UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

MAURÍCIO ELEUTÉRIO DA LUZ

IMPLEMENTAÇÃO DE UMA FERRAMENTA COMPUTACIONAL PARA PROTOTIPAGEM RÁPIDA DE SISTEMAS DE CONTROLE

MEDIANEIRA

2022

MAURÍCIO ELEUTÉRIO DA LUZ

IMPLEMENTAÇÃO DE UMA FERRAMENTA COMPUTACIONAL PARA PROTOTIPAGEM RÁPIDA DE SISTEMAS DE CONTROLE

Implementation of a computational tool for rapid control systems prototyping

Trabalho de conclusão de curso de graduação apresentada como requisito para obtenção do título de Bacharel em Engenharia Elétrica da Universidade Tecnológica Federal do Paraná (UTFPR).
Orientador: Alex Lemes Guedes.

MEDIANEIRA

2022

MAURÍCIO ELEUTÉRIO DA LUZ

IMPLEMENTAÇÃO DE UMA FERRAMENTA COMPUTACIONAL PARA PROTOTIPAGEM RÁPIDA DE SISTEMAS DE CONTROLE

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia Elétrica da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 02/junho/2022

_____

Alex Lemes Guedes
Mestrado
Universidade Tecnológica Federal do Paraná

_____

Diogo Marujo
Doutorado
Universidade Tecnológica Federal do Paraná

_____

Juliano Scholz Slongo
Mestrado
Universidade Tecnológica Federal do Paraná

MEDIANEIRA

2022

AGRADECIMENTOS

Agradeço primeiramente a Deus, pelas oportunidades que me fizeram chegar até aqui.

Agradeço à minha família pelo apoio em todos os sentidos para cursar e concluir um curso de ensino superior.

Aos meus professores, desde o primeiro dia de aula até este momento.

Meu reconhecimento às diversas que pessoas que dispuseram de seu tempo e paciência e colaboraram para que este trabalho fosse concluído.

Minha gratidão a Universidade Tecnológica Federal do Paraná, por proporcionar as condições, o ambiente e os meios para que durante estes anos da graduação pudesse aprender.

RESUMO

O projeto de sistemas de controle possui diversas etapas, como modelagem, simulação, e implementação do controlador. No ambiente acadêmico buscam-se formas de automatizar a implementação, de forma que os alunos de disciplinas de controle possam focar no modelo matemático do controlador. Uma solução para mitigar esse problema é a utilização de prototipagem rápida de sistemas de controle com o método de projeto baseado em modelos. Neste contexto, este trabalho apresenta a implementação de uma ferramenta *Computer-Aided Control System Design* (CACSD), para a prototipagem rápida de sistemas de controle. O software consiste em um ambiente de modelagem de diagrama de blocos, um sistema de simulação em tempo real e sistema de visualização. A ferramenta desenvolvida foi capaz de implementar a prototipagem rápida de sistemas de controle simples, permitindo o uso parcial do método *Model Based Design* com as etapas de modelagem, implementação e testes de sistemas de controle. Os resultados das simulações foram condizentes com o software Matlab, bem como seu desempenho de tempo real mostrou-se suficiente para a simulação de sistemas com passo de integração de 100 milissegundos ou mais.

Palavras-chave: sistemas de controle digital; projeto auxiliado por computador; simulação; engenharia de sistemas.

ABSTRACT

The design of control systems has several stages, such as modeling, simulation, and controller implementation. In the academic environment, ways are sought to automate the implementation, so that students of control disciplines can focus on the mathematical model of the controller. A solution to mitigate this problem is the use of rapid prototyping of control systems with the model-based design method. In this context, this work presents the implementation of a Computer-Aided Control System Design (CACSD) tool for rapid prototyping of control systems. The software consists of a block diagram modeling environment, a real-time simulation system and visualization system. The developed tool was able to implement the rapid prototyping of simple control systems, allowing the partial use of the Model Based Design method with the steps of modeling, implementation and testing of control systems. The simulation results were consistent with the Matlab software, and its real-time performance proved to be sufficient for the simulation of systems with an integration step of 100 milliseconds or more.

Keywords: digital control systems; computer aided design; simulation; systems engineering.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

CACSD	Computer aided control system design
MBD	Model Based Design
HIL	Hardware in the loop
SIL	Software in the loop
MIL	Model in the loop
PIL	Processor in the loop
RCP	Rapid Control Prototyping
JSON	Java script object notation
ASCII	American Standard Code for Information Interchange

SUMÁRIO

1 INTRODUÇÃO

Um sistema de controle tem como objetivo modificar a resposta de um processo a fim de que ele execute uma tarefa de forma adequada, atingindo requisitos de qualidade e de desempenho (NISE, 2013). Num contexto de aquecimento, por exemplo, o sistema de controle é o processo que consegue modular de forma adequada a potência de uma resistência para que o ambiente a ser controlado apresente a temperatura desejada de forma rápida e com variações dentro de um padrão especificado.

Atualmente, a maior parte dos controladores aplicados são digitais. Neste caso, uma das etapas para a implementação do sistema de controle é a programação do mesmo em uma linguagem de programação que depende da plataforma de *hardware* utilizada, por exemplo c/c++ (para sistemas microcontrolados e microprocessados), linguagem estruturada ou lista de instruções (CLP's, controladores lógicos programáveis), ou linguagens de descrição de *hardware*, como VHDL ou Verylog (para CPLD's e FPGA's) (PETER SCHWARZ, 2009).

A etapa de programação requer tempo e esforço que poderiam ser utilizados na otimização da estratégia de controle. Assim, procuram-se maneiras de aproximar as fases de modelagem matemática dos problemas da fase de implementação, automatizando a geração de código, de forma que os projetistas foquem na resolução matemática do problema, e não em esforços de programação. No ambiente acadêmico esse problema é bem visível nas disciplinas de sistemas de controle, com a grande quantidade e complexidade do conteúdo teórico. Isso diminui o tempo disponível para atividades práticas, que poderiam fornecer uma experiência mais realista acerca da aplicação das teorias de controle. Assim, é desejável uma ferramenta que possibilite aos alunos testarem, de forma rápida e efetiva, as técnicas de controle aprendidas, em protótipos de sistemas dinâmicos, por exemplo um pêndulo invertido (BUCHER, 2019).

Como uma resposta a essa necessidade surgiu o *Model Based Design* (MBD - Projeto Baseado em Modelo). Trata-se de uma metodologia de desenvolvimento de *software* para sistemas de controle, comunicação e processamento de sinais na qual um modelo abstrato de um sistema, planta e controlador, é iterativamente refinado até se obter um sistema que atenda as

especificações desejadas de projeto (KELEMENOVÁ, KELEMEN, *et al.*, 2013). No processo de MBD, uma das etapas possíveis é a prototipagem rápida de controle, em inglês RCP (*Rapid Control Prototyping*), que é um conjunto de técnicas e métodos que possibilitam rapidamente implantar o sistema modelado, fazendo a transição de modelo abstrato a um protótipo funcional de sistema de controle através, por exemplo, da geração automática de código (PETER SCHWARZ, 2009). Para se utilizar MBD e RCP, são necessários *softwares* conhecidos pelo acrônimo de CACSD (*Computer Aided Control System Design* - Projeto de Sistema de Controle Auxiliado por Computador) (CHIN, 2013). Eles permitem por meio de um ambiente de modelagem visual em conjunto com linguagens de computação, análisar, simular e a implantar sistemas de controle. Exemplos desses softwares são o Matlab/Simulink (THE MATHWORKS, INC., 2021) e o Scilab/Xcos (ESI GROUP, 2021).

Nesse trabalho será descrita a implementação de um software CACSD que permita modelar, simular e implantar sistemas de controle por meio de diagramas de blocos funcionais, a fim de permitir a prototipagem rápida de sistemas de controle nas atividades de laboratório do ambiente acadêmico. O trabalho abordará aspectos de simulação de sistemas dinâmicos e programação orientada a objetos e permitirá a integração com sistemas reais através de comunicação serial.

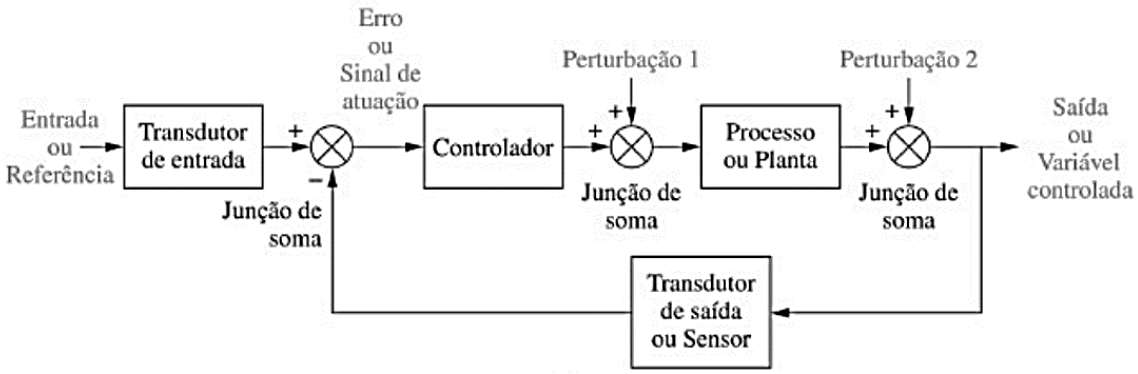
Este documento está dividido nas seguintes partes: Na Introdução, é mostrado o contexto do trabalho e rapidamente apresenta os conceitos de MBD e RCP, assim como os objetivos. No capítulo 2, são apresentados de forma mais aprofundada os conceitos de MBD, RCP e CACSD, além do estado da arte em *softwares* CACSD e simulação de sistemas em tempo real. No capítulo 3 é descrita a implementação do projeto, consistindo na arquitetura de *software* utilizada, nos padrões de *design* de programação aplicados no projeto, além do funcionamento do sistema de simulação de sistemas dinâmicos. No capítulo 4 são apresentados os resultados de teste e validação do *software* desenvolvido e por final na Conclusão são realizadas as considerações finais sobre o trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Modelagem de Sistemas

Um sistema pode ser definido como um conjunto de elementos interligados que funcionam em conjunto para alcançar um objetivo, podendo ser tanto físico, como um sistema de controle, quanto um modelo abstrato, como por exemplo o sistema econômico (OGATA, 2005). Um sistema de controle é um sistema cujo objetivo é, dado um valor de entrada, obter uma saída desejada, conforme um desempenho especificado, de forma a controlar outro sistema (NISE, 2013). Um modelo de sistema de controle tem a forma apresentada na Figura 1.

Figura 1 - Sistema de controle típico em malha fechada.



Fonte: Nise (2013).

O modelo apresentado na Figura 1 é de um sistema de controle em malha fechada, onde a diferença entre o sinal de referência e a saída é utilizada para definir a ação de controle. Um sistema pode ser classificado como sendo de tempo contínuo ou de tempo discreto (LATHI, 2008). Sistema de tempo contínuo é aquele em que os sinais de entrada e saída são definidos em uma faixa contínua de valores. Sistemas de tempo discreto são aqueles que processam sinais que possuem valor apenas em intervalos discretos de tempo. Normalmente em sistemas digitais, como os microcontroladores e microprocessadores são de tempo discreto, as operações são realizadas conforme um sinal discreto chamado *Clock*.

Para permitir simulações e análises, os sistemas devem ser representados de forma abstrata através de um modelo. Existem diferentes formas de se representar um modelo, por exemplo, um sistema de controle pode ser representado em termos de funções de transferência ou representado no espaço de estados, sendo possível alternar entre as duas representações. Os conjuntos de regras que

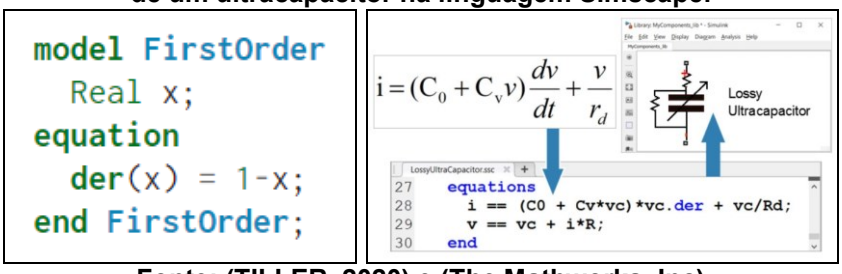
definem como um modelo deve ser representado são chamados de linguagens de modelagem, e podem ser tanto textuais, quanto gráficas (XIAO, 2007).

2.1.1 Linguagens de modelagem textual

Existem várias linguagens textual. Uma delas, a notação matemática, utiliza-se de expressões matemáticas de diversos campos, como a lógica, cálculo e álgebra, para construir um modelo de um sistema. Para sistemas de controle, as formas mais utilizadas são conjuntos de equações diferenciais e funções de transferência. A linguagem matemática é também usada em outras linguagens de modelagem, por exemplo para especificar comportamentos de um sistema em linguagens como a Modelica (TILLER, 2020) e Simscape (THE MATHWORKS), por meio de equações algébrico-diferenciais.

A Modelica é uma linguagem de modelagem de sistemas orientada a objetos, aplicada na simulação de sistemas de vários domínios, como sistemas elétricos, mecânicos, hidráulicos e sistemas de controle. O Simscape é uma linguagem de modelagem de sistemas multidomínio baseada em equações e foi criada pela The Mathworks, Inc. A Figura 2 mostra a representação de um sistema de primeira ordem na linguagem Modelica e a definição de um ultracapacitor na linguagem Simscape, respectivamente.

Figura 2 - Representação de um sistema de primeira ordem na linguagem Modelica e definição de um ultracapacitor na linguagem Simscape.



Fonte: (TILLER, 2020) e (The Mathworks, Inc)

Um exemplo de aplicação de linguagens de modelagem textual na indústria são Verilog-AMS e VHDL-AMS que permitem a modelagem e implementação de sistemas mistos digital/analógico (HUSS, 2003).

2.1.2 Linguagens de Modelagem Visual

Linguagens de modelagem visual utilizam-se de formas gráficas, como diagramas, para representar o sistema a ser modelado. No Diagrama de blocos, por exemplo, cada bloco representa um componente do sistema, com suas entradas e saídas, e as linhas representam as interconexões entre os elementos. Um sistema modelado por diagrama de blocos tipicamente é do tipo causal, ou seja, o fluxo de informação tem um sentido definido. O comportamento de cada componente do sistema pode ser especificado em outra linguagem, como a matemática, utilizando uma função de transferência ou sua representação no espaço de estados.

A UML(*Unified Modeling Language* – Linguagem de Modelagem Unificada) é um exemplo de linguagem visual. Ela é utilizada para a modelagem de sistemas de *software* que se utilizam do paradigma de programação orientada a objetos. Utiliza 12 diagramas para modelar diferentes aspectos de *softwares* além da linguagem OCL (*Object Constraint Language* – Linguagem de Restrição de Objetos) para definir restrições e especificações de projeto (ERICH GAMMA, 2005).

Observa-se que existem dois importantes aspectos em um modelo: o estrutural e o comportamental. O aspecto estrutural se refere à forma como os elementos de um sistema, os subsistemas, são relacionados. O aspecto comportamental diz respeito à funcionalidade de cada subsistema e do sistema como um todo. Linguagens de modelagem gráficas, como os diagramas de blocos, enfatizam o aspecto estrutural, enquanto linguagens como a notação matemática facilitam expressar o aspecto comportamental. Dessa forma, é possível utilizar mais de uma linguagem para representar de forma apropriada um sistema (SCHWARZ, 2009).

2.1.3 Projeto Baseado em Modelo

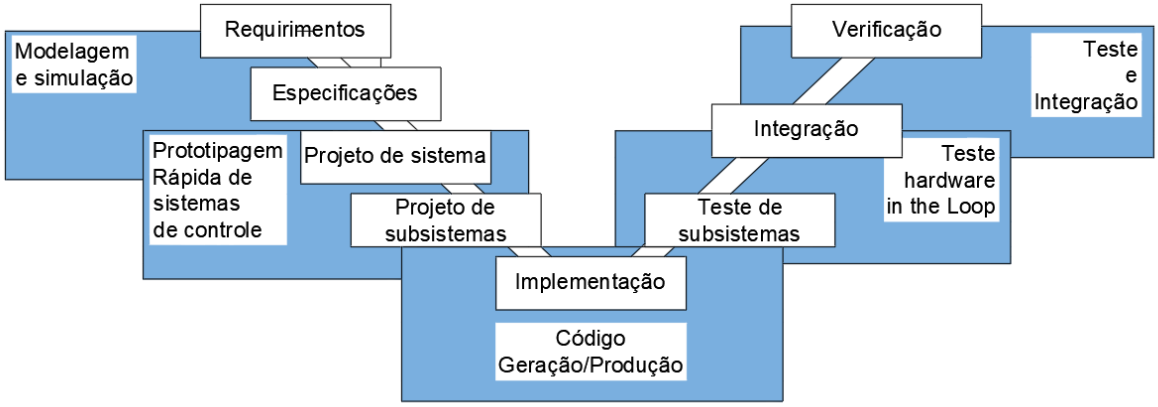
Projeto baseado em Modelo (*Model Based Design* - MBD), é uma abordagem de projeto de sistemas de controle, processamento de imagens e sistemas de comunicação, na qual uma linguagem de modelagem visual é utilizada para construir um modelo matemático que será analisado, simulado, testado e refinado iterativamente até satisfazer as especificações de projeto (KELEMENOVÁ, KELEMEN, *et al.*, 2013). Essas especificações podem ser tanto de natureza matemática, como a resposta ao degrau, máximo sobressinal, rejeição a

perturbações e taxa de amostragem, quanto especificações mais gerais, como velocidade de processamento, armazenamento, etc. No MBD, o foco está em se projetar um sistema cujo comportamento, definido pelo modelo matemático, atenda as especificações. O processo de projeto em MBD pode ser dividido nas seguintes etapas:

- Modelagem da planta: esse modelo, que pode ser obtido através de identificação de sistemas ou modelagem por princípios físicos, geralmente é simplificado em relação ao comportamento real, em virtude da complexidade ou do desconhecimento de todas as leis físicas que o governam.
- Projeto do sistema de controle: definição dos requisitos que o controlador deve satisfazer. Ferramentas matemáticas e técnicas de controle são aplicadas para analisar e construir um modelo do controlador, tais como PID, espaço de estados, lógica fuzzy, lugar das raízes, etc.
- Simulação da planta e do controlador: está presente ao longo do processo de MBD, sendo realizada simultânea e iterativamente com as outras etapas, servindo para verificar se as especificações do projeto estão sendo cumpridas. No contexto de simulação, surgem termos como simulação *off-line* e simulação de tempo real, SIL (*Software in the loop*), MIL (*Model in the Loop*), PIL (*Processor in the loop*), HIL (*Hardware in the loop*) e prototipagem rápida de sistemas de controle, os quais serão explicados posteriormente.
- Implementação: realização física do modelo matemático encontrado através uma linguagem de programação adequada, como c/c++ para microcontroladores, VHDL para hardware programável, Ladder para CLP, etc.
- Validação e teste: testes são realizados a fim de validar o sistema de controle e a planta. Eles podem ser feitos tanto com a simulação *off-line* ou em tempo real, quanto por prototipagem do sistema de controle. Por fim o teste do sistema pode ser implantado na plataforma de *hardware* final (CLP, sistemas embarcados, PC, etc) e integrado na planta a ser controlada.

As etapas do processo MDB são realizadas de forma iterativas, algumas vezes ocorrendo de forma simultânea. A Figura 3 demonstra a integração e sobreposição das diversas etapas de MDB.

Figura 3 - Etapas de *Model Based Design*.



Fonte: Adaptado de Simon Abourida (2005).

2.2 Simulação em Projeto baseado em Modelo

Simulação é o processo de executar um modelo matemático e computacional de um sistema real a fim de imitar seu funcionamento e com o objetivo de estudar seu comportamento (PEGDEN, 1990). No contexto de MDB, podemos dizer que a simulação tem o objetivo de, a partir do modelo matemático da planta e do controlador, fornecer dados para a verificação e validação quanto aos requisitos de projeto. Caso observado que o controlador simulado não atende os requisitos, ele pode ser reprojetado e simulado novamente, e assim iterativamente o sistema de controle é refinado até atingir resultados aceitáveis.

Em MDB podemos destacar duas formas de se realizar simulações: a que utiliza geração de código e a que interpreta o modelo. Na geração de código, um algoritmo transforma o modelo de um sistema em linguagens como C/C++ e Fortran, compila esse código e o vincula ao sistema de simulação do *software* CACSD. Toda vez que for feita alguma alteração no modelo, é necessário gerar código novamente e recompilar. Além disso, a comunicação entre o sistema CACSD e compilador é necessária, aumentando a complexidade na implantação de ferramentas MBD. Na interpretação de modelo, um modelo é simulado por uma máquina virtual (CABOT, 2010), por exemplo em um diagrama de blocos, cada bloco pode ter seu comportamento implementado em uma linguagem de programação, então para simular o interpretador invoca essa função.

As simulações utilizadas em MBD podem ser dos seguintes tipos (KELEMENOVÁ, KELEMEN, *et al.*, 2013):

1. Simulação *Offline*: Neste caso a preocupação é apresentar o comportamento do sistema simulado por um determinado período de tempo, sem se preocupar com o momento em que os dados requeridos são apresentados. A simulação pode utilizar escalas de tempo mais lentas ou mais rápidas do que o tempo real do sistema. Por exemplo: a simulação de 50 anos do crescimento populacional de determinado país pode ser feita em alguns segundos. Por outro lado, a simulação da dinâmica de partículas subatômicas é muito mais demorada que no mundo físico.
2. Simulação de tempo real: Neste tipo de simulação, a escala de tempo é a mesma do sistema a ser simulado. No contexto de sistemas de controle, geralmente isso é feito utilizando um solucionador de passo fixo. Como os *softwares* de simulação são normalmente executados em sistemas operacionais como Windows, Linux ou MAC, que não satisfazem os requisitos de sistema operacional de tempo real, essa simulação é feita em uma unidade computacional própria para esse tipo de simulação. Dada sua característica de operar em tempo discreto, sistemas de controle digitais normalmente operam versões discretizadas dos algoritmos de controle.
3. MIL – *Model in the Loop*: um protótipo virtual da planta e do controlador é simulado. Geralmente é uma simulação *off-line* onde não há geração de código para o controlador. É o primeiro estágio de simulação, feito no próprio ambiente de modelagem.
4. SIL – *Software in the Loop*: Nesta etapa de simulação é gerado o código do controlador a partir do diagrama de blocos do mesmo, porém executado no mesmo ambiente de desenvolvimento, geralmente uma simulação *offline*. Seu objetivo é validar o código gerado que será posteriormente implantado na plataforma de *hardware* final. O código gerado é compilado na forma de uma biblioteca compartilhada e então ligado dinamicamente ao *software* de simulação. Nota-se que para isso é necessário que o ambiente de desenvolvimento tenha interação com um compilador.
5. PIL – *Processor in the loop*: O objetivo desta etapa é avaliar o desempenho da plataforma de *hardware* em processar o algoritmo de

controle, por exemplo, se a plataforma consegue executar os cálculos com o passo de tempo definido. Deve ser providenciado, juntamente com o código do algoritmo de controle, uma forma de comunicação com o ambiente de desenvolvimento, pois normalmente nesta etapa o controlador não estará integrado a planta e esta será simulada. Para realizar a comunicação com o simulador é gerado um código adicional, (geralmente para comunicação serial via USB), ou são utilizados *debuggers*, que interagem com o *hardware* sem necessidade de geração de código. Pode–se citar o protocolo JTAG como maneira de realizar a segunda forma, tal protocolo normalmente é utilizado no modo de depuração da maioria dos ambientes de programação para microcontroladores e FPGA's.

6. HIL – *Hardware in the loop*: Neste tipo de simulação a planta é simulada em tempo real, podendo o controlador ser simulado na mesma plataforma da planta ou o mesmo ser implantado na plataforma de destino e integrada à simulação da planta. O objetivo desta simulação é ser a mais realista possível, para isso pode ser necessário executar passos da ordem de microssegundos, o que pode requerer alta capacidade de processamento dependendo da quantidade de cálculos que devem ser feitos.

2.3 Prototipagem Rápida de Sistemas de Controle

RCP (Prototipagem rápida de sistemas de controle - *Rapid Control Prototyping*), pode ser definido como um conjunto de métodos e técnicas que permitem rapidamente implantar e testar protótipos de sistemas de controle. Isso é alcançado principalmente pela geração automática de código a partir de modelos do sistema de controle (PETER SCHWARZ, 2009). No contexto de RCP, protótipo significa a implantação do algoritmo de controle em uma plataforma de *hardware* diferente da desejada ao final do processo de projeto. Neste sentido, podemos afirmar que as técnicas de simulação MIL, SIL, PIL e HIL são formas de prototipagem rápida de controle.

Observa-se que RCP requer: um *hardware* para aquisição e geração de sinais que serão integrados com a planta a ser controlada, e um ambiente de modelagem capaz de gerar código automaticamente. A combinação entre PC, como

a plataforma de *hardware*, juntamente com sistemas operacionais, como Windows ou Linux têm sido utilizada para a realização de RCP pois oferece a vantagem de em um mesmo ambiente realizar todas as etapas do projeto de controle. Isso é particularmente adequado em atividades de ensino e laboratório, como pode ser visto no trabalho de (BUCHER e BALEMI, 2006), no qual são utilizados Matlab e uma versão modificada do Linux(RTAI) para executar simulações HIL.

Como mencionado anteriormente, para se realizar HIL, é necessário um sistema de processamento que execute tarefas em tempo real. Originalmente o Windows e o Linux não são sistemas de tempo real, pois, ao executar uma tarefa, não há determinismo de em quanto tempo essa tarefa será executada porque isso depende de quantas aplicações estão sendo utilizadas, qual a capacidade de processamento etc. No entanto, existem esforços para adicionar características de tempo real ao Windows e o Linux. Dentre eles pode-se citar:

1. Linux
    a. RTAI - (*RealTime Application Interface*): Desenvolvido pelo Departamento de ciência e tecnologia aeroespacial da Universidade Politécnica de Milão, é uma extensão ao Linux que adiciona capacidades de tempo real por meio de uma camada de abstração de *hardware* (RTAI, 2018).
    b. PREEMPT_RT: Uma extensão ao Linux que adiciona capacidades de tempo real ao agendador de tarefas (FOUNDATION, 2013).
2. Windows
    a. RTX64: Desenvolvido pela empresa IntervalZero, esse sistema captura as capacidades de processamento de núcleos em microprocessadores com mais de um núcleo, e instala um sistema de tempo real. Os usuários do Windows podem então adicionar tarefas por meio de bibliotecas compartilhadas (dll). Essa extensão é utilizada por exemplo pelo *software* Labview para prover capacidades de tempo real no Windows (INTERVALZERO, 2020).

2.4 *Computer Aided Control System Design* – CACSD

Computer Aided Control System Design (CACSD), pode ser definido como qualquer *software* capaz de auxiliar em alguma das etapas do projeto em sistemas de controle. Algumas características comuns nesses *softwares* são:

- Modelagem: possuem editores gráficos com características CAD básicas, como desenho de elementos gráficos, conexão de elementos entre si, ferramentas arrasta e solta (*drag and drop*), bem como biblioteca básica de blocos para linguagens de modelagem e programação visuais. Existem muitos editores de linguagens gráficas, sendo os mais conhecidos e reconhecidos como estado da arte o Simulink, Xcos e Labview. Alguns CACSD possuem suporte para linguagens de modelagem textual, como o Matlab com o Simscape e o Xcos com a Modelica.
- Simulação: *Softwares* CACSD se utilizam de sistemas de computação simbólica para a manipulação de equações e solucionadores numéricos de sistemas dinâmicos, sendo eles implementados no *software* ou realizado a integração com bibliotecas *open source* de solucionadores numéricos, como a Odeint (KARSTEN AHNERT, 2012) ou SUNDIALS (LLNL, 2020).
- Análise: os *softwares* CACSD normalmente possuem editores de texto para alguma linguagem de *script* para computação científica embutida, como é o caso do Matlab e Scilab, além da integração com outras linguagens como Python, R e Júlia. Essas linguagens possuem bibliotecas de funções que permitem analisar sistemas de controle. Alguns exemplos são funções que obtém gráficos de bode ou do lugar das raízes a partir de uma descrição de função de transferência, bem como funções para determinar se essa função de transferência representa um sistema de controle estável. Uma característica comum desses editores é a capacidade de visualização das variáveis no escopo global, o que permite rapidamente inspecionar o código e verificar resultados.

2.5 Simulação de sistemas dinâmicos

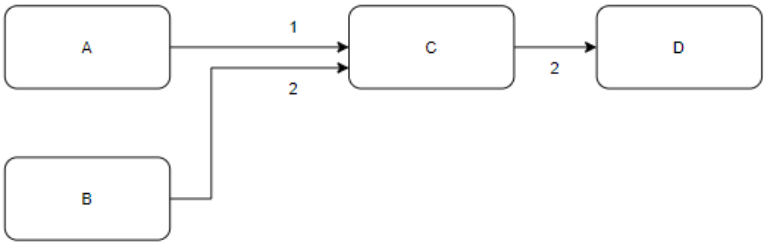
Nesta seção será apresentada a base teórica das técnicas utilizadas para a simulação de sistemas dinâmicos.

2.5.1 Simulação por meio de diagramas de fluxo de dados síncronos

Diagrama de fluxo de dados síncrono (SDF) é uma forma de computação proposta por Edward A. Lee, em 1987 (MESSERSCHMITT e LEE, 1987). Visualmente um SDF é apresentado como um diagrama de blocos no qual cada bloco executa uma função que têm como argumento os valores das entradas do bloco e coloca o resultado nas portas de saída. Toda vez que um dado em alguma porta de entrada é atualizado uma computação é disparada, e os valores das portas de saída de todos os blocos são atualizados. Como é conhecido as dependências de dados entre os blocos, pode-se obter previamente a ordem em que os blocos serão computados.

Na Figura 4 pode-se observar um exemplo de SDF. Como os blocos A e B não têm nenhum dado de entrada, não importará se a ordem de execução começará com o bloco A ou o bloco B. O bloco C só poderá computar após os blocos A e B e o bloco D somente após o bloco C. Dessa forma têm-se duas sequências possíveis de computação: ABCD ou BACD.

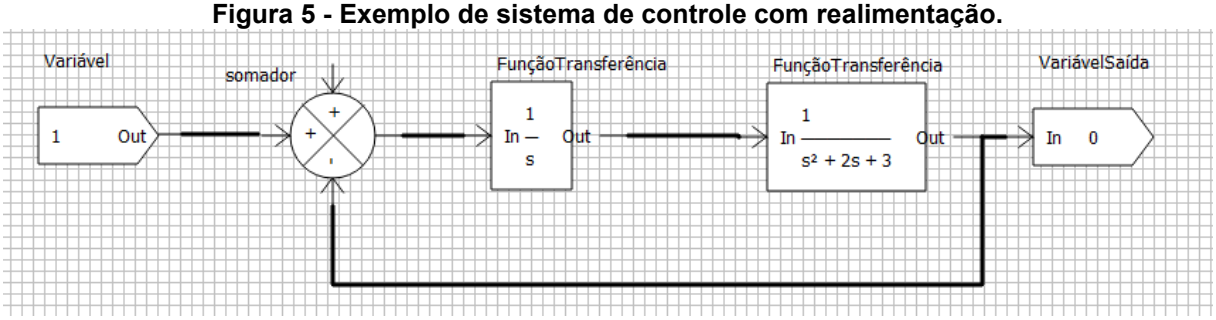
Figura 4 - Um diagrama de fluxo de dados síncrono.



Fonte: Autoria própria.

Na Figura 5 pode-se observar um exemplo de sistema de controle modelado em um diagrama de blocos. Observa-se que sistemas de controle podem ser modelados como diagramas de fluxo de dados síncronos (S. BHATTACHARYYA, K. MURTHY e A. LEE, 1996) (INRIA, 2021). Uma simulação de sistema dinâmico utilizando formalismo SDF normalmente será uma simulação a tempo discreto, ou seja, a cada intervalo de tempo definido, chamado de *Passo de Integração*, os blocos do sistema de controle são computados na ordem previamente calculada.

Figura 5 - Exemplo de sistema de controle com realimentação.

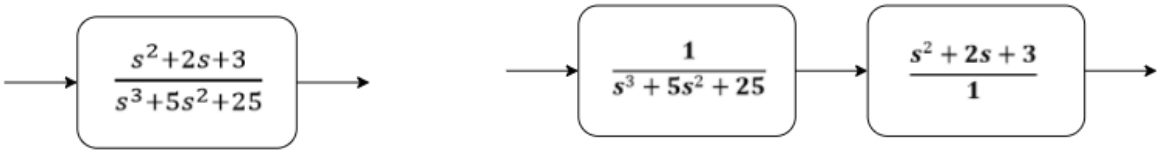


Fonte: Autoria própria.

2.5.2 Representação de funções de transferência em diagramas de blocos

Como demonstrado em (NISE, 2013), uma função de transferência pode ser representada como um diagrama de blocos, considerando que tanto o numerador quanto o denominador são polinômios de “s”, e, portanto, são combinação linear de s. A variável s em um sistema de controle representa um derivador puro, o qual é obtido ao colocar um integrador em malha fechada. Como polinômios são combinações lineares, ao substituir a variável “s” por integradores em malha fechada, pode-se representar o numerador e o denominador de uma função de transferência como uma combinação linear de integradores em cascata, e por sua vez a função de transferência como uma multiplicação entre o numerador e o denominador. A função de transferência da Figura 6 pode ser representada pelo diagrama de blocos da Figura 7.

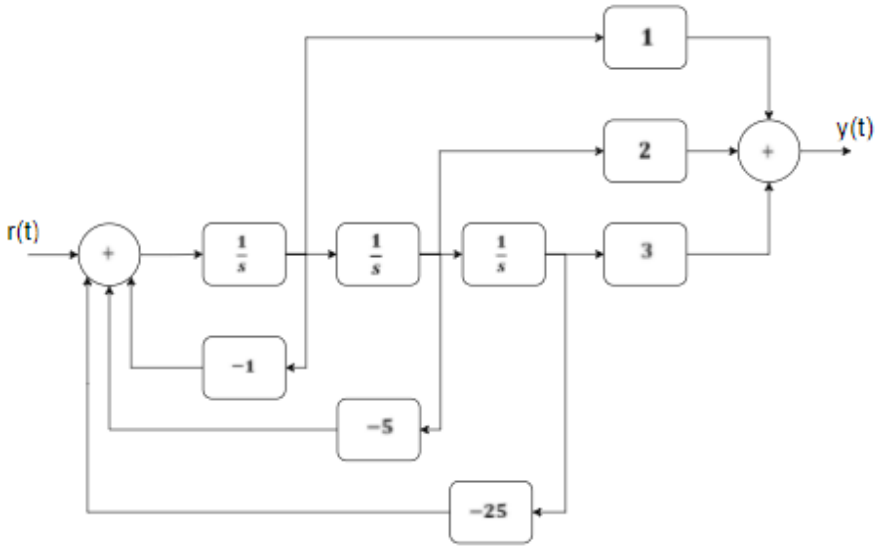
Figura 6 - Exemplo de função de transferência. (a) é equivalente a (b).



(a) (b)

Fonte: Adaptado de Nise (2013).

Figura 7 - Representação em diagrama de blocos da função de transferência da Figura 6.



Fonte : Nise (2013).

2.5.3 Simulação em tempo real e medidas de performance

No contexto de sistemas de controle, sistemas de tempo real são aqueles em que o tempo de execução das tarefas é conhecido, ou seja, determinista, e obedece às restrições de tempo impostas pela planta a ser controlada. Nesse sentido um sistema de tempo real não precisa ser necessariamente o mais rápido, mas o tempo de reação aos sinais recebidos deve ser rápido o suficiente para satisfazer os requisitos de controle (C. BUTAZZO, 2011). Assim, simulação em tempo real pode ser definida aquela em que o simulador cumpre com os requisitos de sistemas de tempo real.

Um sistema de tempo real com boa performance é aquele em que o tempo gasto na atualização entre os valores de entrada e os valores de saída não ultrapasse cinco por cento do tempo total da iteração (LEDIN, 2009). No *software* desenvolvido nesse trabalho isso significa que a execução de todos os blocos de função não deve ultrapassar cinco por cento do tempo de integração. Ultrapassar esse valor significa adicionar blocos de atraso na função de transferência e, dessa forma, o sistema modelado, ao ser simulado, terá sua resposta modificada em virtude da velocidade de processamento (CELLIER e KOFFMAN, 2006).

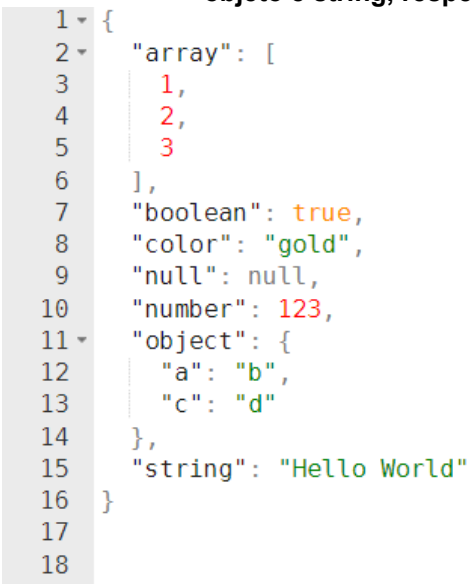
2.6 O Framework Qt

Desenvolvido pela norueguesa Trolltech, O *framework* Qt é um conjunto de bibliotecas multiplataforma em C++, com muitas classes que fornecem funcionalidades diversas, como gráficos 2D e 3D, integração com banco de dados e linguagens de *script*, desenvolvimento de GUI e comunicação Ethernet. Qt é distribuído em versões com licenças comerciais e *open source*, aliado com a grande quantidade de bibliotecas, e extensa documentação, além de ser multiplataforma possibilitou seu uso por muitas empresas. Muitos *softwares* CAD o utilizam, por exemplo Blender 3D, Eagle e Protheus. As distribuições do Qt sob as licenças *open source* LGPL 2.1 e LGPL 3 permitem que seja utilizado comercialmente sem a necessidade de pagamento de *royalties* (THE QT COMPANY, 2022).

2.7 JSON

JSON, acrônimo em inglês para *Java Script Object Notation*, é um formato de arquivo de dados, legível por humanos, geralmente armazenado de forma textual (Introducing JSON, 2021). Os dados em JSON são sempre descritos como um par Chave/valor. Os dados podem ser do tipo *objeto (object), lista (array) e valor (value)*. Os dados do tipo valor podem ser cadeias de caracteres (*strings*), números, os valores booleanos (*bool*) *true* e *false*, lista de valores e objetos, além da palavra reservada *null*, que indica valor nulo ou indefinido. Valores do tipo *objeto* podem conter outros dados aninhados, o que permite a descrição hierárquica dos dados. Neste trabalho, JSON será utilizado como um formato de armazenamento dos diagramas de blocos, uma vez que, por sua característica de ser legível, e apresentar os dados de forma hierárquica facilita descrição dos blocos de funções e seus parâmetros. Além disso, o fato de arquivos e dados JSON serem facilmente manipulados via QT corrobora com a escolha de utilização deste formato.

Figura 8 - Exemplo de arquivo JSON. Os valores são do tipo array, bool, string, null, número, objeto e string, respectivamente.



```
{
  "array": [
    1,
    2,
    3
  ],
  "boolean": true,
  "color": "gold",
  "null": null,
  "number": 123,
  "object": {
    "a": "b",
    "c": "d"
  },
  "string": "Hello World"
}
```

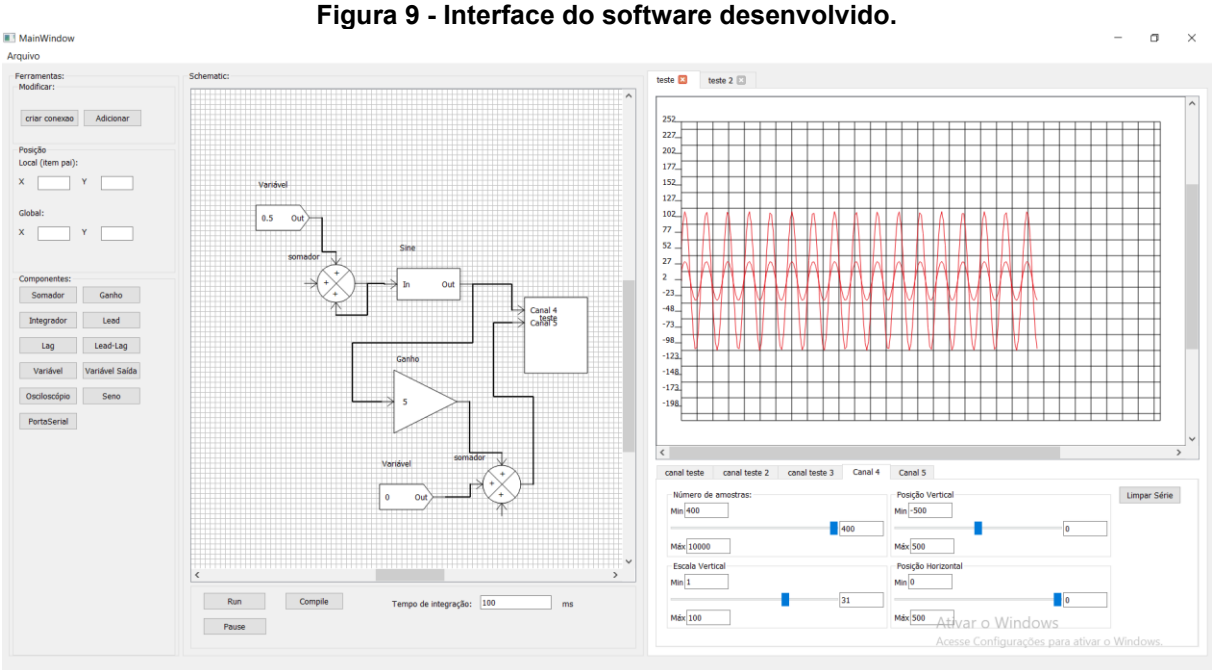
Fonte: Autoria própria.

3 MATERIAIS E MÉTODOS

3.1 Arquitetura

Neste trabalho foi proposto a implementação de um *software* para a Prototipagem rápida de sistemas de controle, a saber um *software* do tipo CACSD, para auxiliar nas etapas de modelagem e implementação do projeto de sistema de controle. Para atender esses requisitos foi necessário o desenvolvimento de três subsistemas: o ambiente de modelagem, o sistema de simulação em tempo real, e um sistema que permita ao usuário visualizar graficamente variáveis do modelo e modificá-las, tal como um supervisório. Adicionalmente, foram implementados dois outros subsistemas, que são a interface do usuário e o sistema de desfazer/refazer, responsável por gerenciar as ações feitas nos programas e fornecer a funcionalidade de desfazer ou refazer ações. Na Figura 9 pode-se ver um modelo em diagrama de blocos, abaixo do modelo existem botões que comandam o sistema de simulação e à direita o sistema de visualização.

Figura 9 - Interface do software desenvolvido.



Fonte (Autoria própria).

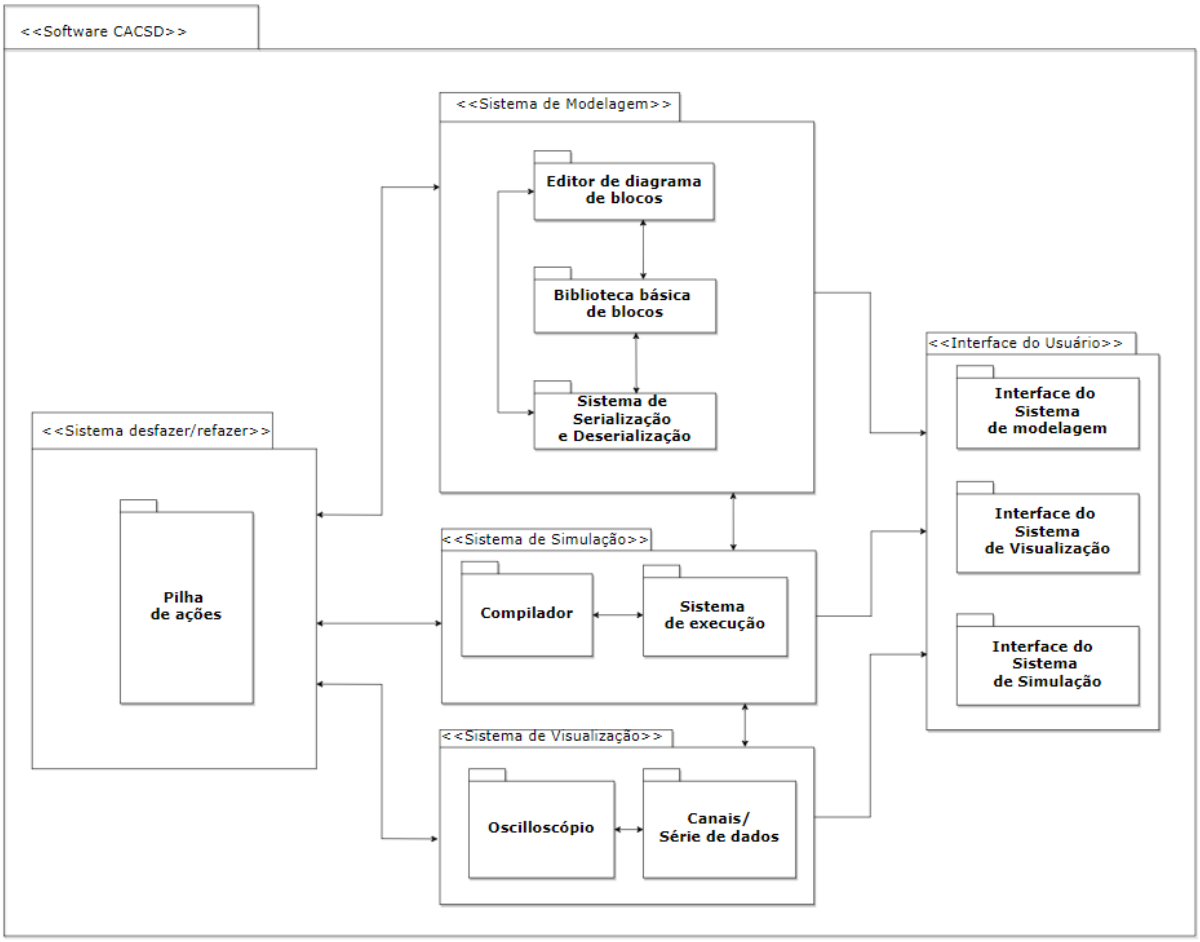
O desenvolvimento do *software* foi dividido em módulos funcionais, de modo que alterações em um módulo não cause inconsistências ou grandes alterações em módulos que dependam dele.

A seguir são relacionados os módulos e suas funcionalidades:

1. Sistema de modelagem:
    a. Editor de diagrama de blocos:
        i. Criar e excluir blocos
        ii. Desenhar e excluir conexões
    b. Sistema de serialização/desserialização
        i. Salvar e carregar diagramas de blocos salvos
    c. Biblioteca básica com os seguintes blocos:
        i. Variável de entrada e de saída
        ii. Somador
        iii. Ganho
        iv. Multiplicador
        v. Divisor
        vi. Timer
        vii. Funções matemáticas: trigonométricas como seno, cosseno e funções diversas como exponencial, e logaritmo.
        viii. Integrador
        ix. Derivador
        x. Função de transferência
        xi. Funções lógicas: and, or, nor, e de comparação
        xii. Polinômio
        xiii. Blocos que integram o modelo ao sistema de visualização: o bloco osciloscópio.
        xiv. Bloco composto: este bloco terá sua funcionalidade expressa em termos de outros blocos.
        xv. Bloco de comunicação serial

2. Sistema de simulação
    a. Classes que descrevam blocos funcionais e o diagrama de blocos
    b. Compilação o diagrama de blocos
    c. Execução de simulações em tempo real e *off-line*
3. Sistema de visualização
    a. Gráficos para a visualização de variáveis, inclusive em tempo real
    b. Controles gráficos para alteração de valores das variáveis
4. Interface do Usuário
    a. Integração dos três subsistemas anteriores em uma interface gráfica.
5. Subsistema de desfazer/refazer – (Undo/Redo):
    a. Facilitação para a criação de comandos
    b. Gerenciamento das ações executadas no *software*.

Figura 10 - Diagrama de pacotes UML 2.0 do software desenvolvido.



Fonte: Autoria própria.

3.2 A interface do usuário

A interface de usuário integra todos os serviços fornecidos pelos demais subsistemas. Ao iniciar o programa, é instanciado um editor gráfico, a interface para a biblioteca de blocos, o sistema de visualização, o sistema de simulação e o desfazer/refazer. Em seguida integra os subsistemas, por exemplo fornecendo uma referência do sistema de desfazer/refazer para o Editor de diagramas, o sistema de visualização e a paleta de ferramentas. Dessa forma há um desacoplamento entre a interface do usuário e a implementação de cada subsistema, possibilitando por exemplo que se possa alterar o *layout* e o estilo da interface de usuário sem causar alterações nos outros subsistemas. Essa integração é feita majoritariamente com o sistema de sinais e *slots* fornecido pelo Qt.

3.3 O sistema de desfazer/refazer

Neste sistema cada modificação realizada no diagrama de blocos, em cada bloco e conexão, e no sistema de visualização, é encapsulada em uma pilha relacionada com a ação de desfazer, de forma que ali estarão todas as informações necessárias para desfazer uma ação. Quando solicitado desfazer uma ação, a última ação guardada é desfeita e movida para outra pilha, dessa vez de ações a serem refeitas.

Isso adiciona uma sobrecarga extra de trabalho, pois para cada ação deve ser implementada duas vezes, uma implementação pura, e outra encapsulada nos objetos responsáveis por esta tarefa. Isso é necessário para que não haja acoplamento entre o sistema de desfazer/refazer e o objeto sobre o qual irá operar.

3.4 O Sistema de Modelagem

Como mencionado anteriormente, o sistema de modelagem compreende o Editor de diagrama de blocos, a biblioteca de blocos e o sistema de serialização/desserialização. O editor de diagrama de blocos pode ser dividido em:

1. Objetos gráficos 2D: Há bibliotecas que auxiliam na construção deste bloco, porém foi necessário implementar funções que tratam de eventos de *mouse*, como por exemplo a funcionalidade de *scroll* quando o botão do meio é girado, ou deslocar a visualização quando clicar com o botão esquerdo do mouse em um espaço vazio e arrastar.
2. Cenas: esta parte é responsável por gerenciar os objetos gráficos como os blocos e as conexões por meio de listas que os descrevem. Toda vez que um bloco ou conexão é adicionado ou removido a respectiva lista é atualizada. Cada vez que um bloco é adicionado, também é atualizado uma instância na parte relacionada com a simulação.

Observa-se que o Sistema de Modelagem implementa as duas formas de modelagem de sistemas descritas no capítulo 2: a gráfica, por meio do editor de diagramas de blocos, e a textual, no qual os diagramas são descritos em arquivos JSON com o sistema de serialização/desserialização.

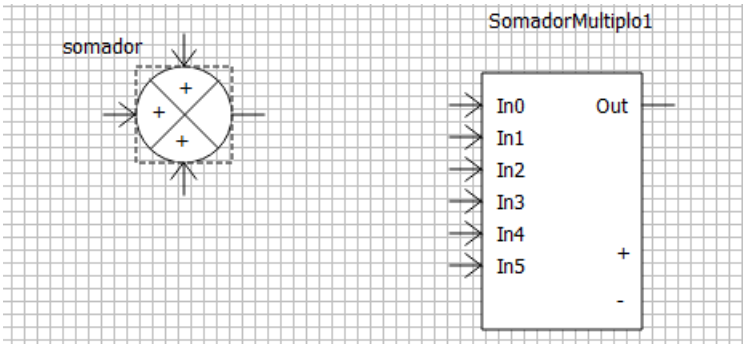
3.4.1 Biblioteca de blocos

A biblioteca de blocos contém um conjunto de funções que permite expressar os sistemas de controle mais comuns. Foi utilizado a classe base c++ *QGraphicsPathItem* fornecida pelo QT, a qual permite representar desenhos 2D como um conjunto de formas básicas, como linhas, arcos e círculos. Também é fornecida a capacidade de receber eventos de *mouse*, o que permite manipular esses objetos 2D ao reimplementar as funções que recebem os eventos de *mouse.* A classe *QGraphicsPathItem* não fornece nenhuma funcionalidade matemática para a representação de blocos de sistemas de controle, portanto foi necessário implementar essa característica nos blocos. Um destaque foi dado aos blocos Função de Transferência, responsável por representar sistemas dinâmicos lineares e o bloco de Comunicação Serial, imprescindível para proporcionar a capacidade de comunicação com *hardware* externo.

3.4.1.1 Somador

Foram implementados dois blocos somadores, um com número fixo de entradas, e seu visual é semelhante ao encontrados nos livros de teoria de controle, e outro bloco, o MultiSomador, com número variável de entradas que pode ser alterado pelo usuário clicando nos símbolos de mais e de menos. Esse bloco é importante para a implementação da função de transferência, pois conforme muda a quantidade de integradores, muda a quantidade necessária de entradas no bloco somador.

Figura 11 - A esquerda bloco somador com entradas fixas e à direita bloco com número de entradas variável.

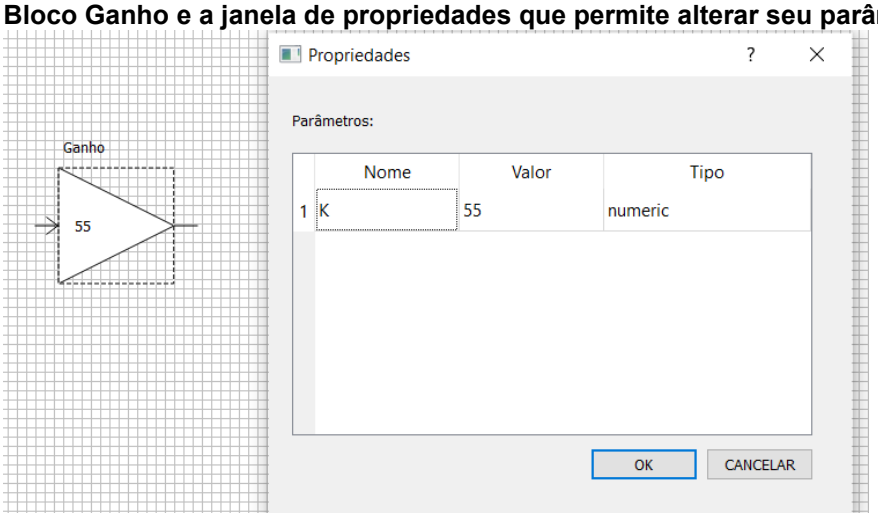


Fonte: Autoria própria.

3.4.1.2 Ganho

Este bloco realiza a multiplicação do valor da porta de entrada por seu parâmetro “K” e coloca na porta de saída.

Figura 12 - Bloco Ganho e a janela de propriedades que permite alterar seu parâmetro "K".

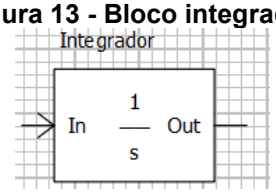


Fonte: Autoria própria.

3.4.1.3 Integrador

Este bloco é essencial para a implementação do bloco função de transferência. Sua funcionalidade é baseada na integral de Euler discreta, então o valor obtido na porta de saída é a soma entre o valor de saída atual com o valor de entrada vezes o tempo de integração.

Figura 13 - Bloco integrador.

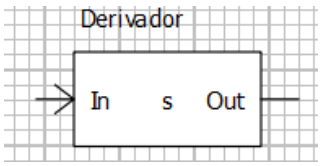


Fonte: Autoria própria.

3.4.1.4 Derivador

A funcionalidade deste bloco é baseada em equações de diferença a tempo discreto, seu valor de saída é igual ao valor de saída anterior menos o atual e dividido pelo tempo de integração.

Figura 14 - Bloco derivador.

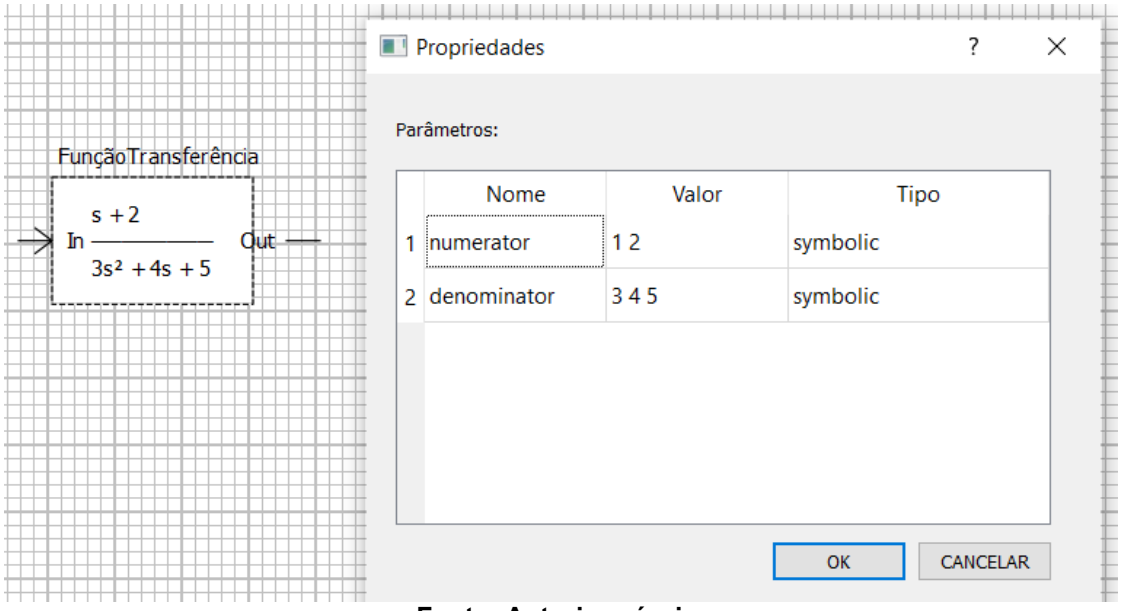


Fonte: Autoria própria.

3.4.1.5 Função de transferência

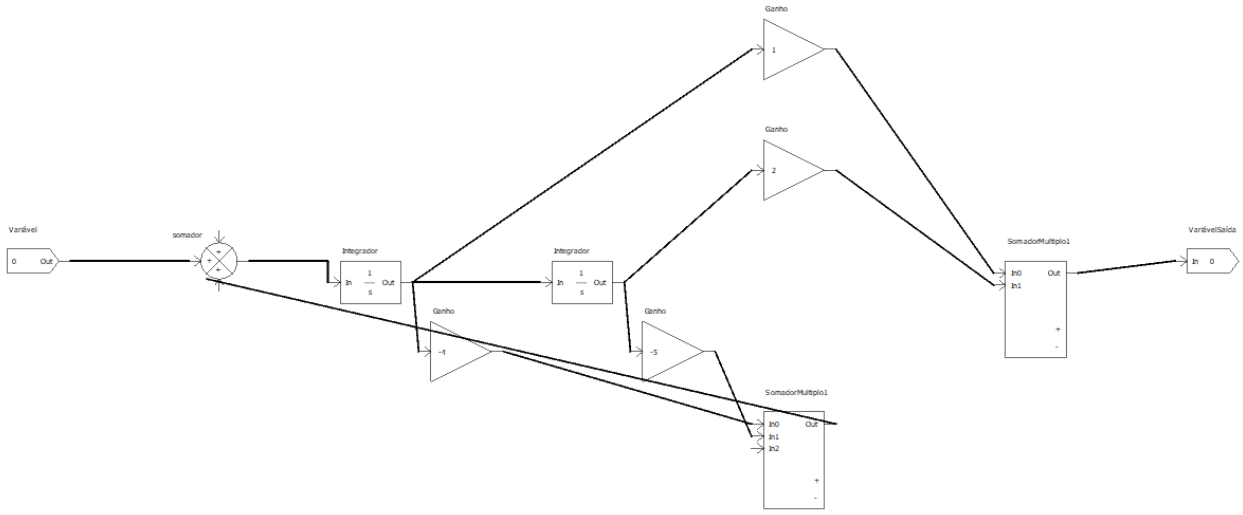
Este é o bloco mais complexo implementado. Sua funcionalidade é baseada em converter uma função de transferência em um diagrama de blocos, tal como descrito no capítulo dois. O usuário, após definir os coeficientes do numerador e do denominador da função de transferência têm a possibilidade de visualizar os blocos internos desse bloco. Assume-se que os valores iniciais de saída dos integradores começam em zero. Como internamente o bloco de função de transferência é um diagrama de blocos, é possível alterá-lo para adicionar funcionalidades como valores iniciais na saída dos integradores (por meio da adição de um somador e uma variável) ou representar funções não lineares, no entanto, na implementação atual, ao alterar o diagrama interno, a descrição da equação do bloco não é alterada.

Figura 15 - Bloco Função de transferência e a caixa de diálogo na qual é possível alterar os coeficientes do numerador e do denominador.



Fonte: Autoria própria.

Figura 16 - Visualização interna do bloco de transferência da Erro! Fonte de referência não encontrada..

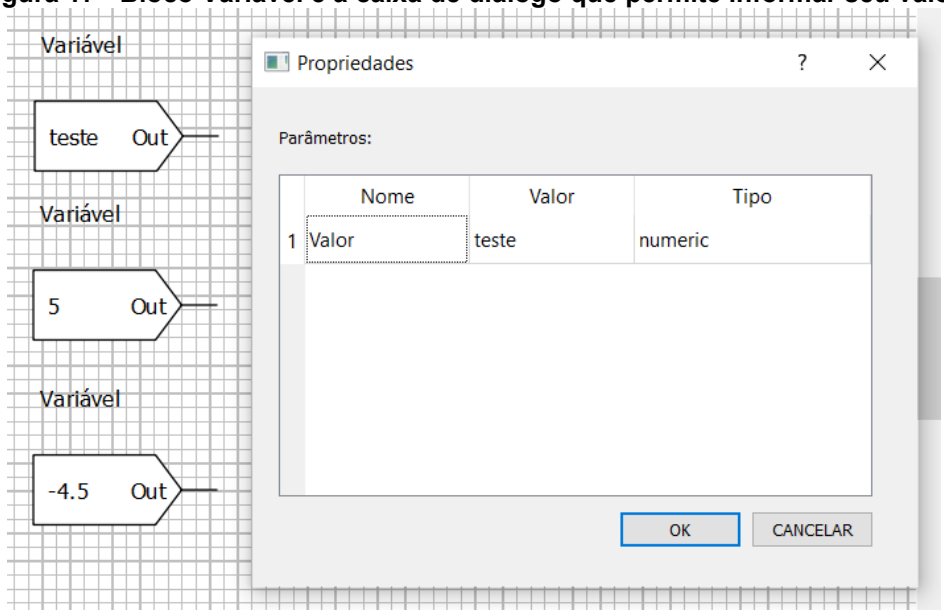


Fonte: Autoria própria.

3.4.1.6 Variável

O bloco *Variável* permite a inserção de valores de entrada no diagrama de blocos, podendo armazenar tanto valores numéricos quanto caracteres ou cadeias de caracteres. Seu valor pode ser alterado dinamicamente enquanto a simulação está acontecendo, permitindo, por exemplo, aplicar um degrau na entrada de uma função de transferência. Ao mesmo tempo serve como interface entre o usuário e o sistema de controle implementado.

Figura 17 - Bloco Variável e a caixa de diálogo que permite informar seu valor.

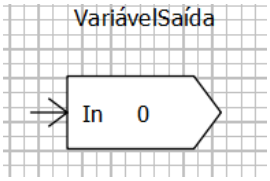


Fonte: Autoria própria.

3.4.1.7 Variável de saída

Este bloco permite visualizar valores de saída de outros blocos, funcionando como interface de visualização e inspeção do funcionamento do sistema de controle.

Figura 18 - Bloco Variável de Saída.

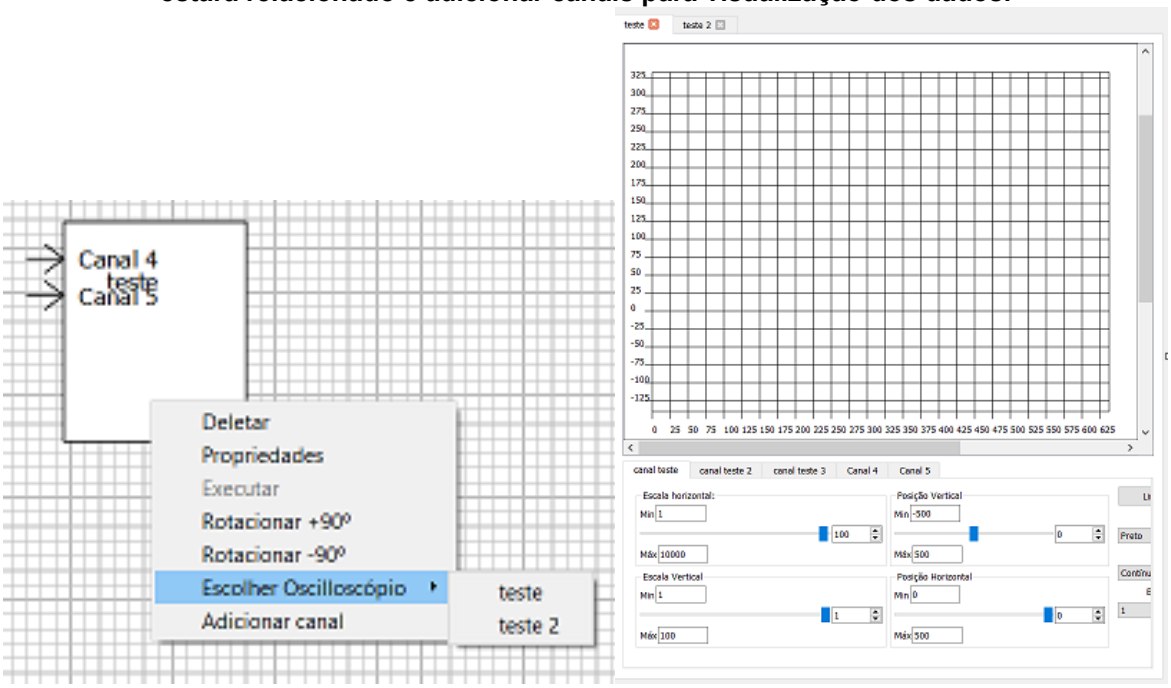


Fonte: Autoria própria.

3.4.1.8 Osciloscópio

Este bloco relaciona valores de saída a canais nos gráficos do sistema de visualização. Ao clicar com o botão direito em cima do bloco, um menu é mostrado, no qual pode-se escolher em qual gráfico os valores serão visualizados e qual canal.

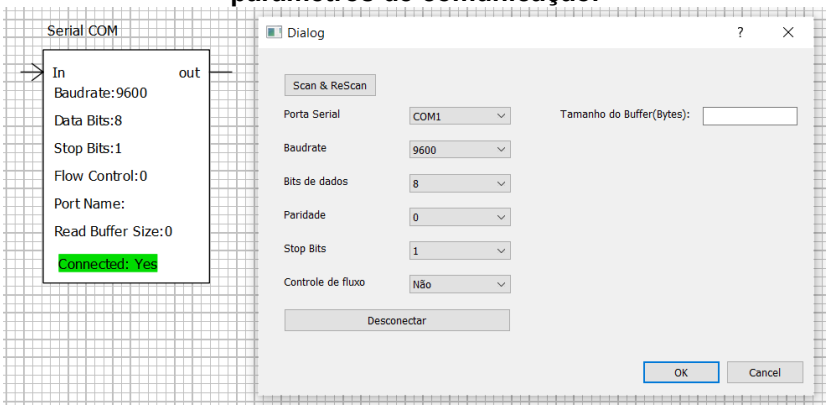
Figura 19 - Bloco Osciloscópio. É possível ver o menu onde pode-se escolher a qual gráfico estará relacionado e adicionar canais para visualização dos dados.



Fonte: Autoria própria.

Este bloco permite a comunicação serial através das portas USB disponíveis no computador. Ao clicar com o botão direito do mouse e escolher a opção “Propriedades” é possível modificar os parâmetros de comunicação serial, bem como conectar ou desconectar à porta serial. É um bloco importante para a prototipagem de sistema de controle, pois ele que faz a interface entre a planta e o sistema de controle modelado.

Figura 20 - Bloco de comunicação serial e a janela de propriedades que permite escolher os parâmetros de comunicação.

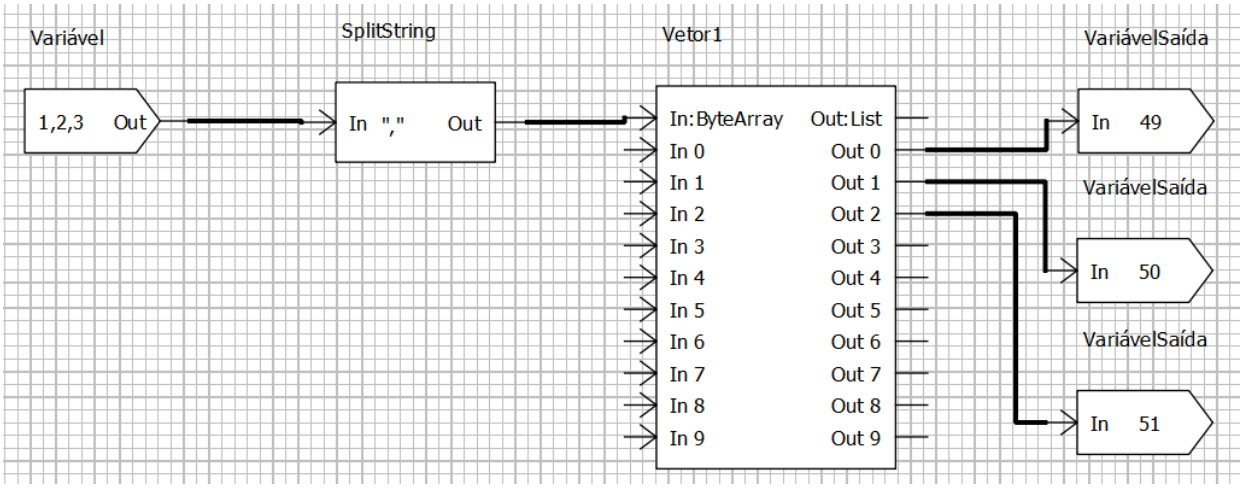


Fonte: Autoria própria.

3.4.1.9 String split

Este bloco permite dividir uma cadeia de caracteres em várias partes. Essa divisão é feita conforme um caractere divisor a ser informado, por exemplo vírgula ou ponto. Essa funcionalidade é útil ao processar os dados provenientes do bloco de comunicação serial.

Figura 21 - Bloco Split String e exemplo em funcionamento. Os caracteres são convertidos nos valores inteiros da tabela ASCII.



Fonte: Autoria própria.

3.4.1.10 Vetor

Este bloco possui funcionalidade parecida à um multiplexador e demultiplexador. Se os dados que chegarem à sua porta “In:ByteArray” forem uma

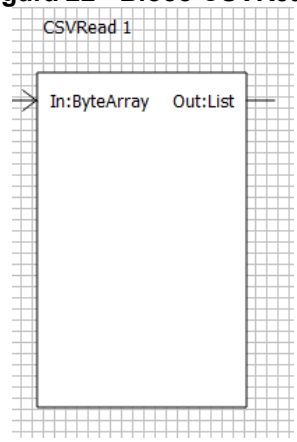
lista de valores (QByteArray – uma classe de objetos do QT), ele colocará cada valor em uma porta de saída, funcionando assim como um vetor no qual é possível acessar um índice individualmente. Se valores forem colocados nas portas “In0” até “In9”, ele construirá uma lista de valores acessível na porta de saída “Out:List”.

Essas funcionalidades são úteis para tratar os dados recebidos da porta serial, e também para montar sequências de dados a serem enviados para a serial, como mostra a Figura 21.

3.4.1.11 Csv read

Este bloco recebe um fluxo de dados no formato CSV (Comma Separated Values), e constrói uma lista de valores. É útil para ao receber valores no formato CSV enviados a partir da porta serial. Dessa forma é possível implementar protocolos simples de comunicação baseados em comunicação serial.

Figura 22 - Bloco CSVRead.

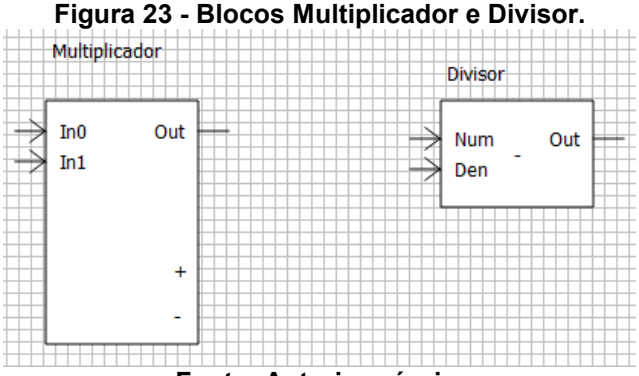


Fonte: Autoria própria.

3.4.1.12 Multiplicador e Divisor

O bloco Multiplicador possui quantidade variável de portas de entrada, sendo no mínimo duas, e efetua a multiplicação entre todos os valores. Já o bloco Divisor efetua a divisão entre o valor na porta “Num” pelo valor da porta “Den”. Não foi implementado formas de tratar erro de divisão por zero, de forma que se ocorrer o valor da saída será zero.

Figura 23 - Blocos Multiplicador e Divisor.

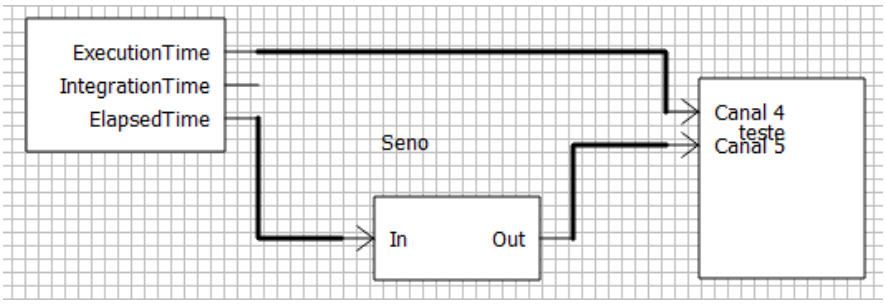


Fonte: Autoria própria.

3.4.1.13 Model Info

Este bloco permite obter, em tempo de execução, informações relativas à simulação, como o tempo transcorrido, o tempo de integração e o tempo demandado na execução do ciclo anterior. Com o tempo transcorrido e um bloco de função matemática é possível implementar um gerador de sinais. O tempo de execução fornece um diagnóstico do desempenho do sistema de controle, informando a necessidade de aumentar o tempo de integração ou a possibilidade de diminuir.

Figura 24 - Exemplo de aplicação com o bloco ModelInfo (primeiro bloco da esquerda para a direita).



Fonte: Autoria própria.

3.4.1.14 Outros Blocos

Foram implementados os seguintes blocos de funções matemáticas: seno, cosseno, tangente, exponencial e raiz quadrada. Foram acrescentados ainda blocos lógicos e comparadores, com funções E, OU, NÃO, comparador Menor, Igual, Maior e Diferente.

3.5 Sistema de serialização e desserialização

O sistema de serialização e desserialização é responsável por salvar os modelos e carregar modelos salvos. Além disso, o sistema de visualização também deve ter suas informações salvas, como por exemplo as configurações dos gráficos, do contrário toda vez que um modelo é carregado necessitaria configurar novamente o sistema de visualização.

O sistema de serialização/desserialização usa arquivos JSON como base para criar um arquivo de texto que permita reconstruir um diagrama de blocos e o sistema de visualização relacionado. Para isso, cada bloco, conexão, e instância de visualização terá uma descrição em JSON. O documento JSON é dividido em quatro partes: os blocos, as conexões e as configurações do sistema de visualização e simulação. Essa divisão busca imitar a forma como um sistema será modelado: primeiro adicionam-se blocos ao modelo, depois eles são conectados, e por fim o sistema de visualização e de simulação são configurados. Na prática isso significa que o arquivo JSON possui quatro objetos principais: *FunctionBlocks, Connections, SimulationInfo e Visualization*. No objeto *FunctionBlocks* é armazenado uma lista dos blocos e sua descrição. No objeto *Connections* é armazenado uma lista das conexões, contendo uma descrição, os blocos e as respectivas portas conectadas além dos pontos geométricos que definem a geometria de cada conexão. No objeto *SimulationInfo* é armazenado o tempo de integração. Embora contenha apenas uma informação nesse momento, o fato de se ter um objeto destinado a guardar as informações da simulação permite aprimoramentos futuros. Por último, no objeto *Visualization*, é armazenado uma lista dos gráficos com os respectivos canais e suas propriedades.

A descrição em JSON tem os atributos nome, posição, rotação, tipo do bloco, a descrição das portas de entradas e saídas bem como o nome de cada porta.

A implementação do sistema de serialização e desserialização foi realizada por meio da implementação das funções virtuais *serialize e desserialize* e *getType* na classe *Serializable*. A função *serialize* tem por objetivo fornecer as informações para armazenamento. Ela retorna um objeto da classe QJsonObject (fornecida pelo QT), que é uma descrição em JSON dos dados a serem armazenados. Isso permite que ao implementar uma classe derivada, possa-se reaproveitar a implementação já

feita na classe base ao chama-la na reimplementação da função *serialize*, uma vez que a implementação da classe base retorna valores, pode-se utilizá-los na reimplementação de serialize na classe derivada, Dessa forma foi possível garantir modularidade e a capacidade de expandir a biblioteca de blocos e outras partes que precisem serem armazenadas sem grandes esforços relacionados ao armazenamento das informações. Como exemplo pode-se citar a classe *FunctionBlock*, a qual retorna na função *serialize* uma descrição em JSON do nome do bloco, a rotação e a posição, assim a classe derivada *Gain*, que é responsável por implementar o bloco ganho, na função *serialize,* só tem o adicional de armazenar o valor do ganho, com os valores da posição, rotação obtidos ao chamar a implementação da classe base *FunctionBlock*.

A funcionalidade de carregar diagramas de blocos salvos é realizada pelas funções *desserialize*, *getType* e *load.* Quando um bloco é armazenado, uma das informações é o tipo do bloco, por exemplo se é um bloco somador ou uma função de transferência. Toda vez que um bloco diferente é implementado é necessário reimplementar essa função, pois ela permite que a função *load* saiba qual bloco instanciar ao abrir um arquivo JSON. A função *load* por sua vez possui uma tabela de busca, em que associa cada tipo de item a ser armazenado com uma função capaz de reconstruí-lo a partir de uma descrição JSON.

Como demonstrado, uma preocupação ao desenvolver o sistema de serialização e desserialização foi implementar de forma que permita a evolução do *software*.

3.6 O Sistema de simulação

O sistema de simulação foi implementado como um interpretador de modelo inspirado no formalismo de linguagens de programação de fluxo de dados, tal qual o Scicos/Xcos (INRIA, 2015). Cada bloco de função é implementado na linguagem C++ e a simulação consiste em gerar uma ordem de execução apropriada de cada bloco. O passo de integração necessário para implementar comportamento em tempo real é fornecido pela classe QTimer do framework Qt.

As funcionalidades do sistema de simulação são fornecidas pelas classes que descrevem um bloco funcional (entrada, saída e função), um grafo que armazena os blocos executáveis e as conexões entre eles, um algoritmo para gerar a sequência correta de execução dos blocos (compilador) e o algoritmo de

simulação que realiza as simulações *off-line* e de tempo real percorrendo a sequência de execução dos blocos, descobertas através da compilação.

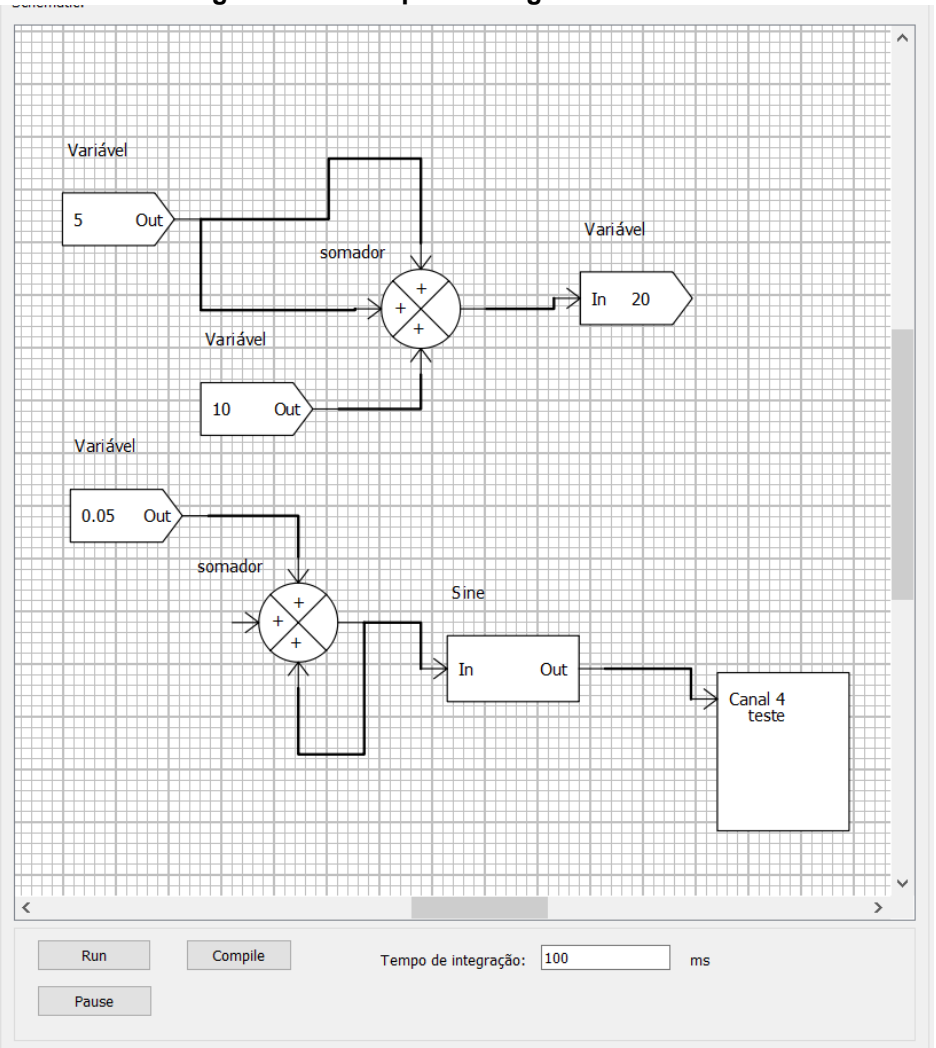
A descrição de cada bloco é feita em termos de entradas, saídas e uma função que mapeia as entradas para as saídas. Dessa forma, sistemas não lineares e funções de difícil modelagem matemática também podem ser representados, como por exemplo o bloco de comunicação serial e o bloco que faz a integração com o sistema de visualização. Isso é análogo ao encontrado em linguagens de modelagem como a Modelica, Simscape ou HDL's. Uma importante consideração é que uma saída de um bloco pode estar conectada a várias entradas de blocos seguintes, mas uma entrada só pode estar conectada a uma saída, devido à própria definição matemática do que é função.

A simulação é executada percorrendo a lista de objetos criados através do mapeamento das conexões. O código de cada objeto é executado quando a simulação chega a ele, de forma que há um valor de entrada e um valor de saída após a execução do bloco. Essa forma de simulação permite que um modelo seja alterado com a simulação em andamento, constituindo assim um sistema RCP – *by-pass.* Para o caso de simulação *off-line*, a lista é percorrida até a quantidade de iterações informada pelo usuário, por exemplo, se for desejado simular um minuto, e o tempo de integração informado for de 100 milissegundos, a lista é percorrida 600 vezes. Para a simulação de tempo real, o sistema de simulação captura o evento gerado pelo timer do sistema operacional, que será gerado no intervalo de tempo definido pelo usuário. A cada disparo do *timer*, todos os blocos da lista são percorridos. Observe que todo o diagrama deve ter sua execução garantida nesse intervalo de tempo, do contrário, não serão satisfeitos os requisitos de tempo real.

3.6.1 O compilador

O compilador é responsável por gerar a ordem de execução da simulação dos blocos através da análise das conexões entre os blocos. Para isso as listas de blocos e as dependências são analisadas a fim de criar a sequência correta de execução do diagrama em questão. Para demonstrar melhor o funcionamento do compilador, será analisado o diagrama construído na Figura 25.

Figura 25 - Exemplo de diagrama de blocos.



Fonte: Autoria própria.

O diagrama construído na Figura 25 possui dois ramos independentes. O primeiro é constituído por duas variáveis conectadas à entrada de um somador, e uma variável de saída conectada à saída do somador. O segundo ramo contém uma variável, um somador realimentado, um bloco seno e um bloco Osciloscópio, que integra o sistema de visualização.

O programa possui uma lista dos blocos que não tem nenhum outro conectado à sua saída ou que não tem saídas, de forma que para o exemplo dado essa lista têm a variável de saída com valor 20 e o bloco Osciloscópio.

A compilação inicia analisando a variável com valor 20, em seguida verifica que há o bloco somador conectado a ela, então novamente verifica que as três entradas do somador estão ocupadas. Observe que duas entradas se referem ao mesmo bloco, a variável com valor 5. Conforme o diagrama é percorrido, uma lista

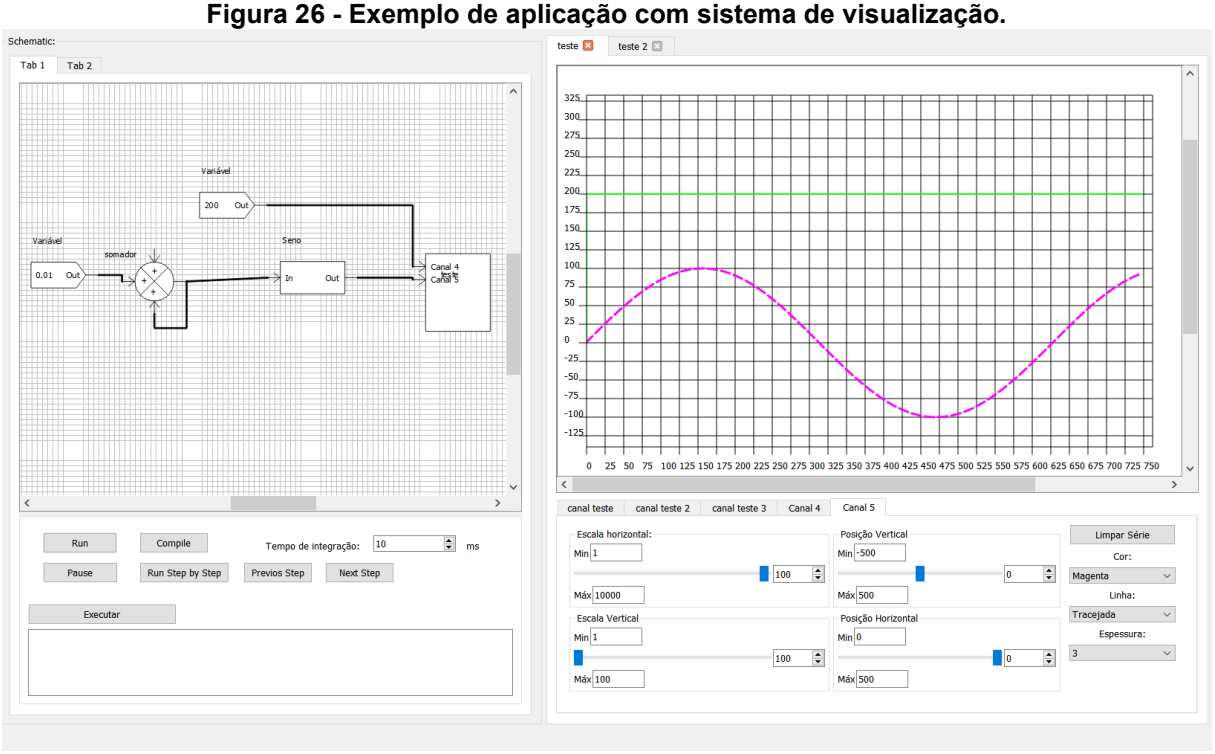
com os blocos que já foram verificados é atualizada e utilizada para que não haja referências repetidas de um bloco na lista de execução, pois isso faria que ele fosse executado mais de uma vez. Após analisar o diagrama a lista de execução poderá ter a seguinte forma: Variável 5, variável 10, somador, Variável 20, variável 0.05, somador, seno, Osciloscópio.

3.7 Sistema de visualização

Optou-se por desenvolver um sistema de visualização simples, sem a possibilidade de construção de interfaces customizadas, tal como o Matlab ou Labview, porém funcional. Para isso, foi escolhido um sistema que permita múltiplas visualizações por meio de vários osciloscópios virtuais, os quais permitem a visualização de múltiplas variáveis simultaneamente. O sistema de visualização funciona em conjunto com o bloco Osciloscópio, que é o responsável por pegar os valores da simulação e fornecer à respectiva série de dados, aqui chamada de “canal”.

Cada Osciloscópio virtual pode ter vários canais, com cada canal podendo ter configurações como tipo, largura e cor da linha, deslocamento vertical e horizontal e escala horizontal e vertical, como mostra a Figura 26. Para uma melhor visualização, foram implementados controles de escala para as séries de dados, sendo eles a escala vertical, horizontal e os deslocamentos horizontal e vertical. Os valores no eixo horizontal representam o tempo transcorrido em segundos, alterar o controle de escala horizontal para 100 significa nesse caso que um intervalo de 100 plotado é referente a 1 segundo de simulação. Para uma verificação dos valores foi implementado uma função em que ao posicionar o mouse sobre uma série de dados plotada, os valores em x (eixo horizontal) e y (eixo vertical) são mostrados.

Figura 26 - Exemplo de aplicação com sistema de visualização.



Fonte: Autoria própria.

4 RESULTADOS

Foi implementado o *software* CACSD para prototipagem rápida de sistemas dinâmicos de acordo com a arquitetura definida previamente e os passos descritos. Para validar o funcionamento do *software* foram feitos testes, divididos em 3 partes:

1. Testes de performance da execução em tempo real: Por meio do bloco *ModelSimulationInfo* é possível obter o tempo de execução de cada ciclo. Com esse dado é possível obter a razão entre o tempo de execução de cada ciclo e o tempo de integração. Foram realizados testes com diferentes números de blocos e gráficos.
2. Validação dos resultados de simulação: Foram construídos diversos diagramas e, por meio do sistema de visualização, foram comparados os valores obtidos com sistemas equivalentes simulados no *software* Matlab. Foi dado especial atenção ao bloco Função de Transferência, uma vez que é parte fundamental para que o *software* simule de forma fidedigna o sistema modelado.
3. Exemplo de aplicação: Nesta seção um exemplo de sistema de controle é aplicado de forma a servir de guia para uso e também demonstrar a capacidade do *software* desenvolvido de modelar e executar um sistema de controle.

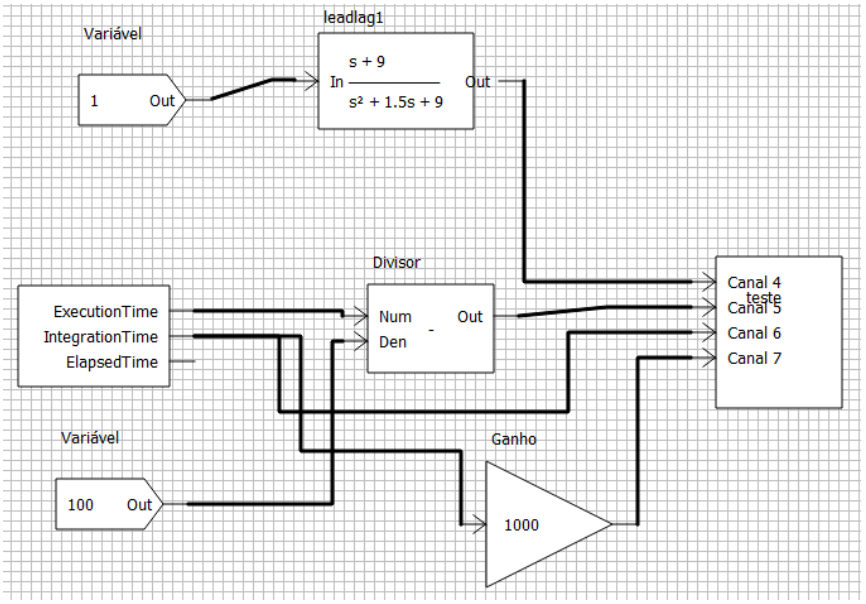
Os testes foram executados em um notebook com processador AMD Ryzen 5 4600H (3 a 4GHz), 8Gb de memória DDR4 3200 MHz, 256Gb de armazenamento SSD e sistema operacional Windows 10.

4.1 Testes de performance de tempo real

Apresenta-se, dois testes de performance. Os diagramas considerados são apresentados na Figura 27 e Figura 28. No primeiro caso, mais simples, foi utilizado tempo de integração de 10ms, enquanto no segundo caso, com grau de complexidade muito maior, foi utilizado 100ms.

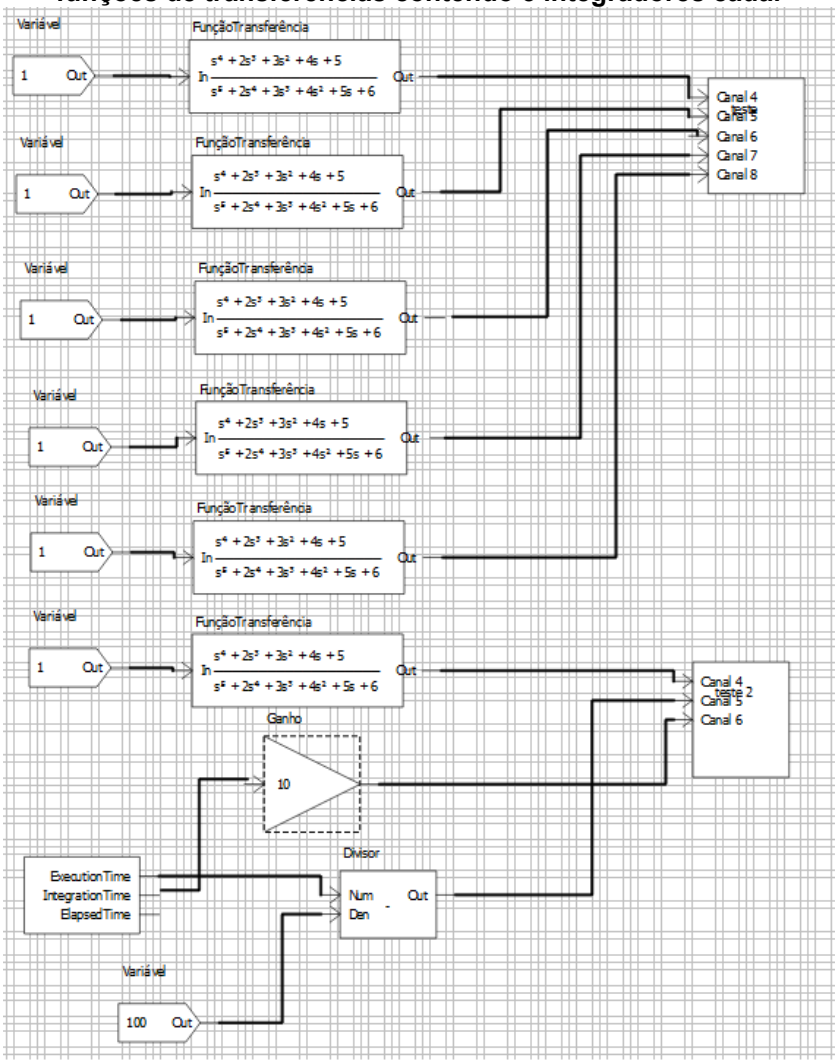
.

Figura 27 - Diagrama 1 analisado no teste de performance de tempo real. Tempo de integração de 10ms.



Fonte: Autoria própria.

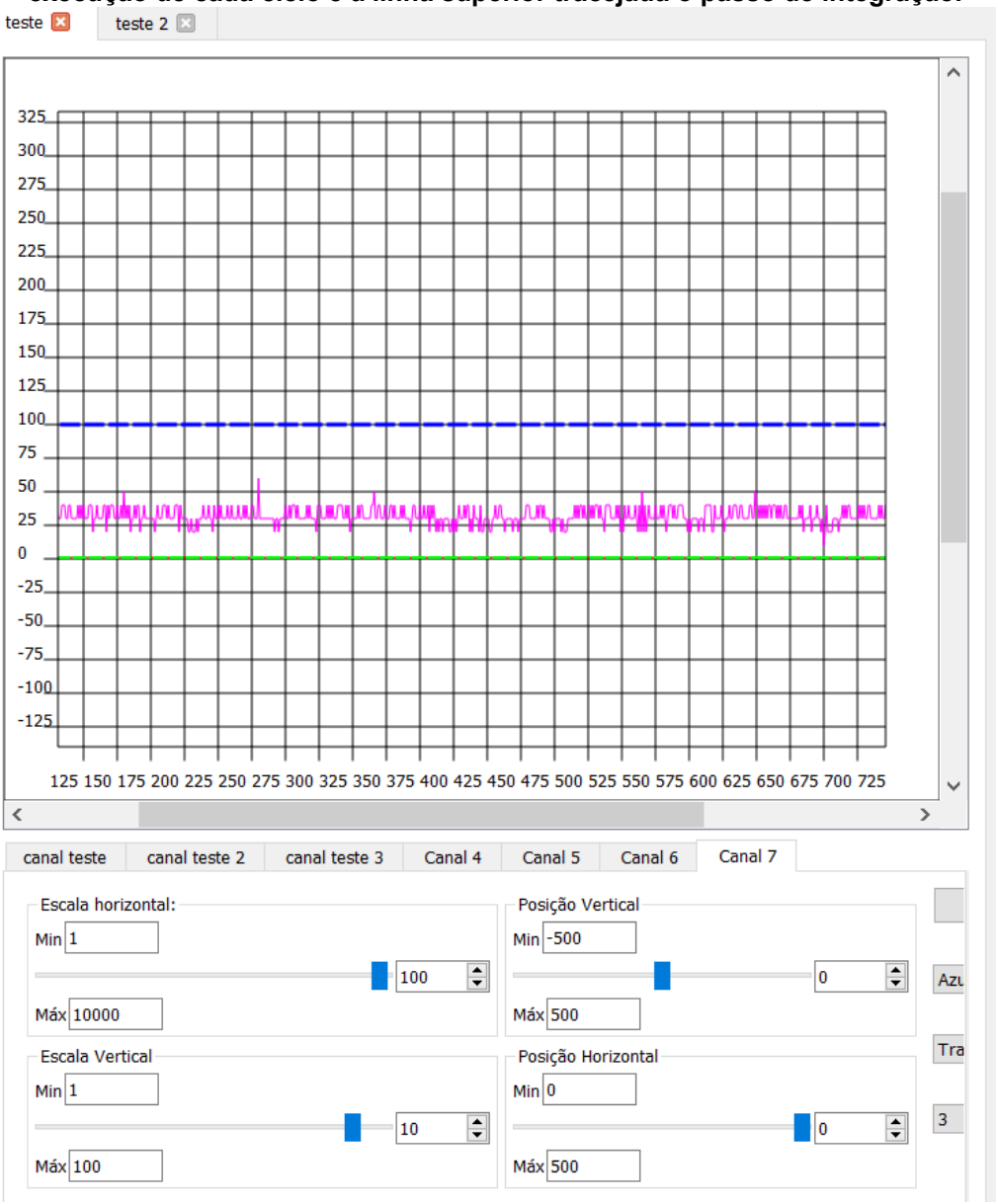
Figura 28 - Diagrama 2 analisado no teste de performance de tempo real. Utilização de 6 funções de transferências contendo 5 integradores cada.



Fonte (autoria própria).

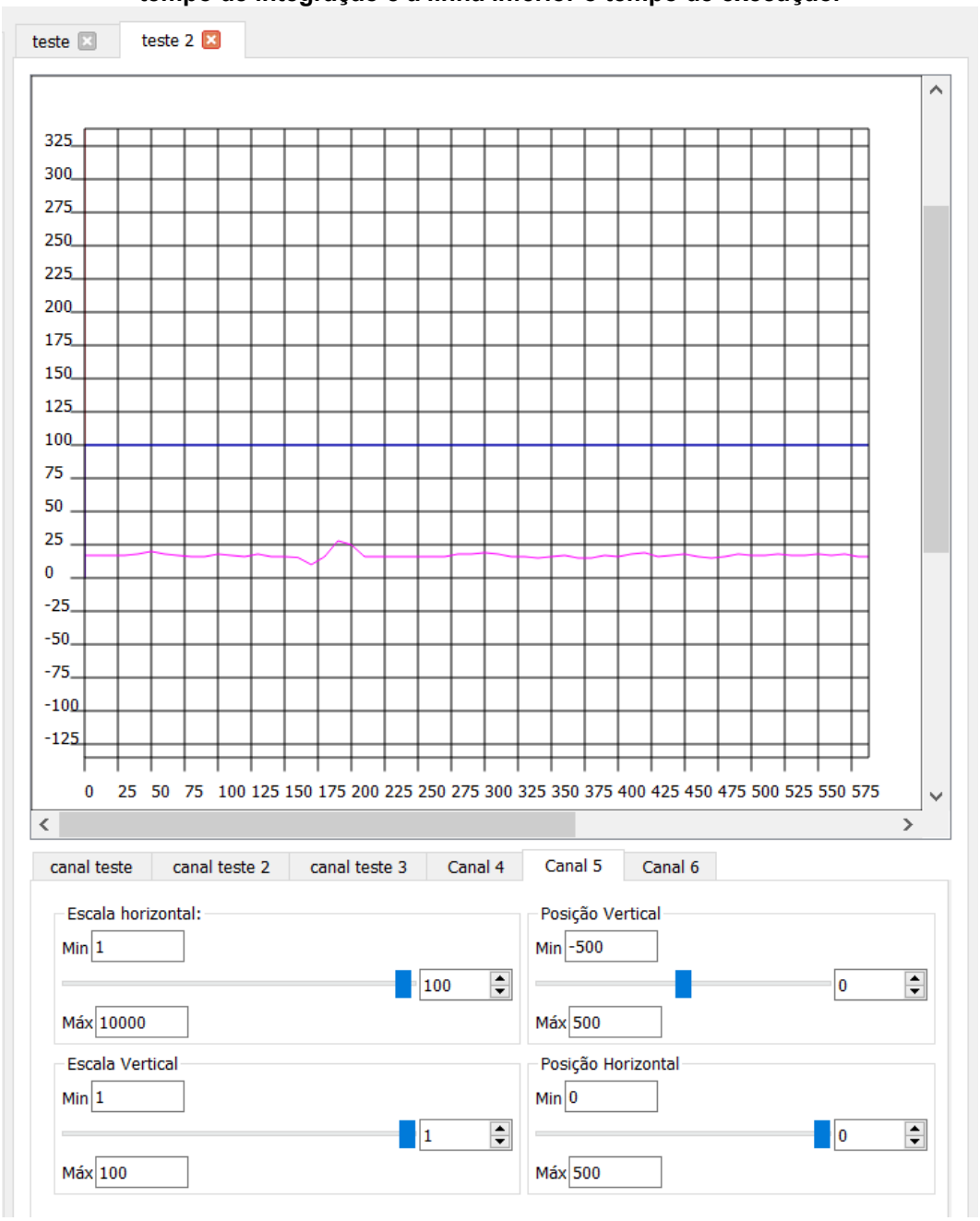
O gráfico da Figura 29 mostra o tempo de iteração para o primeiro teste de performance. Nota-se um valor médio de 2,5 milissegundos de execução, o que corresponde a 20% do tempo de integração. Da mesma forma, na Figura 30**Erro! Fonte de referência não encontrada.** é mostrado o tempo de iteração do segundo teste, que atingiu, em média, o valor de 20 milissegundos. Neste segundo caso o tempo de iteração não poderia ser o de 10ms como no primeiro caso, pois acarretaria em não cumprir o conceito básico de que o tempo de iteração deve ser menor que o tempo de integração.

Figura 29 - Resultado do primeiro teste de performance, a segunda linha indica o tempo de execução de cada ciclo e a linha superior tracejada o passo de integração.



Fonte: Autoria própria.

Figura 30 - Resultado do segundo teste de performance. A primeira linha superior representa o tempo de integração e a linha inferior o tempo de execução.



Fonte: Autoria própria.

4.2 Validação dos resultados de simulação

Nesta etapa foram construídos diagramas iguais na ferramenta desenvolvida e no Simulink, e os resultados foram então comparados. Também foi analisado a capacidade do sistema de simulação em realizar simulações de tempo real por meio da verificação de tempos de simulação e latência. Para os testes de simulação em tempo real serão desenvolvidas rotinas na linguagem C++ que serão capazes de avaliar o tempo de execução de cada iteração, e a variação do mesmo.

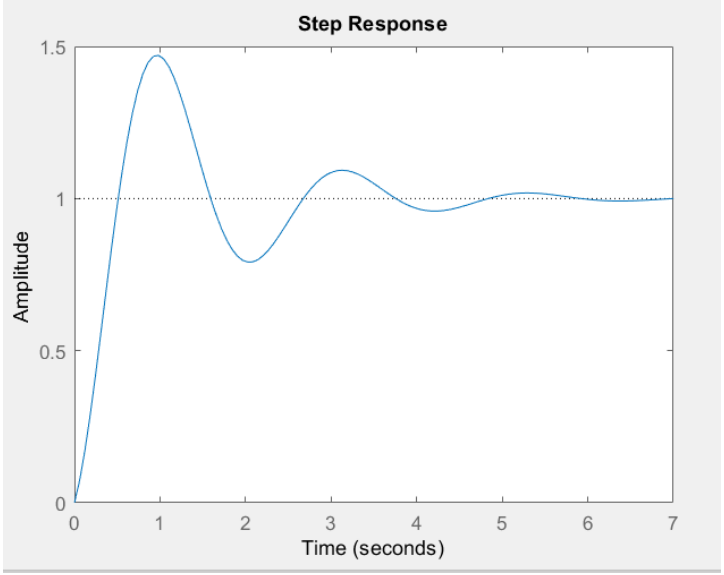
O principal objetivo nesta etapa dos testes é verificar o funcionamento dos blocos Função de Transferência, uma vez que eles são essenciais para a descrição de sistemas dinâmicos. Os testes consistem em criar funções de transferência no Matlab, simular entradas ao degrau e comparar visualmente com os resultados obtidos no *software* desenvolvido. Embora foram feitos muitos testes, para exemplificar será mostrado apenas dois testes referentes a uma mesma função de transferência.

Figura 31 - Código no Matlab para obter a resposta ao degrau em malha aberta de função de transferência "g".

```
>> s=tf('s');
>> g=(s+9)/(s^2+1.5*s+9)

g =

       s + 9
  ---------------
  s^2 + 1.5 s + 9

Continuous-time transfer function.

>> step(g)
```

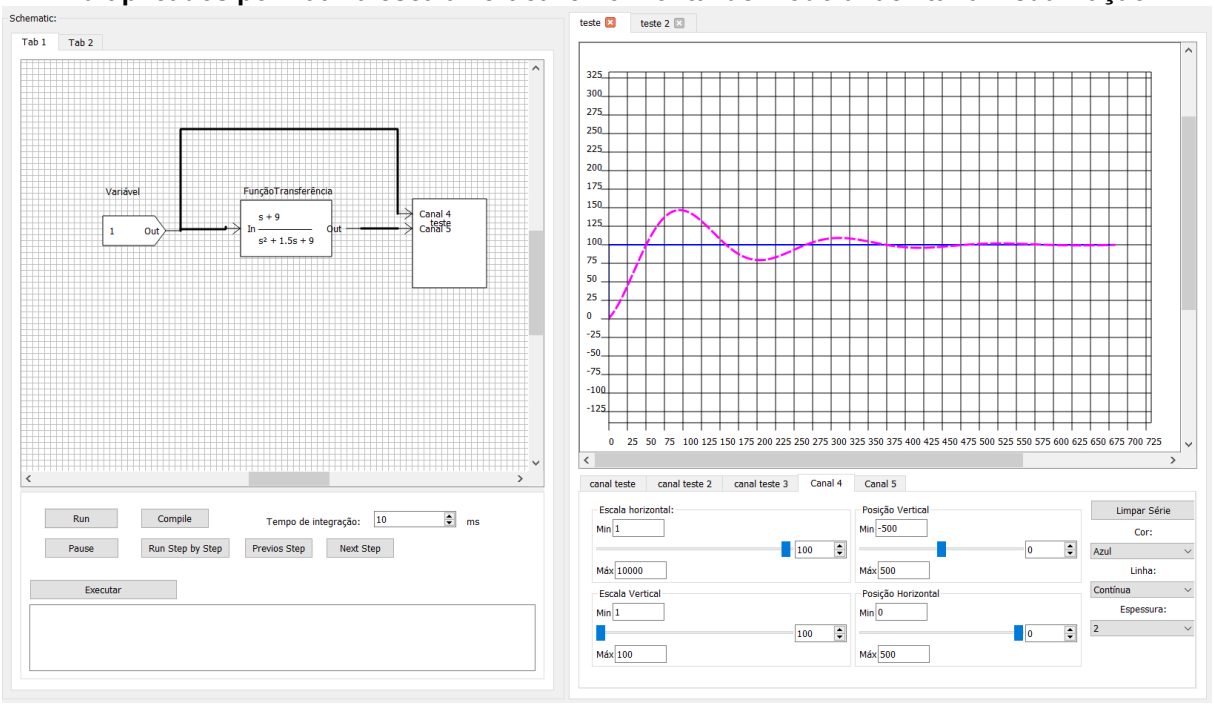
Fonte: Autoria própria.

Figura 32 - Resultado da resposta ao degrau em malha aberta da função de transferência realizado no Matlab da função de transferência "g".



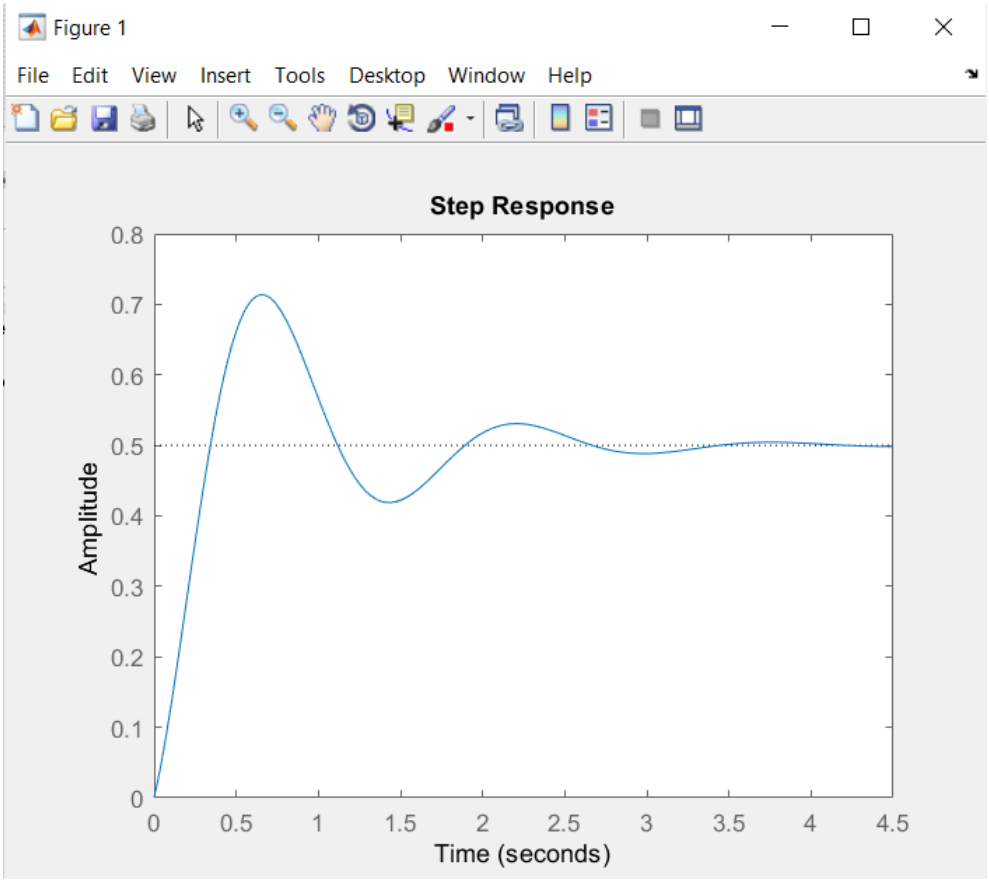
Fonte: Autoria própria.

Figura 33 - Teste de resposta ao degrau em malha aberta de função de transferência. Os valores obtidos estão condizentes com a simulação no Matlab da função "g". Os valores estão multiplicados por 100 na escala vertical e horizontal de modo a facilitar a visualização.



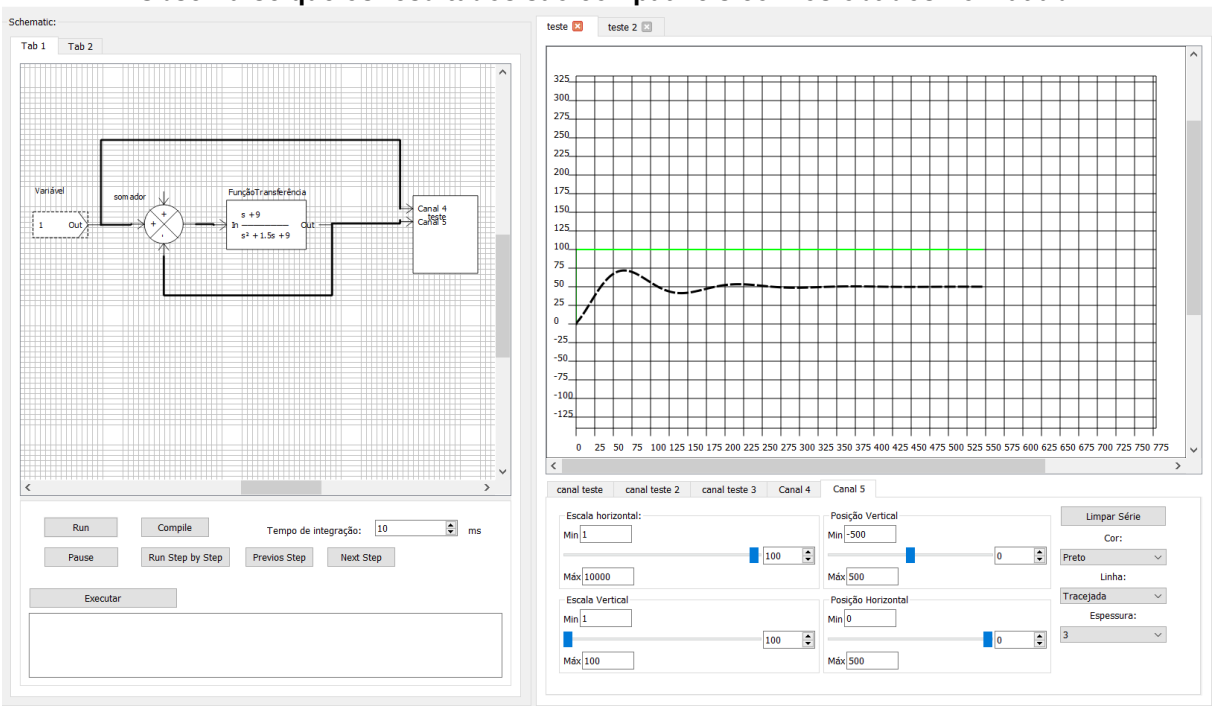
Fonte: Autoria própria.

Figura 34 - Resposta ao degrau em malha fechada da função de transferência "g" obtida no Matlab.



Fonte: Autoria própria.

Figura 35 - Resposta ao degrau em malha fechada obtida de função de transferência. A linha tracejada refere-se a sáida da função de transferência, a entrada é a linha verde e contínua. Observa-se que os resultados são compatíveis com os obtidos no Matlab.



Fonte: Autoria própria.

Observa-se que tanto para malha aberta quanto em malha fechada os resultados obtidos no *software* foram compatíveis com os obtidos no Matlab.

4.3 Exemplo de aplicação

Com o intuito de demonstrar as capacidades do *software* desenvolvido em implementar um sistema de controle, foi modelado e simulado em tempo real um sistema, no qual têm-se uma função de transferência, constituindo a dinâmica do sistema, tanto a planta quanto o controlador, um bloco de comunicação serial para a comunicação com a parte física, e os blocos “CSVRead” e “Vetor” para o tratamento dos dados provenientes da comunicação serial. Um bloco “Osciloscópio” foi adicionado para se verificar visualmente a saída do sistema. Um bloco “Variável de Saída” foi adicionado para verificar o valor proveniente da comunicação serial. O sistema modelado é apresentado na Figura 36.

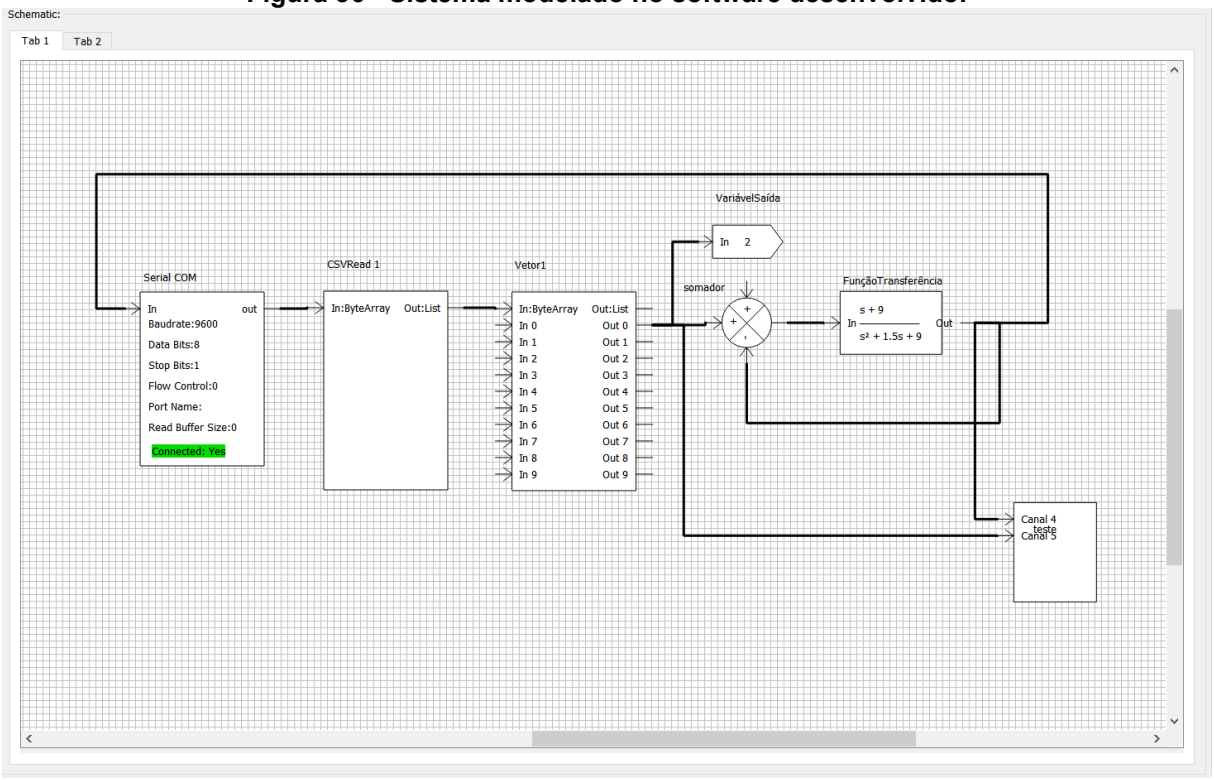
A comunicação serial foi feita via *software*, utilizando-se de um programa para criar um par de portas seriais conectadas, COM1 e COM2. Esse par funciona

no *hardware* das portas USB reais do notebook utilizado. Na Figura 37 e Figura 38 pode-se verificar as configurações de ambas as portas seriais. A partir da interface do *software* terminal foram enviados os valores desejados para a saída da função de transferência.

Observa-se que da comunicação serial é retirado somente o valor desejado para a saída do sistema, porém em uma aplicação com uma planta física, a função de transferência representaria somente o controlador, e pela porta serial seriam obtidos os valores de sensores. No exemplo da Figura 36 apenas uma função de transferência é simulada, porém uma planta a ser controlada pode ter múltiplos processos, com um controlador para cada um, modelado e obtendo os dados a partir das portas de saída do bloco Vetor.

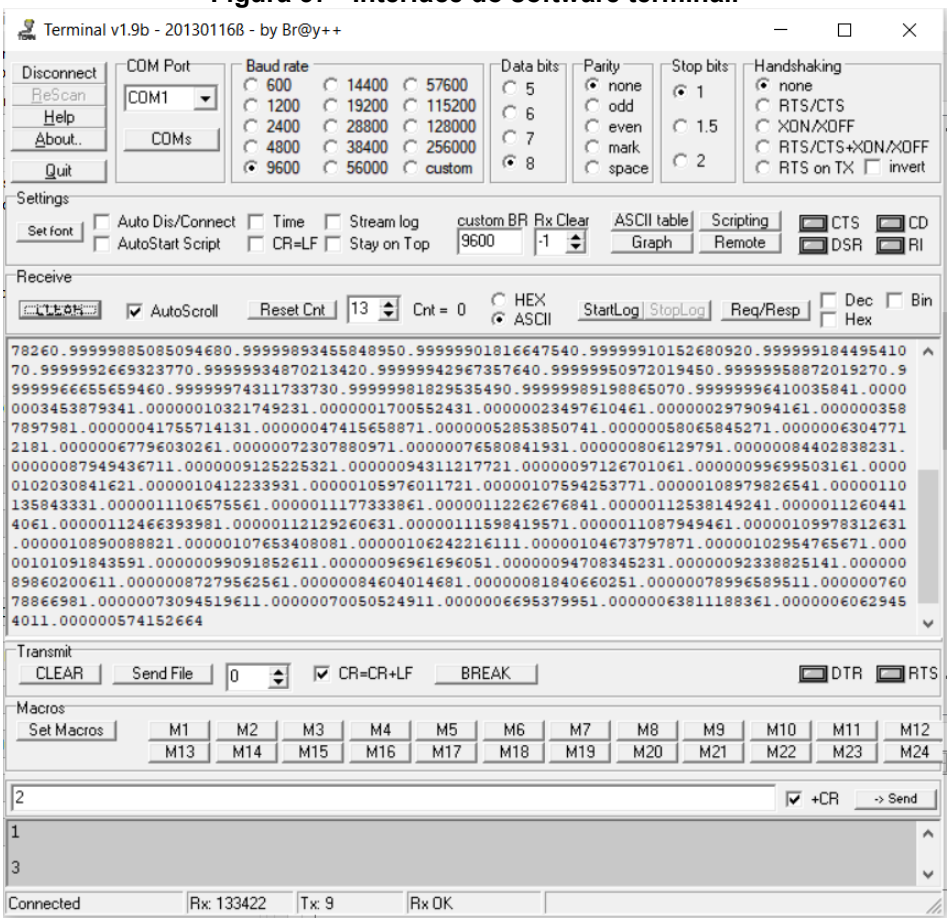
A parte dinâmica do sistema modelado, a função de transferência, foi escolhida de forma arbitrária, apenas para exemplificar o funcionamento do *software*. No entanto ao aplicar em um sistema real, é necessário executar as etapas de identificação do sistema a ser controlado para então poder projetar o controlador, etapas que não foram feitas no *software* desenvolvido, em virtude de seu escopo se restringir às etapas de modelagem e implementação do MBD.

Figura 36 - Sistema modelado no software desenvolvido.



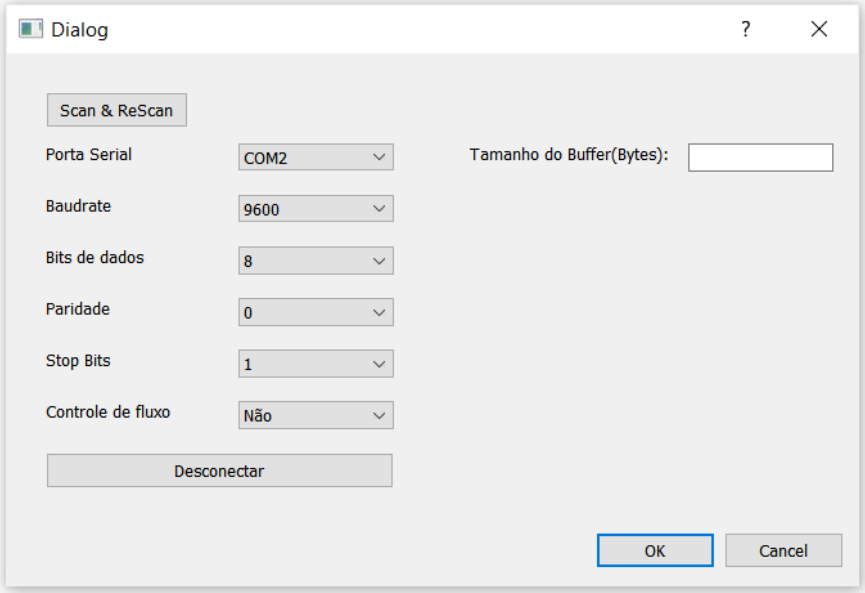
Fonte: Autoria própria.

Figura 37 - Interface do software terminal.



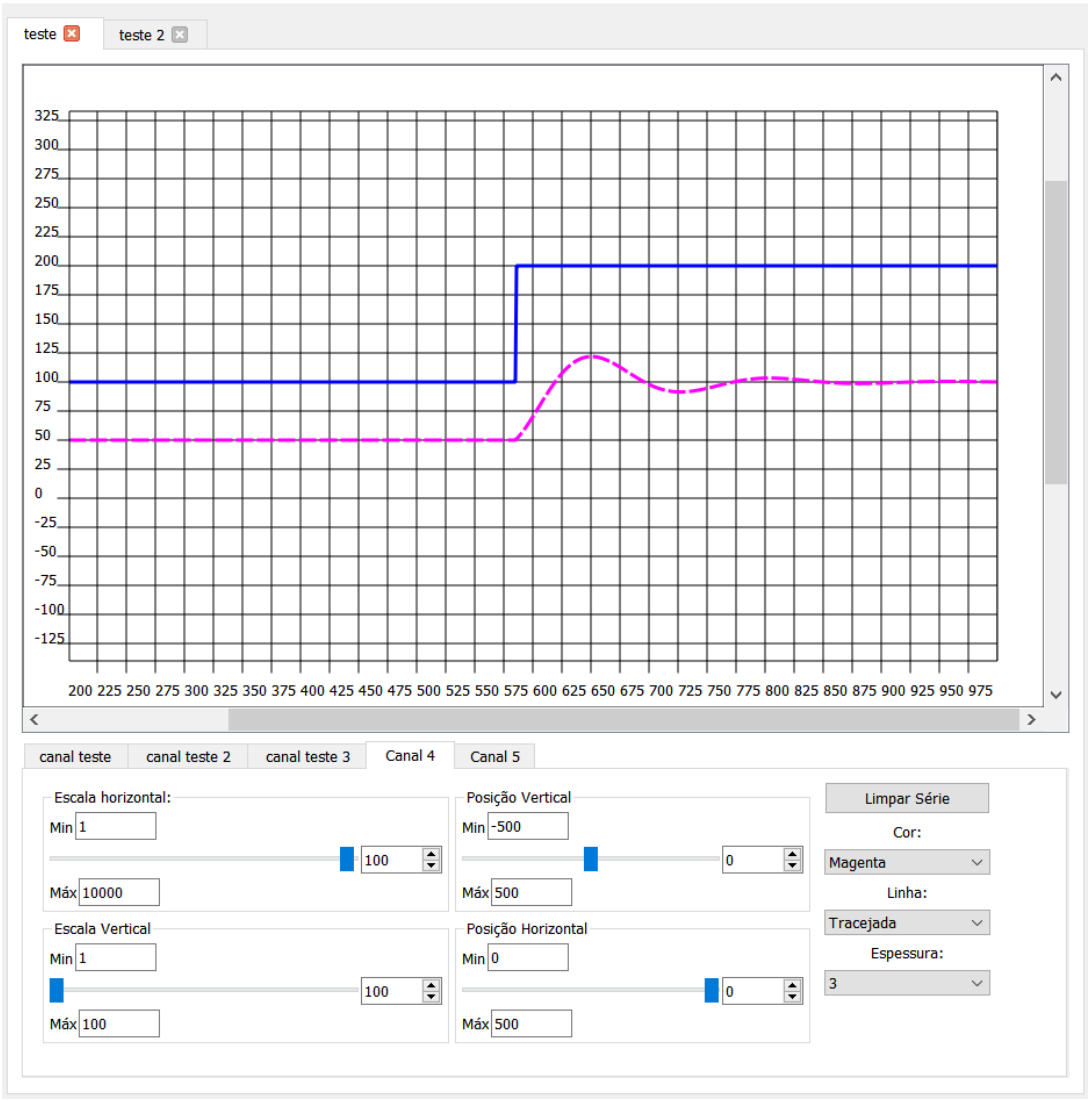
Fonte: Autoria própria.

Figura 38 - Configuração do bloco de comunicação serial.



Fonte: Autoria própria.

Figura 39 - Visualização dos valores de simulação. Observa-se o momento em que na interface do software Terminal o valor 2 é enviado (linha em azul contínua). Na linha magenta tracejada a saída do sistema simulado.



Fonte: Autoria própria.

5 CONCLUSÃO

Atualmente existe uma variedade de ferramentas e esforços que buscam auxiliar no projeto de sistemas de controle, para que se possa diminuir o tempo de implantação. Uma solução que busca mitigar esse problema é a utilização das técnicas de MBD e RCP. Como esta solução envolve identificar, analisar, projetar, testar e implantar sistemas de controle, tarefas que requerem conhecimento de várias áreas de estudo, exigindo compreensão tanto em modelagem matemática quanto em computação, construir uma ferramenta que possa implementar MBD e RCP torna-se uma tarefa complexa. Embora existam outros trabalhos em que ferramentas de modelagem são apresentadas, os detalhes e as decisões de projeto de implementação normalmente não são abordados. Neste sentido, o projeto proposto contribui com trabalhos futuros ao apresentar um roteiro de como implementar uma ferramenta deste tipo, com o referencial teórico servindo como base para estudos mais aprofundados e aprimoramentos na ferramenta desenvolvida.

Ainda nesse contexto, destaca-se o desenvolvimento e apresentação do sistema de serialização/desserialização e o sistema de simulação como as maiores contribuições deste trabalho. A forma como o sistema de serialização funciona serve como modelo para outros tipos de *softwares*, bem como sua analogia com as linguagens de modelagem permite que aprimoramentos bem como estudos formais possam ser realizados nesse sentido. A implementação relativamente simples do sistema de simulação possibilita que seja implantado em *hardware* embarcado com características de tempo real, além de que modificações no compilador possam ser utilizadas para a geração de código c/c++ ou HDL, possibilitando por exemplo a execução mais rápida dos modelos de controle bem como a implantação definitiva do sistema modelado.

No decorrer do desenvolvimento do software a preocupação em torná-lo modular esteve presente, de forma a prepará-lo para trabalhos futuros, como o incremento da biblioteca básica de blocos, as ações possíveis para os blocos e a otimização da sua performance de tempo real.

Pode-se afirmar que o *software* desenvolvido permite a adoção parcial do MBD, com as etapas de modelagem, teste, simulação e implantação de sistemas de

controle além de permitir RCP *by-pass* por meio do sistema de simulação. Isto significa que as etapas de análise e identificação de sistemas ainda devem ser feitas com ferramentas complementares, tais como as já mencionadas no texto. Conforme explicado no capítulo 2, em ferramentas CACSD, a análise matemática e a identificação de sistemas geralmente é realizada por linguagens de script próprias para computação científica, tais como Matlab, Scilab, Python, Júlia e R. Então, fazendo a integração entre o *software* desenvolvido e alguma das linguagens de *scripts* mencionadas possibilitaria realizar todo o processo de MBD nesta ferramenta.

Os resultados das simulações de sistemas dinâmicos foram condizentes com o *software* Matlab, porém vários aspectos de *softwares* estado da arte em simulação não foram implementados, tais como o fato de que os sistemas aqui simulados não levam em conta as condições iniciais em integradores e derivadores. Essas condições podem ser simuladas, mas o usuário deve por si mesmo implementá-las modificando a estrutura interna dos blocos função de transferência, ou construir funções de transferência a partir dos blocos de integradores e derivadores, ganhos e somadores. A implementação dos blocos de função de transferência a partir da transformação das mesmas em diagramas de blocos permite que ferramentas de simulação de sistemas dinâmicos sejam implementadas sem exigir conhecimentos da teoria de espaço de estados. O fato de permitir visualizar o diagrama de blocos correspondente pode servir como auxílio no ensino de sistemas de controle, uma vez que os alunos podem utilizá-la para validar suas conversões de funções de transferência para diagrama de blocos, além da evolução dos valores das variáveis de estado ao longo de uma simulação. Outras consequências dessa abordagem é que é possível alterar a estrutura interna das funções de transferência permitindo a modelagem de funções não lineares.

Neste *software* um requisito para que a RCP seja implementada é que haja meios de colher informações da planta a ser controlada por meio de comunicação serial, isso significa que em um sistema de controle essa ferramenta implementa somente o controlador, exigindo trabalho adicional ao integrar comunicação serial com a planta.

No âmbito de sistemas de tempo real, é possível afirmar que os sistemas de controle aqui simulados em que seja necessário modelar mais de uma função de transferência devem ter um passo de integração de 100 milissegundos ou mais, para

que tenha uma boa performance. Isto é suficiente para sistemas de controle de temperatura e pequenas plantas de controle, como as normalmente utilizadas nos laboratórios de ensino, por exemplo o controle de um pêndulo invertido. Observou-se que o tempo de execução entre cada iteração varia, então é necessária uma futura análise matemática das consequências dessa condição. Essa variação no tempo de execução está relacionada com a quantidade de blocos do sistema modelado, dessa forma faz-se necessário executar ciclos de simulação para cada modelo, com o bloco *ModelSimulationInfo* fornecendo essa informação, a fim de alterar o tempo de integração da simulação para otimizar seu desempenho em tempo real. Em *softwares* similares as simulações são realizadas por meio da geração de código, bem como a destinação de *threads* específicas para a execução da simulação, dessa forma permitindo maior velocidade na execução do modelo. Neste trabalho as simulações não são executadas em uma *thread* específica, então uma sugestão é implementar futuramente e refazer os testes de performance.

Quanto a flexibilidade e a capacidade da representação de sistemas de controle, observa-se que a biblioteca contém os blocos essenciais para modelar sistemas simples, em que a computação do fluxo de dados seja suficiente para fornecer a lógica do modelo. Em outros *softwares* é possível adicionar blocos que alteram o fluxo de dados, tais como *switches* ou a presença de entradas nos blocos que permitem a execução ou não, dessa forma facilitando a implementação de estruturas de *software* tais como máquinas de estado. A biblioteca de blocos não permite a adição de blocos definidos pelo usuário, o que é imprescindível em termos de reuso de *software* e extensibilidade. A implementação de um sistema de bibliotecas com blocos definidos pelo usuário, bem como a inclusão de blocos que possam alterar o fluxo de dados podem ser abordados em trabalhos futuros.

O sistema de visualização implementado permite por meio de fácil configuração a visualização dos resultados, no entanto não possui mecanismo para exportar os dados das simulações, o que permitiria a análise posterior em outros softwares. Uma das formas de intercâmbio de dados em softwares de análise de dados é o CSV, esta ferramenta possui o bloco CSVReader, o qual lê dados no formato CSV, e o bloco vetor no qual é possível enviar pela porta serial nesse formato, assim, para trabalhos futuros a funcionalidade de exportar os dados de simulação pode ser rapidamente implementada ao reutizar o código dos blocos CSVReader e Vetor.

REFERÊNCIAS

BUCHER, R. Practical experiences with Python and Linux RT at the SUPSI Laboratory. **Advances in Control Education - 12th ACE 2019™**, 07 Julho 2019.

BUCHER, R.; BALEMI, S. **Rapid Controller Prototyping with Matlab/Simulink and Linux**. Control Engineering Practice. [S.l.]: [s.n.]. 2006. p. 12.

C. BUTAZZO, G. **Hard Real-Time computing systems**. New York: Springer, 2011.

CABOT, J. Executable models vs code-generation vs model interpretation. **https:** //modeling-languages.com/, 2010. Disponivel em: <https://modeling-languages.com/executable-models-vs-code-generation-vs-model-interpretation-2/>. Acesso em: 4 outubro 2020.

CELLIER , F. E.; KOFFMAN, E. **Continuous System Simulation**. [S.l.]: Springer, 2006.

CHIN, C. S. **Computer-Aided Control Systems Design**. New York: Taylor & Francis Group, 2013.

DSPACE. Bypassing – Externally or Directly on the ECU. **https:** //www.dspace.com/, 2020. Disponivel em: <https://www.dspace.com/en/inc/home/products/systems/functp/bypassing.cfm>. Acesso em: 10 out. 2020.

ERICH GAMMA, R. H. R. J. J. V. **Padrões de projeto:** soluções reutilizáveis de. 1. ed. Porto Alegre: Bookman, 2005.

ESI GROUP. Scilab. **www.scilab.org**, 4 outubro 2021. Disponivel em: <https://www.scilab.org>.

FOUNDATION, T. L. Intro to Real-Time Linux for Embedded Developers. **linuxfoundation.org**, 2013. Disponivel em: <https://www.linuxfoundation.org/blog/2013/03/intro-to-real-time-linux-for-embedded-developers/>. Acesso em: 10 out. 2020.

HUSS, S. A. **Model Engineering in Mixed-Signal Circuit Design:** A Guide to Generating Accurate Behavioral Models in VHDL-AMS. New York: KLUWER ACADEMIC PUBLISHERS, 2003.

INRIA. Frequently asked questions about Scicos. **http:** //www.scicos.org/, 2015. Disponivel em: <http://www.scicos.org/FAQ.html>. Acesso em: outubro 2020.

INRIA. Presentation Of Scicos. **www.scicos.org**, 2 maio 2021. Disponivel em: <http://www.scicos.org/PRESENTATION/scicos_presentation.html>.

INTERVALZERO. Windows RTX64 RTOS by IntervalZero. **IntervalZero RTOS Platform**, 5 setembro 2020. Disponivel em: <https://www.intervalzero.com/rtos/windows-rtx64-rtos-by-intervalzero/>. Acesso em: 05 setembro 2020.

INTRODUCING JSON. **www.json.org**, 2 maio 2021. Disponivel em: <https://www.json.org/json-en.html>.

KARSTEN AHNERT, M. M. ODEINT. **odeint:** solving ODEs in C++, 2012. Disponivel em: <http://headmyshoulder.github.io/odeint-v2/>. Acesso em: 5 Outubro 2020.

KELEMENOVÁ, T. et al. Model Based Design and HIL Simulations. **American Journal of Mechanical Engineering 1**, 18 novembro 2013. 276-281.

LATHI, B. P. **Sinais e Sistemas Lineares**. 2. ed. Porto Alegre: Bookman, 2008.

LEDIN, J. A. Hardware-in-the-Loop Simulation. **Embedded Systems Programming**, fevereiro 2009.

LLNL, L. L. N. L. SUNDIALS: SUite of Nonlinear and DIfferential/ALgebraic Equation Solvers. **https:** //computing.llnl.gov/, 2020. Disponivel em: <https://computing.llnl.gov/projects/sundials>. Acesso em: 4 outubro 2020.

MESSERSCHMITT, D. G.; LEE, E. A. Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing. **IEEE Transactions on Computers**, Janeiro 1987.

NISE, N. S. **Engenharia de sistemas de controle**. 6. ed. Rio de Janeiro: LTC, 2013.

OGATA, K. **Engenharia de Controle Moderno**. [S.l.]: Pearson, 2005.
PEGDEN, C. D. . S. R. E. . R. P. **Introduction to Simulation Using SIMAN**. New York: McGraw-Hill, 1990.

PETER SCHWARZ, J. U. RAPID PROTOTYPING FOR MODEL AND CONTROLLER. **CONTROL SYSTEMS, ROBOTICS AND AUTOMATION - Vol. IV**, p. 287, 2009.

POPOVICI, K.; MOSTERMAN, P. J. **Real Time Simulation Technologies - Principles, Methods and Applications**. Boca Raton: CRC Press, 2013.

RTAI. RTAI - the RealTime Application Interface for Linux. **RTAI - Real Time Application Interface Official Website**, 2018. Disponivel em: <https://www.rtai.org/>. Acesso em: 7 out. 2020.

S. BHATTACHARYYA, S.; K. MURTHY, P.; A. LEE,. APGAN and RPMC: Complementary Heuristics for Translating DSP Block Diagrams into Efficient Software Implementations. **Design Automation for Embedded Systems Journal**, 17 Maio 1996.
SCHWARZ, P. Modeling Languages for Continuous and Discrete Systems. In: (EOLSS), E. O. L. S. S. **Encyclopedia of Life Support Systems (EOLSS)**. [S.l.]: UNESCO - Encyclopedia of Life Support Systems (EOLSS), 2009.

SIMON ABOURIDA, C. D. J. B. **Real-Time and Hardware-In-The-Loop Simulation of Electric Drives and Power Electronics:** Process, problems and solutions. Proceedings - The 2005 International Power Electronics Conference. Niigata - Japão: Leibniz Information Centre for Science and Technology University Library. 4-8 Abril 2005. p. 6.

THE MATHWORKS, I. **Simscape**. Disponivel em: <https://www.mathworks.com/products/simscape.html>. Acesso em: 10 outubro 2020.

THE MATHWORKS, INC. Simulink. **www.mathworks.com**, 4 outubro 2021. Disponivel em: <https://www.mathworks.com/products/simulink.html>.

THE QT COMPANY. Customer Cases. **https:** //www.qt.io, 26 abril 2022. Disponivel em: <https://resources.qt.io/customer-cases>.

TILLER, M. M. Home. **Modelica By Example**, 2020. Disponivel em: <https://mbe.modelica.university/>. Acesso em: 10 outubro 2020.

XIAO, H. A metamodel for the notation of graphical modeling languages. **Computer Software and Applications Conference, 2007. COMPSAC 2007 - Vol. 1. 31st Annual International, Volume 1**, 24-27 Julho 2007. 219-224.