

Kelly Cristina Félix Machado

**Simulação de uma Linha de Envase em
Realidade Virtual Controlada via CLP Virtual
com Comunicação Modbus TCP**

Uberlândia, MG

2026

Kelly Cristina Félix Machado

Simulação de uma Linha de Envase em Realidade Virtual Controlada via CLP Virtual com Comunicação Modbus TCP

Projeto de Trabalho de Conclusão de Curso da Engenharia de Controle e Automação da Universidade Federal de Uberlândia - UFU - Campus Santa Mônica, como requisito para a obtenção do título de Graduação em Engenharia de Controle e Automação.

Universidade Federal de Uberlândia - UFU
Faculdade de Engenharia Elétrica - FEELT

Orientador: Prof. Dr. Josué Silva de Morais
Coorientador: Prof. Dr. Alexandre Cardoso

Uberlândia, MG

2026

Kelly Cristina Félix Machado

Simulação de uma Linha de Envase em Realidade Virtual Controlada via CLP Virtual com Comunicação Modbus TCP/ Kelly Cristina Félix Machado. – Uberlândia, MG, 2026-
Orientador: Prof. Dr. Josué Silva de Moraes

Trabalho de Conclusão de Curso – Universidade Federal de Uberlândia - UFU
Faculdade de Engenharia Elétrica - FEELT, 2026.

1. Automação Industrial. 2. Realidade Virtual. I. Josué Silva de Moraes. II. Universidade Federal de Uberlândia. III. Graduação em Engenharia de Controle e Automação. IV. Simulação de uma Linha de Envase em Realidade Virtual Controlada via CLP Virtual com Comunicação Modbus TCP.

CDU 02:141:005.7

Dedico este trabalho à memória de minha avó Rita Cecília, que foi quem permitiu que eu me tornasse a pessoa que sou hoje. Com carinho e gratidão, levarei sempre seus ensinamentos em minha jornada e suas lembranças em meu coração.

Agradecimentos

À energia superior que rege e equilibra todo o universo, seja ela qual for.

A mim mesma, por ter tido a coragem de dar o primeiro passo para mudar a minha realidade e por ter persistido na busca pelos meus sonhos.

À minha família, em especial à minha vó Rita Cecília e ao meu tio Rosenberg Machado, por todo o apoio ao longo dessa etapa crucial da minha vida. Sem vocês, eu não teria tido a motivação necessária para superar esse desafio.

À minha namorada, Aline Machado, e a toda a sua família, por terem me acolhido e me oferecido apoio nos momentos em que mais precisei. Sem vocês, essa conquista não teria sido possível.

Aos meus amigos, por tornarem essa jornada mais leve e por contribuírem para dias mais felizes durante esse percurso.

A todos os funcionários da UFU, pelo cuidado e dedicação em garantir um ambiente adequado para os alunos.

Aos professores, pela paciência, dedicação e pelos conhecimentos transmitidos ao longo da graduação, fundamentais para a minha formação profissional.

Ao orientador e ao coorientador, pelo suporte, pelos ensinamentos e por todo o apoio oferecido durante o desenvolvimento deste trabalho.

Às políticas sociais que possibilitaram meu ingresso e permanência na universidade, sem as quais eu não teria tido a oportunidade e a honra de fazer parte da história desta instituição.

“Quando se nasce pobre, ser estudioso é o maior ato de rebeldia contra o sistema.”
(Autor Desconhecido)

Resumo

O presente trabalho propõe o desenvolvimento de uma linha de envase de garrafas simulada em um ambiente de realidade virtual, controlada por meio de um CLP virtual programado com lógica em linguagem Ladder e integrada por meio do protocolo de comunicação industrial Modbus TCP. A proposta demonstra como tecnologias associadas à Indústria 4.0 podem ser integradas a sistemas tradicionais de automação industrial. O estudo abrange o desenvolvimento dos modelos tridimensionais da simulação utilizando o software Autodesk Inventor, a implementação das mecânicas e dos processos dinâmicos no software Unity, o desenvolvimento do CLP virtual e da lógica de controle em linguagem Ladder no software CODESYS, bem como a configuração da comunicação Modbus TCP entre os ambientes. Além disso, a aplicação foi desenvolvida para execução em realidade virtual, sendo validada por meio do óculos *Meta Quest 3S*. Os testes realizados permitiram identificar pontos de melhoria e gargalos de desempenho, entretanto, de forma geral, o sistema atingiu os objetivos propostos, mostrando-se uma solução viável para a identificação de falhas, validação do controle e análise de processos industriais em um ambiente tridimensional interativo e de baixo custo experimental.

Palavras-chave: Modbus TCP; Indústria 4.0; Automação industrial; Realidade virtual; Simulação; CLP.

Abstract

This work proposes the development of a bottle filling line simulated in a virtual reality environment, controlled by a virtual PLC programmed using Ladder logic and integrated through the Modbus TCP industrial communication protocol. The proposal demonstrates how technologies associated with Industry 4.0 can be integrated with traditional industrial automation systems. The study covers the development of the simulation's three-dimensional models using the Autodesk Inventor software, the implementation of mechanical behaviors and dynamic processes in the Unity software, the development of the virtual PLC and control logic in Ladder language using the CODESYS software, as well as the configuration of Modbus TCP communication between the environments. In addition, the application was developed for execution in virtual reality and validated using the Meta Quest 3S headset. The conducted tests allowed the identification of performance bottlenecks and improvement opportunities; however, overall, the system achieved the proposed objectives, proving to be a viable solution for fault identification, control validation, and analysis of industrial processes in an interactive three-dimensional environment with low experimental cost.

Key-words: Modbus TCP; Industry 4.0; Industrial automation; Virtual reality; Simulation; PLC.

Lista de ilustrações

Figura 1 – Controle em malha aberta.	18
Figura 2 – Controle em malha fechada.	19
Figura 3 – Estrutura do sistema de controle.	19
Figura 4 – CLP Rockwell 1763-116bwa.	20
Figura 5 – Estrutura do controlador lógico programável.	21
Figura 6 – Ciclo de scan.	22
Figura 7 – Exemplo de programação em Ladder.	22
Figura 8 – Exemplo de programação em Instruction List.	23
Figura 9 – Esquema de camadas do CODESYS.	24
Figura 10 – Interface inicial do CODESYS.	25
Figura 11 – Modelo <i>client/server</i>	27
Figura 12 – Framing de dados nos modos ASCII e RTU.	28
Figura 13 – Framing da mensagem nos modos ASCII e RTU.	28
Figura 14 – Endereçamento de dados no Modbus.	29
Figura 15 – Códigos das funções.	29
Figura 16 – Encapsulamento da mensagem.	30
Figura 17 – Ambiente em realidade virtual.	31
Figura 18 – Objetos virtuais sobrepondo a realidade.	32
Figura 19 – Diferença entre objeto tridimensional sem iluminação e com iluminação.	34
Figura 20 – Aba de ferramentas com recursos de projeção.	35
Figura 21 – Aba de ferramentas com recursos de análise.	35
Figura 22 – Interface principal do Unity.	36
Figura 23 – Processo de rotulagem.	38
Figura 24 – Processo de enchimento e vedação.	39
Figura 25 – Estrutura de comunicação Modbus TCP entre cliente (CLP virtual) e servidor (Unity).	40
Figura 26 – Modelo tridimensional da máquina envasadora.	41
Figura 27 – Exemplo de texturização dos modelos tridimensionais.	42
Figura 28 – Vista frontal do cenário.	43
Figura 29 – Vista lateral do cenário.	43
Figura 30 – Interface principal.	46
Figura 31 – Interface de não objetivo cumprido.	46
Figura 32 – Interface de objetivo cumprido.	47
Figura 33 – Implementação do servidor via C#.	49
Figura 34 – Definição de endereço IP e porta do servidor no CODESYS.	49
Figura 35 – Variáveis Modbus configuradas no CODESYS.	50

Figura 36 – Variáveis auxiliares.	51
Figura 37 – Lógica de <i>first scan</i> e parte da escrita de variáveis em linguagem Ladder. . .	52
Figura 38 – Trecho da lógica de controle do botão de acionamento e da contagem de garrafas.	52
Figura 39 – Lógica de reinicialização de variáveis.	53
Figura 40 – Funcionamento do botão de acionamento.	55
Figura 41 – Esteira transportadora em funcionamento durante a simulação.	56
Figura 42 – Máquina de envase em funcionamento durante a simulação.	56
Figura 43 – Máquina de selagem em funcionamento durante a simulação.	57
Figura 44 – Máquina de rotulagem em funcionamento durante a simulação.	57
Figura 45 – Captura de tráfego Modbus TCP em <i>loopback</i>	58
Figura 46 – Captura de tráfego Modbus TCP em dispositivos distintos.	59
Figura 47 – Comparação de cores e sombras entre o ambiente de desenvolvimento (superior) e de <i>build</i> (inferior).	61
Figura 48 – Taxa de quadros por segundo da aplicação em ambiente de desenvolvimento, com simulação à esquerda e cena inicial à direita.	61
Figura 49 – Taxa de quadros por segundo da aplicação em versão compilada (<i>build</i>). . .	62

Lista de tabelas

Tabela 1 – Comandos de Atuação das Máquinas do Processo	44
Tabela 2 – Mapeamento de Variáveis Modbus do Sistema	48

Lista de abreviaturas e siglas

IoT	Internet of Things
CLP	Controlador Lógico Programável
DT	Digital Twin
TCP/IP	Transmission Control Protocol/Internet Protocol
FPS	Frames Per Second
CPU	Central Processing Unit
GPU	Graphics Processing Unit

Sumário

1	INTRODUÇÃO	14
1.1	Justificativas	14
1.2	Objetivos	15
1.2.1	Objetivos Específicos	15
2	REFERENCIAIS TEÓRICOS	17
2.1	Automação Industrial	17
2.1.1	Conceitos de Automação	17
2.1.2	Controle de Processos Industriais	18
2.2	Controladores Lógicos Programáveis (CLPs)	20
2.2.1	Princípios de Funcionamento do CLP	20
2.2.2	CLP Virtual	23
2.3	Comunicação Industrial	25
2.3.1	Protocolos de Comunicação em Automação	25
2.3.2	Protocolo Modbus	26
2.3.3	WireShark	30
2.4	Simulação e Realidade Virtual	30
2.4.1	Simulação de Processos Industriais	30
2.4.2	Realidade Virtual na Automação	31
2.4.3	Manufatura Virtual e <i>Digital Twins</i>	32
2.5	Modelagem 3D de Ambientes Industriais	33
2.5.1	Fundamentos de Modelagem 3D	33
2.5.2	Uso do Autodesk Inventor	34
2.6	Simulação Interativa em Unity	35
2.6.1	Unity como Ferramenta de Simulação	35
2.6.2	Integração com Sistemas de Automação	37
2.6.3	Meta Quest Developer Hub	37
2.7	Processo de Envase	38
2.7.1	Conceito e Etapas de um Sistema de Envase	38
3	METODOLOGIA	40
3.1	Visão Geral do Sistema	40
3.2	Modelagem 3D no Autodesk Inventor	41
3.2.1	Criação dos Modelos	41
3.2.2	Preparação para Exportação	41
3.3	Desenvolvimento da Simulação no Unity	42

3.3.1	Importação e Configuração dos Modelos	42
3.3.2	Criação de Scripts de Interação e Controle	44
3.3.3	Interface e Visualização do Processo	45
3.3.4	Integração da Simulação com Realidade Virtual	47
3.4	Integração via Protocolo Modbus TCP	48
3.4.1	Implementação do Servidor no Unity	48
3.4.2	Configuração do Cliente no CODESYS	49
3.5	Desenvolvimento do Controle no CODESYS	50
3.5.1	Configuração das Variáveis e Integração com o Modbus TCP	50
3.5.2	Implementação da Lógica de Controle em Linguagem Ladder	51
3.6	Testes e Validação do Sistema	53
4	RESULTADOS E DISCUSSÕES	55
4.1	Simulação da Linha de Envase	55
4.2	Comunicação via Protocolo Modbus TCP	58
4.3	Lógica de Controle Implementada em Linguagem Ladder	59
4.4	Visualização do Processo em Ambiente de Realidade Virtual	60
5	CONCLUSÃO	63
	REFERÊNCIAS BIBLIOGRÁFICAS	65
	APÊNDICES	70
	APÊNDICE A – CÓDIGO LADDER NO CODESYS	71
	APÊNDICE B – CÓDIGO DO SERVER MODBUS NO UNITY	75
	APÊNDICE C – CÓDIGO DA MÁQUINA DE ENVASE NO UNITY	77
	APÊNDICE D – CÓDIGO DA MÁQUINA DE SELAGEM NO UNITY	83
	APÊNDICE E – CÓDIGO DA MÁQUINA DE ROTULAGEM NO UNITY	85

1 Introdução

A indústria de alimentos e bebidas é um dos setores mais relevantes da economia brasileira, respondendo por cerca de 10,8% do PIB nacional (Associação Brasileira da Indústria de Alimentos (ABIA), 2025). Nesse contexto, a busca por eficiência produtiva é essencial para garantir competitividade e sustentabilidade, especialmente em um cenário de crescente demanda e exigência por qualidade.

A fabricação de bebidas envolve dezenas de processos, dentre os quais se destaca o envase, um dos últimos da cadeia produtiva. Nessa etapa, a bebida produzida é transferida para seu recipiente final, seja ele uma garrafa ou uma lata, de modo a garantir a integridade do produto, preservando suas propriedades físicas e químicas e protegendo-o contra contaminações e violações (International Labour Organization, 2025). Esta é uma etapa considerada limitante em relação à produtividade quando comparada aos demais processos, devido às perdas significativas identificadas nela.

O controle dos equipamentos que compõem a linha de envase é geralmente realizado por um CLP (Controlador Lógico Programável), responsável por realizar a leitura e processamento dos dados recebidos via sensores, implementar a lógica dos processos e acionar os atuadores correspondentes às tarefas necessárias. A comunicação entre os dispositivos de campo e o CLP ocorre por meio de protocolos industriais, sendo o Modbus TCP comumente utilizado, devido à sua simplicidade, confiabilidade e ampla adoção nas indústrias tradicionais (TAPIA et al., 2023).

Com o avanço da indústria 4.0, surgiram métodos de mitigar as restrições de produtividade encontradas nas linha de envase, por meio da aplicação de tecnologias como IoT (Internet of Things) e realidade virtual. Esta última permite projetar de forma integrada o controle de uma linha de envase, simulando o comportamento dos equipamentos em diferentes cenários, buscando otimizar o processo, antecipando falhas e aumentando o desempenho produtivo.

Com base nisso, o presente trabalho abordará o desenvolvimento de modelos digitais capazes de representar, de forma precisa, a estrutura e comportamento dos equipamentos que fazem parte de uma linha de envase. Esses modelos serão integrados a um CLP virtual por meio do protocolo industrial Modbus TCP, com o objetivo de realizar o controle do processo utilizando a linguagem Ladder. Por fim, tanto o modelo quanto os resultados do controle poderão ser visualizados em ambiente de realidade virtual, utilizando um óculos de realidade imersiva.

1.1 Justificativas

Entre os diversos processos envolvidos na fabricação de bebidas, o envase é frequentemente apontado como o principal gargalo de produção, tendo como ponto crítico a máquina de

enchimento, pois este é um equipamento de menor velocidade nominal comparado aos demais da cadeia produtiva. Além disso, estudos indicam que as principais causas da baixa eficiência nessa etapa estão relacionadas a perdas de qualidade e paradas por falhas mecânicas ou operacionais (SANTOS, 2008). A automação industrial surge como alternativa eficaz para corrigir essas limitações, promovendo o balanceamento da linha e a fluidez do processo.

No contexto da Indústria 4.0, tecnologias como IoT (*Internet of Things*) e realidade virtual permitem a análise e integração de processos físicos e virtuais, empregando o conceito de manufatura virtual, que consiste em utilizar modelos digitais para representar de forma precisa a estrutura do sistema e simular seus comportamentos lógicos e físicos (IWATA et al., 1995).

Dessa forma, evidencia-se a necessidade de demonstrar, de forma prática, como essas tecnologias podem ser aplicadas na otimização de uma linha de envase de garrafas. O estudo busca mostrar a integração entre as novas tecnologias da indústria 4.0 e os sistemas já utilizadas na indústria, bem como o CLP e a comunicação via protocolo Modbus TCP.

1.2 Objetivos

Este trabalho tem como principal objetivo desenvolver um modelo digital, em ambiente de realidade virtual, que represente de forma fiel uma linha de envase de garrafas, implementando, em seguida, o controle desse processo por meio de um CLP virtual que se comunicará com a simulação via protocolo Modbus TCP.

1.2.1 Objetivos Específicos

- Desenvolver modelos tridimensionais representativos dos equipamentos que compõem a linha de envase, bem como do ambiente no qual o processo está inserido, utilizando o software Autodesk Inventor;
- Implementar a lógica de funcionamento dos modelos tridimensionais, de forma a reproduzir o comportamento físico e operacional dos equipamentos, por meio do software Unity 3D;
- Estabelecer a comunicação entre o software de simulação Unity 3D e o software de controle CODESYS, utilizando o protocolo industrial Modbus TCP;
- Implementar a lógica de controle do processo em linguagem Ladder, por meio de um CLP virtual disponibilizado pelo software CODESYS;
- Desenvolver interfaces de interação e supervisão do processo, possibilitando o monitoramento em tempo real da simulação;
- Integrar a simulação a uma plataforma de realidade virtual, permitindo a visualização imersiva do processo;

- Validar o sistema por meio da interação e visualização em um ambiente imersivo, avaliando a efetividade do controle projetado e identificando possíveis pontos de falha.

2 Referenciais Teóricos

2.1 Automação Industrial

2.1.1 Conceitos de Automação

A automação pode ser compreendida como o conjunto de atividades voltadas a auxiliar ou substituir a mão de obra humana na execução de determinadas tarefas, como descreve Groover (2010). Os primeiros sistemas considerados automatizados foram os moinhos e rodas d'água, os quais convertiam energia cinética em energia mecânica, aplicando-a em atividades agrícolas e de produção. Com o surgimento da máquina a vapor, o conceito de automação começou a se popularizar nas indústrias, caracterizando a 1ª Revolução Industrial (SAKURAI; ZUCHI, 2018). Porém, a atuação autônoma das máquinas se dava de forma limitada, uma vez que havia grande necessidade de intervenção humana para realizar a integração entre os equipamentos.

Outro fator determinante para a evolução da automação nas indústrias foi o surgimento da energia elétrica, quando as máquinas a vapor foram substituídas por equipamentos elétricos, dando início à 2ª Revolução Industrial. O período também foi marcado pela criação do processo de semi-automatização, por Henry Ford, que revolucionou a indústria automobilística com a implementação de linhas de montagem parcialmente automatizadas (SILVA; GASPARIN, 2015).

Seguindo para a 3ª Revolução Industrial, houve um grande avanço tecnológico que introduziu grandes inovações para a indústria, como descobertas do campo da informática, robótica, telecomunicações, entre outros (SAKURAI; ZUCHI, 2018). Com base nesse contexto, a busca pelo aprimoramento das técnicas de automação levou ao investimento crescente em novas tecnologias nas fábricas, resultando na 4ª Revolução Industrial.

Com o desenvolvimento tecnológico e, conseqüentemente, a maior acessibilidade das tecnologias em termos de custo, o antigo problema de limitação na integração entre os equipamentos pôde ser solucionado com a utilização de IoT e *Big Data Analytics*, que permitiram a otimização dos processos e possibilitaram a troca de dados entre os sistemas automatizados que controlam os equipamentos industriais (SACOMANO et al., 2019). Essa é uma das principais características da 4ª Revolução Industrial, período em que o setor industrial se encontra atualmente.

Ao longo de todos esses períodos da história, a automação industrial demonstrou sua importância ao contribuir para o alcance de metas constantemente buscadas pelo setor, como a melhoria das condições de operação, o aumento da qualidade dos produtos, a ampliação da segurança, a flexibilidade em relação aos parâmetros do processo e o crescimento da produtividade, além de proporcionar maior controle sobre a cadeia produtiva (COSTA; LISBOA; SANTOS, 2003).

2.1.2 Controle de Processos Industriais

O controle automático pode ser definido como o método de manter determinadas variáveis de um processo dentro de faixas específicas de valores, sem a necessidade de intervenção direta de um operador humano. Garcia (2015) destaca que a aplicação desse conceito nas indústrias de processo dá origem ao termo *controle de processos*, que consiste na manipulação das variáveis de um sistema produtivo com o objetivo de obter o melhor desempenho possível.

O tipo de controle pode ser classificado de acordo com a intervenção do atuador, sendo dividido em controle manual e controle automático. No controle manual, destaca-se a necessidade de um operador para aplicar energia ao sistema por meio do atuador (RIBEIRO, 2020). Já no controle automático, o momento em que o atuador é acionado é determinado por um equipamento de controle, como um CLP.

Outra forma de classificar os sistemas de controle baseia-se na forma como a resposta de saída é considerada para a determinação do sinal de entrada do processo. Quando o controle atua aplicando sinais de entrada de forma independente da saída do sistema, ele é denominado controle de malha aberta. Por outro lado, quando o controle utiliza informações da saída para ajustar a entrada, com base no erro entre o valor de referência e o valor medido, ele é denominado controle de malha fechada.

O controle em malha aberta apresenta uma estrutura simples, composta basicamente por um sinal de entrada que é processado pelo controlador e enviado à planta, a qual responde a esse estímulo, como mostra a Figura 1.

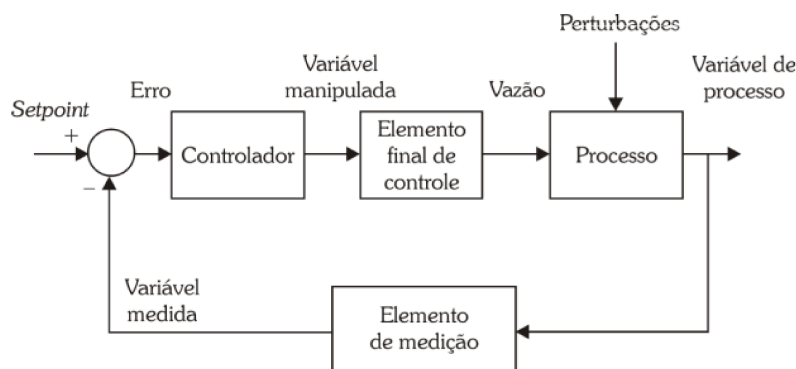
Figura 1 – Controle em malha aberta.



Fonte: Franchi (2014).

O controle em malha fechada, por sua vez, apresenta uma estrutura mais elaborada (Fig. 2), levando em consideração um *setpoint*, ou seja, o valor de referência que a variável de saída deve atingir. A entrada é aplicada ao sistema, processada pelo controlador e enviada à planta, assim como no sistema em malha aberta. No entanto, nesse caso, há um sinal de realimentação (*feedback*) proveniente de sensores instalados na saída do processo. O valor medido é comparado ao setpoint, determinando o erro da saída e permitindo o ajuste em tempo real do sinal de entrada (RIBEIRO, 2020).

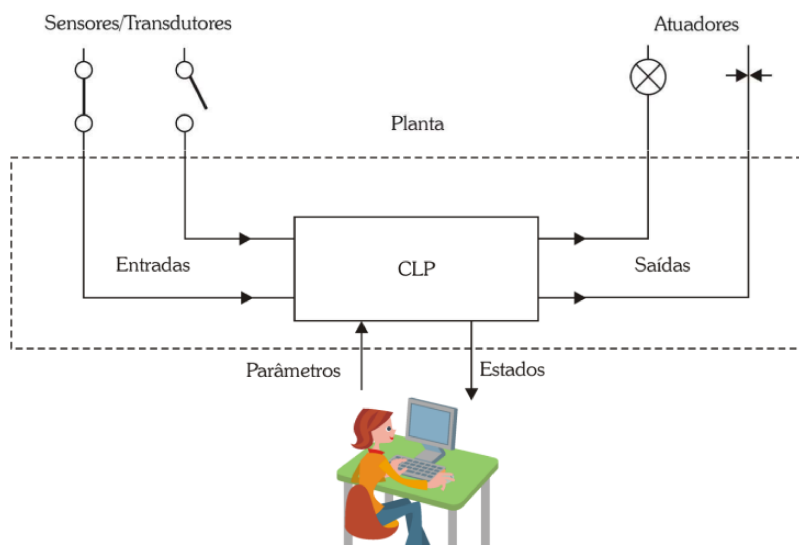
Figura 2 – Controle em malha fechada.



Fonte: Franchi (2014).

A estrutura de um sistema de controle (Fig. 3), de forma geral, é composta por alguns elementos básicos: sensores, controladores, atuadores e indicadores. Os sensores são responsáveis por converter as grandezas físicas ou químicas do processo em valores mensuráveis, que podem ser interpretados em forma de dados. Os atuadores, como o próprio nome indica, são dispositivos que atuam diretamente sobre o processo, executando ações conforme os comandos enviados pelo controlador. O controlador realiza a leitura dos sinais provenientes dos sensores, processa essas informações e, de acordo com a lógica de controle implementada, envia os sinais de comando aos atuadores. Por fim, os indicadores apresentam, de maneira mais amigável ao operador, os valores das variáveis de processo em tempo real, possibilitando a supervisão e o acompanhamento do sistema.

Figura 3 – Estrutura do sistema de controle.



Fonte: Franchi (2014).

A automação, aplicada ao controle de processos, destaca-se na diminuição de custos com retrabalhos e perdas, por meio de um planejamento mais eficiente. Em uma linha de envase, por exemplo, diversos processos são controlados, como o da esteira, da válvula de enchimento

e dos sensores de nível. Nesse contexto, estudos apontam que, a efetivação da automação no controle de processos resultou em um aumento significativo da eficiência produtiva e uma notável diminuição nos custos de produção (PEREIRA; JÚNIOR, 2023).

2.2 Controladores Lógicos Programáveis (CLPs)

2.2.1 Princípios de Funcionamento do CLP

De acordo com a definição de Rosário (2009), um Controlador Lógico Programável (CLP) (Fig. 4) é um dispositivo microprocessado que, por meio de interfaces de entrada e saída, permite a execução de tarefas dedicadas à automação e à supervisão de processos industriais. O surgimento desses dispositivos na década de 1970 revolucionou os sistemas de comando e controle industrial, trazendo grandes vantagens, como a possibilidade de reprogramação, que transferiu as modificações antes realizadas em *hardware* para o ambiente de *software* (ROGGIA; FUENTES, 2016).

Figura 4 – CLP Rockwell 1763-116bwa.

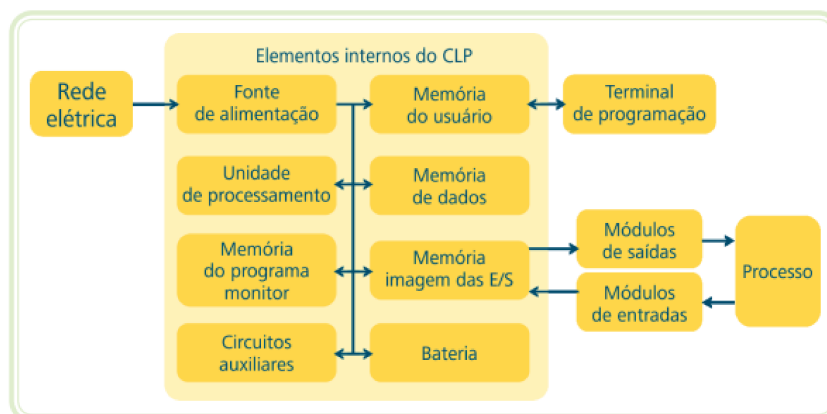


Fonte: Rockwell Automation (2025).

Os CLPs foram desenvolvidos inicialmente para substituir os sistemas de controle baseados em relés e contadores. Porém, com o passar do tempo, esses dispositivos evoluíram a ponto de realizarem também o controle em nível analógico.

A estrutura que compõe o CLP é formada por diversos elementos internos e externos, que incluem terminais de programação, módulos de entrada e saída, e interfaces de conexão com a rede, conforme ilustrado na Figura 5.

Figura 5 – Estrutura do controlador lógico programável.



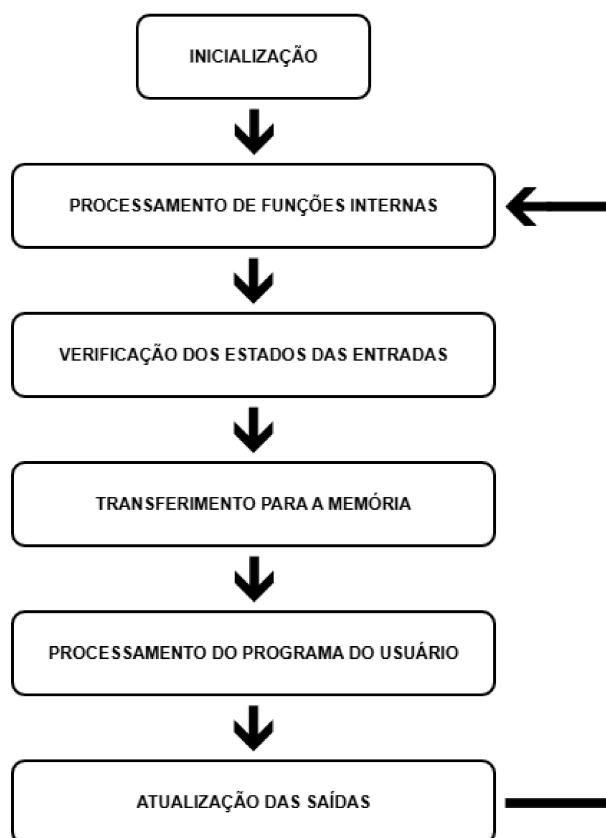
Fonte: Roggia e Fuentes (2016).

Cada elemento que compõe o CLP possui funções específicas, conforme descreve Roggia e Fuentes (2016). A fonte de alimentação tem como finalidade converter a tensão da rede (110 a 220 Vca) para o valor de alimentação dos circuitos eletrônicos (5 a 12 Vcc), manter a bateria do dispositivo carregada e fornecer tensão para as entradas e saídas do CLP (12 ou 24 Vcc). A bateria garante que o relógio interno do CLP, que funciona em tempo real, assim como os programas e parâmetros salvos, sejam mantidos mesmo após a queda de energia. A unidade de processamento (CPU) é responsável pelos cálculos aritméticos, pelo controle e pelo funcionamento lógico de todos os circuitos.

Outro elemento muito importante do CLP é a memória, que pode ser classificada em quatro tipos: memória do monitor, memória do usuário, memória de dados e memória de imagem das entradas e saídas (ROGGIA; FUENTES, 2016). A memória do monitor armazena o programa principal do CLP, que funciona como um sistema operacional e não pode ser alterado pelo usuário. A memória do usuário armazena o programa desenvolvido pelo usuário, utilizando memórias RAM, EPROM, EEPROM ou FLASH-EPROM. A memória de dados armazena informações do programa do usuário, como temporizadores, senhas e parâmetros variados. Por fim, a memória de imagem das entradas e saídas registra os estados de cada uma delas, funcionando como uma tabela consultada pelo CLP durante o processamento do programa do usuário.

O funcionamento de um CLP baseia-se na execução cíclica de três passos principais, descritos por Baptista (2015): leitura das entradas, execução da lógica de controle e atualização das saídas. Esse processo é conhecido como varredura, ou ciclo de *scan*, sendo o tempo de scan definido como o tempo necessário para que o dispositivo complete um ciclo completo. No primeiro passo, o controlador lê os valores de entrada de cada canal, seja digital ou analógico, e armazena o estado correspondente na memória. Em seguida, os dados lidos são processados pela lógica de controle, aplicando operações e interpretando os valores de forma a gerar resultados que serão utilizados na etapa seguinte. No terceiro e último passo, o controlador atualiza os estados das saídas com base nos resultados obtidos, completando assim um ciclo de *scan* (Fig. 6).

Figura 6 – Ciclo de scan.

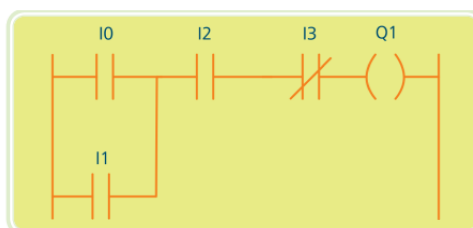


Fonte: Adaptado de Baptista (2015).

De acordo com a norma IEC 61131-3, o desenvolvimento de programas para CLPs pode ser realizado em uma dentre quatro linguagens de programação: *Ladder Diagram (LD)*, *Function Block Diagram (FBD)*, *Instruction List (IL)* e *Structured Text (ST)*. As duas primeiras são linguagens de representação gráfica, enquanto as demais são linguagens textuais.

A linguagem *Ladder*, também conhecida como linguagem de contatos, é utilizada para o desenvolvimento de circuitos lógicos simples e combinacionais, baseados em contatos que representam os módulos de entrada e bobinas que correspondem às variáveis de saída (Fig. 7). Essa linguagem também dispõe de blocos funcionais destinados a operações específicas, como comunicação, temporização, contagem e registro. Seu entendimento é facilitado pelo fato de sua representação gráfica assemelhar-se aos esquemas elétricos clássicos (BAPTISTA, 2015).

Figura 7 – Exemplo de programação em Ladder.

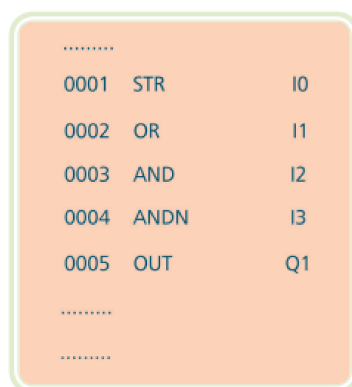


Fonte: Roggia e Fuentes (2016).

A linguagem *Function Block Diagram* baseia-se na representação de portas lógicas para o desenvolvimento de programas em CLPs, sendo amplamente utilizada por técnicos em eletrônica digital devido à familiaridade com esses símbolos (ROGGIA; FUENTES, 2016).

A *Instruction List* é uma linguagem textual convencional, em que toda a lógica é descrita por meio de instruções de máquina para que o equipamento possa interpretá-las corretamente (Fig. 8). Apesar de sua robustez, apresenta como desvantagem a dificuldade na identificação de erros durante a programação, uma vez que não possui representação gráfica como as linguagens anteriores, exigindo maior nível de entendimento por parte do programador. O mesmo ocorre com a *Structured Text*, que, embora mais evoluída como descreve Baptista (2015), demanda um conhecimento mais avançado de programação para ser plenamente compreendida e utilizada de forma eficiente .

Figura 8 – Exemplo de programação em Instruction List.

A imagem mostra um exemplo de programação em Instruction List (IL) dentro de um retângulo laranja com uma borda verde. O código é exibido em uma fonte monoespaçada azul. No topo e no fundo do bloco de código, há linhas de pontos de separação. O código em si consiste em cinco linhas, cada uma com um endereço numérico de quatro dígitos, um código de instrução de três letras e um destino de dois dígitos.

0001	STR	I0
0002	OR	I1
0003	AND	I2
0004	ANDN	I3
0005	OUT	Q1

Fonte: Roggia e Fuentes (2016).

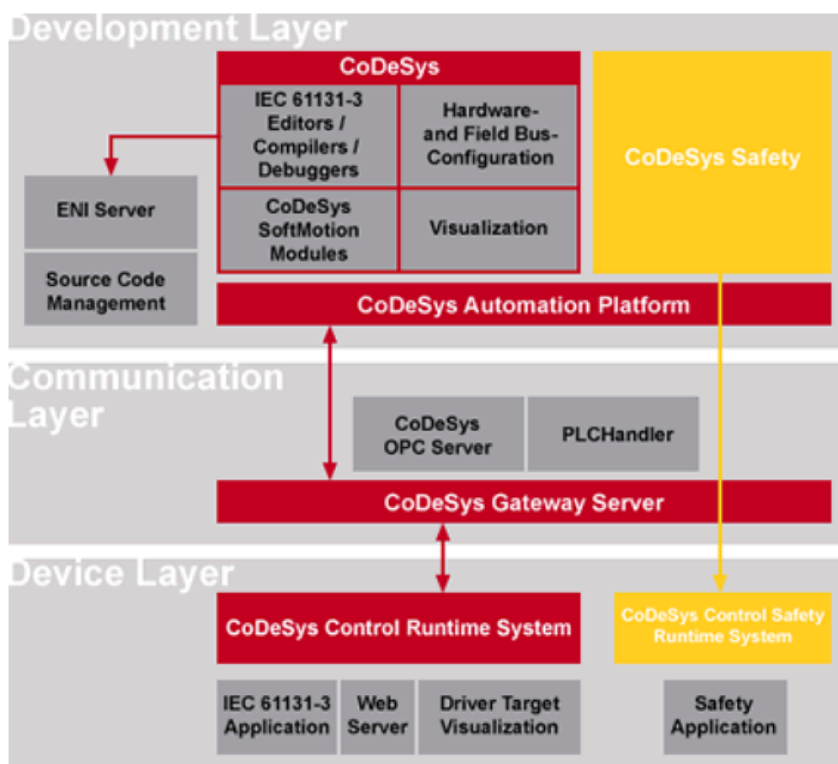
2.2.2 CLP Virtual

Um CLP virtual é um *software* que replica de forma fiel as funcionalidades de um controlador industrial real, permitindo o desenvolvimento de lógicas de controle e a realização de testes sem a necessidade do equipamento físico, tornando essas tecnologias mais acessíveis financeiramente (JUNIOR, 2021). A réplica virtual encontra aplicação em diversas áreas, com destaque para o ensino, simulação e prototipagem, aumentando a segurança ao possibilitar a execução de tarefas e testes em um ambiente controlado.

Uma ferramenta que se destaca na área de automação é o CODESYS (*Controller Development System*), um *software* criado para ser uma plataforma de desenvolvimento de programas para CLPs, controladores de automação programáveis (PCAs) e outros dispositivos de automação (CODESYS GmbH, 2025). Além disso, o CODESYS é amplamente utilizado por ser gratuito e por ser um dos poucos softwares padronizados de acordo com a norma IEC 61131-3 (SOUZA, 2014). Entre suas funcionalidades, destaca-se a disponibilização de CLPs virtuais.

O sistema do *software* pode ser resumido em 3 camadas (*layers*), sendo elas: *Development Layer*, *Device Layer* e *Communication Layer* (Fig. 9). A *Development Layer* corresponde à camada de interação com o usuário, permitindo o desenvolvimento de programas. A *Device Layer* é a camada de nível mais baixo, responsável pela interação com o hardware. É importante salientar que, para que um dispositivo seja programado com a ferramenta, é necessário que ele possua o *Runtime System* do CODESYS implementado. Por fim, a *Communication Layer* gerencia a comunicação entre as duas camadas anteriores, utilizando um *OPC Server* que possibilita a troca de dados (SILVA, 2017).

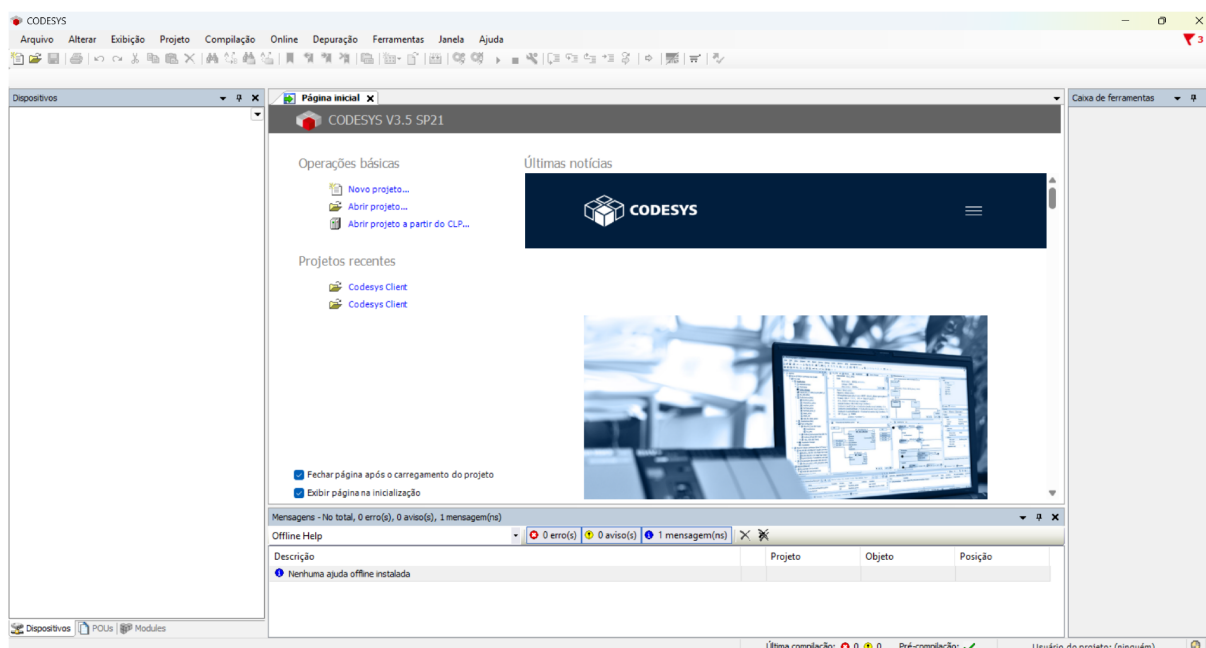
Figura 9 – Esquema de camadas do CODESYS.



Fonte: 3S-Smart Software Solutions GmbH (2015).

A configuração do CLP virtual no CODESYS é simples, sendo necessário um *runtime* de simulação, como o *CODESYS Control Win SL (x64)*, que é instalado junto ao software. Com o runtime instalado, basta abrir o programa e, através de uma interface intuitiva (Fig. 10), criar um novo projeto, selecionar o dispositivo simulado pelo runtime, escolher a linguagem de programação (como Ladder ou ST), realizar o login e fazer o download do programa para o controlador. Após esses passos, o CLP virtual estará em modo *STOP*, sendo necessário selecionar o modo *RUN* para iniciar a simulação.

Figura 10 – Interface inicial do CODESYS.



Fonte: Autoral.

2.3 Comunicação Industrial

2.3.1 Protocolos de Comunicação em Automação

Para que dois dispositivos possam se comunicar, é necessário que existam regras que determinem os termos dessa comunicação, permitindo a identificação do significado de cada *bit* enviado e recebido durante a troca de dados. Para isso, utilizam-se os protocolos de comunicação, que são conjuntos de regras responsáveis por definir os parâmetros de transmissão, como a formatação e codificação dos dados (sintaxe), as informações para identificação de erros (semântica) e a velocidade da troca de dados (sincronização) (GRIGOLETTO, 2012).

No contexto da automação, com o surgimento do conceito de *Internet das Coisas (IoT)*, os protocolos de comunicação passaram a desempenhar um papel ainda mais fundamental. Tornou-se necessária a padronização desses protocolos em nível global, de forma a garantir a interoperabilidade entre dispositivos, ou seja, assegurar que equipamentos de diferentes fabricantes possam se comunicar entre si. Essa capacidade de integração é o que possibilita que grandes linhas de produção mantenham suas operações de forma precisa, contínua e sincronizada (TELLES, 2024).

Os protocolos utilizados em ambientes industriais devem apresentar alta confiabilidade e segurança na transmissão de dados, visto que qualquer falha na comunicação pode gerar prejuízos significativos. Com base nisso, conforme destaca Telles (2024), os protocolos industriais evoluíram e se adaptaram para acompanhar o avanço tecnológico dos dispositivos, garantindo comunicações eficazes e compatíveis com os mais diversos ambientes fabris. Entre os protocolos

mais utilizados na indústria, destacam-se: *DeviceNet*, *Profibus*, *Modbus*, *Profinet*, *Ethernet/IP* e *OPC UA*.

O *DeviceNet* é empregado em comunicações ponto a ponto e utiliza um cabo de par trançado, que além de reduzir interferências eletromagnéticas, também fornece alimentação elétrica ao dispositivo, dispensando o uso de fonte externa. Seu desenvolvimento baseou-se no protocolo *CAN* (*Controller Area Network*), o que, segundo Granermann (2020), garante uma comunicação confiável, rápida e robusta.

O *Profibus* é um protocolo amplamente utilizado por sua robustez e suporte à comunicação em tempo real, permitindo programação, configuração e atualização dos dispositivos sem a necessidade de interromper o funcionamento da rede (DEMÉTRIO; JUNIOR, 2025). Além disso, dispõe de mecanismos que facilitam a detecção de falhas e o diagnóstico de dispositivos, tornando a manutenção mais ágil e eficiente. A evolução desse protocolo resultou no *ProfiNet*, que utiliza a *Ethernet* padrão (IEEE 802.3) como base de infraestrutura, possibilitando melhor integração com as redes corporativas e maior velocidade na transmissão de dados.

Outro protocolo baseado em *Ethernet* é o *Ethernet/IP*, que se destaca pela capacidade de transmissão em tempo real entre dispositivos conectados a uma rede. Além disso, oferece recursos de segurança, como priorização de pacotes para evitar perdas de dados e criptografia, assegurando a integridade das informações transmitidas (DEMÉTRIO; JUNIOR, 2025).

O *OPC UA* (*Open Platform Communications Unified Architecture*) é reconhecido por sua flexibilidade, sendo projetado para operar em diferentes níveis de comunicação (DEMÉTRIO; JUNIOR, 2025). Oferece recursos avançados de segurança, como criptografia, autenticação de usuário e verificação de integridade dos dados. Também suporta arquiteturas *client/server* e *peer-to-peer*, o que o torna adequado para aplicações que exigem soluções personalizadas.

Por fim, o *Modbus* é um dos protocolos mais tradicionais da indústria, projetado para estabelecer comunicação entre controladores lógicos programáveis (CLPs) e resistir a ruídos e interferências (GRIGOLETTO, 2012). Conquistou ampla adoção devido à sua simplicidade, robustez e baixo custo, além da facilidade de instalação e manutenção. O protocolo atua na camada de aplicação do modelo OSI e baseia-se na arquitetura *client/server*, na qual um dispositivo realiza as requisições de dados e os demais apenas respondem.

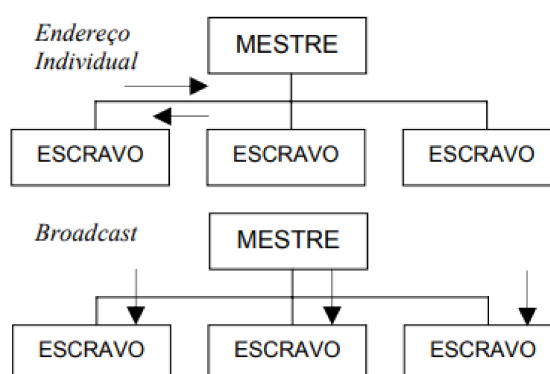
2.3.2 Protocolo Modbus

O protocolo Modbus foi desenvolvido no final da década de 1970 pela fabricante de equipamentos Modicon Industrial Automation Systems para uso em sistemas de automação industrial. Em 2004, tornou-se livre de taxas de licenciamento quando a Schneider Electric (controladora da Modicon) transferiu seus direitos para a *Modbus Organization*. Esse fator, aliado à facilidade de adaptação a diversos meios físicos de comunicação, contribuiu significativamente para a popularização do protocolo (CORDEIRO, 2019).

O Modbus não define um meio físico padrão, podendo ser implementado em diferentes interfaces, como RS-232 (*Recommended Standard 232*), RS-485 (*Recommended Standard 485*) e Ethernet TCP/IP (no caso do Modbus TCP). A conexão via RS-232 é utilizada para transmissões ponto a ponto, com velocidade limitada a aproximadamente 115 kbps e alcance de até 30 metros. Já o padrão RS-485 é amplamente utilizado em ambientes industriais, permitindo taxas de até 12 Mbps (podendo chegar a 50 Mbps em alguns casos) e distâncias de até 1200 metros, com até 32 dispositivos na mesma rede. O padrão Ethernet, por sua vez, apresenta maior eficiência em termos de velocidade e alcance, podendo atingir taxas de 100 Mbps a 10 Gbps, dependendo do tipo de cabo empregado (CORDEIRO, 2019).

O modelo de comunicação do Modbus baseia-se na arquitetura *client/server* (Fig. 11), na qual um único dispositivo (*client*) inicia a comunicação, enquanto os demais (*servers*) aguardam passivamente pelas requisições. A troca de mensagens pode ocorrer de forma individualizada (*query*) ou em *broadcast*, quando os dados são enviados simultaneamente a todos os dispositivos da rede (NASCIMENTO; LUCENA, 2003).

Figura 11 – Modelo *client/server*.



Fonte: Nascimento e Lucena (2003).

Conforme descrito por Nascimento e Lucena (2003), o protocolo Modbus pode operar em dois modos de transmissão: ASCII (*American Standard Code for Information Interchange*) e RTU (*Remote Terminal Unit*). No modo ASCII, o *framing* (estrutura de dados) é formado por caracteres ASCII, enquanto no modo RTU é composto por bytes binários. Em ambos os modos, cada palavra de dados (caractere ou byte) do *framing* é composta sempre por uma sequência de 1 *start bit*, *data bits* (7 para ASCII e 8 para RTU), *bit* de paridade (par, ímpar ou sem paridade) e *stop bit* (1 caso haja paridade e 2 caso não haja paridade) (Fig. 12). O modo ASCII utiliza sempre 10 bits por caractere, enquanto o RTU utiliza 11 bits por byte, independentemente da configuração.

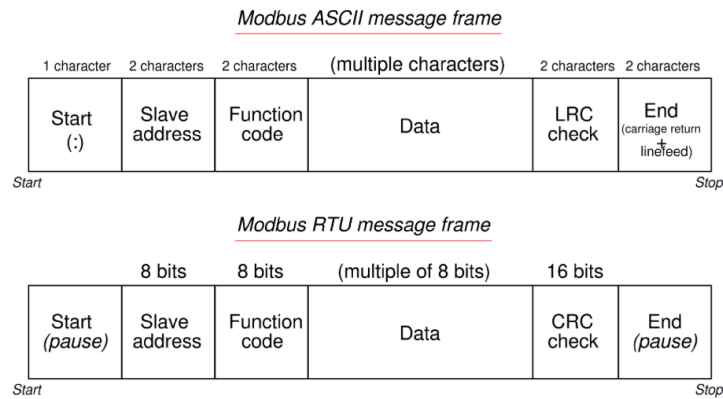
Figura 12 – Framing de dados nos modos ASCII e RTU.



Fonte: Adaptado de ARTERY Technology (2024).

Além do *framing* de dados, que define a estrutura de cada caractere ou byte transmitido, existe também o *framing* da mensagem, que descreve os campos que compõem a comunicação completa. No modo ASCII, o início da mensagem é identificado pelo caractere “:”, enquanto no modo RTU essa detecção ocorre por meio de uma pausa mínima equivalente a 3,5 vezes o tempo de uma palavra de dados (SOUZA, 2010). Após a identificação do início, seguem-se os campos de endereço do server, código da função solicitada, dados da mensagem e, por fim, os campos de verificação de erro (LRC no ASCII e CRC no RTU) (Fig. 13).

Figura 13 – Framing da mensagem nos modos ASCII e RTU.



Fonte: InstrumentationTools Editorial Staff (2022).

No protocolo Modbus, os dados armazenados são classificados em valores discretos e analógicos, que por sua vez se dividem entre dados somente de leitura (entradas) e de leitura e escrita (saídas) (Fig. 14). Os valores discretos são endereçados de 00001–09999 (saídas, *coils*) e de 10001–19999 (entradas, *discrete inputs*). Já os valores analógicos utilizam os intervalos de

30001–39999 (entradas analógicas, *input registers*) e 40001–49999 (saídas analógicas, *holding registers*) (THOMAS, 2008).

Figura 14 – Endereçamento de dados no Modbus.

ENDEREÇO	TIPO DE REGISTRO
00001 – 9999	Coil
10001 – 19999	Discrete Input
30001 – 39999	Input Register
40001 – 49999	Holding Register

Fonte: Adaptado de Thomas (2008).

Para a interação com esses dados, é necessário informar no campo de função o código correspondente à operação desejada. Cada código representa uma ação específica, como leitura ou escrita de registradores, conforme apresentado na Figura 15.

Figura 15 – Códigos das funções.

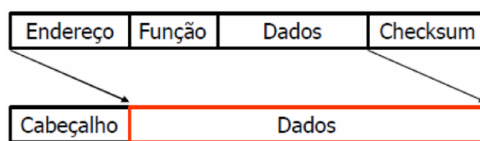
CÓDIGO	FUNÇÃO
1	Read Coils
2	Read Contacts
5	Write a single coil
15	Write multiple coils
3	Read holding registers
4	Read inputs registers
6	Write single register
16	Write multiple registers
22	Mask write register
23	Read/write multiple registers

Fonte: Adaptado de Thomas (2008).

Existe ainda a versão Modbus TCP/IP, que permite a troca de dados em redes Ethernet, oferecendo maior flexibilidade de comunicação entre os dispositivos conectados, sem uma diferenciação rígida entre *clients* e *servers*. Os *gateways* estão disponíveis para o Modbus com o modo RTU (SOUZA, 2010) e a comunicação ocorre pela porta padrão 502.

Nessa versão, a mensagem transmitida mantém a mesma estrutura do Modbus tradicional, porém é encapsulada em um pacote TCP/IP, aproveitando a infraestrutura das redes Ethernet, como LAN e WAN (Fig. 16). Esse encapsulamento introduz algumas diferenças na forma de endereçamento e na verificação de erros. Conforme Souza (2010), a identificação dos dispositivos não é feita por endereços Modbus convencionais, mas por identificadores únicos, utilizados na comunicação por meio de *gateways* e *bridges*, que possibilitam integrar diversos dispositivos sob um mesmo endereço IP. Já a verificação de erros é realizada pelos próprios recursos disponibilizados pelos protocolos Ethernet e TCP/IP.

Figura 16 – Encapsulamento da mensagem.



Fonte: Souza (2010).

Entre as principais vantagens do Modbus TCP/IP estão a alta velocidade de transmissão de dados, que pode alcançar até 10 Gbps (CORDEIRO, 2019), e a flexibilidade de integração. No entanto, como se trata de uma comunicação não determinística devido ao encapsulamento das mensagens, não é possível garantir tempos precisos de resposta (SOUZA, 2010).

2.3.3 WireShark

O Wireshark é um software amplamente utilizado para análise de protocolos de rede, que disponibiliza ferramentas para a captura e inspeção do tráfego de dados que ocorre em uma rede de comunicação. Por meio desse recurso, é possível realizar uma análise em nível de pacote, permitindo a visualização detalhada das mensagens transmitidas entre os dispositivos durante uma conexão, incluindo informações como endereço de origem e destino, tipo de protocolo utilizado e conteúdo dos pacotes (Wireshark Foundation, 2025).

Além disso, o Wireshark possibilita a filtragem e organização dos dados capturados de acordo com critérios específicos, como protocolos, portas ou endereços IP, facilitando a identificação de padrões de comunicação e eventuais falhas na transmissão (Wireshark Foundation, 2025). No contexto deste projeto, a ferramenta foi utilizada como apoio na análise da comunicação via protocolo Modbus TCP, auxiliando na verificação da latência, estabilidade da conexão e correta troca de dados entre o cliente e o servidor.

2.4 Simulação e Realidade Virtual

2.4.1 Simulação de Processos Industriais

De acordo com a definição de Bandyopadhyay e Bhattacharya (2014), a simulação pode ser entendida como a reprodução de um sistema por meio de um protótipo, servindo como base para a realização de experimentos e testes com o objetivo de identificar falhas e problemas inerentes ao sistema. Assim, qualquer programa que imite um sistema ou processo pode ser caracterizado como uma simulação.

No contexto industrial, as simulações podem ser classificadas em dois tipos principais: a evento discreto e a evento contínuo. A simulação a evento discreto considera sistemas compostos por variáveis que mudam em instantes específicos de tempo, enquanto a simulação a evento contínuo baseia-se em sistemas formados por variáveis que variam continuamente ao longo do

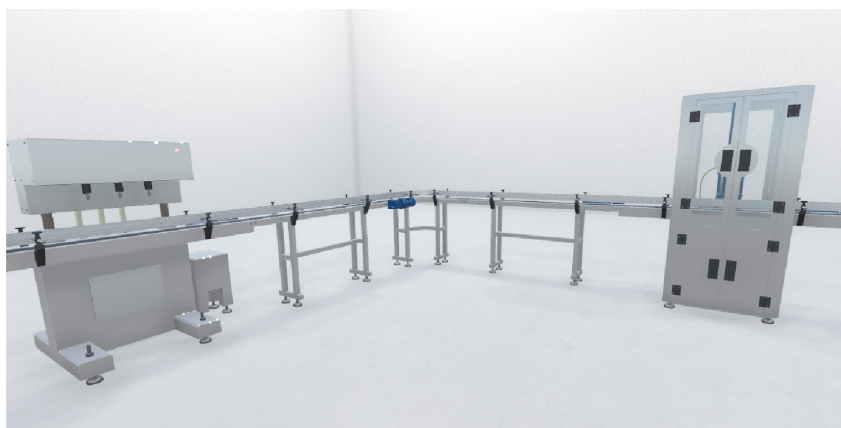
tempo (BANDYOPADHYAY; BHATTACHARYA, 2014). Há ainda os sistemas híbridos, que combinam características de ambos os tipos, apresentando variáveis discretas e contínuas em um mesmo processo.

No desenvolvimento de sistemas de automação, a aplicação de simulações mostra-se altamente promissora, pois oferece diversos benefícios (SOUZA, 2024), como a redução de custos, o aprimoramento da qualidade dos produtos e a otimização dos processos produtivos. Isso se deve ao fato de que a simulação possibilita identificar pontos de falha, gargalos e necessidades de manutenção preventiva, contribuindo para o desenvolvimento de sistemas de automação mais eficientes e confiáveis.

2.4.2 Realidade Virtual na Automação

A realidade virtual pode ser definida, de acordo com Kirner e Siscouto (2007), como uma interface avançada de um programa de computador, na qual o usuário pode visualizar, movimentar-se e interagir com um ambiente composto por modelos tridimensionais de objetos computacionais, em tempo real (Fig. 17). Há também o conceito de realidade aumentada que, ao contrário do que se imagina, apresenta diferenças significativas em relação à realidade virtual. A realidade aumentada (Fig. 18) consiste na sobreposição do ambiente real com objetos virtuais, mesclando o mundo físico e o digital, de modo a causar a impressão de que ambos coexistem no mesmo espaço (KIRNER; SISCOUTO, 2007).

Figura 17 – Ambiente em realidade virtual.



Fonte: Autoral.

Na indústria, a realidade virtual possui diversas aplicações. Uma delas é o treinamento de operadores (FUCHTER; SCHLICHTING, 2018), permitindo a simulação de ambientes de risco ou em altura, de forma a criar um ambiente seguro e controlado, no qual o aprendizado ocorre de maneira didática e sem grandes consequências em casos de erro. Também pode ser utilizada na supervisão de plantas industriais (BUCIOLI et al., 2006), possibilitando a visualização de dados do processo em tempo real e o controle remoto de forma interativa.

Figura 18 – Objetos virtuais sobrepondo a realidade.



Fonte: Kirner e Siscouto (2007).

Outra aplicação relevante da realidade virtual, em conjunto com a simulação de processos, é a validação de projetos de automação (FREITAS; FONSECA; MARQUES, 2007). Essa tecnologia auxilia na identificação de pontos de melhoria e na análise geral da planta, ao permitir a visualização tridimensional e o acompanhamento em tempo real dos dados do processo, promovendo o desenvolvimento e validação eficazes de sistemas de automação aplicados a ambientes simulados (SANTOS; CARVALHO, 2016).

2.4.3 Manufatura Virtual e *Digital Twins*

A abordagem que faz uso de recursos computacionais, como realidade virtual, modelagem tridimensional e simulação, para representar de forma fiel os sistemas de manufatura e seus comportamentos físicos e lógicos, define o conceito de manufatura virtual (IWATA et al., 1995). Essa abordagem possibilita a análise de diversos cenários dentro de uma cadeia produtiva, em múltiplos níveis de especificação, auxiliando nas tomadas de decisão e na otimização de projetos.

A partir da manufatura virtual, surge o conceito de *Digital Twin* (Gêmeo Digital), que se refere a modelos altamente realistas de processos produtivos, capazes de representar não apenas o comportamento, mas também o estado em tempo real dos sistemas, permitindo o monitoramento preciso dos processos (JÚNIOR, 2023). A aplicação dos *Digital Twins* (DT) em conjunto com tecnologias como *Internet of Things* (IoT) e *Big Data Analytics* possibilita o fluxo contínuo de grandes volumes de dados entre as diversas camadas e estruturas do processo até o *software* responsável pela simulação, gerenciamento e análise.

Os conceitos de manufatura virtual e *Digital Twin* constituem alguns dos pilares da Indústria 4.0. Em um cenário de alta competitividade, no qual as empresas buscam constantemente otimizar seus processos, essas tecnologias surgem como soluções avançadas, permitindo redução de custos por meio da previsão de falhas e manutenções, o desenvolvimento de sistemas mais seguros e a tomada de decisões mais assertivas, impulsionando novos modelos de negócio (DITZEL et al., 2020).

2.5 Modelagem 3D de Ambientes Industriais

2.5.1 Fundamentos de Modelagem 3D

A modelagem 3D consiste na criação de representações tridimensionais de objetos por meio da sua manipulação em um espaço virtual, conforme descreve Cardoso (s.d.). A modelagem pode ser dividida em dois tipos: modelagem por polígonos e modelagem por superfícies. A primeira utiliza formas primitivas, como cubos e esferas, que são combinadas para formar estruturas mais complexas, sendo amplamente empregada em jogos e animações. Já a modelagem por superfícies utiliza formas mais suaves, arredondadas e precisas, sendo comumente aplicada no design industrial (CARDOSO, s.d.). Além disso, alguns conceitos servem de base para a modelagem tridimensional, como geometria, texturização, escala, iluminação e renderização.

A geometria é uma das principais áreas envolvidas na modelagem, uma vez que diversos conceitos dessa disciplina se fazem presentes na construção de modelos tridimensionais, como figuras geométricas, polígonos, áreas, dimensões, profundidade, superfícies e arestas (SAÚDE, 2019). Assim, é fundamental possuir uma base sólida de conhecimentos geométricos para viabilizar a criação de modelos tridimensionais coerentes. Esse conhecimento também contribui para o desenvolvimento da noção espacial do objeto projetado, facilitando o processo de modelagem.

Outro conceito essencial é o de escala e proporção, pois um modelo que não respeita as proporções reais do objeto que representa perde a fidelidade e compromete sua funcionalidade e qualidade final. Esse conceito está fortemente relacionado à álgebra linear, que permite, por exemplo, a transformação de escala em três dimensões, aplicável à variação de um ponto (X_1, Y_1, Z_1) em uma escala (S_x, S_y, S_z) por meio de transformações matriciais (SAÚDE, 2019), como demonstrado em 2.1.

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \quad (2.1)$$

A textura consiste na aplicação de imagens ou padrões bidimensionais sobre superfícies tridimensionais, conferindo detalhamento e identidade visual ao modelo, sendo responsável por permitir a identificação do material representado (metal, madeira, vidro, etc.) (CARDOSO, s.d.). Associado a esse processo está o UV mapping, que consiste no mapeamento da textura sobre a malha do objeto, garantindo o encaixe correto da textura em cada superfície do modelo (CARDOSO, s.d.).

De acordo com Cardoso (s.d.) iluminação é o que proporciona profundidade à geometria projetada, permitindo a identificação do objeto como sendo tridimensional. Os contrastes de luz e sombra são o que possibilitam o reconhecimento das superfícies de um objeto em um sistema

de três coordenadas, uma vez que estes estão representados em uma superfície plana que no caso é a tela de um computador. Sem as luzes presentes no ambiente ou sem as arestas que contornam a superfície de um modelo tridimensional, não seria possível identificá-lo como tal, como demonstra a figura 19.

Figura 19 – Diferença entre objeto tridimensional sem iluminação e com iluminação.



Fonte: Adaptado de Abad (2016).

A renderização representa a etapa final da modelagem, na qual os objetos criados são processados e transformados em imagens realistas. Nesse estágio, ocorre o cálculo da interação da luz com o ambiente virtual, considerando as configurações definidas nas etapas anteriores (como textura, iluminação e sombras) (CARDOSO, s.d.). Esse processo utiliza algoritmos e técnicas de processamento gráfico para simular o comportamento da luz no ambiente digital.

Na criação de modelos destinados a simulações industriais, utiliza-se como referência desenhos técnicos (D'ADDARIO, 2025), que apresentam as formas e dimensões de cada componente de uma peça ou equipamento por meio de vistas superiores, laterais, inferiores e seções em corte, com o objetivo de evidenciar detalhes internos ou saliências não visíveis em vistas convencionais. Além disso, imagens reais da peça também são utilizadas para complementar o processo de modelagem, uma vez que oferecem uma riqueza visual maior em relação aos desenhos técnicos, permitindo maior precisão e realismo na projeção tridimensional.

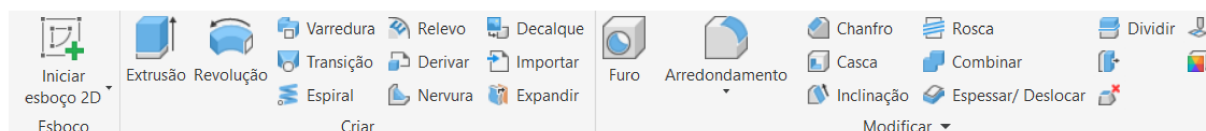
2.5.2 Uso do Autodesk Inventor

Existem diversas ferramentas utilizadas na projeção de modelos tridimensionais voltados à indústria, em geral baseadas em *softwares* de desenho assistido por computador (CAD). Um deles é a solução desenvolvida pela *Autodesk*, o Autodesk Inventor. O *software* oferece uma ampla variedade de recursos que permitem o desenvolvimento de projetos de máquinas, chapas metálicas, cabos, sistemas hidráulicos, entre outros (Autodesk Inc., 2025).

O *software* dispõe de ferramentas que facilitam a criação de peças e máquinas com precisão, como o recurso de esboço, que permite a elaboração de desenhos bidimensionais com dimensões especificadas, servindo de base para a geração de modelos tridimensionais por meio da extrusão. A extrusão consiste em gerar um volume 3D a partir do deslocamento da superfície de um esboço em uma direção perpendicular ao plano do desenho. Além disso, é possível criar peças por meio de revoluções, varreduras, chanfros, arredondamentos, combinações e interseções

de corpos (Fig. 20). Também é possível gerar automaticamente o desenho técnico das peças, contendo vistas ortogonais, cortes, cotas e demais informações do projeto.

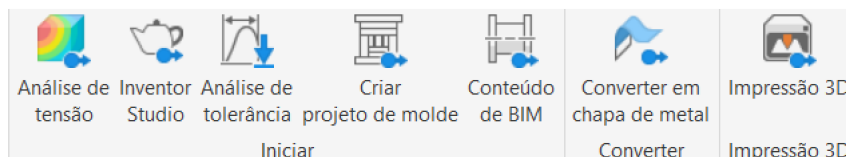
Figura 20 – Aba de ferramentas com recursos de projeção.



Fonte: Autoral.

O Autodesk Inventor permite a exportação de modelos em diversos formatos, como DWG, STEP, OBJ e PDF, o que possibilita sua aplicação em diferentes contextos, desde o desenvolvimento de chapas metálicas até peças para impressoras 3D. Além disso, dispõe de ferramentas de análise estrutural, como análise de tensões e tolerâncias (Fig. 21). Os modelos gerados podem ainda ser exportados para outros motores de simulação, como o Unity. Entretanto, é importante atentar-se à definição correta da escala durante a exportação, uma vez que o *software* permite trabalhar com unidades que variam de milímetros a quilômetros. Um erro nessa configuração pode comprometer todo o projeto, especialmente em aplicações que exigem alta precisão dimensional, como o projeto de chapas metálicas.

Figura 21 – Aba de ferramentas com recursos de análise.



Fonte: Autoral.

2.6 Simulação Interativa em Unity

2.6.1 Unity como Ferramenta de Simulação

A plataforma Unity é uma *game engine*, ou seja, uma plataforma voltada ao desenvolvimento de jogos, simulações, animações e outros conteúdos interativos, oferecendo recursos que simplificam a criação de ambientes virtuais (Unity Technologies, 2025b). Trata-se de uma ferramenta multiplataforma, permitindo o desenvolvimento de aplicações para diferentes sistemas operacionais e dispositivos, incluindo experiências imersivas em realidade virtual e aumentada. Além disso, disponibiliza ferramentas que abrangem desde a criação até a fase de monetização dos projetos desenvolvidos. Sua aplicação não se restringe à área de entretenimento, sendo amplamente utilizada em contextos industriais, arquitetônicos, de design de produtos e em pesquisas voltadas à simulação de sistemas.

A estrutura básica de um projeto Unity é composta pelos seguintes elementos: *GameObjects*, *Components*, *Scripts*, Física e Iluminação (Unity Technologies, 2025a). Os *GameObjects*

representam todos os elementos que podem existir em uma cena, funcionando como blocos de construção que atuam como contêineres para os Components. Os *Components* atribuem propriedades e comportamentos aos objetos, determinando sua aparência, posição, tamanho e demais características. Os *Scripts* são arquivos desenvolvidos em linguagem C#, responsáveis por implementar a lógica da aplicação, permitindo a personalização e a extensão das funcionalidades da plataforma. A Física é o que possibilita a simulação do comportamento de objetos em situações como quedas e colisões, assegurando interações realistas de acordo com os parâmetros definidos. Por fim, a Iluminação permite a simulação do comportamento da luz, criando sombras, brilhos e efeitos visuais que conferem realismo à cena.

A interface principal do Unity dispõe de janelas que permitem o acesso direto à maior parte dos recursos da plataforma (Fig. 22). A área central, denominada *Scene View*, possibilita a visualização e edição completa da cena. A *Game View* apresenta a cena sob a perspectiva da câmera principal, simulando a visão do jogador. A *Project Window* reúne todos os arquivos relacionados ao projeto, como modelos, scripts, animações e cenas, enquanto a *Hierarchy Window* organiza e exibe todos os GameObjects presentes na cena. Já a *Inspector Window* permite configurar os Components associados a cada objeto (Unity Technologies, 2025a).

Figura 22 – Interface principal do Unity.



Fonte: Unity Technologies (2025a).

Entre os Components disponíveis, destacam-se o *Collider* e o *Rigidbody*, fundamentais para a simulação física de objetos. O *Collider* define o espaço físico ocupado por um objeto, o que nem sempre coincide com seu formato visual, enquanto o *Rigidbody* permite que o objeto seja considerado nos cálculos da física (sem ele, o objeto permanece estático) (Unity Technologies,

2025a). Esses dois elementos são essenciais em projetos de simulação, pois possibilitam a reprodução fiel do comportamento físico de máquinas e peças, aproximando o ambiente virtual da realidade industrial e dos conceitos de *Digital Twins*.

No campo das simulações industriais e do desenvolvimento de *Digital Twins*, o Unity destaca-se não apenas pela capacidade de representar sistemas e equipamentos de forma realista, mas também pela facilidade de integração com outros softwares e sistemas (SOUZA, 2021), por meio de APIs e bibliotecas disponíveis na *Unity Asset Store* e em diversos sites da internet. Essa interoperabilidade torna a ferramenta uma opção robusta e versátil para aplicações em manufatura digital e automação.

2.6.2 Integração com Sistemas de Automação

Na indústria, o Unity se destaca por facilitar a integração com ferramentas e equipamentos utilizados no desenvolvimento de sistemas automatizados, como os Controladores Lógicos Programáveis (CLPs). Além disso, permite a comunicação por meio de protocolos industriais, como o protocolo Modbus, utilizando a biblioteca *EasyModbus* (EasyModbusTCP, 2025).

A biblioteca *EasyModbus* é um projeto open source que implementa o protocolo de comunicação Modbus nos modos RTU e TCP. A comunicação Modbus é baseada em uma arquitetura do tipo *client/server* e, por isso, a biblioteca possui duas classes principais: *ModbusClient* e *ModbusServer*. A classe *ModbusServer* é responsável por definir os parâmetros da conexão (como endereço IP e porta) e pelo armazenamento dos dados, enquanto a classe *ModbusClient* realiza o envio de requisições, bem como a leitura e escrita de dados de entrada e saída (SOARES, 2021).

A implementação da biblioteca *EasyModbus* via *script* na plataforma Unity possibilita a integração da ferramenta com Controladores Lógicos Programáveis reais, que atuam em tempo real em um processo, ou ainda com CLPs virtuais, permitindo o desenvolvimento e teste de projetos em um ambiente controlado. É importante ressaltar que, ao utilizar a biblioteca, a definição de qual dispositivo será o *client* e qual será o *server* depende do objetivo da aplicação.

Além disso, como a plataforma Unity possui suporte a aplicações em realidade virtual, seu uso permite a imersão em ambientes industriais e projetos voltados à automação, possibilitando a interação em tempo real com a planta simulada e uma análise visual mais detalhada do processo e dos resultados obtidos.

2.6.3 Meta Quest Developer Hub

Para a análise de desempenho de aplicações em realidade virtual, existem ferramentas específicas que permitem o monitoramento de métricas como FPS (*Frames Per Second*), uso de CPU (*Central Processing Unit*) e uso de GPU (*Graphics Processing Unit*). No ecossistema Meta Quest, o Meta Quest Developer Hub disponibiliza recursos para inspeção e diagnóstico

de aplicações executadas diretamente no dispositivo, auxiliando na identificação de gargalos de desempenho e na validação da experiência do usuário em ambientes imersivos (Meta Platforms, Inc., 2025).

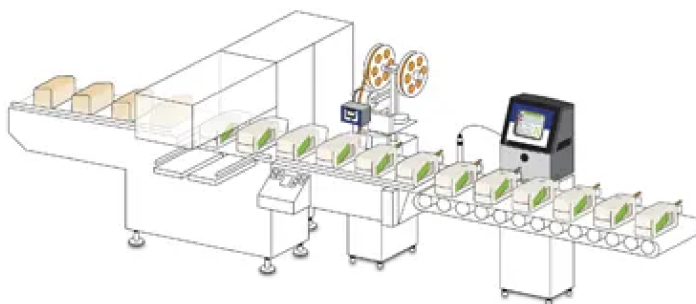
2.7 Processo de Envase

2.7.1 Conceito e Etapas de um Sistema de Envase

Um sistema de envase consiste na última etapa da produção de um líquido, compreendendo o conjunto de equipamentos e sistemas responsáveis pelo enchimento, rotulagem, vedação, secagem, empacotamento e paletização do produto. Cada etapa da linha de envase varia conforme o tipo de líquido a ser envasado e o recipiente que armazenará o produto, como latas, potes ou garrafas de vidro ou PET (CINTRA, 2023). As três principais etapas que compõem uma linha de envase são o enchimento, a vedação e a rotulagem.

Na etapa de rotulagem (Fig. 23), ocorrem os processos relacionados à aplicação dos rótulos nos recipientes. Os adesivos ficam armazenados em uma bobina na máquina que, ao ser acionada, libera-os para o corte nas dimensões finais. Em seguida, os rótulos passam por uma câmara a vácuo responsável por segurá-los e aplicar uma fina camada de cola (CINTRA, 2023). Por fim, os rótulos são fixados na superfície dos recipientes, que seguem então para a etapa de enchimento.

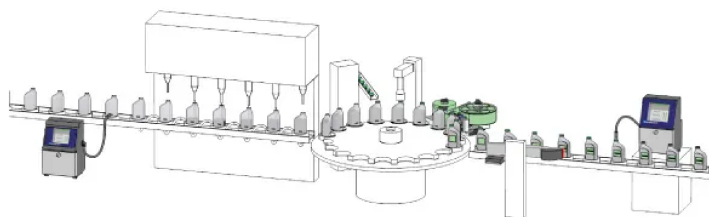
Figura 23 – Processo de rotulagem.



Fonte: Discher (2025).

Durante a etapa de enchimento (Fig. 24), o líquido produzido é transferido para os recipientes, mantendo o nível ideal previamente especificado, de modo a garantir a uniformidade da quantidade de produto. As máquinas de enchimento podem possuir um ou múltiplos bicos enchedores, sendo esta última configuração voltada à agilidade do processo, permitindo preencher um maior número de recipientes em menor tempo. Essa etapa exige rigor quanto às condições de higiene, pois é o ponto de contato direto com o produto final (Videojet Technologies Inc., 2025).

Figura 24 – Processo de enchimento e vedação.



Fonte: Videojet Technologies Inc. (2025).

Após o enchimento, os recipientes seguem para o abastecedor de tampas e, posteriormente, para a máquina de vedação (Fig. 24), onde são devidamente lacrados, assegurando a conservação das propriedades físicas e químicas do produto (International Labour Organization, 2025).

As linhas de envase são comumente apontadas como gargalos de produção, o que torna a automação desses processos essencial. A automação das etapas envolvidas em uma linha de envase garante um sistema mais eficiente e equilibrado, podendo aumentar significativamente a produtividade, inclusive em comparação a linhas semi-automáticas (PEREIRA; JÚNIOR, 2023). Além disso, o controle automatizado contribui para a fluidez do processo e redução de falhas mecânicas, uma das causas da baixa eficiência nas indústrias de envase (SANTOS, 2008).

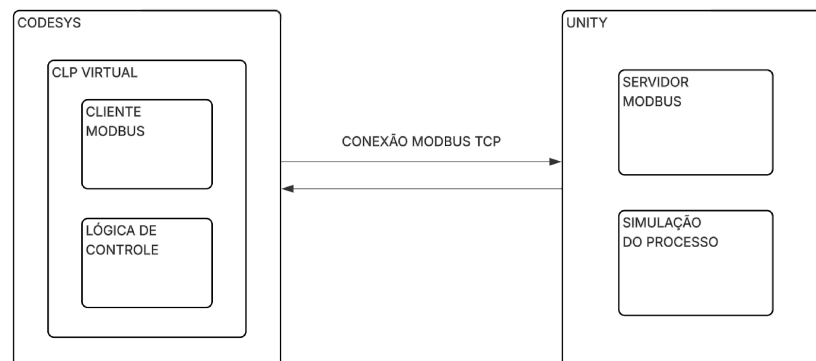
3 Metodologia

3.1 Visão Geral do Sistema

O sistema desenvolvido consiste na simulação de uma linha de envase de garrafas, controlada por meio de um CLP virtual que contém a lógica de controle implementada em linguagem Ladder. Essa lógica é responsável por processar os dados provenientes dos sensores da simulação e enviar os comandos necessários aos atuadores do sistema, sendo a troca de informações realizada por meio do protocolo industrial Modbus TCP.

O sistema é dividido em dois ambientes principais. O primeiro corresponde à simulação desenvolvida na *game engine* Unity, enquanto o segundo refere-se ao software CODESYS, responsável por emular o CLP virtual e permitir o desenvolvimento da lógica de controle em linguagem Ladder. A comunicação entre esses dois ambientes ocorre por meio do protocolo Modbus TCP, no qual o Unity atua como cliente e o CODESYS como servidor (Fig. 25).

Figura 25 – Estrutura de comunicação Modbus TCP entre cliente (CLP virtual) e servidor (Unity).



Fonte: Autoral.

Com o objetivo de organizar e facilitar o desenvolvimento do projeto, o trabalho foi dividido em cinco etapas principais: (I) modelagem tridimensional dos equipamentos, (II) desenvolvimento da simulação no Unity, (III) integração entre o software de simulação e o software de controle via protocolo Modbus TCP, (IV) desenvolvimento da lógica de controle da simulação em linguagem Ladder e (V) realização dos testes e validação do sistema.

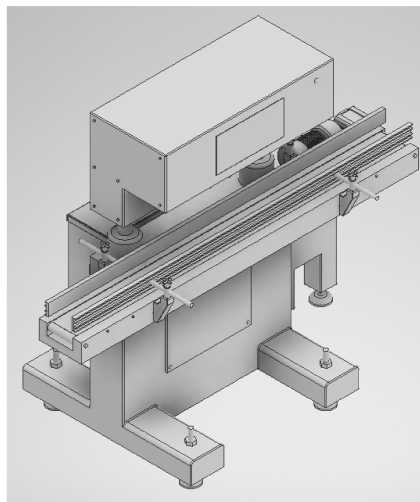
A aplicação foi concebida de forma a permitir a prática do desenvolvimento de lógicas de controle aplicadas a processos industriais, utilizando a linguagem Ladder em um ambiente de simulação. Para isso, o sistema apresenta um objetivo operacional a ser cumprido pelo usuário, que consiste no desenvolvimento do programa em linguagem Ladder responsável por controlar o processo de envase de nove garrafas de suco de laranja, por meio de um CLP virtual com comunicação via protocolo Modbus TCP.

3.2 Modelagem 3D no Autodesk Inventor

3.2.1 Criação dos Modelos

Todos os dispositivos visualmente necessários à simulação da linha de envase, tais como sensores, atuadores, esteiras e máquina envasadora (Fig. 26), bem como aqueles utilizados apenas para referência contextual, como os elementos do cenário, foram modelados por meio do software Autodesk Inventor. A ferramenta permitiu o desenvolvimento preciso dos modelos por meio de recursos como esboços, extrusões, arredondamentos, entre outros, possibilitando a representação detalhada de cada componente da simulação.

Figura 26 – Modelo tridimensional da máquina envasadora.



Fonte: Autoral.

Para garantir a funcionalidade dos dispositivos no ambiente de simulação, as peças identificadas como dinâmicas foram modeladas de forma independente na hierarquia do objeto principal. Essa abordagem possibilitou a manipulação individual dos componentes, bem como a implementação de lógicas de movimento, como rotação e translação, e de interações personalizadas para cada peça.

A modelagem tridimensional foi desenvolvida com base em imagens de referência disponíveis publicamente, utilizadas para inferir proporções e características visuais dos equipamentos. Dessa forma, os modelos desenvolvidos não representam cópias fiéis de equipamentos reais, mas sim representações visuais e funcionais adequadas ao contexto da simulação proposta.

3.2.2 Preparação para Exportação

Após a finalização da modelagem, visando facilitar a manipulação das peças no ambiente de simulação, cada componente foi renomeado de forma padronizada, permitindo uma identificação mais clara e organizada dos objetos. A aplicação de texturas não foi realizada no software

Autodesk Inventor, sendo essa etapa executada posteriormente no ambiente de edição do Unity, a fim de evitar possíveis conflitos de texturização entre os dois softwares.

Em seguida, foi realizada a verificação e padronização da escala dos dispositivos, garantindo que todos os modelos estivessem sob a mesma referência dimensional. Embora o ajuste de escala possa ser efetuado posteriormente no Unity, essa padronização prévia contribui para uma melhor visualização e organização inicial do ambiente de simulação.

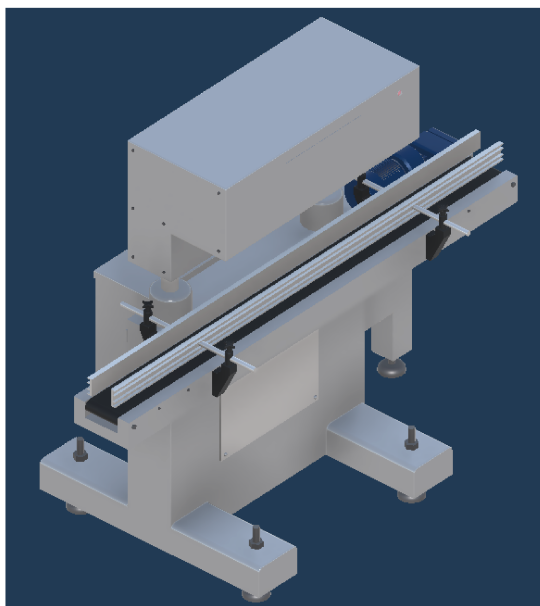
Por fim, definiu-se a extensão de exportação dos objetos, sendo escolhida a extensão OBJ por permitir a manipulação individual das peças no Unity. Essa escolha se justifica pelo fato de o Autodesk Inventor não disponibilizar, de forma nativa, a opção de exportação no formato FBX, que apresenta maior compatibilidade com a game engine utilizada.

3.3 Desenvolvimento da Simulação no Unity

3.3.1 Importação e Configuração dos Modelos

Ao iniciar o ambiente do Unity, o primeiro passo consistiu na importação dos modelos tridimensionais desenvolvidos. Em seguida, foram criados os materiais e texturas correspondentes a cada elemento visual da cena, os quais foram posteriormente atribuídos às malhas dos respectivos *GameObjects* (Fig. 27). Além dos aspectos visuais, também foram adicionados componentes físicos, como *Colliders* e *Rigidbody*s, possibilitando a aplicação de física, o controle de eventos de colisão e a interação entre os objetos da simulação.

Figura 27 – Exemplo de texturização dos modelos tridimensionais.



Fonte: Autoral.

Após a configuração individual dos modelos, realizou-se a composição do cenário (Fig. 28 e 29), organizando os elementos de modo a representar uma linha de envase típica. No entanto, considerou-se um espaço físico reduzido, por se tratar de uma simulação limitada do processo, contemplando apenas três de suas etapas principais. Nessa fase, também foram delimitadas as áreas destinadas à locomoção dentro da cena.

Figura 28 – Vista frontal do cenário.



Fonte: Autoral.

Figura 29 – Vista lateral do cenário.



Fonte: Autoral.

Por fim, foram inseridos elementos de refinamento visual, como iluminação e efeitos gráficos, com o objetivo de representar o ambiente industrial de forma mais fiel à realidade. Nesta etapa, foi tomado um cuidado especial quanto à quantidade de fontes de luz utilizadas, uma vez que o uso excessivo pode comprometer o desempenho do sistema em termos de performance.

3.3.2 Criação de Scripts de Interação e Controle

Após a finalização da parte visual da simulação, foram desenvolvidos os scripts responsáveis pelas mecânicas de funcionamento de cada máquina, com o objetivo de reproduzir o comportamento físico e operacional dos sistemas industriais, especialmente no que se refere aos movimentos das peças e às ações executadas por cada equipamento, como envasar, selar e rotular. Para essa etapa, foram utilizados códigos desenvolvidos em C#, em conjunto com animações de movimento, de modo a replicar as dinâmicas de deslocamento e interação entre os elementos que compõem o processo. Além disso, as mecânicas foram implementadas para operar de acordo com os comandos enviados às máquinas, de modo que cada comando esteja associado a uma ação específica, conforme descrito na Tabela 1.

Tabela 1 – Comandos de Atuação das Máquinas do Processo

Equipamento	Comando	Descrição
Máquina de Envase	3	Retorno da máquina à posição inicial de operação
	11	Acompanhamento do deslocamento das garrafas durante o processo de envase
Máquina de Selagem	7	Acionamento do cilindro responsável pela selagem das garrafas
Máquina de Rotulagem	25	Acionamento do sistema de aplicação de rótulos

Fonte: Autoral.

O mesmo procedimento foi adotado para os dispositivos de leitura de dados, como sensores de presença, os quais foram implementados por meio da utilização de *Colliders* e do tratamento de eventos gerados a partir das interações com esses componentes, permitindo a reprodução do comportamento típico desses dispositivos em um ambiente industrial.

Visando facilitar o entendimento da dinâmica de implementação da lógica de controle via CLP por parte do usuário da aplicação, bem como permitir a validação do sistema de forma independente da comunicação Modbus, foram desenvolvidos dois contextos distintos de execução do processo. O primeiro representa o cenário ideal, no qual toda a lógica de controle é implementada diretamente por meio de scripts em C#, permitindo a visualização do funcionamento esperado da linha de envase. O segundo corresponde ao cenário denominado “real”, no qual ocorre efetivamente a comunicação via protocolo Modbus TCP, com a troca de

dados entre os *softwares* de simulação e controle, possibilitando a validação prática da lógica desenvolvida em linguagem Ladder. A comutação entre esses dois contextos é realizada por meio da ativação de uma variável booleana presente no script de gerenciamento da aplicação.

Para a administração dos eventos da cena como um todo, foi desenvolvido um script de gerenciamento denominado *GameManager*, responsável por controlar aspectos como os pontos de criação e remoção de *GameObjects*, o início e término das simulações, bem como a transição entre cenas. O *GameManager* foi implementado seguindo o padrão *Singleton*, garantindo a existência de apenas uma instância ativa em toda a aplicação. Além disso, o script foi configurado para persistir entre as cenas, permitindo o acesso contínuo às informações de controle. Em seu conteúdo, foi criada uma variável estática que possibilita o acesso global ao gerenciador, tornando viável a manipulação e visualização de variáveis globais, como a velocidade das esteiras, o estado de início do processo e o encerramento da simulação.

Na fase de polimento da aplicação, foram desenvolvidos scripts adicionais destinados à implementação de efeitos sonoros associados aos movimentos das máquinas, das esteiras, interação com botões da interface e trilha sonora. Além disso, foi utilizado o recurso de áudio espacializado (3D) disponibilizado pelo próprio componente de áudio do Unity, permitindo que os sons fossem percebidos de acordo com a posição do usuário no ambiente virtual, contribuindo para uma maior sensação de imersão e realismo na simulação.

3.3.3 Interface e Visualização do Processo

Para o desenvolvimento das interfaces gráficas, foram utilizados como referência visual, no processo de design, protótipos criados com o auxílio de ferramentas de IA generativa, os quais foram posteriormente adaptados para a implementação no sistema.

A interface principal foi desenvolvida em uma cena específica (Fig. 30), na qual o cenário foi composto pela aplicação de uma imagem panorâmica em 360° da linha de envase da cena principal, inserida no *Skybox* (componente responsável pelo plano de fundo que envolve toda a cena). Essa interface apresenta quatro botões principais que, da esquerda para a direita, permitem o acesso às seguintes funcionalidades: visualização dos detalhes da conexão Modbus, como endereço IP, porta de comunicação e dispositivos endereçados no protocolo; início da simulação em modo ideal; acesso às instruções de controle e aos objetivos da simulação; e, por fim, o botão central, responsável por iniciar a simulação controlada pela lógica Ladder implementada no CLP virtual. Os botões de início da simulação, tanto no modo ideal quanto no modo controlado, direcionam o usuário para a cena principal onde está localizada a linha de envase.

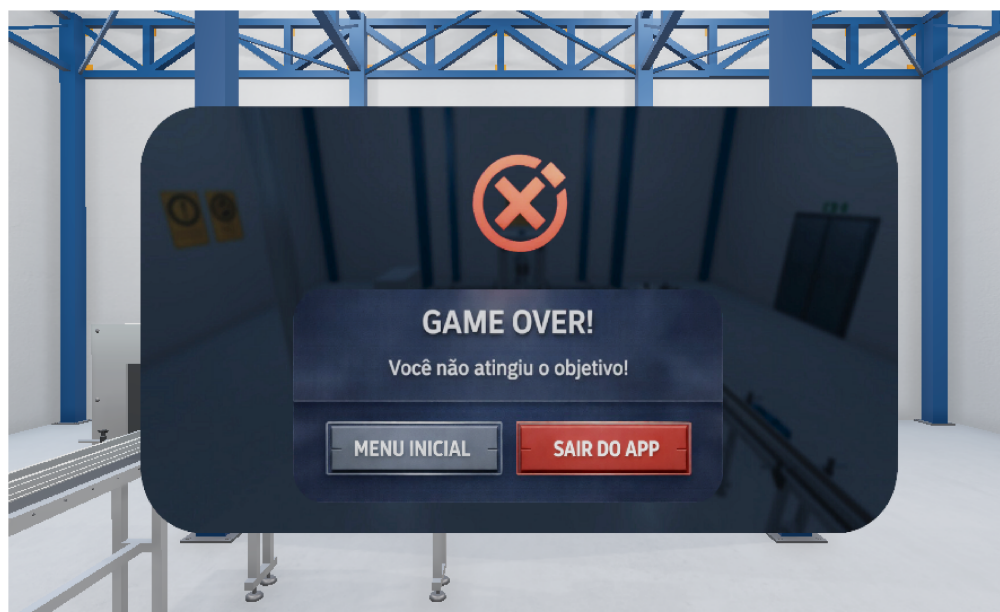
Figura 30 – Interface principal.



Fonte: Autoral.

Além da interface principal, foram desenvolvidas duas interfaces adicionais: a interface de objetivo cumprido e a interface de objetivo não cumprido. Ambas estão associadas à cena da linha de envase, porém são ativadas apenas em situações específicas durante a execução da simulação. A interface de objetivo não cumprido (Fig. 31) é exibida sempre que algum dos requisitos da simulação não é atendido, seja por falha em alguma das etapas do processo, causada por erros na implementação da lógica no CODESYS ou pela interrupção do sistema, como no caso da queda de uma garrafa. Essa interface contém botões que permitem retornar à interface principal ou encerrar a aplicação.

Figura 31 – Interface de não objetivo cumprido.



Fonte: Autoral.

Já a interface de objetivo cumprido (Fig. 32) é apresentada ao final da simulação, quando todos os requisitos estabelecidos são atendidos com sucesso, indicando a correta implementação da lógica de controle em linguagem Ladder no CLP virtual. Antes de sua exibição, é executada uma animação representando o empacotamento das garrafas, simbolizada por uma caixa que se desloca até o local onde a interface é apresentada. Essa interface contém, além dos mesmos botões de navegação da interface anterior, um texto de parabenização e a caixa animada exibindo os créditos do projeto em uma de suas faces.

Figura 32 – Interface de objetivo cumprido.



Fonte: Autoral.

3.3.4 Integração da Simulação com Realidade Virtual

Para viabilizar a execução da simulação em realidade virtual, foi adotado o SDK da Meta (*Oculus*), o qual fornece suporte para interação por meio de controladores, rastreamento de mãos e renderização otimizada, sendo amplamente utilizado no desenvolvimento de aplicações para *headsets* da linha *Meta Quest*. Essa abordagem permitiu a validação da aplicação utilizando o óculos de realidade virtual *Meta Quest 3S*, garantindo compatibilidade com os recursos nativos do dispositivo. Para isso, foi necessária a configuração do *XR Plug-in Management* no Unity, com a definição do *Oculus* como provedor de XR e a seleção do respectivo backend de execução.

Para a implementação dos componentes de interação em realidade virtual, foram utilizados os recursos disponibilizados pelo *Meta Building Blocks*. Esse conjunto de ferramentas possibilitou a criação do *GameObject* que representa o usuário no ambiente virtual, incluindo a câmera principal, responsável pela visualização da cena, e os controladores, que permitem a locomoção do usuário, a orientação da câmera e a interação com os elementos do ambiente. Além disso, as interfaces gráficas e os scripts responsáveis pela interação por meio dos controladores de realidade virtual também foram implementados com o auxílio dos componentes fornecidos pelo

Meta Building Blocks, garantindo uma integração consistente entre a simulação e os dispositivos de entrada em realidade virtual.

3.4 Integração via Protocolo Modbus TCP

3.4.1 Implementação do Servidor no Unity

O primeiro passo anterior à implementação do servidor Modbus no Unity consistiu na definição dos papéis de cada dispositivo envolvido na comunicação, de modo a estabelecer claramente o tipo de dado, o endereço e os comandos associados a cada elemento do sistema (Tab. 2). Essa definição prévia facilitou a etapa posterior de implementação das lógicas de transmissão e recepção de dados, bem como a organização dos scripts responsáveis por cada dispositivo da simulação.

Tabela 2 – Mapeamento de Variáveis Modbus do Sistema

Variável	Endereço	Tipo Modbus
Motor de Abastecimento de Garrafas	00001	Coil
Motor da Esteira Transportadora	00002	Coil
Botão de Partida	00003	Coil
Sensor de Início de Processo	10001	Discrete Input
Sensor de Posição da Máquina de Envase	10002	Discrete Input
Sensor de Posição da Máquina de Selagem	10003	Discrete Input
Sensor de Posição da Máquina de Rotulagem	10004	Discrete Input
Sensor de Contagem	10005	Discrete Input
Sensor de Fim de Curso	10006	Discrete Input
Máquina de Envase	40001	Holding Register
Máquina de Selagem	40002	Holding Register
Máquina de Rotulagem	40003	Holding Register

Fonte: Autoral.

A implementação do servidor no Unity foi realizada por meio de um *script* denominado *ModbusServer*, desenvolvido em C# com o auxílio da biblioteca *EasyModbus*. Nesse script, foi efetuada a configuração do servidor (Fig. 33), incluindo a atribuição de um endereço IP e de uma porta específica (1502), responsáveis por permitir o recebimento das requisições provenientes do cliente, representado pelo *software* CODESYS.

Figura 33 – Implementação do servidor via C#.

```
0 references
private void Start() {
    try {
        modbusServer.Port = 1502;
        modbusServer.LocalIPAddress = IPAddress.Any;
        modbusServer.Listen();
    }
    catch (System.Exception e) {
        Debug.LogError("Erro ao iniciar o servidor Modbus: " + e.Message);
    }
}
```

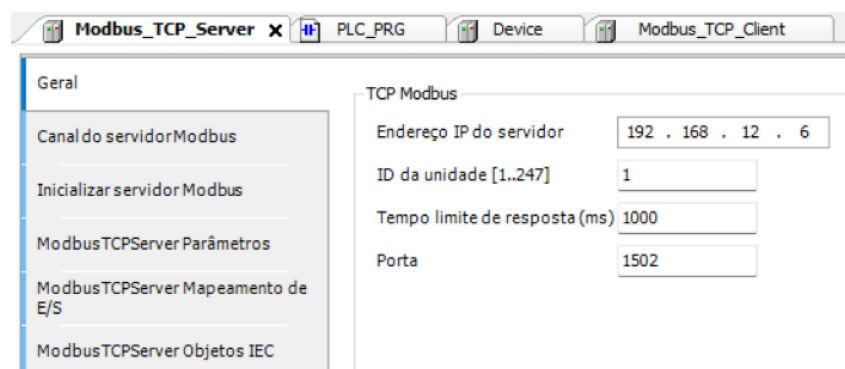
Fonte: Autoral.

Nos *scripts* de controle do processo, associados às máquinas e aos sensores da simulação, foram implementadas rotinas em C# responsáveis pelo envio e recebimento de dados de acordo com os registradores Modbus previamente definidos para cada dispositivo. Essas rotinas foram estruturadas de modo que cada evento físico da simulação (como o acionamento de um motor ou a detecção de presença por um sensor) estivesse diretamente associado a uma variável Modbus. Dessa forma, o comportamento dos modelos tridimensionais passou a ser influenciado diretamente pelas instruções enviadas pelo CLP virtual, estabelecendo um ciclo completo de controle, supervisão e resposta entre o sistema de controle e o ambiente simulado.

3.4.2 Configuração do Cliente no CODESYS

No *software* CODESYS, foi configurado o CLP virtual responsável pelo controle do processo, sendo selecionado o *runtime CODESYS Control Win V3 x64*, de acordo com as configurações do computador utilizado para a execução das simulações. Após a definição do dispositivo virtual, realizou-se a configuração do cliente Modbus no ambiente do CODESYS. Para isso, o dispositivo cliente foi criado e, em sua interface de configuração, foram definidos o endereço IP e a porta do servidor Modbus (Fig. 34), possibilitando o estabelecimento da comunicação entre o CLP virtual e o ambiente de simulação desenvolvido no Unity.

Figura 34 – Definição de endereço IP e porta do servidor no CODESYS.



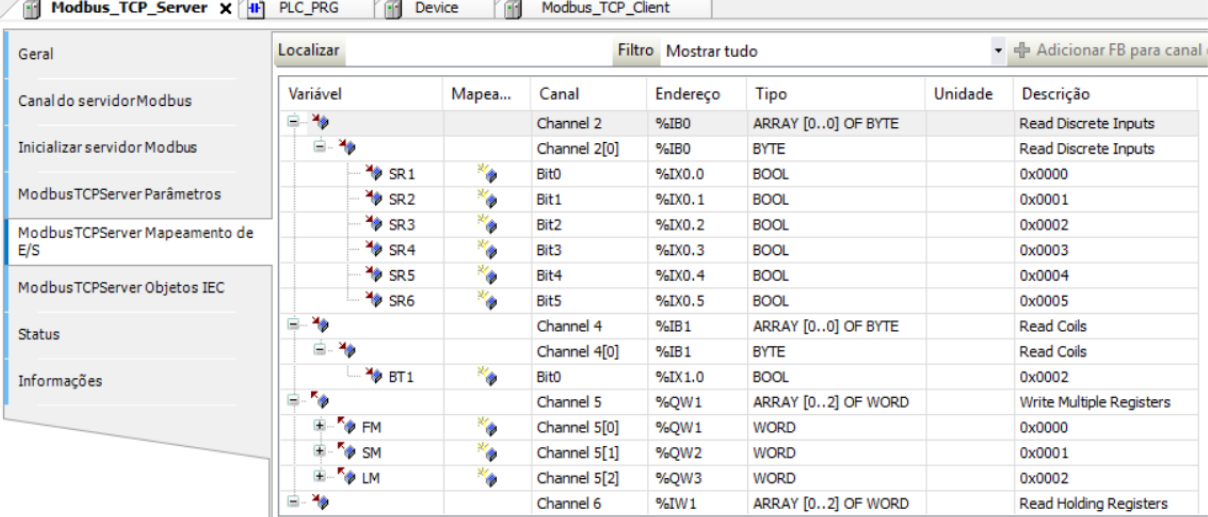
Fonte: Autoral.

3.5 Desenvolvimento do Controle no CODESYS

3.5.1 Configuração das Variáveis e Integração com o Modbus TCP

Os registradores foram organizados de acordo com a definição realizada na etapa de implementação do servidor no Unity, na qual foram estabelecidos os endereços correspondentes a cada dispositivo do sistema. A partir dessa definição, foram criadas, na interface de configuração do servidor, todas as variáveis envolvidas no processo (Fig. 35). Dessa forma, foi necessário atentar-se cuidadosamente aos endereços e aos tipos de dados associados a cada variável, como *holding registers*, *coils* e *discrete inputs*, a fim de possibilitar a implementação correta das funções de leitura e escrita. Essa organização garantiu a correspondência adequada entre os registradores do servidor no Unity e os registradores do cliente no CODESYS.

Figura 35 – Variáveis Modbus configuradas no CODESYS.



Variável	Mapea...	Canal	Endereço	Tipo	Unidade	Descrição
		Channel 2	%IB0	ARRAY [0..0] OF BYTE		Read Discrete Inputs
		Channel 2[0]	%IB0	BYTE		Read Discrete Inputs
SR1		Bit0	%IX0.0	BOOL		0x0000
SR2		Bit1	%IX0.1	BOOL		0x0001
SR3		Bit2	%IX0.2	BOOL		0x0002
SR4		Bit3	%IX0.3	BOOL		0x0003
SR5		Bit4	%IX0.4	BOOL		0x0004
SR6		Bit5	%IX0.5	BOOL		0x0005
		Channel 4	%IB1	ARRAY [0..0] OF BYTE		Read Coils
		Channel 4[0]	%IB1	BYTE		Read Coils
BT1		Bit0	%IX1.0	BOOL		0x0002
		Channel 5	%QW1	ARRAY [0..2] OF WORD		Write Multiple Registers
FM		Channel 5[0]	%QW1	WORD		0x0000
SM		Channel 5[1]	%QW2	WORD		0x0001
LM		Channel 5[2]	%QW3	WORD		0x0002
		Channel 6	%IW1	ARRAY [0..2] OF WORD		Read Holding Registers

Fonte: Autoral.

Além disso, foi necessária a implementação de variáveis auxiliares para apoiar o desenvolvimento da lógica de controle e gerenciar a troca de dados entre as variáveis de entrada e saída da comunicação Modbus (Fig. 36). Essas variáveis atuaram como armazenamento temporário das informações, contribuindo para a sincronização dos dados e para a correta execução do processo de controle.

Figura 36 – Variáveis auxiliares.

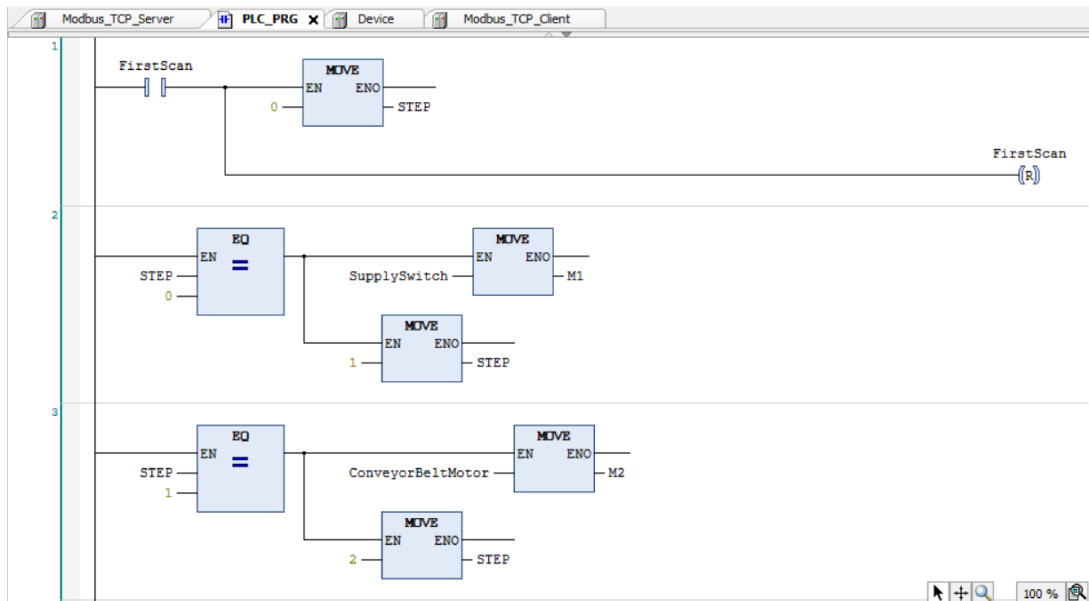
```
PROGRAM PLC_PRG
VAR
    FirstScan : BOOL := TRUE;
    STEP: INT;
    SupplySwitch: BOOL := FALSE;
    ConveyorBeltMotor: BOOL := FALSE;
    StartProcess: BOOL := FALSE;
    CountingSensor: BOOL := FALSE;
    StartSensorFM: BOOL := FALSE;
    PositionSensorFM: BOOL;
    PositionSensorSM: BOOL;
    PositionSensorLM: BOOL;
    LimitSwitch : BOOL;
    CommandFM: WORD;
    CommandSM: WORD;
    CommandLM: WORD;
    AUX1: BOOL := FALSE;
    AUX2: BOOL := FALSE;
    AUX3: BOOL := FALSE;
    AUX4: BOOL := FALSE;
    CTU1: CTU;
    CV1: WORD;
    CTU2: CTU;
    CV2: WORD;
    F_TRIG: F_TRIG;
    R_TRIG: R_TRIG;
    R_TRIG2: R_TRIG;
    R_TRIG3: R_TRIG;
END_VAR
```

Fonte: Autoral.

3.5.2 Implementação da Lógica de Controle em Linguagem Ladder

A programação foi desenvolvida em linguagem Ladder, conforme a norma IEC 61131-3, representando a lógica de acionamento dos equipamentos e leitura dos sensores. Inicialmente, foi implementada uma lógica com o objetivo de replicar o sinal conhecido como *first scan* (Fig. 37), comum em CLPs industriais, que é ativado sempre que o controlador passa do modo *STOP* para o modo *RUN*. Como o CLP utilizado no projeto trata-se de um dispositivo virtual, que não disponibiliza nativamente essa funcionalidade, tornou-se necessário implementar uma variável equivalente. Essa variável foi utilizada para inicializar o sistema e dar início à sequência de leitura e escrita das variáveis envolvidas no processo.

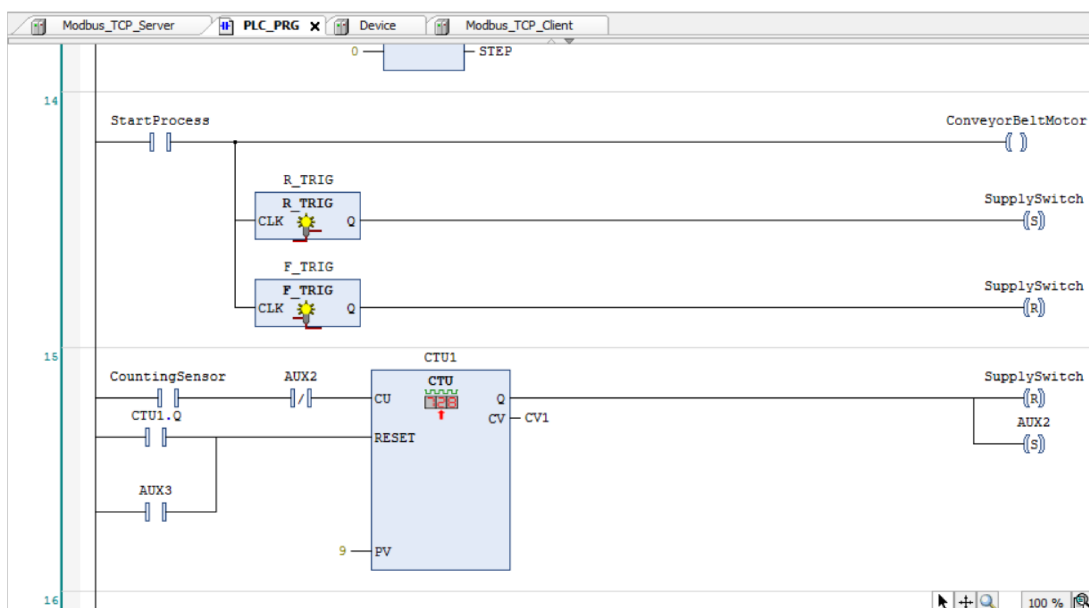
Para garantir um gerenciamento adequado das operações de leitura e escrita, foi implementada uma lógica de execução sequencial do tipo passo a passo (Fig. 37), controlada por uma variável denominada *STEP*. Essa abordagem permitiu a aplicação ordenada das funções associadas às variáveis de entrada e saída, assegurando que, em cada passo, fossem realizadas corretamente as trocas de dados entre as variáveis auxiliares (utilizadas na lógica de controle) e as variáveis físicas vinculadas à comunicação Modbus.

Figura 37 – Lógica de *first scan* e parte da escrita de variáveis em linguagem Ladder.

Fonte: Autoral.

Após a implementação da lógica de leitura e escrita dos dados, foi desenvolvida a lógica de controle do processo propriamente dita, na qual foram definidos os comportamentos dos atuadores com base nos sinais coletados pelos sensores. Nessa etapa, foi implementada a lógica que permite o acionamento da esteira e do abastecedor de garrafas por meio do botão associado ao sistema, garantindo o fornecimento controlado de apenas nove garrafas (Fig. 38). Além disso, as máquinas responsáveis pelos processos de envase, selagem e rotulagem foram configuradas para operar nos instantes adequados, de acordo com a detecção dos sensores de posição ao longo da linha.

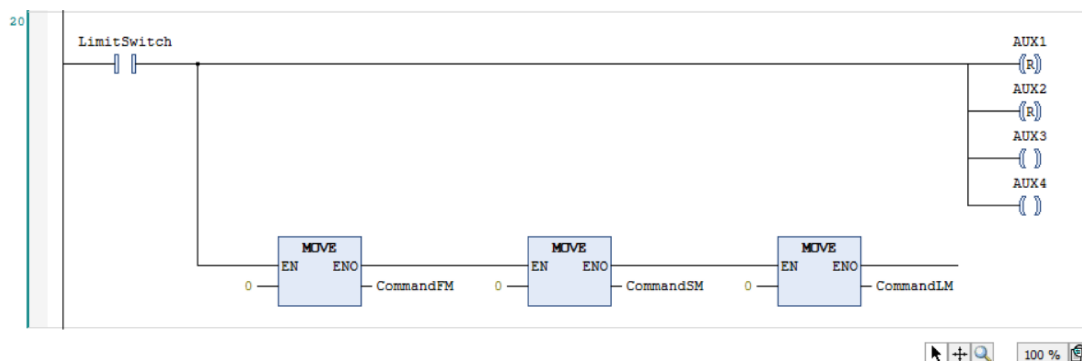
Figura 38 – Trecho da lógica de controle do botão de acionamento e da contagem de garrafas.



Fonte: Autoral.

Por fim, foi desenvolvida uma lógica específica para a reinicialização das variáveis auxiliares (Fig. 39), evitando que dados provenientes de uma simulação anterior fossem propagados para execuções subsequentes, o que poderia ocasionar comportamentos inesperados no sistema. Essa lógica de reinicialização foi configurada para ser acionada sempre que o sensor de fim de curso, localizado ao final da linha de envase, fosse detectado, garantindo a correta preparação do sistema para um novo ciclo de operação.

Figura 39 – Lógica de reinicialização de variáveis.



Fonte: Autoral.

3.6 Testes e Validação do Sistema

Para a validação completa do sistema desenvolvido, foram analisados os seguintes aspectos: a simulação da linha de envase, a comunicação via protocolo Modbus TCP, a lógica de controle implementada em linguagem Ladder e a visualização do processo em ambiente de realidade virtual.

Na validação da simulação da linha de envase, foram observados aspectos relacionados ao desempenho das mecânicas das máquinas e equipamentos que compõem a planta. Avaliou-se cuidadosamente se os movimentos executados pelos modelos tridimensionais reproduziam, de forma funcional e coerente, os movimentos reais desses dispositivos, garantindo que cada equipamento desempenhasse corretamente sua função individual dentro do processo simulado.

Para verificar a efetividade da comunicação estabelecida entre o cliente no CODESYS e o servidor desenvolvido no Unity, foram inicialmente realizados testes de leitura das variáveis pelo cliente, a partir dos sensores simulados. Nessa etapa, avaliou-se se as informações transmitidas pelo servidor da simulação eram corretamente recebidas e interpretadas pelo CLP virtual. Em seguida, foram realizados testes envolvendo os atuadores, nos quais se verificou se os comandos enviados pelo cliente resultavam nos comportamentos esperados na simulação, como o acionamento de motores e a alteração do estado dos dispositivos.

Após a confirmação do funcionamento básico da comunicação, foi realizada a avaliação de outros aspectos relevantes da conexão, incluindo a latência entre o envio e o recebimento das mensagens, a estabilidade da comunicação ao longo do tempo e a precisão na transmissão

dos dados. Para essa análise, foi utilizado o *software* Wireshark, que auxiliou na inspeção dos pacotes trocados entre o cliente e o servidor. Essas avaliações tiveram como objetivo assegurar que o sistema operasse de forma contínua, confiável e sem perdas significativas de dados durante a execução da simulação.

A validação da lógica de controle foi realizada por meio da comparação entre o desempenho da simulação ideal e da simulação real. Considerou-se que a lógica de controle estaria validada caso a simulação real, controlada pelo CLP virtual via comunicação Modbus TCP, reproduzisse de forma equivalente o comportamento observado na simulação ideal, na qual a lógica de controle foi implementada diretamente por meio de scripts em C#.

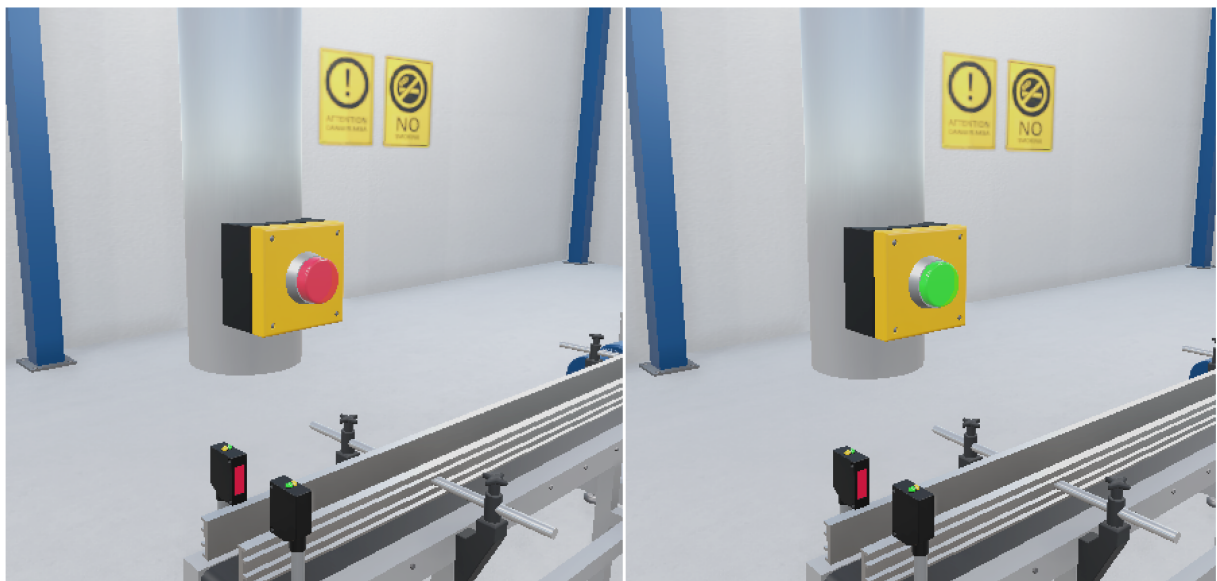
Por fim, a validação da visualização em realidade virtual foi realizada por meio do *Unity Profiler*, recurso disponibilizado nativamente pelo Unity para análise de desempenho da aplicação, incluindo métricas como FPS, uso de CPU e uso de GPU. Inicialmente, as análises foram conduzidas com a aplicação em modo *Play* no ambiente de desenvolvimento, possibilitando o uso do depurador. Posteriormente, foi gerada uma build da aplicação para avaliação do desempenho diretamente no óculos de realidade virtual, permitindo verificar o comportamento do sistema em condições reais de execução. Para o teste da build, foi utilizada a ferramenta Meta Quest Developer Hub. Para a análise dessa *build*, foi utilizada a ferramenta Meta Quest Developer Hub, a qual possibilitou o monitoramento da taxa de quadros por segundo (FPS).

4 Resultados e Discussões

4.1 Simulação da Linha de Envase

O primeiro dispositivo verificado na simulação foi o botão de acionamento acoplado ao cilindro abastecedor. Esperava-se que, ao ser acionado, o botão alterasse sua cor para verde, indicando o estado ativo, e que, ao ser pressionado novamente, retornasse à cor original (vermelho), sinalizando o estado inativo. No modo *Play* do Unity, o botão foi acionado por meio dos controladores do *Meta Quest 3S*, e os resultados demonstraram que essa funcionalidade foi implementada com sucesso, uma vez que o botão apresentou exatamente o comportamento esperado (Fig. 40).

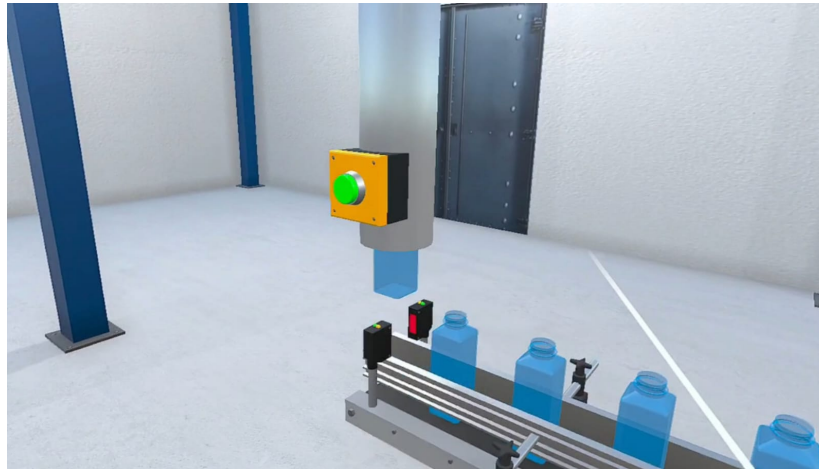
Figura 40 – Funcionamento do botão de acionamento.



Fonte: Autoral.

Em seguida, foi analisado o comportamento das esteiras transportadoras. Esperava-se que estas fossem capazes de transportar as garrafas fornecidas pelo abastecedor a uma velocidade adequada, suficientemente baixa para evitar quedas e suficientemente alta para impedir o acúmulo de garrafas sob o abastecedor. Os movimentos observados na simulação indicaram que as mecânicas das esteiras foram implementadas com êxito, uma vez que cumpriram corretamente sua função no processo de transporte (Fig. 41).

Figura 41 – Esteira transportadora em funcionamento durante a simulação.



Fonte: Autoral.

Posteriormente, foram realizados testes visuais com a máquina de envase, com o objetivo de verificar a correta execução de seus movimentos. O equipamento deveria realizar quatro ações principais: deslocar os cilindros de envase para a posição inicial, acompanhar o deslocamento das garrafas ao longo da esteira, realizar o movimento vertical de descida dos cilindros em direção às garrafas e, por fim, efetuar o enchimento com o líquido. Os resultados indicaram que a máquina foi capaz de executar corretamente todos os quatro movimentos previstos. No entanto, observou-se que, diferentemente do cenário ideal, na simulação real (na qual o comportamento do sistema é determinado pelos dados transmitidos via comunicação Modbus) ocorreu um atraso no movimento vertical de descida dos cilindros, ocasionando um desalinhamento visual no encaixe com as garrafas (Fig. 42). Esse comportamento pode ser atribuído à latência característica da comunicação entre o ambiente de simulação e o ambiente responsável pela lógica de controle. Apesar disso, de forma geral, os movimentos da máquina mostraram-se satisfatórios do ponto de vista funcional para a simulação da linha de envase.

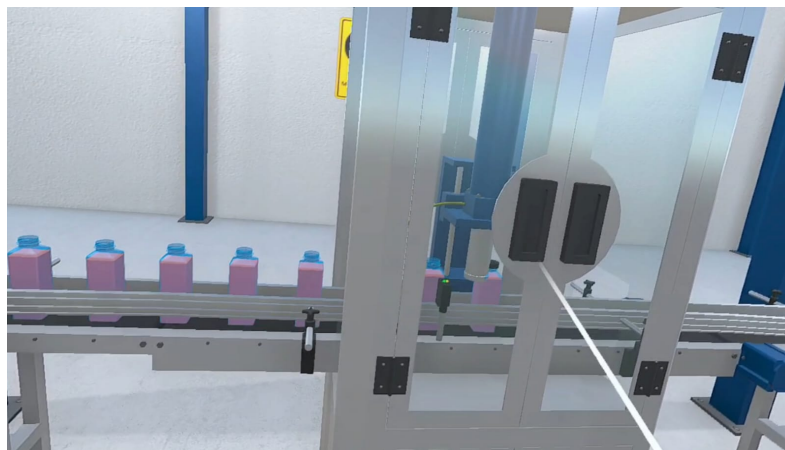
Figura 42 – Máquina de envase em funcionamento durante a simulação.



Fonte: Autoral.

Na máquina de selagem das garrafas, eram esperados três movimentos principais: a descida vertical em direção às garrafas, o retorno à posição inicial e a ação de selagem propriamente dita. As mecânicas implementadas atenderam aos requisitos funcionais necessários para a execução do processo. Entretanto, assim como observado na máquina de envase, foram identificados pequenos desvios visuais nos movimentos verticais, também associados a atrasos temporais (Fig. 43). Nesse caso, a divergência em relação ao comportamento visual esperado está relacionada à implementação das movimentações por meio de scripts utilizando a função *update()* do Unity, que é executada a cada quadro (*frame*) e pode variar conforme a taxa de quadros por segundo (FPS), em vez de utilizar animações pré-definidas, as quais oferecem maior precisão e consistência em termos de tempo.

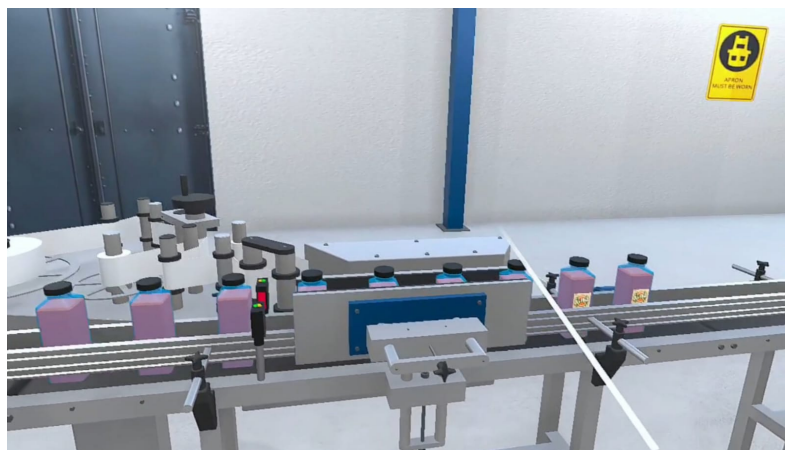
Figura 43 – Máquina de selagem em funcionamento durante a simulação.



Fonte: Autoral.

Por fim, a última máquina analisada foi a rotuladora, um equipamento de funcionamento mais simples, cujo resultado visual esperado consistia na aplicação dos rótulos nas garrafas que passassem por ela. O comportamento observado atendeu plenamente às expectativas, cumprindo seu objetivo funcional dentro do processo de simulação da linha de envase (Fig. 44).

Figura 44 – Máquina de rotulagem em funcionamento durante a simulação.



Fonte: Autoral.

4.2 Comunicação via Protocolo Modbus TCP

Para a verificação da comunicação Modbus TCP entre os dois ambientes da aplicação, foram realizados testes básicos de troca de dados. Inicialmente, verificou-se se as variáveis de leitura configuradas no cliente CODESYS estavam sendo atualizadas corretamente de acordo com os dados enviados pelos sensores simulados no Unity. Em seguida, avaliou-se se os comandos enviados pelo cliente atuavam efetivamente no comportamento dos atuadores presentes na simulação. Em ambas as etapas, a troca de dados ocorreu de forma satisfatória, confirmando que a conexão entre os dois softwares foi estabelecida com sucesso.

Além da validação funcional da comunicação, buscou-se avaliar a qualidade da conexão por meio da análise da latência da comunicação Modbus TCP. Para isso, foi utilizado o *software* Wireshark, que possibilitou a captura do tráfego de pacotes durante a comunicação Modbus TCP, bem como a identificação dos instantes de envio e recebimento das mensagens. Como auxílio, foi aplicado um filtro para capturar exclusivamente os pacotes trafegados pela rede Modbus TCP, facilitando a análise dos dados relevantes.

Em um primeiro momento, o tráfego foi capturado (Fig. 45) com a aplicação executando em modo *Play* no Unity, ou seja, com cliente e servidor Modbus operando no mesmo dispositivo, por meio da interface de *loopback*. A partir dos dados coletados, foi possível calcular a latência observando-se a diferença entre os instantes de captura das mensagens de requisição (*request*) e resposta (*response*), conforme a expressão:

$$T_{\text{latência}} = T_{\text{response}} - T_{\text{request}}$$

$$T_{\text{latência}} = 0,262784 - 0,262484$$

$$T_{\text{latência}} = 0,3\text{ms}$$

Figura 45 – Captura de tráfego Modbus TCP em *loopback*.

No.	Time	Source	Destination	Protocol	Length	Info
86	0.140887	127.0.0.1	127.0.0.1	Modbus/TCP	56	Response: Trans: 17795; Unit: 1, Func: 15: Write Multiple Coils
90	0.161267	127.0.0.1	127.0.0.1	Modbus/TCP	56	Query: Trans: 17796; Unit: 1, Func: 2: Read Discrete Inputs
92	0.161384	127.0.0.1	127.0.0.1	Modbus/TCP	54	Response: Trans: 17796; Unit: 1, Func: 2: Read Discrete Inputs
96	0.181317	127.0.0.1	127.0.0.1	Modbus/TCP	56	Query: Trans: 17797; Unit: 1, Func: 1: Read Coils
98	0.181443	127.0.0.1	127.0.0.1	Modbus/TCP	54	Response: Trans: 17797; Unit: 1, Func: 1: Read Coils
102	0.201388	127.0.0.1	127.0.0.1	Modbus/TCP	63	Query: Trans: 17798; Unit: 1, Func: 16: Write Multiple Registers
104	0.201500	127.0.0.1	127.0.0.1	Modbus/TCP	56	Response: Trans: 17798; Unit: 1, Func: 16: Write Multiple Registers
108	0.221920	127.0.0.1	127.0.0.1	Modbus/TCP	56	Query: Trans: 17799; Unit: 1, Func: 3: Read Holding Registers
110	0.222036	127.0.0.1	127.0.0.1	Modbus/TCP	59	Response: Trans: 17799; Unit: 1, Func: 3: Read Holding Registers
141	0.241933	127.0.0.1	127.0.0.1	Modbus/TCP	58	Query: Trans: 17800; Unit: 1, Func: 15: Write Multiple Coils
143	0.242052	127.0.0.1	127.0.0.1	Modbus/TCP	56	Response: Trans: 17800; Unit: 1, Func: 15: Write Multiple Coils
157	0.262484	127.0.0.1	127.0.0.1	Modbus/TCP	56	Query: Trans: 17801; Unit: 1, Func: 2: Read Discrete Inputs
161	0.262784	127.0.0.1	127.0.0.1	Modbus/TCP	54	Response: Trans: 17801; Unit: 1, Func: 2: Read Discrete Inputs
165	0.282495	127.0.0.1	127.0.0.1	Modbus/TCP	56	Query: Trans: 17802; Unit: 1, Func: 1: Read Coils
167	0.282620	127.0.0.1	127.0.0.1	Modbus/TCP	54	Response: Trans: 17802; Unit: 1, Func: 1: Read Coils
171	0.303033	127.0.0.1	127.0.0.1	Modbus/TCP	63	Query: Trans: 17803; Unit: 1, Func: 16: Write Multiple Registers
173	0.303143	127.0.0.1	127.0.0.1	Modbus/TCP	56	Response: Trans: 17803; Unit: 1, Func: 16: Write Multiple Registers
179	0.323543	127.0.0.1	127.0.0.1	Modbus/TCP	56	Query: Trans: 17804; Unit: 1, Func: 3: Read Holding Registers
181	0.323666	127.0.0.1	127.0.0.1	Modbus/TCP	59	Response: Trans: 17804; Unit: 1, Func: 3: Read Holding Registers
185	0.344619	127.0.0.1	127.0.0.1	Modbus/TCP	58	Query: Trans: 17805; Unit: 1, Func: 15: Write Multiple Coils
187	0.344738	127.0.0.1	127.0.0.1	Modbus/TCP	56	Response: Trans: 17805; Unit: 1, Func: 15: Write Multiple Coils
191	0.364674	127.0.0.1	127.0.0.1	Modbus/TCP	56	Query: Trans: 17806; Unit: 1, Func: 2: Read Discrete Inputs
193	0.364798	127.0.0.1	127.0.0.1	Modbus/TCP	54	Response: Trans: 17806; Unit: 1, Func: 2: Read Discrete Inputs

Fonte: Autoral.

Na sequência, foi gerada uma *build* da aplicação, a qual foi executada no óculos de realidade virtual *Meta Quest 3S*, permitindo a captura do tráfego Modbus TCP entre dois

dispositivos distintos, ambos conectados à mesma rede Wi-Fi (Fig. 46). De forma análoga ao procedimento anterior, utilizou-se o Wireshark com o mesmo filtro de protocolo, e a latência foi novamente calculada com base na diferença entre os instantes de requisição e resposta.

$$T_{\text{latência}} = T_{\text{response}} - T_{\text{request}}$$

$$T_{\text{latência}} = 0,379294 - 0,374842$$

$$T_{\text{latência}} = 4,452\text{ms}$$

Figura 46 – Captura de tráfego Modbus TCP em dispositivos distintos.

No.	Time	Source	Destination	Protocol	Leng	Info
77	0.293209	192.168.12.5	192.168.12.6	Modbus/TCP	66	Query: Trans: 522; Unit: 1, Func: 1: Read Coils
78	0.299506	192.168.12.6	192.168.12.5	Modbus/TCP	64	Response: Trans: 522; Unit: 1, Func: 1: Read Coils
85	0.313744	192.168.12.5	192.168.12.6	Modbus/TCP	73	Query: Trans: 523; Unit: 1, Func: 16: Write Multiple Registers
86	0.317426	192.168.12.6	192.168.12.5	Modbus/TCP	66	Response: Trans: 523; Unit: 1, Func: 16: Write Multiple Registers
90	0.333679	192.168.12.5	192.168.12.6	Modbus/TCP	66	Query: Trans: 524; Unit: 1, Func: 3: Read Holding Registers
92	0.341962	192.168.12.6	192.168.12.5	Modbus/TCP	69	Response: Trans: 524; Unit: 1, Func: 3: Read Holding Registers
94	0.354320	192.168.12.5	192.168.12.6	Modbus/TCP	68	Query: Trans: 525; Unit: 1, Func: 15: Write Multiple Coils
95	0.357959	192.168.12.6	192.168.12.5	Modbus/TCP	66	Response: Trans: 525; Unit: 1, Func: 15: Write Multiple Coils
97	0.374842	192.168.12.5	192.168.12.6	Modbus/TCP	66	Query: Trans: 526; Unit: 1, Func: 2: Read Discrete Inputs
100	0.379294	192.168.12.6	192.168.12.5	Modbus/TCP	64	Response: Trans: 526; Unit: 1, Func: 2: Read Discrete Inputs
102	0.395373	192.168.12.5	192.168.12.6	Modbus/TCP	66	Query: Trans: 527; Unit: 1, Func: 1: Read Coils
103	0.398741	192.168.12.6	192.168.12.5	Modbus/TCP	64	Response: Trans: 527; Unit: 1, Func: 1: Read Coils
105	0.415358	192.168.12.5	192.168.12.6	Modbus/TCP	73	Query: Trans: 528; Unit: 1, Func: 16: Write Multiple Registers
107	0.419959	192.168.12.6	192.168.12.5	Modbus/TCP	66	Response: Trans: 528; Unit: 1, Func: 16: Write Multiple Registers
109	0.436025	192.168.12.5	192.168.12.6	Modbus/TCP	66	Query: Trans: 529; Unit: 1, Func: 3: Read Holding Registers
110	0.440192	192.168.12.6	192.168.12.5	Modbus/TCP	69	Response: Trans: 529; Unit: 1, Func: 3: Read Holding Registers
112	0.456435	192.168.12.5	192.168.12.6	Modbus/TCP	68	Query: Trans: 530; Unit: 1, Func: 15: Write Multiple Coils

Fonte: Autoral.

Os resultados obtidos indicaram que a latência nos dois cenários (comunicação em *loopback* e comunicação entre dispositivos distintos) apresentaram valores reduzidos, porém com uma clara vantagem para o caso em que cliente e servidor estavam executando no mesmo dispositivo. Esse comportamento é esperado, uma vez que, no modo *loopback*, os pacotes não percorrem meios físicos de transmissão, enquanto na comunicação entre dispositivos os dados precisam atravessar a interface de rede, o roteador e o meio sem fio, estando sujeitos a interferências e atrasos adicionais.

4.3 Lógica de Controle Implementada em Linguagem Ladder

Para a validação da lógica de controle implementada em linguagem Ladder no CLP virtual do CODESYS, foi analisado o comportamento de cada equipamento a partir dos comandos enviados pelo CLP, considerando os dados provenientes dos sensores da simulação. A lógica de controle foi considerada implementada com êxito quando a simulação pôde ser executada integralmente sem a ocorrência de erros no processo, permitindo o cumprimento do objetivo proposto, no qual as nove garrafas deveriam ser corretamente abastecidas, seladas e rotuladas ao final do ciclo. Tal condição foi atendida, demonstrando o funcionamento adequado da lógica desenvolvida.

Além disso, a simulação ideal, executada por meio de scripts em C#, foi utilizada como referência para a comparação do desempenho e do comportamento dos dispositivos ao longo de todo o processo de envase. Os resultados obtidos mostraram-se satisfatórios, uma vez que não foram observadas diferenças significativas entre a simulação ideal e a simulação controlada via CLP no que se refere ao comportamento dos sensores e atuadores durante a execução do processo.

4.4 Visualização do Processo em Ambiente de Realidade Virtual

O primeiro aspecto avaliado durante a execução da aplicação em realidade virtual foi a capacidade de interação dos controladores com os objetos e interfaces das cenas, bem como o controle do usuário em relação à locomoção e ao ajuste de visão. Em ambos os casos, os resultados obtidos foram satisfatórios, uma vez que foi possível executar sem dificuldades todas as tarefas propostas por meio dos controladores do óculos de realidade virtual. Esse resultado evidencia que os modelos de interação disponibilizados pelo *Meta Building Blocks* atenderam adequadamente às necessidades do projeto.

Em um segundo momento, observou-se a existência de diferenças na visualização do ambiente quando a aplicação era executada em modo *Play* no ambiente de desenvolvimento do Unity, em comparação com a execução da *build* gerada para o *Meta Quest 3S*. Esse comportamento é esperado, uma vez que a GPU do computador utilizado no desenvolvimento é dedicada e significativamente mais potente do que a GPU integrada do óculos, que opera sob um sistema Android e possui restrições mais severas de desempenho. Dessa forma, no ambiente de desenvolvimento foram observadas iluminação e sombras mais nítidas, além de cores mais claras, enquanto na execução nativa no *headset* as luzes e sombras apresentaram menor nível de detalhamento e as cores mostraram-se levemente mais escuras (Fig. 47 e ??). Ressalta-se, contudo, que essas diferenças se restringem ao aspecto visual da renderização, não afetando a funcionalidade da aplicação, a qual permaneceu plenamente operacional e adequada ao objetivo proposto.

Figura 47 – Comparação de cores e sombras entre o ambiente de desenvolvimento (superior) e de *build* (inferior).

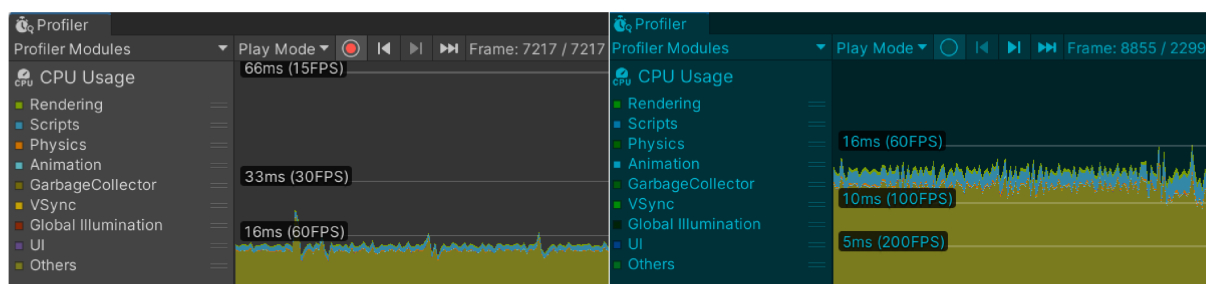


Fonte: Autoral.

Outro aspecto avaliado foi a percepção sonora na aplicação. Os efeitos sonoros foram configurados de modo a proporcionar uma experiência espacializada, permitindo ao usuário perceber variações de volume em função da distância em relação à fonte sonora, bem como a distinção entre os canais esquerdo e direito, de acordo com a posição do usuário no ambiente virtual. Os resultados demonstraram que a implementação do áudio ocorreu de forma efetiva, uma vez que todas essas características foram claramente percebidas durante a simulação.

Na sequência, foram analisados os valores de taxa de quadros por segundo (FPS) para as duas situações de execução: no ambiente de desenvolvimento e na *build* executada diretamente no *Meta Quest 3S*. No primeiro caso, utilizou-se a ferramenta Unity Profiler, que permitiu a captura dos dados de desempenho em tempo real durante a execução da aplicação. Os dados coletados (Fig. 48) indicaram que a aplicação manteve, de forma geral, uma taxa de quadros próxima de 60 FPS, atingindo valores superiores, próximos a 100 FPS, na cena inicial, a qual contém apenas a interface principal.

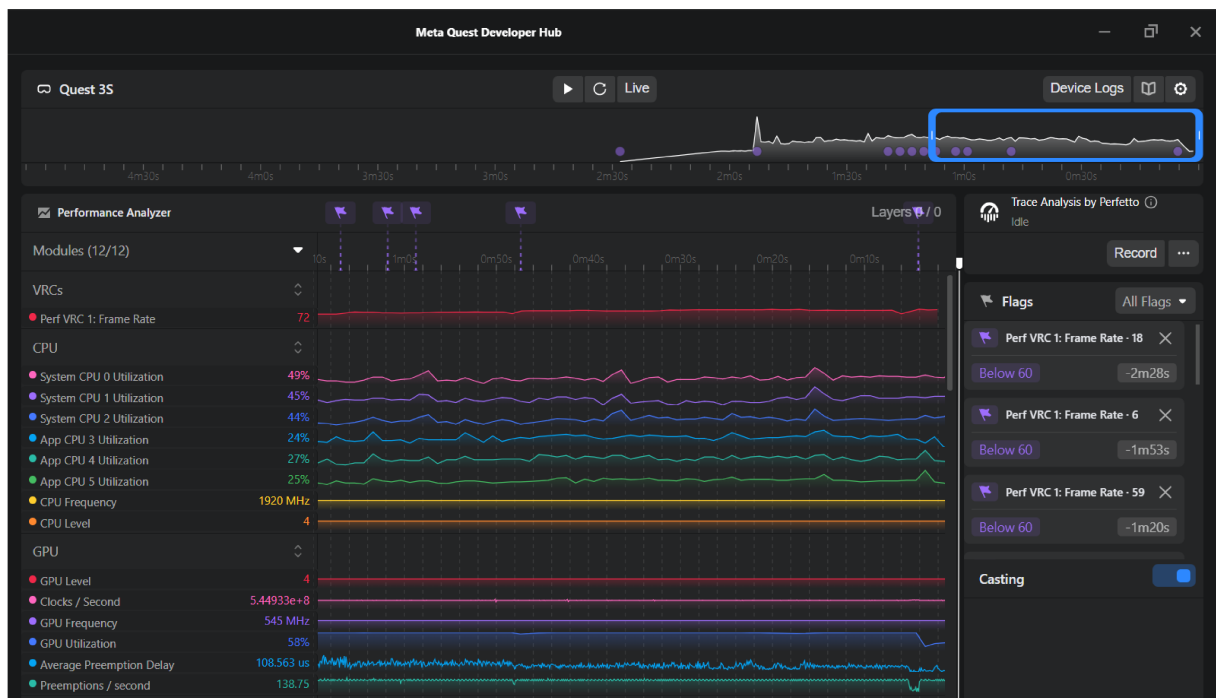
Figura 48 – Taxa de quadros por segundo da aplicação em ambiente de desenvolvimento, com simulação à esquerda e cena inicial à direita.



Fonte: Autoral.

Para a análise do desempenho da *build* executada no *Meta Quest 3S*, os dados foram obtidos por meio do Meta Quest Developer Hub (Fig. 49). Nesse cenário, foi observada uma taxa de quadros em torno de 72 FPS, valor levemente superior ao obtido durante a execução no ambiente de desenvolvimento. Essa diferença é justificada pelo fato da *build* ser otimizada para execução nativa no *headset*, enquanto no modo *Play* há o custo de processamento adicional do Editor do Unity, bem como o custo de renderização e espelhamento das imagens para o óculos via conexão USB.

Figura 49 – Taxa de quadros por segundo da aplicação em versão compilada (*build*).



Fonte: Autoral.

Considerando que o produto final do projeto corresponde à *build* destinada à execução direta no *Meta Quest 3S*, a taxa de FPS obtida mostrou-se plenamente satisfatória, garantindo uma experiência estável e confortável para o usuário durante a simulação.

Dessa forma, com base nos resultados gerais obtidos ao longo do desenvolvimento da simulação em realidade virtual da linha de envase, controlada por meio de um CLP virtual com comunicação via Modbus TCP, conclui-se que o sistema apresentou desempenho adequado e comportamento consistente. O ambiente desenvolvido demonstrou-se uma solução viável para a validação de lógicas de controle implementadas em CLPs, bem como para a análise de processos industriais em um ambiente tridimensional interativo e de baixo custo experimental.

5 Conclusão

Com a ascensão da Indústria 4.0, observou-se a ampliação do uso de tecnologias avançadas em contextos industriais, promovendo a integração entre aspectos tradicionais das fábricas e soluções inovadoras, com o objetivo de otimizar processos produtivos e antecipar falhas. Nesse contexto, o presente trabalho propôs o desenvolvimento da simulação de um processo de envase de garrafas em ambiente de realidade virtual, controlado por meio de um CLP virtual programado em linguagem Ladder e integrado por meio do protocolo industrial Modbus TCP, evidenciando que tecnologias inovadoras podem operar de forma harmoniosa com sistemas legados da automação industrial.

O desenvolvimento do projeto possibilitou a aplicação de diversos conhecimentos adquiridos ao longo do curso, abrangendo áreas como redes industriais, instrumentação industrial, automação e computação gráfica. Dessa forma, o trabalho reforça a importância da interdisciplinaridade no repertório técnico de um(a) engenheiro(a), especialmente para sua atuação em projetos alinhados aos conceitos e demandas da Indústria 4.0.

Ao final, os resultados obtidos mostraram-se compatíveis com os objetivos propostos, sendo possível acompanhar, em um ambiente imersivo, o funcionamento das principais etapas de um processo de envase de garrafas. A aplicação apresentou um nível satisfatório de detalhamento visual e mecânicas de movimento coerentes com os dispositivos reais, apesar da identificação de pequenos atrasos de sincronização, especialmente nas máquinas de envase e selagem, os quais não comprometeram a funcionalidade geral da simulação. No que se refere à comunicação Modbus TCP, esta se mostrou estável e confiável, embora tenha sido identificada uma latência inerente à comunicação, a qual não impactou de forma significativa o desempenho do processo. A implementação do CLP virtual e da lógica de controle em linguagem Ladder também foi bem-sucedida, permitindo o controle remoto da planta simulada.

Foram identificadas algumas limitações ao longo do desenvolvimento, destacando-se a qualidade visual da aplicação executada diretamente no óculos Meta Quest 3S, na qual observou-se um ambiente mais escuro e com sombras menos definidas quando comparado à execução em ambiente de desenvolvimento. Tal limitação está associada às restrições de *hardware* do *headset* e pode ser mitigada futuramente por meio da aplicação de técnicas específicas de otimização voltadas para plataformas mobile. Outra limitação refere-se ao fato de a simulação não representar uma linha de envase completa, porém vale ressaltar que as etapas implementadas mostraram-se suficientes para validar o principal objetivo do trabalho, que é demonstrar a viabilidade da integração entre diferentes tecnologias.

A aplicação desenvolvida mostrou-se uma alternativa segura e eficiente para o teste e validação de lógicas de controle implementadas em CLPs, permitindo a visualização em tempo

real da atuação do código em um ambiente tridimensional de baixo custo experimental e sem riscos operacionais. Nesse sentido, o projeto apresenta potencial de expansão futura, podendo ser integrado a CLPs físicos reais, estendido para representar uma linha de envase completa ou outros processos industriais, bem como utilizado como ferramenta educacional ou integrado a tecnologias de IoT, evoluindo para o conceito de gêmeo digital (*Digital Twin*).

De forma geral, os resultados do projeto indicam que o mesmo foi concluído com sucesso, apesar das limitações identificadas, configurando-se como uma demonstração concreta de que a integração de tecnologias provenientes de diferentes áreas é viável e constitui um dos pilares fundamentais da Indústria 4.0, desempenhando um papel relevante na modernização e otimização dos processos produtivos no contexto industrial contemporâneo.

Referências Bibliográficas

3S-Smart Software Solutions GmbH. *Device Directory – Devices programmable with CODESYS*. 2015. Disponível em: <<http://devices.codesys.com/device-directory.html>>. Acesso em: 21 out. 2025. Citado na página 24.

ABAD, J. D. *Modelo de Iluminación Phong: Tutorial*. 2016. Disponível em: <<https://acodigo.blogspot.com/2016/04/modelo-de-iluminacion-phong-tutorial.html>>. Acesso em: 25 out. 2025. Citado na página 34.

ARTERY Technology. *Modbus on AT32 MCU*. 2024. Disponível em: <<https://medium.com/@arterychip/modbus-on-at32-mcu-5aadb2c586c0>>. Publicado em 10 abr. 2024; Acesso em 22 out. 2025. Citado na página 28.

Associação Brasileira da Indústria de Alimentos (ABIA). *Números do setor*. 2025. Disponível em: <<https://www.abia.org.br/numeros-setor>>. Acesso em: 12 out. 2025. Citado na página 14.

Autodesk Inc. *Autodesk Inventor: Overview of 3D CAD Software*. 2025. Acesso em: 26 out. 2025. Disponível em: <<https://www.autodesk.com/products/inventor/overview>>. Citado na página 34.

BANDYOPADHYAY, S.; BHATTACHARYA, R. *Discrete and Continuous Simulation: Theory and Practice*. Boca Raton, FL: CRC Press, 2014. ISBN 978-1-4665-6282-0. Citado 2 vezes nas páginas 30 e 31.

BAPTISTA, G. T. Trabalho de Conclusão de Curso (Bacharel em Ciências Náuticas), *Controlador Lógico Programável*. Rio de Janeiro, RJ, Brasil: [s.n.], 2015. Orientador: Professor Engenheiro José Barbosa da Silva Filho. Disponível em: <<https://repositorio.marinha.mil.br/bitstream/ripcmb/29484/1/00000bf5.pdf>>, Acesso em: 19 out. 2025. Citado 3 vezes nas páginas 21, 22 e 23.

BUCIOLI, A. A. B. et al. Usando realidade virtual e aumentada na visualização da simulação de sistemas de automação industrial. —, 2006. Acesso em: 25 out. 2025. Disponível em: <<https://tinyurl.com/vr-automacao>>. Citado na página 31.

CARDOSO, L. d. C. *Modelagem e Animação 3D*. Livros1, s.d. Acesso em: 25 out. 2025. Disponível em: <<https://www.livros1.com.br/pdf-read/livar/MODELAGEM-E-ANIMA%C3%87%C3%83O-3D.pdf>>. Citado 2 vezes nas páginas 33 e 34.

CINTRA, J. P. A. Trabalho de Conclusão de Curso (Graduação em Engenharia Mecânica), *Aplicação da metodologia PDCA para implementação de sistema de manutenção autônoma em linha de envase*. Fortaleza, CE: [s.n.], 2023. Acesso em: 26 out. 2025. Disponível em: <https://repositorio.ufc.br/bitstream/riufc/78079/3/2023_tcc_jpacintra.pdf>. Citado na página 38.

CODESYS GmbH. *CODESYS – The leading automation software for industrial control systems*. 2025. Disponível em: <<https://www.codesys.com/>>. Acesso em: 21 out. 2025. Citado na página 23.

CORDEIRO, J. A. F. Monografia (Engenharia de Controle e Automação), *Estudo dos Principais Protocolos de Redes Industriais Utilizadas no Brasil: AS-I, Modbus e Profibus*. Ouro Preto, MG, Brasil: [s.n.], 2019. Orientadora: Profª Drª Karla Boaventura Pimenta Palmieri. Disponível em: <<https://repositorio.ufop.br/>>, Acesso em: 21 out. 2025. Citado 3 vezes nas páginas 26, 27 e 30.

COSTA, I. M.; LISBOA, S. N. D.; SANTOS, T. P. Automação industrial. *DCA447 - Departamento de Engenharia de Computação e Automação, Universidade Federal do Rio Grande do Norte*, Natal, RN, Brasil, 2003. Disponível em: <http://www.dca.ufrn.br/~affonso/FTP/DCA447/trabalho1/trabalho1_15_doc.doc>. Acesso em: 19 out. 2025. Citado na página 17.

DEMÉTRIO, D. J.; JUNIOR, O. R. Comparativo de redes de comunicação de robôs industriais. *Revista de Engenharia e Tecnologia (RET)*, v. 17, n. 2, 2025. Acesso em: 21 out. 2025. Disponível em: <<https://revistas.uepg.br/index.php/ret/article/view/23166/209209218647>>. Citado na página 26.

DISCHER, M. *Integrating Coding, Marking, and Labeling Equipment with Bagging Machines*. 2025. Disponível em: <<https://www.videojet.com/us/homepage/resources/glossary/bagging-machines.html>>. Acesso em: 26 out. 2025. Citado na página 38.

DITZEL, L. F. S. et al. Digital twin na manufatura: uma revisão sistemática da literatura. *Brazilian Journal of Development*, v. 6, n. 9, p. 66183–66197, 2020. Disponível em: <<https://ojs.brazilianjournals.com.br/ojs/index.php/BRJD/article/view/16286/13317>>. Citado na página 32.

D'ADDARIO, J. J. P. Desenho técnico e indústria 4.0: a integração com simulação, impressão 3d e realidade aumentada. *RCMOS – Revista Científica Multidisciplinar O Saber*, v. 5, n. 1, p. —, 2025. Acesso em: 25 out. 2025. Disponível em: <<https://submissoesrevistarcmos.com.br/rcmos/article/view/1074/2475>>. Citado na página 34.

EasyModbusTCP. *Modbus Library for .NET/Java and Python – Communication library and professional tools for industrial communication*. 2025. Disponível em: <<http://easymodbustcp.net/en/>>. Acesso em: 26 out. 2025. Citado na página 37.

FRANCHI, C. M. *Controle de Processos Industriais: Princípios e Aplicações*. São Paulo: Érica, 2014. Citado 2 vezes nas páginas 18 e 19.

FREITAS, E. F. d.; FONSECA, M. d. O.; MARQUES, R. M. Utilização de técnicas de simulação para desenvolvimento, testes e validação de projetos de automação industrial. *Tecnologia em Metalurgia e Materiais*, v. 4, n. 1, p. 18–23, 2007. Acesso em: 25 out. 2025. Disponível em: <<https://tecnologiамmm.com.br/article/10.4322/tmm.00401004/pdf/1573492069-4-1-18.pdf>>. Citado na página 32.

FUCHTER, S. K.; SCHLICHTING, M. S. Utilização da realidade virtual voltada para o treinamento industrial. *Revista Científica Multidisciplinar Núcleo do Conhecimento*, 2018. Acesso em: 25 out. 2025. Disponível em: <<https://tinyurl.com/vr-treinamento-industrial>>. Citado na página 31.

GARCIA, C. *Controle de Processos Industriais: Estratégias Convencionais*. São Paulo: Blucher, 2015. v. 1. Citado na página 18.

GRANERMANN, A. Trabalho de Conclusão de Curso, *Desenvolvimento de um sistema robotizado de encaixotamento de potes para uma empresa de produtos lácteos*. Blumenau, SC: [s.n.], 2020. Acesso em: 21 out. 2025. Citado na página 26.

GRIGOLETTO, V. O. Trabalho de Conclusão de Curso (Graduação em Engenharia Elétrica com ênfase em Eletrônica), *Protocolos de comunicação para automação de sistemas de energia: análise teórica e aplicação prática*. São Carlos, SP: [s.n.], 2012. Citado 2 vezes nas páginas 25 e 26.

GROOVER, M. P. *Automação Industrial*. 4. ed. São Paulo: Book Libre, 2010. Citado na página 17.

InstrumentationTools Editorial Staff. *Background of Modbus – ASCII and RTU Data Frames*. 2022. Disponível em: <<https://instrumentationtools.com/background-of-modbus-ascii-and-rtu-data-frames/>>. Atualizado em 8 nov. 2022; Acesso em 22 out. 2025. Citado na página 28.

International Labour Organization. *Encyclopaedia of Occupational Health and Safety*. Geneva: International Labour Organization, 2025. Citado 2 vezes nas páginas 14 e 39.

IWATA, K. et al. A modelling and simulation architecture for virtual manufacturing systems. *Annals of the CIRP*, v. 44, p. 399–402, jan. 1995. Citado 2 vezes nas páginas 15 e 32.

JUNIOR, E. R. Trabalho de Conclusão de Curso (Engenharia de Controle e Automação), *Simulação Aplicada a uma Linha de Produção de Pães Automatizada*. Uberlândia, MG: [s.n.], 2021. Orientador: Prof. Dr. Renato Santos Carrijo. Disponível em: <<https://repositorio.ufu.br/handle/123456789/29484>>, Acesso em: 21 out. 2025. Citado na página 23.

JÚNIOR, A. A. d. O. *A importância dos softwares de simulação dentro da Indústria 4.0: uma análise da inserção do Digital Twin nos contextos industriais*. 2023. Monografia (Graduação em Engenharia de Produção) — Escola de Minas, Universidade Federal de Ouro Preto, Ouro Preto, MG. Acesso em: 25 out. 2025. Disponível em: <https://monografias.ufop.br/bitstream/35400000/5784/7/MONOGRAFIA_Import%C3%A2nciaSoftwaresSimula%C3%A7%C3%A3o.pdf>. Citado na página 32.

KIRNER, C.; SISCOUTO, R. *Realidade Virtual e Aumentada: Conceitos, Projeto e Aplicações*. João Pessoa, PB, Brasil: Laboratório de Tecnologia e Educação (LABTEVE), Universidade Federal da Paraíba, 2007. Acesso em: 25 out. 2025. Disponível em: <http://www.de.ufpb.br/~labteve/publi/2007_svrps.pdf>. Citado 2 vezes nas páginas 31 e 32.

Meta Platforms, Inc. *Meta Quest Developer Hub*. 2025. Disponível em: <<https://developer.oculus.com/meta-quest-developer-hub/>>. Acesso em: 10 fev. 2026. Citado na página 38.

NASCIMENTO, J. M. A. d.; LUCENA, P. B. d. Trabalho da disciplina DCA447 - Redes para Automação Industrial, *Protocolo Modbus*. Natal, RN: [s.n.], 2003. Disponível em: <https://www.dca.ufrn.br/~affonso/FTP/DCA447/trabalho3/trabalho3_13.pdf>. Citado na página 27.

PEREIRA, I. P.; JÚNIOR, E. F. A importância da automatização no sistema de envase. *Interface Tecnológica*, v. 20, n. 2, 2023. Acesso em: 19 out. 2025. Disponível em:

<https://revista.fatectq.edu.br/interfacetecnologica/pt_BR/article/view/1764/1007>. Citado 2 vezes nas páginas 20 e 39.

RIBEIRO, M. A. *Controle de Processo*. 8. ed. São Paulo: Editora Érica, 2020. 8ª edição. Citado na página 18.

Rockwell Automation. *CLP Rockwell 1763L16BWA*. 2025. Disponível em: <<https://www.mercadolivre.com.br/clp-rockwell-1763l16bwa/up/MLBU2233443230>>. Imagem do equipamento consultada em 19 out. 2025. Citado na página 20.

ROGGIA, L.; FUENTES, R. C. *Automação Industrial*. Santa Maria, RS: Universidade Federal de Santa Maria, Colégio Técnico Industrial de Santa Maria, Rede e-Tec Brasil, 2016. ISBN 978-85-9450-001-4. Citado 4 vezes nas páginas 20, 21, 22 e 23.

ROSÁRIO, J. M. *Automação Industrial*. São Paulo: Editora Baraúna, 2009. Copyright © 2009 by Editora Baraúna SE Ltda. ISBN 978-85-7923-000-4. Citado na página 20.

SACOMANO, J. B. et al. *Indústria 4.0: Conceitos, Tecnologias e Aplicações*. São Paulo: Novatec Editora, 2019. Acesso em: 18 out. 2025. Disponível em: <https://books.google.com.br/books?hl=pt-BR&lr=&id=PNCuDwAAQBAJ&oi=fnd&pg=PA17&dq=industria+4.0&ots=o1TXBvJFVi&sig=YmssL9Pcu0B-b97b5DucoRXD2Pg&redir_esc=y#v=onepage&q&f=false>. Citado na página 17.

SAKURAI, R.; ZUCHI, J. D. As revoluções industriais até a indústria 4.0. *Interface Tecnológica*, v. 15, n. 2, 2018. Acesso em: 19 out. 2025. Disponível em: <https://revista.fatectq.edu.br/interfacetecnologica/pt_BR/article/view/386/335>. Citado na página 17.

SANTOS, G. J.; CARVALHO, H. d. *Projeto e desenvolvimento de interface tridimensional para sistema supervisor e de treinamento*. São Paulo, SP, Brasil: [s.n.], 2016. Trabalho de formatura. Disponível em: <<https://bdta.abcd.usp.br/directbitstream/3644fb69-d650-4783-8620-13f882d92a89/GREGORIO%20JUODINIS%20SANTOS%20PMR16.pdf>>. Citado na página 32.

SANTOS, G. M. P. M. dos. Monografia (Especialização em Engenharia de Produção), *A eficiência global de equipamentos (OEE) na indústria de bebidas: um estudo de caso numa linha de envase de cerveja – garrafa retornável*. Fortaleza, CE: [s.n.], 2008. Citado 2 vezes nas páginas 15 e 39.

SAÚDE, A. V. *Computação Gráfica e Processamento de Imagens*. Londrina, PR: Editora e Distribuidora Educacional S.A., 2019. Acesso em: 25 out. 2025. Disponível em: <https://cm-cls-content.s3.amazonaws.com/201901/INTERATIVAS_2_0/COMPUTACAO_GRAFICA_E_PROCESSAMENTO_DE_IMAGENS/U1/LIVRO_UNICO.pdf>. Citado na página 33.

SILVA, M. C. A. d.; GASPARIN, J. L. *A Segunda Revolução Industrial e suas influências sobre a Educação Escolar Brasileira*. 2015. Acesso em: 19 out. 2025. Disponível em: <http://www.histedbr.fe.unicamp.br/acer_histedbr/seminario/seminario7/TRABALHOS/M/Marcia%20CA%20Silva%20e%20%20Joao%20L%20Gasparin2.pdf>. Citado na página 17.

SILVA, R. F. P. Relatório de Estágio (Mestrado em Engenharia Eletrotécnica), *Um Controlador Programável Baseado em CoDeSys - Estágio na Bresimar Automação*. Coimbra, Portugal: [s.n.], 2017. Citado na página 24.

SOARES, W. S. Trabalho de Conclusão de Curso (Bacharelado em Engenharia de Controle e Automação), *Utilização de um ambiente virtual didático para o ensino de programação para CLPs*. São Paulo: [s.n.], 2021. Acesso em: 26 out. 2025. Disponível em: <<https://bdta.abcd.usp.br/directbitstream/4168de2b-0965-4777-80c1-f15942bc57f1/WESLEY%20SILVA%20SOARES%20TCC21.pdf>>. Citado na página 37.

SOUZA, E. L. da S. Monografia (Graduação em Engenharia Elétrica), *Redes de Comunicação Industrial*. Campinas, SP: [s.n.], 2010. Orientadora: Prof.^a M.^a Débora Meyhofer Ferreira. Disponível em: <<https://lyceumonline.usf.edu.br/salavirtual/documentos/1971.pdf>>, Acesso em: 22 out. 2025. Citado 3 vezes nas páginas 28, 29 e 30.

SOUZA, J. V. O. *Simulações e gêmeos digitais na Indústria 4.0: desafios e benefícios para o setor automobilístico*. Ouro Preto, MG: [s.n.], 2024. Monografia (Graduação em Engenharia de Controle e Automação). Disponível em: <<https://monografias.ufop.br/handle/35400000/7094>>. Citado na página 31.

SOUZA, L. C. Iniciação Científica – IC, *Estudo e Aplicação de Linguagens de Programação Utilizando o Software CODESYS*. Jataí, GO, Brasil: [s.n.], 2014. Citado na página 23.

SOUZA, V. d. S. Dissertação (Mestrado em Informática), *Digital Twin: Uma proposta de geração de ambientes virtuais baseado em comissionamento virtual utilizando motor gráfico Unity3D*. Manaus, AM: [s.n.], 2021. Acesso em: 26 out. 2025. Disponível em: <https://tede.ufam.edu.br/bitstream/tede/8533/5/Disserta%C3%A7%C3%A3o_ViniciusSouza_PPGI.pdf>. Citado na página 37.

TAPIA, E. et al. Assessing industrial communication protocols to bridge the gap between machine tools and software monitoring. *Sensors*, v. 23, n. 12, p. 5694, 2023. Disponível em: <<https://www.mdpi.com/1424-8220/23/12/5694>>. Citado na página 14.

TELLES, A. *Protocolos e interoperabilidade para IoT*. São Paulo: Editora Senac São Paulo, 2024. (Série Universitária). ISBN 9788539646128. Citado na página 25.

THOMAS, G. *Introduction to the Modbus Protocol*. 2008. Disponível em: <<https://www.ccontrols.com/pdf/Extv9n4.pdf>>, Volume 9, Issue 4, “Extension”, © Contemporary Control Systems, Inc.; Acesso em: 22 out. 2025. Citado na página 29.

Unity Technologies. *Unity Documentation | Docs and Guides to Work with the Unity Ecosystem*. 2025. Acesso em: 26 out. 2025. Disponível em: <<https://docs.unity.com/>>. Citado 3 vezes nas páginas 35, 36 e 37.

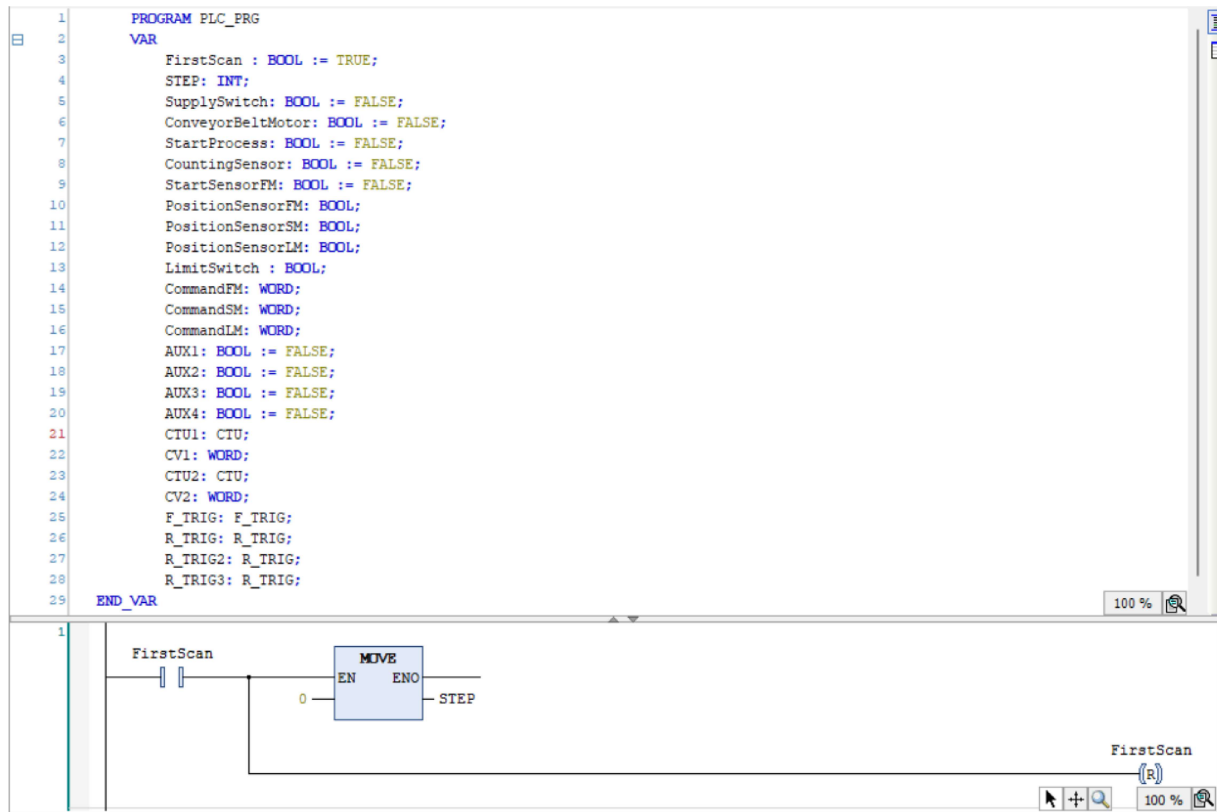
Unity Technologies. *Unity Real-Time Development Platform | 3D, 2D, VR and AR Engine*. 2025. Acesso em: 26 out. 2025. Disponível em: <<https://unity.com/>>. Citado na página 35.

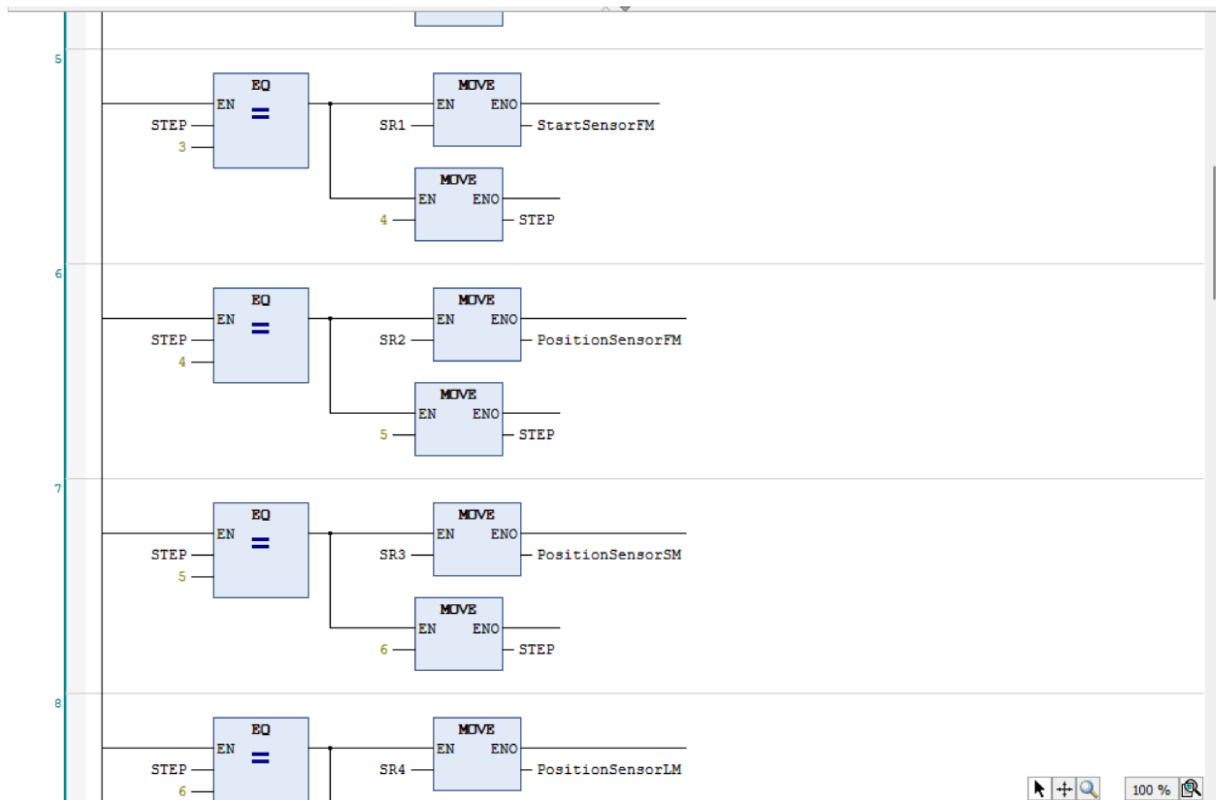
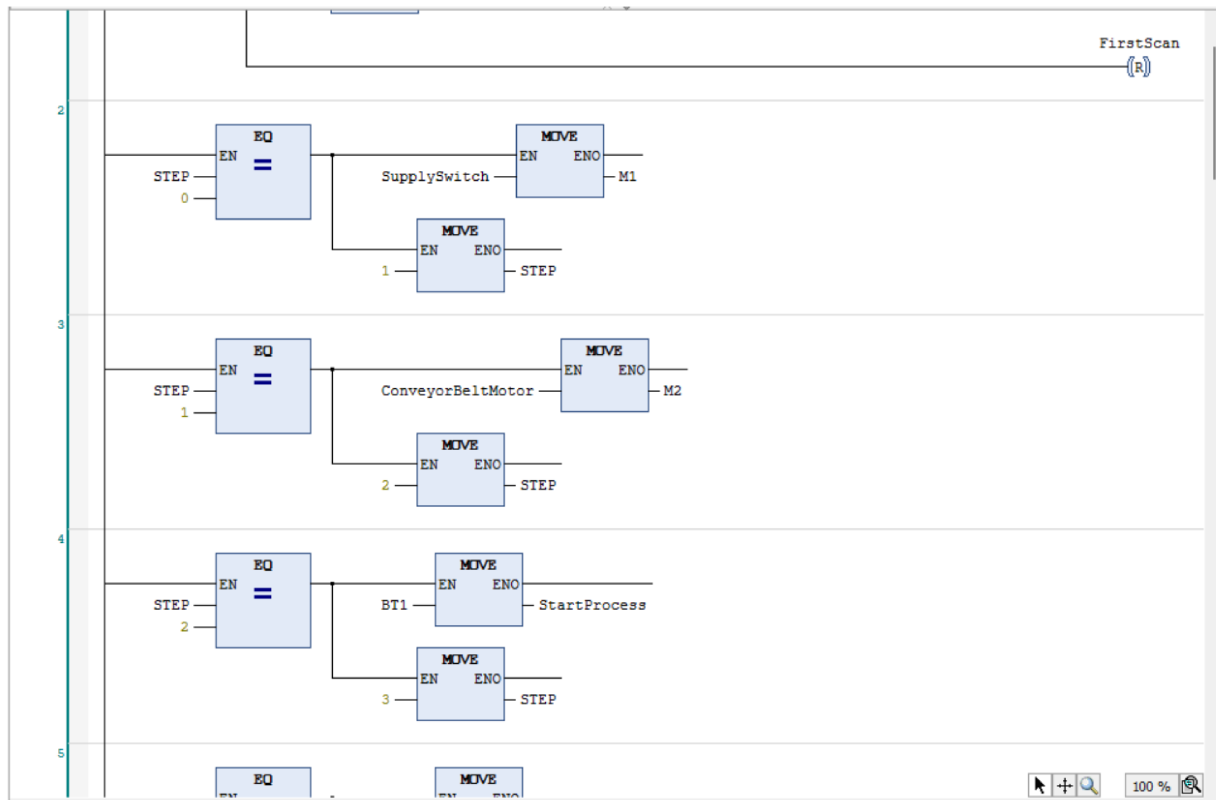
Videojet Technologies Inc. *What is a bottle filling machine?* 2025. Disponível em: <<https://www.videojet.com/us/homepage/resources/glossary/bottle-filling-machines/bottle-filling-machine.html>>. Acesso em: 26 out. 2025. Citado 2 vezes nas páginas 38 e 39.

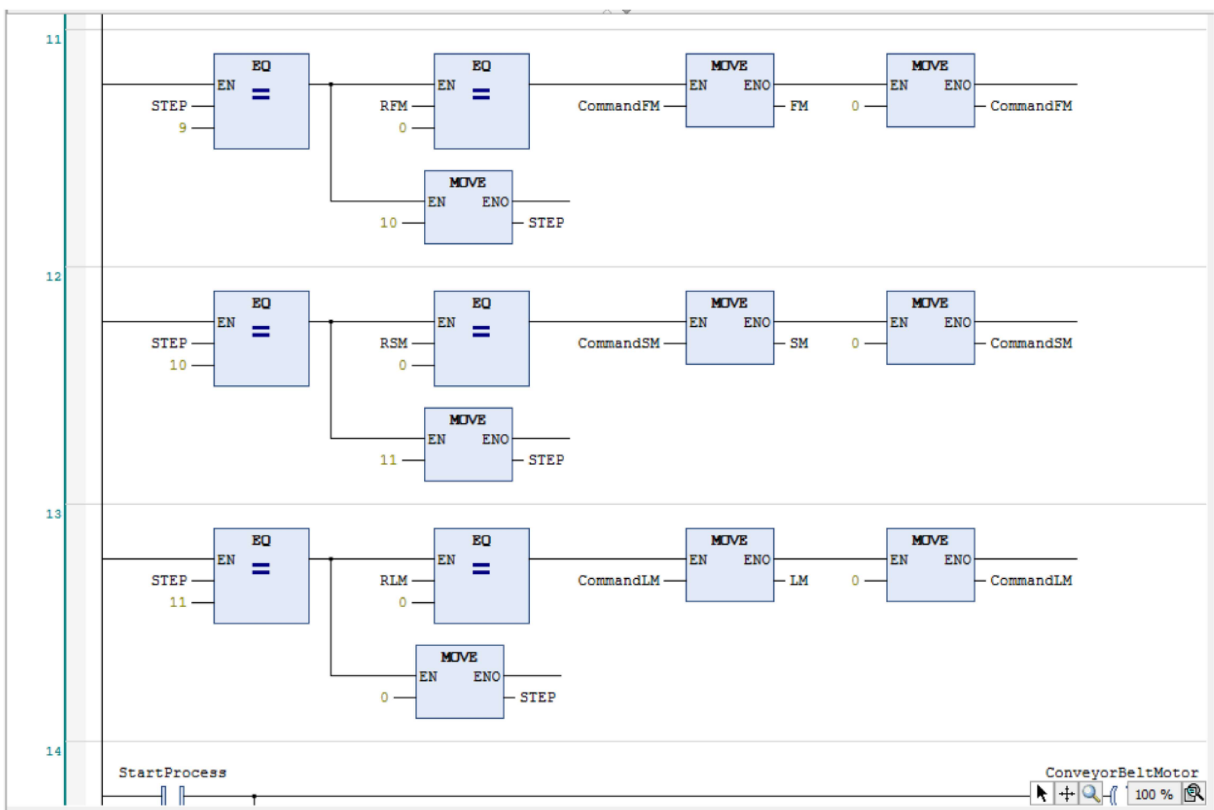
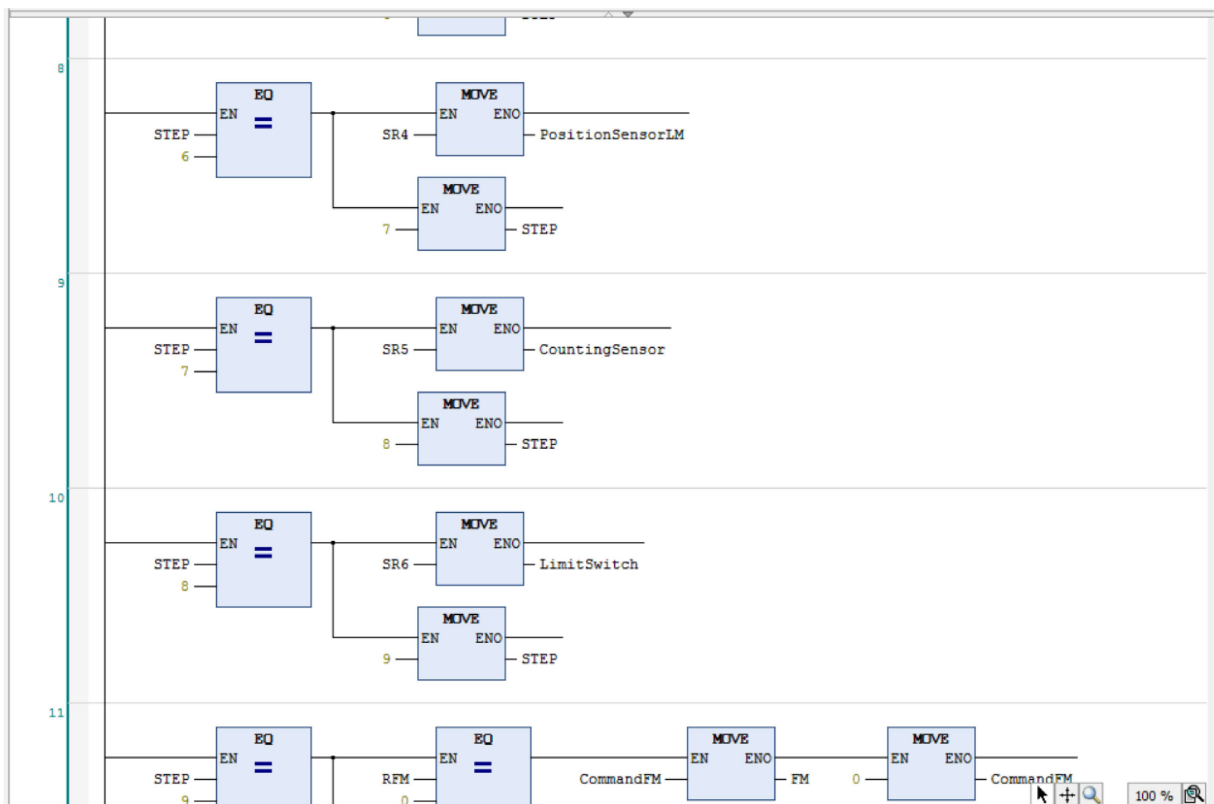
Wireshark Foundation. *Wireshark: Network Protocol Analyzer*. 2025. Disponível em: <<https://www.wireshark.org/>>. Acesso em: 10 fev. 2026. Citado na página 30.

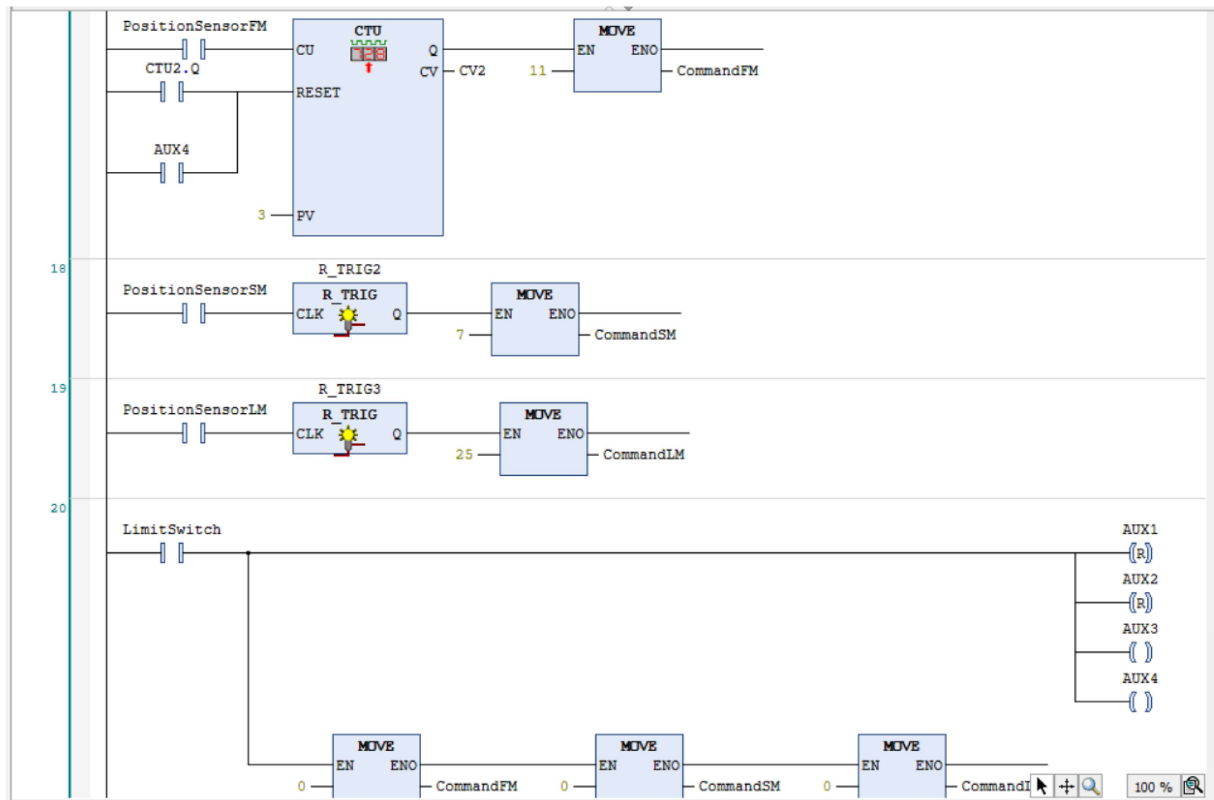
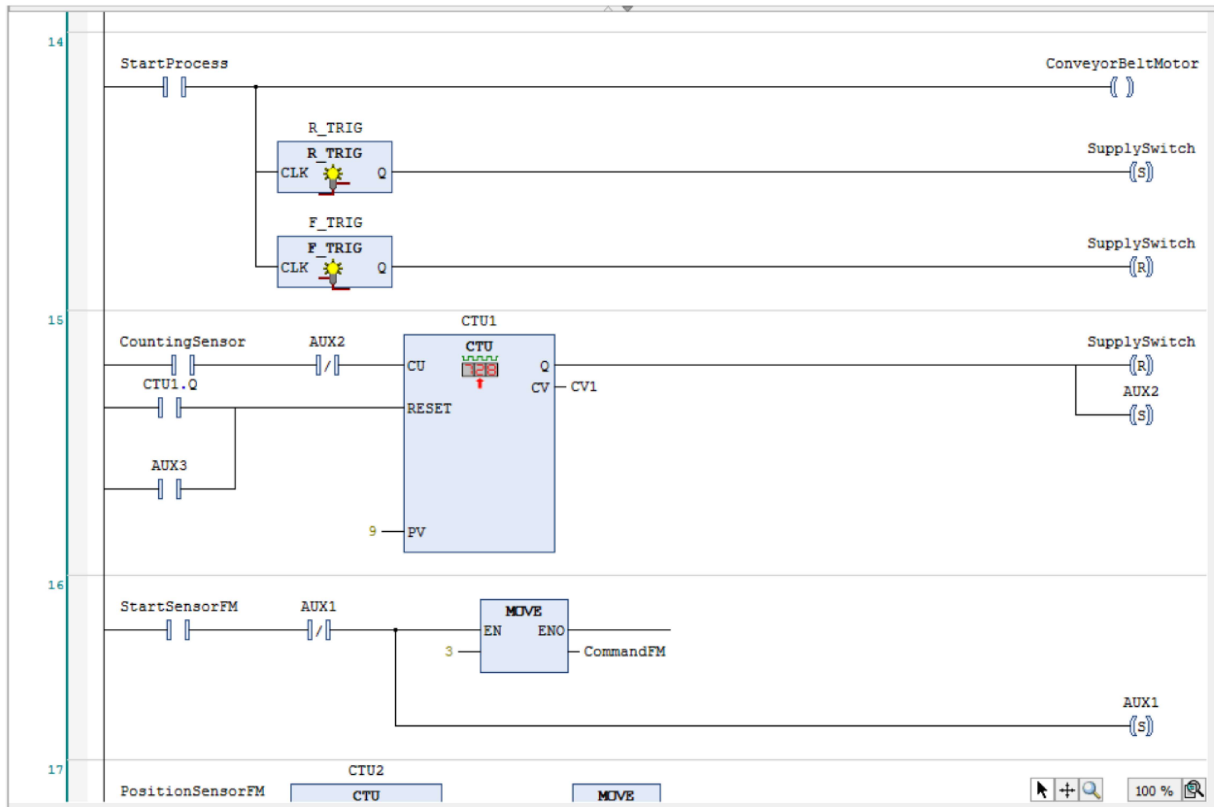
Apêndices

APÊNDICE A – Código Ladder no Codesys









APÊNDICE B – Código do Server Modbus no Unity

```
Kelly Cristina Félix Machado, last month | 1 author (Kelly Cristina Félix Machado)
1  using UnityEngine;
2  using EasyModbus;
3  using System.Net;
4
5  1 reference | Kelly Cristina Félix Machado, last month | 1 author (Kelly Cristina Félix Machado)
6  public class ModbusServerUnity : MonoBehaviour {
7
8      15 references
9      public ModbusServer modbusServer = new ModbusServer();
10
11      3 references
12      public static ModbusServerUnity InstanceModbus {get; private set;}
13      // private int coolDown = 100;
14
15      0 references
16      void Awake() {
17          if(InstanceModbus != null && InstanceModbus != this) {
18              Destroy(this);
19          } else {
20              InstanceModbus = this;
21              DontDestroyOnLoad(gameObject);
22          }
23      }
24
25      0 references
26      private void Start() {
27          try {
28              modbusServer.Port = 1502;
29              modbusServer.LocalIPAddress = IPAddress.Any;
30              modbusServer.Listen();
31          }
32          catch (System.Exception e) {
33              Debug.LogError("Erro ao iniciar o servidor Modbus: " + e.Message);
34          }
35      }
36  }
```

```
1 reference
31 public void ResetVariables()
32 {
33     modbusServer.coils[1] = false;
34     modbusServer.coils[2] = false;
35     modbusServer.coils[3] = false;
36     modbusServer.discreteInputs[1] = false;
37     modbusServer.discreteInputs[2] = false;
38     modbusServer.discreteInputs[3] = false;
39     modbusServer.discreteInputs[4] = false;
40     modbusServer.discreteInputs[5] = false;
41     modbusServer.holdingRegisters[1] = 0;
42     modbusServer.holdingRegisters[2] = 0;
43     modbusServer.holdingRegisters[3] = 0;
44 }
45
0 references
46 private void OnApplicationQuit() {
47     ResetVariables();
48     modbusServer.StopListening();
49 }
50
51 }
```

APÊNDICE C – Código da Máquina de Envase no Unity

```

Kelly Cristina Félix Machado, last month | 1 author (Kelly Cristina Félix Machado)
1  using System;
2  using System.Collections.Generic;
3  using System.Data.Common;
4  using Unity.Mathematics;
5  using UnityEngine;
6
0 references | Kelly Cristina Félix Machado, last month | 1 author (Kelly Cristina Félix Machado)
7  public class FillingMachineController : MonoBehaviour
8  {
9      [Header("Objects")]
10     11 references
11     public GameObject fillingNozzlesStructure;
12     2 references
13     public List<GameObject> fillingNozzles;
14     9 references
15     public GameObject machineGreenLed;
16     9 references
17     public GameObject machineRedLed;
18     1 reference
19     public ConveyorBeltController belt;
20     [HideInInspector]
21     4 references
22     public List<GameObject> bottlesFilling;
23
24     [Header("Parameters")]
25     3 references
26     public float speed = 10f;
27
28     [Header("Logic Variables")]
29     [HideInInspector]
30     4 references
31     public bool isActive;
32     3 references
33     public bool isFollowing;
34     3 references
35     public bool isReturning;
36     [HideInInspector]

```

```
27      6 references
28      public bool isOnTarget;
29
30      [Header("Effects")]
31      1 reference
32      public GameObject juiceVFX;
33      4 references
34      private GameObject juiceVFXObject;
35      4 references
36      private List<GameObject> juiceVFXObjects;
37      1 reference
38      public AudioClip audioOnFilling;
39      6 references
40      private bool isFirstPlaySound = true;
41      7 references
42      private AudioSource audioSource;
43      // [Header("ModBus Helper")]
44      // Start is called before the first frame update
45      0 references
46      void Start()
47      {
48          audioSource = GetComponent<AudioSource>();
49          bottlesFilling = new List<GameObject>();
50          juiceVFXObjects = new List<GameObject>();
51          isOnTarget = false;
52          ModbusServerUnity.InstanceModbus.modbusServer.holdingRegisters[1] = 0;
53      }
54
55      // Update is called once per frame
56      0 references
57      void Update(){
58
59          if (GameManager.Instance.isGameOver){
60              audioSource.Stop();
61              return;
62          }
63      }
```

```

55         if (!GameManager.Instance.isIdealSimulation)
56         {
57             if (ModbusServerUnity.InstanceModbus.modbusServer.holdingRegisters[1] == 3)
58             {
59                 isActive = true;
60             }
61             else if (ModbusServerUnity.InstanceModbus.modbusServer.holdingRegisters[1] == 11)
62             {
63                 isFollowing = true;
64             }
65         }
66
67         if (isActive)
68         {
69             if (isFirstPlaySound && GameManager.Instance.isAudioOn){
70                 audioSource.Play();
71                 isFirstPlaySound = false;
72             }
73             InitialPosition();
74         }
75         if(isFollowing){
76             if (isFirstPlaySound && GameManager.Instance.isAudioOn){
77                 audioSource.Play();
78                 audioSource.PlayOneShot(audioOnFilling);
79                 isFirstPlaySound = false;
80             }
81             FollowingBottles();
82         }
83         if(isReturning){
84             ReturnPositionY();
85         }
86     }
87 }
88

```

```

89 1 reference
90 public void InitialPosition(){
91     if (fillingNozzlesStructure != null)
92     {
93         var targetPosition = new Vector3(0,0,0.18f);
94
95         if (fillingNozzlesStructure.transform.localPosition.z >= targetPosition.z) {
96             isActive = false;
97             audioSource.Stop();
98             isFirstPlaySound = true;
99             ModbusServerUnity.InstanceModbus.modbusServer.holdingRegisters[1] = 0;
100             machineGreenLed.SetActive(false);
101             machineRedLed.SetActive(true);
102             return;
103         }
104
105         if(machineGreenLed != null && machineRedLed != null){
106             machineGreenLed.SetActive(true);
107             machineRedLed.SetActive(false);
108         }
109
110         fillingNozzlesStructure.transform.localPosition = new Vector3(0, 0, fillingNozzlesStructure.transform.localPosition.z+(speed * Time.deltaTime));
111     }
112 }
113
114
115
116 1 reference
117 public void FollowingBottles(){
118     if(machineGreenLed != null && machineRedLed != null){
119         machineGreenLed.SetActive(true);
120         machineRedLed.SetActive(false);
121     }
122 }

```

```

122
123     if(!isOnTarget){
124         foreach (GameObject obj in fillingNozzles)
125         {
126             if (obj != null)
127             {
128
129                 var targetPosition = new Vector3(obj.transform.localPosition.x,-0.145f,obj.transform.localPosition.z);
130                 obj.transform.localPosition = new Vector3(0, obj.transform.localPosition.y (speed * Time.deltaTime), obj.transform.localPosition.z);
131
132                 if (obj.transform.localPosition.y <= targetPosition.y) {
133                     isOnTarget = true;
134                     juiceVFXobject = Instantiate(juiceVFX);
135                     juiceVFXobject.transform.SetParent(obj.transform,false);
136                     juiceVFXobject.transform.localPosition = new Vector3(obj.transform.localPosition.x,1.275f,obj.transform.localPosition.z*1.35f);
137                     juiceVFXobjects.Add(juiceVFXobject);
138                 }
139             }
140         }
141     }
142 }
143
144 if(isOnTarget){
145     FillingBottles();
146 }
147
148 if (fillingNozzlesStructure != null)
149 {
150
151     var targetPosition = new Vector3(fillingNozzlesStructure.transform.localPosition.x,fillingNozzlesStructure.transform.localPosition.y, 0.18f);
152     fillingNozzlesStructure.transform.localPosition = new Vector3(0, 0, fillingNozzlesStructure.transform.localPosition.z -(belt.GetCurrentSpeed()*Time.deltaTime));
153

```

```

154         if (fillingNozzlesStructure.transform.localPosition.z <= targetPosition.z) {
155             ModbusServerUnity.InstanceModbus.modbusServer.holdingRegisters[1] = 0;
156             isFollowing = false;
157             audioSource.Stop();
158             isFirstPlaySound = true;
159             machineGreenLed.SetActive(false);
160             machineRedLed.SetActive(true);
161         } else if (fillingNozzlesStructure.transform.localPosition.z <= targetPosition.z+0.04){
162             isReturning = true;
163         }
164     }
165 }
166
167 }
168
169 1 reference
170 public void ReturnPositionY(){
171
172     RemoveBottles();
173
174     if(machineGreenLed != null && machineRedLed != null){
175         machineGreenLed.SetActive(true);
176         machineRedLed.SetActive(false);
177     }
178
179     if(isOnTarget){
180         foreach (GameObject obj in fillingNozzles)
181         {
182             if (obj != null)
183             {
184
185                 var targetPosition = new Vector3(obj.transform.localPosition.x,0,obj.transform.localPosition.z);
186                 obj.transform.localPosition = new Vector3(0, obj.transform.localPosition.y+(speed * Time.deltaTime), obj.transform.localPosition.z);

```

```
187         if (obj.transform.localPosition.y >= targetPosition.y) {
188             machineGreenLed.SetActive(false);
189             machineRedLed.SetActive(true);
190             isOnTarget = false;
191             isReturning = false;
192             isActive = true;
193         }
194     }
195 }
196 }
197 }
198
199 }
200
201 1 reference
202 public void FillingBottles(){
203     foreach (GameObject bottle in bottlesFilling)
204     {
205         if (bottle != null)
206         {
207             if(bottle.activeSelf == false){
208                 bottle.SetActive(true);
209             }
210
211             Vector3 scale = bottle.transform.localScale;
212             scale.y += 1f * Time.deltaTime;
213
214             if(bottle.transform.localScale.y <= 1){
215                 bottle.transform.localScale = scale;
216             }
217         }
218     }
219 }
220
```



```
221     }
222
223     1 reference
224     public void RemoveBottles()
225     {
226         for (int i = bottlesFilling.Count - 1; i >= 0; i--)
227         {
228             bottlesFilling.RemoveAt(i);
229         }
230
231         if(juiceVFXObjects != null){
232             foreach (GameObject effect in juiceVFXObjects){
233                 Destroy(effect);
234             }
235         }
236     }
237
238 }
239
```

APÊNDICE D – Código da Máquina de Selagem no Unity

```
You, last month | 2 authors (Kelly Cristina Félix Machado and one other)
1  using UnityEngine;
2  using System.Collections.Generic;
3
0 references | Kelly Cristina Félix Machado, last month | 1 author (Kelly Cristina Félix Machado)
4  public class LabelingSensor : MonoBehaviour
5  {
6      3 references
      public List<GameObject> bottlesLabeling;
7      4 references
      private bool isFirstPlaySound = true;
8      6 references
      private AudioSource audioSource;
9      // Start is called before the first frame update
10     0 references
      void Start()
11     {
12         audioSource = GetComponent<AudioSource>();
13     }
14
15     // Update is called once per frame
16     0 references
      void Update()
17     {
18         if (GameManager.Instance.isGameOver){
19             audioSource.Stop();
20             return;
21         }
22
23         if (GameManager.Instance.startProcess && isFirstPlaySound && audioSource!=null && GameManager.Instance.isAudioOn)
24         {
25             isFirstPlaySound = false;
26             audioSource.Play();
27         }
28         else if (!GameManager.Instance.startProcess && !isFirstPlaySound && audioSource!=null && GameManager.Instance.isAudioOn)
29         {
30             isFirstPlaySound = true;
```

```
31         audioSource.Stop();
32     }
33
34     if (!GameManager.Instance.isIdealSimulation)
35     {
36         if (ModbusServerUnity.InstanceModbus.modbusServer.holdingRegisters[3] == 25)
37         {
38             foreach(GameObject bottle in bottlesLabeling){
39                 bottle.transform.Find("Label").gameObject.SetActive(true);
40             }
41             ModbusServerUnity.InstanceModbus.modbusServer.holdingRegisters[3] = 0;
42         }
43     }
44 }
45
46 0 references
47 void OnTriggerEnter(Collider other)
48 {
49     if (GameManager.Instance.isGameOver){
50         return;
51     }
52
53     if(other.CompareTag("Bottle")){
54         if (GameManager.Instance.isIdealSimulation)
55         {
56             other.gameObject.transform.Find("Label").gameObject.SetActive(true);
57         }
58         else
59         {
60             ModbusServerUnity.InstanceModbus.modbusServer.discreteInputs[4] = true;
61             bottlesLabeling.Add(other.gameObject);
62         }
63     }
64 }
```

```
0 references
65 void OnTriggerExit(Collider other)
66 {
67     if (GameManager.Instance.isGameOver){
68         return;
69     }
70
71     if (!GameManager.Instance.isIdealSimulation)
72     {
73         if (other.CompareTag("Bottle"))
74         {
75             ModbusServerUnity.InstanceModbus.modbusServer.discreteInputs[4] = false;
76             bottlesLabeling.Remove(other.gameObject);
77         }
78     }
79 }
80
81 }
82 }
```

APÊNDICE E – Código da Máquina de Rotulagem no Unity

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  0 references | Kelly Cristina Félix Machado, last month | 1 author (Kelly Cristina Félix Machado)
6  public class SealingMachineController : MonoBehaviour
7  {
8      13 references
9      public GameObject sealingNozzleStructure;
10     2 references
11     public float speed = 10f;
12     [HideInInspector]
13     4 references
14     public bool isOnTarget;
15     4 references
16     private bool isOnInitialPosition;
17     4 references
18     private bool isSealing;
19     4 references
20     private bool isMachineOn;
21     3 references
22     private float coolDown = 20;
23     1 reference
24     public AudioClip audioOnShot;
25     7 references
26     private AudioSource audioSource;
27     4 references
28     private bool isFirstPlaySound = true; Kelly Cristina Félix Machado, last month • first version finished
29
30     0 references
31     void Start()
32     {
33         audioSource = GetComponent<AudioSource>();
34         isOnTarget=false;
35         isOnInitialPosition=true;
36         isSealing = false;
37         isMachineOn = false;
38         ModbusServerUnity.InstanceModbus.modbusServer.holdingRegisters[2] = 0;
39     }

```

```

29     void Update()
30     {
31         if (GameManager.Instance.isGameOver){
32             audioSource.Stop();
33             return;
34         }
35
36         if (GameManager.Instance.startProcess && isFirstPlaySound && audioSource!=null && GameManager.Instance.isAudioOn)
37         {
38             isFirstPlaySound = false;
39             audioSource.Play();
40         }
41         else if (!GameManager.Instance.startProcess && !isFirstPlaySound && audioSource!=null && GameManager.Instance.isAudioOn)
42         {
43             isFirstPlaySound = true;
44             audioSource.Stop();
45         }
46
47         if (!GameManager.Instance.isIdealSimulation)
48         {
49             if (ModbusServerUnity.InstanceModbus.modbusServer.holdingRegisters[2] == 7 && isMachineOn == false)
50             {
51                 isOnTarget = true;
52                 if (GameManager.Instance.isAudioOn)
53                 {
54                     audioSource.PlayOneShot(audioOnShot);
55                 }
56                 ModbusServerUnity.InstanceModbus.modbusServer.holdingRegisters[2] = 0;
57             }
58         }
59
60         SealingBottles();
61     }
62

```

```

63     public void SealingBottles(){
64         if (sealingNozzleStructure != null){
65             if(isOnTarget){
66                 isMachineOn = true;
67                 var targetPosition = new Vector3(sealingNozzleStructure.transform.localPosition.x,-0.11f,sealingNozzleStructure.transform.localPosition.z);
68
69                 if (sealingNozzleStructure.transform.localPosition.y <= targetPosition.y) {
70                     isOnTarget = false;
71                     isSealing = true;
72                 }
73
74                 sealingNozzleStructure.transform.localPosition = new Vector3(0, sealingNozzleStructure.transform.localPosition.y-(speed * Time.deltaTime),
75                 sealingNozzleStructure.transform.localPosition.z);
76             }
77             if(isSealing){
78                 coolDown--;
79                 if(coolDown<=0){
80                     coolDown = 20;
81                     isSealing = false;
82                     isOnInitialPosition = false;
83                 }
84             }
85             if(isOnInitialPosition==false){
86                 var initialPosition = new Vector3(sealingNozzleStructure.transform.localPosition.x,0,sealingNozzleStructure.transform.localPosition.z);
87                 sealingNozzleStructure.transform.localPosition = new Vector3(0, sealingNozzleStructure.transform.localPosition.y+(speed * Time.deltaTime),
88                 sealingNozzleStructure.transform.localPosition.z);
89
90                 if (sealingNozzleStructure.transform.localPosition.y >= initialPosition.y) {
91                     isOnInitialPosition = true;
92                     isMachineOn = false;
93                 }
94             }
95         }
96     }
97 }

```