GABRIEL FEUERWERKER

SIMULAÇÃO COMPUTACIONAL DA LÓGICA DE CONTROLE DE UMA BANCADA DIDÁTICA

São Paulo
2021

GABRIEL FEUERWERKER

SIMULAÇÃO COMPUTACIONAL DA LÓGICA DE CONTROLE DE UMA BANCADA DIDÁTICA

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Mecatrônico.

São Paulo
2021

GABRIEL FEUERWERKER

SIMULAÇÃO COMPUTACIONAL DA LÓGICA DE CONTROLE DE UMA BANCADA DIDÁTICA

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Mecatrônico.

Orientador:

Prof. Dr. Fabrício Junqueira

São Paulo
2021

Dedicado à minha família e amigos, que me apoiaram durante a elaboração do trabalho e durante todo o curso na Escola Politécnica.

AGRADECIMENTOS

Ao orientador Prof. Dr. Fabrício Junqueira pelo seu apoio, incentivo, dedicação e orientação que foram essenciais para a realização deste projeto.

À FESTO por disponibilizar o modelo 3D utilizado neste trabalho.

Aos meu pai, Roberto Moisés Feuerwerker, à minha mãe, Tatiana Feuerwerker, e à minha irmã, Mariana Feuerwerker, por serem minha fonte de motivação e sempre acreditarem em mim, em especial nos momentos em que me faltou força durante a graduação.

À minha namorada, Maria Clara Santos Braune, por incentivar o melhor em mim e por entender os momentos em que me ausentei por dedicação a este trabalho.

Aos docentes da Escola Politécnica, em especial aos que me instruíram ao longo do quarto e quinto ano, que se desdobraram para manter as atividades escolares da melhor forma possível durante o atual cenário pandêmico.

Aos meus colegas de sala pelo companheirismo ao longo de toda esta jornada e que sempre me estenderam a mão quando foi preciso.

RESUMO

A implementação do código de controle do CLP na bancada MPS FESTO na disciplina *PMR3305 - Sistemas a Eventos Discretos* - é o momento em que o aluno tem a possibilidade de testar e visualizar o resultado do seu trabalho. Entretanto, as bancadas só podem ser acessadas no laboratório da POLI, o que foi dificultado durante a pandemia do novo Corona Vírus. Neste contexto, o objetivo desse trabalho é projetar e implementar um modelo virtual de uma das bancadas MPS FESTO e uma sistemática para os alunos usarem juntamente ao *CodeSys*, de forma que possam testar a lógica de controle desenvolvida em SFC - *Sequential Function Charts* - antes de ser implementado presencialmente no laboratório de *PMR3305 - Sistemas a Eventos Discretos*.

Palavras-Chave – Petri Net, OPC-UA Protocol, Visual Components, FESTO Workstation, CodeSys, Digital twin, Programming CLP

ABSTRACT

The implementation of the control code of CLP in the MPS FESTO bench during the PMR3305 discipline - Systems to Discrete Events - is a moment where the student has the possibility to test and visualize the results of his own work. Nonetheless, the benches can only be acessed in a POLI laboratory, which has been hampered during the pandemic of the Corona Virus. In this scenario, the objective of this work is to simulate virtually the MPS FESTO benches, in a way that the code developed in SFC - Sequential Function Charts - that refers to the control system, can be studied and validated before being implemented in person in the PMR3305 laboratory - Systems to Discrete Events.

Keywords – Petri Net, OPC-UA Protocol, Visual Components, FESTO Workstation, CodeSys, Digital twin, Programming CLP

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Com a evolução da computação, através de *hardwares* e *softwares* com velocidades de processamento e armazenamento cada vez maiores, a humanidade foi capaz de resolver problemas complexos de forma mais rápida, como explicado por (HARARI, 2015). Uma das formas de se resolver problemas complexos é por meio da simulação computacional, como em (NEUMANN, 1947) que é uma das primeiras publicações sobre o tema. Nela os autores abordam o uso da simulação computacional para resolver problemas probabilísticos, no caso através de um método - *Monte Carlo Simulation*.

Com o passar dos anos, o desenvolvimento da computação possibilitou uma amplitude maior de opções para simulações em diferentes áreas do conhecimento. Dentre esta gama de aplicações começou-se a utilizar o recurso das simulações computacionais para propósitos educacionais, tanto em ensino superior quanto básico, como pode ser observado em (FONTOURA, 2019), onde é proposto um simulador do processo de produção de cerveja a ser utilizado pelos alunos de graduação e pós-graduação de engenharia química da Universidade de Vassouras. A intenção é que os alunos possam acompanhar as respostas do processo de produção da cerveja sem a necessidade de implementá-lo, economizando tempo e investimento financeiro.

1.1 CONTEXTO

Com a pandemia causada pelo novo Coronavírus (SARS-Cov-2) e a adoção do distanciamento social proposta por (DORIA, 2020) para o Estado de São Paulo, diversas atividades presenciais - como comércio e ensino - foram impactadas e precisaram ser reinventadas. No varejo, houve um aumento na demanda do chamado comércio eletrônico, ou *e-commerce*, como explicado por (ALVARENGA, 2021). Na educação, escolas e faculdades tiveram suas portas fechadas, como foi o caso da Universidade de São Paulo (USP) desde o anuncio de (AGOPYAN, 2020).

Na Escola Politécnica da USP (POLI), para não comprometer o calendário escolar

e o aprendizado dos alunos, as disciplinas tiveram que se adaptarem ao modelo virtual de ensino. Aulas expositivas puderam ser substituídas por videoconferências utilizando ferramentas como *Google Meets* e *Zoom*. Provas e entregas de trabalho foram realizadas por plataformas digitais de ensino, como o *e-disciplinas*. Porém, aulas práticas, como experiências laboratoriais, exigiram maior expertise por parte dos docentes devido à dificuldade de se adaptar uma atividade laboratorial prática em uma atividade virtual sem comprometer o aprendizado.

A disciplina *Sistemas a Eventos Discretos* (PMR3305) lecionada no sexto semestre do curso de graduação em engenharia mecatrônica da POLI é dividida em parte teórica e prática como descrito em (USPDIGITAL, 2016). De forma resumida, na parte teórica são apresentados aos alunos conceitos fundamentais de sistemas a eventos discretos, modelagem de sistemas de controle sequenciais, Redes de Petri para lógica de controle e metodologia de projeto de sistemas de controle. Enquanto que na parte prática, os alunos participam da atividade de construção de modelos de sistemas de automação e análise destes modelos por simulação discreta, desenvolvimento de programas de controle para controladores programáveis e, por fim, teste destes programas nas bancadas *MPS FESTO*. Estas bancadas didáticas simulam uma pequena linha de produção e é nesta etapa em que os alunos visualizam o resultado do trabalho de estruturar, modelar e implementar a lógica de controle; sendo este um momento de aprendizado e satisfação dos alunos da disciplina.

No cenário de pandemia apresentado, com aulas remotas, o corpo docente de PMR3305 teve a tarefa de lecionar a disciplina de forma virtual. Nos anos de 2020 e 2021, a parte prática consistiu em modelar as Redes de Petri com base nos requisitos das bancadas e simular a Rede de Petri no *software PIPE* para verificação das propriedades e existência de travamentos. A validação final, que costumava ser a implementação nas *MPS FESTO* em laboratório, foi realizada pelo professor da disciplina que, por conta da experiência, já sabia se a Rede de Petri condizia com o funcionamento da bancada ou não.

Em meio a este cenário, os alunos de PMR3305 não tiveram a experiência de testar e visualizar o funcionamento da lógica de controle implementada na bancada, uma vez que estavam restritos às aulas remotas.

1.2 MOTIVAÇÃO

O trabalho desenvolvido apresenta motivação acadêmica e pessoal por parte do autor que, como aluno do curso de engenharia mecatrônica da Escola Politécnica da USP, afetado pelas dificuldades impostas pelo ensino à distância durante a pandemia do SARS-Cov-2, gostaria de deixar aos demais alunos desta Escola um trabalho que possa ser explorado e desenvolvido visando a melhoria contínua da qualidade de ensino da POLI e da formação de colegas engenheiros.

1.3 OBJETIVOS

Este trabalho foi desenvolvido visando projeto e implementação de uma ferramenta que possibilite os alunos de PMR3305 a realizarem atividades laboratoriais sem a necessidade da presença física na Escola.

Ao longo do texto será apresentado o processo de desenvolvimento de um sistema funcional da simulação computacional de uma bancada *MPS FESTO*, de modo que os discentes possam testar a implementação da lógica de controle em linguagem SFC (*Sequential Function Chart*), através da visualização do funcionamento de um modelo animado da bancada *MPS FESTO*.

1.4 ORGANIZAÇÃO DO TEXTO

No capítulo 2 é apresenta a revisão bibliográfica sobre as bases que apoiam este projeto.

No capítulo 3 é feita a descrição do projeto, detalhando o sistema proposto.

No capítulo 4 é feita a descrição da implementação do projeto.

Por fim, no capítulo 5 é apresentada a conclusão do projeto.

2 REVISÃO BIBLIOGRÁFICA

Para alcançar os objetivos deste trabalho, foi realizada uma revisão bibliográfica sobre tópicos da disciplina PMR3305, como controladores lógico programáveis e suas principais linguagens de programação. Também foram estudados os *softwares CodeSys* e *Visual Components*, por serem as ferramentas de implementação do sistema. Além do estudo sobre as bancadas MPS FESTO, que é o objeto a ser digitalizado.

2.1 CONTROLADORES LÓGICO PROGRAMÁVEIS

Existem registros de máquinas que operam com controle de sistema sequencial desde o século XVIII, como citado em (MIYAGI, 1996), as máquinas de tear automáticas com cartões perfurados ou uma moenda automática por esteira já são exemplos de sistemas a eventos discretos.

No início, o controle de SED poderia ser reduzido por um "operador", um "dispositivo de controle" e um "objeto de controle". Com a evolução tecnológica, a partir dos anos 50, os SED passaram a trabalhar com conceitos de "monitoração" e "atuação". Ou seja, o "operador" irá acompanhar o sistema por um painel de monitoramento e o objeto de controle será alterado por um sistema de atuação, que responde a um dispositivo de controle.

Em 1968, a divisão *Hydramatic* da *General Motors* (Estados Unidos) divulgou uma especificação técnica de dez itens (TODORA, 2009) para buscar empresas com interesse de produzir o controlador programável. São eles:

- Os controladores devem ser facilmente programáveis, com operações sequenciais facilmente alteráveis;
- Devem ser de fácil manutenção;
- Devem possuir características operacionais de alta confiabilidade;

- Devem possuir dimensões menores que os painéis à relés para diminuição de gastos;
- Deve ser apto a mandar dados para um sistema central;
- Deve ter preço competitivo em relação aos dispositivos à relés;
- Deve receber sinais de entrada na ordem de 115V CA;
- Deve ser capaz de enviar sinais de saída de 115V CA;
- Devem possibilitar expansões na forma de módulos para atender sistemas de maior porte;
- Cada unidade deve possibilitar a expansão de no mínimo 4000 palavras na memória do programa.

Assim, em 1969, a *Bedford Associates* (Estados Unidos), lançou o primeiro Controlador Lógico Programável chamado 084 (BALL, 2015) seguindo as especificações divulgadas pela divisão *Hydramatic*. A partir dos anos 70, as novas gerações de controladores que seguiam essas especificações foram batizados de CLP's - Controladores Lógico Programáveis.

Como explicado em (MARTINS, 2020), um CLP é um dispositivo eletrônico para automação. Composto por uma C.P.U., memória e dispositivos (Input/Output), o CLP é programável afim de realizar tarefas de inter travamento, temporização, contagem, operações matemáticas, controle em malha aberta ou malha fechada; podendo controlar sistemas industriais complexos.

Como explicado em (LEWIS, 2011), com a difusão dos CLP's surgiram diversas normas para padronizar a programação destes controladores, como a francesa NFC-03-190 e a alemã DIN 40719-6. Entretanto, nenhuma destas normas foi tão impactante quanto a IEC61131 da *Electrotechnical Comission*, a primeira a receber aceitação na industria mundial. Para seu desenvolvimento foram montados times de especialistas, conforme (IEC, 2003). Como indicado na Tabela 1, o time 3 ficou responsável pela tarefa de desenvolver e padronizar as linguagens de programação dos CLP's, o que ficou conhecido como norma IEC 61131-3.

Tabela 1: Partes da Norma IEC 61131

Time	Título	Conteúdo
1	*General Information*	Terminologias e conceitos
2	*Equipment requirements and tests*	Verificação e fabricação mecânica e eletrônica
3	*Programmable Languages*	Estrutura das linguagens do CLP
4	*User guidelines*	Guia para escolha, uso e manutenção do CLP
5	*Communications*	Conectividade com outros dispositivos
6	*Reserved*	Está reservado
7	*Fuzzy Control Programming*	Funcionalidades de software
8	*Guidelines for the Application and Implementation of Programming Languages*	Guias para implementar as linguagens da IEC 1131-3

2.2 LINGUAGENS DE PROGRAMAÇÃO PARA CLP'S

A normal IEC 61131-3 (IEC, 2003) separa as linguagens em dois grupos: Linguagens Textuais e Linguagens Gráficas. As Linguagens Textuais padronizadas são a IL (*Instruction List*) e a ST (*Structured Text*). A IL é uma lista simples de comando, executando o conjunto de funções declaradas de forma sequencial. Já a ST é mais complexa, sendo uma linguagem de alto nível capaz de estruturar programas com processamentos numéricos, operações de comparação e comandos. As Linguagens Gráficas padronizadas são a LD (*Ladder Diagram*) e a FBD (*Function Block Diagram*). A LD é uma linguagem bastante difundida e se assemelha à notação de diagramas de lógicas de relés. Já a FBD é constituída de blocos de funções que processam fluxos de dados.

A disciplina PMR3305 utiliza a linguagem SFC (*Sequential Function Chart*) para automação das Bancadas FESTO. Como discutido em (MIYAGI, 1996), a SFC não é considerada uma linguagem e sim um elemento comum de descrição pela IEC 61131-3. Entretanto, por ser tratada como linguagem pelos principais *softwares* para programação de CLP's - como o CodeSys - e por ser mencionada por (MIYAGI, 1996) como uma linguagem gráfica de programação, este trabalho irá se referir à SFC como linguagem.

A SFC é uma linguagem de programação gráfica baseada em GRAFCET, na qual o

programador consegue definir o comportamento sequencial do sistema de controle. Por ser uma linguagem de fácil utilização, ter poder de descrever problemas dinâmicos e por evitar ambiguidade, (BONFATTI; MONARI; SAMPIERE, 1999) descreve a SFC como uma das principais linguagens.

2.3 CODESYS

Criado pela empresa S3 – *Smart Software Solutions*, o CodeSys é um software de controle e automação de CLP's. Este programa possui uma versão gratuita e também é utilizado em cursos de graduação da Escola Politécnica da Universidade de São Paulo. Contudo, não apresenta uma interface gráfica tri-dimensional que permita a observação dos processos.

O trabalho de (VOGEL et al., 2015) apresenta uma implementação completa desta ferramenta juntamente com um simulador para criar um mecanismo de manejo de placas de cultura utilizadas em laboratório por meio da lógica de controle de Redes de Petri.

O CodeSys permite o interfaceamento com softwares de animações 3D, como o Visual Components, por possuir um servidor para o estabelecimento de comunicação OPC-UA.

2.4 VISUAL COMPONENTS

O *Visual Components* é um *software* para simulação de manufaturas 3D. Segundo o site institucional (COMPONENTS, 2021), o *software* foi desenvolvido em 1999 por Scott Walter, Mika Anttila e Juha Renfors com o objetivo tornar a simulação de manufaturas 3D mais acessível às empresas do setor.

Em 2016, o *software* recebeu uma atualização, sendo renomeado como *Visual Components 4.0*. Nessa nova versão, a aplicação conta com uma *Python API* para programação da simulação e compatibilidade para estabelecer comunicação com *server's* OPC-UA.

O *Visual Components 4.0* existe apenas em versões pagas, existindo um servidor de licença dentro da Escola Politécnica para uso educacional.

2.5 PROTOCOLO DE COMUNICAÇÃO OPC-UA

O protocolo de comunicação OPC-UA é um padrão para troca de dados dentro do ambiente de automação industrial, desenvolvido por (OPC-FOUNDATION, 2021). Para funcionamento desta especificação é necessário estabelecer um servidor - *OPC server* - e seus clientes - *OPC Client* - sendo viável interagir com mais de um cliente simultaneamente.

Como explicado em (OPC-FOUNDATION, 2021), inicialmente o padrão foi desenvolvido apenas para sistemas *Windows*. Tal restrição fez com que a *OPC Foundation* aprimorasse o protocolo, criando a especificação OPC-UA, a qual apresenta uma estrutura de plataforma aberta, sendo compatível com qualquer sistema.

Para mais informações sobre o protocolo de comunicação OPC-UA, pode-se consultar o caso de uso (ROSSOW, 2018) onde foi realizado a implementação da comunicação entre duas ferramentas de modelagem, o OpenModelica e o Xcos. Ou pode-se consultar o próprio site (OPC-FOUNDATION, 2021)

2.6 BANCADA MPS FESTO

As bancadas MPS FESTO simulam um modulo de produção individual, onde cada um apresenta uma função diferente. As bancadas pode ser consultadas na Tabela 2, construída com base nas informações dadas em (POLA, 2013). Vale pontuar que estão presentes no laboratório de *Sistemas a Eventos Discretos* apenas as bancadas 1,2,3,5 e 7.

Tabela 2: Descrição das FESTO MPS *stations*

Bancada	Nome	Função
1	*Distributing*	Fornecer peças de trabalho
2	*Testing*	Realiza a verificação da peça de acordo com critérios estabelecidos
3	*Processing*	Transporta as peças por uma mesa giratória, realiza a indexação da peça e realiza uma simulação de usinagem no orifício da peça
4	*Buffer*	Armazenar até 5 peças
5	*Sorting*	Realiza a classificação das peças e as separa por tipo ou por cor
6	*Separating*	Realiza a separação de peças, baseado na profundidade do orifício, ou na altura da peça
7	*Handling*	Realiza a manipulação de peças
8	*Pick and Place*	Manipula peças do tipo medidores
9	*Fluidc Muscle Press*	Realiza o controle de pressão exercido pelo músculo pneumático
10	*Punching*	Responsável por furar a tampa do cilindro,

Cada bancada é composta por um controlador, um painel de controle e uma planta. Considerando o principal objetivo deste trabalho, que é o desenvolvimento de um sistema funcional de simulação computacional de uma das bancada, os principais objetos de estudo nas bancadas são os sensores, atuadores e sinais gerados. Para isso foram levantados, com base no (POLA, 2013), os atuadores, sensores e sinais da bancada Handling, que será o módulo de implementação do trabalho.

O objeto do trabalho será o módulo Handling, responsável por manipular as peças entre células através de uma garra. Conforme Figura 1, o módulo Handling é composto por uma garra com movimentação de translação em dois eixos, sendo possível deslocar a peça de trabalho do Módulo *buffer* à rampa ou à próxima estação.

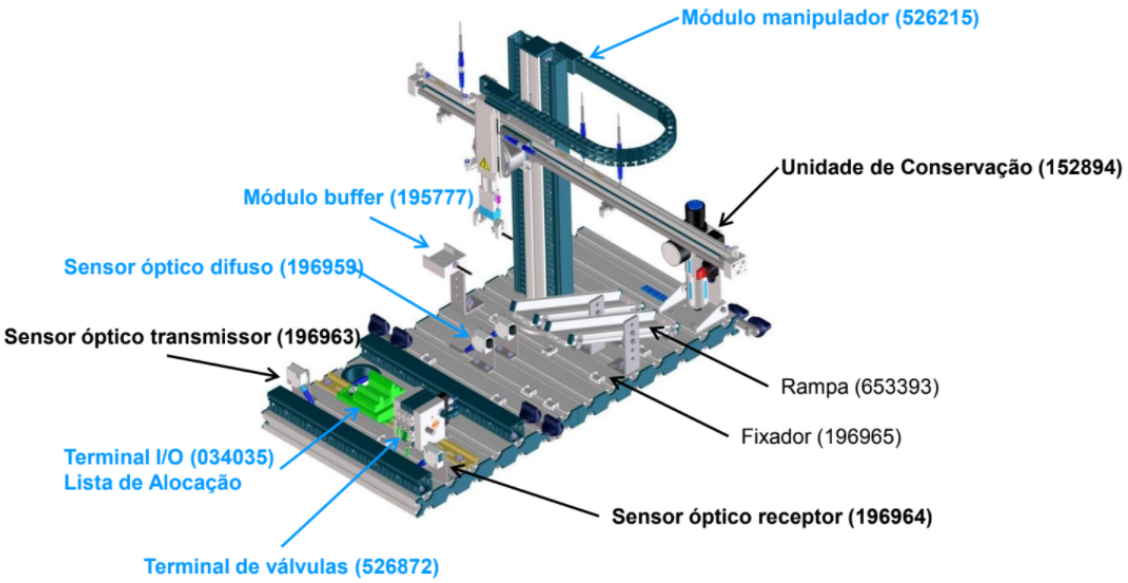

Figura 1: Módulo Handling

FONTE: (POLA, 2013)

A Figura 2 mostra os sinais de entrada e saída do módulo Handling

Terminal de I/O Entradas digitais (IN)	Descrição	Terminal de I/O Saídas digitais (OUT)	Descrição
DI 0	Peça disponível no buffer	DO 0	Move módulo Handling para próxima estação
DI 1	Módulo Handling na posição buffer	DO 1	Recua módulo Handling para estação anterior
DI 2	Módulo Handling na posição próxima estação	DO 2	Avança módulo da garra
DI 3	Módulo Handling na posição da rampa	DO 3	Abre garra
DI 4	Garra avançada	DO 4	
DI 5	Garra recuada	DO 5	
DI 6	Sensor da garra	DO 6	
DI 7	Próxima estação livre	DO 7	Estação ocupada

Figura 2: Sinais Handling

FONTE: (POLA, 2013)

Já a Tabela 3 relaciona os sensores da planta com os sinais de entrada.

Tabela 3: Sinais de entrada e sensores

Sinal	Sensor
DI 0	Sensor de posição no buffer
DI 1	Sensor de posição no curso horizontal da garra perpendicular ao buffer
DI 2	Sensor de posição no curso horizontal da garra perpendicular à próxima estação
DI 3	Sensor de posição no curso horizontal da garra perpendicular à rampa
DI 4	Sensor de posição para medir o avanço da garra
DI 5	Sensor de posição para medir recuo da garra
DI 6	Sensor óptico na garra que identifica se a peça de trabalho possui cor ou não (cor preta)
DI 7	Sinaliza se a próxima estação de trabalho está livre

3 METODOLOGIA

O sistema deve simular o funcionamento da bancada MPS FESTO. Para isso, deve existir um modelo capaz de simular os atuadores, sensores e sinais de entrada e saída da bancada; e que troque informações com o *software* em que a lógica de controle do CLP está implementada.

3.1 REQUISITOS

Dado que o objetivo do trabalho é simular o funcionamento da bancada MPS FESTO, o projeto deve:

- Trabalhar com um modelo 3D similar à bancada física *Handling* existente no laboratório de *Sistemas a Eventos Discretos*.
- Reproduzir a animação da movimentação da bancada
- Gerar sinais de saída para o emulador do CLP, similares aos dos sensores reais da bancada, detectando peça de trabalho ou posições pré estabelecidas dos atuadores.
- Transformar sinais de controle, provenientes do emulador do CLP, em movimento dos atuadores ou peça de trabalho.

3.2 DESCRIÇÃO DO SISTEMA

Primeiro, deve-se desenvolver o programa da lógica de controle do CLP da bancada escolhida. Para isso, pode-se primeiro modelar a lógica de funcionamento em Rede de Petri para identificar a existência de travamentos.

Para receber e enviar sinais para a animação, deve-se estabelecer um servidor OPC-UA. No caso da utilização do *CodeSys* como *software* de programação do CLP, basta adicionar a aplicação *Symbol Configuration*, selecionar as variáveis correspondentes aos

sinais de entrada e saída do controlador e compilar. Em seguida, deve-se adicionar a aplicação *Matrikon FLEX OPC UA server* e configurar os parâmetros desejáveis para se estabelecer a comunicação OPC-UA, como a interface de *Ethernet*, endereço de IP, número da porta a ser utilizada, número de sessões requeridas, taxa de ciclo do servidor, politica de segurança e o tipo de autentificação que será requerida dos clientes OPC.

No programa em que a animação foi implementada, deve-se solicitar acesso ao servidor OPC-UA. No caso da utilização do *Visual Components*, deve-se ir na aba de conectividade, adicionar um novo servidor OPC-UA, indicar o endereço do servidor desejado, informar o tempo de leitura desejado e parear as variáveis do programa onde a lógica de controle foi implementada com os sinais configurados no modelo 3D da bancada. Por fim, a animação pode ser inicializada.

Para sumarizar, a estrutura do sistema descrito está representada na Figura 3.

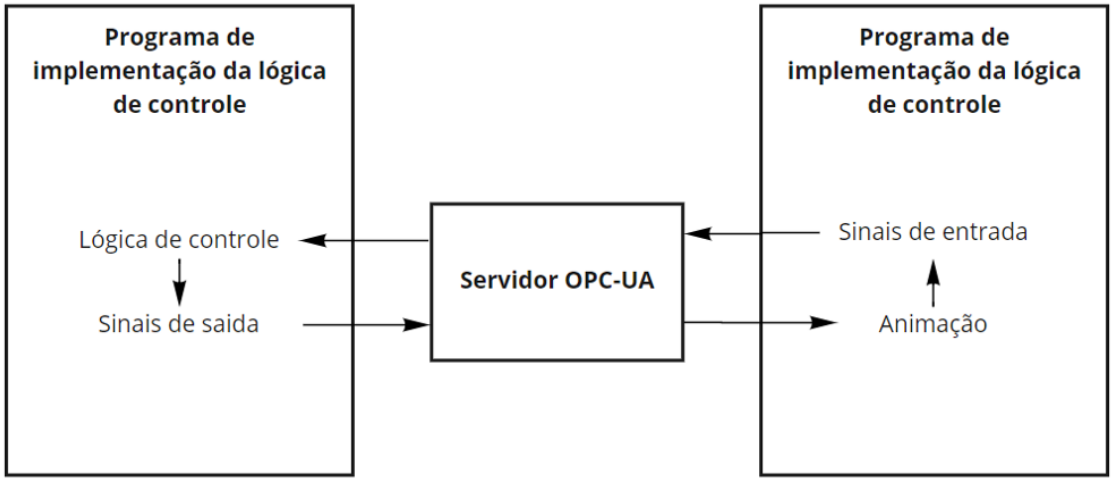

Figura 3: Estrutura do sistema descrito

4 IMPLEMENTAÇÃO DO PROJETO

Para implementação do projeto foram utilizados os *software Visual Components, CodeSys e Matrikon*. No *Visual Components* foi modelado a bancada *Handling* com seus respectivos sensores e atuadores. Também foi nesse *software* que foi implementado o código em *Python* que gera a animação 3D do modelo. No *CodeSys* foi implementado a lógica de controle em SFC. Já o *Matrikon* foi utilizado para monitorar os sinais trocados entre o *CodeSys* e o *Visual Components* por protocolo OPC-UA, com a finalidade de facilitar a identificação dos erros durante a implementação.

4.1 IMPLEMENTAÇÃO DA LÓGICA DE CONTROLE

A implementação da lógica de controle foi realizada no *Codesys*, através da linguagem gráfica SFC. A escolha do *software* foi baseada sob duas abordagens: proximidade com a experiencia real do laboratório, uma vez que este é o programa utilizado em sala de aula; e compatibilidade para estabelecer comunicação OPC-UA em sua mais nova versão. A versão 2.3 do *Codesys* é a utilizada pelos alunos na matéria PMR3305. Porém, neste projeto, será utilizada a versão 3.5, também compatível com o controlador utilizado na disciplina, o CPX-CEC. Entretanto, apenas a versão 3.5, possui um servidor para comunicação OPC-UA.

4.1.1 O PROGRAMA EM SFC

O programa desenvolvido é uma simplificação do programa original disponibilizado pela FESTO, dado que o foco da simulação computacional proposta neste projeto é a movimentação dos atuadores responsáveis pela movimentação do módulo, sendo ignorados os sinais relativos a LED's, botões e telas. Sendo assim, os sinais utilizados no programa são apresentados na Tabela 4.

Tabela 4: Sinais utilizados no programa

Sinal	função
DI 0	Indica a disponibilidade da peça de trabalho no *buffer*
DI 1	Indica que o módulo esta na posição do *buffer*
DI 2	Indica que o módulo esta na posição da próxima estação
DI 3	Indica que o módulo esta na posição da rampa
DI 4	Indica que a garra esta avançada
DI 5	Indica que a garra esta recuada
DI 6	Indica se a peça de trabalho possui cor (não é preta)
DI 7	Indica que a próxima estação de trabalho está livre
DO 0	Move o módulo para a próxima estação
DO 1	Recua o módulo para a estação anterior
DO 2	Avança a garra
DO 3	Abre a garra
DO 7	Indica que a estação Handling está ocupada

O programa desenvolvido conta com 13 estados. O primeiro deles é o *Init*, o estado de inicialização do programa, em que DO1 e DO7 são falsos. Com o sinal DI 0 verdadeiro - disponibilidade da peça de trabalho no *buffer* - o programa avança ao *passo0*. No *passo0* DO2 e DO7 são verdadeiros, gerando o comando de avanço da garra e indicando que a estação esta ocupada. Quando o sinal DI4 for verdadeiro - indicando que a garra está avançada - o programa entra no *passo1*. No *passo1* DO3 é falso, encerrando o comando de abertura da garra. Depois de 1 segundo, o programa entra no *passo2*, com DO2 falso, encerrando o comando de avanço da garra. Com DI5 verdadeiro, indicando o recuo completo da garra, o programa entra no *passo3*. A estrutura do programa descrito até o *passo3* pode ser verificada na Figura 4.

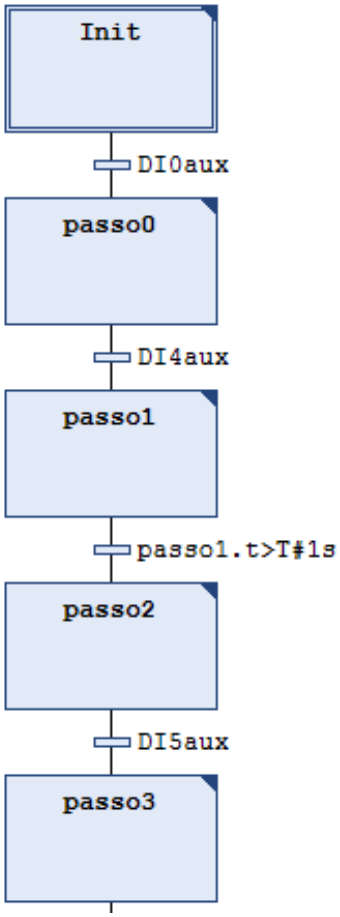

Figura 4: Estrutura do programa descrito até o *passo3*

Após o *passo3* existe duas transições. Se DI6 for verdadeiro, indicando cor (não preta) na peça de trabalho, o programa entra no *passo11*. Caso contrario, com DI6 falso, indicando não cor (preta) na peça de trabalho, o programa entra no *passo4*. Tanto no *passo11* quanto no *passo4* DO0 é verdadeiro, gerando o comando para movimentação do módulo para a próxima estação. A transição do *passo11* para o *passo12* é o sinal DI2, o qual indica que o módulo está na posição da próxima estação. Já a transição do *passo4* para o *passo5* é o sinal DI3, que indica que o módulo está na posição da rampa. Tanto no *passo12* quanto no *passo5* DO0 é falso, encerrando o sinal que movimenta o módulo para a próxima estação. A transição do *passo12* para o *passo6* é a negação do sinal DI7, indicando que a próxima estação de trabalho está livre. Já a transição do *passo5* para o *passo6* é sempre verdadeira, uma vez que não existe uma condição que impeça a soltura da peça de trabalho na rampa. A estrutura do programa descrito do *passo3* ao *passo6* pode ser verificada na Figura 5.

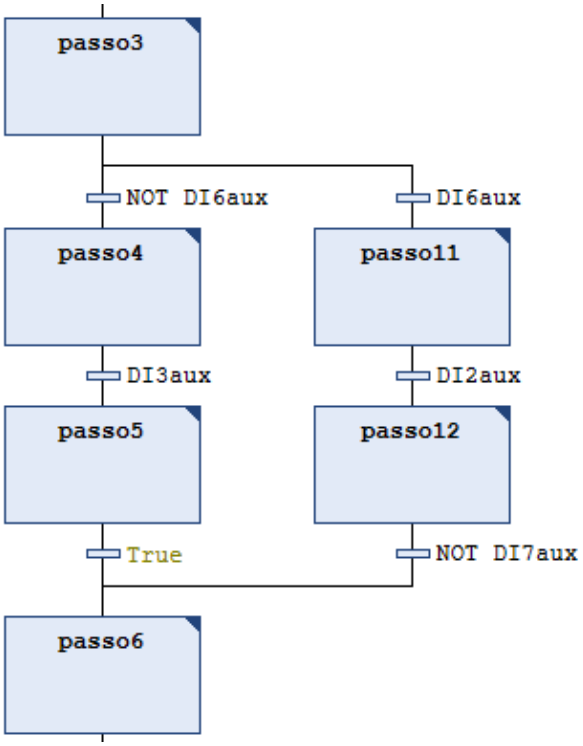

Figura 5: Estrutura do programa descrito do *passo3* ao *passo6*

No *passo6* DO2 é verdadeiro, gerando o sinal de avanço da garra. Com DI4 verdadeiro, sinal gerado quando a garra está avançada, O programa entra no *passo7*, gerando DO3 verdadeiro, solicitando a abertura da garra. Depois de 1 segundo o programa entra no *passo9*, onde DO2 é falso, parando o comando de avanço da garra. Com DI5 verdadeiro, indicando o recuo da garra, o programa entra no *passo10*, onde DO1 é verdadeiro, recuando a módulo para a estação anterior. Com DI1 verdadeiro, indicando que o módulo está na posição do *buffer*, o programa retorna ao estado de *Init*. A estrutura do programa descrito do *passo6* ao *passo12* pode ser verificada na Figura 6.

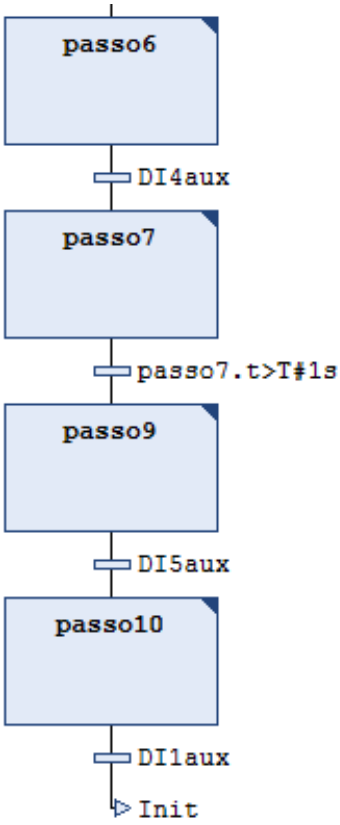

Figura 6: Estrutura do programa descrito do *passo6* ao *passo12*

4.2 SIMULAÇÃO 3D

Para a simulação 3D da Bancada FESTO, foi escolhido o *Visual Components*. Este é um *software* projetado para a modelagem e simulação de modelos 3D, com grande aplicação na industria para planejamento de *layout*, simulação de produção e verificação do funcionamento de CLP's.

4.2.1 MODELO UTILIZADO

Para a simulação gráfica, foi utilizado o modelo em CAD do módulo *Handling* disponibilizado pela FESTO. As peças de trabalho foram construídas com base nas dimensões reais das utilizadas em laboratório. Também foi construída uma bancada para apoio da peça de trabalho antes desta entrar no *buffer*. O modelo final utilizado pode ser verificado na Figura 7 (vista frontal), Figura 8 (vista lateral) e Figura 9 (vista superior).

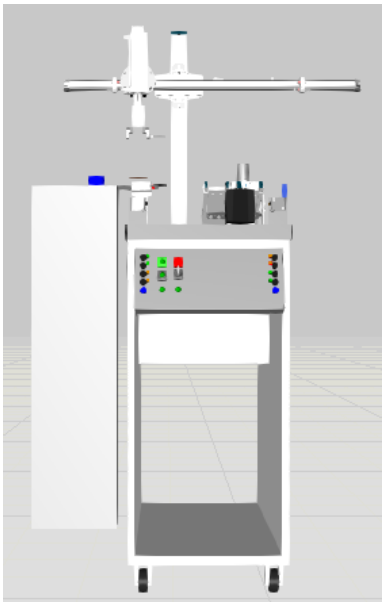

Figura 7: Vista frontal do modelo final utilizado

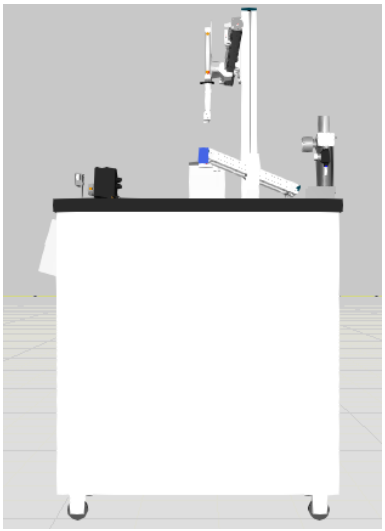

Figura 8: Vista lateral do modelo final utilizado

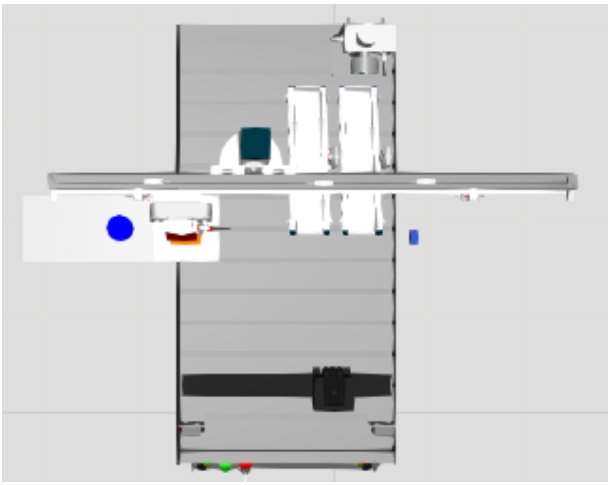

Figura 9: Vista superior do modelo final utilizado

4.2.2 ESTRUTURA DA ANIMAÇÃO

A estrutura da animação é composta pela definição de articulações, servos controladores, sensores e sinais. O comportamento da estrutura é definida em código de *Python*.

4.2.2.1 ARTICULAÇÕES

Foram quatro articulações definidas para a animação do modelo:

- Esteira_Garra: movimento no trilho, deslocando do *buffer* à próxima estação.
- Elevador_Garra: movimento de sobe e desce do braço.
- Garra_1: movimento de abre e fecha de um dos dentes da garra.
- Garra_2: movimento de abre e fecha do outro dente da garra.

A articulação Esteira_Garra é a maior na hierarquia de movimentação, ou seja, todas as outras articulações se movimentam em relação à esta. Em seguida vem a articulação Elevador_Garra, contendo o movimento das articulações Garra_1 e Garra_2.

4.2.2.2 SERVOS CONTROLADORES

Foram definidos três servos controladores para movimentar as articulações:

- Servo_Esteira: controla o movimento da articulação Esteira_Garra.
- Servo_Elevador: controla o movimento da articulação Elevador_Garra.
- Servo_Garra: controla o movimento das articulações Garra_1 e Garra_2.

4.2.2.3 SENSORES

Foram definidos sete sensores para gerar os sinais de entrada do controlador:

- Sensor_Disponível: verifica a disponibilidade da peça de trabalho no *buffer*.
- Sensor_Buffer: verifica se o braço está na posição do *buffer*.
- Sensor_Prox_Est: verifica se o braço está na posição da próxima estação.
- Sensor_Rampa: verifica se o braço está na posição da rampa.
- Sensor_Garra_Avan: verifica se o braço está avançado.
- Sensor_Garra_Recu: verifica se o braço está recuado.
- Sensor_Cor: verifica se a peça de trabalho tem cor (não preta).

Todos os sensores implementados no projeto são sensores de volume, onde deve ser definida uma área de leitura do sensor, o intervalo de medição, a necessidade de se identificar algum material especifico e qual é o tipo de sinal atrelado. Todos os sensores foram implementados com o mesmo intervalo de medição, no caso 0.01 segundos. O Sensor_Cor está programado para detectar apenas peças de cor azul.

Com relação a areá de atuação, ela é definida por dois eixos de coordenadas, se formando entre os eixos X e Y.

A Figura 10 mostra a posição das coordenadas globais de referencia utilizadas no projeto.

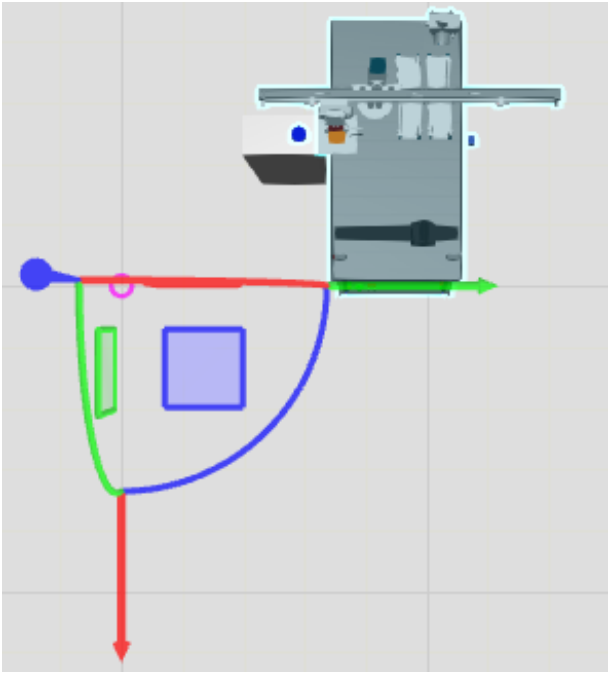

Figura 10: Coordenadas globais de referencia do projeto

A Tabela 5 descreve a posição dos dois planos cartesianos de cada sensor com relação às coordenadas globais.

Tabela 5: Posição dos planos cartesianos com relação às coordenadas globais

Plano cartesiano	Sensor correspondente	X	Y	Z	Rx	Ry	Rz
DIO_1	Sensor_Disponível	-399	695	922	0	0	0
DIO_2	Sensor_Disponível	-351	741	922	0	0	180
DI1_1	Sensor_Buffer	-420	718	1163	0	0	0
DI1_2	Sensor_Buffer	-362	718	1163	0	0	0
DI2_1	Sensor_Prox_Est	-420	1082	1163	0	90	0
DI2_2	Sensor_Prox_Est	-362	1082	1163	0	0	0
DI3_1	Sensor_Rampa	-420	907	1163	0	90	0
DI3_2	Sensor_Rampa	-362	907	1163	0	90	0
DI4_1	Sensor_Garra_Avan	-375	568	921	0	0	0
DI4_2	Sensor_Garra_Avan	-375	1254	921	0	0	0
DI5_1	Sensor_Garra_Recu	-367	542	1110	0	0	0
DI5_2	Sensor_Garra_Recu	-367	1254	1110	0	0	0
DI6_1	Sensor_Cor	-367	542	1110	0	0	0
DI6_2	Sensor_Cor	-375	1254	921	0	0	0

4.2.2.4 SINAIS

Foram definidos sete sinais de entrada:

- DI0: sinal booleano atrelado ao Sensor_Disponível.
- DI1: sinal booleano atrelado ao Sensor_Buffer.
- DI2: sinal booleano atrelado ao Sensor_Prox_Est.
- DI3: sinal booleano atrelado ao Sensor_Rampa.
- DI4: sinal booleano atrelado ao Sensor_Garra_Avan.
- DI5: sinal booleano atrelado ao Sensor_Garra_Recu.
- DI6: sinal booleano atrelado ao Sensor_Cor.

Importante pontuar que não foi implementado o sinal relativo ao DI 7 da lógica de controle implementada no *CodeSys*. Dado que o projeto é referente à simulação de apenas uma bancada, o sinal DI7 será sempre falso, simulando como se a próxima bancada estivesse sempre disponível.

4.2.2.5 PYTHON SCRIPT

A ultima etapa da animação 3D consiste em desenvolver um código em *Python* para programar os movimentos da bancada.

A biblioteca utilizada foi a *vcScript* para utilizar os métodos de animação do *Visual Components*. No começo do código foram definidos os objetos.

O programa principal roda dentro da função *OnRun()* em laço até que a aplicação do *Visual Components* se encerre. Os principais blocos do programa estão comentados para facilitar a compreensão do código implementado.

O código desenvolvido pode ser encontrado acessando este link.

4.3 COMUNICAÇÃO OPC-UA

Para que o código implementado em SFC no *CodeSys* possa gerar uma lógica de controle para a o modelo virtual da bancada MPS FESTO, é preciso estabelecer uma comunicação entre o *CodeSys* e o *Visual Components*.

Para implementação da comunicação OPC-UA foi necessário configurar o servidor OPC-UA no *CodeSys*, instalar o programa de interface de leitura de variáveis *Matrikon*, identificar o servidor no *Visual Components* e parear as variáveis do *CodeSys* com as variáveis do *Visual Components*.

4.3.1 CONFIGURAÇÃO CODESYS

Para se estabelecer um servidor OPC-UA no *CodeSys* deve-se instalar duas aplicações. A primeira delas é o *Matrikon FLEX OPC UA Server*, onde será definida a interface de *Ethernet*, endereço de IP, número da porta utilizada, número de sessões permitidas, taxa de ciclo do servidor, politica de segurança e tipo de autenticação utilizada pelos clientes.

A Figura 11 mostra a configuração do *Matrikon FLEX OPC UA Server* utilizada neste trabalho.

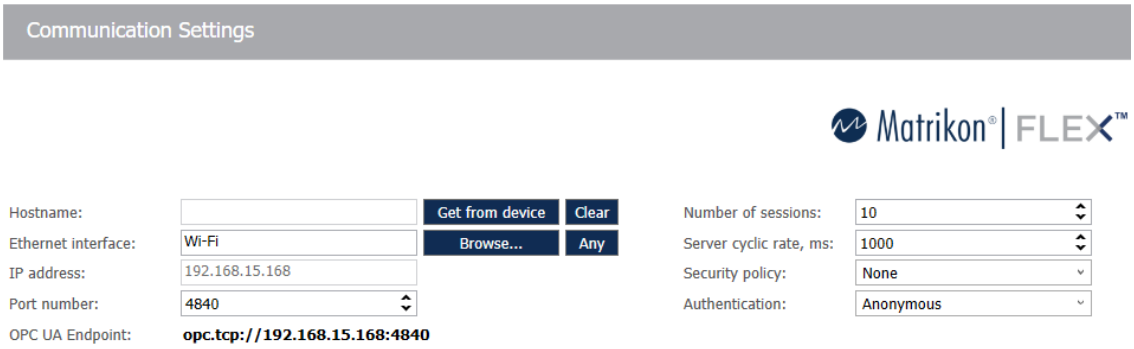

Figura 11: Configuração do *Matrikon FLEX OPC UA Server* utilizada neste trabalho

A segunda aplicação é o *Symbol Configuration*, onde serão selecionadas as variáveis e que tipo de acesso - leitura e escrita - que o *CodeSys* irá permitir. A Figura 12 mostra as variáveis selecionadas e o tipo de acesso permitido, que no caso todas estão com permissão para leitura e escrita. Ou seja, os clientes OPC-UA poderão ler e alterar o valor das variáveis.

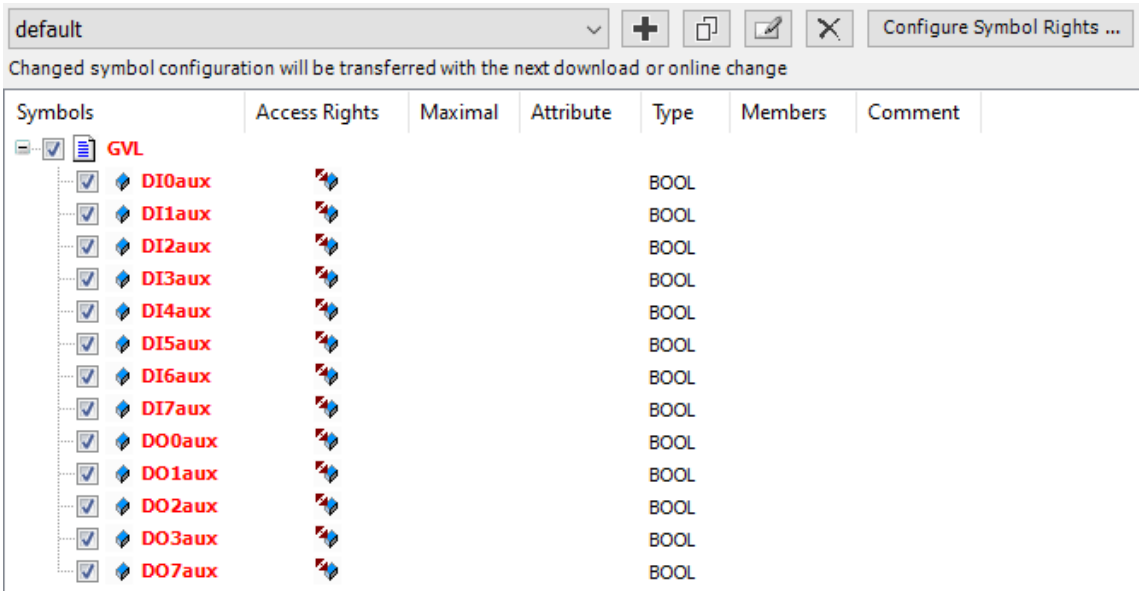

Figura 12: Configuração do *Symbol Configuration* utilizada neste trabalho

Por último, o emulador do controlador deverá ser inicializado, seguido da inicialização do programa da lógica de controle

4.3.2 CONFIGURAÇÃO MATRIKON

O *Matrikon* é o *software* utilizado para leitura das variáveis, gerando uma interface gráfica onde é possível ler e editar o valor das variáveis. Para sua utilização, deve-se instalar na máquina o *software Matrikon OPC UA Explorer*.

Com a aplicação aberta, deve-se solicitar a adição de um novo servidor. Em seguida, selecionar o método de conexão, que no caso deste trabalho será manual, onde a URL do servidor do *CodeSys* deve ser fornecida.

Com a conexão estabelecida, basta criar um painel com as variáveis desejadas. Neste painel será possível ler e escrever o valor das variáveis que possuem permissão concedida pelo *Symbol Configuration* no *CodeSys*. Um exemplo deste painel é apresentado na Figura 13.

Data View - 1

	Display Name	Session Name	Value	Source Timestamp	Server Timestamp
1	DI0aux	OPCUAServer@Ma...	false	2021-12-05 ...	2021-12-05 ...
2	DI1aux	OPCUAServer@Ma...	false	2021-12-05 ...	2021-12-05 ...
3	DI2aux	OPCUAServer@Ma...	false	2021-12-05 ...	2021-12-05 ...
4	DI3aux	OPCUAServer@Ma...	false	2021-12-05 ...	2021-12-05 ...
5	DI4aux	OPCUAServer@Ma...	false	2021-12-05 ...	2021-12-05 ...
6	DI5aux	OPCUAServer@Ma...	false	2021-12-05 ...	2021-12-05 ...
7	DI6aux	OPCUAServer@Ma...	false	2021-12-05 ...	2021-12-05 ...
8	DI7aux	OPCUAServer@Ma...	false	2021-12-05 ...	2021-12-05 ...
9	DO0aux	OPCUAServer@Ma...	false	2021-12-05 ...	2021-12-05 ...
10	DO1aux	OPCUAServer@Ma...	false	2021-12-05 ...	2021-12-05 ...
11	DO2aux	OPCUAServer@Ma...	false	2021-12-05 ...	2021-12-05 ...
12	DO3aux	OPCUAServer@Ma...	false	2021-12-05 ...	2021-12-05 ...
13	DO7aux	OPCUAServer@Ma...	false	2021-12-05 ...	2021-12-05 ...

Figura 13: Exemplo de painel para monitoramento e edição das variáveis

4.3.3 CONFIGURAÇÃO VISUAL COMPONENTS

A ultima etapa do estabelecimento da comunicação OPC-UA do projeto é feita no *Visual Components*. Para isso, deve-se ir ao painel de conectividade e solicitar a adição de um novo servidor OPC-UA. Em seguida, deve-se inserir a URL do servidor do *CodeSys*, o mesmo utilizado no *Matrikon* e, se desejável, alterar as configurações da comunicação. Em caso de sucesso no estabelecimento da comunicação OPC-UA entre servidor e cliente, aparecerá um *True* (verdadeiro) na situação da conexão, conforme indicado na Figura 14.

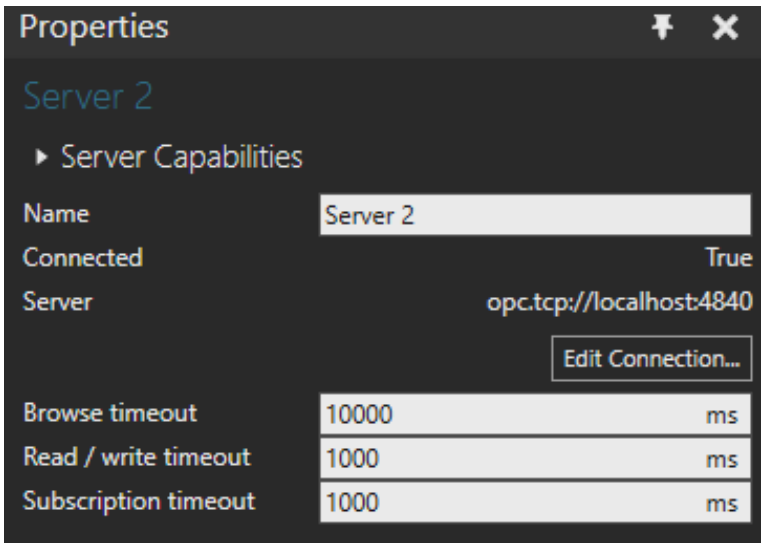

Figura 14: Conexão do cliente do *Visual Components* com o servidor do *CodeSys*

Por último, deve-se parear as variáveis do *Visual Components* com as do *CodeSys*. Para isso, primeiro deve-se indicar as variáveis que irão enviar informação da simulação para o servidor no botão *Simulation to server*. Em seguida, deve-se indicar as variáveis que irão receber informação do servidor para a simulação no botão *Server to simulation*.

No caso deste projeto, as variáveis pareadas em *Simulation to server* estão listadas na Tabela 6. Já as variáveis pareadas em *Server to Simulation* estão listadas na Tabela 7

Tabela 6: Variáveis pareadas em *Simulation to server*

Visual Components	*CodeSys*
DI0	DI0aux
DI1	DI1aux
DI2	DI2aux
DI3	DI3aux
DI4	DI4aux
DI5	DI5aux
DI6	DI6aux

Tabela 7: Variáveis pareadas em *Server to simulation*

Visual Components	*CodeSys*
DO0	DO0aux
DO1	DO1aux
DO2	DO2aux
DO3	DO3aux
DO7	DO7aux

5 CONCLUSÃO

Para permitir um melhor aproveitamento, por parte dos alunos, das atividades de laboratório, é interessante buscar formas de se levar a experiência restrita ao laboratório à casa do aluno. Desta forma, o aluno poderá se preparar melhor para a atividade em laboratório, não ficando restrito a um tempo determinado de aula. Além disso, em tempos de pandemia e distanciamento social, pode-se adaptar as atividades de laboratório, sem comprometer o aprendizado do aluno.

Este trabalho é o primeiro na Escola Politécnica da USP a propor a integração entre o *Codesys* e o *Visual Components* com a finalidade de gerar uma simulação computacional da movimentação das bancadas MPS FESTO. Por este motivo, com os resultados da pesquisa aqui presente e com o resultado positivo do sistema aqui proposto; a disciplina *PMR3305 - Sistemas a Eventos Discretos* poderá se beneficiar do conteúdo deste trabalho através da implementação das bancadas virtuais nas próximas turmas.

Ainda é necessário o aprimoramento da simulação, modelando as demais bancadas presentes no laboratório. Nesse sentido, é interessante que as modelagens sejam desenvolvidas de forma que exista compatibilidade entre os módulos, o que não foi entregue neste trabalho. Esta compatibilidade deve permear formas de se parear componentes distintos, possibilitando que a peça de trabalho percorra todos os módulos, assim como acontece nas bancadas físicas do laboratório.

REFERÊNCIAS

AGOPYAN, V. *Novas medidas de restrição sobre coronavírus para a comunidade*. 2020. Disponível em: ⟨https://jornal.usp.br/institucional/reitor-divulga-novas-medidas-de-restricao-sobre-coronavirus-para-comunidade-universitaria/⟩.

ALVARENGA, D. *Com pandemia, comércio eletrônico tem salto em 2020 e dobra participação no varejo brasileiro*. 2021. Disponível em: ⟨https://g1.globo.com/economia/noticia/2021/02/26/com-pandemia-comercio-eletronico-tem-salto-em-2020-e-dobra-participacao-no-varejo-brasileiro.ghtml⟩.

BALL, K. The dawn of the programmable logic controller (plc). *Spring 2015 Edition of PULSE.*, 2015.

BONFATTI, F.; MONARI, P.; SAMPIERE, U. *IEC 1131-3 programming methodology*. [S.l.]: Cj International, 1999.

COMPONENTS, V. *About Us*. 2021. Disponível em: ⟨https://www.visualcomponents.com/about-us/⟩.

DORIA, J. *Medida de quarentena do Decreto nº 64.881 - 22/03/2020*. 2020. Disponível em: ⟨https://www.al.sp.gov.br/repositorio/legislacao/decreto/2020/decreto-64994-28.05.2020.html⟩.

FONTOURA, C. R. de O. Using the process simulator in chemical engineering study: an application in the beer production process. *Brazilian Journal of Development*, 2019.

HARARI, Y. N. *Sapiens*. [S.l.]: LPM, 2015.

IEC. *International Standard*. [S.l.], 2003.

LEWIS, R. *Programming Industrial Control Systems Using IEC 1131-3*. [S.l.]: The Institution of Engineering and Technology, 2011.

MARTINS, E. R. *Tecnologias, Métodos e Teorias na Engenharia de Computação*. Paraná, Brasil: Atena, 2020.

MIYAGI, P. E. *Controle Programável*. São Paulo, Brasil: Edgar Blucher, 1996.

NEUMANN, J. von. Statistical methods in neutron diffusion. 1947.

OPC-FOUNDATION. *What is OPC?* 2021. Disponível em: ⟨https://opcfoundation.org/about/what-is-opc/⟩.

POLA, D. F. S. R. *Treinamento MPS FESTO*. [S.l.], 2013.

ROSSOW, A. B. Integration of modeling and simulation tools using opc-ua protocol. *XL International Sodebras Congress*, 2018.

TODORA, J. G. *The PLC/PAC Tutorial*. 2009. Disponível em: ⟨http://theplctutor.com/history.html⟩.

USPDIGITAL. *Disciplina: PMR3305 - Sistemas a Eventos Discretos*. 2016. Disponível em: ⟨https://uspdigital.usp.br/jupiterweb/obterDisciplina?nomdis=&sgldis=pmr3305⟩.

VOGEL, M. et al. Petrijet platform technology: An automated platform for culture dish handling and monitoring of the contents. *Journal of Laboratory Automation*, 2015.