UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Christian Diefenbach

VTBSim – Programa para Simulação de Sistemas Dinâmicos

Uma Ferramenta para Ensino de Controle

Porto Alegre
2004

Christian Diefenbach

VTBSim – PROGRAMA PARA SIMULAÇÃO DE SISTEMAS DINÂMICOS

Uma ferramenta para ensino de controle

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica (PPGEE), da Universidade Federal do Rio Grande do Sul (UFRGS), como parte dos requisitos para a obtenção do título de Mestre em Engenharia Elétrica.

Área de concentração: Automação e Instrumentação Eletro-Eletrônica

ORIENTADOR: Dr. João Manoel Gomes da Silva Jr.

Porto Alegre
2004

Christian Diefenbach

VTBSim – PROGRAMA PARA SIMULAÇÃO DE SISTEMAS DINÂMICOS

Uma ferramenta para ensino de controle

Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia Elétrica, e aprovada em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____
Prof. Dr. João Manoel Gomes da Silva, Jr., UFRGS
Doutor Universite de Toulouse III (Paul Sabatier) (1997)

Banca Examinadora:

Prof. Dr. Carlos Eduardo Pereira - UFRGS / Eng. Elétrica
Doutor TECHNISCHE UNIVERSITAT STUTTGART (1995)

Prof. Dr. Jorge Trierweiler - UFRGS / Eng. Química
Doutor pela Universität Dortmund (1996)

Prof. Dr. Eugenio Castelan - UFSC / Depto Automação e Controle
Doutor pela Universite de Toulouse III (Paul Sabatier) (1992)

Prof. Dr. Walter Fetter Lages - UFRGS / Eng. Elétrica
Doutor em Eng. Eletrônica pelo Instituto Tecnológico de Aeronáutica (1998)

Coordenador do PPGEE: _____
Prof. Dr. Carlos Eduardo Pereira - UFRGS/Eng. Elétrica
Doutor pela Technische Universitat Stuttgart (1995)

Porto Alegre, Dezembro de 2004.

DEDICATÓRIA

Dedico este trabalho aos meus pais e à minha esposa, em especial pela dedicação e apoio em todos os momentos.

AGRADECIMENTOS

Ao Programa de Pós-Graduação em Engenharia Elétrica, PPGEE, pela oportunidade de realização deste trabalho, em minha área de pesquisa.

Aos colegas da Activision, em especial Wade Brainerd, pelo inestimável auxílio na implementação da interface gráfica do usuário.

Ao Prof. Dr. João Manoel Gomes da Silva, Jr., pela paciência e perseverança de orientar um aluno à distância.

RESUMO

No ambiente universitário, um grande esforço é feito para disponibilizar ferramentas que facilitem aos graduandos, desde o início do curso, a fazer uma correlação entre conteúdos matemáticos abstratos e sua aplicação em uma realidade física, objetivando não apenas a melhor compreensão e fixação de conceitos físicos e matemáticos, mas estimulando o interesse do aluno a estes conceitos. Este trabalho propõe uma ferramenta de simulação de topologias variáveis, que possibilita uma realimentação visual de um sistema descrito através dos dispositivos físicos que o constituem, permitindo ao aluno a interatividade com processo simulado, e uma mais fácil associação com a dinâmica do sistema estudado, pontos-chave na utilização de experimentos para fixação e compreensão de conceitos de controle.

Esta ferramenta implementará um algoritmo de simulação em blocos, onde cada bloco é processado individualmente em termos de suas entradas e saídas, proporcionando uma relação direta entre cada aspecto do experimento físico e cada bloco funcional utilizado na simulação. Como os blocos são simulados separadamente, existe também uma relação direta entre as grandezas mensuráveis no experimento físico e as grandezas utilizadas na interligação entre os blocos na simulação.

Palavras-chaves: Engenharia Elétrica. Simulação. Aprendizado de Controle. Ferramentas de Ensino.

ABSTRACT

In the university environment, a great effort is done to make available tools that allow undergraduate students, from the beginning of the course, to correlate abstract mathematical concepts and their application in a physical reality, aiming not only at a better comprehension and fixation of physical and mathematical concepts, but also stimulating the student interest in this concepts. This work proposes a variable topology simulation tool, which allows a visual feedback of a system described using physical devices, enabling the student to interact with the simulated system, key points in the utilization of experiments for the comprehension and fixation of control concepts.

This tool will implement an block simulation algorithm , where each block is processed individually, in terms of its inputs and outputs, providing a direct relantionship between between each aspets of the physical experiment and each functional block used in the simulation. Since the blocks are simulated separately, there is also a direct relantionship between the measurable values on the physical experiment and the values used on the interconnection between blocks on the simulation.

Keywords: Electrical Engineering. Simulation. Control Learning. Teaching Tools.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

CSV: Comma Separated Values (Valores Separados por Vírgula)

GPL: GNU General Public License

PPGEE: Programa de Pós-Graduação em Engenharia Elétrica

XML: EXtensible Markup Language (Linguagem de Marcação Extensível)

1 INTRODUÇÃO

No ensino de sistemas de controle, uma das principais dificuldades se encontra na transição entre os modelos matemáticos e a compreensão dos dispositivos físicos que estão sendo modelados. O estudo de modelos matemáticos auxilia na compreensão das ferramentas que permitem o cálculo de dispositivos físicos, mas são necessárias outras técnicas para transmitir a idéia do que estes modelos matemáticos representam em termos de dispositivos físicos.

A partir da última década do século passado, muitos sistemas foram propostos e utilizados com sucesso no ensino de controle. Mesmo com a atual facilidade de acesso a componentes eletrônicos, microprocessadores e computadores, ainda são necessários meios de relacionar os conteúdos estudados em sala de aula, e a aplicação destes conceitos à realidade física, ou seja, são necessárias soluções para diminuir a distância entre a teoria e a prática. Uma das alternativas é a implementação de experimentos virtuais dedicados, localmente em um computador (Scopinho, 2002; Dormido, 2003), ou através de programas comercias de simulação (Coelho, 2001). Com o avanço da internet e das tecnologias de comunicação, outra solução é a disponibilização, para um grande número de alunos, de uma única bancada de teste, monitorada e controlada à distância (Miele, 2001; Casini, 2001; Jochheim, 1999; Ko, 2001). Outras abordagens dispensam até mesmo a implementação física do experimento, utilizando ferramentas de simulação como MatLab e SimuLink e disponibilizando uma interface de acesso remoto via World Wide Web (Granado, 2003; Tilbury, 1999; Exel, 1999; Schmid, 1998).

O sistema proposto implementa uma interface gráfica modular para simulação de sistemas físicos em blocos, focando especialmente em experimentos de laboratório utilizados regularmente no ensino de controle. Ao invés de proporcionar uma ferramenta para descrição

do sistema a partir de modelos matemáticos, os blocos representam diretamente os dispositivos físicos que são parte do sistema, abstraindo o modelo matemático da interface com o aluno.

Assim sendo, a ferramenta proposta serve como o elemento de ligação entre o modelo matemático e o experimento físico. Cada bloco funcional do experimento físico é representado por um bloco autonomo na ferramenta, e cada bloco é processado separadamente pelo algoritmo de simulação. Desta forma é possível traçar uma relação direta entre o aspecto real e o aspecto virtual do experimento. Como cada um dos blocos é simulado separadamente, sempre existirá também uma relação direta entre as grandezas observáveis no experimento físico e as grandezas utilizadas na ligação entre os blocos.

O objetivo é proporcionar uma solução cuja utilização seja simples, segura, que aproxime o modelo matemático do dispositivo físico, e que tenha um aproveitamento eficiente do tempo disponível ao aluno, mas que seja também suficientemente complexa para apresentar um desafio adequado para a correta compreensão dos conceitos estudados, incluindo, opcionalmente, o impacto de considerações práticas como saturação de sensores e controladores, limitação de atuadores, ruídos e não-linearidades.

A interface gráfica tem funcionalidades para iniciar e parar a simulação, retornar a simulação ao início e avançar a simulação até um determinado tempo definido pelo usuário. Também serão implementados recursos para gravação de arquivos padronizados (valores separados por vírgula, .CSV) que poderão ser importados em qualquer outro programa para posterior análise.

O esquemático é definido totalmente através da interface gráfica. Não haverá necessidade de definir modelos matemáticos dos dispositivos físicos; apenas, opcionalmente, seus parâmetros.

O presente trabalho está organizado como segue:

O capítulo 2 trata da problemática do ensino de controle e o estado da arte. As abordagens mais comuns são discutidas brevemente. Alguns exemplos de ferramentas existentes são listados com suas características básicas; a motivação para este trabalho é apresentada e os objetivos são traçados. Uma breve descrição da ferramenta a ser desenvolvida é também apresentada.

O capítulo 3 descreve, em maiores detalhes, como esta ferramenta foi implementada. São apresentados os conceitos básicos, as estruturas de dados, os algoritmos utilizados e o processo de simulação passo a passo.

O capítulo 4 complementa o capítulo 3, focando nas funções necessárias para criar novos blocos, e as considerações necessárias durante a programação, para uma correta comunicação entre o bloco e o algoritmo de simulação.

O capítulo 5 descreve, em maiores detalhes, os recursos implementados na estrutura básica da ferramenta, e como estes são utilizados,

O capítulo 6 faz o estudo dos blocos fornecidos com a ferramenta e sua implementação. Cada um dos blocos é analisado separadamente em função de suas entradas e saídas, funções, equações e algoritmos utilizados.

O capítulo 7 fornece exemplos de utilização do sistema. São feitas comparações com resultados analíticos, assim como são observadas as conseqüências das não-linearidades, que podem ser inseridas no sistema simulado.

O capítulo 8 apresenta as conclusões e perspectivas de trabalhos futuros, e o capítulo 9 contém as referências bibliográficas.

2 Definição da Problemática e Estado da Arte

2.1 Definição da Problemática

O ensino dos conceitos matemáticos de sistema de controle e sua aplicação física necessitam uma parte prática, que complemente a teoria e coloque à prova os conceitos estudados, ajudando o aluno a fixar os conhecimentos e aplicá-los no dia-a-dia.

Esta função pode ser desempenhada pelas aulas práticas, com acesso a bancadas, componentes e instrumentos, utilizando experimentos com o objetivo de aplicar um determinado grupo de conceitos e conhecimentos, bem como criatividade, na solução dos problemas propostos. Embora esta parte prática seja um aspecto indispensável ao estudo de engenharia elétrica, é possível utilizar outros métodos que, em muitas circunstâncias, substituem com vantagens as aulas de laboratório.

Laboratórios práticos possuem um custo elevado para aquisição e manutenção, ocupam muito espaço e requerem muito tempo para a preparação, experimentação e, dependendo do caso, coleta de resultados. O uso de pessoal especializado, cuja responsabilidade é manter e organizar os laboratórios, também representa uma significativa parcela do orçamento das instituições de ensino. A conseqüência disto é a limitação da quantidade de aulas e do tempo de utilização, o uso de uma bancada por mais de um aluno, bem como a restrição do uso do laboratório fora do contexto das aulas práticas.

Outra consideração importante é a segurança. Experimentos que lidam com partes móveis, temperaturas elevadas e, de forma geral, eletricidade, são uma causa potencial de acidentes, principalmente quando estão aos cuidados de alunos ainda inexperientes quanto à forma correta e segura de lidar com estes equipamentos.

A utilização de sistemas reais, controlados remotamente, resolve em parte o problema de espaço e preparação de experimentos, mas ainda possui custo elevado e não podem ser controlados por mais de um aluno simultaneamente, enquanto outros alunos podem apenas acompanhar o experimento (Jochheim, 1999; Casini, 2001).

Outra solução é a utilização de laboratórios virtuais controlados remotamente. Neste caso, duas abordagens são utilizadas: a primeira utiliza programas dedicados, que imitam a interface utilizada em programas comercias, para minimizar o tempo de aprendizado para utilização do programa. A segunda opção utiliza programas comercias para a simulação dos experimentos, mas oferece uma interface remota simplificada, que dispensa o aluno da formalidade matemática e do aprendizado formal da linguagem utilizada por estes programas. Esta técnica possui a vantagem de poder ser utilizada por vários alunos simultaneamente, cada um interagindo com sua própria versão do experimento (Granado, 2003; Tilbury, 1999; Exel, 1999; Schmid, 1998).

A alternativa mais comum implementa um laboratório virtual localmente, em um computador. Desta forma, o experimento possui as seguintes vantagens:

a) Mobilidade: O aluno pode utilizar o experimento onde quer que haja acesso a um computador.
b) Disponibilidade: O uso não está sujeito às horas estipuladas pelo laboratório.
c) Flexibilidade: O experimento pode ser configurado independentemente da disponibilidade de recursos físicos.
d) Eficiência: Experimentos podem ter seu tempo acelerado ou atrasado, podem ser alterados, repetidos, analisados e comparados em uma fração do tempo necessário a um experimento real.

e) Segurança: O uso seguro de dispositivos físicos, como motores e geradores, requer técnicas e normas de segurança nem sempre do conhecimento dos alunos, além do custo extra da supervisão que, às vezes, se faz necessária. O uso destes mesmos dispositivos virtuais não apresenta o mesmo problema.

Todos os métodos utilizados em laboratórios didáticos visam objetivos semelhantes, ou seja, dar ao aluno a oportunidade de utilizar seus conhecimentos teóricos em experiências que aproximam o modelamento matemático da realidade física, fornecendo os meios de fixar e validar os conceitos estudados em aula. Um outro aspecto menos abordado, é de que o experimento não apenas testa o conhecimento, ele também dá ao aluno a certeza e a segurança de ter compreendido os conceitos utilizados no modelamento dos dispositivos físicos.

O custo, neste caso, depende unicamente da utilização de programas comerciais para a simulação do experimento.

2.2 Análise do Estado da Arte

Existem várias alternativas de sistemas para simulação de sistemas dinâmicos, que podem ser escolhidos dependendo da aplicação, características e custo do sistema. A lista apresentada a seguir não pretende analisar todas as alternativas existentes, apenas citar as características gerais disponíveis em termos de programas de simulação de sistemas dinâmicos.

Scilab

Programa para simulação de esquemáticos baseado em blocos hierárquicos. O esquemático é definido através destes blocos, que embora tenham a capacidade de modelar dispositivos físicos, são gerados a partir de modelos matemáticos. Inclui um pacote de aplicativos para cálculo numérico, disponibilizando um ambiente aberto para simulação de

sistemas científicos e de engenharia. Pode ser estendido utilizando C++ ou Fortran, possui um interpretador e uma linguagem de programação de alto nível. É distribuído gratuitamente com uma licença semelhante a GNU General Public License (GPL) (Scilab, 2004).

Octave: Linguagem interativa de alto-nível, compatível em grande parte com o MatLab, voltada principalmente a aplicações de cálculo numérico. Possui uma interface de linha de comando para solução numérica de problemas lineares e não-lineares, e executar outras operações matemáticas em uma linguagem semelhante ao MatLab. Possui, também, a capacidade de plotar gráficos 2D e 3D, mas sua interface é limitada ao modo equivalente à linha de comando encontrado no Matlab, pois não possui os meios de editar um modelo ou um sistema utilizando uma interface gráfica. É distribuído gratuitamente com uma licença GPL (Octave, 2004).

ACSL

É um grupo de programas que combinam a linguagem ACSL, um conjunto de bibliotecas, sistema para plotagem de gráficos, uma interface gráfica, uma linguagem para macros e um compilador de esquemáticos. É possível utilizar a interface gráfica para definir modelos e esquemáticos. É distribuído comercialmente pela AEgis Techlologies (Aegis, 2004).

Modelica

É uma linguagem de modelagem orientada a objeto, projetada para permitir, de forma conveniente, a modelagem orientada a componentes de sistemas físicos complexos, isto é, sistemas que contenham subcomponentes elétricos, eletrônicos, mecânicos, hidráulicos, térmicos e sistemas de controle (Modelica, 2004). Sendo uma linguagem para definição de sistemas complexos, Modelica não define uma interface gráfica, embora existam sistemas comerciais como o Dymola (Dynasim, 2004) e MathModelica (MathCore, 2004), que

fornecem uma interface gráfica para definição de sistemas em Modelica. Utilizando estas ferramentas é possível definir um esquemático e suas conexões através da interface gráfica, utilizando blocos definidos pelas bibliotecas disponíveis, como a biblioteca hidráulica, corpos rígidos e transmissão automotiva. Outros projetos, como o Research center for Object Oriented Modeling and Simulation e The Open Source Modelica Project, visam desenvolver um tradutor de Modelica para C/C++.

Ptolemy

É um programa desenvolvido em Java, cujo princípio é fornecer uma linguagem e um sistema de simulação usando modelos computacionais, que governem a interação entre diferentes componentes. O projeto visa estudar modelagem, simulação e projeto de sistemas de tempo real. Disponibiliza, também, uma interface gráfica em blocos chamada Vergil, que possibilita a definição e simulação de esquemáticos (Ptolemy, 2004).

Berkeley Madonna

Possivelmente o sistema mais rápido e flexível para solução de sistemas de equações diferenciais disponível atualmente. Foi desenvolvido na universidade de Berkeley, Califórnia, e é utilizado no desenvolvimento de modelos matemáticos voltados à pesquisa e ao ensino. Possui interfaces gráficas para modificação de parâmetros, edição de gráficos de fluxo de sinais e reações químicas, mas não para esquemáticos. É distribuído comercialmente pela Kagi Software (Madonna, 2004).

2.3 Motivação para o Trabalho

2.3.1 Objetivos

Como visto na seção precedente, existem várias alternativas de ferramentas para simulação de sistemas dinâmicos. A proposta deste trabalho, no entanto, é fornecer uma ferramenta didática complementar, com as seguintes características:

Fornecer uma ligação direta entre a definição do sistema a nível de blocos e os dispositivos físicos envolvidos na construção das malhas de controle.

a) Ser de fácil instalação e operação.
b) Ter a capacidade de gerar esquemáticos rapidamente, a partir de dispositivos e sistemas físicos pré-definidos.
c) Rodar, eficientemente, em sistemas com recursos limitados.
d) Ser compatível com os principais sistemas operacionais.
e) Oferecer recursos que facilitem a utilização no ensino de controle.
f) Ser distribuído gratuitamente.

O foco será dado no aspecto didático da utilização de sistemas de simulação, voltado ao ensino de controle. Ao invés de fornecer ao aluno um sistema que auxilie na busca da solução, este programa se propõe a apresentar os diferentes tipos de problemas que serão encontrados em sistemas físicos reais, assim como suas características e conseqüências.

Desta forma, o aluno terá uma ferramenta que pode ser utilizada tanto dentro ou fora de aula, que fornece os meios para testar e visualizar de forma rápida e eficiente os diferentes aspectos dos sistemas de controle e, desta forma, melhor entender e fixar os conceitos estudados em aula.

2.3.2 Descrição da Ferramenta a ser Desenvolvida

A ferramenta implementará um simulador em blocos, não-hierárquico, de tempo contínuo, baseado no modelo de fluxo de sinais (Liu, 1998)

A ferramenta deverá fornecer uma interface gráfica com o usuário, que permita a descrição do esquemático e o controle da simulação. Isto inclui posicionamento dos blocos, configuração dos parâmetros dos blocos, definição das interconexões entre os blocos, controles para iniciar, parar, avançar e reiniciar uma simulação. Também serão implementadas funcionalidades que facilitam o uso deste programa como ferramenta de ensino, tais como a inclusão de anotações direto no esquemático, definição do nome do esquemático e do nome do autor, e a possibilidade de exportar os dados simulados para outros programas de visualização e análise.

Enquanto alguns programas focam no aspecto da modelagem matemática de sistemas (e.g. Simulink) e outros na simulação voltada ao projeto de sistemas (e.g. Modelica), o programa proposto deverá oferecer uma conexão mais direta entre o modelo matemático e os dispositivos físicos utilizados na simulação. Ou seja, deverá fornecer blocos que representem diretamente os dispositivos físicos a serem simulados. Esta ferramenta também deverá fornecer os meios para realimentar ao aluno, em tempo real, a respeito da dinâmica e do estado do sistema.

A arquitetura deverá ser aberta, utilizando uma plataforma que possa ser compilada em diferentes sistemas operacionais, assim como em diferentes linguagens, sem necessidade de modificar o código fonte da ferramenta.

Os dispositivos implementados através dos blocos poderão simular comportamentos lineares ou não-lineares, dependendo do tipo de bloco. Os blocos podem incluir, por exemplo, ruído, saturação, histerese e zona morta. Também serão implementados blocos para simulação

de dispositivos físicos, como motores, bombas, tanques e fornos. Para permitir a modelagem de um circuito de controle completo, serão implementados medidores de parâmetros físicos, como tacômetros, medidores de pressão e termômetros.

De forma a complementar o sistema de controle, serão utilizados amplificadores de erro, funções de transferência, PIDs e outras funções matemáticas, assim como geradores de sinal configuráveis.

A visualização do estado do sistema será feita através de blocos desenvolvidos para isto. Por exemplo, o Bloco Multímetro mostrará continuamente a tensão em cada uma de suas duas entradas. O Bloco Osciloscópio implementará um osciloscópio simplificado de quatro canais. Alguns blocos poderão fornecer uma realimentação visual de seu estado atual; o Bloco Tanque, por exemplo, mostrará seu nível diretamente no esquemático.

Outras funcionalidades incluirão copiar, colar, excluir, desfazer, refazer, além gravar e ler esquemáticos em um formato multiplataforma e multilíngue.

Dependendo do tipo, blocos poderão implementar equações diferenciais como forma de descrição do processo simulado. Neste caso as equações serão resolvidas utilizando o método de Runge-Kutta de quarta ordem. (Numerical Recipes, 1992)

A partir de uma definição válida dos blocos, a arquitetura da ferramenta deverá fornecer os meios para transferir sinais da saída de um bloco para a entrada de outro(s), baseado nas conexões do esquemático, assim como gerenciar o algoritmo de simulação passo a passo, de forma que a implementação dos blocos não tenha que ter conhecimento da topologia do esquemático.

A arquitetura modular da ferramenta permite que usuários, com conhecimentos de programação em C++ e conhecimentos básicos da biblioteca wxWidgets, possam implementar novos blocos sem alterar a estrutura do programa. O processo de criação de

novos blocos é simples e objetivo, e a forma mais simples para a definição de novos blocos é copiar um dos blocos fornecidos com o programa e alterar suas características de acordo com o comportamento desejado.

3 Projeto Conceitual da Ferramenta

Esta seção trata das escolhas e métodos utilizados na implementação da ferramenta de simulação. Serão abordados os métodos de simulação, formatos para entrada de esquemáticos, e sistemas de propagação de sinais entre blocos.

3.1 Métodos de Simulação

3.1.1 Simulação Orientada à Equações

A simulação orientada à equações consiste em definir cada bloco como uma equação, e unir todos os blocos do esquemático utilizando equações capazes de definir o sistema simulado como um todo.

A vantagem deste sistema é que este oferece maior precisão nos resultados e o tempo de simulação é reduzido, quando comparado com um sistema de simulação em blocos.

As desvantagens são as dificuldades em simular eventos discretos

3.1.2 Simulação Seqüencial em Blocos

O método de simulação levou em consideração duas alternativas: simulação sequencial em blocos e simulação orientada a equações.

4 Implementação do Programa

Esta seção descreve em detalhes os requisitos necessários e os métodos utilizados para implementação da ferramenta de simulação proposta.

4.1 Projeto Multiplataforma

Um dos requisitos do sistema é que este seja capaz de rodar nos principais sistemas operacionais disponíveis atualmente. Duas ferramentas para desenvolvimento de sistemas multiplataforma foram analisadas com este objetivo.

4.1.1 Biblioteca Fox-Toolkit

A Biblioteca Fox-Toolkit (Fox-Toolkit, 2004) é um sistema de código fonte aberto, capaz de fornecer uma estrutura sobre a qual outros programas podem ser desenvolvidos e compilados em diferentes plataformas. É distribuído gratuitamente através da licença GPL, e oferece suporte para Windows, Linux, FreeBSD, Sun OS/Solaris, HP-UX, IBM AIX e DEC Alpha.

4.1.2 Biblioteca wxWidgets

A Biblioteca wxWidgets (wxWidgets, 2004) é muito semelhante a Fox-Toolkit, em termos de licenças e funcionalidade. Duas características, no entanto, foram levadas em consideração na escolha da wxWidgets como base para a ferramenta de simulação:

Suporte nativo para MacOS.

Disponibilidade de um sistema simples para suporte multilíngüe da interface gráfica com o usuário.

4.2 Estrutura do Programa

O programa foi implementado utilizando a linguagem C++. A estrutura de classes fornece uma hierarquia e uma divisão de funções de forma a facilitar a estruturação e manutenção do programa.

4.2.1 Conceitos da Linguagem de Programação

A linguagem de programação utilizada no desenvolvimento desta ferramenta foi C++. Para um melhor entendimento deste trabalho e da linguagem utilizada, serão relembrados alguns conceitos de programação orientada a objeto (Object Oriented Programming – OOP) (Campione, 2000).

Objetos

Um objeto é um bloco de programa formado por variáveis e métodos relacionados. Cada objeto descreve um estado e um comportamento aplicado a este objeto.

Classes

Uma classe é um gabarito ou protótipo, que define as variáveis e os métodos comuns a todos os objetos de um mesmo tipo.

Herança

Uma classe herda seu estado e comportamento de sua classe base, e então define métodos e variáveis específicos à classe herdada. Herança fornece um mecanismo poderoso e natural para organizar e estruturar programas.

Funções Virtuais

Funções virtuais, ou métodos virtuais, são funções definidas na classe base que podem substituídas, de forma a ter seu comportamento alterado na classe herdada.

4.2.2 Conceitos Básicos da Ferramenta

Estes são alguns conceitos utilizados na descrição da estrutura do programa.

Esquemático

Conjunto de blocos funcionais, com suas posições e estados, assim como a definição das interconexões entre estes blocos.

Blocos

Unidades funcionais capazes de simular um comportamento em função de suas entradas e saídas.

Conexões

São as estruturas que identificam as interconexões entre blocos. Cada conexão referencia um bloco e uma porta de saída, e um bloco e uma porta de entrada. Os valores de entrada e saída, em si, são mantidos pelas portas.

Portas

As portas se distinguem das conexões por serem as estruturas de dados que mantém os valores associados a cada uma das portas de saída de um bloco, assim como referências às portas de saídas em outros blocos para cada uma das portas de entrada conectadas.

4.2.3 Classes Definidas

Classe Painel (class vtbFrame)

Esta classe foi herdada da classe base wxFrame fornecida por wxWidgets. Ela é responsável pela implementação dos recursos básicos da interface gráfica do usuário, como menus, barra de ferramentas e acesso a arquivos. Apenas uma instância desta classe é criada para o programa.

Classe Paleta (class vtbPalette)

Esta classe foi herdada da classe base wxPanel fornecida por wxWidgets. Ela é responsável pela implementação da barra de blocos disponíveis para utilização no esquemático, à esquerda da interface gráfica com o usuário. Durante a inicialização do programa, todos os blocos disponíveis são registrados junto à esta classe. O programa implementa o recurso de arrastar e soltar, para posicionar blocos no esquemático.

Classe Esquemático (class vtbSchematic)

Esta classe também foi herdada da classe base wxPanel fornecida por wxWidgets. Ela é responsável pela implementação do esquemático em si, assim como todas as funções associadas ao esquemático.

As responsabilidades desta classe incluem:

a) Manter a lista e posição de todos os blocos.

b) Manter, criar e destruir conexões entre blocos.

c) Verificar a consistência das conexões entre blocos (ver seção 3.3.1 e 3.3.2).

d) Copiar e colar blocos.

e) Desfazer e refazer alterações ao esquemático.

f) Acessar a caixa de diálogo de propriedades dos blocos.

g) Ordenar os blocos para simulação (ver seção 3.4).

h) Administrar a simulação de cada um dos blocos.

Classe Bloco (class vtbBlock)

Esta classe não é uma classe herdada. Ela foi projetada de forma a fornecer as funções e comportamentos necessários à sua utilização junto ao esquemático. As responsabilidades desta classe incluem:

a) Criar, manter e destruir as portas de saída.

b) Atualizar as referências das portas de saída nas portas de entrada em outros blocos.

c) Implementar as funções necessárias ao esquemático para simular o sistema como um todo.

d) Criar e manter a caixa de diálogo com os parâmetros de cada instância utilizada no esquemático.

Classe Conexão (class vtbConnection)

Esta classe não é uma classe herdada. Ela foi projetada de forma armazenar uma ligação entre dois blocos. Cada objeto desta classe referencia dois blocos e duas portas, uma de entrada e uma de saída.

Esta classe não implementa nenhum método.

4.2.4 Interface Gráfica

A Figura 1 mostra a interface gráfica do usuário, e a função de cada uma das classes que fazem parte da estrutura básica do programa.

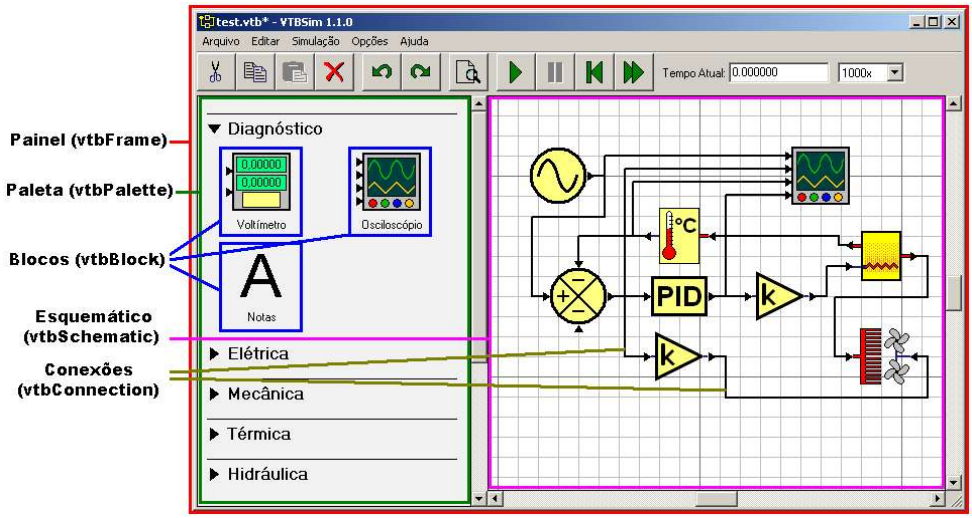

Figura 1 – Interface gráfica do usuário

4.3 Definição e Classificação dos Objetos Utilizados

Esta seção expande a explicação da classe Esquemático e da classe Bloco, as funções e relacionamentos entre estas classes, e a forma que estas características são utilizadas para simular um sistema dinâmico. Também são abordadas as portas, seus tipos e aplicações.

4.3.1 Portas de Entrada e Saída

A definição de uma porta como entrada ou saída é dada em função dos valores a ela relacionados. Como exemplo, uma porta do tipo elétrica de saída é definida como gerando uma tensão, e as portas de entrada e ela conectada consomem um valor de corrente dependendo desta tensão. Neste aspecto, a escolha da tensão como valor primário e corrente como uma função da tensão não é totalmente arbitrário, embora conceitualmente estes valores poderiam ser trocados. Como a estrutura interna do programa relativa à topologia do sistema simulado não contém a noção de malhas, apenas de nós, a escolha da tensão como valor primário se faz necessária em função do algoritmo de simulação.

A mesma lógica de simulação por nós pode ser aplicada à outros tipos de portas, fazendo pressão o valor principal na porta tipo hidráulica, velocidade (RPM) na porta mecanica, temperatura na porta tipo térmica.

Uma outra forma de abordar esta questão é definir que o valor principal é partilhado por todas as portas ligadas à um determinado nó, enquanto valores secundários aplicados à porta de saída são o somatório das contribuições individuais de cada uma das portas ligadas a este nó, conforme a Lei de Kirchoff.

4.3.2 Tipos de Portas e Suas Utilizações

Portas são os elementos de ligação em cada um dos blocos. Através das portas que se estabelecem as conexões que irão propagar sinais no esquemático, ligando uma porta de saída em um bloco a uma porta de entrada em outro(s) bloco(s).

O programa suporta 4 tipos de portas: elétrica, mecânica, hidráulica e térmica. Cada uma pode ser de entrada ou saída, e possui variáveis associadas de acordo com as grandezas que se deseja propagar de um bloco para outro.

A propagação sinais se dá conforme o método de fluxo de sinais (Liu, 1998), de uma porta de saída de um bloco para a porta de entrada de outro bloco (por exemplo, a alimentação de um motor). De acordo com o tipo de porta e do tipo de bloco, também é possível transferir valores de uma porta de entrada de volta para uma porta de saída. Por exemplo, se a saída de tensão de um amplificador for ligada a um motor, está conexão ira propagar o sinal de tensão da saída do amplificador à alimentação do motor. Por outro lado, é possível utilizar a mesma conexão entre amplificador e motor para propagar o valor da corrente do motor de volta ao amplificador, ou seja, de uma porta de entrada a uma porta de saída. No caso da conexão de mais de uma entrada à mesma saída, ou utilizando o exemplo anterior, mais de um motor ao mesmo amplificador, é necessário calcular a corrente total consumida pelos motores. Isto é feito aplicando a Lei dos Nós de Kirchoff. É necessário então, propagar a tensão do amplificador a todos os motores, e propagar reversamente o somatório das correntes de todos os motores de volta ao amplificador. Este método chama-se modelo da Lei de Conservação (Liu, 1998), que neste caso, se refere à conservação da corrente. A mesma técnica é utilizada em todos os outros tipos de portas. Por exemplo, são implementadas as versões equivalentes da Lei de Kirchoff para corrente, mas para conservação de fluxo de líquidos na porta hidráulica, torque na porta mecânica e calor na porta térmica.

Ao longo desta subseção, serão explicados os tipos de porta disponíveis, assim como os identificadores internos associados a cada uma das portas e às suas grandezas.

Porta tipo Elétrica (identificador: ptElectric)

Esta porta implementa apenas duas grandezas: tensão e corrente. Todos os blocos, que implementam funções matemáticas, utilizam este tipo de porta como entrada e saída.

Tabela 1 – Valores da Porta Elétrica

Valor	Unidade	Identificador	Propagação
Tensão	V	pelVoltage	Direta
Corrente	A	pelCurrent	Reversa

Porta tipo Hidráulica (identificador: ptHidraulic)

Esta porta implementa apenas valores de pressão, fluxo e temperatura. Outros dois valores auxiliares, fluxo de entrada e temperatura de entrada, são utilizados para avaliar corretamente a temperatura do fluxo de líquido que está passando pela porta.

Tabela 2 – Valores da Porta Hidráulica

Valor	Unidade	Identificador	Propagação
Pressão	kPa	phdPressure	Direta
Fluxo	l/min	phdFlow	Reversa
Temperatura	°C	phdTemp	Igual ao fluxo
Fluxo de Entrada	l/min	phdFlowIn	Reversa
Temperatura de Entrada	°C	phdTempIn	Reversa

Para avaliar a temperatura do fluxo que está entrando ou saindo de uma porta hidráulica, no caso de existirem múltiplas entradas ligadas a uma saída, as seguintes regras são aplicadas:

a) A temperatura do fluido de saída é sempre a temperatura interna do fluido no tanque, ou seja, a temperatura de fluidos entrando no tanque não afeta a temperatura de saída.

b) A temperatura do fluido é propagada na mesma direção do fluxo.

c) A temperatura do fluido de entrada é a média ponderada, pelo fluxo, da temperatura de todos os fluidos de entrada.

Este último item é o caso em que se utiliza o valor auxiliar Fluxo de Entrada e Temperatura de Entrada. Para calcular de que maneira a temperatura de vários fluxos ligados a uma mesma saída de um tanque interagem, é necessário determinar a topologia para as ligações físicas no tanque, ou seja , de que forma a temperatura de fluxo saindo do tanque é afetada diretamente pela temperatura de um fluxo entrando no tanque.

O modelo implementado define que não há propagação direta entre fluxos, isto é, que fluxos de entrada afetam somente a temperatura do tanque (e não outros fluxos), e fluxos de saída tem a temperatura igual à do tanque. Então, as variáveis Fluxo de Entrada e Temperatura de Entrada armazenam a média ponderada de temperatura e o fluxo total de entrada, e as variáveis Fluxo e Temperatura armazenam o fluxo total de saída e a temperatura do tanque. O bloco que implementa a saída hidráulica é responsável pela avaliação e combinação dos dois valores de fluxo e dos dois valores de temperatura de acordo com seu algoritmo interno de simulação.

A direção de propagação da variável Temperatura é definida como “igual ao fluxo”. Isto é importante para dispositivos de passagem de fluxo de líquido (por exemplo, uma vávula), onde todo o fluxo que entra, sai no mesmo instante de tempo. Para manter a consistência com o comportamento de um fluxo real, a variável Temperatura deve ser propagada na mesma direção do fluxo.

Porta tipo Mecânica (identificador: ptMechanic)

Esta porta implementa as grandezas de velocidade angular, torque e momento de inércia. O momento de inércia é utilizado para estimar, de forma direta, o pólo mecânico do sistema formado pela interconexão de portas mecânicas.

Tabela 3 – Valores da Porta Mecânica

Valor	Unidade	Identificador	Propagação
Velocidade	RPM	pmcSpeed	Direta
Torque	N/m	pmcTorque	Reversa
Inércia	Nm/rad/s^2	pmcInertia	Reversa

Porta tipo Térmica (identificador: ptThermic)

Esta porta implementa apenas valores de temperatura e calor. O calor é utilizado para transferir energia calorífica entre blocos ligados por portas do tipo térmica.

Tabela 4 – Valores da Porta Térmica

Valor	Unidade	Identificador	Propagação
Temperatura	°C	ptpTemp	Direta
Calor	J	ptpHeat	Reversa

4.3.3 Limitações Intrínsecas da Interface Gráfica

A interface gráfica implementa propositadamente duas limitações ao tipo de conexão que pode ser estabelecida entre blocos.

Combinação entre Entradas e entre Saídas

A conexão é sempre estabelecida de uma porta de saída para uma porta de entrada. Ligações entre duas entradas ou duas saídas não são permitidas pela interface gráfica.

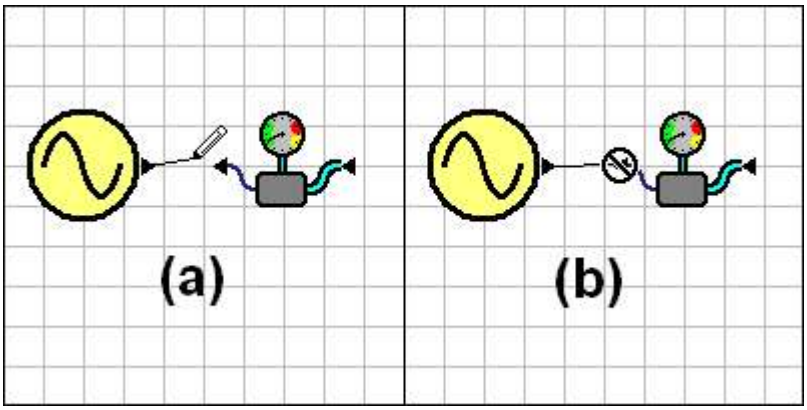

Figura 2 – Bloqueio da ligação entre portas de saída

Combinação de Diferentes Tipos de Portas

Conexões só podem ser estabelecidas entre portas do mesmo tipo, ou seja, portas elétricas só podem ser ligadas a portas elétricas, portas mecânicas só podem ser ligadas a portas mecânicas; o mesmo é válido para os outros tipos de portas.

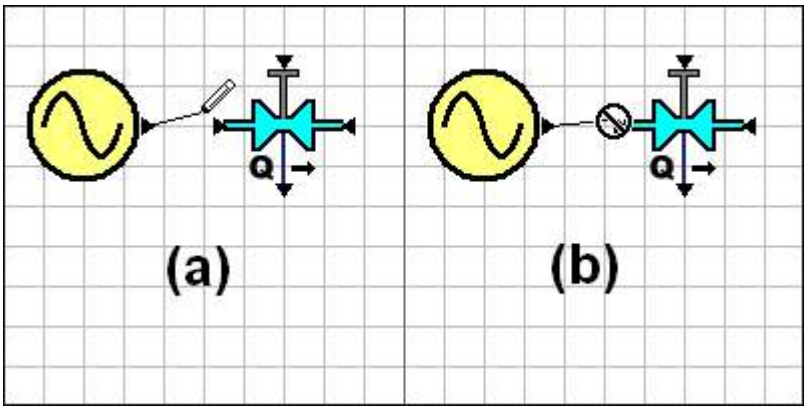

Figura 3 – Bloqueio da ligação entre portas de diferentes tipos

4.3.4 Tipos de Blocos e suas Utilizações

Existem dois tipos básicos de blocos, sob o ponto de vista da ordem em que os mesmos são simulados: blocos ordenados e blocos não-ordenados. Esta distinção é importante para o método de simulação empregado pela classe Esquemático.

Blocos ordenados

Blocos ordenados são blocos que precisam ser avaliados segundo uma ordem causal definida, de acordo com a maneira que estão ligados entre si no esquemático. Isto inclui dois tipos de blocos:

Blocos cujas saídas não dependem das entradas anteriores, isto é, possuem propagação instantânea. Nestes blocos, as variações na entrada causam variações na saída sem que seja necessário avançar o passo da simulação. Por exemplo, o bloco Amplificador, onde a saída é sempre uma função algébrica, igual à entrada multiplicada pelo ganho. Neste tipo de bloco, a saída é mantida sincronizada porque é garantida a ordem causal definida pelo esquemático.

Blocos que possuem entradas não-elétricas. Estes blocos são ordenados para que valores de propagação reversa, associados a portas de saída (e.g. Torque em portas mecânicas), sejam avaliados na ordem correta. Por exemplo, veja a Figura 4, abaixo:

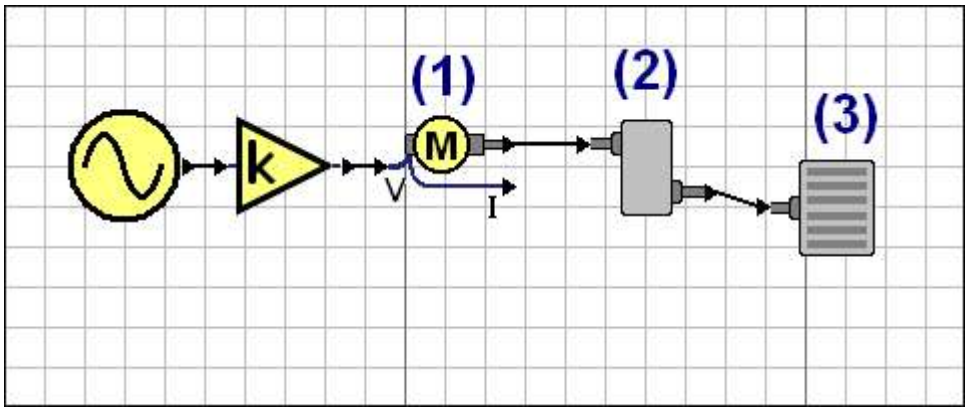

Figura 4 – Exemplo de propagação de sinais através de portas não-elétricas

Neste exemplo, os blocos devem ser simulados na ordem 1, 2, 3. Isto é necessário, para que o sinal de velocidade do motor (1) seja propagado para o redutor (2), e deste para a carga (3), no mesmo passo da simulação. Da mesma forma, é possível garantir que a propagação reversa de grandezas (no caso, torque e inércia) chegue da carga (3) até o motor (1) se a ordem inversa de simulação for seguida, ou seja, 3, 2, 1 (ver subseção 3.5.4).

A exceção é feita aos blocos que possuem entradas elétricas devido a uma limitação intrínseca ao método de simulação, que não propaga reversamente a corrente através de uma seqüência ordenada de blocos no esquemático. Utilizando o mesmo exemplo da Figura 4, O sinal de tensão é propagado do gerador de sinais, ao amplificador e à entrada do motor. Embora seja possível propagar reversamente a corrente consumida pelo motor para a saída de tensão do amplificador, o algoritmo implementado não suporta a transferência desta grandeza de volta ao gerador de sinais.

Blocos não-ordenados

Blocos não-ordenados são blocos cujas saídas dependem da entrada no instante atual, e também das entradas anteriores, ou seja, estes blocos retêm os seus estados atuais, de um passo da simulação para o próximo.

Como o programa não garante a ordem em que estes blocos serão simulados, é necessário que todos os blocos não-ordenados amostrem suas entradas no instante atual, antes que qualquer outro bloco altere o valor de suas saídas.

4.3.5 Limitações na Topologia de Conexões

Devido à necessidade de certos blocos serem ordenados para a simulação, a definição de dependências circulares (loops algébricos) para estes blocos é ilegal.

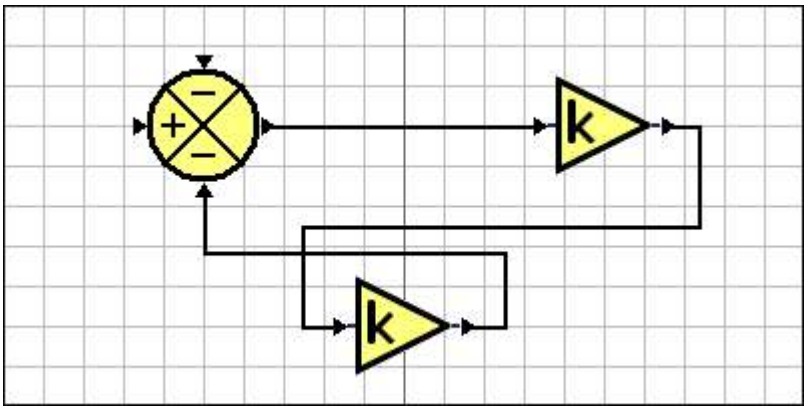

Figura 5 – Configuração ilegal com dependência circular

A interface gráfica não é capaz detectar estas topologias ilegais, como a mostrada na Figura 5, enquanto as conexões estão sendo definidas. Por outro lado, quando o esquemático tentar ordenar os blocos para a simulação, será gerado um erro.

4.3.6 Conexões

As conexões são objetos controlados pela classe vtbConnection. Cada elemento define o bloco de origem e o índice da porta no bloco de origem (sempre uma porta de saída), assim como o bloco de destino e o índice da porta de destino (sempre uma porta de entrada).

Cada ligação no esquemático possui um item da classe vtbConnection definindo esta conexão. Este objeto é controlado pela classe Bloco e pela classe Esquemático, isto é, cada bloco sabe quais conexões pertencem a ele, e o esquemático tem a lista de todas as conexões entre todos os blocos.

4.4 Ordenação dos Blocos

Conforme explicado na subseção 3.3.3, esta fase é necessária para gerar uma ordem de simulação dos blocos, ou seja, estabelecer uma ordem causal na propagação das grandezas, de

forma a garantir o sincronismo dos sinais através dos diferentes tipos de blocos no esquemático.

Outra responsabilidade desta função é estabelecer as referências entre as portas de entrada e as portas de saída. Cada porta de entrada possui um ponteiro para referenciar diretamente os valores da porta de saída à qual está conectada. Durante a verificação das dependências entre blocos, cada bloco define os ponteiros de todas as portas de entrada ligadas às suas portas de saída.

Esta função é chamada pela classe Esquemático, apenas uma vez durante cada simulação, quando esta é iniciada ou reiniciada.

Este evento é implementado pela classe Esquemático, através da função vtbSchematic::SetCorrectOrder( ).

Esta é a primeira função executada quando o usuário inicia uma simulação.

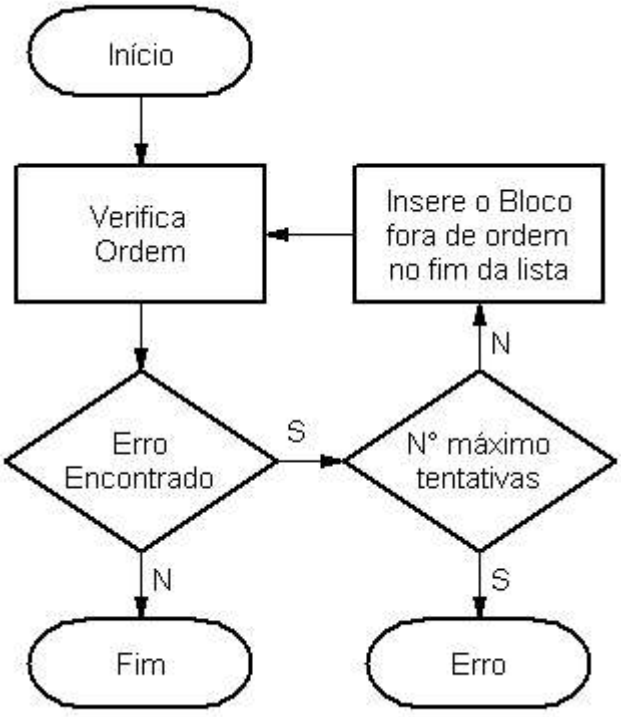

Figura 6 – Algoritmo de ordenação dos blocos

Os blocos que precisam ser ordenados (ver subseção 3.3.3) são verificados através de uma função derivada do algoritmo Bolha e do algoritmo de Inserção.

O algoritmo Bolha funciona iterando sobre todos os itens de uma lista, dois a dois, e trocando de lugar os itens que estiverem fora de ordem. O algoritmo de Inserção funciona iterando sobre todos os itens de uma lista, e recolocando cada item na posição correta.

O algoritmo de ordenação da ferramenta funciona de forma semelhante. Cada bloco possui um indicador se este bloco já foi avaliado para um determinado passo da simulação. Para determinar a ordem correta, a classe Esquemático itera sobre a lista dos blocos, e cada bloco verifica, através de sua lista de conexões, se algum bloco ordenado dependente já foi avaliado; se for encontrado, o bloco que está verificando suas conexões é colocado no fim da lista, e uma nova iteração é iniciada.

Este processo se repete até que nenhum erro de ordenação seja encontrado, ou o número máximo de tentativas tenha expirado. Caso uma ordenação válida não seja encontrada, um erro é indicado através da interface gráfica. Esta função é executada apenas uma vez, sempre que a simulação for iniciada ou reiniciada.

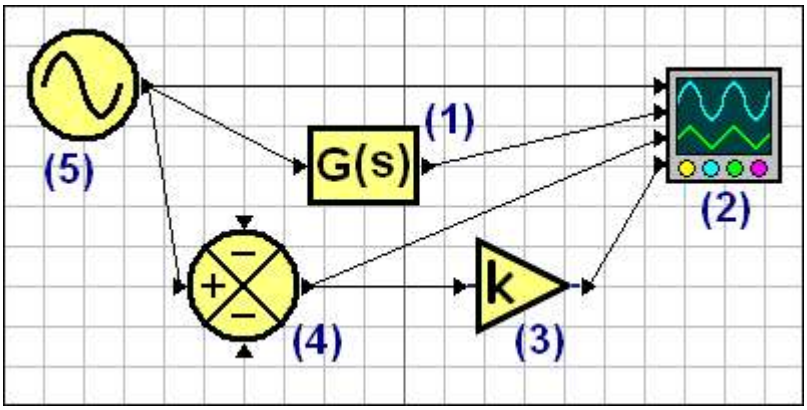

Figura 7 – Exemplo de ordenação da classe Esquemático

Por exemplo, a tabela 5 mostra os passos seguidos pelo esquemático ao tentar ordenar o sistema da figura 7. Para cada par de blocos fora de ordem, o primeiro bloco é colocado no fim da lista:

Tabela 5 – Processo de ordenação de blocos passo a passo

Passo	Ordem					Observação (erro de ordenação)
1	2	3	4	5	1	amplificador (3) após osciloscópio (2)
2	3	4	5	1	2	amplificador de erro (4) após amplificador (3)
3	4	5	1	2	3	gerador (5) após amplificador de erro (4)
4	5	1	2	3	4	amplificador (3) após osciloscópio (2)
5	5	1	3	4	2	amplificador de erro (4) após amplificador (3)
6	5	1	4	2	3	amplificador (3) após osciloscópio (2)
7	5	1	4	3	2	-

Note que o bloco Função de Transferência nunca será considerado fora de ordem e posto no fim da lista, pois é um bloco não ordenado (ver seção 3.3.3).

4.5 Processo de Simulação

O programa implementa um algoritmo de simulação utilizando um método híbrido entre o método de fluxo de sinais e o método da lei de conservação (Liu, 1998), através da avaliação de blocos funcionais, onde o tempo avança com passo fixo.

O algoritmo segue uma seqüência de eventos, que devem ser executados para cada passo durante a simulação.

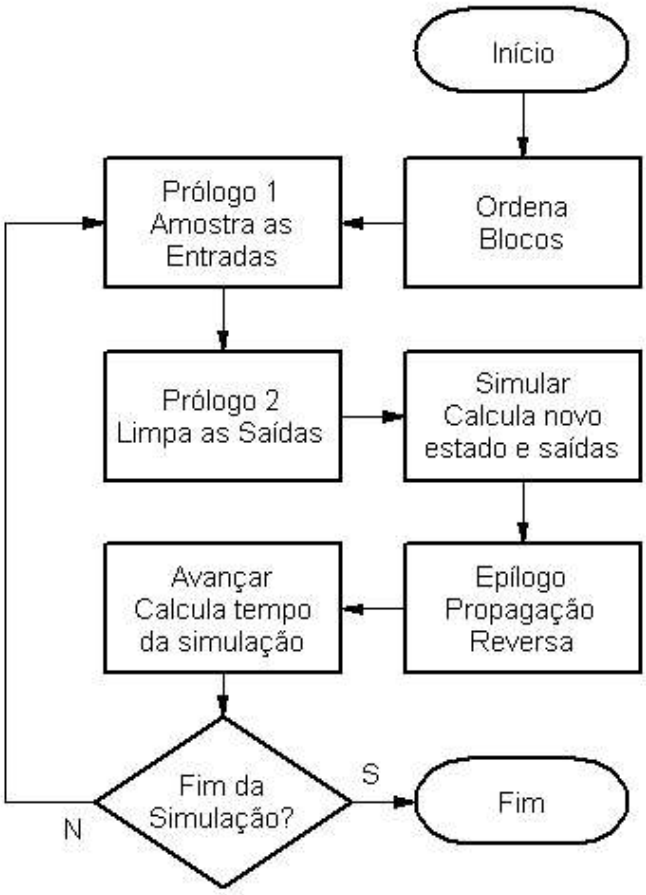

Figura 8 – Algoritmo de Simulação do Esquemático

Cada uma das fases, mostradas no algoritmo da O algoritmo segue uma seqüência de eventos, que devem ser executados para cada passo durante a simulação., é aplicada a todos os blocos do esquemático. Estas são as funções que implementam diretamente o algoritmo de simulação em cada um dos blocos, mas nem todas estas funções precisam ser necessariamente substituídas no bloco herdado. A seguir, detalharemos cada uma destas fases.

4.5.1 Prólogo 1

Esta função é definida como parte da classe Bloco, através da função virtual vtbBlock::Prolog1( ).

Blocos que dependem da entrada atual e das entradas anteriores, ou possuem entradas não-elétricas (ver subseção 3.3.3) não são ordenados. Assim sendo, o algoritmo não garante a ordem em que estes blocos serão avaliados e, conseqüentemente, alterarão suas saídas.

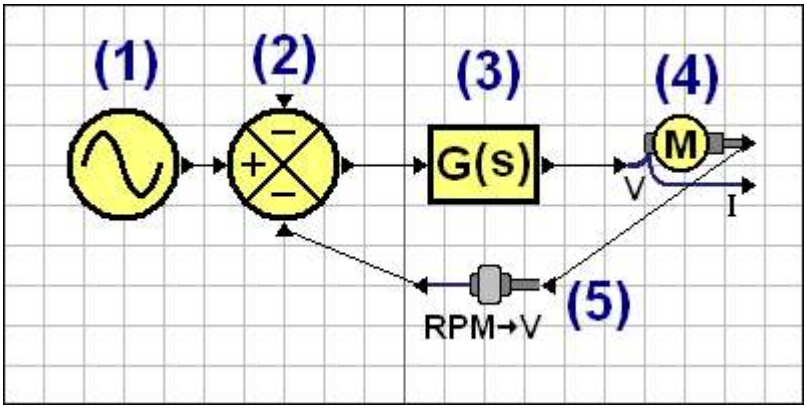

Figura 9 – Exemplo de uso da função Prólogo 1

No exemplo da figura 9, são mostrados 2 blocos não-ordenados: o bloco 3 (Função de Transferência) e o bloco 4 (Motor). Como, para cada passo da simulação, o algoritmo não garante qual deles será avaliado primeiro, e duas situações poderiam ocorrer para o instante t:

No primeiro caso, o bloco (3) calcula a saída para o instante t +1 antes do bloco (4). Por conseqüência, o bloco (4) utilizaria a entrada do instante t +1 para calcular suas saídas, ao invés de usar a entrada do tempo t.

O inverso aconteceria se o bloco 4 fosse avaliado antes do bloco 3. Sua saída seria propagada através dos blocos 5 e 2, e seria aplicada a entrada do bloco 3.

Para isto, foi implementada a função virtual vtbBlock::Prolog1( ). Cada bloco não-ordenado herdado da classe base vtbBlock deve fornecer uma implementação desta função. O objetivo é amostrar todas as entradas de todos os blocos não-ordenados, antes de dar a qualquer outro bloco a oportunidade de mudar o valor nas suas saídas. Esta é a primeira fase em cada passo da simulação do esquemático.

4.5.2 Prólogo 2

Esta função é definida como parte da classe Bloco, através da função virtual vtbBlock::Prolog2( ).

Portas de entrada, de qualquer tipo, têm a capacidade de propagar valores de volta a porta de saída a elas conectadas. Por exemplo, portas de entrada tipo elétrica propagam o valor de corrente de volta a porta de saída que está gerando o sinal de tensão.

A interface gráfica permite somente a ligação de uma saída para cada entrada, mas permite a ligação de múltiplas entradas a partir da mesma saída. O processo de propagar o valor de saída para múltiplas entradas é simples, pois não há conflitos; apenas um valor existe para ser distribuído entre as entradas. A recíproca é mais complexa; existem possivelmente múltiplas entradas capazes de propagar valores (e.g. corrente, torque) de volta para a porta de saída.

Para resolver esta situação, foi aplicada a Lei dos Nós de Kirchoff, ou seja, a corrente na porta de saída será o somatório das correntes de todos os nós de entrada a ela conectadas. O raciocínio equivalente foi aplicado a todos ou outros tipos de porta, onde o valor aplicado à porta de entrada é o somatório dos valores propagados reversamente às portas de saída.

O propósito da função Prólogo 2, então, é zerar todos os somadores dos valores em que se aplica a Lei dos Nós. Durante a fase Epílogo, cada bloco que tenha portas de entrada conectadas pode, opcionalmente, somar valores às portas de saída.

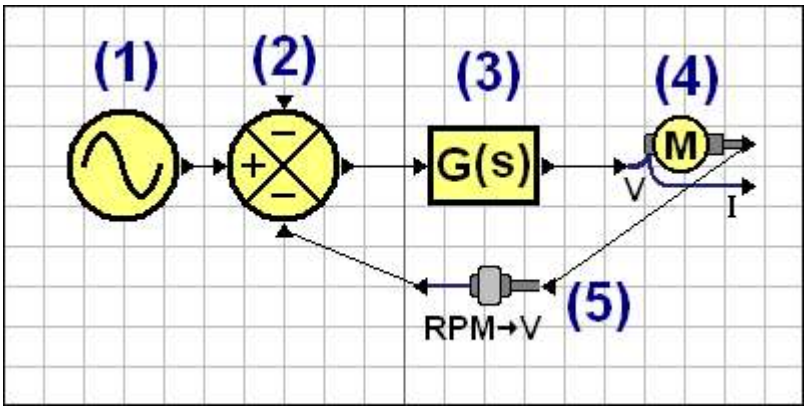

Figura 10 – Exemplo de uso da função Prólogo 2

No exemplo da figura 10, a função Prólogo 2 zera o valor de fluxo na saída do bloco 1. Posteriormente os blocos 2 e 3, ligados através das conexões a e b, irão somar seus valores individuais de fluxo ao acumulador mantido pela porta de saída hidráulica do bloco 1.

Abaixo, temos a tabela dos tipos de porta e os valores que devem ser zerados pela função Prólogo 2 e que, posteriormente, podem ser propagados de volta às portas de saída utilizando a Lei dos Nós.

Tipo de Porta Valores zerados pela função Prólogo2:

Tabela 6 – Lista de valores propagados de volta às saídas

Elétrica	Corrente
Mecânica	Torque, Inércia
Hidráulica	Fluxo, Temperatura, Fluxo de Entrada, Temperatura de Entrada
Temperatura	Calor

4.5.3 Simular

Esta função é definida como parte da classe Bloco, através da função virtual vtbBlock::Simulate(double step_time). Cada bloco deve implementar sua própria versão desta função, de acordo com o objetivo do bloco.

Esta função implementa a principal fase de cada passo do algoritmo de simulação. A classe Esquemático chama cada um dos blocos, de acordo com a ordem em que foram colocados pela função vtbSchematic::SetCorrectOrder( ) (ver subseção 3.4). Cada bloco, por sua vez, calcula sua(s) saída(s) usando como parâmetro(s) o(s) valor(es) presente na(s) entrada (s) no instante em que esta função for chamada (blocos ordenados, subseção 3.3.3), ou então, usando como parâmetro(s) o(s) valore(s) que foram amostrados previamente pela função Prólogo 1 (blocos não-ordenados, ver subseção 3.3.3).

Após calcular a saída para o instante t+1, utilizando um avanço de tempo definido pelo parâmetro double step_time, o bloco coloca o novo valor na sua porta de saída. Blocos ordenados utilizam este novo valor, por sua vez, para calcular as suas próprias saídas. A principal conseqüência disto, é que ao final de um passo de simulação, todos os blocos que possuem propagação instantânea (que não dependem de entradas em intervalos anteriores para calcular suas saídas) tem suas saídas atualizadas e coerentes em função de suas entradas.

4.5.4 Epílogo

Esta função é definida como parte da classe Bloco, através da função virtual vtbBlock::Epilog( ).

O objetivo desta função é garantir a consistência na propagação reversa, isto é, a propagação de valores das entradas para as saídas dos blocos. Neste caso, a classe Esquemático chama esta função para todos os blocos, na ordem reversa em que estes se encontram para as outras fases da simulação de cada um dos passos, ou seja, o oposto da ordenação estabelecida pela função vtbSchematic::SetCorrectOrder( ) (ver subseção 3.4)

Esta função, então, permite que blocos que propagam valores através de suas entradas, tenham estes valores atualizados, em função dos valores propagados por outros blocos, para a suas saídas. Por exemplo:

Conforme a figura 4 durante a fase de simulação, a velocidade de saída do bloco 1 (Motor) será propagada à entrada do bloco 2 (Redutor), e deste para o bloco 3 (Carga). Durante a fase Epílogo, o torque e a inércia serão propagados do bloco 3 para o bloco 2, e o bloco 2 irá então calcular a inércia e o torque total e propagá-los de volta para a saída do bloco 1. Desta forma, no início do próximo passo de simulação, o bloco 1 terá o valor atualizado de todas as cargas ligadas à sua saída.

4.6 Gerador de Ruído

A classe Bloco implementa o suporte para a simulação de ruído através das funções vtbBlock::PrepareNoise( ) e vtbBlock::GetNoise( ).

O ruído encontrado normalmente em sistemas físicos não apresenta uma distribuição uniforme de amplitude. A amplitude de ruído somado ao sinal de interesse ao longo do tempo apresenta uma distribuição normal, ou seja, uma amplitude descrita estatisticamente por uma curva gaussiana. A este tipo de ruído se dá o nome de ruído Gaussiano. Sendo assim, se utilizarmos uma função de geração de números pseudo-randômicos na escala em que se deseja simular ruído, este apresentará uma distribuição uniforme, e não uma distribuição normal ou Gaussiana (Johnson, 2003).

O ruído Gaussiano pode ser implementado a partir de um gerador de números pseudo-randômicos com distribuição uniforme. Para tanto, e necessário primeiramente um gerador pseudo-randômico. Esta função foi implementada a partir da biblioteca Mersenne Twister. Esta biblioteca foi escolhida por apresentar uma qualidade muito superior às implementações

padrão da função rand() disponíveis no sistema operacional, com período mínimo garantido de 219937-1 (Matsumoto, 2004).

Sejam U_1 e U_2 dois números randômicos com distribuição uniforme na faixa (0,1]. Dois números randômicos com distribuição Gaussiana G_1 e G_2 podem ser obtidos utilizando a transformação de Box-Muller: (Ross, 1997)

$$G_1 = \sqrt{-2 \cdot \log(U_1)} \cdot \text{sen}(2 \cdot \pi \cdot U_2) \quad \textbf{(1)}$$

$$G_2 = \sqrt{-2 \cdot \log(U_1)} \cdot \cos(2 \cdot \pi \cdot U_2) \quad \textbf{(2)}$$

Abaixo, temos a comparação entre um sinal de ruído utilizando uma distribuição uniforme, e um sinal de ruído Gaussiano gerado utilizando a transformação de Box-Muller:

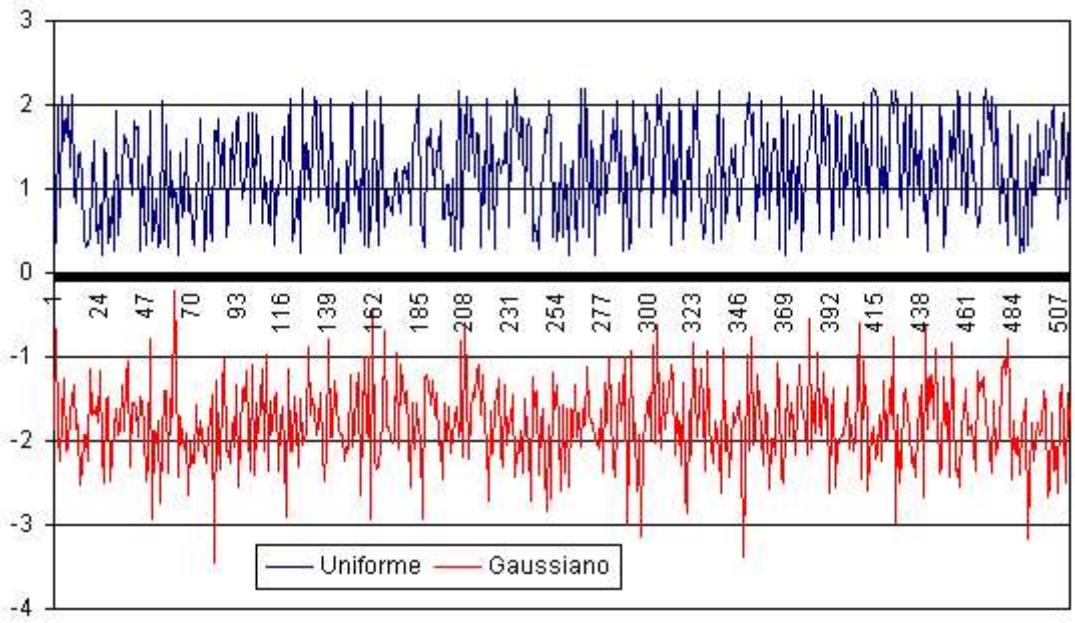

Figura 11 – Comparação no domínio tempo entre ruído uniforme e Gaussiano

Embora sejam semelhantes em aparência, a comparação da distribuição em freqüência dos dois métodos para geração de ruído mostra a resposta mais uniforme obtida com um sinal de ruído Gaussiano.

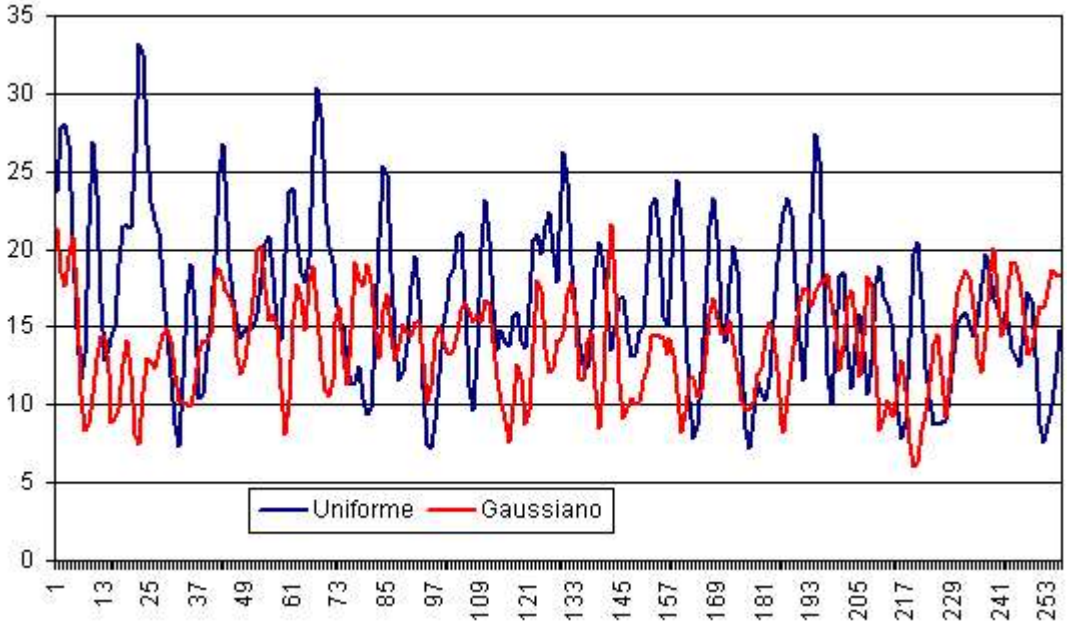

Figura 12 - Comparação no domínio freqüência entre ruído uniforme e Gaussiano

Este método para geração de ruído é utilizado nos blocos Amplificador, Tacômetro, Medidor de Pressão, Termômetro e Gerador de Sinais. Cada um dos blocos pode ser configurado individualmente, utilizando uma escala logarítmica em dB, ou seja, a cada 20 unidades, a amplitude do ruído é multiplicada por dez.

A transformação de Box-Muller gera um sinal com média zero e variância unitária. Para modificarmos a amplitude do ruído em função da variância de uma amostra G, fazemos (Ross, 1997):

$$G_V = \sqrt{\delta} \cdot G \qquad \textbf{(3)}$$

Seja K_R um ganho fixo do ruído adicionado ao sinal de interesse, e C_R o valor determinado pelo usuário através da interface gráfica para ajustar a amplitude do ruído. Sendo assim, se escolhermos um valor absoluto de ganho K_R, a variância da distribuição normal de amplitude do ruído, em função do valor de controle de ruído do bloco C_R, é dada por:

$$\delta = \left(K_R \cdot 10^{\frac{C_R}{20}}\right)^2 \quad \textbf{(4)}$$

Então, para o valor escolhido na ferramenta, K_R igual a 1x10-4, e para o valor C_R determinado pelo usuário variando entre 0 e 100, a variância do sinal de ruído ficará entre os seguintes valores:

$$\delta_{MIN} = \left(0.0001 \cdot 10^{\frac{0}{20}}\right)^2 = 10nV \quad \textbf{(5)}$$

$$\delta_{MAX} = \left(0.0001 \cdot 10^{\frac{100}{20}}\right)^2 = 100V \quad \textbf{(6)}$$

5 Estrutura e Definição de Novos Tipos de Blocos

A extensibilidade da ferramenta através da definição de novos blocos é uma de suas principais características. A estrutura dos blocos foi definida de forma a implementar uma interface genérica entre o algoritmo de simulação e cada uma das diferentes classes de blocos, assim como para facilitar a definição de novas classes de bloco herdadas da classe vtbBlock. A ferramenta foi definida de forma a assumir o maior número possível de tarefas, simplificando a responsabilidade dos blocos.

Este capítulo fornece informações mais detalhadas sobre a estrutura dos blocos, a qual servirá também de guia para a construção de novos blocos. A maneira mais fácil de abordar esta tarefa é partindo de um bloco já existente, e modificando seu comportamento, de acordo com as especificações do novo bloco.

A seguir, serão explicadas quais são as funções que devem ser implementadas ou modificadas, quais funções são obrigatórias e quais são as funções opcionais, e quais são as considerações necessárias na programação. Também serão detalhadas várias funcionalidades do algoritmo de simulação, cuja compreensão seja necessária na definição de um novo bloco.

As funções vtbBlock::Prolog1(), vtbBlock::Prolog2(), vtbBlock::Simulate(), vtbBlock::Epilog(), vtbBlock::PrepareNoise() e vtbBlock::GetNoise() já foram abordadas na subseção 3.5 como parte da explanação do algoritmo de simulação, e não serão expostas neste capítulo.

5.1 Definindo Novos Tipos de Blocos

A definição de um novo tipo de bloco se dá através da criação de uma classe derivada da classe vtbBlock. O usuário então define funções (métodos) para substituir o

comportamento padrão inplementado pela classe principal, incluindo funções para inicializar, simular, ler a gravar os dados relativos a cada instancia do novo tipo de bloco no esquemático.

A maneira mais simples de criar novos tipos de blocos consiste em alterar um bloco pré-existente, adequando suas funções de acordo com o novo comportamento desejado. Os demais sub-ítens deste capítulo cobrem em maiores detalhes a utilização e funções dos métodos implementados na classe vtbBlock.

5.2 Nomenclatura

O nome do novo bloco é arbitrário, mas é recomendado manter o padrão utilizado pelos blocos anteriores, ou seja, "vtbBlock_xxx", onde "xxx" é um mnemônico relativo ao bloco que está sendo definido. Todos os exemplos deste capítulo utilizarão "Teste" como o mnemônico do bloco que está sendo definido, ou seja, o nome do bloco será vtbBlock_Teste.

5.3 Considerações Multilíngües

Originalmente, o programa utiliza strings na língua inglesa, que podem ser traduzidas para qualquer outra língua.

O programa utilizou um gerador de dicionários chamado poEdit. Todas as strings no programa que precisam ser traduzidas, devem utilizar uma macro que é identificada pelo programa poEdit e adicionada à lista de strings que necessitam tradução. Esta macro é definida pelo caracter sublinhado "_". Por exemplo: _("string que precisa ser traduzida").

Maiores detalhes de como utilizar poEdit e gerar novos dicionários, podem ser obtidos na documentação do próprio programa (poEdit, 2004).

Um exemplo de código de bloco, onde fica evidenciado a utilização das funções descritas, pode ser encontrado no Apêndice A.

5.4 A Caixa de Diálogo de Propriedades

A Caixa de Diálogo de cada bloco não é uma função em si, mas sim uma classe herdada de wxDialog. Cada classe herdada deverá implementar 3 funções:

Construtor

O construtor é uma função que cria todos os controles e define seus posicionamentos na caixa de diálogo. O Gerador de Sinais (vtbSignalgen.cpp) contém um bom exemplo de um construtor para a caixa de diálogo.

TransferDataToWindow( )

Esta função copia os valores do bloco para a caixa de diálogo. Sempre que esta função for chamada, os valores na caixa de diálogo irão refletir os valores atuais do bloco.

TransferDataFromWindow( )

Esta função copia os valores da caixa de diálogo de volta para o bloco. Qualquer validação dos valores definidos pelo usuário deve ser feita nesta função.

Maiores detalhes a respeito do construtor da caixa de diálogo, e quais são as funcionalidades que podem ser utilizadas, podem ser encontrados na documentação da biblioteca wxWidgets (wxWidgets, 2004).

5.5 Transferência de Valores Entre Blocos

A transferência de valores entre blocos se dá de forma direta, sem necessidades de cópias, já que cada grupo de conexões partilha os valores relativos a um nó, desta forma:

Cada porta de saída é responsável por alocar e manter a lista de valores relativos a um nó, enquanto todos as outras portas de entrada conectadas a esta porta de saída apenas referenciam os valores através de um ponteiro para a porta de saída.

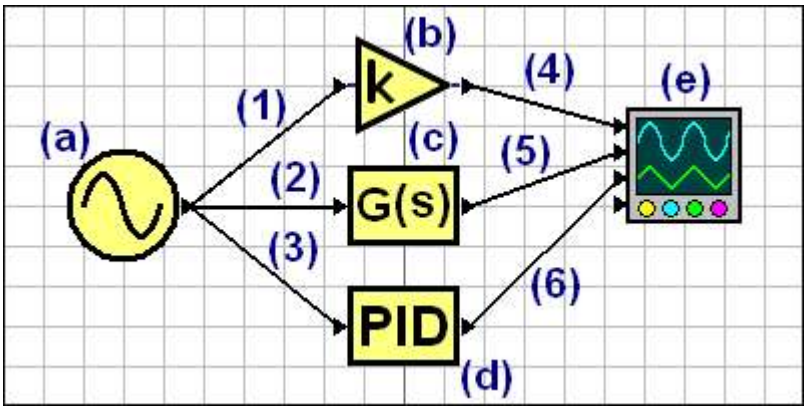

Figura 13 – Método de ligação entre blocos

Como exemplo, a figura 13 mostra a ligação entre cinco blocos diferentes. As conexões 1, 2 e 3 formam um nó e partilham o mesmo grupo de valores, estes valores são criados e mantidos pelo bloco (a), e apenas referenciados pelos blocos (b), (c) e (d). Já no caso das conexões 4, 5 e 6, cada uma forma um nó separado, e possui um conjunto de valores mantidos pelos blocos (b), (c) e (d), respectivamente, enquanto as portas do bloco (e) possuem os ponteiros que referenciam os valores das portas de saída.

A alocação de memória para as portas de saída é uma necessidade imposta pelo algoritmo de simulação. Como uma porta de saída pode ser ligada a várias portas de entrada, e uma porta de entrada pode ser ligada a uma e apenas uma porta de saída, a escolha foi feita para que a responsabilidade de manter os valores das conexões seja da porta de saída. Desta forma, como a interface garante que duas portas de entrada ou duas portas de saída não podem ser conectadas, existe sempre uma, e apenas uma, porta capaz de manter os valores para qualquer grupo de conexões.

Desta forma, durante o algoritmo de simulação, quando um bloco escreve um valor em uma porta, todas as portas a ela conectadas têm acesso a este valor, sem necessidade de nenhum outro processamento.

5.6 Construtor do Bloco

Função: vtbBlock_Teste::vtbBlock_Teste() : vtbBlock()

Esta função é chamada quando o bloco é criado, de forma a ser colocado na paleta de blocos disponíveis para utilização no esquemático.

Esta função deve realizar quatro tarefas:

a) Inicializar o ponteiro da caixa de diálogo para NULL, caso este exista.

b) Inicializar todos os parâmetros do bloco para seus valores padrão.

c) Alocar a memória que irá armazenar os valores relativos a todas as portas de saída deste bloco.

d) Inicializar os valores das portas de saída. Isto pode ser feito localmente ou chamando a função vtbBlock::ResetBlock().

5.7 Cópia do Bloco

Função: vtbBlock_Teste::vtbBlock_Teste(const vtbBlock_Teste a) : vtbBlock(a)

Esta função é chamada pela função vtbBlock_Teste::Clone();

Esta função é um outro tipo de Construtor, com a diferença que seus valores iniciais a estado são copiados de outro bloco do mesmo tipo. A função deverá inicializar o ponteiro da caixa de diálogo, alocar as portas de saída, inicializar as portas de saída, mas os valores associados aos parâmetros e o estado atual do bloco, ao invés de serem os valores padrão, serão os valores copiados do outro bloco sendo clonado.

O bloco clonado é identificado pelo parâmetro "a" na chamada da função.

5.8 Destruidor do Bloco

Função: vtbBlock_Teste::~vtbBlock_Teste()

Esta função é chamada automaticamente pelo sistema básico da linguagem C++. O programa não necessita chamar esta função explicitamente, uma vez que ela é chamada automaticamente sempre que um bloco for destruído.

Esta função tem apenas duas responsabilidades:

a) Liberar a memória alocada para as portas de saída

b) Destruir a caixa de diálogo, caso esta tenha sido criada pela função vtbBlock_Teste::Properties()

5.9 Clonagem do Bloco

Função: vtbBlock* vtbBlock_Teste::Clone()

Esta função é chamada sempre que um novo bloco for colocado no esquemático. O bloco pode ser clonado de outro bloco já existente no esquemático ou clonado do bloco existente na paleta de blocos.

Esta é uma função simples, normalmente implementada diretamente na definição da classe vtbBlock_Teste. Sua implementação cria uma cópia do bloco atual através do construtor da subseção 4.6, e retorna um ponteiro para a cópia.

5.10 Nome Traduzido do Bloco

Função: wxString vtbBlock_Teste::GetName()

Esta função é chamada quando a ferramenta precisa obter o nome do bloco, para ser utilizado na interface gráfica do usuário.

Função normalmente implementada na definição da classe herdada. Sua implementação retorna uma string utilizando a macro para tradução da interface (subseção 4.2). Esta string é utilizada para referenciar o nome do bloco na interface gráfica do usuário.

5.11 Nome Fixo do Bloco

Função: wxString vtbBlock_Teste::GetFixName()

Esta função é chamada quando a ferramenta precisa obter o nome do bloco, para ser utilizado no salvamento e leitura de arquivos. O usuário não tem acesso a este nome através da interface.

Função implementada na definição da classe herdada. Sua implementação retorna uma string sem utilizar a macro para tradução. Esta string é utilizada para referenciar o nome do bloco quando for necessário gravar ou ler um arquivo de esquemático. Isto é necessário para que arquivos gerados por diferentes linguagens da ferramenta sejam compatíveis entre si.

5.12 Ícone do Bloco

Função: wxBitmap* vtbBlock_Teste::GetBitmap()

Esta função é chamada quando a ferramenta precisa obter uma referência para a imagem padrão do bloco. Esta função não mostra a imagem na interface gráfica.

Esta função deve retornar um ponteiro para uma imagem do tipo wxBitmap. Todos os ícones dos blocos não são lidos de arquivos externos; eles são inseridos diretamente no código fonte utilizando o formato XMP. Este formato é incluído na ferramenta através de uma diretiva #include, e então lido pelo construtor Bitmap(), que é parte da biblioteca wxWidgets.

Uma das opções para gerar uma imagem do tipo XMP, é utilizar o aplicativo ImageMagick (ImageMagick, 2004).

5.13 Desenho do Bloco

Função: void vtbBlock_Teste::Paint(wxDC& dc, const wxPoint& WindowPos)

Esta função é chamada quando a ferramenta precisa mostrar o bloco na interface gráfica, tanto na paleta quanto no esquemático.

A implementação desta função no bloco herdado é opcional. Esta função deve ser substituída quando o bloco necessita implementar um algoritmo diferenciado para sua representação na interface gráfica. O bloco Tanque, por exemplo, substitui esta função para representar graficamente o nível do tanque diretamente no esquemático.

A documentação da biblioteca wxWidgets possui maiores detalhes sobre as funções gráficas disponíveis.

5.14 Categoria do Bloco

Função: vtbBlockCategory vtbBlock_Teste::GetCategory()

Esta função é chamada pela classe Paleta durante a inicialização da ferramenta.

Esta função determina a qual grupo o bloco pertence. Isto determina em qual grupo o bloco será inserido na paleta de blocos, que é parte da interface gráfica. A ferramenta implementa 5 categorias de blocos:

Tabela 7 – Categorias dos blocos

Categoria	Descrição dos Blocos	Identificador
Diagnóstico	Blocos auxiliares e de medição	bcDiagnostic
Elétrica	Possuem apenas portas do tipo Elétrica	bcElectrical
Mecânica	Possuem portas do tipo Mecânica	bcMechanical
Térmica	Possuem portas do tipo Térmica	bcThermal
Hidráulica	Possuem portas do tipo Hidráulica	bcHydraulic

5.15 Número de Portas

Função: int vtbBlock_Teste::GetNumPorts()

Esta função é chamada quando a ferramenta precisa obter o número de portas do bloco, tanto durante a simulação quanto para a mostrar o bloco na interface gráfica e estabelecer conexões entre blocos.

Esta função deve retornar o número de portas que o bloco possui, tanto de entrada quanto de saída. Normalmente implementada diretamente na definição da classe.

5.16 Direção das Portas

Função: vtbPortDirection vtbBlock_Teste::GetPortDirection(int Idx)

Esta função é chamada quando a ferramenta precisa consultar se uma determinada porta é de entrada ou de saída, tanto durante a simulação quanto para a mostrar o bloco na interface gráfica e estabelecer conexões entre blocos.

Esta função determina quais portas são de entrada e quais portas são de saída. O algoritmo do simulador identifica qual porta do bloco está sendo consultada através do parâmetro Idx.

5.17 Lado das Portas

Função: vtbPortSide vtbBlock_Teste::GetPortSide(int Idx)

Esta função é chamada quando a ferramenta precisa consultar se uma determinada porta esta à direita, à esquerda, acima ou abaixo do bloco, de forma a mostrar o bloco na interface gráfica e estabelecer conexões entre blocos.

A implementação desta função no bloco herdado é opcional. Esta função deve ser substituída quando se deseja alterar o padrão de direcionamento das portas. Se esta função não for substituída, todas as portas de entradas serão colocadas à esquerda do bloco, e todas as portas de saída serão colocadas à direita do bloco. O algoritmo do simulador identifica qual porta do bloco está sendo consultada através do parâmetro Idx.

O bloco Amplificador de Erro contém um excelente exemplo de implementação desta função.

5.18 Posição das Portas

Função: wxPoint vtbBlock_Teste::GetPortPos(int Idx)

Esta função é chamada quando a ferramenta precisa obter a posição de uma porta, de forma a mostrar o bloco na interface gráfica e estabelecer conexões entre blocos.

A implementação desta função no bloco herdado é opcional. Esta função deve ser substituída quando se deseja alterar o posicionamento padrão das portas. O posicionamento padrão distribui as portas de forma eqüidistante em cada um dos lados do bloco. O algoritmo do simulador identifica qual porta do bloco está sendo consultada através do parâmetro Idx.

O bloco Tanque contém um exemplo de implementação desta função.

5.19 Tipo das Portas

Função: wxPoint vtbBlock_Teste::GetPortType(int Idx)

Esta função é chamada quando a ferramenta consultar o tipo de uma porta, de forma a validar as conexões entre blocos na interface gráfica.

Esta função identifica o tipo de cada uma das portas de um bloco, conforme definição dada na subseção 3.3.1. O algoritmo do simulador identifica qual porta do bloco está sendo consultada através do parâmetro Idx.

5.20 Inicialização do Bloco

Função: void vtbBlock_Teste::ResetBlock()

Esta função é chamada quando o algoritmo de simulação é reiniciado, ou quando o bloco é criado pela primeira vez.

A implementação desta função no bloco herdado é necessária apenas quando o bloco possuir variáveis que precisem ser inicializadas para um valor definido antes de iniciar a simulação. A implementação desta função na classe base zera todos os valores das portas de saída deste bloco. Uma implementação típica inicializa as variáveis exclusivas do bloco herdado, e então chama a função da classe base para inicialização das portas.

5.21 Tipo de Ordenação do Bloco

Função: bool vtbBlock_Teste::IsSorted()

Esta função é chamada pela classe Esquemático, durante a fase de ordenação dos blocos, para consultar se este bloco necessita ser ordenado ou não.

A implementação desta função no bloco herdado é obrigatória apenas quando o bloco for do tipo não ordenado, já que a função virtual na classe base retorna verdadeiro.

5.22 Propriedades do Bloco

Função: void vtbBlock_Teste::Properties()

Esta função é chamada quando o usuário aciona o comando Propriedades, para cada um dos blocos selecionados no esquemático.

A implementação desta função no bloco herdado é necessária apenas quando o bloco implementar uma caixa de diálogo para entrada dos parâmetros de configuração. Neste caso, a única responsabilidade desta função é criar a caixa de diálogo, caso esta ainda não exista, e mostrá-la para o usuário.

Caso o bloco não possua parâmetros que necessitem de configuração, esta função fica livre para qualquer outra aplicação, acionada pelo comando Propriedades (Ver subseção 5.1.2). Por exemplo, o bloco Chave de Entrada utiliza esta função para inverter a posição da chave quando a entrada de controle não está conectada.

5.23 Salvar e Carregar o Bloco

Função: void vtbBlock_Teste::Save()

Função: void vtbBlock_Teste::Load()

Estas funções são chamadas quando o usuário aciona o comando Salvar ou Abrir.

A implementação destas funções no bloco herdado é necessária apenas quando o bloco possuir parâmetros de configuração. Como regra geral, sempre que um bloco implementar uma caixa de diálogo, ele deverá implementar as funções de salvar e carregar o bloco, de forma a reter as configurações definidas pelo usuário.

O formato utilizado para os arquivos de esquemáticos da ferramenta é XML. A biblioteca wxWidgets contém o suporte necessário para criar e ler arquivos neste formato. Para maiores detalhes sobre o suporte XML podem ser encontrados na documentação da biblioteca wxWidgets.

Além de salvar e carregar as configurações do bloco, estas funções devem chamar as funções equivalentes na classe base; pois a classe base salva e carrega a posição do bloco e suas conexões.

O bloco Amplificador é um bom exemplo de implementação destas funções.

6 Recursos e Detalhes de Operação da Ferramenta

Este capítulo explica os recursos e os detalhes de operação da ferramenta.

6.1 Barra de Título e Menus

A barra de títulos contém o nome e versão do programa, assim como o nome do esquemático que está sendo utilizado. Caso ainda um nome ainda não tenha sido dado ao esquemático, a barra mostrará o nome padrão “SemNome”. Um asterisco à direita do nome indica que o esquemático contém alterações não salvas.

Figura 14 – Barra de Título e Menus

A seguir serão detalhados cada um dos menus, com suas funções. As teclas de atalho, ou seja, teclas que podem ser utilizadas para acessar funções de qualquer ponto sem utilizar o menu, são sempre indicadas ao lado dos itens de menu a que correspondem. A letra sublinhada nos menus indica a tecla de navegação.

6.1.1 Menu Arquivo

Item Novo

Fecha o esquemático atual e abre um novo esquemático vazio. Caso o esquemático contenha alterações, o programa pedirá uma confirmação do usuário antes de prosseguir.

Item Abrir

Fecha o esquemático atual e abre um esquemático gravado previamente. Caso o esquemático atual contenha alterações, o programa pedirá uma confirmação do usuário antes de prosseguir.

Item Salvar

Grava o esquemático utilizando o nome e localização atual. Caso o esquemático ainda não tenha um nome, o programa abre uma caixa de diálogo para perguntar o nome e localização do arquivo.

Item Salvar Como...

Grava o esquemático utilizando um novo nome e/ou localização. Neste caso, o programa sempre abre uma caixa de diálogo para perguntar o nome e localização do arquivo.

Item Título e Autor

Esta é uma funcionalidade que permite uma melhor identificação do esquemático e do autor, além do nome do arquivo. Estes valores são gravados juntamente com o esquemático.

Item Fechar

Fecha o programa. Caso o esquemático atual contenha alterações, o programa pedirá uma confirmação do usuário antes de prosseguir.

Arquivos Recentes

Lista dos arquivos mais recentes utilizados no programa, para facilitar o acesso rápido. Para abrir um arquivo, simplesmente selecione um item da lista. Caso o esquemático atual contenha alterações, o programa pedirá uma confirmação do usuário antes de prosseguir.

6.1.2 Menu Editar

Item Desfazer

Desfaz, começando pela última, as alterações feitas no esquemático pelo usuário.

Item Refazer

Refaz, começando pela última, as alterações desfeitas no esquemático pelo comando Desfazer.

Item Recortar

Retira os blocos selecionados do esquemático, e os coloca na área de transferência.

Item Copiar

Coloca os blocos selecionados na área de transferência, sem retirá-los do esquemático.

Item Colar

Coloca uma cópia dos blocos da área de transferência para o esquemático.

Item Limpar

Retira os blocos selecionados do esquemático, sem colocá-los na área de transferência.

Item Selecionar Tudo

Seleciona todos os blocos no esquemático

Item Selecionar nada

Limpa a seleção de blocos do esquemático

Item Inverter Seleção

Faz com que todos blocos não selecionados sejam selecionados, e vice-versa.

Item Propriedades

Abre a caixa de diálogo de propriedades do bloco para todos os blocos selecionados.

6.1.3 Menu Simulação

Item Iniciar

Inicia a simulação. A simulação irá continuar até que o usuário selecione Pausar ou Reiniciar.

Item Pausar

Para a simulação, mas mantém o tempo atual e o estado de todos os blocos inalterado.

Item Reiniciar

Caso a simulação esteja rodando, para a simulação. Zera o tempo atual e retorna todos os blocos aos seus estados iniciais.

Item Avançar Até...

Abre uma caixa de diálogo para entrada de um instante de tempo. A simulação será iniciada e irá continuar até que o tempo selecionado pelo usuário seja alcançado.

Submenu Selecionar Taxa

Este submenu expande em um grupo de opções para a taxa de avanço da simulação. Esta taxa representa a relação entre o tempo real para o usuário, e o tempo real para a simulação. Por exemplo, selecionando uma taxa de 10x, o tempo na simulação passará 10 vezes mais rápido do que para o usuário, ou seja, a cada segundo que passa para o usuário, 10 segundos se passam para a simulação.

Outra conseqüência da escolha da taxa de avanço, é a alteração do passo da simulação, ou seja, do intervalo de tempo utilizado nos algoritmos de cálculo numérico utilizados em cada um dos blocos. Seja T_A a taxa de avanço escolhida pelo usuário. O passo de simulação Δt será dado por:

$$\Delta t = \frac{T_A}{10000} \quad \textbf{(7)}$$

É necessário algum cuidado ao utilizar esta função, pois como a ferramenta não implementa um algoritmo de detecção de erros ou de passo adaptativo, é possível definir um passo que torne o resultado impreciso ou até mesmo instável, dependendo dos pólos existentes no sistema simulado.

6.1.4 Menu Opções

Item Cor de Fundo

Abre uma caixa de diálogo que permite ao usuário selecionar a cor de fundo para a área do esquemático na interface gráfica.

Item Cor de Fundo

Abre uma caixa de diálogo que permite ao usuário selecionar o espaçamento entre as linhas de grade, e a cada quantas linhas de grade será colocada uma linha de grade principal.

Item Mostrar Grade

Inverte a seleção do item Mostrar Grade, que determina se o usuário deseja que a grade seja plotada na área de esquemático da interface gráfica

Item Fixar na Grade

Inverte a seleção do item Fixar na Grade, que determina se o usuário deseja que o posicionamento dos blocos na área de esquemático da interface gráfica, só possa ser feito alinhando o centro do bloco às linhas de grade.

Item Selecionar Linguagem

Permite ao usuário trocar a linguagem utilizada na interface gráfica. Para esta alteração ter efeito, é necessário reiniciar o programa. Até o momento, estão disponíveis Português (Brasil) e Inglês.

6.1.5 Menu Ajuda

Item Sobre

Abre uma caixa de mensagem com informações de versão e autores da ferramenta, assim como o endereço eletrônico da página desta ferramenta na Internet.

6.2 Barra de Ferramentas

Os botões da barra de ferramentas são equivalentes à funções já descritas na subseção 5.1. Nesta subseção serão apenas descritos quais são as funções na barra de ferramentas e suas funções correspondentes no menu.

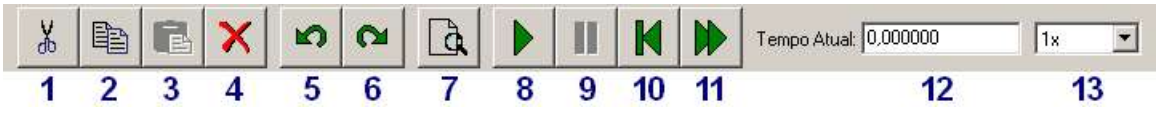

Figura 15 – Barra de Ferramentas

1) Recortar Blocos Selecionados – Item Recortar
2) Copiar Blocos Selecionados – Item Copiar
3) Colar Blocos – Item Colar
4) Limpar Blocos Selecionados – Item Limpar
5) Desfazer Mudanças – Item Desfazer
6) Refazer Mudanças – Item Refazer

7) Mudar Propriedades do Bloco – Item Propriedades
8) Iniciar Simulação – Item Iniciar
9) Pausar Simulação – Item Pausar
10) Reiniciar Simulação – Item Reiniciar
11) Avançar Simulação até... – Item Avançar até...
12) Este campo não está disponível no menu. Ele indica o tempo real (para a simulação) transcorrido desde o início da simulação.
13) Taxa de Simulação – Item Selecionar taxa

6.3 Paleta de Blocos

A paleta de blocos contém todos os blocos disponíveis para uso no esquemático. Como normalmente todos os blocos disponíveis não cabem na paleta simultaneamente, se utiliza a barra de rolagem a direita para que seja mostrada a zona de interesse na paleta.

6.4 Definindo um Esquemático

Para definir um novo esquemático a partir de um esquemático vazio, deve-se em primeiro lugar, escolher e posicionar alguns blocos no esquemático. Para isso, é necessário clicar e arrastar um bloco da paleta de blocos até a área do esquemático na interface gráfica. Para mudar o bloco de lugar, bastar clicar e arrastar o bloco diretamente na área do esquemático.

Em segundo lugar, são definidas as conexões entre os blocos. Para gerar uma nova conexão, clique em uma porta de saía e arraste o ponteiro do mouse até a porta de entrada à qual se deseja conectar. As ligações são feitas sempre iniciando na porta de saída para a porta

de entrada. Caso as portas sejam do mesmo tipo e não haja uma conexão anterior na porta de entrada (ver subseção 3.3.2), uma nova conexão será estabelecida.

Opcionalmente, alguns blocos permitem configurar parâmetros utilizados pelos seus algoritmos internos de simulação. Para configurar um bloco, deve-se dar um duplo clique sobre o bloco, ou selecionar o bloco e acionar o item Propriedades, no menu ou na barra de ferramentas.

6.5 Simulando o Esquemático

Para iniciar a simulação do esquemático, existem duas opções.

Caso não se queira definir um ponto no tempo onde a simulação deva parar, basta acionar o comando Iniciar, no menu ou na barra de ferramentas. A simulação será iniciada, e transcorrerá de acordo com taxa definida pelo submenu Selecionar Taxa ou no item Taxa de Simulação da barra de ferramentas.

Se por outro lado se deseja fixar o ponto onde a simulação deva pausar, utiliza-se o comando Avançar até..., disponível no menu e na barra de ferramentas. O programa irá abrir uma caixa de diálogo para que o usuário entre com o valor do tempo em que a simulação deve pausar. A simulação não será iniciada automaticamente; para isso deve ser utilizado o comando Iniciar. O usuário pode pausar e iniciar a simulação a qualquer momento, mesmo que o tempo limite definido para a simulação ainda não tenha transcorrido. Quando o tempo definido houver transcorrido, não será mais possível iniciar a simulação até que esta seja reiniciada ou um novo tempo limite (maior que o anterior) seja definido.

6.6 Observando a Saída dos Dados

Observar que a entrada é sempre em Volts. Para medir outros tipos de valores como temperatura, velocidade e pressão, é necessário o uso de transdutores específicos para cada tipo de porta, como o termômetro, o tacômetro e o medidor de pressão.

Esta subseção detalhará a operação de um dos blocos predefinidos pela ferramenta: o bloco Osciloscópio. Este bloco implementa um osciloscópio de 4 canais simplificado.

A ferramenta é capaz de plotar gráficos e gerar arquivos de Valores Separados por Vírgula (Comma Separated Values – CSV) utilizando o bloco Osciloscópio.

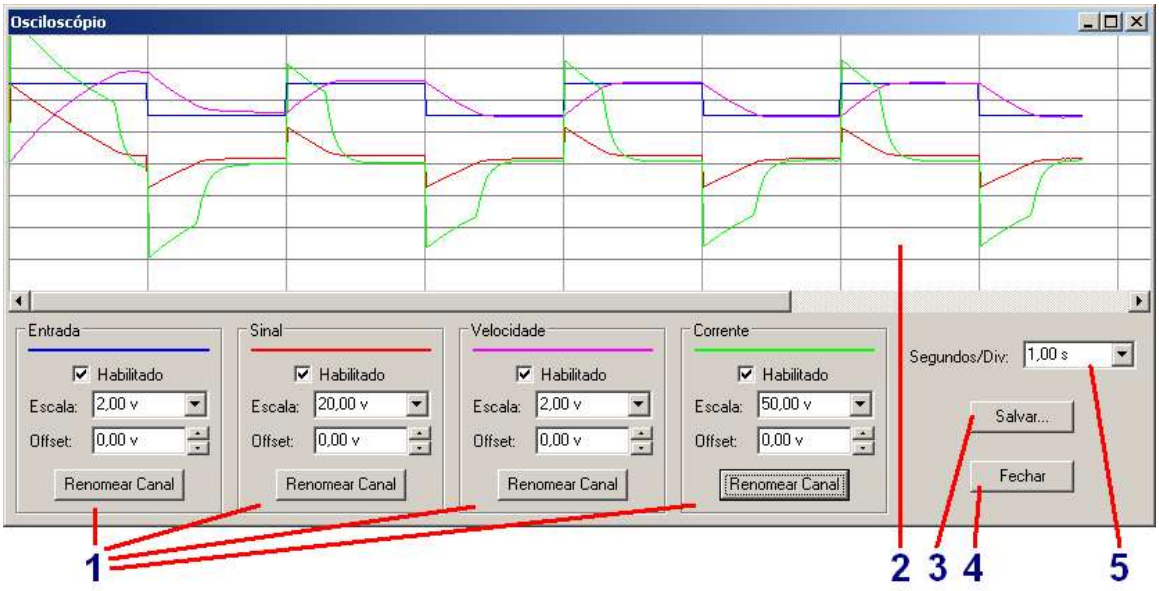

Figura 16 – Janela de propriedades do bloco Osciloscópio

Conforme a figura 16, a janela de propriedades do bloco Osciloscópio é dividida em quatro subgrupos para controle dos canais (1), área de plotagem dos gráficos (2), o botão para salvar arquivos em formato CSV, o botão para fechar esta janela (4), e o controle da escala de tempo por divisão horizontal (5).

O osciloscópio mantém sempre pelo menos as últimas 100 divisões horizontais na memória, ou seja, se a escala horizontal é de 100 milissegundos, um mínimo 10 segundos é mantido na memória. O controle da memória é feito da seguinte forma: quando o número de divisões armazenadas chega a 150, as primeiras 50 são descartadas e as últimas 100 são mantidas, e assim sucessivamente. Desta forma sempre há entre 100 e 150 divisões horizontais armazenadas na memória, para cada um dos quatro canais. Como a cada divisão horizontal o osciloscópio armazena 100 valores, o total será entre 10.000 e 15.000 amostras por canal, na memória.

Cabe ressaltar que o intervalo horizontal do osciloscópio não tem relação com a taxa de avanço da simulação. Por exemplo, se a taxa de avanço é de 1x, e o osciloscópio está definido para 1 segundo por divisão, teremos (ver equação 7) 100 passos de simulação para cada valor amostrado pelo osciloscópio.

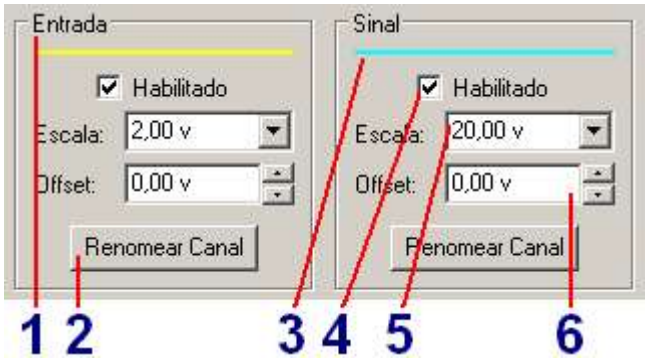

Figura 17 – Controles individuais dos canais do osciloscópio

A figura 17 mostra os controles para cada um dos canais do osciloscópio. Para facilitar a identificação, a cada um dos quatro canais pode ser atribuído um nome (1) utilizando-se o botão Renomear Canal (2). A cor utilizada para plotar o gráfico deste canal também é identificada (3), e o canal pode ser plotado ou não, dependendo do estado do controle Habilitado (4). A escala vertical, em Volts por divisão, é definida pelo controle Escala (5). O

offset é ajustado através do controle Offset (6), e em intervalos de 1/5 da divisão vertical, ou seja, se a escale é de 2 Volts/divisão, o offset é ajustado em incrementos de 0,4 Volts.

A figura 18 mostra um exemplo de geração de arquivo CSV, relativo aos primeiros 150 milissegundos do gráfico mostrado na figura 16.

```
Time,Entrada,Sinal,Velocidade,Corrente
0.000100,5.000372,0.000000,-0.000164,0.000000
0.010100,4.997799,49.930637,0.058113,193.921265
0.020100,4.995694,49.390244,0.159872,228.834030
0.030100,4.999554,48.768730,0.271629,233.015106
0.040000,4.998826,48.128948,0.381557,231.453827
0.050000,5.000479,47.470036,0.490799,228.815475
0.060000,5.000549,46.857426,0.599503,226.004486
0.070000,4.999701,46.204815,0.706337,223.190826
0.080000,5.000677,45.583992,0.812118,220.406143
0.090000,5.001285,44.971901,0.916361,217.656036
0.100000,4.997199,44.305317,1.018851,214.941193
0.110000,4.999605,43.673473,1.120660,212.261353
0.120000,5.000181,43.085426,1.220846,209.616119
0.130000,4.999566,42.468914,1.319033,207.005035
0.140000,4.998913,41.855598,1.417697,204.427689
0.150100,4.997648,41.258080,1.515079,201.858353
```

Figura 18 – Exemplo de geração de arquivo CSV

7 Biblioteca Pré-Definida de Blocos

Este capítulo explica os métodos utilizados na implementação de cada um dos blocos fornecidos na ferramenta.

7.1 Bloco Notas

Figura 19 – Bloco Notas

Este bloco não possui portas ou um algoritmo de simulação. Sua função é ser um painel de texto, onde o usuário pode colocar anotações e explicações a respeito do esquemático.

7.2 Bloco Gerador de Sinais

Figura 20 – Bloco Gerador de Sinais

O Gerador de sinais é um bloco ordenado, que pode ser utilizado para gerar ondas quadradas ou senoidais, e suas propriedades ajustáveis incluem de freqüência, amplitude, offset e ruído.

Seja A a amplitude do sinal, O_S o offset, f a freqüência e G(t) uma função randômica de ruído gaussiano (ver subseção 3.6). Então, para a onda senoidal:

$$y(t) = O_S + A \cdot sin(2 \cdot \pi \cdot f \cdot t) + G(t) \quad \textbf{(8)}$$

E para onda quadrada:

$$\begin{aligned} \forall sin(t) \geq 0 : y(t) &= O_S + A + G(t) \\ \forall sin(t) < 0 : y(t) &= O_S - A + G(t) \end{aligned} \quad \textbf{(9)}$$

7.3 Bloco Amplificador de Erro

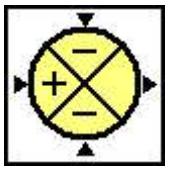

Figura 21 – Bloco Amplificador de Erro

O Amplificador de Erro é um bloco ordenado, porque sua propagação é instantânea (o bloco implementa uma função algébrica). Normalmente utilizado para implementar realimentações. Possui uma entrada positiva e duas entradas negativas. Este bloco não possui propriedades ajustáveis ou ruído. Seja $P_1(t)$ a entrada positiva e $N_1(t)$ e $N_2(t)$ as entradas negativas, no instante t. A saída é dada por:

$$y(t) = P_1(t) - N_1(t) - N_2(t) \quad \textbf{(10)}$$

7.4 Bloco Amplificador

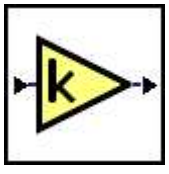

Figura 22 – Bloco Amplificador

O Amplificador é um bloco ordenado, com apenas uma entrada e uma saída. Este bloco possui propriedades ajustáveis para ganho, tensão máxima de saída, tensão mínima de saída, zona morta, histerese e ruído.

Neste bloco, o ruído é determinado pelo ajuste de sua propriedade para o ruído e também pelo ganho, ou seja, para um mesmo ajuste de ruído, um bloco com o dobro do ganho terá ruído com o dobro da amplitude.

7.5 Bloco Chave de Entrada

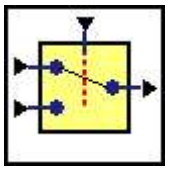

Figura 23 – Bloco Chave de Entrada

Este é um bloco ordenado, que implementa uma chave controlada eletricamente, de 1 pólo e 2 posições. O ponto de chaveamento é fixo em 0 Volts. Este bloco não possui propriedades, mas se a entrada de controle não estiver conectada e o usuário acionar o comando Propriedades para este bloco, a chave mudará de posição.

Sejam $V_1(t)$ e $V_2(t)$ as entradas nos terminais, e $V_C(t)$ a entrada de controle no instante t, então:

$$\begin{aligned} \forall V_C(t) \geq 0 : y(t) = V_1(t) \\ \forall V_C(t) < 0 : y(t) = V_2(t) \end{aligned} \qquad \textbf{(11)}$$

A Figura 23 mostra a chave ligada ao terminal V_1.

7.6 Bloco Função de Transferência

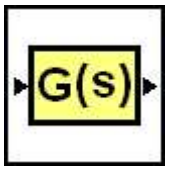

Figura 24 – Bloco Função de Transferência

Este é um bloco não ordenado, com uma entrada e uma saída, que implementa uma função de transferência no formato:

$$G(s) = \frac{Y(s)}{U(s)} \tag{12}$$

Através da caixa de diálogo de propriedades, o usuário define a ordem da função de transferência e cada um dos termos dos polinômios Y(s) e U(s). Estes polinômios são então transformados para uma representação matricial no espaço de estados, utilizando a forma canônica controlável (Ogata, 2003a), modificada para facilitar a implementação. Seja uma função de transferência em s:

$$\frac{Y(s)}{U(s)} = \frac{b_n \cdot s^n + b_{n-1} \cdot s^{n-1} + \cdots + b_1 \cdot s + b_0}{s^n + a_{n-1} \cdot s^{n-1} + \cdots + a_1 \cdot s + a_0} \tag{13}$$

Então a representação no espaço de estados, para uma função de transferência de ordem n, para uma entrada u e uma saída y, será dada por:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \vdots \\ \dot{x}_{n-1} \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} -a_{n-1} & -a_{n-2} & \cdots & -a_2 & -a_1 & -a_0 \\ 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & 0 \\ 0 & 0 & \cdots & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \cdot u \qquad \textbf{(14)}$$

$$y = \left[b_0 - a_0 \cdot b_n \mid b_1 - a_1 \cdot b_n \mid \cdots \mid b_{n-1} - a_{n-1} \cdot b_n \right] \times \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} + b_n \cdot u \qquad \textbf{(15)}$$

As equações 14 e 15 assumem que o coeficiente an é igual a 1. O algoritmo implementado na ferramenta difere do descrito acima, porque todos os termos são normalizados em função do termo an. Este passo foi omitido para simplificar a descrição do método utilizado. Este sistema de equações é então resolvido utilizando o algoritmo de Runge-Kutta multivariável de quarta ordem.

7.7 BLOCO PID

Figura 25 – Bloco PID

Este é um bloco não ordenado, com uma entrada e uma saída, que implementa um controlador PID no formato:

$$G_C(s) = K_P \cdot \left(1 + \frac{1}{T_I \cdot s} + \frac{T_D \cdot s}{s + P_D}\right) \quad \textbf{(16)}$$

Onde K_P é o ganho, T_I é o tempo integral e T_D é o tempo derivativo. P_D é o pólo associado ao controle derivativo, que foi tornado explícito na implementação, de forma a dar ao usuário a oportunidade de ajustar o seu valor.

As funções de transferência se tornam então:

$$G_C(s) = K_P \quad \textbf{(17)}$$

$$G_C(s) = \frac{K_P.s + \frac{K_P}{T_I}}{s} \quad \textbf{(18)}$$

$$G_C(s) = \frac{K_P.(P_D.T_D + 1)s + K_P.P_D}{s + P_D} \quad \textbf{(19)}$$

$$G_C(s) = \frac{K_P(P_D.T_D + 1).s^2 + K_P(P_D + \frac{1}{T_I}).s + \frac{K_P.P_D}{T_I}}{s^2 + P_D.s} \quad \textbf{(20)}$$

Onde as equações 17, 18, 19 e 20 representam os controladores tipo P, PI, PD e PID, respectivamente.

Na implementação da ferramenta, cada um dos estágios do controlador (P, I e D) é computado separadamente, utilizando o algoritmo numérico implementado na ferramenta.

7.8 Bloco Motor

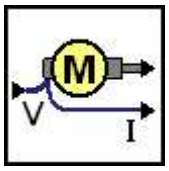

Figura 26 – Bloco Motor

Este é um bloco não ordenado, com uma entrada de tensão, uma saída tipo mecânica e uma saída de tensão indicativa da corrente consumida no motor. A grandeza da saída de monitoração da corrente é igual a 1 Volt/Ampère consumido pelo motor.

A caixa de diálogo de propriedades permite o ajuste das seguintes características do motor:

Tabela 8 – Características do bloco Motor

Característica	Unidade
Constante do Motor	$\frac{Nm}{A}$ ou $\frac{V \cdot s}{rad}$
Inércia	$\frac{Nm \cdot s^2}{rad}$
Resistência	Ohm
Indutância	mH
Atrito	$\frac{Nm \cdot s}{rad}$

Em unidades SI, a constante de torque do motor K_T e a constante da força contra-eletromotriz K_E são numericamente iguais, e definidas pela constante do motor, conforme a tabela 8. Nas equações abaixo, os termos K_T e K_E são mantidos separados para maior clareza.

Este bloco implementa um motor DC utilizando um sistema de dois pólos, onde as variáveis de estado são a corrente e a velocidade angular (Ogata, 2003b).

O torque se relaciona com a corrente segundo a equação:

$$T = K_T \cdot i \tag{21}$$

Onde K_T é a constante de torque (ou de armadura) do motor.

A força contra-eletromotriz é calculada a partir da velocidade angular do rotor:

$$e = K_e \cdot \dot{\theta} \tag{22}$$

Aplicando a Lei de Newton e a Lei de Kirchoff às equações 21 e 22. podemos obter:

$$\begin{aligned} J \cdot \ddot{\theta} + b\dot{\theta} &= K_T \cdot i - T_C \\ L\frac{\partial i}{\partial t} + R \cdot i &= V - K_E \cdot \dot{\theta} \end{aligned} \tag{23}$$

Onde T_C é o torque de carga. Escolhendo velocidade angular e corrente como as variáveis de estado e podemos escrever as equações acima no espaço de estados:

$$\begin{aligned} &\omega = \dot{\theta} \\ &\frac{d}{dt}\begin{bmatrix} \omega \\ i \end{bmatrix} = \begin{bmatrix} -\frac{B}{J} & \frac{K_T}{J} \\ -\frac{K_E}{L} & -\frac{R}{L} \end{bmatrix} \times \begin{bmatrix} \omega \\ i \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{L} \end{bmatrix} \cdot V \end{aligned} \tag{24}$$

A equação 24 é a forma mais comumente encontrada na literatura, mas esta forma ignora a parcela do torque de carga. Como portas do tipo mecânica também implementam uma entrada de torque, podemos adicionar uma segunda entrada, de torque de carga T_C, da seguinte forma:

$$\frac{d}{dt}\begin{bmatrix}\omega \\ i\end{bmatrix}=\begin{bmatrix}-\frac{B}{J} & \frac{K_T}{J} \\ -\frac{K_E}{L} & -\frac{R}{L}\end{bmatrix}\times\begin{bmatrix}\omega \\ i\end{bmatrix}+\begin{bmatrix}-\frac{1}{J} & 0 \\ 0 & \frac{1}{L}\end{bmatrix}\times\begin{bmatrix}T_C \\ V\end{bmatrix} \quad \textbf{(25)}$$

A equação 25 é calculada numericamente pelo programa, em função do tempo e das entradas. As saídas são obtidas diretamente das variáveis de estado, apenas convertendo rad/s em RPM.

7.9 Bloco Caixa de Redução

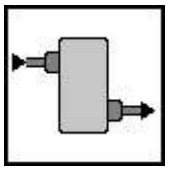

Figura 27 – Bloco Caixa de Redução

A Caixa de Redução é um bloco ordenado, com uma entrada e uma saída do tipo mecânica. Este bloco possui propriedades ajustáveis para a relação de velocidade entre a entrada e a saída N, a inércia total (Nm/rad/s^2) e o atrito dinâmico (Nm/rad/s).

Os valores de inércia e atrito são calculados utilizando a velocidade de entrada na caixa de redução. Uma porta tipo mecânica tem 3 grandezas associadas: velocidade, inércia e torque (ver tabela 3). Seja ω_E a velocidade de entrada, ω_S a velocidade de saída, T_E o torque na entrada, T_S o torque na saída, J_E a inércia equivalente na entrada, J_S a inércia na saída, J_R a inércia da caixa de redução e B_R o atrito dinâmico da caixa de redução. Considerando que a velocidade é propagada da entrada para a saída do bloco, e inércia e torque são propagados da saída para a entrada, as equações para este bloco ficam então:

$$\omega_S = \frac{\omega_E}{N} \tag{26}$$

$$T_E = \omega_E \cdot B_R + \frac{T_S}{N} \tag{27}$$

$$J_E = J_R + \frac{J_S}{N} \tag{28}$$

7.10 Bloco Carga Mecânica

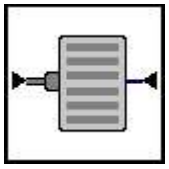

Figura 28 – Bloco Carga Mecânica

A Carga Mecânica é um bloco ordenado, com uma entrada do tipo mecânica e uma entrada tipo elétrica. Efeito de um freio acionado eletricamente Este bloco possui propriedades ajustáveis de inércia total J (Nm/rad/s^2), atrito dinâmico mínimo B_{MIN} (Nm/rad/s) e atrito dinâmico máximo B_{MAX} (Nm/rad/s). A variação do atrito dinâmico entre o mínimo e o máximo é controlada pela tensão na entrada elétrica V_C na faixa entre 0 e 10 Volts:

$$B_C = B_{MIN} + \frac{V_C \cdot \left(B_{MAX} - B_{MIN}\right)}{10} \tag{29}$$

Seja ω_C a velocidade, T_C o torque e B_C o atrito dinâmico; a equação para este bloco fica então:

$$T_C = \omega_C \cdot B_C \quad \textbf{(30)}$$

O valor da inércia é colocado diretamente na porta de entrada.

7.11 Bloco Tacômetro

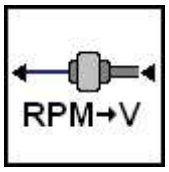

Figura 29 – Bloco Tacômetro

O Tacômetro é um bloco ordenado, com uma entrada do tipo mecânica e uma saída elétrica. Este bloco possui propriedades ajustáveis de ganho (mV/RPM) e nível de ruído.

Como o ganho K_V é dado em milivolts, a tensão de saída E_S, em função da velocidade V_{RPM}, é dada por:

$$E_S = \frac{V_{RPM} \cdot K_V}{1000} + G \quad \textbf{(31)}$$

Onde G é o valor randômico do ruído gaussiano (ver subseção 3.6) adicionado à saída.

7.12 Bloco Tanque

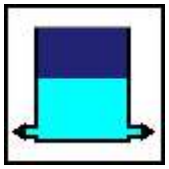

Figura 30 – Bloco Tanque

Este é um bloco não ordenado, com duas saídas do tipo hidráulica. Este bloco possui propriedades ajustáveis de altura do tanque (m), seção transversal (m^2), nível inicial (m), nível atual (m), temperatura inicial (°C) e temperatura atual (°C). Retirar os atuais. O nível inicial e a temperatura inicial são os valores assumidos pelo bloco Tanque no início da simulação.

As duas saídas se comportam de forma idêntica, e seus efeitos são combinados para o cálculo do estado do tanque. Seja L_T o nível do tanque, então a pressão P_S (kPa) é dada por:

$$P_S = L_T \cdot g \tag{32}$$

Onde g é a aceleração da gravidade: 9.81 m/s^2. A densidade do líquido é fixada pelo algoritmo em $1g/cm^3$.

Todos os blocos que utilizam portas do tipo hidráulico precisam implementar um algoritmo para avaliar corretamente a temperatura em função dos fluxos de entrada e saída (ver subseção 3..3.1 – porta tipo hidráulica).

Os fluxos são dados em l/s (litros por segundo). Cada porta possui as seguintes grandezas: fluxo Q, fluxo de entrada Q_E, temperatura no tanque T_T, temperatura de entrada T_E e pressão P_S. Seja C a seção transversal do tanque e Δt o passo da simulação, então e equação para a variação de nível e temperatura para cada passo da simulação fica sendo:

$$\frac{\partial L_T}{\partial t} = \frac{(Q + Q_E)}{C \cdot 1000} \partial t \tag{33}$$

$$\frac{\partial T_T}{\partial t} = \frac{(T_E - T_T) \cdot Q_E}{1000 \cdot C \cdot L_T} \partial t \tag{34}$$

As equações acima descrevem o cálculo para uma das duas portas para manter a clareza. A versão implementada pelo bloco leva em consideração as duas portas de saída hidráulica.

7.13 BLOCO BOMBA

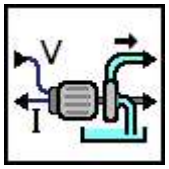

Figura 31 – Bloco Bomba

Este é um bloco não ordenado, com uma entrada de tensão, uma saída tipo mecânica, uma saída de tensão indicativa da corrente consumida no motor e uma porta hidráulica. A grandeza da saída de monitoração da corrente é igual a 1 Volt/Ampère consumido pelo motor.

Este bloco foi implementado a partir do bloco de simulação de motores. Consulte a subseção 6.8 para os detalhes de implementação do motor da bomba.

Conectada ao eixo do motor, foi implementada uma bomba centrífuga, que se comporta como uma carga, através de um torque aplicado ao eixo.

A pressão máxima de saída da bomba é dada por (Koppel, 2004):

$$P_{Max} = K_P \cdot \omega^2 \quad \textbf{(35)}$$

O fluxo máximo é dado por:

$$Q_{Max} = K_Q \cdot \omega \quad \textbf{(36)}$$

A curva de perda de pressão em função do fluxo de saída de uma bomba centrífuga varia de acordo com o tipo de bomba. Neste bloco será utilizada uma bomba com perda quadrática de pressão em função do fluxo. Então a pressão de saída será dada por:

$$P = P_{Max} \cdot \sqrt{1 - \left(\frac{Q}{Q_{Max}} \right)^2} \quad \textbf{(37)}$$

A eficiência de uma bomba centrífuga muda em função da velocidade de operação, fluxo de saída e pressão de saída, de forma não linear. Para a implementação deste bloco, estimamos a eficiência hidráulica da bomba como sendo 80%. Para calcular a potência consumida pela bomba, utilizamos também a pressão de saída e o fluxo. A potência no eixo é dada pelo torque multiplicado pela velocidade angular, então:

$$T \cdot \omega = \frac{P \cdot Q}{\frac{80\%}{100\%}} \Rightarrow T = \frac{P \cdot Q}{0.8 \cdot \omega} \quad \textbf{(38)}$$

Este torque é então utilizado como carga na segunda variável de entrada na equação do motor. Sendo assim, embora a eficiência hidráulica da bomba seja 80%, a eficiência total leverá em consideração as perdas do motor elétrico utilizado para esta bomba.

7.14 Bloco Válvula

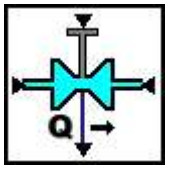

Figura 32 – Bloco Válvula

A Válvula é um bloco ordenado, com duas entradas tipo hidráulica, uma saída tipo elétrica para monitoração do fluxo através da válvula e uma entrada tipo elétrica para controle de abertura da válvula. Além de uma válvula controlada eletricamente, este bloco implementa toda a tubulação que liga a válvula à(s) porta(s) de saída hidráulica, em ambos os lados.

A caixa de diálogo de propriedades permite o ajuste das seguintes características da válvula:

Tabela 9 – Parâmetros do bloco Válvula

Característica	Unidade
Fator de Fluxo à 5V	$\frac{l}{s \cdot \sqrt{kPa}}$
Diâmetro do Cano	mm
Comprimento do Cano	m
Ganho do Fluxo	$\frac{V \cdot s}{l}$

A tensão de controle V_C da válvula é fixa entre 0 e 10 Volts. Tensões negativas ou acima de 10 Volts são consideradas como válvula totalmente fechada ou totalmente aberta, respectivamente.

O ganho do fluxo G_Q é utilizado como o ganho da saída V_Q de monitoração do fluxo Q, segundo a equação:

$$V_Q = Q \cdot G_Q \tag{39}$$

O cálculo da vazão em função do tempo leva em consideração o coeficiente de vazão da válvula, as dimensões da tubulação e a inércia do fluido. O escoamento é laminar e a densidade do fluido é considerada como 1 g/cm3.

Seja P_D a pressão diferencial na válvula, P_V a perda de pressão na válvula, P_B a perda de pressão de Bernoulli e J o coeficiente de variação de fluxo pela pressão. Assim, a variação do fluxo pelo tempo em l/s^2 é dada por:

$$\frac{\partial Q}{\partial t} = \frac{P_D - P_V - P_B}{J} \quad \textbf{(40)}$$

O coeficiente de variação de fluxo do fluido em função da pressão dentro da tubulação (J) em $kPa.s^2/l$, é dado por:

$$J = \frac{10 \cdot L}{C} \quad \textbf{(41)}$$

Onde L é o comprimento da tubulação em metros e C é a área da seção transversal em cm^2. A pressão diferencial P_D em kPa é dada pela diferença de pressão entre as conexões da válvula:

$$P_D = P_1 - P_2 \quad \textbf{(42)}$$

Este modelo para simulação de válvulas não leva em consideração as limitações mecânicas do sistema, como fluxo máximo e pressão máxima. Assim sendo precisamos de um método para descrever a autoridade da válvula sem recorrer aos seus valores nominais. O método encontrado para tanto foi especificar o fator de fluxo K_V da válvula para uma abertura de 50%, ou seja, quando a tensão de controle for 5 Volts.

Para otimizar o controle ao longo da faixa de operação da válvula, e permitir que ela seja operada totalmente aberta (V_C = 10 Volts) e totalmente fechada (V_C = 0 Volts), a tensão de controle V_C modifica o fator de fluxo segundo a seguinte equação:

$$K_C = \left(\frac{V_C}{10 - V_C} \right) \cdot K_V \quad \textbf{(43)}$$

Onde K_V é o fator de fluxo da válvula para uma abertura de 50%, e K_C é o fator de fluxo em função da tensão de controle. Assim, a perda de pressão na válvula será dada por:

$$P_V = \left(\frac{Q}{K_C} \right)^2 \quad \textbf{(44)}$$

É necessária a utilização de um algoritmo separado para o cálculo da perda de pressão para as condições de válvula totalmente aberta e totalmente fechada, de forma a evitar divisões por zero durante o cálculo.

Para válvula totalmente aberta, o algoritmo assume uma perda de pressão na válvula P_V igual a zero. Nesta situação, a perda de pressão é estimada unicamente pela perda de pressão por fricção no cano.

De forma a estimar a perda de pressão ao longo da tubulação, assumiremos que o fluxo é turbulento, e utilizaremos a equação de Hazen-Williams (ToolBox, 2004a):

$$Pc = \frac{23.342 \cdot 10^7 \cdot \left(\frac{100}{C_R} \right)^{1.852} \cdot Q^{1.852} \cdot L}{D^{4.8655}} \quad \textbf{(45)}$$

C é a constante de rugosidade de Hazen-Williams. O algoritmo de simulação utiliza C_R igual a 150, relativa à rugosidade de um cano de polivinil cloreto (PVC).

A partir da equação de Bernoulli (ToolBox, 2004b), para um fluido de densidade 1 g/cm^3, podemos calcular a perda e pressão em função da velocidade do fluxo:

$$P_B = \frac{V^2}{2} \tag{46}$$

A velocidade pode ser calculada a partir do fluxo e da seção transversal da válvula, e é dada por:

$$V = \frac{10 \cdot Q}{C} \tag{47}$$

Substituindo as equações 41, 42, 43, 44, 45, 46 e 47 na equação 40, obtemos:

$$\frac{\partial Q}{\partial t} = \frac{P_1 - P_2 - \left(\frac{Q}{K_V}\right)^2 - \frac{\left(\frac{10 \cdot Q}{C}\right)^2}{2} - \frac{23.342 \cdot 10^7 \cdot \left(\frac{100}{C_R}\right)^{1.852} \cdot Q^{1.852} \cdot L}{D^{4.8655}}}{\frac{10 \cdot L}{C}} \tag{48}$$

Esta equação diferencial ordinária é resolvida numericamente pelo algoritmo implementado no programa.

7.15 Bloco Medidor de Pressão

Figura 33 – Bloco Medidor de Pressão

O Medidor de Pressão é um bloco ordenado, com uma entrada do tipo hidráulica e uma saída elétrica. Este bloco possui propriedades ajustáveis de ganho (V/kPa) e nível de ruído.

A partir do ganho K_P e da pressão de entrada P_E, a tensão de saída V_S é dada por:

$$V_S = P_E \cdot K_P + G \tag{49}$$

Onde G é o valor randômico do ruído gaussiano (ver subseção 3.6) adicionado à saída.

7.16 Bloco Forno

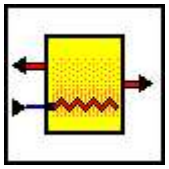

Figura 34 – Bloco Forno

Este é um bloco não ordenado, com duas saídas do tipo térmica e uma entrada elétrica da resistência de aquecimento. O forno é modelado por duas variáveis de estado, a temperatura da resistência de aquecimento T_R e a temperatura do forno T_F. Este bloco possui propriedades ajustáveis de valor da resistência R (Ohms), capacidade térmica da resistência C_R (cal/°C), coeficiente de condução térmica entre a resistência e o forno K_R (cal/°C/s), capacidade térmica do forno C_F (cal/°C) e coeficiente de perda do forno K_F (cal/°C/s).

Seja q o fluxo de calor nas portas térmicas, V a tensão na entrada, e considerando que 1 caloria é igual a 4,184 Joules, as equações diferenciais das variáveis de estado são (Ogata, 2003):

$$\dot{T}_R = \frac{V^2}{R \cdot C_R \cdot 4.184} - \frac{(T_R - T_F) \cdot K_R}{C_R} \tag{50}$$

$$\dot{T}_F = \frac{(T_R - T_F) \cdot K_R}{C_F} - \frac{q}{C_F} \tag{51}$$

Escrevendo na forma matricial no espaço de estados:

$$\begin{bmatrix} \dot{T}_R \\ \dot{T}_F \end{bmatrix} = \begin{bmatrix} -\frac{K_R}{C_R} & \frac{K_R}{C_R} \\ \frac{K_R}{C_F} & -\frac{K_R + K_F}{C_F} \end{bmatrix} \times \begin{bmatrix} T_R \\ T_F \end{bmatrix} + \begin{bmatrix} \frac{1}{R \cdot C_R \cdot 4.184} & 0 \\ 0 & -\frac{1}{C_F} \end{bmatrix} \times \begin{bmatrix} V^2 \\ q - 20 \cdot K_F \end{bmatrix} \quad \textbf{(52)}$$

Na implementação deste bloco, as perdas de calor do forno são calculadas levando em consideração uma temperatura ambiente de 20°C. Esta perda está inserida na equação através dos dois termos em K_F. Este sistema de equações diferenciais é resolvido numericamente pelo algoritmo Runge-Kutta de quarta ordem.

7.17 Bloco Dissipador de Calor

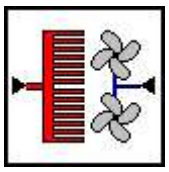

Figura 35 – Bloco Dissipador de Calor

O Dissipador de Calor é um bloco ordenado, com uma entrada do tipo térmica e uma entrada tipo elétrica. Este bloco possui propriedades ajustáveis de capacidade térmica C_D (cal/°C), coeficiente de condução térmica na entrada K_E (cal/°C/s), coeficiente de dissipação térmica mínimo K_{MIN} (cal/°C/s) e coeficiente de dissipação térmica máximo K_{MAX} (cal/°C/s). A variação do coeficiente de dissipação térmica K_D entre o mínimo e o máximo é controlado pela tensão na entrada elétrica V_C na faixa entre 0 e 10 Volts:

$$K_D = K_{MIN} + \frac{V_C \cdot (K_{MAX} - K_{MIN})}{10} \quad \textbf{(53)}$$

Seja T_E a temperatura na entrada térmica e T_D a temperatura do dissipador. O fluxo térmico q é dado por:

$$q = (T_E - T_D) \cdot K_C \quad \textbf{(54)}$$

Considerando a temperatura ambiente como fixa em 20 °C, a equação diferencial da temperatura do dissipador pode ser escrita como:

$$T_D = \frac{q - (T_D - 20) \cdot K_D}{C_D} \quad \textbf{(55)}$$

O valor do fluxo térmico q é colocado diretamente na porta de entrada térmica.

7.18 Bloco Termômetro

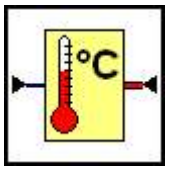

Figura 36 – Bloco Termômetro

O Termômetro é um bloco ordenado, com uma entrada do tipo térmica e uma saída elétrica. Este bloco possui propriedades ajustáveis de ganho (mV/°C) e nível de ruído.

Como o ganho K_T é dado em milivolts, a tensão de saída V_S, em função da temperatura T_E, é dada por:

$$V_S = \frac{T_E \cdot K_T}{1000} + G \quad \textbf{(56)}$$

Onde G é o valor randômico do ruído gaussiano (ver subseção 3.6) adicionado à saída.

7.19 Bloco Multímetro

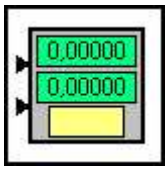

Figura 37 – Bloco Multímetro

O Multímetro é um bloco ordenado, com duas entradas elétricas. Este bloco não possui propriedades, e é utilizado para monitorar a tensão em suas entradas e mostrar o valor diretamente no esquemático.

7.20 Bloco Osciloscópio

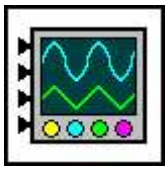

Figura 38 – Bloco Osciloscópio

O Osciloscópio é um bloco ordenado, com quatro entradas elétricas. Detalhes a respeito da utilização deste bloco são dados na subseção 5.6.

8 Exemplos de Esquemático

8.1 Oscilador

Este bloco mostra um exemplo onde um oscilador é iniciado em função do ruído de um amplificador de ganho 10, com saída limitada entre –3 Volts e +3 Volts.

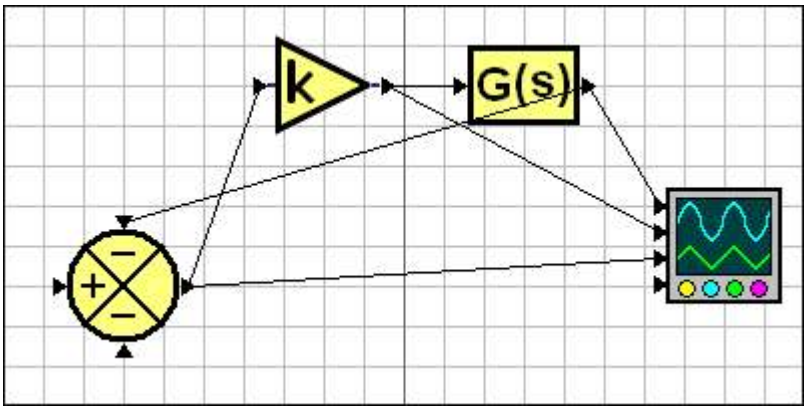

Figura 39– Exemplo de um amplificador com partida própria

A função de transferência implementada no circuito da Figura 39, é dada por:

$$G(s) = \frac{1000000}{s^3 + 300 \cdot s^2 + 30000 \cdot s + 1000000} \quad \textbf{(57)}$$

Esta função de transferência gera um pólo triplo em 100 radianos, ou 15,9 Hertz. O gráfico da figura 40 mostra uma freqüência de oscilação de 27,51 Hertz. Calculando a freqüência para um ângulo de fase de 180° a partir da função de transferência obtemos 27,566 Hertz. A freqüência de oscilação foi encontrada a partir do arquivo CSV exportado pelo osciloscópio. O erro entre a freqüência calculada e a freqüência simulada é de 0,2%.

Outro experimento realizado para validar o algoritmo foi determinar algebricamente o ganho necessário no amplificador para obter um ganho unitário com atraso de 180°; este

ganho é de 8. Após a partida do oscilador, o ganho foi alterado para 8, e o gráfico obtido está na figura 41. Não houve alteração mensurável de amplitude ao longo do gráfico. Para períodos maiores de tempo, foi determinado que o ganho que mantém o oscilador estabilizado é de 7,996. Isto representa um erro de 0,05% em relação ao valor calculado.

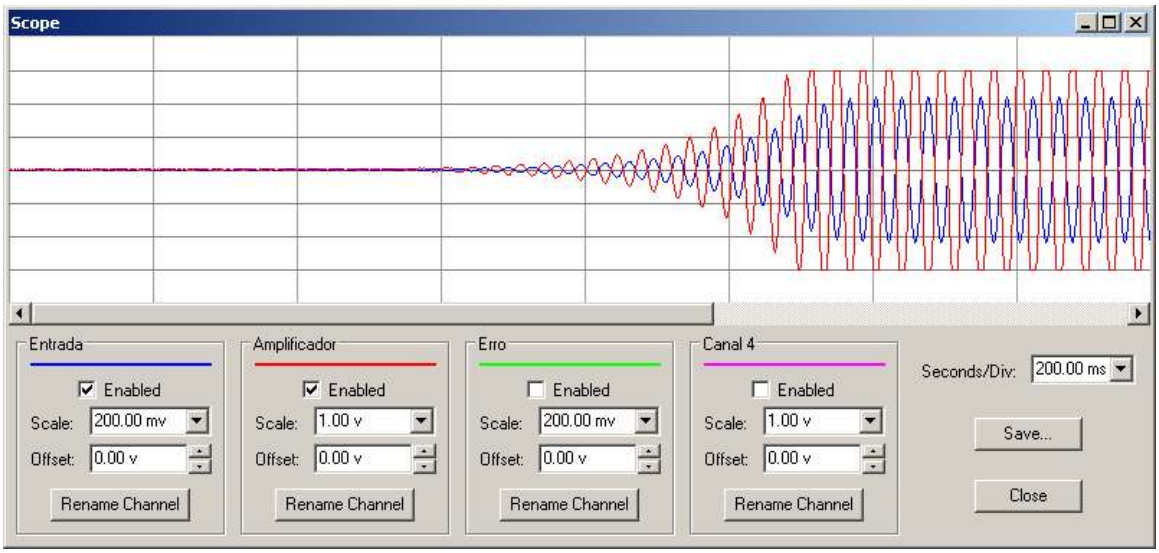

Figura 40 – Sinais de saída do oscilador com partida própria

Figura 41 – Oscilador com ganho unitário para 180°

8.2 Controle de Velocidade do Motor

Este exemplo mostra o controle de velocidade em um motor. A realimentação é feita através de um tacômetro com saída de 5 mV/RPM. O controle foi implementado com um controlador PI de ganho 10 e tempo integral 0,2 segundos. O amplificador de potência tem ganho 15 e tensão máxima de saída de 100 V.

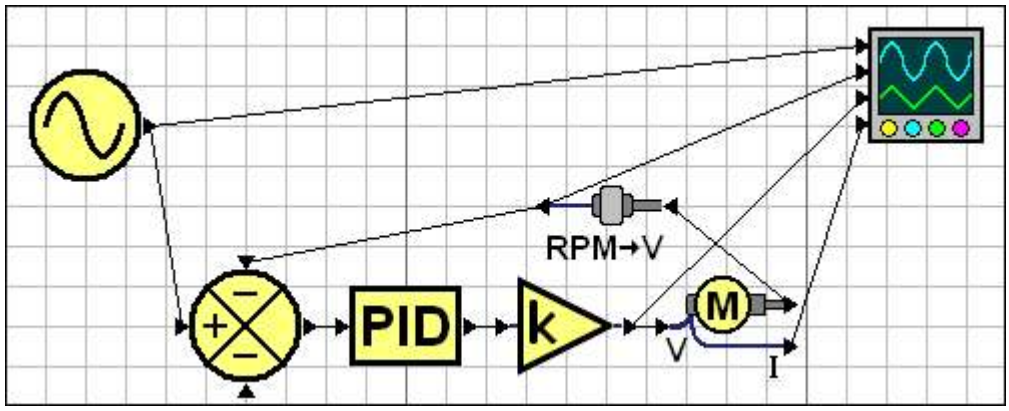

Figura 42 – Circuito para controle de velocidade de um motor

Fazendo o set point variar entre 0,4 Volts e 0,6 Volts, ou seja, com a velocidade de 80 e 120 RPM, temos o gráfico da figura 43.

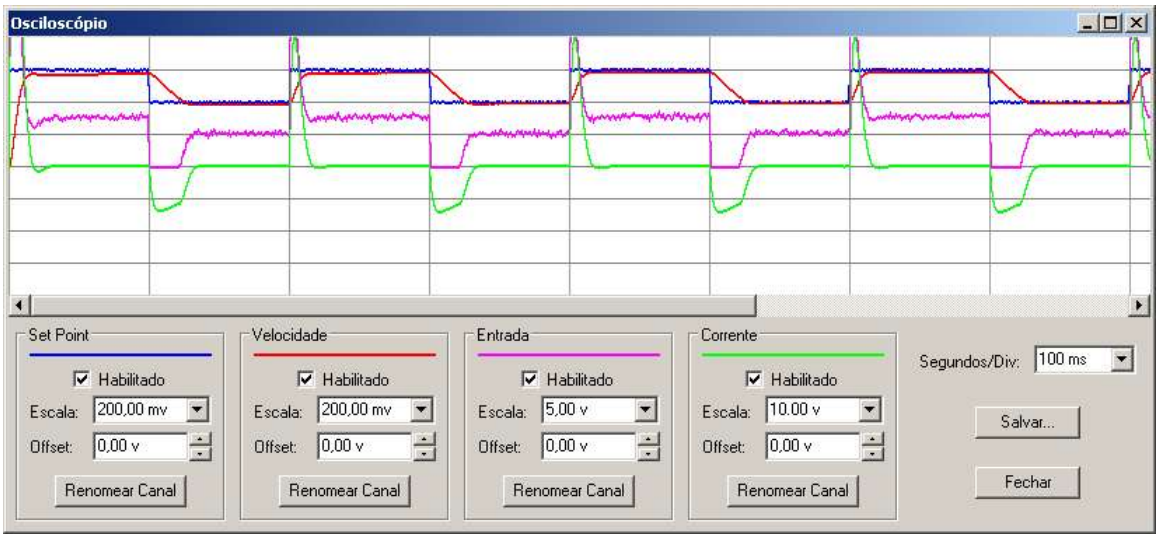

Figura 43 – Motor controlado entre 80 e 120 RPM

Multiplicando o set point por 10, e modificando a escala do osciloscópio da mesma forma, temos o resultado da figura 44.

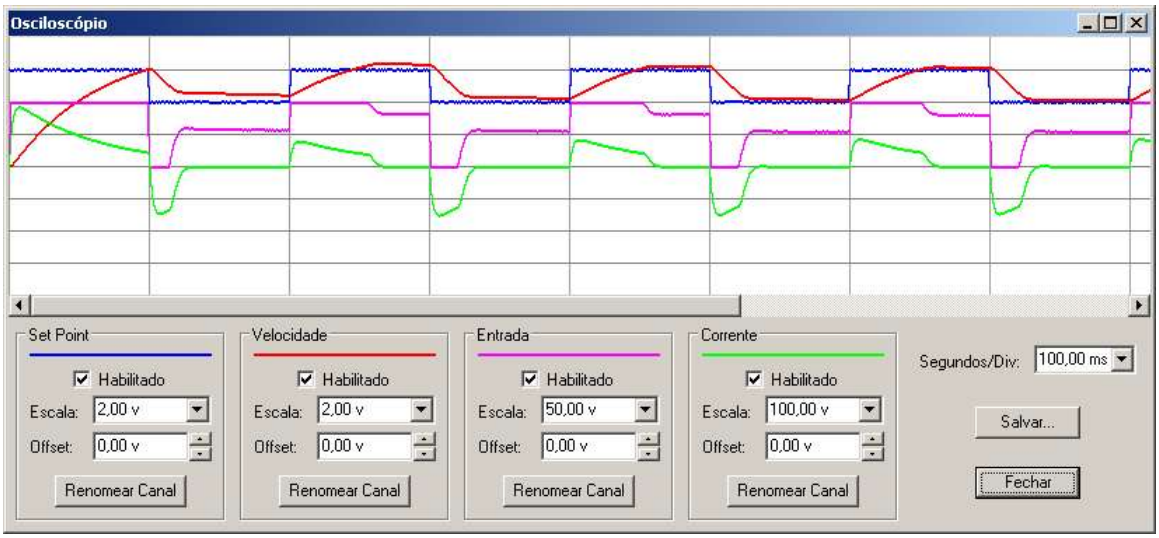

Figura 44 – Motor controlado entre 800 e 1200 RPM

O ruído no primeiro caso é bem mais visível porque as amplitudes, assim como as escalas, são dez vezes menores. Desconsiderando a diferença na escala, a característica da corrente na figura 43 e na figura 44 é a mesma para saltos negativos no set point, pois em

ambos os casos a Entrada não baixa de zero Volts. Já para saltos positivos de set point, a forma de onda da corrente muda completamente, pois a aceleração do motor na figura 44 é limitada pela saturação do amplificador de potência (Entrada) em 100 Volts.

8.3 Controle de Temperatura de um Forno

Este exemplo mostra o controle de temperatura em um forno. A realimentação é feita através de um termômetro com saída de 20 mV/°C. O controle foi implementado com um controlador PI de ganho 10 e tempo integral 25000 segundos. O amplificador de potência tem ganho 10 e tensão máxima de saída de 100 V.

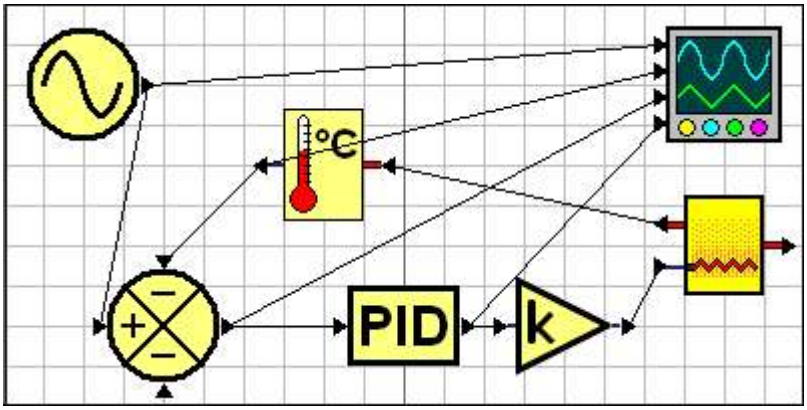

Figura 45 – Circuito de controle de temperatura em um forno

A simulação foi acelerada em 1000 vezes, de forma que uma hora passava a cada 3,6 segundos. A entrada mostra o ponto de ajuste variando entre 500 °C e 600 °C, ao longo de 11 horas e 15 minutos, ou seja, 40,5 segundos de simulação.

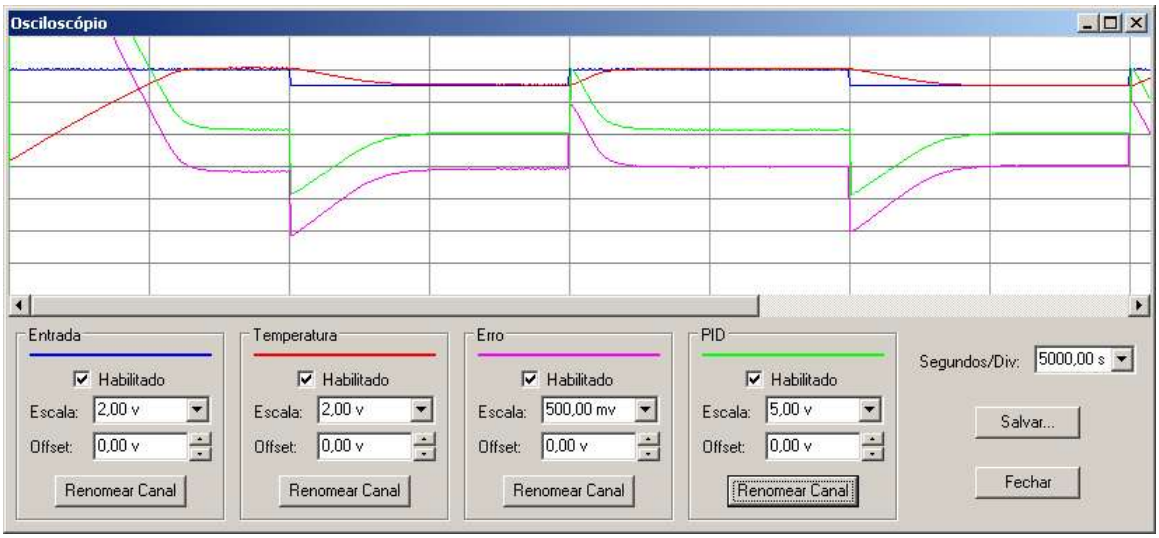

Figura 46 – Sinais de saída para o forno e o controle

Pontos de interesse incluem o fato de que o overshoot na temperatura só ocorre para o salto inicial de 20 °C para 600 °C, e não no salto de 500 °C para 600 °C. Este fato ocorre devido à saturação do amplificador de potência, que opera na capacidade máxima até que a tensão de saída do PID baixe a menos de 10 Volts, 5700 segundos após o início da simulação. Isto faz com que o controlador PID permaneça com um sinal de erro em sua entrada por um tempo mais elevado, causando o windup da parcela integral do PID.

Outro fato a ser observado é que a estabilização para variações positivas do ponto de ajuste é mais rápida do que variações negativas.

9 Conclusões e Perspectivas

Este trabalho analisou as ferramentas de simulação disponíveis atualmente e propôs uma ferramenta complementar, cujo objetivo é auxiliar alunos de engenharia na compreensão e fixação dos conceitos de sistemas de controle, aproximando a teoria e modelos matemáticos da realidade dos dispositivos físicos.

O algoritmo escolhido foi o de fluxo de sinais, com casos específicos resolvidos através do algoritmo de conservação. Esta escolha de algoritmos permitiu a implementação de sistemas dinâmicos sem a necessidade de um algoritmo de relaxamento sucessivo.

A linguagem utilizada na implementação foi C++, sempre fazendo uso do paradigma orientado a objetos. Isto facilita a organização do programa, e a implementação de novos blocos se torna simples e objetiva.

Blocos de todos os tipos disponíves na ferramenta foram apresentados, demonstrando as diversas características e capacidades disponíveis. Foram dados exemplos de esquemáticos, e resultados numéricos foram comparados com a solução algébrica do sistema, mostrando uma precisão aceitável para sistemas didáticos.

A interface é simples e a definição de novos esquemáticos é rápida e eficiente. Também foram implementadas funcionalidades para facilitar o uso em aplicações didáticas.

O uso desta ferramenta em aplicações industriais ainda requer melhoramentos no algoritmo de simulação, mas foram criados uma estrutura de objetos, uma interface gráfica e o know-how que permitem a expansão ou modificação do algoritmo para atender a diferentes aplicações de simulação de sistemas dinâmicos.

Embora as funcionalidades implementadas até então, terem sido desenvolvidas para atender as necessidades básicas de uma ferramenta para ensino de controle, ainda existem outras funcionalidades que podem ampliar os benefícios da utilização desta ferramenta:

Criar novos blocos, com outras funcionalidades e algoritmos, demonstrando aspectos não explorados nos blocos fornecidos com a ferramenta.

Implementar um algoritmo de passo variável, de forma que a ferramenta se ajuste automaticamente em função da ordem de grandeza das dinâmicas (constantes de tempo) existentes no sistema simulado.

Implementar um algoritmo mais gerérico, com base no algoritmo de conservação, de forma a flexibilizar os tipos de ligações que podem ser estabelecidas entre blocos, permitindo, por exemplo, ligar duas saídas a uma mesma entrada, ou até mesmo abstrair completamente o conceito de entradas e saídas.

Permitir que blocos sejam simulados em passos diferentes, para que blocos com dinâmicas mais rápidas possam ser simulados com passos menores que blocos com dinâmicas mais lentas.

Implementar um maior número de bloco animados e adicionar sons como forma de realimentação na interface com o usuário.

A ferramenta implementada atende as características necessárias, definidas no início do trabalho. A possibilidade de usar blocos que simulem dispositivos físicos, e diretamente observar os efeitos das diferentes formas de utilização destes blocos, poderá se mostrar mais um meio eficiente de aproximar os conhecimentos teóricos da realidade física.

Referências

ADVANCED CONTINUOUS SIMULATION LANGUAGE. **The Time Proven Standard for Continuous Simulation.** Disponível em: <http://www.aegistg.com/ACSLCUT/ACSLCUT.html> Acesso em: 26 nov. 2004.

BERKELEY MADONNA. **Modelling and Analysis of Dynamic Systems.** Disponível em: <http://www.berkeleymadonna.com/>. Acesso em: 26 nov. 2004.

CAMPIONE, M.; WALRATH, K.; HUML, A. **The Java(TM) Tutorial:** a short course on the basics. [S.l.]: Addison-Wesley Professional, 2000. ISBN: 0-2017-0393-9.

CASINI, M.; PRATTICHIZZO, D.; VICINO, A. The Automatic Control Telelab: a remote control engineering laboratory. In: IEEE CONFERENCE ON DECISION AND CONTROL, 40., 2001, Orlando, Florida USA. **Proceedings...** [S.l.]: IEEE, 2002. ISBN: 0-7803-7061-9.

COELHO, A. A. R.; et al. Learning Lab for Understanding Control Theory of Signals and Linear Systems. In: IEEE CONFERENCE ON DECISION AND CONTROL, 40., 2001, Orlando, Florida USA. **Proceedings...** [S.l.]: IEEE, 2002. ISBN: 0-7803-7061-9.

DORMIDO, S.; ESQUEMBRE, F. The Quadruple-Tank Process: an interactive tool for control education. In: EUROPEAN CONTROL CONFERENCE ECC´03, Cambridge, England. **Proceedings...** [S.l.]: University of Cambridge, UK, 2003.

DYNASIM. **Dymola Multi-Engineering Modelling and Simulation.** Disponível em: <http://www.dynasim.se/>. Acesso em: 26 nov. 2004.

ENGINEERING TOOLBOX. Hazen-Williams Equation. In: ENGINEERING TOOLBOX F**luid Dynamics**. Disponível em: <http://www.engineeringtoolbox.com/21_797.html>. Acesso em 28 nov. 2004.

ENGINEERING TOOLBOX. Bernoulli Equation. In: ENGINEERING TOOLBOX: **Fluid Dynamics**. Disponível em: <http://www.engineeringtoolbox.com/21_183.html>. Acesso em 28 nov. 2004.

EXEL, M.; MICHAU; F.; GENTIL, S. A Web-Based Simulation Workshop for Digital Control. In: EUROPEAN CONTROL CONFERENCE ECC99, Karlsruhe, Alemanha. **Proceedings...** [S.l.]: Springer, 1999. ISBN 1-8523-3122-4

FOX TOOLKIT. **C++ Based Toolkit for Developing Graphical User Interfaces.** Disponível em: <http://www.fox-toolkit.org/>. Acesso em: 28 nov. 2004.

GRANADO E.; et al. Web based Design of Virtual Teaching in the Laboratory of Automatic Control. In: EUROPEAN CONTROL CONFERENCE ECC´03, Cambridge, Inglaterra. **Proceedings...** [S.l.]: University of Cambridge, UK, 2003.

IMAGEMAGICK. **A Robust Collection of Tools and Libraries to Read, Write, and Manipulate an Image.** Disponível em <http://imagemagick.sourceforge.net/>. Acesso em: 4 dec. 2004.

JOCHHEIM, A.; ROHRIG, C. The Virtual Lab for Teleoperated Control of Real Experiments. In: IEEE CONFERENCE ON DECISION & CONTROL, 38.,1999, Phoenix, Arizona USA. **Proceedings...** Phoenix, Arizona USA: IEEE, 1999. ISBN: 0-7803-5250-5.

JOHNSON, D. **White Gaussian Noise.** Disponível em <http://cnx.rice.edu/content/m11281/latest/?format=pdf>. Acesso em: 26 nov. 2004

KO, C.C.; et al. A Webcast Virtual Laboratory on a Frequency Modulation Experiment. In: IEEE CONFERENCE ON DECISION AND CONTROL, 40., 2001, Orlando, Florida USA. **Proceedings...** Orlando, Florida USA: IEEE, 1999. ISBN: 0-7803-7061-9.

KOPPEL, T. **Lectures Pumps and Pumping**, Department of Mechanics – Tallinn Technical University. Disponível em: <www.tut.fi/units/ymp/bio/board/5901012/ Pumps_I.pdf>. Acesso em 28 nov. 2004.

LIU, J. Continuous Time and Mixed-Signal Simulation in Ptolemy II. **Memo M98/74**. Berkeley: UCB/ERL, EECS UC, july 1998.

MATHCORE. **Technologies for Full-System Simulation:** about MathModelica. Disponível em: <http://www.mathcore.com/products/mathmodelica/index.shtml>. Acesso em: 26 nov. 2004.

MATSUMOTO, M.; NISHIMURA, T. **Mersenne Twister:** very fast random number generator. Disponível em <http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html>, acesso em: 26 nov. 2004

MIELE, D.A.; POTSAID, B.; WEN, J.T. An Internet-based Remote Laboratory for Control Education In: AMERICAN CONTROL CONFERENCE, 2001, Arlington, Virginia USA. **Proceedings...** Arlington, Virginia USA: IEEE, 1999. ISBN: 0-7803-6495-3

MODELICA. **Object-Oriented Modelling Language.** Disponível em <http://www.modelica.org/>. Acesso em: 26 nov. 2004.

OCTAVE. **A High-level Interactive Language Iintended for Numerical Computations.** Disponível em: <http://www.octave.org/>. Acesso em: 26 nov. 2004.

OGATA, K. Análise de Sistemas de Controle no Espaço de Estados. In: OGATA, K. **Engenharia de Controle Moderno.** New Jersey, USA: Prentice Hall, 2003. p.154-156,616-618. ISBN: 85-87918-23-0

OGATA, K. **System Dynamics**. [S.l.]: New Jersey, USA: Prentice Hall, 2003. ISBN 0-1314-2462-9

POEDIT. **Cross-platform Gettext Catalogs Editor.** Disponível em: <http://poedit.sourceforge.net/>. Acesso em: 4 dez. 2004.

PRESS, W. H. **Numerical Recipes In C:** the art of scientific computing. Cambridge, UK: Cambridge University Press, 1992. ISBN: 0-521-43108-5

PTOLEMY PROJECT. **Heterogeneous Modeling, Simulation, and Design of Concurrent Systems**. Disponível em <http://ptolemy.eecs.berkeley.edu/>. Acesso em: 26 nov. 2004.

ROSS, S. M. **A First Course in Probability**. 5.ed. [S.l.]: Prentice Hall College Division, 1997. ISBN: 0-137-46314-6

SCHMID, C. The Virtual Control Lab VCLab for Education on the Web. In: AMERICAN CONTROL CONFERENCE, 1998, Philadelphia, Pennsylvania USA. **Proceedings...** Philadelphia, Pennsylvania USA: IEEE, 1998. ISBN: 0-7803-4530-4.

SCILAB. **A Scientific Software Package for Numerical Computations.** Disponível em: <http://www.scilab.org/>. Acesso em: 26 nov. 2004.

SCOPINHO, B., et al. Ambiente Integrado Para Análise, Projeto E Sintonia De Sistemas De Controle. In: CONGRESSO BRASILEIRO DE AUTOMÁTICA, 14., 2002, Natal. **Anais...** Natal: UFRN, 2002.

SOARES, R. P. **Desenvolvimento de um Simulador Genérico de Processos Dinâmicos**. 2003, 162p. Tese (Mestrado em engenharia) – Programa de Pós-Graduação em Engenharia Química, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2003.

TILBURY, D.; LUNTZ, J.; MESSNER, W. Controls Education on the WWW: tutorials for MATLAB and SIMULINK. In: AMERICAN CONTROL CONFERENCE, 1999, Philadelphia, Pennsylvania USA. **Proceedings...** Philadelphia, Pennsylvania USA: IEEE, 1998. ISBN: 0-7803-4530-4.

WXWIDGETS. **An Open Source C++ GUI Framework for Cross-Platform Programming** , Disponível em: <http://www.wxwidgets.org/>. Acesso em: 26 nov. 2004.

ZEILMANN, R. P.; et al. A Web-based Remote Laboratory for Control Education. In: IFAC SYMPOSIUM ON INTELLIGENT COMPONENTS AND INSTRUMENTS FOR CONTROL APPLICATIONS (SICICA'03), 5., Aveiro, Portugal. **Proceedings...** Aveiro, Portugal: Pergamon Press, 2003. ISBN: 0-08-044010-X.

ZEILMANN, R. P. **Uma Estratégia para Controle e Supervisão via Internet de Processos Industriais,** 2002. Tese (Mestrado em Engenharia) – Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal do Rio Grande do Sul, 2002.

Apêndice A

Os capítulos 3 e 4 mostraram a estrutura dos blocos e os métodos utilizados na sua implementação. Este apêndide fornece um exemplo de código fonte, a implementação do bloco Motor.

Arquivo vtbBlock_Motor.h

```
#ifndef vtbBlock_Motor_h
#define vtbBlock_Motor_h

#include "vtbBlock.h"

#define MOTOR_ORDER 2

class vtbBlock_Motor : public vtbBlock
{
public:
   vtbBlock_Motor();
   vtbBlock_Motor( const vtbBlock_Motor& a );
   virtual ~vtbBlock_Motor();
   virtual vtbBlock* Clone() { return new vtbBlock_Motor( *this ); }
   virtual wxString GetName() { return _("Motor"); }
   virtual wxString GetFixName() { return ("Motor"); }
   wxBitmap* GetBitmap();
   vtbBlockCategory GetCategory() { return bcMechanical; }

   virtual int GetNumPorts() { return 3; }
   virtual vtbPortDirection GetPortDirection( int Idx );
   virtual vtbPortType GetPortType( int Idx );

   virtual void ResetBlock();
   virtual void Prolog1();
   virtual bool IsSorted() { return false; }
   virtual bool Simulate( double step_size );

   virtual void Load( wxXmlNode* Node );
   virtual void Save( wxXmlNode* Node );
   virtual void Properties();

   class vtbBlock_MotorDialog* Dialog;

   // Input values
   double mValIn[2];
   double mInertiaIn;

   // Simulator parameters
   double mState[MOTOR_ORDER];
   double mDiff[MOTOR_ORDER];
   double mMatA[MOTOR_ORDER][MOTOR_ORDER];
   double mMatB[MOTOR_ORDER];
   // Torque constant and Counter EMF constant
   double mMotorK;
   // Moment of inertia of the rotor (J)
   double mMotorJ;
   // Electric resistance (R)
```

```
   double mMotorR;
   // Electric inductance (L)
   double mMotorL;
   // Damping ratio of the mechanical system (b)
   double mMotorB;

   // Block specific functions
   bool Calculate_TF();
};

#endif
```

Arquivo vtbBlock_Motor.cpp

```
#include "vtbGlobal.h"
#include "vtbBlock_Motor.h"

#include "VTBSim.h"
#include "vtbFrame.h"
#include "vtbSchematic.h"

#include "icons/motor.xpm"

class vtbBlock_MotorDialog : public wxDialog
{
public:
   vtbBlock_Motor* Block;

   wxTextCtrl* MotorKCtrl;
   wxTextCtrl* MotorJCtrl;
   wxTextCtrl* MotorRCtrl;
   wxTextCtrl* MotorLCtrl;
   wxTextCtrl* MotorBCtrl;

   vtbBlock_MotorDialog( vtbBlock_Motor* Block, wxWindow* parent ) :
      wxDialog( parent, -1, _("Motor Properties"), wxDefaultPosition ),
      Block( Block )
   {
      wxBoxSizer* Sizer = new wxBoxSizer(wxVERTICAL);
      SetSizer(Sizer);

      wxStaticBox* StaticBox = new wxStaticBox(this, wxID_ANY, _("Motor"));
      wxStaticBoxSizer*  StaticBoxSizer  =  new  wxStaticBoxSizer(StaticBox,
wxHORIZONTAL );
      Sizer->Add(StaticBoxSizer, 0, wxGROW|wxALL, 5);

      wxFlexGridSizer* GridSizer = new wxFlexGridSizer( 0, 2, 5, 5 );
      StaticBoxSizer->Add( GridSizer, 0, wxGROW|wxALL, 5 );

      GridSizer->Add( new wxStaticText( this, wxID_ANY, _("Motor Constant
(Nm/A or V/rad/s): ") ), 0, wxALIGN_RIGHT|wxALIGN_CENTER_VERTICAL );
      MotorKCtrl = new wxTextCtrl( this, wxID_ANY, wxT("") );
      GridSizer->Add( MotorKCtrl );

      GridSizer->Add(   new   wxStaticText(   this,   wxID_ANY,   _("Inertia
(Nm/rad/s^2): ") ), 0, wxALIGN_RIGHT|wxALIGN_CENTER_VERTICAL );
      MotorJCtrl = new wxTextCtrl( this, wxID_ANY, wxT("") );
      GridSizer->Add( MotorJCtrl );
```

```
    GridSizer->Add( new wxStaticText( this, wxID_ANY, _("Resistance (ohm):
") ), 0, wxALIGN_RIGHT|wxALIGN_CENTER_VERTICAL );
    MotorRCtrl = new wxTextCtrl( this, wxID_ANY, wxT("") );
    GridSizer->Add( MotorRCtrl );

    GridSizer->Add( new wxStaticText( this, wxID_ANY, _("Inductance (mH):
") ), 0, wxALIGN_RIGHT|wxALIGN_CENTER_VERTICAL );
    MotorLCtrl = new wxTextCtrl( this, wxID_ANY, wxT("") );
    GridSizer->Add( MotorLCtrl );

    GridSizer->Add(   new   wxStaticText(   this,   wxID_ANY,   _("Friction
(Nm/rad/s): ") ), 0, wxALIGN_RIGHT|wxALIGN_CENTER_VERTICAL );
    MotorBCtrl = new wxTextCtrl( this, wxID_ANY, wxT("") );
    GridSizer->Add( MotorBCtrl );

    wxBoxSizer* ButtonSizer = new wxBoxSizer(wxHORIZONTAL);
    Sizer->Add(ButtonSizer, 0, wxALIGN_CENTER_HORIZONTAL|wxALL, 5);

    wxButton* OKButton = new wxButton( this, wxID_OK, _("&OK") );
    OKButton->SetDefault();
    ButtonSizer->Add(OKButton, 0, wxALIGN_CENTER_VERTICAL|wxALL, 5);
    ButtonSizer->Add(new wxButton( this, wxID_CANCEL, _("&Cancel") ), 0,
wxALIGN_CENTER_VERTICAL|wxALL, 5);
    ButtonSizer->Add(new wxButton( this, wxID_APPLY, _("&Apply") ), 0,
wxALIGN_CENTER_VERTICAL|wxALL, 5);

    Sizer->Fit( this );
    Sizer->SetSizeHints( this );
  }

  virtual bool TransferDataToWindow()
  {
    MotorKCtrl->SetValue( wxString::Format( wxT("%f"), Block->mMotorK ) );
    MotorJCtrl->SetValue( wxString::Format( wxT("%f"), Block->mMotorJ ) );
    MotorRCtrl->SetValue( wxString::Format( wxT("%f"), Block->mMotorR ) );
    MotorLCtrl->SetValue( wxString::Format( wxT("%f"), Block->mMotorL ) );
    MotorBCtrl->SetValue( wxString::Format( wxT("%f"), Block->mMotorB ) );
    return true;
  }

  virtual bool TransferDataFromWindow()
  {
    Block->mMotorK = atof( MotorKCtrl->GetValue() );
    Block->mMotorJ = atof( MotorJCtrl->GetValue() );
    Block->mMotorR = atof( MotorRCtrl->GetValue() );
    Block->mMotorL = atof( MotorLCtrl->GetValue() );
    Block->mMotorB = atof( MotorBCtrl->GetValue() );
    TransferDataToWindow();

    // Calculate the matrices based on the settings
    Block->Calculate_TF();

    wxGetApp().Frame->Schematic->SetChanged();
    return true;
  }

private:
  DECLARE_EVENT_TABLE()
};

BEGIN_EVENT_TABLE( vtbBlock_MotorDialog, wxDialog )
END_EVENT_TABLE()
```

```
vtbBlock_Motor::vtbBlock_Motor() : vtbBlock()
{
  // Reset the properties dialog pointer
  Dialog = NULL;

  // Default settings
  mMotorK = 0.625;
  mMotorJ = 0.04f;
  mMotorR = 0.5f;
  mMotorL = 6.0f;
  mMotorB = 0.0005f;

  // Calculate the matrices based on the settings
  Calculate_TF();

  // Allocate all output ports
  PortData[1] = (float*)malloc( MAX_VARS * sizeof(float) );
  PortData[2] = (float*)malloc( MAX_VARS * sizeof(float) );

  // Reset the block
  ResetBlock();
}

vtbBlock_Motor::vtbBlock_Motor( const vtbBlock_Motor& a ) : vtbBlock( a )
{
  int cc, cd;

  // Reset the properties dialog pointer
  Dialog = NULL;

  // Copy the settings
  mMotorK = a.mMotorK;
  mMotorJ = a.mMotorJ;
  mMotorR = a.mMotorR;
  mMotorL = a.mMotorL;
  mMotorB = a.mMotorB;
  for(cc = 0; cc < MOTOR_ORDER; ++cc)
  {
    mState[cc]= a.mState[cc];
    mDiff[cc]= a.mDiff[cc];
    mMatB[cc]= a.mMatB[cc];
    for(cd = 0; cd < MOTOR_ORDER; ++cd)
    {
      mMatA[cc][cd] = a.mMatA[cc][cd];
    }
  }

  // Allocate all output ports
  PortData[1] = (float*)malloc( MAX_VARS * sizeof(float) );
  PortData[2] = (float*)malloc( MAX_VARS * sizeof(float) );

  // Copy the input ports
  PortData[0] = a.PortData[0];
  // Copy the output ports
  PortData[1][pmcSpeed] = a.PortData[1][pmcSpeed];
  PortData[1][pmcTorque] = a.PortData[1][pmcTorque];
  PortData[1][pmcInertia] = a.PortData[1][pmcInertia];
  PortData[2][pelVoltage] = a.PortData[2][pelVoltage];
  PortData[2][pelCurrent] = a.PortData[2][pelCurrent];
}
```

```
vtbBlock_Motor::~vtbBlock_Motor()
{
   if ( Dialog )
     Dialog->Destroy();

   // Free all output ports
   free( PortData[1] );
   free( PortData[2] );
}

vtbPortDirection vtbBlock_Motor::GetPortDirection( int Idx )
{
   switch(Idx)
   {
   default:
   case 0: return pdInput;
   case 1:
   case 2: return pdOutput;
   }
}

vtbPortType vtbBlock_Motor::GetPortType( int Idx )
{
   switch(Idx)
   {
   default:
   case 0:
   case 2: return ptElectric;
   case 1: return ptMechanic;
   }
}

wxBitmap* vtbBlock_Motor::GetBitmap()
{
   static wxBitmap Bitmap( motor );
   return &Bitmap;
}

void vtbBlock_Motor::ResetBlock()
{
   vtbBlock::ResetBlock();

   // Reset current mState
   for(int cc = 0; cc < MOTOR_ORDER; ++cc)
   {
     mState[cc]= 0.0f;
     mDiff[cc]= 0.0f;
   }
}

// This function is overridden because this block is NOT instant
void vtbBlock_Motor::Prolog1()
{
   // Get the values from the connected inputs
   if( PortData[0] ) mValIn[1] = PortData[0][pelVoltage];
   else mValIn[1] = 0.0f;
   // Get the values associated with the outputs
   mValIn[0]= PortData[1][pmcTorque];
   mInertiaIn = PortData[1][pmcInertia];
}

bool vtbBlock_Motor::Simulate( double step_size )
```

```
{
  int cc, cd;

  // Calculate the dynamic elements on the state variable matrix
  mMatA[0][0] = - mMotorB / ( mMotorJ + mInertiaIn );
  mMatA[0][1] = mMotorK / ( mMotorJ + mInertiaIn );
  mMatB[0]= -1.0f / ( mMotorJ + mInertiaIn );

  // Calculate the derivatives of the mState variables
  for(cc = 0; cc < MOTOR_ORDER; ++cc)
  {
    mDiff[cc] = mMatB[cc] * mValIn[cc];
    for(cd = 0; cd < MOTOR_ORDER; ++cd)
    {
      mDiff[cc] += mMatA[cc][cd] * mState[cd];
    }
  }

  // Calculate the new mState values
  for(cc = 0; cc < MOTOR_ORDER; ++cc)
  {
    mState[cc] += mDiff[cc] * step_size;
  }

  // Calculate the output, in RPM and Amps
  PortData[1][pmcSpeed] = mState[0] * RAD_TO_RPM;
  PortData[2][pelVoltage] = mState[1];
  // Current associated with the voltage input
  if( PortData[0] ) PortData[0][pelCurrent] = mState[1];

  return true;
}

void vtbBlock_Motor::Load( wxXmlNode* BlockNode )
{
  vtbBlock::Load( BlockNode );
  for ( wxXmlNode* Node = BlockNode->GetChildren(); Node; Node = Node-
>GetNext() )
  {
    if ( Node->GetName() == wxT("MotorK") )
      mMotorK = atof( Node->GetPropVal( wxT("Value"), wxT("0.625") ) );
    else if ( Node->GetName() == wxT("MotorJ") )
      mMotorJ = atof( Node->GetPropVal( wxT("Value"), wxT("0.04") ) );
    else if ( Node->GetName() == wxT("MotorR") )
      mMotorR = atof( Node->GetPropVal( wxT("Value"), wxT("0.5") ) );
    else if ( Node->GetName() == wxT("MotorL") )
      mMotorL = atof( Node->GetPropVal( wxT("Value"), wxT("6.0") ) );
    else if ( Node->GetName() == wxT("MotorB") )
      mMotorB = atof( Node->GetPropVal( wxT("Value"), wxT("0.0005") ) );
  }

  // Recalculate the matrix, since it is not stored in the file
  Calculate_TF();
}

void vtbBlock_Motor::Save( wxXmlNode* BlockNode )
{
  vtbBlock::Save( BlockNode );

  wxXmlNode* MotorKNode = new wxXmlNode( wxXML_ELEMENT_NODE, wxT
("MotorK") );
```

```
  MotorKNode->AddProperty(      new      wxXmlProperty(      wxT("Value"),
wxString::Format( wxT("%f"), mMotorK ), NULL ) );
  BlockNode->AddChild( MotorKNode );

  wxXmlNode*   MotorJNode   =   new   wxXmlNode(   wxXML_ELEMENT_NODE,   wxT
("MotorJ") );
  MotorJNode->AddProperty(      new      wxXmlProperty(      wxT("Value"),
wxString::Format( wxT("%f"), mMotorJ ), NULL ) );
  BlockNode->AddChild( MotorJNode );

  wxXmlNode*   MotorRNode   =   new   wxXmlNode(   wxXML_ELEMENT_NODE,   wxT
("MotorR") );
  MotorRNode->AddProperty(      new      wxXmlProperty(      wxT("Value"),
wxString::Format( wxT("%f"), mMotorR ), NULL ) );
  BlockNode->AddChild( MotorRNode );

  wxXmlNode*   MotorLNode   =   new   wxXmlNode(   wxXML_ELEMENT_NODE,   wxT
("MotorL") );
  MotorLNode->AddProperty(      new      wxXmlProperty(      wxT("Value"),
wxString::Format( wxT("%f"), mMotorL ), NULL ) );
  BlockNode->AddChild( MotorLNode );

  wxXmlNode*   MotorBNode   =   new   wxXmlNode(   wxXML_ELEMENT_NODE,   wxT
("MotorB") );
  MotorBNode->AddProperty(      new      wxXmlProperty(      wxT("Value"),
wxString::Format( wxT("%f"), mMotorB ), NULL ) );
  BlockNode->AddChild( MotorBNode );
}

void vtbBlock_Motor::Properties()
{
  if ( !Dialog )
    Dialog = new vtbBlock_MotorDialog( this, wxGetApp().Frame );
  Dialog->Show();
}

bool vtbBlock_Motor::Calculate_TF()
{
  // State variables are speed (w) and current (I)
  mMatA[0][0] = - mMotorB / mMotorJ;
  mMatA[0][1] = + mMotorK / mMotorJ;
  mMatA[1][0] = - 1000.0f * mMotorK / mMotorL;
  mMatA[1][1] = - 1000.0f * mMotorR / mMotorL;
  mMatB[0]= -1.0f / mMotorJ;
  mMatB[1]= 1000.0f / mMotorL;

  return true;
}
```

Arquivo motor.xpm

```
/* XPM */
static char *motor[] = {
/* columns rows colors chars-per-pixel */
"64 64 29 1",
"  c black",
". c #4D4D26",
"X c #686834",
"o c #7C7C3E",
"O c gray30",
```

```
"+ c #686868",
"@ c #7C7C7C",
"# c navy",
"$ c #8C8C46",
"% c #9A9A4D",
"& c #A7A754",
"* c #B2B259",
"= c #BDBD5F",
"- c #C7C763",
"; c #D0D068",
": c #E1E171",
"> c #E9E975",
", c #808080",
"< c gray55",
"1 c #A7A7A7",
"2 c #B2B2B2",
"3 c gray74",
"4 c #FFFF80",
"5 c gray78",
"6 c #D0D0D0",
"7 c #E1E1E1",
"8 c #E9E9E9",
"9 c gray94",
"0 c None",
/* pixels */
"0000000000000000000000000000000000000000000000000000000000000000000000000000",
"0000000000000000000000000000000000000000000000000000000000000000000000000000",
"0000000000000000000000000000000000000000000000000000000000000000000000000000",
"0000000000000000000000000000000000000000000000000000000000000000000000000000",
"0000000000000000000000000000000000000000000000000000000000000000000000000000",
"0000000000000000000000000000000000000000000000000000000000000000000000000000",
"0000000000000000000000000000000000000000000000000000000000000000000000000000",
"0000000000000000000000000          00000000000000000000000000000000000000000000",
"0000000000000000000000000            000000000000000000000000000000000000000000",
"0000000000000000000000     4444444     000000000000000000000000000000000000000000",
"0000000000000000000000   4444444444444   000000000000000000000000000000000000000000",
"0000000000000000000000   444444444444444   000000000000000000000000000000000000000000",
"0000000000000000000000   44444444444444444   000000000000000000000000000000000000000000",
"0000000000000000000000   4444444444444444444   000000000000000000000000000000000000000000",
"0000000000000000000000  444444444444444444444  000000000000000000000000000000000000000000",
"0000000000000000000000  4444    .44444.    4444  000000000000000000000000000000000000000000",
"0000000000000000000000      4444     :444:     4444          000000000000000000000000000000",
"0000000000000000000000 ,,,,  4444     =444*     4444  ,,,,,,, 000000000000000000000000000000",
"0000000000000000000000 ,,,  44444   X o444o X   44444  ,,,,,, 000000000000000000000000000000",
"0000000000000000000000 ,,,  44444   &  444  &   44444  ,,,,,,          000000000000000000000",
"0000000000000000000000 ,,,  44444   -X ;4; .-   44444  ,,,,,, ,,,,,,,,,000000000000000000000",
"0000000000000000000000 ,,,  44444   :% &4& %:   44444  ,,,,,, ,,,,,,,,,000000000000000000000",
"0000000000000000000000 ,,,  44444   :- .4. -:   44444  ,,,,,, ,,,,,,,,,000000000000000000000",
"0000000000000000000000 ,,,  44444   :>  ;  >:   44444  ,,,,,,          000000000000000000000",
"0000000000000000000000 ,,,  44444   :4.    4:   44444  ,,,,,, 000000000000000000000000000000",
"0000000000000000000000 ,,,,  4444   :4$   $4:   4444  ,,,,,,, 000000000000000000000000000000",
"0000000000000000000000      4444   44=   =44   4444          000000000000000000000000000000",
"00000000000000###0  4444   44:   :44   4444  000000000000000000000000000000000000000000",
"00000000000####00  444   444   444   444  000000000000000000000000000000000000000000",
"0000000000#####00   4444444444444444   000000000000000000000000000000000000000000",
"000000000#######000   44444444444444   000000000000000000000000000000000000000000",
"0000#########0##0000   444444444444   000000000000000000000000000000000000000000",
"0000########00##00000   4444444444444   000000000000000000000000000000000000000000",
"0000######0000###00000     4444444     000000000000000000000000000000000000000000",
"00000000000000##0000000             00000000000000000000000000000000000000000000",
"00000000000000##000000000         000000000000000000000000000000000000000000000",
"00000000000000##0000000000000000000000000000000000000000000000000000000000000",
```

```
"00000000000000000##0000000000000000000000000000000000000000000000000000",
"00O 0000000 O000###0000000000000000000000000000000000000000000000000000",
"002 5000005 20000###0000000000000000000000000000000000000000000000000000",
"009 +00000+ 900000####0000000000000000000000000000000000000000000000000",
"000< 70007 <0000000##############################################000",
"0006 10001 600000000O##############################################000",
"0000O 009 O000000000000000000000000000000000000000000000000000000000000",
"00001 303 1000000000000000000000000000000000000000000000000000000000000",
"00008 +0+ 8000000000000000000000000000000000000000000000000000000000000",
"00000@ 1 +0000000000000000000000000000000000000000000000000000000000000",
"000006   500000000000000000000000000000000000000000000    000000",
"000000   0000000000000000000000000000000000000000000000  0000000",
"0000001 1000000000000000000000000000000000000000000000000  0000000",
"0000000000000000000000000000000000000000000000000000000000  0000000",
"0000000000000000000000000000000000000000000000000000000000  0000000",
"0000000000000000000000000000000000000000000000000000000000  0000000",
"0000000000000000000000000000000000000000000000000000000000  0000000",
"0000000000000000000000000000000000000000000000000000000000  0000000",
"0000000000000000000000000000000000000000000000000000000000  0000000",
"0000000000000000000000000000000000000000000000000000000000  0000000",
"000000000000000000000000000000000000000000000000000000000    000000",
"0000000000000000000000000000000000000000000000000000000000000000000000",
"0000000000000000000000000000000000000000000000000000000000000000000000",
"0000000000000000000000000000000000000000000000000000000000000000000000",
"0000000000000000000000000000000000000000000000000000000000000000000000",
"0000000000000000000000000000000000000000000000000000000000000000000000",
"0000000000000000000000000000000000000000000000000000000000000000000000"
};
```