.

3.3.4 Guias

Os nós de guia são pequenas peças destinadas a manter o fio de nylon rente às falanges dos dedos. para isso, é necessário colar um nó por falange, com exceção das pontas dos dedos, que receberão tampas com um encaixe ajustado. Para colar cada um dos nós, é ideal vestir as luvas e fazer marcações visando atingir maior precisão. Em seguida, é recomendado usar uma chave de fenda, marcador ou qualquer objeto que simule a circuferência de um dedo e prosseguir com a colagem dos nós. O procedimento é similar com as tampas para cada dedo, mas é crucial testar o encaixe antes de prosseguir com a colagem.

3.4 Conexão dos fios

Após a montagem da luva, é necessário realizar a ligação dos fios de energia, sinal e terra referentes aos potenciômetros e aos servomotores. É possível soldar os fios diretamente à esp32, mas a mesma possui apenas um pino de alimentação com 3.3V, então é necessário fazer alguns ajustes para que todos os potenciômetros sejam alimentados. O processo é análogo para os pinos de força dos servomotores e para o aterramento de todos os componentes. Visando maior conforto e facilidade de manutenção, foi usada uma placa de prototipagem em conjunto com alguns *headers*, de modo a criar barramentos para alimentação e aterramento, além de fornecer conexões diretas para os fios de sinal. Para os potenciômetros, foi realizada a solda de cabos com um *jumper* de três vias para facilitar a conexão com os *headers*. Desta forma, os *jumpers* ficarão fixos aos *headers* e a ESP32 é acoplada por cima das conexões, encapsulando toda a eletrônica entre a mesma e a placa universal. O diagrama das conexões feitas neste projeto é apresentado na imagem abaixo:

A alimentação para os componentes eletrônicos virá da conexão fornecida pela porta USB do computador, mas é possível usar uma fonte de alimentação externa, como baterias ou um *power bank* com conexões distintas para cada voltagem.

3.5 Configuração do *firmware* e *softwares* compatíveis

O *firmware* disponibilizado é de código aberto e permite ao usuário modificá-lo completamente, sendo possível mudar a ordem dos pinos, alocar mais componentes eletrônicos como botões e *joysticks*, escolher o protocolo de comunicação usado pela ESP32, personalizar o nome do dispositivo, entre outras funcionalidades específicas. Antes de compilá-lo, é necessário efetuar o *download* do *driver* de conexão USB para UART disponibilizado pela Silicon Labs(37), além de de efetuar o download do programa Arduino IDE.

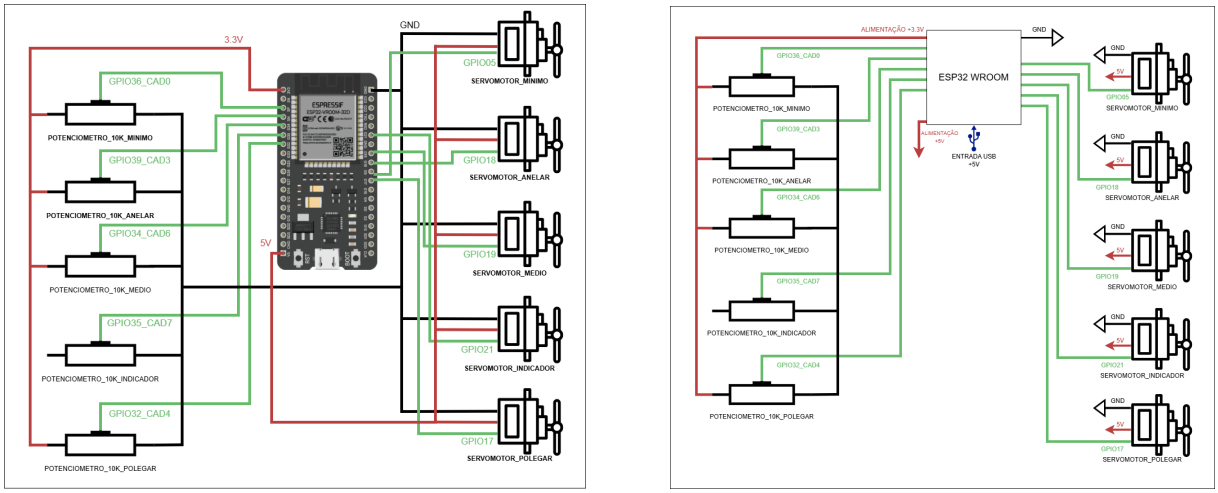


Figura 3.7: Diagrama de conexões elétricas para as LucidVR Gloves usando a ESP32. Adaptado de (8).

Com a interface do Arduino instalada e executada, é necessário baixar algumas bibliotecas específicas para que a mesma possa compilar o código da forma correta. Inicialmente é necessário adicionar uma URL do gerenciador de placas da ESP32, fornecido pela Espressif(38), na aba de preferências localizada no canto superior esquerdo. Em seguida, na aba de ferramentas, deve-se acessar a seção de placas disponíveis e em seguida acessar o item 'gerenciador de placas'. Um menu de busca será aberto e nele é possível buscar pela placa ESP32 e instalar os arquivos necessários.

A instalação do gerenciador de placas da ESP32 é responsável por fornecer predefinições de todos os modelos associados a esta linha, sendo necessário buscar pelo modelo mais aproximado ao da placa utilizada na montagem do projeto. A escolha de uma placa diferente também influencia nos pinos usados, então recomenda-se usar o modelo ESP32-WROOM e escolher a opção 'ESP32-WROOM DA Module'. Por fim, é necessário adicionar a biblioteca 'esp32servo' através da seção 'Gerenciar bibliotecas', também presente na aba 'Ferramentas'.

Por fim, é necessário validar algumas configurações antes de compilar o código. A tabela abaixo indica as variáveis a serem revistas e explica suas respectivas funções. A única configuração alterada durante a montagem deste projeto indica o sentido invertido dos potenciômetros, pois os mesmos foram invertidos durante a construção dos módulos hápticos.

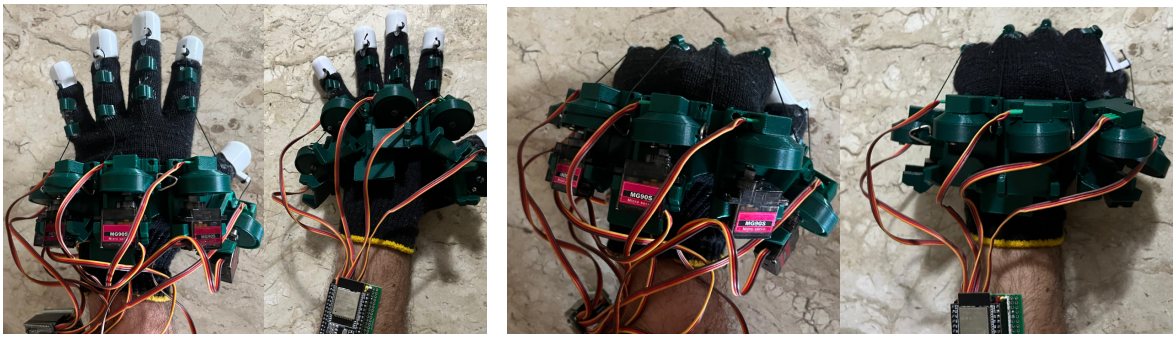
Após a revisão dos parâmetros, o processo de carregamento do código pode ser iniciado. Basta conectar a ESP32 na porta USB, checar a porta COM habilitada para a mesma e então iniciar o processo de carregamento através do botão 'Carregar', presente no canto superior esquerdo da tela, logo abaixo da aba 'Arquivo'. Ao término do carregamento é possível monitorar a comunicação serial pelo Arduino IDE e checar a validade dos valores recebidos.

Tabela 3.3: Tabela de configurações

Variável	**Valor usado**	**Descrição**
COMMUNICATION	COMM_SERIAL	Define o protocolo a ser usado. Para ativar conexão bluetooth, usa-se o valor COMM_BTSERIAL.
BTSERIAL_DEVICE_NAME	”lucidgloves-left”ou ”lucidgloves-right”	Nome associado ao dispositivo a ser configurado. É necessário inserir um nome para cada mão.
FLIP_POTS	true	Indica se os potenciômetros foram invertidos no processo de montagem.
USING_FORCE_FEEDBACK	true	Indica se a luva usará *feedback* háptico através dos servomotores.
PIN_CALIB	0	Indica qual pino será usado para ativar o processo de calibração.
USING_CALIB_PIN	false	Indica se o processo será ativado ao apertar o botão de calibração.

Capítulo 4

Resultados e discussões



(a) (b)

Figura 4.1: Luva completamente construída e vestida por um usuário. A construção com os módulos hápticos pode ser vista à esquerda, quanto a construção sem os módulos pode ser vista à direita.

O capítulo anterior descreve os materiais e processos necessários para construir uma luva RV de baixo custo e comunicação serial com o computador. Assim que o processo de compilação é finalizado a comunicação serial com o computador já é inicializada, de modo que a ESP32 envia Strings separadas por quebra de linha e contendo um conjunto de informações de cada potenciômetro identificado por uma letra maiúscula com um valor inteiro referente à sua resistência atual escrito ao lado. O conjunto de informações enviadas varia conforme a configuração do driver, pois é possível acoplar joysticks e botões de interação e de simulação de gestos. A escolha do padrão de mensagens escolhido se baseia no SDK chamado OpenVR, desenvolvido pela empresa Valve, de modo a garantir compatibilidade das LucidVR Gloves com todos os aplicativos compatíveis com este kit de desenvolvimento. A mesma empresa também desenvolveu o aplicativo SteamVR, que possui compatibilidade com a maior parte dos HMDs vendidos no mercado, e uma *Application Programming Interface* (API) de acesso ao OpenVR com o mesmo nome, disponível para uso na *Game Engine* Unity, tornando-a ideal para criar experiencias em RV com foco no uso destas luvas como dispositivos de entrada pois as mesmas serão reconhecidas independente das suas escolhas de configuração e do HMD usado, além de ser possível acoplá-las com *trackers* externos, a exemplo do HTC Vive. A comunicação entre

a luva e todos os aplicativos citados é garantida através de um driver chamado OpenGloves, provido por um dos contribuidores do projeto das luvas, e através dele que é feita a calibração de posição das mãos e do uso dos servomotores, além dos testes de *feedback* háptico e de conexão Bluetooth, caso a mesma seja utilizada.

Por outro lado, tendo em vista que o projeto utilizado é *open-source*, implementações proprietárias podem ser feitas visando descentralizar o uso das Lucid Gloves com as plataformas citadas anteriormente. A comunicação serial pode ser usada de forma isolada ou com outros protocolos, permitindo seu uso em outros softwares ou até mesmo em navegadores, com o auxílio de conexão via *socket* e uma devida interface de tratamento dos dados. O exemplo de código abaixo, escrito na linguagem Python, mostra o processo de extração dos dados da luva e do seu envio para uma conexão *socket* via *User Datagram Protocol* (UDP).

serial_bridge.py

```python
import serial
import socket
import re

SERIAL_PORT = 'COM5'
BAUD_RATE = 115200
HOST = '127.0.0.1'
PORT = 65432

ser = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=1)

with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as sock:
    while True:
        try:
            line = ser.readline().decode('utf-8').strip()
            if line:
                matches = re.findall(r'([A-Z])(\d+)', line)
                data = {label: int(value) for label, value in matches if label in 'ABCDE'}
                message = ",".join(f"{k}:{v}" for k, v in data.items())
                print(message)
                sock.sendto(message.encode(), (HOST, PORT))
        except Exception as e:
            print("Erro:", e)
```

Figura 4.2: Programa feito na linguagem Python para decodificar dados produzidos pela LucidVr Glove e encaminhá-los via conexão Socket. Fonte: Autor

O software de Modelagem 3D Blender possui suporte ao uso de *scripts* e possui uma API na linguagem Python, que permite o uso de conexão socket para recebimento dos dados e permite tratamento e uso dos mesmos dentro do programa. O código exibido abaixo exemplifica o uso das LucidGloves como uma dataglove dentro do Blender usando conexão socket e sua API. A aplicação utiliza um cenário pré-montado contendo um corpo animável com duas mãos e segmentos curvos definidos como caminhos para cada dedo seguir. O código é responsável por buscar os objetos de interesse, denominados de ossos, e definir o uso cinemática inversa nos mesmos, onde a restrição seriam os caminhos predefinidos. Por fim, uma *thread* é criada para manter a conexão ativa e invocar as funcionalidades de tratamento e aplicações dos dados recebidos. A arquitetura do sofware é indicada na figura 4.3.

O código presente na figura 4.2 foi adaptado para exibir o tempo entre envio dos pacotes, visando medir o tempo entre transmissão de pacotes via conexão serial e testar a eficiência da luva. O tempo entre cada log de envio variou entre 16ms e 17ms. O firmware da luva foi programado com uma taxa de envio de 115200$baud$ e cada pacote enviado mede exatamente 100 bytes ou 800 bits, permitindo estimar o intervalo de tempo ideal para o envio de cada pacote através

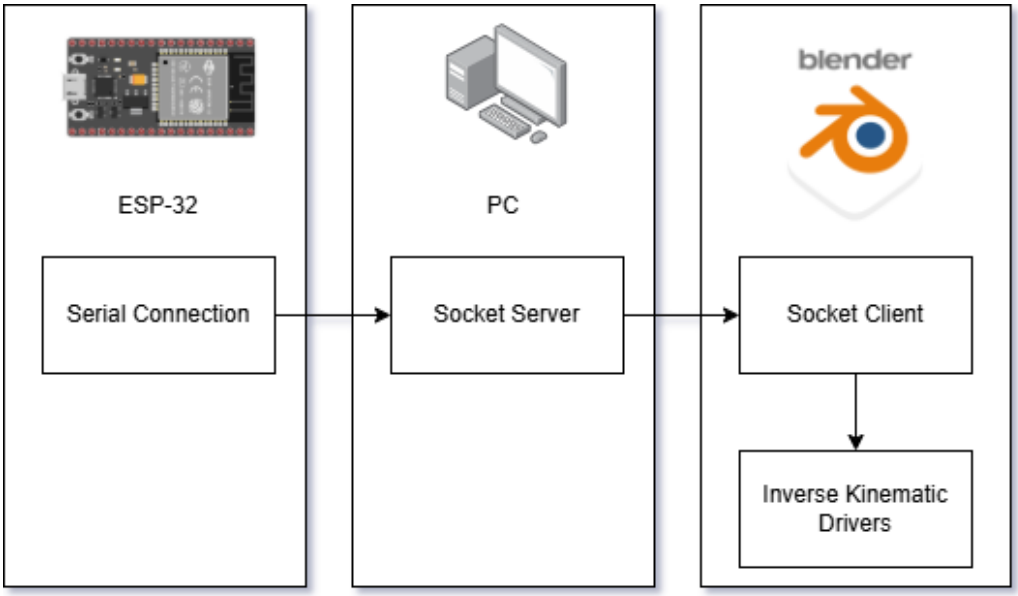
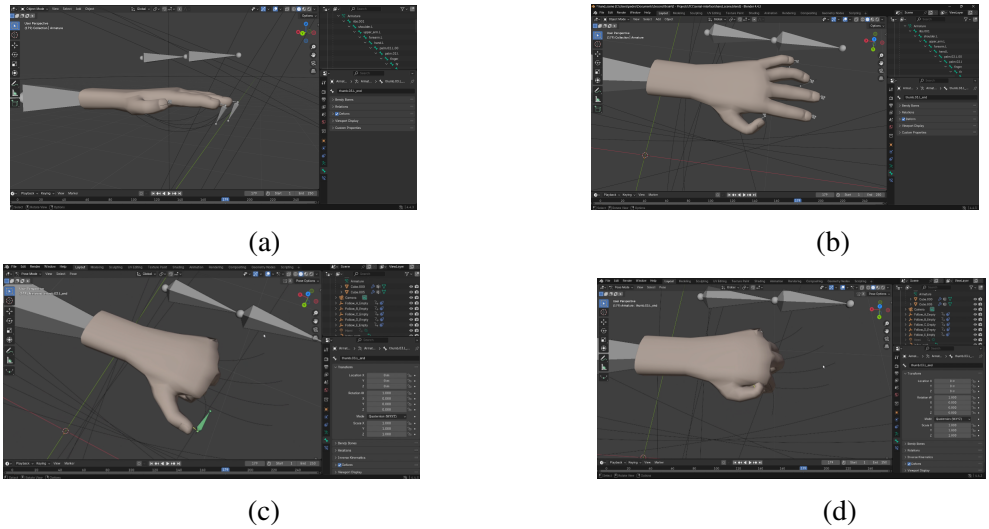


Figura 4.3: Diagrama de arquitetura do *software* baseados em entidades e conexões das mesmas. Fonte: Autor.

do quociente entre a quantidade de bits por pacote e a *baud rate*, que resulta em $\approx 6.94ms$. Apesar do tempo médio de envio ser maior que o dobro do valor ideal, não há impactos signficativos em sua performance. Outras métricas de tempo, como o intervalo para transmissão do pacote socket, não foram medidas por se tratar de uma implementação feita usando conexão local, tornando-as irrelevantes no resultado final. O programa pode ser considerado *plug-and-play*, pois não são necessários muitos ajustes para executá-lo, basta informar a porta USB utilizada, executar os scripts de transmissão e recebimento de dados em seus respectivos programas e caso a luva esteja conectada via USB, já estará pronta para uso.

A luva LucidVR se mostra ideal para experimentos e prototipagens, mas é possível implementar melhorias em sua usabilidade e no seu design. Cada luva tem um peso de $\approx 280g$, mas a obrigatoriedade do uso de um controle ou *tracker* acoplado ao mesmo aumenta consideravelmente o seu peso, fato que pode prejudicar a imersão do usuário (14). Durante a fase de testes foi utilizado o óculos Meta Quest 2, cujos controles pesam $151g$, e o peso resultante das luvas, do controle e da placa ESP32 foi de $444g$. O posicionamento da placa também é um fator preocupante, pois o autor do guia oficial cita o uso de um invólucro específico para manter a placa presa à tira de velcro, mas além do pulso não ser uma localização ideal para manter este componente, o arquivo de impressão deste invólucro não é disponibilizado no site oficial do projeto e é necessário procurar modelos feitos por outros usuários. O uso das LucidGloves como DataGloves desobriga a necessidade dos módulos hápticos, de modo que uma versão exclusiva para esse uso diminuiria o peso total do conjunto em $\approx 67g$ por luva além de reduzir a quantidade de pinos e fios usados, abrindo espaço para designs mais compactos e focados na precisão dos dados obtidos com o uso de mais sensores. Outro item a ser melhorado envolve a exposição das conexões elétricas, fator que dificulta a usabilidade e faz com que a luva não seja visualmente



(a) (b)

(c) (d)

Figura 4.4: Capturas de tela do *software* Blender com o cenário montado para aplicação. A figura (a) foi feita sem a execução do cliente socket, enquanto as figuras (b), (c) e (d) foram feitas após inicialização do mesmo.

atrativa. Implementações de design com seus componentes totalmente cobertos podem mudar este aspecto. A Senseglove Nova 2([28]) Apresenta funcionamento similar ao da LucidGlove e possui um design mais arrojado, de modo que o uso de impressões 3d para montar invólucros inspirados na luva da SenseGlove mudariam a apresentação da LucidGlove sem impactar no seu custo de produção.

Outro contexto a ser considerado envolve o uso das LucidGloves sem o auxílio das conexões USB. A incorporação de baterias junto dos elementos citados anteriormente incrementa tanto o peso quanto o custo de produção por luva. O autor do projeto recomenda o uso de *powerbanks* de 20000mAh, que custam entre 80$R$$ a 120$R$$ e pesam em média 380g. Além disso o design final desconsidera o uso das baterias como elemento funcional, limitado a algumas soluções contidas no tutorial em vídeo do projeto. Um design inspirado na CyberGlove 3 ([5]) ou na HaptX G1([30]) traria uma proposta mais dedicada ao uso deste dispositivo com baterias.

Quanto ao uso de sensores protocolos para aumento de precisão dos dados, a placa ESP32 pode funcionar com o protocolo *Bluetooth Low Energy* (BLE), usado como forma de localizar dispositivos em um raio de até 5 metros. Pela disposição dos pinos usados, também é possível acoplar uma IMU e implementar reconhecimento dos dados do sensor para transmissão de posição via comunicação serial. A implantação de sensores 6 ou 9-DoF possibilitaria o uso das LucidGloves como dispositivos independentes, desobrigando acoplamento de controles ou trackers em alguns softwares e em HMDs que não possuem controles, trazendo novas possibilidades e fomentando a produção novos softwares e integrações que não dependam de controles para mapear posição das mãos.

O processo descrito no capítulo 3 e as considerações deste capítulo expandem o guia feito pelo autor original e sanam lacunas apresentadas ao longo do processo, além de propor uma solução compacta e robusta para a organização dos componentes eletrônicos e considerar



Figura 4.5: Kit completo contendo par de luvas para uso. Contém dois cabos micro-usb e dois suportes para controles meta quest.

alterações do modelo atual baseado em dispositivos já estabelecidos no mercado. É esperado que qualquer pessoa com conhecimento básico em eletrônica possa segui-lo e confeccionar o par de luvas com facilidade e precisão.

Capítulo 5

Conclusão

Este trabalho alcançou com êxito a implementação e documentação de um par de luvas de baixo custo como dispositivo de entrada para aplicações em Realidade Virtual, utilizando o projeto *open source* LucidVR Gloves como base. A proposta demonstrou ser tecnicamente viável, apresentando funcionalidade adequada para experimentos de interação virtual, com destaque para o uso das luvas em um ambiente independente do ecossistema padrão, centrado em programas como a Game engine Unity, a plataforma Steam e softwares focados em RV desenvolvidos para a mesma. O programa consiste no uso de uma conexão UDP para enviar dados da Luva ao Software Blender e movimentar dedos de uma mão virtual de modo análogo aos de uma mão real, com auxílio de cinemática inversa.

A partir da análise do processo de construção e configuração, foi possível identificar soluções que simplificam a montagem, tornando o projeto mais acessível a pesquisadores, desenvolvedores e entusiastas da área. A reprodução do dispositivo, conforme descrita, não exige conhecimentos avançados de engenharia, o que contribui significativamente para a democratização de tecnologias imersivas.

Adicionalmente, o trabalho expande a literatura existente ao fornecer uma abordagem sistematizada para a montagem de luvas de RV, ressaltando aspectos críticos como a escolha de componentes, a adaptação de peças impressas em 3D e a configuração de firmware compatível com diferentes ambientes de desenvolvimento.

Apesar dos resultados satisfatórios, reconhece-se algumas limitações, como a necessidade de controles ou trackers externos e o consequente aumento de peso atrelado ao par de luvas, além da ausência de sensores inerciais que poderiam melhorar a precisão posicional. Tais aspectos indicam possibilidades para desenvolvimentos futuros, como a incorporação de módulos IMU, a adoção de comunicação sem fio via Bluetooth Low Energy e a evolução do firmware para proporcionar maior flexibilidade e autonomia. Também existem limitações físicas em aspectos como comforto, facilidade de uso, requisitos funcionais e apelo visual, que poderiam ser facilmente resolvidas executando produções similares a modelos já presentes no mercado.

Conclui-se, portanto, que este trabalho contribui de forma relevante para o avanço de soluções acessíveis no domínio da Realidade Virtual, ao mesmo tempo em que abre caminhos

para investigações e aprimoramentos subsequentes, como melhorias no código produzido pelo autor e execução das propostas de design sugeridas no capítulo 4.

Bibliografia

[1] V. R. Society, "History of virtual reality." https://www.vrs.org.uk/virtual-reality/history.html. Acesso em: 10 abr. 2025.

[2] W. Torres, L. Santos, G. Melo, A. Oliveira, P. Nascimento, G. Carvalho, T. Neves, A. Martins, and Ícaro Araújo, "A framework for real-time gestural recognition and augmented reality for industrial applications," *Sensors*, vol. 24, no. 8, p. 2407, 2024. Publicado em 10 abr. 2024. Acesso em: 08 jun. 2025.

[3] E. V. Laboratory, "Sayre glove (first wired data glove)." https://www.evl.uic.edu/research/2162. Acesso em: 13 abr. 2025.

[4] I. Everden, "The history of virtual reality." https://www.sciencefocus.com/future-technology/the-history-of-virtual-reality, 2016. Acesso em: 13 abr. 2025.

[5] C. S. LLC, "Cyberglove." https://www.cyberglovesystems.com/cyberglove-iii. Acesso em: 13 abr. 2025.

[6] H. Mattede, "Potenciômetro – o que é e como funciona!." Disponível em: https://www.mundodaeletrica.com.br/potenciometro-o-que-e-como-funciona/. Acesso em: 08 jun. 2025.

[7] uPesy Electronics, "Esp32 pinout reference: Gpio pins ultimate guide," 2023. Disponível em: https://www.upesy.com/blogs/tutorials/esp32-pinout-reference-gpio-pins-ultimate-guide. Acesso em: 08 jun. 2025.

[8] LucidVR, "LucidGloves Prototype 4 ESP 32 WiringDiagram," 2025. Disponível em: https://github.com/LucidVR/lucidgloves/wiki/Prototype-4-Wiring-Diagram. Acesso em: 09 mai. 2025.

[9] MediaArtTube, "Myron krueger - videoplace, responsive environment, 1972-1990s," 2008. Publicado em: 2008-04-07. Acesso em: 08 jun. 2025.

[10] VintageCG, "Videoplace (1985)," 2009. Publicado em: 2009-08-14. Acesso em: 08 jun. 2025.

[11] Knowledge Media Design Institute - KDMI, "Kmdi speaker series: Tom furness - cobwebs in a virtual brain," 2023. Publicado em: 2021-08-30. Acesso em: 08 jun. 2025.

[12] R. Palombo, S. Weber, M. Wyszynski, and B. Niehaves, "Glove versus controller: the effect of vr gloves and controllers on presence, embodiment, and cognitive absorption," *Frontiers in Virtual Reality*, vol. Volume 5 - 2024, 2024.

[13] M. Izumi, H. Hagiya, Y. Otsuka, Y. Soejima, S. Fukushima, M. Shibata, S. Hirota, T. Koyama, F. Otsuka, and A. Gofuku, "Effectiveness of sensing gloves–applied virtual reality education system on hand hygiene practice: A randomized controlled trial," *American Journal of Infection Control*, vol. 53, no. 1, pp. 65–69, 2025.

[14] Q. Wang, B. Kang, and P. O. Kristensson, "Supporting playful rehabilitation in the home using virtual reality headsets and force feedback gloves," in *2022 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 504–513, 2022.

[15] J. Jankowski and A. G. and, "Usability evaluation of vr interface for mobile robot teleoperation," *International Journal of Human–Computer Interaction*, vol. 31, no. 12, pp. 882–889, 2015.

[16] R. Freire, B. R. Glowacki, R. R. Williams, M. Wonnacott, A. Jamieson-Binnie, and D. R. Glowacki, "Omg–vr: Open-source mudra gloves for manipulating molecular simulations in vr," 2020. Publicado em: 5 maio 2020. Acesso em: 13 mai. 2025.

[17] M. Shigapov, V. Kugurakova, and E. Zykov, "Design of digital gloves with feedback for vr," in *2018 IEEE East-West Design Test Symposium (EWDTS)*, pp. 1–5, 2018.

[18] Varjo Technologies Oy, "Degrees of freedom in vr/xr," 2025. Disponível em: https://varjo.com/learning-hub/degrees-of-freedom-in-vr-xr/. Acesso em: 09 mai. 2025.

[19] B. Lang and J. Batallé, "An introduction to positional tracking and degrees of freedom (dof)," 2013. Disponível em: https://www.roadtovr.com/introduction-positional-tracking-degrees-freedom-dof/. Acesso em: 09 mai. 2025.

[20] D. Barnard, "Degrees of freedom (dof): 3-dof vs 6-dof for vr headset selection," 2025. Disponível em: https://virtualspeech.com/blog/degrees-of-freedom-vr. Acesso em: 09 mai. 2025.

[21] Mechatech, "What is a 3 dof vs 6 dof in vr?," 2025. Disponível em: https://www.mechatech.co.uk/journal/what-is-a-3dof-vs-6dof-in-vr. Acesso em: 17 mai. 2025.

[22] Bryan Siepert, "Adafruit 9-dof orientation imu fusion breakout – bno085: Overview," 2020. Disponível em: https://learn.adafruit.com/adafruit-9-dof-orientation-imu-fusion-breakout-bno085/overview. Acesso em: 08 jun. 2025.

[23] R. Seibel and N. Rochester, "Communication devic," 1962.

[24] B. O. R. et al, "Virtual reality exposure therapy for ptsd vietnam veterans: A case study," *Journal of Traumatic Stress*, vol. 12, no. 2, pp. 263–271, 1999. Publicado em abr. 1999.Acesso em: 08 jun. 2025.

[25] T. G. Zimmerman, J. Lanier, C. Blanchard, S. Bryson, and Y. Harvill, "A hand gesture interface device," 1987. Documento disponibilizado digitalmente. Acesso em: 13 abr. 2025.

[26] A. Mulder, "How to build an instrumented glove based on the powerglove flex sensors," *PCVR Magazine*, vol. 16, pp. 10–14, 1994. Documento disponibilizado digitalmente. Acesso em: 13 abr. 2025.

[27] V. Angelov, E. Petkov, G. Shipkovenski, and T. Kalushkov, "Modern virtual reality headsets," in *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, pp. 1–5, 2020.

[28] SENSEGLOVE, "Senseglove nova." https://www.senseglove.com/product/nova-2/. Acesso em: 20 abr. 2025.

[29] Manus, "Manus : Enabling digital hands." https://www.manus-meta.com/. Acesso em: 20 abr. 2025.

[30] HaptX, "Haptx gloves g1." https://haptx.com/gloves-g1/, 2024. Acesso em: 20 abr. 2025.

[31] B. VR, "Bifrost pulse - touch beyond reality." https://www.indiegogo.com/projects/bifrost-pulse#/updates/all. Acesso em: 20 abr. 2025.

[32] Udexreal, "Udcap, silk-like vr glove for steamvr." https://www.kickstarter.com/projects/udexreal/udcap-silk-like-vr-gloves-for-steamvr. Acesso em: 20 abr. 2025.

[33] Diver-X, "Contact glove." https://www.kickstarter.com/projects/diver-x/contact-glove. Acesso em: 20 abr. 2025.

[34] bHaptics, "Tactglove dk2." https://www.bhaptics.com/tactsuit/tactglove-dk2/. Acesso em: 20 abr. 2025.

[35] Meta, "Meta quest 3: Mixed reality headset - shop now." https://www.meta.com/quest/quest-3/?srsltid=AfmBOoqe6YPYp4EJnVY6YWJXFTLTWjVYZT-DRF-t3lQ077gRPNSA9Du-. Acesso em: 20 abr. 2025.

[36] A. McEvoy, N. Murray, and C. Keighrey, "Building a low-cost immersive haptic feedback glove for virtual reality," in *2024 16th International Conference on Quality of Multimedia Experience (QoMEX)*, pp. 58–61, 2024.

[37] Silicon Labs, "CP210x USB to UART Bridge VCP Drivers." https://www.silabs.com/developer-tools/usb-to-uart-bridge-vcp-drivers?tab=downloads, 2025. Acesso em: 09 mai. 2025.

[38] Espressif Systems, "package_esp32_index.json — Índice de Pacotes para Arduino-ESP32," 2025. Disponível em: https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json. Acesso em: 09 mai. 2025.

[39] P. Almeida, "Lucid dataglove blender." https://github.com/PAlmeida-dev/lucid-dataglove-blender, 2025. Accessed: 2025-06-02.

Apêndices

Apêndice A

Códigos utilizados no projeto

Listing A.1: Servidor feito através da linguagem Python.

```
import serial
import socket
import re
import time

SERIAL_PORT = 'COM5'   # e.g., 'COM3' on Windows or '/dev/
   ttyUSB0' on Linux
BAUD_RATE = 115200
HOST = '127.0.0.1'
PORT = 65432

ser = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=0.5)

previous_time = time.perf_counter()  # high precision time calc
with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as sock:
    while True:
        try:
            line = ser.readline().decode('utf-8').strip()
            if line:
                current_time = time.perf_counter()
                delta_time = current_time - previous_time
                previous_time = current_time

                print(f"Tempo␣entre␣leituras:␣{delta_time␣*␣
                    1000:.3f}␣ms")

                matches = re.findall(r'([A-Z])(\d+)', line)
                data = {label: int(value) for label, value in
                    matches if label in 'ABCDE'}
                print(data)
                message = ",".join(f"{k}:{v}" for k, v in data.
                    items())
```

```
                sock.sendto(message.encode(), (HOST, PORT))
        except Exception as e:
            print("Erro:", e)
```

Listing A.2: Cliente *socket* a ser executado no Blender.

```
import bpy
import socket
import threading
import random

# Configura  o do socket (conforme o script externo)
HOST = '127.0.0.1'
PORT = 65432

sock = None
running = True
socket_thread = None
arm = None
setup = True
# Dicion rio global de valores dos dedos
finger_values = {'A': 0, 'B': 0, 'C': 0, 'D': 0, 'E': 0}

# Nome do objeto Armature e dos ossos
# === Configura  o geral ===
ARMATURE_NAME = "Armature"

CONTROL_BONE_MAPPING = {
    'A': 'thumb.03.L_end',
    'B': 'finger_index.03.L_end',
    'C': 'finger_middle.03.L_end',
    'D': 'finger_ring.03.L_end',
    'E': 'finger_pinky.03.L_end',
}

BONE_MAPPING = {
    'A': 'thumb.03.L',
    'B': 'finger_index.03.L',
    'C': 'finger_middle.03.L',
    'D': 'finger_ring.03.L',
    'E': 'finger_pinky.03.L',
}

CURVE_MAPPING = {
    'A': 'thumb_path',
    'B': 'index_path',
    'C': 'middle_path',
    'D': 'ring_path',
```

```
    'E': 'pinky_path',
}

PROP_MAPPING = {
    'A': 'thumb_control',
    'B': 'index_control',
    'C': 'middle_control',
    'D': 'ring_control',
    'E': 'pinky_control',
}

def lucid_dataglove_handling(data):
    global finger_values
    try:
        decoded = data.decode('utf-8')
        parts = decoded.split(',')
        for part in parts:
            label, value = part.split(':')
            finger_values[label] = min(1.0,(int(value) / 4095.0)
                )
    except Exception as e:
        print(f"Erro no socket: {e}")

# Handler que atualiza os ossos
def update_bones():
    global arm, finger_values
    for label in finger_values:
        arm[PROP_MAPPING[label]] = finger_values[label]
    arm.animation_data.drivers.update()
    return

def start_sock_thread():
    global sock, running

    def execute():
        global sock,running
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        try:
            sock.bind((HOST, PORT))
            sock.settimeout(1.0)
            print(f"Escutando em {HOST}:{PORT}")
            print("Esperando dados da luva...")
            while running:
                try:
                    data, addr = sock.recvfrom(1024)
                    print(f"dados recebidos{data}")
                    bpy.app.timers.register(lambda data=data:
```

```
                        lucid_dataglove_handling(data))
                    update_bones()
                except socket.timeout:
                    continue
                except BlockingIOError:
                    pass  # sem novos dados
                except Exception as e:
                    if running:
                        print(f"Erro ao receber dados: {e}")
        except Exception as e:
            print(f"Conex o socket falhou em {HOST}:{PORT}. 
                Error: {e}")
            print("Outra conex o pode estar usando a porta")
        finally:
            if sock:
                sock.close()
                print("Socket Fechado")
    thread = threading.Thread(target=execute, daemon=True)
    thread.start()
    self.report({'INFO'}, "Leitura UDP iniciada")
    return thread

def end_sock_thread():
    global running, sock, socket_thread
    running = False
    if sock:
        sock.close()
    if socket_thread:
        socket_thread.join(2.0)  # Wait for thread to finish,
            but not forever
    print("Conex o interrompida")



def setup_armature():
    armature = bpy.data.objects.get(ARMATURE_NAME)
    if not armature or armature.type != 'ARMATURE':
        raise ValueError(f"Objeto '{ARMATURE_NAME}' n o 
            encontrado ou n o  um Armature.")
    return armature

def setup_IK():
    global arm
    bpy.ops.object.mode_set(mode='OBJECT')
    bpy.context.view_layer.objects.active = arm
    arm.select_set(True)
    bpy.ops.object.mode_set(mode='POSE')
```

```
    for label in BONE_MAPPING:
        target_bone_name = CONTROL_BONE_MAPPING[label]
        bone_name = BONE_MAPPING[label]

        try:
            pbone = arm.pose.bones[bone_name]
            ik = pbone.constraints.new(type='IK')
            ik.target = arm
            ik.subtarget = target_bone_name
            ik.chain_count = 2 if bone_name.startswith("thumb")
                else 3
            print(f"IK␣adicionado␣em␣{bone_name}␣  ␣{
                target_bone_name}")
        except KeyError:
            print(f"Osso␣'{bone_name}'␣ou␣controlador␣'{
                target_bone_name}'␣n o␣encontrado.")
        except Exception as e:
            print(f"Erro␣ao␣adicionar␣IK␣para␣{bone_name}:␣{e}")

# === Setup Follow Path com Empty intermedi rio ===
def setup_follow_path_with_empty(label):
    global arm
    curve_name = CURVE_MAPPING[label]
    prop_name = PROP_MAPPING[label]
    display_name = prop_name.split("_")[0]
    ctrl_bone_name = CONTROL_BONE_MAPPING[label]

    curve_obj = bpy.data.objects.get(curve_name)
    if not curve_obj:
        print(f"Curva␣'{curve_name}'␣n o␣encontrada.")
        return

    # Cria propriedade customizada
    if prop_name not in arm:
        arm[prop_name] = 0.0
    if not hasattr(bpy.types.Object, prop_name):
        setattr(bpy.types.Object, prop_name, bpy.props.
            FloatProperty(
            name=display_name,
            description=f"Controle␣do␣dedo␣{display_name}",
            min=0.0,
            max=1.0,
            default=0.0
        ))

    # Recupera osso de controle
    pbone = arm.pose.bones.get(ctrl_bone_name)
```

```
    if not pbone:
        print(f"Osso␣de␣controle␣'{ctrl_bone_name}'␣n o␣
            encontrado.")
        return

    # Cria um Empty auxiliar
    empty_name = f"Follow_{label}_Empty"
    if empty_name in bpy.data.objects:
        empty = bpy.data.objects[empty_name]
    else:
        empty = bpy.data.objects.new(empty_name, None)
        bpy.context.collection.objects.link(empty)
        empty.empty_display_type = 'SPHERE'

    # Constraint Follow Path no Empty
    follow = empty.constraints.get("Follow␣Path")
    if not follow:
        follow = empty.constraints.new(type='FOLLOW_PATH')
    follow.target = curve_obj
    follow.use_curve_follow = True
    follow.use_fixed_location = True
    follow.offset_factor = 0.0

    # Driver no offset_factor
    fcurve = follow.driver_add("offset_factor")
    driver = fcurve.driver
    driver.type = 'SCRIPTED'
    var = driver.variables.new()
    var.name = label
    var.type = 'SINGLE_PROP'
    var.targets[0].id = arm
    var.targets[0].data_path = f'["{prop_name}"]'
    driver.expression = label

    # Bone segue o Empty via Copy Transforms
    copy = pbone.constraints.get("Copy␣Transforms")
    if not copy:
        copy = pbone.constraints.new(type='COPY_TRANSFORMS')
    copy.target = empty

if __name__ == "__main__":
    try:
        arm = setup_armature()
        if not setup:
            setup_IK()
            for label in CONTROL_BONE_MAPPING:
                setup_follow_path_with_empty(label)
```

```
            setup = True
        socket_thread = start_sock_thread()
    except Exception as e:
        print(f"Erro ao inicializar o listener: {e}")
```

Apêndice B

Instruções de uso

Para conseguir executar o código do servidor corretamente em primeiro lugar é necessário ajustar o valor da variável ‘SERIAL_PORT‘ no arquivo A.1. É possível visualizar a porta utilizada com auxílio do software Arduino IDE, basta clicar na placa selecionada e visualizar a porta que a mesma utiliza. Em seguida, as variáveis ‘HOST‘ e ‘PORT‘ podem ser personalizadas, desde que tenham valores iguais nos códigos A.1 e A.2. Com o uso do terminal, ao executar o comando ‘python glove_serial_read_socket_send.py‘ o servidor será aberto com os dados de *host* e a porta inseridos. É recomendado usar o Python 3.11 ou superior.

Para executar o código do cliente, é necessário ter a versão ‘4.4.3‘ do Blender. Caso o mesmo esteja sendo executado pela primeira vez em um cenário que não esteja configurado, a variável ‘setup‘ do arquivo A.2 deve ter seu valor definido como ‘False‘. Para inserir o *script* em uma cena basta slecionar a aba *scripting*, presente na parte superior da tela. O código funcionará corretamente se o cenário possuir os arquivos inseridos com os nomes listados no Apêndice A.2 e estiver com os parâmetros de conexão socket iguais aos do servidor.

Uma cópia da implementação exposta acima foi divulgada na plataforma GitHub(39), junto do cenário usado no capítulo 4.

Anexos

Anexo A

Códigos do *firmware* para ESP32-WROOM

Listing A.1: Arquivo 'main.ino'

```
#define ALWAYS_CALIBRATING CALIBRATION_LOOPS == -1

#define CALIB_OVERRIDE true
#if USING_CALIB_PIN && COMMUNICATION == COMM_SERIAL && PIN_CALIB
    == 0 && !CALIB_OVERRIDE
  #error "You␣can't␣set␣your␣calibration␣pin␣to␣0␣over␣usb.␣You␣
      can␣calibrate␣with␣the␣BOOT␣button␣when␣using␣bluetooth␣
      only.␣Set␣CalibOverride␣to␣true␣to␣override␣this."
#endif

ICommunication* comm;
int loops = 0;
void setup() {
  #if COMMUNICATION == COMM_SERIAL
    comm = new SerialCommunication();
  #elif COMMUNICATION == COMM_BTSERIAL
    comm = new BTSerialCommunication();
  #endif
  comm->start();

  setupInputs();

  #if USING_FORCE_FEEDBACK
    setupServoHaptics();
  #endif

}

void loop() {
```

```
if (comm->isOpen()){
  #if USING_CALIB_PIN
  bool calibButton = getButton(PIN_CALIB) != INVERT_CALIB;
  if (calibButton)
    loops = 0;
  #else
  bool calibButton = false;
  #endif

  bool calibrate = false;
  if (loops < CALIBRATION_LOOPS || ALWAYS_CALIBRATING){
    calibrate = true;
    loops++;
  }

  int* fingerPos = getFingerPositions(calibrate, calibButton);
  bool joyButton = getButton(PIN_JOY_BTN) != INVERT_JOY;

  #if TRIGGER_GESTURE
  bool triggerButton = triggerGesture(fingerPos);
  #else
  bool triggerButton = getButton(PIN_TRIG_BTN) !=
      INVERT_TRIGGER;
  #endif

  bool aButton = getButton(PIN_A_BTN) != INVERT_A;
  bool bButton = getButton(PIN_B_BTN) != INVERT_B;

  #if GRAB_GESTURE
  bool grabButton = grabGesture(fingerPos);
  #else
  bool grabButton = getButton(PIN_GRAB_BTN) != INVERT_GRAB;
  #endif

  #if PINCH_GESTURE
  bool pinchButton = pinchGesture(fingerPos);
  #else
  bool pinchButton = getButton(PIN_PNCH_BTN) != INVERT_PINCH;
  #endif

  bool menuButton = getButton(PIN_MENU_BTN) != INVERT_MENU;

  comm->output(encode(fingerPos, getJoyX(), getJoyY(),
      joyButton, triggerButton, aButton, bButton, grabButton,
      pinchButton, calibButton, menuButton));

  #if USING_FORCE_FEEDBACK
```

```
      char received[100];
      if (comm->readData(received)){
        int hapticLimits[5];
        //This check is a temporary hack to fix an issue with
            haptics on v0.5 of the driver, will make it more
            snobby code later
        if(String(received).length() >= 10) {
           decodeData(received, hapticLimits);
           writeServoHaptics(hapticLimits);
        }
      }
    #endif
    delay(LOOP_TIME);
  }
}
```

Listing A.2: Arquivo 'Encoding.ino'

```
/*struct inputData {
  int* flexion;
  int joyX;
  int joyY;
  bool joyClick;
  bool triggerButton;
  bool aButton;
  bool bButton;
  bool grab;
  bool pinch;
};

struct outputData{
  int* hapticLimits;
};
*/

#if ENCODING == ENCODING_LEGACY
//legacy encoding
char* encode(int* flexion, int joyX, int joyY, bool joyClick,
    bool triggerButton, bool aButton, bool bButton, bool grab,
    bool pinch, bool calib, bool menu){
  static char stringToEncode[75];

  sprintf(stringToEncode, "%d&%d&%d&%d&%d&%d&%d&%d&%d&%d&%d&%
     d\n",
  flexion[0], flexion[1], flexion[2], flexion[3], flexion[4],
  joyX, joyY, joyClick,
  triggerButton, aButton, bButton, grab, pinch
  );
```

```
  return stringToEncode;
}
//legacy decoding
void decodeData(char* stringToDecode, int* hapticLimits){
    byte index = 0;
    char* ptr = strtok(stringToDecode, "&");  // takes a list of
        delimiters
    while(ptr != NULL)
    {
        hapticLimits[index] = atoi(ptr);
        index++;
        ptr = strtok(NULL, "&");  // takes a list of delimiters
    }
}
#endif

#if ENCODING == ENCODE_ALPHA
//alphabetic encoding
char* encode(int* flexion, int joyX, int joyY, bool joyClick,
   bool triggerButton, bool aButton, bool bButton, bool grab,
   bool pinch, bool calib, bool menu){
  static char stringToEncode[75];
  int trigger = (flexion[1] > ANALOG_MAX/2) ? (flexion[1] -
     ANALOG_MAX/2) * 2:0;
  sprintf(stringToEncode, "A%dB%dC%dD%dE%dF%dG%dP%d%s%s%s%s%s%s%
     s%s\n",
  flexion[0], flexion[1], flexion[2], flexion[3], flexion[4],
  joyX, joyY, trigger, joyClick?"H":"",
  triggerButton?"I":"", aButton?"J":"", bButton?"K":"", grab?"L"
     :"", pinch?"M":"", menu?"N":"", calib?"O":""
  );
  return stringToEncode;
}

//legacy decoding
void decodeData(char* stringToDecode, int* hapticLimits){
  hapticLimits[0] = getArgument(stringToDecode, 'A'); //thumb
  hapticLimits[1] = getArgument(stringToDecode, 'B'); //index
  hapticLimits[2] = getArgument(stringToDecode, 'C'); //middle
  hapticLimits[3] = getArgument(stringToDecode, 'D'); //ring
  hapticLimits[4] = getArgument(stringToDecode, 'E'); //pinky
  //Serial.println("Haptic: "+ (String)hapticLimits[0] + " " + (
     String)hapticLimits[1] + " " + (String)hapticLimits[2] + "
     " + (String)hapticLimits[3] + " " + (String)hapticLimits[4]
      + " ");
}
```

```
int getArgument(char* stringToDecode, char command){
  char* start = strchr(stringToDecode, command);
  if (start == NULL)
    return -1;
  else
    return atoi(start + 1);
}

#endif
```

Listing A.3: Arquivo 'gesture.ino'

```
bool grabGesture(int *flexion){
  return (flexion[PINKY_IND] + flexion[RING_IND] + flexion[
      MIDDLE_IND] + flexion[INDEX_IND]) / 4 <= ANALOG_MAX/2 ?
      0:1;
}

bool pinchGesture(int *flexion){
  return (flexion[INDEX_IND] + flexion[THUMB_IND]) / 2 <=
      ANALOG_MAX/2 ? 0:1;
}

bool triggerGesture(int *flexion){
  return flexion[INDEX_IND]<=(ANALOG_MAX/2)?0:1;
}
```

Listing A.4: Arquivo 'haptics.ino'

```
#if USING_FORCE_FEEDBACK

#if defined(ESP32)
  #include "ESP32Servo.h"
#else
  #include "Servo.h"
#endif

Servo pinkyServo;
Servo ringServo;
Servo middleServo;
Servo indexServo;
Servo thumbServo;

void setupServoHaptics(){
  pinkyServo.attach(PIN_PINKY_MOTOR);
  ringServo.attach(PIN_RING_MOTOR);
  middleServo.attach(PIN_MIDDLE_MOTOR);
  indexServo.attach(PIN_INDEX_MOTOR);
```

```
  thumbServo.attach(PIN_THUMB_MOTOR);
}

//static scaling, maps to entire range of servo
void scaleLimits(int* hapticLimits, float* scaledLimits){
  for (int i = 0; i < 5; i++){
    scaledLimits[i] = 180.0f - hapticLimits[i] / 1000.0f * 180.0
        f;
  }
}

//dynamic scaling, maps to the limits calibrated from your
    finger
void dynScaleLimits(int* hapticLimits, float* scaledLimits){
  //will be refactored to take min and max as an argument

  /* this implementation of dynamic scaling relies on the
     assumption
   * that the servo reaches 2/3 of the potentiometer's range,
   * and that 0 degrees is geared to the start of the
      potentiometer.
   * Different hardware types may need to handle dynamic scaling
      differently.
   */
  for (int i = 0; i < sizeof(hapticLimits); i++){
    scaledLimits[i] = hapticLimits[i] / 1000.0f * 180.0f;
  }
}

void writeServoHaptics(int* hapticLimits){
  float scaledLimits[5];
  scaleLimits(hapticLimits, scaledLimits);
  if(hapticLimits[0] >= 0) thumbServo.write(scaledLimits[0]);
  if(hapticLimits[1] >= 0) indexServo.write(scaledLimits[1]);
  if(hapticLimits[2] >= 0) middleServo.write(scaledLimits[2]);
  if(hapticLimits[3] >= 0) ringServo.write(scaledLimits[3]);
  if(hapticLimits[4] >= 0) pinkyServo.write(scaledLimits[4]);
}

#endif
```

Listing A.5: Arquivo 'ICommunication.ino'

```
//Interface for communication

class ICommunication {

  public:
```

```
    virtual bool isOpen() = 0;

    virtual void start() = 0;

    virtual void output(char* data) = 0;

    virtual bool readData(char* input) = 0;
};
```

Listing A.6: Arquivo 'input.ino'

```
// Requires RunningMedian library by Rob Tillaart
#if ENABLE_MEDIAN_FILTER
  #include <RunningMedian.h>
  RunningMedian rmSamples[5] = {
      RunningMedian(MEDIAN_SAMPLES),
      RunningMedian(MEDIAN_SAMPLES),
      RunningMedian(MEDIAN_SAMPLES),
      RunningMedian(MEDIAN_SAMPLES),
      RunningMedian(MEDIAN_SAMPLES)
  };
#endif

int maxFingers[5] = {0,0,0,0,0};
int minFingers[5] = {ANALOG_MAX, ANALOG_MAX, ANALOG_MAX,
   ANALOG_MAX, ANALOG_MAX};

void setupInputs(){
  pinMode(PIN_JOY_BTN, INPUT_PULLUP);
  pinMode(PIN_A_BTN, INPUT_PULLUP);
  pinMode(PIN_B_BTN, INPUT_PULLUP);

  pinMode(PIN_MENU_BTN, INPUT_PULLUP);

  #if !TRIGGER_GESTURE
  pinMode(PIN_TRIG_BTN, INPUT_PULLUP);
  #endif

  #if !GRAB_GESTURE
  pinMode(PIN_GRAB_BTN, INPUT_PULLUP);
  #endif

  #if !PINCH_GESTURE
  pinMode(PIN_PNCH_BTN, INPUT_PULLUP);
  #endif

  #if USING_CALIB_PIN
  pinMode(PIN_CALIB, INPUT_PULLUP);
```

```
  #endif
}

int* getFingerPositions(bool calibrating, bool reset){
  int rawFingers[5] = {NO_THUMB?0:analogRead(PIN_THUMB),
      analogRead(PIN_INDEX), analogRead(PIN_MIDDLE), analogRead(
      PIN_RING), analogRead(PIN_PINKY)};

  //flip pot values if needed
  #if FLIP_POTS
  for (int i = 0; i <5; i++){
    rawFingers[i] = ANALOG_MAX - rawFingers[i];
  }
  #endif

  #if ENABLE_MEDIAN_FILTER
  for (int i = 0; i < 5; i++){
    rmSamples[i].add( rawFingers[i] );
    rawFingers[i] = rmSamples[i].getMedian();
  }
  #endif

  //reset max and mins as needed
  if (reset){
    for (int i = 0; i <5; i++){
      maxFingers[i] = 0;
      minFingers[i] = ANALOG_MAX;
    }
  }

  //if during the calibration sequence, make sure to update max
     and mins
  if (calibrating){
    for (int i = 0; i <5; i++){
      if (rawFingers[i] > maxFingers[i])
        #if CLAMP_FLEXION
          maxFingers[i] = ( rawFingers[i] <= CLAMP_MAX )?
              rawFingers[i] : CLAMP_MAX;
        #else
          maxFingers[i] = rawFingers[i];
        #endif
      if (rawFingers[i] < minFingers[i])
        #if CLAMP_FLEXION
          minFingers[i] = ( rawFingers[i] >= CLAMP_MIN )?
              rawFingers[i] : CLAMP_MIN;
        #else
          minFingers[i] = rawFingers[i];
```

```
        #endif
    }
  }

  static int calibrated[5] = {511,511,511,511,511};

  for (int i = 0; i<5; i++){
    if (minFingers[i] != maxFingers[i]){
      calibrated[i] = map( rawFingers[i], minFingers[i],
          maxFingers[i], 0, ANALOG_MAX );
      #if CLAMP_ANALOG_MAP
        if (calibrated[i] < 0)
          calibrated[i] = 0;
        if (calibrated[i] > ANALOG_MAX)
          calibrated[i] = ANALOG_MAX;
      #endif
    }
    else {
      calibrated[i] = ANALOG_MAX / 2;
    }
  }
  return calibrated;

}

int analogReadDeadzone(byte pin){
  int raw = analogRead(pin);
  if (abs(ANALOG_MAX/2 - raw) < JOYSTICK_DEADZONE * ANALOG_MAX /
      100)
    return ANALOG_MAX/2;
  else
    return raw;
}

int getJoyX(){
  #if JOYSTICK_BLANK
  return ANALOG_MAX/2;
  #elif JOY_FLIP_X
  return ANALOG_MAX - analogReadDeadzone(PIN_JOY_X);
  #else
  return analogReadDeadzone(PIN_JOY_X);
  #endif
}

int getJoyY(){
  #if JOYSTICK_BLANK
  return ANALOG_MAX/2;
```

```
  #elif JOY_FLIP_Y
  return ANALOG_MAX - analogReadDeadzone(PIN_JOY_Y);
  #else
  return analogReadDeadzone(PIN_JOY_Y);
  #endif
}

bool getButton(byte pin){
  return digitalRead(pin) != HIGH;
}
```

Listing A.7: Arquivo 'lucidgloves-firmware.ino'

```
/*
 * LucidGloves Firmware Version 4
 * Author: Lucas_VRTech - LucidVR
 * lucidvrtech.com
 */

#include "AdvancedConfig.h"

//This is the configuration file, main structure in _main.ino
//CONFIGURATION SETTINGS:
#define COMMUNICATION COMM_SERIAL //Which communication protocol
    to use
//serial over USB
  #define SERIAL_BAUD_RATE 115200

//serial over Bluetooth
  #define BTSERIAL_DEVICE_NAME "lucidgloves-left"

//ANALOG INPUT CONFIG
#define FLIP_POTS  true  //Flip values from potentiometers (for
   fingers!) if they are backwards

//Gesture enables, make false to use button override
#define TRIGGER_GESTURE true
#define GRAB_GESTURE    true
#define PINCH_GESTURE   true


//BUTTON INVERT
//If a button registers as pressed when not and vice versa (eg.
   using normally-closed switches),
//you can invert their behaviour here by setting their line to
   true.
//If unsure, set to false
#define INVERT_A false
```

```
#define INVERT_B false
#define INVERT_JOY false
#define INVERT_MENU false
#define INVERT_CALIB false
//These only apply with gesture button override:
#define INVERT_TRIGGER false
#define INVERT_GRAB false
#define INVERT_PINCH false


//joystick configuration
#define JOYSTICK_BLANK false //make true if not using the
   joystick
#define JOY_FLIP_X false
#define JOY_FLIP_Y false
#define JOYSTICK_DEADZONE 10 //deadzone in the joystick to
   prevent drift (in percent)

#define NO_THUMB false //If for some reason you don't want to
   track the thumb

#define USING_CALIB_PIN true //When PIN_CALIB is shorted (or it'
   s button pushed) it will reset calibration if this is on.

#define USING_FORCE_FEEDBACK true //Force feedback haptics allow
    you to feel the solid objects you hold
#define SERVO_SCALING false //dynamic scaling of servo motors

#if defined(ESP32)
  //(This configuration is for ESP32 DOIT V1 so make sure to
     change if you're on another board)
  #define PIN_PINKY     36
  #define PIN_RING      39
  #define PIN_MIDDLE    34
  #define PIN_INDEX     35
  #define PIN_THUMB     32
  #define PIN_JOY_X     33
  #define PIN_JOY_Y     25
  #define PIN_JOY_BTN   26
  #define PIN_A_BTN     27
  #define PIN_B_BTN     14
  #define PIN_TRIG_BTN  12 //unused if gesture set
  #define PIN_GRAB_BTN  13 //unused if gesture set
  #define PIN_PNCH_BTN  23 //unused if gesture set
  #define PIN_CALIB     12 //button for recalibration (You can
     set this to GPIO0 to use the BOOT button, but only when
     using Bluetooth.)
```

```
  #define DEBUG_LED 2
  #define PIN_PINKY_MOTOR     5  //used for force feedback
  #define PIN_RING_MOTOR      18 //^
  #define PIN_MIDDLE_MOTOR    19 //^
  #define PIN_INDEX_MOTOR     21 //^
  #define PIN_THUMB_MOTOR     17 //^
  #define PIN_MENU_BTN        27

//PINS CONFIGURATION
#elif defined(__AVR__)
  //(This configuration is for Arduino Nano so make sure to
     change if you're on another board)
  #define PIN_PINKY     A0
  #define PIN_RING      A1
  #define PIN_MIDDLE    A2
  #define PIN_INDEX     A3
  #define PIN_THUMB     A4
  #define PIN_JOY_X     A6
  #define PIN_JOY_Y     A7
  #define PIN_JOY_BTN   7
  #define PIN_A_BTN     8
  #define PIN_B_BTN     9
  #define PIN_TRIG_BTN  10 //unused if gesture set
  #define PIN_GRAB_BTN  11 //unused if gesture set
  #define PIN_PNCH_BTN  12 //unused if gesture set
  #define PIN_CALIB     0 //button for recalibration
  #define DEBUG_LED     LED_BUILTIN
  #define PIN_PINKY_MOTOR     2 //used for force feedback
  #define PIN_RING_MOTOR      3 //^
  #define PIN_MIDDLE_MOTOR    4 //^
  #define PIN_INDEX_MOTOR     5 //^
  #define PIN_THUMB_MOTOR     6 //^
  #define PIN_MENU_BTN        8
#endif
```

Listing A.8: Arquivo 'SerialBTCommunication.ino'

```
//only compiles if BTSerial is set because it won't compile for
   a non-compatible board
#if COMMUNICATION == COMM_BTSERIAL
#include "BluetoothSerial.h"
class BTSerialCommunication : public ICommunication {
  private:
    bool m_isOpen;
    BluetoothSerial m_SerialBT;

  public:
    BTSerialCommunication() {
```

```
      m_isOpen = false;
    }

    bool isOpen(){
      return m_isOpen;
    }

    void start(){
      Serial.begin(115200);
      m_SerialBT.begin(BTSERIAL_DEVICE_NAME);
      Serial.println("The␣device␣started,␣now␣you␣can␣pair␣it␣
          with␣bluetooth!");
      m_isOpen = true;
    }

    void output(char* data){
      m_SerialBT.print(data);
    }

    bool readData(char* input){
      /*byte size = m_SerialBT.readBytesUntil('\n', input, 100);
      input[size] = NULL;*/
      String message = m_SerialBT.readStringUntil('\n');
      strcpy(input, message.c_str());
      return input != NULL && strlen(input) > 0;
    }
};
#endif
```

Listing A.9: Arquivo 'SerialCommunication.ino'

```
class SerialCommunication : public ICommunication {
  private:
    bool m_isOpen;

  public:
    SerialCommunication() {
      m_isOpen = false;
    }

    bool isOpen(){
      return m_isOpen;
    }

    void start(){
      Serial.setTimeout(10);
      Serial.begin(SERIAL_BAUD_RATE);
      m_isOpen = true;
```

```
    }

    void output(char* data){
      Serial.print(data);
      Serial.flush();
    }

    bool readData(char* input){
      byte size = Serial.readBytesUntil('\n', input, 100);
      input[size] = NULL;
      return input != NULL && strlen(input) > 0;
    }
};
```

Anexo B

Instruções de uso

Para compilar o *firmware* corretamente, em adição ao processo descrito na seção 3.5, é necessário alocar todos os arquivos do anexo A.1 em uma única pasta. Em seguida, deve-se abrir o arquivo ‘lucidgloves-firmware.ino‘ no *software* Arduino IDE, conectar a ESP32 usando uma porta USB e clicar no botão de *upload*, representado por uma seta para a direita.

É importante validar se o sinal transmitido pelos potenciômetros está no sentido correto. o valor associado aos potênciometros deve aumentar à medida que o usuário fecha a mão e diminuir caso contrário. Se o comportamento inverso for identificado, é necessário alterar o valor da variável ‘FLIP_POTS‘, presente no arquivo ‘lucidgloves-firmware.ino‘.