Controle Robótico Remoto via Godot e ESP32

UNIVERSIDADE JORGE AMADO - Ciências da computação (GUILHERME COSTA ANDRADE, JOÃO FELIPE DA SILVA PINTO SANTANA, LUISA SARDINHA DOS SANTOS SILVA E PEDRO ALEXANDRE COSTA COUTINHO) TÍTULO: DESENVOLVIMENTO E ANÁLISE DE UM SISTEMA DE TELEOPERAÇÃO DE BAIXA LATÊNCIA COM REALIDADE VIRTUAL Salvador 2025

RESUMO

Este Trabalho apresenta o desenvolvimento e a análise de um protótipo de sistema de teleoperação que utiliza Realidade Virtual (VR) para o controle de atuadores físicos. O principal objetivo do projeto é investigar arquiteturas de comunicação que minimizem a latência, um fator crítico para aplicações de tempo real como cirurgias remotas e operações industriais de precisão. Foram exploradas e comparadas duas abordagens distintas: a primeira, baseada em um serviço de banco de dados em nuvem (Google Firebase), e a segunda, utilizando uma conexão local e direta via protocolo UDP (User Datagram Protocol), inicialmente, e HTTP local, na versão atual. Os resultados preliminares indicam que a arquitetura baseada em nuvem introduz uma latência proibitiva para aplicações de tempo real, enquanto a comunicação local demonstrou ser significativamente mais estável. O trabalho contribui com uma análise prática das arquiteturas e estabelece uma base para futuras investigações sobre a otimização de sistemas de teleoperação em VR, com foco na solução do complexo problema de Cinemática Inversa. Palavras-chave: Teleoperação, Realidade Virtual, Baixa Latência, ESP32, Godot Engine, Cinemática Inversa.

ABSTRACT

This project presents the development and analysis of a teleoperation system prototype that uses Virtual Reality (VR) to control physical actuators. The main objective of the project is to investigate communication architectures that minimize latency, a critical factor for real-time applications such as remote surgeries and precision industrial operations. Two distinct approaches were explored and compared: the first, based on a cloud database service (Google Firebase), and the second, using a direct local connection via the User Datagram Protocol (UDP), initially, and local HTTP in the current version. Preliminary results indicate that the cloud-based architecture introduces prohibitive latency for real-time applications, whereas the local communication proved to be significantly more stable. This work contributes a practical analysis of these architectures and establishes a foundation for future

research on the optimization of VR teleoperation systems, focusing on solving the complex Inverse Kinematics problem. Keywords: Teleoperation, Virtual Reality, Low Latency, ESP32, Godot Engine, Inverse Kinematics.

Parte I: Elementos Acadêmicos e Contextuais

1. Introdução

1.1 Contextualização, Motivação e Problema de Pesquisa

A convergência de tecnologias digitais e físicas, um pilar da Indústria 4.0, tem impulsionado o desenvolvimento de sistemas ciberfísicos cada vez mais sofisticados. Dentre eles, os sistemas de teleoperação — que permitem a um operador humano controlar um robô ou dispositivo à distância — ganham destaque por seu potencial de transformar setores como a medicina e a manufatura. A adição da Realidade Virtual (VR) a esses sistemas oferece um nível de imersão e intuição de controle sem precedentes. A eficácia de um sistema de teleoperação é, contudo, intrinsecamente dependente da sua capacidade de resposta em tempo real. A latência, definida como o tempo de atraso entre a ação do operador e a reação correspondente do dispositivo remoto, é o principal obstáculo para a viabilidade de aplicações críticas. Arquiteturas de comunicação baseadas em serviços de nuvem, embora escaláveis, introduzem uma latência significativa e variável, tornando-as inadequadas. Portanto, o problema central desta pesquisa é: Como projetar uma arquitetura de comunicação entre um ambiente de VR e um atuador físico que minimize a latência para viabilizar a teleoperação em tempo real? O objetivo de longo prazo é aplicar a solução a um braço mecânico de 5 Graus de Liberdade (GDL).

1.2 Objetivo Geral e Específicos

Objetivo Geral: Desenvolver e avaliar um protótipo de sistema de teleoperação de baixa latência, utilizando o headset Meta Quest 2 como interface de Realidade Virtual e um microcontrolador ESP32 como controlador de atuadores físicos. **Objetivos Específicos:**

- Garantir a Comunicação de Baixa Latência: Investigar, implementar e validar arquiteturas de comunicação (nuvem vs. local) com o objetivo de alcançar baixa latência ($\le 150ms$) entre o Meta Quest 2 e o microcontrolador ESP32.
- Desenvolver o Protótipo da Garra: Projetar e construir o protótipo de uma garra mecânica de 5 Graus de Liberdade (GDL) controlada por servos, capaz de replicar movimentos de um braço humano.
- Implementar a Cinemática Inversa: Elaborar uma solução de Cinemática Inversa (CI) que mapeie as coordenadas de posição e orientação da interface VR (Meta Quest 2) para os ângulos dos 5 servos motores da garra.
- Validar a Viabilidade de Teleoperação: Demonstrar a viabilidade técnica do sistema para controle preciso em tempo real, validando o potencial para a realização de ações

de teleoperação a longa distância.

1.3 Justificativa e Estrutura do Documento

A solução para o problema da latência pode expandir drasticamente o campo de aplicação da teleoperação. Este trabalho se justifica pela necessidade de investigar e validar arquiteturas de comunicação local e direta, fornecendo um caminho para sistemas mais seguros, confiáveis e eficientes. Este documento está dividido em quatro partes principais: a Parte I estabelece o contexto e a base teórica; a Parte II detalha a evolução do projeto em uma linha do tempo, incluindo as versões que falharam e os aprendizados; a Parte III descreve a arquitetura e implementação da versão atual (V4.0); e a Parte IV apresenta os resultados, conclusões e o roadmap para trabalhos futuros.

2. Referencial Teórico e Revisão Bibliográfica

2.1 Conceitos Fundamentais e Sistemas Imersivos

A Realidade Virtual é definida como um ambiente tridimensional, gerado por computador, que pode ser explorado e com o qual um usuário pode interagir de forma imersiva (TORI, 2018). A sensação de "presença" é o principal diferencial da VR. Dispositivos como o Meta Quest 2 são classificados como sistemas de VR standalone, pois integram todo o processamento e rastreamento em um único headset, facilitando a aplicação em diversos cenários (JERALD, 2015). O Godot Engine, com o plugin Godot XR Tools, é utilizado pela sua versatilidade e recursos nativos para capturar input imersivo (como o Function Pointer e o nó 2dviewIn3d) e lidar com programação de redes. ESP32: Microcontrolador de baixo custo, ideal para projetos de IoT e robótica devido à sua conectividade Wi-Fi e Bluetooth integrada. Sistemas de Teleoperação: Consiste no controle de um sistema robótico (teleoperador) por um operador humano a partir de um local remoto. O desafio central, conforme apontado por (SHERIDAN, 1992), reside na qualidade do canal de comunicação, que deve garantir a transmissão de comandos e o recebimento de feedback com a menor latência possível.

2.2 Latência e Protocolos de Comunicação

Latência: O atraso em uma rede é composto por múltiplos fatores (tempo de propagação, processamento, etc.) (KUROSE; ROSS, 2017). Em sistemas de tempo real, uma latência elevada (acima de 150-200ms) é considerada prejudicial à performance da tarefa (KUROSE; ROSS, 2017). **Protocolos de Comunicação:**

- API (Interface de Programação de Aplicações): Conjunto de regras utilizadas na Arquitetura A (Firebase).
- HTTP (Protocolo de Transferência de Hipertexto): Protocolo básico da web, utilizado na Versão 3.0 (local).
- UDP (Protocolo de Datagrama de Usuário): É um protocolo não orientado à conexão que envia pacotes de forma rápida, sem garantir entrega ou ordem. Por seu baixo

overhead (sobrecarga) e alta velocidade, o UDP é o protocolo de escolha para sistemas de controle (TANENBAUM, 2011). Foi testado na Versão 1.5 **e é o protocolo em uso na V4.0 (Semana 6)**.
- MQTT (Protocolo de Mensagens de Telemetria de Fila): Protocolo leve, futuro alvo para comunicação assíncrona e eficiente.

2.3 Trabalhos Correlatos

A pesquisa em teleoperação via VR é um campo ativo. Por exemplo, o trabalho de

$$AUTOR, Ano$$

explorou o uso de WebSockets para o controle de um braço robótico, atingindo latências na ordem de 80ms em redes locais. Já

$$AUTOR, Ano$$

propôs um algoritmo preditivo para compensar a latência em redes de longa distância, mas com complexidade computacional elevada. O presente trabalho se diferencia ao focar na comparação de arquiteturas de comunicação fundamentalmente distintas (nuvem vs. local) para um sistema embarcado de baixo custo como o ESP32.

3. Metodologia e Planejamento Inicial

3.1 Abordagem de Desenvolvimento

Este trabalho caracteriza-se como uma pesquisa aplicada e experimental. A metodologia adotada foi baseada no desenvolvimento iterativo de protótipos e na análise quantitativa da latência de arquiteturas de sistema distintas. O projeto seguiu uma progressão lógica, iniciando com uma prova de conceito e evoluindo para a arquitetura otimizada em ciclos curtos de prototipagem incremental.

3.2 Arquitetura dos Sistemas e Ferramentas

Arquitetura A (Baseada em Nuvem - V1.0): O ambiente VR (Godot) envia requisições HTTP PUT para um banco de dados Firebase. O ESP32 monitora as alterações e atua. **Arquitetura B (Local e Direta - V2.0 / V3.0 / V4.0):** O ESP32 é configurado como um Access Point Wi-Fi. A comunicação é realizada diretamente (via UDP na V2.0 **e V4.0**, e HTTP na V3.0) entre o Meta Quest 2 e o ESP32. **Ferramentas:** Headset VR (Meta Quest 2), Microcontrolador (ESP32-DevKitC V4), Atuadores (LEDs e protoboard), Motor de Jogo (Godot Engine 4.2), Linguagens (GDScript e C++ para o Firmware), Simulador de Teste (Python 3.x).

3.3 Procedimentos de Avaliação (Latência)

Para a análise de latência, os testes foram conduzidos em duas fases:

- Ambiente Simulado: O cliente VR (Godot) e o servidor (simulador em Python) foram executados na mesma máquina, comunicando-se através da interface de rede de loopback (127.0.0.1). Um registro de tempo (timestamp) era marcado no envio e outro no recebimento, sendo a latência a diferença.
- Hardware Real (V3.0): A medição foi realizada entre o clique no controle do Meta Quest 2 e o acionamento físico do LED controlado pelo ESP32, fornecendo uma métrica mais precisa das condições reais de rede local.

Requisitos Funcionais e Não-Funcionais: Requisitos Iniciais (Semana 1):

- Software (Godot/MQ2): Interface gráfica, conexão com o microcontrolador, captação das posições das mãos/dedos, transmissão de dados de posição.
- Robótica (Braço Mecânico): Capacidade de replicar movimentos de um braço orgânico, força igual ou superior à de um braço médio, capacidade de agarrar objetos pequenos. **Requisitos Revisados (Semana 2):**
- Software (Godot/MQ2): Interface gráfica, conexão com o ESP32, captação das posições dos controles (substituindo o Hand Tracking), transmissão do status dos botões pressionados.
- Robótica: (Mantidos, mas com foco na garra de 5 servos motores).

Parte II: A Linha do Tempo e o Processo de Engenharia

4. Linha do Tempo de Desenvolvimento e Iterações

4.1 Versão 0.x - Tentativas Iniciais (O Que Deu Errado)

- **Problema/Ideia:** Inicialmente, o projeto focou em usar o Hand Tracking nativo do Meta Quest 2 para capturar os movimentos e posições da mão.
- **Resultado e Falha:** O rastreamento de mãos se mostrou altamente impreciso e instável para a necessidade de controle fino do braço robótico, levando ao abandono dessa abordagem.
- **Aprendizado (Lesson Learned):** Para alcançar a precisão necessária, a captação de dados deve ser feita através dos controles físicos do Meta Quest 2. Além disso, o foco de desenvolvimento mudou de movimentos de dedo para o mapeamento dos botões do controle para a ativação inicial de funções (Semana 2).

4.2 Versão 1.x - O Primeiro Protótipo Funcional (A Falha Firebase)

- **Arquitetura (V1.0 - Semana 3):** O primeiro protótipo funcional visava acender LEDs reais (simulados) a partir de cliques em botões na interface Godot. A arquitetura de

comunicação foi: Godot (MQ2) $\rightarrow$ API REST $\rightarrow$ Firebase (Banco de Dados Online) $\rightarrow$ ESP32 (Arquitetura A, conforme Seção 3.2).

- **Código Godot (V1.0 - Godot to Firebase):**
    - O Node3D.gd e Button.gd foram desenvolvidos para encapsular o status dos botões e enviá-lo via requisição PUT para o Firebase (Anexo A.1).
- **Resultado e Falha:** A latência (delay) era extremamente alta (maior que 2 segundos em alguns casos), o sistema era completamente dependente da estabilidade da internet e da performance do Firebase. Concluiu-se que esta arquitetura é inviável para qualquer aplicação de teleoperação que exija resposta em tempo real.
- **Aprendizado:** A comunicação precisa ser direta, local e com o mínimo de intermediários possível.

4.3 Versão Atual (V2.x/3.x) - A Busca pela Baixa Latência

- **V2.0 (Simulação UDP - Semana 4):** A primeira tentativa de comunicação direta foi via UDP (Protocolo de Datagrama de Usuário), utilizando o ESP32 como Access Point (roteando uma rede Wi-Fi local). Para testar a ideia, um ESP32 simulado em Python foi usado, comunicando-se via loopback (127.0.0.1:8888) com o Godot (Arquitetura B - UDP).
    - **Resultado do Teste:** Embora o sistema fosse estável e independente da internet, o delay permaneceu alto (aproximadamente 1 segundo) na simulação, levantando dúvidas sobre a performance do Godot e Python em loops.
- **V3.0 (HTTP Local - Semana 5):** A versão atual implementou a conexão direta, utilizando HTTP para transferir o status dos botões do Meta Quest 2 diretamente para o ESP32 (que atua como Access Point).
    - **Implementação:** O aplicativo foi migrado para o modo AR (Realidade Aumentada). A arquitetura de input do Godot foi aprimorada, migrando do ineficiente ray_cast_vr para o Function Pointer do plugin Godot XR Tools. O ESP32 foi configurado com sucesso para acender LEDs.
    - **Situação Atual:** Esta versão atingiu latências de 70ms a 218ms no hardware real (MQ2 $\leftrightarrow$ ESP32), um resultado satisfatório para o estágio de prototipagem e que valida o princípio da comunicação local, embora a latência ainda seja alvo de otimização.

4.4 Versão 4.0 - Otimização UDP (Semana 6 - Atual)

- **Status:** Implementação dos "Recursos Prioritários" (Seção 8.1 do doc. original) e migração bem-sucedida para UDP.
- **Protocolo:** Estamos usando o protocolo UDP para passar informações entre o VR e o ESP.
- **Rastreamento:** Estamos passando o eixo XYZ e ABC (posição e rotação) das duas mãos.
- **Latência:** A taxa de transmissão está estável e menor que 20ms (operando a 20Hz).

- **Interação:** O acionamento dos botões (testado com LEDs) está sendo feito pelos gatilhos principais dos controles, sem a necessidade de mirar em nenhum lugar, como planejado.

Parte III: Documentação Técnica Atual (O Guia de Onboarding)

5. Arquitetura e Implementação da Versão Atual (V4.0)

5.1 Godot Engine - Implementação Detalhada (Frontend)

O Godot é responsável pela interface AR e pela lógica de envio de pacotes. A arquitetura de interface foi aprimorada, utilizando o plugin Godot XR Tools.

- **Interação:** A detecção de clique foi migrada do ineficiente ray_cast_vr para o sistema de Function Pointer (Recurso do Godot XR Tools). O objetivo prioritário da Semana 6 **era** configurar esta interação para ser acionada diretamente pelo gatilho principal do controle, eliminando a necessidade de mirar em um lugar específico. **Isso foi concluído (V4.0)**, garantindo um acionamento instantâneo, mais intuitivo e de baixa latência.
- **Interface de Botões:** Os botões foram implementados utilizando o nó 2dviewIn3d do Godot XR Tools. Este recurso permite que interfaces 2D comuns sejam renderizadas e manipuladas em um ambiente 3D de forma interativa. Os estados dos botões (agora lidos dos gatilhos) são mapeados para um JSON (futuramente bytes puros) para otimizar o envio via rede. O Godot é compilado para o Android (para o Meta Quest 2).

5.2 Firmware ESP32 - Implementação Detalhada (Hardware)

O ESP32 está configurado para operar no modo Access Point (AP), criando uma rede Wi-Fi dedicada que é acessada pelo Meta Quest 2. O firmware (Anexo B) utiliza bibliotecas de servidor web HTTP (V3.0) para receber as requisições e decodificar o payload (a mensagem) em comandos para os servos motores. **Nota: O firmware está defasado (V3.0) e precisa ser atualizado para receber os pacotes UDP da V4.0.**

5.3 Protocolo de Comunicação Unificado (V4.0)

- **Atual:** Comunicação **UDP** direta na rede local (`Rede_ESP_VR`). A mensagem (atualmente JSON) contém o estado dos botões (via gatilhos) e os dados de tracking (XYZ/ABC).
- **Próxima Etapa:** (Mantida do doc original) Busca por protocolos como WebSocket ou MQTT. *(Embora o UDP V4.0 esteja funcional, a otimização futura para Bytes Puros (Seção 8.1) ou mudança de topologia ainda está pendente)*.

6. Resultados e Validação

6.1 Testes de Performance

- **V1.0 (Firebase - Arquitetura A):** Latência > 2000ms. Inviável.
- **V2.0 (UDP Simulado):** Latência estável, mas alta (~1000ms) na simulação.
- **V3.0 (HTTP no MQ2** $\leftrightarrow$ **ESP32 - Arquitetura B):** Latência medida entre 70ms (melhor caso) e 218ms (pior caso).
- **V4.0 (UDP no MQ2** $\leftrightarrow$ **ESP32 - Semana 6): Latência de transmissão de pacotes inferior a 20ms (20Hz).**

6.2 Validação de Requisitos

Os requisitos de interface gráfica, conexão com o ESP32, e transmissão do estado dos botões foram validados. O requisito de baixa latência ($\le 150ms$) foi **atingido e superado na V4.0 (< 20ms)**.

Parte IV: Conclusão e Futuro

7. Conclusão

7.1 Síntese dos Resultados e Contribuições

Este trabalho demonstrou que uma arquitetura baseada em serviços de nuvem como o Firebase, apesar da facilidade de implementação, introduz uma latência proibitiva para aplicações de tempo real. A arquitetura **V4.0 (UDP)**, proposta e baseada em uma conexão local e direta, provou ser significativamente mais estável, rápida (< 20ms) e adequada para o desenvolvimento de sistemas de teleoperação responsivos. A principal contribuição do trabalho é a validação prática da abordagem de rede local dedicada para sistemas de baixo custo como o ESP32.

7.2 Limitações do Projeto Atual

O projeto tem três limitações principais:

1. **Mapeamento Cinemático:** O problema mais complexo é o mapeamento dos movimentos da garra (5 servos motores) para as coordenadas XYZ (posição) e ABC (orientação) dos controles, o que requer uma cinemática inversa eficiente.
2. **Firmware Defasado:** O firmware do ESP32 (Anexo B) está configurado para HTTP (V3.0) e precisa ser atualizado para receber e processar os pacotes UDP (V4.0) que o Godot já está enviando.
3. **Formato de Dados:** Os dados ainda são enviados como JSON (V4.0), o que gera *parsing* no ESP32. A migração para Bytes Puros (planejada na Seção 8.1 do doc.

original) ainda não foi feita.

8. Trabalhos Futuros (O Roadmap para Colaboradores)

8.1 Próximos Passos (Pós-Semana 6)

(Esta seção foi atualizada para refletir os novos objetivos definidos na Semana 6)

1. **Atualizar Firmware ESP32:** Refatorar o `TCC_ESP32.ino` (Anexo B) para parar de usar `WebServer.h` (HTTP) e implementar `WiFiUDP.h` para receber e processar os pacotes UDP das portas 4210 (tracking) e 4211 (botões).
2. **Calcular Cinemática Inversa (CI) no Godot:** Iniciar o cálculo dos ângulos de cada servo (responsáveis por cada articulação) diretamente no Meta Quest, como sugere a "Resolução 2" (Seção 8.2).
3. **Refazer a Garra (Protótipo Físico):** Reconstruir a garra (o primeiro protótipo foi feito com palitos) com materiais mais robustos para testar os servos.
4. **Analisar Streaming de Câmera:** Investigar a viabilidade de utilizar uma câmera acoplada à garra para passar informação para uma tela (viewport) dentro do Godot, permitindo ao operador ver e operar a garra sem estar presente.
5. **Implementar Modo Híbrido (AR/VR):** Permitir a alteração entre o modo AR (visão local) e o modo VR (usando a câmera), visando a operação a longa distância.

8.2 Desafios a Longo Prazo: Cinemática Inversa (CI)

O maior e mais complexo desafio é a resolução da Cinemática Inversa (CI) do braço robótico de 5 Graus de Liberdade (GDL). Enquanto a Cinemática Direta (CD) calcula a posição final da garra (o Efetor Final) dadas as posições angulares dos servos, a CI faz o oposto: calcula os ângulos de cada um dos 5 servos motores a partir de uma posição e orientação (XYZ + ABC) desejada pelo operador (CRAIG, 2005). A complicação reside no fato de que o modelo da garra permite que múltiplos servos atuem no mesmo eixo espacial, criando uma redundância que exige um algoritmo de decisão eficiente (o "melhor caminho" para mover a garra). O primeiro passo para resolver este desafio é a modelagem geométrica do braço, que geralmente é realizada utilizando a convenção de Parâmetros de Denavit-Hartenberg (D-H) para estabelecer a relação matemática entre as articulações (CRAIG, 2005; ROSÁRIO, 2011). **Abordagem (Pós-Semana 6):** O foco inicial para o mapeamento da garra será limitar a CI a modelos teóricos e/ou modelos de baixa veracidade em relação ao modelo robótico real.

Três soluções foram propostas, cada uma com seus prós e contras em termos de processamento e latência:

1. **Cinemática Inversa Baseada em Aprendizado de Máquina (IA):**
    - *Descrição:* Treinar um modelo de IA (como uma Rede Neural) para mapear a posição XYZ/ABC do controle para os 5 ângulos de servo.
    - *Vantagem:* Pode encontrar soluções otimizadas para caminhos complexos.

    - *Desvantagem:* Exige alto poder de processamento (provavelmente no Godot/MQ2) e grandes quantidades de dados de treinamento.
2. **Cinemática Inversa no Godot (Dedução e Envio Angular):**
    - *Descrição:* O Godot executaria o cálculo da CI para os eixos principais (ombro e cotovelo) e enviaria os 5 valores angulares (joint-space) diretamente para o ESP32.
    - *Vantagem:* Alivia a carga de processamento do ESP32, garantindo que o tempo de resposta se mantenha baixo. **Esta é a abordagem atual escolhida (conforme 'Resolução 2' definida na Semana 6).**
3. **Cinemática Inversa no ESP32 (Função Analítica/Algoritmo):**
    - *Descrição:* Criar uma função analítica ou um algoritmo de decisão no firmware do ESP32 para calcular os ângulos. Esta é a opção preferida por sua natureza self-contained (autocontida).
    - *Vantagem:* O sistema se torna totalmente independente e a latência de comunicação é minimizada.
    - *Desvantagem:* O ESP32 tem recursos de processamento limitados, tornando os cálculos complexos de CI um desafio de performance.

A próxima fase do projeto focará em desenvolver e testar a **solução 2**.

8.3 Guia para Contribuição

$$TEXTO: Link para o repositório, passos para compilar e rodar o projeto.$$

REFERÊNCIAS

Observação: A lista abaixo é um exemplo de formatação ABNT para TCC. Você deve preencher os dados exatos (cidade, editora, ano exato) após confirmar a obra.

CRAIG, John J. Introduction to Robotics: Mechanics and Control. 3. ed. Upper Saddle River, NJ: Prentice-Hall, 2005.

JERALD, Jason. The VR Book: Human-Centered Design for Virtual Reality. Nova York: Morgan & Claypool Publishers, 2015.

KUROSE, James F.; ROSS, Keith W. Redes de Computadores e a Internet: Uma Abordagem Top-Down. 7. ed. São Paulo: Pearson Education, 2017.

ROSÁRIO, João Manuel Dias da Silva. Princípios de Robótica. Évora: Escola de Ciência e Tecnologia, Universidade de Évora, 2011.

SHERIDAN, Thomas B. Telerobotics, Automation, and Human Supervisory Control. Cambridge, MA: MIT Press, 1992.

TANENBAUM, Andrew S.; WETHERALL, David J. Redes de Computadores. 5. ed. São Paulo: Pearson Prentice Hall, 2011.

TORI, Romero; HOUNSELL, Marcelo da Silva (org.). Introdução a Realidade Virtual e Aumentada. Porto Alegre: Sociedade Brasileira de Computação (SBC), 2018.

ANEXOS

ANEXO A – CÓDIGO-FONTE DA ARQUITETURA FIREBASE

(Cole aqui os códigos da Fase 3: Node3d.gd e Button.gd)

ANEXO B – CÓDIGO-FONTE DA ARQUITETURA UDP/HTTP LOCAL

(Cole aqui os códigos da Fase 4 e 5: O código final do Godot e o simulador em Python. **Nota: Os códigos precisam ser atualizados para refletir a V4.0/UDP.**)