Wesley Silva Soares

Utilização de um ambiente virtual didático para o ensino de programação para CLPs

Brasil

2021, v-0.1

Wesley Silva Soares

Utilização de um ambiente virtual didático para o ensino de programação para CLPs

Trabalho de conclusão de curso apresentado à Escola Politécnica da Universidade de São Paulo, a fim de obter o título de Engenheiro Mecatrônico

Universidade de São Paulo – USP

Escola Politécnica

Orientador: Prof. Dr. Marcos Ribeiro Pereira Barretto

Brasil

2021, v-0.1

Catalogação-na-publicação

Soares, Wesley
Utilização de um ambiente virtual didático para o ensino de programação para CLPs / W. Soares -- São Paulo, 2021.
50 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1.Treinamento para CLPs 2.Fábrica virtual I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos II.t.

Wesley Silva Soares

Utilização de um ambiente virtual didático para o ensino de programação para CLPs

Trabalho de conclusão de curso apresentado à Escola Politécnica da Universidade de São Paulo, a fim de obter o título de Engenheiro Mecatrônico

---

Prof. Dr. Marcos Ribeiro Pereira Barretto
Orientador

Professor
Convidado 1

Professor
Convidado 2

Brasil
2021, v-0.1

À Aline, por ser a pessoa a quem devo tudo,
ao Jorge, por ser o meu exemplo,
Pamela por estar ao meu lado,
Dedico este trabalho.

Agradecimentos

Primeiramente, agradeço à Deus.

Aos Professores e funcionários do departamento de Engenharia Mecatrônica da Escola Politécnica, principalmente ao professor Marcos Barretto pela orientação no trabalho. E ao funcionário Cássio pela disponibilidade e atenção quando precisei.

Agradeço, também, aos meus amigos de longa data, pelo apoio e incentivo durante a criação deste trabalho.

Resumo

Cada vez mais, processos industriais utilizam alguma forma de automação. O uso de automação na indústria aumenta a eficiência dos processos, reduzindo custos e uso de materiais, além de reduzir riscos aos operadores. Para isso, a formação de um profissional da área deve ser sólida, abrangendo satisfatoriamente os conceitos relevantes ao tema, como programação de CLPs, Controladores Lógico Programáveis, por exemplo. Atualmente, o ensino pode ser feito por meio de aparatos físicos em laboratório, que possuem alto custo financeiro, além de necessitar que o aluno esteja presente no local do experimento. Outra abordagem é a utilização de ferramentas comerciais de simulação que facilitam o acesso dos alunos à tipos diferentes cenários de treinamento possíveis, porém, no contexto didático, também tem custo elevado de aquisição. Visto isso, neste trabalho é proposta uma solução que pode auxiliar o ensino de automação industrial, de forma virtual e que sua aplicação seja economicamente viável. Que simula cenas compostas por componentes comuns ao ambientes de produção industrial. Para isso, foi utilizada a plataforma de desenvolvimento de simulações 3D *Unity* e comunicação com controladores externos por meio do protocolo *modbus*.

Palavras-chaves: Treinamento para CLPs. Fábrica virtual. Ambiente de aprendizagem.

Abstract

More and more, industrial processes use some type of automation. The use of industrial automation raises the process efficiency, reducing costs and material needs, in addition to reducing operating health risks. For that, the formation of a professional in automation must be solid, covering the subject relevant concepts, like, as an example, PLC (Programmable Logic Controller) programming, in a satisfactory way. Nowadays, the learning can be done using laboratory physical components, that have high costs, in addition to the need that the students have to be in the laboratory. Another approach consists in using commercial simulation tools, making the student access easier to more possible training scenarios, however, in a didactic context, also have a high implementation cost. With that said, in this paper is proposed a solution that can be used as an auxiliary in the teaching of industrial automation, virtually and being cost effective. Simulating scenes that have virtual components commonly seen in industrial contexts. For that, the 3D development engine Unity was used along with the Modbus protocol in order to communicate with external controllers.

Key-words: PLC Training. Virtual factory. Learning environment.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CLP	Controlador Lógico Programável
LD	*Ladder Diagram*
FBD	*Function Block Diagram*
OPC	*Open Platform Communications*
SFC	*Sequential Function Chart*
ST	*Structured Text*

Sumário

1 Introdução

1.1 Motivação

A indústria está cada vez mais ligada à processos que aplicam alguma forma de automação. O aumento da eficiência proporcionado por um controle automatizado oferece redução do custo financeiro, melhora da qualidade de vida dos funcionários e oferece redução do impacto ambiental, como redução do uso de insumos e de energia. A automatização de uma planta pode substituir máquinas antigas, pouco eficientes energeticamente, por outras mais modernas que necessitam de menos energia e com maior produtividade. Atividades repetitivas, executadas por pessoas, são substituídas por máquinas capazes de trabalhar com mais velocidade e repetibilidade, evitando que pessoas sejam expostas à potenciais danos, como lesões por esforço (1). Além disso, o processo automatizado aumenta o nível de segurança de uma planta industrial, pois pode gerar alertas com antecedência e executar procedimentos de emergência caso algo esteja fora de parâmetros seguros de operação previamente estabelecidos. Para isso, uma tecnologia largamente utilizada são os CLPs, Controladores Lógicos Programáveis, dispositivos que, de acordo com as variáveis de entrada, executam rotinas pré estabelecidas e geram saídas para controlar um sistema.

Dito isso, para que a formação de um engenheiro que atua na área de automação seja satisfatória, é necessário que se tenha familiaridade com essas tecnologias.

Em uma enquete com 48 alunos da Escola Politécnica da USP onde 34 informaram ser da engenharia mecatrônica, 6 da área elétrica e 8 de outras ênfases. Destes alunos, 31,1% estavam no sétimo semestre do curso.

O questionário foi uma autoavaliação sobre o quão familiarizado o aluno se sentia com linguagens de programação que, segundo o índice TIOBE (2), possuem alta popularidade. As respostas poderiam ser: Desconheço; Conheço, mas nunca tive contato; Básico; Intermediário; Avançado. A Tabela 1 apresenta a porcentagem das respostas que expressaram alguma forma de familiaridade, ou seja, básico, intermediário ou avançado. Em seguida, foi feito o mesmo questionamento, mas relacionado à linguagens de programação utilizadas em CLPs. O resultado mostrado na Tabela 2.

Pode-se notar que Python e C/C++ são as linguagens de melhores resultados em familiaridade com os alunos, sendo estas usadas em matérias ao longo do curso de engenharia. Já, em relação às utilizadas em CLPs, o quadro se inverte e, a maior parte dos que responderam, afirmam que desconhecem estas linguagens.

Tabela 1 – Resumo das respostas sobre linguagens de programação

Linguagem	**Familiaridade**
Python	93.75%
Java	35.42%
Javascript	31.25%
C/C++	81.25%
C#	22.92%
VBA	31.25%
PHP	12.50%
Ruby	33.33%

Fonte: Autor

Tabela 2 – Resumo das respostas sobre linguagens usadas em CLPs

Linguagem	**Familiaridade**
Ladder Diagram	60.42%
Sequantial Flow Charts	45.83%
Function Block Diagram	20.83%
Structured Text	18.75%
Instruction List	14.58%

Fonte: Autor

Atualmente, na Escola Politécnica da USP, para aplicar os conceitos de programação para CLPs, são utilizadas as bancadas didáticas da Festo. Porém, não é possível a sua utilização fora do período de aula, sendo necessário o acompanhamento de um monitor. Visto isso, foi proposta a viabilidade da utilização de um ambiente virtual, contendo elementos de fábrica que interagem com comandos de um controlador, podendo replicar aspectos das bancadas físicas, por exemplo.

Por não necessitar estar em laboratório, um ambiente virtual é mais acessível à alunos interessados em ter mais contato com os conceitos expostos em aula, também possui maior flexibilidade na modificação de cenários onde soluções podem ser aplicadas e, além disso, elimina o disco de danos, por não possuir componentes físicos.

1.2 Objetivo

Este trabalho tem o objetivo de montar um ambiente virtual de uma fábrica onde seja possível simular didaticamente a programação de CLPs para controlar processos de fabricação. Similar, às bancadas didáticas presentes em laboratório, porém com mais flexibilidade de layout de processos e de acesso por não ser necessário estar presente fisicamente para testar o funcionamento. Como prova de conceito, serão montados dois cenários simulando componentes observados em fábricas, como sensores, esteiras e atuadores. Sendo possível que um programa externo a este ambiente seja capaz de interagir com os componentes por meio de um protocolo de comunicação industrial, não sendo necessariamente possível que este programa "saiba"que não está interagindo com componentes reais.

2 Revisão Bibliográfica

Visto que o objetivo do trabalho é de propor uma alternativa às limitações inerentes ao ensino de programação para CLPs, quando se utiliza somente dispositivos físicos, como o alto custo dos equipamentos (3) e a necessidade de se estar presente no laboratório, uma solução possivelmente viável é uma ferramenta que possibilite a simulação de um ambiente físico; Possa se comunicar com um programa de controle externo em tempo real e que seja de baixo custo de implantação. Essas características foram resumidas como "Treinamento para CLPs", "Fábrica virtual"e "Ambiente de aprendizagem". As palavras-chave *"PLC training"*, *"virtual factory"*, e *"learning environment"* foram utilizadas nas ferramentas de busca de artigos científicos *Scopus*, *Google Scholar* e *IEEE Explorer* entre Março e Julho de 2021, e os artigos apresentados neste trabalho foram selecionados por apresentarem conceitos relevantes ao tema.

2.1 Trabalhos Relacionados

Segundo,(3), "No contexto da Industria 4.0, um dos maiores desafios é como verificar e validar complexos sistemas automatizados de manufatura, baseados em novas tecnologias de fabricas inteligentes". Para eles, por ser geralmente feito nas plantas reais, os teste com CLPs podem gerar incidentes que vão de atrasos no andamento projeto a danos aos equipamentos da linha, levando a prejuízos financeiros. Com isso, foi feito o estudo de uma ferramenta de comissionamento virtual, que funcionasse de forma que a comunicação com CLP fosse feita em tempo real, a fim de reduzir tempo e custos. "Comissionamento virtual consiste em replicar o comportamento de um ambiente físico de manufatura usando um sistema de software com o objetivo de providenciar um ambiente virtual, proporcionando que engenheiros de automação/robótica possam validar suas lógicas de automação (CLP ou robô controlador) e HMI (Interface Homem Máquina) antes do comissionamento"Tebani et al.(3).

O estudo apresenta quatro abordagens que podem ser feitas num comissionamento de uma planta automatizada. A primeira é testar o funcionamento do CLP utilizando diretamente o controlador real com os equipamentos da planta, situação que pode causar danos aos componentes físicos, caso haja algum erro no programa carregado no controlador. A segunda é chamada de *"Software in the Loop"*, onde um controlador virtual atua sobre equipamentos também virtuais, que segundo o estudo, é útil para a verificação da robustez do sistema de controle, porém, como não contém todas as características do sistema real, os resultados dos testes não são tão confiáveis comparativamente por não sem em tempo real. A terceira abordagem consiste em testar o CLP real em um ambiente virtual a fim de testar

o comportamento do controlador antes da sua implementação real. Finalmente, a quarta abordagem é testar o software de controle em um CLP virtual, ligado aos componentes da planta real. Esta ultima abordagem leva à riscos parecidos com a primeira abordagem, por expor os equipamentos da planta à um programa que não foi completamente validado.

O estudo também expõe o conceito de Gêmeo Digital, *Digital Twin*, onde é feito um modelo virtual que simula o comportamento de um sistema físico real. Com isso, dados obtidos no modelo virtual são validados no sistema real, que por sua vez os resultados são utilizados para aprimorar o modelo. Foi proposta uma abordagem onde um ambiente virtual, simulando uma planta real, comunicava-se com um CLP físico a fim de testar o comissionamento do software de controle. Com isso, reduzir o tempo do comissionamento real. Para isso, foi usado a ferramenta de modelagem *Dymola* para replicar os componentes físicos, e suas propriedades relevantes à automação, de uma máquina de colocar tampas em frascos de perfume. Em seguida, o software de controle foi carregado em um CLP físico que fazia a comunicação com o modelo. Esta comunicação foi feita por meio do protocolo TCP/IP, onde o programa do CLP decodificava para decimal os dados em ASCII vindos do modelo.

O experimento de Tebani et al.(3) faz, de maneira bem sucedida, a comunicação em tempo real entre um modelo virtual que simula uma entidade física e um CLP real. O CLP da solução proposta por este trabalho será virtual, porém há a necessidade da comunicação em tempo real para que a simulação seja satisfatória.

Para a praticar do uso de CLPs, pesquisadores da *Center of Tongji University* (4), China, fizeram o uso da simulação tanto do CLP, quanto do ambiente. Tendo como resultado uma ferramenta de simulação viável e de menor custo comparada à uma solução física. Para isso, o CLP virtual utilizado foi o PLCSim, programa de simulação de CLP criado pela Siemens, sendo este capaz de executar programas criados para diversos controladores da empresa (5), facilitando o desenvolvimento de projetos por diminuir a necessidade de que cada vez que seja necessário testar a implementação de uma funcionalidade, o programa de controle tenha de ser carregado para a memória de um dispositivo físico. O ambiente virtual simulava um cruzamento contendo semáforos em cada uma de suas direções. A construção do ambiente foi feita utilizando a plataforma *Unity*(6), que será abordada adiante neste trabalho. No ambiente virtual, cada direção do cruzamento possuía um semáforo, onde as luzes eram controladas pelo CLP virtual. As lâmpadas e os botões de *Start* e *Stop* foram associados cada um a um endereço de I/O. Os acionamentos vinham do CLP por meio do protocolo de comunicação OPC e interpretados dentro do ambiente da *Unity*.

O artigo mostrado é relevante ao trabalho por manter o ambiente e o controlador como dispositivos virtuais comunicando-se em tempo real, porém a aplicação apresentada, mesmo explorando os conceitos de controle de entradas e saídas, não é um cenário de

produção de uma fábrica.

No artigo de Vaananen, Horelli e Katajisto(7), é dito que a melhor maneira de se ensinar os conceitos de programação para CLPs, é utilizando dispositivos reais, porém estes possuem alto preço para ser comprado e poucas possibilidades de alterações de cenário. Uma alternativa viável é a utilização de ferramentas de simulação, que são mais versáteis em relação à possibilidades de cenários e complexidades, além de que "instrumentos virtuais podem sempre ser operados em um modo básico e o treinamento ser recomeçado após um possível distúrbio" Vaananen, Horelli e Katajisto(7) e num momento posterior, ser apresentado os equipamentos reais. Foi criada uma linha de montagem virtual, análoga a um modelo real disponível fisicamente, utilizando a ferramenta de desenvolvimento de ambientes 3D OGRE, *Object-Oriented Graphics Rendering Engine*.O ambiente virtual poderia se comunicar com um *softPLC* contendo o programa de controle. Além da planta industrial, utilizando o conceito de *Hardware in the loop*, onde a resposta de hardware para o controlador é feita por meio de simulação, foi criada uma simulação para automação residencial contendo sensores como de gás carbônico e fumaça, detecção de vazamentos, controle de temperatura e de luzes. O modelo simulava tanto os sistemas dinâmicos como a variação de temperatura de um ambiente, como o comportamento de moradores interagindo com os itens da casa e mudando os estados dos equipamentos, como ligar ou desligar aparelhos.

Segundo o experimento do *Vidyalankar Institute of Technology* (8), Mumbai, India, ensinar os conceitos de CLPs, sem que seja possível que os alunos testem na prática os conceitos ensinados em aula, é uma tarefa difícil, "As principais dificuldades no aprendizado de programação para CLP utilizando diagramas *ladder* são o entendimento dos conceitos e visualização sobre o uso de relês, contatos, chaves etc" (Narayanan e Deshpande(8)). Além disso, afirma que os kits didáticos para laboratório possuem uma estrutura fixa, o que limita o numero de problemas propostos. Com isso, "Além de possibilitar o ajuste do currículo convencional do curso, um laboratório virtual possibilita que estudantes desenvolvam seus próprios programas e identifiquem problemas relacionados à implementação" (Narayanan e Deshpande(8)). Nesse estudo, foi desenvolvida uma ferramenta chamada *Virtual Lab*, contendo um simulador de *Ladder Diagram*, onde foi possível estudar tanto os conceitos de software e hardware de um CLP. Os experimentos foram montados de forma a introduzir os conceitos de programação para CLPs, como operações lógicas com os contatos de entrada e uso de contadores e *timers*, de forma gradativa e intuitiva.

Foi aplicado um questionário para 120 professores de engenharia que fizeram o curso no laboratório, a fim de avaliar a ferramenta utilizada. De maneira geral, foi visto que a ferramenta obteve uma aceitação positiva, porém menos de metade das respostas foram positivas sobre a comparação dos experimentos virtuais com o mundo real. Algumas características da ferramenta foram notadas como, por exemplo, aspectos de segurança em

ambiente real não considerados no ambiente virtual proposto. Foi notada a possibilidade dos experimentos virtuais ser mostrados em aula e exercícios pedidos como dever de casa. Os professores, em seguida, utilizaram a ferramenta em aulas de laboratório e foi feito outro questionário com os alunos, tendo também um resultado positivo. Os alunos relataram a falta de uma reposta mais realista dos componentes. "Eles tiveram a sensação de que se sistemas reais como motores, relês, chaves de contato etc, fossem animados e incluídos, a experiência teria sido mais animadora" (Narayanan e Deshpande(8)).

Como resultado, foi visto que os alunos obtiveram alto interesse nas tarefas propostas e no uso dos materiais de apoio disponíveis. Também notou-se que após o uso da ferramenta, o nível de entendimento dos alunos em relação aos conceitos do curso foi superior comparado aos anos anteriores.

Neste outro estudo da *Claude Bernard University Lyon 1*, França, foi proposto o uso de um ambiente virtual para o ensino de conceitos da indústria 4.0. Para isso, foi utilizado o *FactoryIO* (9), ferramenta de simulação de ambiente de fábrica, dispondo de dispositivos virtuais que simulam equipamentos industriais como esteiras, sensores, atuadores e robores industriais. O *FactoryIO* possui uma interface amigável de um ambiente 3D. O controle dos equipamentos virtuais pode ser feito tanto por uma interface interna, por meio de rotinas programadas em FDB, quanto por meio de dispositivos externos, como CLPs físicos ou virtuais. Foi o utilizado o *CodeSys* como ferramenta de desenvolvimento para automação. O *Codesys* possibilita tanto escrever, quanto executar programas em uma série de linguagens utilizadas em CLPs.

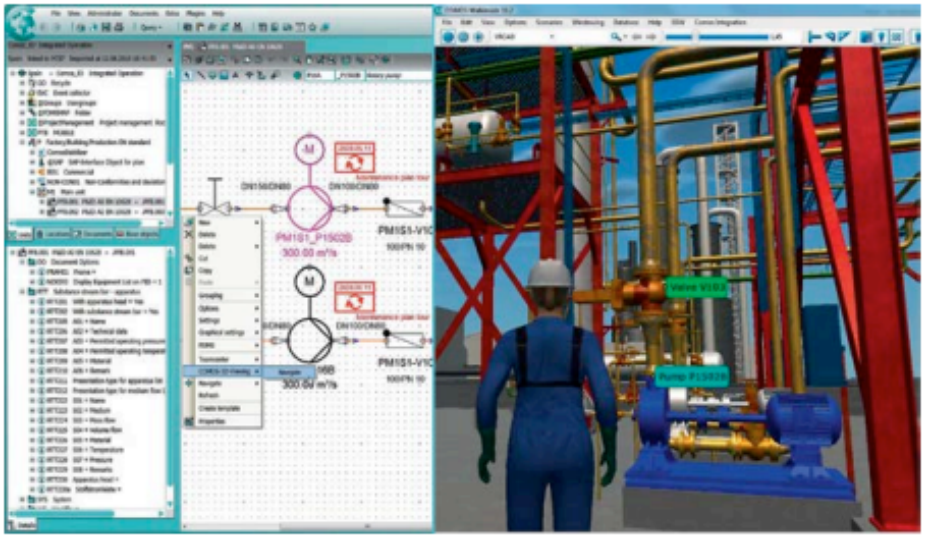
O treinamento foi divido em três partes. A primeira foi um cenário contendo uma esteira por onde passavam caixas a serem contadas. O objetivo desta primeira cena foi a familiarização dos alunos com o programa de simulação, *FacrotyIO*, e com o *CodeSys*, introduzindo conceitos da programação em *Ladder Diagram*, uma linguagem gráfica que se estrutura por representar a lógica de programação por meio de contatos e bobinas de forma análoga a um circuito de relés.(10, p. 37). No segundo cenário, era necessário transferir duas caixas ao mesmo tempo para o local determinado, para isso o programa deveria ser escrito em *Structured Text*, ST, linguagem textual de alto nível. Nesse caso, era necessário checar se existia duas caixas na esteira para que o atuador liberasse a passagem e quando não houvesse nenhuma, bloquear novamente. No terceiro cenário era preciso classificar caixas de acordo com seu tamanho, para isso foi requisitado a programação de uma maquina de estados em ST que executasse essa tarefa. Ao fim do treinamento, foi aplicada uma pesquisa para os 20 alunos participantes, sendo perguntas sobre o quão útil foi o treinamento e os graus de dificuldade de utilização das ferramentas propostas. Tendo resultados positivos de aceitação dos alunos sobre a ferramentas de simulação e desenvolvimento e sobre os problemas propostos.

2.2 Ferramentas comerciais

2.2.1 *COMOS Walkinside*

Uma ferramenta utilizada na indústria para simular o comportamento de plantas reais é o *COMOS Walkinside*(11), distribuída pela empresa de automação Siemens. O *COMOS Walkinside* faz um *Digital Twin* de uma fábrica, ou seja, replica e simula virtualmente todos os componentes e parâmetros relevantes, criando assim, um gêmeo digital de uma planta. Com isso, é possível que sejam feitos testes de comportamento do software do controlador, definir possíveis parâmetros de funcionamento ou testar a segurança do sistema frente à possíveis falhas de componentes.

Figura 1 – *COMOS Walkinside*



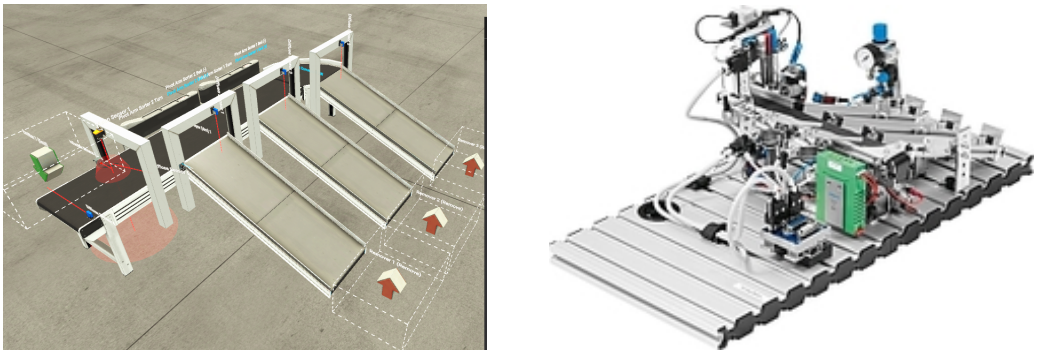
Fonte: Siemens(12)

Com o *COMOS Walkinside*, é possível treinar os operadores de uma planta que, por meio de realidade virtual, podem transitar pela planta virtual, Figura 1 e executar treinamentos de tarefas possivelmente arriscadas de serem feitas no mundo real, como um protocolo de ação em caso de incêndio ou falha catastrófica de algum componente. Além disso, segundo a empresa, o *COMOS Walkinside* proporciona maior disponibilidade da planta, por trazer mais capacidade de planejamento de paradas, aumento da eficiência da produção e transparência nos processos da planta.

A distribuição da ferramenta é feita por meio de licença dentro do pacote COMOS, que é a plataforma que faz o gerenciamento de ciclo de vida de uma planta, sendo possível fazer o planejamento, das instalações desde o fase de projeto; Documentação dos sistemas, tanto elétricos, hidráulicos e de automação e, também o planejamento de manutenções preventivas. No contexto didático, os custos de aquisição desta ferramenta são elevados.

2.2.2 *FactoryIO*

Outra ferramenta utilizada para a simulação de um ambiente de uma fábrica, é o *FactoryIO*(9) desenvolvido pela empresa *Real Games*. Essa ferramenta trás uma série de componentes virtuais que simulam dispositivos como sensores, esteiras, manipuladores, tanques, etc. Contém também objetos que podem interagir com estes componentes, como caixas, pallets e peças que podem ser montadas. Com estes componentes, é possível montar uma grande quantidade de cenários que simulam um ambiente de produção de uma fábrica onde se pode aplicar o controle desses componentes utilizando CLPs, como visto na Figura 2(a) que faz referência ao módulo de *sorting* da bancada didática da Festo, Figura 2(b) . Pode-se criar aplicações simples, como detecção de um objeto e acionamento e uma esteira, até aplicações mais complexas envolvendo por exemplo controle de nível de um tanque por meio do fluxo de entrada e saída de fluido.



(a) *Sorter* montado no *Factory IO*.

(b) Módulo *"MPS sorting station D"* da Festo.

Figura 2 – Uso do *FactoryIO* para representar um modelo físico

Fonte: (a)-Autor; (b)-Festo(13)

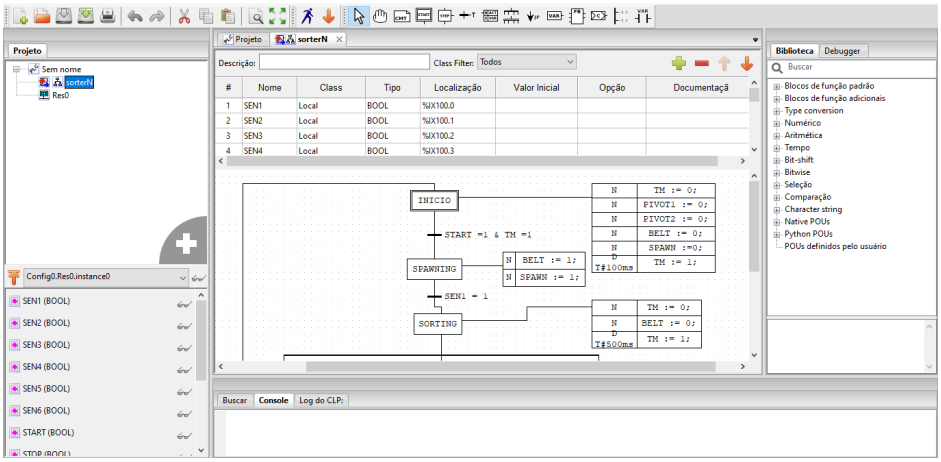
O *FactoryIO* pode se comunicar por meio de uma série de protocolos, como: As ehternet proprietárias de Allen-Bradley e da Siemens; Com WinSPS-S7 e Grafcet-Studio; *Modbus TCP*; *OPC Client DA e UA* ou por meio de USB 4750 e 4704. Além de poder ter um editor interno de FBD onde é possível desenvolver aplicações de controle para as cenas. Cada tipo de comunicação possui uma licença específica sendo a *Starter Edition* € 36,00 ao ano ou € 99,00 uma única vez, que disponibiliza o simulador interno de CLP, a de menor custo; A *Ultimate Edition* € 25,00 ao mês, € 253,00 ao ano ou € 695,00 uma vez. A versão que possibilida a comunicação por *Modbus TCP*, está disponível por € 15,00 ao mês, € 144,00 ou €395,00 uma vez. Valores disponíveis na página da desenvolvedora em Junho de 2021. Além das licenças pagas, a versão *Ultimate Edition* é disponível por um período de teste de 30 dias. Nota-se que o *FactoryIO* é uma poderosa ferramenta para o estudo de CLPs, porém atualmente tem seu custo elevado para ser adquirido para o uso em laboratório.

3 Ferramentas Utilizadas

3.1 *OpenPLC*

Para desenvolver programas para PCs, existem ferramentas usadas tanto para uma linguagem específica, como o *Dev-C++*(14) para linguagem C/C++, como uma IDE que serve para diversas linguagens, como o *Microsoft visual Studio Code*(15). Para CLPs, geralmente essas soluções são proprietárias do fabricante do hardware, como o PCS7 da Siemens, ou *Factory Talk* da Rockwell. O *OpenPLC* (16) é uma ferramenta de desenvolvimento de software para CLPs que é *Open Source* e pode gerar programas para uma série de diferentes CLPs e controladores compatíveis com *Arduino*. Nele é possível gerar e simular o funcionamento de programas escritos em *Ladder Diagram*, LD, *Sequential Flow Charts*, SFC, *Structured Text*,ST e *Instruction List*, IL. O *OpenPLC* possui um módulo de *Runtime*, onde é possível executar os programas criados no editor, em computadores com sistemas operacionais *Windows* ou *Linux*. Por conter as características apontadas anteriormente, OpenPLC será a ferramenta de desenvolvimento dos programas para CLP utilizada neste trabalho.

Figura 3 – *OpenPLC* - Ambiente de desenvolvimento

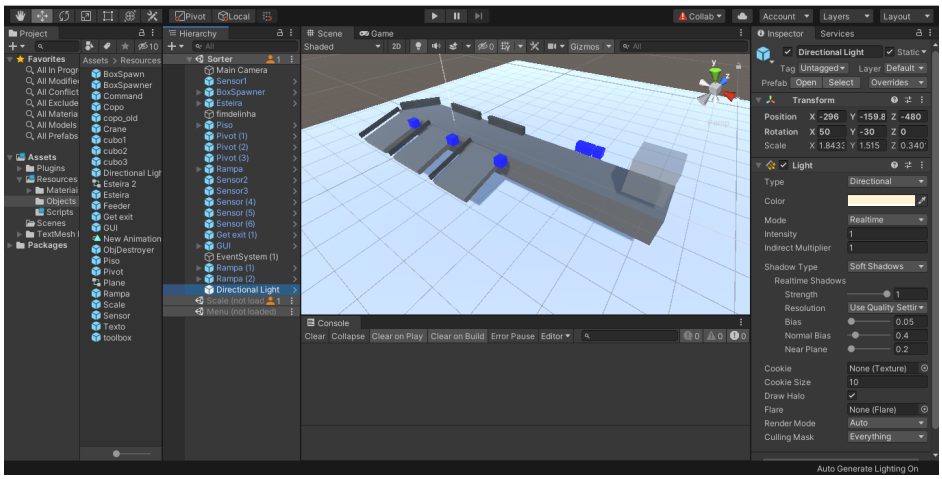


Fonte: Autor

3.2 *Unity*

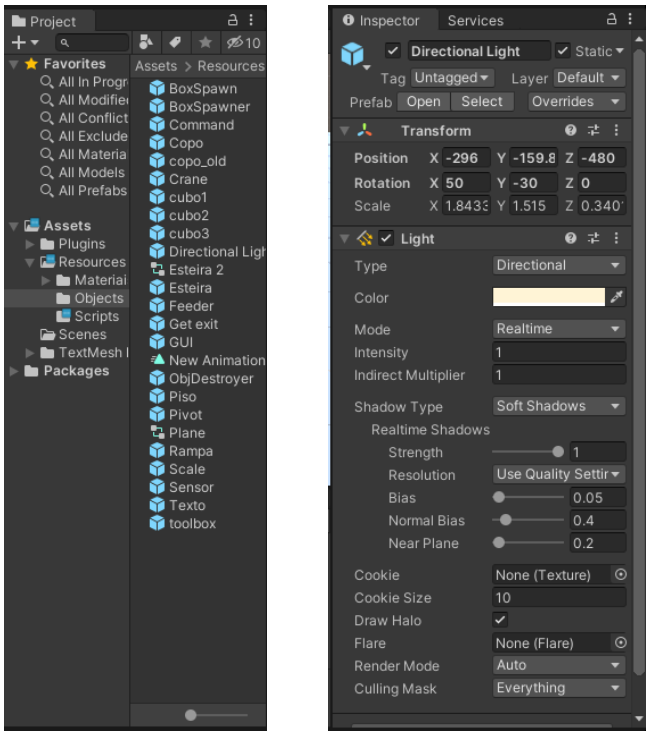
A *Unity* é uma plataforma de desenvolvimento 2D e 3D, contendo ferramentas que proporcionam a renderização, efeitos de iluminação e simulação de colisões entre objetos dispostos em uma cena. Dentro do ambiente de desenvolvimento, é possível associar *scripts* escritos em C# aos objetos, com isso é possível que a interações entre objetos sejam calculadas em tempo real, de acordo com o comportamento programado no *scripts*. Segundo os criadores (6), a *Unity* está presente em 70% entre os 1000 maiores jogos para plataformas móveis e em 50% dos jogos incluindo consoles, PC e mobile. Neste trabalho, a *Unity* será utilizada como a plataforma para criar as cenas e os objetos que interagem com os comandos vindos do CLP. O comportamento desses objetos serão programados em C#, utilizando o *Visual Studio Code*. Os objetos criados neste ambiente podem ser instanciados em mais de uma cena, garantindo um caráter modular ao conjunto de recursos disponíveis, mostrados adiante.

Figura 4 – *Unity* - Ambiente de desenvolvimento



Fonte: Autor

O Ambiente de desenvolvimento da *Unity* pode ser disposto conforme a Figura 4, onde à esquerda estão localizadas as abas que contém a hierarquia do projeto, Figura 5(a), e a pasta que contém os recursos em detalhe. À direita está posicionada a aba que detalha as propriedades do objeto selecionado, Figura 5(b). Ao centro é mostrada a cena que está sendo criada, Figura 6, com os seus objetos instanciados à esquerda.



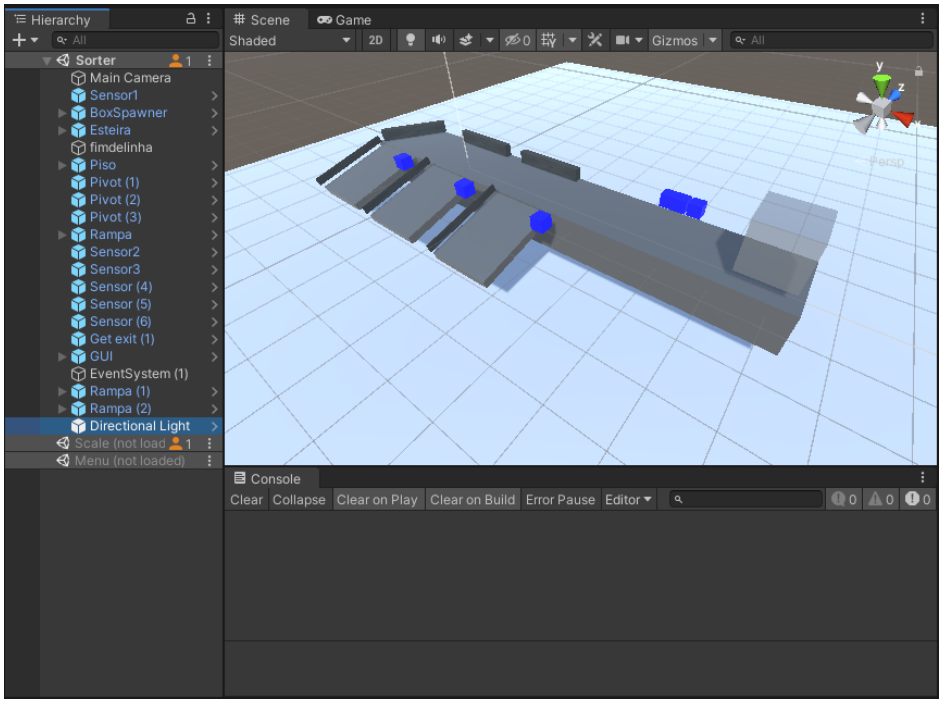
(a) Aba de hierarquia do projeto.

(b) Aba dos recursos

Figura 5 – *Unity* - Detalhe das abas de propriedades utilizadas

Fonte: Autor

Figura 6 – *Unity* -Detalhe da aba de cena



Fonte: Autor

3.3 *EasyModbus*

A *EasyModbus* (17) é uma biblioteca *Open source* que implementa o protocolo de comunicação modbus RTU e TCP. A *EasyModbus* é implementada nas plataformas .Net, Java e Python. O protocolo modbus (18) é um protocolo de comunicação industrial do tipo cliente-servidor, onde o cliente faz a requisição de um comando a ser respondido pelo servidor. Os comandos correspondem a leitura ou gravação de sinais, podendo ser digitais *inputs*, entradas e *coils*, saídas, ou analógicos *registers*.

A *EasyModbus* possui duas classes principais que possuem métodos e propriedades próprios. A classe *ModbusClient* contém os métodos e propriedades dos dispositivos do tipo cliente, por exemplo: Conexão com o servidor: *void Connect(string ipAddress, int port)*; Leitura do estado das entradas digitais: *bool[] ReadDiscreteInputs(int startingAddress, int quantity)*; Escrita em uma bobina: *void WriteSingleCoil(int startingAddress, bool value)*. A classe *ModbusServer*, por sua vez, trata das propriedades e métodos do lado do servidor, contendo propriedades como a porta que será utilizada para a comunicação *public int Port* ou os estados de cada registrador: *public InputRegisters inputRegisters*. Neste trabalho, o protocolo *modbus* será o meio de comunicação entre o CLP e o ambiente virtual, implementado utilizando a biblioteca *EasyModbus*

4 Solução proposta

4.1 Visão geral

Como visto anteriormente, é possível aplicar ferramentas de simulação como auxiliar no ensino de automação industrial, existem ferramentas que abordam esse tema como o *Factory IO*, porém, são de custo elevado. Este trabalho tem como uma premissa, utilizar ferramentas gratuitas para o desenvolvimento. Com isso, foi decidido a utilização da *Unity* como plataforma de desenvolvimento do ambiente virtual e do *OpenPLC* como ferramenta para o criar e executar os programas de controle.

A abordagem de desenvolvimento do ambiente virtual foi a de criar componentes independentes que podem ser aplicados em diversas cenas, apenas alterando parâmetros relevantes, sem que seja necessário mudanças em sua estrutura. Estes componentes serão detalhados adiante. Foram criadas cenas onde os componentes foram dispostos de maneira que pudessem interagir entre si.

Como forma de comunicar o software de controle com o hardware virtual, foi utilizada a biblioteca *EasyModbus* para fazer o gerenciamento do protocolo modbus. Sendo possível o uso de diversas ferramentas de controle que utilizam este protocolo.

Os comportamentos foram escritos em scripts em C#, pondendo ser acessados no repositório do *Github* (19), que podem ser implementados em mais de um tipo de componente.

4.2 Projeto de componentes

Nesta seção serão apresentados os componentes utilizados, suas propriedades e scripts associados. OS componentes utilizados neste trabalho foram escolhidos por serem de uso comum em aplicações de automação, além de serem exemplos básicos de utilização de entradas e saídas digitais e analógicas. Por exemplo, uma esteira é um exemplo de componente que é acionado por meio do software de controle, que numa aplicação real é movida usualmente por motor. Já o sensor é uma entrada digital que pode representar, simplificadamente, diversos componentes de detecção como sensores de presença ou chaves de fim de curso. Os endereços de acionamento e de leitura são definidos durante a montagem das cenas. Quando se mantém o ponteiro do mouse sobre um componente, é mostrado seu endereço associado na interface gráfica.

4.2.1 Alimentador

O Alimentador, Figura 7, simula o comportamento de um alimentador industrial, componente que faz o carregamento de materiais em um contexto industrial. No modelo apresentado neste trabalho, o Alimentador pode ter sua vazão de material definida na montagem da cena.

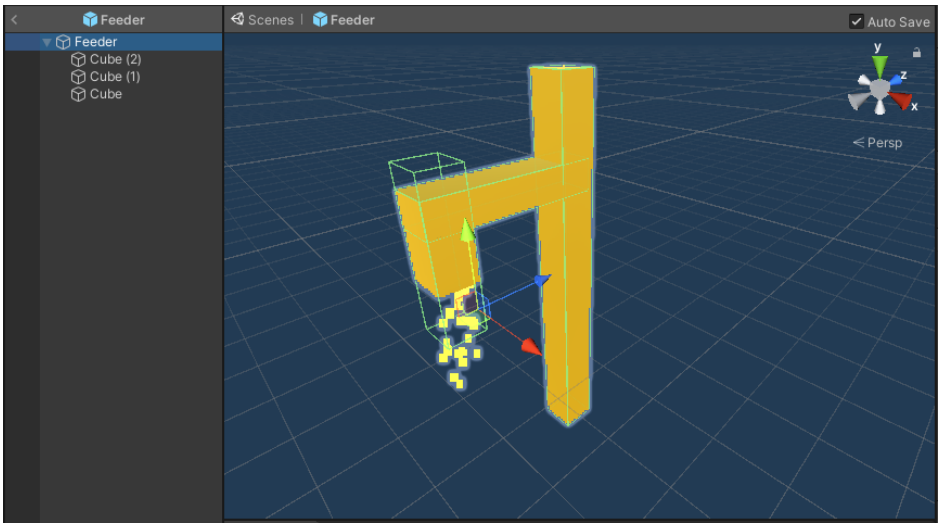
Para construir o Alimentador, foi criado um objeto vazio, *Empty*, que carrega as propriedades vistas acima, além disso, foram criados três cubos, objetos do tipo *Cube*, para formar a sua estrutura, no cubo da extremidade foi posto um *Box Collider*, colisor em caixa, com a propriedade do tipo *Trigger*, disparo. Ao colocar a propriedade como *Trigger*, a *Unity* ignora o colisor como um corpo rígido e apenas dispara o evento de detecção.

Tabela 3 – Propriedades do Alimentador

Propriedade	**Tipo**	**Descrição**
Addr	STRING	Endereço mostrado na interface gráfica
Coil	INT	Número da *coil* que ativa o alimentador
Ligado	BOOL	Estado do alimentador
Rate	REAL	Velocidade que o Alimentador irá preencher os recipientes
Scripts		*"Feeder.cs"* ; *"Show Addr.cs"*

Fonte: Autor

Figura 7 – Alimentador



Fonte: Autor

4.2.2 Balança

A Balança, Figura 8, registra a massa dos objetos que estão em contato com a sua face superior. Além do sensor de massa, neste trabalho, a balança tem a funcionalidade de esteira, com isso, é possível criar cenários onde a esteira só faz a liberação de um objeto que atinja determinada massa.

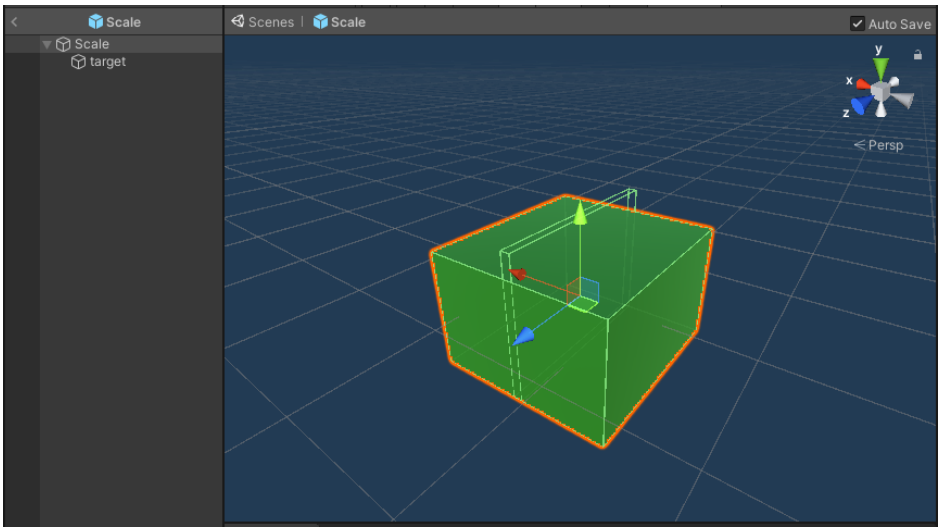
Para a construir a balança, foi criado um cubo que, além do *Box Collider* criado originalmente, foi adicionado outro *Box Collider* no centro, com a propriedade de *Trigger*, com o objetivo de disparar a detecção do objeto que está em cima da esteira.

Tabela 4 – Propriedades da Balança

Propriedade	**Tipo**	**Descrição**
Addr	STRING	Endereço mostrado na interface gráfica
Coil	INT	Número da *coil* que ativa a esteira da Balança
Ligado	BOOL	Estado do alimentador
Reg	INT	Número do registrador que guarda a massa na Balança
Scripts		"Esteira.cs"; *"Scale.cs"* ; *"Show Addr.cs"*

Fonte: Autor

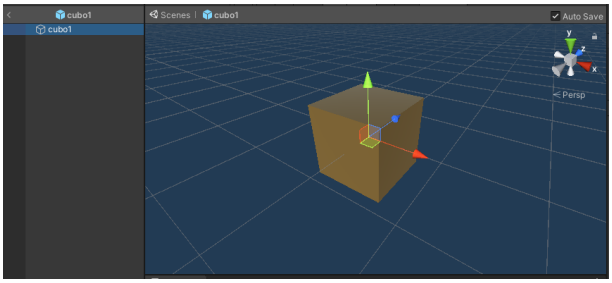
Figura 8 – Balança



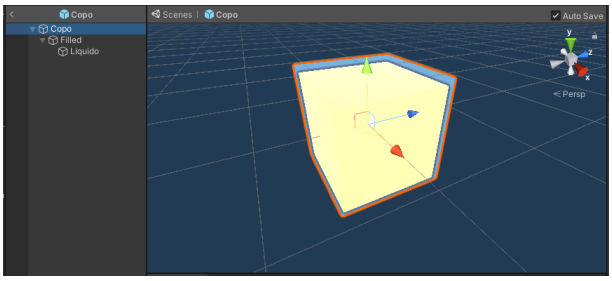
Fonte: Autor

4.2.3 Caixa

As Caixas são os objetos que são deslocados nas esteiras e manipulados pelo manipulador. Estes objetos são instanciados nas cenas por meio do *Spawner*. Neste projeto foram criados quatro tripos de caixas, três que são sólidas, 9(a) e se diferenciam pelas cores laranja, cinza e verde. Além da caixa do tipo "copo", Figura 9(b) que possui a propriedade de representar o seu nível de enchimento.



(a) Caixa



(b) Copo

Figura 9 – Caixa e Copo

Fonte: Autor

Para contruir as caixas sólidas, foram criados cubos e, em cada tipo de cor, foi atribuído um material diferente. Para construir a caixa do tipo "copo", foi criado um objeto vazio, e dentro dele criado um cubo com um material com transparência atribuído. Dentro do cubo transparente, foi colocado um cubo sólido que, por meio da propriedade *filled*, tem seu volume alterado.

Tabela 5 – Propriedades da Caixa

Propriedade	**Tipo**	**Descrição**
changeMass	BOOL	Caixa está alterando a massa
changeRate	FLOAT	Taxa de variação da massa
Crane	GAMEOBJECT	Manipulador que está movendo a caixa
naCrane	BOOL	Caixa está em um Manipulador
naesteira	BOOL	Caixa está em movimento em uma esteira
Target	VECTOR3	Ponto de destino da caixa
Velocidade	FLOAT	Velocidade de deslocamento da caixa
Scripts		"ObjEsteira"

Fonte: Autor

Tabela 6 – Propriedades do Copo

Propriedade	**Tipo**	**Descrição**
Todas as propriedades da Caixa Enchendo	BOOL	Copo está enchendo
Filled	FLOAT	Proporção de enchimento do Copo
Rate	FLOAT	Taxa de enchimento do Copo
Scripts		"Copo.cs"

Fonte: Autor

4.2.4 Desviador

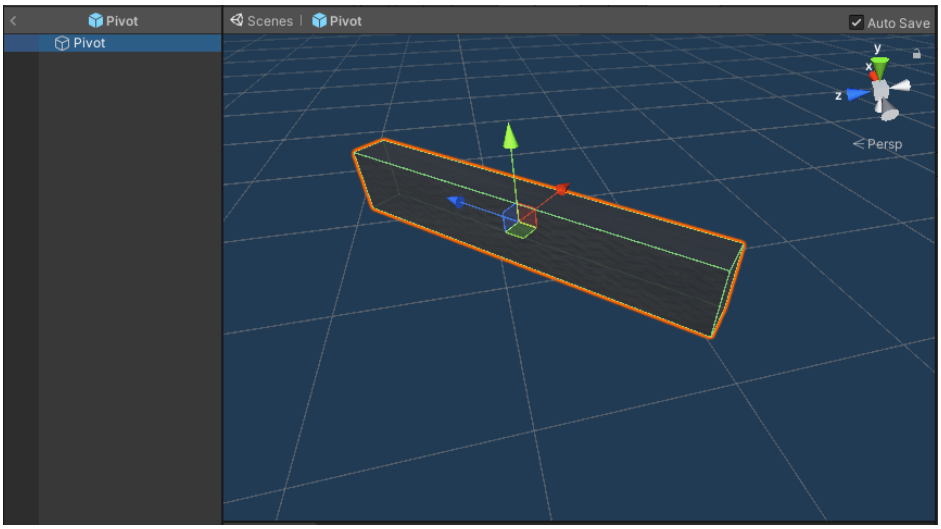
O Desviador, Figura 10, é um componente adicionado na esteira com o objetivo de alterar a direção dos objetos que estão sendo transportados. A construção do Desviador foi feita com um cubo redimensionado para as dimensões desejadas.

Tabela 7 – Propriedades do Desviador

Propriedade	**Tipo**	**Descrição**
Acionado	BOOL	Estado do Desviador
Addr	STRING	Endereço mostrado na interface gráfica
Coil	INT	Número da *coil* que ativa o alimentador
Scripts		*"Pivot.cs"*; *"Show Addr.cs"*

Fonte: Autor

Figura 10 – Desviador



Fonte: Autor

4.2.5 Esteira e Rampa

Este projeto contém dois tipos distintos de esteiras transportadoras, a Esteira e a Rampa. O primeiro tipo, Esteira Figura 11(a), transporta os objetos que estão na parte de cima somente quando está acionado. O segundo, Rampa figura 11(b), não contém acionamentos e os objetos sempre irão se deslocar quando estiverem em contato, servindo como um componente passivo.

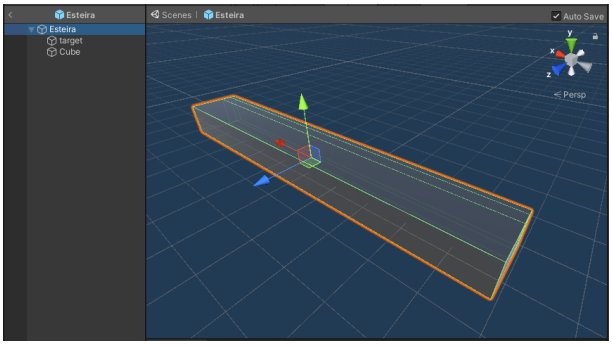
Para construir a Esteira, foi criado um cubo como *Game Object* principal, no qual o *Box Collider* foi deslocado da linha de centro, para que o objeto transportado troque de direção somente quando está completamente sobre a segunda esteira. Ao diminuir o tamanho do colisor, foi necessário criar outro cubo, como filho do principal, para servir de apoio estrutural.

A Rampa foi construída também com o *Box Collider* deslocado, além disso, o cubo principal foi rotacionado em 15º, e colocado um anteparo para impedir que os objetos caiam.

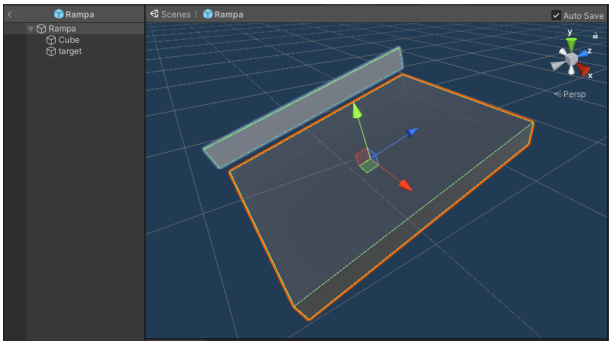
Tabela 8 – Propriedades da Esteira

Propriedade	**Tipo**	**Descrição**
Coil	INT	Endereço da *coil* que ativa a Esteira
Ligado	BOOL	Estado da Esteira
IsRampa	BOOL	Define se é uma Rampa
Scripts		*"Esteira.cs"*; *"Show Addr.cs"*

Fonte: Autor



(a) Esteira



(b) Rampa

Figura 11 – Esteira e Rampa

Fonte: Autor

4.2.6 Manipulador

Neste projeto, o Manipulador, Figura 12 é o componente que dá a capacidade de mover objetos livremente, dentro de seu volume de controle. O manipulador possui três

graus de liberdade, sendo dois com atuadores lineares e um angular, Z, Y, Θ. Além do posicionamento, é possível ativar o atuador que "captura", os objetos em seu alcance.

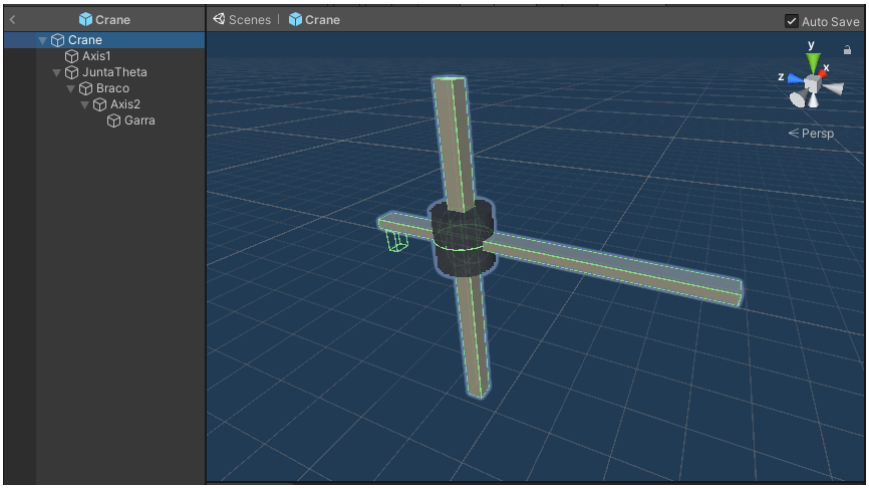
Para construir o Manipulador foi criado um objeto vazio, *Empty*, como *Game Object* principal, como filhos foi criado um cubo como o eixo Y e um cilindro representando a junta Θ. Como filho da junta, foi adicionado objeto vazio tendo como filho um cubo como eixo Z, contendo outro objeto vazio com um colisor em modo *trigger* representando a garra. Essa estrutura, onde um objeto está instanciado como filho de outro, foi utilizada para que a manipulação das coordenadas seja feita de maneira local, ou seja, é possível definir a origem de um objeto filho como o centro do objeto pai. Com isso, quando é feita a rotação ou translação de um objeto pai, as coordenadas locais dos objetos filhos não se alteram.

Tabela 9 – Propriedades do Manipulador

Propriedade	**Tipo**	**Descrição**
Fbk Angle	FLOAT	Leitura do Angulo Θ atual
Fbk Y	FLOAT	Leitura da posição atual do eixo Y
Fbk Z	FLOAT	Leitura da posição atual do eixo Z
Grab	BOOL	Estado da garra do Manupulador
Grab Coil	INT	Endereço da *Coil* da garra
Manual	BOOL	Modo de operação do Manupulador
Reg Angle	INT	Endereço do *Register* da junta Θ
Reg Y	INT	Endereço do *Register* do eixo Y
Reg Z	INT	Endereço do *Register* do eixo Z
Set Angle	FLOAT	Setpoint do Angulo Θ
Set Pos Y	FLOAT	Setpoint da posição Y
Set Pos Z	FLOAT	Setpoint da posição Z
Speed	FLOAT	Velocidade de movimentação do Manipulador
Scripts		*"Crane.cs"*

Fonte: Autor

Figura 12 – Manipulador



Fonte: Autor

4.2.7 Piso

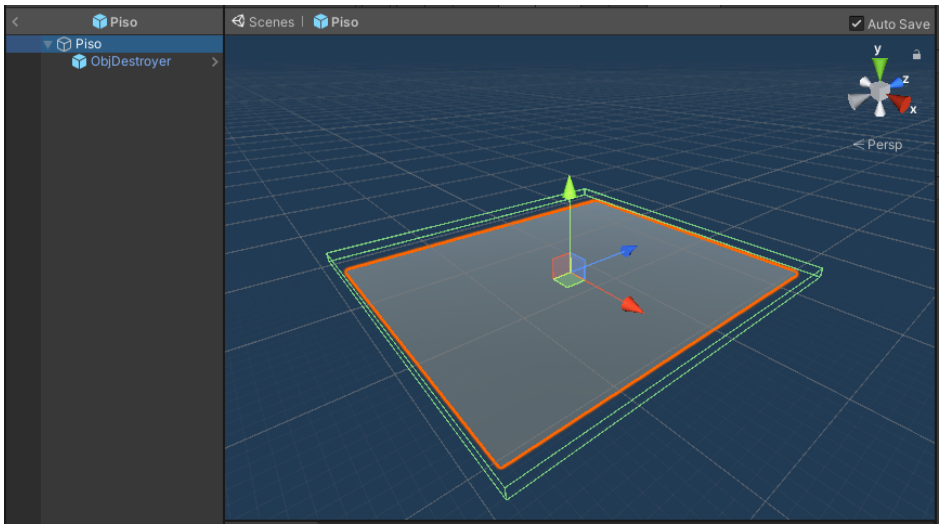
O Piso tem o papel de manter a simulação menos sobrecarregada, destruindo as caixas que entram em contato com ele, sendo um componente passivo. Para isso, foi criado um objeto do tipo plano, *Plane*, contendo um *Box Collider* que detecta as colisões.

Tabela 10 – Propriedades do Piso

Propriedade	**Tipo**	**Descrição**
Scripts		*"Destruidor.cs"*;

Fonte: Autor

Figura 13 – Piso



Fonte: Autor

4.2.8 Sensor

Os objetos do tipo Sensor, Figura 14, deste projeto são equipamentos que emitem um sinal quando um objeto, do tipo que foi definido previamente, passa por sua região de detecção. Conferindo a este tipo de objeto a característica de um dispositivo de detecção (10, p. 7).
O Sensor pode ser configurado atribuindo um valor à variável *senstype*. Com isso o Sensor pode detectar somente um tipo de caixa (laranja: 1; verde: 2; cinza: 3) ou detectar qualquer caixa que passe em sua região de detecção (*senstype* = 0).

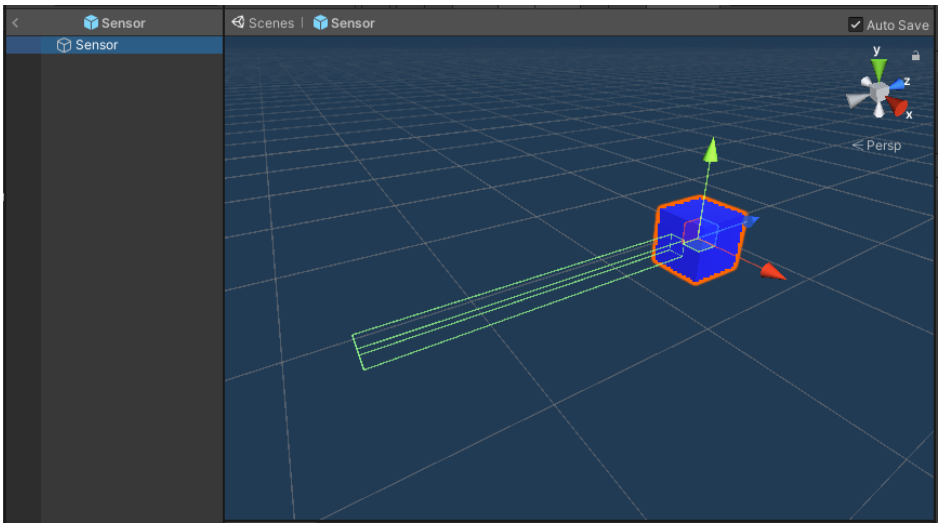
Para construir o Sensor, foi criado um cubo onde o seu *Box Collider* foi definido como do tipo *Trigger* e suas dimensões foram alteradas para cobrir uma região além do cubo original como visto na Figura 14. O Sensor tem sua cor modificada, de azul para branco, quando está acionado.

Tabela 11 – Propriedades do Sensor

Propriedade	**Tipo**	**Descrição**
Addr	INT	Endereço mostrado na interface gráfica
Numero	INT	Endereço da *Discrete Input* do Sensor
Senstype	INT	Modo de detecção do Sensor
Scripts		*"Sensor.cs"*; *"ShowAddr.cs"*

Fonte: Autor

Figura 14 – Sensor



Fonte: Autor

4.2.9 *Spawner*

O *Spawner*, Figura 15, tem a função de ser uma fonte de caixas nas cenas. Quando ativado, o *Spawner* cria caixas onde ele está posicionado. Enquanto houver uma caixa dentro da sua região de criação, o *Spawner* não irá criar outra caixa, mesmo se estiver ativado.
É possível determinar previamente, durante a montagem da cena, sua velocidade e quais tipos de caixas cada *Spawner* irá criar.

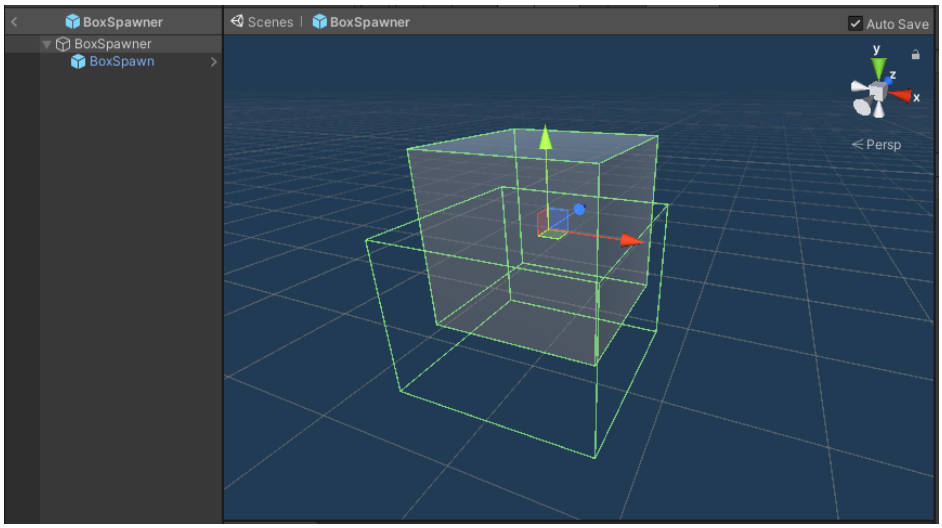
Para construir o *Spawner*, foi criado um objeto vazio dentro de um cubo. Este objeto vazio está associado ao *script* que cria as caixas nas cenas. Ele contém um *Box Collider* do tipo *Trigger* onde é definida a região de criação das caixas. O cubo contém o *script* que mostra o endereço da *coil* associada ao componente. Foi atribuído um material translúcido ao cubo com o objetvo de indicar visualmente que o *Spawner* cria caixas, mas não é uma.

Tabela 12 – Propriedades do *Spawner*

Propriedade	**Tipo**	**Descrição**
Addr	INT	Endereço mostrado na interface gráfica
Coil	INT	Endereço da *Coil* do *Spawner*
Cube Type 1	GAMEOBJECT	Primeiro tipo de caixa que a ser criada
Cube Type 2	GAMEOBJECT	Segundo tipo de caixa que a ser criada
Cube Type 3	GAMEOBJECT	Terceiro tipo de caixa que a ser criada
Contador	FLOAT	Contador de intervalo de criação
Intervalo	FLOAT	Intervalo de criação de caixas
Scripts		*"Cubespawner.cs"*; *"ShowAddr.cs"*

Fonte: Autor

Figura 15 – Spawner



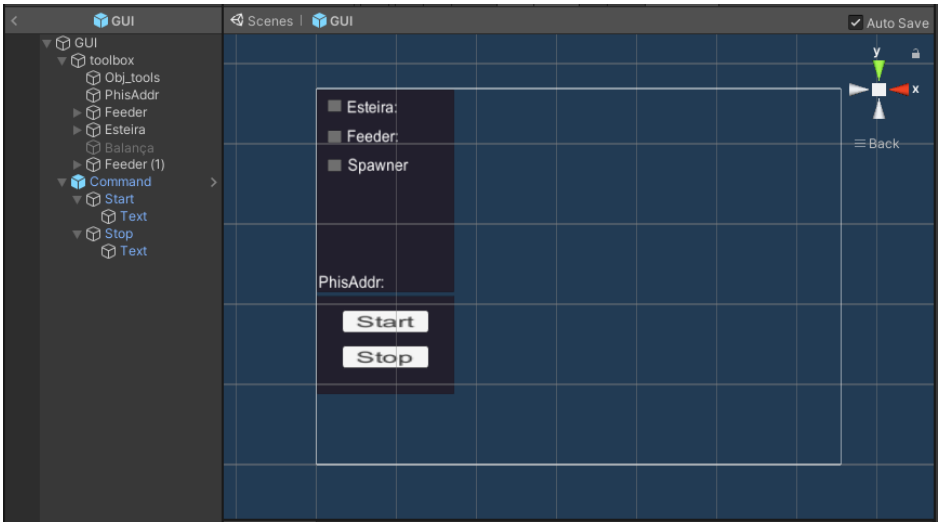
Fonte: Autor

4.2.10 GUI - Interface Gráfica

A Interface Gráfica, Figura 16, contém elementos que exibem os estados dos elementos que podem ser acionados na cena. Quando um equipamento está ligado, seu nome na interface gráfica é mostrado na cor verde, quando está desligado, é mostrado em vermelho. Além de mostrar o estado, é possível ligar e desligar manualmente estes equipamentos. Caso o equipamento ter uma saída do tipo analógico, este valor é mostrado ao lado do nome. Quando o usuário mantém o ponteiro do mouse sobre algum componente da cena, o endereço de entrada ou saída associado é mostrado na interface.
Na Interface Gráfica, também estão presentes dois botões, *Start* e *Stop*, sendo possível o uso como sinais para iniciar ou parar um programa externo.

Para construir a Interface Gráfica, primeiro foi criado um elemento do tipo *Canvas* onde foram inseridos dois *Panels* um listando os componentes que podem ser acionados, cada um com um botão associado à sua respectiva *Coil*. Esta lista é montada na criação da cena, sendo necessário definir o nome e a *Coil* de cada um dos componentes listados. O outro *Panel* contém os botões que podem ser usados como comandos para programas externos, associados cada um a uma *Discrete Input*.

Figura 16 – GUI



Fonte: Autor

4.3 Scripts auxiliares

Cada script deste projeto é responsável por uma classe que define um comportamento específico da aplicação, estes scripts foram escritos em C#, linguagem que a *Unity* é capaz de interpretar. Os scripts desenvolvidos neste projeto podem ser vistos neste repositório do *GitHub* 19. Os componentes possuem um ou mais scripts que implementam as suas propriedades. Um script pode ser usado por mais de um componente diferente que compartilham a mesma propriedade, como o script *"ShowAddr.cs"* que implementa a funcionalidade de exibir o endereço de um componente quando se mantém o ponteiro do mouse sobre ele, mas pode não estar associado a nenhum componente de cena, como o *FactoryManager.cs*.

Os scripts deste projeto utilizam algumas funções nativas do ambiente da *Unity*, como:

Start(): Que é executada uma vez quando um objeto é instanciado. Geralmente atribui configurações relativas ao objeto.

Update(): É executada a cada vez que há uma atualização de quadro na cena. pode ser utilizada para atualizar os estados do objeto. Adiante neste capítulo, quado é dito que determinada função é executada a cada quadro ou *frame*, a chamada dessa função é feita por meio da *Update()*.

OnCollisionEnter(Collision co): Esta função e suas varições como *OnCollisionStay(Collision co)*, fazem o tratamento de uma colisão com um *Collider* de um objeto.

OnTriggerEnter(Collider co): Faz o tratamento dos disparos dos colisores do tipo *Trigger* de um objeto.

O funcionamento de cada script é descrito a seguir e seus respectivos códigos estão

como apêndices.

4.3.1 *ButtonUI.cs*

O script *ButtonUI.cs* contém a classe *ButtonUI* que implementa o uso dos botões da interface gráfica, esta classe possui três funções:

ManChangeState(int coil): Troca o estado da *coil* selecionada. Este recurso possibilita que o acionamento de um componente, como uma esteira, possa ser feita manualmente pelo usuário.

ManFlipInput(int input): Troca o estado da *discrete input* selecionada. Este recurso possibilita que o estado de uma entrada, como um sensor, possa ser alterado manualmente pelo usuário.

StartALevel(string levelName) Carrega a cena selecionada.

4.3.2 *CameraControl.cs*

O script *CameraControl.cs* possibilita que a câmera da cena seja movimentada. A movimentação câmera é feita por meio das setas. A rotação é feita ao manter a tecla *Shift* pressionada em conjunto com uma seta.

4.3.3 *Copo.cs*

O script *Copo.cs* implementa o comportamento da caixa do tipo "Copo". O script verifica a massa da caixa e faz a atualização da sua coordenada Z local por meio da função *fill()*. A coordenada Z é vinculada à variável *filled* do tipo *float* que, por sua vez, é a massa da caixa dividia por 30, sendo limitada entre 0 e 0.95.

4.3.4 *Crane.cs*

O script *Crane.cs* faz o controle do posicionamento e da manipulação de objetos do manipulador. A cada *frame*, são executadas as funções: *getgrab()* ; *walk_Y()* ; *walk_Z()* ; *walk_theta()* sendo responsáveis pela captura de objetos pela garra e pela atualização das posições Y, Z e Θ.

getgrab(): Caso a *Coil* de atuação da garra estiver acionada, a próxima caixa que estiver no volume de alcance da garra será sinalizada como sendo manipulada pelo Manipulador.

walk_Y(): Faz a movimentação do eixo Y em direção ao *setpoint*. Para isso, o valor da coordenada Y local é adicionada de k, valor relacionado ao inverso da distância entre a posição atual e o *setpoint*. Quando essa diferença é menor que 0.001, arbitrariamente

pequena, é assumido que o destino foi alcançado. Caso o limite máximo ou mínimo seja atingido, a propriedade de *halo* é acionada, simulando uma luz de alerta de fim de curso.

walk_Y(): Faz a movimentação do eixo Y em direção ao *setpoint*. Seu funcionamento é análogo à *walk_Y()*.

walk_theta(): Faz a movimentação da junta Θ em direção ao *setpoint*. Seu funcionamento é análogo à *walk_Y()*, porém não possui limitação de valor de movimentação em seu eixo.

4.3.5 *CubeSpawner.cs*

O script *Crane.cs* implementa o *Spawner*. A cada quadro, é verificado se o *Spawner* está ativado e incrementa o valor de seu contador de tempo de emissão. Caso esse tempo alcance o limite definido para o objeto, a função *spawn()* é chada.

spawn(): Cria uma instância de uma caixa dentre os tipos definidos para um determinado *Spawner* de forma aleatória. Depois de instanciada, é definida para uma massa inicial para a caixa, de forma aleatória dentro de um limite arbitrário de 1kg a 5kg.

4.3.6 *Destruidor.cs*

O script *Destruidor.cs* implementa a função de remover os objetos que tocam o chão. Para isso, cada objeto que entra em contato com o piso, são destruídos com o comando *Destroy(co.gameObject);* onde *co* é o objeto que colidiu com o piso.

4.3.7 *Esteira.cs*

O script *Esteira.cs* faz o comportamento da esteira mover as caixas que estão em contato. Para isso, quando uma caixa entra em contato com o colisor da esteira, é enviado uma mensagem para o objeto da caixa, informando o estado da esteira, se está se movendo ou não, este estado é atualizado a cada quadro. Além disso, é informado para a caixa o ponto para onde deve se mover.

4.3.8 *FactoryManager.cs*

O script *FactoryManager.cs* foi criado como controlador de comunicação *ModBus* entre uma cena e o ambiente externo. Todas as interações entre os componentes e o que é externo ao ambiente do programa de simulação são concentradas e mediadas pelo objeto *FactoryManager*. É por meio dele, que é possível ler e escrever nos registradores e bobinas vinculados aos componentes das cenas. Este objeto é "auto instanciado" no sistema quando algum outro objeto faz uma requisição à ele, por exemplo, o *FactoryManager* passa a existir

no contexto da cena quando um sensor define o seu estado na *discrete input* vinculada a ele. A classe *FactoryManager* contém as seguintes funções:

ChecaCoil(int coil): Retorna o valor da bobina *coil*.

SetCoil(int coil, bool estado): Atribui o valor da variável *estado* à bobina *coil*.

FlipCoil(int coil): Inverte o valor da bobina *coil*.

SetReg(int sensor, bool estado): Atribui o valor da variável *estado* à *discrete input sensor*.

FlipInput(int sensor): Inverte o valor da *discrete input sensor*.

LeInputReg(int reg): Retorna o valor do registrador de entrada *reg*.

SetInputReg(int reg, int value): Atribui o valor da variável *value* ao registrador de saída *reg*.

LeHoldingReg(int reg): Retorna o valor do registrador de saída *reg*.

stopConnection(): Interrompe o servidor *ModBus*.

4.3.9 *Feeder.cs*

O script *Feeder.cs* implementa o comportamento do Alimentador. Para isso, guarda qual foi a caixa que entrou em sua região de alcance e, a cada quadro, envia a mensagem de alterar ou não sua massa e qual é a taxa.

4.3.10 *ObjEsteria.cs*

O script *ObjEsteria.cs* implementa o comportamento das Caixas. A cada quadro são atualizadas a massa e a posição da caixa. Este script contém as seguintes funcões:

UpdateMass(): Caso esteja definido que a caixa deva ter sua massa altera, faz essa alteração, de acordo com a taxa atribuída à variavel *changeRate*;

UpdatePosition(): Caso a caixa esteja em uma esteira ligada, a caixa irá se mover em direção a um ponto definido pela esteira à esta caixa. Caso esteja sendo movida por um manipulador, os efeitos da gravidade são desativados e a sua posição é a mesma que a garra do manipulador, deslocada de 0.5 no eixo Y.

EstadoEsteira(bool estado): Define se a caixa deve se mover ou não em direção ao alvo.

Fill(float rate): Define que a caixa deve ter sua massa alterada e atribui a taxa de variação à variável *changeRate*, de acordo com a variável *rate*;

NoFill(): Define que a caixa não deve ter sua massa alterada.

4.3.11 *PhisAdr.cs*

O script *Feeder.cs* contém a função ShowAddr(string addr), que atribui o valor do enderço ao texto mostrado na interface gráfica.

4.3.12 *Pivot.cs*

O script *Pivot.cs* implementa o comportamento do Desviador. Quando instanciado, sua posição inicial é guardada e caso esteja acionado, sua posição é alterada para desviar as caixas da esteira, sofrendo um deslocamento de Vector3(-0.025F,0F,-0.695F) e uma rotação de Vector3(0,30,0). Quando não está acionado, retorna à posição inicial.

4.3.13 *RenderOrder.cs*

O script *RenderOrder.cs* é um auxiliar para que as texturas de um objeto seja renderizada, criada no quadro, na ordem ordem correta, ou seja, garante que algo que um componente que esteja dentro de outro, seja representado dentro desse componente.

4.3.14 *Scale.cs*

O script *Scale.cs* implementa o comportamento da Balança. Quando uma caixa passa por cima da balança, sua massa é atribuída ao registrador vinculado à esta balança. Quando a caixa sai da região de *trigger*, o registrador recebe 0 como valor.

4.3.15 *sensor.cs*

O script *sensor.cs* implementa as funcionalidades do Sensor. Para isso, quando uma objeto dispara o *trigger* deste sensor, é chamada a função *checatag(string tag)* que verifica se é uma caixa do tipo que este sensor é capaz de detectar, caso positivo, atribui *true* à *discrete input* vinculada ao sensor. Caso o sensor for do tipo 0, que detecta objetos de qualquer tipo que entra em contato, a função *checatag(string tag)* não é chamada e atribui *true* à sua entrada discreta. Quando ele está em *true*, altera a cor para branco. Quando não detecta nenhum objeto de seu tipo, mantém a sua *discrete input* em *false* e a sua cor original.

4.3.16 *ShowAddr.cs*

O script *ShowAddr.cs* é um auxiliar da interface gráfica que, quando se mantém o ponteiro do mouse sobre um objeto, mostra o valor do seu endereço *modbus* na interface gráfica. Além de mudar a cor deste objeto para amarelo. Para isso, procura o objeto cujo nome é *"PhisAddr"* e atribui a propriedade *addr* do componente que está em destaque.

Quando o mouse não está sobre este objeto, sua cor volta ao original e é mostrado "-" na região de endereço da interface gráfica.

4.3.17 *UpdTextoUI.cs*

O script *UpdTextoUI.cs* implementa o comportamento de indicar o estado de um componente da cena na interface gráfica. Para isso, captura o estado do entrada, registrador ou bobina associado ao componente e caso seja digital, mostra mostra em verde ou vermelho se *true* ou *false* respectivamente. Caso for um registrador que contém valores numéricos, mostra o valor deste registrador, como o valor lido de uma balança, por exemplo.

4.4 Projeto de cenas

A construção de cada cena é feita utilizando os componentes descritos anteriormente. Cada cena pode ter diversos dispositivos do mesmo tipo, de acordo com a necessidade de cada aplicação. Uma cena pode ser montada para um propósito específico, como a Cena 1, que seu propósito é separar os objetos de uma linha, mas também uma cena pode ter mais de uma aplicação, como a Cena 2. Animações das cenas em execução estão disponíveis neste repositório do *GitHub* (19).

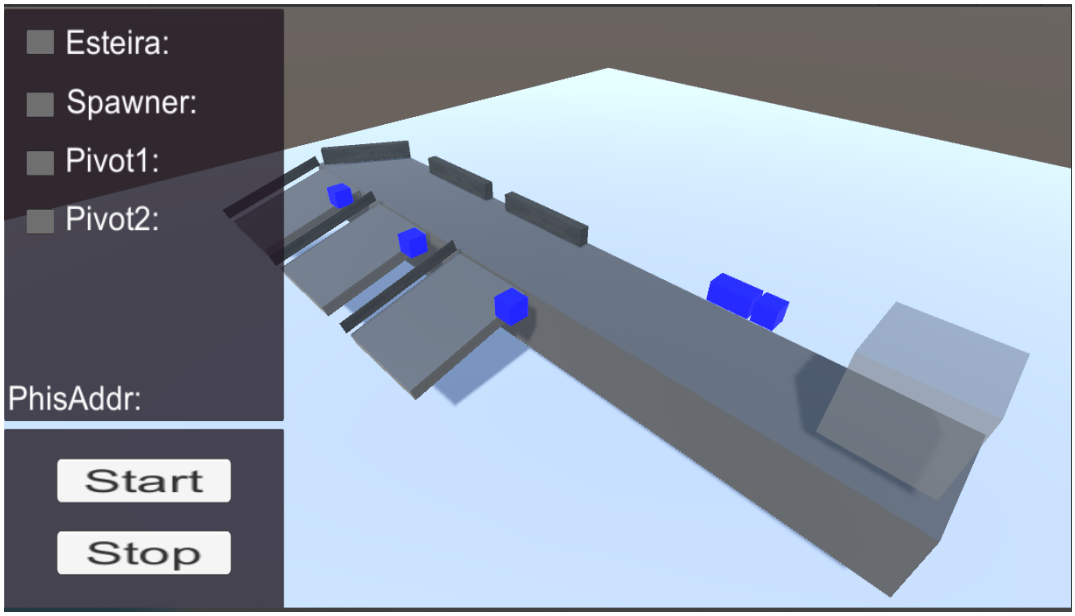
4.4.1 Cena 1 - Sorter

A primeira cena a ser montada no projeto foi a *Sorter*, inspirada no módulo *"MPS sorting station D"* da bancada didática de treinamento em automação industrial da Festo. Esta cena consiste em uma esteira principal que transporta caixas de três tipos diferentes, que devem ser separadas cada uma em uma linha específica que são rampas. Para isso, são posicionados três sensores na esteira principal, um de presença, que detecta qualquer objeto que esteja na sua região de alcance, outro que detecta somente caixas do tipo A e outro somente do tipo B, em analogia aos sensores que são sensíveis a materiais metálicos e sensores que são ativados de acordo com a cor do objeto. Além dos sensores da esteira principal, são posicionados sensores de presença em cada rampa. Na esteira principal, também são posicionados desviadores que direcionam as caixas às suas respectivas rampas.

4.4.2 Cena 2 - Loader | Crane

A Cena 2 é montada de forma que pode ser utilizada para exercícios que envolvem qualquer combinação entre o manipulador, balança e alimentador. Por exemplo, pode-se desejar que o alimentador carregue uma caixa até determinada massa e seja transportada entre as esteiras; Pode ser adicionada a tarefa de remover da linha principal, utilizando

Figura 17 – Visão da Cena 1 *Sorter*



Fonte: Autor

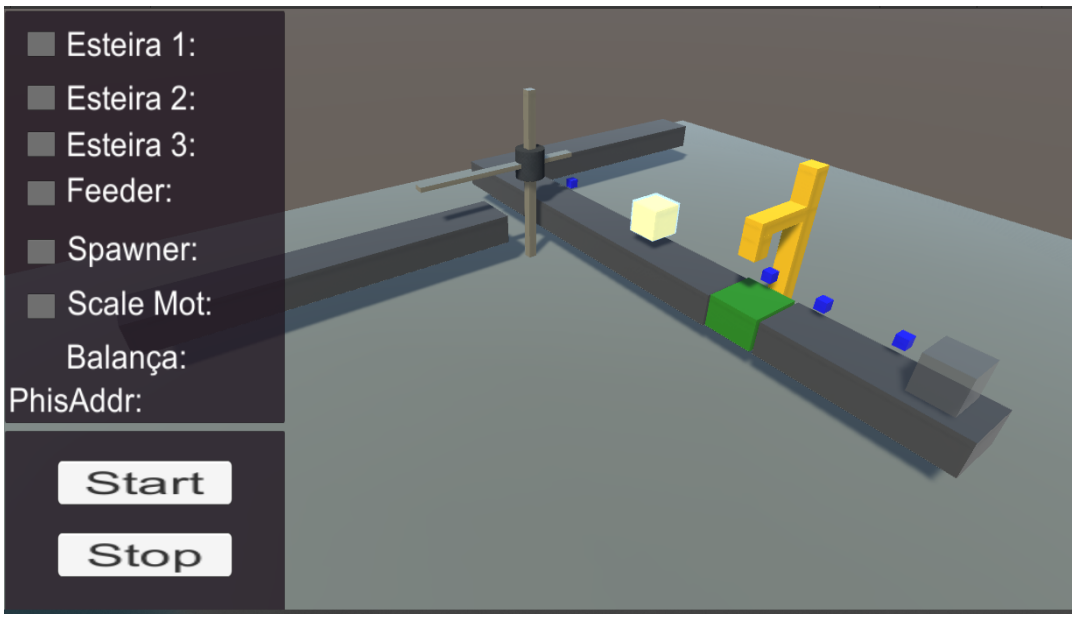
o manipulador, as caixas que não foram carregadas corretamente pelo alimentador; Ou somente usar o manipulador para mover caixas de uma esteira para outra. Para montar a Cena 2, foi criada uma esteira principal contendo uma esteira em formato de L, composta de duas esteiras dispostas perpendicularmente, uma balança e uma esteira para a entrada das caixas. Na esteira para entrada, foi colocado um *spawner* para gerar as caixas e dois sensores de presença, um na posição central e outro posicionado na extremidade oposta ao *spawner*. Ao lado da balança, foi colocado um alimentador e um sensor de presença. O manipulador foi posicionado na região central de uma das partes da esteira em L, junto a um sensor de presença. Próximo ao manipulador, foi posicionada outra esteira formando uma linha secundária.

4.5 Programas de controle

Os programas de controle criados são exemplos de como as cenas podem ser controladas por meio de componentes externos. Os programas foram desenvolvidos utilizando a ferramenta *OpenPLC* e utilizando SFCs, *Sequential Flow Diagrams*. Estes programas podem ser escritos em outras linguagens de programação para CLPs, ou mesmo utilizando outras ferramentas que possuem comunicação *ModBus TCP*, como a própria aplicação de demonstração que está inclusa ao fazer o *download* da biblioteca *EasyModbus*.

No editor do *OpenPLC*, as variáveis de entrada nos programas de controle, são atribuídas à endereços físicos utilizando o prefixo IX% para digirais e IW para analógicas, as de saída são QX% e QW. Sendo que, como serão atribuídas utilizando *modbus*, o valor do primeiro endereço é 100.0 para as digitais e 100 para as analógicas. No lado do

Figura 18 – Visão da Cena 2 - Loader | Crane



Fonte: Autor

programa de simulação, o primeiro endereço tanto de entrada e saída, digital e analógica, é 1. Portanto o endereço da entrada digital 100.0 no programa de controle, é a *discrete input* 1 da cena.

O Programa 1 foi escrito utilizando todos os recursos disponíveis na cena. Já os Programas 2 e 3 utilizam apenas parte dos recursos da cena que fazem controle. Mostrando assim, que é possível que a mesma cena possui flexibilidade de tipos de soluções. É possível, também, que a mesma cena seja controlada por mais de um programa de controle funcionando independentes um do outro.

4.5.1 Programa 1 - Sorter

O programa de controle da Cena 1 faz o comando da cena utilizando os endereços da esteira; dos sensores da esteira principal e das rampas; dos desviadores e do *spawner*. Além dos botões *Start* e *Stop* da interface gráfica. Esses endereços são vinculados à variáveis, como mostrado na Figura 19, onde os sensores SEN1, SEN2 e SEN3 são referentes aos da esteira pricipal, e os SEN4 SEN5 e SEN6 são das rampas. Há, também, uma variável auxiliar para controle de tempo.

O fluxo do programa, Figura 20, funciona colocando todos os acionamentos em um estado seguro, desativando todos. Após isso, o sistema entra no estado *SPAWNING*, onde a esteira e o *Spawner* são ligados até que o sensor de presença seja acionado, iniciando o estado *SORTING* que para a esteira e o *Spawner*, verifica qual é o tipo de caixa detectada. De acordo com o sensor ativado, o desviador PIVOT1 ou PIVOT2 é acionado, ou nenhum dos dois é acionado, caso nem o SEN2 nem o SEN3 estejam ativados. A esteira é ligada e

Figura 19 – Variáveis do programa de controle do *Sorter*

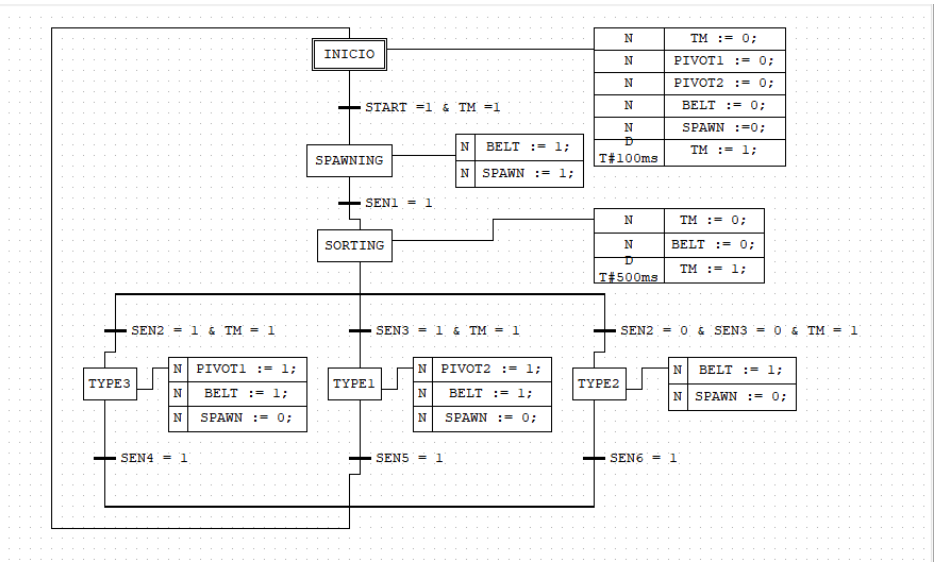
Descrição: Class Filter: Todos

#	Nome	Class	Tipo	Localização	Valor Inicial
1	SEN1	Local	BOOL	%IX100.0	
2	SEN2	Local	BOOL	%IX100.1	
3	SEN3	Local	BOOL	%IX100.2	
4	SEN4	Local	BOOL	%IX100.3	
5	SEN5	Local	BOOL	%IX100.4	
6	SEN6	Local	BOOL	%IX100.5	
7	START	Local	BOOL	%IX100.6	
8	STOP	Local	BOOL	%IX100.7	
9	BELT	Local	BOOL	%QX100.0	
10	SPAWN	Local	BOOL	%QX100.1	
11	PIVOT1	Local	BOOL	%QX100.2	
12	PIVOT2	Local	BOOL	%QX100.3	
13	TM	Local	BOOL		

Fonte: Autor

o fluxo volta ao início.

Figura 20 – Programa de controle do *Sorter*



Fonte: Autor

4.5.2 Programa 2 - Crane

O programa 2 é uma demonstração do uso do manipulador, movendo caixas da esteira principal, para a auxiliar. O programa faz uso do manipulador, do *spawner* e das esteiras, Figura 21, porém seu foco de controle está no manipulador.

O fluxo do programado do manipulador, Figura 22, começa em uma posição inicial, preparado para capturar uma caixa. Quando o sensor SEN1 é ativado, é enviado o comando para ativar a garra GRAB que captura a caixa, após um tempo de 500ms, é enviado o comando para os eixos irem para a posição de de descarga, aguardando 2000ms, a caixa é solta e retorna ao passo inicial que espera o sinal do sensor.

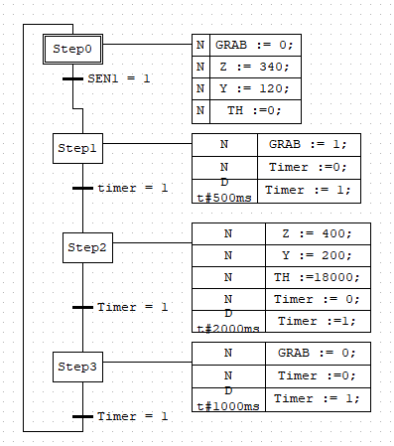
Figura 21 – Variáveis do programa de controle da Cena 2 - Manipulador

Descrição: Class Filter: Todos

#	Nome	Class	Tipo	Localização	Valor Inicial
1	SEN1	Local	BOOL	%IX100.3	
2	GRAB	Local	BOOL	%QX100.5	
3	A	Local	BOOL	%QX100.0	1
4	A0	Local	BOOL	%QX100.1	1
5	A1	Local	BOOL	%QX100.2	1
6	A2	Local	BOOL	%QX100.3	1
7	A3	Local	BOOL	%QX100.4	1
8	Z	Local	INT	%QW100	
9	Y	Local	INT	%QW101	
10	TH	Local	INT	%QW102	
11	Timer	Local	BOOL		

Fonte: Autor

Figura 22 – Programa de controle da Cena 2 - Manipulador



Fonte: Autor

4.5.3 Programa 3 - Loader

O programa 3 utiliza a balança em conjunto com o alimentador para encher as caixas do tipo copo criadas pelo *spawner*, até um valor definido pelo programa de controle. As variáveis e seus endereços podem ser visto na Figura 23.

Figura 23 – Variáveis do programa de controle da Cena 2 - *Loader*

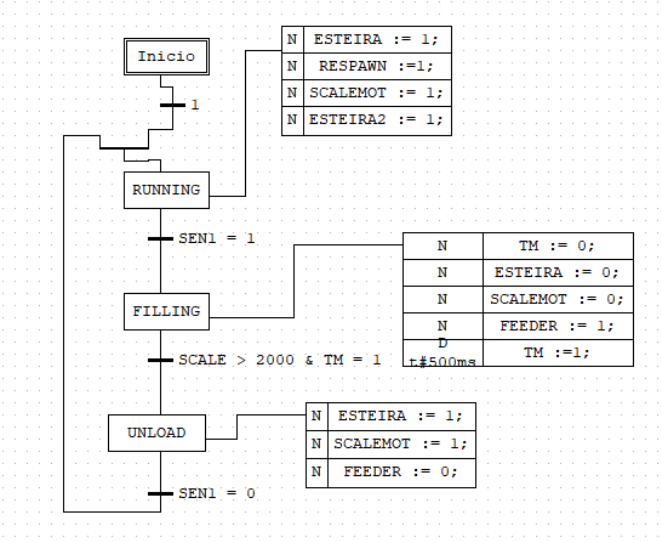
Descrição: Class Filter: Todos

#	Nome	Class	Tipo	Localização	Valor Inicial
1	ESTEIRA	Local	BOOL	%QX100.0	
2	SCALEMOT	Local	BOOL	%QX100.1	
3	ESTEIRA2	Local	BOOL	%QX100.2	
4	RESPAWN	Local	BOOL	%QX100.3	
5	FEEDER	Local	BOOL	%QX100.4	
6	SEN1	Local	BOOL	%IX100.0	
7	SEN2	Local	BOOL	%IX100.1	
8	SEN3	Local	BOOL	%IX100.2	
9	SCALE	Local	INT	%IW100	
10	TM	Local	BOOL		

Fonte: Autor

O programa 3, Figura 24, é iniciado ligando a esteira principal, o motor da balança e o *spawner*. Quando o sensor que está próximo ao alimentador é ativado, o motor da balança é desligado e enviado o comando de acionamento do alimentador, que fica ligado até que o

Figura 24 – Programa de controle da Cena 2 - *Loader*



Fonte: Autor

valor transmitido pela balança atinja o definido no programa, com isso, o alimentador é desligado e o motor da balança religado, e retorna-se ao estado inicial.

5 Resultados obtidos

A solução proposta resultou em um programa de simulação de ambiente de fábrica, utilizando a ferramenta *Unity*, que pode se comunicar com o ambiente externo por meio do protocolo *modbus*. As cenas simuladas funcionaram com êxito onde os componentes criados interagem entre si, a fim de executar os passos definidos por seus programas de controle. Há a possibilidade de posteriormente mais componentes serem criados, além melhorias serem feitas nos modelos deste trabalho, como aprimoramento gráfico e inclusão de animações.

6 Comentários finais

Visto que a solução obtida neste trabalho executa de forma satisfatória a proposta de ser uma ferramenta didática que auxilia no aprendizado de automação industrial, pode-se assumir que é um complemento às bancadas físicas que são utilizadas em laboratório. Além de ser uma alternativa economicamente viável às soluções comerciais atualmente existentes.

Referências

1 FENERICK, J. A.; VOLANTE, C. R. Evolução das indústrias, os benefícios da automação e as perspectivas do mercado da robótica no brasil e no mundo. *Revista Interface Tecnológica*, Interface Tecnologica, v. 17, n. 1, p. 734–745, jul. 2020. Disponível em: <https://doi.org/10.31510/infa.v17i1.805>. Citado na página 13.

2 TIOBE Index | TIOBE - The Software Quality Company. 2020. Disponível em: <https://www.tiobe.com/tiobe-index/>. Citado na página 13.

3 TEBANI, K. et al. Real-time communication between PLC and Dymola for virtual commissioning application. In: *2020 4th International Conference on Advanced Systems and Emergent Technologies (IC_ASET)*. Hammamet, Tunisia: IEEE, 2020. p. 83–88. ISBN 9781728163567. Disponível em: <https://ieeexplore.ieee.org/document/9318223/>. Citado 2 vezes nas páginas 15 e 16.

4 WANG, H. et al. Development of three dimensional virtual PLC experiment model based on unity3d. In: *2017 First International Conference on Electronics Instrumentation & Information Systems (EIIS)*. IEEE, 2017. Disponível em: <https://doi.org/10.1109/eiis.2017.8298660>. Citado na página 16.

5 SIEMENS. *SIMATIC S7-1500 S7-PLCSIM Advanced Function Manual*. Postfach 48 48 90026 NÜRNBERG GERMANY, 2016. Disponível em: <https://cache.industry.siemens.com/dl/files/153/109739153/att_895955/v1/s7-plcsim_advanced_function_manual_en-US_en-US.pdf>. Citado na página 16.

6 UNITY, T. *Unity Real-Time Development Platform | 3D, 2D VR & AR Engine*. 2021. Disponível em: <https://unity.com/>. Citado 2 vezes nas páginas 16 e 22.

7 VAANANEN, M.; HORELLI, J.; KATAJISTO, J. Virtual learning environment concept for PLC-programming - case: Building automation. In: *2010 2nd International Conference on Education Technology and Computer*. IEEE, 2010. Disponível em: <https://doi.org/10.1109/icetc.2010.5529409>. Citado na página 17.

8 NARAYANAN, G.; DESHPANDE, A. Learning Automation Made Easy through Virtual Labs. In: *2016 International Conference on Learning and Teaching in Computing and Engineering (LaTICE)*. Mumbai, India: IEEE, 2016. p. 60–65. ISBN 9781509025046. Disponível em: <http://ieeexplore.ieee.org/document/7743154/>. Citado 2 vezes nas páginas 17 e 18.

9 FACTORY I/O – Next-Gen PLC Training. Disponível em: <https://factoryio.com/>. Citado 2 vezes nas páginas 18 e 20.

10 MIYAGI, P. *Controle programaável : fundamentos do controle de sistemas a eventos discretos*. Saão Paulo: Edgard Blucher, 1996. ISBN 852120079X. Citado 2 vezes nas páginas 18 e 32.

11 COMOS Virtual reality and field operator training. Disponível em: <https://new.siemens.com/br/pt/produtos/automacao/software-industria/software-engenharia-comos/walkinside.html>. Citado na página 19.

12 SIEMENS, S. D. I. S. *COMOS Walkinside*. 2020. Disponível em: <https://assets.new.siemens.com/siemens/assets/api/uuid:0469b115-a980-4635-bd15-fa31e3ecfee5/comos-walkinside.pdf>. Citado na página 19.

13 FESTO. *MPS sorting station D – Combining opto and inductive sensors - MPS stations - Learning factory kits - Factory automation & Industry 4.0 - Learning Systems - Festo Didactic*. Disponível em: <https://www.festo-didactic.com/int-en/learning-systems/factory-automation-industry-4.0/learning-factory-kits/mps-stations/mps-sorting-station-d-combining-opto-and-inductive-sensors.htm#prd_det_accessories>. Citado na página 20.

14 BLOODSHED Software - Dev-C. 2020. Disponível em: <https://www.bloodshed.net/devcpp.html>. Citado na página 21.

15 VISUAL Studio Code - Code Editing. Redefined. Microsoft, 2016. Disponível em: <https://code.visualstudio.com/>. Citado na página 21.

16 THE OPENPLC PROJECT | openplcproject.com. Disponível em: <https://www.openplcproject.com/>. Citado na página 21.

17 EasymodbusTCP Modbus Library for .NET/Java and Python – Communication library and professional tools for industrial communication. Disponível em: <http://easymodbustcp.net/en/>. Citado na página 24.

18 THE Modbus Organization. Disponível em: <https://modbus.org/>. Citado na página 24.

19 SOARES, W. S. *Wessilsoares/TCC_Virtual_Env*. 2021. Original-date: 2021-07-14T00:37:29Z. Disponível em: <https://github.com/Wessilsoares/TCC_Virtual_Env>. Citado 3 vezes nas páginas 25, 35 e 40.