Diógenes Azevedo Evangelista

Plataforma de controle usando sistema operacional em tempo real

Belo Horizonte - MG,Brasil

17/12/2009

Diógenes Azevedo Evangelista

Plataforma de controle usando sistema operacional em tempo real

Monografia apresentada para obtenção do Grau de Bacharel em Engenharia de Controle e Automação pela Universidade Federal de Minas Gerais

Orientador:
Maria Auxiliadora Muanis Persechini

ESCOLA DE ENGENHARIA
UNIVERSIDADE FEDERAL DE MINAS GERAIS

Belo Horizonte - MG,Brasil

17/12/2009

Resumo

Neste trabalho é desenvolvida, para os Laboratórios de Controle de Processos do Departamento de Engenharia Eletrônica da UFMG, uma plataforma de software e hardware capaz de executar controladores do tipo PID industrial. A característica genérica da plataforma desenvolvida permite que ela seja utilizada em diferentes processos e plantas pilotos existentes nesses laboratórios.

A plataforma é baseada em um computador de uso genérico, uma placa de aquisição de dados e softwares de distribuição livre. Entre os softwares de distribuição livre foram selecionados para este projeto o sistema operacional Linux, a extensão RTAI e um driver do projeto Comedi que realiza a comunicação com a placa de aquisição de dados. A partir da utilização desses recursos de software e hardware é desenvolvido, em linguagem C, um aplicativo de software para a execução do controlador. Grande parte dos softwares utilizados também são de código aberto, o que permite que os detalhes de implementação sejam melhor estudados e entendidos.

O aplicativo de software desenvolvido utiliza programação multithread e memória compartilhada. O aplicativo possui cinco threads: uma para a interface com o usuário; uma para copiar os parâmetros da interface para a memória compartilhada; uma para gravar os dados em arquivo; uma para executar o algoritmo PID discretizado, além de um filtro passa-baixas e um filtro anti-spike; e uma para criar as estruturas de tempo real e criar as últimas três threads citadas. Esse tipo de programação faz com que o programa fique melhor estruturado, pois permite que cada thread cuide de uma funcionalidade específica e possa ser executada de forma independente, mas colaborativa.

Para testar a plataforma, a planta piloto SCVT (Sistema de Controle de Vazão e Temperatura de Ar) é utilizada. Com os recursos do programa desenvolvido, são feitas coletas de dados para caracterização da planta piloto. A função de transferência no domínio de Laplace da malha de temperatura da planta piloto é obtida por meio de técnicas de modelagem. Com base nessa função de transferência e nos requisitos de controle do processo, um controlador PID é projetado e os parâmetros dos filtros utilizados são escolhidos. Em seguida, para controlar a planta piloto, os parâmetros calculados são inseridos no programa desenvolvido. Por último, são apresentados os resultados do controle de temperatura do ar da planta piloto realizado com a plataforma desenvolvida.

Abstract

In this work, a software and hardware platform capable of executing industrial PID controller is developed to the Processes Control Laboratories of the Electronic Engineering Department of UFMG. The developed platform has a generic characteristic, which permits its use in different processes and pilot plants that exist in the laboratories.

The platform is based on a generic computer, a data acquisition board and free softwares. Were selected, between the free softwares, the operating system Linux, the RTAI patch and a driver of the Comedi project, which provides the communication with the data acquisition board. Using these software and hardware resources, is developed, in C language, a software application to the execution of the controller. Most softwares used are also open-source, permitting a better study and understanding of the implementation details.

The developed software application uses multithread programming and shared memory. The applicative has five threads: one to manage the interface with the user; one to place a copy of the interface parameters in the shared memory; one to persist the data; one to execute the discrete PID algorithm, besides of a low-pass filter and an anti-spike filter; and one to create the real time structures and the last three threads cited. This kind of programming contributes to a better structured program, for it permits each thread to take care of one specific functionality and to be executed independently, but collaboratively.

The SCTV (System of Air Flow and Temperature Control) pilot plant is used to test the platform. Data is collected to the characterization of the pilot plant with resources of the developed program. The transfer function in the domain of Laplace of the pilot plant's temperature loop is obtained by means of modeling techniques. Based on this transfer function and on the requirements of the process control, a controller PID is projected and the parameters of the used filters are chosen. After that, the calculated parameters are inserted in the developed program to control the pilot plant. At last, the results of the control of the pilot plant‘s air temperature with the developed platform are presented.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

K_c : Ganho proporcional;

K_d : Ganho derivativo;

K_i : Ganho integral;

T_i : Tempo integral;

T_d : Tempo derivativo;

T : Intervalo de amostragem;

PV : Variável controlada;

MV : Variável manipulada;

SP : Referência (Setpoint);

e : Erro;

u : Ação de controle;

A_p : Ação proporcional;

R_i : Realimentação integral;

R_d : Realimentação derivativa;

α : Parâmetro alfa do filtro derivativo;

β : Parâmetro beta do filtro passa-baixas;

y : Saída do sensor de temperatura;

y' : Saída do filtro anti-spike;

V : Volts;

s : Domínio da frequência contínuo (Domínio de Laplace);

z : Domínio da frequência discreto;

τ : Constante de tempo da malha de temperatura;

τ_c : Constante de tempo desejada;

K : Ganho estático da malha de temperatura;

1 Introdução

Entre os controladores retroalimentados, o PID é um dos mais conhecidos, estudados e utilizados. O algoritmo desse controlador precisa de 3 parâmetros: Proporcional, Derivativo e Integral. Existem infinitas combinações que resultam em controladores que impõem à planta comportamentos distintos. Por isso, esse tipo de controlador pode ser utilizado nas mais diversas plantas desde que se escolham valores adequados para estes parâmetros.

Neste trabalho é apresentada a implementação de uma plataforma de software e hardware capaz de executar controladores do tipo PID industrial. Essa plataforma é baseada em um computador e softwares de distribuição livre. Apesar de ser implementado apenas o controlador PID industrial, a plataforma é desenvolvida de forma que outros controladores possam ser implementados facilmente.

Um PC de uso genérico é usado com o sistema operacional Linux. O computador possui CPU INTEL Dual Core E2.140 com dois núcleos de 1,6 GHz e 1Gb de memória RAM. A distribuição usada é a Ubuntu, ela é estável e possui comunidade ativa de suporte. Novas versões do Ubuntu são lançadas freqüentemente e inclui atualizações de segurança pela Canonical. O Ubuntu possui boa infra-estrutura de tradução e acessibilidade e inclui apenas softwares livres [www.ubuntu br.org, 2009].

Os componentes de software são desenvolvidos em linguagem C. Essa linguagem possui diversas IDEs (*Integrated Development Environment*) de código aberto para Linux e bibliotecas de uso geral. Além disso, possibilita o uso fácil de todos os recursos necessários para programação em tempo real e concorrente como semáforos, temporizadores, threads entre outros. Existem outras linguagens que possuem IDEs de código aberto e os recursos citados, mas preferiu-se usar a linguagem C. [Seixas Filho and Szuster, 1993] mostra como usar os recursos para programação concorrente e apresenta exemplos de código em C.

O Linux padrão pode ser considerado um sistema operacional em tempo real para intervalos de tempo acima de um determinado limite. Esse limite pode ser um fator que impeça o uso do Linux padrão em determinadas aplicações. Para que o valor desse limite seja reduzido, é

instalada uma extensão para o kernel do Linux chamada RTAI. Assim, o Linux com o RTAI pode ser considerado um sistema de tempo real para intervalos de tempo acima de um novo limite, que é menor do que o limite do Linux padrão. Isso faz com que a plataforma desenvolvida possa ser usada em um número maior de aplicações. Além disso, o RTAI fornece diversos serviços que facilitam a programação voltada para aplicações de tempo real [www.rtai.org/, 2009].

Para aquisição de dados do processo é utilizada a placa de aquisição de dados PCI-6221 da National Instruments. Essa placa possui, entre outros, 16 entradas analógicas de 16 bits e 2 saídas analógicas de 16 bits [NI, 2008]. Na prática, a placa é responsável pela interface entre a plataforma e o processo.

Para acesso aos dados da placa de aquisição é utilizado um driver de comunicação que faz parte do projeto Comedi [www.comedi.org/, 2009]. Esse projeto desenvolve drivers, ferramentas e bibliotecas de código aberto e fornece um API que inclui, entre outras, funções de leitura e escrita na placa.

A plataforma desenvolvida é validada ao ser utilizada para modelar e controlar uma planta piloto. A planta piloto utilizada é o Sistema de Controle de Vazão e Temperatura de Ar (SCVT) [Magalhães, 2008].

1.1 Objetivos

Este trabalho tem o objetivo de desenvolver, usando apenas softwares de distribuição livre, uma plataforma de software e hardware para implementação de algoritmos de controle. Inicialmente, como estudo de caso, um controlador PID industrial é implementado. Esse controlador deve ser capaz de controlar, após ajuste dos parâmetros, processos do tipo SISO (*single input single output*) se os sinais de entrada e saída do processo respeitarem os limites de tensão da placa de aquisição utilizada. Como teste da plataforma, objetiva-se modelar uma planta de temperatura e vazão de ar e controlá-la usando a plataforma desenvolvida. Para isso é necessário:

- Instalar um ambiente computacional com sistema operacional em tempo real;
- Implementar o algoritmo de um controlador PID nesse ambiente;
- Instalar a placa de aquisição de dados e integrá-la a plataforma;
- Modelar e controlar a planta piloto de vazão e temperatura de ar.

1.2 Justificativa

Existem diversos tipos de controladores retroalimentados, mas escolheu-se o PID porque ele é um dos mais conhecidos, estudados e utilizados na área acadêmica e é largamente utilizado no ambiente industrial.

Uma plataforma de controle baseada em um computador e softwares de distribuição livre permite que esse tipo de controlador seja amplamente utilizado pelos alunos e professores em práticas de laboratório e outras atividades.

O fato dessa plataforma, e a maioria de seus componentes, ser de código aberto permite que os detalhes de implementação sejam melhor estudados e entendidos de forma que esses conhecimentos não fiquem restritos a ambientes corporativos ou proprietários.

Após o desenvolvimento deste trabalho, a plataforma desenvolvida será disponibilizada para uso nas disciplinas de Laboratório de Controle e Automação I e II. Tendo em vista a característica genérica da plataforma a ser desenvolvida ela poderá ser utilizada em diferentes processos e plantas pilotos existentes nesses laboratórios.

1.3 Revisão bibliográfica

Linux é o termo geralmente usado para designar qualquer sistema operacional que utilize o núcleo Linux. A primeira versão do kernel Linux foi desenvolvida pelo finlandês Linus Torvalds, inspirado no sistema Minix.

Segundo [Terry Bollinger, 2003] o Linux está disponível sob licença GPL (*General Public License*). Essa licença garante ao usuário o direito de :

- Decidir onde e quando usar o software;
- Estudar como ele funciona (o acesso ao código é uma precondição);
- Modificá-lo;
- Publicar possíveis modificações;
- Vender a versão original ou a modificada.

Um detalhe importante é que qualquer produto desenvolvido que se baseie no código fonte de um software sob licença GPL deve ter todas as cópias lançadas dentro da mesma licença e

acompanhadas do código fonte. Porém, isso não se aplica à softwares que simplesmente usam outros softwares GPL ou que apenas utilizam o mesmo sistema de um software GPL.

Existem diversas distribuições de Linux de propósito geral, entre elas: Ubuntu, Kubuntu, Kurumin, OpenSuSE, Red Hat Linux, Fedora, Mandriva. Segundo [www.distrowatch.com/, 2009], página especializada em catalogar o desempenho e uso das muitas distribuições Linux, Ubuntu é a mais utilizada atualmente.

Segundo [www.ubuntu br.org, 2009], uma nova versão do Ubuntu é lançada a cada seis meses e cada versão nova possui suporte completo, incluindo atualizações de segurança pela Canonical por pelo menos 18 meses. O Ubuntu sempre será gratuito, e não cobrará adicionais por uma ”versão enterprise”ou atualizações de segurança. O CD de instalação possui apenas softwares livres.

Existem diversos sistemas operacionais disponíveis no mercado. Em [C.Schmidt et al., 2002] é analisado o desempenho dos sistemas operacionais: Windows NT, Windows 2K, Linux padrão, Linux/RT, QNX e VxWorks. Os melhores resultados são obtidos com o QNX e com o VxWorks. Maiores detalhes do desempenho do QNX podem ser obtidos em [M.Sacha, 1995]. Segundo [C.Schmidt et al., 2002], as grandes desvantagens do QNX e do VxWorks são: não serem de distribuição livre e serem complicados de configurar/programar. Neste trabalho, foi escolhido o sistema operacional Linux com a extensão RTAI, pois ambos são de distribuição livre, possuem um grande número de drivers compatíveis e possuem uma comunidade grande e ativa.

Segundo [Regnier et al.,], o kernel padrão do Linux falha na oferta de garantias temporais típicas dos sistemas de tempo real. Existem muitas extensões para adicionar capacidades de tempo real nesse sistema operacional. Entre elas: Preempt-RT, RTAI, RTLinux e Xenomai. Em [Regnier et al.,], duas delas (Preempt-RT e Xenomai) são analisadas. [Vitale et al., 2004] cita algumas vantagens da implementação de um sistema de controle em Linux com o RTAI. As principais vantagens são:

- O sistema com RTAI possui um tempo de resposta baixo, na ordem de μs, o que permite a implementação de controladores mais avançados que exijam menor tempo de resposta;
- O RTAI é um produto gratuito e, por essa razão, possibilita o uso de controladores redundantes e o teste em paralelo de vários algoritmos de controle;
- No RTAI não é preciso desenvolver programas para se conseguir a integração dos diferentes softwares.

[A.Barbalacel et al., 2008] mostra comparações de desempenho, em várias situações, entre

o VxWorks, o Linux com o Xenomai, o Linux com RTAI e o Linux padrão. Apenas o VxWorks não é de distribuição livre. Em todas as comparações o RTAI foi melhor que o Xenomai e o Linux padrão. Um dos testes mostra que o RTAI apresentava delay de 71,8 μs e um jitter de 0,15 μs e que apenas o VxWorks apresentava um resultado melhor com um delay de 69,20 μs e um jitter de 0,5 μs.

Além disso, [Meghwani and Kulkarni, 2008] observaram que com o RTAI é possível executar aplicações que não necessitam de requisitos de tempo real (por exemplo, TCP/IP e interfaces gráficas) sem se afetar o desempenho das aplicações em tempo real.

[André de Almeida, 2006] usa Linux Debian em conjunto com RTAI para desenvolver, implementar e testar um sistema de controle para um pêndulo invertido. Primeiramente, é desenvolvido um modelo matemático para o pêndulo através de equações diferencias que descrevam a física do processo. Depois, é feita a instalação e configuração do sistema operacional, através da inclusão de bibliotecas de tempo real no Linux padrão, obtendo assim um sistema operacional que possui determinismo no tempo, uma das características indispensáveis para o controle. Por último, são elaboradas estratégias de controle para serem implementadas no Linux de tempo real.

[Flávio Mota, 2006] também usa Linux em conjunto com RTAI para desenvolver gerenciadores de dispositivo GPS para sistema operacional Linux com e sem preempção de tempo real. Uma interface de alto nível é fornecida aos desenvolvedores de programas de aplicação, essa interface é composta por rotinas computacionais que interagem com um receptor GPS de forma transparente. O trabalho tem como objetivo eximir o usuário da necessidade de conhecer profundamente os detalhes de comunicação e gerenciamento do receptor GPS, tanto para programas de aplicação desenvolvidos para Linux padrão, quanto para programas de aplicação desenvolvidos para serem executados em ambiente de tempo real.

Segundo [Bennett, 1988], um sistema ligado por dispositivos físicos a um processo externo opera em tempo real se as ações desse sistema estão relacionadas com os eventos do processo externo. Nesse Projeto Final de Curso, a plataforma desenvolvida está ligada à planta piloto por meio de uma placa de aquisição de dados. A plataforma deve medir as variáveis periodicamente e reagir de forma a manter o sistema operando em torno do ponto de operação desejado.

Para que a placa funcione corretamente no ambiente Linux é necessário um driver desenvolvido para esse sistema operacional. O site [www.comedi.org/, 2009] informa que o projeto Comedi fornece diversos drivers de código aberto para placas de aquisição de dados. Em [I.So et al., 2007] um sistema de aquisição de dados que usa Linux, RTAI e driver Comedi é descrito. O sistema é capaz de adquirir dados em períodos de 10 ms enquanto executa as demais

aplicações que não são de tempo real.

Um programa, independentemente da aplicação, pode possuir uma interface gráfica. A taxa de atualização da interface, geralmente, é baixa. Por outro lado, em aplicações em tempo real, a taxa de amostragem é, normalmente, mais rápida que a necessidade de atualização da interface gráfica. Por isso, a separação entre a interface (lenta) e o código (rápido) é comum em aplicações em tempo real. Essas partes podem ser desenvolvidas em linguagens diferentes, por exemplo, em [Martínez-Ortiz, 2005] as rotinas em tempo real foram programadas em C e a interface em C++.

Surge, então, uma necessidade de comunicação entre as partes. Em [Martínez-Ortiz, 2005] a comunicação entre os módulos em tempo real no kernel e a interface no espaço de usuário é feita usando canais FIFO providos pelo RTAI. Em [Vitale et al., 2004], a comunicação entre processos é baseada no padrão POSIX e usa threads e queues. Em [Meghwani and Kulkarni, 2008] a comunicação entre o controlador e a interface gráfica é feita usando-se memória compartilhada.

Os controladores podem ser executados em tempo real no espaço kernel. Para isso, as rotinas em tempo real podem ser carregadas dinamicamente na forma de módulos kernel com o comando *insmod* e removidas com o comando *rmmod*. Segundo [Meghwani and Kulkarni, 2008], o risco de desenvolver uma aplicação em espaço kernel é que o acesso à memória é desprotegido. Para evitar quebra do sistema, o programa deve ser primeiramente testado em espaço usuário e, depois, mudado para espaço kernel. Porém, nada impede que o aplicativo seja usado em modo usuário desde que a consequente perda de desempenho não afete a aplicação.

1.4 Organização do Texto

No capítulo 1, é apresentado o contexto em que o projeto está inserido, os objetivos do projeto e quais são as motivações para realizá-lo. Também é feita uma revisão bibliográfica que embasa o restante do trabalho.

No capítulo 2, todos os recursos de software e hardware utilizados para o desenvolvimento da plataforma de controle são apresentados. Alguns testes de desempenho são realizados para validar o sistema operacional em tempo real.

No capítulo 3, a implementação da plataforma é descrita em detalhes. A função dos recursos de programação em tempo real, como threads e semáforos, com relação à plataforma é explicada. Também é mostrado o algoritmo PID utilizado e sua versão discretizada.

No capítulo 4, a planta piloto é descrita mais detalhadamente e um controlador PID é projetado e implementado. Em seguida, o resultado do uso da plataforma para o controle da planta é apresentado.

Por último, no capítulo 5, são apresentadas as conclusões e várias sugestões para trabalhos futuros.

2 *Descrição da plataforma de desenvolvimento*

Neste capítulo, os principais componentes de software e hardware usados na plataforma serão descritos. Os principais componentes de hardware são: o computador e a placa de aquisição. Os componentes de software mais importantes são: o sistema operacional, a extensão para tempo real e o driver da placa de aquisição de dados. Além disso, serão mostrados os resultados dos testes feitos no Linux e no RTAI.

A plataforma de desenvolvimento deve:

- Se comportar como um sistema de tempo real, dentro de limites de tempo a serem definidos;
- Deve também permitir uma comunicação fácil com os hardwares, entre eles a placa de aquisição de dados;
- Ser composta de softwares de distribuição livre.

A Figura 2.1 mostra a relação entre os principais hardwares e softwares utilizados, mostra também a forma como a planta piloto é conectada à plataforma. É possível perceber que a placa de aquisição de dados faz a interface entre o ambiente computacional e a planta, e possibilita a coleta de dados para a modelagem e o controle.

2.1 Componentes de software

Na Figura 2.2, é possível ver o relacionamento entre o aplicativo desenvolvido e os principais componentes de software utilizados. A plataforma de controle desenvolvida faz uso de funções do sistema operacional (Ubuntu), da extensão RTAI e do driver da placa de aquisição de dados (Comedi).

No apêndice A, é apresentado um tutorial detalhado para instalação do RTAI e do Comedi.

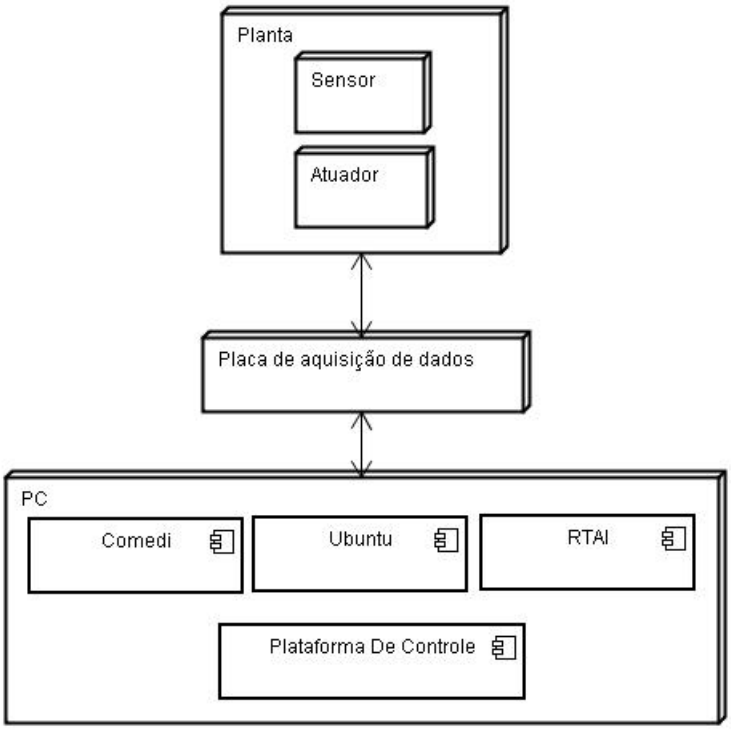


Figura 2.1: Diagrama de Implementação.

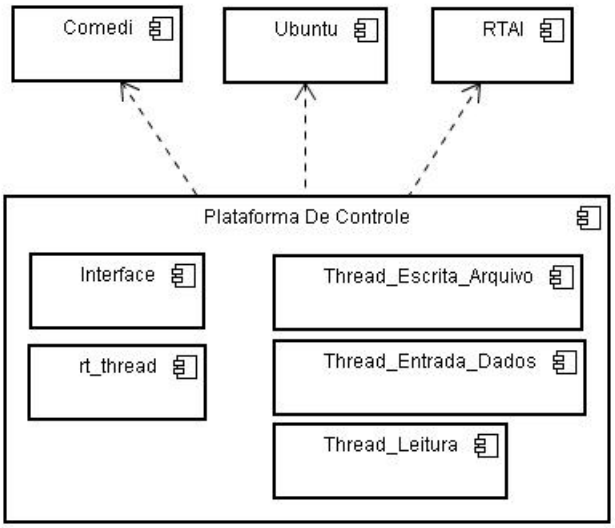


Figura 2.2: Diagrama de Componente.

2.1.1 Sistema operacional utilizado

O Linux é um sistema operacional de distribuição livre e está atualmente disponível para muitos processadores diferentes e para uma grande variedade de dispositivos.

O centro do sistema operacional Linux é o kernel. Porém devido a natureza *open-source* do kernel, que permite a qualquer um modificá-lo, o Linux está disponível em uma grande

variedade de distribuições. Uma distribuição Linux é um conjunto de softwares, ferramentas e utilitários, tendo como base o kernel do Linux.

Kernel do Linux

O kernel é o núcleo do sistema operacional e dá aos softwares a capacidade de acessar o hardware. É a parte do sistema responsável pelo gerenciamento de baixo nível do hardware e software. Ele pode incluir todos os drivers de dispositivos suportados pelo sistema.

Para manter a compatibilidade com o maior número possível de dispositivos, as distribuições incluem quase todos os drivers de dispositivos disponíveis para o Linux. Para evitar que isso torne o kernel muito grande, um kernel básico é criado com os drivers mais importantes e os demais drivers são incluídos como módulos.

Os módulos oferecem a vantagem de poderem ser carregados e descarregados dinamicamente conforme necessário, sem ficarem o tempo todo consumindo memória RAM e recursos do sistema. Alguns programas utilizam um módulo no Kernel para ter acesso direto ao hardware e assim rodar com um melhor desempenho.

É possível editar o kernel para otimizá-lo ao computador em uso. Nesse projeto a opção mais importante com relação ao desempenho é indicar qual processador está sendo utilizado, que é o INTEL Dual Core E2.140. Isto faz com que o kernel seja compilado com otimizações para a arquitetura, o que resulta em um ganho de desempenho. Outras modificações também são feitas para diminuir a latência, mais detalhes no apêndice A.

Para instalar o RTAI é preciso usar um kernel vanilla. Utilizou-se o 2.6.23, pois ele é compatível com a versão do RTAI e a versão do Ubuntu utilizadas. Um kernel vanilla é um kernel estável, padronizado e retirado do site *http://www.kernel.org* e pode ser diferenciado dos demais pela numeração. Um kernel comum é um kernel vanilla modificado e possui como sufixo o número do vanilla usado, por exemplo 2.6.23.xx.

Distribuição utilizada

A distribuição de Linux usada foi a Ubuntu, ela é estável e possui comunidade ativa de suporte. Novas versões do Ubuntu são lançadas freqüentemente, incluem atualizações de segurança pela Canonical e podem ser baixadas gratuitamente em *www.ubuntu-br.org*. O Ubuntu possui boa infra-estrutura de tradução e acessibilidade e, segundo [www.ubuntu br.org, 2009], inclui apenas softwares livres.

É utilizada a versão 8.04 para desktop, que é uma versão LTS (Long-term Support). Em geral, nas versões LTS, todos os aplicativos e componentes já foram testados e considerados estáveis, além da garantia de suporte por três anos para desktop e cinco para servidor.

O Ubuntu utiliza uma configuração clássica do GNOME. Segundo [www.gnome.org, 2009], o GNOME é Software Livre e parte do Projeto GNU, que se dedica a dar a usuários e desenvolvedores controle sobre seus desktops, softwares e dados. Isso consiste, basicamente, em garantir o direito: de usar o software para qualquer fim, estudar o seu código-fonte, modificá-lo e redistribuí-lo, modificado ou não. O GNOME entende que usabilidade é criar programas que sejam de fácil utilização para todos e não simplesmente abarrotá-los com recursos.

2.1.2 Extensão para tempo real

Segundo [Bennett, 1988], sistemas em tempo real são sistemas em que:

1. O início da execução de um ciclo é determinado pela passagem do tempo ou por um evento externo;
2. Os resultados da execução de um ciclo dependem do valor da variável tempo no momento da execução ou dependem do tempo necessário para a execução completa do ciclo.

Um aplicativo que implementa um controlador é um sistema de tempo real, pois:

1. A execução de um ciclo deve ocorrer periodicamente, ou seja, o início da execução é determinado pela passagem do tempo;
2. Os resultados da execução de um ciclo dependem do tempo necessário para a execução completa do ciclo. Um atraso no cálculo da ação de controle muda a forma como a planta reage, pois se o tempo para o cálculo deixar de ser desprezível, pode ocorrer a introdução de atraso fracionário. A introdução de atraso fracionário faz com que a validade do algoritmo precise ser reavalidado, pois as equações se tornam inválidas.

Ainda segundo [Bennett, 1988], os sistemas de tempo real podem ser divididos em duas categorias:

- Os sistemas que precisam que o tempo de execução médio dos ciclos, obtido em um intervalo de tempo fixo, seja menor que um valor máximo especificado.
- Os sistemas em que os resultados de um ciclo precisam ser obtidos, sempre, dentro de um período de tempo máximo especificado.

Essas categorias são apenas uma referência, pois, por exemplo, em um aplicativo que implementa um controlador é preciso que:

- O cálculo da ação de controle ocorra em um tempo menor que o período máximo, que é especificado com base no intervalo de amostragem;
- O tempo médio decorrido entre as leituras da variável controlada seja próximo do tempo de amostragem especificado (T).

Porém, caso, eventualmente, as restrições não sejam cumpridas, o desempenho do controlador não será seriamente afetado. Por exemplo, segundo [Bennett, 1988], no caso específico de um sistema de aquecimento, o desempenho do controlador não será seriamente afetado se uma amostra for perdida ocasionalmente ou se $9,95 \leq T \leq 10,05$ ms para um sistema em que o T definido seja de 10 ms.

Cada aplicação deve ser analisada individualmente para determinar a tolerância da aplicação à variação do intervalo de amostragem, à variação do tempo necessário para o cálculo da ação de controle e os efeitos da perda de amostras. Pode haver casos em que nenhum tipo de atraso ou perda de amostras é tolerado. Mas não é parte do escopo desse trabalho definir como essa tolerância pode ser definida.

Como mencionado anteriormente, o Linux não é nativamente um sistema em tempo real (para intervalos de tempo na ordem de μs) e existem diversas extensões com o propósito de torná-lo um sistema operacional em tempo real. Escolheu-se o RTAI, pois ele possui uma comunidade maior e mais ativa, é citado em um número maior de artigos, é gratuito e apresenta melhor desempenho, segundo a revisão bibliográfica realizada. O RTAI oferece diversos serviços que facilitam a programação voltada para aplicações com restrições desse tipo. Além de permitir a coexistência de tarefas críticas no tempo com tarefas comuns.

Segundo [Dozio and Mantegazza, 2003], o projeto do RTAI começou no Departamento de Engenharia Aeroespacial da Politécnico de Milão. Essa extensão instala uma HAL (Hardware Abstraction Layer) genérica. HAL é uma camada abstrata que intercepta todas as interrupções de software e de hardware e, para cada uma, decide se o Linux ou o RTAI será responsável pelo tratamento da interrupção.

O guia [www.rtai.org/, 2006] informa que o RTAI oferece os mesmos serviços do kernel, mas com as características de um sistema de tempo real, principalmente o determinismo e a preemptibilidade. Ele consiste basicamente em um expedidor de interrupções: intercepta as interrupções e as redirecionam para o Linux, caso necessário. Não é uma modificação do

kernel, mas utiliza a HAL para adquirir informação do Linux e comandar a execução de algumas funções fundamentais. O RTAI trata o Linux como um tarefa de segundo plano que é executada quando não há nada para fazer nas tarefas de tempo real. Maiores detalhes em [Dozio and Mantegazza, 2003].

A versão do RTAI utilizada foi a 3.6.2, obtida no site *https://www.rtai.org/* em março de 2009.

Segundo [Regnier et al.,], nas plataformas de tempo real, eventos de temporizadores ou de hardware são utilizados para disparar tarefas. Uma tarefa periódica fica suspensa a espera de um evento. Quando o evento ocorre, a requisição de interrupção associada aciona o tratador correspondente que, por sua vez, acorda a tarefa. O intervalo de tempo entre os instantes de ocorrência do evento e o início da execução da tarefa associada é chamado de latência.

No apêndice A, é mostrado que a latência, medida pelo RTAI, em modo kernel é menor que 17 μs e em modo usuário a latência é de cerca de 20 μs. De um modo geral, executar uma aplicação em espaço kernel faz com que ela tenha um melhor desempenho com relação à mesma aplicação executada em modo usuário. Porém, executar em modo kernel pode levar à quebra do sistema, pois o acesso à memória é desprotegido. Segundo [Lineo, 1993], existem duas formas de executar o aplicativo em espaço kernel: compilando o aplicativo como um módulo kernel; ou usando a função *rt_make_hard_real_time*. A primeira opção é a que apresenta melhor desempenho, porém a segunda opção é utilizada neste trabalho por ser a mais simples de implementar.

2.1.3 Driver de comunicação

O site [www.comedi.org/, 2009] informa que o projeto Comedi desenvolve drivers, ferramentas e bibliotecas de código aberto para aquisição de dados. O Comedi consiste em uma coleção de drivers para uma variedade de placas de aquisição de dados e uma biblioteca em espaço kernel (Kcomedilib) que provê uma interface com os dispositivos conveniente para tarefas em tempo real.

O projeto Comedi disponibiliza também o Comedilib que é uma biblioteca que provê a mesma interface do Kcomedilib, mas em espaço usuário e de forma mais amigável ao desenvolvedor. O Comedilib inclui documentação, programas demonstrativos e aplicativos para configuração e calibração.

Os drivers são implementados como módulos do kernel e fornecem funcionalidades comuns entre si, porém, para placas de aquisição de dados diferentes. Cada driver de baixo nível

é implementado em um módulo individual. Na plataforma desenvolvida, o driver específico utilizado é o ni_pcimio. As principais características dos drivers comedi são:

- Prover suporte para tempo real para a maioria das placas;
- Possuir uma biblioteca de alto nível: Comedilib;
- Requer que a versão do kernel do Linux seja 2.6 ou superior.

Há várias funções síncronas básicas disponíveis no comedi:

- Aquisição analógica simples: lê um valor analógico do canal indicado;
- Aquisição analógica múltiplas: lê consecutivamente n valores analógicos do canal indicado, o número de amostras é limitado a 100 porque a função é bloqueante;
- Escrita analógica simples: escreve um valor analógico no canal indicado;
- Aquisição digital simples: lê um valor digital do canal indicado;
- Escrita digital simples: escreve um valor digital no canal indicado.

Além das funções de leitura e escrita na placa, o driver também inclui funções de conversão dos valores usados internamente pela placa (de 0 a 65536 DAC) em valores físicos como tensão (de -10 a 10 V).

2.2 Componentes de hardware

Um PC de uso genérico é utilizado como base para a plataforma. O computador utilizado possui CPU INTEL Dual Core E2.140 com dois núcleos de 1,6 GHz e 1Gb de memória RAM.

Utilizou-se a placa PCI-6221 da National Instruments para aquisição de dados. Segundo [Instruments, 2007], ela possui entre outros:

- 16 canais de entradas analógicas de 16 bits com freqüência de até 250 KSamples/s ,
- 2 canais de saídas analógicas de 16 bits com freqüência de até 833 KSsamples/s,
- 24 canais de Entrada/Saída digital com freqüência de até 1 MHz,
- Intervalo de entrada de -10V a 10V.

Durante o controle da planta, as variáveis de interesse estão disponíveis em forma de tensão(V) e a placa adquire os dados e os transforma para a forma digital, ou seja, um número entre 0 e 65535 (DAC). A ação de controle é então calculada com base no valor obtido. Em seguida, a placa transforma o valor digital da ação de controle, em DAC, em uma tensão que é aplicada na entrada do atuador.

Por isso, é bom saber como os valores DAC são convertidos em tensão e vice-versa. A Figura 2.3 mostra os valores de tensão obtidos ao se aplicar valores DAC. Os valores foram aplicados utilizando-se a plataforma desenvolvida, em modo de operação manual, e a tensão na saída analógica foi medida com um multímetro.

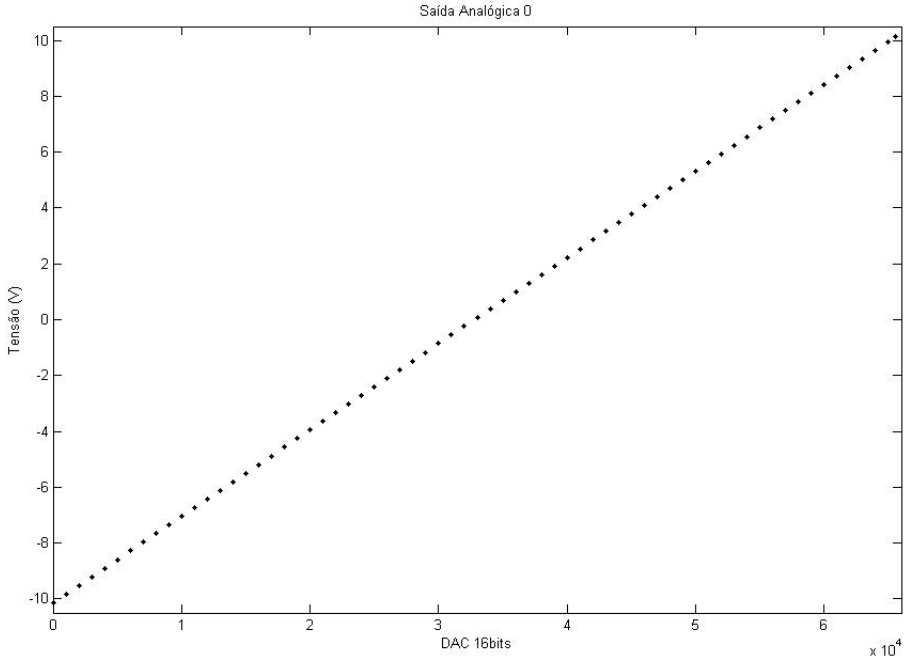


Figura 2.3: Curva de DAC versus tensão do canal 0 de saída analógica.

A Figura 2.4 mostra os valores DAC obtidos ao se aplicar diversas tensões na entrada analógica 0. Os valores de tensão foram gerados a partir da saída analógica, utilizando-se a plataforma desenvolvida em modo de operação manual. A saída analógica foi conectada à entrada analógica e a tensão foi medida com um multímetro.

Na Figura 2.3 e na Figura 2.4, é possível perceber que tanto a curva de conversão de analógico para digital quanto a curva de conversão de digital para analógico são lineares e apresentam a conversão entre tensão e valores DAC conforme esperado.

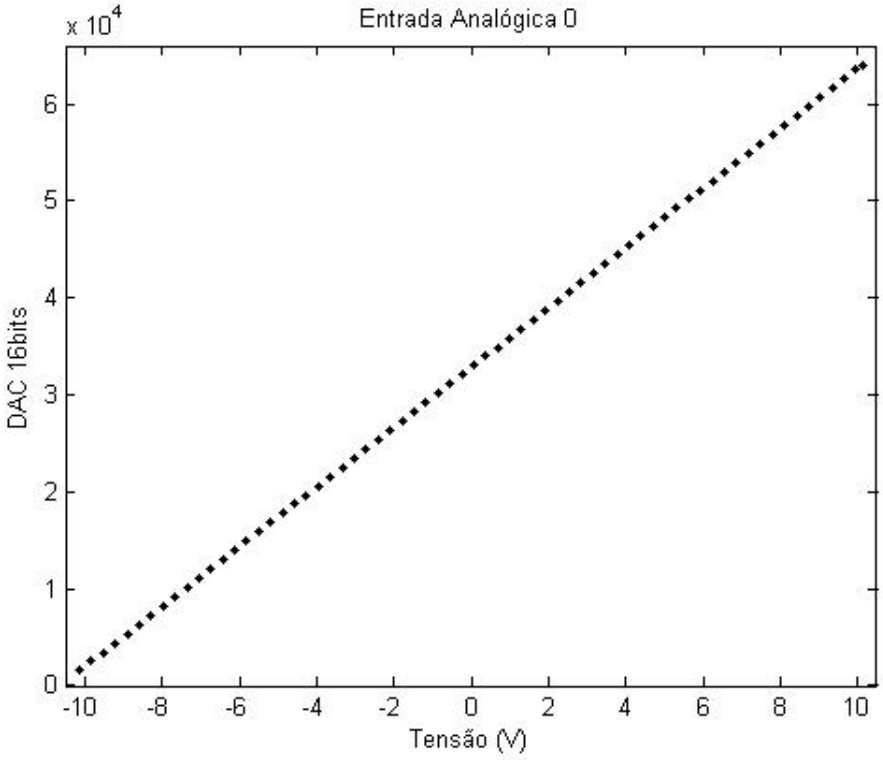


Figura 2.4: Curva de DAC versus tensão do canal 0 de entrada analógica.

2.3 Comparação de desempenho

Como dito anteriormente, a plataforma de controle desenvolvida deve se comportar como um sistema de tempo real e cumprir certas restrições, dentro de limites de tempo a serem definidos. Porém, a definição do limite de tempo e da tolerância ao não cumprimento das restrições depende da aplicação. Essa seção tem como objetivo mostrar o comportamento da plataforma desenvolvida de forma a permitir que, definida uma aplicação, seja possível decidir se a plataforma desenvolvida é adequada a essa aplicação. Não é parte do escopo desse trabalho mostrar como os limites de tempo e a tolerância podem ser definidos.

2.3.1 Linux com baixa latência

A seguir será mostrado, por meio de testes, qual o comportamento do Linux sem a extensão para tempo real. É importante ressaltar que o kernel do Linux utilizado sofreu modificações para a redução da latência, portanto, não é o kernel padrão.

Um programa foi desenvolvido para realizar os testes. Ele é composto de três threads:

- Uma que realiza cálculos matemáticos ininterruptamente;

- Uma que fica em um *loop*. Esse *loop* consiste em abrir um arquivo, escrever nele várias vezes e fechar o arquivo.

- Uma que gera uma onda quadrada de amplitude de 1V. No início de cada ciclo, o momento atual é armazenado em um *buffer*, em seguida, o nível de tensão na saída da placa de aquisição é modificado: se estiver em 1V muda para 2V, se tiver em 2V muda para 1V. Essa thread está associada a um *Timer* que faz com que ela seja executada a cada T segundos. O teste acaba quando essa thread é executada 10000 vezes. Após o fim do teste, o valores armazenados no *buffer* são salvos em arquivo.

Além disso, um osciloscópio digital é conectado à saída da placa de aquisição de dados, para comprovar que o intervalo de tempo T foi obedecido.

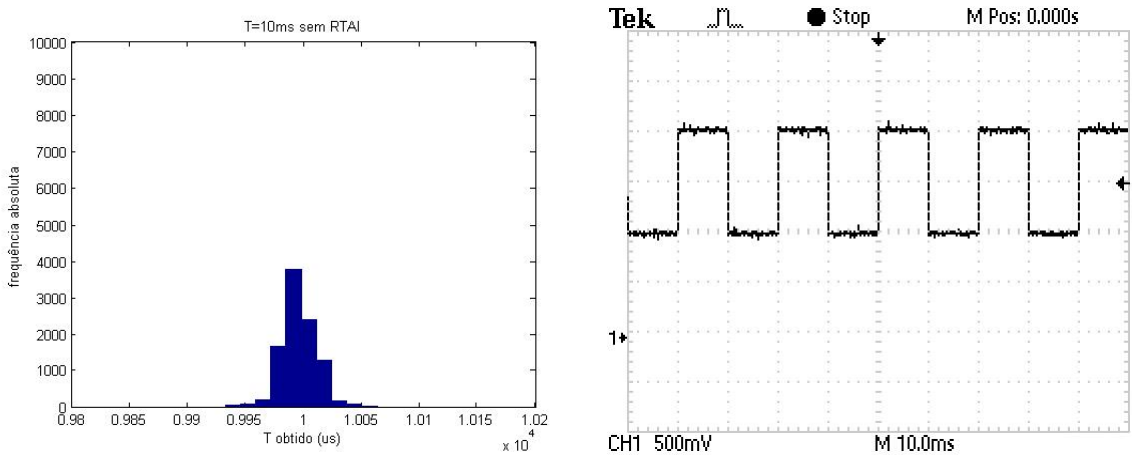
Dessa forma, durante a execução dos testes, o processador fica 100% ocupado e duas fontes de informação são geradas: o formato da onda obtida com o osciloscópio e um arquivo com o momento do início de cada uma das 10000 execuções da terceira thread.

Com o arquivo obtido, é possível gerar um histograma do intervalo de tempo decorrido entre cada execução da terceira thread. Com o histograma, é possível mostrar o intervalo médio, que deve ser igual ao valor definido para T, e a variação desse intervalo. Com a onda obtida, é possível mostrar se o intervalo de tempo medido pelo temporizador é confiável para um certo valor de T.

A Figura 2.5a e a Figura 2.5b mostram os resultados do teste realizado com T=10ms. Na Figura 2.5b, cada divisão vertical representa 500mV e cada divisão horizontal representa 10ms. Pode-se perceber que a onda gerada respeita a amplitude de 1V e o intervalo de tempo especificado. A Figura 2.5a mostra que o intervalo médio é igual a 10 ms e a variação desse intervalo é de 0,08 ms (0,8%).

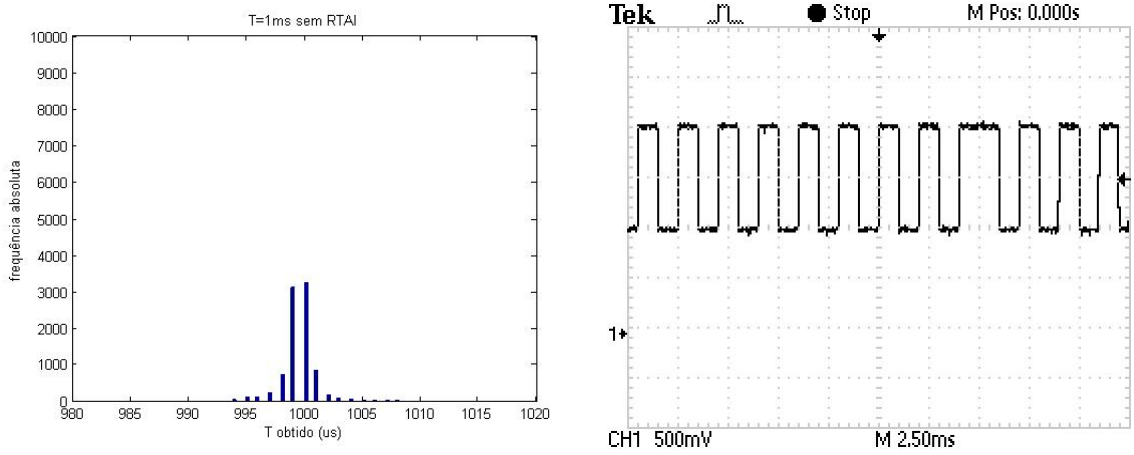
A Figuras 2.6a e a Figura 2.6b mostram os resultados do teste realizado com T=1ms. Na Figura 2.6a, pode-se perceber que o intervalo médio é igual a 1ms e a variação do intervalo é de 10 μs (1%). Na Figura 2.6b, cada divisão vertical representa 500mV e cada divisão horizontal representa 2,5 ms. A onda gerada mostra que, às vezes, o intervalo é de 2ms, ou seja, o dobro do especificado.

A Figura 2.7a e a Figura 2.7b mostram os resultados do teste realizado com T=100 μs. O intervalo médio deveria ser de 100 μ s, mas é igual a 1ms, dez vezes maior que o especificado. Em mais de 6% dos casos o intervalo é de 2ms. Na Figura 2.6b, cada divisão vertical representa 500mV e cada divisão horizontal representa 2,5 ms. A onda gerada confirma que o intervalo especificado não foi respeitado. Portanto, conclui-se que o Linux sem RTAI, mesmo



(a) (b)

Figura 2.5: Resultados do teste realizado sem RTAI para T=10ms.



(a) (b)

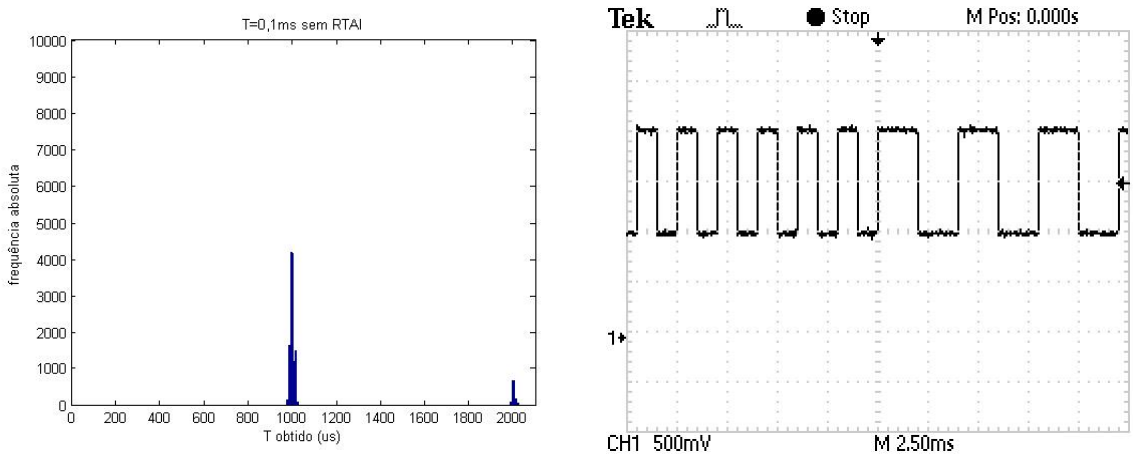
Figura 2.6: Resultados do teste realizado sem RTAI para T=1ms.

com baixa latência, não se comporta como um sistema em tempo real para aplicações que envolvam períodos inferiores a 1ms.

2.3.2 Linux com RTAI

A seguir será mostrado qual o comportamento do Linux com a extensão para tempo real RTAI. Os testes são idênticos aos descritos na subseção 2.3.1, a diferença é que a terceira thread do programa, neste teste, faz uso dos recursos providos pelo RTAI.

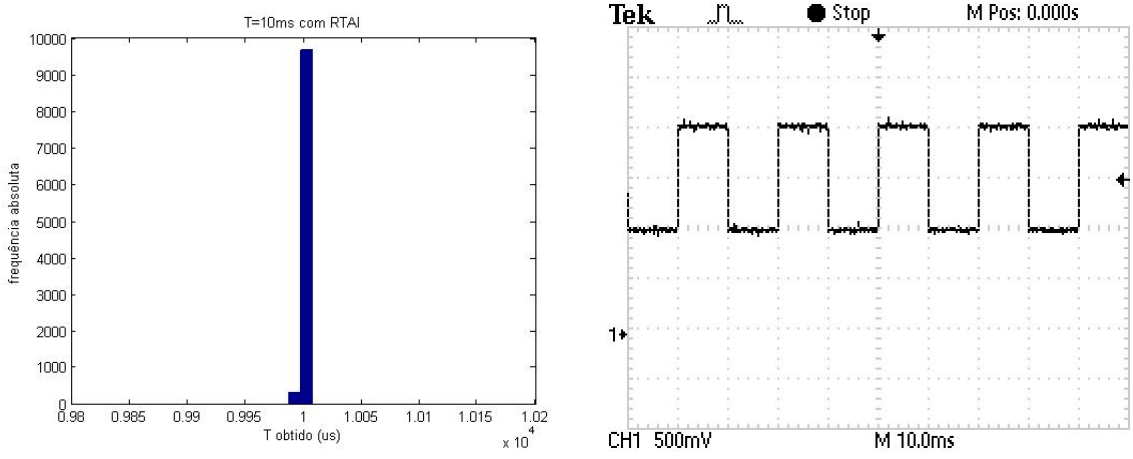
Dessa forma, durante a execução dos testes, o processador também fica 100% ocupado e duas fontes de informação são geradas: o formato da onda obtida com o osciloscópio e um arquivo com o momento do início de cada uma das 10000 execuções da terceira thread.



(a) (b)

Figura 2.7: Resultados do teste realizado sem RTAI para T=100 μs.

A Figura 2.8a e a Figura 2.8b mostram os resultados do teste realizado com T=10ms. Na Figura 2.8a, pode-se perceber que o intervalo médio é igual a 10ms e a variação do intervalo é de 12 μs (0,12%). Essa figura mostra que a variação obtida é bem menor do que a obtida com o Linux sem RTAI, por exemplo, com o RTAI em mais de 90% das vezes o intervalo pertence a mesma faixa de valores enquanto que sem RTAI esse número não chega a 40%. Na Figura 2.8b, cada divisão vertical representa 500mV e cada divisão horizontal representa 10ms. Pode-se perceber que a onda gerada respeita a amplitude de 1V e o intervalo de tempo especificado.

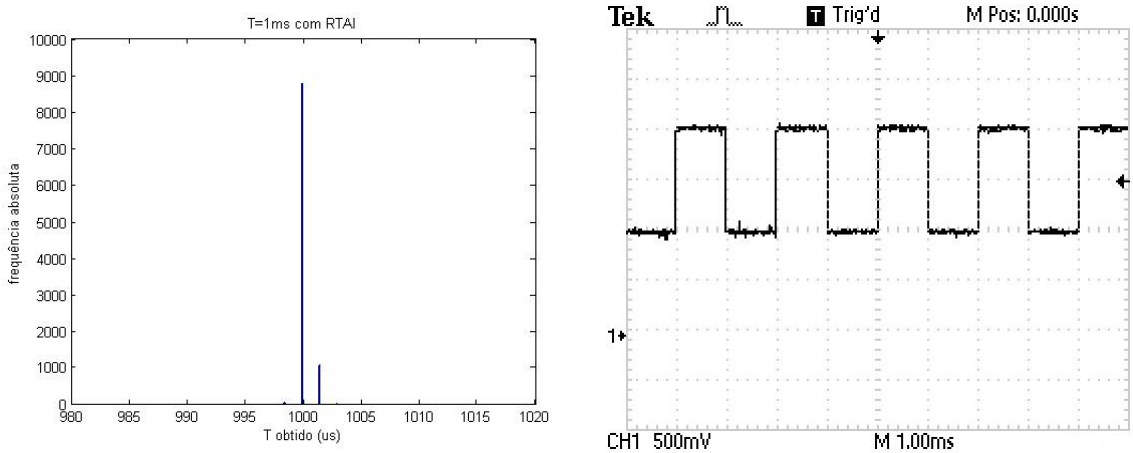


(a) (b)

Figura 2.8: Resultados do teste realizado com RTAI para T=10ms.

A Figuras 2.9a e a Figura 2.9b mostram os resultados do teste realizado com T=1ms. Na Figura 2.9a, pode-se perceber que o intervalo médio é igual a 1ms e a variação do intervalo é de 4 μs (0,4%). Essa figura mostra que a variação obtida é bem menor do que a obtida com o Linux sem RTAI, por exemplo, com o RTAI em quase 90% das vezes o intervalo pertence

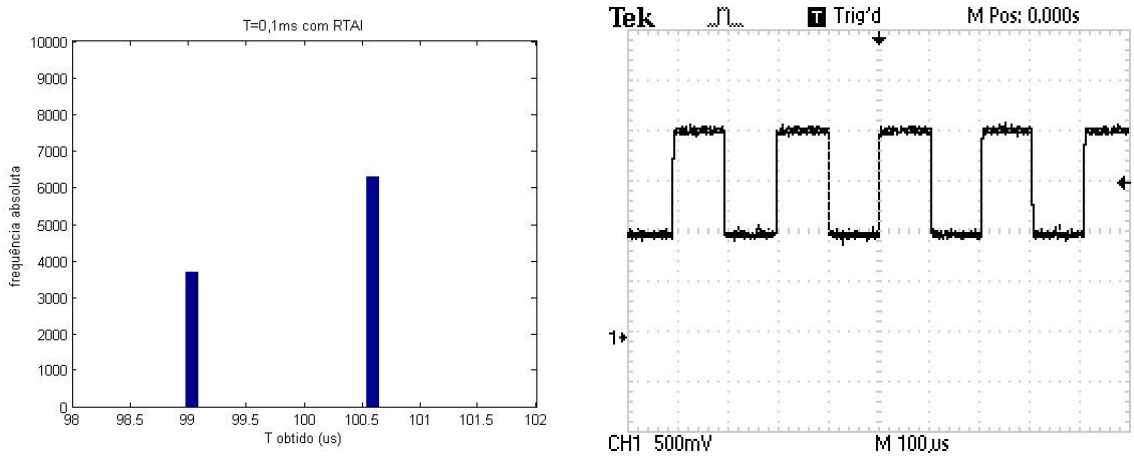
a mesma faixa de valores enquanto que sem RTAI esse número não chega a 35%. Na Figura 2.9b, cada divisão vertical representa 500mV e cada divisão horizontal representa 1ms. Pode-se perceber que a onda gerada respeita a amplitude de 1V e o intervalo de tempo especificado.



(a) (b)

Figura 2.9: Resultados do teste realizado com RTAI para T=1ms.

A Figuras 2.10a e a Figura 2.10b mostram os resultados do teste realizado com T=100 μs. Na Figura 2.10a, pode-se perceber que o intervalo médio é igual a 100 μs, ou seja, igual ao especificado. A figura também mostra que a variação do intervalo é de 1 μs (1%). Na Figura 2.10b, cada divisão vertical representa 500mV e cada divisão horizontal representa 100μs. A onda gerada mostra que, a amplitude de 1V e o intervalo de tempo especificado são respeitados. Portanto, conclui-se que o Linux com RTAI e com 100% da capacidade de processamentos utilizada se comporta como um sistema em tempo real para aplicações que envolvam intervalos de tempo da ordem de 100 μs.



(a) (b)

Figura 2.10: Resultados do teste realizado com RTAI para T=100 μs.

2.3.3 Intervalo mínimo de amostragem

Existem dois intervalos de tempo que são importantes para o controle: o intervalo de amostragem e a duração do ciclo do algoritmo de controle. A Figura 2.11 mostra que o intervalo de amostragem é o tempo entre a leitura de um ciclo e a leitura do ciclo imediatamente anterior. Mostra também que a duração do ciclo do algoritmo de controle é o intervalo de tempo entre o momento da leitura de PV e o momento da escrita de MV, e inclui, portanto, o tempo para o cálculo da ação de controle.

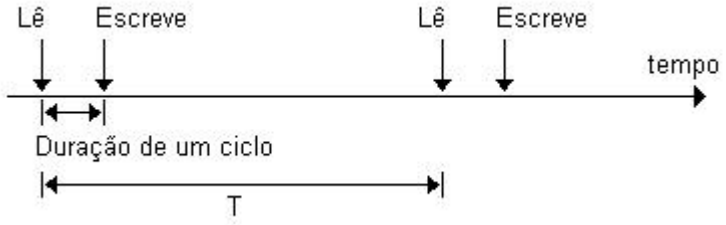


Figura 2.11: Representação de dois intervalos de tempo importantes para o controle.

Na prática, esses dois intervalos não possuem valores exatamente determinísticos, pois apresentam uma certa variação em torno de um valor médio. Para determinar o comportamento desses intervalos, o aplicativo desenvolvido, descrito no capítulo 3, é utilizando com diferentes valores de intervalo de amostragem (T). São obtidos, em forma de arquivo, o momento de início e de fim da execução de cada ciclo. Para cada valor de T, dois histogramas são feitos com os dados de 10000 ciclos: um para o intervalo de amostragem e um para o tempo de duração do ciclo de controle. É importante ressaltar que esse teste foi feito sem carga e com o algoritmo PID industrial, ou seja, a carga principal é o próprio aplicativo que implementa o algoritmo PID.

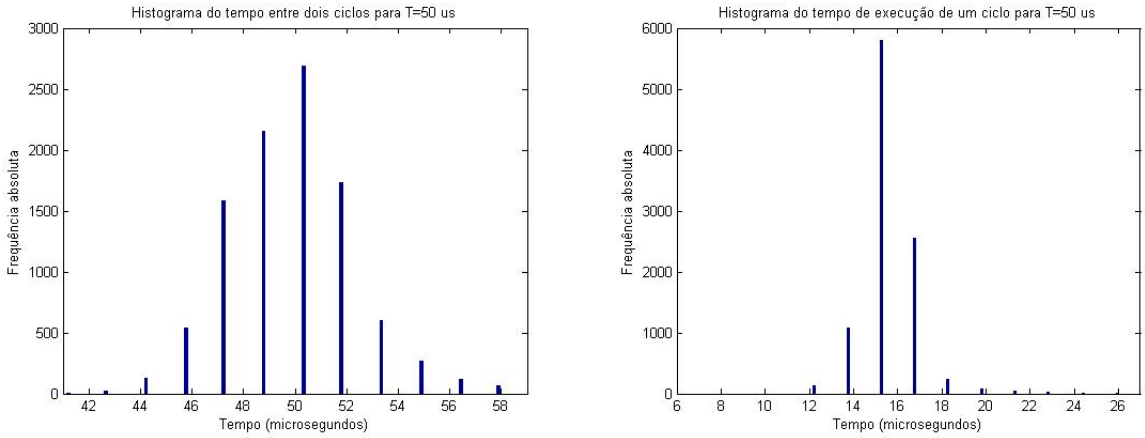
A Figura 2.12a e a Figura 2.12b mostram, respectivamente, a distribuição do intervalo de amostragem e a distribuição do tempo de duração do ciclo de controle, para T=50μs.

A Figura 2.13a e a Figura 2.13b mostram, respectivamente, a distribuição do intervalo de amostragem e a distribuição do tempo de duração do ciclo de controle, para T=60μs.

A Figura 2.14a e a Figura 2.14b mostram, respectivamente, a distribuição do intervalo de amostragem e a distribuição do tempo de duração do ciclo de controle, para T=100μs.

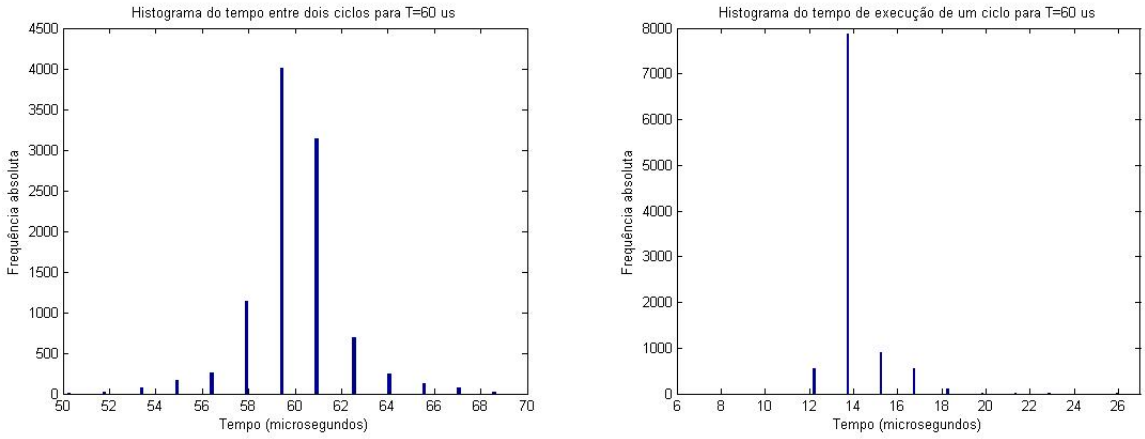
Para cada valor de T, é calculada a porcentagem dos ciclos que apresenta:

1. Tempo de duração do ciclo de controle menor que 20 μs;
2. Intervalo de amostragem dentro da faixa de $T \pm 10\%$;



(a) (b)

Figura 2.12: Comportamento do controlador PID para T=50 μs.



(a) (b)

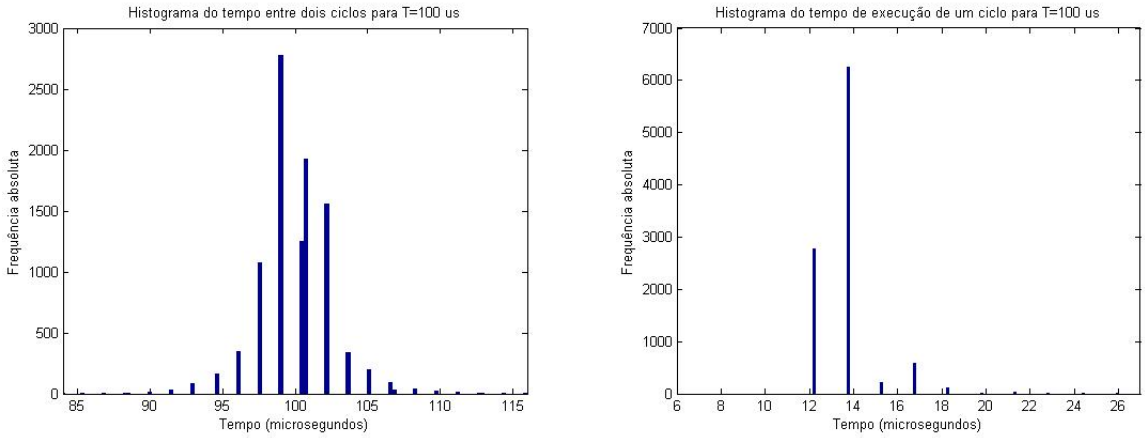
Figura 2.13: Comportamento do controlador PID para T=60 μs.

3. Intervalo de amostragem dentro da faixa de $T \pm 5\%$;

4. Intervalo de amostragem dentro da faixa de $T \pm 1\%$.

A Tabela 2.1 mostra os resultados do cálculos expressos em porcentagem. Pode-se perceber que quanto maior é o intervalo de amostragem menor é a variação em T. Pode-se perceber também que quanto mais restrita é a variação no intervalo de amostragem, maior é o intervalo de amostragem que cumpre a restrição.

Como mencionado anteriormente, a tolerância à variação no intervalo de amostragem depende da aplicação. Porém, para definir o intervalo mínimo da plataforma, é estabelecido que uma variação de 10% é aceitável. Assim, o intervalo mínimo da plataforma é de 60 μs, pois ela cumpre a restrição em 97,66% da vezes. Como mostra a tabela 2.1, para $T \geq 60\mu s$ o tempo



(a) (b)

Figura 2.14: Comportamento do controlador PID para T=100 μs.

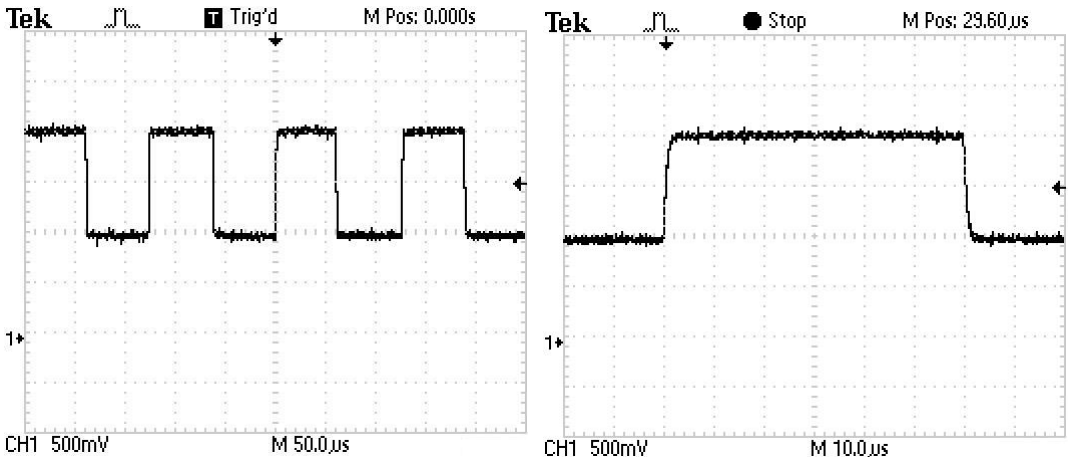
Tabela 2.1: Variação do intervalo de amostragem e da duração do ciclo de controle.

T	50 μs	60 μs	100 μs
Ciclo de controle com duração menor que 20μs	98,80	99,61	99,77
Intervalo de amostragem dentro da faixa de $T\pm10\%$	95,63	97,66	99,30
Intervalo de amostragem dentro da faixa de $T\pm5\%$	65,70	89,81	92,70
Intervalo de amostragem dentro da faixa de $T\pm1\%$	26,86	40,12	59,54

máximo de duração do ciclo de controle é 20 μs em mais de 99,60% das vezes.

O intervalo mínimo de amostragem definido para a plataforma desenvolvida é de 60 μs. Isso não significa que o valor mínimo deve ser utilizado, pois a definição do intervalo de amostragem depende da dinâmica do processo e a variação considerada aceitável depende da aplicação.

Também é importante, para o controle, saber como a tensão de saída da placa de aquisição de dados de comporta quando $T = 60\mu s$. O aplicativo utilizado é o mesmo da seção 2.3.2, por isso é esperado que seja gerada uma onda quadrada com amplitude de 1V e que a tensão de saída mude de valor a cada 60 μs. Na Figura 2.15a, cada divisão vertical representa 500mV e cada divisão horizontal representa 50μs. Essa figura mostra que a tensão de saída se comporta conforme o esperado. Na Figura 2.15b, cada divisão vertical representa 500mV e cada divisão horizontal representa 10μs. Essa figura mostra mais detalhadamente o comportamento da tensão de saída. É possível perceber que existe uma certa dinâmica de subida e de descida da tensão aplicada. Ou seja, a tensão atinge o valor especificado em menos 2μs. Portanto, conclui-se que o conjunto driver e placa de aquisição de dados não acrescenta nenhuma restrição ao intervalo mínimo proposto.



(a) (b)

Figura 2.15: Resultados do teste realizado com T=60 μ s.

3 *Implementação do controlador PID*

Neste capítulo, são descritos o controlador PID industrial utilizado, o modo como é feita a discretização do algoritmo desse controlador e detalhes da implementação. São mostrados também a função dos recursos de programação em tempo real utilizados, como threads e semáforos, e como é realizada a comunicação entre as diversas threads.

Apesar de ser implementado apenas o controlador PID industrial, a plataforma de desenvolvimento mostrada no capítulo 2 pode ser utilizada para implementar diversos tipos de controladores. Portanto, nada impede que, futuramente, a plataforma de controle desenvolvida seja ampliada com outros controladores.

É importante ressaltar que, para que a plataforma de controle seja a mais genérica possível, não é feita a transformação das variáveis *PV*, *MV* e *SP* para unidades de engenharia. Durante o desenvolvimento, para essas variáveis, é utilizado o valor absoluto da transformação A/D e D/A, ou seja um número entre 0 e 65535 DAC.

3.1 Discretização do PID

Segundo [Dorf and Bishop, 2001], a equação ideal de um controlador PID tem a seguinte forma:

$$u(s) = \left(K_c + \frac{K_i}{s} + sK_d \right) e(s). \tag{3.1}$$

Onde e representa o erro, u a ação de controle, K_c o ganho proporcional , K_i o ganho integral e K_d o ganho derivativo.

A equação 3.1 pode ser reescrita como:

$$u(s) = K_c \left(1 + \frac{1}{sT_i} + sT_d \right) e(s). \tag{3.2}$$

Onde T_i representa o tempo integral e T_d representa o tempo derivativo.

Segundo [Persechini, 2008], na prática, não existe apenas uma estrutura para representar

um controlador PID, para diferentes aplicações existem diferentes formas. Neste projeto, o controlador PID industrial é implementado na plataforma de controle conforme descrito em [Clarke, 1984]. A equação do controlador PID industrial é:

$$u(s) = K_c \left(1 + \frac{1}{sT_i}\right) \left(SP(s) - \frac{1 + sT_d(1+\alpha)}{1 + \alpha sT_d} PV(s)\right) \tag{3.3}$$

onde *PV* representa a variável controlada e *SP* representa a referência. Na Figura 3.1 está representado o algoritmo PID industrial que é discretizado e implementado conforme [Clarke, 1984].

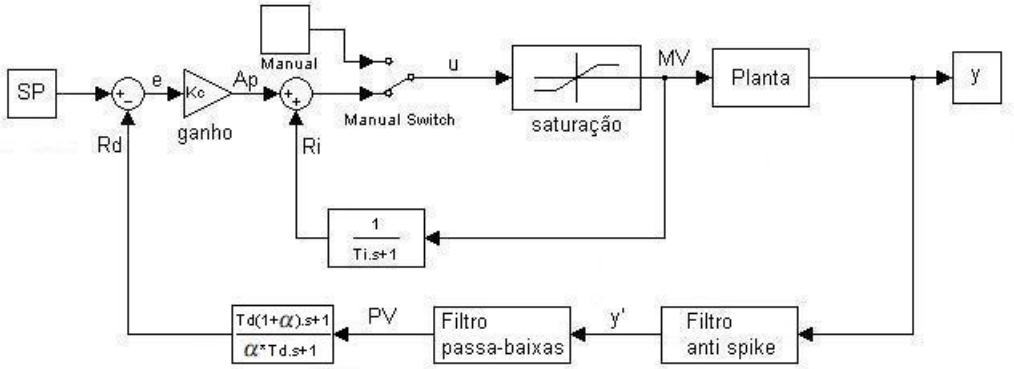


Figura 3.1: Esquema do algoritmo PID contínuo utilizado.

Segundo [Clarke, 1984], controlador PID industrial tem muitas das características desejáveis em um controlador usado em ambiente industrial. Essas características serão descritas na seção 3.2. Todo o desenvolvimento do projeto é baseado na equação 3.3 e na Figura 3.1, por isso as variáveis citadas são relativas a essa equação e a essa figura.

É importante ressaltar que os parâmetros K_c, T_i e T_d do controlador PID industrial, equação 3.3, não são iguais em valor aos parâmetros do PID clássico, equação 3.1. Segundo [Clarke, 1984], para α=0, os parâmetros calculados para um controlador clássico estão relacionados aos parâmetros de um controlador industrial pelas seguintes equações:

$$K_{c_{clas}} = K_c \left(1 + \frac{T_d}{T_i}\right) \tag{3.4}$$

$$T_{d_{clas}} = \frac{T_i T_d}{T_i + T_d} \tag{3.5}$$

$$T_{i_{clas}} = T_i + T_d \tag{3.6}$$

Onde, os parâmetros do PID clássico foram designados nas equações 3.4, 3.5 e 3.6 por $K_{c_{clas}}$, $T_{i_{clas}}$ e $T_{d_{clas}}$. Para $\alpha \neq 0$, as equações 3.4, 3.5 e 3.6 continuam válidas desde que o valor de α seja pequeno. Não faz parte do escopo deste trabalho definir os limites de α para o uso das equações citadas.

Segundo [Clarke, 1984], tendo em vista a forma de discretização, existe uma condição para que o termo derivativo possa ser utilizado:

$$2\alpha T_d - T > 0; \tag{3.7}$$

$$2\alpha T_d > T. \tag{3.8}$$

Portanto, depois de escolher os parâmetros T_d, α e T é preciso verificar se a condição 3.8 é satisfeita. Caso a condição 3.8 não seja satisfeita, então o valor de T_d deve ser zero.

3.2 Cuidados ao se discretizar o PID

Teoricamente, para se obter o algoritmo discreto de um PID basta discretizar a estrutura selecionada. Na prática, alguns cuidados devem ser tomados, entre eles:

- Limitação da ação de controle;
- Transferência suave de manual para automático;
- Impedimento do chute derivativo;
- Filtragem do sinal derivativo.
- Anti Wind-up

Para o controle de plantas industriais, devem ser utilizados algoritmos que levem as diversas características do ambiente industrial em consideração. O algoritmo PID industrial é utilizado como estudo de caso neste trabalho por incluir todos os cuidados citados [Clarke, 1984].

3.2.1 Limitação da ação de controle

Consiste em limitar o valor calculado de $u(k)$ e utilizar este valor nos cálculos relativos à próxima amostragem, até que o valor calculado seja novamente menor que o limite estipulado. Ou seja, caso o valor de $u(k)$ ultrapasse os limites estipulados, é o valor limitado que será utilizado nos cálculos seguintes.

Na implementação do controlador, é considerada uma saturação máxima e mínima, para evitar que a saída do controlador esteja fora dos limites da capacidade da placa de aquisição de dados ou do atuador. A capacidade de escrita da placa é de 0 a 65535 DAC, correspondente a

faixa de tensão entre –10V a 10V, e o atuador da planta deve receber um sinal de tensão entre 0 a 10 V. De acordo com a Figura 2.3, a placa de aquisição deve trabalhar entre 32780 e 65535 DAC para que a tensão no atuador esteja entre 0 e 10V. Como mostra a Figura 3.1, as ações de saturação foram inseridas tanto para o modo de operação manual quanto para o automático, de forma a proteger a planta em todos os casos.

3.2.2 Transferência suave de manual para automático

O controlador industrial deve garantir uma mudança suave entre o modo de operação manual e o automático. Isso pode ser obtido garantindo que o PID apresente, no momento da troca de manual para automático, um valor para u aproximadamente igual ao valor de MV escolhido pelo operador em modo manual. Na prática, basta que R_i também seja calculado em modo manual.

Como mostra a Figura 3.1, quando se opera em modo manual, o termo R_i não está mais em uma malha fechada e está relacionado com MV pela equação:

$$\frac{R_i(s)}{MV(s)} = \frac{1}{sT_i + 1}, \tag{3.9}$$

que é exatamente a equação de um filtro passa-baixas. Dessa forma, o valor de R_i tende ao de MV.

Porém, o sucesso dessa transferência suave depende do operador, uma vez que ele deve selecionar um valor adequado de MV e mantê-lo por um certo tempo, antes de trocar para o modo automático.

3.2.3 Impedimento do chute derivativo

Quando ocorre uma mudança na referência, pode ocorrer uma grande variação no erro que resulta em um alto valor do termo derivativo. Para se evitar esse efeito, o termo derivativo deve ser calculado apenas sobre o valor da PV e não do erro.

3.2.4 Filtragem do sinal derivativo

Em princípio, o ganho da resposta em freqüência do termo derivativo cresce sem limite a medida que a freqüência aumenta. Na prática, o termo derivativo deve ser limitado para as freqüências acima da faixa de passagem da planta, na qual predominam as componentes de ruído do processo e de medição. Essa filtragem pode ser conseguida usando, na ação derivativa,

um filtro (F) da forma:

$$F(s) = \frac{1}{1 + \alpha s T_d} \tag{3.10}$$

3.2.5 Anti Wind-up

Segundo [Clarke, 1984], se, normalmente durante a partida da planta, o erro permanece alto por um grande intervalo de tempo e o erro é integrado, o termo integral pode se tornar maior que o limite estabelecido para a variável manipulada e isso faz com que a ação de controle sature. Assim, mesmo quando PV atinge o valor de SP, a ação de controle pode continuar saturada até que o termo integral seja reduzido. Essa situação resulta em um indesejável *overshoot*.

O erro também pode permanecer alto por um grande intervalo de tempo se ocorrer uma saturação do atuador. Assim o termo integral pode se tornar muito grande se não for houver uma ação anti wind-up. Por isso, deve ser feita a desaturação da ação integral, que consiste em limitar o valor do termo integral [Clarke, 1984]. Neste trabalho, o valor limite é o valor em que a ação de controle satura.

Para evitar o wind-up, o controlador PID industrial implementa o ganho integral em uma realimentação positiva interna ao controlador. Dessa forma, o valor do termo integral (R_i) é limitado ao valor máximo da ação de controle.

3.3 Processamento de sinais

O sinal recebido do sensor (y) não pode ser utilizado diretamente no controlador PID. Antes, ele passa por dois filtros: o passa-baixas e o anti-spike.

3.3.1 Filtro anti-spike

Segundo [Magalhães, 2008], um problema presente na malha de temperatura da planta piloto SCVT é a presença de ruído *spike*. O referido ruído possui a característica de provocar variação muito brusca na medição de uma variável e, na amostragem seguinte, o valor da variável medida retorna ao seu valor original, ou próximo disso. Se esse tipo de ruído não for filtrado, variações bruscas e indesejáveis na variável manipulada também podem ocorrer, prejudicando a eficácia do controlador (aumento de variabilidade no sinal de controle).

De acordo com a Figura 3.1 a entrada do filtro é y, a saída é y'. No filtro anti-spike, quando uma amostra do sinal obtida desvia-se bruscamente de sua média, essa amostra é considerada

um *spike*. No caso de ocorrer um *spike*, o valor da amostra é substituído por um valor próximo do valor da amostra anterior [Clarke, 1984]. A seguir é mostradao o código em C de uma das possíveis implementações do filtro anti-spike.

```
unsigned int filtroAntiSpike(unsigned int yEntrada){
    int D;
    int Delta;
    int moduloDelta;
    unsigned int ySaida=0;
    if(Dmin<=0){
      yAnterior=yEntrada;
      ySaida= yEntrada;
    }
    else{
        D=(int)Dmin*pow(2,S);
        Delta=yEntrada-yAnterior;
        moduloDelta=Modulo(Delta);
        if(moduloDelta>D){
            if(Delta>0)
                ySaida=yAnterior+(unsigned int)D;
            else
                ySaida=yAnterior+(unsigned int)(-D);
            S++;
        }
        else{
            ySaida=yEntrada;
            S--;
        }
        if(S<0)
            S=0;
        yAnterior=ySaida;
    }
    return ySaida;
}
```

O parâmetro do filtro é Dmin, que indica a janela inicial e a janela mínima para o filtro

anti-spike. Dmin deve ter um valor próximo da maior variação, entre duas amostras, esperada. A variável S, que possui valor inicial igual a zero, junto com Dmin são responsáveis por definir a janela corrente. É possível perceber que a janela não possui um valor fixo. A amostra obtida (yEntrada) não será considerada um spike se:

$$yAnterior - 2^S Dmin \leq yEntrada \leq yAnterior + 2^S Dmin. \tag{3.11}$$

Apesar de ser desejável a presença do filtro anti-spike na plataforma de controle, ele não faz parte do algoritmo PID industrial. Por isso, o filtro anti-spike pode ser desativado escolhendo-se um valor menor ou igual a zero para o parâmetro Dmin.

3.3.2 Filtro passa-baixas

O filtro derivativo pode não ser suficiente para condicionar o sinal. Por isso, adiciona-se um filtro passa-baixas auxiliar, que recebe como entrada a saída do filtro anti-spike. A função transferência em z do filtro passa-baixas é:

$$\frac{PV}{y'} = \frac{(1-\beta)z^{-1}}{1-\beta z^{-1}}. \tag{3.12}$$

Para $0 \leq \beta \leq 1$.

Em equação de diferenças:

$$PV(k) = \beta PV(k-1) + (1-\beta)y'(k). \tag{3.13}$$

Para desabilitar esse filtro, basta fazer $\beta = 0$ na equação 3.13. Isso resulta em:

$$PV(k) = y'(k). \tag{3.14}$$

3.4 Desenvolvimento do aplicativo para controle

Para implementar o controlador PID é desenvolvido, em linguagem C, um programa multithread. Considerando a interface com o usuário como uma thread, existem 5 threads. A função de cada uma pode ser resumida em:

- Interface: cria a interface com o usuário e permite que ele escolha os parâmetros do controlador, o modo de operação, o intervalo de amostragem e as constantes relacionadas com os filtros implementados;

- rt_thread: cria as demais threads que usam métodos do RTAI e os semáforos que protegem as seções críticas;

- Thread_Leitura: lê os dados da porta A/D da placa de aquisição de dados, executa o algoritmo PID e escreve na porta D/A da placa de acordo com o intervalo de amostragem especificado;

- Thread_Escrita_Arquivo: cria um histórico com as principais variáveis envolvidas na operação, ou seja, PV, SP, MV, modo de operação (manual/automático) e tempo de início e fim de cada ciclo;

- Thread_Entrada_Dados: copia os valores dos parâmetros, escolhidos pelo usuário, da interface para as variáveis globais usadas pelo algoritmo PID industrial em Thread_Leitura.

São utilizadas várias threads para possibilitar uma melhor organização, por limitações técnicas e para separar o código que é executado periodicamente do código que é executado apenas quando um determinado evento ocorre. Por exemplo, threads que utilizam a biblioteca GTK+ não podem usar funções do RTAI. Além disso, é recomendada a separação entre o código que usa funções do Linux padrão e o código que usa funções do RTAI, pois toda vez que uma função do Linux é utilizada, o controle da operação deixa de ser do RTAI e passa a ser do Linux padrão, ou seja, perde-se o determinismo temporal provido pelo RTAI.

O motivo de criação de cada thread pode ser resumido em:

- Interface: a interface, geralmente, é lenta e precisa ser separada das threads rápidas, além disso o uso da biblioteca GTK+ impossibilita o uso de funções do RTAI;

- rt_thread: Como a interface não pode usar funções do RTAI essa thread é responsável por criar as estruturas, providas pelo RTAI, necessárias para o correto funcionamento das demais threads, como os semáforos que protegem as seções críticas. Só depois disso, as demais threads podem ser criadas. Além disso, o código dessa thread só é executado nos momentos de abrir e fechar o aplicativo;

- Thread_Leitura: para uma melhor organização, o código responsável pela execução do algoritmo PID é separado do restante do código. Essa thread é executada periodicamente, com período igual ao intervalo de amostragem;

- Thread_Escrita_Arquivo: o código dessa thread faz uso de funções do Linux padrão para manipular arquivo, por isso deve ser separado do restante do código;

- Thread_Entrada_Dados: para copiar os valores dos parâmetros da interface para as variáveis globais usadas pelo algoritmo PID industrial o código dessa thread faz uso de funções do Linux padrão. Além disso, esse código só é executado quando o usuário clica no botão Enviar.

A comunicação entre as threads é feita com o uso de memória compartilhada, ou seja, os dados são armazenados em variáveis globais e podem ser lidos e modificados por qualquer thread. Para controlar o acesso às variáveis globais, são usados semáforos que garantem que apenas uma thread de cada vez possa acessar as variáveis. Essa forma de comunicação foi escolhida porque apresenta as seguintes vantagens:

- Possui baixo overhead;
- Não há perda de informação;
- É simples de implementar.

A thread Interface cria a janela de interface com o usuário. É importante ressaltar que a interface foi feita com o uso da biblioteca GTK+ que está disponível sob licença GPL. Primeiramente, a janela de interface é exibida com o botão Enviar desabilitado e os botões Ajuda e Inicializar habilitados, conforme a Figura 3.2a. Antes de clicar em Inicializar, o usuário deve preencher os seguintes campos:

- Arquivo: é o nome do arquivo que será criado para salvar os dados obtidos durante a execução do controlador PID;
- T: o intervalo de amostragem, em milissegundos, é o intervalo entre duas amostragens. Nesse período, o controlador deve ser capaz de calcular ação de controle e enviar esse valor ao atuador. O valor mínimo utilizado foi de 60 μs;
- P: parâmetro proporcional (K_c) do controlador PID industrial, equação 3.3;
- I: parâmetro integral (T_i) do controlador PID industrial, equação 3.3;
- D: parâmetro derivativo (T_d) do controlador PID industrial, equação 3.3;
- Alfa: parâmetro para ajuste do filtro associado ao termo derivativo do controlador PID industrial, equação 3.3;
- Beta: parâmetro para ajuste do filtro auxiliar, equação 3.13;

- Dmin: parâmetro para o filtro anti-spike. Define o tamanho inicial da janela que é usada para definir se ocorreu spike.

(a) (b) (c)

Figura 3.2: Tela e mensagem da interface com usuário.

Se a inicialização for bem sucedida a mensagem de sucesso da Figura 3.2c será exibida. O usuário pode então fazer o controle manual ou trocar para controle automático, como mostra a Figura 3.2b. Ao clicar em Enviar, a thread armazenará os parâmetros (K_c, T_i, T_d, α, β, Dmin, SP ou MV e modo de operação) em variáveis globais e sinalizará um evento que deve ser recebido pela thread Thread_Entrada_Dados.

A inicialização consiste em:

- criar um arquivo com o nome selecionado;
- desabilitar o botão Inicializar;
- habilitar o botão Enviar;
- inicializar o controlador em modo manual;
- inicializar a placa de aquisição de dados;
- criar thread rt_thread.

Se o usuário optar por sair do programa, a thread Interface sinaliza para as demais threads o fim da execução. Cada thread fica responsável por concluir suas operações e terminar.

A thread denominada rt_thread faz com que todas as threads e suas variáveis sempre estejam na memória. Isso diminui o tempo necessário para o sistema operacional retomar a execução

dela e de qualquer thread que venha a ser criada por ela. Também é responsável por criar as demais threads de tempo real e criar os semáforos que são usados para acessar as variáveis globais. Em seguida, fica esperando um evento oriundo da interface, que indica a solicitação do usuário de encerrar o aplicativo. Quando o evento é recebido, os semáforos são destruídos e a thread encerra sua execução.

A thread Thread_Leitura, primeiramente, cria um *timer* com o período igual ao intervalo de amostragem (T). Isso faz com que a thread seja acordada de T em T milissegundos. Depois, essa thread entra em um *loop* que executa as seguintes ações:

1. verifica o *flag* de fim de execução. Caso o *flag* esteja setado, a thread encerra;
2. lê da placa o valor da variável controlada;
3. entra na seção crítica;
4. executa o algoritmo do controlador PID industrial;
5. escreve na placa o valor da variável manipulada;
6. armazena os valores em variáveis globais do tipo buffer circular;
7. armazena o valor -1 na próxima posição dos buffers circular;
8. sai da seção crítica;
9. dorme (o *timer* é responsável por acordar a thread no momento certo).

A Thread_Escrita_Arquivo, primeiramente, cria um *timer* com o período dez vezes maior que o intervalo de amostragem. Isso faz com que essa thread seja executada de 10T em 10T milissegundos. Depois essa thread fica em um *loop* que executa as seguintes ações:

1. verifica o *flag* de fim de execução. Caso o *flag* esteja setado, a thread encerra;
2. entra na da seção crítica;
3. copia os dados dos buffers circulares para buffers auxiliares até que se obtenha o valor -1;
4. sai da seção crítica;
5. transfere os dados copiados nos buffers auxiliares para o arquivo criado pela thread Interface durante a inicialização;

6. dorme (o *timer* é responsável por acordar a thread no momento certo).

A Thread_Entrada_Dados, primeiramente, cria um mapa de eventos com os eventos que serão monitorados. Depois, a thread fica em um *loop* que consiste em esperar a chegada de um evento e tratar esse evento. Se o evento for SIGUSR1, a thread:

1. entra na seção crítica;
2. copia os valores dos parâmetros da interface para as variáveis globais que serão usadas durante a execução do algoritmo do PID industrial em Thread_Leitura e
3. sai da seção crítica.

Se o evento for SIGUSR2, que sinaliza que o usuário deseja terminar o programa, a thread encerra a execução.

4 Testes da plataforma

Nesse capítulo a planta piloto é brevemente apresentada, em seguida, utilizando-se a plataforma desenvolvida, são realizadas coletas de dados para a modelagem da malha de temperatura da planta piloto. Após a obtenção do modelo, um controlador PID é projetado e a plataforma desenvolvida é novamente utilizada para a implementação do controlador.

Como dito anteriormente, para que a plataforma de controle seja a mais genérica possível, não é feita a transformação das variáveis *PV*, *MV* e *SP* para unidades de engenharia. O valor dessas variáveis é o valor absoluto da transformação A/D e D/A, ou seja um número entre 0 e 65535 DAC. Porém, em determinados momentos, serão feitas referências à tensão e a potência para situar melhor o leitor.

4.1 Planta Piloto

Segundo [Magalhães, 2008], o Sistema de Controle de Vazão e Temperatura de Ar, denominado SCVT, foi concebido no Laboratório de Controle de Processos Industriais, LCPI, com o principal objetivo de representar, em escala reduzida, o processo real de secagem de pelotas de minério de ferro. A planta piloto tem características semelhantes às características das plantas reais de pelotização, tais como comportamento dinâmico não-linear, acoplamento entre as variáveis e ser multi-variável.

O sistema é composto de um ventilador, que é acionado por um conjunto inversor de frequência - motor, que insufla ar através de um duto (túnel) de alumínio. Nesse duto, há uma resistência elétrica que aquece o ar insuflado pelo ventilador. Logo na saído duto, há o conjunto de medição (instrumentação), composto pelos sensores de temperatura e vazão de ar.

Pode-se perceber que existem duas malhas de controle no SCVT: uma de temperatura e uma de vazão. Neste trabalho foi utilizada apenas a malha de temperatura. A seguir, são descritas em detalhes as características do sensor e do atuador da malha de temperatura.

4.2 Sensor

O sensor de temperatura realiza a medição dessa variável por meio de um termistor com coeficiente de temperatura negativo (NTC) que é ligado em uma Ponte de Wheatstone. Variações de temperatura no termistor provocam mudanças no valor da resistência, causando um desequilíbrio na ponte de modo a alterar a tensão. Essa variação de tensão é amplificada. A Figura 4.1 apresenta um diagrama esquemático do conjunto do sensor de temperatura (NTC - Ponte de Wheatstone)[Magalhães, 2008].

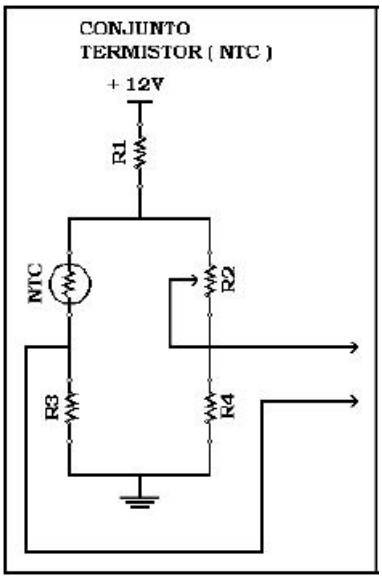


Figura 4.1: Diagrama esquemático do sensor de temperatura - Termistor (NTC) - Ponte de Wheatstone.

4.3 Atuador

O atuador da malha de temperatura é composto pelo circuito de potência e resistências elétricas responsáveis pelo aquecimento da corrente de ar interna do duto. Podem ser usadas três resistências cônicas de estufa padrão, com 400W de potência cada, o que resultaria em uma potência total máxima de 1200W. Apesar de o sistema ter capacidade para três resistências, apenas uma foi utilizada neste trabalho. O circuito de aquecimento do ar foi limitado ao uso de apenas uma resistência cônica devido ao fato de o sensor de vazão ter um faixa de operação limitada de 0 a 50 ^{o}C. O uso de três resistências faria com que a temperatura pudesse alcançar 85 ^{o}C, ou seja, prejudicaria a malha de vazão.

A energia térmica entregue ao ar, a cada instante, é função do ângulo de disparo do TRIAC. Se o ângulo de disparo é 0^{o} (sinal de controle de temperatura igual 0 V), a tensão de saída

TRIAC é máxima. Assim, a potência elétrica entregue às resistências também é máxima (400W). Para o ângulo de 180^o, a tensão de saída e a potência são nulas (sinal de controle de temperatura igual a 10 V). O valor do ângulo de disparo do TRIAC é convertido por uma interface que recebe sinais de tensão de 0 a 10 V e envia pulsos ao *gate* do TRIAC em instantes de tempo correspondentes ao ângulo de disparo definido. O circuito de potência (controle de disparo) é sempre sincronizado com a frequência da rede elétrica.

4.4 Conexão da planta piloto à plataforma

Neste projeto, não se modificou a planta piloto, porém um *borne* intermediário foi necessário para se fazer a ligação entre a planta e a plataforma desenvolvida. Na Figura 4.2, é possível ver como os principais componentes estão interligados.

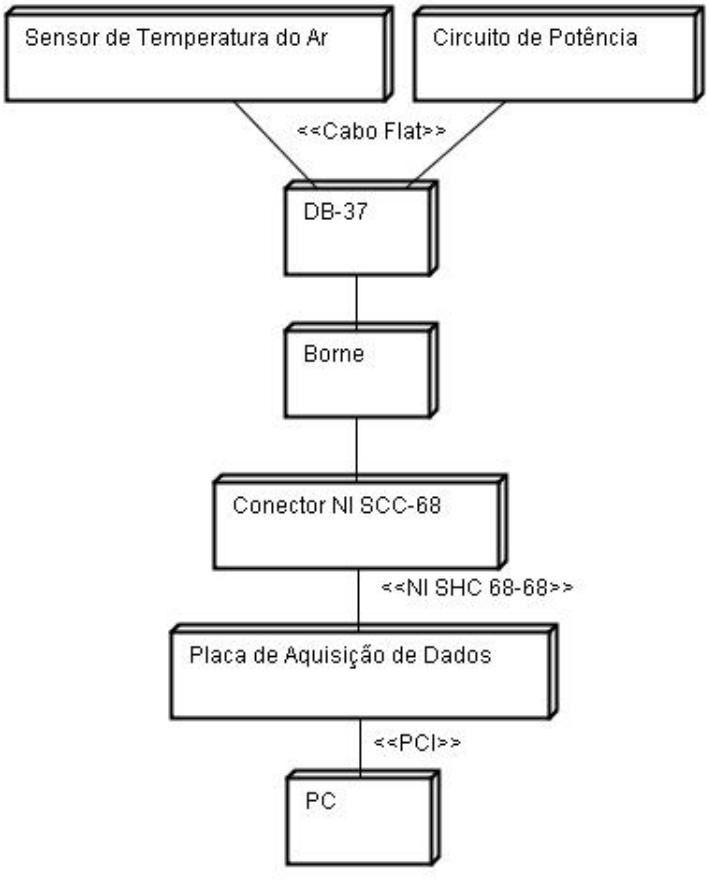


Figura 4.2: Diagrama de Ligação.

A placa de aquisição de dados se conecta à placa mãe do computador por meio de um entrada PCI. Um cabo NI SHC 68-68 liga a placa ao conector NI SCC-68. O conector possui locais demarcados para cada entrada e saída onde se conecta fios que se ligam ao *borne*. Dez das doze posições do *borne* estão ligadas, por meio de fios, a pinos do conector DB-37. Esse

conector se liga à planta por meio de um cabo Flat de 34 vias. Internamente à planta, as vias 11 e 12 estão conectadas ao circuito de potência e resistências elétricas e as vias 22 e 23 estão ligadas ao sensor de temperatura.

A Tabela 4.1 mostra os pinos do conector que estão ligados borne, que por sua vez está ligado ao conector BD-37, e a função de cada posição do *borne*.

Tabela 4.1: Tabela de Ligação.

Pino NI SCC-68	Borne	Pino DB-37	Função
-	1	5	Liga/Desliga - Motor
-	2	6	Gnd
52	3	7	Liga/Desliga - Aquecedor
4	4	8	Gnd
-	5	9	Saída Analógica (0-10V) Inversor
-	6	10	Gnd
22	7	11	Saída Analógica (0-10V) Aquecedor
54	8	12	Gnd
-	9	-	Canal 1 conversor A/D - Sensor de Vazão
-	10	-	Gnd
68	11	23	Canal 1 conversor A/D - Sensor de Temperatura
34	12	22	Gnd

4.5 Modelagem da malha de temperatura da planta piloto

Segundo [Magalhães, 2008], a frequência da tensão aplicada ao motor deve estar entre 20 Hz e 100 Hz. Durante a modelagem da malha de temperatura, a vazão de ar é mantida constante, com frequência de 20 Hz aplicada ao motor. Isso faz com que a vazão seja a mínima possível e, portanto, a constante de tempo da malha de temperatura seja a menor possível.

Primeiramente é preciso decidir em qual região de operação da planta o modelo da malha será obtido. Para isso é preciso obter a curva da relação estática entre entrada e a saída. Por isso, são aplicados oito valores de MV à planta. Para cada valor de MV, espera-se a temperatura estabilizar e obtém-se o valor de PV. A curva da relação estática entre entrada e a saída, presente na Figura 4.3, mostra que a malha apresenta um comportamento mais linear entre 30 e 60% da potência, ou seja entre 4 e 7 Volts, ou ainda para MV entre 45882 e 55708 DAC. Portanto, a malha de temperatura será modelada para operação nessa faixa de potência.

A malha de temperatura é modelada como um sistema de primeira ordem com atraso puro

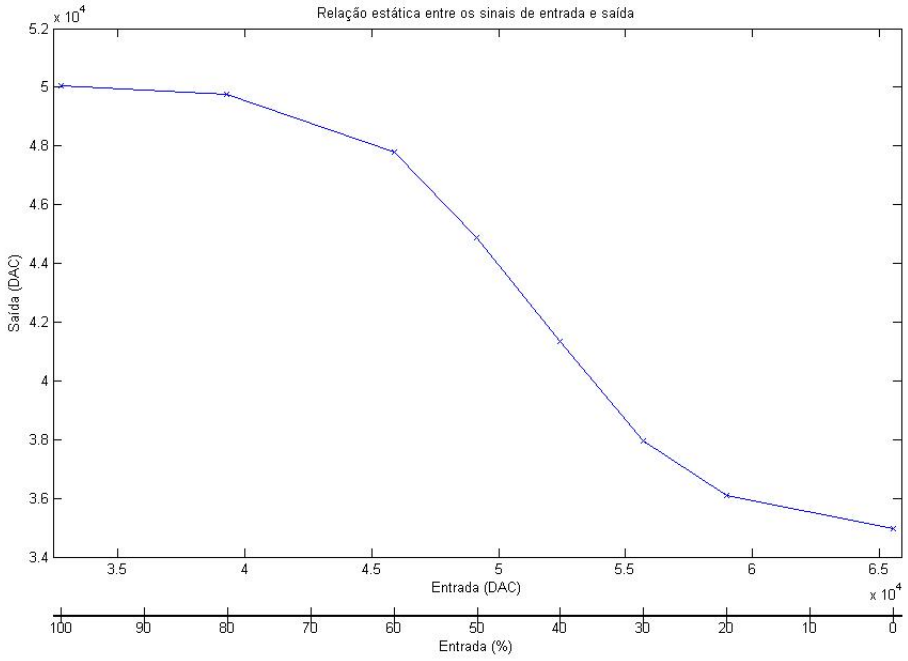


Figura 4.3: Relação estática entre entrada e saída.

de tempo θ e um *offset*, ou seja:

$$G(s) = \frac{Ke^{-\theta s}}{\tau s + 1} + \frac{Offset}{s}. \tag{4.1}$$

Esse modelo foi escolhido por ser simples e, como é mostrado posteriormente, representar satisfatoriamente o comportamento da malha de temperatura.

O *offset* é obtido por meio de regressão linear considerando a região linear da curva de relação estática entre entrada e a saída. A Figura 4.4 mostra a curva de relação estática e a curva da regressão linear cuja equação é:

$$PV = 94371 - 1,0115MV. \tag{4.2}$$

Dessa forma, o valor do *offset* é 94371. É importante ressaltar que o efeito da temperatura ambiente está incluído no *offset*.

Para se obter o comportamento dinâmico, são aplicados degraus na MV de 30% para 50% da potência. Duas massas de dados são utilizadas: uma para a modelagem e uma para a validação do modelo obtido. A primeira massa de dados é mostrada na Figura 4.5, onde no instante de tempo t=520 s o valor de MV muda de 55708 DAC para 49157 DAC.

O atraso puro de tempo (θ) é obtido por inspeção visual e possui valor de 20 s. O ganho é

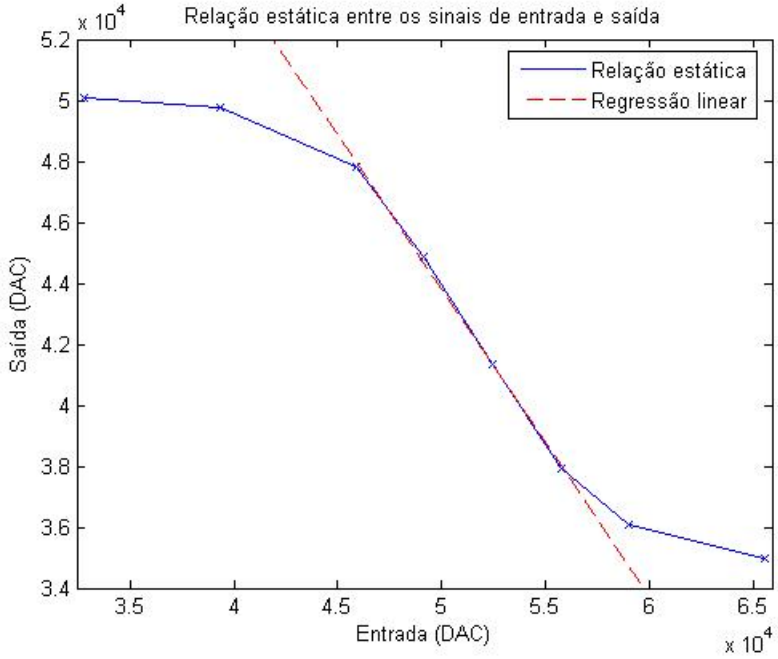


Figura 4.4: Regressão linear na curva de relação estática entre entrada e saída.

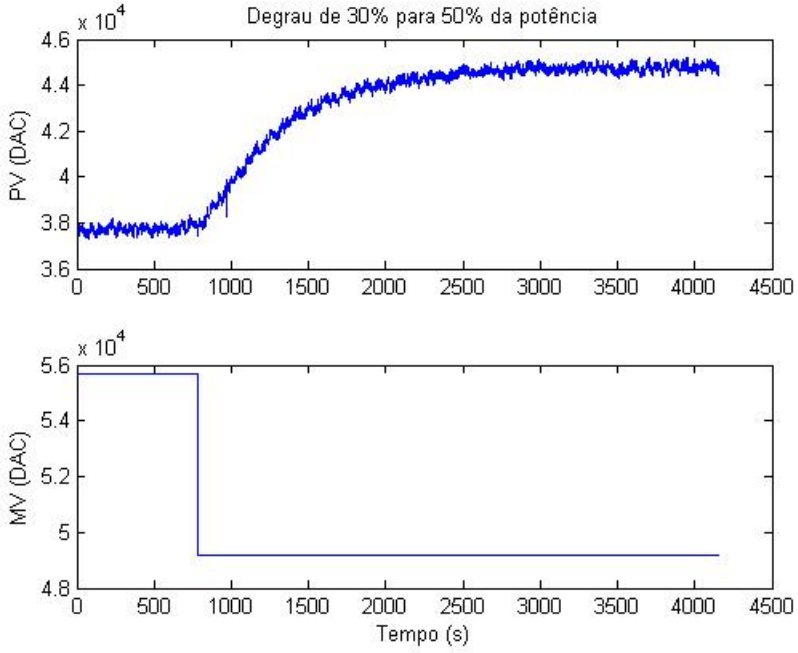


Figura 4.5: Massa de dados utilizada para a modelagem.

definido como:

$$K = \frac{\Delta PV}{\Delta MV} \tag{4.3}$$

em que ΔPV representa a variação na variável controlada (PV) e ΔMV representa a variação da variável manipulada (MV).

O método da resposta complementar [Doebelin, 1990] é utilizado para definir a constante de tempo (τ). Nesse método, a curva do logaritmo da resposta complementar (w) é traçada em função do tempo. Os valores de $w(t)$ são obtidos pela equação:

$$w(t) = ln|1 - \frac{PV_a(t)}{\Delta PV}| \tag{4.4}$$

em que $PV_a(t) = PV(t+\theta)$ desconta o efeito do atraso puro de tempo. Em seguida, o ajuste linear da assíntota de $w(t)$ é feito, para valores altos de t, conforme a equação:

$$w(t) = at + b. \tag{4.5}$$

Para utilizar o método da resposta complementar, primeiramente, desloca-se o zero das curvas de PV e MV, ou seja, subtrai-se 37710 de MV e 55708 de PV em todas as amostras da massa de dados. Em seguida, é feito o ajuste linear da assíntota de w. A Figura 4.6 mostra a assíntota estimada usando $w(t)$ para $100 < t < 1000$. O coeficiente a da assíntota é relacionada com τ por meio de:

$$a = \frac{-1}{\tau}. \tag{4.6}$$

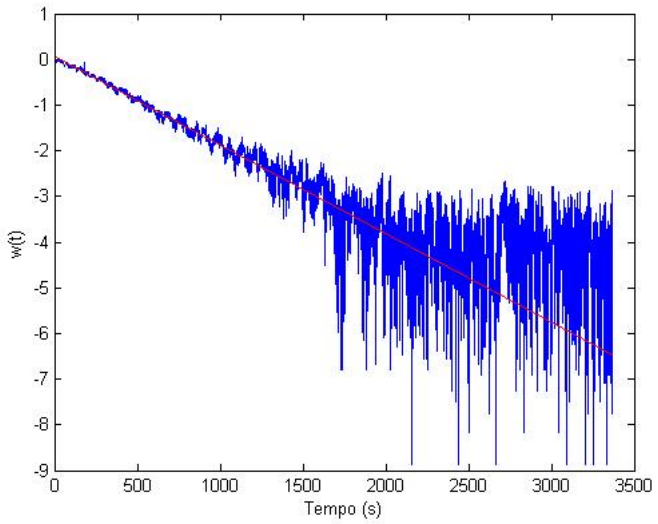


Figura 4.6: Curva do logaritmo da resposta complementar ($w(t)$).

Utilizando as equações 4.3 e 4.6, obtém-se o modelo para a parte dinâmica da malha de temperatura. Portanto,

$$G(s) = \frac{-1,084e^{-20s}}{515,6s+1}. \tag{4.7}$$

A Figura 4.7 e a Figura 4.8 mostram a saída do modelo (sem *offset*) a uma entrada em degrau com amplitude de -6551 comparada, respectivamente, com a primeira e a segunda massa

de dados. Ambas as massas de dados tiveram o zero deslocado. É possível perceber que a parte dinâmica do modelo proposto representa satisfatoriamente a dinâmica da malha de temperatura.

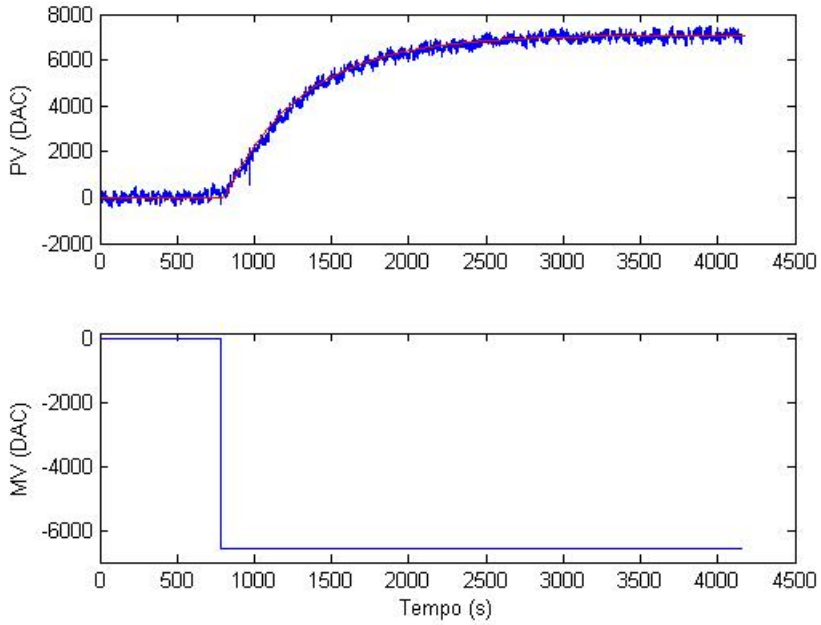


Figura 4.7: Saída do modelo comparado com a massa de dados utilizada na modelagem.

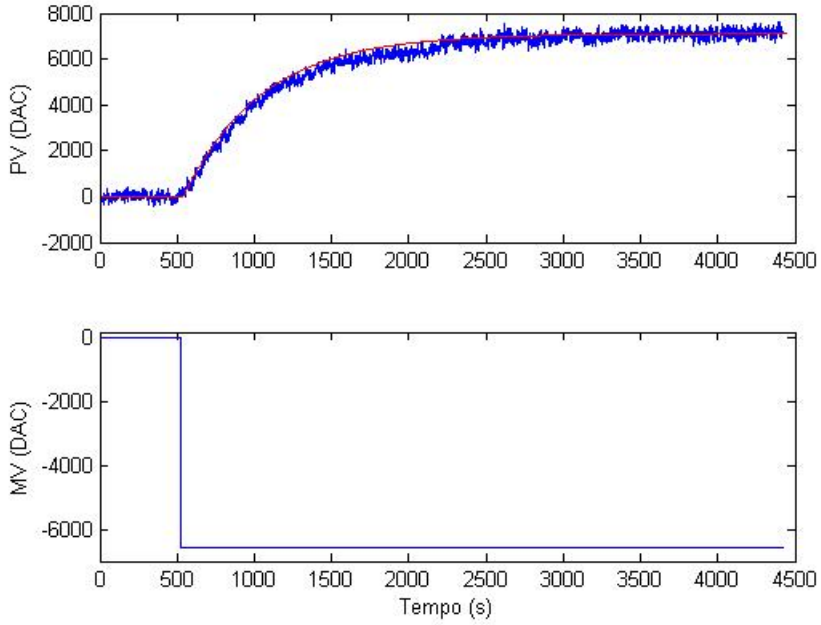


Figura 4.8: Saída do modelo comparado com a segunda massa de dados.

Finalmente, o modelo completo da malha de temperatura é:

$$G(s) = \frac{-1,084e^{-20s}}{515,6s+1} + \frac{94371}{s}. \tag{4.8}$$

4.6 Projeto do controlador PID

Ao fechar a malha de temperatura, é esperado que:

- A rejeição a perturbação aumente, ou seja, caso ocorra uma mudança na temperatura ambiente ou na vazão, o controlador deve ser capaz de compensar essa mudança;
- A malha responda mais rapidamente a uma mudança no *SP* que a malha aberta, ou seja, o valor de *PV* deve atingir o valor do *SP* em um intervalo menor que observado em malha aberta;
- O valor de *PV*, após a temperatura estabilizar, seja igual ao valor de *SP*, ou seja, que não exista erro em estado estacionário;
- Não ocorra overshoot, pois a planta não possui nenhum sistema de resfriamento;
- Manter o atraso puro em 20s, pois não é desejável aumentá-lo e não é possível reduzi-lo.

Portanto, pode-se definir os objetivos do projeto do controlador como:

- Reduzir a constante de tempo para menos de 250s;
- Não apresentar erro em estado estacionário;
- Não apresentar overshoot;
- Manter o atraso puro de tempo;
- Aumentar a rejeição a perturbação.

Uma função transferência que representa o comportamento desejado em malha fechada é:

$$G_d = \frac{e^{-20s}}{180s+1}, \tag{4.9}$$

O projeto do controlador é feito por meio do método do Modelo Interno. [Seborg et al., 1989] mostra que dado um sistema que seja estável em malha aberta e tenha função transferência representável pela equação 4.1, é possível impor a esse sistema o comportamento correspondente a:

$$f = \frac{e^{-\theta s}}{\tau_c s+1}. \tag{4.10}$$

Para isso, basta fechar a malha e utilizar um controlador PID clássico cujos parâmetros K_c, T_i e T_d são calculados segundo as equações:

$$Kc = \frac{1}{K}\frac{2(\frac{\tau}{\theta})+1}{2(\frac{\tau_c}{\theta})+1}; \tag{4.11}$$

$$T_i = \frac{\theta}{2} + \tau; \tag{4.12}$$

$$T_d = \frac{\tau}{2(\frac{\tau}{\theta})+1}. \tag{4.13}$$

[Seborg et al., 1989] recomenda que sejam respeitadas as seguintes restrições:

$$\frac{\tau_c}{\theta} > 0,8; \tag{4.14}$$

$$\tau_c > \frac{\tau}{10}. \tag{4.15}$$

Utilizando τ_c=180 s,θ=20 s e τ=515,6 s nas equações 4.11, 4.12 e 4.13, as restrições 4.14 e 4.15 são obedecidas e os valores K_c=-2,5520, T_i=525,6 e T_d=9,8097 são obtidos. Porém, esses valores devem ser convertidos por meio das equações 3.4, 3.6 e 3.5 antes de serem utilizados em um PID industrial. Assim, os valores convertidos são: K_c=-2,5034, T_i=515,6 e T_d=10.

Após o projeto do controlador são projetados o filtro anti-spike, o filtro passa-baixas, cuja equação é a 3.12, e o filtro do termo derivativo, cuja equação é a 3.10.

O valor de Dmin, que é igual a 300, é escolhido de forma a eliminar o spike detectado na massa de dados utilizada para a modelagem. A Figura 4.9 mostra a massa de dados citada e a saída do filtro anti-spike. A saída do filtro é obtida ao submeter essa massa de dados ao filtro. Nessa figura é possível perceber que o filtro retira o spike que ocorre no instante t=967s, além de manter praticamente inalteradas as outras amostras.

A saída do filtro anti-spike ainda possui ruídos de natureza diferente do spike. O valor de α, para o filtro do termo derivativo, é escolhido de forma a reduzir as componentes desse ruído que possuam frequências acima da faixa de passagem da planta. O filtro derivativo, não pode alterar a constante de tempo da planta, por isso, ele deve reduzir o ruído sem alterar o formato da curva. A saída do filtro derivativo é comparada com a saída do filtro anti-spike para diversos valores de α. Por inspeção visual, o valor escolhido para α é igual a 1,5. A Figura 4.10 mostra a saída do filtro anti-spike comparada com a saída do filtro derivativo quando α=1,5. Esse valor de α modifica a ação derivativa, mas a análise do efeito dessa modificação não faz parte do escopo deste trabalho.

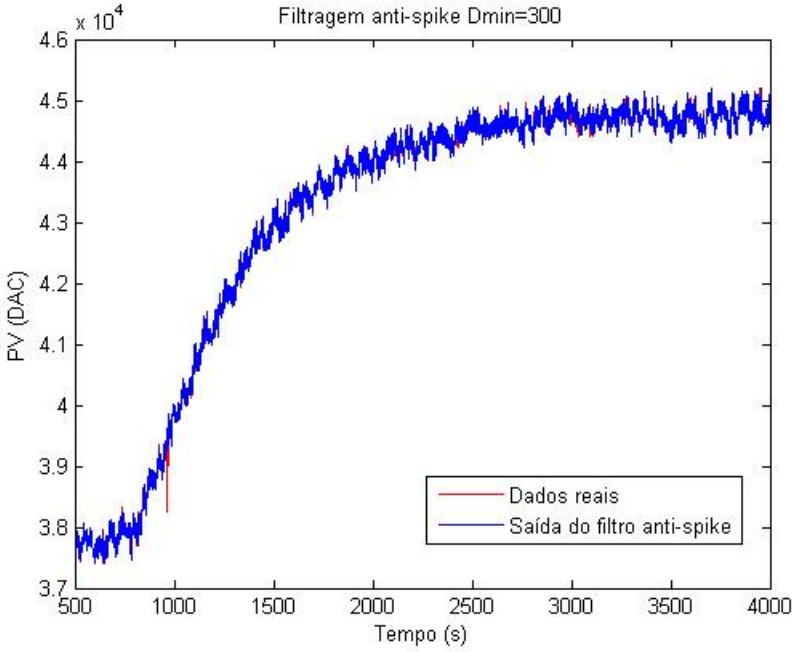


Figura 4.9: Saída do filtro anti-spike quando Dmin=300.

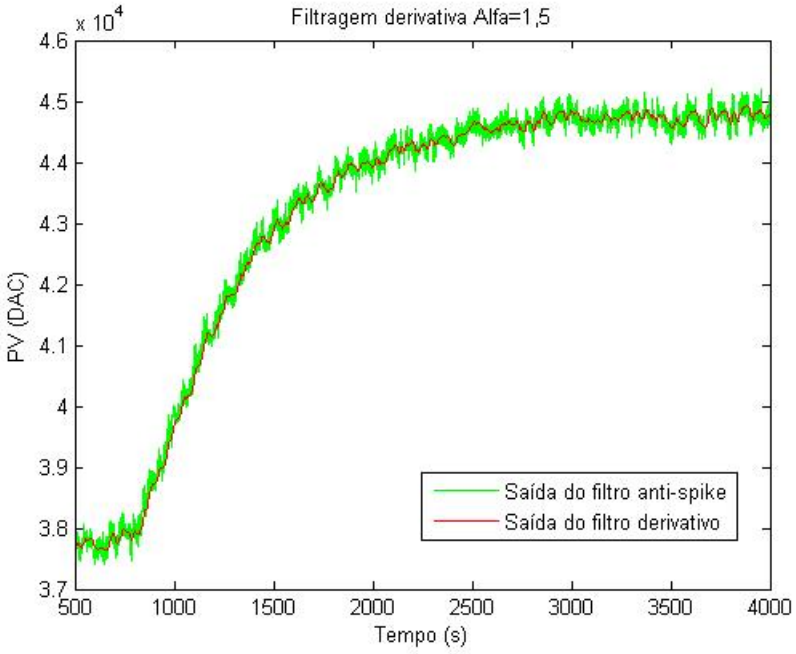


Figura 4.10: Saída do filtro derivativo quando α=1,5.

Como mostra a Figura 4.10, o filtro derivativo com α igual a 1,5 é suficiente para reduzir o ruído de alta frequência presente no sinal. Por isso, não é necessário o uso do filtro passa-baixas digital. Portanto, o valor escolhido para β é zero, ou seja, o filtro passa-baixas é desabilitado.

O intervalo de amostragem é escolhido de forma que ele seja bem menor que a constante de tempo da malha e que o atraso puro de tempo seja um múltiplo inteiro de T. Como a constante

de tempo é de 515,6 s e o atraso puro de tempo é de 20 s, o valor escolhido para T é de 5 segundos.

Como mencionado na seção 3.1, é preciso verificar se a condição para o uso do termo derivativo é satisfeita. Para isso os valores Td=10, α=1,5 e T=5 são substituídos na inequação 3.8 e obtém-se:

$$2 * 1,5 * 10 > 5. \tag{4.16}$$

Portanto, os valores Td=10, α=1,5 e T=5 satisfazem a condição e o termo derivativo pode ser utilizado. É importante resaltar que, geralmente, o valor escolhido para α é 0,1, mas se esse valor fosse utilizado, a condição 3.8 não seria satisfeita e o valor de T_d deveria ser 0. Por isso preferiu-se usar α=1,5, mesmo sabendo que esse valor influencia no termo derivativo projetado.

A Figura 4.11 mostra como é feita a simulação do controlador implementado. Ao comparar essa figura com a Figura 3.1 observa-se que o valor efetivamente enviado para MV é o complemento da saída do controlador. Isso é necessário para que o ganho do controlador seja positivo. Ou seja, dado um projeto de controlador em que o ganho seja negativo, é possível usar um ganho positivo de mesmo módulo para calcular a ação de controle e depois aplicar à planta o valor máximo (65535) subtraído da ação calculada.

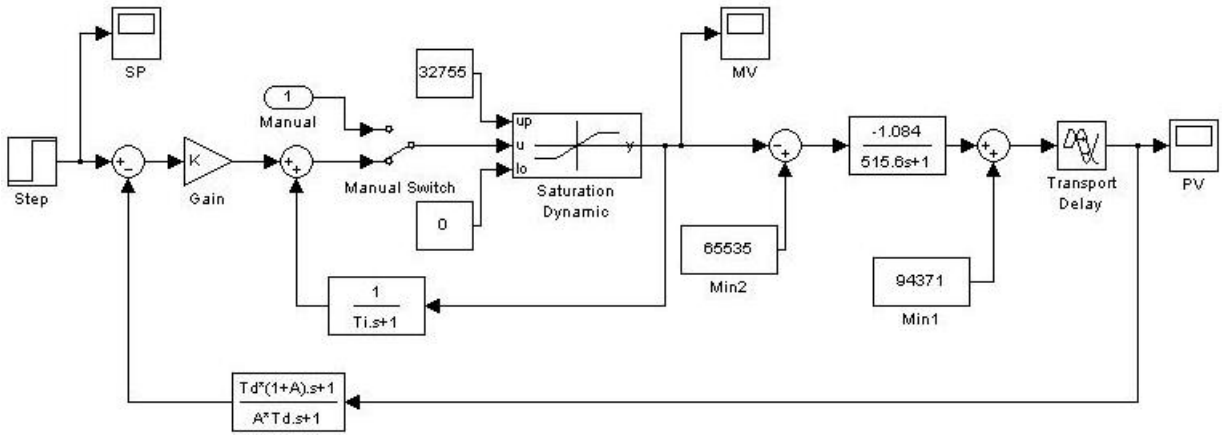


Figura 4.11: Simulação no simulink.

Como o valor calculado de K_c é negativo, é feito o complemento da saída e o ganho do controlador fica então positivo. As modificações necessárias para fazer o complemento da saída são realizadas no código fonte do aplicativo desenvolvido. Portanto, de agora em diante, o controle é feito com o complemento da saída. Isso significa, por exemplo, que MV tem sempre um valor entre 0 e 32755 (não mais entre 32780 e 65535) e que um aumento em MV resulta em um aumento na potência.

A Figura 4.12 mostra o resultado da simulação do comportamento da planta para uma variação no *SP* de 39890 para 43890 DAC.

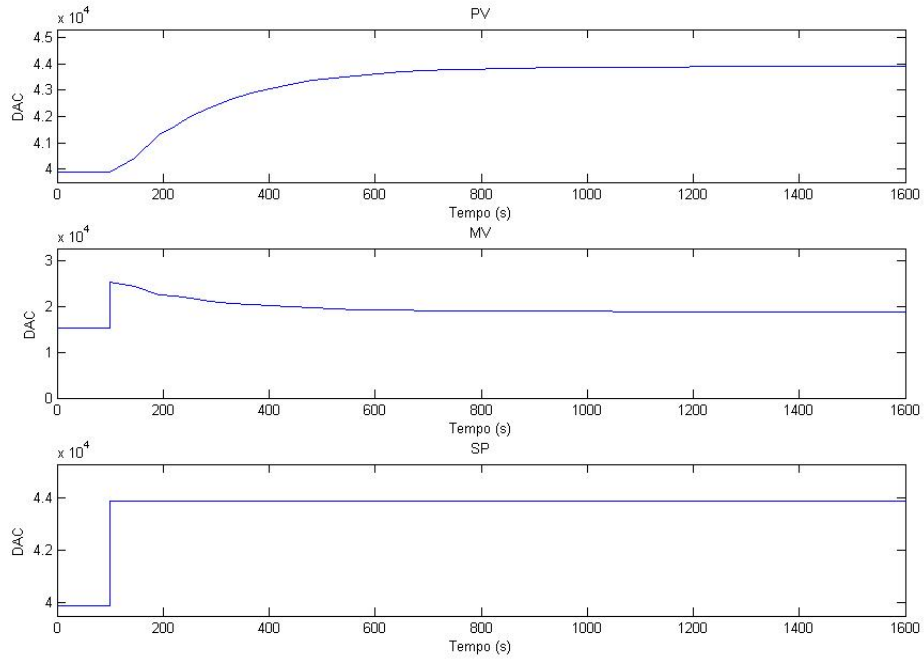


Figura 4.12: Simulação da resposta em malha fechada.

4.7 Resultados Experimentais

A plataforma desenvolvida é utilizada para controlar a temperatura da planta piloto. Os parâmetros projetados na seção anterior são utilizados, exceto o ganho que teve o sinal invertido devido ao uso do complemento da saída. Portanto:

- T=5;
- K_c=2,5034;
- T_i=515,6;
- T_d=10;
- α =1,5;
- β=0;
- Dmin=300.

Os parâmetros são inseridos por meio da interface gráfica. Inicialmente, a temperatura é elevada, em modo manual, de modo que *PV* fique aproximadamente igual a 39890 DAC. Em seguida, o modo de operação é alterado para automático, o valor do *SP* é modificado para 39890 DAC e se espera a temperatura atingir o *SP*. No instante t=3620 s, o valor do *SP* é modificado para 43890 DAC. Aproximadamente no instante t=6800 s, para testar a rejeição à perturbação, a frequência aplicada ao motor é alterada de 20 Hz para 30 Hz, ou seja, a vazão de ar aumenta. A Figura 4.13 mostra o resultado obtido durante o intervalo de tempo de 3244 a 8956 segundos. Nessa figura é possível observar que não ocorre overshoot, não há erro em estado estacionário e o controlador consegue rejeitar a perturbação. Além disso, a temperatura atinge o *SP* em menos de 900 segundos, então, considerando que esse tempo equivale a 4 constantes de tempo, a constante de tempo é menor que 250 segundos. Portanto, todos os objetivos do projeto do controlador foram atendidos.

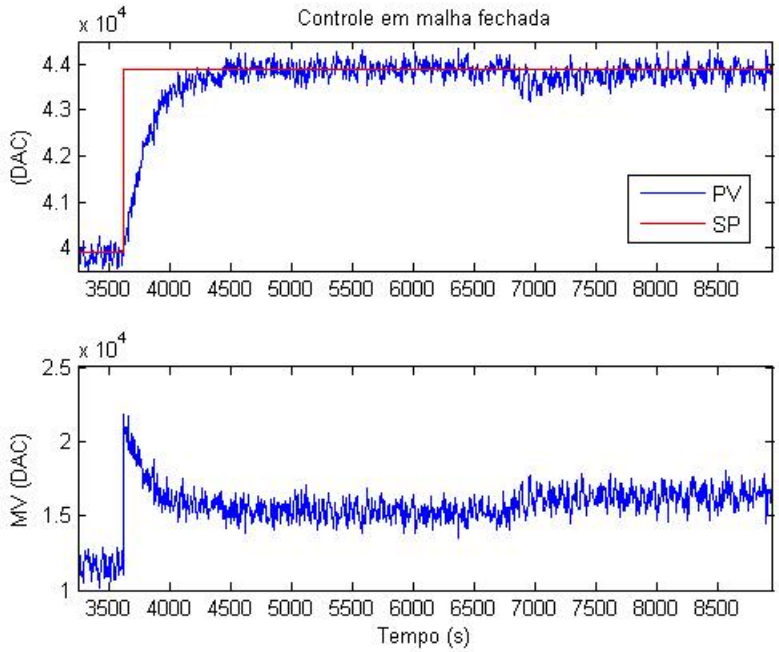


Figura 4.13: Resultado do controle em malha fechada.

Para validar o controlador implementado na plataforma, o comportamento simulado da planta em malha fechada, que é mostrado na Figura 4.12, é comparado com o comportamento real obtido, com o uso do controlador PID desenvolvido, que é mostrado na Figura 4.13. A Figura 4.14 mostra a comparação dos valores de *PV* e *MV* para uma mudança no valor do *SP* de 39890 DAC para 43890 DAC. A perturbação não foi simulada, por isso ela não está na comparação. O comportamento simulado de *PV*, mostrado em linha pontilhada, segue a mesma tendência do comportamento da planta real, mostrado em linha contínua. O comportamento simulado de *MV* também segue a mesma tendência da planta real, porém possui um offset

de aproximadamente 3700 DAC. Essa diferença, provavelmente, ocorre porque a temperatura ambiente no dia em que o controle foi realizado era maior que a temperatura ambiente no dia em que foi feito a coleta de dados para a modelagem. Por isso, a potência necessária para atingir o *SP* é menor que o previsto na simulação.

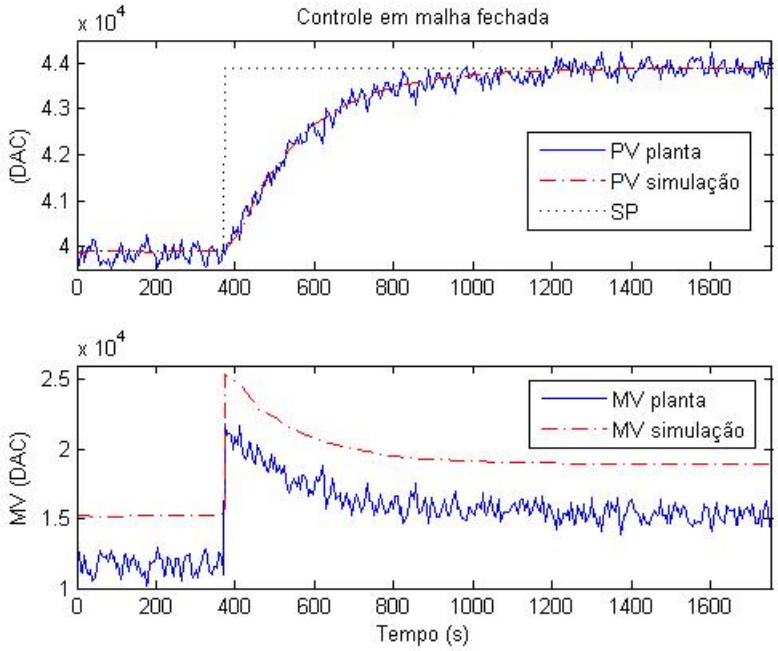


Figura 4.14: Comparação dos resultados reais e os simulados.

A Figura 4.15 mostra os valores de *PV* e *MV* para a mesma mudança no valor do *SP*, porém subtraindo 3700 DAC dos valores de *MV* simulados. Pode-se perceber que, eliminado-se o *offset*, o comportamento simulado de *MV* é mais próximo do real e segue a mesma tendência.

É importante ressaltar que a simulação é feita no domínio de Laplace e o controlador implementado está no domínio *z*. Por isso, obter um comportamento real parecido com o simulado, além de mostrar que o modelo obtido é satisfatório, também mostra que a implementação do controlador PID foi feita de forma adequada.

A Figura 4.16a e a Figura 4.16b mostram que a ação de controle é limitada e que não ocorre wind-up. Na Figura 4.16a, até o instante t=120 s, o *SP* é menor que a temperatura ambiente, por isso *MV* satura em zero. No instante t=120 s, o *SP* é alterado para um valor acima da temperatura ambiente, pode-se perceber que não ocorre wind-up, pois *MV* não permanece saturado. Na Figura 4.16b, no instante t=0s, o *SP* é alterado para um valor muito acima da temperatura ambiente, por isso *MV* satura em 32755. No instante t=6680s o *SP* é reduzido, pode-se perceber que não ocorre wind-up, pois *MV* não permanece saturado.

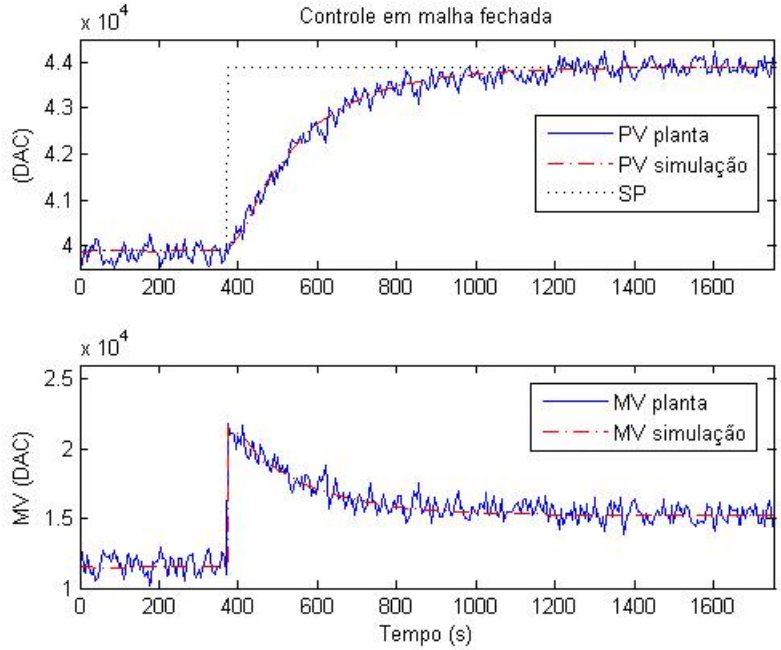
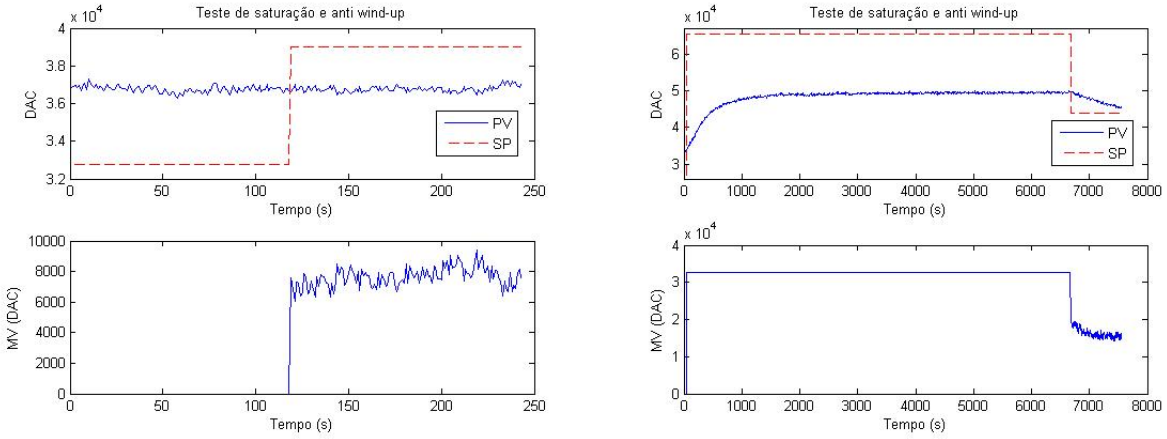


Figura 4.15: Comparação dos resultados com o *MV* simulado modificado.



(a) (b)

Figura 4.16: Resultado dos testes de saturação e de anti wind-up.

O último teste é relativo a mudança suave de manual para automático. Nesse teste, o modo de operação inicial é manual e o valor de *MV* é 9825. Depois que a temperatura estabiliza (t=1260s), o modo de operação é alterado para automático e o *SP* escolhido é 39890. A Figura 4.17 mostra os valores de *PV* ao longo do teste. Mostra também que o valor de *MV*, após a transição para automático, permanece próximo ao valor anterior. É preciso observar que a variação brusca em t=1265 ocorre devido ao erro de 1780 presente nesse instante que, multiplicado pelo ganho de 2,5034, resulta em um aumento repentino de 4456 DAC em *MV*.

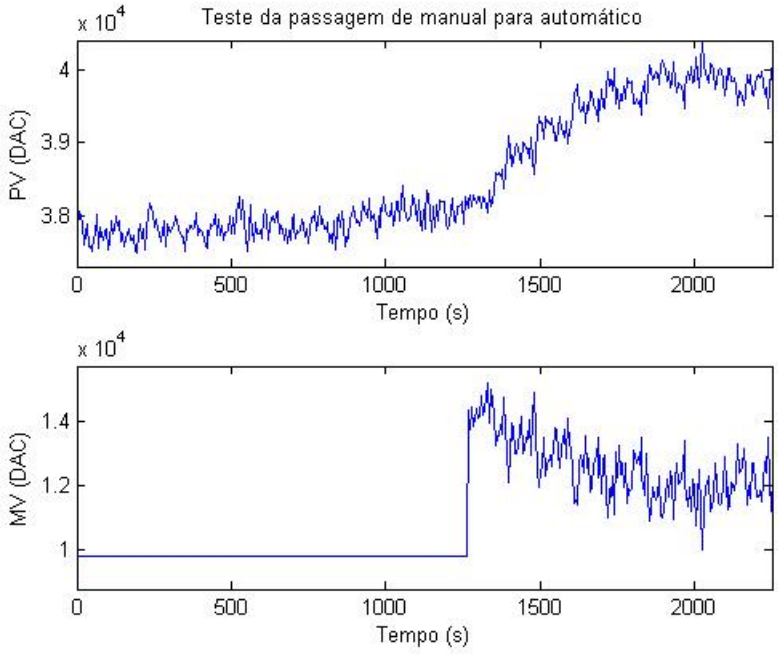


Figura 4.17: Teste de mudança suave de manual para automático.

5 *Conclusão*

Neste capítulo são apresentadas as conclusões obtidas ao longo deste projeto e sugestões para trabalhos futuros.

5.1 Conclusões

O uso de softwares livres permite que a plataforma desenvolvida neste trabalho possa ser reproduzida com custos reduzidos. Além disso, mesmo que os códigos dos softwares utilizados, como o driver da placa de aquisição de dados, não tenham sido modificados, usar softwares de código aberto possui a vantagem de permitir que os detalhes de implementação sejam melhor estudados e entendidos de forma que esses conhecimentos não fiquem restritos a ambientes corporativos ou proprietários.

A placa de aquisição de dados é instalada e integrada à plataforma de forma bem sucedida com o uso do driver de comunicação Comedi. Na seção 2.2 é mostrado que tanto a curva de conversão de analógico para digital quanto a curva de conversão de digital para analógico são lineares e apresentam a conversão entre tensão e valores DAC conforme esperado.

Neste trabalho é desenvolvido uma plataforma de controle que permite a implementação de diversos algoritmos de controle, porém apenas o controlador PID industrial é implementado. Como demonstrado na seção 2.3, o controlador PID implementado apresenta uma variação em T. Por isso, cada aplicação deve ser analisada individualmente para que seja possível decidir se a aplicação tolera a variação que a plataforma apresenta.

A plataforma é utilizada para modelar a malha de temperatura da planta piloto SCVT. O modelo obtido é utilizado para projetar um controlador PID por meio do método do Modelo Interno conforme descrito em [Seborg et al., 1989]. O comportamento da planta piloto em malha fechada é simulado e se obtêm um resultado próximo do comportamento real da planta. Portanto, a plataforma é utilizada com sucesso para modelar e controlar a malha de temperatura da planta piloto.

Portanto, a integração dos componentes de software e hardware é realizada de forma a permitir que a plataforma seja utilizada para fins didáticos nas disciplinas de Laboratório de Controle e Automação I e II.

5.2 Sugestões para trabalhos futuros

A plataforma desenvolvida permite implementar diversos controladores. Por isso, sugere-se que outros controladores sejam implementados de forma a ampliar as opções de uso da plataforma de controle.

O filtro passa-baixas pode ser transformado em um filtro anti-aliasing, para isso a frequência de amostragem do filtro precisa ser maior que a do algoritmo de controle. Sugere-se transformar o filtro passa-baixas digital em um filtro anti-aliasing.

Durante os testes com a planta, percebe-se que a interface não está completa, pois não é possível visualizar os valores correntes de MV e PV. Por isso, sugere-se acrescentar essas informações à interface.

O desempenho do controlador pode ser um pouco melhor se ele for transformado em um módulo do Linux. Por isso, sugere-se transformar o controlador PID em um módulo caso seja necessário um melhor desempenho.

APÊNDICE A – Tutorial: Instalando o RTAI

Há pouca documentação em português sobre como instalar o RTAI e o Comedi, e a maioria dos documentos em inglês pressupõe que se tenha conhecimento prévio no processo de compilação do kernel. Este documento se propõe a cobrir o processo de configuração, compilação e instalação do kernel, do RTAI e do Comedi.

A.1 Introdução

Este tutorial é feito para que todos os passos para instalar o RTAI e o Comedi estejam registrados e possa, no futuro, ser usado para configurar outros computadores. Os trabalhos [Bensen, 2007], [Deen, 2007] e [Monteiro, 2008] são usados neste tutorial. Porém, algumas modificações são feitas, pois as versões do Ubuntu, do kernel e do RTAI são diferentes das citadas nesses trabalhos.

A.2 Requisitos

Antes de prosseguir com o tutorial, é preciso garantir alguns requisitos:

- Um computador PC;
- Linux instalado - Para confecção deste tutorial, a distribuição Ubuntu 8.10 é usada. Esta distribuição provê facilidades (apt-get) para instalar pacotes externos;
- gcc e g++ versão 3.4 - Pode-se conferir a versão do gcc e g++ no computador executando:

  $gcc –version

  $g++ –version

  Para instalar os compiladores no Ubuntu, execute:

  $sudo apt-get install gcc -3.4

$sudo apt-get install g++ -3.4

Para consultar informações para a instalação acesse http://gcc.gnu.org/install/.

- make - Preferencialmente uma versão acima da 3.80. Make é a ferramenta responsável por executar os scripts de compilação (”Makefile”). Normalmente, esta ferramenta vem instalada por padrão em todas as distribuições. Em todo caso, para instalar, execute:

  $sudo apt-get install make

- module-init-tools - Versão igual ou superior a 3.0. Ferramentas necessárias para carregar módulos, entre outras utilidades. Para instalar, execute:

  $sudo apt-get install module-init-tools

- libncurses5-dev - Menu de configuração do kernel. Esse pacote instala as bibliotecas necessárias para rodar o menu de configuração do kernel.

  $apt-get install libncurses5-dev

- Também foi necessário instalar: $apt-get install patch

Após instalar todos os pacotes necessários, é preciso selecionar as versões do kernel e do RTAI para prosseguir com a instalação.

A.3 Versões do RTAI

No momento em que este tutorial está sendo escrito, a última versão estável do RTAI é a 3.6.2. Ela pode ser encontrada na página do RTAI: www.rtai.org.

A.4 Baixando o RTAI

Baixe o código do RTAI 3.6.2.tar.bz2 no site www.rtai.org e o descompacte. O diretório padrão para deixar o código-fonte do RTAI e do kernel é o /usr/src, mas é necessária a permissão de super-usuário para alterar a pasta.

A.5 Baixando os sources do Linux

Normalmente, o kernel específico de uma distribuição é modificado para melhorar alguns aspectos. Porém, muitas vezes, essas mudanças são incompatíveis com o RTAI. Por isso, é

necessário baixar um kernel sem modificações, também conhecido como ”kernel vanilla”. Normalmente, o source do linux também é baixado no diretório /usr/src. Para acessar esse diretório, digite:

$cd /usr/src

O primeiro passo aqui é decidir qual versão do kernel baixar. Cada versão do RTAI tem suporte para algumas versões do kernel. Para verificar quais são as versões suportadas, veja quais *patches* estão disponíveis na pasta <caminho para rtai>rtai-3.6.2/base/arch/i386/patches. Nessa pasta há arquivos com nomes parecidos com esses:

hal-linux-2.4.32-i386-1.2-07.patch

hal-linux-2.4.33-i386-1.3-03.patch

hal-linux-2.4.34-i386-1.3-03.patch

hal-linux-2.6.15-i386-1.3-07.patch

hal-linux-2.6.16-i386-1.3-08.patch

hal-linux-2.6.17-i386-1.5-02.patch

hal-linux-2.6.23-i386-1.12-03.patch

README

O kernel mais novo suportado é o 2.6.23, por isso ele é usado como exemplo. Na página http://www.kernel.org/pub/linux/kernel/v2.6/ encontra-se os sources de várias versões do kernel. Clique no link do kernel desejado, linux-2.6.23.tar.gz nesse caso.

Após baixar o source do kernel, descompacte o mesmo. Agora, haverá uma pasta correspondente à versão do kernel baixado.

OBS: A numeração do kernel pode ter um parâmetro adicional, totalizando assim 4 números (e.g. 2.6.23.11), porém os *patches* do RTAI se destinam apenas às versões com 3 números (vanilla). Provavelmente, o *patch* funcionará com os kernels com a numeração extra, porém estes kernels não são completamente testados e suportados.

A.6 Aplicando o patch do RTAI

Aplicar o patch no kernel é muito simples. Considerando que a estrutura de diretórios seja a seguinte:

/usr/src

—-/linux-2.6.23

—-/rtai-3.6.2

Entre na pasta onde o kernel foi descompactado:

$cd /usr/src/linux-2.6.23

Agora aplique o patch da seguinte forma:

$patch -p1 -b ../rtai-3.6.2/base/arch/i386/patches/hal-linux-2.6.23-i386-1.12-03.patch

Pronto, o patch do RTAI foi aplicado ao kernel.

A.7 Configurando o Kernel

Não entra no escopo deste tutorial verificar todas as opções de configuração do kernel, portanto este tópico é apenas superficial.

A configuração do kernel é a parte mais difícil da instalação. Uma boa idéia é copiar o arquivo de configuração do kernel que está sendo usado, afinal de contas ele está funcionando.

Para isso vá ao diretório /boot e copie um arquivo com nome parecido com config-2.6.27-11-generic, onde 2.6.27 é a versão do kernel. Copie esse arquivo para a pasta onde o novo kernel está (.../linux-2.6.23). Renomeie ele para .config.

Agora que se possui um arquivo de configuração consistente, pode-se alterar o que for necessário no kernel. Se houver mais de um compilador instalado na distribuição utilizada, é preciso especificar qual deles deve ser usado. Para isso, basta adicionar alguns parâmetros quando for configurar o kernel:

$make menuconfig CC=/usr/bin/gcc-3.4 CXX=/usr/bin/g++-3.4

Lembre-se de adaptar os caminhos ao seu caso. Depois de o menu de configuração do kernel ser exibido, é preciso carregar o arquivo de configuração que foi copiado previamente. Para isso, entre em ”Load Alternate Configuration File”, digite ”.config”e precione Enter.

Algumas opções do kernel devem ser verificadas antes da compilação ser iniciada.

- Processor type and features ->Processor family: Core 2/newer Xeon.
- Processor type and features ->Timer frequency: 1000Hz.

- Processor type and features ->Preemption Model :Preemptible Kernel (Low-Latency Desktop).
- Processor type and features ->Use register arguments: esta opção deve ser desabilitada.
- Power management options (ACPI, APM) ->Power Management support: esta opção deve ser desabiltada, pois ela aumenta a latência do sistema.
- Power management options (ACPI, APM) ->CPU Frequency scaling: esta opção deve ser desabilitada.
- Enable loadable module support: esta opção deve ser habilitada (normalmente está por padrão). Com ela, é possível carregar os módulos do RTAI mais tarde.
- Loadable module support ->Module versioning support: esta opção não deve ser habilitada, pois segundo o FAQ de instalação do RTAI, ela causa alguns problemas para o mesmo.
- Kernel hacking ->Compile the kernel with frame pointers: esta opção deve ser desabilitada.

Para fazer a nova configuração do kernel ser a mais parecida possível com a que está em uso, faça:

$sudo cp /boot/config-‘uname -r‘ ./.config

$sudo make oldconfig

Aperte Enter para todos os prompts que surgirem. Agora deve-se configurar o novo kernel:

$sudo make menuconfig

Para que o novo kernel tenha um nome mais significativo, um sufixo pode ser adicionado ao nome, por exemplo -rtai-3.6.2. Para fazer isso vá em: General setup >Local version - append to kernel release = -rtai-3.6.2

O processo de compilação do kernel é algo que exige tentativas e erros para chegar a um bom resultado, e quanto mais se conhecer o hardware, mais o kernel poderá ser otimizado. Por exemplo, se a placa de rede é conhecida, escolha a opção correspondente no kernel para ela (de preferência estática, e não como módulo), e desabilite o suporte a todas as outras placas. Desta forma, o tempo de compilação será reduzido, pois menos módulos serão compilados, e o kernel ficará mais rápido, pois o suporte à rede é interno, não precisando carregar módulos e fazer chamadas para o mesmo.

A.8 Compilando o Kernel

No Ubuntu, o melhor modo de instalar um kernel customizado é criar um arquivo ".deb"de instalação do kernel. Para isso, é preciso ter os seguintes pacotes instalados:

$apt-get install kernel-package fakeroot

Altere o flag CFLAGS_KERNEL na linha 299 do make file para:

$CFLAGS_KERNEL = -fno-tree-scev-cprop

Execute os seguintes comandos na pasta onde está o kernel para gerar os pacotes de instalação:

$make-kpkg clean

```
$ fakeroot make-kpkg --initrd --append-to-version=-rtai
\kernel_image kernel\_headers
```

Após o parâmetro *–append-to-version*= pode-se colocar qualquer nome que se achar conveniente.

Dependendo da configuração do kernel, e das configurações do seu computador, a compilação pode demorar algumas horas, portanto seja paciente. Assim que a compilação terminar, haverá dois arquivos ".deb"na pasta /usr/src. Agora é possível instalar o kernel com o seguinte comando:

$dpkg -i *.deb

Agora temos que criar o initrd, o responsável por carregar os módulos na inicialização do sistema. O utilitário usado agora é o mkinitramfs:

$mkinitramfs -o /boot/initrd.img-2.6.23 2.6.23

Pronto. Se tudo deu certo, quando o computador reiniciar, haverá uma nova opção no grub. É só escolher ela.

Agora edite o gerenciador de boot, nesse caso o GRUB. O arquivo de configuração do GRUB é /boot/grub/menu.lst. Adicione uma nova opção ao menu, que deve ser algo parecido com o trecho a seguir, variando de acordo com as configurações da máquina (e.g. partições):

```
title Ubuntu 8.10, kernel 2.6.23-RTAI-3.6.2
uuid da6abf6f-c132-4bd4-8985-01edcd761a0e
kernel /boot/vmlinuz-2.6.23 root=UUID=da6abf6f-c132-4bd4-8985-01edcd761a0e ro
```

```
locale=pt_BR quiet splash
initrd /boot/initrd.img-2.6.23
```

Lembre-se, isto é só um exemplo, e pode ter variações dependendo da distribuição. Procure analisar as opções já existentes no arquivo e se basear nelas para criar a nova. Feito isso, reinicie o computador.

A.9 Configurando o RTAI

Se tudo deu certo, o novo kernel deve estar funcionando. Agora é preciso configurar o RTAI. Depois de entrar na pasta onde o RTAI foi descompactado, o primeiro passo é criar uma nova pasta onde ele será compilado.

$cd /usr/src/rtai-3.5

$mkdir build

$cd build

Agora, podemos configurar o RTAI propriamente:

$make -f ../makefile menuconfig

As seguintes opções devem ser verificadas:

- General ->Instalation Directory: onde o RTAI será instalado. O padrão é /usr/realtime
- General ->Linux Build Tree: mude esta opção para algo como /usr/src/linux-headers-2.6.23-rtai.
- Machine (x86) ->Number of CPUs (SMP-only) = 2 (2 processadores)
- Base system ->Other features ->ativar (*):Mathfuns support in kernel

Agora salve e saia da configuração.

A.10 Instalando o RTAI

A instalação do RTAI é extremamente simples. Primeiro se compila o RTAI:

$make

Após o processo de compilação, se instala o RTAI com o comando:

$make install

Se ocorrer o erro:

```
../../../base/include/rtai\_posix.h:1212: /usr/include/bits/fcntl2.h:51:
error: call to '__open_missing_mode' declared with attribute error: open
with O\_CREAT in second argument needs 3 arguments
```

vá na linha 1212 do arquivo /base/include/rtai_posix.h:1212 e substitua o código

```
fd = open(name, O\_CREAT | O\_WRONLY);
```

por :

```
fd = open(name, O\_CREAT | O\_WRONLY, S\_IRWXU | S\_IRWXG | S\_IRWXO);
```

Pronto. Se tudo isso ocorreu sem problemas, reinicie o computador, escolha o kernel do RTAI, e use o sistema RTAI.

A.11 Testando a Instalação

A distribuição do RTAI inclui três programas de teste, cada um com uma versão para ser executada no modo kernel e outra para ser utilizada no modo usuário. Um dos programas testa a latência, outro a preempção e o terceiro, o chaveamento entre tarefas. Para utilizá-lo, basta entrar no diretório correspondente e executar o comando ./run.

Estes testes possibilitam identificar se o RTAI está instalado e funcionando corretamente. Como o principal objetivo é identificar a correta instalação do RTAI, eles foram executados sem carga na CPU.

Nos testes de latência, o *overrun* deve ser igual a zero o tempo todo. Nos testes de preempção, os valores de *jitter* devem ficar dentro de limites aceitáveis para a aplicação. No teste de chaveamento, o tempo de chaveamento entre tarefas deve ficar dentro de limites aceitáveis para a aplicação. Como o teste de latência é o mais importante, apenas os resultados dele serão mostrados.

A.11.1 Testes de latência

Segundo [Regnier et al.,], nas plataformas de tempo real, eventos de temporizadores ou de hardware são utilizados para disparar tarefas. Uma tarefa periódica fica suspensa a espera de um evento. Quando o evento ocorre, a requisição de interrupção associada aciona o tratador correspondente que, por sua vez, acorda a tarefa. O intervalo de tempo entre os instantes de ocorrência do evento e o início da execução da tarefa associada é chamado de latência. Caso um evento ocorra uma segunda vez antes de a tarefa ter recebido o primeiro evento, ocorre o overrun.

O RTAI subtrai dos valores um tempo que ele julga ser o utilizado para manipular os dados. Os valores negativos para latência mínima significam que o RTAI superestimou o tempo necessário para isso.

Por padrão, as aplicações de teste ficam em /usr/realtime/testsuite. Vamos executar o teste de latência em modo usuário e em modo kernel, para verificarmos se o sistema funciona como deveria. Para testar em modo usuário, execute:

$cd /usr/realtime/testsuite/user/latency

$./run

O teste mostrado a seguir foi feito no espaço usuário e mostra, entre outros, a latência mínima, média e máxima.

```
 ## RTAI latency calibration tool ##
# period = 100000 (ns)
# average time = 1 (s)
# use the FPU
# start the timer
# timer_mode is oneshot

RTAI Testsuite - USER latency (all data in nanoseconds)
2009/04/2 15:01:34
RTH|    lat min|    ovl min|    lat avg|    lat max|    ovl max|   overruns
RTD|       -642|       -694|       3949|      18472|      18472|          0
RTD|       -697|       -697|       4018|      19087|      19087|          0
RTD|       -558|       -697|       3991|      17244|      19087|          0
RTD|       -667|       -697|       4048|      17843|      19087|          0
```

```
RTD|       -689|       -697|       4031|      17968|      19087|          0
RTD|       -694|       -697|       4054|      17321|      19087|          0
RTD|       -672|       -697|       4039|      17383|      19087|          0
RTD|       -697|       -697|       3967|      17972|      19087|          0
2009/04/2 15:01:54
>>> S = 98.696, EXECTIME = 0.0298652
```

Agora, testaremos em modo kernel:

$cd /usr/realtime/testsuite/kern/latency

$./run

O teste mostrado a seguir foi feito no espaço kernel.

```
## RTAI latency calibration tool ##
# period = 100000 (ns)
# avrgtime = 1 (s)
# do not use the FPU
# start the timer
# timer_mode is oneshot

RTAI Testsuite - KERNEL latency (all data in nanoseconds)
RTH|    lat min|    ovl min|    lat avg|    lat max|    ovl max|   overruns
RTD|      -1348|      -1487|       2914|      16581|      16581|          0
RTD|      -1347|      -1487|       2792|      16657|      16657|          0
RTD|      -1382|      -1487|       2904|      16214|      16657|          0
RTD|      -1345|      -1487|       2796|      16350|      16657|          0
RTD|      -1382|      -1487|       2882|      16554|      16657|          0
RTD|      -1355|      -1487|       2816|      16544|      16657|          0
RTD|      -1366|      -1487|       2858|      16151|      16657|          0
RTD|      -1343|      -1487|       2832|      16209|      16657|          0
```

Quando o modo kernel for testado, pode aparecer uma mensagem de erro, informando que um dispositivo não foi encontrado, parecida com:

```
Error opening /dev/rtf3
```

Para resolver esse problema, copie e cole o código abaixo em um arquivo, conceda permissão de execução, e rode o script:

```
\textit{\#!/bin/bash
mknod -m 666 /dev/rtai_shm c 10 254
for n in 'seq 0 31'
do
f=/dev/rtf\textdollar n
mknod -m 666 \textdollar f c 150 \textdollar n
done}
```

A.12 Instalando o Comedi

Este projeto usou o comedi-0.7.76. Primeiramente, instale o RTAI como descrito acima. Faça o download do comedi-0.7.76. Descompacte comedi.tar.gz

$sudo -s

$cp -r /.../comedi /usr/src/

$cd /usr/src/comedi-0.7.76

Para fazer o arquivo de configuração:

$./autogen.sh

Para o próximo comando, use o diretório do código fonte do kernel e o diretório do código binário do RTAI

$./configure –with-linuxdir=/usr/src/linux-headers-2.6.23-rtai –with-rtaidir=/usr/realtime

$make && make install && make dev

Faça o download do comedilib-0.8.1.Depois descompacte o arquivo comedilib.tar.gz

$cp -r /.../comedilib /usr/src/

$cd /usr/src/comedilib-0.8.1

$sudo -s

Para fazer o arquivo de configuração:

$./autogen.sh

$./configure

$make

$make install

$depmod -a

Recompile e reinstale o RTAI com suporte ao Comedi. Para isso, vá ao diretório do rtai e digite:

$make menuconfig

As seguintes opções devem ser verificadas:

- Add-on ->Comedi support over LXRT: ativar (*)
- Add-on ->COMEDI installation directory :/usr/src/comedi-0.7.76 - local onde o Comedi será instalado. O instalador vai adicionar /include/linux/comedilib.h ao caminho informado

Digite:

$make && make install

A.13 Testando o Comedi

Agora que o Comedi consegue se comunicar com a placa, deve-se comunicar com o Comedi:

$cat /proc/comedi

Para obter informações da placa digite:

$cd /usr/src/comedilib-0.8.1/demo

$sudo ./board_info

Isso mostra, por exemplo, que o subdevice 0 é para entrada analógica

Para ligar o driver à /dev/comedi0

$comedi_config /dev/comedi0 ni_pcimio

Se aparecer uma mensagem de erro dizendo que /dev/comedi0 não existe, vá ao diretório do comedi digite:

$make dev

$LD_LIBRARY_PATH=/usr/local/lib/:/usr/src/comedilib-0.8.1/lib/.libs/:$LD_LIB RARY_PATH

$echo ”/usr/local/lib”>>/etc/ld.so.conf /sbin/ldconfig

Toda vez que o linux reiniciar, deve-se ligar o driver à /dev/comedi0 e ler os módulos do RTAI e Comedi. Por isso use o seguinte script:

```
#!/bin/bash
echo "Criando Dev"
cd /usr/src/comedi-0.7.76/
make dev
echo "Load Comedi drivers"
if [ $(id -u) = "0" ]; then
        echo "loading rtai modules"
        /sbin/insmod /usr/realtime/modules/rtai_hal.ko
        /sbin/insmod /usr/realtime/modules/rtai_lxrt.ko
        /sbin/insmod /usr/realtime/modules/rtai_sem.ko
        /sbin/insmod /usr/realtime/modules/rtai_mbx.ko
        /sbin/insmod /usr/realtime/modules/rtai_fifos.ko
        /sbin/insmod /usr/realtime/modules/rtai_tbx.ko
        /sbin/insmod /usr/realtime/modules/rtai_bits.ko
        /sbin/insmod /usr/realtime/modules/rtai_mq.ko
        /sbin/insmod /usr/realtime/modules/rtai_shm.ko
        /sbin/insmod /usr/realtime/modules/rtai_tasklets.ko
#       /sbin/insmod /usr/realtime/modules/rtai_math.ko
        /sbin/insmod /usr/realtime/modules/rtai_netrpc.ko
#       /sbin/insmod /usr/realtime/modules/rtai_rtdm.ko
        /sbin/insmod /usr/realtime/modules/rtai_signal.ko
        # then load the comedi driver and assign driver to
        # /dev/comedi0
        echo "assign driver to /dev/comedi0"
        /sbin/modprobe ni_pcimio
#       /sbin/insmod /usr/src/comedi/comedi/drivers/ni_pcimio.ko
        /usr/local/sbin/comedi_config /dev/comedi0 ni_pcimio
else
        echo "Only superuser can load the Comedi driver."
fi
```

Teste o Comedi:

$cd /usr/src/comedilib-0.8.1/testing

$make

$./comedi_test

Leia alguns dados:

$cd /usr/src/comedilib-0.8.1/demo

$./cmd -n 1 -N 200 -p

Se aparecer a mensagem:

./rtai_user_mode_task: error while loading shared libraries: liblxrt.so.1: cannot open shared object file: No such file or directory

digite:

$export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/realtime/lib

Referências Bibliográficas

[NI, 2008] (2008). *DAQ Getting Started Guide.* National Instruments.

[A.Barbalacel et al., 2008] A.Barbalacel, Luchetta, A., Manduchi, G., Soppelsa, M. M. A., and Taliercio, C. (2008). Performance comparison of vxworks, linux, rtai and xenomai in a hard real-time application. *IEEE*, 55:435–439. Issue 1 Part 1.

[André de Almeida, 2006] André de Almeida, R. (2006). Controle do pêndulo utilizando plataforma linux de tempo real. Monografia final de curso de Engenharia de Controle e Automação - UFMG.

[Bennett, 1988] Bennett, S. (1988). *Real-Time Computer Control: An Introduction*. Prentice Hall.

[Bensen, 2007] Bensen, R. (2007). Tutorial: Instalando rtai. Obtido em http://pet.inf.ufsc.br/files/tutorial_rtai.pdf, acessado em 07/07/2009.

[Clarke, 1984] Clarke, D. W. (1984). Pid algorithms and their computer implementation. *Trans Inst*, 6(6):305–316.

[C.Schmidt et al., 2002] C.Schmidt, D., Deshpande, M., and O'Ryan, C. (2002). Operating system performance in support of real-time middleware. *Workshop on Object-Oriented Real-Time Dependable Systems*, pages 199–206.

[Deen, 2007] Deen, B. (2007). Ct – i/o linkdriver with comedi. Obtido em http://www.ce.utwente.nl/rtweb/publications/index.php?body=student_titles&Year=2007. Acessado em 09/07/2009.

[Doebelin, 1990] Doebelin, E. O. (1990). *Measurement Systems: Application and Design*. McGraw-Hill, 4 edition.

[Dorf and Bishop, 2001] Dorf, R. C. and Bishop, R. H. (2001). *Sistemas de Controle Modernos*. LTC Editora.

[Dozio and Mantegazza, 2003] Dozio, L. and Mantegazza, P. (2003). Linux real time application interface (rtai) in low cost high performance motion control.

[Flávio Mota, 2006] Flávio Mota, H. d. D. (2006). Desenvolvimento de um gerenciador de dispositivo gps em sistema linux padrão e de tempo real. Monografia final de curso de Engenharia de Controle e Automação - UFMG.

[Instruments, 2007] Instruments, N. (2007). Ni 622x specifications. Obtido em http://sine.ni.com/nips/cds/view/p/lang/en/nid/14132, acessado em 21/05/2009.

[I.So et al., 2007] I.So, Siddons, D., Caliebe, W., and Khalid, S. (2007). Hard real-time quick exafs data acquisition with all open source software on a commodity personal computer. *Nuclear Instruments and Methods in Physics Research A*, A 582:190–192.

[Lineo, 1993] Lineo, I. (1993). *DIAPM RTAI Programming Guide 1.0.* www.cs.ru.nl/lab/rtai/.

[Magalhães, 2008] Magalhães, E. (2008). Implementação de estratégias de controle avançado a uma planta piloto de controle de vazão e temperatura de ar. Master's thesis, Universidade Federal de Minas Gerais.

[Martínez-Ortiz, 2005] Martínez-Ortiz, F. (2005). On the use of a real time application interface under linux in the electrochemistry laboratory. aplication to chronopotentiometry. *Journal of Electroanalytical Chemistry*, 574:239–250.

[Meghwani and Kulkarni, 2008] Meghwani, A. and Kulkarni, A. M. (2008). Development of a laboratory model of sssc using rtai on linux plataform. *Sãdhanã*, pages 643–661. Part 5.

[Monteiro, 2008] Monteiro, J. (2008). Rtai installation complete guide. Obtido em https://www.rtai.org/RTAICONTRIB. Acessado em 09/07/2009.

[M.Sacha, 1995] M.Sacha, K. (1995). Measuring the real-time operating system performance. *Seventh Euromicro Workshop on Real-Time Systems*, pages 34–40.

[Persechini, 2008] Persechini, P. M. A. M. (2008). Laboratório de controle e automação ii - implementação de controladores digitais. Roteiro de aula.

[Regnier et al.,] Regnier, P., Lima, G., and Barreto, L. Avaliação do determinismo temporal no tratamento de interrupções em plataformas de tempo real linux. www.lisha.ufsc.br/wso/wso2008/papers/.

[Seborg et al., 1989] Seborg, D. E., Edgar, T. F., and Mellichamp, D. A. (1989). *Process Dynamics and Control*. John Wiley Sons.

[Seixas Filho and Szuster, 1993] Seixas Filho, C. and Szuster, M. (1993). *Programação concorrente em ambiente Windows- Uma visão de automação*. Editora da UFMG.

[Terry Bollinger, 2003] Terry Bollinger, T. M. C. (2003). Use of free and open-source software (foss) in the u.s. department of defence. Version 1.2.04. http://cio-nii.defense.gov/sites/oss/.

[Vitale et al., 2004] Vitale, V., Centioli, C., F.Iannone, Mazza, G., Panella, M., Pangione, L., Podda, S., and Zaccariah, L. (2004). Real time linux operating system for plasma control on ftu-implementation advantages and first experimental results. *Fusion Engineering and Designs*, 71:71–76.

[www.comedi.org/, 2009] www.comedi.org/ (2009). Acessado em 18/04/2009.

[www.distrowatch.com/, 2009] www.distrowatch.com/ (2009). Acessado em 05/05/2009.

[www.gnome.org, 2009] www.gnome.org (2009). Acessado em 26/08/2009.

[www.rtai.org/, 2006] www.rtai.org/ (2006). Rtai: a beginner's guide. Acessado em 20/05/2009.

[www.rtai.org/, 2009] www.rtai.org/ (2009). Acessado em 18/04/2009.

[www.ubuntu br.org, 2009] www.ubuntu br.org (2009). Acessado em 18/04/2009.