Universidade Federal de Campina Grande
Centro de Engenharia Elétrica
Departamento de Engenharia Elétrica

José Wesley Feitosa Oliveira

Trabalho de Conclusão de Curso

Estudo de Metodologia para Geração Automática de Código para CLP

Campina Grande - PB
Dezembro de 2018

José Wesley Feitosa Oliveira

Trabalho de Conclusão de Curso

Estudo de Metodologia para Geração Automática de Código para CLP

Trabalho de Conclusão de Curso submetido à Coordenação de Curso de Graduação de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia Elétrica.

Área de Concentração: Controle e Automação.

Orientador: George Acioli Júnior

Campina Grande - PB

Dezembro de 2018

José Wesley Feitosa Oliveira

Trabalho de Conclusão de Curso

Estudo de Metodologia para Geração Automática de Código para CLP

Trabalho de Conclusão de Curso submetido à Coordenação de Curso de Graduação de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia Elétrica.

Trabalho aprovado em: Campina Grande - PB, / /

George Acioli Júnior, UFCG
Professor Orientador

Rafael Bezerra Correia Lima, UFCG
Professor Avaliador

Campina Grande - PB

Dezembro de 2018

Dedico este trabalho aos meus avós que cuidaram de mim durante toda minha infância com muito carinho, amor e afeto, e infelizmente, não puderam me ver chegar até aqui.

Agradecimentos

Agradeço aos meus pais, que nunca mediram esforços ao fazer da educação dos seus filhos uma prioridade. Por terem sido sempre o ombro amigo na hora do desalento e a voz sábia no momento da dúvida. Por nunca terem desistido e sempre lutado pelos seus.

Agradeço à minha família pelo porto seguro que representam na certeza de que nunca conhecerei a solidão.

Agradeço aos amigos que foram essenciais nessa caminhada. Aos amigos feitos em Campina Grande por terem dividido os bônus e ônus de morar longe de casa e pela certeza de que seremos eternos colegas de profissão.

Agradeço ao professor George que me orientou com muita dedição neste trabalho, sempre dando um norte nas dificuldades que surgiram ao longo da pesquisa.

Agradeço aos funcionários Tchai, Adail e aos coordenadores Damásio e Mário, pela amizade e apoio durante esses anos e por sempre atenderem aos alunos da melhor maneira possível.

“Demore o tempo que for
para decidir o que você
quer da vida e depois
que decidir não recue
ante nenhum pretexto,
porque o mundo tentará
te dissuadir. ”

Friedrich Nietzsche

Resumo

O CLP surge no final dos anos 60 com a proposta de substituir os relés eletromecânicos nos sistemas de controle de motores industriais. Com o seu sucesso, inúmeros fabricantes foram se fixando no mercado, dessa maneira, objetivando padronizar o uso dos CLPs é fundada, em 1992, a PLCOpen®. De maneira geral, a PLCOpen® atua na promoção de comitês técnicos (TCs) e comitês promocionais (PCs), que normatizam o uso dos CLPs, a principal norma ligada à PLCOpen® é a IEC 61131-3, responsável pelas linguagens de programação. Diante do exposto, o *software* MATLAB® desenvolveu uma ferramenta capaz de gerar de maneira automática códigos de programação para CLPs em Diagramas Ladder e Texto Estruturado, estando essas duas linguagens dentro dos padrões internacionais propostos pela IEC 61131-3. Por fim, o presente trabalho detalha os campos de atuação da PLCOpen® e traz o método de geração de código automático pelo MATLAB®, bem como a validação do código gerado nos CLPs do Laboratório de Automação.

Palavras-chave: CLP, PLCOpen®, IEC 61131-3, IDE, Linguagens de programação.

Abstract

The CLP emerged in the late 1960s with the proposal to replace electromechanical relays in industrial motor control systems. With its success, numerous manufacturers have been setting themselves in the market, in order to standardize the use of PLCs, PLCOpen® was founded in 1992. In general, PLCOpen® is dedicated to the promotion of technical committees (TCs) and promotional committees (PCs), which regulate the use of PLCs, the main standard linked to PLCOpen® is IEC 61131-3, responsible for programming languages. In the light of the above, MATLABtextsuperscript® has developed a tool capable of automatically generating programming codes for CLPs in Ladder Diagrams and Structured Diagrams, both of which are within the international standards proposed by IEC 61131-3. Finally, the present work details the fields of action of PLCOpen® and brings the method of generation of automatic code by MATLAB®, as well as the validation of the code generated in the PLCs of the Automation Laboratory.

Keywords: CLP, PLCOpen®, IEC 61131-3, IDE, Programming language

Lista de ilustrações

Lista de abreviaturas e siglas

CLP	Controlador Lógico Programável
GM	*General Motors*
IL	*Instruction List*
LD	*Ladder Diagram*
FBD	*Function Block Diagram*
ST	*Structured Text*
IEC	*International Electrotechnical Commission*
TC	*Technical Committees*
PC	*Promotional Committees*
PSE	*Programming Support Environment*
XML	*Extensible Markup Language*
IDE	*Integrated Development Environment*
POU	*Program Organization Unit*
CI	Circuito Integrado
SFC	*Sequential Function Chart*

Sumário

1 INTRODUÇÃO

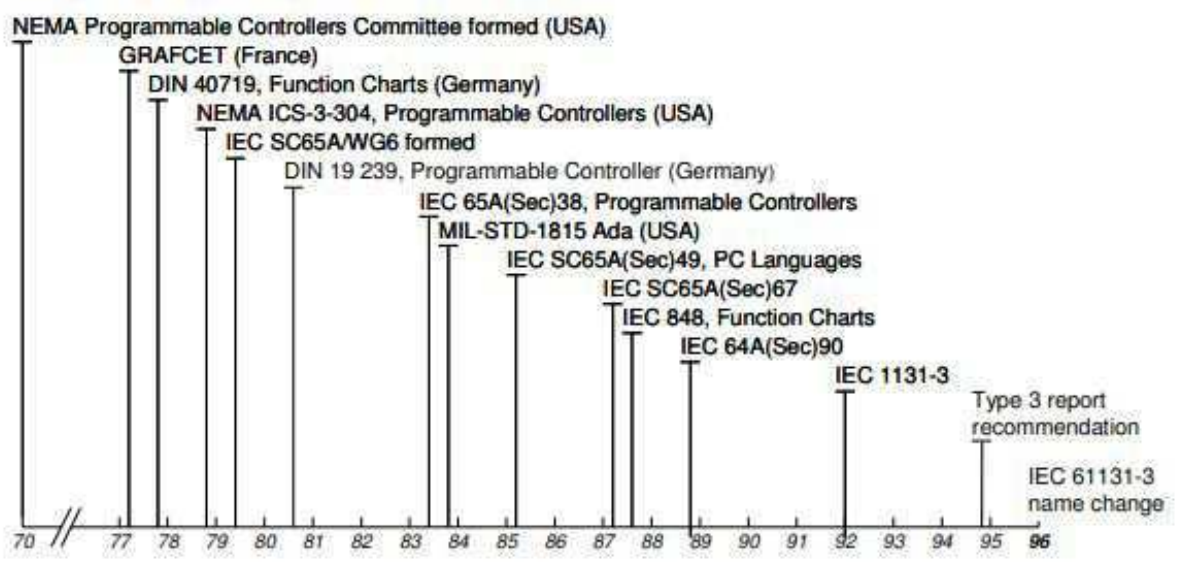
Automação industrial pode ser definida como a aplicação de tecnologias de *software*, *hardware* e equipamentos específicos em processos produtivos. O principal objetivo da automação industrial é implementar projetos que sejam capazes de aumentar a autonomia dos processos de fabricação e reduzir ao máximo o esforço humano na cadeia de valor. Atualmente, trata-se de um conceito intrinsecamente ligado à indústria 4.0, que engloba Sistemas ciber-físicos, Internet das Coisas e Computação em Nuvem para originar "fábricas inteligentes". (1)

A necessidade de um sistema automático surgiu na revolução industrial, onde o trabalho humano passou a ser considerado muito ineficiente face às altas demandas da época, assim despontaram os primeiros sistemas controlados por peças mecânicas, que posteriormente foram substituídas por relés e contatores.

O primeiro CLP foi desenvolvido pela GM no final da década de 60, com intuito de optimizar o seu sistema de produção. Com o sucesso alcançado pela GM, várias outras empresas começaram a se utilizar desse equipamento, e assim surgiram diferentes fabricantes. De circuitos programados apenas em *hardware* utilizando linguagem *Assembly*, atingiu-se o estado da arte com equipamentos que podem ser reprogramados quantas vezes seja necessário, ou seja, o mesmo dispositivo pode atuar em diferentes atividades de controle, bastando, apenas, uma nova programação. Como foi supracitado, diversas empresas iniciaram no ramo da produção de CLPs, e com o intuito de regulamentar a maneira de programar os CLPs, surge em 1992 na Holanda, a PLCOpen®, uma associação mundial que promove a interdependência entre *hardware* e código de programação dos CLPs.

A figura a seguir apresenta a linha do tempo descritiva da evolução dos esforços de normalização e de definição de linguagens de programação de CLPs, a começar pelo ano de 1970, um ano após a instalação do primeiro CLP na fábrica da GM.(2)

Figura 1 – Linha do tempo da evolução das linguagens de programação.



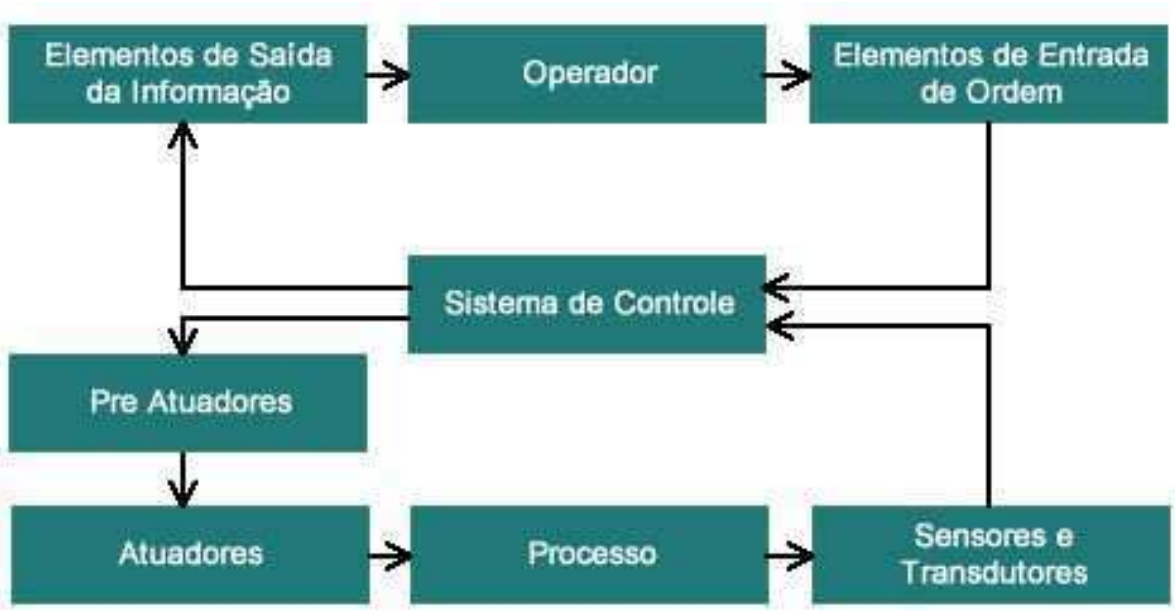
Fonte: `https://edisciplinas.usp.br/pluginfile.php/4203394/mod_resource/content/0/IEC_61131-3.pdf`, 2018.

A automação industrial de um sistema é um procedimento mediante o qual as tarefas de produção que são realizadas por operadores humanos são transferidas a um conjunto de elementos tecnológicos levando-se em consideração possíveis eventualidades que possam ocorrer mantendo sempre a segurança e a qualidade. Cada vez mais os segmentos de produção industrial, geração e distribuição de energia, transportes e muitos outros requerem um número crescente de novos sistemas e máquinas automatizadas. Isto se deve ao aumento da produção, aos custos mais baixos de componentes de automação e máquinas, a qualidade e estabilidade de novos produtos e à necessidade de substituir trabalhos perigosos e monótonos dos operadores.(3)

Na era atual, a automação se embasa na projeção e implantação de sistemas ciber-físicos, que controlam processos materiais e gerenciam as tomadas de decisões de forma totalmente descentralizada. Com a internet das coisas, esses sistemas ganharam a habilidade de "dialogar" com diversas máquinas simultaneamente e enviar informações em tempo real para gestores e supervisores. Contudo, sistemas mecânicos e eletrônicos mais simples ainda não foram abandonados, pois existem processos em que a mão de obra humana se faz fundamental à qualidade do produto. Além disso, sistemas ciber-físicos apresentam custo mais elevado, o que, às vezes, impossibilita sua implantação.(1)

A figura abaixo mostra um esquemático de como opera um sistema industrial automatizado.

Figura 2 – Funcionamento de um sistema industrial automazido



Fonte: `https://www.citisystems.com.br/o-que-e-automacao-industrial/`, 2018.

Os sensores e transdutores são os elementos que convertem as grandezas físicas em sinais elétricos, os atuadores e pré-atuadores fazem o papel final do sistema de controle, sendo esses elétricos (motores), pneumáticos (esteiras) ou hidráulicos (quando a carga demanda uma força muito alta para os motores elétricos). Um exemplo de atuação dos CLPs é o controle de operação dos atuadores via válvula solenoide (no caso dos atuadores pneumáticos) ou via inversor de frequência (no caso dos motores elétricos).

O CLP é considerado o cérebro da automação industrial por controlar os equipamentos e processos. Sua vantagem é que ele possui as características de um computador, mas a diferença de ter sido projetado especialmente para trabalhar em ambientes industriais dos mais limpos aos mais agressivos. Outra vantagem que ele possui é que a programação é mais intuitiva com a utilização da lógica Ladder (lógica de programação que reproduz os diagramas elétricos em blocos lógicos e blocos de função). (3)

1.1 Objetivos

Dito isto, o objetivo desse trabalho foi estudar uma metodologia de geração automática de código para os CLPs, o que traz ainda mais comodidade e versatilidade aos sistemas controlados, contudo os códigos gerados e a maneira como são implementados nas IDEs de cada fabricante de controladores lógicos precisam estar de acordo com a IEC 61131-3. Sendo assim, também foi feito um estudo dessa norma, bem como das outras normas que a complementam, o conjunto IEC 61131, que aborda nove normas para os

fabricantes e profissionais da Automação Industrial. A IEC 61131 é parte principal da PLCOpen®, o detalhamento dessa instituição foi outro objetivo do trabalho.

1.2 Organização do Trabalho

Além do presente capítulo, onde consta uma introdução de toda a pesquisa, a continuidade do trabalho ficou dividida da seguinte forma:

- Capítulo 2: Fundamentação Teórica. Neste capítulo está a base teórica utilizada, que se trata do estudo da PLCOpen® e da IEC 61131-3.
- Capítulo 3: Desenvolvimento das atividades. Neste capítulo foi detalhado o uso da ferramenta para a geração automática de código e a implementação do código gerado na IDE da Rockwell Automation® (Studio 5000).
- Capítulo 4: Conclusão.

2 FUNDAMENTAÇÃO TEÓRICA

Para o desenvolvimento deste trabalho, faz-se necessário o conhecimento sobre Automação Industrial, lógica e linguagens de programação usadas nos CLPs, bem como das normas que fazem a sua padronização. Ter uma boa familiaridade com a ferramenta *Simulink*® fornecida pelo *software* MATLAB® também é importante.

2.1 PLCOpen®

A PLCopen, como uma organização ativa no Controle Industrial, está criando uma alta eficiência para o desenvolvimento do software aplicativo: em um único projeto assim como em um alto volume de produtos. É embasada em ferramentas padrões disponíveis, para as quais extensões são e serão definidas. Com resultados como a biblioteca para Motion Control, Segurança, Especificação XML, Níveis de Reutilização e Conformidade, a PLCOpen® faz sólidas contribuições para a comunidade, estendendo a independência entre o software e o hardware, assim como a reutilização de código e integração com ferramentas externas de *software.*(4)

A principal atividade da PLCopen® está na norma IEC 61131-3, única norma que regulamenta a programação de CLPs. A importância da padronização se dá pelo fato de que o código seja reaproveitado por diversas pessoas e em qualquer equipamento. Dentro da norma existem quatro linguagens de programação: *Instruction List* (Lista de Instruções - IL), *Ladder Diagram* (Diagrama Ladder - LD), *Function Block Diagram* (Diagrama de Blocos Funcionais - FBD) e *Structured Text* (Texto Estruturado - ST).

2.1.1 Comitês Técnicos e Promocionais

No total a PLCopen® possui seis TCs (*Technical Committees*) e quatro PCs (*Promotional Committees*), são eles:

- TC1 - Normas
- TC2 - Funções
- TC3 - Certificação
- TC4 - Comunicação
- TC5 - Segurança
- TC6 - Esquemas XML

- PC1, PC3 e PC4 - Divulgação da importância da padronização da programação de controle industrial
- PC2 - Treinamento

O Comitê Técnico 1 (TC1) abrange nove normas, a principal delas sendo a IEC 61131-3 e as demais atuam de maneira a completar esta, todas elas fazem parte da Norma Internacional IEC 61131 desenvolvida pelo *International Electrotechnical Commission* (Comitê Eletrotécnico Internacional - IEC).

O Comitê Técnico 2 (TC2) define bibliotecas comuns de Funções e Blocos Funcionais para áreas de aplicação específicas. Exemplos são as bibliotecas de Blocos Funcionais para aplicações de Segurança (*Safety*) e para *Motion Control*. Esta padronização integra os aspectos de segurança e controle de movimento no controle industrial (4)

O Comitê Técnico 3 (TC3) define um sistema de certificação ao qual uma PSE (*Programming Support Environment*) se encontra face à IEC 61131-3, denominada Nível de Conformidade, esse TC é responsável por apontar um subconjunto da norma ao qual a PSE integra. Dentro dessa certificação existem três níveis: Nível Certificação, Nível Reutilizção e Nível Básiso.

Nível Certificação é o nível mais alto. Com a ampla faixa de áreas de aplicação da norma IEC 61131-3, nem todas implementações utilizam exatamente os mesmos tipos de dados. Para comportar isto, a certificação de acordo com o Nível Conformidade, CL, implica que o fornecedor da PSE selecione os tipos de dados pelo seu produto em concordância com a declaração de conformidade;

Nível Reutilização é dedicado a fazer que as unidades de programação Função e Blocos Funcionais sejam reutilizáveis em diferentes PSEs na mesma linguagem de programação.

Nível Básico este é um nível de entrada, que mostra o comprometimento com a norma. Isso facilita o trabalho dos usuários que trabalham com mais de um fabricante, permitindo uma interpretação mais uniforme da norma.

O Comitê Técnico 4 (TC4) abrange a área da relação entre comunicação e as linguagens de programação, permitindo que cada serviço ou dispositivo se conecte independentemente e incorporação da tecnologia *OPC UA*

O Comitê Técnico 5 (TC5) fornece condições de segurança para aplicação da IEC 61131-3, onde se visa desenvolver um *software* seguro e que proporcione orientações no uso dos aspectos de confiabilidade.

O Comitê Técnico 6 (TC6) trabalha na especificação do esquema XML para todas as linguagens, o que proporciona uma facilidade no intercâmbio e na integração com as ferramentas de *software*.

Os Comitês Promocionais 1, 3 e 4 cumprem a tarefa de divulgar a PLCOpen® pelo mundo, mostrando todos os benefícios da padronização. Isso inclui o *website*, publicação de periódico próprio (PLCopening), participação em congressos e conferências e também organização dos próprios eventos. A divisão dos PCs é feita por região, onde o PC1 fica responsável pela Europa, o PC3 pelos Estados Unidos e o PC4 pelo Japão

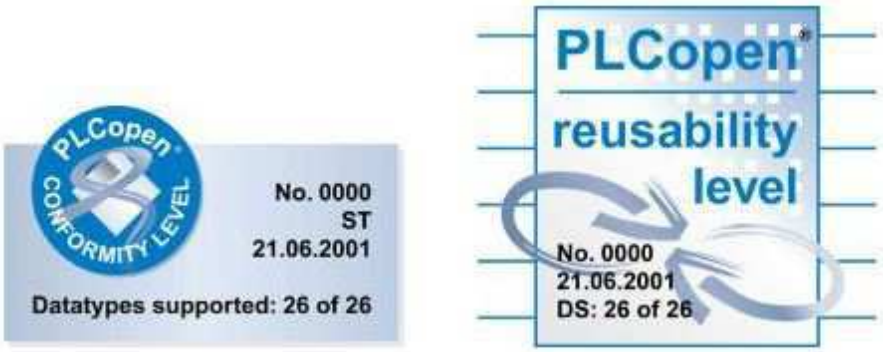
O Comitê Promocional 2 (PC2) fornece os requisitos de treinamento para o uso da IEC 61131-3. É importante frisar que a PLCOpen® não fornece os treinamentos, apenas especifíca os requisitos básicos dos mesmo.

Abaixo encontram-se alguns selos fornecidos pela PLCOpen® fornecidos aos seus membros.

Figura 3 – Selo de certificação para empresas autorizadas a utilizarem os blocos de *Motion Control*.

Fonte: plcopen.org, 2018.

Figura 4 – Selos de Nível de Conformidade e Nível de Reusabilidade



Fonte: plcopen.org, 2018.

Figura 5 – Selo de certificação para os critérios de comunicação com a parceria entre PLCOpen® e OPC UA

Fonte: plcopen.org, 2018.

Figura 6 – Selo de certificação de segurança

Fonte: plcopen.org, 2018.

Figura 7 – Selo de certificação para membros habilitados à geração de diagrama XML

Fonte: plcopen.org, 2018.

Figura 8 – Selo de certificação para empresas habilitadas a prestarem treinamentos da IEC 61131-3

Fonte: plcopen.org, 2018.

2.2 IEC 61131-3

A IEC 61131-3 é uma norma internacional, sediada pela International Electrotechnical Commission, IEC. A convenção desta norma aconteceu para ser americana. Outros países membros são: Alemanha, França, Inglaterra, Japão e Holanda.

Inicialmente era vista apenas como uma norma para CLP's, atraindo a atenção da maior região de usuários e fornecedores: a Europa. A popularidade na América do Norte aumentou quando surgiram os primeiros soft-controladores. (SoftPLC), os quais são computadores pessoais que executam um programa que se comporta como um CLP. O documento do grupo OMAC é o principal exemplo disto.

Muitos fabricantes japoneses de CLP's começaram adotando pacotes europeus para programação de CLP's para o mercado europeu. Contudo, hoje o Japão está entre os mais aquecidos mercados para os fornecedores de pacotes de programação. (4)

Segundo a PLCOpen®, pode-se dividir a IEC 61131-3 em duas partes: Elementos Comuns e Linguagens de Programação.

2.2.1 Elementos Comuns

Dentro dos Elementos Comuns, tem-se: Tipos de Dados, Variáveis, Configuração, Recursos e Tarefas, Unidades de Organização de Programas e o Sequenciamento Gráfico de Funções (SFC). A seguir será exposto cada um desses elementos mais detalhadamente.

Tipos de Dados a definição dos tipos de dados previne erros na fase inicial, como, por exemplo, a divisão de uma data por um inteiro, alguns tipos de dados definidos são: *Boolean, Integer, Real, Byte, Word, Date, Time_of_Day* e *String*;

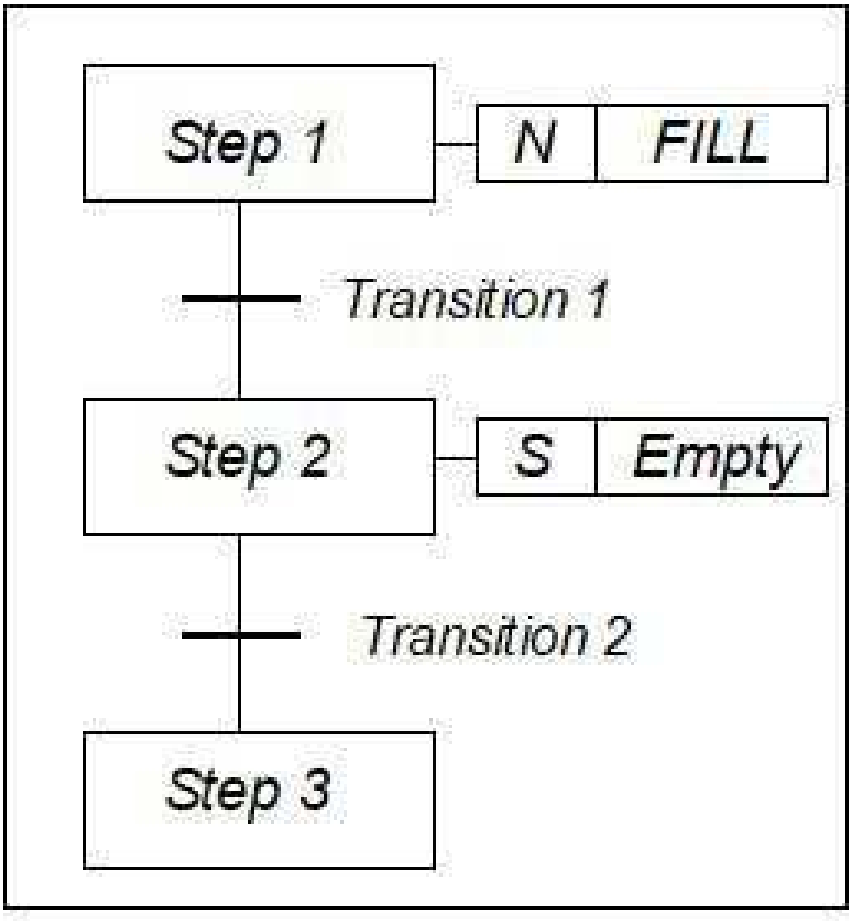
Variáveis são associadas aos endereços de *hardware* (entradas e saídas, por exemplo) nas configurações, recursos e programas. Normalmente, seu escopo é local, podendo ser reutilizadas em outras partes sem conflito, se forem de escopo global devem receber a declaração 'VAR_GLOBAL';

Configuração, Recursos e Tarefas A configuração é o nível mais alto do *software*, onde o problema de controle é formulado. Os recursos estão inseridos na configuração, e podem ser definidos como um elemento com capacidade de processamento dos programas IEC. Seguindo a lógica, as tarefas estão inseridas nos programas, e controlam a execução de um conjunto de programas ou blocos funcionais, são executadas periodicamente ou quando ocorrer algum evento específico, como mudança de uma variável.

Unidades de Organização de Programas As POUs (Unidades de Organização de Programas) constitutem os Programas, Blocos Funcionais e Funções. Existem algu-

mas funções padrões como, por exemplo, ADD(*addition*), ABS (*absolute*), SQRT, SINus e COSinus, e as funções criadas pelos usuários, que podem ser aproveitadas o quanto forem necessárias. Os Blocos Funcionais são semelhantes aos CIs, onde representa-se uma função de controle e escondem seu conteúdo, são intuitivos e preservam os estados passados, podendo ser programados na linguagem "C"e reutilizados em diversos programas ou projetos. Os programas são constituídos de funções e blocos funcionais. O SFC é utilizado para descrever de modo sequêncial um programa de controle, essa ferramenta auxilia a organização interna do programa e ajuda a decompor o problema em partes gerenciáveis. O SFC consiste de Passos, interligados com blocos de Ações e Transições. Cada passo representa um estado particular do sistema sendo controlado. Uma transição é associada com uma condição, a qual, quando verdadeira, causa a desativação do passo anterior à mesma e a ativação do passo seguinte. Passos são ligados com blocos de ações, desempenhando uma determinada ação de controle. Cada elemento pode ser programado em qualquer linguagem IEC, incluindo o próprio SFC. (4)

Figura 9 – Exemplo de um SFC

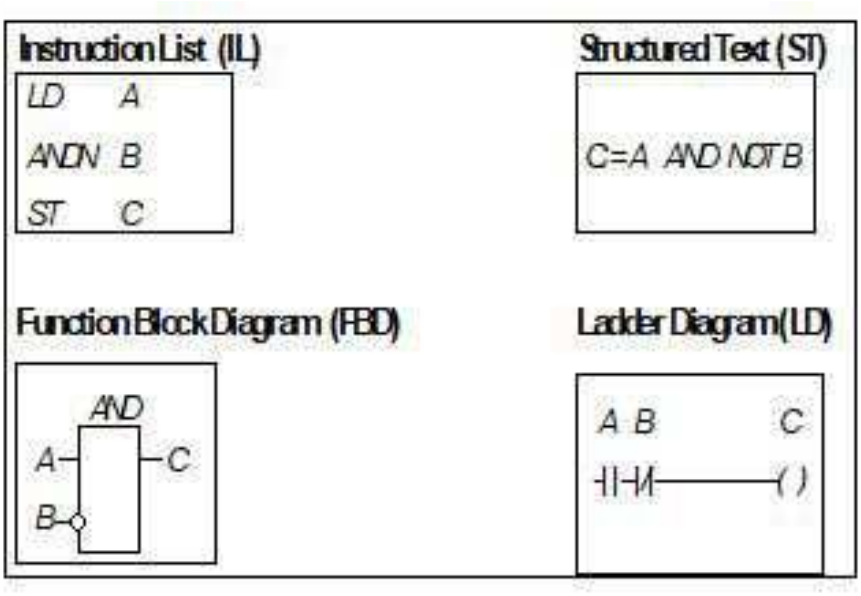


Fonte: plcopen.org, 2018.

2.2.2 Linguagens de Programação

Aqui são definidas as linguagens de programação normatizadas pela IEC 61131-3, como já foi citado anteriormente, dispomos de quatro linguagens, duas textuais (Lista de Instruções e Texto Estruturado) e duas gráficas (Diagrama Ladder e Blocos Funcionais). Na figura a seguir é mostrada uma lógica simples de controle implementada para essas quatro linguagens.

Figura 10 – Comparação entre as quatro linguagens



Fonte: plcopen.org, 2018.

A variedade de linguagens interligadas é bastante eficaz, pois a depender de alguns fatores como, por exemplo: formação do programador, problema a ser resolvido, nível da descrição do problema, estrutura do sistema de controle ou interface com outras pessoas/departamentos, é proporcionado uma variedade de maneiras de como resolver tais problemas.

Diagrama Ladder é o equivalente a lógica de relés, sendo desenvolvida para substituir esses mecanismos.

Lista de Instruções foi criada com base no *Assembly*, sendo muito semelhante a essa linguagem.

Blocos de Funções expressa o processo de controle como um conjunto de blocos gráficos interligados, se assemelha com os diagramas de blocos de estudos de sinais e circuitos eletrônicos.

Texto Estruturado é uma linguagem de alto nível, baseada em Ada, Pascal e "C", contém todas as características de uma linguagem de programação, como as funções condicionais (IF-THEN-ELSE e CASE OF) e iterativas (FOR, WHILE e REPEAT).

A norma IEC 61131-3 trabalha em conjunto com outras normas regulamentadas pela IEC e está inserida no Comitê Técnico 1 (TC1), que já foi mencionado anteriormente. Além da IEC 61131-3, existem outras oito normas contidas nesse Comitê Técnico, são elas:

IEC 61131-1 Informações gerais estabelece as definições e identifica as principais características relevantes para a seleção e aplicação dos CLPs e seus periféricos associados;

IEC 61131-2 Requisitos e testes de equipamento especifica requisitos de equipamentos e testes relacionados para controladores programáveis (CLP) e seus periféricos associados;

IEC 61131-3 Linguagens de Programação define um conjunto mínimo com os elementos básicos de programação, regras sintáticas e semânticas para as linguagens de programação mais utilizadas;

IEC 61131-4 Diretrizes do Usuário fornece um relatório técnico contendo informações gerais e diretrizes de aplicação do padrão para o usuário final de controladores programáveis;

IEC 61131-5 Especificação de serviço de mensagens define a comunicação de dados entre controladores programáveis e outros sistemas eletrônicos;

IEC 61131-6 Segurança funcional conforme a norma IEC 61131-1, fornece os requisitos e periféricos associados que devem ser usados como subsistema lógico de um sistema relacionado com a segurança;

IEC 61131-7 Programação de controle *fuzzy* efine elementos básicos de programação para o controle de lógica *fuzzy*, conforme usado em controladores programáveis;

IEC 61131-8 Diretrizes para as linguagens de programação fornece um guia de desenvolvedores de software para as linguagens de programação definidas na 61131-3;

IEC 61131-9 Interface de comunicação digital *single-drop* fornece os padrões de comunicação para pequenos sensores e atuadores, também conhecido por "IO-Link".

Por fim, vamos fazer uma comparação com um CLP convencional: este contém um recurso, executando uma tarefa, controlando um programa, processando de forma cíclica. A IEC 61131-3 acrescenta muito mais capacidade, tornando-o aberto para o futuro. Um futuro que inclui multi-processamento e programas disparados por eventos. E este futuro não está longe: basta olhar para os sistemas distribuídos ou sistemas de controle de tempo-real. A IEC 61131-3 é apropriada para uma ampla faixa de aplicações, sem a necessidade de se aprender linguagens de programação adicionais.

A norma IEC 61131-3 causará um grande impacto em toda indústria de controle industrial. Certamente a norma não ficará restrita para o mercado de CLPs convencionais. Atualmente, a norma já é adotada no mercado de *Motion Control*, sistemas distribuídos e sistemas de controle baseados em PC/Softlogic, incluindo pacotes SCADA. E as áreas de aplicação continuam crescendo.(4)

3 DESENVOLVIMENTO DAS ATIVIDADES

3.1 Introdução ao Simulink PLC Coder™

A ferramenta computacional utilizada para a geração automática de código foi a Simulink PLC Coder™ disponível no MATLAB®, capaz de gerar códigos em Texto Estruturado e Linguagem Ladder, além do esquema em XML, a partir do Simulink®, Stateflow® e Funções do MATLAB®, neste trabalho foi usado o Simulink®.Ele também fornece relatórios de geração de código com métricas de código estático e rastreabilidade bidirecional entre modelo e código. Os arquivos gerados podem ser importados para as IDEs dos CLPs de maneira rápida e eficiente. (5) e (6)

As principais características da ferramenta, de acordo com a MathWorks®, são:

- Geração automática de diagramas de texto estruturado e Ladder da IEC 61131-3;
- Suporte Simulink® , incluindo subsistemas reutilizáveis, blocos controladores PID e tabelas de consulta;
- Suporte Stateflow® , incluindo máquinas de estado, funções gráficas e tabelas de verdade;
- Suporte a MATLAB® , incluindo instruções if-else, construções de loop e operações matemáticas;
- Suporte para vários tipos de dados, incluindo booleano, inteiro, enumerado e ponto flutuante, bem como vetores, matrizes, barramentos e parâmetros ajustáveis;
- Criação de bancada de teste

A seguir encontram-se as IDEs que são suportadas pelo PLC Coder™:

- 3S CoDeSys 2.3
- 3S CoDeSys 3.3
- B&R Automation Studio 3.0
- Beckhoff TwinCAT 2.11
- KW-Software MULTIPROG 5.0
- Phoenix Contact PC WORX 6.0

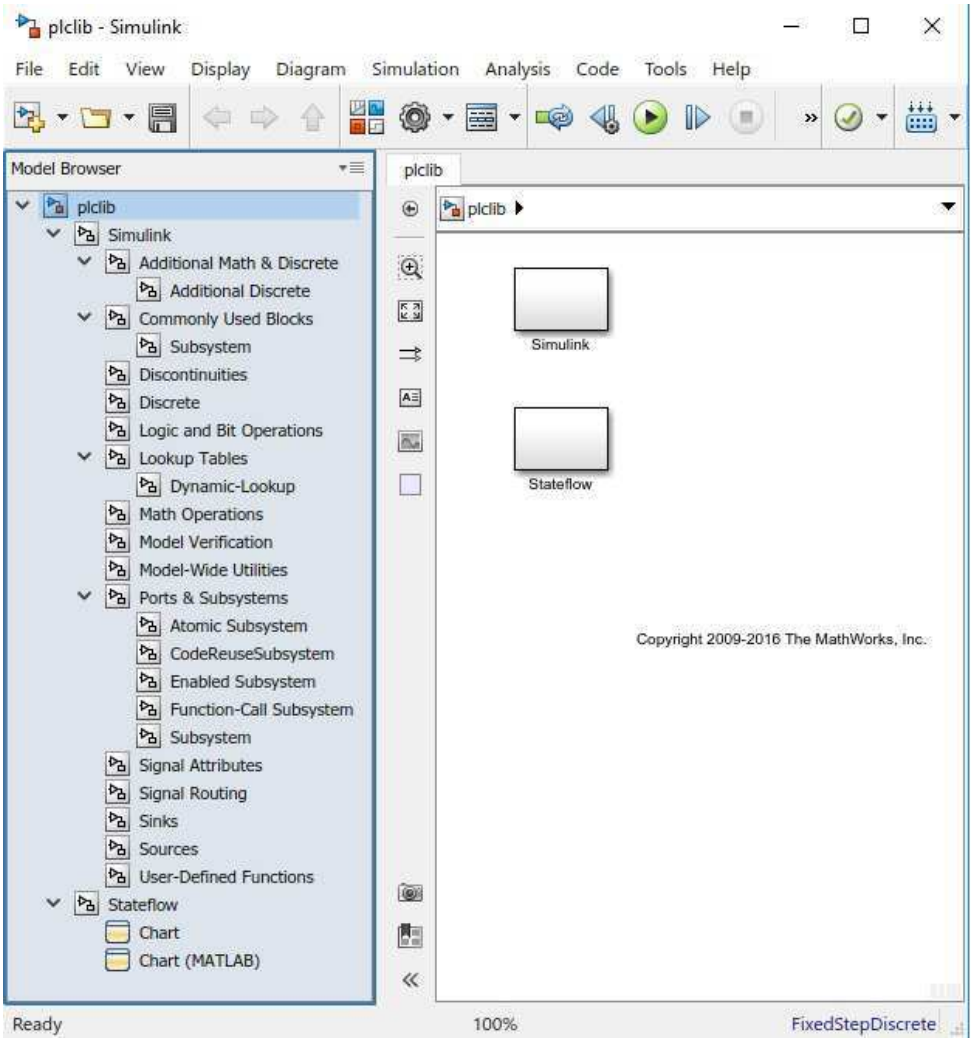
- Rockwell RSLogix 5000: AOI
- Rockwell RSLogix 5000: Routine
- Siemens SIMATIC Step 7 5.4
- Generic
- PLCOpen XML
- IndraWorks Engineering
- OMRON Sysmac Studio

As IDEs da CoDeSys, da Rockwell Automation e o PLCOpen XML possuem suporte tanto para o Texto Estrururado, quanto para o diagrama em Ladder, as demais possuem suporte apenas para o Texto Estruturado.

3.2 Executando a geração automática de código

Inicialmente, é importante saber quais blocos do Simulink® podem ser utilizados para geração dos códigos, para isso basta inserir o comando 'plclib' na janela de comandos (*Command Window*). O resultado será a janela mostrada a seguir:

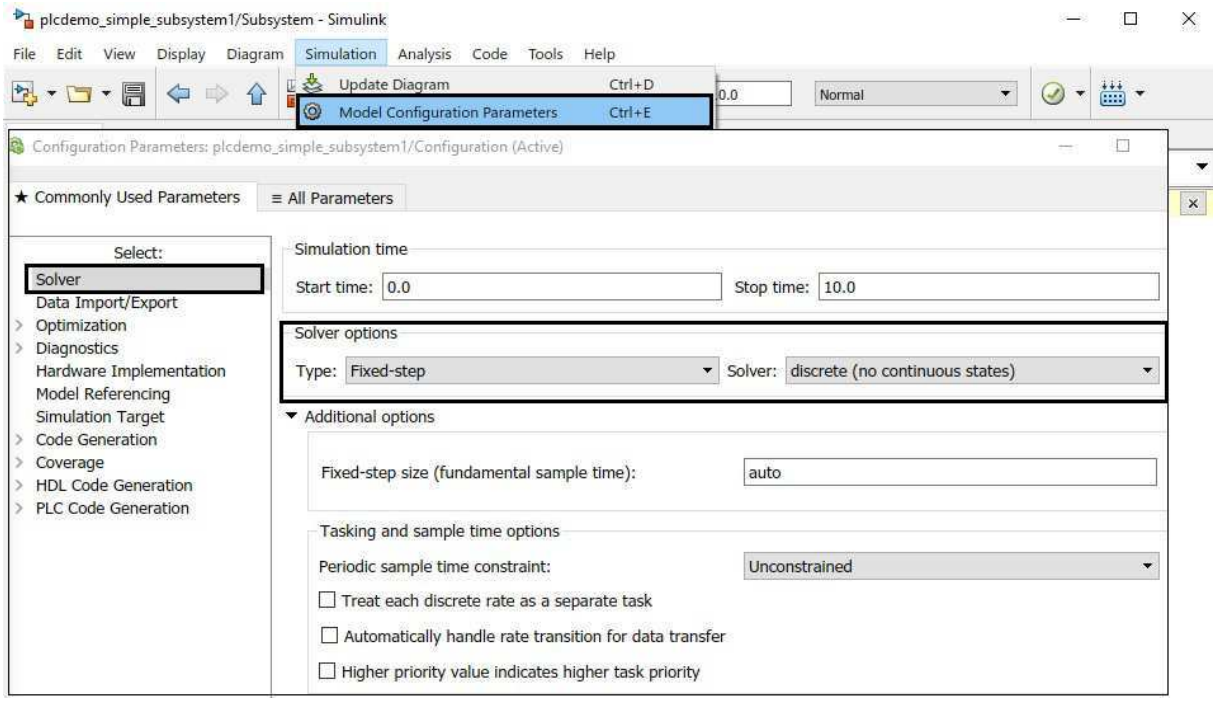
Figura 11 – Blocos suportados pelo Stateflow® e Simulink®



Fonte: Autoria própria, 2018

Para gerar o código, é necessário a alteração de alguns parâmetros de simulação, tipo de solução discreta com passo fixo.

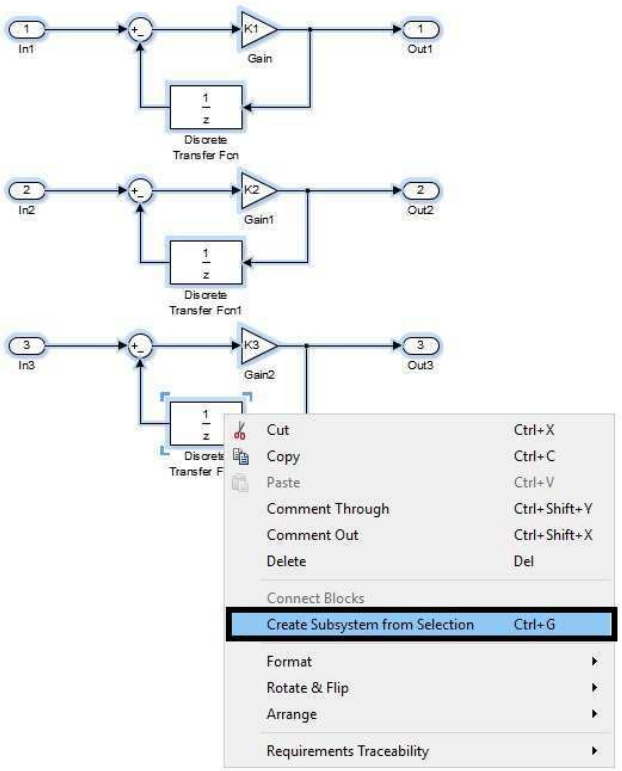
Figura 12 – Mudando os parâmtros de simulação



Fonte: Autoria própria, 2018

A partir de um modelo criado no Simulink®, selecione todos os componentes que se deseja gerar o código e crie um subsistema, em seguida é preciso salvar e executar.

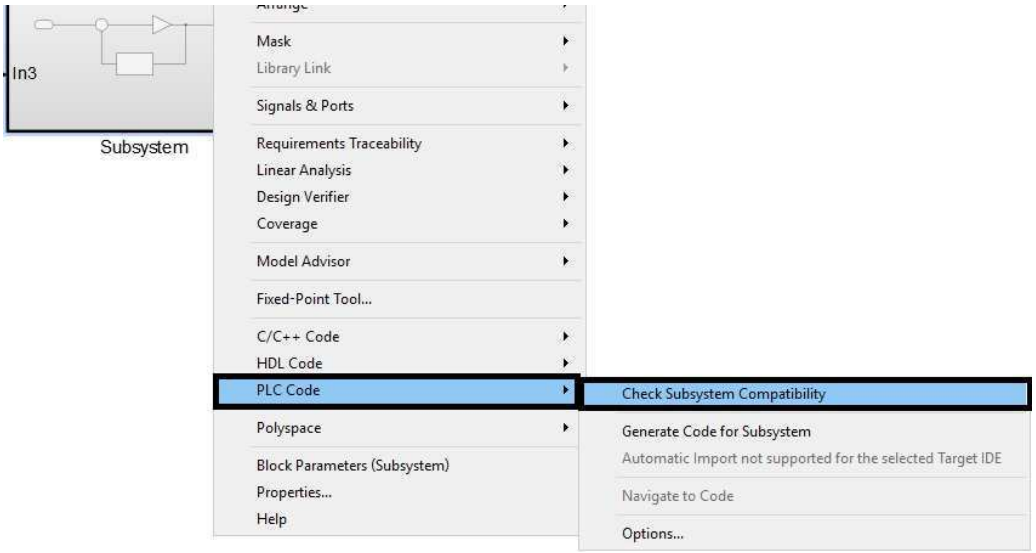
Figura 13 – Criando subsistema com os blocos que iram ser utilizados no código



Fonte: Autoria própria, 2018

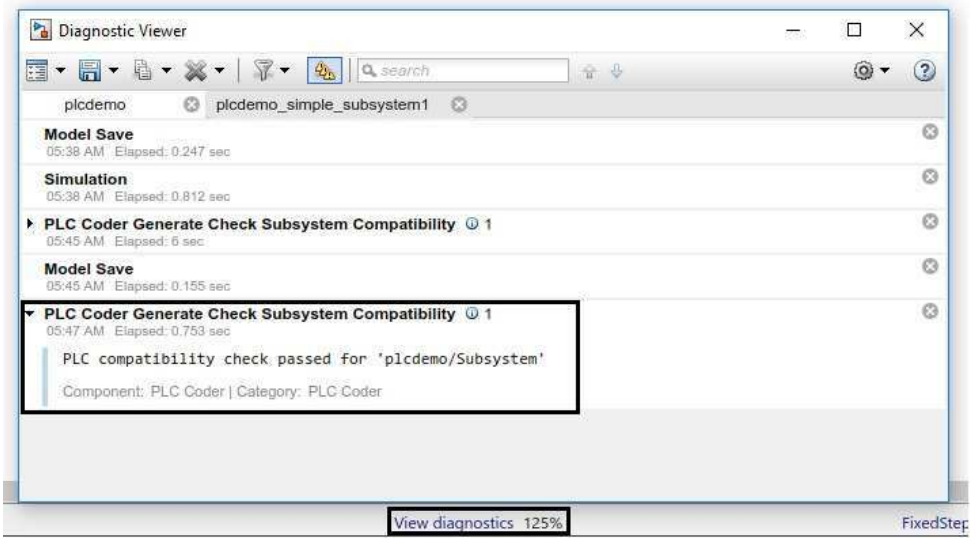
Com o subsistema criado, verifique se este é compatível com a geração automática de código para CLPs. Caso haja algum erro, se abrirá uma janela informando o que foi feito indevidamente. A janela de diagnósticos também pode ser acessada na parte inferior da tela.

Figura 14 – Verificando se o subsistema é compatível para a geração automática de código



Fonte: Autoria própria, 2018

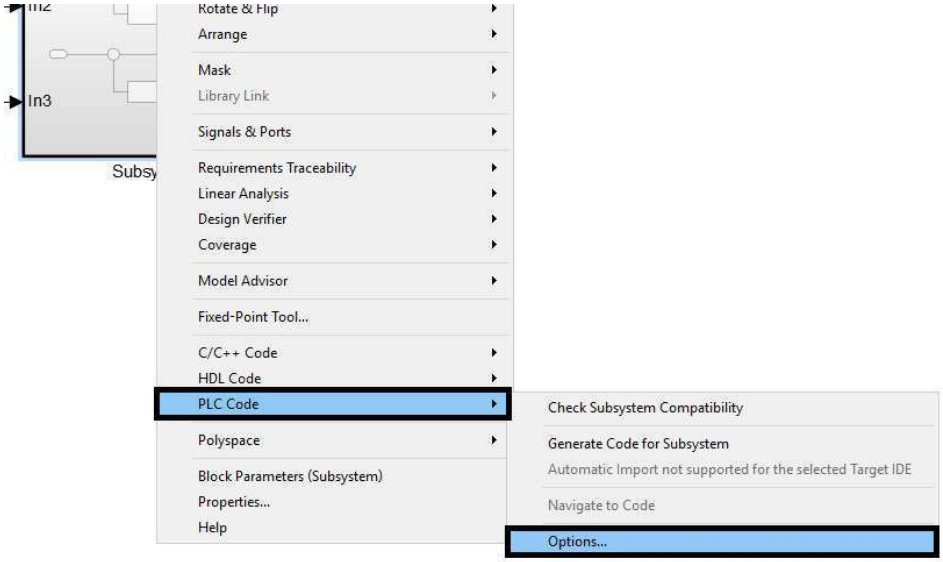
Figura 15 – Diagnóstico da verificação



Fonte: Autoria própria, 2018

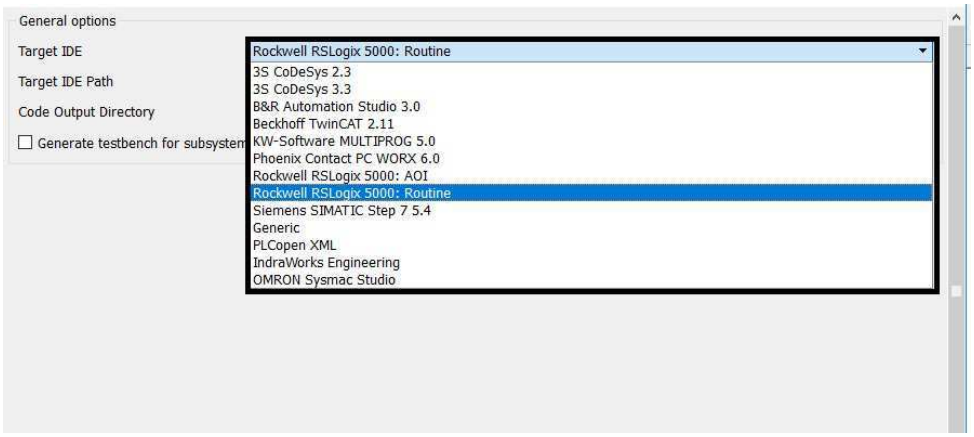
Feito isto, o próximo passo é gerar o código para o CLP. As figuras a seguir ilustram o que deve ser feito para que o código seja gerado.

Figura 16 – Abrir o campo 'options' em 'PLC CODE'



Fonte: Autoria própria, 2018

Figura 17 – Escolher a IDE desejada



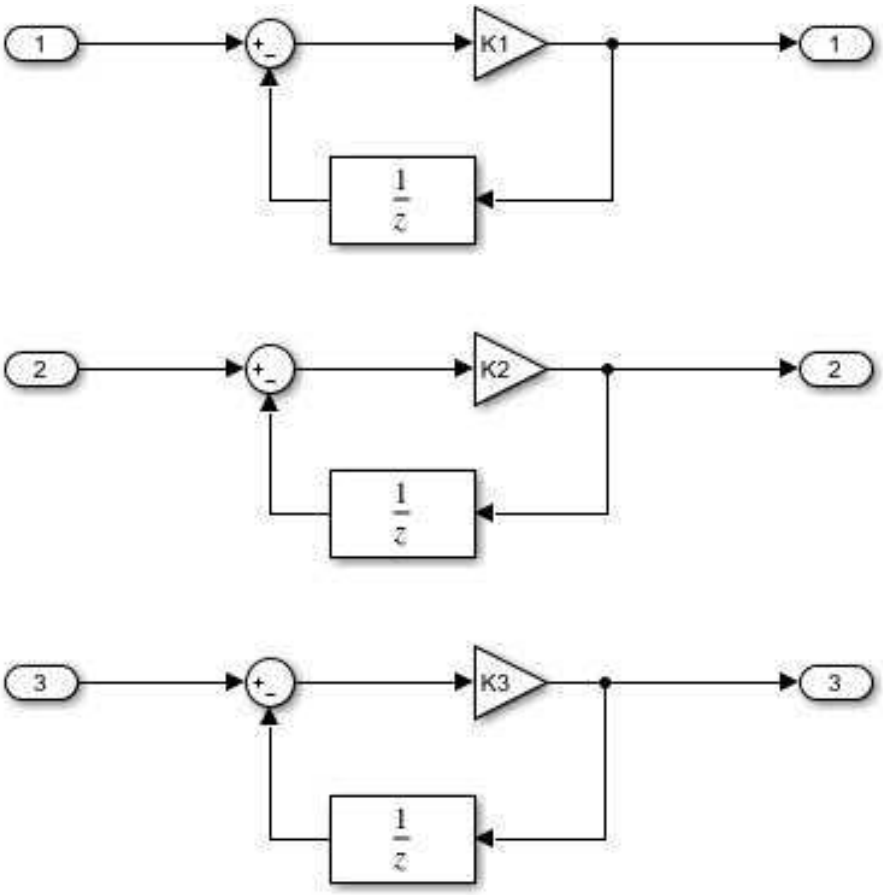
Fonte: Autoria própria, 2018

Caso não seja alterada, o arquivo com o código estará no diretório criado pelo MATLAB® 'plcscr' dentro do diretório onde o projeto foi salvo.

3.3 Utilização do código gerado

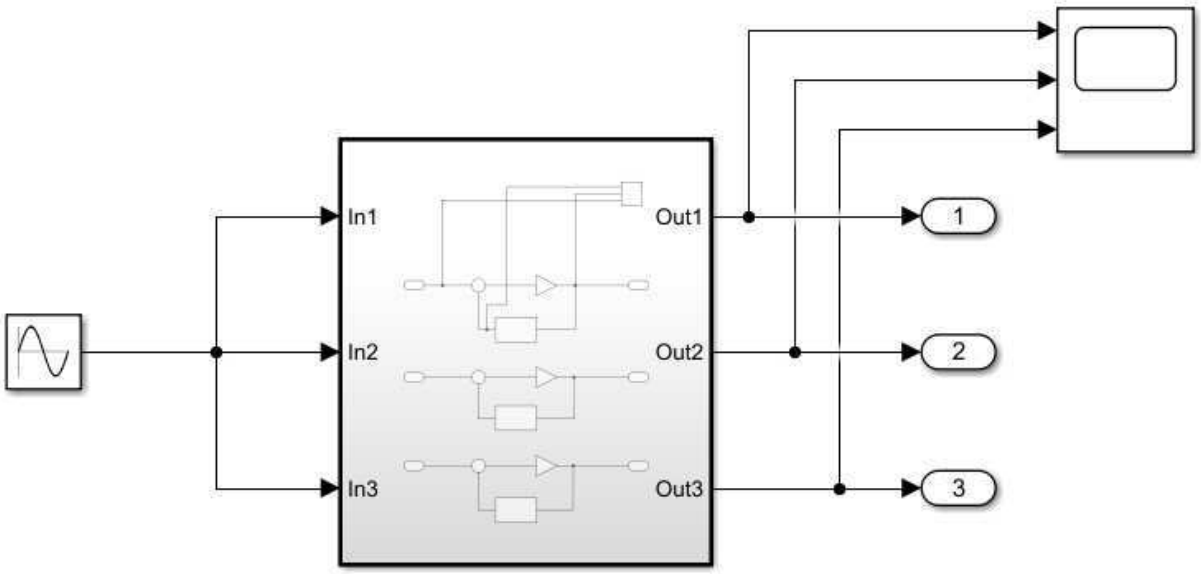
Nesta secção utilizou-se o código gerado anteriormente para a IDE Studio 5000®, da fabricante Rockwell Automation® a fim de ser validado nos CLPs do Laboratório de Automação. As figuras abaixo mostram o subsistema criado no MATLAB®.

Figura 18 – Subsistema criado no MATLAB®



Fonte: Autoria própria, 2018

Figura 19 – Subsistema conectado com entradas e saídas



Fonte: Autoria própria, 2018

Esquema XML gerado para o subsistema.

Listing 3.1 – Test

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <project><!--(*
3  *
4  * File: ajust_param.xml
5  *
6  * IEC 61131-3 Structured Text (ST) code generated for subsystem ...
       "ajust_param/Subsystem"
7  *
8  * Model name                        : ajust_param
9  * Model version                     : 1.1
10 * Model creator                     : Automacao
11 * Model last modified by            : Automacao
12 * Model last modified on            : Thu Dec 06 15:46:26 2018
13 * Model sample time                 : 0.2s
14 * Subsystem name                    : ajust_param/Subsystem
15 * Subsystem sample time             : 0.2s
16 * Simulink PLC Coder version        : 2.5 (R2018a) 06-Feb-2018
17 * ST code generated on              : Thu Dec 06 15:50:15 2018
18 *
19 * Target IDE selection              : PLCopen XML
20 * Test Bench included               : No
21 *
22 *)
23 -->
24    <fileHeader companyName="The Mathworks, Inc." ...
          companyURL="www.mathworks.com" contentDescription="" ...
          creationDateTime="2018-12-06T15:50:15" productName="Simulink ...
          PLC Coder" productRelease="(R2018a)" productVersion="2.5"/>
25    <contentHeader name="ajust_param">
26       <coordinateInfo>
27          <fbd>
28             <scaling x="0" y="0"/>
29          </fbd>
30          <ld>
31             <scaling x="0" y="0"/>
32          </ld>
33          <sfc>
34             <scaling x="0" y="0"/>
35          </sfc>
36       </coordinateInfo>
37    </contentHeader>
38    <types>
39       <dataTypes/>
```

```
40        <pous>
41            <pou name="Subsystem" pouType="functionBlock">
42                <interface>
43                    <inputVars>
44                        <variable name="ssMethodType">
45                            <type>
46                                <SINT/>
47                            </type>
48                        </variable>
49                        <variable name="In1">
50                            <type>
51                                <LREAL/>
52                            </type>
53                        </variable>
54                        <variable name="In2">
55                            <type>
56                                <LREAL/>
57                            </type>
58                        </variable>
59                        <variable name="In3">
60                            <type>
61                                <LREAL/>
62                            </type>
63                        </variable>
64                    </inputVars>
65                    <outputVars>
66                        <variable name="Out1">
67                            <type>
68                                <LREAL/>
69                            </type>
70                        </variable>
71                        <variable name="Out2">
72                            <type>
73                                <LREAL/>
74                            </type>
75                        </variable>
76                        <variable name="Out3">
77                            <type>
78                                <LREAL/>
79                            </type>
80                        </variable>
81                    </outputVars>
82                    <localVars>
83                        <variable name="DiscreteTransferFcn_states">
84                            <type>
85                                <LREAL/>
86                            </type>
```

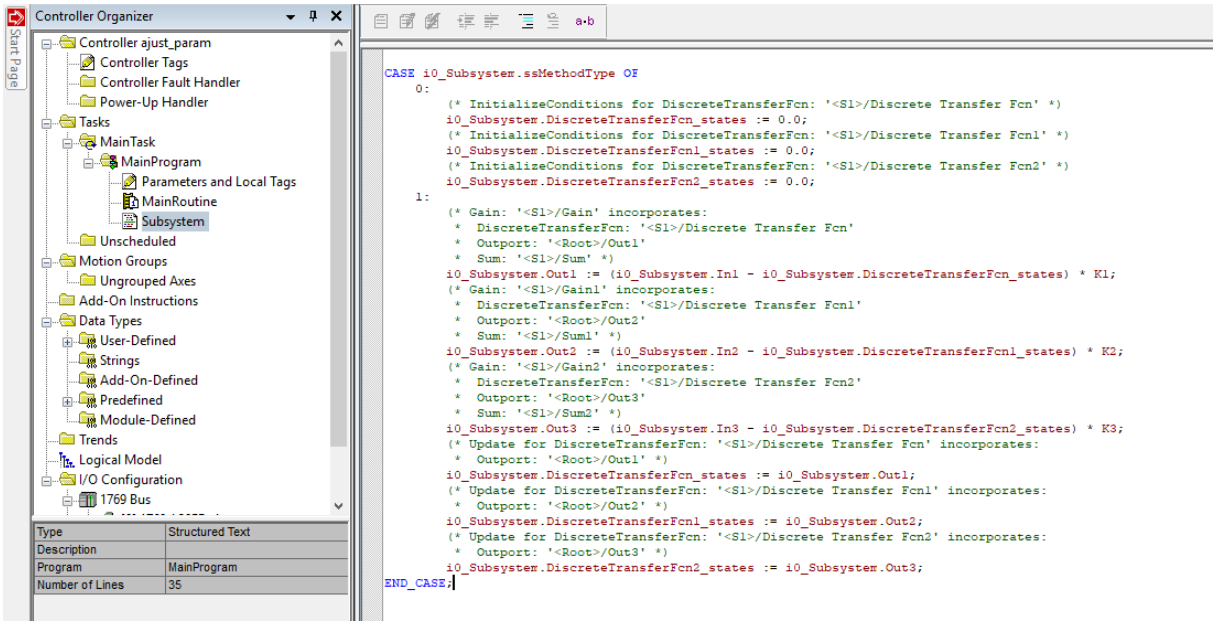
```
87                      </variable>
88                      <variable name="DiscreteTransferFcn1_states">
89                          <type>
90                              <LREAL/>
91                          </type>
92                      </variable>
93                      <variable name="DiscreteTransferFcn2_states">
94                          <type>
95                              <LREAL/>
96                          </type>
97                      </variable>
98                  </localVars>
99              </interface>
100             <body>
101                 <ST>
102                     <xhtml xmlns="http://www.w3.org/1999/xhtml">
103 <![CDATA[
104
105 CASE ssMethodType OF
106     0:
107         (* InitializeConditions for DiscreteTransferFcn: ...
               '<S1>/Discrete Transfer Fcn' *)
108         DiscreteTransferFcn_states := 0.0;
109         (* InitializeConditions for DiscreteTransferFcn: ...
               '<S1>/Discrete Transfer Fcn1' *)
110         DiscreteTransferFcn1_states := 0.0;
111         (* InitializeConditions for DiscreteTransferFcn: ...
               '<S1>/Discrete Transfer Fcn2' *)
112         DiscreteTransferFcn2_states := 0.0;
113     1:
114         (* Gain: '<S1>/Gain' incorporates:
115          *  DiscreteTransferFcn: '<S1>/Discrete Transfer Fcn'
116          *  Outport: '<Root>/Out1'
117          *  Sum: '<S1>/Sum' *)
118         Out1 := (In1 - DiscreteTransferFcn_states) * K1;
119         (* Gain: '<S1>/Gain1' incorporates:
120          *  DiscreteTransferFcn: '<S1>/Discrete Transfer Fcn1'
121          *  Outport: '<Root>/Out2'
122          *  Sum: '<S1>/Sum1' *)
123         Out2 := (In2 - DiscreteTransferFcn1_states) * K2;
124         (* Gain: '<S1>/Gain2' incorporates:
125          *  DiscreteTransferFcn: '<S1>/Discrete Transfer Fcn2'
126          *  Outport: '<Root>/Out3'
127          *  Sum: '<S1>/Sum2' *)
128         Out3 := (In3 - DiscreteTransferFcn2_states) * K3;
129         (* Update for DiscreteTransferFcn: '<S1>/Discrete Transfer ...
               Fcn' incorporates:
```

```
130          *  Outport: '<Root>/Out1' *)
131         DiscreteTransferFcn_states := Out1;
132         (* Update for DiscreteTransferFcn: '<S1>/Discrete Transfer ...
               Fcn1' incorporates:
133          *  Outport: '<Root>/Out2' *)
134         DiscreteTransferFcn1_states := Out2;
135         (* Update for DiscreteTransferFcn: '<S1>/Discrete Transfer ...
               Fcn2' incorporates:
136          *  Outport: '<Root>/Out3' *)
137         DiscreteTransferFcn2_states := Out3;
138 END_CASE;
139
140 ]]>
141 </xhtml>
142                 </ST>
143             </body>
144         </pou>
145     </pous>
146   </types>
147   <instances>
148     <configurations>
149       <configuration name="Configuration">
150         <resource name="ajust_param_res">
151           <globalVars>
152             <variable name="K1">
153               <type>
154                 <LREAL/>
155               </type>
156               <initialValue>
157                 <simpleValue value="0.1"/>
158               </initialValue>
159             </variable>
160             <variable name="K2">
161               <type>
162                 <LREAL/>
163               </type>
164               <initialValue>
165                 <simpleValue value="0.2"/>
166               </initialValue>
167             </variable>
168             <variable name="K3">
169               <type>
170                 <LREAL/>
171               </type>
172               <initialValue>
173                 <simpleValue value="0.3"/>
174               </initialValue>
```

```
175                         </variable>
176                     </globalVars>
177                 </resource>
178             </configuration>
179         </configurations>
180     </instances>
181 </project>
```

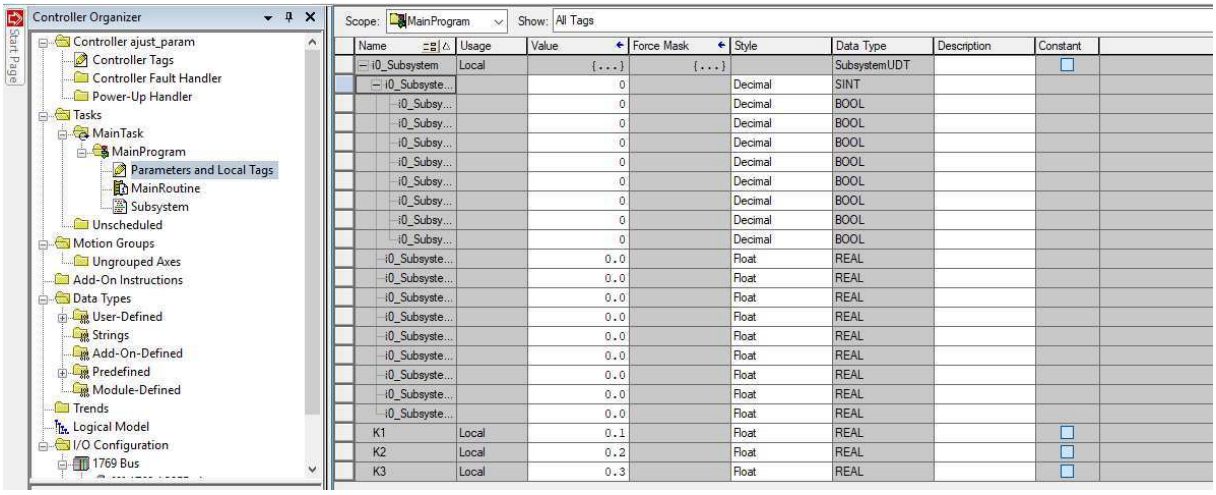
Código gerado para o Studio 5000® e as *tags* correspondentes a este código.

Figura 20 – Código em Texto Estruturado gerado para o Studio 5000®



Fonte: Autoria própria, 2018

Figura 21 – Tags geradas automaticamente para dar suporte ao código



Fonte: Autoria própria, 2018

Um subsistema simples, onde três entradas passam por um parâmetro de ganho e saída retorna como realimentação negativa após passar por uma Transformada Z. O código mostra todos os passos comentados, as *tags* com os valores deifinidos para K1, K2 e K3, e as demais iniciadas em zero.

Contudo alguns imprevistos impossibilitaram a validação, via aplicação do código nos CLPS. A versão do Studio 5000® utilizada não é compatível com o equipamento presente no laboraótio, e a versão do RS Logix 5000® (outra IDE da Rockwell Automation®) na qual fazemos a programação dos CLPs não é compatível com o código gerado pelo MATLAB®, ou seja, um problema de compatibilidade entre IDE, CLP e MATLAB® impossibilitou a validação prática do código.

4 CONCLUSÃO

No presente trabalho foi feita a abordagem teórica sobre a Automação Industrial, mostrando como a mesma funciona e tem evoluído através dos anos, desde o fim do Século XIX, com a Revolução Industrial. Posteriormente é feito um estudo dos CLPs, que constituem o cérebro dos processos automáticos, a maneira como são programados e em que partes do meio industrial são utilizados. Em seguida apresenta-se as normas de programação e a PLCOpen®, trabalhando em conformidade. O desenvolvimento de IDEs e CLPs passou a ser normatizado e, dessa maneira, procurou-se atingir o máximo de compatibilidade e reuso de códigos. Por fim, foi estudado uma maneira de geração automática de código para ser implementadas nos CLPs do Laboratório de Automação.

Por fim, o resultado foi bastante satisfatório, pois permitiu a análise da instituição PLCOpen®, até então pouco estudada no meio acadêmico e o uso de uma ferramenta muito útil para a portabilidade e economia de tempo. A oportunidade de estudo e trabalho em uma IDE e CLP de uso profissional, também foi enriquecedora, visto que, na academia os estudos teóricos acabam ocupando quase todo o espaço de carga-horária, sobrando assim pouco tempo para atividades práticas.

Bibliografia

1 O que é Automação Industrial? Disponível em: <https://www.siembra.com.br-/noticias/o-que-e-automacao-industrial/>. Acesso em: 13/12/2018. Citado 2 vezes nas páginas 11 e 12.

2 USP. *A Norma IEC 61131*. [S.l.]. Disponível em: <https://edisciplinas.usp.br-/pluginfile.php/4203394/mod_resource/content/0/IEC_61131-3.pdf>. Acesso em: 12/12/2018. Citado na página 11.

3 SILVEIRA, C. B. *O que é Automação Industrial*. Disponível em: <https://www-.citisystems.com.br/o-que-e-automacao-industrial/>. Acesso em: 13/12/2018. Citado 2 vezes nas páginas 12 e 13.

4 PLCOPEN®. *PLCOpen®*. Disponível em: <http://plcopen.org/>. Acesso em: 03/12/2018. Citado 5 vezes nas páginas 15, 16, 20, 21 e 24.

5 MATHWORKS®. *Generate IEC 61131-3 Structured Text for PLCs and PACs*. Disponível em: <https://www.mathworks.com/products/sl-plc-coder.html>. Acesso em: 06/12/2018. Citado na página 25.

6 GAIAO, P. B. T. Relatório de Estágio, *Relatório de Estágio Supervisionado*. 2018. Citado na página 25.