UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Pedro Barros Torreão Gaião

RELATÓRIO DE ESTÁGIO SUPERVISIONADO

CAMPINA GRANDE
Agosto de 2018

PEDRO BARROS TORREÃO GAIÃO

RELATÓRIO DE ESTÁGIO SUPERVISIONADO

Relatório de Estágio Supervisionado submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para obtenção do título de Graduado em Engenharia Elétrica.

Campina Grande
Agosto de 2018

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

PEDRO BARROS TORREÃO GAIÃO

Este Relatório de Estágio Supervisionado foi julgado adequado para a obtenção do título de Bacharel em Engenharia Elétrica, sendo aprovado em sua forma final pela banca examinadora:

Orientador: Prof. George Acioli Júnior
Universidade Federal de Campina Grande - UFCG

Professor Convidado: Rafael Bezerra Correia Lima
Universidade Federal de Campina Grande - UFCG

Campina Grande
Agosto de 2018

Este trabalho é dedicado à minha família e a todos que de alguma forma contribuíram para que eu conseguisse realizar esse sonho.

Agradecimentos

Agradeço em primeiro lugar a Deus por olhar por mim em todos os momentos de minha vida e por mais essa graça concedida.

Agradeço aos meus pais e irmãs, por todo amor e carinho, e aos meus tios, João Bosco e Vera Lúcia, por ajudar a tornar esse sonho uma realidade.

Agradeço ao meu orientador Professor George Acioli Júnior e ao meu supervisor Professor Péricles Rezende Barros, tanto pela oportunidade quanto por todos os conselhos, paciência e ajuda nesse período.

Aos meus amigos e colegas de curso pelo suporte e companheirismo ao longo da graduação.

Aos meus professores, todos os meus professores, cuja dedicação e conhecimento contribuíram significamente para a minha formação como aluno.

"Existe um tempo certo para cada coisa,
momento oportuno para cada propósito debaixo do Sol:
Tempo de nascer, tempo de morrer; tempo de plantar,
tempo de colher.
(Bíblia Sagrada, Eclesiastes 3)

Resumo

Este relatório apresenta as atividades realizadas pelo aluno Pedro Barros Torreão Gaião durante o Estágio Supervisionado no Laboratório de Instrumentação Eletrônica e Controle, pertencente ao Departamento de Engenharia Elétrica da Universidade Federal de Campina Grande, sob orientação do Professor George Acióli Júnior e supervisão do Professor Péricles Rezende Barros.

Palavras-chave: CLP, Texto Estruturado, Ladder, Controle Preditivo baseado em Modelo, DeltaV.

Abstract

This paper presents the activities executed by the student Pedro Barros Torreão Gaião during his Supervised Internship in the Laboratory of Electronic Instrumentation and Control, belonging to the Department of Electrical Engineering of the Federal University of Campina Grande, under the orientation of Professor George Acióli Júnior and the supervision of Professor Péricles Rezende Barros.

Keywords:PLC, Structured Text, Ladder, Model Predictive Control, DeltaV.

Lista de ilustrações

Lista de abreviaturas e siglas

CLP	Controlador Lógico Programável
PLC	*Programmable Logic Controller*
ST	*Structured Text*
UFCG	Universidade Federal de Campina Grande
DEE	Departamento de Engenharia Elétrica
LIEC	Laboratório de Instrumentação Eletrônica e Controle
MPC	*Model Predictive Control*
DMC	*Dynamic Matrix Control*
SDCD	Sistema Digital de Controle Distribuído
IHM	Interface Homem Máquina

Sumário

1 Introdução

1.1 Contexto

A disciplina Estágio Supervisionado é componente obrigatório da grade curricular do Curso de Engenharia Elétrica da Universidade Federal de Campina Grande e tem como principal objetivo possibilitar que o aluno em final de curso utilize os conhecimentos adquiridos nas disciplinas ao longo da sua formação para desempenhar atividades que o ajudem a se preparar para o que será encontrado no âmbito profissional. Este relatório contém a descrição das tarefas realizadas entre o período de 07 de maio a 26 de julho de 2018 com carga horária de 16 horas semanais, totalizando 185 horas sob a orientação do professor George Acioli Júnior e supervisão do professor Péricles Rezende Barros.

O estágio foi realizado no Laboratório de Instrumentação Eletrônica e Controle (LIEC) do Departamento de Engenharia Elétrica (DEE) da Universidade Federal de Campina Grande, situado na Rua Aprígio Veloso, 882, no Bairro Universitário, no município de Campina Grande, estado da Paraíba. O LIEC foi fundado em 1973 e fica situado no Setor C do Campus I da UFCG. Nele são desenvolvidas pesquisas nas áreas de Controle de Processos, Automação Industrial e Instrumentação Eletrônica. A fachada do laboratório está representada na Figura 1.

Figura 1 – Fachada do Laboratório de Instrumentação Eletrônica e Controle - LIEC (Fonte: *http://liec.ufcg.edu.br/*).

1.2 Atividades

Do ponto de vista das atividades realizadas, estas podem ser separadas em duas categorias principais. A primeira foi o estudo e devida documentação dos procedimentos e configurações necessárias para geração de código para controladores lógico programáveis (CLP) utilizando a ferramenta *Simulink PLC Coder* do software MATLAB, da fabricante *MathWorks*. A ferramenta *Simulink PLC Coder* possibilita tanto a geração de código baseado em texto, utilizando o Texto Estruturado (*ST - Structured Text*), quando a geração de código baseado em representações gráficas, utilizando a linguagem de programação *Ladder*. A segunda categoria de atividades compreende a documentação dos procedimentos e configurações necessárias à utilização do bloco de Controle Preditivo por Modelo fornecido pelo Sistema Digital de Controle Distribuído (SDCD), modelo *DeltaV*, da fabricante *Emerson Process Management*.

1.3 Organização

Como foi citado na seção anterior, o presente trabalho está organizado em duas partes principais, uma referente à geração de código para CLP e outra referente à utilização do bloco de controle preditivo por modelo do sistema *DeltaV*. Além dessa divisão por categoria, o relatório está organizado em capítulos. Segue abaixo uma breve descrição dos conteúdos abordados em cada capítulo:

Capítulo 2: apresenta uma breve introdução sobre os controladores lógico programáveis, seus elementos básicos e princípios de operação.

Capítulo 3: apresentação da ferramenta *Simulink PLC Coder* e dos procedimentos para geração de código para CLP.

Capítulo 4: visão geral sobre o SDCD, seus princípios de funcionamento e utilização do bloco MPC.

Capítulo 5: considerações finais.

Parte I

GERAÇÃO DE CÓDIGO PARA CLP

2 Controlador Lógico Programável

2.1 Apresentação

Um Controlador Lógico Programável (CLP) é um computador industrial especializado capaz de armazenar funções de sequência, temporização, contagem, aritmética, manipulação de dados e comunicação para controlar máquinas e processos industriais. O CLP tem sua origem no controle de sistemas baseado em dispositivos eletromecânicos, conhecidos como relés. Antes do CLP ser amplamente difundido na indústria, todo sistema de controle automático era implementado através da utilização de relés, chaves, temporizadores, contadores, etc. Para que estes sistemas fossem montados, eram necessários enormes paineis cheios de relés e muitas fiações, que levavam muito tempo para serem montados e tornava a manutenção um pesadelo. Além de tomarem muito espaço e exigirem muito tempo e trabalho para serem montados, caso um problema fosse identificado ou a lógica precisasse ser alterada, significava, na maioria dos casos, que toda a fiação da unidade precisaria ser refeita, o que obviamente elevava os custos. Adicionalmente, por se tratarem de dispositivos eletromecânicos, os relés possuíam uma vida útil, o que causava muitas interrupções operacionais. Como se não bastasse todas as desvantagens mencionadas, como não era possível testar o sistema antes de colocá-lo para funcionar, qualquer pequena falha no diagrama esquemático ou erro de montagem, poderia resultar em prejuízos elevados [HANSSEN].

Os primeiros CLPs era relativamente simples e tinham como único objetivo substituir a lógica a relé. Gradualmente, novas funcionalidades foram sendo incorporadas, tais como contadores, temporizadores, entradas e saídas analógicas, funções aritméticas, etc. Nos dias atuais as unidades são capazes de efetuar funções matemáticas complexas incluindo integração numérica e diferenciação, além de operarem a elevadas velocidades de processamento. Os controladores lógico programáveis encontram aplicações em praticamente todos os segmentos industriais, sendo alguns dos setores mais comuns, a indústria automotiva, petroquímica, a manufatura, a mineração, a indústria alimentícia [BRYAN L. A.; BRYAN].

No entanto, conforme o número de fabricantes de CLPs aumentou, aumentou também a diversidade de protocolos de comunicação proprietários. A falta de padronização juntamente ao contínuo desenvolvimento tecnológico, transformou a comunicação de CLPs em um emaranhado de protocolos de comunicação incompatíveis, o que até hoje representa um problema embora muitos fabricantes hoje em dia ofereçam soluções para comunicação utilzando protocolos conhecidos e padronizados. Várias linguagens de programação também foram desenvolvidas, como as linguagens de programação gráfica Ladder (LD), que lembra

diagramas elétricos, o Diagrama de Blocos de Função (FBD - *Function Block Diagram*), e o Gráfico de Funções Sequencias (SFC -*Sequential Function Chart*), e as linguagens de programação baseadas em texto (*text-based*), como a Lista de Instruções (IL -*Instruction List*) e o Texto Estruturado (ST - *Structured Text*), sendo a última uma linguagem de mais alto-nível que fornece associação com linguagens como Pascal e C. Todas as linguagens mencionas foram incorporadas pelo padrão internacional IEC 61131-3 (*International Electrotechnical Commission*, 2013).

Devido à grande diversidade encontrada no mercado, os CLPs podem ser encontrados em diferentes configurações e tamanhos, sendo mais comuns a configuração compacta e a modular. Na configuração compacta, representada na Figura 2, todos os elementos constituintes do controlador como, por exemplo, a unidade de processamento, a memória e as interfaces de entradas/saídas, estão contidas em um único dispositivo. A principal vantagem desse modelo é o baixo custo, no entanto, possui baixa flexibilidade. No caso da configuração modular, representada na Figura 3, todos os elementos constituintes do controlador estão separados em módulos. O custo é mais elevado, porém possibilita grande flexibilidade, permitindo ao usuário montar o sistema da maneira que melhor atende sua estratégia de controle.

Figura 2 – CLP Compacto.

Figura 3 – CLP Modular

2.2 Elementos Básicos

Esquematicamente, pode-se dividir um CLP em seis unidades principais, como pode ser visualizados na Figura 4. Essas unidades principais são a unidade de processamento, a memória, a fonte de alimentação, os módulos de entrada e saída, e os módulos de comunicação, e são conectadas entre si através de barramentos. Esses barramentos, por sua vez, podem ser de quatro tipos:

- Barramento de dados: utilizado para transferência de dados entre o processador, a memória e os módulos de entrada/saída;
- Barramento de endereço: utilizado para transferir os endereços de memória de onde os dados serão obtidos e para onde serão enviados;
- Barramento de controle: utilizado para sincronização e controle;
- Barramento do sistema: utilizado para comunicação das entradas e saídas.

Para entender como um CLP funciona, é necessário ter uma noção básica sobre a função de cada uma das unidades principais.

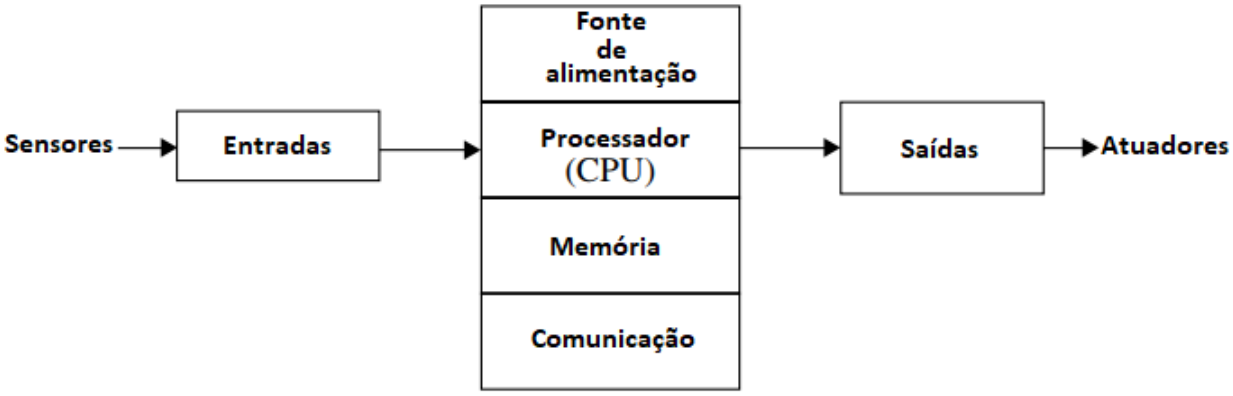


Figura 4 – Elementos básicos de um CLP.

2.2.1 Processador

Consiste de um microprocessador responsável por controlar as funções do CLP. É no processador que são efetuados os cálculos e o controle do fluxo de informações e de como o programa opera. O processador recebe os dados de entrada proveniente dos sensores, executa o programa armazenado e envia para as saídas os sinais apropriados para controle dos dispositivos atuadores.

2.2.2 Memória

Armazenam os dados de entrada e saída bem como o conjunto de procedimentos que operam as funções do CLP. O tamanho da memória varia de um fabricante de CLP para o outro e podem ainda ser voláteis ou não. Um CLP tem normalmente as seguintes unidades de memória:

- ROM (*Read-only memory*): esse tipo de memória é não-volátil, ou seja, mantém os dados armazenados quando a alimentação é desligada. Além disso, permite apenas leitura, ou seja, é utilizada para armazenamento permanente do sistema operacional e dos dados do sistema. Como a informação armazenada em uma memória ROM não pode ser deletada ou alterada, uma memória programável apagável somente de leitura (EPROM - *Erasable programmable read-only memory*) pode ser utilizada quando se deseja atualizar o sistema operacional do CLP.
- RAM (*Random access memory*): a memória de acesso aleatório pode ser utilizada tanto para leitura quanto para a escrita de dados, e é responsável pelo armazenamento dos programas do controlador. Esse tipo de memória é volátil, o que significa que os dados são perdidos quando a alimentação é desligada, razão pela qual os CLPs possuem uma bateria de modo a evitar que o código do programa não seja perdido.

2.2.3 Fonte de alimentação

Todo CLP deve ser conectado à uma fonte alimentação que, por sua vez, pode ser um módulo separado, no caso dos CLPs modulares, ou pode ser integrada ao restante do sistema, como no caso dos CLPs compactos.

2.2.4 Unidade de comunicação

Todos os CLPs possuem uma conexão para cabos de programação, paineis de operação, impressoras ou redes. Essa unidade incorpora um ou mais protocolos de comunicação e vários padrões são utilizados, tanto para a porta de programação, quanto para conexão com outros dispositivos.

2.2.5 Entradas e Saídas

Os módulos de entrada e saída possibilitam a comunicação do CLP com o mundo externo e são especificados de acordo com os sinais de entrada e saída associados com o objetivo da aplicação. Esses módulos podem ser discretos ou analógicos, como chaves de fim de curso, transdutores de pressão, botões, solenóides entre outros. Cada entrada e saída possui um endereço específico de memória que pode ser acessado pelo usuário através do código do programa.

2.3 Princípios de operação

Durante sua operação, o primeiro procedimento que a CPU realiza é a leitura dos dados de entrada dos dispositivos de campo através das interfaces de entrada e o armazenamento desses dados na memória. Após esse procedimento, a CPU executa o programa armazenado na memória do sistema que, por sua vez, pode estar escrito em qualquer das linguagens já mencionadas. Por fim, a CPU escreve ou atualiza os dispositivos de saída através das interfaces de saída. Esse processo de leitura das entradas, resolução do programa e atualização das saídas é chamado de ciclo varredura ou ciclo da CPU, e o tempo entre uma atualização e outra é chamado de tempo de varredura. Quanto maior for o programa ou menor a capacidade de processamento da CPU, maior será o tempo de varredura. Um resumo desse processo está representado na Figura 5

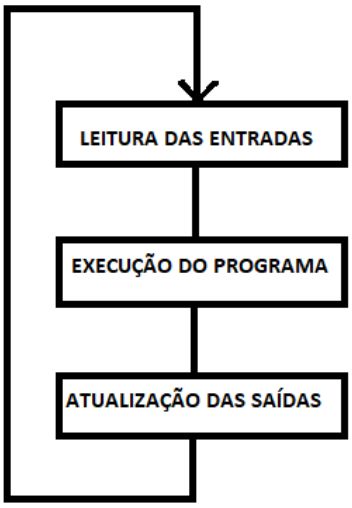


Figura 5 – Ciclo da CPU.

2.4 Texto estruturado

Como o nome indica, o texto estruturado (ST) é uma linguagem baseada em texto e se trata de uma linguagem de alto nível onde muitas operações e instruções podem ser realizadas com uma única linha de comando. Essa linguagem se destaca especialmente na programação de funções aritméticas complexas, manipulação de tabelas, processamento de números e no manuseio de dados estruturados.

Em um primeiro momento, pode parecer melhor utilizar uma linguagem de programação gráfica no desenvolvimento de aplicações com CLPs pelo fato da representação dos elementos utilizados facilitar a compreensão. No entanto, a utilização de uma linguagem baseada em texto na maioria das vezes resulta em uma programação mais rápida, programas menores em comparação com a linguagem Ladder, e uma lógica de fácil entendimento.

Além disso, algumas operações não tem como ser implementadas através de uma linguagem de programação gráfica.

Para usuários habituados a programar em linguagens de alto nível como a linguagem de programação C, o texto estruturado não apresenta nenhuma dificuldade de entendimento e/ou implementação visto que ambas compartilham muitas funcionalidades como, por exemplo, a utilização de declarações condicionais `IF ... ELSE` e `CASE`, e a utilização de iteração com laços de repetição como `FOR` e `WHILE`. Abaixo segue um pequeno exemplo de um programa utilizando essa linguagem.

```
PROGRAM exemploST
        VAR
           x : BOOL;
        END_VAR
        x := TRUE;
        REPEAT
           x := FALSE;
           UNTIL
              x := FALSE;
        END_REPEAT;
END_PROGRAM;
```

Figura 6 – Texto estruturado - Exemplo.

Nesse exemplo, observa-se que o programa começa com `PROGRAM`, seguido do nome do programa `exemploST`, e termina com `END_PROGRAM`. Tudo que se encontra entre essas duas linhas de código é considerado parte do programa. Continuando, a linha de código `VAR` indica o início da declaração de variáveis, onde uma variável `x` é declarada como sendo do tipo booleana. A linha de código `END_VAR`, por sua vez, finaliza a declaração das variáveis. Em seguida, tem-se a atribuição do valor booelano `TRUE` à variável `x`. A linha de código `REPEAT` sinaliza o início de um laço de repetição que termina com a declaração `END_REPEAT`.

2.5 Linguagem Ladder

Um diagrama Ladder é basicamente uma representação das etapas de um programa utilizando contatos de relé e bobinas. O diagrama Ladder é projetado com os contatos do lado esquerdo e as bobinas do lado direito. Isso é um resquício de quando os sistemas de controle eram baseados em relés e diagramas elétricos muito similares eram utilizados. O diagrama de contatos (Ladder) consiste em um desenho formado por duas linhas verticais, que representam os pólos positivo e negativo de uma bateria, ou fonte de alimentação genérica. Entre as duas linhas verticais são desenhados ramais horizontais com os contatos e as bobinas. Esses contatos são elementos gráficos associados com uma variável booleana

que representam os estados das entradas do CLP e podem ser representados de diversas formas, sendo as mais básicas o contato normalmente aberto e o normalmente fechado. Na Figura 7 estão representados os elementos básicos da linguagem Ladder.

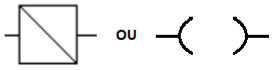
Nomeclatura	Abreviação	Símbolo
Contato Normalmente Aberto	NA	
Contato Normalmente Fechado	NF	
Bobina ou Saída	---	

Figura 7 – Linguagem Ladder - Elementos básicos.

Geralmente, um diagrama Ladder é baseado no princípio de que, se uma condição ou combinação de condições é satisfeita, então uma ou mais ações (eventos ou instruções) serão executadas. O CLP executa o programa em Ladder sempre da esquerda para a direita e de cima para baixo, verificando primeiramente se os contatos satisfazem uma determinada condição para que possa então fornecer ou não corrente para a bobina. Uma vez que o CLP executa uma parte do diagrama, essa mesma parte só é executada novamente no próximo ciclo de varredura.

Como um exemplo simples de implementação da linguagem Ladder, tem-se a porta lógica AND representada na Figura 8. Observa-se que, para que corrente passe para a bobina `O0001` é necessário que ambos os contatos `I0000` e `I0001` estejam fechados, isto é, possuam nível lógico alto (valor booleano `TRUE`), e que nenhuma outra combinação dos valores das entradas permite a passagem de corrente para `O0001`.

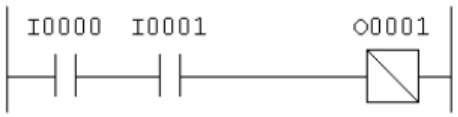


Figura 8 – Linguagem Ladder - Porta Lógica AND.

3 *Simulink PLC Coder*

3.1 Apresentação

3.1.1 Descrição do produto

O Simulink PLC Coder gera Texto Estruturado e Diagramas Ladder em formato PLCopen XML independentes de hardware a partir de modelos do Simulink e de diagramas de estado no Stateflow®. O código ou diagrama gerado segue a norma internacional IEC 61131-3, responsável pelo desenvolvimento do padrão de linguagens de programação de controladores lógico programáveis. Os códigos são gerados no formato PLCopen XML e outros formatos de arquivos compatíveis com diversos ambientes de desenvolvimento integrado (IDEs - *integrated development environments*) incluindo o CODESYS da fabricante 3S-Smart Software Solutions e o Studio 5000 da fabricante Rockwell Automation®.

3.1.2 Características principais

Dentre as muitas funcionalidades do *Simulink PLC Coder*, algumas podem ser destacadas:

- Geração automática de Texto Estruturado e Diagramas Ladder;
- Suporte a diversos ambientes de desenvolvimento integrado (IDEs);
- Suporte ao Simulink, incluindo sistemas reutilizáveis e blocos de controladores PID;
- Suporte ao Stateflow, incluindo máquinas de estado, funções gráficas, e tabelas verdade;
- Suporte ao MATLAB, incluindo declarações `if-else`, loops, e operações matemáticas.

O *Simulink PLC Coder* traz a abordagem de *Model-Based Design* para dentro do domínio dos controladores lógico programáveis, possibilitando que os projetistas e programadores passem mais tempo otimizando e melhorando seus algoritmos através de experimentos e protótipos, que podem ser implementados rapidamente com o uso desta ferramenta, ao invés de programando CLPs. Em uma aplicação típica, é possível utilizar um modelo no Simulink para simular um projeto a ser implementado em um CLP. Uma vez que o modelo satisfaz os requisitos de projeto, pode-se utilizar a ferramenta de verificação de compatibilidade do *Simulink PLC Coder* que verifica a conformidade entre a semântica

do modelo e os blocos da IDE do CLP a ser utilizado. Após verificar a compatibilidade, o *Simulink PLC Coder* pode ser utilizado para gerar código em texto estruturado que implemente o projeto incorporado no modelo. Caso o usuário não esteja satisfeito com o código gerado, é possível retornar quando quiser ao modelo original, modificá-lo e gerar novamente o código. Utilizando o *Simulink PLC Coder*, também é possível efetuar a geração de código em diagrama Ladder para para qualquer aplicação a partir de um diagrama de estados no Stateflow.

O *Simulink PLC Coder* pode ser utilizado como uma ferramenta de projeto e teste de algoritmos de controle para qualquer aplicação industrial que utilize controladores lógico programáveis e pode se revelar uma ferramenta muito útil na integração de sistemas.

3.2 Plataformas suportadas

O *Simulink PLC Coder* é compatível com diversos ambientes de desenvolvimento integrado disponíveis no mercado. Segue abaixo duas listas de plataformas compatíveis, a primeira lista abrange as plataformas suportadas para geração de código em Texto Estruturado, e a segunda lista abrange as plataformas suportadas para geração de código em linguagem de programação Ladder.

- IDE's suportadas para geração de código em Texto Estruturado:
  - 3S-Smart Software Solutions CODESYS Version 2.3, 3.3 e 3.5
  - B&R Automation Studio 3.0 ou 4.0
  - Beckhoff TwinCAT 2.11 ou 3
  - KW-Software MULTIPROG®5.0 ou 5.5
  - OMRON Sysmac Studio Version 1.04, 1.05, 1.09 ou 1.12
  - Phoenix Contact®PC WORX target IDE
  - Rexroth IndraWorks version 13V12 IDE
  - Siemens SIMATIC®STEP®7 Version 5.3, 5.4 ou 5.5
  - Rockwell Automation RSLogix 5000 Series Version 17, 18, 19 ou 20 e Rockwell Studio 5000 Logix Designer Version 21 ou 24
- IDE's suportadas para geração de código em linguagem Ladder:

- 3S-Smart Software Solutions CODESYS Version 2.3, 3.3 e 3.5
- PLCopen XML
- Rockwell Automation RSLogix 5000 Series Version 17, 18, 19 ou 20 e Rockwell Studio 5000 Logix Designer Version 21 ou 24

3.3 Geração de código em texto estruturado

3.3.1 Fluxo de tarefas

Fluxo de tarefas básico para geração de código com o Simulink PLC Coder:

1. Definir e projetar um modelo no Simulink
2. Identificar os componentes do modelo para os quais se deseja gerar o código a ser importado para o CLP
3. Colocar os componentes selecionados em um subsistema
4. Escolher a IDE alvo
5. Selecionar um modo de solução
6. Configurar o bloco de subsistema como atômico
7. Checar a compatibilidade do modelo com o Simulink PLC Coder
8. Simular o modelo
9. Configurar os parâmetros do modelo para geração do código
10. Examinar o código gerado
11. Importar o código para a IDE do CLP

3.3.2 Configurações necessárias

O Simulink PLC Coder gera código apenas para subsistemas de tarefa única. Para subsistemas de múltiplas tarefas é necessário configurar o modo de atribuição de tarefas para tarefa única antes de selecionar o modo de solução. Isso é feito na configuração do modelo, no painel de solução, desmarcar a opção 'Tratar cada taxa discreta como uma tarefa separada'.

Com relação ao modo de solução, caso o modelo possua passo variável, deve-se utilizar o modo de solução contínuo. Caso o modelo possua passo fixo, utilize o modo de solução discreto de passo fixo.

3.3.3 Exemplo

O modelo no Simulink a ser utilizado está representado na Figura 9.

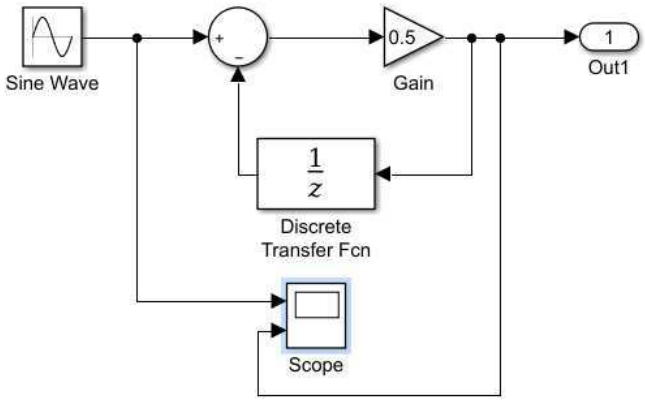


Figura 9 – Modelo - Simulink.

Em seguida, configura-se o modelo para utilizar o modo de solução discreto com passo fixo. No menu superior, selecione a aba **Simulation** → **Model Configuration Parameters** → **Solver** → **Type: Fixed-step** → **Solver: discrete**, seguindo a Figura 10. Salve o modelo.

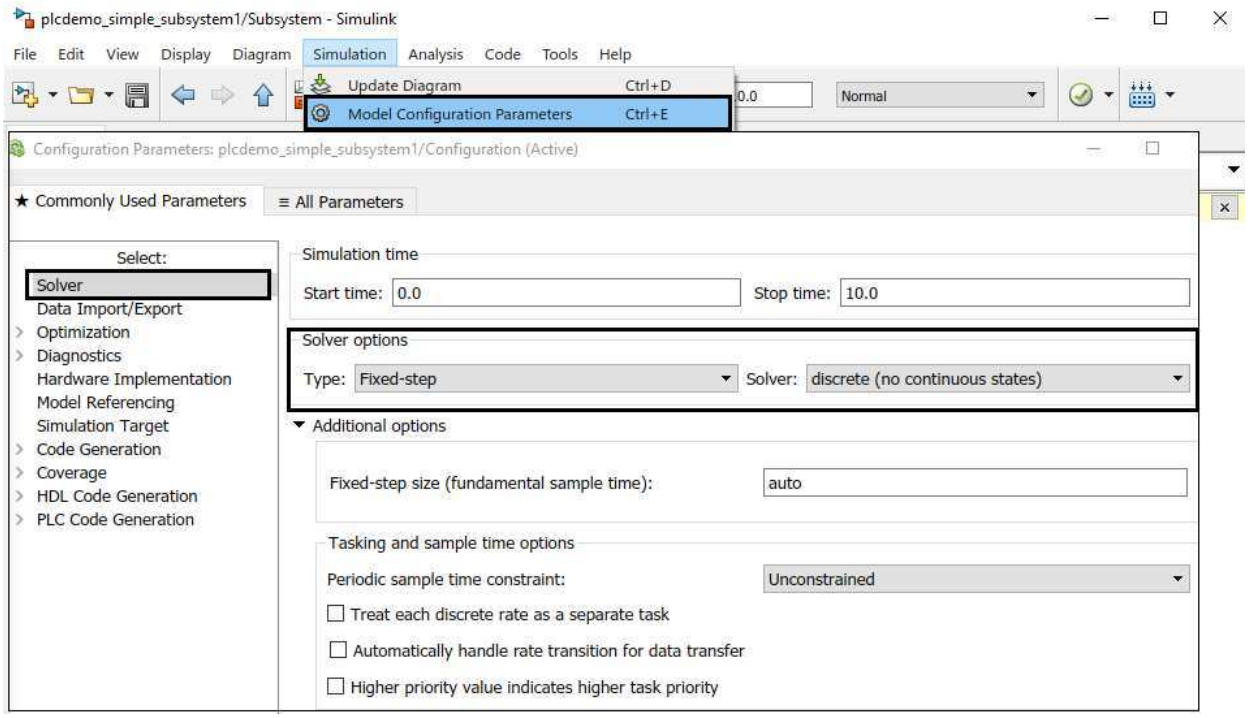


Figura 10 – Modo de solução.

Selecione os componentes para os quais se deseja gerar o código e crie um subsistema como mostrado na Figura 11:

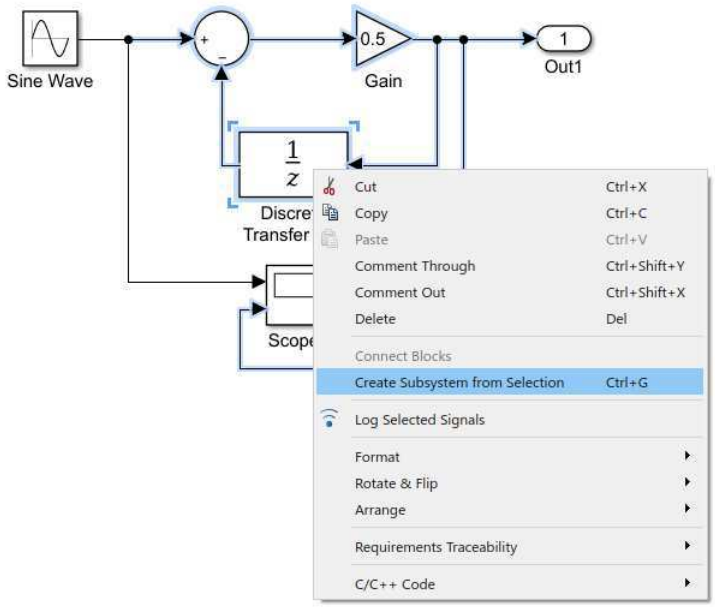


Figura 11 – Criação do subsistema.

Salve o modelo com o novo subsistema e, clicando com o botão direito sobre ele, selecione a opção **Block Parameters (Subsystem) → Treat as atomic unit**:

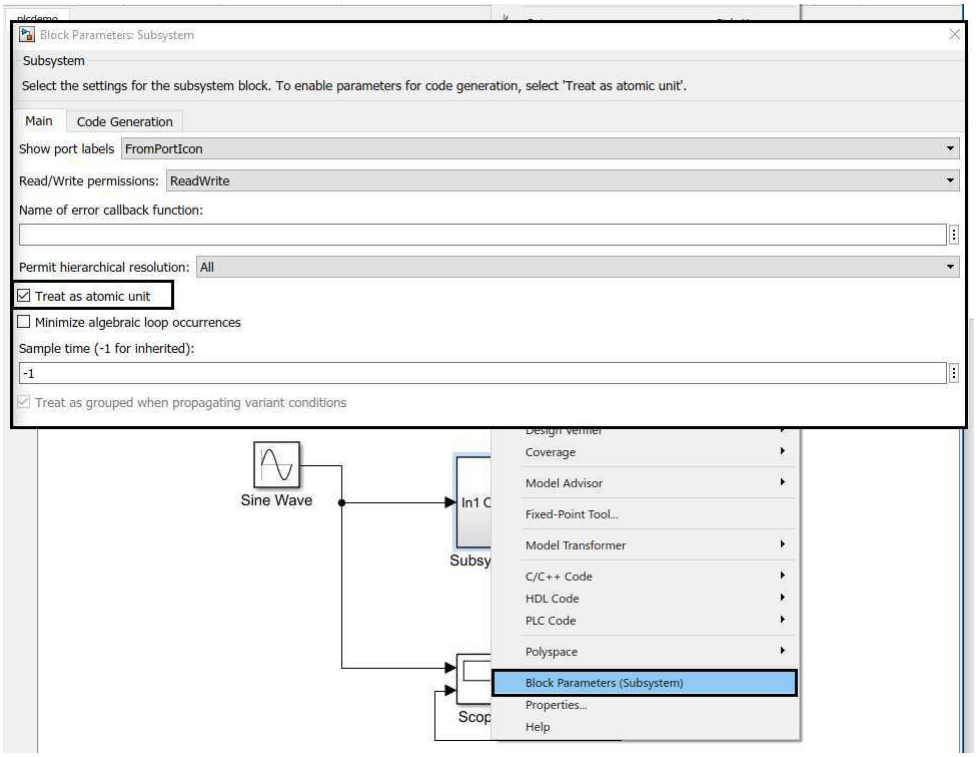


Figura 12 – Configuração do subsistema.

Simule o modelo e salve. Em seguida, verifique a compatibilidade do sistema para geração de código selecionando **PLC Code → Check Subsystem Compatibility**, como representado na Figura 13:

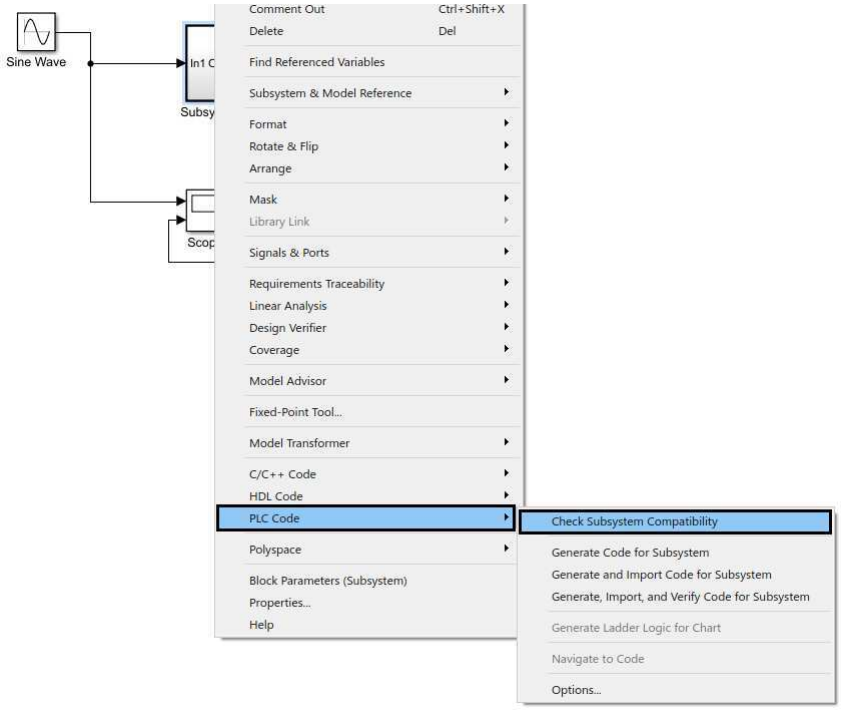


Figura 13 – Verificação de compatibilidade do subsistema.

O codificador verifica se o modelo satisfaz os critérios do *Simulink PLC Coder*. Finalizada a verificação, aparece a opção *View diagnostics* na parte inferior da janela, onde é possível visualizar se o subsistema passou na verificação de compatibilidade, como mostrado na Figura 14. Caso a verificação falhe, o usuário deve revisar todos os passos até então.

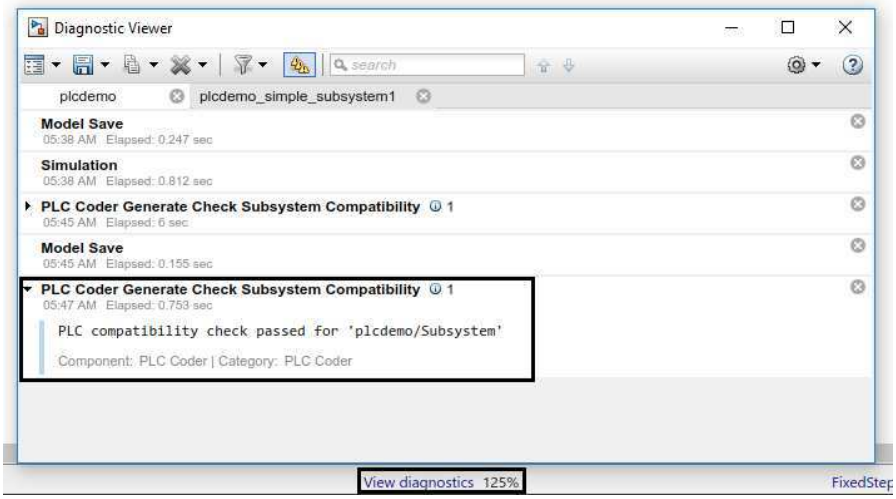


Figura 14 – Diagnóstico de verificação.

Efetuada a verificação de compatibilidade, é possível gerar o código em texto estruturado clicando com o botão direito sobre o subsistema e selecionando **PLC Code → Options → Target IDE: 3S CODESYS 3.5 → Apply → Generate code**. Esse procedimento está mostrado na Figura 15. Na opção Target IDE, o usuário deve selecionar a plataforma para a qual o código será gerado, nesse caso, utilizou-se o ambiente de desenvolvimento CODESYS versão 3.5. O nome do arquivo de código gerado tem o formato `modelname.exp`, por exemplo, `plcdemo`. A extensão `.exp` é característica do ambiente CODESYS, caso o ambiente de desenvolvimento alvo fosse o Studio 5000 da fabricante Rockwell Automation, a extensão do arquivo seria do tipo `.L5X`.

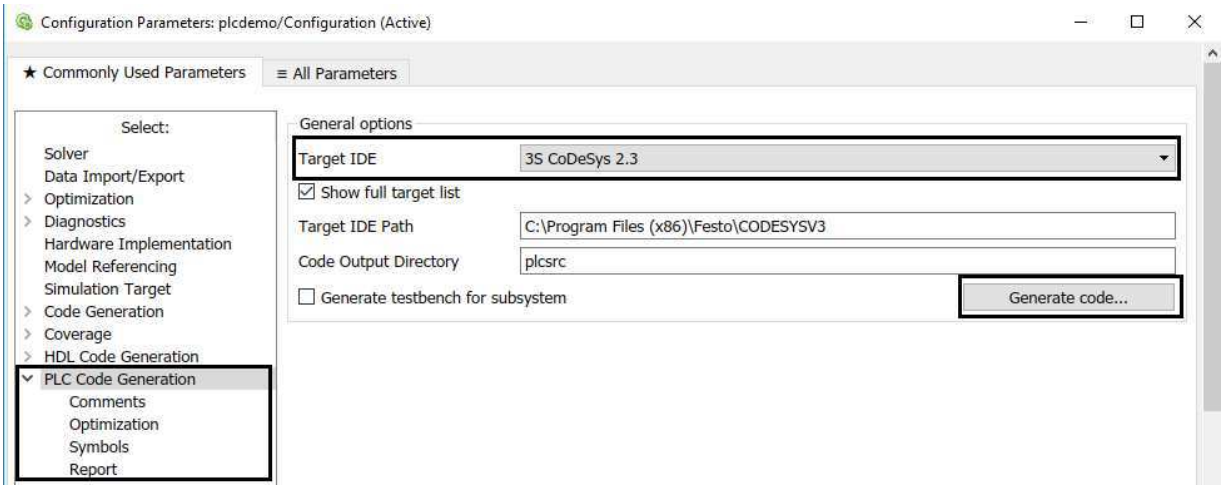


Figura 15 – Geração de código em texto estruturado.

É possível visualizar o código gerado a partir da importação do arquivo correspondente para o ambiente de desenvolvimento escolhido. Analisando o código gerado apresentado na Figura 16, na seção superior da figura são declaradas as variáveis de entrada e saída e os seus tipos correspondentes, onde `SINT` significa *Short Integer* e se refere a um valor inteiro entre -128 e 127, e `LREAL` significa *Long Real Number* e se refere à um número real entre $\pm 10^{\pm 308}$. Na seção inferior da figura estão contidas as instruções do código gerado para o subsistema. Essa seção consiste basicamente de uma estrutura condicional do tipo `CASE` que analisa o valor numérico da variável `ssMethodType`, caso seja igual à `SS_INITIALIZE`, é atribuído o valor zero à variável `DiscreteTransferFcn_states`, caso `ssMethodType` seja igual a `SS_STEP`, o valor da saída `Out1` é atualizado com o resultado proveniente da solução da função do subsistema e atribuído a `DiscreteTransferFcn_states`.

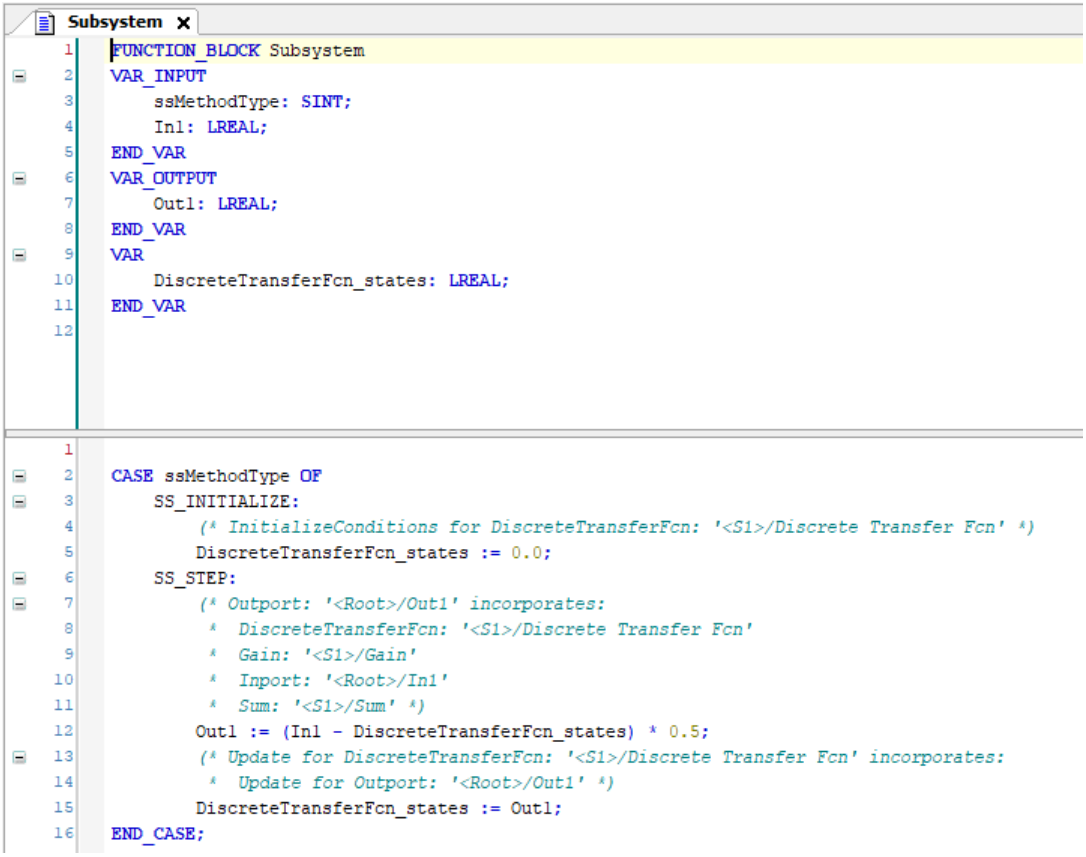


Figura 16 – Código em texto estruturado.

3.4 Geração de código em linguagem Ladder

Como mencionado no capítulo anterior, o Diagrama Ladder é uma linguagem de programação gráfica utilizada para desenvolver software para controladores lógico programáveis (CLPs) e é uma das linguagens padronizadas pela norma IEC 61131. Utilizando o Simulink PLC Coder, é possível gerar Diagramas Ladder a partir de diagramas de fluxo no Stateflow. Uma vez que o projeto esteja completo e o diagrama gerado, pode-se importar o código gerado para a IDE de preferência e visualizar o diagrama.

3.4.1 Fluxo de tarefas

1. Antes da geração do Diagrama Ladder deve-se confirmar que o as entradas e saídas do diagrama de fluxo são Booleanas e que cada estado no diagrama de fluxo corresponde à uma saída

2. Gere o código correspondente ao Diagrama Ladder à partir do diagrama de fluxo no Stateflow

3. Importe o código correspondente ao Diagrama Ladder para a IDE que será utilizada e valide o diagrama utilizando o parâmtros de teste gerados

3.4.2 Configurações necessárias

Para utilizar os diagramas de estado do Stateflow para geração de código para CLPs, é necessário que algumas propriedades sejam configuradas previamente, são elas:

- As entradas e saídas do diagrama devem ser do tipo booleana pois elas corresponderão aos terminais de entrada e saída do CLP;
- Cada estado no diagrama deve correponder à uma saída. A saída é verdadeira se o estado estiver ativo;
- As condições de transição devem envolver apenas operações do tipo booleana entre as entradas.

3.4.3 Exemplo

O diagrama de estados implementado no Stateflow a ser utilizado para geração de código em linguagem Ladder está representado na Figura 17.

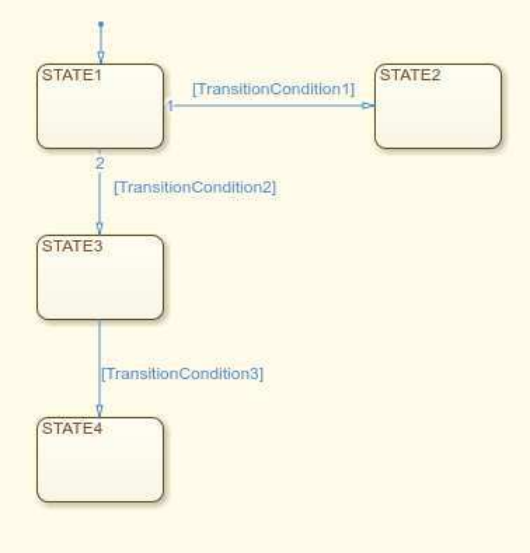


Figura 17 – Diagrama de estados.

A configuração de cada estado do diagrama como uma saída é feita da forma ilustrada na Figura 18, clicando com o botão direito do mouse sobre o estado, selecionando a opção **Properties → Create output for monitoring**.

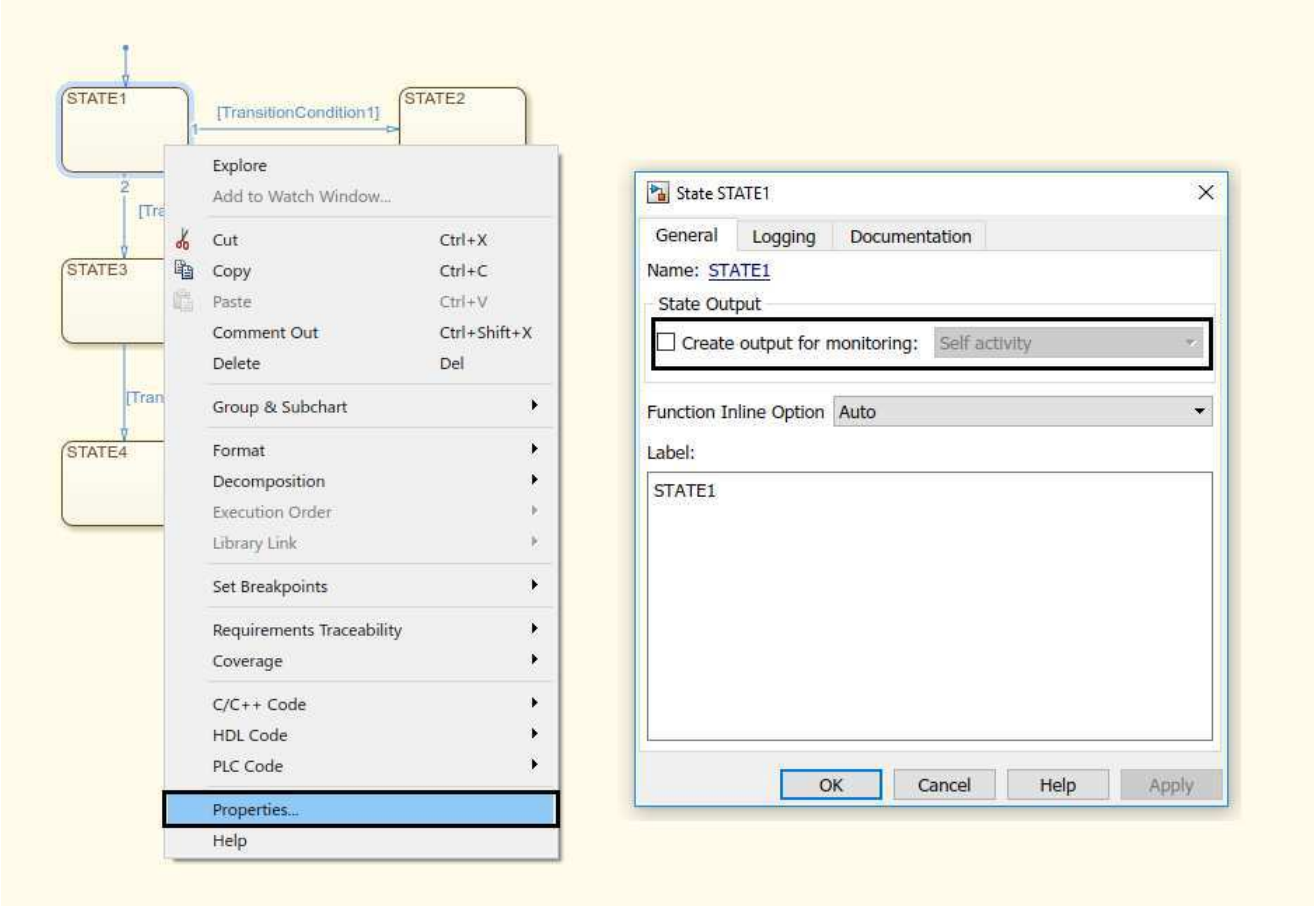


Figura 18 – Diagrama - Saída para monitoração.

Em seguida, é necessário configurar as entradas e saídas como sendo do tipo booleano. Para fazer isso é só selecionar a entrada correspondente no painel **Symbols** e configurá-la no painel **Property Inspector** → **Type: boolean**, como ilustrado na Figura 19.

O próximo passo é modificar a linguagem de ação de MATLAB para C clicando com o botão direito do mouse sobre o símbolo do MATLAB no canto inferior esquerdo da tela e selecionando **Properties** → **Action Language: C** como está demonstrado na Figura 20.

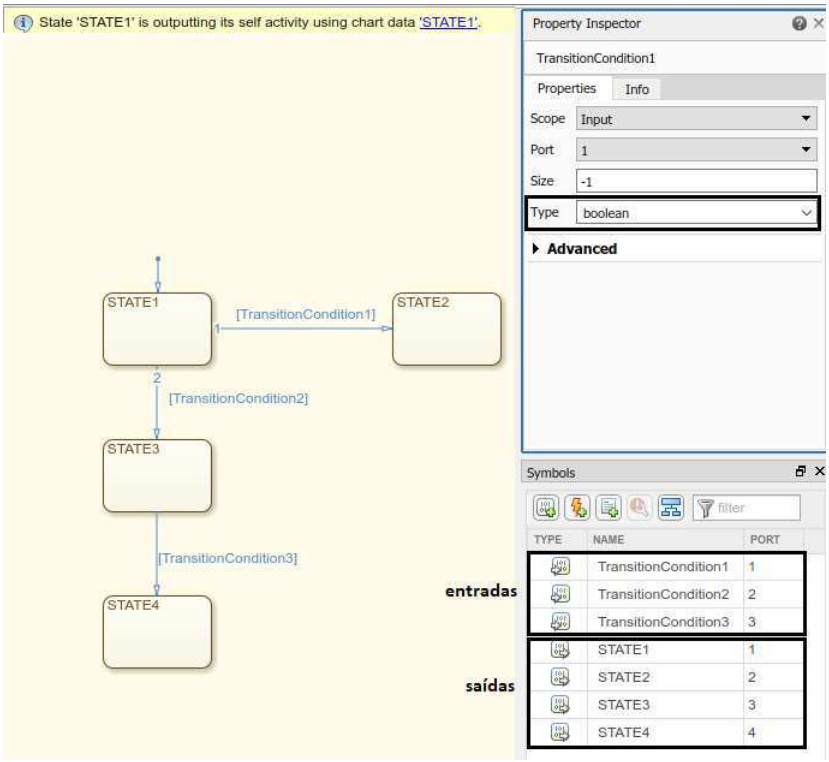


Figura 19 – Diagrama - configuração das E/S.

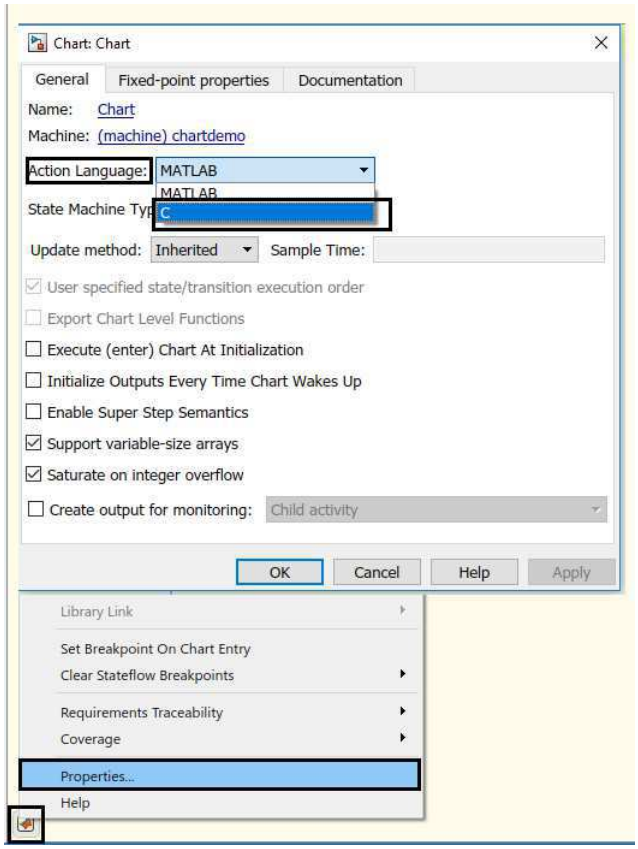


Figura 20 – Linguagem.

Finalizados todos os procedimentos anteriores, deve-se verificar a compatibilidade do diagrama gerado com a ferramenta *Simulink PLC Coder*. Para fazer isso, deve-se clicar com o botão direito do mouse sobre o diagrama de estados na janela do Simulink, e selecionar **PLC Code → Check Subsystem Compatibility**. Na Figura 21 está representado esse passo assim como a janela de análise da verificação, caso a verificação falhe, recomenda-se a revisão do processo até então.

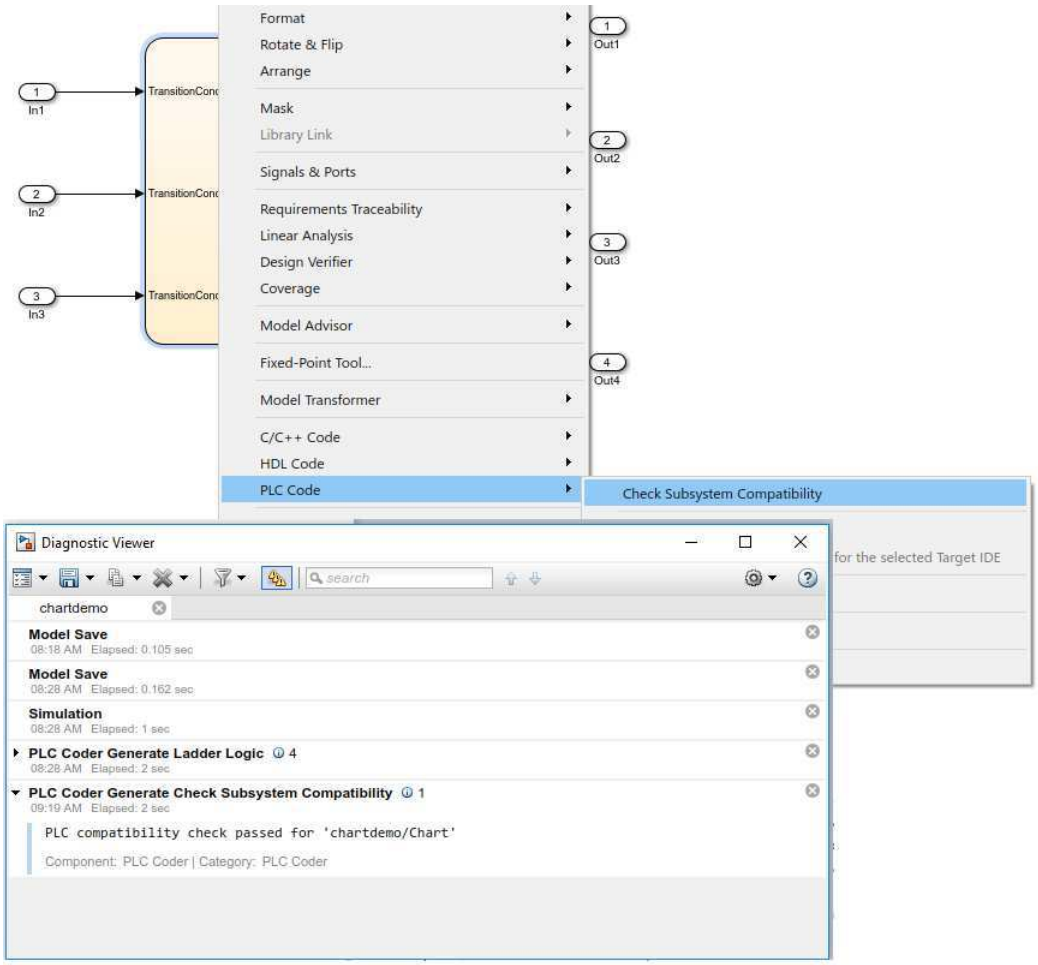


Figura 21 – Verificação de compatibilidade.

Após a verificação de compatibilidade, o diagrama de estados está pronto para servir de base para a geração de código em linguagem Ladder. Para fazer isso, deve-se clicar com o botão direito do mouse sobre o diagrama de estados na janela do Simulink, e selecionar **PLC Code → Options → Target IDE → Generate code**. Na opção Target IDE, o usuário deve selecionar a plataforma para a qual o código será gerado, nesse caso, utilizou-se o ambiente de desenvolvimento CODESYS versão 3.5. A execução desse passo está apresentada na Figura 22 e a janela de diagnótico da operação pode ser visualizada na Figura 23.

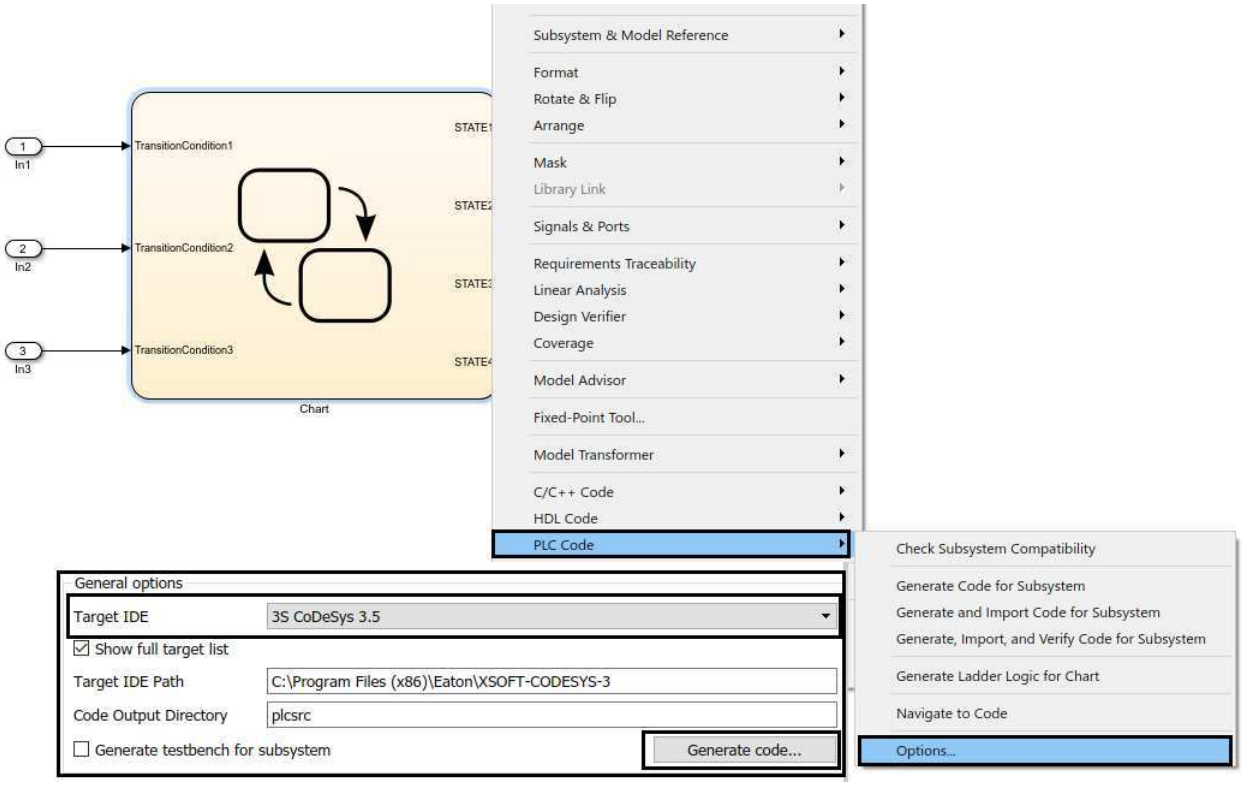


Figura 22 – Gerar código.

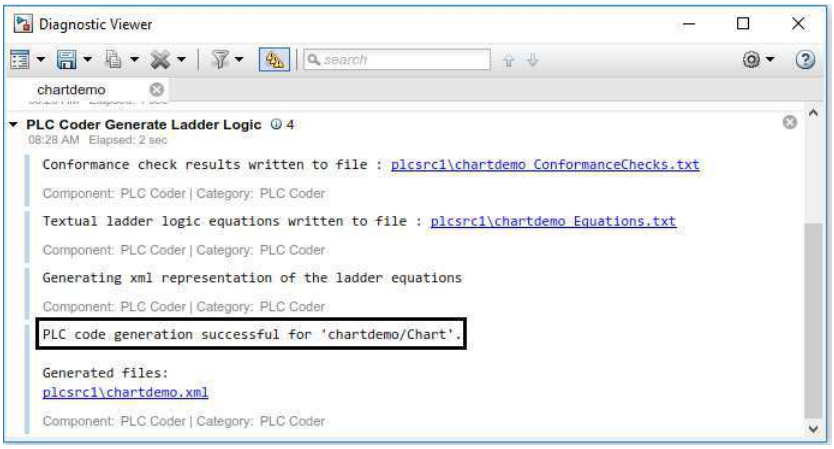


Figura 23 – Diagnóstico.

É possível visualizar o código gerado a partir da importação do arquivo correspondente para o ambiente de desenvolvimento escolhido. Analisando o código gerado no ambiente CODESYS, na Figura 24 tem-se a declaração das variáveis utilizadas, dividida em três partes, na primeira parte tem-se a declaração das variáveis de entrada onde é possível visualizar as condições de transição definidas pelo usuário, as variáveis de incialização definidas pela ferramenta e o método de solução. Na segunda parte da declaração de variáveis tem-se as variáveis de saídas definidas pelo usuário, correspondente aos estados. Por fim, na terceira parte da declaração de variáveis, observam-se as variáveis criadas pela ferramenta para organização e manipulação de dados no digrama Ladder.

Na Figura 25, está representado o código em linguagem Ladder gerado que, a partir de uma breve análise, percebe-se que descreve corretamente o diagrama de estados construído previamente. Por exemplo, para que a saída `STATE2` seja ativada, é necessário que os contatos correspondentes ao estado anterior `STATE1` e a condição de transição correspondente `TransitionCondition1` estejam ativados.

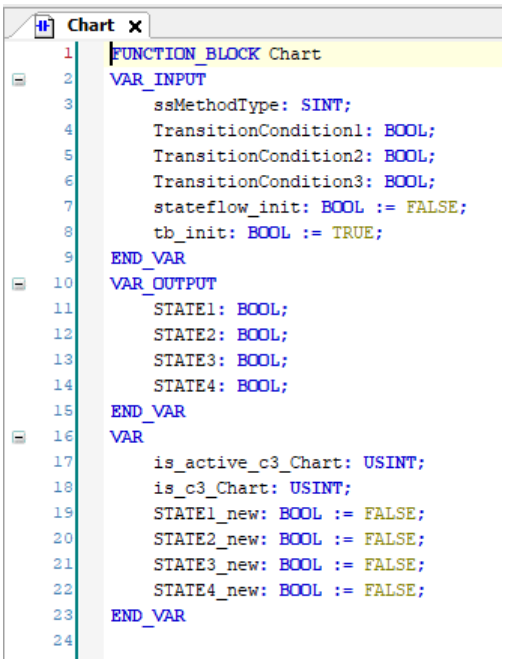


Figura 24 – Código gerado - declaração das variáveis.

3.5 Blocos do Simulink e do Stateflow suportados

Para visualizar uma lista de blocos do Simulink e do Stateflow que podem ser utilizados com a ferramenta *Simulink PLC Coder* na geração de código para CLPs, basta digitar o comando `plclib` na janela de comandos (*Command Window*) do MATLAB. O resultado desse comando está representado na Figura 26 e apresenta a lista de blocos disponíveis. O codificador é capaz de gerar código para os subsistemas que contém os blocos listados.

Mais detalhes sobre os procedimentos citados neste capítulo e mais funcionalidades da ferramenta *Simulink PLC Coder* podem ser encontrados no guia do usuário [MathWorks 2017].

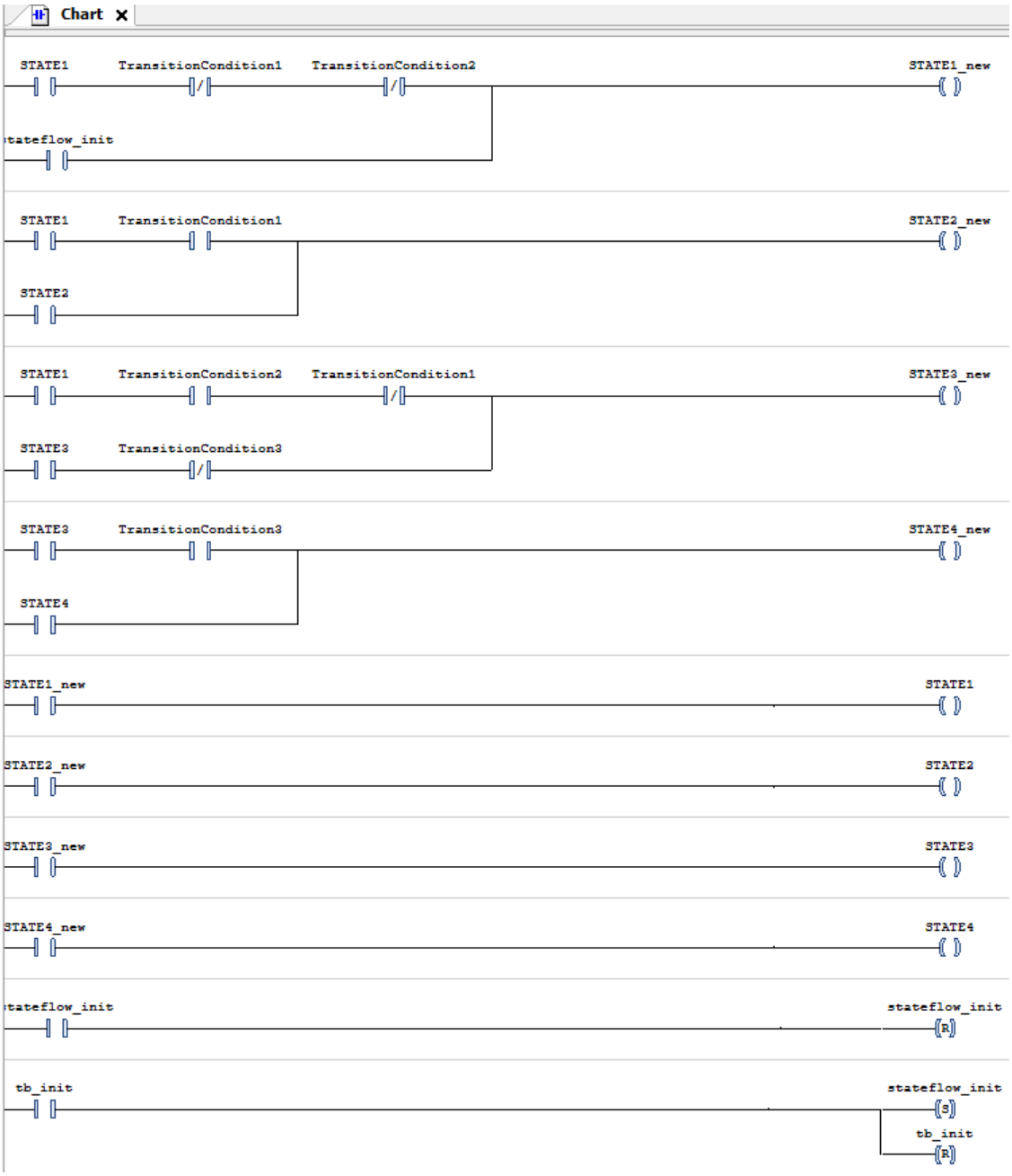


Figura 25 – Código gerado - Diagrama Ladder.

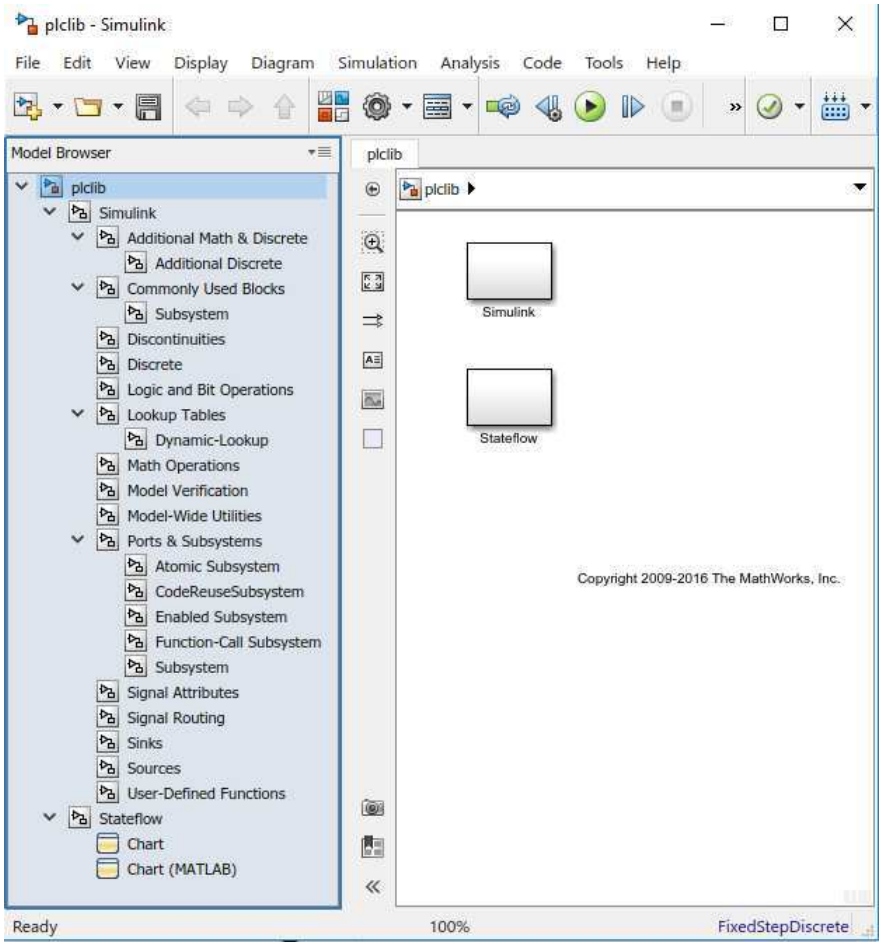


Figura 26 – Lista de blocos suportados.

Parte II

DELTAV: BLOCO MPC

4 Sistema Digital de Controle Distribuído

O principal objetivo deste capítulo é apresentar o bloco de controle preditivo fornecido pela *Emerson Process Management* em seu Sistema Digital de Controle Distribuído (SDCD), modelo *DeltaV*, e os procedimentos básicos de configuração. Será considerado que o leitor possui conhecimento dos aspectos teóricos do Controle Preditivo baseado em Modelo (MPC), em especial do Controle Preditivo por Matriz Dinâmica (DMC), e que possui familiaridade com o sistema *DeltaV* e sabe como operá-lo. Caso o leitor tenha dificuldades em acompanhar as próximas seções, um exemplo muito completo e didático de aplicação do *DeltaV* para automação de uma Planta Didática SMAR PD3, que simula um processo evaporador, foi apresentado em [MOREIRA 2016].

4.1 Apresentação

Nesta seção os aspectos básicos relacionados ao SDCD são introduzidos. Por definição, um SDCD é um sistema digital de controle em que os respectivos estão distribuídos pelo sistema formando subsistemas interconectados por uma rede para comunicação redundante de monitoramento, o que possibilita uma melhor organização entre as estações ao longo do processo com diversos níveis de hierarquia. A arquitetura desse sistema é composta por três entidades básicas. A primeira é a interface com o processo responsável por adquirir e processar os dados obtidos do processo, além de poder enviar sinais ou comandos aos atuadores e à outras interfaces. A segunda é a IHM, composta por várias estações dedicadas espalhadas na planta, exibindo apenas as informações necessárias para aquela área, além de uma sala de controle central para supervisão de todos os processos. Por fim, a terceira entidade é a rede de comunicação, que faz o papel de via de dados e faz a comunicação entre as estações que fazem tanto interação com o processo quanto com os operadores, integrando o processo como um todo [MOREIRA 2016].

Dentre as principais vantagens de se utilizar um Sistema Digital de Controle Distribuído pode-se citar a facilidade de se modificar/ampliar o sistema após a instalação original, a possibilidade de se utilizar estratégias de multitarefas e a menor necessidade de cabeamento comparado a um sistema centralizado. No entanto, algumas desvantagens também se fazem presentes como, por exemplo, problemas de compatibilidade entre dispositivos devido ao sistema ser proprietário, alto custo de implementação e a necessidade de mão-de-obra devidamente qualificada para operação [MOREIRA 2016].

Com relação às possibilidades de aplicação, os SDCDs são bastante utilizados em processos complexos que exigem um monitoramento mais rigoroso e onde se procura evitar a interrupção do funcionamento da planta à todo custo, tanto devido a questões financeiras quanto pela necessidade se manter a segurança do ambiente de trabalho. Alguns exemplos são as refinarias, indústrias automotivas, petroquímicas e as indústrias de manufatura de produtos brutos e caros no geral. Dentre as estratégias de controle disponíveis para SDCDs, pode-se mencionar o controle preditivo, lógica fuzzy e as redes neurais.

4.2 DeltaV

Dentre os Sistemas Digitais de Controle Distribuído disponíveis no mercado, uma opção muito confiável é o sistema *DeltaV*, da fabricante *Emerson Process Management*. Se tratando da sua configuração, o próprio fabricante disponibiliza ambientes exclusivos de configuração, programação, criação de telas de operação, supervisão dos cartões e aquisição de dados. Estes estão contidos no DeltaV User Station Software e são disponibilizados junto com o sistema. Os principais aplicativos presentes são:

- *DeltaV Operate* - permite a visualização, criação e edição de telas de operação para o sistema supervisório.
- *DeltaV Configurarion Studio* - é constituído por três programas:
  - *Explorer* - utilizado para visualizar e editar a configuração do sistema;
  - *Control Studio* - utilizado para a implementação de rotinas, alarmes e referenciamento de entradas e saídas aos cartões;
  - *Configuration Assistant* - ferramenta utilizada para auxiliar na montagem e configuração do sistema.
- *DeltaV Diagnostics* - monitora as redes e dispositivos conectados, seus estados e suas configurações.

4.3 Bloco MPC

Presumindo que o leitor tenha conhecimento e saiba operar o sistema *DeltaV*, após efetuar o *login* na máquina, o usuário deve selecionar o **Menu Iniciar** → **Programas** → **DeltaV Explorer**. Após criar/selecionar a AREA onde o MPC será implementado, o usuário deve clicar com o botão direito sobre a mesma e selecionar a opção **Open with Control Studio** para abrir o ambiente de edição. No **Control Studio**, o usuário deve selecionar a aba **Advanced Control** e arrastar o bloco MPC, como o da Figura 27, para o ambiente de edição. As entradas e saídas do bloco MPC são configuradas no ambiente

DeltaV Control Studio da maneira que melhor represente a estrutura de controle desejada e são brevemente descritas a seguir:

- CNTRL - Variável de controle (*Controlled Variable*);
- MNPLT - Variável manipulada (*Manipulated Variabe*);
- CNSTR - Restrição (*Constraint*);
- DSTRB - Perturbação (*Disturbance*).

Figura 27 – *DeltaV* - Bloco MPC.

O próximo passo é definir o número de variáveis de controle, perturbações e restrições. Para fazer isso, o usuário deve clicar com o botão direito do mouse sobre o bloco MPC, e selecionar a opção **Extensible Parameters**. Uma janela será exibida na qual consta o número de variáveis de controle, perturbações e restrições *default*, para alterá-los basta clicar sobre uma das variáveis e selecionar a opção **Modify**, uma segunda janela será aberta na qual o usuário entra com o número desejado. Todo esse procedimento está ilustrado na Figura 28, onde `NOF_CNSTR` significa número de restrições (*Number OF CoNSTRaints*), `NOF_CNTRL` significa número de parâmetros de controle (*Number OF CoNTRoL parameters*) e `NOF_DSTRB` significa número de parâmetros de perturbação (*Number OF DiSTRBance parameters*).

Após definir o número de entradas do bloco MPC, o usuário deve configurar os parâmetros relacionados a essas variáveis. Para fazer isso o usuário deve clicar com o botão direito do mouse sobre o bloco MPC, e selecionar a opção **Properties**. Uma janela como a da Figura 29 será aberta e nela é possível selecionar o parâmetro desejado e configurar suas propriedades. Por exemplo, no caso da variável controlada, pode-se ajustar os limites do *setpoint*, e no caso da variável manipulada, é possível definir os limites do sinal de saída.

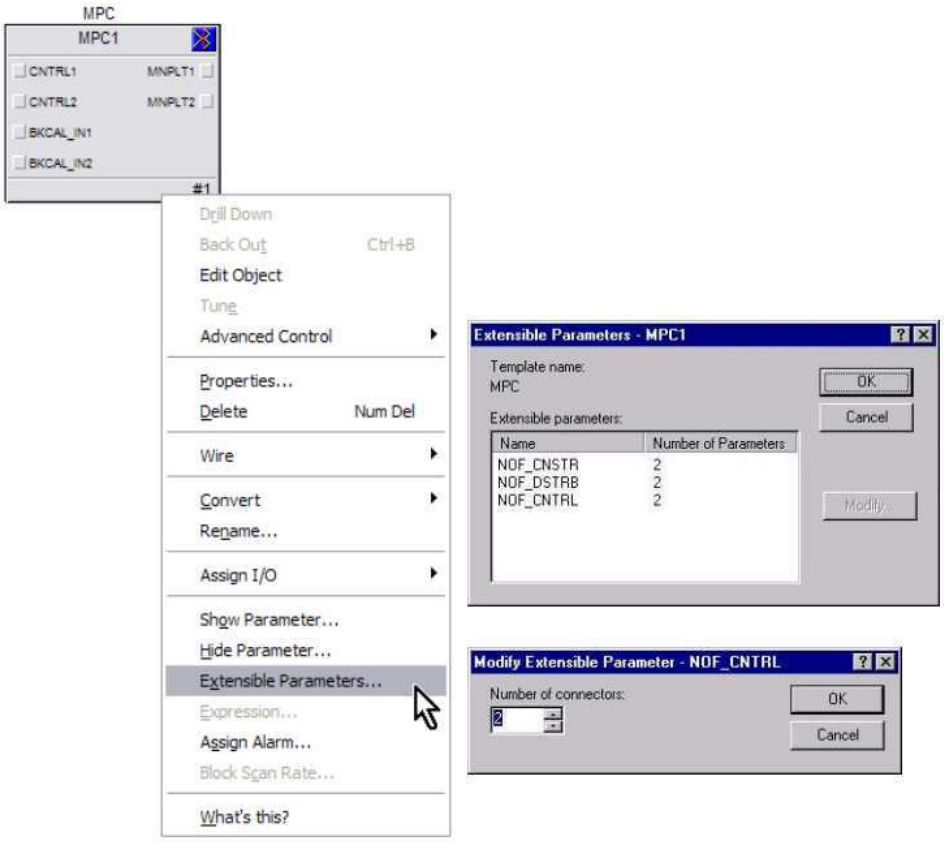


Figura 28 – Configuração do número de entradas e saídas.

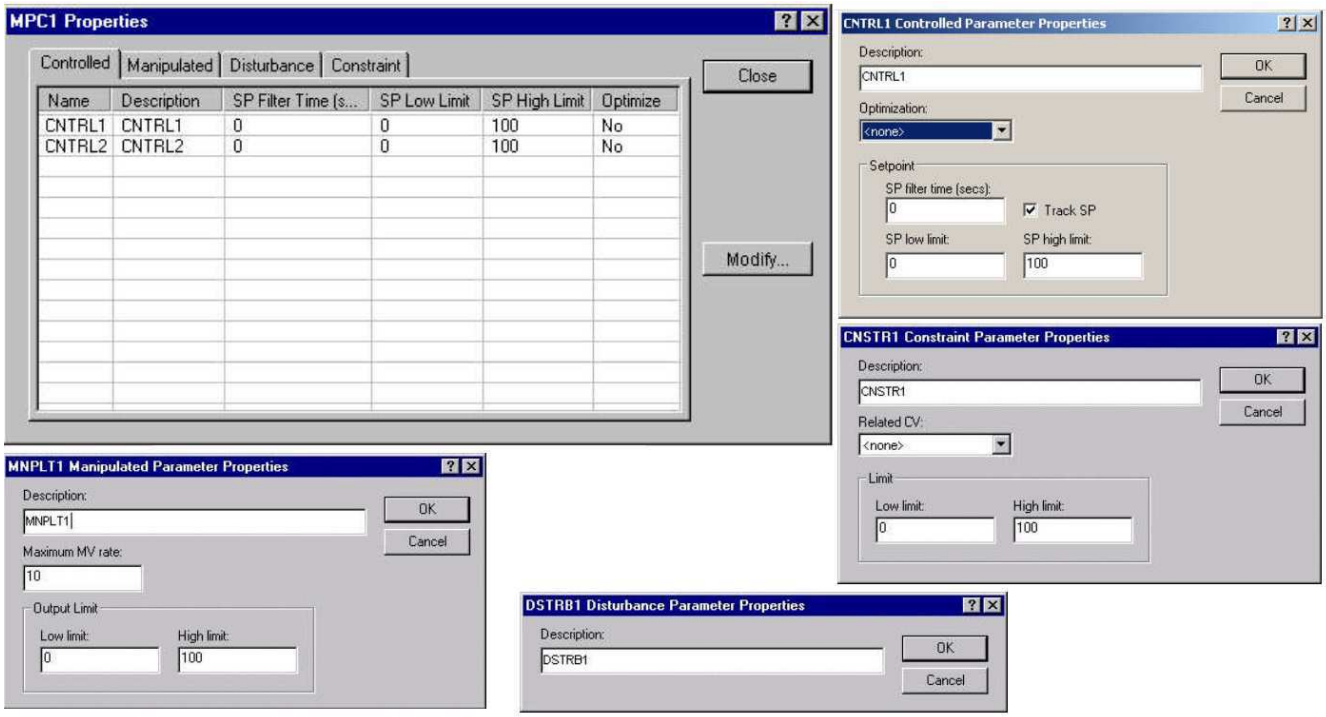


Figura 29 – Configuração dos parâmetros.

Em seguida, o usuário utiliza o bloco MPC configurado para construir a estratégia de controle desejada. Definida a estratégia de controle, o usuário só precisa conectar os blocos de Entrada/Saída às entradas e saídas do bloco MPC. Caso o usuário tenha criado um módulo separado para o controle MPC, o procedimento também é simples, basta selecionar a aba **IO** no ambiente **Control Studio** e arrastar os blocos de entrada (*Analog Input*) e saída (*Analog Output*) para a área de edição. Para configurá-los, deve-se clicar com o botão direito sobre o bloco entrada/saída, selecionar a opção **Assign I/O → To Signal Tag**, e atribuir a tag desejada. Na Figura 30 está representada um exemplo da utilização do bloco MPC para uma estratégia de controle de malha simples, e na Figura 31 um exemplo da utilização do bloco MPC para uma estratégia de controle em cascata. É uma boa prática quando se utiliza blocos de diferentes módulos, incluir no projeto um bloco de ajuste de escala `SCLR`. A entrada `BKCAL_IN` corresponde à entrada de realimentação do bloco MPC.

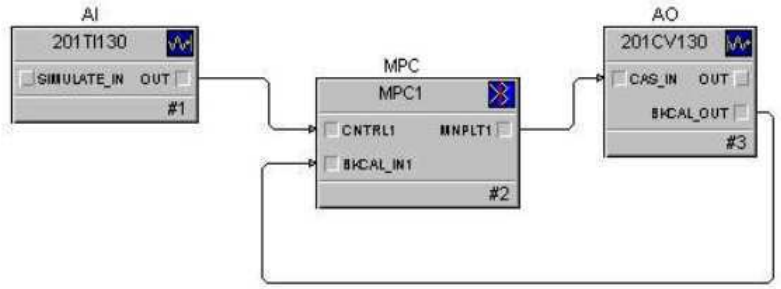


Figura 30 – MPC - Malha Simples.

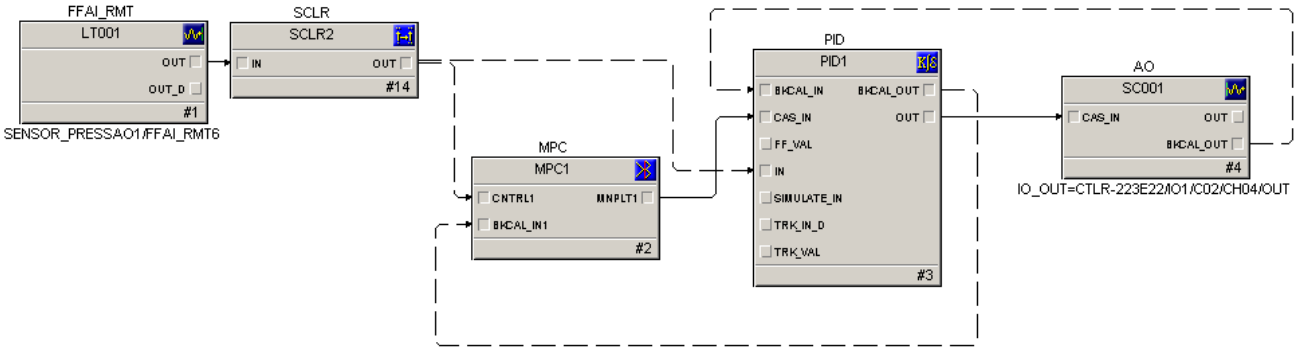


Figura 31 – MPC - Cascata.

Finalizada essa etapa de configuração, é hora de fazer a identificação do modelo que será utilizado na predição das saídas futuras e cálculo das ações de controle. Para isso, a ferramenta *DeltaV Predict* será utilizada.

4.3.1 *DeltaV Predict*

É utilizando a ferramenta *DeltaV Predict* que um modelo de resposta ao degrau é identificado, através da execução de um teste automatizado que coleta os dados do processo e os utiliza para criar a resposta ao degrau. A partir do modelo obtido, e efetuando-se a devida validação do mesmo, é possível gerar de maneira automatizada o controlador MPC. Caso o modelo do processo tenha sido corretamente identificado, a configuração padrão do controlador gerado fornece desempenho ótimo. Caso contrário, ajustes podem ser efetuados. Adicionalmente, a ferramenta *DeltaV Predict* fornece um ambiente de simulação que permite testar o controlador gerado quantas vezes forem necessárias e à uma velocidade muito maior do que seria possível na realidade. Após conseguir resultados satisfatórios, o usuário pode testar o controlar no processo real.

Para abrir o *DeltaV Predict*, o usuário deve clicar com o botão direito do mouse sobre o bloco MPC e selecionar **Advanced Control** → **Predict** como ilustrado na Figura 32. Uma janela como a da Figura 34 será aberta. Na seção *Test Process* definem-se os parâmetros de teste, o tamanho do degrau a ser aplicado na variável manipulada, o tempo que o processo leva para atingir o regime permanente e o número de ciclos de teste. Antes de clicar em testar é necessário assegurar que o sistema está operando em regime permanente e que o tempo de regime (T_{ss}) configurado é suficiente para cobrir toda a resposta do processo, uma boa estimativa é utilizar a equação:

$$T_{ss} = TempoMorto + 4 * ConstanteTempoProcesso \tag{4.1}$$

Com relação ao número de ciclos de teste utilizado, é recomendado a utilização de dois ciclos ao invés de apenas um. O tempo de duração do teste vai depender do tempo de regime escolhido e do número de ciclos de teste, caso se escolha apenas um ciclo, o tempo de duração do teste é de $7 * T_{ss}$ segundos, e caso dois ciclos sejam utilizados, o tempo de duração do teste é de $14 * T_{ss}$ segundos. Logo, para processos de dinâmica muito lenta, os testes tendem a demorar bastante. Em seguida, pode-se iniciar o teste clicando em **Test**.

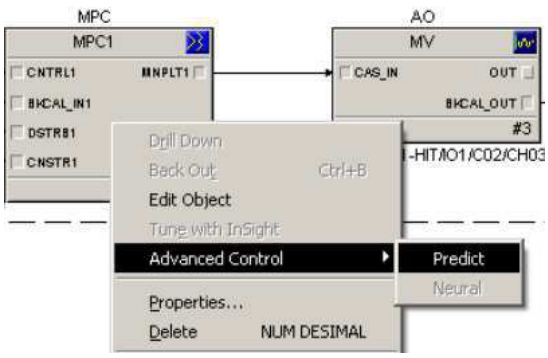


Figura 32 – MPC - Abrir o *DeltaV Predict*.

Finalizado o teste, o usuário pode gerar um modelo clicando em **Autogenerate**. O modelo gerado fica disponível na parte superior esquerda da janela do *DeltaV Predict*, onde está destacado em vermelho na Figura 34. Clicando duas vezes nesta opção, uma janela parecida com a da Figura 34 será aberta, na qual é possível visualizar o modelo de resposta ao degrau identificado.

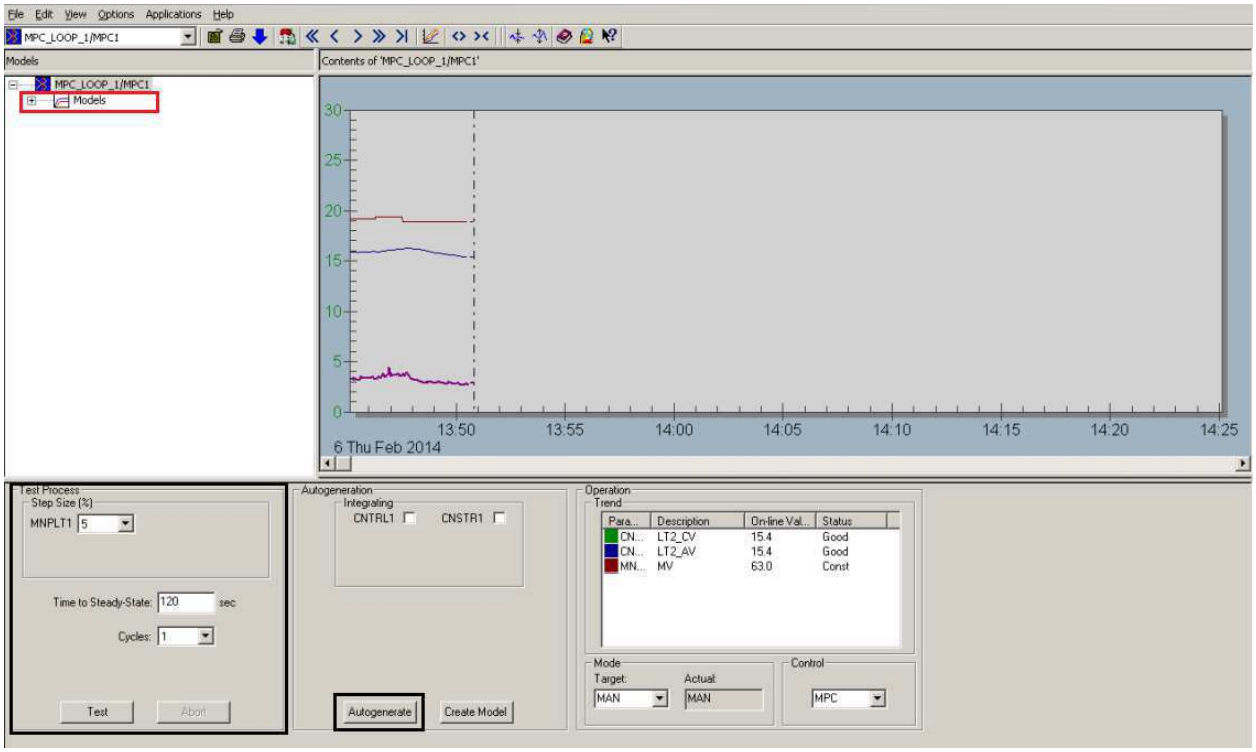


Figura 33 – *DeltaV Predict* - Parâmetros.

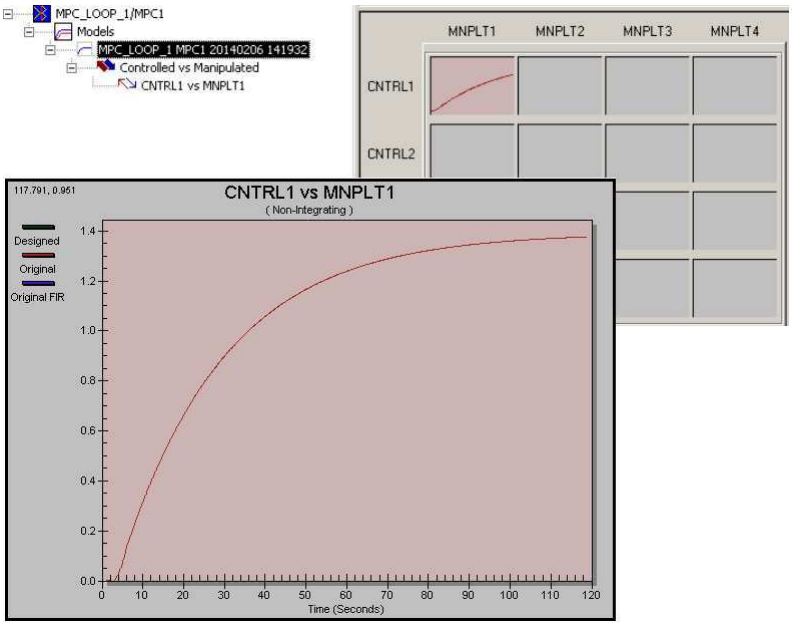


Figura 34 – *DeltaV Predict* - Modelo identificado.

Para validar o modelo identificado, o usuário pode clicar em **Options** → **Expert** no menu superior do *DeltaV Predict*. Em seguida, clicando com o botão direito sobre o modelo identificado no menu lateral do *DeltaV Predict* e selecioando a opção **Verify against original data**, o usuário é apresentado a uma janela como a da Figura 35, na qual tem-se a comparação entre os valores das amostras originais e os valores preditos com a utilização do modelo. Além disso, a ferramenta fornece dois índices de desempenho que podem auxiliar na análise da qualidade do modelo, o erro quadrático médio (MSE - *Mean Square Error* ou MSD - *Mean Square Deviation*) e o coeficiente de determinação (R^2) [GAIÃO 2018] .

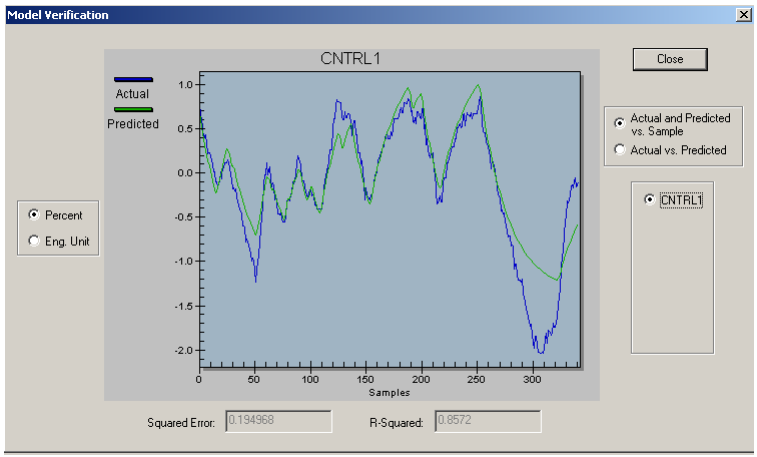


Figura 35 – *DeltaV Predict* - Verificação do Modelo.

Caso o modelo esteja satisfatório, o usuário pode gerar o controlador clicando com o botão direito sobre o modelo identificado no menu lateral do *DeltaV Predict* e selecionado a opção **Generate controller**. Após isso, uma janela como a da Figura 36 será exibida, onde o usuário definirá os parâmetros da matriz de pesos sobre os erros preditos e da matriz de pesos sobre as variações da ação de controle. Feito isso, o controlador é gerado.

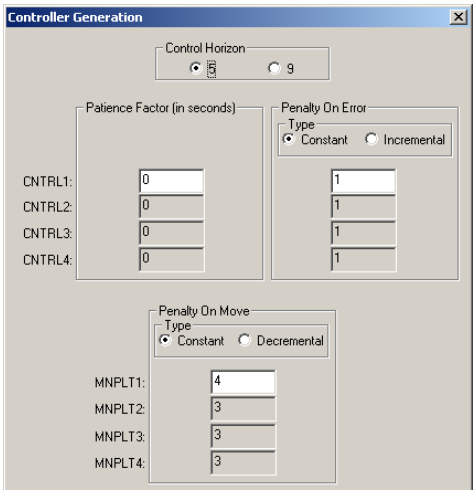


Figura 36 – *DeltaV Predict* - Matrizes de penalidades.

Por fim, o usuário pode utilizar a aplicação *MPC Simulate* para validar o controlador gerado, ajustando os valores da variável controlada e observando o comportamento do processo e, quando satisfeito com os resultados, aplicar o controlador ao processo real e ajustá-lo através do *MPC Operate*, que possui uma tela de operação similar à representada na Figura 37. Como aplicação dos procedimentos mencionados neste capítulo, tem-se o Trabalho de Conclusão de Curso entitulado *Controle Preditivo aplicado a uma Planta Didática* [GAIÃO 2018].

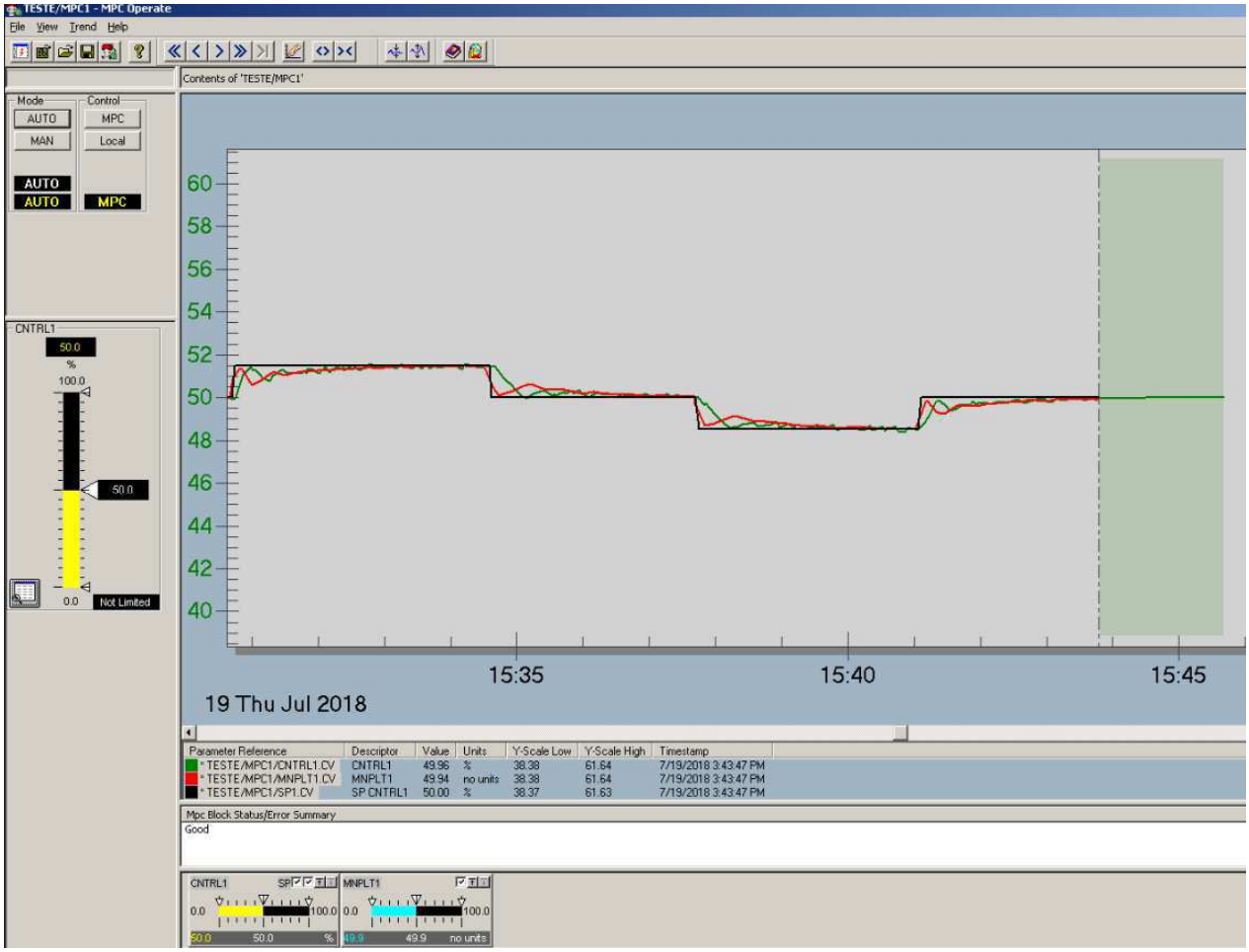


Figura 37 – *MPC Operate*.

5 Considerações Finais

Na primeira parte deste trabalho foi apresentado a ferramenta *Simulink PLC Coder* e foi feito o estudo e devida documentação dos procedimentos e configurações necessárias para geração de código para controladores lógico programáveis (CLP). Foram vistos pequenos exemplos de geração de código baseado em texto, utilizando o Texto Estruturado (*ST - Structured Text*), e geração de código baseado em representações gráficas, utilizando a linguagem de programação *Ladder*. A segunda categoria de atividades compreendeu a documentação dos procedimentos e configurações necessárias à utilização do bloco de Controle Preditivo por Modelo fornecido pelo Sistema Digital de Controle Distribuído (SDCD), modelo *DeltaV*, da fabricante *Emerson Process Management*. Todo o processo, desde a configuração do bloco MPC, a identificação e validação do modelo, e à geração do controlador foi descrito e exemplificado. Por fim, conclui-se que as atividades desempenhadas e o trabalho empregado durante este estágio foram de grande valia e forneceram uma ótima oportunidade de preparo para o que será encontrado no âmbito profissional.

Referências

BRYAN L. A.; BRYAN, E. A. *Programmable Controllers: Theory and Implementation.* 2. ed.. ed. [S.l.]: Industrial Text. Citado na página 17.

GAIÃO, P. B. T. Trabalho de Conclusão de Curso, *Controle Preditivo Aplicado a uma Planta Didática.* 2018. Citado 2 vezes nas páginas 48 e 49.

HANSSEN, D. H. *Programmable Logic Controllers.* 1. ed.. ed. [S.l.]: Wiley. Citado na página 17.

MATHWORKS. *Simulink PLC Coder: User's Guide.* [S.l.], 2017. Disponível em:<https://www.mathworks.com/products/sl-plc-coder.html>. Acesso em 20 jul. 2018. Citado na página 37.

MOREIRA, L. J. S. Relatório de Estágio, *Relatório de Estágio Supervisionado.* 2016. Citado na página 41.