



UNIVERSIDADE ESTADUAL PAULISTA
“JÚLIO DE MESQUITA FILHO”
Instituto de Ciência e Tecnologia
Câmpus de Sorocaba

JOHNNATAN RODRIGUES DE OLIVEIRA

Sistema embarcado real aplicado ao controle de processo de uma planta
virtual: Unity, Linux, Mqtt e Raspberry

Sorocaba

2024

JOHNNATAN RODRIGUES DE OLIVEIRA

**Sistema embarcado real aplicado ao controle de processo de uma planta
virtual: Unity, Linux, Mqtt e Raspberry**

Trabalho de Conclusão de Curso apresentado à
Universidade Estadual Paulista (UNESP),
Instituto de Ciência e Tecnologia, Sorocaba,
como parte dos requisitos para obtenção do grau
de Bacharel(a) em Engenharia de Controle e
Automação.

Orientador: Prof. Dr. Ivando Severino Diniz

Sorocaba

2024

O48s

Oliveira, Johnnatan Rodrigues de

Sistema embarcado real aplicado ao controle de processo de uma
planta virtual: Unity, Linux, Mqtt e Raspberry / Johnnatan
Rodrigues de Oliveira. -- Sorocaba, 2024
68 p. : il., tabs.

Trabalho de conclusão de curso (Bacharelado - Engenharia de
Controle e Automação) - Universidade Estadual Paulista (UNESP),
Instituto de Ciência e Tecnologia, Sorocaba
Orientador: Ivando Severino Diniz

1. Simulação por computador. 2. Controle de processo. 3. Realidade
virtual. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Dados fornecidos pelo autor(a).

JOHNNATAN RODRIGUES DE OLIVEIRA

**Sistema embarcado real aplicado ao controle de processo de uma planta
virtual: Unity, Linux, Mqtt e Raspberry**

Trabalho de Conclusão de Curso apresentado ao Instituto de Ciência e Tecnologia de Sorocaba, Universidade Estadual Paulista (UNESP), como parte dos requisitos para obtenção do grau de Bacharel(a) em Engenharia de Controle e Automação.

Data da defesa: 18/12/2024

BANCA EXAMINADORA:

Prof. Dr. Ivando Severino Diniz

UNESP – Instituto de Ciência e Tecnologia – Campus de Sorocaba

Prof. Dr. Admilson Irio Ribeiro

UNESP – Instituto de Ciência e Tecnologia – Campus de Sorocaba

Prof. Dr. Nilson Cristino da Cruz

UNESP – Instituto de Ciência e Tecnologia – Campus de Sorocaba

Agradecimentos

Agradeço imensamente e dedico esta dissertação a várias pessoas que foram fundamentais na minha jornada acadêmica e pessoal.

Primeiramente, aos meus pais, Celso e Vilma, pela vida, criação, suporte e amor incondicional. A estrutura e o carinho com que me proporcionaram me permitiram tornar-me quem sou hoje.

Ao meu orientador, Ivando, pela paciência, carisma, ensinamentos e motivação. Seus incentivos e a fomentação de curiosidade por áreas do conhecimento, através de suas muitas ideias brilhantes, me inspiram a buscar e aplicar a aprendizagem contínua com paixão.

À minha namorada, Mayra, por seu carinho, amor, compreensão e companhia. Sua presença torna os dias mais, leves, felizes e completos, sendo uma força motriz que me impulsiona a melhorar a cada dia.

Às *Avengers*, *Anna*, *Fer*, *Blenda* e *Beea*, amigas incríveis que tornaram o percurso acadêmico memorável. Juntos, enfrentamos desafios, celebramos conquistas e criamos memórias inesquecíveis, tornando esta jornada especial e divertida. É um orgulho afirmar que essas são pessoas extraordinárias no que fazem e uma companhia indispensável, tornando cada momento do curso valioso e gratificante.

Agradeço também aos membros da banca avaliadora por suas valiosas contribuições e sugestões, que certamente enriqueceram este trabalho.

De forma especial a UNESP por fornecer um ambiente acadêmico estimulante e por apoiar o desenvolvimento intelectual e profissional. A infraestrutura, os recursos e o corpo docente da instituição foram fundamentais para a realização do projeto e assim como crescimento profissional.

Por fim, agradeço a todos que, de alguma forma, contribuíram para a realização deste projeto.

Oliveira, J. R. **Sistema embarcado real aplicado ao controle de processo de uma planta virtual: Unity, Linux, Mqtt e Raspberry** 2024. 68p. Trabalho de Conclusão de Curso (Bacharelado em Engenharia de Controle e Automação) – Instituto de Ciência e Tecnologia, Universidade Estadual Paulista, Sorocaba, 2024.

RESUMO

A aplicação da Realidade Virtual (RV) no ensino de controle de processos representa uma inovação significativa, proporcionando uma alternativa prática e imersiva aos tradicionais laboratórios físicos. Este projeto desenvolveu uma infraestrutura robusta integrando cliente, servidor, dispositivos e rede para criar um ambiente virtual interativo. O cliente, operando em um ambiente web ou Unity, oferece a interface gráfica para interação do usuário. O servidor, configurado em um dispositivo rodando containers Docker, gerencia a comunicação entre todos os componentes, utilizando serviços como Node.js, MongoDB e MQTT. Microcontroladores ESP32 e ESP01 são programados para interagir com sensores e atuadores, enviando dados para o servidor via MQTT. A rede local, conectada através de roteadores ou hotspots móveis, assegura a comunicação eficiente entre todos os elementos. Simulações, incluindo a planta MPS-PA e cenários customizados, demonstram a flexibilidade e abrangência do ambiente virtual. Comparações com sistemas físicos, como a estação de trabalho MPS-PA Compact da Festo, revelam que o sistema virtual é capaz de replicar com precisão as funcionalidades dos sistemas físicos, oferecendo vantagens adicionais em termos de acessibilidade, segurança e custo. A análise dos resultados confirma a eficácia do ambiente virtual em atender aos objetivos educacionais, destacando a integração de tecnologias avançadas e superando as limitações dos métodos tradicionais de ensino. Conclui-se que a plataforma desenvolvida é uma ferramenta eficaz para o ensino de controle de processos, complementando a aprendizagem teórica com uma experiência prática e interativa, e preparando os alunos para as demandas da Indústria 4.0.

Palavras-chave: Realidade Virtual. Controle de processos. Laboratório virtual. Tecnologias no ensino de engenharia.

Oliveira, J. R. **Real-World Embedded System for Process Control of a Virtual Plant: Unity, Linux, MQTT, and Raspberry**. 2024. 68p. Final Paper (Bachelor's degree in Control and Automation Engineering) – Institute of Science and Technology, São Paulo State University, Sorocaba, 2024.

ABSTRACT

The application of Virtual Reality (VR) in process control education offers a practical and immersive alternative to traditional physical laboratories. This project developed a robust infrastructure integrating client, server, devices, and network to create an interactive virtual environment. Operating in a web or Unity environment, the client provides the graphical interface for user interaction. Configured on a device running Docker containers, the server manages communication between components using services like Node.js, MongoDB, and MQTT. ESP32 and ESP01 microcontrollers interact with sensors and actuators, sending data to the server via MQTT. The local network, connected through routers or mobile hotspots, ensures efficient communication. Simulations, including the MPS-PA plant and customized scenarios, demonstrate the flexibility and comprehensiveness of the virtual environment. Comparisons with physical systems, such as Festo's MPS-PA Compact workstation, show that the virtual system accurately replicates the functionalities of physical systems, offering advantages in accessibility, safety, and cost. Results confirm the effectiveness of the virtual environment in meeting educational objectives, highlighting the integration of advanced technologies and surpassing traditional teaching methods. The developed platform is an effective tool for process control education, complementing theoretical learning with practical and interactive experience, preparing students for Industry 4.0 demands.

Keywords: Virtual Reality. Process control. Virtual laboratory. Engineering education technologies.

LISTA DE FIGURAS

Figura 1 - Disposição dos componentes da plataforma.....	20
Figura 2- Exemplo de Código na IDE do Arduino.....	26
Figura 3- Interface do ProBuilder no Unity para Modelagem 3D.....	31
Figura 4 - Modelagem Detalhada no Blender dos Componentes 3D.....	32
Figura 5 - Configuração dos Modelos Detalhados no Unity.....	33
Figura 6 - Diagrama de Expressão Regular para Identificação de Âncoras.....	36
Figura 7 - Arquitetura Geral do Servidor Node.js.....	39
Figura 8 - Orquestração dos Serviços no Docker Compose.....	40
Figura 9 - Exposição de Portas e Rotas com Express.js.....	42
Figura 10 - Ferramentas de Depuração MQTT.....	45
Figura 11 - Interações e Configuração do MongoDB.....	46
Figura 12 -Telas Iniciais: Login, Registro.....	54
Figura 13 -Interação do Usuário com o Ambiente Virtual na tela principal.....	55
Figura 14 -Interatividade com o ambiente.....	55
Figura 15 -MPS-PA Festo Usada como Base para o Sistema.....	57
Figura 16 -Controle de nível com ESP01 como CLP.....	59
Figura 17 -Controle de Nível com ESP32 e Entradas Adicionais.....	59

LISTA DE ABREVIATURAS E SIGLAS

- **API:** Application Programming Interface
- **CLI:** Command Line Interface
- **CRUD:** Create, Read, Update, Delete
- **DB:** Database
- **ESP01 e ESP 01:** Microcontrolador compacto da Espressif Systems
- **FBX:** Autodesk Filmbox (formato de arquivos 3D)
- **FFT:** Fast Fourier Transform (Transformada Rápida de Fourier)
- **GPU:** Graphics Processing Unit
- **HTTP:** Hypertext Transfer Protocol
- **IDE:** Integrated Development Environment
- **IoT:** Internet of Things (Internet das Coisas)
- **JSON:** JavaScript Object Notation
- **MQTT:** Message Queuing Telemetry Transport
- **MPS-PA:** MPS-PA Compact Workstation (Festo)
- **NPM:** Node Package Manager
- **RA:** Realidade Aumentada
- **RGB:** Red, Green, Blue (código de cores)
- **RV:** Realidade Virtual
- **TCP:** Transmission Control Protocol
- **UDP:** User Datagram Protocol
- **VR:** Virtual Reality (Realidade Virtual)
- **WS:** WebSocket

SUMÁRIO

1	INTRODUÇÃO	11
2	REFERENCIAL TEÓRICO	13
2.1	Realidade Virtual	13
2.2	Gêmeos Digitais	14
2.3	Controle de Processos	14
2.4	Tecnologias Utilizadas	16
2.4.1	Unity	16
2.4.2	Docker	16
2.4.2.1	Rede Bridge em Docker	17
2.4.2.2	Volumes em Docker	17
2.4.3	Node.js	17
2.4.4	MongoDB	18
2.4.5	MQTT	18
2.4.6	ESP32 e ESP01	19
3	MATERIAIS E MÉTODOS	20
3.1	Materiais da estruturais da plataforma	20
3.1.1	Cliente:	20
3.1.2	Servidor:	20
3.1.3	Dispositivo:	20
3.1.4	Modem/Roteador (Rede):	21
3.2	Ambiente de Desenvolvimento	21
3.2.1	Hardware e Software Utilizados	21
3.3	Desenvolvimento do Ambiente Virtual	22
3.3.1	Modelagem e Implementação no Unity	22
3.4	Integração dos Sistemas	23
3.4.1	Gestão da Integração com Node.js e Docker	24

3.4.2 Criação de APIs com Express	24
3.5 Desenvolvimento do Controlador	25
3.5.1 Seleção e Configuração dos Microcontroladores	25
3.5.2 Ferramentas e Procedimentos de Programação.....	26
3.5.3 Processo de Programação e Integração.....	27
3.6 Desenvolvimento da Interface Web.....	28
3.6.1 Materiais Utilizados	28
3.6.2 Métodos e Procedimentos	28
4 DESENVOLVIMENTO.....	30
4.1 Requisitos do Sistema	30
4.2 Implementação do Ambiente Virtual no Unity	30
4.2.1 Modelagem 3D dos Componentes.....	30
4.2.2 Configuração dos Sistemas de Controle.....	34
4.2.3 Interações do Usuário com Sliders, Toggles e Buttons	37
4.3 Configuração do Servidor Node.js	39
4.3.1 Estrutura Geral do Servidor	39
4.3.2 Orquestração dos Serviços Docker	40
4.3.3 Exposição de Portas e Rotas.....	41
4.4 Integração MQTT.....	43
4.4.1 Configuração do Broker MQTT.....	43
4.4.2 Estrutura dos Tópicos MQTT	43
4.4.3 Implementação da Comunicação MQTT no Servidor Node.js.....	44
4.4.4 Comunicação entre Servidor Node.js e Interface Web.....	44
4.4.5 Testes e Depuração.....	45
4.5 Armazenamento de Dados com MongoDB.....	45
4.5.1 Configuração do Banco de Dados MongoDB	46

4.5.2 Estrutura de Dados	46
4.5.3 Integração com o Servidor Node.js.....	48
4.5.4 Operações CRUD	48
4.5.5 Rotas da API.....	49
4.6 Containerização com Docker	50
4.6.1 Configuração do Docker Compose	50
4.6.2 Criação de Redes e Volumes	51
4.6.3 Comandos e Execução	52
4.6.4 Desenvolvimento da Configuração com Docker	52
4.7 Desenvolvimento do Controlador com ESP-32 e ESP01	53
5 RESULTADOS	54
5.1 Fluxo de Navegação do Usuário no Ambiente Virtual	54
5.2 Sincronização em Tempo Real e Comunicação entre Dispositivos	56
5.3 Estudo de Caso: Controle de Nível com Gêmeo Digital	57
6 DISCUSSÃO	60
6.1 Análise dos Resultados Obtidos	60
6.2 Comparação com Trabalhos Relacionados	60
6.3 Sugestões para Trabalhos Futuros	61
7 CONCLUSÃO	63
REFERÊNCIAS	64

1 INTRODUÇÃO

A modelagem de sistemas de controle constitui uma das áreas centrais na engenharia de controle e automação, sendo crucial para garantir a eficiência e a qualidade dos processos industriais (Hästbacka, Vepsäläinen, Kuikka, 2011). A capacidade de modelar e simular sistemas complexos permite aos engenheiros prever e otimizar o comportamento dos processos, garantindo maior segurança, eficiência energética e redução de custos (Mawson e Hughes, 2019). Tradicionalmente, o ensino desses conceitos é realizado principalmente por meio de aulas teóricas e demonstrações em laboratórios físicos (Soliman, 2018). No entanto, essa abordagem carece de um ambiente que integre de forma eficaz os aspectos teóricos e práticos, o que pode dificultar a assimilação e a aplicação dos conceitos pelos estudantes, distanciando-os da realidade industrial (Seledtsova, Chechurin, 2019).

Nos últimos anos, a transformação digital e o avanço tecnológico têm revolucionado diversos setores, incluindo a educação (Pedro e Teixeira, 2021). Tecnologias emergentes, como a Realidade Virtual (RV) e os gêmeos digitais, apresentam um grande potencial para transformar o ensino, especialmente em áreas técnicas e científicas (Künz et al., 2022). A RV, por exemplo, tem se mostrado uma ferramenta poderosa para melhorar a compreensão de conceitos abstratos (Marougkas et al., 2023), oferecendo uma experiência imersiva e interativa onde os usuários podem simular diferentes situações e realizar tarefas de maneira segura e controlada (Papanastasiou et al., 2018).

Neste contexto, este Projeto de Final de Curso (PFC) visa explorar uma abordagem inovadora de ensino de controle utilizando RV para modelar um laboratório de controle. O desenvolvimento de ambientes virtuais interativos oferece uma alternativa eficaz e acessível à dependência de laboratórios físicos, que são caros e de difícil acesso (Kharki, Berrada, Burgos, 2021; Wolf, 2010). A utilização de um laboratório virtual permite a reprodução de sistemas de controle de processos complexos, proporcionando aos alunos uma plataforma prática e envolvente para aprender e experimentar.

A necessidade de integrar teoria e prática de forma eficiente (Froyd e Ohland, 2005) motiva a criação deste projeto. A RV, em particular, oferece uma capacidade única de imersão que pode transformar a maneira como os conceitos de controle de processos são ensinados e compreendidos. Ao simular um ambiente de laboratório realista, os estudantes podem interagir com os sistemas de controle de uma maneira que seria impossível em um ambiente exclusivamente teórico. Além disso, o ensino de controle de processos em um ambiente virtual

não só aumenta a acessibilidade ao laboratório como também oferece uma flexibilidade maior na experimentação e no aprendizado. Em ambientes educacionais tradicionais, a limitação de recursos físicos e a necessidade de manutenção constante dos equipamentos podem ser barreiras significativas. Além disso, a utilização de tecnologias avançadas como RV e gêmeos digitais prepara os alunos para as demandas da Indústria 4.0, onde a integração entre sistemas físicos e virtuais é cada vez mais comum.

Para alcançar esse objetivo, o projeto integrará várias tecnologias. Os servidores Node.js serão usados para gerenciar a comunicação entre os componentes do sistema, enquanto o protocolo MQTT facilitará a comunicação em tempo real e o MongoDB será utilizado para armazenamento de dados de simulação. A MPS-PA Compact Workstation da Festo será utilizada como base para o ambiente virtual, permitindo exemplificar diversos conceitos de controle de processos de maneira interativa e prática. A criação do ambiente virtual incluirá o desenvolvimento de programas específicos para microcontroladores ESP32 e ESP01, que se conectarão a sensores e atuadores. A simulação ocorrerá dentro do serviço cpp-app, controlado por um servidor orquestrado pelo Node.js, tudo dentro de um container, permitindo que os alunos interajam com o sistema tanto em ambientes de RV imersivos quanto em navegadores web comuns.

Espera-se que o ambiente virtual desenvolvido sirva como uma ferramenta eficaz para o ensino de controle de processos, complementando a aprendizagem teórica com uma experiência prática e interativa. A análise dos resultados e a comparação com sistemas físicos reais deverão demonstrar a eficácia do ambiente virtual em reproduzir e, em alguns aspectos, melhorar a experiência de aprendizado tradicional.

O documento inicia apresentando os conceitos fundamentais relacionados ao controle de processos, Realidade Virtual, gêmeos digitais e as tecnologias utilizadas no projeto. Na metodologia, são abordadas a infraestrutura necessária, as ferramentas e tecnologias empregadas, a configuração do ambiente, além da estrutura e funcionamento do sistema, e o fluxo de dados e comunicação entre os componentes. Em seguida, no desenvolvimento, detalha-se como os componentes foram modelados, configurados e integrados para a criação da plataforma. Os resultados apresentam as funcionalidades do sistema, exemplos de simulações realizadas e discutem os resultados em relação aos objetivos do projeto. A discussão analisa os resultados obtidos, compara com trabalhos relacionados, discute as limitações do projeto e propõe sugestões para trabalhos futuros.

2 REFERENCIAL TEÓRICO

2.1 Realidade Virtual

A Realidade Virtual (RV) é uma tecnologia que cria ambientes tridimensionais simulados por computador, permitindo aos usuários uma interação imersiva e sensorial com esses ambientes. Esta tecnologia tem suas raízes nos anos 1960, com o desenvolvimento do "Sensorama" por Morton Heilig, e a criação do primeiro sistema de RV com display montado na cabeça por Ivan Sutherland e Bob Sproull nos anos 1960 (Heilig, 1962; Sutherland, 1968). A evolução tecnológica ao longo das décadas subsequentes, especialmente nos campos da computação gráfica e sensores, tem sido fundamental para tornar a RV mais acessível e aplicável em diversas áreas, incluindo a educação e treinamento.

Na educação, a RV é utilizada para criar experiências de aprendizado imersivas que melhoram a compreensão e a retenção de conceitos complexos. Por exemplo, em treinamentos médicos, a RV permite que estudantes pratiquem procedimentos cirúrgicos em um ambiente seguro e controlado, resultando em melhorias significativas nas habilidades práticas e na confiança dos alunos **(Built In, 2022)**. Em treinamentos de segurança, trabalhadores podem aprender a manejar equipamentos perigosos e responder a emergências sem os riscos associados ao treinamento real (Mordor Intelligence, 2022).

Estudos recentes indicam que a utilização da RV pode acelerar o processo de aprendizado. Um estudo da Penn State University mostrou que estudantes que utilizam RV para aprender tarefas complexas completam essas tarefas duas vezes mais rápido do que aqueles que utilizam métodos tradicionais (World Economic Forum, 2021). Essa eficácia é atribuída à capacidade da RV de fornecer feedback imediato e permitir a prática repetida em um ambiente seguro.

A aplicação da RV no ensino e treinamento tem demonstrado ser uma ferramenta poderosa para a educação moderna, oferecendo uma experiência imersiva e interativa que transcende as limitações do aprendizado tradicional. A evolução dessa tecnologia possibilitou que estudantes e profissionais pratiquem habilidades em ambientes seguros e controlados, resultando em um aprendizado mais eficaz e envolvente.

Para avançar nessa discussão, é fundamental explorar um conceito estreitamente relacionado e igualmente inovador: os gêmeos digitais. Enquanto a RV proporciona um ambiente de aprendizado imersivo, os gêmeos digitais complementam essa experiência ao oferecer uma réplica virtual detalhada e funcional de sistemas físicos reais, permitindo uma análise aprofundada e uma interação em tempo real.

2.2 Gêmeos Digitais

Os gêmeos digitais são representações virtuais de objetos, processos ou sistemas físicos, continuamente atualizados com dados em tempo real. Este conceito, popularizado por Michael Grieves em 2002, tem ganhado destaque na Indústria 4.0 devido à sua capacidade de melhorar a eficiência operacional e prever falhas (Grieves & Vickers, 2017).

Na prática industrial, os gêmeos digitais são utilizados para monitorar e otimizar operações, permitindo simulações detalhadas de cenários operacionais e facilitando a tomada de decisões baseada em dados. Por exemplo, na manufatura, esses gêmeos permitem prever a necessidade de manutenção de máquinas, reduzindo o tempo de inatividade e prolongando a vida útil dos equipamentos (Boschert & Rosen, 2016). Na indústria de energia, eles permitem simular operações complexas e testar diferentes estratégias de produção antes de implementá-las, aumentando a eficiência e reduzindo custos (Fuller et al., 2020).

No contexto educacional, os gêmeos digitais oferecem uma plataforma interativa para que estudantes explorem sistemas complexos de maneira prática e segura. Estudantes de engenharia, por exemplo, podem utilizar gêmeos digitais para simular processos de fabricação e estudar o comportamento de diferentes materiais. Esta abordagem prática de "aprender fazendo" é comprovadamente eficaz para a educação técnica (Fuller et al., 2020).

A integração de gêmeos digitais com tecnologias de RV cria um ambiente de aprendizado robusto e dinâmico, onde a teoria e a prática se encontram de maneira fluida. Por exemplo, enquanto a RV pode transportar os estudantes para um ambiente de fábrica virtual, os gêmeos digitais podem fornecer dados em tempo real sobre as operações dessa fábrica, permitindo uma análise mais aprofundada e uma compreensão mais completa dos processos industriais.

2.3 Controle de Processos

A teoria de controle de processos é um campo da engenharia que se concentra na modelagem e regulação do comportamento dinâmico de sistemas. O objetivo principal é manter as variáveis de um processo dentro de limites desejados, garantindo que o sistema opere de maneira eficiente e segura. Isso é alcançado através de técnicas que ajustam automaticamente as entradas do sistema com base em medições das saídas, utilizando controladores projetados especificamente para esta finalidade (Åström & Murray, 2008).

Os fundamentos da teoria de controle envolvem a criação de modelos matemáticos que descrevem o comportamento de sistemas dinâmicos. Esses modelos são frequentemente

expressos por meio de equações diferenciais que representam a relação entre as entradas, estados e saídas do sistema. A partir desses modelos, técnicas de análise e design, como resposta em frequência, análise de estabilidade e síntese de controladores, são aplicadas para garantir que o sistema responda de maneira desejada às perturbações e mudanças de referência (Ogata, 2010).

Entre as variáveis mais medidas e controladas na indústria estão a pressão, temperatura, nível e vazão. A pressão é frequentemente monitorada em processos de fabricação e transporte de gases e líquidos, sendo controlada para garantir a segurança e eficiência operacional. Sensores de pressão são usados para medir a pressão em tanques e tubos, ajustando válvulas para manter níveis adequados (Åström & Murray, 2008). A temperatura é outra variável crucial, controlada em processos como a pasteurização e refrigeração. Termopares e termistores medem a temperatura, enquanto controladores ajustam fontes de calor ou resfriamento para manter a temperatura desejada (Ogata, 2010). O nível de líquidos em tanques também é monitorado para evitar transbordamentos ou operações em vazio, utilizando sensores de nível e controladores de bombas (Seborg et al., 2010). A vazão, por sua vez, é medida em sistemas de distribuição de água e produtos químicos, com sensores de vazão e válvulas de controle ajustando a taxa de fluxo para garantir uma distribuição consistente e eficiente (Dorf & Bishop, 2011).

A análise desses sistemas, por sua vez, é realizada por meio de diversas ferramentas, como o diagrama de Bode, a resposta ao degrau, gráficos de tempo, a Transformada Rápida de Fourier (FFT) e os plots de pólos e zeros. Essas técnicas combinadas permitem uma compreensão profunda do comportamento dos sistemas de controle, facilitando o design e a implementação de soluções eficazes para manter a eficiência e a segurança dos processos industriais.

A análise de sistemas de controle de processos é fundamentada em diversas ferramentas que permitem compreender e projetar a resposta do sistema a diferentes entradas. Uma dessas ferramentas é o diagrama de Bode, que mostra a resposta em frequência de um sistema e ajuda a analisar a estabilidade e a robustez do controle frente a perturbações (Ogata, 2010). A resposta ao degrau, por outro lado, é uma técnica que avalia a resposta temporal do sistema a uma entrada em degrau, fornecendo informações cruciais sobre o comportamento transitório e o regime permanente do sistema (Åström & Murray, 2008). Gráficos de tempo são frequentemente utilizados para visualizar a evolução de variáveis do processo ao longo do tempo, facilitando a identificação de tendências e anomalias. A Transformada Rápida de Fourier (FFT) é empregada para decompor sinais em suas componentes de frequência, sendo útil na análise de vibrações e ruídos em sistemas industriais (Seborg et al., 2010). Além disso, os plots de polos e zeros são

ferramentas gráficas que exibem a localização dos polos e zeros de uma função de transferência no plano complexo, fornecendo insights sobre a estabilidade e a dinâmica do sistema (Dorf & Bishop, 2011). Essas técnicas analíticas combinadas permitem uma compreensão profunda e detalhada do comportamento dos sistemas de controle, facilitando o design e a implementação de soluções eficazes para manter a eficiência e a segurança dos processos industriais.

2.4 Tecnologias Utilizadas

2.4.1 Unity

Unity é uma plataforma de desenvolvimento de software usada para criar experiências interativas em 2D e 3D. É amplamente utilizada no desenvolvimento de jogos, simulações e aplicações de realidade aumentada (RA) e RV. Unity permite a criação de gráficos de alta qualidade e interação em tempo real, oferecendo uma biblioteca extensa de ferramentas e componentes reutilizáveis e suporte a múltiplas plataformas, como Windows, macOS, iOS, Android, e dispositivos de VR/AR (Unity Technologies, 2023).

A plataforma Unity fornece um ambiente integrado que facilita o desenvolvimento, permitindo que desenvolvedores criem mundos virtuais detalhados e adicionem física realista aos objetos. Ela também oferece uma série de ferramentas para animação, áudio, inteligência artificial, e rede, o que facilita a criação de jogos e aplicações complexas. A Asset Store do Unity é um repositório de recursos onde desenvolvedores podem encontrar e compartilhar scripts, modelos 3D, texturas e outros componentes que aceleram o desenvolvimento.

Além disso, Unity suporta o desenvolvimento de aplicações de RV e RA, permitindo a criação de experiências imersivas. Com o Unity, desenvolvedores podem criar protótipos rapidamente, testar novas ideias e implementar feedback iterativamente, o que é crucial para o desenvolvimento ágil.

2.4.2 Docker

Docker é uma ferramenta de contêinerização que permite empacotar uma aplicação e todas as suas dependências em um contêiner. Isso garante que a aplicação funcione de maneira consistente em qualquer ambiente, independentemente das configurações do sistema operacional subjacente (Merkel, 2014). Docker utiliza contêineres, que são ambientes isolados e leves que compartilham o mesmo kernel do sistema operacional, mas possuem seu próprio sistema de arquivos, processos e rede.

Uma das principais vantagens do Docker é a portabilidade dos contêineres, que podem ser facilmente movidos entre diferentes sistemas, desde o computador do desenvolvedor até servidores de produção na nuvem. Isso reduz significativamente os problemas de compatibilidade e acelera o ciclo de desenvolvimento e implantação. Docker também facilita a escalabilidade das aplicações, permitindo o gerenciamento eficiente de múltiplos contêineres em diferentes ambientes.

Docker utiliza o Dockerfile, um script simples que contém as instruções para construir um contêiner, especificando o sistema operacional base, as dependências necessárias e as etapas para configurar a aplicação. Além disso, o Docker Hub é um repositório onde desenvolvedores podem compartilhar e acessar imagens de contêineres pré-construídas, o que agiliza o processo de desenvolvimento.

2.4.2.1 Rede Bridge em Docker

A rede bridge é um dos tipos de rede padrão do Docker. Ela permite que os contêineres conectados a ela se comuniquem entre si, enquanto permanecem isolados do restante do sistema e de outras redes. A rede bridge facilita a configuração de ambientes de desenvolvimento onde múltiplos contêineres precisam interagir, como um contêiner de aplicação web comunicando-se com um contêiner de banco de dados.

2.4.2.2 Volumes em Docker

Volumes em Docker são utilizados para persistir dados gerados e utilizados por contêineres. Eles são armazenados fora do contêiner, permitindo que os dados sobrevivam a reinicializações e exclusões de contêineres. Volumes são essenciais para gerenciar dados de maneira eficiente e segura, especialmente em aplicações que exigem armazenamento persistente, como bancos de dados.

2.4.3 Node.js

Node.js é uma plataforma de runtime que permite executar código JavaScript no servidor. Isso possibilita a criação de aplicações web altamente escaláveis e eficientes, pois o Node.js utiliza um modelo de execução não bloqueante baseado em eventos, o que permite tratar muitas conexões simultaneamente sem sobrecarregar o servidor (Tilkov e Vinoski, 2010). Este modelo é particularmente útil para aplicações em tempo real, como chats, jogos online e sistemas de monitoramento.

Node.js possui um ecossistema rico de módulos e pacotes disponíveis através do npm (Node Package Manager), facilitando a adição de funcionalidades como autenticação de usuários, acesso a bancos de dados, e comunicação em tempo real. A utilização de JavaScript tanto no lado do cliente quanto no servidor permite que os desenvolvedores escrevam código mais coeso e reutilizável, melhorando a eficiência do desenvolvimento e a manutenção do código.

2.4.4 MongoDB

MongoDB é um banco de dados NoSQL que armazena dados em documentos flexíveis no formato JSON, ao invés de usar tabelas e linhas como em bancos de dados relacionais tradicionais (Chodorow, 2013). Essa estrutura permite armazenar e manipular dados complexos de maneira mais eficiente e escalável. MongoDB é especialmente útil em aplicações que lidam com grandes volumes de dados variáveis e que precisam ser capazes de se adaptar rapidamente a mudanças nos requisitos de dados.

Um dos pontos fortes do MongoDB é sua capacidade de distribuir dados em vários servidores, o que melhora a performance e a disponibilidade da aplicação. Além disso, ele oferece funcionalidades como índices completos, consultas ad-hoc, e suporte para operações de agregação, que permitem manipular e consultar os dados de maneira poderosa e flexível. Isso torna o MongoDB uma escolha popular para aplicações web modernas, análise de dados, e projetos de Big Data .

2.4.5 MQTT

MQTT (Message Queuing Telemetry Transport) é um protocolo de comunicação leve projetado para conectar dispositivos em redes com largura de banda limitada e alta latência, como aquelas utilizadas em projetos de Internet das Coisas (IoT). Funciona no modelo de publicação/assinatura, onde dispositivos (clientes) podem publicar mensagens em tópicos específicos e outros dispositivos podem se inscrever nesses tópicos para receber as mensagens (Hunkeler, Truong e Stanford-Clark, 2008).

A eficiência e a baixa sobrecarga do MQTT o tornam ideal para cenários onde a largura de banda e o consumo de energia são preocupações importantes. Ele é frequentemente usado em sistemas de automação residencial, monitoramento remoto, e redes de sensores. Além disso, o MQTT é robusto o suficiente para garantir a entrega de mensagens mesmo em condições de rede não confiáveis, proporcionando uma comunicação confiável entre dispositivos conectados.

2.4.6 ESP32 e ESP01

ESP32 e ESP01 são microcontroladores desenvolvidos pela Espressif Systems, amplamente utilizados em projetos de Internet das Coisas (IoT) devido ao seu baixo custo e alta funcionalidade. O ESP32 é um microcontrolador avançado com Wi-Fi e Bluetooth integrados, múltiplos núcleos de processamento, e uma vasta gama de interfaces periféricas, tornando-o ideal para projetos complexos de IoT que requerem conectividade e processamento robustos(Espressif Systems, 2023).

O ESP01 é uma versão mais simples e compacta do ESP8266, focada em conectividade Wi-Fi. Ele é frequentemente utilizado em projetos que necessitam apenas de funcionalidades básicas de conectividade, como automação residencial e dispositivos conectados. Apesar de suas limitações em comparação com o ESP32, o ESP01 é extremamente popular devido ao seu tamanho reduzido e facilidade de uso.

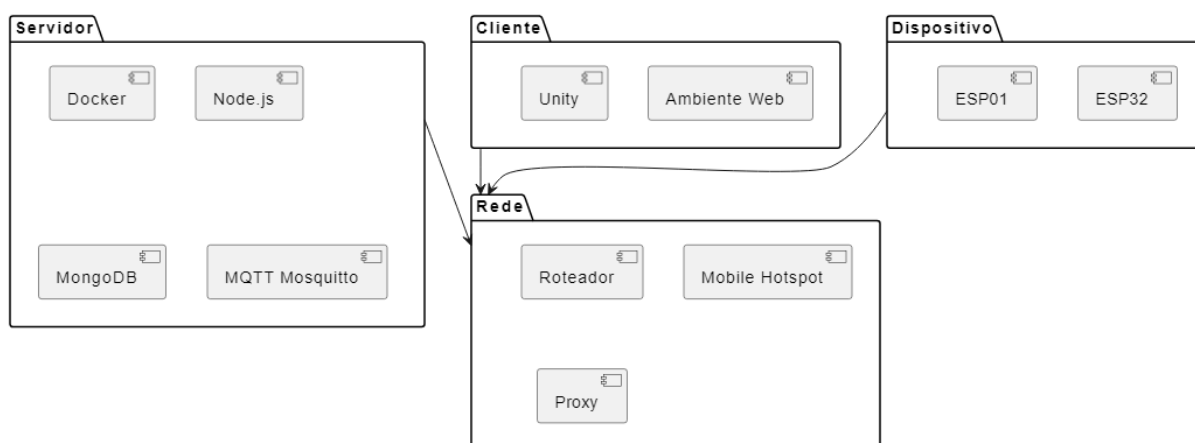
Ambos os microcontroladores são programáveis utilizando a IDE do Arduino, facilitando o desenvolvimento e a implementação de projetos IoT. Eles suportam a comunicação via MQTT, permitindo a integração eficiente com sistemas de monitoramento e controle remoto. A combinação de baixo custo, facilidade de uso e versatilidade faz com que o ESP32 e o ESP01 sejam escolhas populares para desenvolvedores e entusiastas de IoT (Espressif Systems, 2023).

3 MATERIAIS E MÉTODOS

3.1 Materiais da estruturais da plataforma

Para a realização deste projeto, foi montada uma infraestrutura robusta que integra vários componentes essenciais, distribuídos em quatro categorias principais: cliente, servidor, dispositivo e rede. A Figura 1 a seguir ilustra a disposição desses componentes e suas interconexões no sistema:

Figura 1 - Disposição dos componentes da plataforma.



Fonte: Autoria Própria

3.1.1 Cliente:

Ambiente web ou Unity: O cliente é o dispositivo que interage diretamente com a parte visual da plataforma, podendo ser um computador, tablet, celular ou notebook. Este dispositivo executa o ambiente web ou o ambiente Unity, proporcionando a interface gráfica e as funcionalidades de interação ao usuário.

3.1.2 Servidor:

Dispositivo rodando Docker: O servidor é responsável por gerenciar a comunicação entre todos os componentes do sistema. Neste projeto, o servidor foi configurado em um dispositivo que executa containers Docker, podendo ser um computador, notebook ou um Raspberry Pi 4. Este servidor hospeda serviços essenciais como o Node.js, MongoDB e o broker MQTT Mosquitto.

3.1.3 Dispositivo:

Microcontroladores: Os dispositivos utilizados para programação e interação com o sistema incluem o ESP32 e o ESP01. Esses microcontroladores são programados

para coletar dados de sensores e controlar atuadores, enviando e recebendo informações via MQTT para o servidor.

3.1.4 Modem/Roteador (Rede):

Rede Local: Todos os componentes (cliente, servidor e dispositivos) estão conectados através de uma rede local, que pode ser composta por um roteador comum, um celular com mobile hotspot ou redes diferentes interligadas por um proxy utilizando remote.it. Essa infraestrutura de rede garante a comunicação eficiente e sincronizada entre todos os elementos do sistema.

3.2 Ambiente de Desenvolvimento

Para a implementação do ambiente virtual e a configuração do sistema, foi necessário utilizar uma variedade de hardwares e softwares especializados, cada um contribuindo de forma distinta para o desenvolvimento do projeto. A seguir, detalhamos os recursos utilizados em cada etapa do processo.

3.2.1 Hardware e Software Utilizados

A configuração do ambiente de desenvolvimento exigiu uma combinação de dispositivos de hardware e ferramentas de software, cada um desempenhando um papel crítico no processo de criação, simulação e controle do sistema.

- **Unity:** O Unity foi a principal ferramenta utilizada para a criação e simulação do ambiente virtual. Escolhido devido à sua capacidade de gerar protótipos rápidos e por seu suporte robusto a gráficos 3D e realidade aumentada/virtual. O Unity também facilita a integração com outros componentes do sistema, como a biblioteca MQTT.
- **Blender:** Algumas geometrias específicas do ambiente virtual foram modeladas no Blender, uma ferramenta poderosa e flexível para criação de conteúdo 3D. O Blender complementa o Unity na modelagem de objetos detalhados que requerem maior precisão.
- **Visual Studio Code:** Foi selecionada como IDE para o desenvolvimento do código, e utilizado em várias etapas do projeto, desde a escrita de scripts em Node.js até a programação dos microcontroladores devido sua vasta gama de extensões e suporte a múltiplas linguagens o torna ideal para projetos integrados.

- **IDE do Arduino:** Para a programação dos microcontroladores ESP32 e ESP01, a IDE do Arduino foi a ferramenta utilizada, pois é um ambiente de desenvolvimento fácil de usar e permite a integração direta com uma ampla gama de sensores e atuadores.

Configuração e Acesso Remoto

A escrita do código e a configuração dos sistemas foram realizadas em diferentes dispositivos, utilizando métodos de acesso remoto para facilitar a colaboração e o desenvolvimento contínuo.

- **Raspberry Pi:** O Raspberry Pi serviu como uma estação de desenvolvimento acessível remotamente. Utilizado para executar a escrita e a depuração do código, permitindo o acesso remoto via SSH a partir de um computador. Com a extensão apropriada no Visual Studio Code, foi possível desenvolver e gerenciar os scripts diretamente no Raspberry Pi.
- **Computador pessoal dedicado:** Para as tarefas que exigiam maior poder de processamento, como a programação em Unity e a renderização de gráficos complexos, foi utilizado um computador de alta performance. Este computador suportava o desenvolvimento intensivo em Unity, garantindo a fluidez e a eficácia na criação do ambiente virtual.

Essa configuração de hardware e software permitiu uma abordagem flexível e escalável, adaptando-se às necessidades específicas de cada fase do projeto e garantindo a eficiência no desenvolvimento e na execução do ambiente virtual.

3.3 Desenvolvimento do Ambiente Virtual

O desenvolvimento do ambiente virtual envolveu a utilização de ferramentas avançadas de modelagem 3D e programação, permitindo a criação de uma simulação precisa e interativa da MPS-PA Compact Workstation. Este processo foi realizado principalmente com a utilização do Unity 3D e do Blender, integrando diversas tecnologias para assegurar uma comunicação eficiente e uma experiência de usuário intuitiva.

3.3.1 Modelagem e Implementação no Unity

Para recriar o ambiente da MPS-PA Compact Workstation de forma virtual, foi necessário seguir um processo detalhado de modelagem e integração, utilizando os seguintes recursos e ferramentas:

- **Unity 3D:** O Unity 3D foi escolhido como a principal plataforma para a prototipagem e implementação do ambiente virtual devido à sua capacidade de criar rapidamente

modelos interativos e dinâmicos. A ferramenta **ProBuilder**, dentro do Unity, facilitou a construção e a edição direta de geometrias complexas durante a fase de desenvolvimento, permitindo ajustes em tempo real e testes de funcionalidades.

- **Blender:** Algumas partes específicas e detalhadas da estação de trabalho foram modeladas no Blender antes de serem importadas para o Unity. O Blender foi utilizado para criar objetos tridimensionais complexos e precisos que exigiam uma modelagem detalhada, complementando a rapidez do ProBuilder no Unity.
- **Integração MQTT:** Para facilitar a comunicação entre os componentes físicos e o ambiente virtual, foi integrada a biblioteca **M2MqttUnityClient** (VIGANO, 2021). Esta biblioteca permite que o Unity se conecte a um broker MQTT, recebendo e enviando dados em tempo real. Isso é crucial para a simulação de cenários em que os microcontroladores interagem com a interface virtual.
- **TextMeshPro:** A interface de usuário dentro do Unity foi aprimorada com a utilização do **TextMeshPro** juntamente com os recursos de canvas, que oferece opções avançadas de renderização de texto. Essa ferramenta foi crucial para apresentar informações claras e legíveis ao usuário, proporcionando uma interface gráfica rica e interativa.

3.4 Integração dos Sistemas

A integração dos sistemas foi gerida por um servidor Node.js, responsável por unificar os serviços executados em containers Docker. Entre esses serviços, incluíram-se o broker MQTT Mosquitto, um serviço de simulação em C++, e o banco de dados MongoDB. A configuração do Docker envolveu a criação de uma rede bridge para assegurar a comunicação eficiente entre os containers e a utilização de volumes locais para a persistência de dados. O servidor Node.js utilizou o framework Express para criar rotas de API, facilitando a interação entre o ambiente virtual e os dispositivos físicos.

A integração dos diversos sistemas que compõem o projeto é um processo crucial que assegura a comunicação eficiente e a operação harmoniosa entre todos os componentes. Esta seção detalha como os serviços e componentes foram integrados para formar um sistema coeso e funcional.

3.4.1 Gestão da Integração com Node.js e Docker

O núcleo da integração dos sistemas foi gerido por um servidor Node.js, que desempenha um papel central na unificação e coordenação dos diferentes serviços. Aqui está um detalhamento de como essa integração foi estruturada:

- **Node.js como Orquestrador:** O servidor Node.js foi escolhido por sua capacidade de gerenciar múltiplas conexões simultâneas e sua compatibilidade com uma vasta gama de pacotes e frameworks. Ele serviu como o orquestrador principal, gerenciando as interações entre os componentes do sistema, como o ambiente virtual, os microcontroladores e os serviços em containers Docker.
- **Containers Docker:** Docker foi utilizado para containerizar os serviços críticos, o que facilita a manutenção, a escalabilidade e a portabilidade do sistema. Dentro dos containers, foram executados serviços essenciais como:
 - **MQTT Mosquitto:** O broker MQTT Mosquitto é responsável por gerenciar a comunicação de mensagens entre os dispositivos conectados e o ambiente virtual, utilizando o protocolo MQTT para transmitir dados de forma eficiente e em tempo real.
 - **Serviço de Simulação em C++:** Um serviço de simulação desenvolvido em C++ foi containerizado para realizar cálculos complexos ou simulações que suportam o ambiente virtual.
 - **MongoDB:** O banco de dados MongoDB, também executado em um container Docker, foi utilizado para armazenar dados persistentes e estados do sistema, garantindo que todas as informações necessárias fossem mantidas e acessíveis.

Configuração do Docker: Para assegurar que todos os containers pudessem se comunicar de forma eficiente, foi criada uma rede bridge dentro do Docker. Essa configuração permite que os containers compartilhem informações e serviços através de uma rede isolada, melhorando a segurança e o desempenho do sistema. Além disso, a utilização de volumes locais permitiu a persistência de dados, mesmo em caso de reinicialização ou atualização dos containers.

3.4.2 Criação de APIs com Express

O framework Express foi utilizado para desenvolver as rotas de API no servidor Node.js. Essas APIs são fundamentais para a interação entre o ambiente virtual e os dispositivos físicos. A seguir, descrevemos como as APIs foram configuradas e suas funções principais:

- **Roteamento de API:** O Express foi utilizado para criar rotas que facilitam a comunicação entre o ambiente virtual e os microcontroladores. Essas rotas permitem que o ambiente virtual envie comandos para os dispositivos físicos e receba dados de sensores e atuadores.
- **Gestão de Estados e Eventos:** As APIs geridas pelo Express permitem o monitoramento e a atualização do estado do sistema em tempo real. Isso inclui a recepção de dados dos microcontroladores e a emissão de comandos baseados nas interações do usuário com o ambiente virtual.
- **Segurança e Autenticação:** Para garantir a segurança das comunicações e a integridade dos dados, foram implementados mecanismos de autenticação e autorização nas rotas de API, assegurando que apenas dispositivos e usuários autorizados pudessem acessar e modificar o sistema.

3.5 Desenvolvimento do Controlador

Os microcontroladores ESP32 e ESP01 foram programados utilizando a IDE do Arduino. Para a integração com sensores e atuadores, foram utilizados jumpers, protoboards, módulos ESP01-S Relay V0.1, programadores ESP01 e módulos joystick para simular sensores e LEDs.

[adicionar imagem dos componentes e fotinhas com as montagens]

O desenvolvimento do controlador é uma etapa vital do projeto, pois envolve a programação e a configuração dos microcontroladores que interagem diretamente com os sensores e atuadores. Estes dispositivos são essenciais para coletar dados e executar ações no sistema, integrando-se com o ambiente virtual para criar uma simulação interativa e dinâmica.

3.5.1 Seleção e Configuração dos Microcontroladores

Os microcontroladores ESP32 e ESP01 foram escolhidos devido às suas capacidades avançadas e flexibilidade para comunicação e controle. Aqui está um detalhamento de como esses microcontroladores foram configurados e utilizados no projeto:

- **ESP32:** O ESP32 é um microcontrolador robusto, com capacidade de processamento suficiente para suportar tarefas complexas e múltiplas conexões. Ele foi utilizado principalmente para tarefas que requerem uma alta taxa de transferência de dados ou a execução de cálculos complexos. Suas características incluem Wi-Fi integrado, Bluetooth, múltiplas entradas e saídas digitais e analógicas, tornando-o ideal para a coleta e transmissão de dados de sensores para o servidor.

- **ESP01:** O ESP01 é uma versão compacta e mais econômica do ESP8266, ideal para tarefas de controle simples e comunicação básica via Wi-Fi. No projeto, ele foi empregado para operações de controle direto de atuadores e transmissão de dados de sensores em aplicações onde um controlador mais leve é suficiente.

3.5.2 Ferramentas e Procedimentos de Programação

Utilizando a IDE do Arduino foi realizada a programação dos microcontroladores, para isso os seguintes elementos foram utilizados para a programação e integração dos controladores:

- **IDE do Arduino:** Esta ferramenta foi utilizada para escrever e carregar o código nos microcontroladores. A IDE do Arduino oferece uma vasta biblioteca de funções pré-definidas, simplificando a programação de operações complexas, como a leitura de sensores e o controle de atuadores. O ambiente de desenvolvimento foi configurado para suportar os microcontroladores ESP32 e ESP01, com bibliotecas específicas instaladas para cada modelo.

Figura 2- Exemplo de Código na IDE do Arduino.



```

pumpRelay
22 WiFiClient wifiClient;
23 PubSubClient client(server, 8883, NULL, wifiClient);
24
25 void setup() {
26   pinMode(RELAY, OUTPUT);
27   digitalWrite(RELAY, LOW);
28
29   WifiConnect();
30   MqttConnect();
31 }
32
33 void WifiConnect(){
34   WiFi.begin(ssid, password);
35   do{delay(500);}
36   while(WiFi.status() != WL_CONNECTED);
37 }
38
39 void MqttConnect(){
40   do{delay(500);}
41   while (!client.connect(clientId, mqttusr, token,EVT(status),1,true,"PUMPoffline"));
42
43   client.publish(EVT(status), "PUMPonline", true);
44
45   client.setCallback(callback);
46   client.subscribe(CMD(pumpCMD));

```

(aka nodemcu), 28 MHz, 40MHz, DOUT (compatible), 1MB (FS:64KB OTA:~470KB), 2, nonos-sdk 2.2.1+100 (190703), v2 Lower Memory, Disabled, None, Only Sketch, 115200 on COM8

Fonte: Autoria Própria

- **Jumpers e Protoboards:** Para a interconexão dos microcontroladores com os sensores e atuadores, foram utilizados jumpers e protoboards. Esses componentes permitem a

montagem rápida e flexível de circuitos sem a necessidade de soldagem, facilitando a prototipagem e testes.

- **Módulos ESP01-S Relay V0.1:** Esses módulos de relé foram usados para representar o controle de dispositivos de maior potência a partir dos microcontroladores. Eles atuam como intermediários que permitem que os microcontroladores acionem e desativem dispositivos conectados, como motores ou lâmpadas, através de sinais de controle de baixa potência.
- **Programadores ESP01:** Para carregar o código nos módulos ESP01, foram utilizados programadores específicos que permitem a interface direta entre o ESP01 e a IDE do Arduino. Esses programadores simplificam o processo de upload de firmware e facilitam a configuração inicial dos microcontroladores.
- **Módulos Joystick e LEDs:** Para simular entradas de sensores e saídas de atuadores, foram utilizados módulos joystick e LEDs. Os joysticks permitem simular entradas analógicas que poderiam ser representativas de sensores reais, enquanto os LEDs fornecem uma maneira visual de representar saídas de controle, como o acionamento de luzes ou sinalizadores.

3.5.3 Processo de Programação e Integração

1. **Escrita do Código:** A primeira etapa envolveu a escrita de código para cada microcontrolador na IDE do Arduino. O código foi projetado para realizar tarefas específicas, como ler dados de sensores ou enviar comandos para atuadores. Cada microcontrolador foi programado para se comunicar com o servidor através do protocolo MQTT, enviando e recebendo mensagens conforme necessário.
2. **Testes e Depuração:** Após a escrita do código, os microcontroladores foram conectados a protoboards e testados em conjunto com os sensores e atuadores. A depuração foi realizada para garantir que os microcontroladores funcionassem corretamente e se comunicassem de forma eficiente com o servidor e o ambiente virtual.
3. **Integração Final:** Uma vez que os testes iniciais foram concluídos, os microcontroladores foram integrados ao sistema completo, conectando-os à rede e aos outros componentes do projeto. Esse processo envolveu a verificação da comunicação contínua via MQTT e a sincronização das operações entre o ambiente virtual e os dispositivos físicos.

3.6 Desenvolvimento da Interface Web

Nesta seção, são descritos os materiais, métodos e procedimentos utilizados no desenvolvimento da interface web, que possibilita a interação do usuário com o ambiente virtual. A interface foi projetada para oferecer suporte a dispositivos de realidade virtual (RV) e garantir a comunicação em tempo real com o servidor.

3.6.1 Materiais Utilizados

1. Ferramentas de Desenvolvimento:

IDE Visual Studio Code: Utilizado para escrever e depurar o código da interface. A ferramenta suporta uma variedade de extensões que auxiliam no desenvolvimento da aplicação.

Navegadores: Usados para testar e verificar a compatibilidade e o desempenho da interface web. Utilizou-se para teste Chrome, Edge, Meta Quest Browser, Chromium no raspberry PI.

2. Ferramentas de Comunicação em Tempo Real:

WebSocket: Protocolo utilizado para estabelecer uma conexão bidirecional entre o cliente (navegador) e o servidor, facilitando a atualização constante de dados.

3.6.2 Métodos e Procedimentos

1. Planejamento e Definição de Requisitos:

Definição dos requisitos funcionais e não funcionais da interface, incluindo a necessidade de renderização 3D interativa, suporte a dispositivos de RV e comunicação contínua com o servidor.

Planejamento da arquitetura da interface de maneira modular para facilitar a manutenção e a possibilidade de futuras expansões. Utilizou-se uma árvore de diretórios que separa as diversas funcionalidades e recursos em pastas e módulos *.js com funcionalidades implementadas separadamente.

2. Implementação da Renderização 3D:

Inicialização do Three.js: Configuração inicial da biblioteca Three.js, que inclui a criação da cena, da câmera e do renderizador.

Modelagem e Importação de Objetos: Criação ou importação de modelos 3D que representam os componentes do sistema e sua inclusão na cena.

Animação e Interação: Desenvolvimento de funcionalidades que permitem aos usuários manipular objetos 3D usando entradas como através das mãos com o uso do oculus metaquest2, ou do controle, ou até mesmo utilizando teclado/mouse na versão de navegador.

3. Configuração da Comunicação em Tempo Real:

Implementação de WebSocket: Utilizando o recurso *socket.io* para estabelecimento de uma conexão WebSocket para facilitar a comunicação contínua com o servidor, permitindo a troca de dados em tempo real.

Sincronização da Interface: Desenvolvimento de procedimentos para manter a interface web sincronizada com o estado dos dispositivos físicos.

4 DESENVOLVIMENTO

4.1 Requisitos do Sistema

Para desenvolver um ambiente virtual interativo utilizando tecnologias de Realidade Virtual e Digital Twins que auxilie no ensino de conceitos de controle de processos, foram definidos os seguintes requisitos:

- Implementar um ambiente virtual no Unity que permita a visualização e manipulação de sistemas de controle de processos.
- Desenvolver um servidor em Node.js para gerenciar a comunicação entre diferentes componentes do sistema, incluindo a interface de usuário e os dispositivos de entrada.
- Integrar o protocolo MQTT para a comunicação em tempo real entre o ambiente virtual e os dispositivos de controle.
- Utilizar MongoDB para armazenar dados de simulação e interações do usuário.
- Implementar contêineres Docker para facilitar a implantação e a gestão dos componentes do sistema.

As capacidades esperadas para o usuário incluem:

- Um ambiente que permita a visualização e manipulação de sistemas de controle de processos.
- Um sistema que possibilita a visualização da dinâmica do processo e a leitura dos sinais em tempo real.
- Interação com dispositivos físicos e simulações virtuais.
- Programação de microcontroladores para interagir com o sistema, enviando e recebendo sinais, podendo atuar como controladores.

4.2 Implementação do Ambiente Virtual no Unity

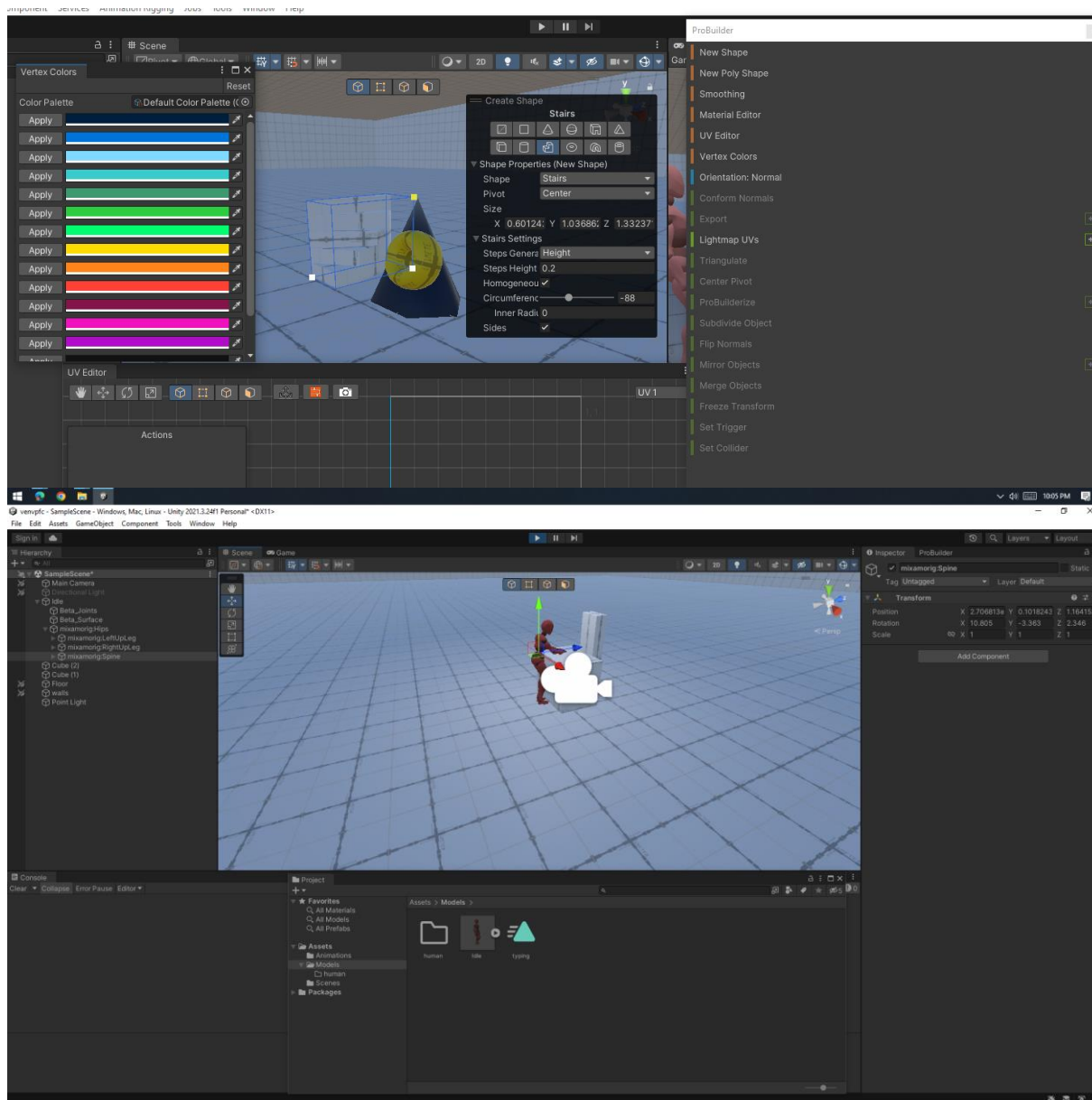
4.2.1 Modelagem 3D dos Componentes

A construção do ambiente virtual envolveu uma série de etapas práticas, focando na modelagem, configuração e integração de componentes tridimensionais, utilizando uma combinação estratégica de ferramentas especializadas.

Na fase inicial, a modelagem foi realizada com o ProBuilder, uma ferramenta nativa do Unity, que permite a criação rápida de protótipos diretamente no ambiente de desenvolvimento. O

ProBuilder facilita a modelagem de elementos básicos através de placeholders, que são versões simplificadas dos modelos finais. Essa abordagem permitiu ajustes rápidos e uma compreensão clara da disposição espacial do ambiente logo no início do projeto. A Figura [Fig. 1 - Interface do ProBuilder no Unity] ilustra a utilização do ProBuilder para a criação dos primeiros blocos que compõem a estrutura preliminar do ambiente.

Figura 3- Interface do ProBuilder no Unity para Modelagem 3D.



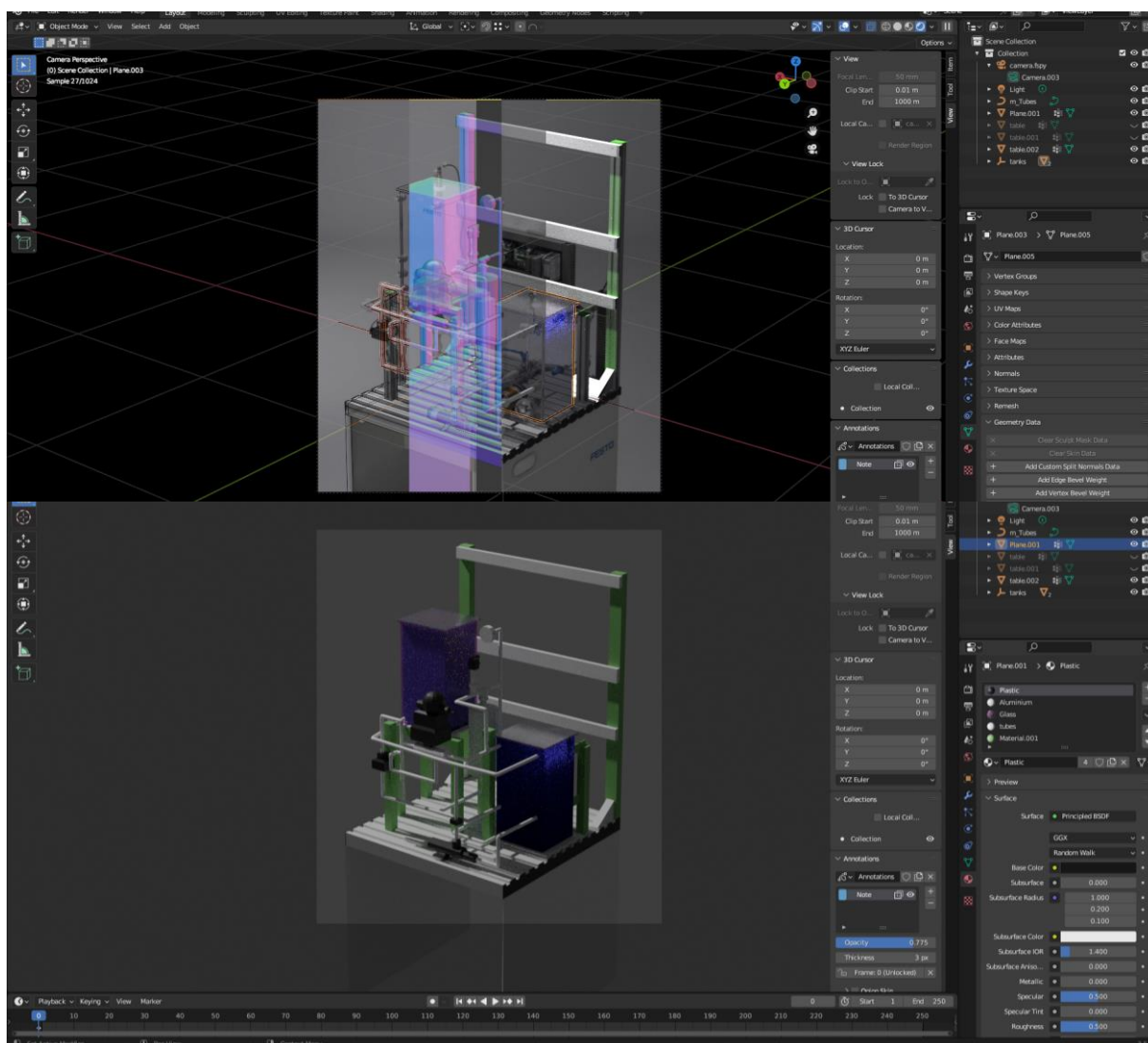
Fonte: Autoria Própria

Na Figura 3, observa-se como o ProBuilder permite a manipulação de formas geométricas simples, essenciais para a prototipagem e testes iniciais de layout. Além da presença de ferramentas para edição de faces, vértices arestas, textura UV, colorização entre outras. Este

método possibilitou uma rápida interação sobre a configuração do ambiente, oferecendo uma base sólida para o desenvolvimento subsequente.

Após a fase de prototipagem com placeholders, o processo avançou para a criação de modelos mais detalhados e finais utilizando o Blender, um software de modelagem 3D amplamente reconhecido por sua capacidade de criar modelos complexos e detalhados. No Blender, cada componente foi refinado para atender aos requisitos visuais e funcionais do projeto. A Figura 4 apresenta a interface do Blender com o modelo tridimensional do MPS-PA em diversas etapas do desenvolvimento através de uma colagem.

Figura 4 - Modelagem Detalhada no Blender dos Componentes 3D.



Fonte: Autoria Própria

Conforme ilustrado na, pode-se ver que o Blender oferece um ambiente robusto para a criação dos modelos, incluindo a aplicação de texturas, materiais e detalhes que possibilita a criação precisa e o realismo para colocação no ambiente virtual. Na figura em questão cores diferentes

garantiu uma abordagem eficiente e robusta para a criação do ambiente virtual, assegurando tanto a rapidez de desenvolvimento quanto a qualidade visual necessária.

4.2.2 Configuração dos Sistemas de Controle

A configuração dos sistemas de controle no Unity envolveu a criação de uma lógica para leitura dos nomes dos objetos na hierarquia através de expressões regulares, a fim de possibilitar identificar e manipular componentes no ambiente virtual de forma direta e genérica, para uma vasta integração com sistemas de controle.

Para viabilizar essa identificação automática, foram implementadas as âncoras, que funcionam como objetos de referência utilizados para controlar propriedades de outros elementos no ambiente virtual, como rotação, translação, escala e estados binários. As âncoras desempenham um papel fundamental na comunicação bidirecional entre o Unity e os dispositivos físicos (ESP32, ESP01, etc.), permitindo o controle simultâneo tanto pelo usuário, via interface gráfica, quanto pelo sistema físico, via sensores ou outros dispositivos externos.

Cada âncora é configurada automaticamente com base na análise dos nomes dos objetos presentes na hierarquia do Unity. Para isso, foi desenvolvida uma lógica de expressões regulares (REGEX) capaz de identificar e extrair as seguintes informações embutidas nos nomes:

- **Tipo de Transformação:** Determina se o controle será de rotação, translação, escala ou estado.
- **Eixos de Controle:** Define quais eixos (X, Y ou Z) serão afetados pela transformação.
- **Intervalo de Valores:** Estabelece os limites de controle, como ângulos de rotação ou posições mínimas e máximas.
- **Tipo de Controle de Entrada:** Indica a forma de controle, que pode ser via slider, toggle ou botão, facilitando a interação do usuário com o sistema.

A configuração bidirecional ocorre por meio do protocolo MQTT, onde o Unity e os microcontroladores (ESP32, ESP01, etc.) trocam informações de forma simultânea. Quando o usuário altera um controle na interface (por exemplo, deslizar um slider), o Unity publica uma mensagem MQTT para atualizar o dispositivo físico. Da mesma forma, quando o dispositivo físico detecta uma alteração (como um sensor mudando de estado), ele publica uma mensagem MQTT que é recebida pelo Unity, atualizando automaticamente o estado da âncora no ambiente virtual. Isso garante a sincronização dinâmica e mantém o sistema virtual e físico alinhados em tempo real.

Estrutura da Lógica

Primeiramente, desenvolveu-se um script em C# para ser utilizado dentro do Unity. O objetivo deste script era percorrer a hierarquia de objetos, localizar aqueles que possuem um padrão específico em seus nomes e configurar esses objetos como âncoras para controle. A expressão regular utilizada para identificar esses objetos foi a seguinte:

```
/#(.*)\.Anchor_([rtsuk])(?:([+-]?[0-9]*\.[0-9]+);?){0,4})(?:([ax])(?:[+-]?[0-9]*\.[0-9]+);?){3})?(?:([map])(?:[+-]?[0-9]*\.[0-9]+);?){2})?(?:([rng])(?:[+-]?[0-9]*\.[0-9]+);?){2})?(?:([slider|toggle|btn]))?.*$/
```

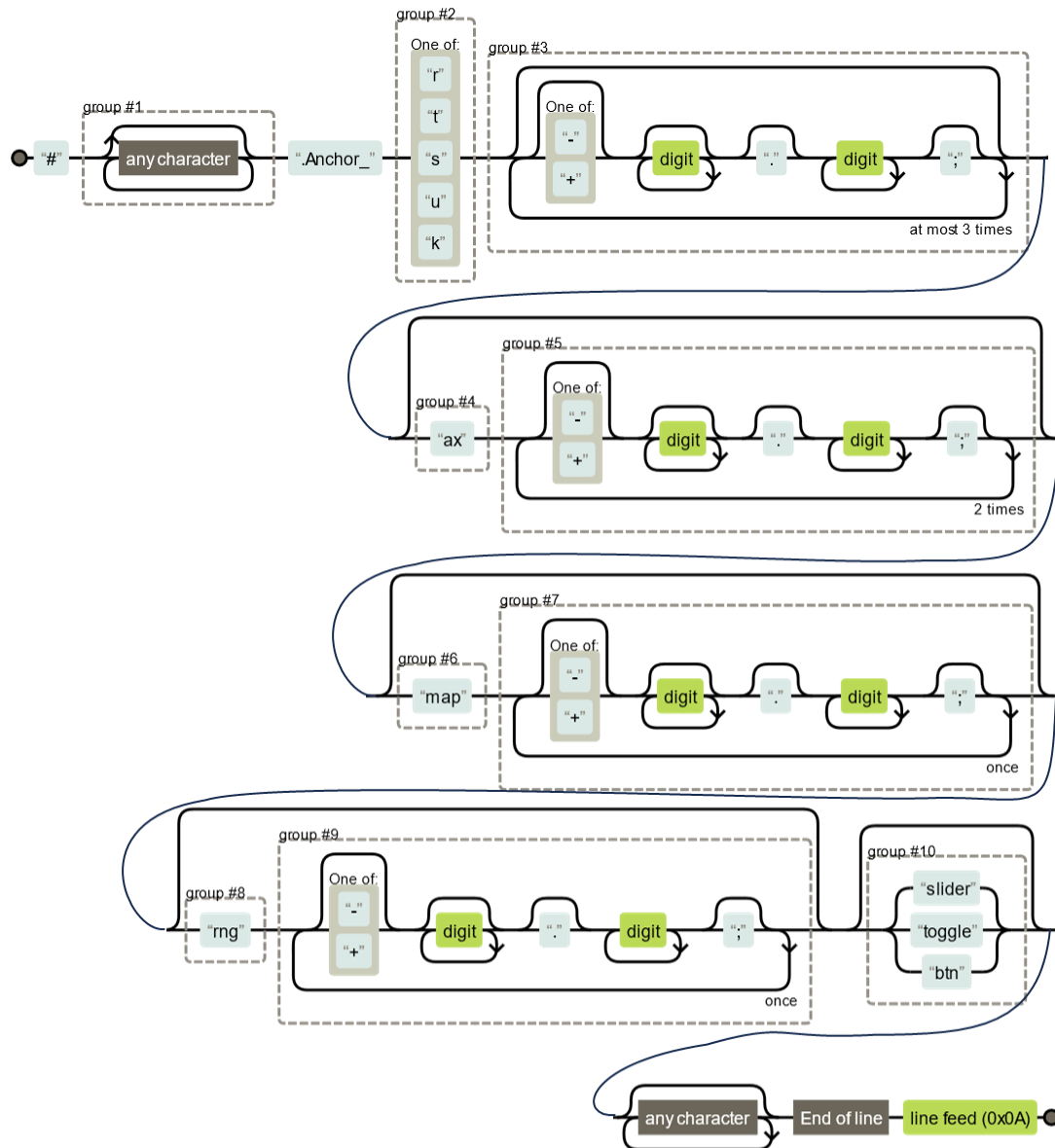
Para uma melhor compreensão da estrutura e dos componentes desta expressão, conforme o diagrama de trilha Figura 6, a expressão regular inicia com # seguido de qualquer caractere, identificando um padrão geral. Em seguida, busca-se por .Anchor_ seguido de um tipo de âncora, que pode ser r (rotação), t (translação), s (escalonamento), u (uniforme) ou k (chave de forma). A expressão também captura os eixos (ax) e mapeamentos (map) aplicáveis, além dos intervalos de valores (rng) caso se trate de uma medida que deseja-se limitar. Finalmente, identifica o tipo de entrada, que pode ser slider, toggle ou btn.

Esta expressão regular permite identificar vários componentes dos nomes dos objetos, como o tipo de âncora (rotação, translação, escalonamento, uniforme ou chave de forma), os eixos aplicáveis, mapeamentos e intervalos de valores, além do tipo de entrada (slider, toggle ou botão), e com ela transformações são realizadas no objeto a fim de reagir a uma entrada do usuário ou sinal recebido por um atuador através de sensores.

O processo de identificação e configuração das âncoras pode ser descrito utilizando os seguintes passos:

1. Início
2. Buscar "devices" na hierarquia
3. Para cada objeto filho
 - Aplicar regex para identificar âncoras
 - Se corresponder, criar dados de âncora
 - Registrar âncora para comunicação MQTT
4. Fim

Figura 6 - Diagrama de Expressão Regular para Identificação de Âncoras.



Fonte: Adaptado de de regexper.com (Jeff Avallone)

Fluxo do Processo de Identificação e Configuração

Conforme descrito, o processo começa com a inicialização do script, que percorre a hierarquia de objetos. Cada nome de objeto é verificado contra a expressão regular. Se corresponder, os dados são extraídos e a âncora é configurada. Finalmente, a âncora é registrada no sistema MQTT para permitir o controle remoto.

Controle de Âncoras Além do fluxo de processo, a lógica implementada para a criação de objetos de referência pode ser descrita utilizando os seguintes passos:

1. Início
2. Para cada âncora identificada
 - Criar objeto de referência
 - Configurar posição e rotação do objeto de referência
 - Associar objeto de referência aos dados de âncora
3. Fim

Conforme descrito, a classe *AnchorManager* é responsável por iniciar a configuração das âncoras e publicar as mudanças no sistema MQTT. A classe *Draggable* lida com as interações do usuário, permitindo a manipulação dos objetos.

Processamento dos Dispositivos

O script foi configurado para ser executado durante a inicialização do ambiente Unity. Ele examina todos os objetos filhos do GameObject "devices", aplicando a expressão regular para identificar e configurar âncoras. Cada âncora identificada tem seus dados processados e preparados para comunicação via MQTT, permitindo o controle remoto dos componentes.

Identificação das Âncoras

Cada objeto é verificado para um padrão específico em seu nome, e se corresponder, é configurado como uma âncora. As informações extraídas incluem o nome da âncora, tipo, valores padrão, eixos, mapeamentos e tipo de entrada.

Configuração das Âncoras

Após a identificação, cria-se um objeto de referência para cada âncora, que é utilizado para calcular transformações e ajustes necessários. Este objeto de referência é posicionado como um irmão do objeto original, compartilhando a mesma hierarquia e transformações básicas.

Integração com MQTT

Cada âncora configurada é registrada com um tópico específico no sistema MQTT. Isso permite que as alterações nos valores das âncoras sejam publicadas e recebidas, facilitando o controle remoto em tempo real dos componentes virtuais.

4.2.3 Interações do Usuário com Sliders, Toggles e Buttons

Para gerenciar as interações do usuário, foi criada uma classe chamada *Draggable*. Esta classe permite a manipulação de objetos no ambiente virtual através de diferentes tipos de entradas, como sliders, toggles e botões. Estes sinais são enviados via MQTT e retornam para ajustar a interface conforme necessário.

Implementação da Classe Draggable

A classe Draggable rastreia a posição inicial do mouse e do objeto quando o usuário começa a interação. Dependendo do tipo de âncora (rotação ou tradução), ela calcula a nova posição ou rotação do objeto com base no movimento do mouse.

Cálculo de Deslocamento

Para rotação, a classe calcula o ângulo de rotação baseado no movimento do mouse ao redor de um eixo especificado. Para translação, calcula a nova posição do objeto ao longo de um eixo específico. Em ambos os casos, os valores resultantes são mapeados e ajustados conforme necessário, e os sinais resultantes são enviados via MQTT.

Aplicação das Transformações

As novas posições, rotações ou escalas são aplicadas ao objeto, e a interface do usuário é atualizada em tempo real para refletir essas mudanças. A lógica interna garante que os valores permaneçam dentro dos intervalos especificados e que as alterações sejam suaves e precisas.

A lógica implementada no script para a interação com o mouse pode ser descrita utilizando os seguintes passos:

Interação com o Mouse:

- Início
 - Quando o usuário pressiona o mouse, registra a posição inicial do mouse e do objeto.
 - Ao mover o mouse, calcula o deslocamento baseado na diferença entre a posição atual e a inicial do mouse.
 - Aplica a transformação correspondente (rotação, translação ou escala) ao objeto.
 - Envia a nova posição, rotação ou escala via MQTT.
 - Se a interação for finalizada, atualiza a interface do usuário em tempo real para refletir as mudanças.

Processamento das Mensagens MQTT:

- Início
 - Registrar âncoras com tópicos MQTT.
 - Monitorar tópicos MQTT.
 - Para cada mensagem recebida:
 - Identificar tipo de transformação (rotação, translação ou escala).
 - Se rotação:
 - Calcular ângulo com base na diferença de posição do mouse.
 - Ajustar rotação do objeto ao longo do eixo definido.
 - Enviar nova rotação via MQTT.
 - Se translação:

- Projetar diferença de posição do mouse ao longo do eixo de atuação.
- Ajustar posição do objeto.
- Enviar nova posição via MQTT.
- Se escala:
 - Calcular fator de escala com base no movimento do mouse.
 - Ajustar escala do objeto, preservando proporções.
 - Enviar nova escala via MQTT.
- Fim

A implementação descrita envolve a configuração precisa de sistemas de controle utilizando expressões regulares para identificar e manipular âncoras em um ambiente Unity, integrando essas âncoras com sistemas de comunicação MQTT para controle remoto. A classe Draggable permite interações do usuário com componentes virtuais, garantindo que as alterações sejam refletidas em tempo real e dentro dos parâmetros estabelecidos.

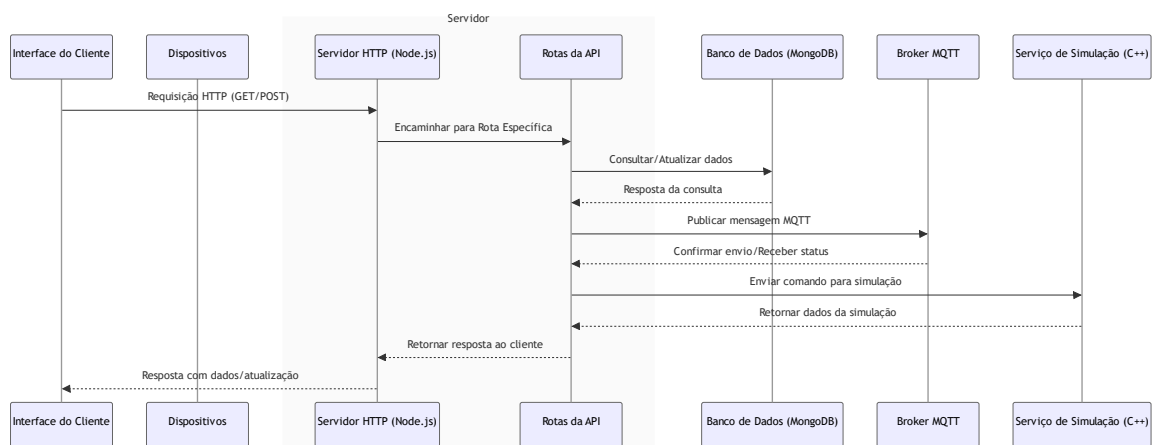
4.3 Configuração do Servidor Node.js

A configuração do servidor Node.js visou orquestrar a comunicação entre os diferentes componentes do sistema, incluindo serviços Docker, broker MQTT, serviço de simulação em C++, e o banco de dados MongoDB. Além disso, o servidor Node.js foi responsável por expor portas e rotas para o acesso à versão web da plataforma, utilizando o framework Express.

4.3.1 Estrutura Geral do Servidor

A estrutura do servidor foi organizada de forma modular, com cada funcionalidade principal sendo delegada a um módulo específico. A lógica central do servidor está contida no arquivo `app.js`, que inicializa o servidor HTTP, configura rotas para a API e integra os diferentes módulos para gerenciar a comunicação entre os serviços Figura 7.

Figura 7 - Arquitetura Geral do Servidor Node.js.



Fonte: Autoria Própria.

1. Inicialização do Servidor:

- O servidor HTTP é iniciado utilizando o módulo `http` do Node.js.
- O aplicativo Express é configurado para servir arquivos estáticos da pasta `public` e processar requisições JSON.

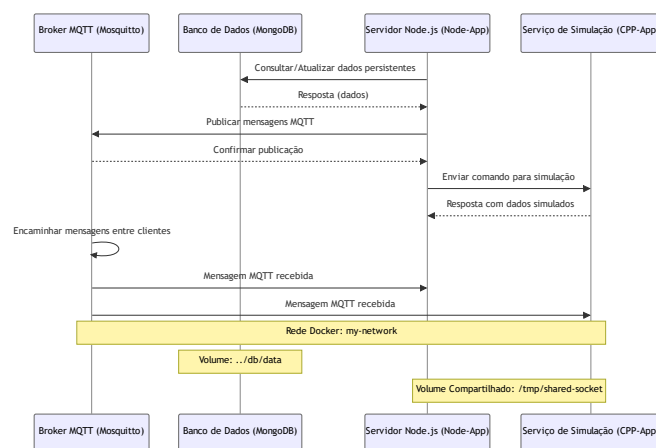
2. Integração de Módulos:

- **Módulo de Banco de Dados:** Um módulo (`db-mongo.js`) foi criado para gerenciar a conexão e operações com o MongoDB. Este módulo é responsável por estabelecer a conexão com o banco de dados, inicializar o cliente MongoDB, e disponibilizar métodos para operações de CRUD.
- **Módulo de Rotas da API:** O módulo `apiRoutes.js` define as rotas da API, facilitando a comunicação entre o servidor e a interface web, bem como o processamento de requisições de dispositivos conectados.
- **Módulo de WebSockets:** A comunicação em tempo real com a interface web é gerenciada pelo módulo `comm-ws.js`, que utiliza a biblioteca `socket.io` para manter uma conexão bidirecional entre o servidor e os clientes.
- **Módulo de MQTT:** A comunicação com o broker MQTT é estabelecida pelo módulo `comm-mqtt.js`, que se conecta ao broker Mosquitto e permite a troca de mensagens entre o servidor e os dispositivos IoT.

4.3.2 Orquestração dos Serviços Docker

A orquestração dos serviços Docker foi configurada através de um arquivo `docker-compose.yml`, que define os serviços e suas dependências Figura 8.

Figura 8 - Orquestração dos Serviços no Docker Compose.



Fonte: Autoria Própria.

1. **Broker MQTT:**

- a. O broker Eclipse Mosquitto é configurado para gerenciar a comunicação de mensagens entre os dispositivos conectados e o ambiente virtual.
- b. O servidor Node.js utiliza a biblioteca `mqtt` para se conectar ao broker, permitindo a publicação e subscrição de tópicos específicos.

2. **Serviço de Simulação em C++:**

- a. O serviço de simulação é executado em um contêiner separado, que é iniciado e gerenciado pelo servidor Node.js através de chamadas ao módulo `child_process`.
- b. Este serviço realiza cálculos complexos necessários para o ambiente virtual e retorna os resultados ao servidor.

3. **Banco de Dados MongoDB:**

- a. O banco de dados MongoDB, containerizado e gerenciado pelo Docker, armazena dados de simulação e interações do usuário.
- b. O módulo `mongoose` no servidor Node.js gerencia a conexão e operações de CRUD no banco de dados.

4.3.3 **Exposição de Portas e Rotas**

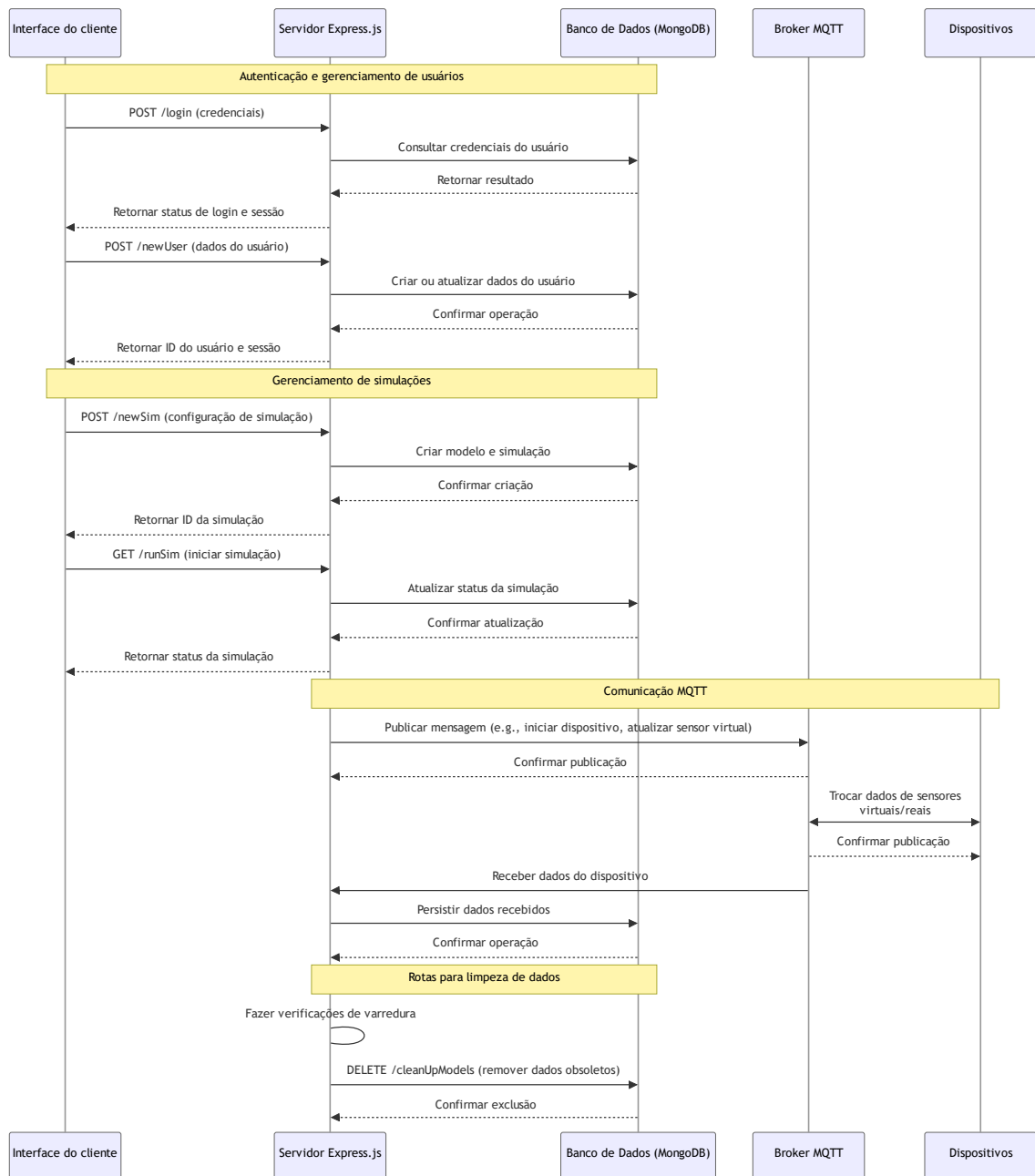
1. O framework Express é utilizado para expor portas e rotas, facilitando o acesso à plataforma web e a comunicação com dispositivos físicos **Rotas de API:**
 - a. As rotas RESTful são definidas para permitir operações de leitura, escrita, atualização e exclusão de dados no MongoDB.
 - b. Estas rotas são essenciais para a interação entre o ambiente virtual e os dispositivos IoT.

Figura 9.

2. **Rotas de API:**

- a. As rotas RESTful são definidas para permitir operações de leitura, escrita, atualização e exclusão de dados no MongoDB.
- b. Estas rotas são essenciais para a interação entre o ambiente virtual e os dispositivos IoT.

Figura 9 - Exposição de Portas e Rotas com Express.js.



Fonte: Autoria Própria.

3. Serviço Web:

- O servidor serve a interface web, permitindo que os usuários acessem e interajam com o ambiente virtual através de navegadores modernos.
- Arquivos estáticos são servidos a partir da pasta `public`, e páginas específicas são mapeadas para diferentes endpoints

4.4 Integração MQTT

A integração com o protocolo MQTT para a troca de dados em tempo real foi configurada utilizando o broker Eclipse Mosquitto. A comunicação é gerenciada pelo servidor Node.js, que possibilita a troca de mensagens entre o ambiente virtual e os dispositivos conectados.

4.4.1 Configuração do Broker MQTT

O broker Eclipse Mosquitto foi configurado através de um contêiner Docker, definido no arquivo `docker-compose.yml`. Este contêiner gerencia a comunicação de mensagens entre os dispositivos conectados e o ambiente virtual, utilizando o protocolo MQTT. A configuração básica do Mosquitto inclui a definição de persistência de dados, localização de logs e a permissão para conexões anônimas.

4.4.2 Estrutura dos Tópicos MQTT

Foi adotado um padrão para a estrutura dos tópicos MQTT para organizar a comunicação entre o ambiente virtual e os dispositivos conectados. Para este projeto, o nome do ambiente foi definido como "johnnPFC", mas a estrutura permite que usuários criem e utilizem seus próprios ambientes.

- `API/#`: Utilizado para enviar um endereço proxy para comunicação via API com o servidor, permitindo que os dispositivos conheçam e comuniquem através de um endereço público com a API.
- `env/device/signal_name`: Onde `env` é o nome do ambiente, `device` é o dispositivo, e `signal_name` é o nome dado ao sinal.

Os sinais podem ser de diferentes formatos, incluindo:

- **Formato simples:** `[numero]` ou `{value:[numero],time:[time_sent]}`
- **Referência para modelo/simulação:** `ref_md1[mongodb_model_id]`

Quando um sinal retorna uma string começando com `ref_md1`, concatenado com o `_id` do modelo, a lógica dos dispositivos (como microcontroladores, sockets web ou Unity) faz a inscrição neste sinal e recebe os valores diretamente do modelo através do tópico `sim_data/<_id do modelo>`.

4.4.3 Implementação da Comunicação MQTT no Servidor Node.js

A comunicação MQTT no servidor Node.js é gerenciada por um módulo específico (`comm-mqtt.js`). Este módulo utiliza a biblioteca `mqtt` para se conectar ao broker Mosquitto e gerenciar a publicação e subscrição de tópicos.

1. Conexão ao Broker MQTT:

- a. O módulo MQTT é inicializado com a URL do broker (`mqtt://mosquitto:1883`).
- b. Ao conectar, o módulo emite eventos (`mqtt-subscribe`, `mqttConnected`) através do `EventEmitter`.

2. Subscrição e Publicação de Tópicos:

- a. Ao conectar-se ao broker, o módulo se inscreve aos tópicos configurados, incluindo `<envs>/#`. (onde `envs` refere-se a cada ambiente criado que contenha usuários online, ou simulações rodando)
- b. Mensagens recebidas nos tópicos são emitidas como eventos (`mqtt-message`), permitindo que outras partes do sistema reajam às mensagens recebidas.
- c. O módulo também escuta por eventos internos (`mqtt-data`) para publicar mensagens em tópicos específicos.

4.4.4 Comunicação entre Servidor Node.js e Interface Web

Para a comunicação com a interface web, foi implementado um módulo de WebSocket (`comm-ws.js`) para gerenciar a comunicação em tempo real com a interface web. Este módulo utiliza a biblioteca `socket.io` para estabelecer uma conexão bidirecional entre o servidor e os clientes web.

1. Inicialização do WebSocket:

- O módulo WebSocket é inicializado e associado ao servidor HTTP.
- Conexões WebSocket são estabelecidas e gerenciadas pelo módulo `socket.io`.

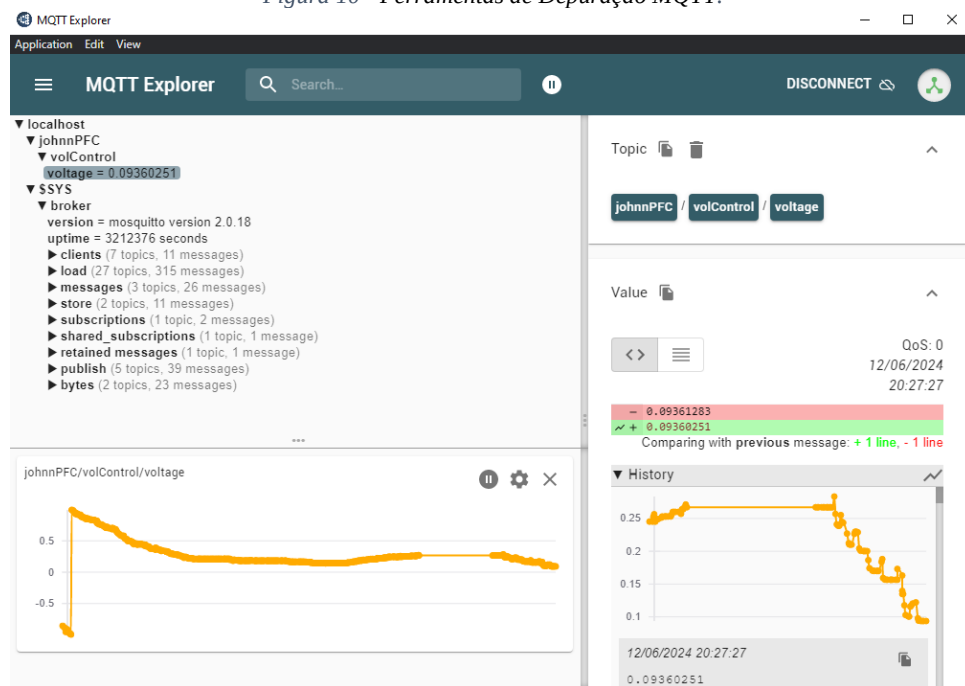
2. Emissão de Eventos MQTT:

- O módulo escuta por eventos WebSocket que começam com `mqtt:/`.
- Esses eventos são emitidos internamente como eventos MQTT (`mqtt-data`), permitindo a publicação de mensagens nos tópicos apropriados.

4.4.5 Testes e Depuração

Para testar e depurar a integração MQTT, foram utilizados o console de desenvolvimento do Chrome e a ferramenta open-source MQTT Explorer (Nordquist, 2024). O console de desenvolvimento do Chrome permitiu a verificação e envio de mensagens MQTT diretamente da interface web, enquanto o MQTT Explorer facilitou a visualização e depuração das mensagens MQTT trocadas entre os dispositivos e o servidor Figura 10.

Figura 10 - Ferramentas de Depuração MQTT.



Fonte: acervo pessoal.

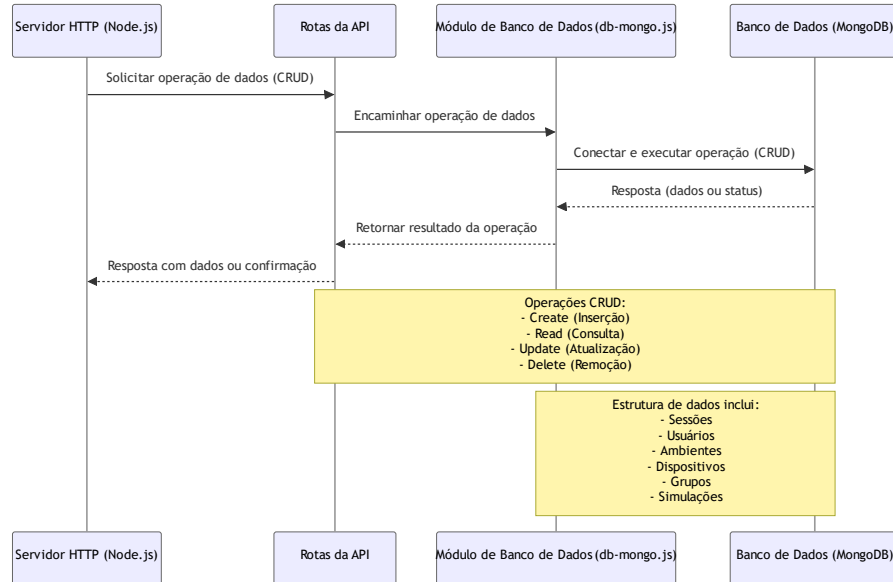
Pela figura pode ser visto a interface do **MQTT Explorer**, mostrando a ferramenta em uso para monitorar e depurar mensagens MQTT. A tela está dividida em três seções principais: à esquerda, a hierarquia dos tópicos MQTT conectados ao broker local; ao centro, detalhes do tópico "johnnPFC/volControl/voltage", mostrando o valor atual da voltagem (0.09360251) e comparando-o com o valor anterior; e abaixo, um gráfico histórico ilustrando a variação da voltagem ao longo do tempo. A interface também destaca métricas do broker Mosquitto (versão 2.0.18) e o histórico de mensagens trocadas, essencial para identificar padrões e resolver problemas na comunicação entre dispositivos e servidor.

4.5 Armazenamento de Dados com MongoDB

Foi utilizada a base de dados MongoDB para armazenar dados de simulação e interações do usuário. A estrutura de dados inclui informações sobre sessões, usuários, ambientes,

dispositivos, grupos e simulações. A configuração do banco de dados, integração com o servidor Node.js e a definição dos esquemas de dados serão descritas a seguir.

Figura 11 - Interações e Configuração do MongoDB.



Fonte: Autoria própria.

4.5.1 Configuração do Banco de Dados MongoDB

O MongoDB foi configurado como um serviço Docker, utilizando o arquivo `docker-compose.yml`. Para garantir a persistência de dados, foi utilizado um volume Docker, assegurando que os dados sejam mantidos mesmo em caso de reinício ou desligamento do contêiner.

4.5.2 Estrutura de Dados

A estrutura de dados do MongoDB foi projetada para armazenar informações sobre sessões, usuários, ambientes, dispositivos, grupos e simulações. A seguir, detalha-se a estrutura de cada coleção:

- **Sessões (session):**
 - ID
 - usr
 - time
 - env
 - IP/mac
 - alias (e.g., unityApp, webApp, microConn)
- **Usuários (user):**
 - user-id
 - name
 - icon (opcional)
 - mail (opcional)

- o login-method (e.g., google, email/user-pwd)
 - o pwd-token (opcional)
 - o loc (array com sessão, posição, tempo)
 - o sessions (array com IDs de sessões válidas)
- **Ambientes (env):**
 - o env-id
 - o users
- **Mapeamento (map):**
 - o geometry (referência de localização)
 - o devices
 - id
 - transform (posição, rotação, escala)
- **Dispositivos (device):**
 - o dev-id
 - o base-id (opcional)
 - o name
 - o icon (opcional)
 - o preview (opcional)
 - o description (opcional)
 - o geometry (referência com diretório do arquivo)
 - o owner
 - o anchors
 - o simulations
- **Grupos (group):**
 - o group-id
 - o name
 - o owners (número)
 - o members (todos)
- **Simulações (simulation):**
 - o state (e.g., running, paused, stopped)
 - o status (e.g., on, off)
 - o log (array com tempo e dados)
 - o models (o primeiro é o ativo)
- **Modelos (model):**
 - o sys
 - value (string)
 - type (e.g., const, SS matrix, ft, fw, dt)
 - o input
 - value (string)
 - type (e.g., const, function, mqtt, sim)
 - dt (tempo de atualização)
 - st (tempo de amostragem)
 - decimation (método)

4.5.3 Integração com o Servidor Node.js

A integração do MongoDB com o servidor Node.js foi realizada utilizando a biblioteca `mongodb` e a criação de um módulo específico (`db-mongo.js`). Este módulo gerencia a conexão com o banco de dados e fornece métodos para realizar operações CRUD (Create, Read, Update, Delete).

1. Conexão ao MongoDB:

- O módulo `db-mongo.js` inicializa a conexão ao MongoDB utilizando a URL do banco de dados (`mongodb://mongodb:27017`) e o nome do banco de dados (`my-database`).
- A conexão é estabelecida e mantida através do cliente `MongoClient` da biblioteca `mongodb`.

2. Definição dos Esquemas de Dados:

- O módulo define e gerencia as coleções de dados, como usuários, sessões, ambientes, dispositivos, grupos, simulações e modelos.
- Métodos específicos são implementados para realizar operações CRUD em cada coleção, como criação, recuperação, atualização e exclusão de documentos.

4.5.4 Operações CRUD

As operações CRUD são implementadas no módulo `db-mongo.js`, fornecendo métodos para manipulação dos dados armazenados no MongoDB. Estes métodos incluem:

- **Criação:** Implementa a inserção de novos documentos nas coleções, como a criação de novos usuários, ambientes e simulações.
- **Leitura:** Realiza a busca de documentos específicos nas coleções, permitindo a recuperação de informações como detalhes de usuários e dispositivos.
- **Atualização:** Permite a modificação de documentos existentes, como a atualização de informações de usuários e estados de simulação.
- **Exclusão:** Implementa a remoção de documentos das coleções, como a deleção de usuários ou sessões antigas.

Como exemplo pode-se visualizar abaixo a implementação realizada para lidar com a coleção 'users'

- **Criação de Usuário:**

```

async function createUser(user) {
  const collection = this.db.collection('users');
  user.createdAt = new Date();
  const result = await collection.insertOne(user);
  return result;
}

```

- **Recuperação de Usuário:**

```

async function getUser(userId) {
  const collection = this.db.collection('users');
  const user = await collection.findOne({ _id: userId });
  return user;
}

```

- **Atualização de Usuário:**

```

async function updateUser(userId, updates) {
  const collection = this.db.collection('users');
  const result = await collection.updateOne({ _id: userId }, { $set: updates });
  return result;
}

```

- **Exclusão de Usuário:**

```

async function deleteUser(userId) {
  const collection = this.db.collection('users');
  const result = await collection.deleteOne({ _id: userId });
  return result;
}

```

4.5.5 Rotas da API

As rotas da API para interação com o MongoDB foram definidas no módulo `apiRoutes.js`. Essas rotas permitem que os clientes realizem operações CRUD através de requisições HTTP. O `userId` é enviado no cabeçalho das requisições POST ou GET para que a operação seja realizada sobre o usuário especificado. A validação dos dados enviados é realizada através de esquemas JSON definidos em um arquivo de validação.

- **Criação de Novo Usuário:**

- Endpoint: POST /newUser
- Cabeçalho: `userId`
- Corpo da Requisição: { `usr`, `pwd`, `mail` }
- Descrição: Cria um novo usuário ou atualiza um usuário anônimo existente.

- **Login de Usuário:**
 - Endpoint: `POST /login`
 - Corpo da Requisição: `{ usr, pwd }`
 - Descrição: Autentica um usuário existente e retorna suas informações.
- **Criação de Nova Sessão:**
 - Endpoint: `GET /newSession`
 - Cabeçalho: `userID`
 - Descrição: Cria uma nova sessão para o usuário especificado.
- **Listagem de Usuários:**
 - Endpoint: `GET /listUsers`
 - Descrição: Retorna uma lista de todos os usuários.
- **Outras Rotas Implementadas:**
- **Criação de Novo Modelo:** `POST /newModel`
- **Criação de Nova Simulação:** `POST /newSimulation`
- **Criação de Novo Ambiente:** `POST /newEnv`

Cada uma destas rotas utilizam o `userID` no cabeçalho (header) para realizar operações específicas de criação, busca, atualização ou deleção no banco de dados MongoDB.

4.6 Containerização com Docker

A utilização do Docker para containerizar os diferentes serviços do projeto permitiu a criação e configuração dos contêineres de forma eficiente e padronizada. A configuração foi gerenciada através de um arquivo Docker Compose, que define os serviços necessários, como Node.js, MQTT e MongoDB. Esta seção detalha como o Docker Compose foi utilizado para configurar e gerenciar os contêineres, e a criação de uma rede bridge para facilitar a comunicação entre eles.

4.6.1 Configuração do Docker Compose

Para gerenciar os contêineres, foi utilizado o Docker Compose com a versão 3.7. O arquivo `docker-compose.yml` especifica os serviços, redes e volumes necessários para a execução dos componentes do sistema.

1. Serviço MQTT:

- **Imagem e Configuração:** Utilizou-se a imagem `eclipse-mosquitto` do Docker Hub. A configuração personalizada do Mosquitto foi realizada

diretamente no arquivo `docker-compose.yml`, criando diretórios necessários e definindo configurações de persistência, logs e permissões para conexões anônimas.

- **Portas e Rede:** A porta 1883 foi exposta para permitir a comunicação MQTT. O serviço foi conectado à rede bridge `my-network`.

2. Serviço MongoDB:

- **Imagem e Configuração:** Utilizou-se a imagem `mongo:3.6` do Docker Hub. Para garantir a persistência dos dados, um volume foi montado no diretório de dados do MongoDB.
- **Portas e Rede:** A porta 27017 foi mapeada para 27018 na máquina host, permitindo o acesso ao banco de dados. O serviço foi conectado à rede `my-network`.

3. Serviço Node.js:

- **Imagem e Configuração:** Criou-se uma imagem personalizada para o serviço Node.js. O Dockerfile correspondente define a base `node:14`, copia os arquivos do projeto, instala as dependências e expõe a porta 3000.
- **Portas e Rede:** A porta 3000 foi exposta para permitir o acesso à aplicação. O contêiner foi conectado à rede `my-network`.
- **Volumes:** Dois volumes foram montados: um para o diretório da aplicação (`../node-app`) e outro para os módulos Node.js (`nodemodules`), garantindo que as mudanças no código e nas dependências sejam refletidas imediatamente.

4. Serviço de Simulação em C++:

- **Imagem e Configuração:** Utilizou-se a imagem `gcc:latest` do Docker Hub. A configuração do contêiner incluiu a instalação de dependências e a compilação do código de simulação.
- **Portas e Rede:** A porta 12345 foi exposta para permitir a comunicação com o serviço de simulação. O contêiner foi conectado à rede `my-network`.

4.6.2 Criação de Redes e Volumes

- **Rede Bridge:** A rede `my-network` foi criada utilizando o driver bridge. Esta rede permite a comunicação interna segura e eficiente entre os contêineres do projeto. A escolha de uma rede bridge garante isolamento dos contêineres e facilita a configuração das regras de comunicação entre eles.

- **Volumes:** Volumes foram utilizados para garantir a persistência de dados e facilitar o desenvolvimento. O volume `../db/data` foi montado no serviço MongoDB para armazenar os dados do banco, enquanto os volumes `../node-app` e `nodemodules` no serviço Node.js garantem que as mudanças no código e nas dependências sejam refletidas imediatamente sem necessidade de reconstrução dos contêineres.

4.6.3 Comandos e Execução

Os comandos definidos no `docker-compose.yml` asseguram que cada contêiner seja configurado corretamente ao iniciar:

1. **Serviço MQTT:** Configuração do Mosquitto para persistência de dados e logs.
2. **Serviço Node.js:** Instalação das dependências Node.js e utilização do `nodemon` para reiniciar automaticamente o servidor em caso de mudanças no código.
3. **Serviço de Simulação:** Atualização do sistema, instalação de dependências específicas e compilação do código de simulação C++.

4.6.4 Desenvolvimento da Configuração com Docker

1. Configuração do Serviço MQTT:

- O contêiner Mosquitto foi configurado para criar os diretórios necessários para dados e logs, além de definir a configuração de persistência. Isso foi feito diretamente no comando de inicialização no `docker-compose.yml`.
- As configurações foram salvas no diretório `/mosquitto/config/mosquitto.conf`, garantindo que o Mosquitto inicie com as configurações corretas toda vez que o contêiner é executado.

2. Configuração do Serviço MongoDB:

- A imagem `mongo:3.6` foi utilizada para criar o contêiner MongoDB, com um volume montado para persistir os dados no diretório `../db/data`. Isso garante que os dados armazenados no MongoDB sejam preservados mesmo que o contêiner seja reiniciado.
- A porta 27017 do contêiner foi mapeada para a porta 27018 do host, permitindo o acesso ao banco de dados tanto de dentro quanto de fora do ambiente Docker.

3. Configuração do Serviço Node.js:

- O Dockerfile do Node.js especifica a utilização da imagem base `node:14`, copiando o código da aplicação e instalando as dependências necessárias. A porta 3000 foi exposta para acesso à aplicação.

- Dois volumes foram montados: um para o diretório da aplicação (`../node-app`) e outro para os módulos Node.js (`nodemodules`). Isso facilita o desenvolvimento contínuo, pois qualquer alteração no código ou nas dependências é refletida imediatamente.

4. Configuração do Serviço de Simulação em C++:

- Utilizou-se a imagem `gcc:latest` para criar o contêiner de simulação. O contêiner foi configurado para instalar dependências específicas e compilar o código de simulação no momento da inicialização.
- A porta 12345 foi exposta para permitir a comunicação com o serviço de simulação.

Essa abordagem de containerização permitiu uma configuração modular e escalável dos serviços, facilitando o desenvolvimento, testes e implantação do sistema. A utilização do Docker Compose para gerenciar a configuração e execução dos contêineres garantiu que os serviços fossem iniciados de maneira coordenada e integrada, assegurando uma comunicação eficiente entre eles.

4.7 Desenvolvimento do Controlador com ESP-32 e ESP01

Programação e Configuração dos Dispositivos

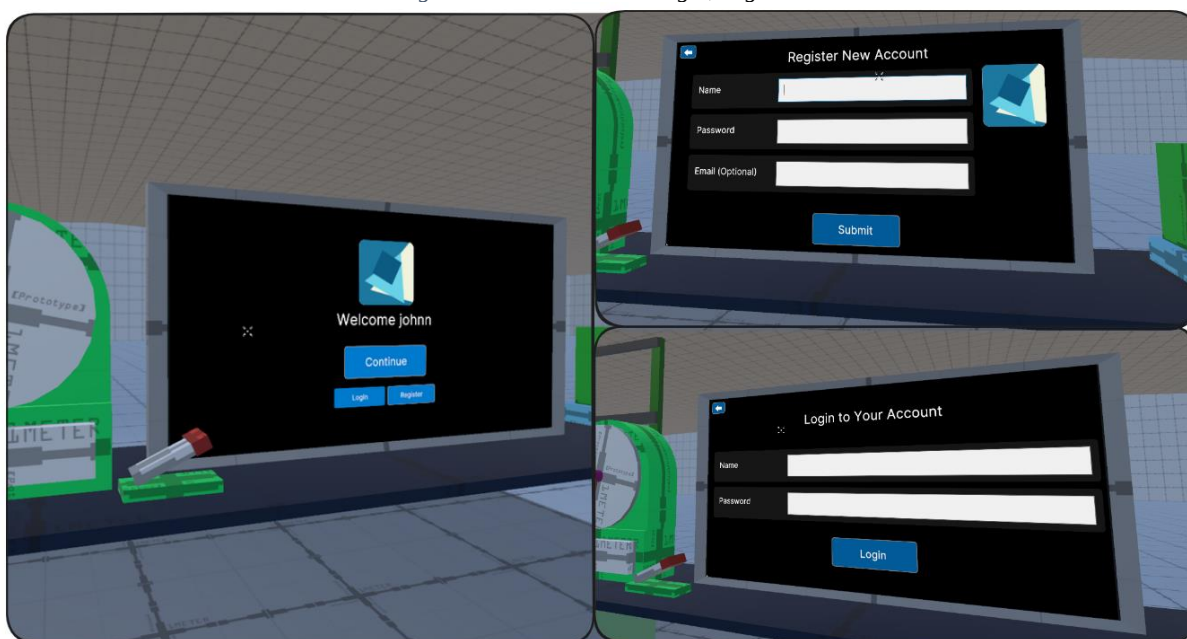
Os microcontroladores ESP32 e ESP01 foram programados para interagir com o sistema, enviando e recebendo sinais via MQTT. A lógica de funcionamento dos microcontroladores inclui conectar-se à internet, subscrever-se a tópicos específicos e processar as entradas para atuar como controlador ou sensor.

5 RESULTADOS

5.1 Fluxo de Navegação do Usuário no Ambiente Virtual

O fluxo de trabalho, começando pelo cliente consiste em fazer o login ou continuar a sessão anterior caso já tenha registro, ou dado o perfil temporário que é criado inicialmente, registra-lo para obtenção de uma persistência de dados. Em seguida, conforme a Figura 12 abrir o menu principal, devido a característica de ser um ambiente adaptado para trabalhar em realidade virtual ou aumentada, tais seleções são realizadas em uma tela física (representação de um televisor), visto que uma interface no espaço de tela não seria viável em uma situação de imersão.

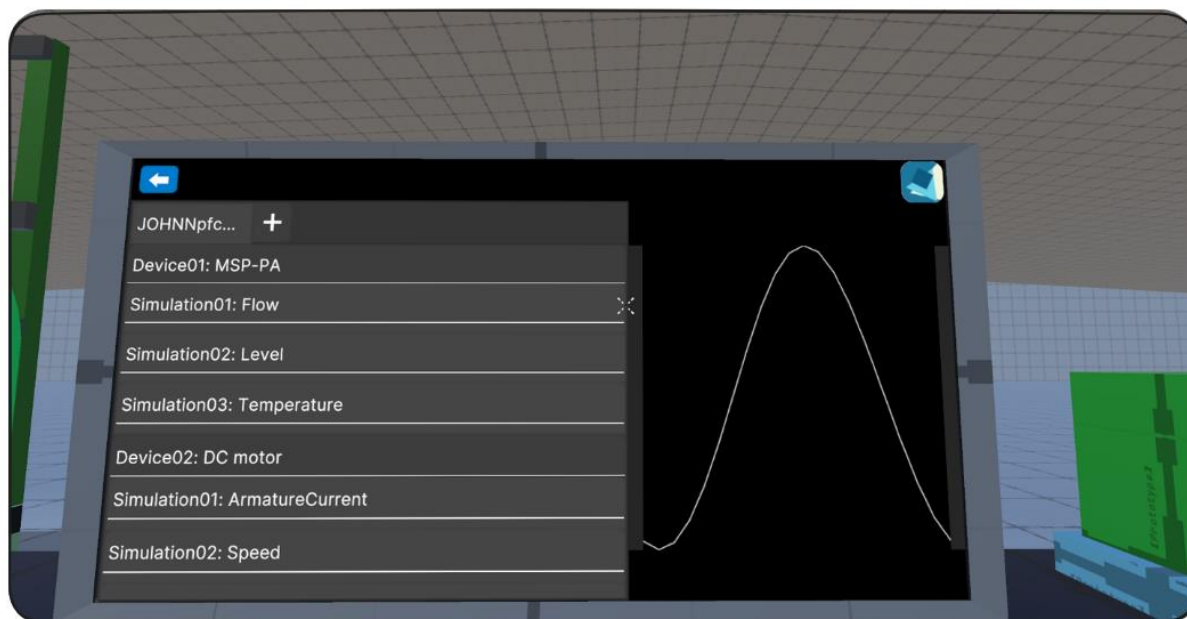
Figura 12 -Telas Iniciais: Login, Registro.



Fonte: acervo pessoal

Na tela principal, Figura 13, é possível selecionar o ambiente e visualizar os dispositivos, as simulações, seus parâmetros e estado atual, assim como fazer alterações e atribuições de valores para as mesmas juntamente com a criação de novos elementos. Para alternar entre os ambientes existentes disponíveis usa-se as abas superiores, e selecionado entre os dispositivos presentes, pode-se expandir ou contrair para uma visualização mais detalhada.

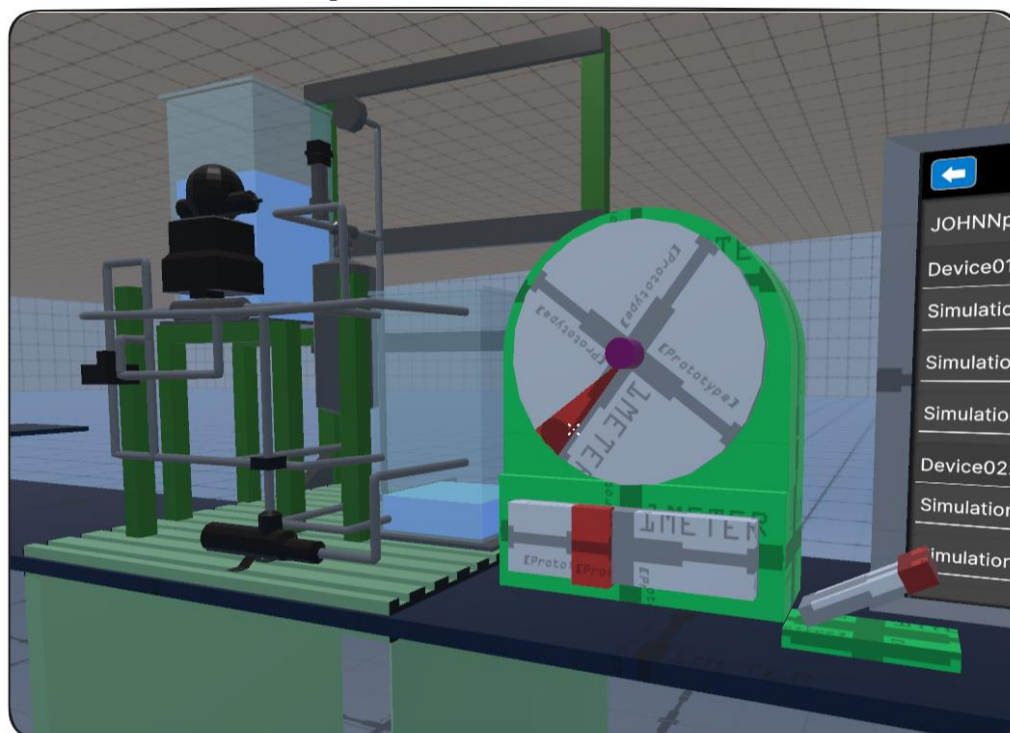
Figura 13 -Interação do Usuário com o Ambiente Virtual na tela principal.



Fonte: acervo pessoal

Navegando pelo ambiente 3D, Figura 14, encontram as representações virtuais dos dispositivos, assim como formas de interagir com os mesmos, as visualizações refletem a uma variável simulada ou lida por sensores.

Figura 14 -Interatividade com o ambiente.



Fonte: acervo pessoal

O usuário pode manualmente alterar o estado de elementos arrastando ou clicando, nos objetos do cenário, tais estados representam grandezas como tensão de uma fonte, posição de uma chave (ligado ou desligada), set-point de velocidade, vazão, posição, ganho de um compensador, etc. Suas interações são traduzidas para transformações nos elementos, através das ancoras, como alteração na posição (*slider* ou *toggle*), escala (*toggle* ou *butttons*), rotação (*slider* ou *toggle*), cor (*toggle* ou *buttons*); conforme descrito na sessão de 4.2.3.

Com a mesma lógica de ancoras, que acopla a transformação de objetos com grandezas físicas, além da interação com o usuário, sinais enviados por sensores ou outras simulações recebidas através udp ou mqtt, possibilitam a visualização como em um gêmeo digital de suas representações.

Desta forma essas implementações permitem a visualização e interação do usuário ou de seus dispositivos microcontrolados com o ambiente virtual e vice-versa.

5.2 Sincronização em Tempo Real e Comunicação entre Dispositivos

Um dos objetivos do trabalho, para que possa ser considerado gêmeo digital, é trabalhar em tempo real, esta consideração depende do contexto, como por exemplo <buscar exemplos>, porém geralmente 20ms já pode ser considerada *real-time* <adicionar cita> desta forma dependendo da aplicação que se deseja simular deve ser escolhido o protocolo de comunicação e a forma de conexão.

A máquina do cliente ou do dispositivo embarcado utilizado pode estar conectada ao servidor através de uma ponte, de um proxy, ou até mesmo estar na mesma máquina, formando uma rede local. Pensando nisso, foi realizada uma análise de latência considerando diferentes cenários de conexão e protocolos de comunicação. Os dados foram coletados ao longo de múltiplos testes, e a latência média foi calculada para avaliar o impacto dessas configurações no desempenho do sistema.

Nos testes realizados, observou-se que o protocolo UDP em uma conexão local (LAN) apresentou a menor latência média, 3,554 ms, o que torna essa configuração a mais adequada para aplicações que exigem respostas em tempo real. Esse desempenho confirma que a conexão LAN, associada ao UDP, é capaz de atender às exigências de baixa latência necessárias para garantir a operação eficiente de um gêmeo digital. Já o protocolo MQTT em conexão local apresentou uma latência média de 77,85 ms, um valor mais elevado em comparação ao UDP, mas que ainda pode ser aceitável em aplicações onde a integridade dos dados é prioritária.

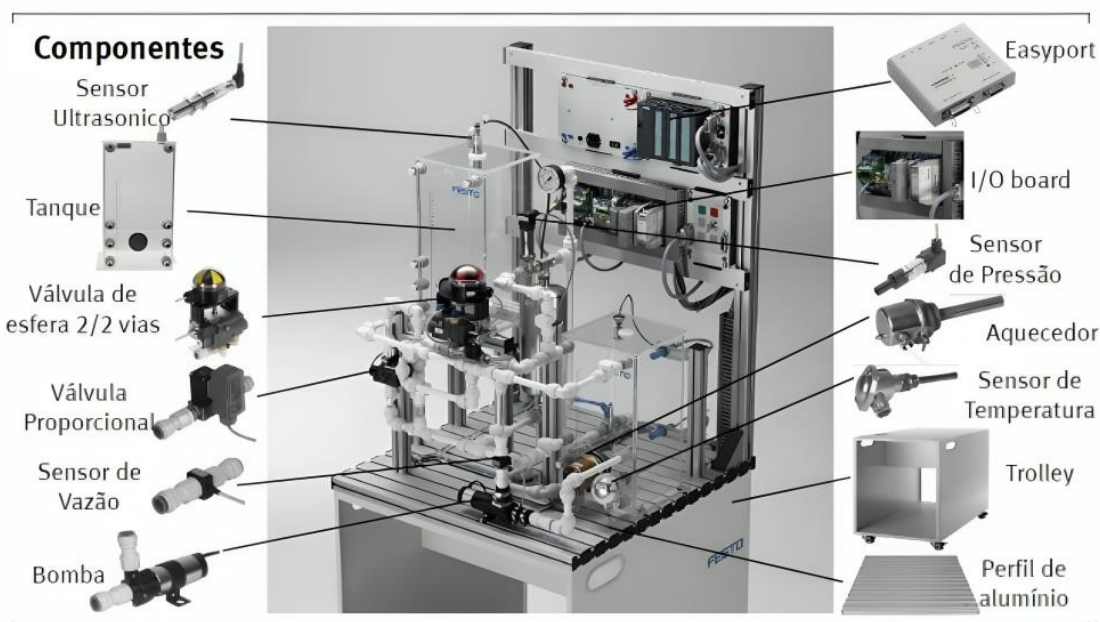
Nas conexões realizadas via proxy, que simulam um ambiente remoto, os valores de latência foram significativamente maiores. O protocolo UDP apresentou uma latência média de 344,85 ms, enquanto o protocolo MQTT teve uma média de 252,31 ms. Esses valores indicam que, embora o uso de proxy seja funcional, ele compromete a capacidade de operação em tempo real do sistema, sendo mais indicado para aplicações menos sensíveis ao tempo ou para testes em ambientes distribuídos.

5.3 Estudo de Caso: Controle de Nível com Gêmeo Digital

Tendo selecionado o protocolo de comunicação e a forma a qual o micro-controlador estará conectado ao servidor, é feita a leitura dos sinais que se desejam controlar, e escrita e publicação dos sinais que serão atuados sobre.

Este foi baseado na MPS-PA workstation da Festo, Figura 15, neste caso o micro-controlador atua sobre a tensão aplicada na bomba e recebe valores de leitura do sensor de nível. Para o projeto da função de transferencia do sistema, foi realizada uma perturbação de degrau unitário na bomba, acionando-a instantaneamente com seu valor máximo e realizada a leitura da vazão, por se tratar de um sistema fechado o nível do tanque superior foi calculado com base nas dimensões das tubulações e tanque de água.

Figura 15 -MPS-PA Festo Usada como Base para o Sistema.



Fonte: (ISETPOINT, 2024)

Com isso obtendo uma aproximação de primeira ordem, da vazão em função da tensão aplicada, e então do nível em função da vazão. Na prática foi projetado um controlador PID para o CLP S7-300 CPU 313C, que controla o sistema físico real.

Para o mesmo experimento no ambiente virtual, pode-se utilizar a modelagem matemática da planta e sensores(quanto menos ideal/mais precisa melhor) no servidor, e projeto do controlador/acionamento em um ESP01 conectado a um relê, para simbolizar o acionamento da bomba assim como feito com uma valvula digital no CLP.

Para um *benchmark* esta é uma modelagem bem simples, visto que os tempos de resposta são consideravelmente altos (ordem de segundos), considerando rede local LAN, com protocolo MQTT, usando broker local, aproximação dos sensores para um ganho unitário, e modelagem da planta de primeira ordem, que pode ser descrito conforme a equação abaixo:

$$G(s) = \frac{K}{\tau s + 1} e^{-\theta s}$$

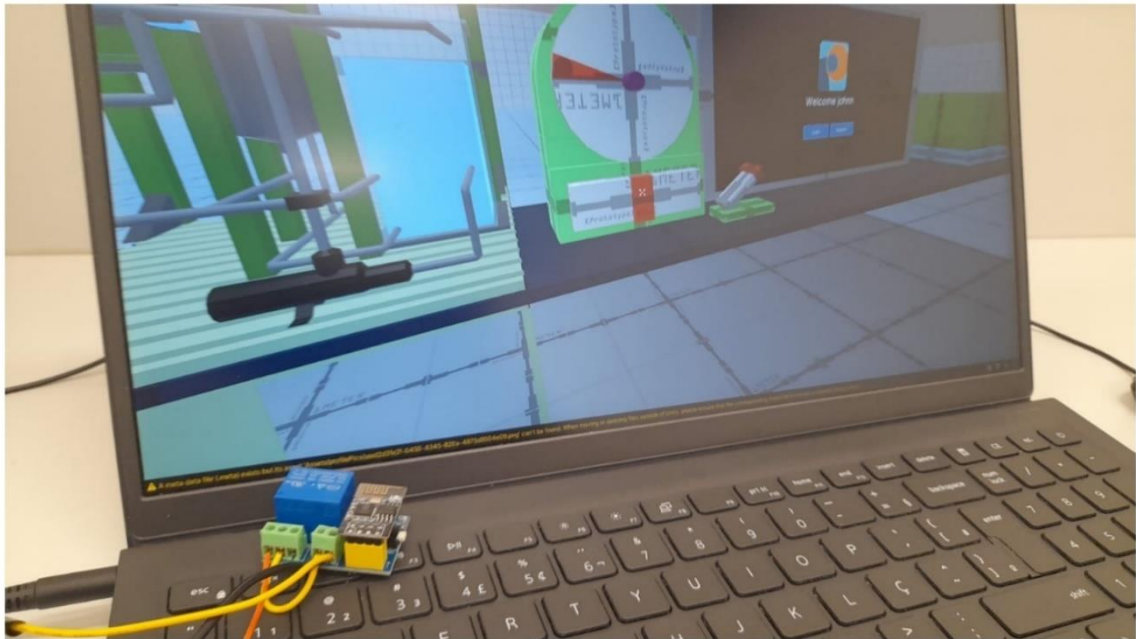
Onde K é o ganho da planta, τ é o tempo característico θ o tempo morto, sabendo que o tempo de subida é dado por $\tau \times \ln\left(\frac{0.9}{0.2}\right)$ e que a vazão demorou 12 segundos, sem tempo morto para chegar a seu valor maximo de 0.61 L/min, os valores de K e τ são respectivamente .61 e 1.3.

O nível é dado pela integração da vazão dividido pela area da sessão transversal do tanque, que neste caso é de um quadrado de 20cm, a vazão a ser considerada é a vazão de entrada da bomba dada pela modelagem feito acima menos a de saída do sistema, que é calculada considerando o que sai do tanque superior para o inferior, isso é feito levando em conta a altura dos tanques e o diametro da tubulação, que é de 15mm.

Desta forma , conforme a Figura 16, o setpoint de nível é atribuido na aplicação em unity, o ESP atua faz o papel do CLP, lendo o sensor de vazão virtual e implementa o PID, retornando um setpoint de tensão.

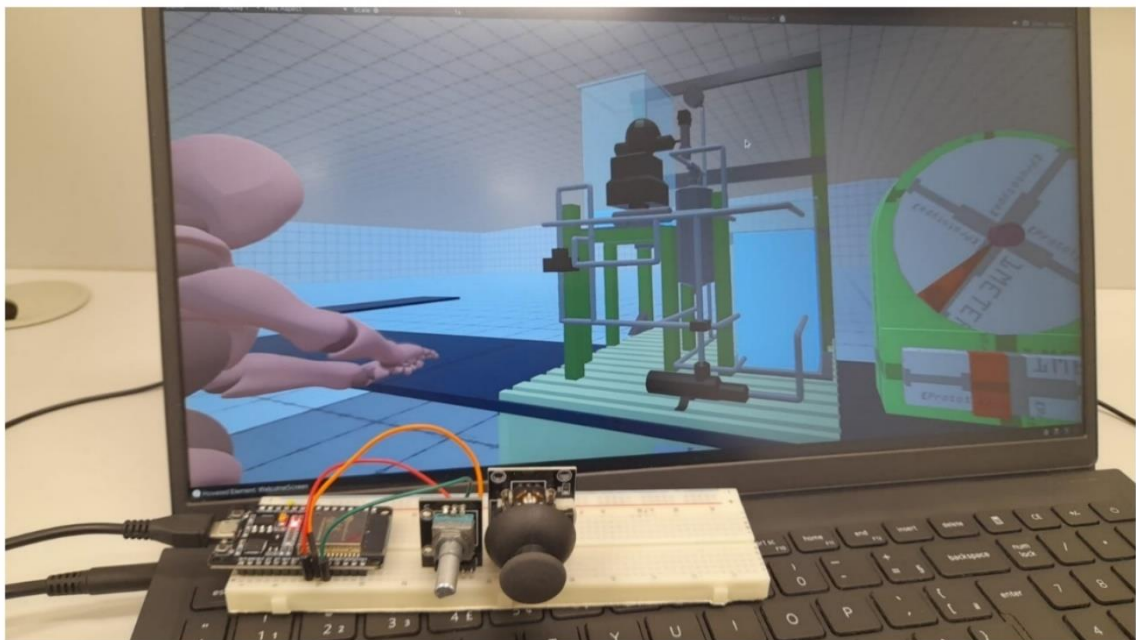
Desta forma, é possível alterar o setpoint de nível, ou vazão e visualizar a tensão sendo alterada. Para deixar isso mais dinâmico possibilitando uma exemplificação de um caso que o set-point e definido por um dispositivo fisico, implementou-se utilizando um ESP32 com potenciômetros e botões (joystick ou encoder incremental rotatório), um caso que pode habilitar ou desabilitar o controlador, manipular diretamente a tensão da bomba, ou um setpoint de vazão, ou nível, dependendo do que for selecionado, conforme a Figura 17.

Figura 16 -Controle de nível com ESP01 como CLP..



Fonte: acervo pessoal

Figura 17 -Controle de Nível com ESP32 e Entradas Adicionais.



Fonte: acervo pessoal

6 DISCUSSÃO

6.1 Análise dos Resultados Obtidos

Os resultados obtidos evidenciam a eficácia do sistema desenvolvido para atender às principais demandas educacionais e experimentais no controle de processos. A plataforma virtual implementada demonstrou ser robusta e altamente versátil, viabilizando a integração eficiente entre dispositivos físicos e o ambiente simulado. O uso de tecnologias como Unity, MQTT e microcontroladores ESP32 e ESP01 possibilitou a criação de um sistema que não apenas simula o comportamento de plantas industriais, mas também permite a interação em tempo real entre os elementos físicos e virtuais. Os testes realizados confirmaram a precisão e a responsividade do sistema ao executar comandos no ambiente virtual e transmiti-los para os dispositivos físicos programados. Além disso, foi possível replicar cenários industriais complexos, como o controle de nível e variáveis da planta MPS-PA, com alto grau de fidelidade.

O sistema se mostrou altamente configurável, permitindo a fácil substituição ou adição de novos modelos e cenários simulados, ampliando significativamente sua aplicabilidade em diferentes contextos educacionais e industriais. Outro aspecto positivo foi a funcionalidade de programação direta dos microcontroladores ESP, que permite interagir com as plantas carregadas na plataforma. Esse recurso representa um diferencial importante, pois possibilita ao usuário experimentar em tempo real diferentes configurações de controle, personalizando tanto o ambiente virtual quanto os dispositivos físicos associados. Tal flexibilidade aumenta a experiência prática e proporciona um aprendizado mais dinâmico e interativo.

6.2 Comparação com Trabalhos Relacionados

O estudo de Feijoo et al. (2021) é um exemplo marcante do uso de ambientes virtuais para controle de processos. O trabalho apresentou uma aplicação específica para a planta MPS-PA utilizando controladores fuzzy, destacando a robustez e a precisão obtidas com essa abordagem. Esse estudo foi inovador ao integrar lógica fuzzy para o controle de variáveis industriais em um ambiente virtual. Contudo, o foco estava em uma aplicação específica, enquanto o presente projeto buscou oferecer uma plataforma genérica, que não apenas implementa a MPS-PA, mas também permite carregar outras plantas e realizar programações personalizadas em microcontroladores para interagir com o sistema em tempo real. Essa flexibilidade amplia o alcance do projeto e o torna aplicável a uma variedade maior de cenários e demandas.

O trabalho iDynamic, desenvolvido por Martins (2020), também é uma referência na área de simuladores virtuais. A plataforma é altamente versátil, suportando simulações de sistemas em múltiplos domínios, como elétrico, hidráulico e pneumático. Além disso, sua funcionalidade de interação em tempo real via WebSocket e a possibilidade de programar novos cenários em Python são características impressionantes que enriquecem as aplicações de controle. Contudo, o projeto aqui desenvolvido vai além ao facilitar a interação entre dispositivos físicos genéricos e o ambiente simulado, oferecendo um nível de integração prática que complementa a versatilidade da programação. Isso permite não apenas simular, mas também criar um vínculo direto entre o mundo físico e o virtual, fortalecendo a experiência educacional e a aplicabilidade industrial.

6.3 Sugestões para Trabalhos Futuros

1. **Implementação Completa de Ferramentas de Análise e Controle:**
 - Integração de diagramas de Bode, gráficos de Nyquist, gráficos de pólos e zeros.
 - Implementação da análise de resposta em frequência e análise de resposta no tempo.
 - Desenvolvimento de funções de transferência e espaço de estados.
 - Integração do filtro de Kalman para análise avançada.
2. **Ampliação de Opções de Plantas ou Dispositivos Industriais:**
 - Adição de novas plantas industriais e dispositivos simulados.
 - Desenvolvimento de novas descrições, diagramas, resultados e objetivos de aprendizagem específicos para cada novo dispositivo.
3. **Criação de Novos Desafios e Tarefas:**
 - Desenvolvimento de tarefas ou desafios adicionais com metas específicas e conquistas.
 - Implementação de sistemas de feedback mais sofisticados, incluindo dicas, sugestões, avisos, mensagens e indicadores mais detalhados.
4. **Expansão da Interatividade e Realismo do Ambiente:**
 - Melhorias na interface gráfica criada com Unity para aumentar o realismo das simulações.
 - Implementação de mais ferramentas de controle e análise dentro do ambiente virtual.
5. **Aprimoramento da Infraestrutura de Comunicação:**
 - Otimização do uso do MQTT para a troca de dados entre Unity e o servidor.
 - Melhoria na integração e desempenho dos containers Docker que executam os componentes do sistema.
6. **Ampliação e Diversificação da Base de Dados:**
 - Expansão do banco de dados MongoDB para armazenar e recuperar uma gama mais ampla de dados de processos físicos.
 - Implementação de novos métodos de armazenamento e recuperação de dados para otimizar a simulação e análise dos processos.
7. **Adaptação e Integração com Novos Dispositivos e Sensores:**

- Desenvolvimento de compatibilidade com uma gama mais ampla de microcontroladores e sensores.
- Integração com novos dispositivos de hardware para expandir as capacidades práticas do ambiente virtual.

8. Desenvolvimento de Recursos Educacionais Adicionais:

- Criação de tutoriais e materiais didáticos adicionais para ajudar os usuários a entender e utilizar melhor as ferramentas de simulação e análise.
- Implementação de módulos de aprendizagem adaptativa para personalizar a experiência de acordo com o nível de conhecimento do usuário.

7 CONCLUSÃO

A aplicação da Realidade Virtual (RV) no ensino de controle de processos, conforme explorado neste projeto, revelou-se uma abordagem promissora e potencialmente vantajosa. A plataforma desenvolvida demonstrou a capacidade de replicar sistemas de controle de processos com um grau significativo de precisão, proporcionando uma experiência de aprendizado que pode ser imersiva e interativa. Embora os laboratórios virtuais ofereçam vantagens notáveis, como flexibilidade e acessibilidade, é importante reconhecer que eles complementam, e não substituem completamente, os métodos tradicionais de ensino.

Os objetivos estabelecidos no início deste trabalho foram, em grande medida, alcançados. A criação de um ambiente que une teoria e prática de forma coerente e eficaz foi realizada com sucesso. No entanto, a eficácia e a viabilidade do ambiente virtual como uma alternativa aos laboratórios físicos tradicionais dependem de diversos fatores, incluindo a infraestrutura tecnológica disponível e a receptividade dos usuários. A plataforma apresentada pode oferecer vantagens adicionais, como segurança aumentada e redução de custos, mas esses benefícios podem variar conforme o contexto de aplicação.

O impacto deste projeto no ensino de controle de processos é relevante, destacando-se como uma contribuição significativa para o uso de tecnologias emergentes na educação. A adoção de RV e gêmeos digitais pode preparar melhor os alunos para as demandas da Indústria 4.0, promovendo uma formação mais dinâmica e alinhada com as necessidades contemporâneas. No entanto, o sucesso da implementação dessas tecnologias depende de uma contínua adaptação e aprimoramento, assim como da aceitação e integração por parte das instituições de ensino e dos alunos.

Em conclusão, o ambiente virtual desenvolvido neste trabalho oferece uma ferramenta útil e inovadora para o ensino de controle de processos. Ele complementa o aprendizado teórico com uma experiência prática que pode ser enriquecedora, ajudando a preparar os alunos para os desafios do mundo moderno. A continuidade das pesquisas e o aprimoramento desta plataforma têm o potencial de expandir ainda mais suas aplicações e eficácia, contribuindo para a evolução dos métodos de ensino em disciplinas técnicas e científicas.

REFERÊNCIAS

AHMAD, Saleh; ALHAYYAS, Shamma; ALMANSOORI, Maryam; ALMENHALI, Noof; ALSUDAIN, Fatima; ALKHALDI, Afrah. **Remote Control of the FESTO MPS PA Compact Workstation for the Development of a Remotely Accessible Process Control Laboratory**. Int. J. Online Biomed. Eng., v. 16, p. 84-103, 2020. DOI: 10.3991/ijoe.v16i05.12809.

ÅSTRÖM, K. J.; MURRAY, R. M. **Feedback Systems: An Introduction for Scientists and Engineers**. Princeton: Princeton University Press, 2008. DOI: 10.2307/j.ctvcm4gh1.

BOSCHERT, S.; ROSEN, R. **Digital Twin—The Simulation Aspect**. In: **Mechatronic Futures**. Springer, 2016. DOI: 10.1007/978-3-319-32156-1_5.

BUILT IN. **Virtual Reality in Education: Benefits, Uses & Examples**. 2022. Disponível em: <https://builtin.com/articles/virtual-reality-in-education>. Acesso em: 11 jun. 2024.

CABELLO, R. **Three.js Documentation**. 2010. Disponível em: <https://threejs.org/docs/>. Acesso em: 11 jun. 2024.

CHODOROW, K. **MongoDB: The Definitive Guide**. Sebastopol: O'Reilly Media, Inc., 2013.

DORF, R. C.; BISHOP, R. H. **Modern Control Systems**. 12. ed. Upper Saddle River: Pearson, 2011.

ESPRESSIF SYSTEMS. **ESP32 Overview & ESP01 Overview**. 2023. Disponível em: <https://www.espressif.com/en/products/socs/esp32>. Acesso em: 11 jun. 2024.

FROYD, J.; OHLAND, M. **Integrated Engineering Curricula**. **Journal of Engineering Education**, v. 94, 2005. DOI: 10.1002/j.2168-9830.2005.tb00835.x.

FULLER, A.; FAN, Z.; DAY, C.; BARLOW, C. **Digital Twin: Enabling Technologies, Challenges and Open Research**. IEEE Access, v. 8, p. 108952-108971, 2020. DOI: 10.1109/ACCESS.2020.2998358.

GRIEVES, M.; VICKERS, J. **Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems**. In: **Transdisciplinary Perspectives on Complex Systems**. Springer, 2017. DOI: 10.1007/978-3-319-38756-7_4.

HÄSTBACKA, David; VEPSÄLÄINEN, Timo; KUIKKA, S. **Model-driven development of industrial process control applications**. J. Syst. Softw., v. 84, p. 1100-1113, 2011. DOI: 10.1016/j.jss.2011.01.063.

HEILIG, M. Sensorama Simulator. US Patent 3,050,870, 14 ago. 1962.

HUNKELER, U.; TRUONG, H. L.; STANFORD-CLARK, A. **MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks**. In: 2008 3rd International Conference on Communication Systems Software and Middleware and Workshops, 2008. IEEE, p. 791-798. DOI: 10.1109/COMSWA.2008.4554519.

ISETPOINT. **Bancada Festo**. Disponível em: <https://isetpoint.wordpress.com/set-point/network/stand-by/atividades/bancada-festo/>. Acesso em: 19 dez. 2024.

KHARKI, Khadija El; BERRADA, K.; BURGOS, D. **Design and Implementation of a Virtual Laboratory for Physics Subjects in Moroccan Universities**. Sustainability, 2021. DOI: 10.3390/SU13073711.

KÜNZ, Andreas; ROSMANN, Sabrina; LORIA, Enrica; PIRKER, J. **The Potential of Augmented Reality for Digital Twins: A Literature Review**. 2022 IEEE Conference on Virtual Reality and 3D User Interfaces (VR), 2022, p. 389-398. DOI: 10.1109/VR51125.2022.00058.

MAROUKAS, Andreas; TROUSSAS, C.; KROUSKA, Akrivi; SGOUROPOULOU, C. **Virtual Reality in Education: A Review of Learning Theories, Approaches and Methodologies for the Last Decade**. Electronics, 2023. DOI: 10.3390/electronics12132832.

MAWSON, Victoria Jayne; HUGHES, B. **The development of modelling tools to improve energy efficiency in manufacturing processes and systems**. Journal of Manufacturing Systems, 2019. DOI: 10.1016/J.JMSY.2019.04.008.

MERKEL, D. **Docker: lightweight linux containers for consistent development and deployment**. Linux Journal, 2014, n. 239, p. 2, 2014.

MORDOR INTELLIGENCE. **Virtual Reality (VR) in Education Market - Size, Share & Industry Analysis**. 2022. Disponível em: <https://mordorintelligence.com>. Acesso em: 11 jun. 2024.

NORDQUIST, Thomas. **MQTT Explorer**. Disponível em: <https://github.com/thomasnordquist/MQTT-Explorer>. Acesso em: 12 jun. 2024.

- OGATA, K. **Modern Control Engineering**. 5. ed. Upper Saddle River: Prentice Hall, 2010.
- PAPANASTASIOU, George P.; DRIGAS, A.; SKIANIS, C.; LYTRAS, Miltiadis Demetrios; PAPANASTASIOU, Effrosyni. **Virtual and augmented reality effects on K-12, higher and tertiary education students' twenty-first century skills**. *Virtual Reality*, v. 23, p. 425-436, 2018. DOI: 10.1007/s10055-018-0363-2.
- PEDRO, Francisco Xavier; TEIXEIRA, Adriano Canabarro. **The Impact of Accelerated Digital Transformation on Educational Institutions**. *Handbook of Research on Developing a Post-Pandemic Paradigm for Virtual Technologies in Higher Education*, 2021. DOI: 10.4018/978-1-7998-6963-4.ch001.
- SEBORG, D. E.; EDGAR, T. F.; MELLICHAMP, D. A. *Process Dynamics and Control*. 3. ed. Hoboken: John Wiley & Sons, 2010.
- SELEDTSOVA, I. A.; CHECHURIN, L. **Blending Traditional and Modern Approaches to Teaching Control Theory**. 2019. p. 632-642. DOI: 10.1007/978-3-030-34983-7_62.
- SOLIMAN, M. **Using Virtual Experiments in Teaching Control Theory**. 2018. p. 70-80. DOI: 10.1007/978-3-030-11434-3_12.
- SUTHERLAND, Ivan E. **A head-mounted three dimensional display**. In: PROCEEDINGS OF THE DECEMBER 9-11, 1968, FALL JOINT COMPUTER CONFERENCE, PART I, 1968, San Francisco, California. New York: Association for Computing Machinery, 1968. p. 757-764. DOI: 10.1145/1476589.1476686.
- TILKOV, S.; VINOSKI, S. Node.js: **Using JavaScript to build high-performance network programs**. *IEEE Internet Computing*, v. 14, n. 6, p. 80-83, 2010. DOI: 10.1109/MIC.2010.145.
- UNITY TECHNOLOGIES. **Unity Documentation**. 2023. Disponível em: <https://docs.unity3d.com/Manual/index.html>. Acesso em: 11 jun. 2024.
- VIGANO, Gian Paolo. **M2MqttUnity**. 2021. Disponível em: <https://github.com/gpvigano/M2MqttUnity>. Acesso em: 11 jun. 2024.
- WOLF, T. **Assessing Student Learning in a Virtual Laboratory Environment**. *IEEE Transactions on Education*, v. 53, p. 216-222, 2010. DOI: 10.1109/TE.2008.2012114.