Marcelo Pereira Nakamura
Pedro Henrique Wehbe Castro Azambuja

Supervisores *Fuzzy* para Controladores PID Aplicação: Controle de Temperatura de um Ambiente Simulado de Cabine de Aeronave Comercial (*Mock-up*)

São Paulo

2013

Marcelo Pereira Nakamura
Pedro Henrique Wehbe Castro Azambuja

Supervisores *Fuzzy* para Controladores PID Aplicação: Controle de Temperatura de um Ambiente Simulado de Cabine de Aeronave Comercial (*Mock-up*)

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Mecatrônico.

Área de Concentração:
Engenharia Mecatrônica

Orientador:
Prof. Dr. Maurício Silva Ferreira

São Paulo
2013

Marcelo Pereira Nakamura
Pedro Henrique Wehbe Castro Azambuja
Supervisores *Fuzzy* para Controladores PID
Aplicação: Controle de Temperatura de um Ambiente Simulado de Cabine de Aeronave Comercial (*Mock-up*)/ Marcelo Pereira Nakamura
Pedro Henrique Wehbe Castro Azambuja. – São Paulo, 2013-
133 p. : il. (algumas color.) ; 30 cm.

Orientador:
Prof. Dr. Maurício Silva Ferreira

Trabalho de Formatura – , 2013.

1. Supervisor Fuzzy. 2. Controlador PID. 3. Controle de Temperatura. 4. Aeronave Comercial. 5. Mock-up. I. Prof. Dr. Maurício Silva Ferreira. II. Escola Politécnica da Universidade de São Paulo. III. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos. IV. Supervisores Fuzzy para Controladores PID

CDU 02:141:005.7

Marcelo Pereira Nakamura
Pedro Henrique Wehbe Castro Azambuja

Supervisores *Fuzzy* para Controladores PID Aplicação: Controle de Temperatura de um Ambiente Simulado de Cabine de Aeronave Comercial (*Mock-up*)

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Mecatrônico.

São Paulo, 24 de novembro de 2013:

Prof. Dr. Maurício Silva Ferreira
Orientador

Professor
Convidado 1

Professor
Convidado 2

São Paulo
2013

Este trabalho é dedicado às nossas famílias.

Agradecimentos

Os autores desejam expressar seus mais sinceros agradecimentos:

Ao Prof. Dr. Maurício Silva Ferreira, pela sua disposição em educar e pela orientação abrangente, essencial e dedicada;

Ao Prof. Dr. Fabrício Junqueira, cujas sugestões foram de fundamental importância para a realização do projeto;

Ao pessoal do experimento *Mock-up* da Embraer, pelos inúmeros conhecimentos práticos transmitidos e pela paciência em responder nossas diversas dúvidas;

Ao Prof. Dr. Lucas Moscato e ao Prof. Dr. Thiago Martins, pela paciência e ajuda ao longo do ano.

"The worthwhile problems are the ones
you can really solve or help solve, the ones
you can really contribute something to.
No problem is too small or too trivial
if we can really do something about it."
(Richard P. Feynman)

Resumo

O objetivo do presente trabalho é obter uma solução de controle que diminua o tempo de subida e o erro em regime das respostas em malha fechada para as temperaturas de interesse do ar em um ambiente simulado de cabine de aeronave comercial (*Mock-up*). Para isso foi desenvolvido um controle adaptativo que combina os controladores PID convencionais instalados na planta com supervisores *Fuzzy* e supervisores *Fuzzy* de *set points*. Os supervisores *Fuzzy* têm como entrada o erro de acompanhamento e a taxa de variação do erro no tempo, atualizando os valores dos set points de vazão e temperatura do ar e os ganhos dos controladores PID de forma adaptativa. Modelos para as plantas de temperatura e vazão foram desenvolvidos e identificados para a realização de simulações em MATLAB/SIMULINK. Os resultados deste trabalho mostram que os supervisores *Fuzzy* diminuem o tempo de subida e o erro em regime para as temperaturas de interesse, de modo que os requisitos de ensaio especificados pelos pequisadores para as respostas em malha fechada são garantidos.

Palavras-chaves: Supervisor *Fuzzy*. Controlador PID. Controle de Temperatura. Aeronave Comercial. *Mock-up*.

Abstract

The purpose of the this work is to obtain a control solution that reduces the rise time and steady state error under closed loop for temperatures of interest in a simulated environment of a commercial aircraft cabin (Mock-up). For this we developed an adaptive control which combines the conventional PID controllers installed in the plant with fuzzy PID supervisors and fuzzy set point supervisors. The Fuzzy supervisors have as input the tracking error and the rate of change of the error in time, updating the values of the air flow rate and air temperature set points and the PID controller's gains adaptively. Models for plant temperature and flow rate were developed and identified for simulations in MATLAB/Simulink. The results of this work show that the fuzzy supervisors decrease the rise time and steady state error for the temperatures of interest, so that the project requirements specified by the researchers for closed-loop responses are met.

Key-words: Fuzzy Supervisor. PID Controller. Temperature Control. Commercial Aircraft. Mock-up.

Lista de ilustrações

Sumário

1 Introdução

1.1 O Experimento

O ambiente simulado de aeronave comercial, ao qual refere-se o termo *Mock-up*, trata-se de um experimento concebido pelo Departamento de Engenharia Mecânica da Escola Politécnica em parceria com a Embraer. As instalações do experimento inspiram-se na linha de jatos E da Embraer e permitem a realização de ensaios de conforto com seres humanos. Este tipo de instalação é um modelo de cabine de passageiros, normalmente montada em feiras aeronáuticas para demonstrar o interior de um novo modelo de aeronave. O principal objetivo dessa iniciativa é adquirir, tendo em mão dados de diversos ensaios,um melhor entendimento sobre a contribuição de variáveis controladas sobre o conforto de passageiros em situações típicas enfrentadas antes e durante um voo comercial, aplicando, como objetivo final, os resultados para um melhoramento das aeronaves produzidas pela Embraer atualmente.
O conforto estudado do passageiro, no ambiente simulado, pode ser dividido em quatro partes fundamentais: conforto vibroacústico, conforto visual, conforto térmico e conforto ligado à pressão ambiente (Figura 1).

Figura 1 – Foto da parte exterior das instalações do experimento

Figura 2 – Foto da sala de ambientação

1.1.1 Instalações do Experimento

O conforto vibroacústico é estudado no experimento por meio da reprodução de sons típicos e vibrações impostas sobre a estrutura da cabine, aos quais os passageiros são submetidos em salas de embarque e dentro do avião. O conforto visual é estudado por meio de diferentes configurações de cores e intensidade das luzes instaladas no ambiente simulado de sala de embarque e dentro do avião. O conforto térmico é estudado por meio de três variáveis controladas principais: umidade, temperatura e vazão do ar em diferentes partes do experimento. O conforto ligado à pressão ambiente é estudado pela variação da pressão do ar dentro da cabine, diretamente relacionado com sensações de incômodo no aparelho auditivo experimentadas pelo passageiro em situações típicas de voo. Foram realizados dezenas de ensaios com até 30 voluntários em 2012, cujos resultados permitiram a elaboração de um modelo estatístico para previsão de conforto. No experimento, os voluntários são recebidos, respondem a um questionário, passam, em alguns ensaios, por avaliação médica e são encaminhados para a sala de ambientação. Neste ambiente, os participantes do experimento são submetidos a condições visuais, acústicas e térmicas típicas de salas de embarque de aeroportos brasileiros e recebem informações sobre o experimento e instruções de procedimento.

Após o "briefing", os passageiros são conduzidos ao interior do *mock-up*, para o início do voo. A parte interior do experimento, que simula uma cabine de aeronave comercial, mostrada na Figura 3, possui trinta assentos e é onde os participantes do experimento são acomodados. Neste espaço, os participantes serão submetidos a diferentes condições

Figura 3 – Foto do interior da cabine

térmicas, visuais, acústicas e ambientais, aproximando o ambiente da realidade de um voo típico. O ambiente de cabine é uma versão simplificada de uma cabine real.

De formar a tornar a simulação mais realista, todas as fases de um voo são contempladas: embarque, decolagem, cruzeiro, serviço de bordo, pouso e desembarque, demarcadas pela variação do som e vibração na cabine. Todas as variáveis são controladas e monitoradas a partir de uma sala onde podem ser visualizados os supervisórios de cada sistema.

1.2 Solução de Controle Original

A solução de controle original utiliza controladores PID SISO [1] para os controles de temperatura e vazão do sistema. Os controladores PID fornecem um algoritmo de controle para dois tipos de atuadores: um conjunto de resistências que aquece o ar na saída dos sistemas de condicionamento e ventiladores centrífugos que são responsáveis pela vazão desse ar condicionado para os sistemas *Liner*, cabine e *gasper*. Descrições e definições detalhadas do sistema controlado e do controle original serão fornecidas no capítulo 2.

1.3 Motivação

O ambiente simulado exige, para que os objetivos principais do experimento sejam satisfeitos, um sistema de controle que atue sobre o sistema controlado de interesse de

[1] SISO: single-input, single-output.

modo que as condições desejadas para os diversos ensaios de conforto sejam garantidas. Deseja-se, em suma, um comportamento programado do sistema controlado, com certo grau de especificidade, para que os testes propostos pelos pesquisadores envolvidos sejam conduzidos de forma satisfatória. Este grau de especificidade foi definido pelos pesquisadores envolvidos na concepção do experimento e apresentado à empresa responsável à época da licitação na forma de um documento. As especificações foram definidas em alto nível, deixando os detalhes de implementação livres para o critério das empresas concorrentes.

Foi constatado, por meio da realização de diversos ensaios, que o controle original não fornece uma curva de saída satisfatória para as temperaturas controladas e para as vazões. O experimento é de natureza térmica, envolvendo curvas de resposta exponenciais com constantes de tempo dominantes, logo os ensaios são naturalmente longos. Isto é normal para um sistema não controlado, mas indesejável em um sistema com solução de controle implementada, de modo que se almeja uma minimização dos tempos de subida e de assentamento para as temperaturas controladas, respeitadas as condições desejadas. É possível identificar, portanto, o primeiro descontentamento dos pesquisadores: os tempos de subida e assentamento das variáveis controladas são muito altos, logo menos ensaios podem ser realizados por dia. Uma diminuição dos tempos de subida e de assentamento é, portanto, um dos requisitos para o melhoramento do controle original.

Outro problema identificado nas curvas de resposta para as temperaturas e para as vazões controladas é o erro em regime. As temperaturas estabilizam-se em valores diferentes dos desejados para a realização dos ensaios. Em alguns ensaios há diferenças de até 5 °C entre as temperaturas lidas e seus respectivos *set points*. Logo, identifica-se o segundo descontentamento dos pesquisadores: as temperaturas obtidas apresentam valores não satisfatórios quando comparadas com as temperaturas desejadas para os experimentos, limitando a abrangência dos ensaios e forçando os pesquisadores e técnicos a utilizar o método de tentativa e erro para o ajuste dos *set points* (vide Figura 4). Uma diminuição do erro em regime para as temperaturas controladas é, portanto, o outro critério para o melhoramento do controle original.

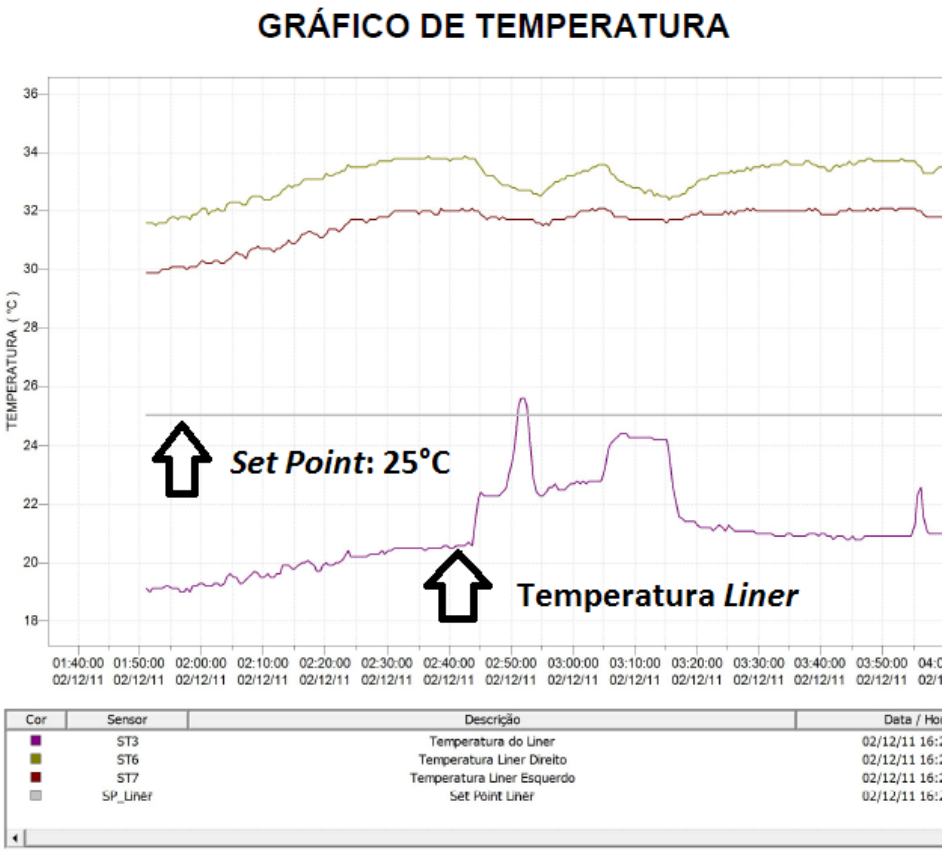

Figura 4 – Resposta de temperatura do *liner* e *set point*

1.4 Problemas Identificados na solução do Controle Original

É absolutamente necessário (ASTROM; HAGGLUND, 2010) entender os problemas fundamentais que geram as respostas indesejadas em malha fechada, antes que qualquer solução de controle possa ser proposta. Torna-se crucial, portanto, identificar de antemão os problemas na solução original, que são:

1. Os controladores PID responsáveis pela ação de controle sobre as temperaturas de interesse apresentam parâmetros fixos, sendo que as plantas controladas apresentam parâmetros variáveis;
2. O sistema controlado apresenta atrasos de transporte;
3. Os controles de vazão e temperatura são independentes.

O conhecimento físico do processo é indispensável à concepção de uma solução de controle. Mesmo uma razoável quantidade de "inteligência" em equipamentos não substitui conhecimento sobre o processo real (ASTROM; HAGGLUND, 2010). Uma síntese criteriosa dos problemas identificados será fornecida no capítulo 3.

1.5 Objetivos

As respostas em malha fechada das variáveis controladas de interesse ficam definidas como as temperaturas dos sistemas *Liner* e cabine. As temperaturas de interesse, toda vez que mencionadas, se referirão às temperaturas dos sistemas *Liner* e cabine. Deseja-se obter um controle que diminua os tempos de subida e assentamento para as temperaturas de interesse assim como o erro em regime. O projeto também visa aproveitar ao máximo os equipamentos disponibilizados pela solução original, otimizando o patrimônio atual da Escola Politécnica. O presente trabalho propõe-se a resolver os problemas identificados e categorizados na seção anterior da forma apresentada na subseção a seguir, prezando pela simplicidade de implementação. Uma justificativa detalhada da solução será apresentada no capítulo 3.

1.5.1 Metodologia

Este projeto compreende:

1. balanço de energia para as interações térmicas entre os sistemas controlados;
2. desenvolvimento do modelo para o ventilador centrífugo;
3. desenvolvimento do modelo de simulação dos sistemas controlados;
4. projeto do sistema de controle;
5. simulação numérica do controle proposto via MATLAB/Simulink;
6. projeto de um protótipo para realização de testes.

1.6 Organização do Texto

Este texto está organizado do seguinte modo: No capítulo 2 será dada uma descrição geral dos sistemas de interesse, identificando seus componentes: controladores, atuadores, plantas e sensores. No capítulo 3 será apresentada uma revisão do segundo método de Ziegler-Nichols, que será aplicado para a sintonia em malha fechada dos controladores PID das vazões, e a metodologia de projeto de supervisores *fuzzy*. No capítulo 3 também será apresentado o desenvolvimento da solução de controle proposta, com o projeto dos supervisores *fuzzy*. No capítulo 4 serão mostrados os resultados obtidos a partir de simulações em MATLAB da solução proposta pelo trabalho. No capítulo 5 será apresentado o projeto executivo do protótipo de *Mock up*. No capítulo 6, serão apresentados os resultados do controle proposto obtidos com o protótipo. Finalmente, também no capítulo 6, as principais conclusões do trabalho serão levantadas e discutidas.

2 Descrição e Modelagem dos Sistemas

2.1 Descrição Geral

Os sistemas controlados do *Mock-up* podem ser subdivididos em três subsistemas independentes, tendo cada um seu próprio sistema de condicionamento de ar: *Gasper*, Cabine e *Liner*. O sistema *Gasper* insufla ar para válvulas ajustáveis difusoras de ar, instaladas no teto da cabine, sobre os assentos, controladas manualmente pelos passageiros durante o voo. O sistema *Gasper* não será abordado neste trabalho. A palavra sistema, no contexto deste trabalho, englobará todos os elementos da malha de controle: controladores, atuadores, sensores e plantas. Quando as palavras "Cabine" ou "*Liner*" forem empregadas individualmente, estarão referindo-se, respectivamente, às massas de ar contidas nas regiões da Cabine e do *Liner*, definidas neste capítulo, representado as plantas a serem controladas.

2.2 Sistema Cabine

2.2.1 Cabine (Planta)

A cabine é a região interna do *Mock-up*, como mostra a Figura 5. O ar insuflado nesta região tem suas condições controladas pelo sistema Cabine. A cabine (planta), propriamente dita, é essencialmente um tanque de ar com dois sistemas de atuação:

1. Conjunto de resistências elétricas para aquecimento do ar da cabine proveniente do sistema de condicionamento;

2. Ventilador centrífugo para controle da vazão volumétrica de ar frio insuflado proveniente do sistema de condicionamento.

Esses sistemas de atuação controlam condições de temperatura e vazão. O sistema de atuação sobre a umidade do ar não será abordado neste trabalho. Essas condições são controladas durante os ensaios, com o intuito de observar seus efeitos no conforto térmico dos passageiros.

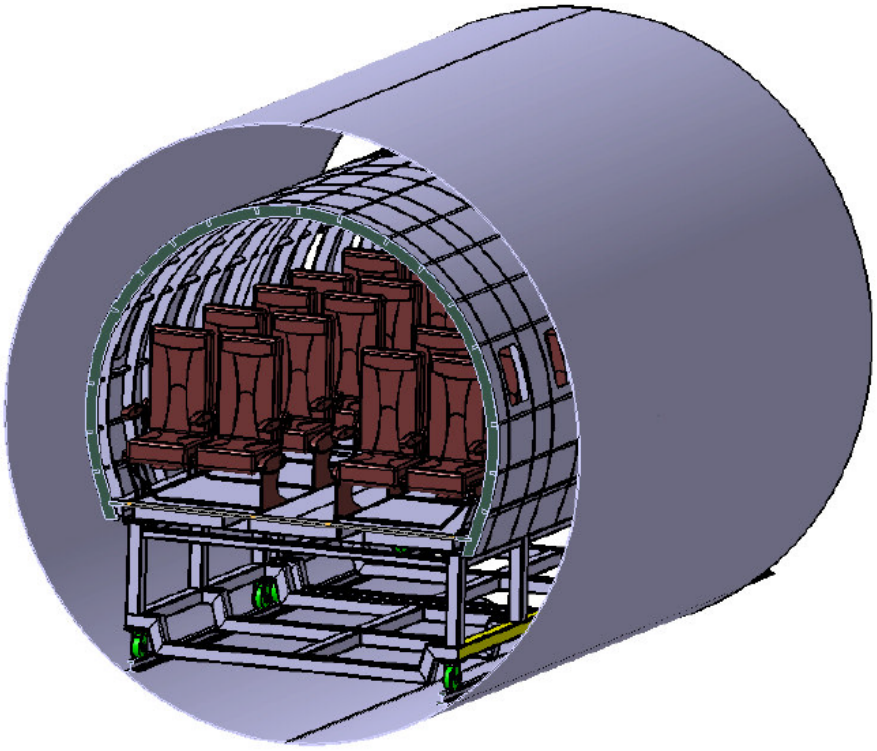

Figura 5 – Vista Isométrica do *Mock-up* integrado

2.2.2 Atuadores do Sistema Cabine

2.2.2.1 Conjunto de Resistências

O controle de temperatura do sistema cabine é realizado por meio do aquecimento do ar de alimentação saído do sistema de condicionamento. Este ar se encontra a uma temperatura bem mais baixa que o limite inferior de operação para a cabine (14 °C), configurada na receita do ensaio. O ar é aquecido pela potência dissipada por efeito Joule no conjunto de resistores. A potência dissipada é controlada por um sinal de voltagem na forma de PWM (Figura 6), enviado ao conjunto de resistências pelo controlador. O conjunto de resistências do sistema cabine (AT3) está em configuração triângulo (Figura 7).

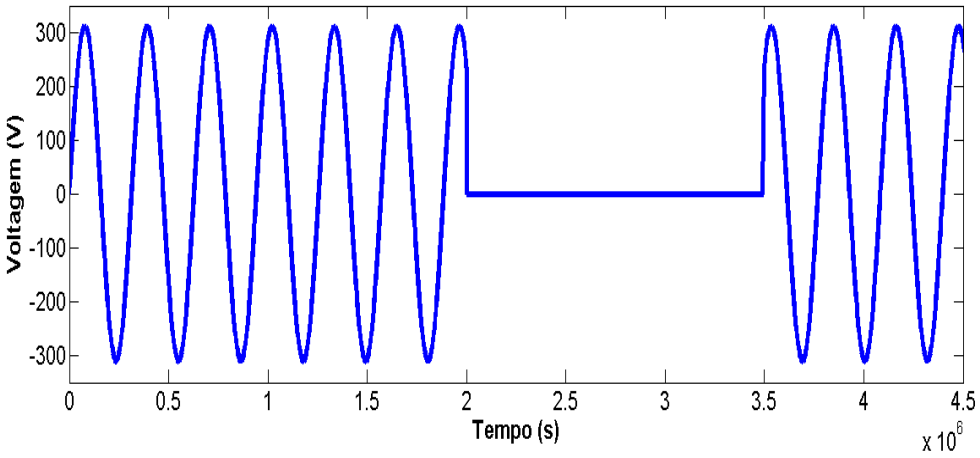

Figura 6 – Formato do PWM enviado pelo controlador

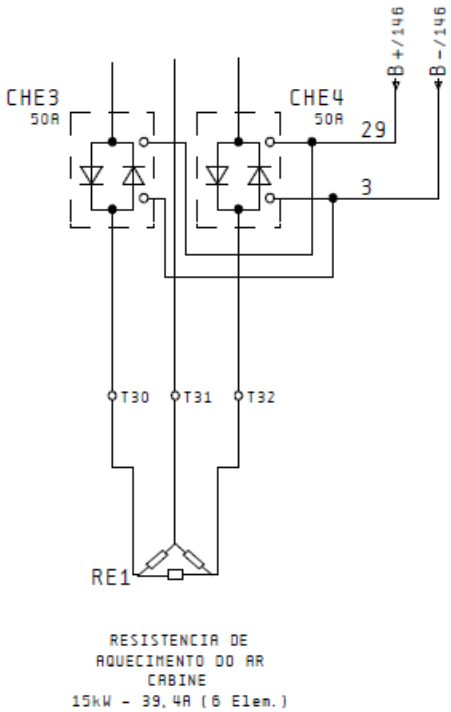

Figura 7 – Conjunto de Resistências do Sistema Cabine

2.2.2.2 Ventilador Centrífugo

O controle de vazão do sistema cabine é realizado por meio de um ventilador centrífugo de 3 CV modelo VTI-300 (VC2) (Figura 8) que retira ar do sistema de condicionamento da cabine. O controle deste ventilador centrífugo é feito em malha fechada, pois há realimentação de valores de vazão para a cabine (OGATA, 2011). A faixa de operação para a vazão foi especificada entre 700 e 1600 m^3/h.

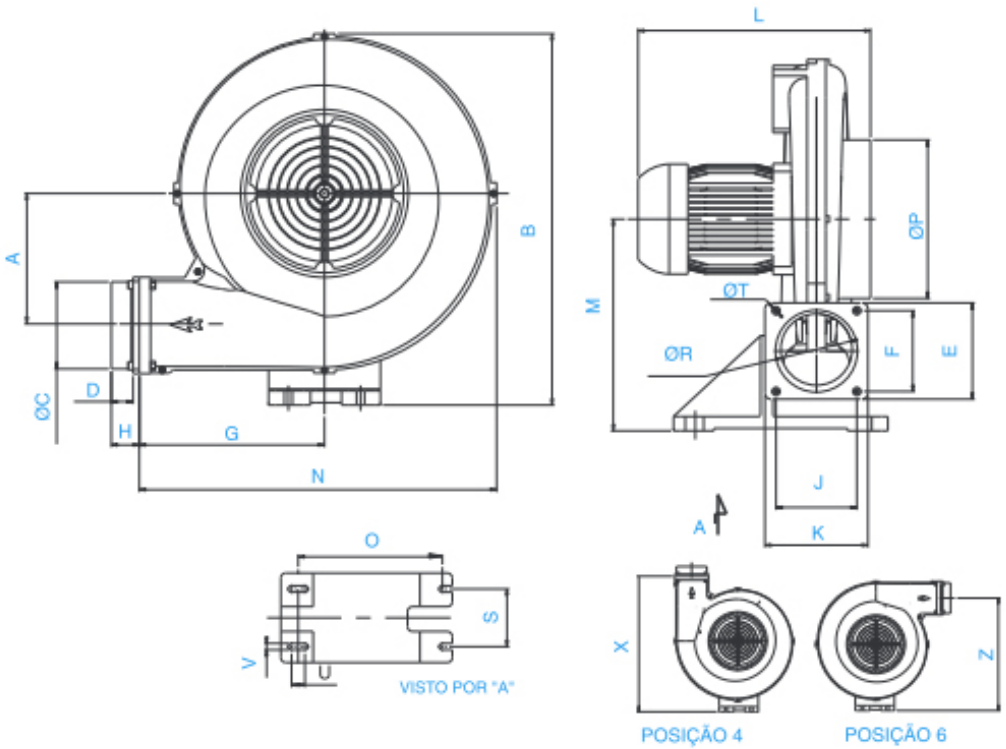

Figura 8 – Ventilador centrífugo (VC2) para controle de vazão de ar na cabine.

Fonte: Manual do Fabricante.

2.2.3 Sensores de Temperatura

A temperatura do ar na região da cabine é medida por três termorresistências (ST2, ST4 e ST5) PT-100 classe A, ligação com três fios, da fabricante Warme do Brasil, indicadas na Figura 9. A leitura fornecida ao controlador é a média das temperaturas medidas pelos três sensores. Os sensores ST4 e ST5 estão instalados nos retornos de ar à esquerda e a direita, respectivamente. O terceiro sensor (ST2) está instalado sob uma das poltronas de passageiros. A temperatura de bulbo seco, de acordo com o especificado, terá seu limite inferior em 14 °C e superior em 37 °C.

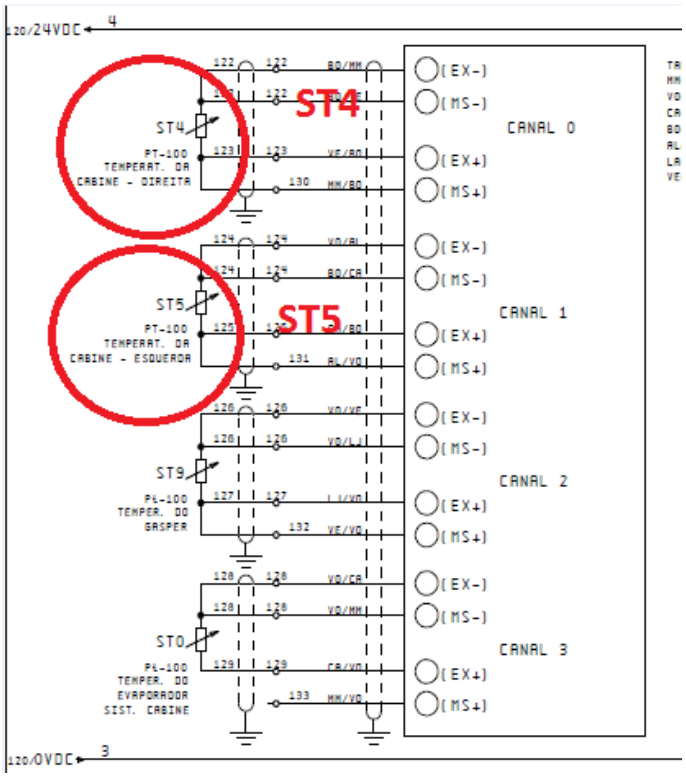

Figura 9 – Esquema Elétrico dos Sensores Instalados na Cabine

2.2.4 Sensores de Vazão

A leitura da vazão é fornecida pelos fluxômetros de massa térmica MV1 e MV2, modelo FT2 da fabricante Fox Thermal Instruments (CONTECH, 2007), instalados nos dutos que levam o ar para a cabine. Este sensor opera a partir do princípio que um sensor aquecido colocado em uma corrente de ar transfere calor na proporção direta da velocidade da massa da corrente. Há dois sensores em diferentes pernas de um circuito em ponte balanceada: um sensor mede a temperatura do fluido e um segundo sensor é mantido a uma temperatura que varia de modo que o $\triangle$T, em relação à temperatura do fluido, permaneça constante e positivo. A energia fornecida ao sensor para manter esse $\triangle$T constante é diretamente proporcional à vazão de ar. A precisão de medida deste sensor é de 1%.

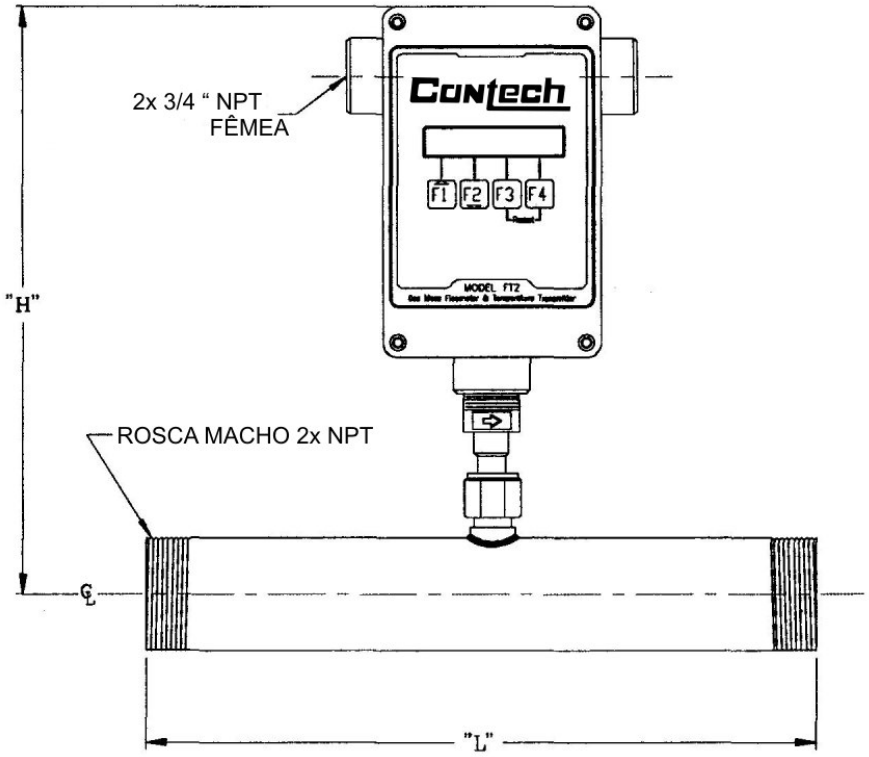

Figura 10 – Medidor de Vazão da CONTECH

Fonte: Manual do FT2, CONTECH 2007.

2.3 Sistema *Liner*

2.3.1 *Liner*(Planta)

O *Liner* é a região existente entre as paredes internas e as paredes externas da cabine, conforme mostra a Figura 11 abaixo. Sua principal função é resfriar a parede interna da cabine de formar a simular as baixas temperaturas das paredes internas de um avião em cruzeiro. O ar insuflado nesta região tem suas condições controladas pelo sistema *Liner*. Este controla condições de temperatura e vazão. O *Liner* (planta), propriamente dito, é essencialmente um espaço anular com ar e dois sistemas de atuação:

1. Conjunto de resistências elétricas para aquecimento do ar do *liner* proveniente do sistema de condicionamento;

2. Ventilador centrífugo para configurar a vazão de ar insuflado proveniente do sistema de condicionamento.

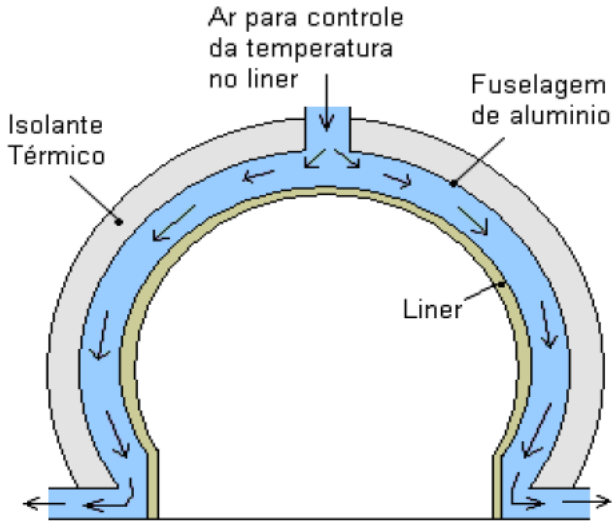

Figura 11 – Ilustração do *Liner*

Esses sistemas de atuação controlam condições de temperatura e vazão. O sistema de atuação sobre a umidade do ar não será abordado neste trabalho. Essas condições são controladas durante os ensaios, com o intuito de observar seus efeitos no conforto térmico dos passageiros.

2.3.2 Atuadores do Sistema *Liner*

2.3.2.1 Conjunto de Resistências

O controle de temperatura do sistema *Liner* é realizado por meio do aquecimento do ar de alimentação saído do sistema de condicionamento a uma temperatura bem mais baixa que o limite de operação (14 °C), configurada na receita do ensaio. O controle de temperatura é realizado de forma análoga ao controle de temperatura da cabine.

O aquecimento é realizado pelo envio de um pulso PWM para um conjunto de resistências (AT1) em configuração duplo triângulo (Figura 12). O *duty cycle* do PWM é controlado pelo PID, de modo que o tempo em nível lógico *high* em relação ao tempo total do pulso determina a potência dissipada pelo conjunto de resistências. Essa potência dissipada aquece o ar na saída do condicionamento que, com certo atraso de transporte, chega à região do *Liner*.

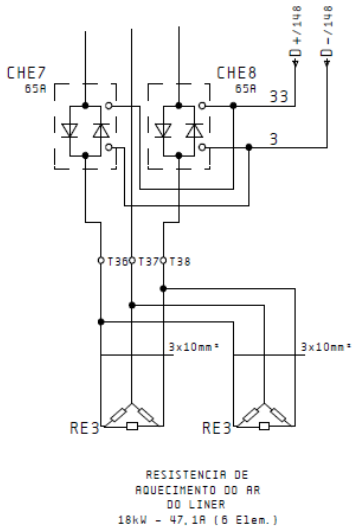

Figura 12 – Conjunto de Resistências do Sistema *Liner*

2.3.2.2 Ventilador Centrífugo

O controle de vazão do sistema *Liner* é realizado por meio de um ventilador centrífugo (VC1), modelo R3G450-AG10-13 da fabricante EBM-Pabst, que retira ar do sistema de condicionamento do *Liner*. Este controle é feito em malha aberta, pois não há realimentação de valores da vazão de saída. Foi levantada uma curva de calibração de vazão mássica de ar em função da velocidade de rotação das pás do ventilador centrífugo. Esse modelo de ventilador possui um controlador PID instalado internamente para controlar a velocidade de rotação. O CLP[1], com uma tabela de interpolação na memória obtida a partir da curva de calibração, manda então um sinal para o ventilador, dependendo da vazão configurada na receita de ensaio. A faixa de operação para a vazão foi especificada entre 1.000 e 5.000 m^3/h.

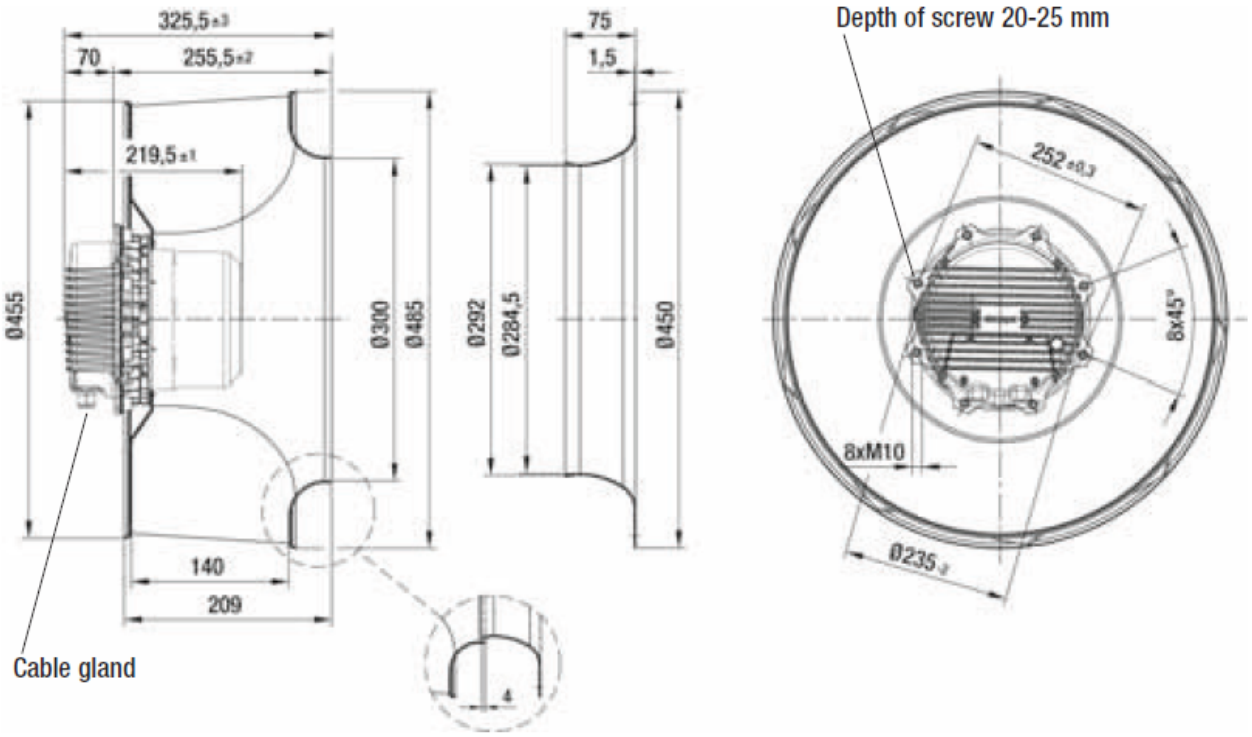

Figura 13 – Ventilador do *Liner*(VC4).

[1] CLP: controlador lógico programável

2.3.3 Sensores de Temperatura

A temperatura do ar no *Liner* é medida por duas termorresistências (ST6 e ST7) PT-100, classe A com três fios, da fabricante Warme do Brasil, indicadas na Figura **??**. A leitura fornecida ao controlador é a média das temperaturas medidas pelos dois sensores. A temperatura de bulbo seco, de acordo com o especificado, terá seu limite inferior em 14 °C e superior em 37 °C. Há também um sensor (ST3) do mesmo tipo na saída do ar proveniente do sistema de condicionamento, medindo a temperatura do ar após a passagem pelo conjunto de resistências.

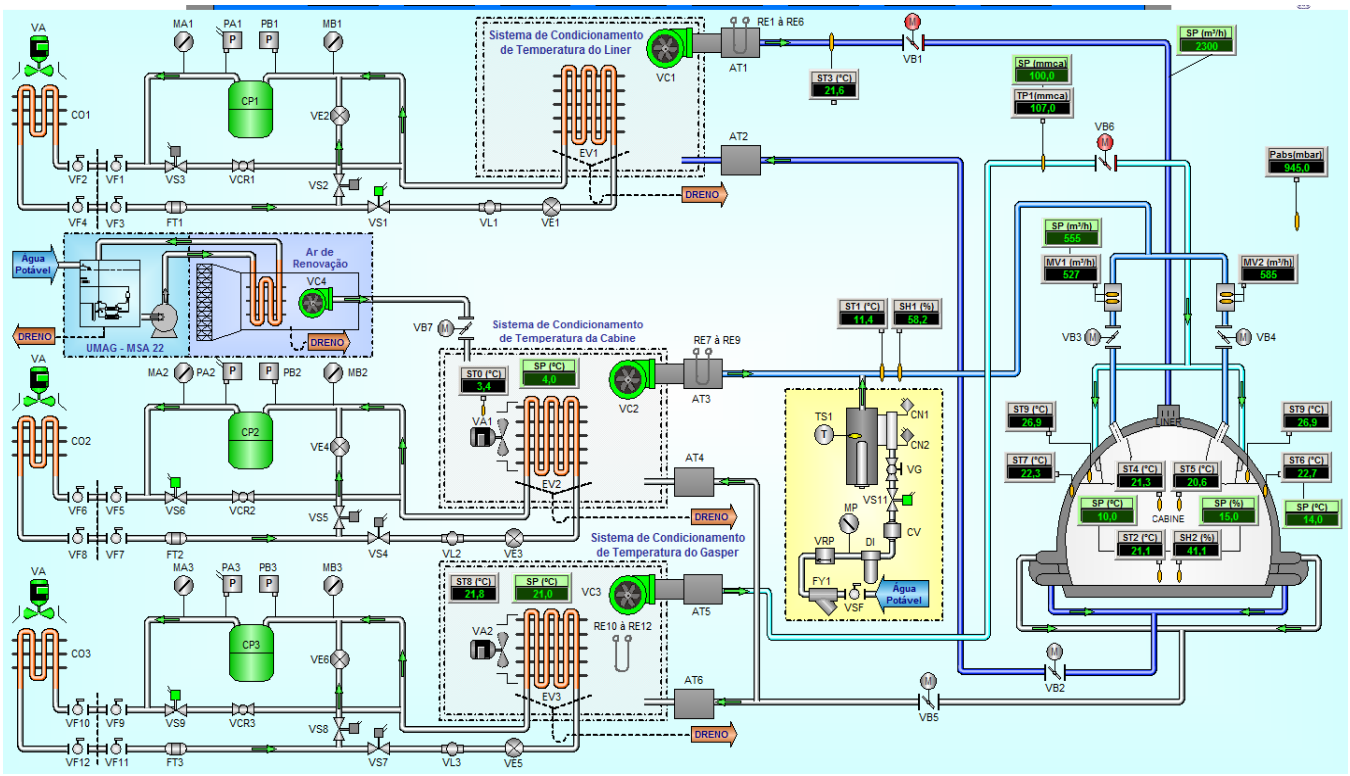

Figura 14 – Supervisório

2.4 Controlador

São utilizados controladores do tipo PID para o controle das temperaturas e das vazões dos sistemas *liner* e cabine, exceto para a vazão do *liner*, em que é utilizada uma tabela de interpolação que relaciona rotação com vazão.

2.4.1 Controlador Lógico Programável (CLP)

O CLP utilizado é o Modicon M340 P342030 da fabricante *Schneider Electric* com sete módulos para conexões analógicas e digitais, uma fonte CPS3500 (alimentação de 135 mA max e 24 V CC) e um processador M340 com conexões RJ45 (TCP/IP Ethernet), DB9 (CANOpen) e USB.

Figura 15 – Modicon 340 da *Schneider Electric*

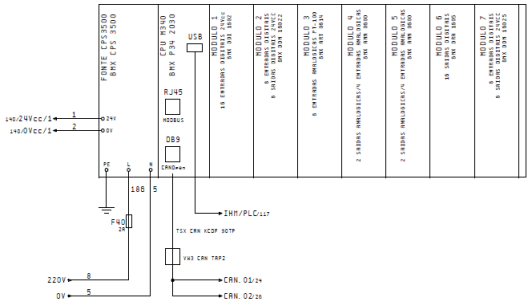

Figura 16 – Diagrama Elétrico Simplificado do Modicon 340

2.4.2 Bloco PID

Os controladores PID são indiscutivelmente os controladores mais utilizados na indústria nas últimas décadas (ASTROM; HAGGLUND, 2010). Sua popularidade é dada principalmente pela constatação de que podem ser implementados de diversas formas e em inúmeras aplicações tecnológicas em geral. A função de transferência do bloco PID implementada pelo Modicon 340 é:

$$PID(s) = K_p + \alpha * \frac{1}{T_i * s} + \alpha * \frac{T_d * s}{1 + \frac{T_d}{N} * s}. \tag{2.1}$$

O fator de escala α é uma constante de ajuste que relaciona o limite de variação das saídas com o limite de variação das entradas.

2.4.2.1 Ação Proporcional K_p

A ação de controle é proporcional ao erro. A constante de proporcionalidade é o ganho do controlador, normalmente adimensional. O ganho define o quanto a variável de

controle deve variar em correspondência a uma variação unitária no sinal de erro.

$$u(t) = K_p e(t) \tag{2.2}$$

2.4.2.2 Ação Integral T_i

A função principal da ação de controle integral é reduzir o erro estacionário, ou seja, fazer com que a saída em regime seja a mais próxima possível do set point. Considerando apenas a ação integral, a saída do controlador será proporcional a integral do sinal de erro ao longo do tempo (TANNURI; CRUZ, 2008):

$$u(t) = \frac{1}{T_i} \int_0^t e(t)dt \tag{2.3}$$

2.4.2.3 Ação Derivativa T_d

A ação de controle derivativa tem por propósito melhorar a estabilidade em malha fechada. A instabilidade do processo pode ser intuitivamente descrita da seguinte forma. Devido à dinâmica do processo, levará algum tempo até que uma mudança na variável controlada possa ser notada na saída do processo. Portanto, o sistema de controle estará atrasado na correção de um erro. A ação derivativa pode ser entendida como uma correção preditiva aproximada, levando em conta a tangente da curva de erro. A ação derivativa em aplicações industriais geralmente acompanha um filtro derivativo representado pelo termo N na equação 2.1.

$$u(t) = T_d \frac{de(t)}{dt} \tag{2.4}$$

2.5 Modelagem dos Sistemas *Liner* e Cabine

As plantas e os atuadores de cada sistema serão modelados a seguir. As plantas serão modeladas por equações diferenciais ordinárias de primeira ordem, considerando: a ação dos atuadores sobre as massas de ar, o acoplamento gerado pela troca de calor por convecção entre *Liner* e cabine, cargas térmicas de passageiros e tripulantes e cargas térmicas de equipamentos elétricos.

2.5.1 Modelagem das Plantas

A modelagem a seguir foi desenvolvida com base nas equações de conservação de energia e transferência de calor (BERGMAN et al., 2011). O intuito desta modelagem é obter um melhor entendimento físico sobre as dinâmicas das plantas. O equacionamento a seguir despreza os gradientes de temperatura das massas de ar dentro das regiões do

liner e cabine (COUGHANOWR; KOPPEL, 1978). Foi verificado experimentalmente, por meio de sucessivas medições de temperatura em vários pontos das regiões de interesse, que a variação das temperaturas de interesse no espaço é desprezível em regime. As temperaturas medidas são médias de sensores instalados. O calor específico do ar para o *liner* e cabine será considerado invariante no tempo. O sistema (*Liner* + Cabine) será considerado isolado do exterior, ou seja, as trocas térmicas dos sistemas de interesse com a vizinhança serão desprezadas. A carga térmica do sistema foi determinada da seguinte maneira:

1. As temperaturas dos sistemas de interesse eram levadas até um valor comum, 10 °C acima das temperaturas ambientes inicialmente medidas para cada região;
2. Quando as temperaturas de interesse estavam praticamente iguais, o sistema de controle era então desligado e observava-se o decaimento das temperaturas de interesse no tempo.

Com a realização desses ensaios, verificou-se que as temperaturas demoravam aproximadamente 90 minutos para variarem 2 °C. Os acoplamentos com a vizinhança foram desprezados. O modelo também considera os passageiros como fornecedores de potência, simplificando a natureza das trocas térmicas do corpo humano com o ar da cabine. As poltronas não serão consideradas no equacionamento. A temperatura da parede interna da cabine será considerada igual à temperatura do ar no *liner*.

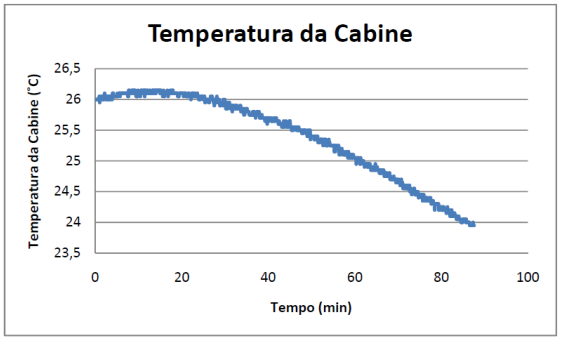

Figura 17 – Decaimento da Temperatura do Ar da Cabine

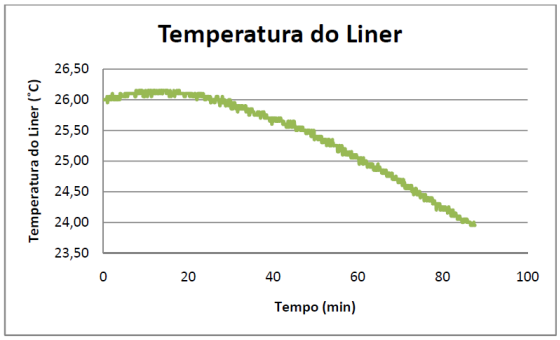

Figura 18 – Decaimento da Temperatura do Ar do Liner

2.5.1.1 Equacionamento

Conservação da energia para o ar para um volume de controle genérico fica:

$$\dot{Q} = \dot{E}_{gerada} + (\dot{E}_{entrada} - \dot{E}_{saída}) \tag{2.5}$$

em que

$$\dot{Q} = m * c * \frac{dT}{dt}$$

$$\dot{E}_{gerada} = \dot{Q}_{pessoas}(t) + \dot{Q}_{elétrica}(t) + P_{Resistência}(t)$$

$$\dot{E}_{entrada} = \dot{m}_{entrada}(t) * c * T_{entrada}$$

$$\dot{E}_{saída} = \dot{m}_{saída}(t) * c * T_{saída}$$

Para o sistema *Liner* + Cabine:

$$\dot{T}(t) = KT(t) + F(t) - C \tag{2.6}$$

em que

$$\dot{T}(t) = \begin{bmatrix} \frac{dT_{cabine}}{dt} \\ \frac{dT_{liner}}{dt} \end{bmatrix}, T(t) = \begin{bmatrix} T_{cabine} \\ T_{liner} \end{bmatrix}, K = \begin{bmatrix} \frac{-h*A-\dot{m}_{s,cabine}*c_{ar,cabine}}{m_{ar,cabine}*c_{ar,cabine}} & h*A \\ h*A & \frac{-h*A-\dot{m}_{s,liner}*c_{ar,liner}}{m_{ar,liner}*c_{ar,liner}} \end{bmatrix}$$

$$F(t) = \begin{bmatrix} \frac{Q_{pessoas}(t)+Q_{elétrica}(t)+R_{resistência,cabine}(t)+\dot{m}_{e,cabine}(t)*c_{ar,cabine}*T_{ref}}{m_{ar,cabine}*c_{ar,cabine}} \\ \frac{R_{resistência,liner}(t)+\dot{m}_{e,liner}(t)*c_{ar,liner}*T_{ref}}{m_{ar,liner}*c_{ar,liner}} \end{bmatrix}, C = \begin{bmatrix} \frac{\dot{m}_{s,cabine}*Tref}{m_{ar,cabine}} \\ \frac{\dot{m}_{s,liner}*Tref}{m_{ar,liner}} \end{bmatrix}$$

- T_{cabine} é a temperatura do ar na cabine;
- T_{liner} é a temperatura do ar no *liner*;
- h é o coeficiente de convecção entre o ar da cabine e a parede interna;
- A é a área da parede interna da cabine;
- $\dot{m}$ é a vazão mássica;
- c é o calor específico do ar a 25 °C;
- T_{ref} é a temperatura de referência;
- $\dot{Q}_{pessoas}$ é a potência fornecida por tripulantes e passageiros dentro da cabine;
- $\dot{Q}_{elétrica}$ é a potência dissipada por equipamentos elétricos dentro da cabine;
- $P_{Resistência}$ é a potência fornecida pelos conjuntos de resistências do *liner* e da cabine.

A vazão de renovação da cabine ($\dot{m}_{s,cabine}$) é constante e igual a 750 $\frac{m^3}{h}$. A vazão de saída do *liner* ($\dot{m}_{s,liner}$) é igual a vazão de entrada ($\dot{m}_{e,liner}$).

2.5.2 Modelagem dos Atuadores

2.5.2.1 Conjunto de Resistências

A potência dissipada por efeito Joule no conjunto de resistências aquece o ar saído do sistema de condicionamento. A potência dissipada é controlada por um pulso de tensão PWM com *duty cycle* controlado. Este sinal PWM é gerado pelo controlador PID. O valor da potência dissipada no conjunto de resistências é:

$$P_{Resistência}(t) = P_{máxima} * DC \tag{2.7}$$

em que

$$DC = \frac{T_{High}}{T_{Total}}. \tag{2.8}$$

DC é o *duty cycle* do sinal PWM.

2.5.2.2 Ventiladores Centrífugos

São utilizados dois ventiladores centrífugos para o controle de vazão de ar de entrada nos dois sistemas. Os motores dos ventiladores são assíncronos trifásicos ligadas à uma rede de 220 V. A rotação dos ventiladores é controlada por pulsos PWM enviados pelo PID aos inversores de frequência, conforme o esquema de VC2 na Figura 19.

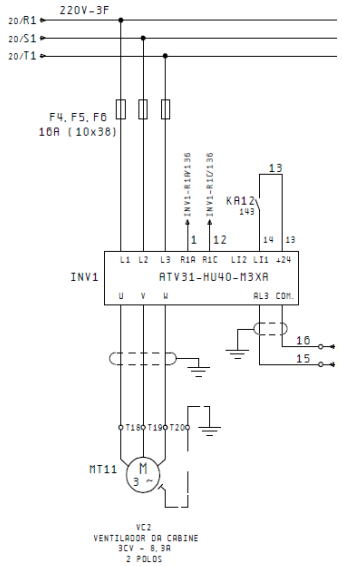

Figura 19 – Ventilador Centrífugo da Cabine

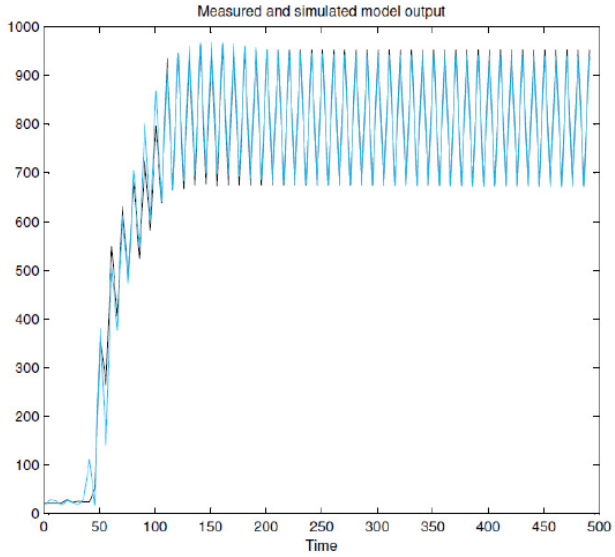

Figura 20 – Comportamento Não Linear do Ventilador Centrífugo

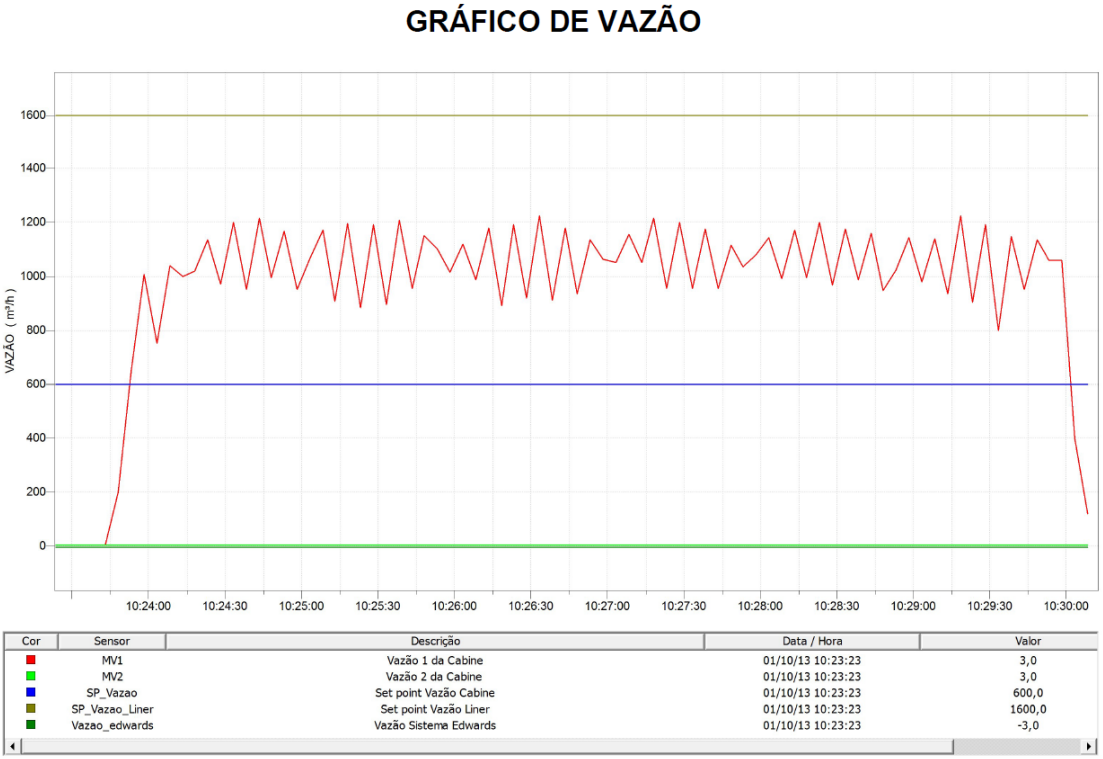

Figura 21 – Comportamento Não Linear do Ventilador Centrífugo

A vazão na saída dos ventiladores centrífugos varia de forma não linear com a rotação. O gráfico 20 mostra a vazão lida a cada 5 segundos (linha preta) na saída do ventilador centrífugo da cabine em malha fechada em comparação com um modelo de Hammerstein-Wiener obtido para o ventilador.

O modelo de Hammerstein-Wiener (em azul na Figura 20) de melhor ajuste (92%) obtido por meio do MATLAB possui as seguintes características:

```
Discrete-time OE model:  y(t) = [B(z)/F(z)]u(t) + e(t)
  B(z) = -0.916 z^-1 + z^-2

  F(z) = 1 - 0.7163 z^-1 - 0.9698 z^-2 + 0.7465 z^-3
```

Figura 22 – Modelo em tempo discreto do Ventilador Centrífugo da Cabine

- uma entrada, uma saída;
- 2 pólos;
- 1 zero;
- atraso.

Parte não linear:

- entrada pw-linear com 10 breakpoints;
- saída pw-linear com 10 breakpoints.

3 Controle Proposto

Neste capítulo serão apresentadas as técnicas de controle utilizadas no trabalho, as principais observações sobre o controle original que motivaram a solução proposta e a matriz de alteração dos ganhos do controlador PID. Inicialmente será revisado o método de sintonia em malha fechada proposto por Ziegler-Nichols. Em seguida, uma breve revisão de metodologia de projeto de supervisores *fuzzy* será apresentada, assim como os fundamentos do controle *fuzzy*. Por fim, será apresentada a matriz de alteração dos ganhos para os supervisores *fuzzy*.

3.1 Segundo Método de Ziegler-Nichols

Este método é aplicado com o sistema operando em malha fechada. Utilizando-se somente a ação de controle proporcional (T_i e $T_d = 0$), aumenta-se o valor do ganho K_p de zero até um valor crítico K_{cr}, a partir do qual o sistema apresenta oscilações mantidas. São determinados experimentalmente os valores para o ganho K_{cr} e período crítico correspondente P_{cr}, conforme indicado na Figura 23. Os valores dos parâmetros do PID são então ajustados de acordo com a tabela mostrada na Figura **??** proposta por Ziegler e Nichols. Se o sinal de saída não apresentar esse comportamento oscilatório, o método não pode ser aplicado (CAMPOS; TEIXEIRA, 2006).

Controlador	K_p	T_i	T_d
P	$0,5K_{cr}$	—	—
PI	$0,45K_{cr}$	$P_{cr}/1,2$	—
PID	$0,6K_{cr}$	$P_{cr}/2$	$P_{cr}/8$

Fonte: Tannuri e Cruz, 2008.

3.2 Metodologia de Projeto dos Supervisores

A metodologia de desenvolvimento tem como objetivo formalizar e estruturar um procedimento para a concepção do projeto de controle (SHAW; SIMOES, 2007). Inicialmente serão identificados e definidos os problemas de controle da solução original. Uma análise qualitativa dos processos envolvidos nos sistemas controlados será então apresentada. Em seguida a solução corretiva de controle poderá ser proposta e justificada. Nesta etapa será utilizada a metodologia de desenvolvimento apresentada por (GUERRA, 1998). Finalmente, serão realizadas simulações para definição dos parâmetros.

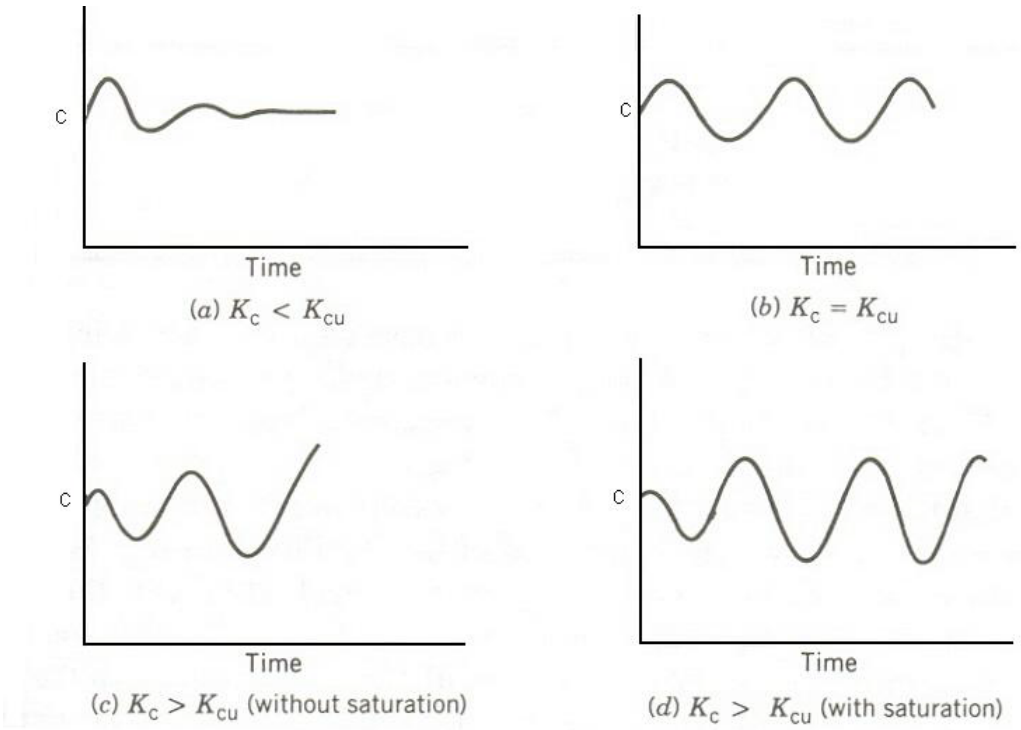

Figura 23 – Sintonia pelo segundo método de ZN

Fonte: Campos e Teixeira, 2006.

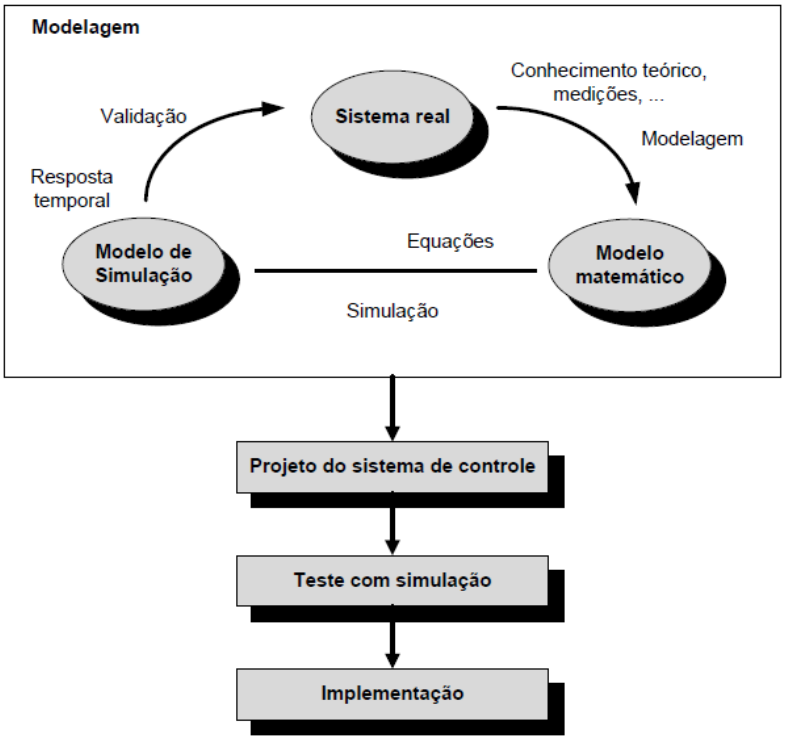

Figura 24 – Projeto de um sistema de controle

Fonte: Zupancic et al., 2004.

3.2.1 Problemas da Solução Original

Os dois problemas principais da solução de controle original para as temperaturas de interesse são:

1. tempos de subida e de assentamento altos;
2. resposta oscilatória com erro em regime.

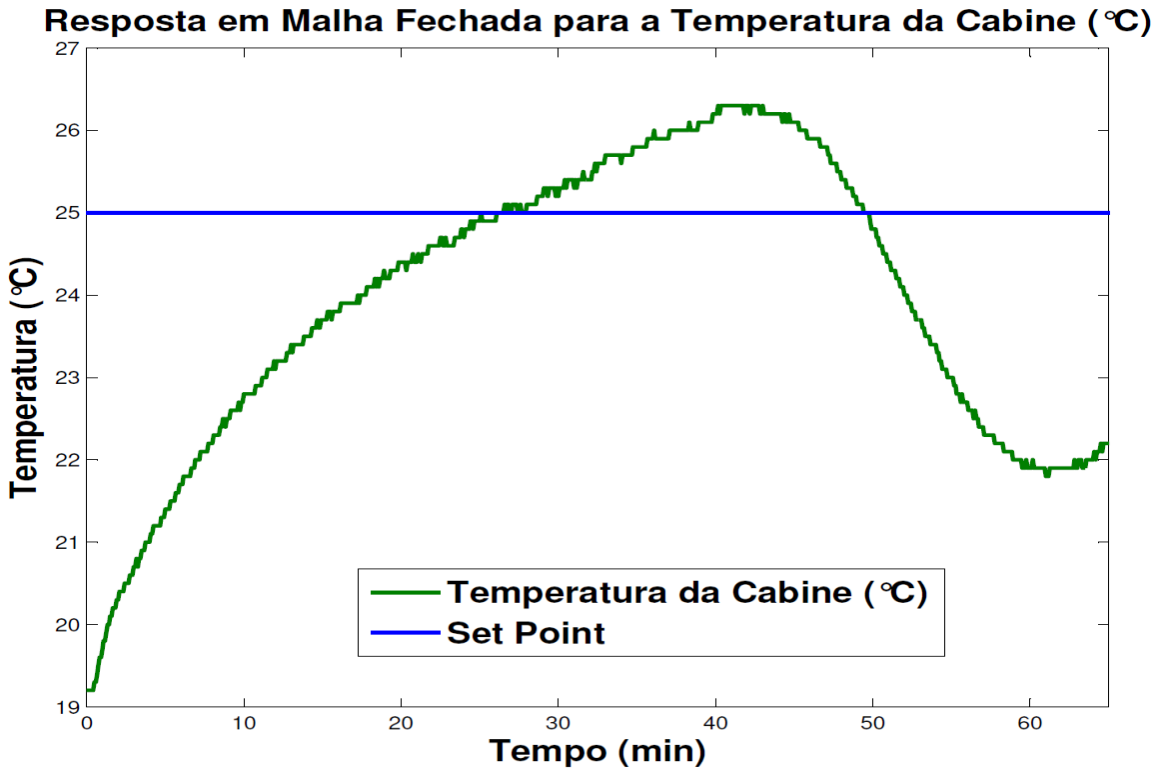

Figura 25 – Resposta em Malha Fechada da Temperatura da Cabine

As cargas térmicas envolvidas no experimento são relativamente altas, conforme apresentado no capítulo introdutório, logo os sistemas de temperatura controlados são naturalmente lentos. Há um atraso para a ação de controle das resistências elétricas (atuadores da temperatura). Isto explica os tempos de subida e de assentamento tipicamente altos. O erro em regime pode ser explicado por diversos fatores: (a) os parâmetros do sistema não são fixos no tempo, pois as condições de ensaio variam, assim como as condições ambientais externas como temperatura ambiente, radiação solar, etc; (b) Há um acoplamento entre as temperaturas do *liner* e da cabine, porquanto há troca térmica por convecção entre esses dois sistemas e (c) inicialmente os controladores PID utilizados foram sintonizados por tentativa e erro pelos operadores, sem a aplicação de métodos heurísticos mais sofisticados.

Os problemas principais da solução de controle original para a vazão da cabine são:

1. resposta oscilatória da vazão da cabine com erro;
2. não há realimentação por sensor para a vazão do sistema *liner*.

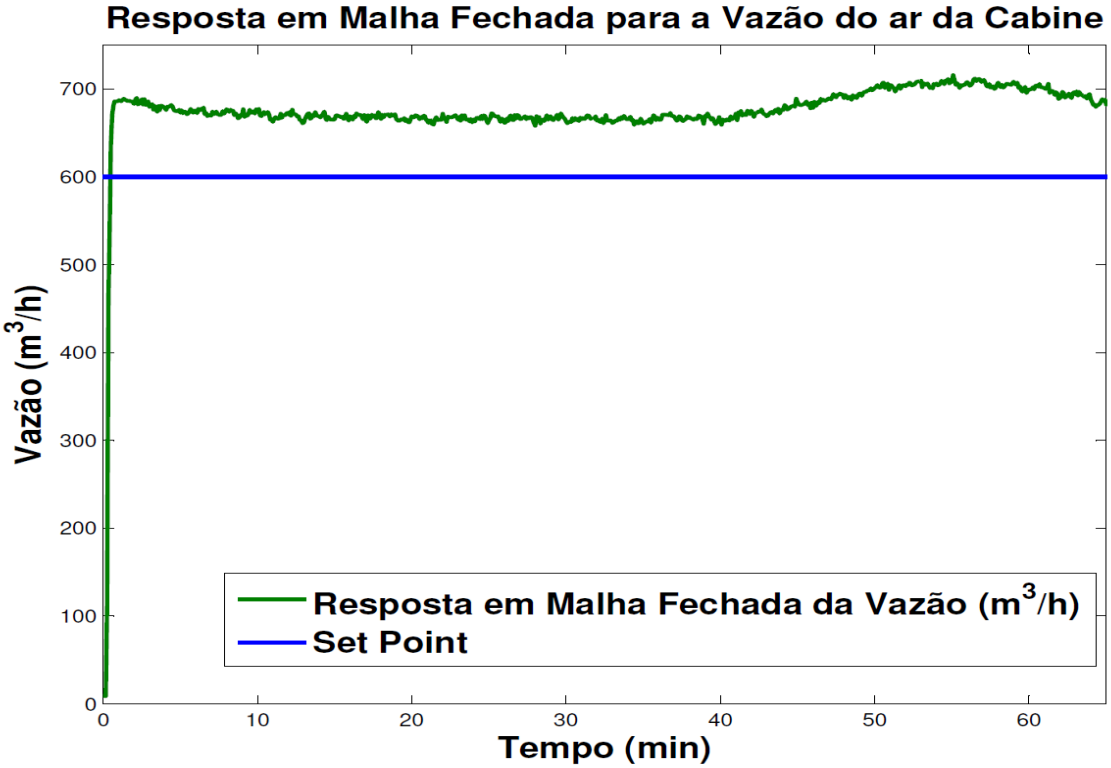

Figura 26 – Resposta em Malha Fechada da Vazão da Cabine

Isto pode ser explicado pelo comportamento não linear dos ventiladores centrífugos para a vazão de ar em relação à rotação das pás. Qualquer variação do diferencial de pressão do ventilador, para uma rotação fixa, implica em uma variação na vazão lida. Para o ventilador centrífugo do sistema *liner*, não há realimentação. Foi levantada uma curva de vazão x rotação e uma tabela de conversão foi inserida no Modicon 340.

3.2.2 Características dos Sistemas Controlados

A solução de controle proposta pauta-se em três características do sistema controlado: atraso, parâmetros variáveis e não linearidade. O atraso é verificado para o sistema de controle das temperaturas, em que há um atraso para que a saída (potência dissipada) dos atuadores (conjunto de resistências) acompanhe a lei de controle do PID, tanto em resfriamento quanto em aquecimento. Os parâmetros do modelo dos sistemas controlados de temperatura variam de acordo com as condições de ensaio e com as condições climáticas do ambiente. O número de passageiros e tripulantes, a potência elétrica dissipada

pelos equipamentos, assim como as condições de escoamento do ar do *liner* podem variar, dependendo do ensaio realizado. Assim, por mais que os controladores PID para as temperaturas sejam sintonizados para uma condição de ensaio, a medida em que os parâmetros da planta vão mudando no tempo, os ganhos calculados por sintonia podem não garantir mais uma resposta em malha fechada adequada. A não linearidade é encontrada na resposta dos ventiladores centrífugos, assim como na variação do coeficiente de convecção entre os sistemas *liner* e cabine para diferentes condições de escoamento. Logo,o controle proposto deverá lidar com atrasos, não linearidades e adaptatividade.

3.2.3 Observações Práticas sobre o Controle Original

Verificou-se que a sintonia pelo segundo método de Ziegler-Nichols (1942) dos parâmetros dos controladores PID melhorava significativamente as respostas em malha fechada para as temperaturas e vazões de interesse em termos de erro em regime.

Verificou-se experimentalmente, para as temperaturas de interesse, que a resposta em malha fechada apresentava melhora significativa em termos de tempos de subida e assentamento quando os *set points* de temperatura e vazão eram modificados. Por exemplo, no aquecimento do sistema, para uma resposta mais rápida, o *set point* de temperatura era configurado em alguns graus acima do desejado e a vazão era configurada em seu valor máximo.

3.3 Solução de Controle Proposta

3.3.1 Abordagem *Fuzzy*

As observações práticas apresentadas na subseção anterior sugerem que os problemas de controle identificados poderiam ser resolvidos, em parte, com um ajuste adaptativo dos ganhos dos controladores PID. Uma forma de acelerar o processo de convergência dos ganhos dos controladores é utilizar ganhos calculados por meio do segundo método de Ziegler-Nichols (1942) como ganhos iniciais. A lentidão causada pelos atrasos no controle dos sistemas de temperatura poderia ser diminuída com a supervisão de *set points* de vazão e temperatura feita por um operador experiente, que tivesse um certo nível de conhecimento sobre o processo (CHIU, 1998). Esta técnica foi empregada com sucesso pela Yokogawa Electric (YOKOGAWA, 1990) para o controle de temperatura em um forno industrial.

A primeira observação prática, feita na seção anterior, sugere o projeto de uma mecanismo de ajuste para os ganhos dos controladores PID, ajustando adaptativamente os ganhos com base nas respostas em malha fechada das variáveis de interesse. Resultados encontrados na bibliografia indicam que a utilização de supervisores *fuzzy* para contro-

ladores PID no controle de processos de primeira ordem com atraso gera respostas com menor tempo de subida e assentamento e erro em regime nulo (Dotoli et al, 2001). A segunda observação sugere que um operador experiente poderia ajustar os *set points* de temperatura e vazão para chegar mais rapidamente às condições de ensaio. Este operador experiente pode ser substituído por um supervisor *fuzzy* para os *set points*, introduzindo um nível hierárquico superior na malha de controle.

3.3.2 Metodologia de Projeto de um Supervisor *Fuzzy*

Existem (LINHARES, 2010) quatro passos inicias para o projeto de um supervisor *fuzzy*: (a) definição do modelo e das características operacionais, (b) definição dos conjuntos *fuzzy*, (c) definição do comportamento do controle e (d) determinação do método de *defuzzificação*. No projeto de supervisores *fuzzy* é necessário a definição de parâmetros estruturais (fixos) e parâmetros de sintonização (variáveis) (GOMIDE; GUDWIN, 1994).

- Definição do Modelo e das Características Operacionais: Estabelece as particularidades da arquitetura do sistema (como sensores e atuadores) e também define as propriedades operacionais específicas do controlador *fuzzy* em questão.
- Definição dos Conjuntos *fuzzy*: Garante que cada variável seja associada a conjuntos previamente identificados e dentro do domínio de cada variável linguística. De forma a garantir melhor suavidade e estabilidade, cada conjunto *fuzzy* pode se superpor aos conjuntos adjacentes.
- Definição do Comportamento do Controle: Envolve a escrita das regras que atrelam as variáveis de entrada às propriedades de saída do modelo. A base de regras está associada com a estrutura do controlador e a sua construção é a etapa mais difícil e crucial no projeto. O projetista tem de definir que informações devem ser utilizadas no algoritmo de controle, que operações estas informações devem sofrer, e que elementos de saída do processo devem ser obtidos a partir do sistema.
- Determinação do Modelo de defuzzificação: Fornece o valor da ação de controle procurada. Deve-se determinar qual o método mais adequado ao modelo, de forma a se obter uma resposta satisfatória.

Os parâmetros estruturais são:

- número de variáveis de saída;
- número de variáveis de entrada;

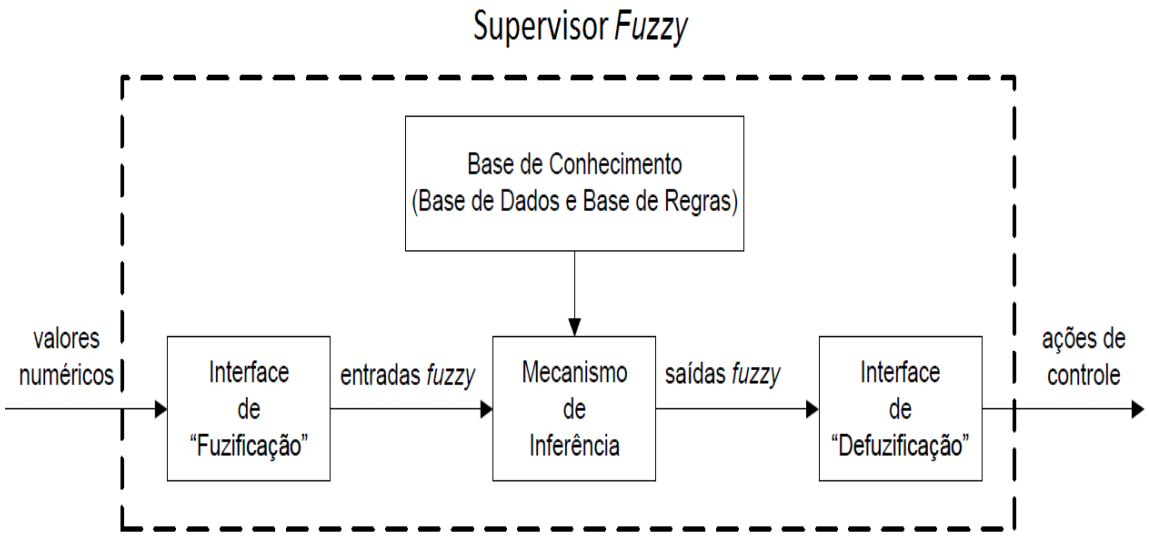

Figura 27 – Estrutura de um supervisor *fuzzy*

Fonte: Adaptada de Linhares, 2010.

- variáveis linguísticas;
- função de pertinência;
- intervalos de discretização e normalização;
- estrutura da base de regras;
- conjunto básico de regras.

Os parâmetros de sintonização são:

- domínio das funções de transferência;
- suporte dos conjuntos *fuzzy*

3.3.3 Projeto de um Supervisor *Fuzzy* para os Controladores PID

O supervisor *fuzzy* deverá alterar os ganhos proporcional (K_p), integral (T_i) e derivativo (T_d) dos controladores PID para as vazões e temperaturas de interesse. O projeto desenvolvido neste trabalho do supervisor *fuzzy* tem por base os trabalhos de (WEI, 2010), (COPELAND; KULDIP, 1994) e (TZAFESTAS; PAPANIKOLOPOULOS, 1990).

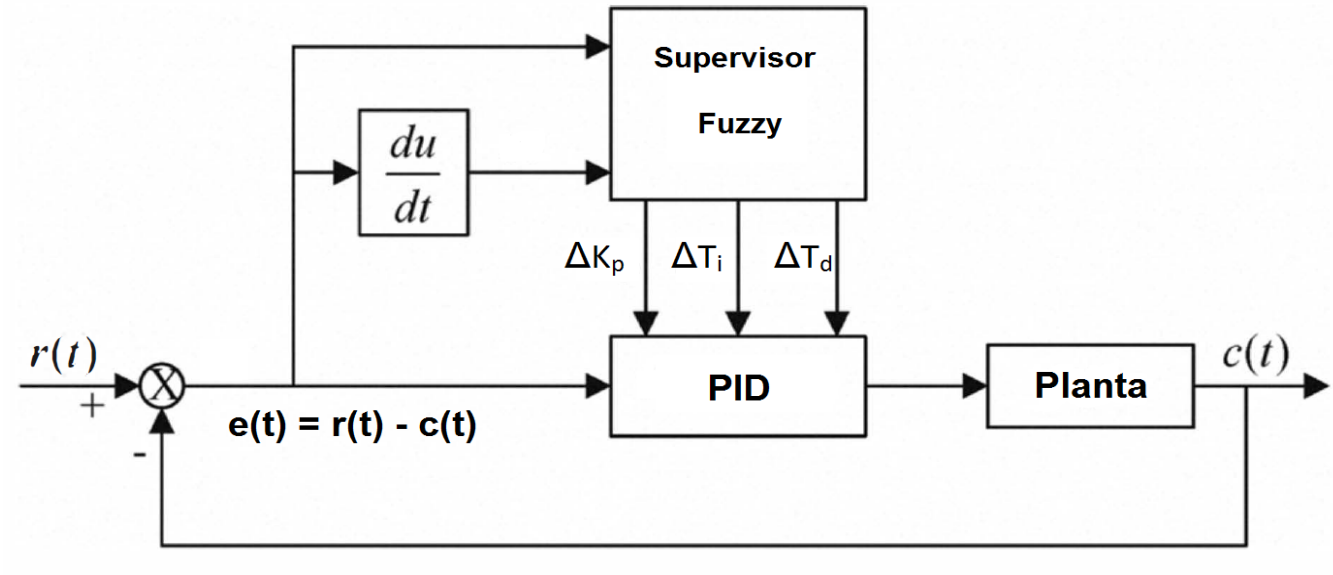

Figura 28 – Supervisor *fuzzy* para PID

3.3.3.1 Entradas e Saídas

As entradas do supervisor serão o erro de acompanhamento ($e(t)$) e a taxa de variação do erro de acompanhamento ($\frac{de(t)}{dt}$). As saídas do supervisor serão as variações (ΔK_p, ΔT_i e ΔT_d) a serem somadas aos ganhos dos controladores a cada iteração.

3.3.3.2 Variáveis Linguísticas e Funções de Pertinência

As variáveis linguísticas possuem valores que são frases na linguagem natural. Esses valores são nomes de conjuntos *fuzzy* em um determinado universo. As variáveis linguísticas utizadas neste trabalho serão: MG(muito grande), G(grande), M (médio), P (pequeno), MP (muito pequeno) e 0 (zero). Essas variáveis linguísticas correspondem, na matriz de controle, aos valores $6, 5, 4, 3, 2, 1$, respectivamente (LEMKE; DE-ZHAO, 1985).

As funções de pertinência são curvas que definem como cada ponto no espaço das entradas é mapeado em um grau de pertinência que varia de 0 a 1. Quanto mais próximo de 1 for o valor da função de pertinência em um conjunto *fuzzy* A para uma variável x qualquer, maior será o grau de pertinência do elemento ao conjunto A. Neste projeto serão utilizadas funções de distribuição gaussiana, pela suavidade e notação concisa.

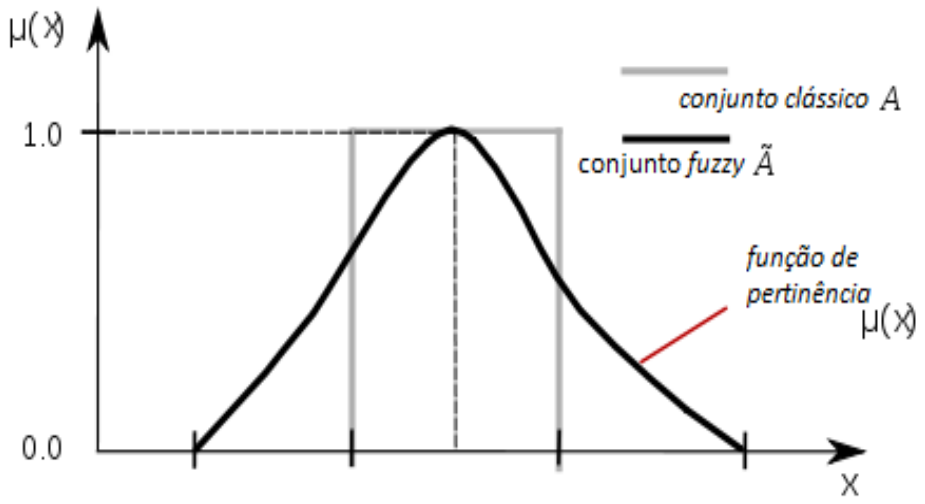

Figura 29 – Função de pertinência gaussiana

3.3.3.3 Estrutura da Base de Regras e Conjunto Básico de Regras

Uma das partes essenciais do supervisor *fuzzy* é o conjunto de regras linguísticas que geram as saídas para os controladores PID. A estrutura básica das regras é uma condicional das duas entradas, gerando uma regra de ajuste para as três saídas:

$$\textit{"Se } e(t) \textit{ for } A \textit{ e } \frac{de(t)}{dt} \textit{ for } B, \textit{ então } \Delta K_p \textit{ é } C, \Delta T_i \textit{ é } D, \Delta T_d \textit{ é } E. \textit{ "}$$

Outra parte essencial do supervisor *fuzzy* é o conjunto básico de regras, ou seja, as ações corretivas que devem ser realizadas sobres os ganhos dos controladores PID com base nas entradas do supervisor. Essas regras representam o conhecimento do operador (ZADEH, 1965). Uma matriz de controle *fuzzy* foi proposta por Macvicar-Whelan (1976) para emular o raciocínio do operador. Esta matriz é baseada nos seguintes princípios:

- Se a saída tem o valor desejado e a derivada do erro é zero, então a saída do controlador deve ser mantida constante;
- Se a saída diverge do valor desejado, a ação então depende do sinais e dos valores do erro e de sua derivada. Se as condições são tais que o erro possa ser corrigido rapidamente por si, então a saída do controlador deve ser mantida constante ou quase constante. Caso contrário, muda-se a saída do controlador para se atingir resultados satisfatórios.

Neste trabalho será a matriz sugerida por (TZAFESTAS; PAPANIKOLOPOULOS, 1990). Os sinais "+"e "−"indicam mudanças positivas e negativas, respectivamente. A figura 30 mostra a matriz de controle *fuzzy*.

ΔE(Xo-X)

	-6	-5	-4	-3	0	-1	-0	+0	+1	+2	+3	+4	+5	+6
-6	-6	-6	-5	-5	-4	-4	-3	-3	-2	-2	-1	-1	0	0
-5	-6	-5	-5	-4	-4	-3	-3	-2	-2	-1	-1	0	0	0
-4	-5	-5	-4	-4	-3	-3	-2	-2	-1	-1	0	0	0	0
-3	-5	-4	-4	-3	-3	-2	-2	-1	-1	0	0	0	+1	+1
-2	-4	-4	-3	-3	-2	-2	-1	-1	0	0	0	+1	+1	+2
-1	-4	-3	-3	-2	-2	-1	-1	0	0	0	+1	+1	+2	+2
-0	-3	-3	-2	-2	-1	-1	0	0	0	+1	+1	+2	+2	+3
+0	-3	-2	-2	-1	-1	0	0	0	+1	+1	+2	+2	+3	+3
+1	-2	-2	-1	-1	0	0	0	+1	+1	+2	+2	+3	+3	+4
+2	-2	-1	-1	0	0	0	+1	+1	+2	+2	+3	+3	+4	+4
+3	-1	-1	0	0	0	+1	+1	+2	+2	+3	+3	+4	+4	+5
+4	-1	0	0	0	+1	+1	+2	+2	+3	+3	+4	+4	+5	+5
+5	0	0	0	+1	+1	+2	+2	+3	+3	+4	+4	+5	+5	+6
+6	0	0	+1	+1	+2	+2	+3	+3	+4	+4	+5	+5	+6	+6

Figura 30 – Matriz de Controle *fuzzy*

Fonte: Papanikolopoulos (1990).

A matriz de controle *fuzzy* considera as seguintes características do controle PID:

- O termo integral é responsável pelo sobressinal, então este deve ser reduzido quando a resposta do sistema ao degrau unitário exceda 1. Por outro lado, um pequeno aumento do termo integral durante a subida na resposta leva a uma diminuição de 10-20% no tempo de subida.

- O termo derivativo é responsável pela suavização das oscilações da resposta ao degrau unitário. Um pequeno aumento no termo derivativo durante a subida e em regime elimina as pequenas oscilações que geralmente ocorrem na resposta.

- Aumentando o termo proporcional, diminui-se o tempo de subida e aumentam-se as oscilações. Então, finalmente, esse termo deve ser diminuído para se evitar comportamento oscilatório.

As regras de mudança dos ganhos dos controladores PID ficam:

$$K_P = K_p + CV\{E, \Delta E\} * k_1 \tag{3.1}$$

$$T_i = T_p + CV\{E, \Delta E\} * k_2 \tag{3.2}$$

$$T_d = T_d + CV\{E, \Delta E\} * k_3 \tag{3.3}$$

em que CV representa o valor defuzzificado da saída guardado na matriz de controle *fuzzy* para e e Δe. Os parâmetros k_1, k_2 e k_3 são importantes no procedimento de ajuste dos controladores PID, já que determinam os limites de variação de cada ganho. Esses limites podem ser determinados considerando diversos critérios de projeto como, por exemplo, o critério de estabilidade de Routh-Hurwitz (ROUTH, 1877), (HURWITZ, 1895). Esses serão os parâmetros de sintonização do supervisor.

3.3.3.4 Intervalos de Discretização e Normalização

Os intervalos de discretização e normalização são fundamentais para a convergência dos ganhos. Os ganhos dos controladores PID serão atualizados simultaneamente, mas serão ajustados por diferentes ganhos (k_1, k_2 e k_3). Estes ganhos definem o intervalo de variação de cada ganho do controlador e são definidos por tentativa e erro durante a simulação.

3.3.4 Projeto de um Supervisor de *Set Points*

O supervisor de *set points* (Figura 31) deverá alterar os *set points* de vazão e temperatura com base na resposta em malha fechada das temperaturas de interesse. O objetivo é reduzir o tempo de subida das temperaturas, chegando mais rapidamente às condições de ensaio. Uma vez atingida a condição de ensaio, o supervisor então irá restaurar os *set points* originais. O projeto deste supervisor é realizado de forma análoga aos supervisores dos controladores PID apresentados na seção anterior.

3.3.4.1 Entradas e Saídas

As entradas do supervisor serão o erro de acompanhamento da temperatura ($e(t)$) e a taxa de variação do erro de acompanhamento ($\frac{de(t)}{dt}$). As saídas do supervisor serão dois ganhos ($K_{SP,V}$ e $K_{SP,T}$) de multiplicação para os *set points* originais.

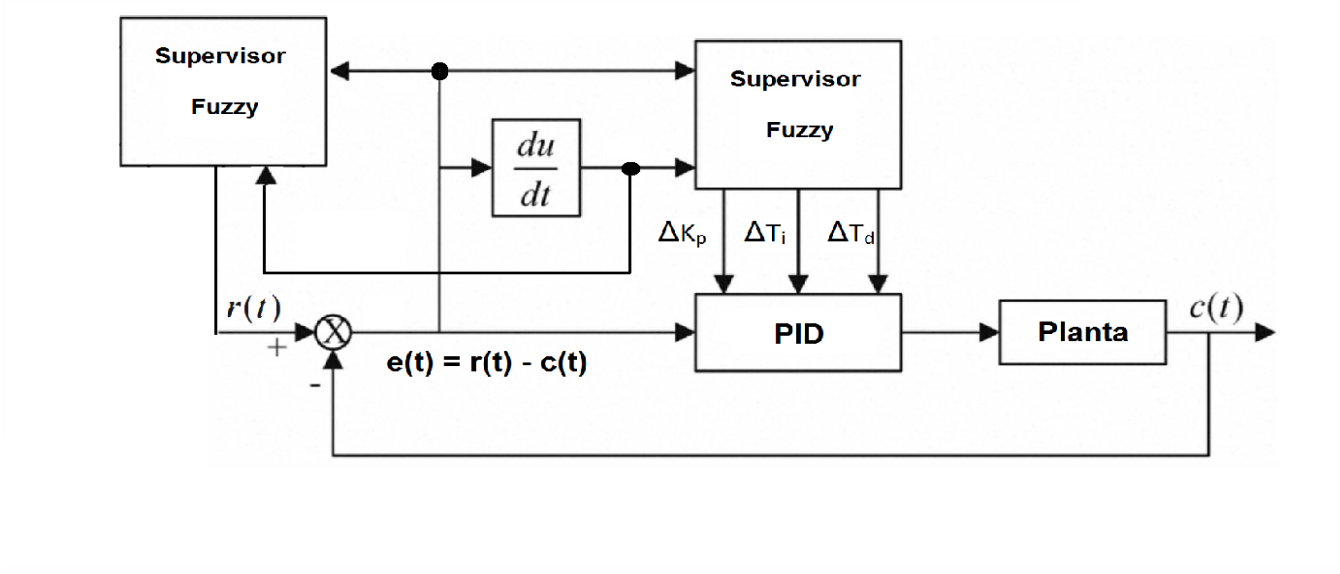

Figura 31 – Supervisor *fuzzy* para *set points*

3.3.4.2 Variáveis Linguísticas e Funções de Pertinência

As variáveis linguísticas e as funções de pertinência serão as mesas utilizadas para o projeto do supervisor *fuzzy* dos controladores PID. A diferença será o intervalo de discretização definido pelos ganhos $K_{SP,V}$ e $K_{SP,T}$.

3.3.4.3 Estrutura da Base de Regras e Conjunto Básico de Regras

A estrutura básicas das regras também será uma condicional, gerando uma regra de ajuste para as duas saídas:

"Se e(t) for A e $\frac{de(t)}{dt}$ *for B, então* $K_{SP,V}$ *é C e* $K_{SP,T}$ *é D.*

O conjunto de regras a serem aplicadas sobre os ganhos para os *set points* é pautado nos mesmos princípios enunciados na seção anterior. A matriz de controle utilizada para o supervisor de *set points* é similar à utilizada para os supervisores dos controladores PID. A mudança de *set points* pode gerar instabilidade na malha interna, logo só será feita para acelerar a resposta do sistema. Quando a temperatura se aproximar do *set point* desejado, o supervisor será desligado.

As regras de mudança dos *set points* ficam:

$$SPV_{artificial} = CV\{E, \Delta E\} * SPV_{original} * K_V \tag{3.4}$$

$$SPT_{artificial} = CV\{E, \Delta E\} * SPT_{original} * K_T \tag{3.5}$$

3.3.4.4 Intervalos de Discretização e Normalização

Os intervalos de discretização e normalização são fundamentais para a alteração dos *set points*. Estes intervalos são definidos pelos ganhos K_V e K_T, que definem o intervalo de variação da vazão e da temperatura.Esses intervalos são limitados pela especificação de operação máxima do sistema: vazão máxima e temperaturas máxima e mínima (ZIEGLER; NICHOLS, 1942).

4 Simulação da Solução Proposta

4.1 Controle de Vazão

A resposta em malha fechada da vazão (Figura 20) apresenta característica típica para aplicação do segundo método de Ziegler-Nichols. A malha utilizada para a sintonia dos controladores dos ventiladores centrífugos é mostrada na Figura 32. Os modelos utilizados para os ventiladores são do tipo Wiener-Hammerstein, cujos parâmetros foram identificados a partir de dados de ensaios no *Mock-up*.

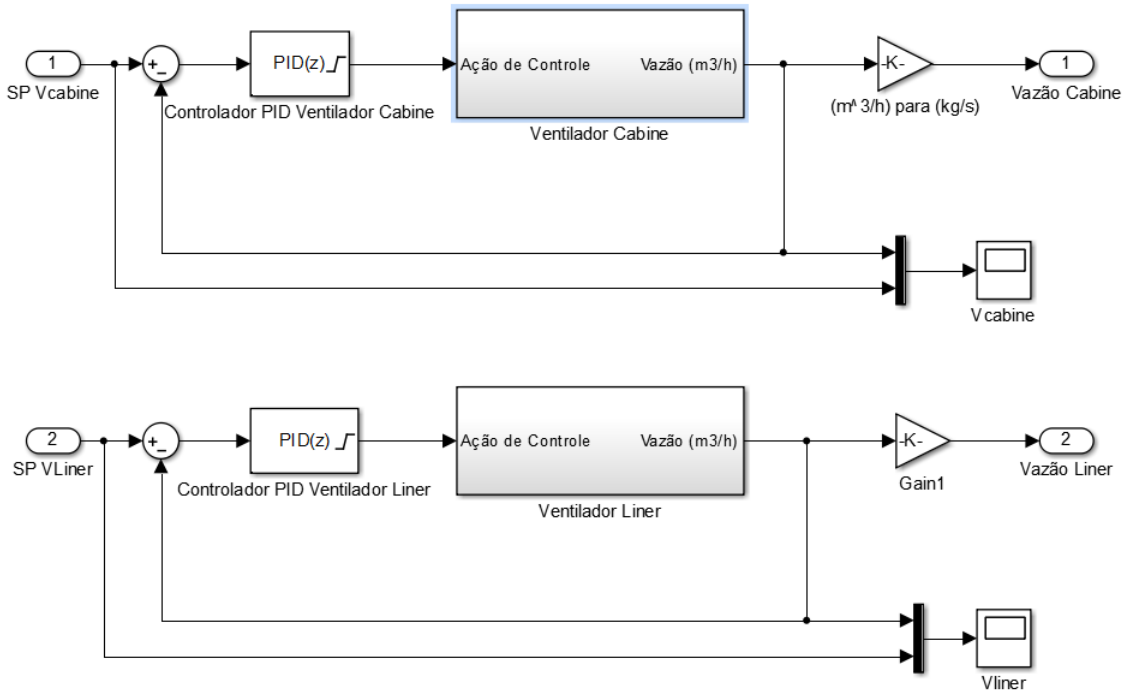

Figura 32 – Malha de Simulação dos Ventiladores

Originalmente o controle de vazão foi sintonizado pelo método da tentativa e erro. No caso do *liner*, não há sensor de vazão instalado na planta. O controle é implementado em malha aberta por uma tabela em que vazão é relacionada com a rotação do ventilador por um polinômio do quarto grau. O sistema de controle de vazão para o ar da cabine apresenta realimentação por um sensor de vazão.

Os gráficos 33 e 34 comparam as respostas dos ventiladores antes e depois da sintonia dos parâmetros pelo segundo método de Ziegler-Nichols.

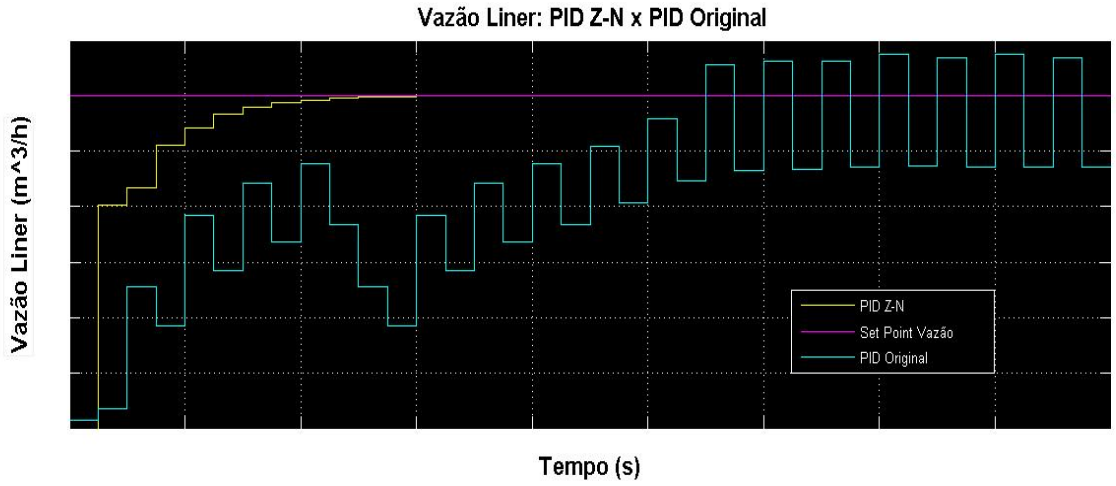

Figura 33 – Gráfico Comparativo para Ventilador Centrífugo do *Liner*

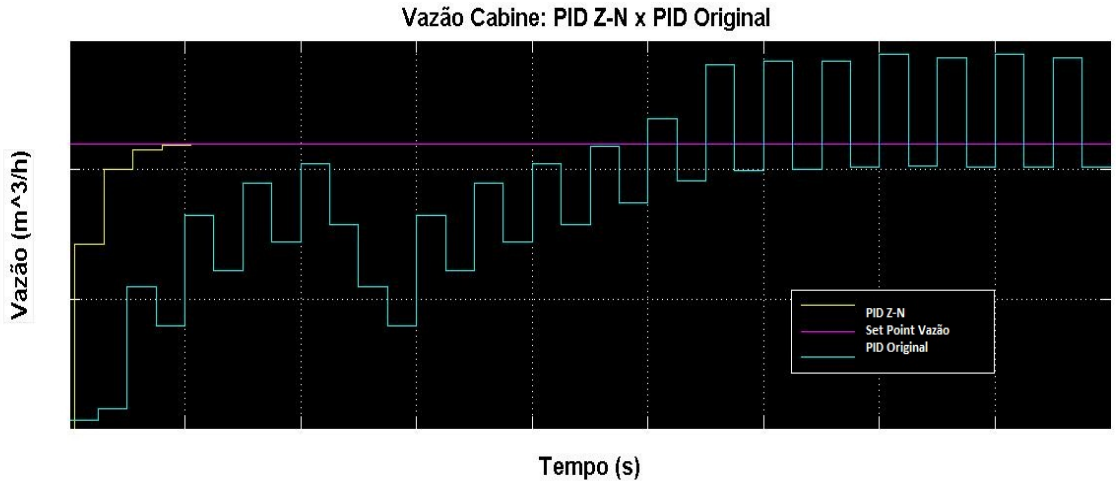

Figura 34 – Gráfico Comparativo para Ventilador Centrífugo da Cabine

O tempo de subida médio para a resposta das vazões dos sitemas *liner* e cabine é 5 segundos, logo a sintonia pelo segundo método de Ziegler-Nichols pode ser feita em tempo relativamente curto. A sintonia dos parâmetros dos controladores PID das malhas de simulação gerou resultados satisfatórios para as respostas das vazões .

4.2 Controle de Temperatura

O controle proposto para a temperatura objetiva a minimização dos tempos de subida e de estabilização, assim como a obtenção de erro em regime nulo. As plantas controladas apresentam parâmetros que variam de acordo com diferentes condições de ensaio (número de tripulantes, número de passageiros, potência elétrica dos equipamentos, coeficiente convectivo entre ar do *liner* e cabine) e ambientais (temperatura do ar externo, radiação solar), assim o supervisor *fuzzy* para os controladores PID de temperatura ajustará adaptativamente os ganhos proporcional, integral e derivativo. O supervisor *fuzzy* dos controladores PID lida com as variações de parâmetros da planta. O supervisor *fuzzy* de *set points* busca minimizar os tempos de subida para as temperaturas de interesse, integrando os controles de temperatura e vazão (COPELAND; KULDIP, 1994). A malha completa do sistema utilizada para a sintonia dos controladores das temperaturas é mostrada na Figura 35.

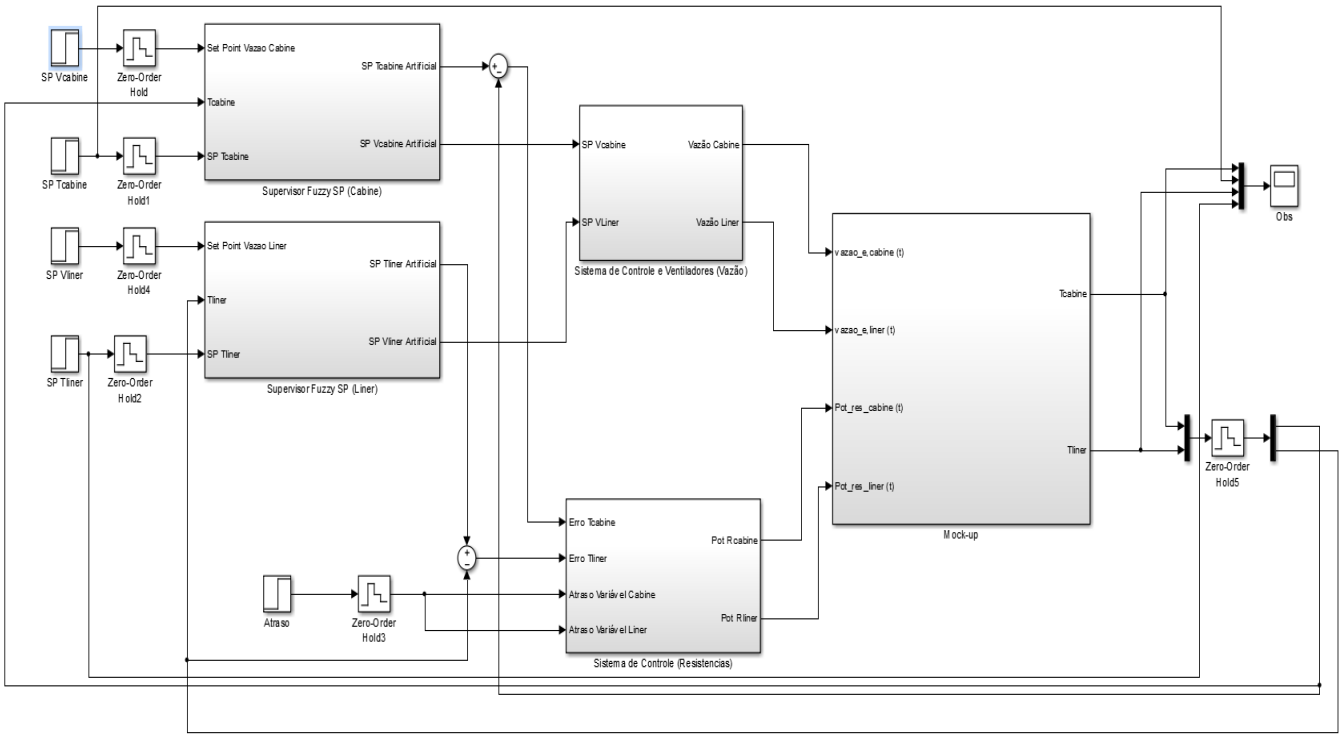

Figura 35 – Malha Completa Utilizada para a Simulação do Controle de Temperatura

A malha do supervisor PID é atualizada a cada 20 segundos, cerca de 4 vezes o tempo de amostragem dos sensores da planta original. A malha do supervisor de *set points* é atualizada a cada 30 segundos.

4.2.1 Supervisor *Fuzzy* para Controladores PID

A figura 36 mostra a malha do supervisor PID, utilizada para atualizar os ganhos dos controladores PID.

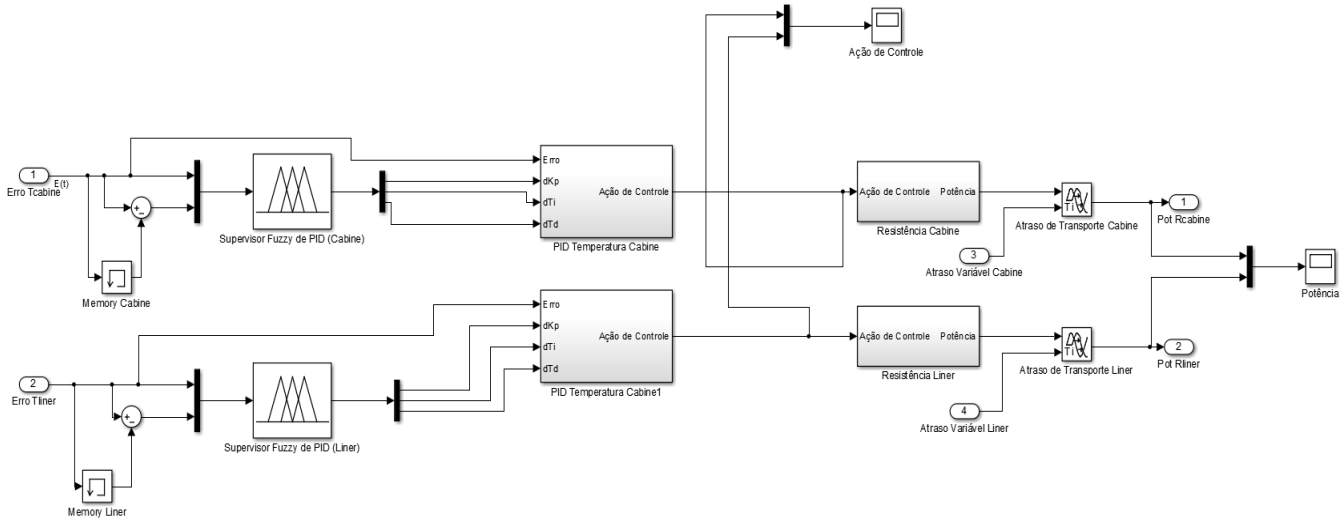

Figura 36 – Universo de Discurso Incremental para o Ganho Proporcional (K_p)

A figura 37 representa o universo de discurso incremental para o ganho proporcional (K_p) do controlador PID.

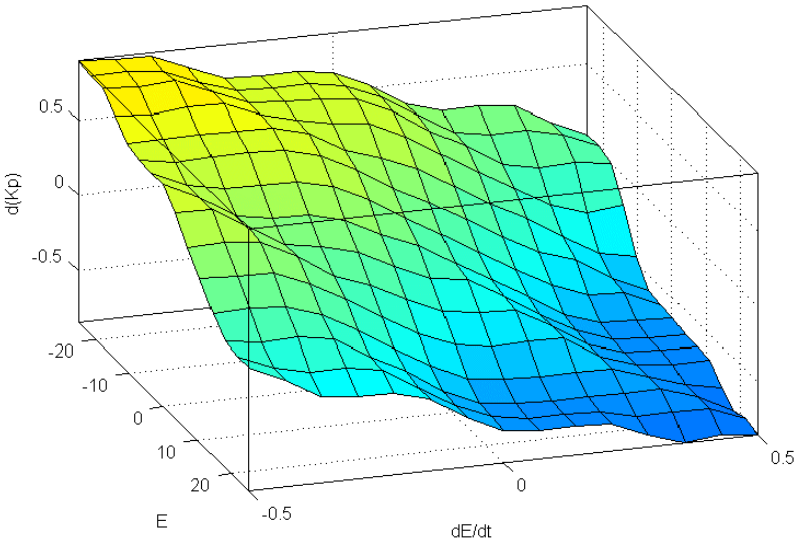

Figura 37 – Universo de Discurso Incremental para o Ganho Proporcional (K_p)

A figura 38 representa o universo de discurso incremental para o ganho integral (T_i) do controlador PID.

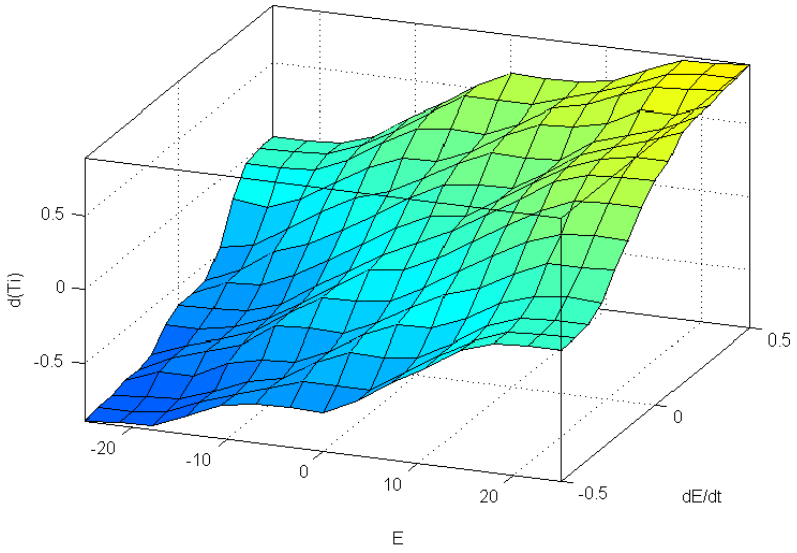

Figura 38 – Universo de Discurso Incremental para o Ganho Integral (T_i)

A figura 39 representa o universo de discurso incremental para o ganho derivativo (T_d) do controlador PID.

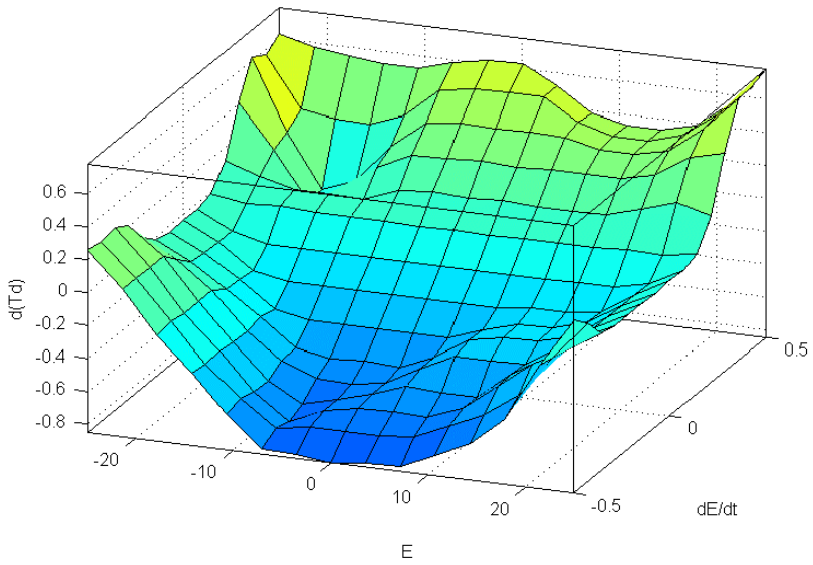

Figura 39 – Universo de Discurso Incremental para o Ganho Derivativo (T_d)

Os valores para os limites de variação dos ganhos são determinados a partir de ganhos iniciais calculados pelo segundo método de Ziegler-Nichols ((MAEDA; MURAKAMI, 1992)) em valores críticos de operação do sistema: pior caso de aquecimento - temperatura externa mínima (14 °C), sem carga térmica de passageiros e equipamentos - e pior caso de refrigeração - temperatura externa máxima (37 °C) com carga térmica de passageiros e equipamentos máxima.

O gráfico 40 compara a resposta para a temperatura da cabine para o controlador

ajustado pelo segundo método de Ziegler-Nichols e pelo supervisor *fuzzy*. Analogamente, o gráfico 41 compara a resposta para a temperatura do *liner*. O resultado da simulação mostra uma melhora significativa em termos de tempo de assentamento (17 minutos) e erro regime (nulo). Os controladores apresentam os mesmos ganhos inicialmente.

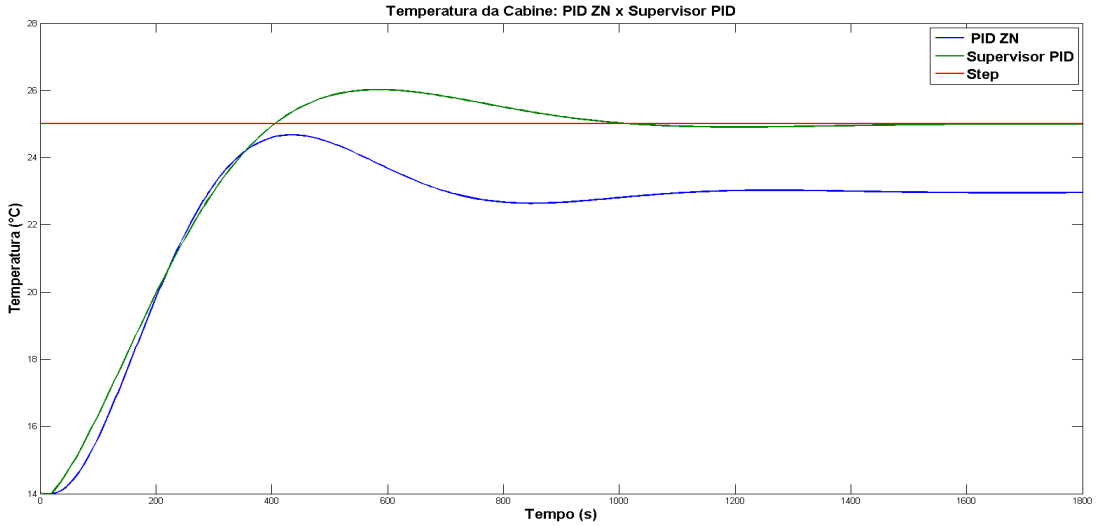

Figura 40 – Gráfico Comparativo para a Temperatura da Cabine

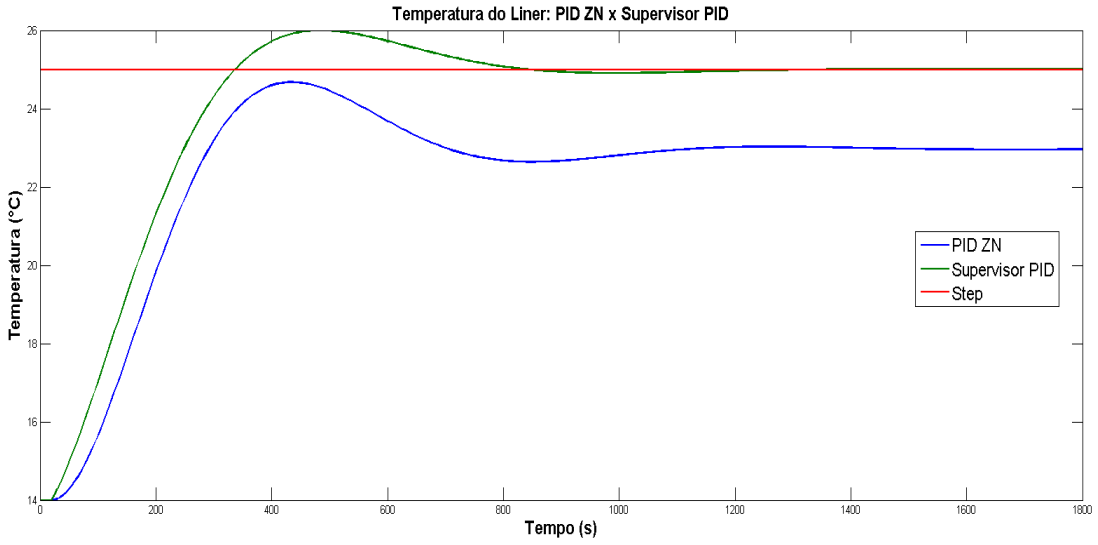

Figura 41 – Gráfico Comparativo para a Temperatura do Liner

4.2.2 Supervisor *Fuzzy* de *Set Points*

A figura 75 representa o universo de discurso para o ganho do *set point* de temperatura.

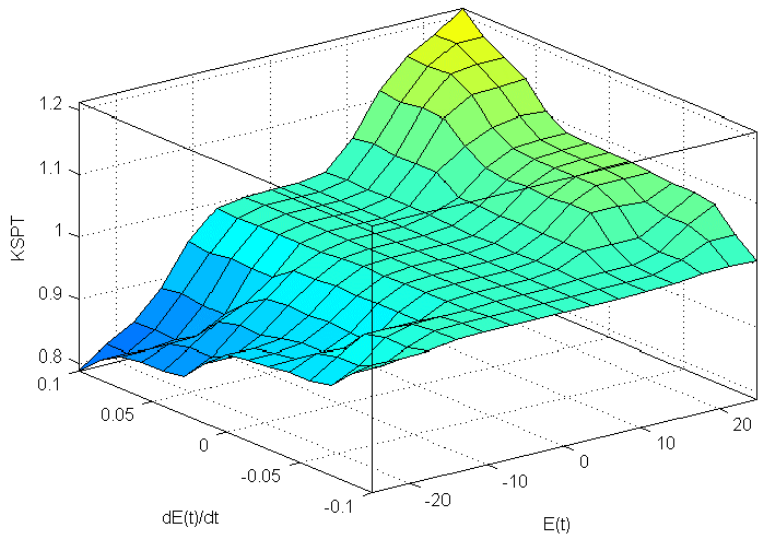

Figura 42 – Universo de Discurso para o Ganho do *Set Point* de Temperatura

A figura 43 representa o universo de discurso para o ganho do *set point* de vazão.

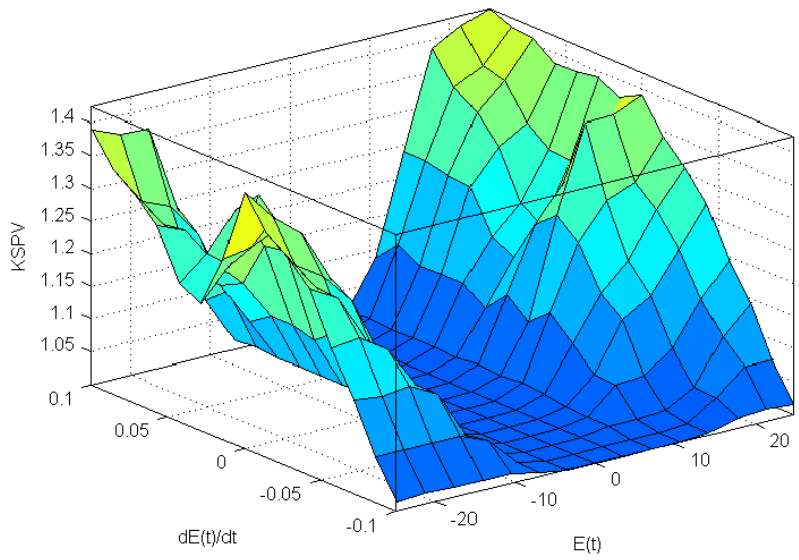

Figura 43 – Universo de Discurso para o Ganho do *Set Point* de Vazão

Seguindo a heurística da matriz de (MACVICAR-WHELAN, 1976), o *set point* de vazão é alterado apenas quando a resposta da temperatura, que representa a variável de interesse para o supervisor, apresenta sinais de lentidão. O *set point* de vazão é modificado para acelerar a resposta de temperatura. No caso de arrefecimento, a maior vazão para os

sistemas aumenta a vazão de ar frio. No caso de aquecimento, o *set point* de temperatura é aumentado, assim como o *set point* de vazão, aumentando a vazão de ar quente para os sistemas *liner* e cabine. O gráfico 44 compara a resposta para a temperatura da cabine com o controlador ajustado pelos três métodos.

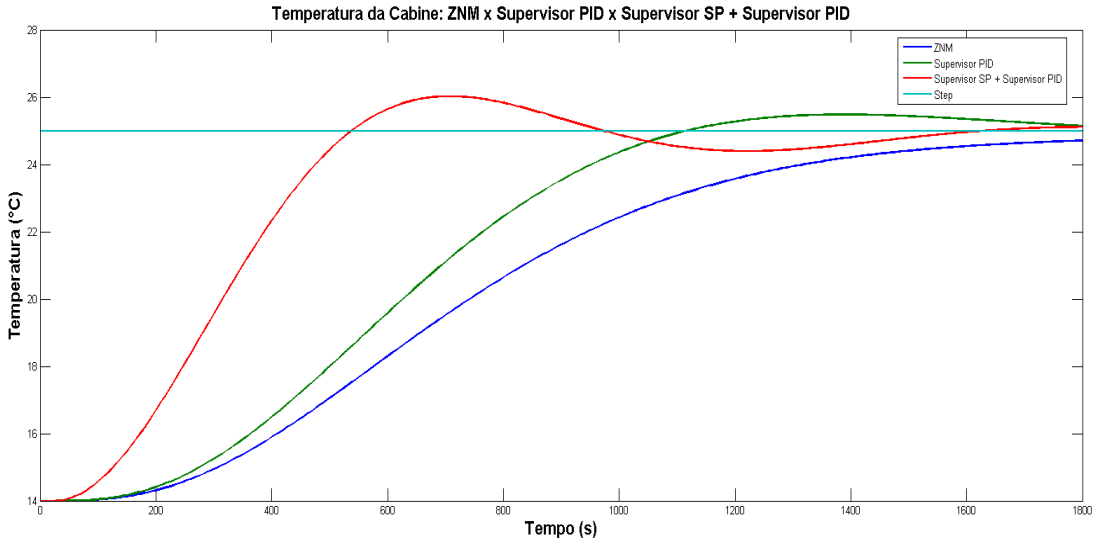

Figura 44 – Gráfico Comparativo para a Temperatura da Cabine

O gráfico 45 compara a resposta para a temperatura do *liner* com o controlador ajustado pelos três métodos.

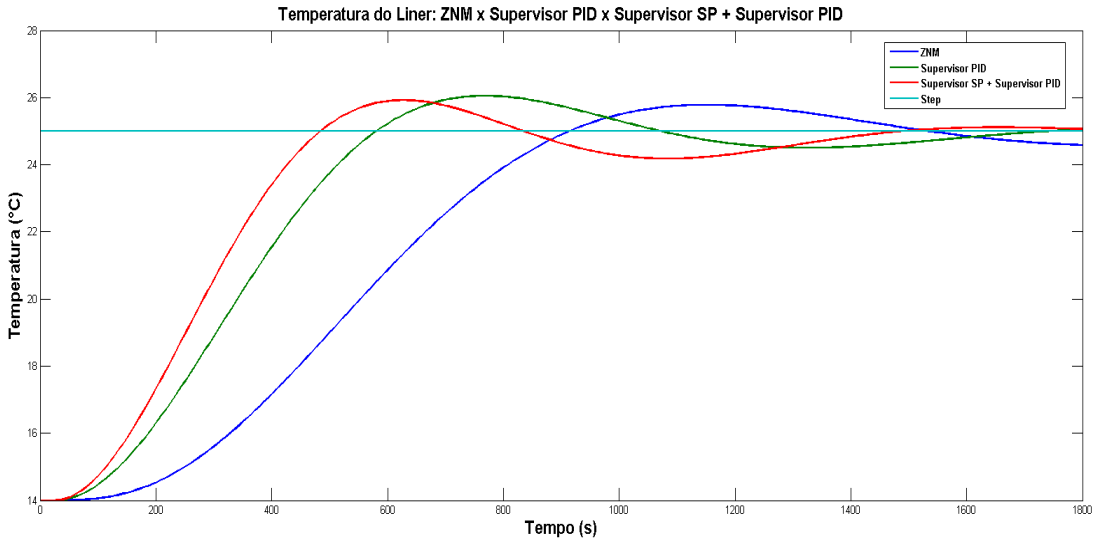

Figura 45 – Gráfico Comparativo para a Temperatura do *Liner*

4.3 Discussão dos Resultados de Simulações

Os resultados obtidos a partir de simulações em Simulink/MATLAB (JANG; GULLEY, 2008) com a solução de controle proposta indicam uma melhoria nas respostas das variáveis de interesse. Em relação ao PID ajustado pelo segundo método de Z-N, o supervisor de PID apresentou uma resposta mais rápida e com erro nulo em regime, apesar de ter apresentado maior sobressinal. A adição de um supervisor de *set points* tornou a resposta do sistema mais rápida e com erro nulo em regime, apesar de também ter apresentado um aumento no máximo sobressinal. O maior problema da implementação de supervisores consiste na determinação do tempo de amostragem das malhas. A malha externa (supervisor de *set points*) precisa ser mais lenta que a malha interna (supervisor do PID).

5 Protótipo

Neste capítulo será apresentado o projeto de um protótipo de *mock-up*, construído para a realização de testes práticos da solução proposta. A construção de um protótipo em miniatura reduz drasticamente os tempos de ensaio, em razão da menor inércia térmica do sistema. Os resultados dos testes práticos com o protótipo serão úteis para a implementação do controle proposto na planta original. O protótipo proposto mantém as principais características de controle da planta original, de modo a, por meio da realização de testes, validar a solução proposta.

5.1 Objetivos e Critérios de Projeto

O objetivo principal da construção do protótipo é o teste da solução de controle proposta. São feitas, portanto, simplificações construtivas de modo a simplificar a construção do protótipo, mantendo as principais características da planta original, importantes do ponto de vista de controle: não linearidades dos ventiladores, atrasos de atuação e variação controlada de parâmetros da planta (carga térmica de equipamentos e passageiros). Deseja-se, portanto, manter o tipo de atuação sobre a planta original, assim como a natureza das trocas térmicas entre os sistemas *liner* e cabine. As características desejadas para o protótipo dos sistemas cabine e *liner* são:

- Dimensões máximas: 450 x 220 x 280 (mm);
- Controle de temperatura pelo aquecimento do ar de entrada;
- Controle de vazão com ventiladores axiais;
- Superfície de interface entre *liner* e cabine, simulando a parede da cabine;
- Razão entre carga térmica variável dentro do ambiente da cabine - simulando o calor dissipado pelos equipamentos elétricos e pelos passageiros e tripulantes - pela massa de ar contida na cabine deve ser mantida constante;
- O protótipo deve ter o mesmo número de sensores de temperatura e vazão da planta original;
- O controle deve apresentar interface com Simulink/MATLAB.

Optou-se pela utilização de materiais acessíveis e leves, como placas de MDF e chapas de alumínio. Foram utilizados sensores, microcontroladores e componentes eletrô-

nicos de dimensões reduzidas a fim de minimizar a massa final da estrutura. As dimensões máximas escolhidas mantém uma razão de redução em relação à planta original.

O controle de temperatura deve ser realizado de forma análoga ao controle realizado na planta original: aquecimento do ar de uma fonte fria por meio de potência dissipada por um conjunto de resistores.

O controle de vazão deve ser feito por meio de ventiladores axiais (*coolers*) utilizados em computadores comerciais.

A superfície de separação entre as regiões do *liner* e da cabine será fabricada a partir de uma chapa de alumínio. Este elemento é importante para se manter a analogia da natureza das trocas térmicas entre a parede e o ar da cabine.

A carga térmica variável dentro da cabine será obtida por meio da instalação de elementos elétricos ativos, que dissipam calor por efeito Joule.

Os modelos simulados foram desenvolvidos em Simulink/MATLAB. Assim, optou-se por aproveitar a versatilidade da *toolbox* de lógica *fuzzy* do software MATLAB e implementar o controle por meio de um PC.

5.2 Projeto

5.2.1 Sistema *Liner* + Cabine

Optou-se pela utilização de placas de MDF de 16 mm de espessura para a confecção dos componentes do sistema *liner* + cabine. A madeira é um isolante térmico, mantendo os sistemas isolados do ambiente. Cantoneiras de alumínio foram escolhidas de modo a garantir uma melhor fixação das laterais de madeira, evitando possíveis trincas no MDF. As cantoneiras também permitem que o protótipo possa ser desmontado várias vezes para o ajuste dos sensores de temperatura, sem danificar as peças de madeira. As peças de alumínio (paredes da cabine e cantoneiras) foram fabricadas a partir de chapas de alumínio de 2 mm. Duas lâmpadas incandescentes de 7 W foram instaladas dentro da região da cabine para simular o calor dissipado por passageiros, tripulantes e equipamentos elétricos dentro da planta original do experimento, mantendo a proporção carga térmica por massa de ar na cabine. A massa de ar na cabine foi calculada considerando o ar como um gás perfeito a 25 °C e 1 atm. A carga térmica total no pior caso de refrigeração foi calculada e apresentada no documento de especificação (EMBRAER/FAPESP, 2009).

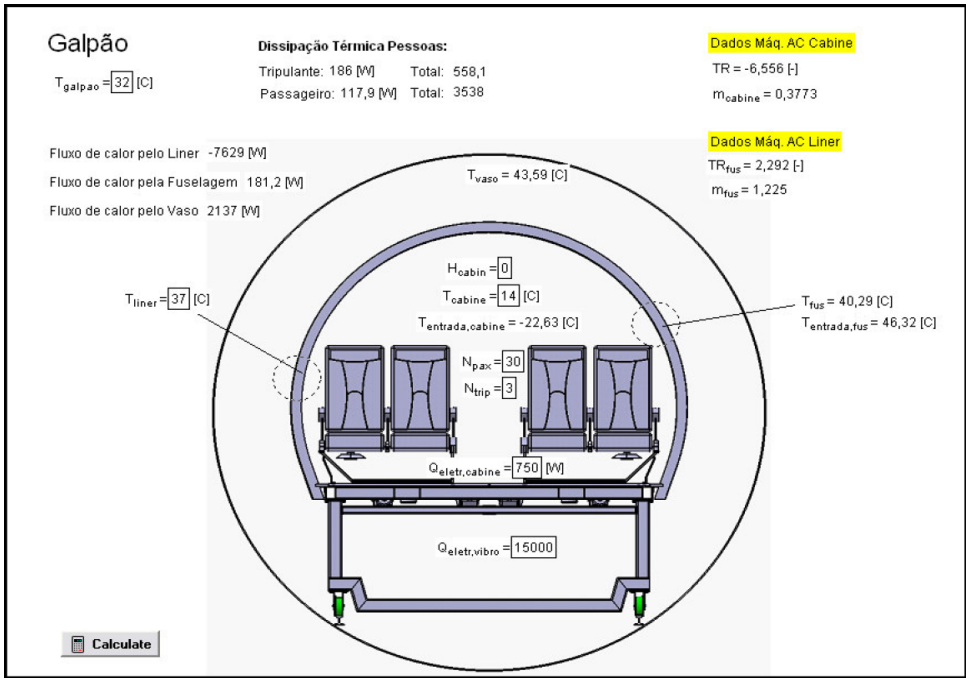

Figura 46 – Modelo matemático do *mock up* configurado para verão, pior caso de refrigeração.

Fonte: Especificação do Sistema de Condicionamento Térmico do Mock-Up Integrado, 2009.

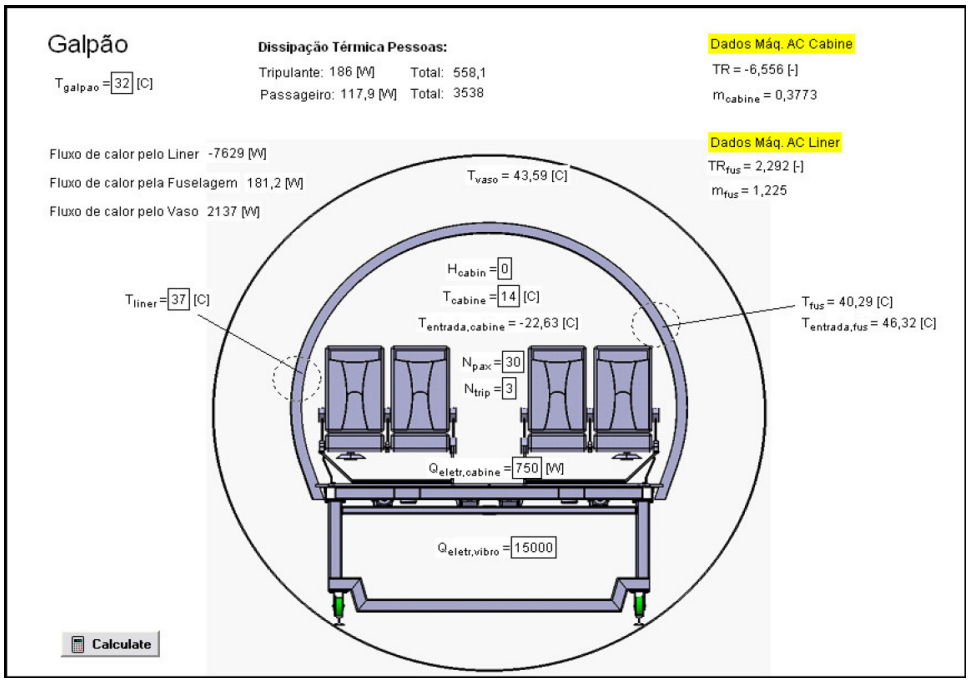

Figura 47 – Modelo matemático do *mock up* configurado para verão, pior caso de refrigeração.

Fonte: Especificação do Sistema de Condicionamento Térmico do Mock-Up Integrado, 2009.

O ar entra para a cabine por meio de um tubo de PVC de 1 polegada de diâmetro, instalado no teto da região da cabine e sustentado por uma braçadeira de alumínio. O ar,

de acordo com o documento de especificação, deve ser distribuído igualmente pela região interna da cabine. Para isso, duas fileiras de furos de 3 mm foram realizados ao longo do comprimento Z do tubo de PVC. As fileiras estão alinhadas a 45° do eixo central X do tubo.

Figura 48 – Região da Cabine

5.2.1.1 Sensores de Temperatura

Para o protótipo, foram utilizados os sensores de temperatura DHT22 modelo AM2303 (49). As principais características deste sensor são:

Figura 49 – Sensor de Temperatura DHT22

- Baixo custo;
- Baixo consumo de energia;
- Estabilidade em longo prazo;

- Acurácia adequada;
- Fácil instalação.

Modelo	AM2303
Tensão de Alimentação	3.3-6 V DC
Sinal de Saída	Sinal Digital *Single-Bus*
Faixa de Operação	-40 a $125\,^{\circ}\mathrm{C}$
Acurácia	$+-0,2\,^{\circ}\mathrm{C}$
Tempo de Medição	2 s

Fonte: Adaptado de AM2303 Aosong Electronics, 2011.

Os sensores foram instalados sobre a parede do *liner*, com um calço de poliacetato de vinila entre as superfícies para evitar o contato direto do sensor com o metal. No total foram utilizados 9 sensores de temperatura:

- 2 sensores instalados na região central interna da cabine;
- 4 sensores instalados no sistema *liner*, sendo um em cada lado da entrada do ventilador;
- 2 sensores instalados nas caixas de aquecimento;
- 1 sensor instalado na parte exterior do protótipo para medir a temperatura do ar ambiente.

5.2.1.2 Ventiladores

Para o protótipo, foram utilizados os ventiladores axiais (*coolers*) *SickleFlow* 120 do fabricante *Cooler Master* (figura 50). Os ventiladores apresentam vazão nominal máxima de 118,4 m^3/h e rotação nominal máxima de 2.000 rpm. A redução dos ventiladores foi feita utilizando a regra de similaridade (ALLEN, 2006) apresentada abaixo:

$$\frac{Q_1}{w_1 D_1^3} = \frac{Q_2}{w_2 D_2^3} \tag{5.1}$$

em que

- Q é a vazão nominal;
- w é a rotação nominal;
- D é o diâmetro das pás.

O modelo escolhido apresenta um pino PWM, por meio do qual será controlada a rotação do ventilador. O ventilador é instalado ao protótipo por meio de um acoplamento parafusado às laterais do conjunto.

Figura 50 – Ventiladores Axiais *SickleFlow*

5.2.2 Sistema de Aquecimento do Ar

No experimento *Mock-up* o controle de temperatura é feito por meio do aquecimento do ar de entrada proveniente dos sistemas de condicionamento de ar. No protótipo, analogamente, o controle de temperatura é feito por meio do controle da potência dissipada por um conjunto de resistores imersos no ar de entrada. No experimento original o ar de entrada pode atingir temperaturas de até 7°C. No protótipo o ar de entrada está disponível a temperatura ambiente.

5.2.2.1 Conjunto de Resistores

Optou-se por resistências de imersão para substituir o conjunto de resistores utilizados na planta original. Para a construção do protótipo, foram utilizados aquecedores elétricos de água comerciais (51). Para cada sistema (*liner* e cabine) foi utilizado um aquecedor de 1.000 W - 127 V do fabricante HD. A potência elétrica enviada aos aquecedores é controlada por meio de um pulso PWM. O *duty cycle* da onda é controlado pelo microcontrolador que chaveia um relé de estado sólido.

Figura 51 – Aquecedor Elétrico

5.2.2.2 Tubulação e Caixa de Aquecimento

Para a tubulação foram utilizados tubos e conexões de PVC com diâmetro nominal de 100 mm. O diâmetro e o comprimento da tubulação dos sistemas liner e cabine foram determinados a partir do atraso de transporte causado pela tubulação:

$$\theta = \frac{L * \pi * D^2}{4 * Q * \rho} \tag{5.2}$$

em que

- L é o comprimento da tubulação;
- D é o diâmetro da tubulação;
- Q é a vazão mássica de ar;
- ρ é a densidade do ar a 25°C.

As caixas de aquecimento foram projetadas para abrigar os aquecedores elétricos. Optou-se pela utilização de painéis de MDF de 16 mm de espessura, revestidos internamente por fita adesiva isolante térmica.

Figura 52 – Foto da Parte Frontal da Caixa de Aquecimento

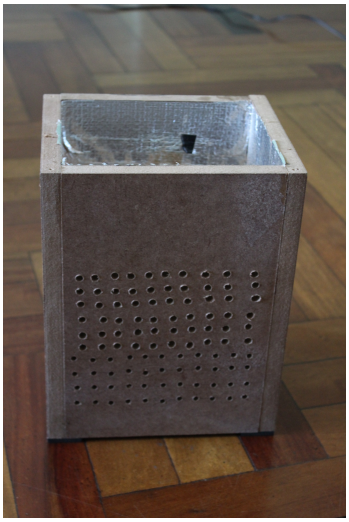

Figura 53 – Foto da Parte Traseira da Caixa de Aquecimento

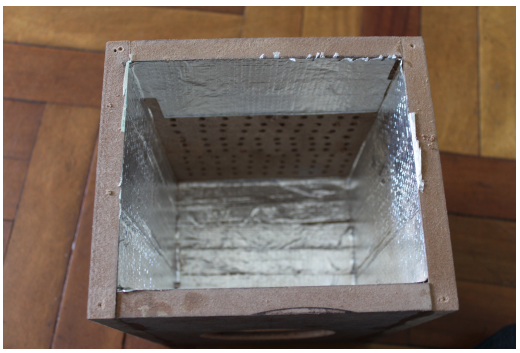

Figura 54 – Foto da Parte Superior da Caixa de Aquecimento

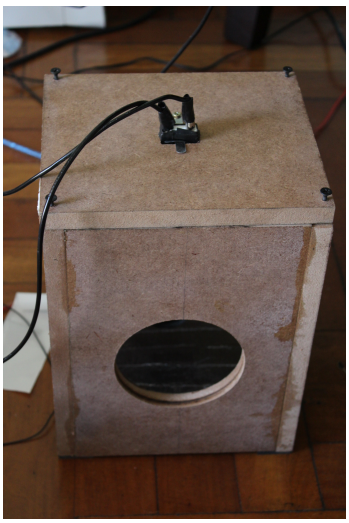

Figura 55 – Foto da Caixa de Aquecimento Montada

5.2.2.3 Sensores de Vazão

Para o protótipo, optou-se pela utilização do sensor FS5 56 do tipo *thermal mass flow*. Este sensor é utilizado para gases e funciona pelo mesmo princípio do sensor utilizado na planta original (FT2). Foram instalados dois sensores desse tipo: um na saída de ar da caixa de aquecimento do sistema *liner* e outro na saída de ar da caixa de aquecimento do sistema cabine.

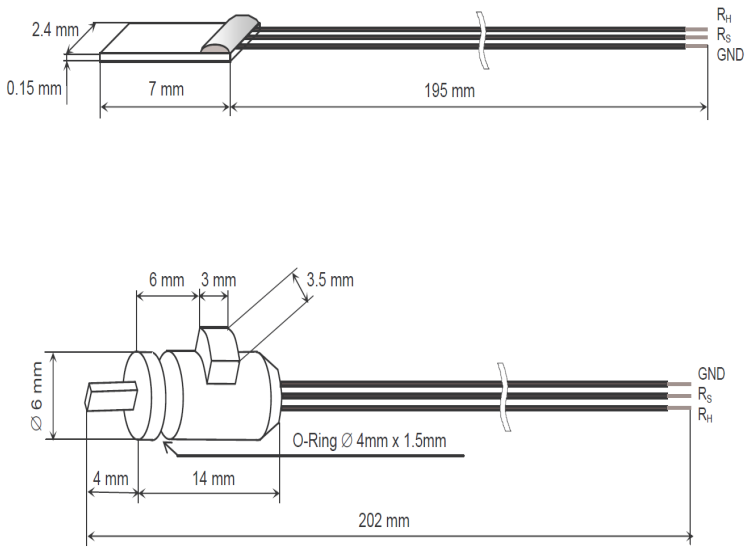

Figura 56 – Sensor de Vazão Mássica

5.2.3 Projeto Elétrico

Dentro do escopo do projeto elétrico estão:

- Microcontrolador;
- Projeto de circuitos para os sensores de temperatura e vazão;
- Projeto de circuitos de controle da temperatura do ar;
- Projeto de circuitos de controle da vazão de ar;
- Projeto de circuito de controle da potência dissipada pelas lâmpadas.

5.2.3.1 Microcontrolador

Optou-se pelo Arduíno MEGA 2560 como microcontrolador. O MEGA é compatível com a grande maioria de *shields* projetados para o Arduíno e apresenta biblioteca disponível para MATLAB. O MEGA 2560 possui 54 portas digitais (entrada/saída) - 14 dos quais podem ser utilizadas como PWM -, 16 entradas analógicas, 4 UART's e oscilador de cristal de 16 MHz.

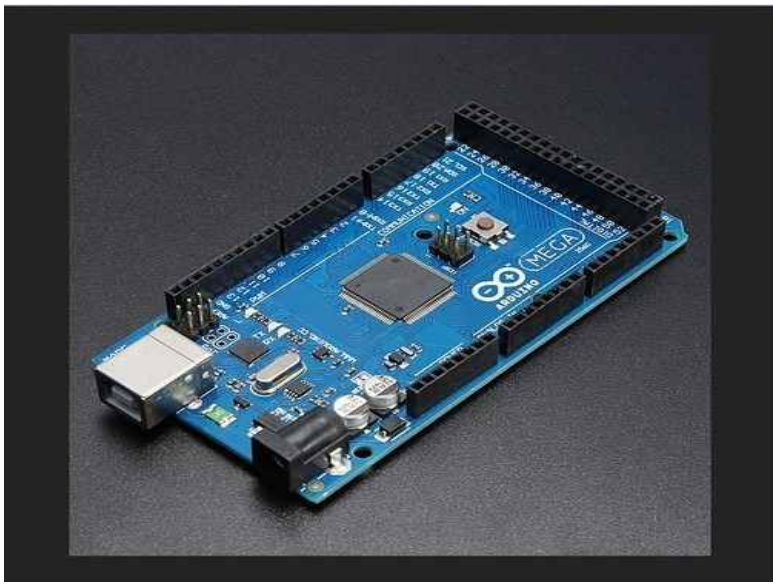

Figura 57 – Arduíno MEGA2560

5.2.3.2 Placas dos Sensores

Os sensores de temperatura DHT22 não exigem a confecção de placas e podem ser diretamente conectados ao Arduíno MEGA2560. Os sensores de vazão utilizados necessitam de amplificadores operacionais, sendo alimentados por uma fonte de 12 V. A figura 59 mostra o circuito projetado para os sensores de vazão.

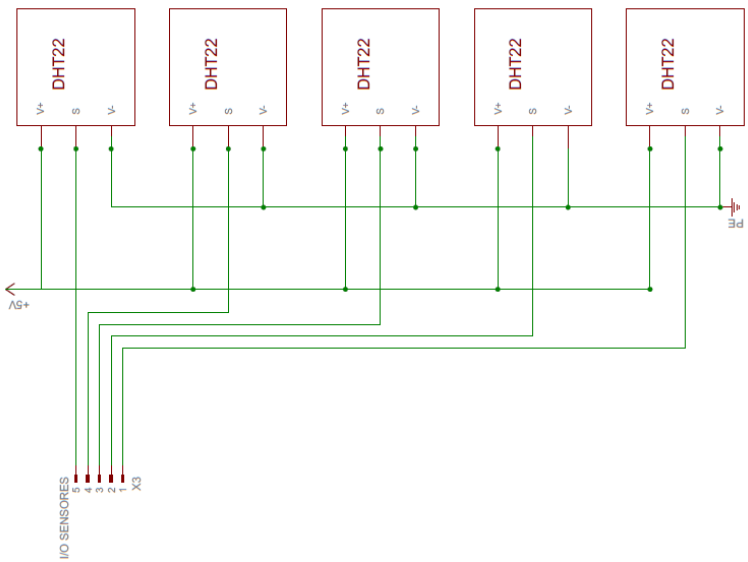

Figura 58 – Circuito para o Sensor de Temperatura

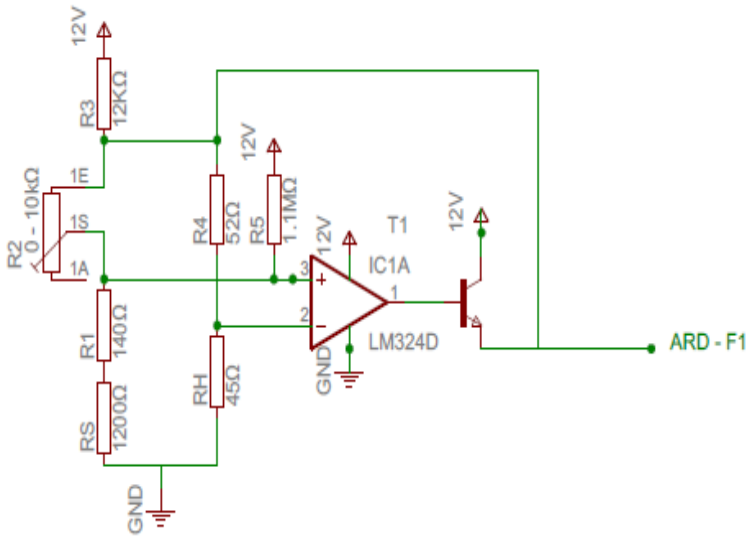

Figura 59 – Circuito para o Sensor de Vazão

5.2.3.3 Controle da Temperatura do Ar

A potência dissipada pelos aquecedores elétricos é controlada por um pulso digital PWM enviado ao relé pelo microcontrolador. O modelo de relé utilizado é o 4815Z e é alimentado por uma tensão de 127 V. O relé é chaveado pelo sinal PWM enviado pelo microcontrolador. A potência dissipada é linear em relação ao valor do *duty cycle*, conforme indicado no gráfico 61.

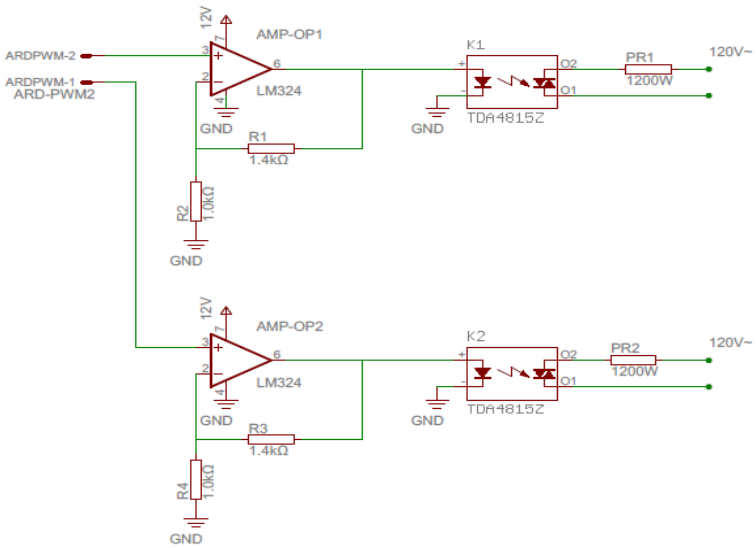

Figura 60 – Circuito do Aquecedor

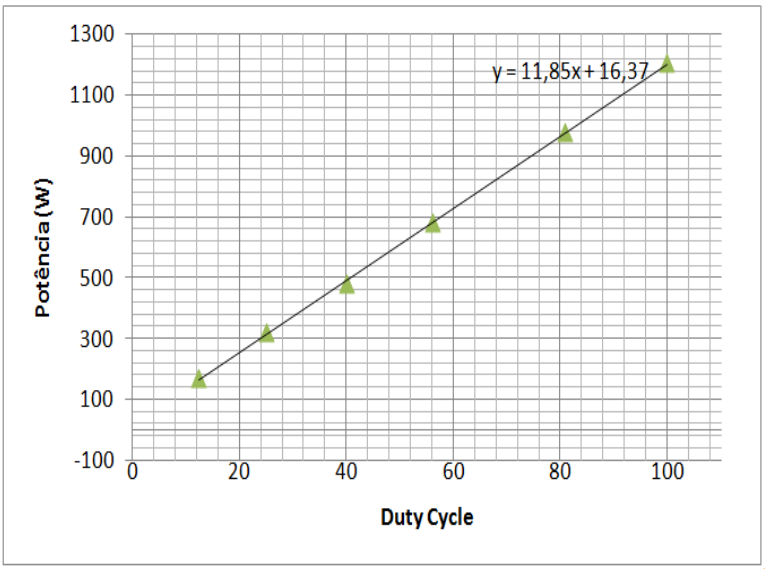

Figura 61 – Potência x *Duty Cycle*

5.2.3.4 Controle da Vazão de Ar

O controle da vazão de ar para os sistemas *liner* e cabine é realizado por meio do controle da rotação dos ventiladores axiais. O controle de rotação é feito por um pulso PWM digital enviado pelo microcontrolador ao ventilador. Os ventiladores são alimentados por uma fonte de 12 V.

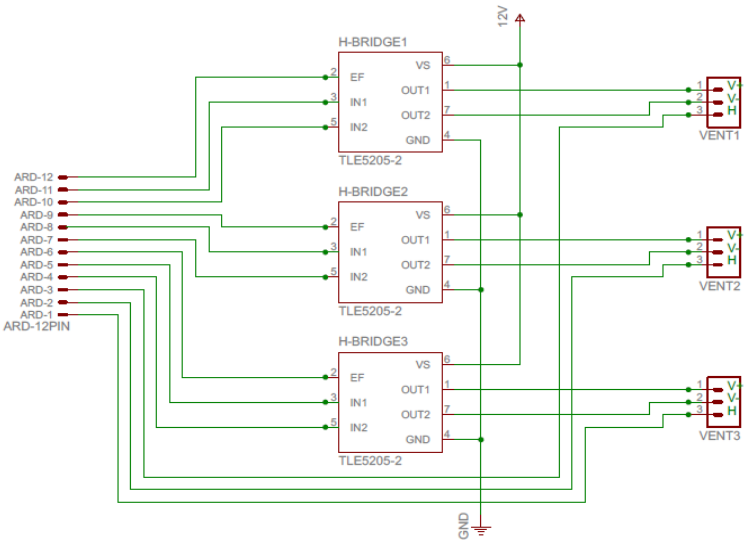

Figura 62 – Circuito do Ventilador

5.2.3.5 Controle da Potência Dissipada pelas Lâmpadas

As lâmpadas foram instaladas na região da cabine do protótipo para simular a carga térmica dissipada por equipamentos, passageiros e tripulantes. Deseja-se que a carga seja

variável para a realização do testes de controle em diferentes situações. Assim, optou-se por um simples *dimmer* ligado a duas lâmpadas de 7 W ligadas em paralelo.

5.2.4 Protótipo Final

As fotos abaixo mostram o protótipo montado.

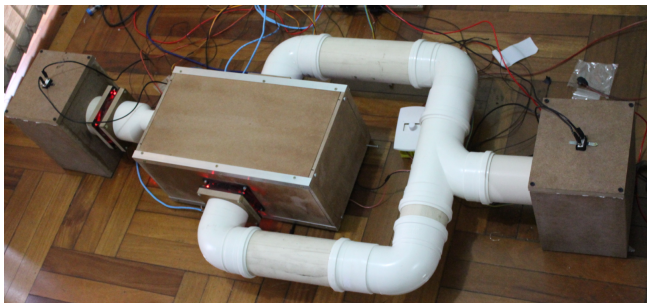

Figura 63 – Vista Superior do Protótipo Montado

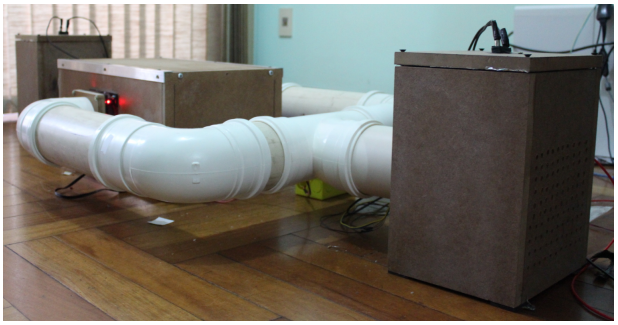

Figura 64 – Vista Frontal do Protótipo Montado

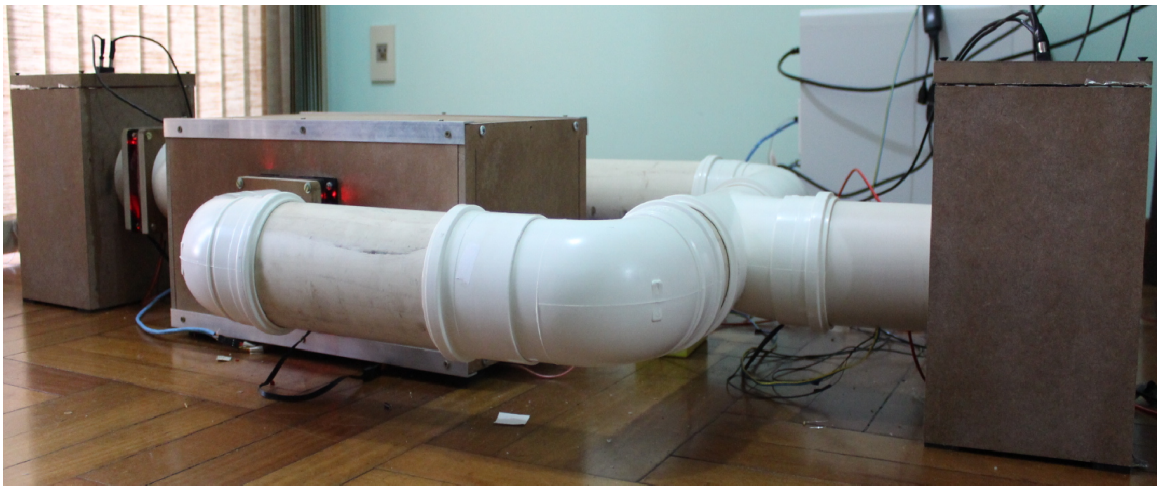

Figura 65 – Circuito do Ventilador

6 Resultados e Conclusões

Neste capítulo serão apresentados o resultados obtidos com a solução proposta aplicada ao protótipo construído, assim como a proposição de modificações sobre a planta atual e as principais conclusões do presente trabalho.

6.1 Controle de Vazão

Para o protótipo, os controladores PID utilizados para os ventiladores foram implementados diretamente no microcontrolador. O Arduíno possui um bloco PID discreto que tem como saída um pulso PWM que controla a rotação dos ventiladores. Os gráficos 68 e 69 mostram um comparativo para a resposta com os controladores sem ajuste e depois de sintonizados com a mudança dos *set points*. Os ganhos foram otimizados por meio de uma rotina de tentativa e erro, similar a descrita por (OGATA, 2011)[1].

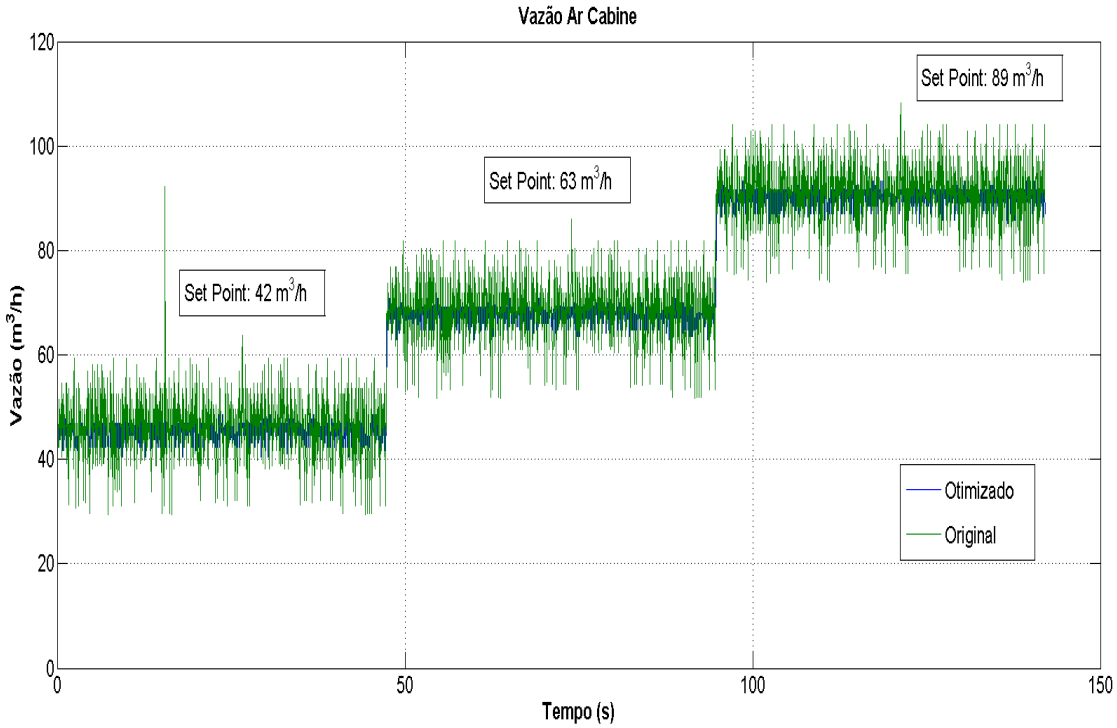

Figura 66 – Resposta da Vazão para o Ar da Cabine

[1] Engenharia de Controle Moderno página 580

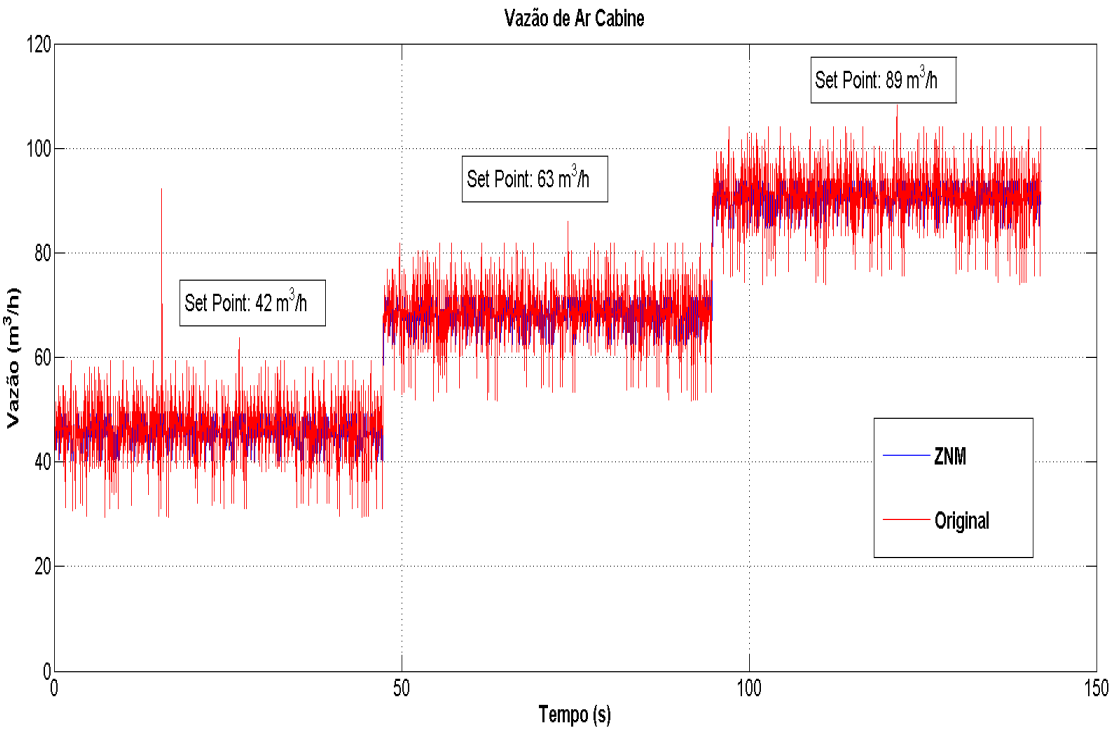

Figura 67 – Resposta da Vazão para o Ar do Liner

A leitura dos medidores de vazão apresenta alguns picos, apesar do filtro aplicado, devido a irregularidades na tubulação. Em ambos os casos, há uma clara melhoria na qualidade da resposta. A tabela indica os ganhos obtidos com os diferentes métodos de ajuste.

	Original	Ziegler-Nichols	Otimizada
K_p	1	3,60	3,12
T_i	1	0,45	0,78
T_d	1	0,12	0,61

Ganhos para o Ventilador da Cabine

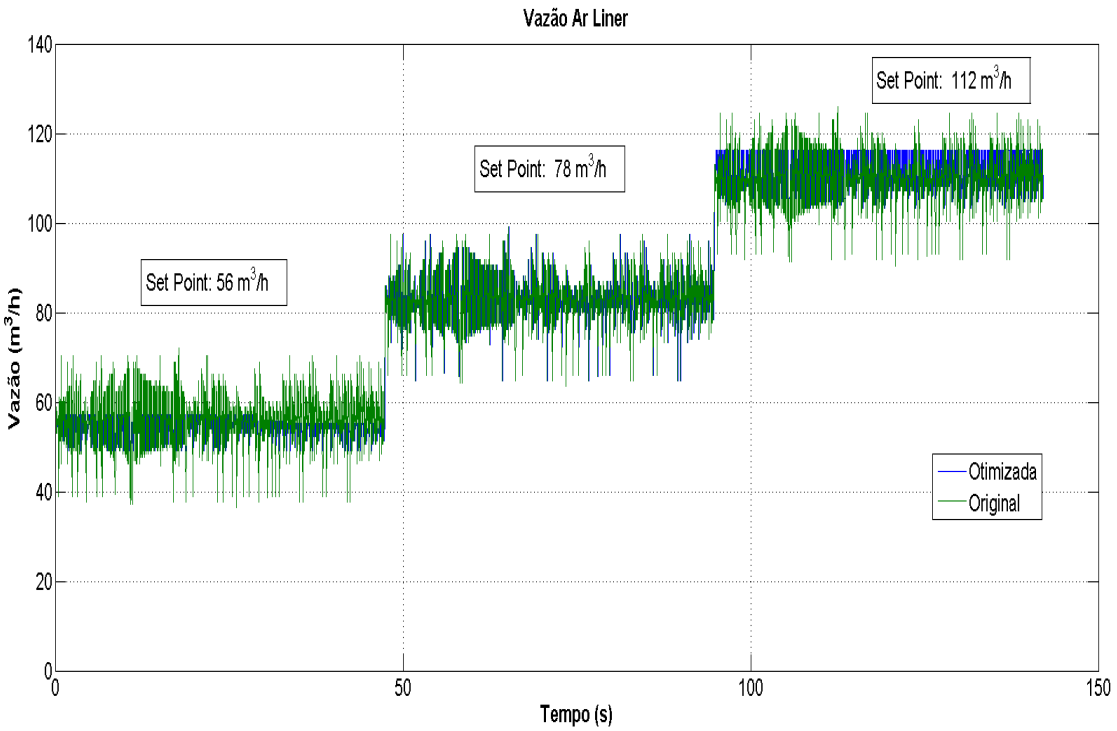

Figura 68 – Resposta da Vazão para o Ar do *Liner*

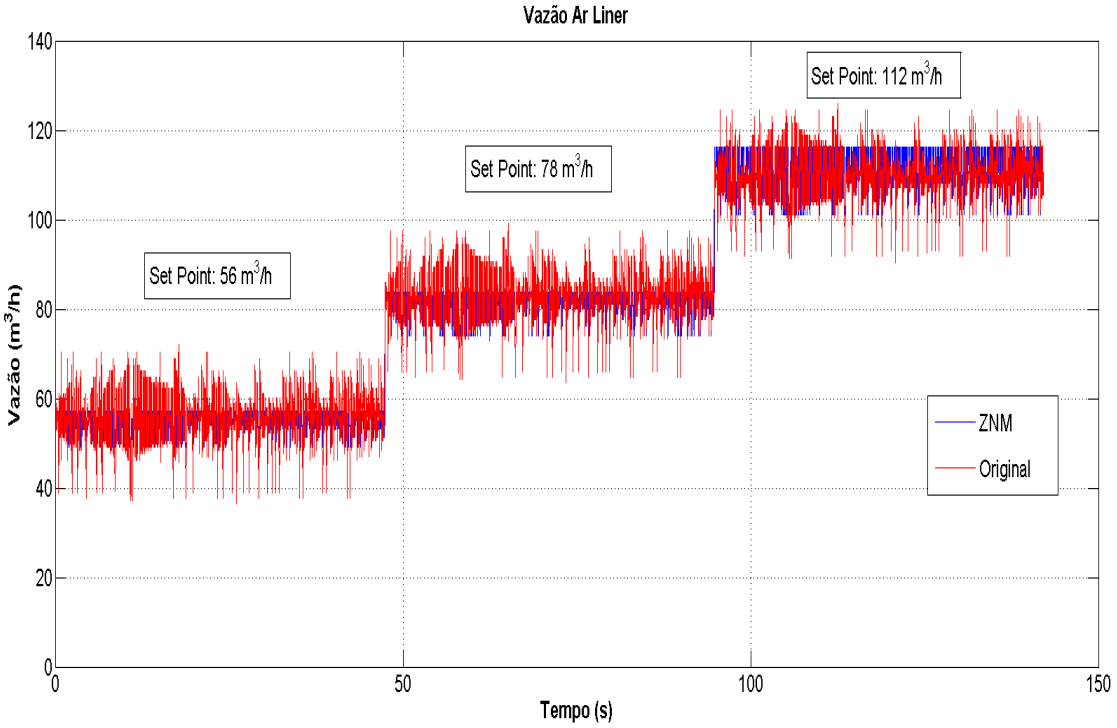

Figura 69 – Resposta da Vazão para o Ar do Liner

	Original	Ziegler-Nichols	Otimizada
K_p	1	5,4	6,3
T_i	1	1,35	2,18
T_d	1	1,06	1,67

Ganhos para o PID do Ventilador da Cabine

6.2 Controle de Temperatura

Os supervisores *fuzzy* foram implementados com o bloco *fuzzy controller* do Simulink. O método de Mamdani (MAMDANI; ASSILIAN, 1975) é utilizado para a defuzzificação das saídas dos supervisores. A ferramenta de visualização do conjunto de regras do supervisor é ilustrada na figura 70.

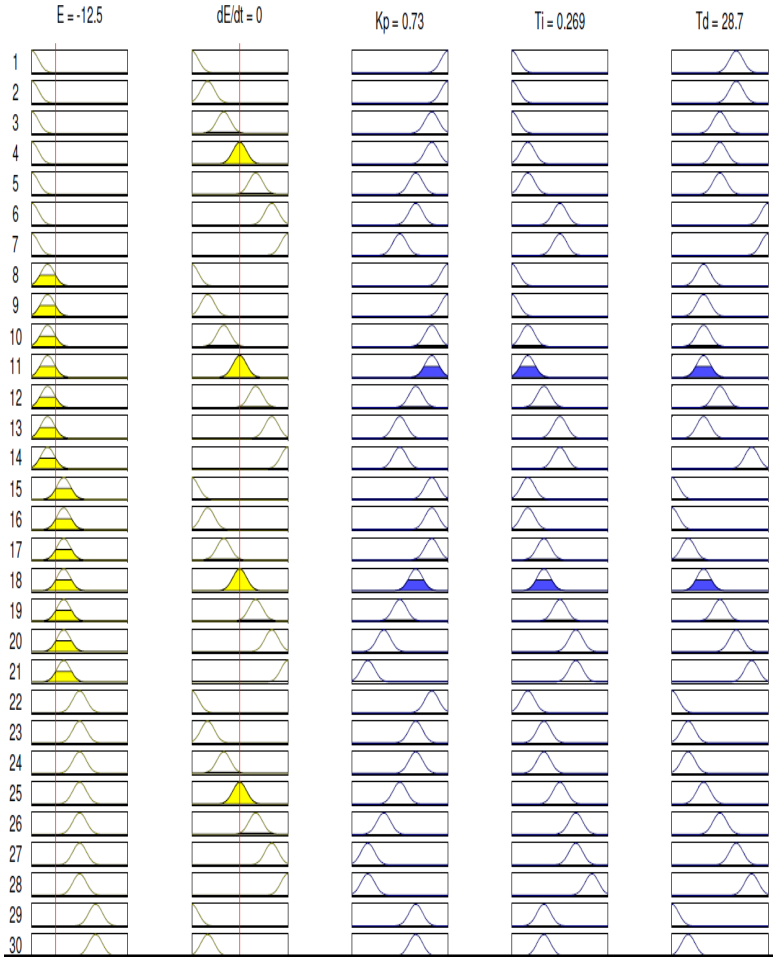

Figura 70 – Ferramenta de Visualização das Saídas Defuzzificadas

As superfícies de variação dos ganhos para os supervisores PID e supervisor de *set points* são mostradas nos gráficos abaixo. Por conveniência, os ganhos dos supervisores de PID para os sistemas *liner* e cabine são os mesmos.

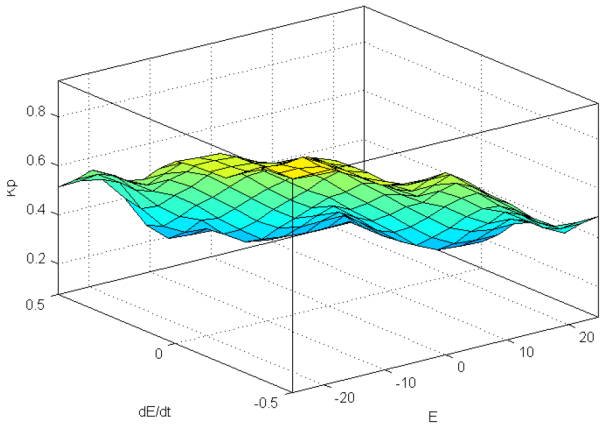

Figura 71 – Superfície do Ganho K_p para os supervisores dos controladores dos sistemas *liner* e cabine

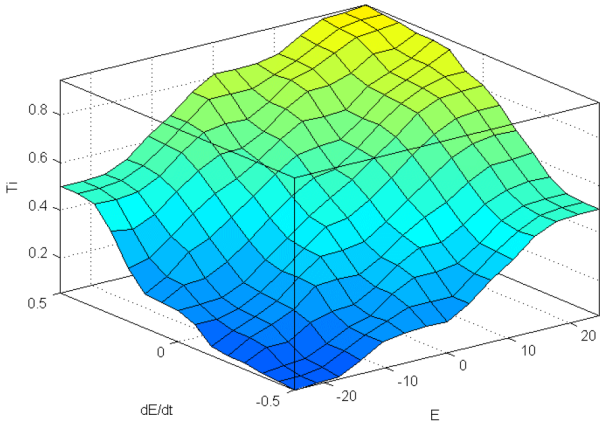

Figura 72 – Superfície do Ganho T_i para os supervisores dos controladores dos sistemas *liner* e cabine

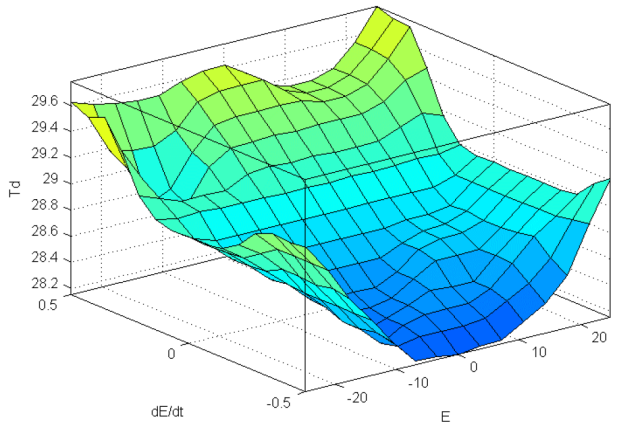

Figura 73 – Superfície do Ganho T_d para os supervisores dos controladores dos sistemas *liner* e cabine

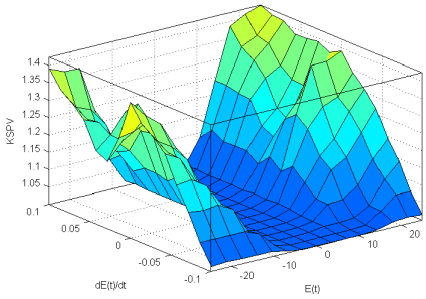

Figura 74 – Superfície de Variação do *Set Point* de Vazão

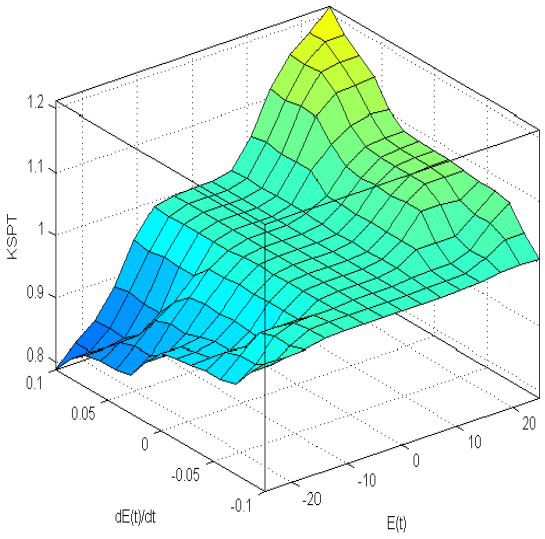

Figura 75 – Superfície de Variação do *set point* de Temperatura

Os testes de aquecimento e resfriamento foram realizados com um degrau de 15 °C em relação a temperatura ambiente, no caso de aquecimento, e 15 °C em relação a temperatura atual do sistema, no caso de resfriamento.

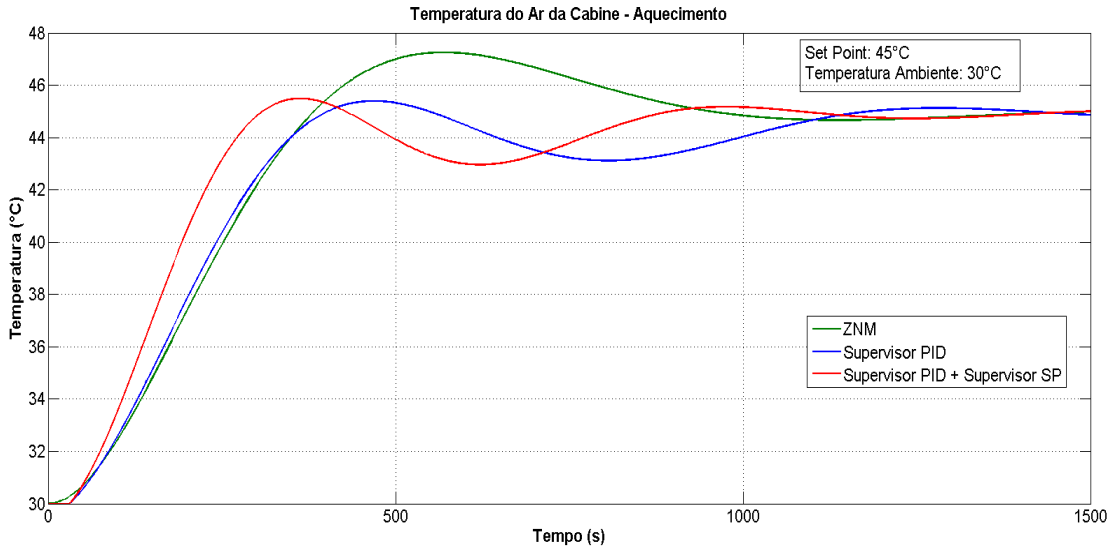

Figura 76 – Aquecimento do Ar da Cabine

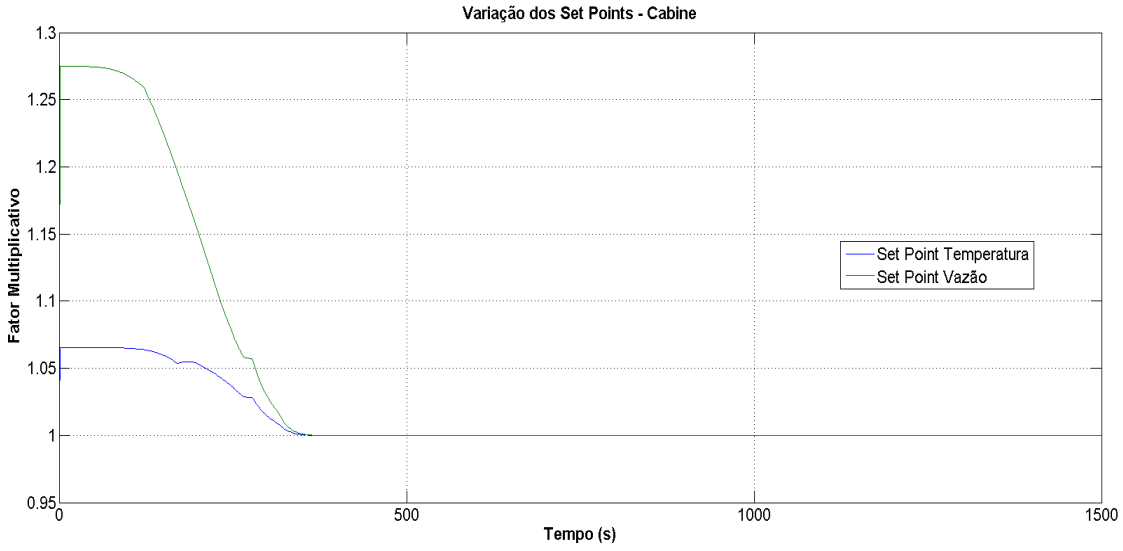

Figura 77 – Variação dos *Set Points* - Aquecimento do Ar da Cabine

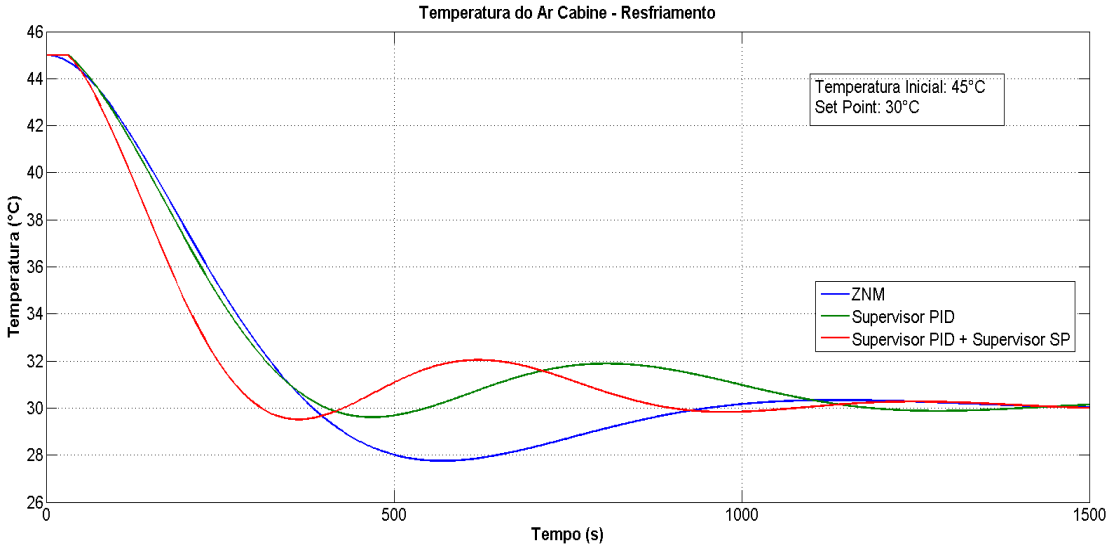

Figura 78 – Resfriamento do Ar da Cabine

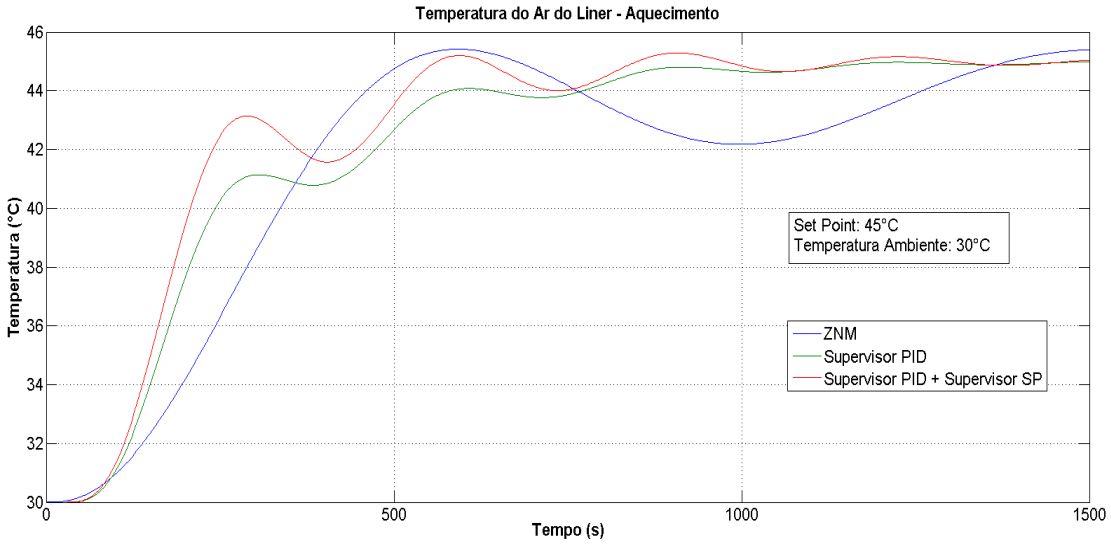

Figura 79 – Aquecimento do Ar do *Liner*

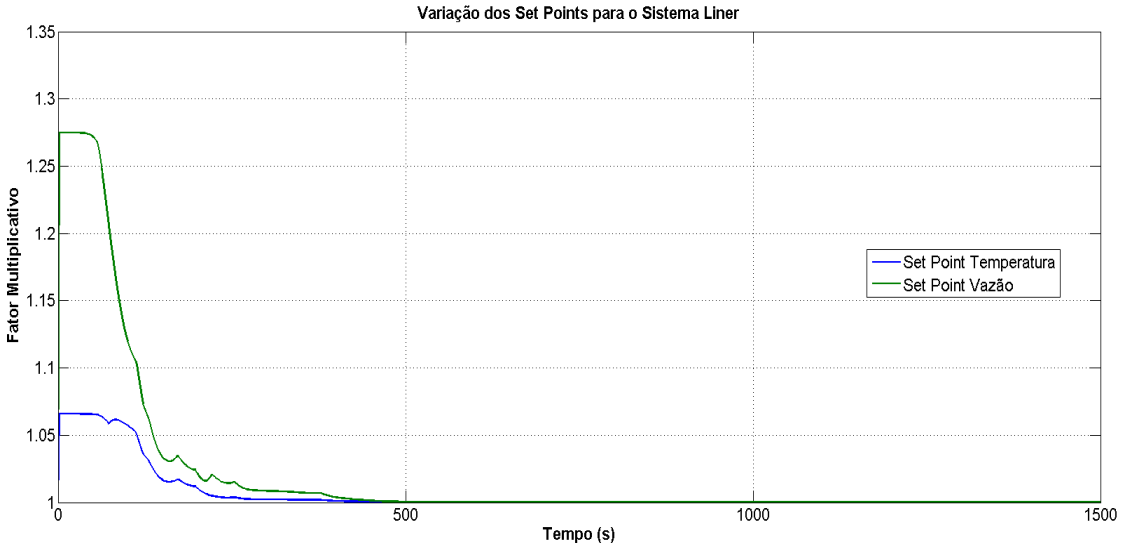

Figura 80 – Variação dos *Set Points* - Aquecimento do Ar do Liner

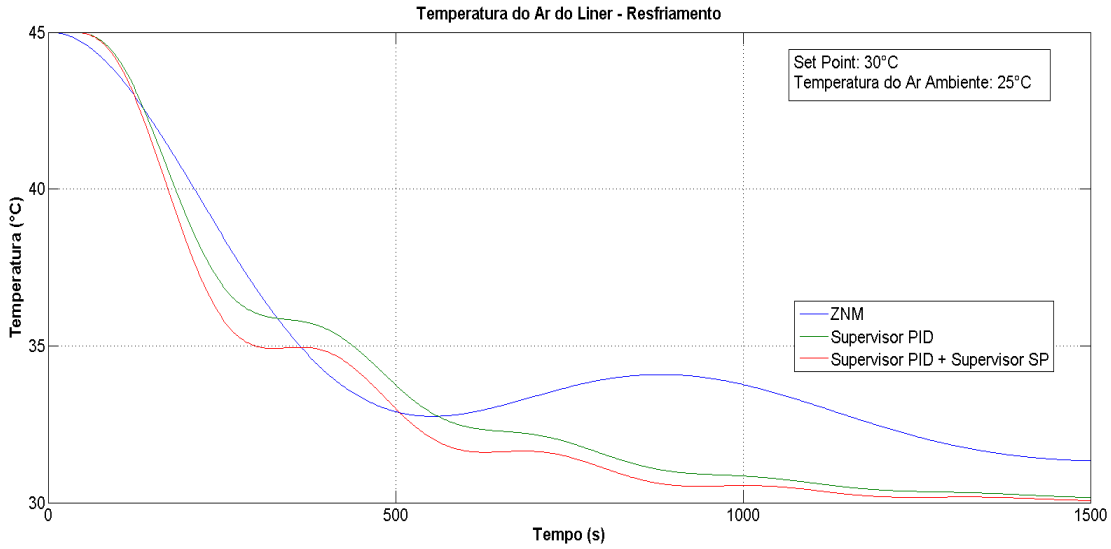

Figura 81 – Resfriamento do Ar do *Liner*

6.3 Modificações Propostas

Nesta seção serão propostas algumas modificações sobre a no *Mock up* tendo por base conclusões obtidas de ensaios com o protótipo e observações realizadas durante ensaios reais na planta.

6.3.1 Modificações no Controle de Temperatura

Há vários elementos construtivos no *mock up* que introduzem algumas características que dificultam o controle da temperatura do ar para os sistemas *liner* e cabine na planta atual:

- *Software*: O programa utilizado para a programação das receitas de ensaios envia os ganhos dos controladores PID antes do início do ensaio e não permite a modificação em tempo real dos ganhos, dificultando a implementação de métodos de adaptatividade e otimização. O ensaio precisa ser parado para o ajuste do controlador.
- Atuadores: São utilizados conjuntos de resistências industriais ligadas em triângulo para o aquecimento do ar oriundo dos sistemas de condicionamento. Além de pouco eficiente em termos da relação energia térmica dissipada/energia elétrica consumida, este método não aquece o ar de forma uniforme e introduz atraso no sistema.
- Vazão de saída da cabine: Não existe controle da vazão do ar de renovação da cabine no ambiente do supervisório.
- Independência entre os controles de temperatura e vazão: O modelo de primeira ordem apresentado no capítulo 2 descreve a relação entre temperatura e vazão, entretanto os controladores PID atuam de forma independente.

A implementação de um *software* com ajuste dos ganhos em tempo real ou até a integração com Simulink/MATLAB facilitaria a implementação de outros métodos de controle (supervisor *fuzzy*, preditor de *Smith*, controladores de ordem superiores). Com a modificação em tempo real dos ganhos dos controladores, tornaria-se mais fácil a otimização do controlador por meio de rotinas computacionais. A vazão de saída da cabine, conforme descrito pelo modelo de primeiro ordem no capítulo 2, é uma variável importante na análise do controle da temperatura do ar na cabine e poderia ser controlada e modificada no supervisório. Os controles de temperatura e vazão, para o caso do *liner*, deveriam ser integrados. Conforme mostrado pela adição do supervisor, a resposta da temperatura se torna bem mais rápida com a integração dos sistemas.

6.3.2 Modificações no Controle de Vazão

O controle de vazão pela atuação de ventiladores se mostrou problemático do ponto de vista do controle por PID. Existem algumas particularidades do controle de vazão (TANNURI; CRUZ, 2008):

- Processo rápido: tempo morto desprezível e constante de tempo da ordem de segundos;

- Sinal medido muito ruidoso: flutuações rápidas oriundas de irregularidades na tubulação e de instabilidade do ventilador;

- Zona morta: para o ventilador axial utilizado, a saída é nula até que a magnitude da entrada exceda um certo valor.

Os principais problemas encontrados no controle de vazão com a utilização de ventiladores foram a saturação e a zona morta. O atrito seco no eixo do motor DC, utilizado para girar as pás do ventilador, gera uma zona morta de controle, tornando o controle de vazão em *set points* baixos pouco efetivo. Esses problemas também existem com a utilização ventilador centrífugo utilizado na planta. O motor de indução trifásico utilizado para acionar o ventilador apresenta zona morta em razão do atrito seco e escorregamento no rotor. Outro problema do ponto de vista do controle é a relação não linear entre vazão e rotação das pás. O controle PID altera a rotação das pás dos ventiladores e é realimentado com a leitura da vazão medida pelos medidores de vazão. Além disso, o sistema liner no *Mock up* não apresenta retroalimentação de medidores de vazão.

Válvulas de controle se mostram uma alternativa mais viável para o controle de vazão de ar, pois há uma vasta bibliografia disponível com modelos desenvolvidos para o controle de válvulas, em que a relação entre vazão volumétrica e ângulo de abertura foi determinada para diferente modelos de válvulas (CHERN; HUANG, 2004), (HUANG; KIM, 1996), (KMIURA; TANAKA, 1995). Válvulas de controle de fluxo também podem ser acionadas por diversos tipos de atuadores, como solenóides, motores de passo, servomotores, ampliando as alternativas de controle. A vazão também determina o atraso de transporte causado pela tubulação, conforme descrito no capítulo anterior, logo é fundamental que a resposta em malha fechada da vazão seja estável, pois um atraso variável no controle de temperatura torna a solução de controle mais complexa.

6.4 Conclusões

Neste trabalho faz-se um estudo sobre a sintonia de controladores PID por meio de supervisores *fuzzy*, assim como a adição de supervisores *fuzzy* de *set points* para a integração dos controles de vazão e temperatura, visando obter uma melhor resposta em malha fechada para as temperaturas do ar do sistema *liner* e cabine. Visa-se também o ajuste dos controladores PID dos ventiladores centrífugos que regulam a vazão volumétrica de ar para os sistemas *liner* e cabine.

Para efetuar o controle, foram realizadas três tarefas: ajuste pelo segundo método de Ziegler-Nichols dos controladores de vazão e temperatura; projeto de um supervisor *fuzzy* para ajuste automático dos ganhos do controladores PID das temperaturas de in-

teresse e projeto de um supervisor de *set points* para as malhas de controle de vazão e temperatura.

Por meio de simulações realizados em Simulink/MATLAB, observa-se que o ajuste pelo segundo método de Ziegler-Nichols dos controladores PID gera respostas satisfatórias para as malhas de vazão, mas não para as temperaturas de interesse. Para os controladores PID de temperatura, a introdução de um supervisor de ganhos *fuzzy* gera uma resposta satisfatória de acordo com os critérios estabelecidos no capítulo introdutório. Quando a ação do supervisor de ganhos é combinada com a ação do supervisor de *set points*, além da resposta em malha fechada desejada ser obtida, observou-se uma diminuição no tempo de subida dos sistemas.

Referências

ALLEN, P. *Similarity Rules*. [S.l.]: Dalhousie University Department of Mechanical Engineering, 2006.

ASTROM, K. J.; HAGGLUND, T. *PID Controllers: Theory, Design and Tuning*. [S.l.]: The Instrumentation, Systems and Automation Society, 2010.

BERGMAN, T. L. et al. *Fundamentos de Transferencia de Calor*. Rio de Janeiro: LTC, 2011. 644 p.

CAMPOS, M. C. M. M.; TEIXEIRA, H. C. G. *Controles Típicos de Equipamento e Processos Industriais*. [S.l.]: Editora Edgard Blucher, 2006.

CHERN, M. J.; HUANG, C. D. Control of volumetric flow rate of ball valve using v-port. *Journal of Fluid Engineering*, v. 126, p. 471–481, 2004.

CHIU, S. Using fuzzy logic in control applications: Beyond fuzzy PID control. *IEEE Control Systems*, p. 100–105, 1998.

CONTECH. *Fluxometro de Massa Térmica e Transmissor de Temperatura*. São Paulo, SP, 2007. Manual.

COPELAND, R. P.; KULDIP, S. R. A fuzzy supervisor for PID control of unknown systems. *IEE Internation Symposium on Intelligent Control*, v. 4, p. 97–115, 1994.

COUGHANOWR, D. R.; KOPPEL, L. B. *Análise e Controle de Processos*. Rio de Janeiro: Guanabara Dois, 1978.

EMBRAER/FAPESP. *Especificação do Sistema de Condicionamento Térmico do Mock-up Integrado Projeto: Conforto de Cabine*. [S.l.]: Escola Politécnica da Universidade de São Paulo, 2009.

GOMIDE, F. A. C.; GUDWIN, R. R. Modelagem , controle, sistemas e lógica fuzzy. *Revista da Sociedade Brasileira de Automática*, p. 602–608, 1994.

GUERRA, R. *Projeto e simulação do controle de atitude autônomo de satélites usando lógica nebulosa*. Dissertação (Mestrado em Engenharia e Tecnologia Espaciais/ Mecânica Espacial e Controle) — Instituto Nacional de Pesquisas Espaciais, 1998.

HUANG, C. D.; KIM, R. Three-dimensional analysis of partially open butterfly valve flows. *ASME Transactions*, v. 118, p. 562–568, 1996.

HURWITZ, A. On the conditions under which an equation has only roots with negative real parts. *Matematische Annelen*, v. 46, p. 273–284, 1895.

JANG, J. S.; GULLEY, N. *Fuzzy Logic Toolbox User's Guide*. [S.l.], 2008.

KMIURA, T.; TANAKA, T. Hydrodinamic characteristics of a butterfly valve-prediction of pressure loss characteristic. *ISA Transactions*, v. 34, p. 319–326, 1995.

LEMKE, H. R. van N.; DE-ZHAO, W. Fuzzy PID supervisor. *24th Conference on Decision and Control*, 1985.

LINHARES, G. C. *Controle das Condições de Conforto térmico num Ambiente Utilizando Lógica Fuzzy*. Dissertação (Mestrado) — Universidade Federal de Ouro Preto, Ouro Preto, 2010.

MACVICAR-WHELAN, P. J. Fuzzy sets for man-machine interaction. *International Journal of Man-Machine Studies*, v. 8, p. 687–697, 1976.

MAEDA, M.; MURAKAMI, S. A self-tuning fuzzy controller. *Fuzzy Sets and Systems*, v. 51, p. 29–40, 1992.

MAMDANI, E. H.; ASSILIAN, S. An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-machine Studies*, v. 7, p. 1–13, 1975.

OGATA, K. *Engenharia de Controle Moderno*. [S.l.]: Pearson Education, 2011.

ROUTH, E. J. *A Treatise on the stability of motion*. [S.l.]: Macmillan, 1877.

SHAW, I. S.; SIMOES, M. G. *Controle e Modelagem Fuzzy*. São Paulo: Blucher, 2007.

TANNURI, E. A.; CRUZ, J. J. *Entendendo e Ajustando Malhas de Controle*. [S.l.]: Escola Politécnica da Universidade de São Paulo, 2008.

TZAFESTAS, S.; PAPANIKOLOPOULOS, N. P. Incremental fuzzy expert PID control. *IEEE Transactions on Industrial Electronics*, v. 37, p. 365–371, 1990.

WEI, J. Research on the temperature control system based on fuzzy self-tuning PID. *ICCDA International Conference on Computer Design and Applications*, 2010.

YOKOGAWA. *UT15/UT14 Digital Indicating Controllers, technical information publication TI*. Tokyo, Japan, 1990. Apostila.

ZADEH, L. A. Fuzzy sets. *Information and Controll*, v. 8, p. 338–358, 1965.

ZIEGLER, J. G.; NICHOLS, N. B. Optimum settings for automatic controller. *ASME. Trans*, v. 1, p. 64–759, 1942.

Anexos

ANEXO A – Desenhos e Vistas

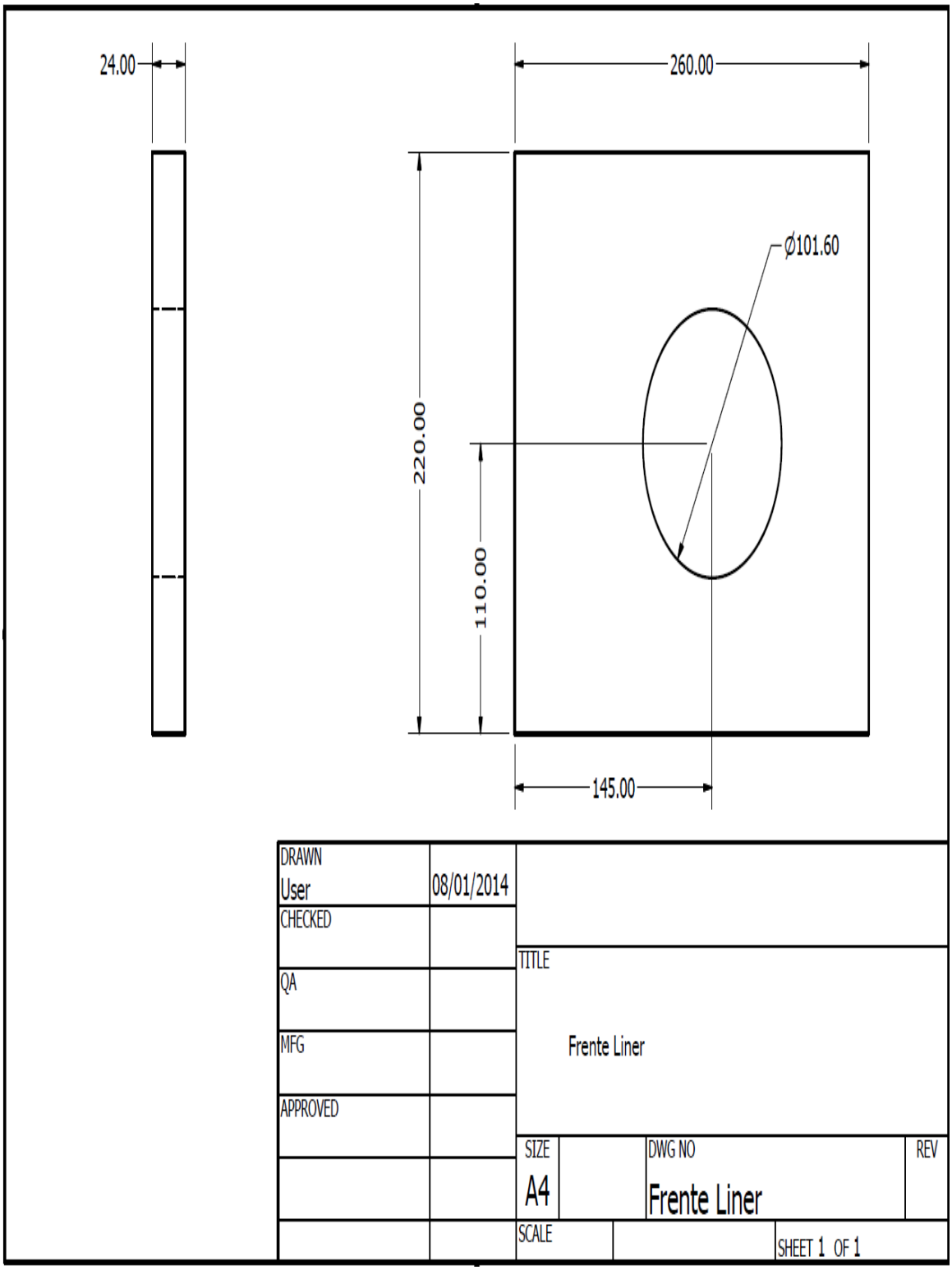

Figura 82 – Frente da Caixa de Aquecimento do Liner

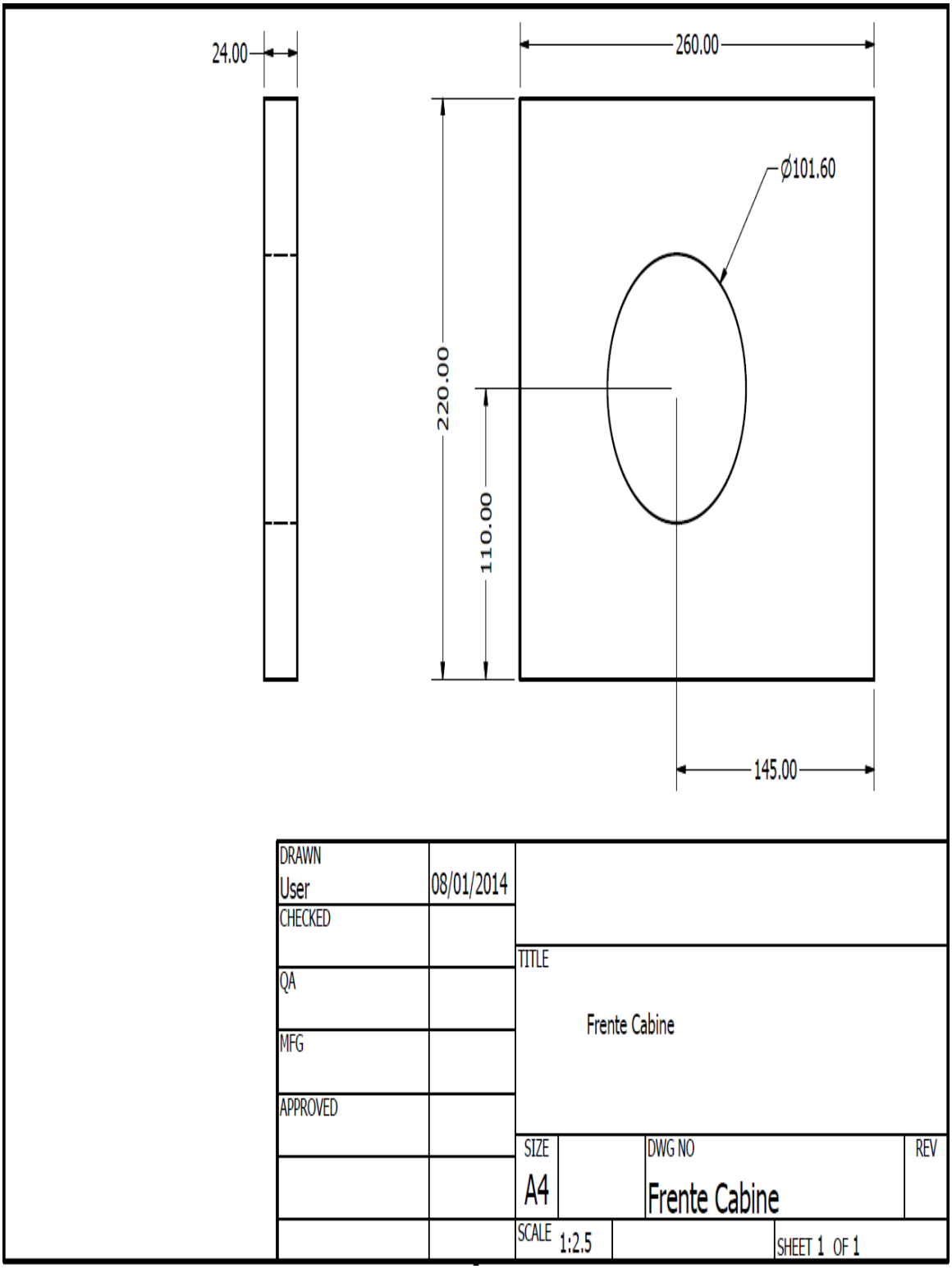

Figura 83 – Frente da Caixa de Aquecimento do Liner

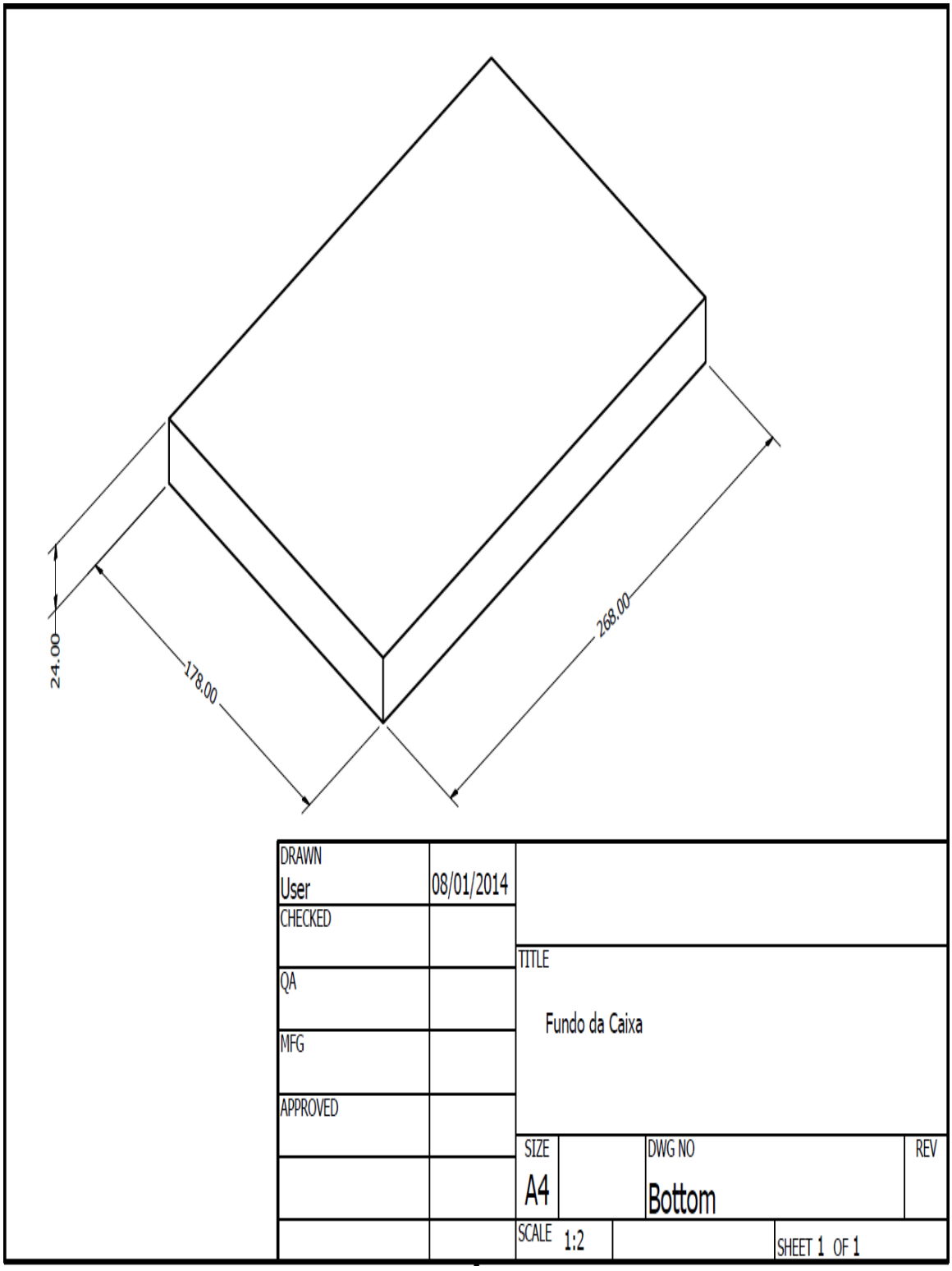

Figura 84 – Fundo da Caixa de Aquecimento da Cabine e Liner

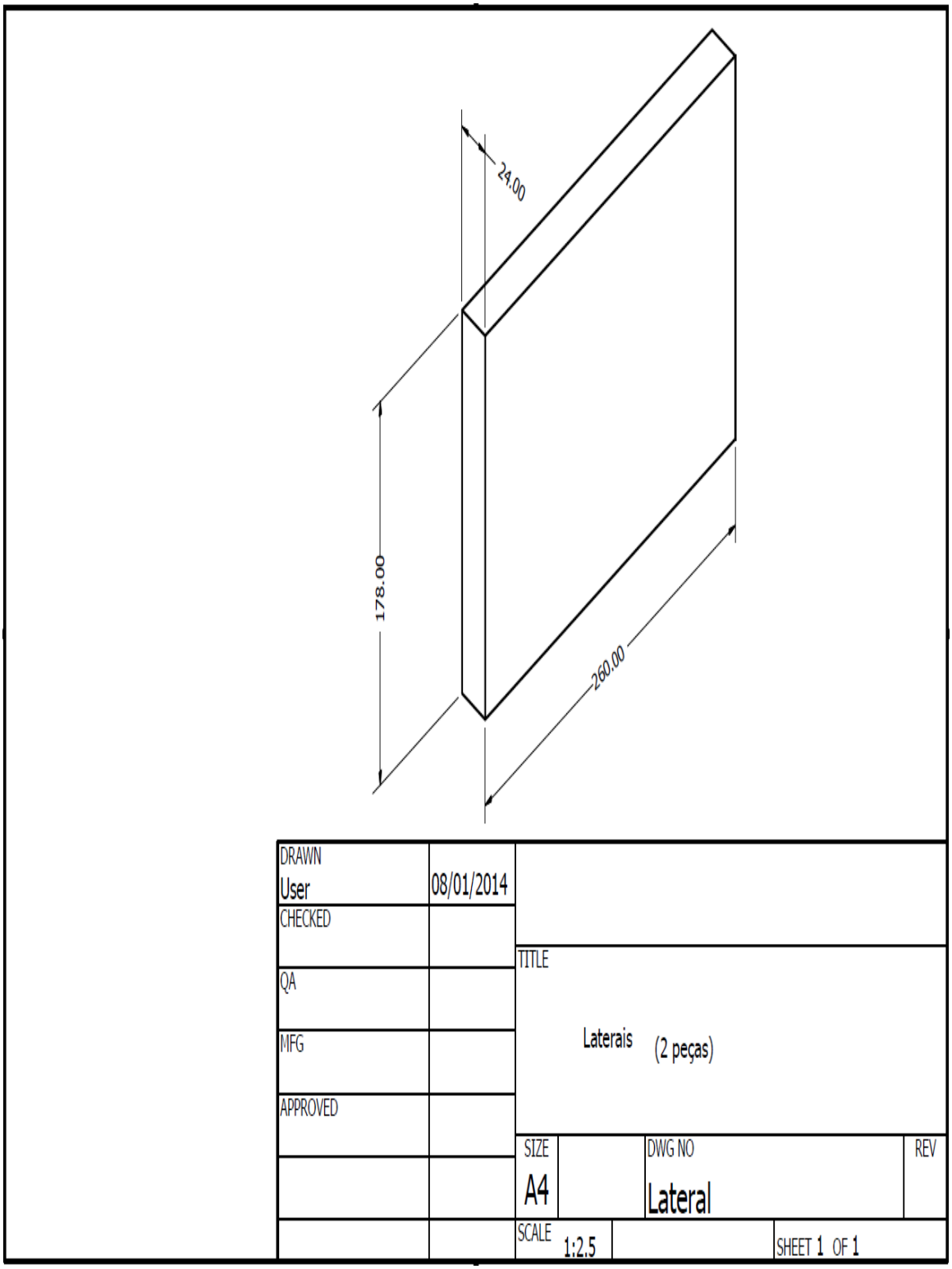

Figura 85 – Lateral da Caixa de Aquecimento da Cabine e Liner

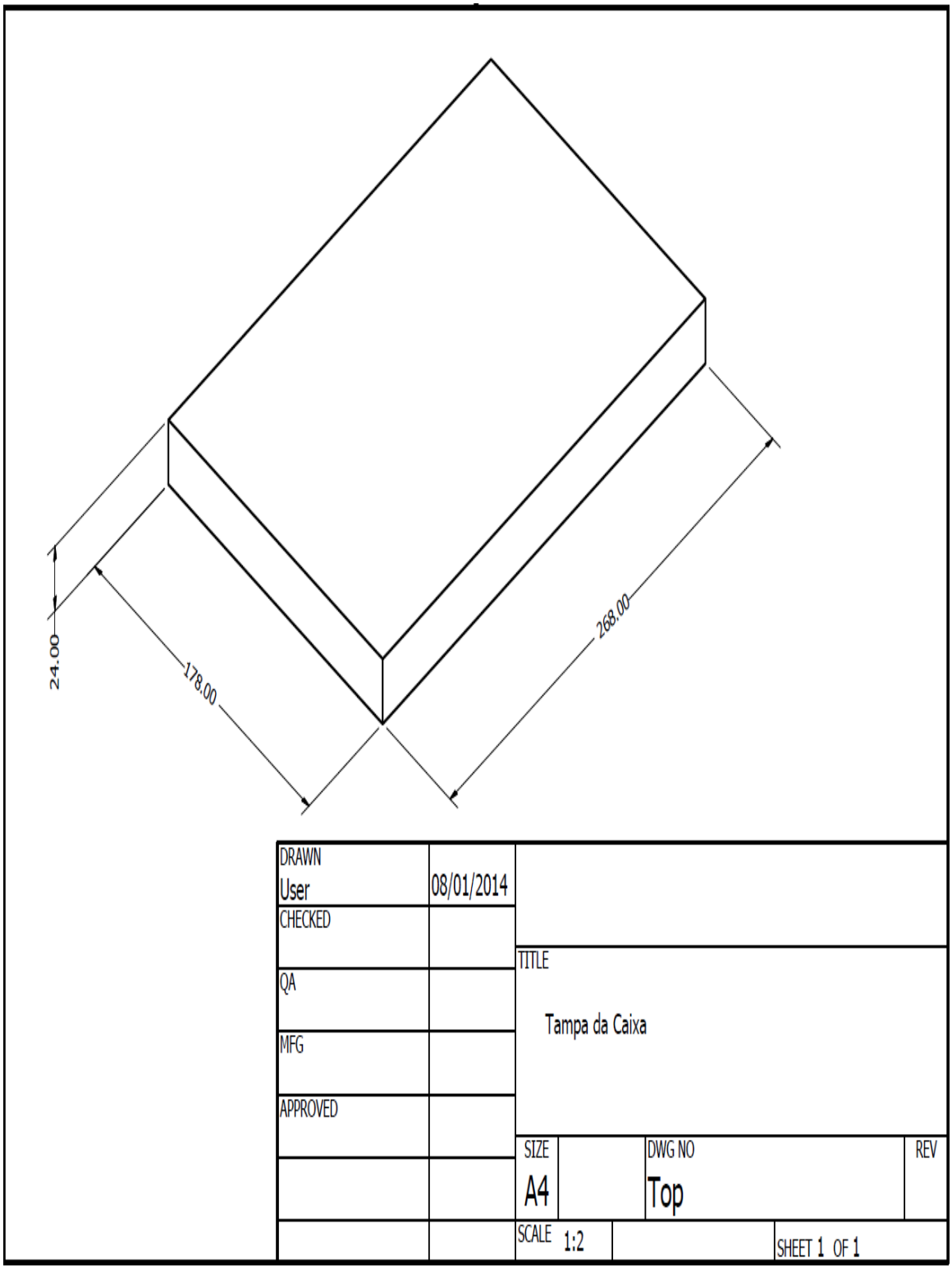

Figura 86 – Tampa da Caixa de Aquecimento da Cabine e Liner

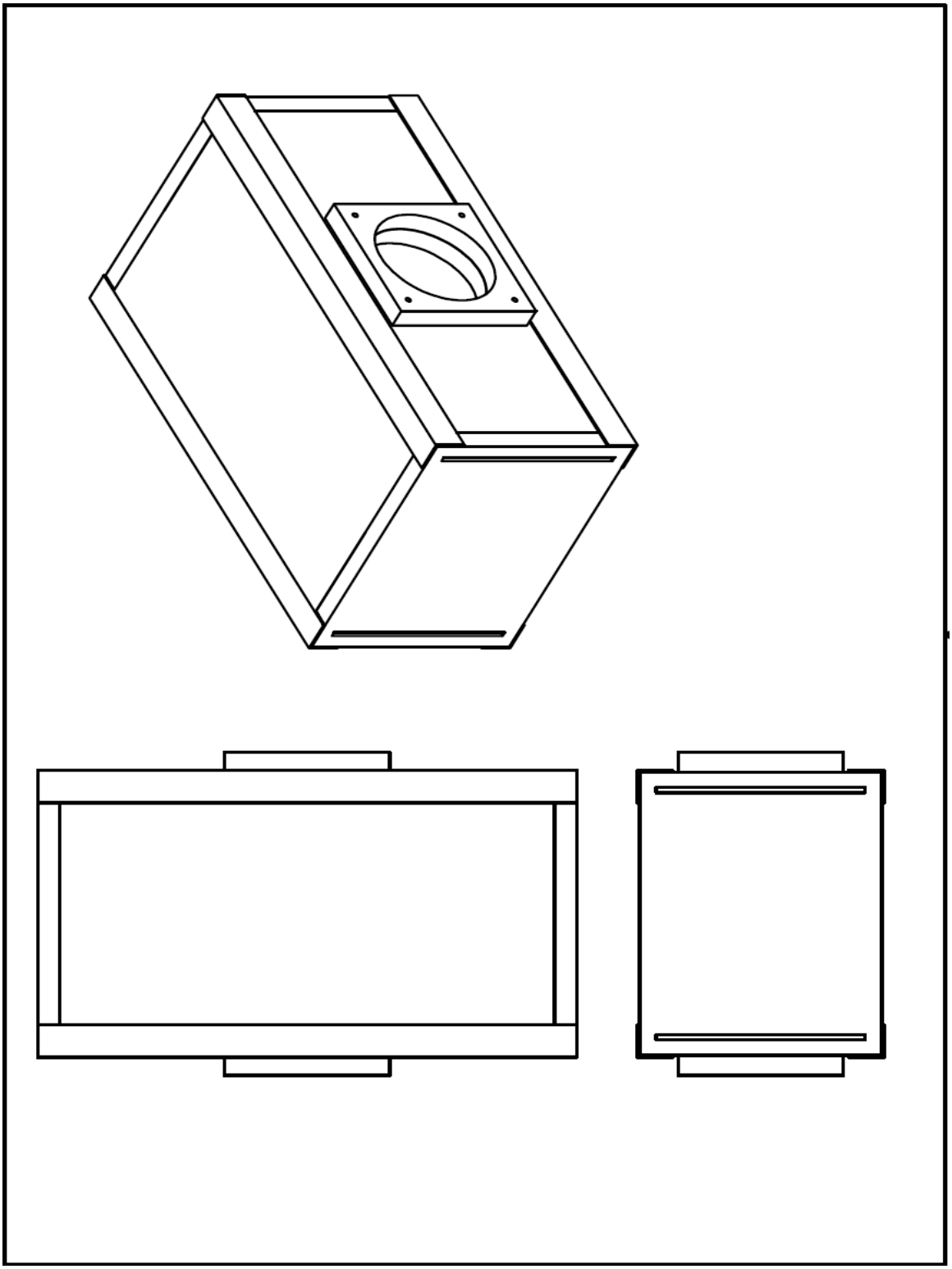

Figura 87 – Protótipo dos Sistemas *Liner* e Cabine

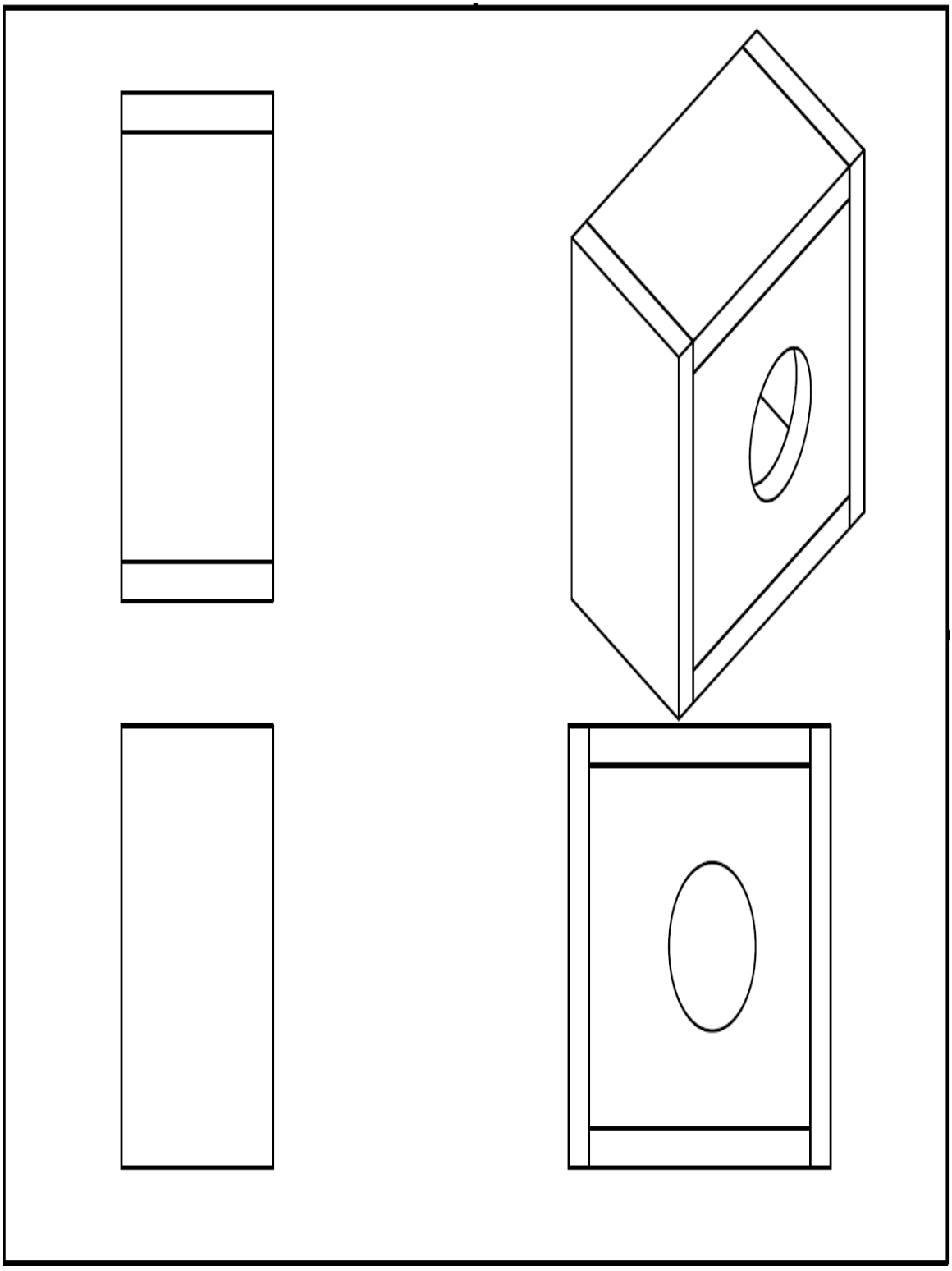

Figura 88 – Caixa de Aquecimento do Liner

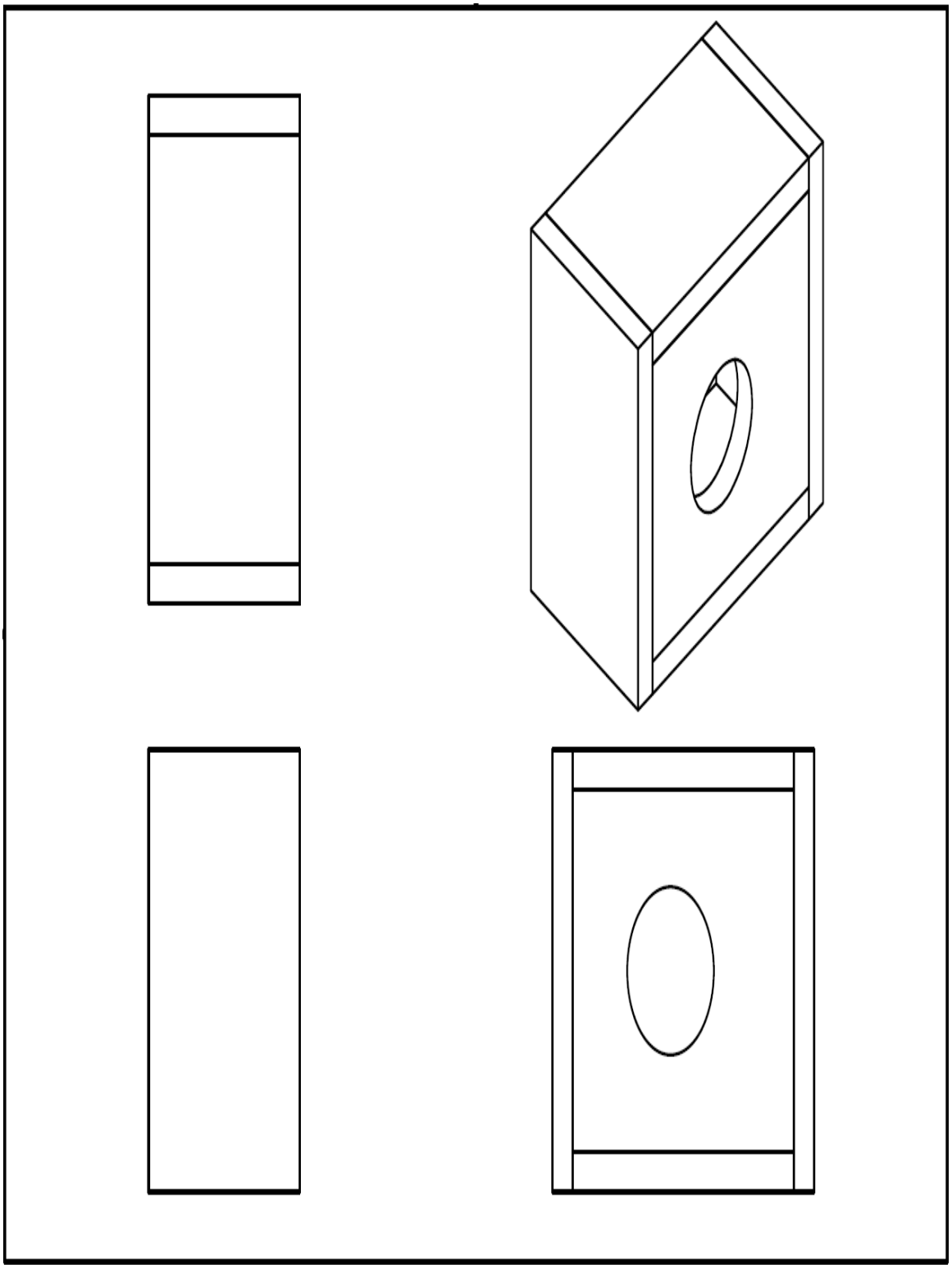

Figura 89 – Caixa de Aquecimento da Cabine

ANEXO B – Circuitos Elétricos

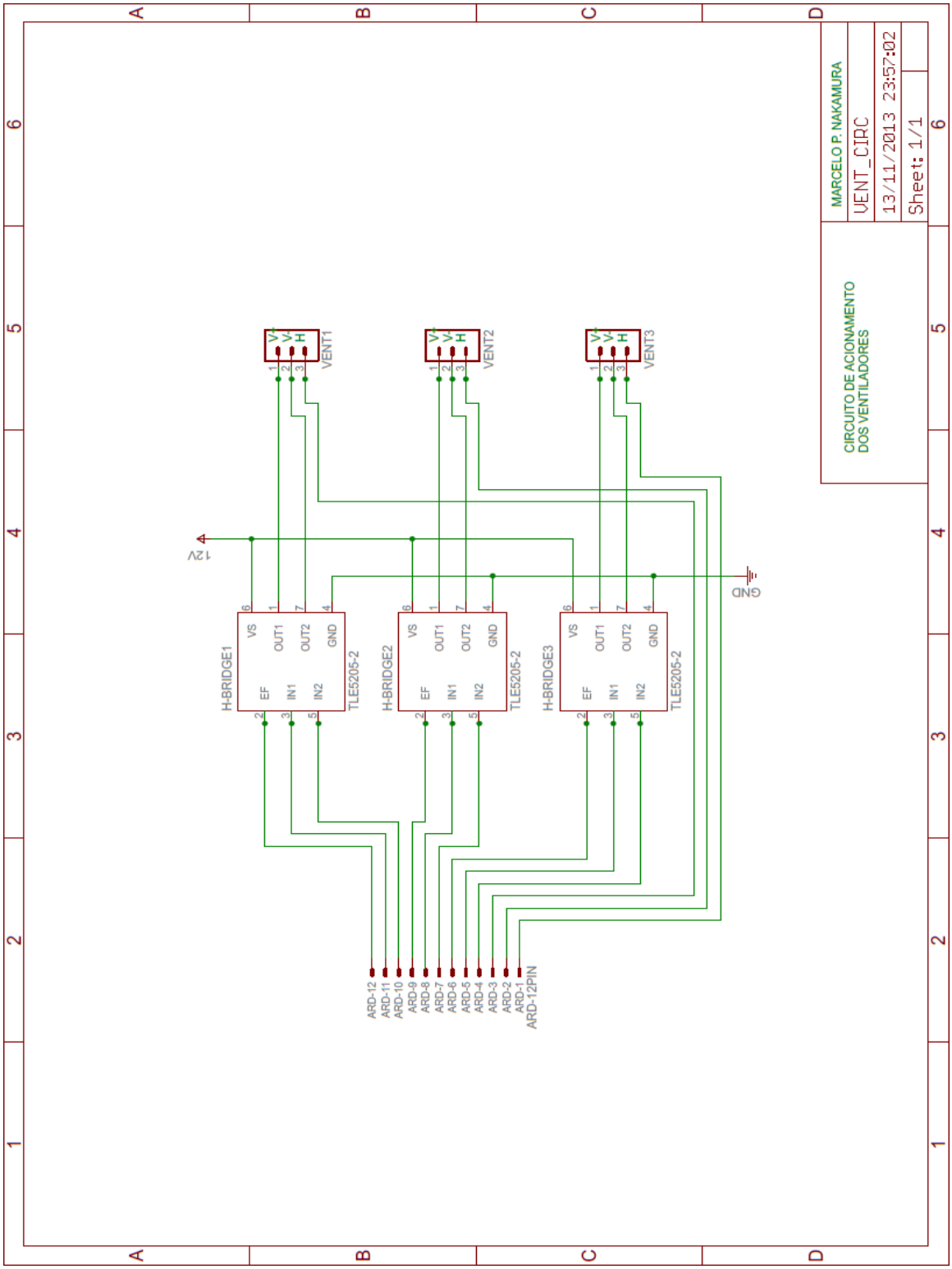

Figura 90 – Circuito de Acionamento dos Ventiladores

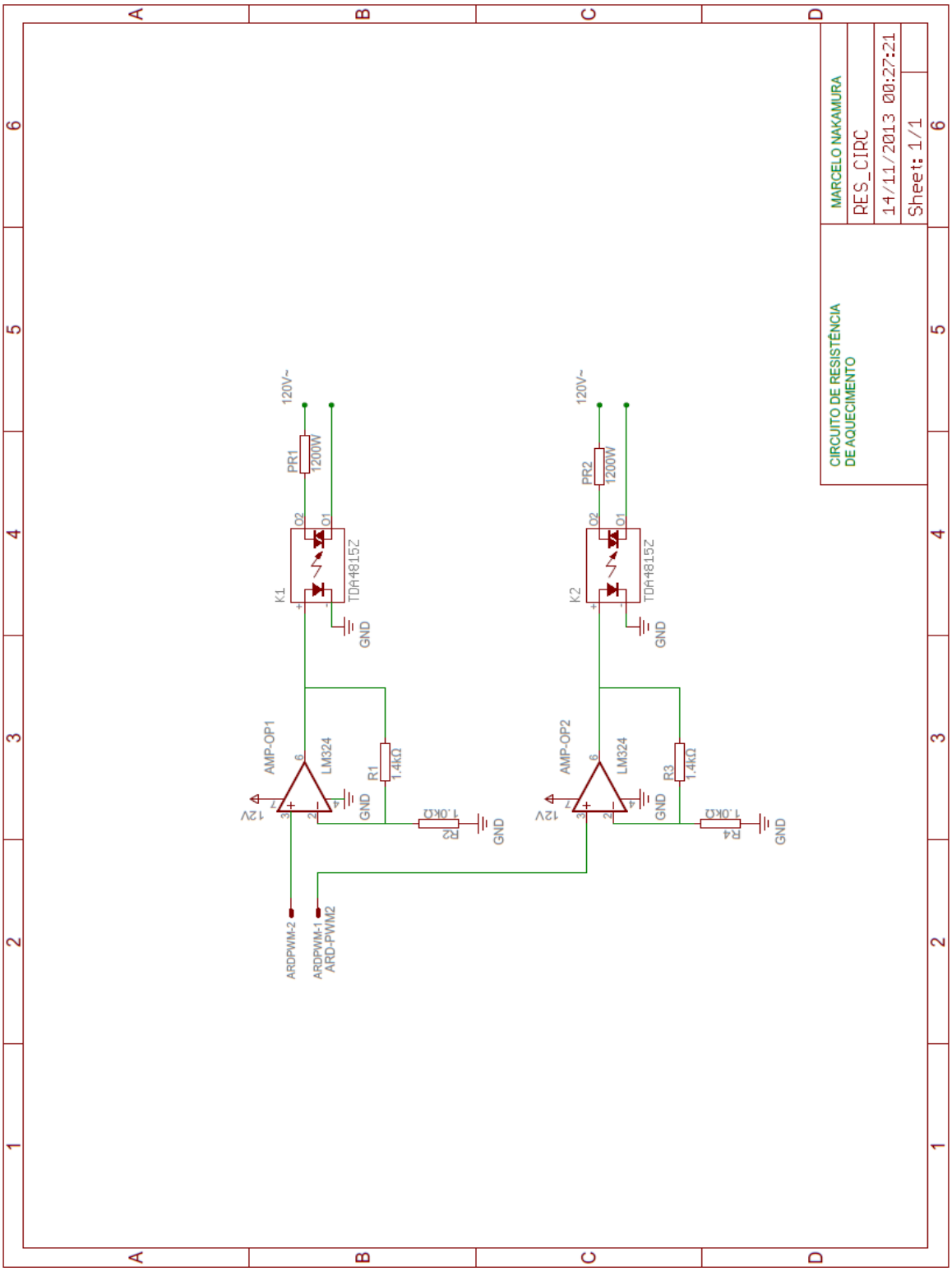

Figura 91 – Circuito de Resistência de Aquecimento

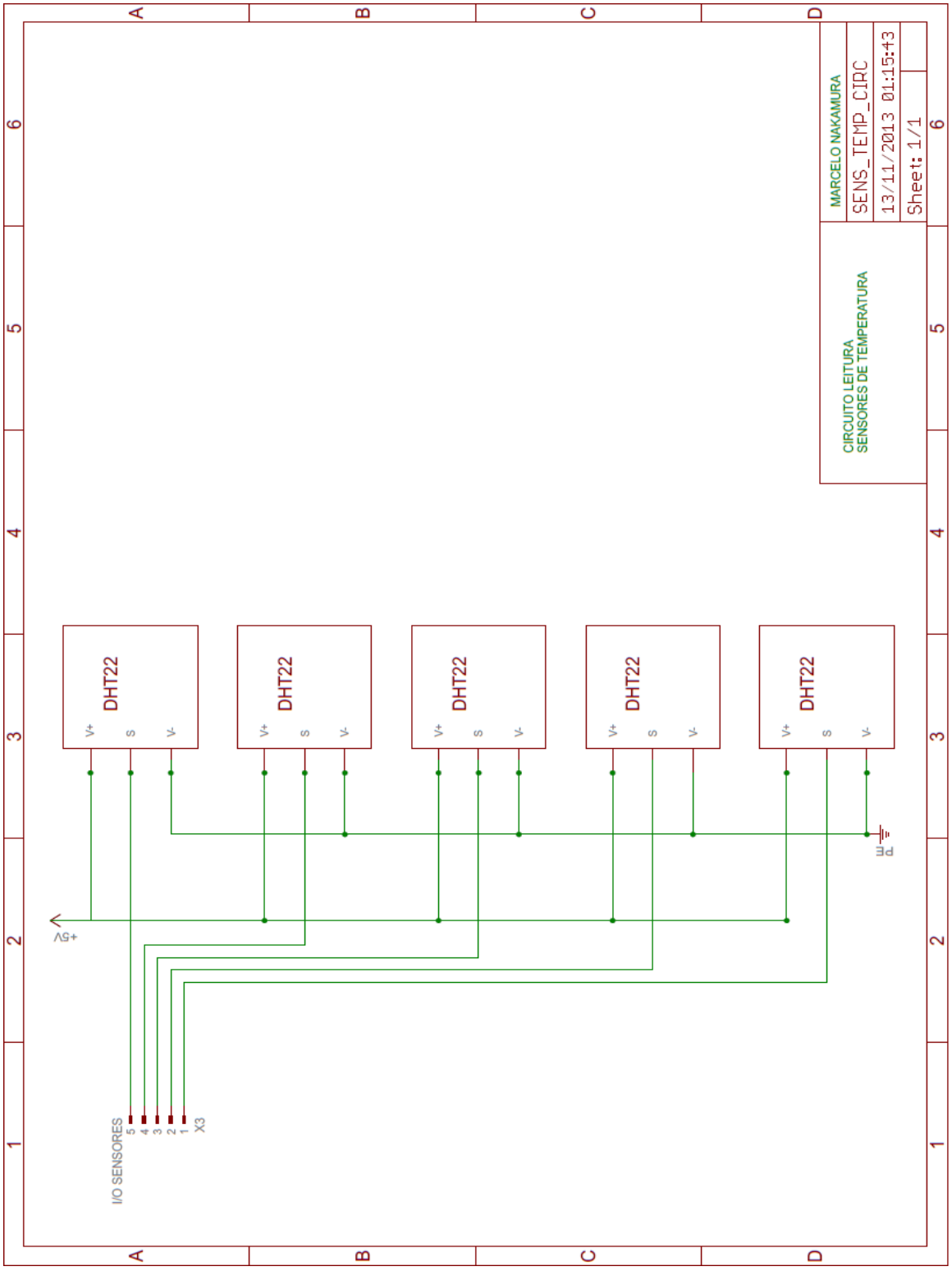

Figura 92 – Circuito de Leitura de Sensores de Temperatura

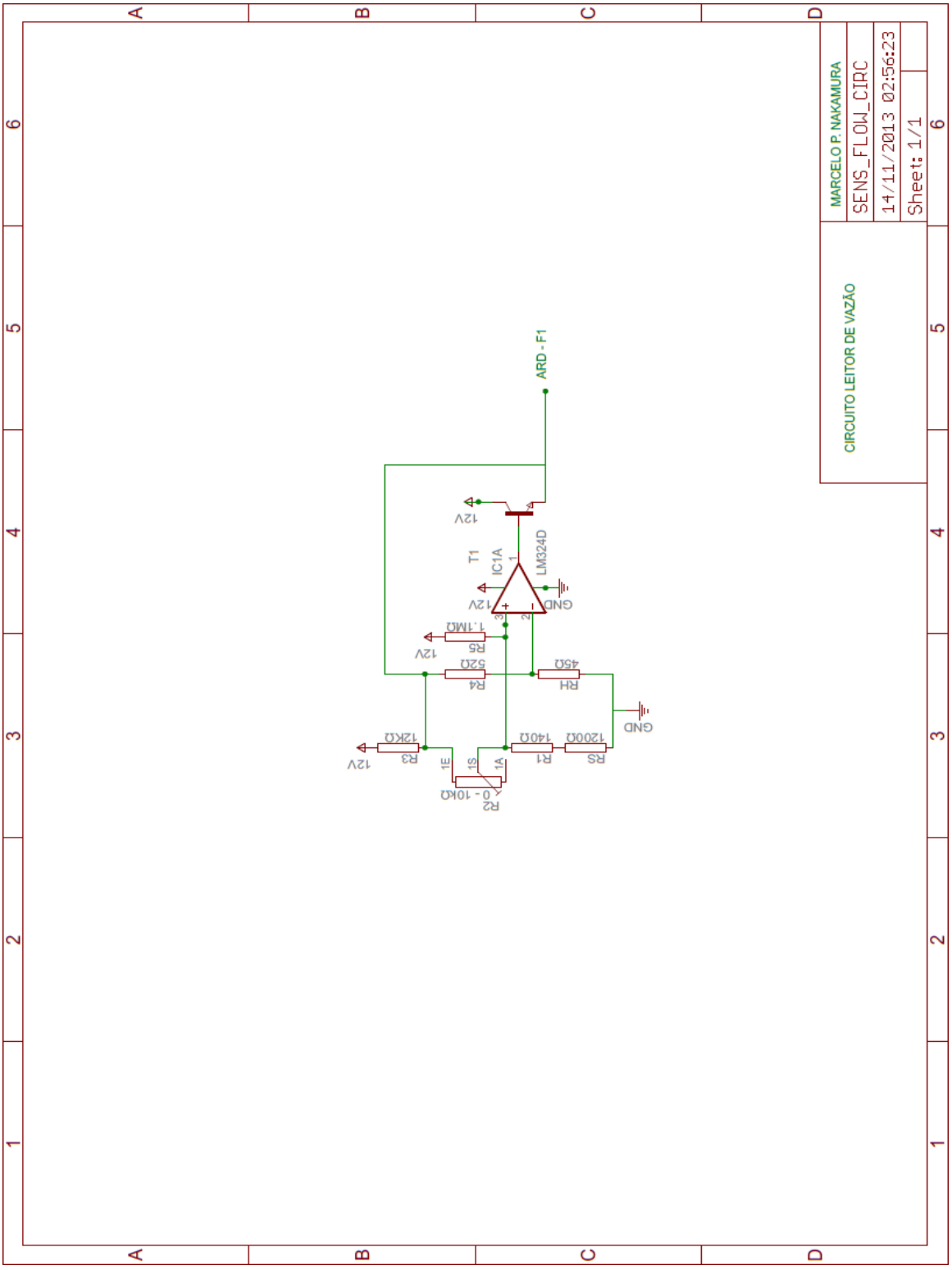

Figura 93 – Circuito do Sensor de Vazão

ANEXO C – Modelos em Simulink/MATLAB

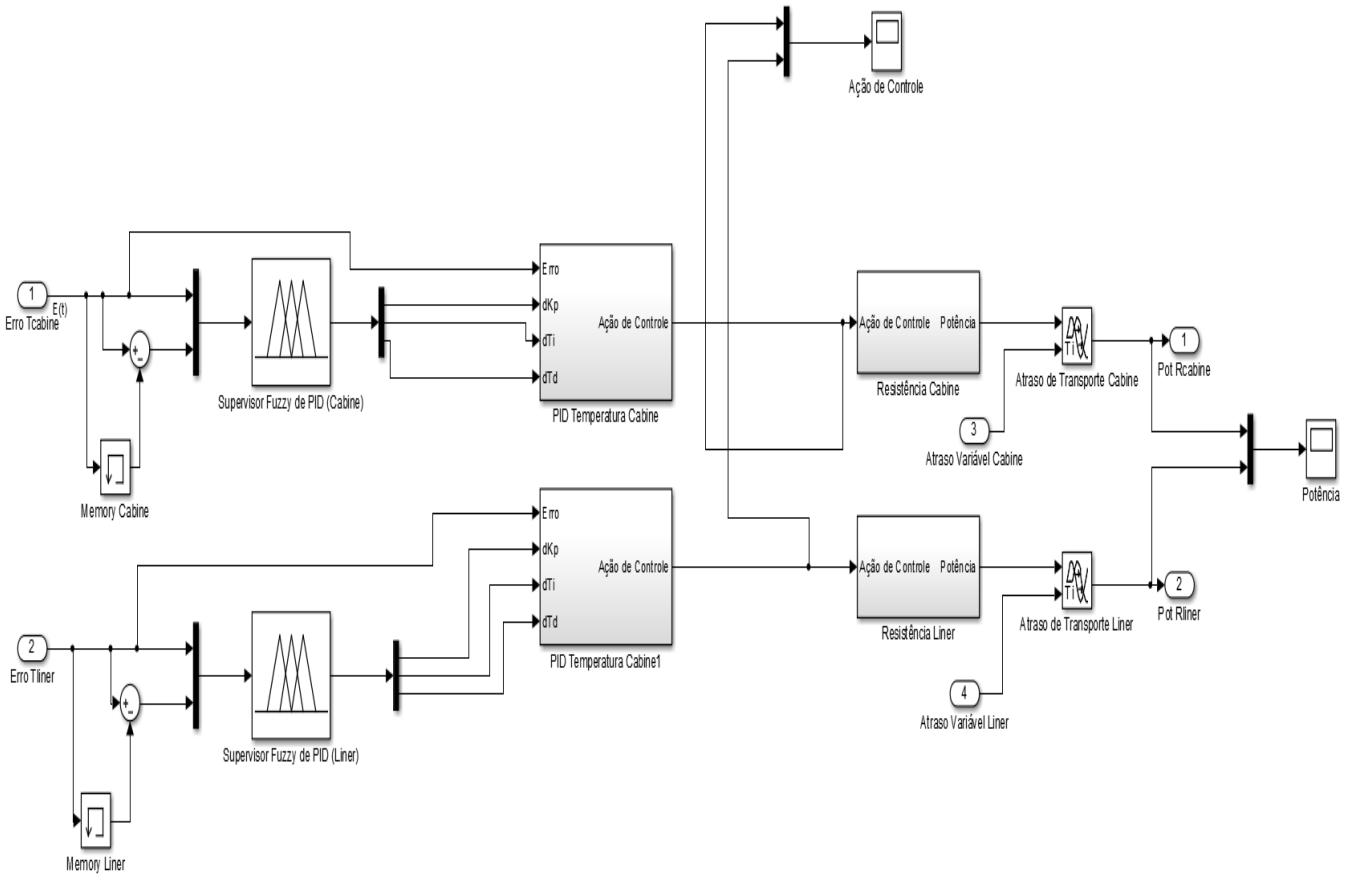

Figura 94 – Malha Completa

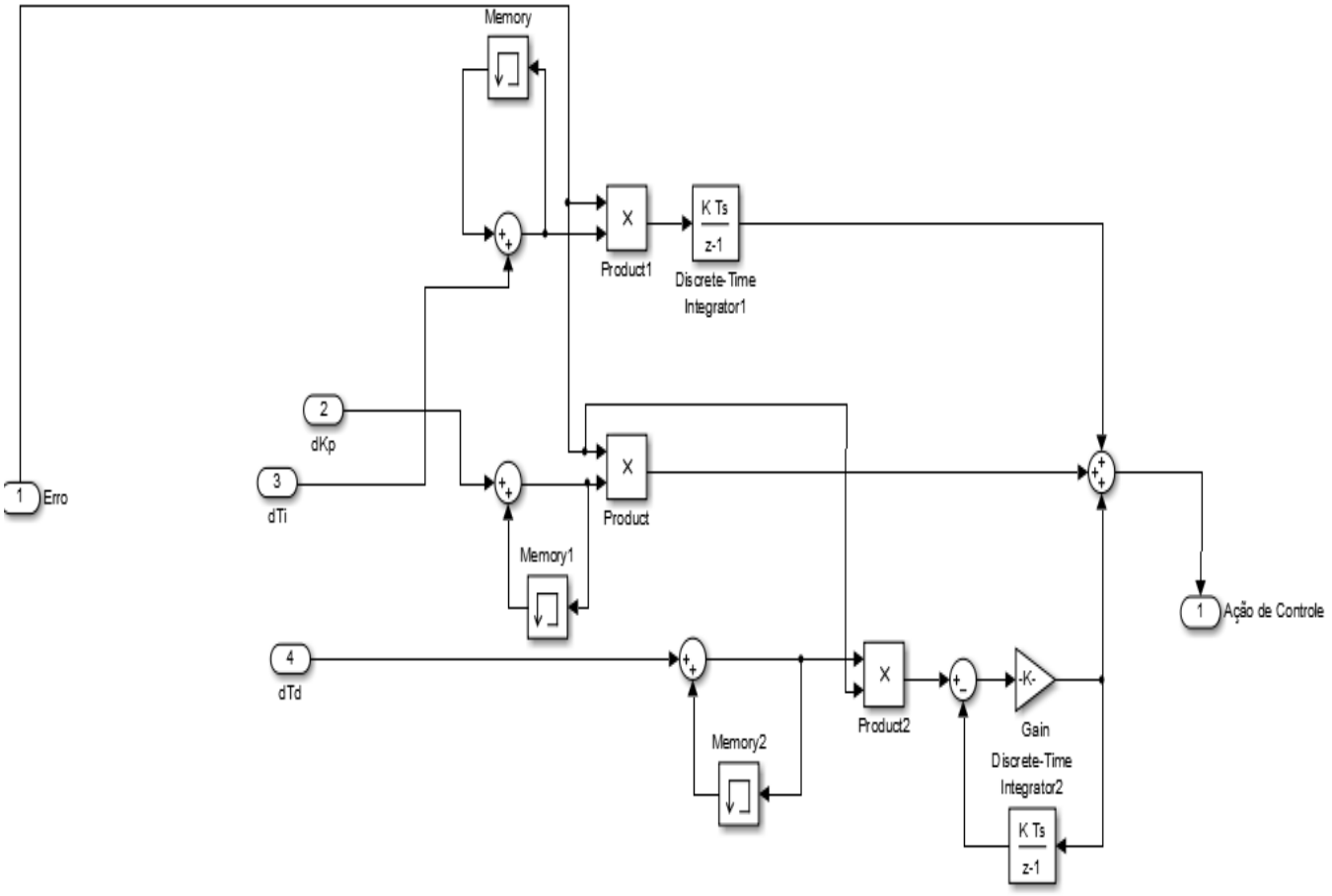

Figura 95 – Controlador PID *Liner*

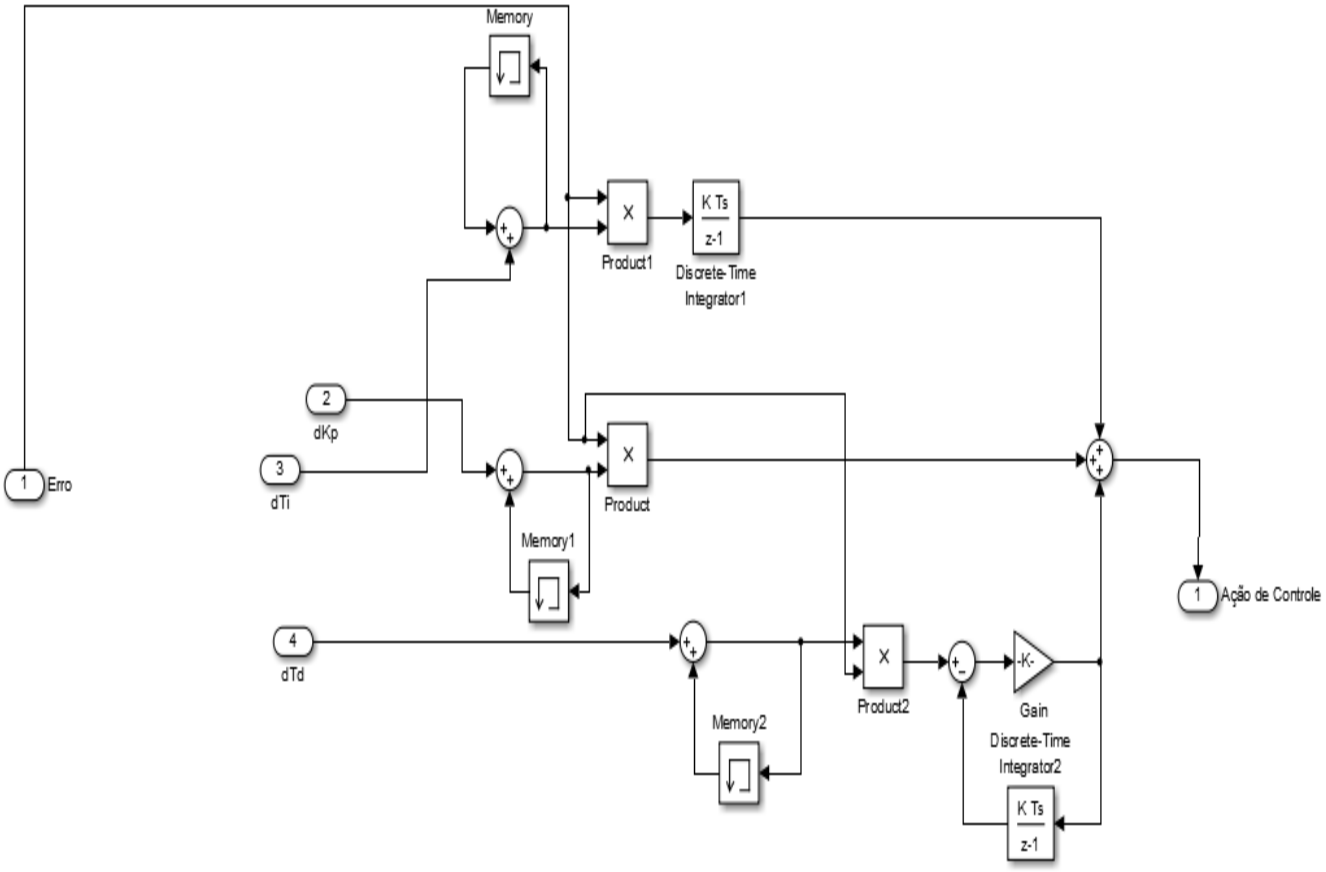

Figura 96 – Controlador PID Cabine

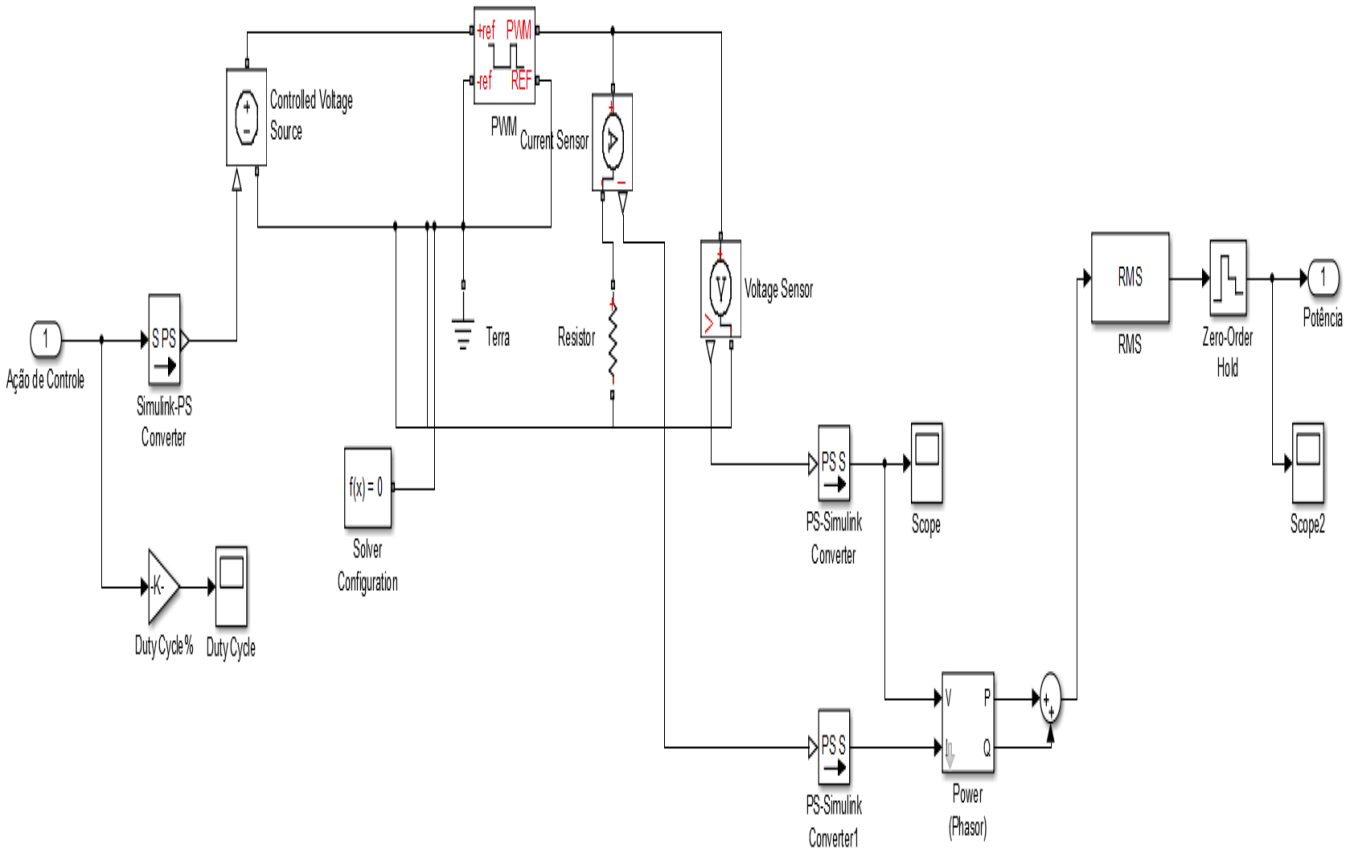

Figura 97 – Modelo Ventilador *Liner*

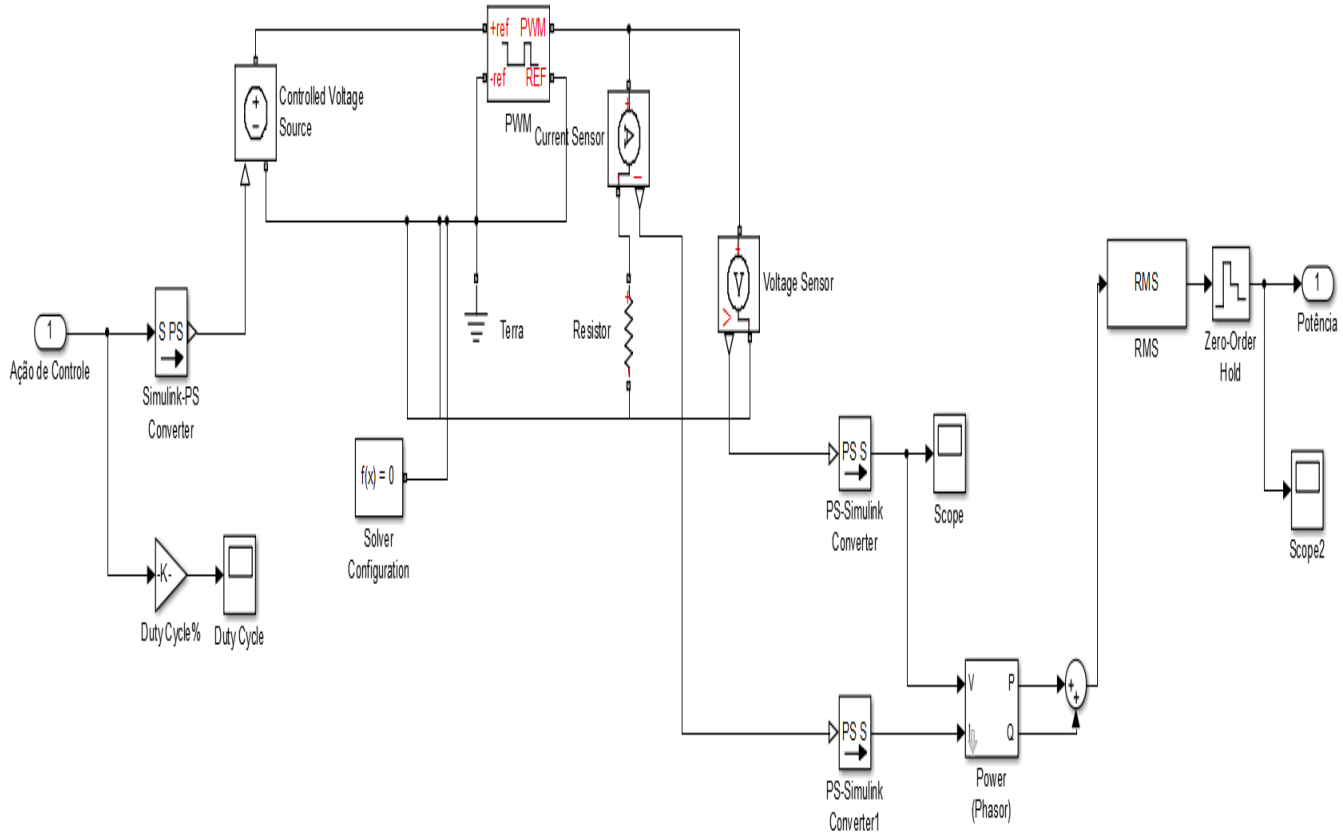

Figura 98 – Modelo Resistência Cabine

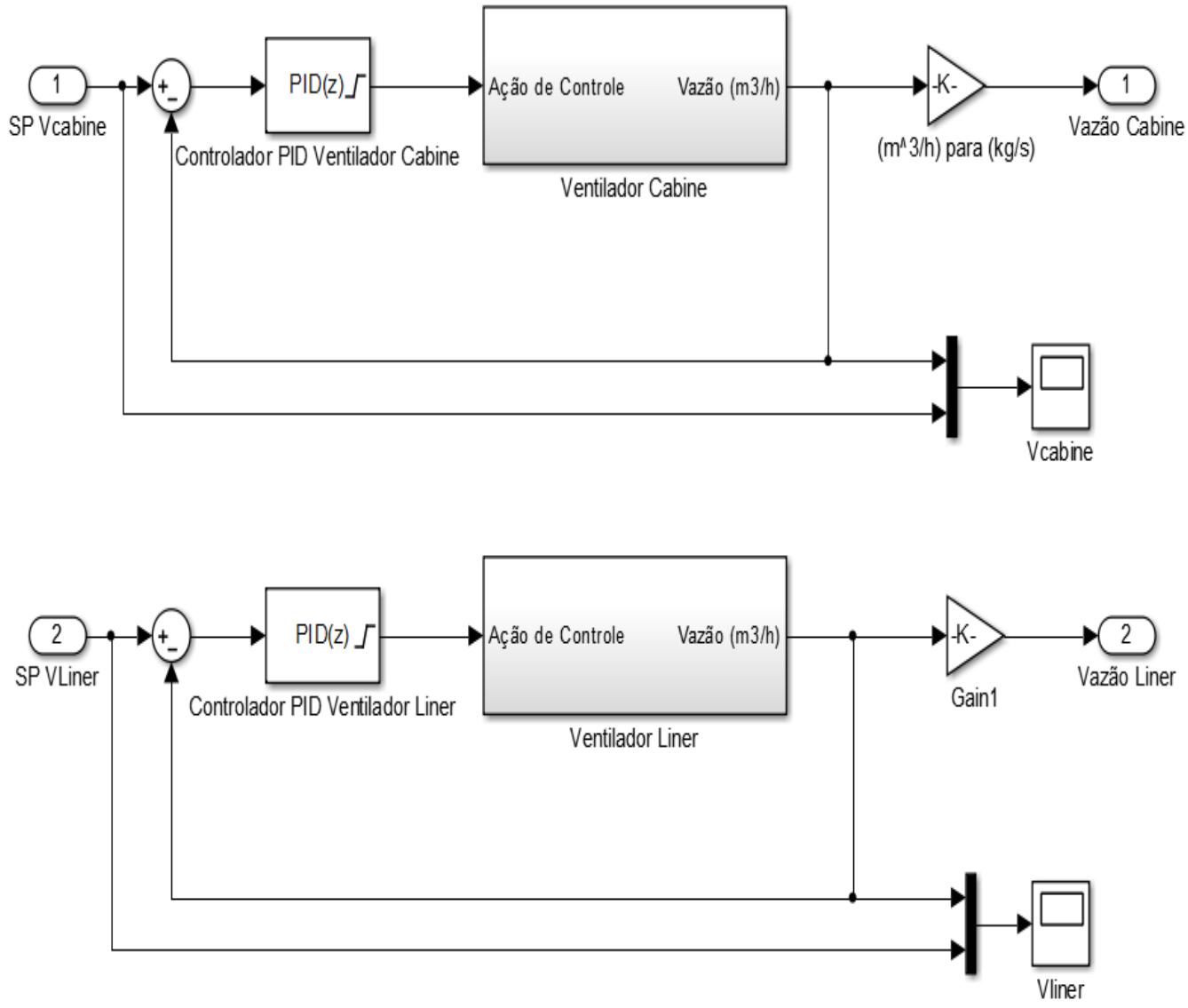

Figura 99 – Malha Resistência *Liner*

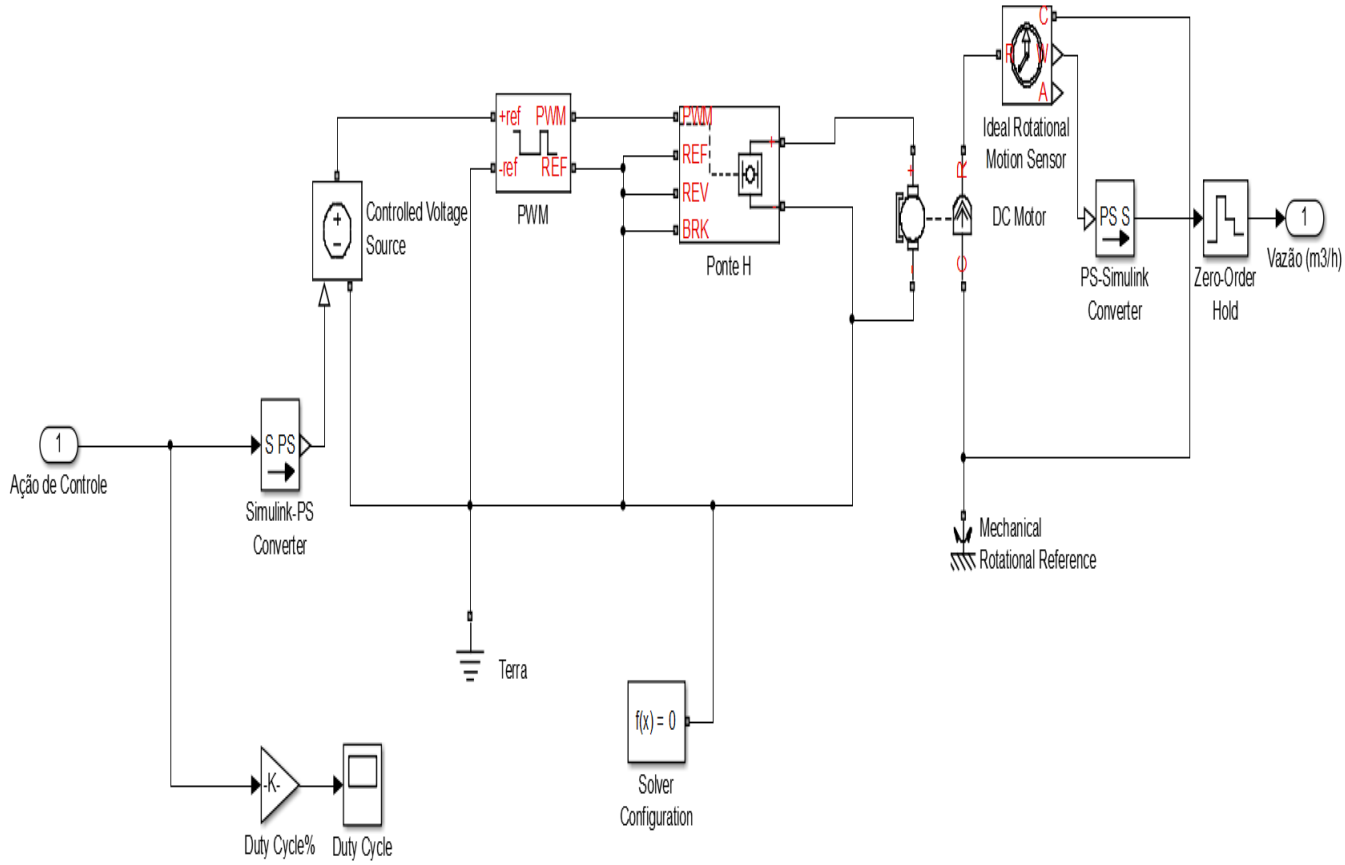

Figura 100 – Malha Ventilador Cabine

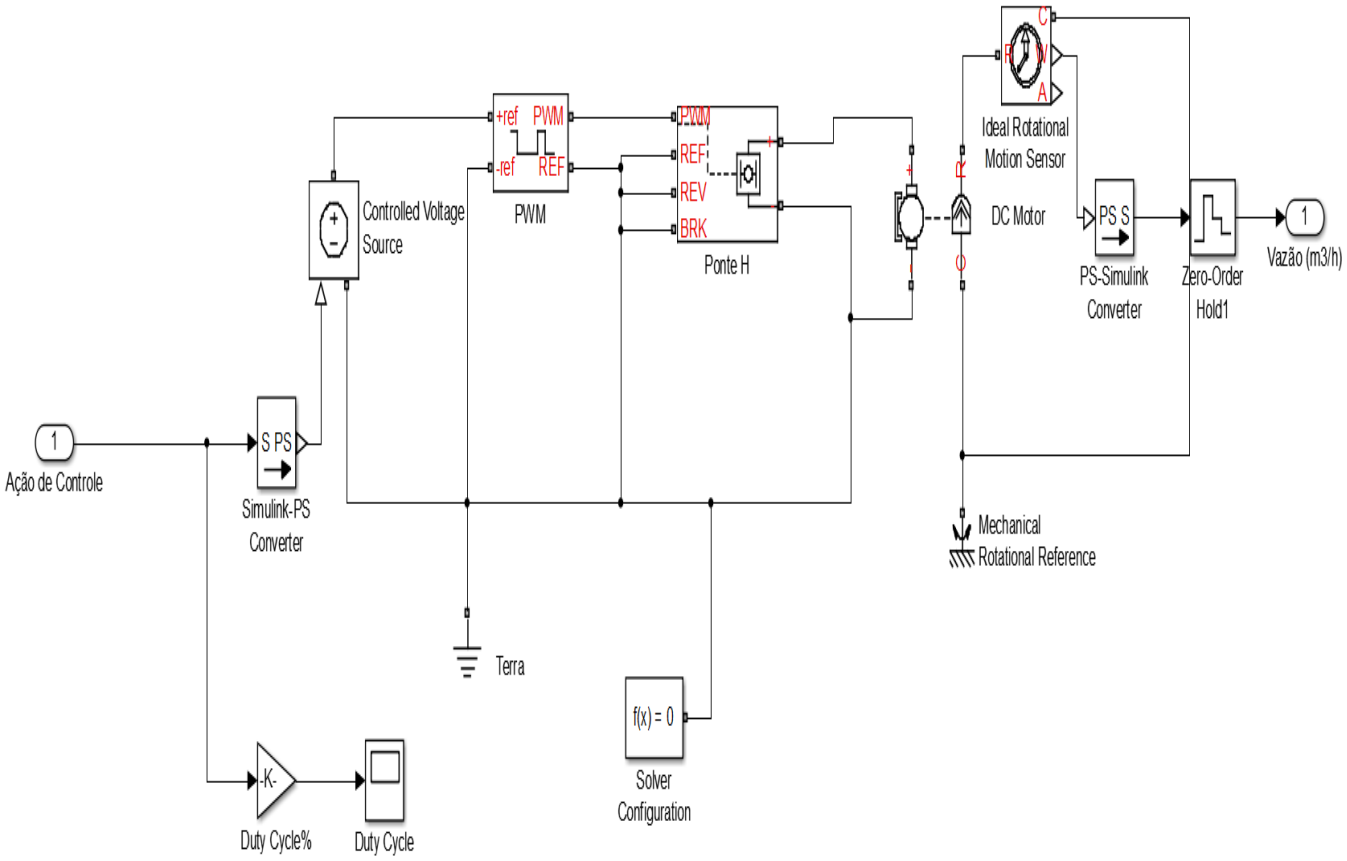

Figura 101 – Modelo Ventilador Liner

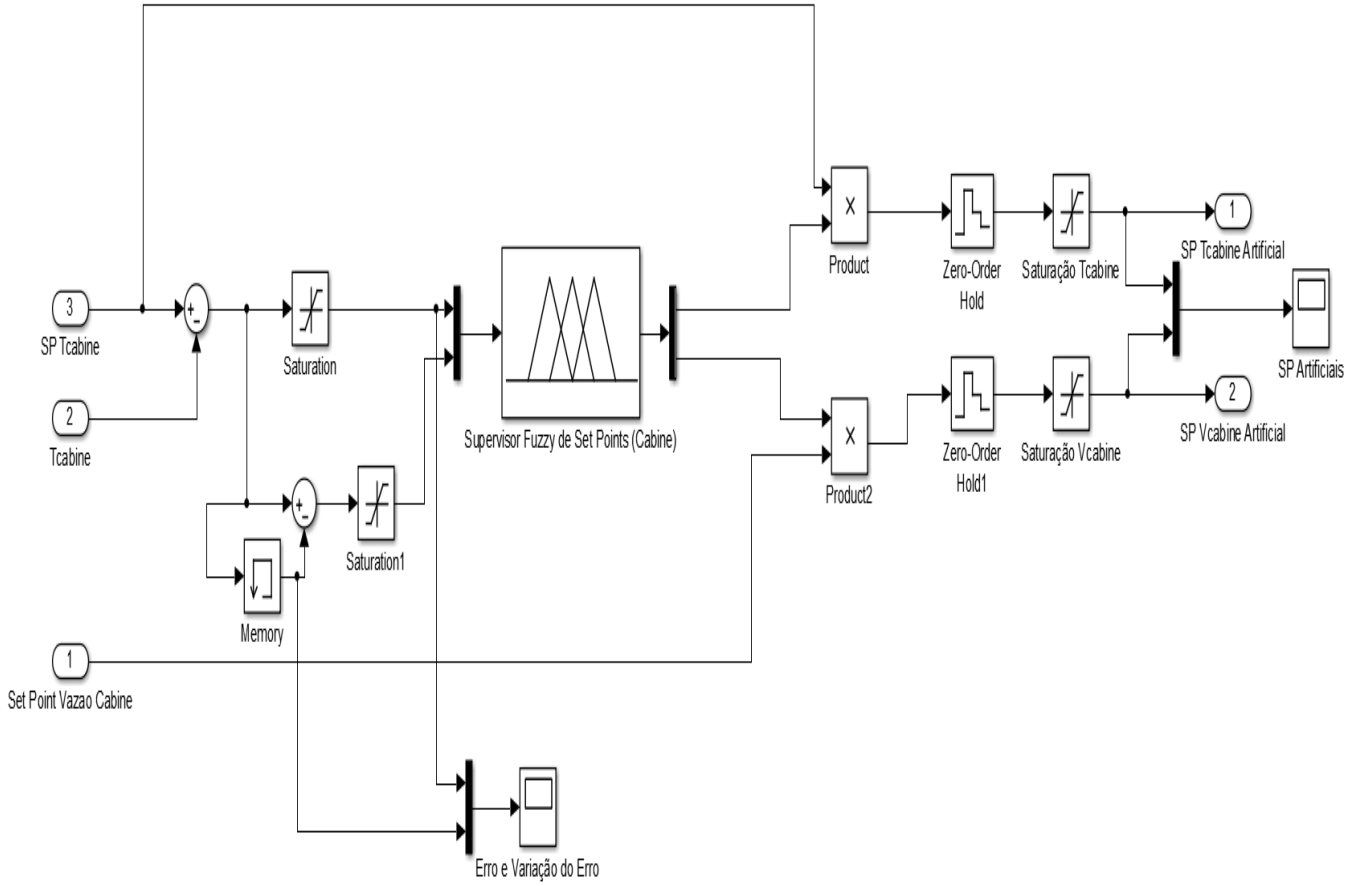

Figura 102 – Supervisor *Set Points Liner*

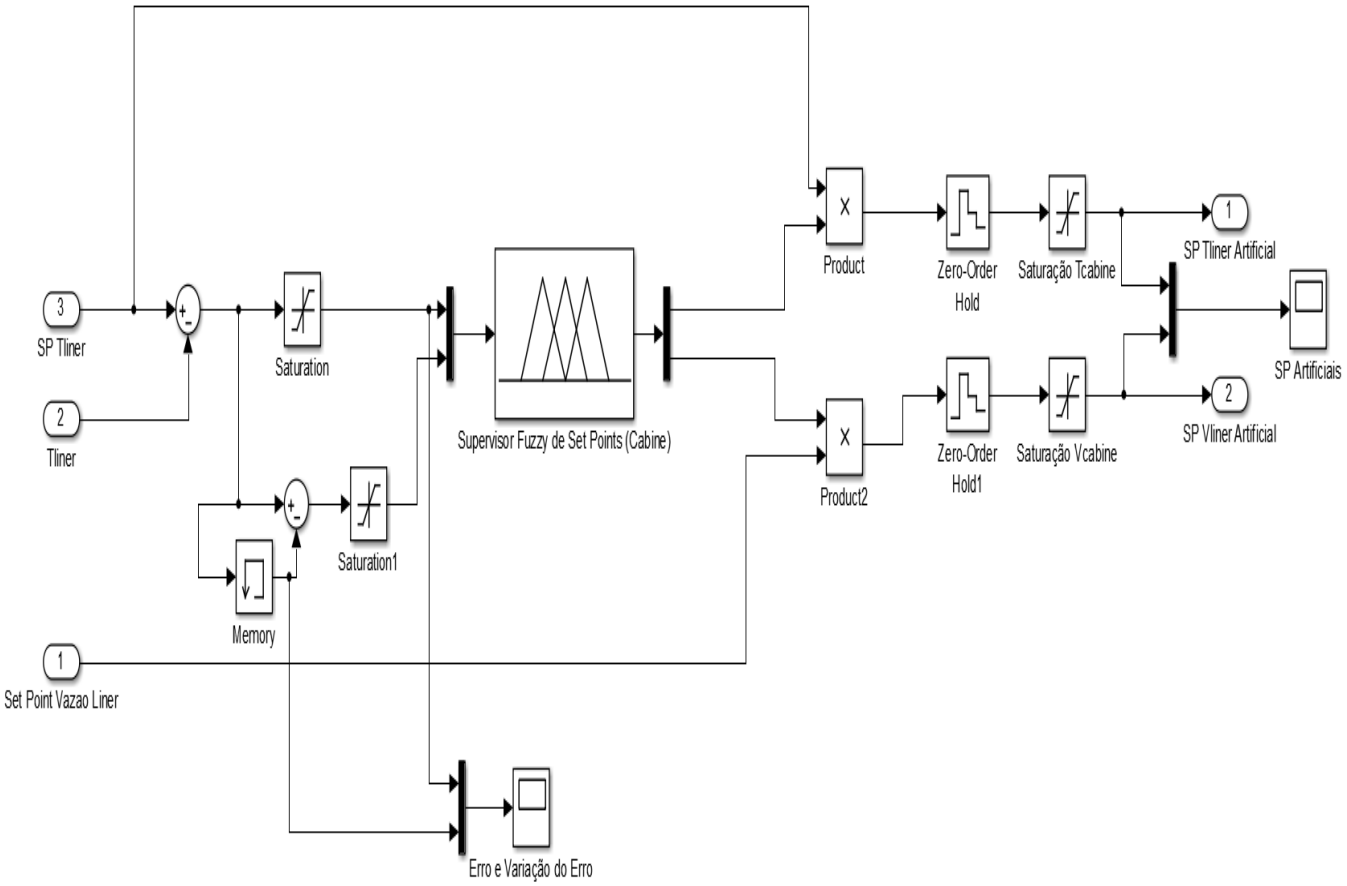

Figura 103 – Supervisor *Set Points* Cabine

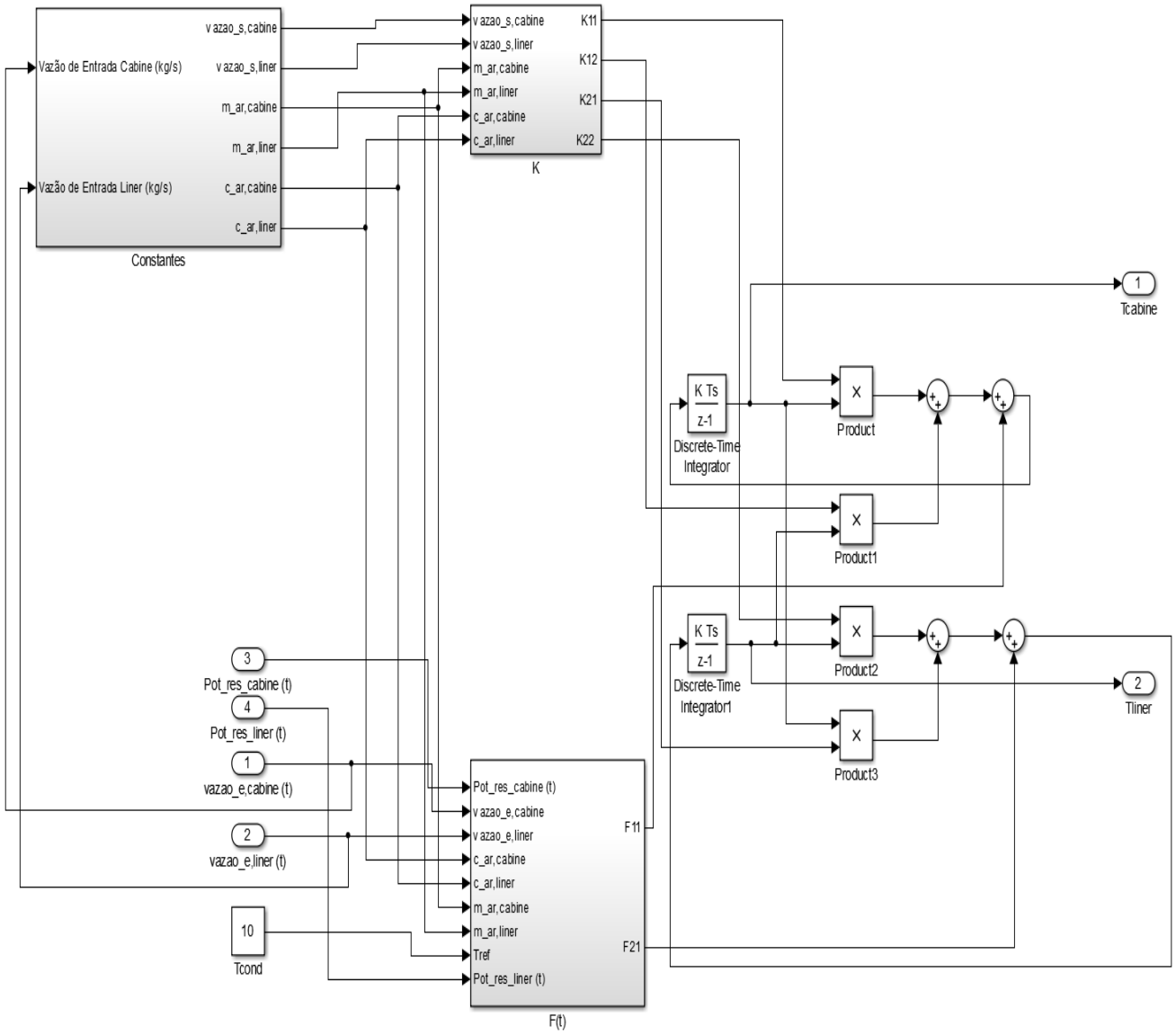

Figura 104 – Modelo *Mock Up*

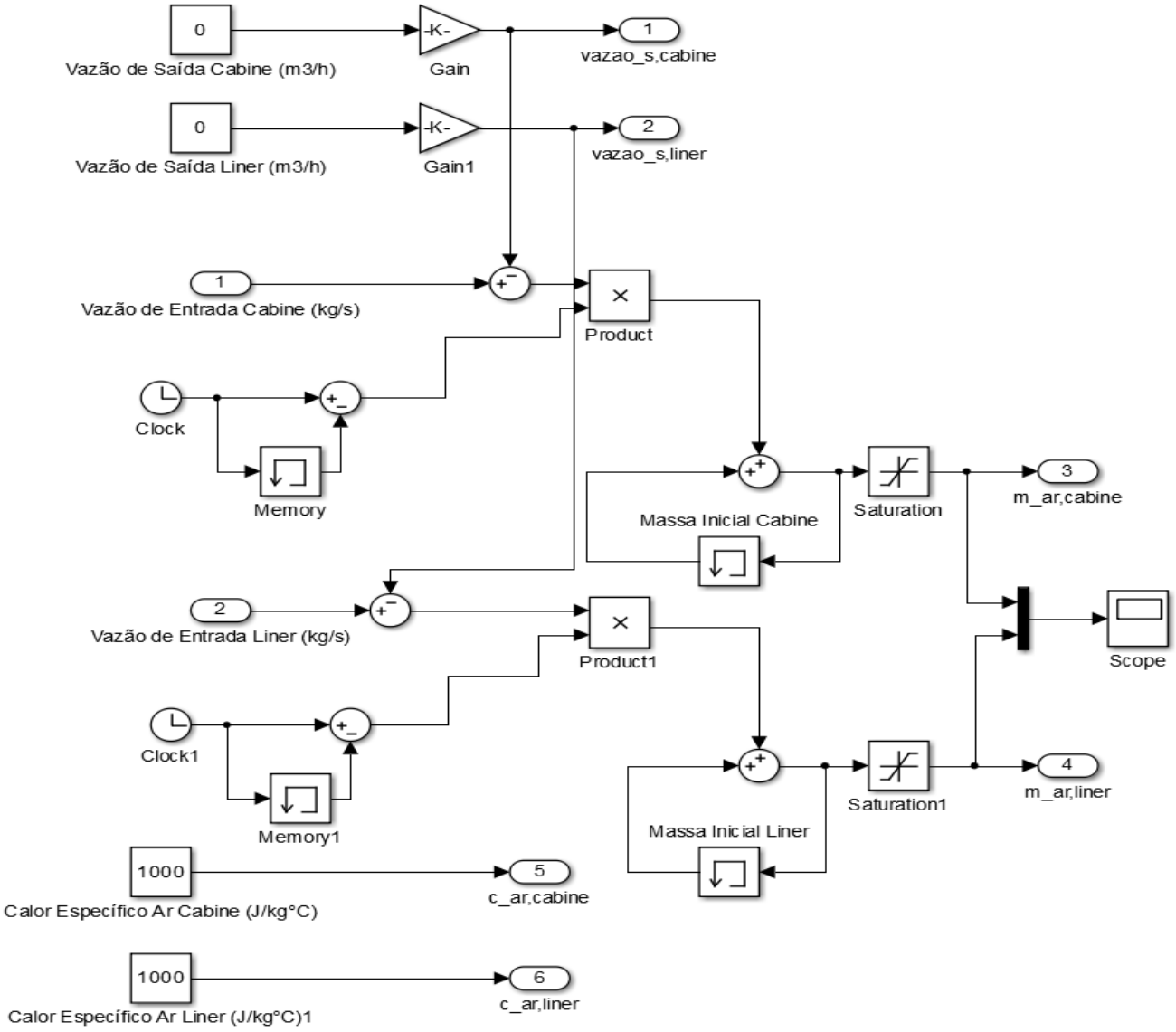

Figura 105 – Bloco de Constantes

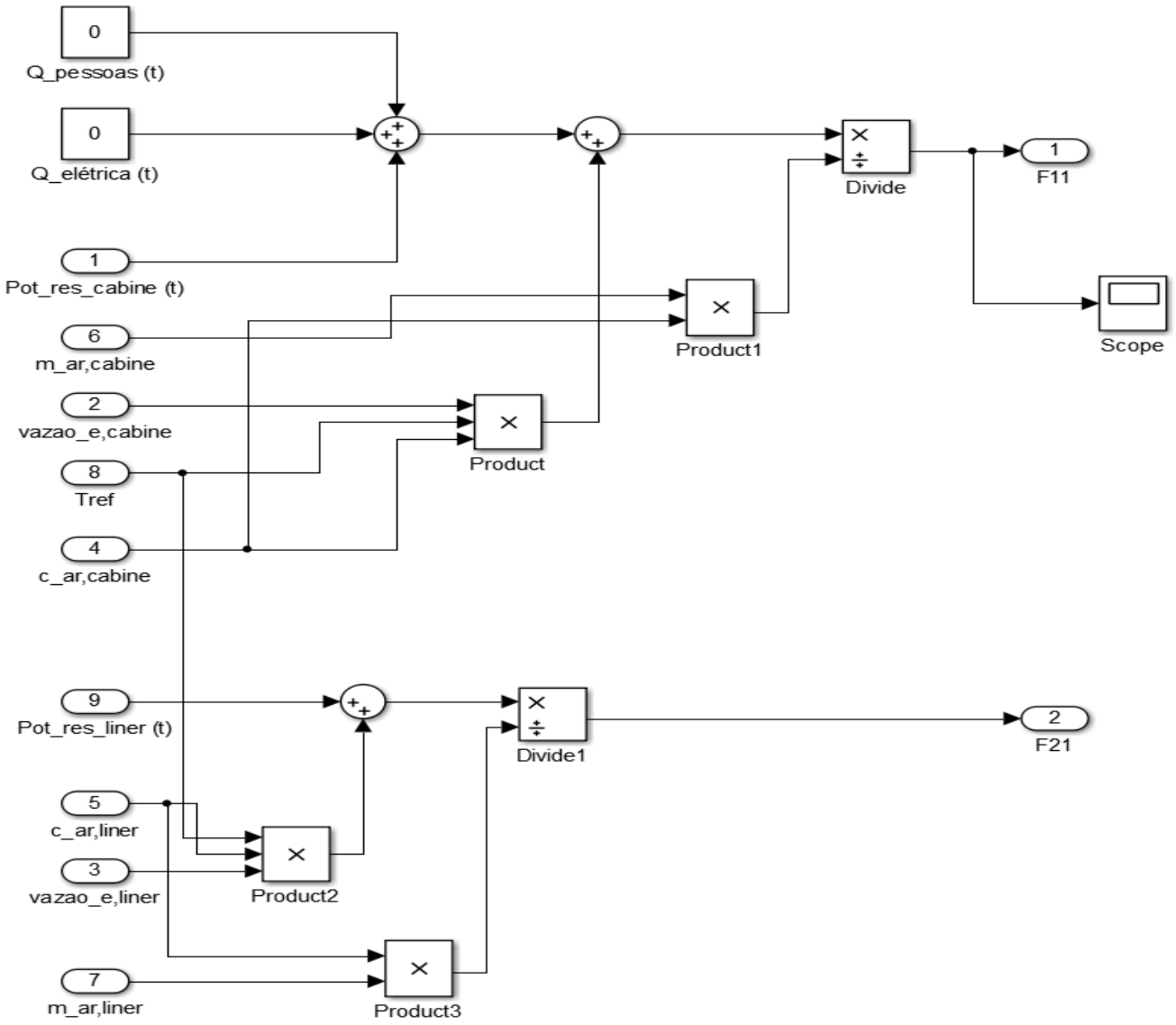

Figura 106 – Bloco de Cargas Térmicas

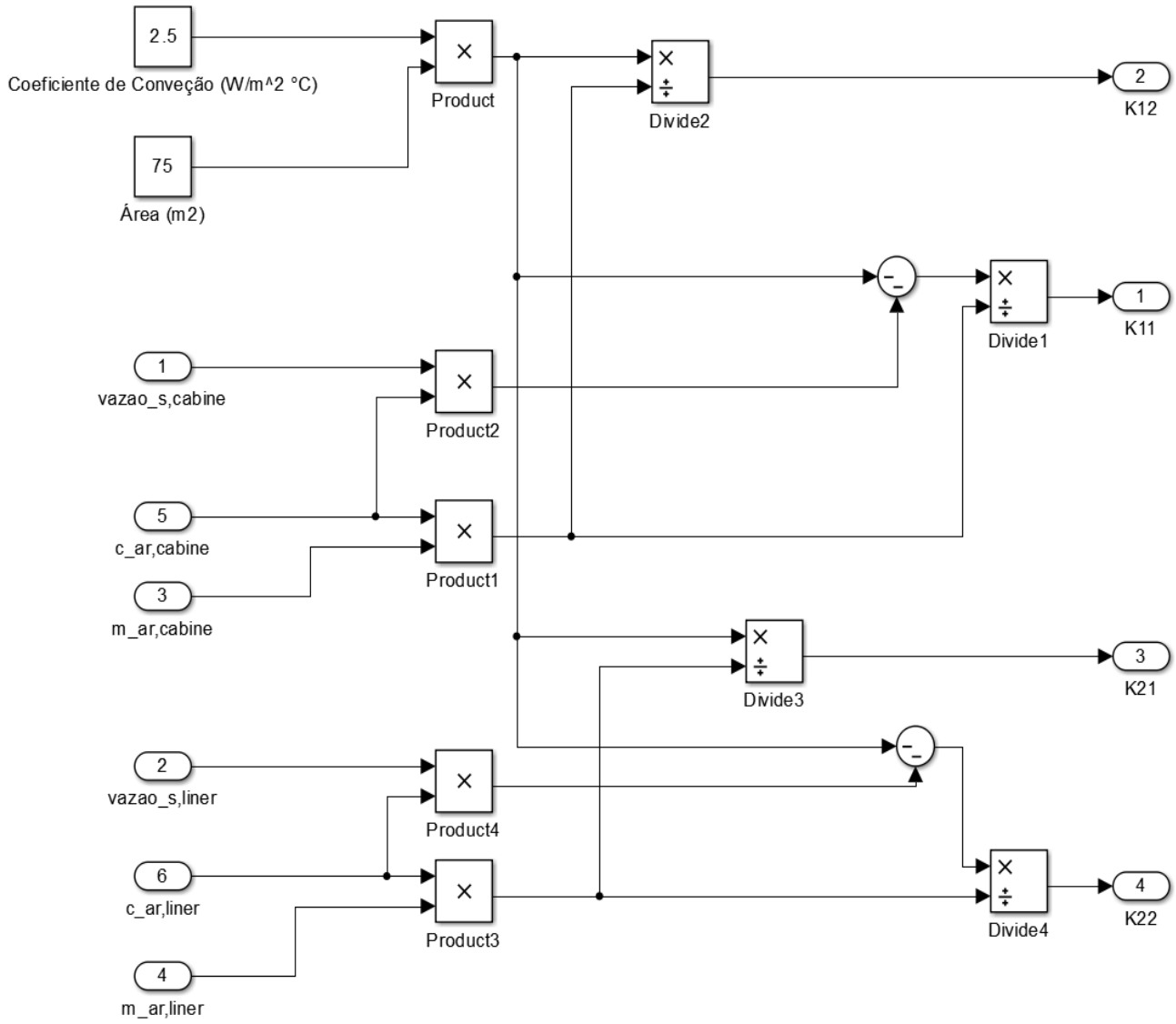

Figura 107 – Bloco de Acoplamento

ANEXO D – Código

Main.c

```
#include <TaskManager.h>

TaskManager _TaskManager;

void setup()
{
 _TaskManager = TaskManager();
}

void loop()
{
        while(1)
                _TaskManager.Run();

}
```

TaskManager

```
/*
 * TaskManager.h
 *
 *  Created on: 23/01/2014
 *      Author: MPN
 */

#ifndef TASKMANAGER_H_
#define TASKMANAGER_H_

#include <Arduino.h>
#include <AirFlowController.h>
#include <SerialController.h>
#include <VentController.h>
#include <TempControl.h>

// Task Manager
static const int TasksNumber = 7;
static const int DhtNumber = 9;
static const int VentNumber = 3;
static const int AirflowSensorNumber = 2;
extern int VENTPIN_HALL[VentNumber];
extern int VENTPIN_PWM[VentNumber];
extern double SETPOINT_PWM[VentNumber];

class TaskManager {
public:
        TaskManager();
```

```
	virtual ~TaskManager();
	void Run();
	void Init();

private:
	SerialController _SerialController;
	AirFlowController _AirFlowController;
	VentController _VentController;
	TempControl _TempController;
	void SerialReceiveMask();
	void ReadTempMask();
	void CalcRotationMask();
	void SetVentPWMMask();
	void GetAirFlowMask();
	void SerialSenderMask();
	void checkSerialCommMask();
	void assignTasks();
	float DHT_READ[DhtNumber-4];
	float AIRFLOW_READ[AirflowSensorNumber];
	void (*TASKS[TasksNumber])(void);
};
#endif /* TASKMANAGER_H_ */


/*
 * TaskManager.cpp
 *
 *  Created on: 23/01/2014
 *      Author: MPN
 */

#include "TaskManager.h"


	int VENTPIN_HALL[VentNumber] = {4,2,3};
	int VENTPIN_PWM[VentNumber] = {8,6,7};
	double SETPOINT_PWM[VentNumber];

TaskManager::TaskManager() {

	int count = 0;

	for(count = 0 ; count < DhtNumber ; count++)
		DHT_READ[count]= 0;
	for(count = 0 ; count < AirflowSensorNumber ; count++)
		AIRFLOW_READ[count]= 0;

	_SerialController.init();
	_TempController = TempControl();
	_TempController.Init();
```

```
	_VentController.Init(VENTPIN_HALL, VENTPIN_PWM,
SETPOINT_PWM);
	_AirFlowController.Init(A2,A3);
	assignTasks();
}

TaskManager::~TaskManager() {
	// TODO Auto-generated destructor stub
}


void TaskManager::Run()
{
	int i = 0;
	for(i = 0; i<TasksNumber; i++)
	{
	 TASKS[i]();
	 delay(1);
	}
}

// Masks to call real controllers' functions
void TaskManager::SerialReceiveMask()
{
	_SerialController.Serial_Receive(SETPOINT_PWM);
}

void TaskManager::ReadTempMask()
{
	_TempController.readTemperatures(DHT_READ);
}

void TaskManager::CalcRotationMask()
{
	_VentController.calculateRotation();
}

void TaskManager::SetVentPWMMask()
{
	_VentController.setVentilatorPWM();
}

void TaskManager::GetAirFlowMask()
{
	_AirFlowController.getAllAirFlowReads(AIRFLOW_READ);
}

void TaskManager::SerialSenderMask()
{
```

```
        _SerialController.Serial_Send(DHT_READ[0],DHT_READ[1],AIRFLOW_
READ[0],AIRFLOW_READ[1]
                                ,DHT_READ[2],DHT_READ[3],DHT_READ[4]);
}

void TaskManager::checkSerialCommMask()
{
        _SerialController.checkSerialCommunication(SETPOINT_PWM);
}

// Assign tasks before running
void TaskManager::assignTasks()
{
TASKS[0] = &SerialReceiveMask;
TASKS[1] = &ReadTempMask;
TASKS[2] = &CalcRotationMask;
TASKS[3] = &SetVentPWMMask;
TASKS[4] = &GetAirFlowMask;
TASKS[5] = &SerialSenderMask;
TASKS[6] = &checkSerialCommMask;
}
```

Serial Controller

```
/*
 * SerialController.h
 *
 *  Created on: 23/01/2014
 *      Author: MPN
 */

#include <Arduino.h>

#ifndef SERIALCONTROLLER_H_
#define SERIALCONTROLLER_H_

#define REQ_BUF_SZ 500
#define ENV_BUF_SZ 23

class SerialController
{

public:
void Init();
void Serial_Send (float ard_temp_cab,float ard_temp_lin, float ard_flow_cab,
float ard_flow_liner,float ard_temp_reslin, float ard_temp_rescab, float
ard_temp_amb);
void Serial_Receive(double* SETPOINT_PWM);
```

```
private:
char Serial_Rx_buffer [REQ_BUF_SZ]; // buffered Serial received data
int req_index;   // index of Serial Receive
bool hasCommStarted;
void processReceiveData(char* header,double* SETPOINT_PWM);
void storeInByte(char inByte);
void StrClear(char *str, char length);
void accelSerialRead(int count);
char StrContains(char *str, char *sfind);
getDigits(int Number, int NumSize, int* NumberDigits);
int timeout;

};
#endif /* SERIALCONTROLLER_H_ */

#include <Arduino.h>
#include <SerialController.h>

/*
 * SerialController.cpp
 *
 *  Created on: 20/01/2014
 *      Author: MPN
 */

void SerialController::init()
{
        Serial.begin(9600);
        Serial1.begin(9600);
        Serial2.begin(9600);
        Serial3.begin(115200);
        req_index = 0;
        hasCommStarted = false;
}

// Serial_Send
void SerialController::Serial_Send (float ard_temp_cab,float ard_temp_lin, float
ard_flow_cab, float ard_flow_liner,
 float ard_temp_reslin, float ard_temp_rescab, float ard_temp_amb)
{
  int TEMP_CAB = (int)(10*ard_temp_cab);
  int TEMP_LIN = (int)(10*ard_temp_lin);
  int FLOW_CAB = (int)(10*ard_flow_cab);
  int FLOW_LIN = (int)(10*ard_flow_liner);
  int TEMP_RES_CAB = (int)(10*ard_temp_rescab);
  int TEMP_RES_LIN= (int)(10*ard_temp_reslin);
  int TEMP_RES_AMB = (int)(10*ard_temp_amb);

  int TEMP_LIN_DIGITS[3];
  int TEMP_CAB_DIGITS[3];
```

```
  int FLOW_LIN_DIGITS[3];
  int FLOW_CAB_DIGITS[3];
  int TEMP_RESLIN_DIGITS[3];
  int TEMP_RESCAB_DIGITS[3];
  int TEMP_RESAMB_DIGITS[3];

  getDigits(TEMP_LIN,3,TEMP_LIN_DIGITS);
  getDigits(TEMP_CAB,3,TEMP_CAB_DIGITS);
  getDigits(FLOW_LIN,3,FLOW_LIN_DIGITS);
  getDigits(FLOW_CAB,3,FLOW_CAB_DIGITS);
  getDigits(TEMP_RES_CAB,3,TEMP_RESLIN_DIGITS);
  getDigits(TEMP_RES_LIN,3,TEMP_RESCAB_DIGITS);
  getDigits(TEMP_RES_AMB,3,TEMP_RESAMB_DIGITS);

   char Serial_Tx_Buffer[ENV_BUF_SZ];

  Serial_Tx_Buffer[0] =  TEMP_CAB_DIGITS[0]+ '0';
  Serial_Tx_Buffer[1] =  TEMP_CAB_DIGITS[1]+ '0';
  Serial_Tx_Buffer[2] =  TEMP_CAB_DIGITS[2]+ '0';
  Serial_Tx_Buffer[3] =  TEMP_LIN_DIGITS[0]+ '0';
  Serial_Tx_Buffer[4] =  TEMP_LIN_DIGITS[1]+ '0';
  Serial_Tx_Buffer[5] =  TEMP_LIN_DIGITS[2]+ '0';
  Serial_Tx_Buffer[6] =  FLOW_CAB_DIGITS[0]+ '0';
  Serial_Tx_Buffer[7] =  FLOW_CAB_DIGITS[1]+ '0';
  Serial_Tx_Buffer[8] =  FLOW_CAB_DIGITS[2]+ '0';
  Serial_Tx_Buffer[9] =  FLOW_LIN_DIGITS[0]+ '0';
  Serial_Tx_Buffer[10] = FLOW_LIN_DIGITS[1]+ '0';
  Serial_Tx_Buffer[11] = FLOW_LIN_DIGITS[2]+ '0';
  Serial_Tx_Buffer[12] = TEMP_RESCAB_DIGITS[0]+ '0';
  Serial_Tx_Buffer[13] = TEMP_RESCAB_DIGITS[1]+ '0';
  Serial_Tx_Buffer[14] = TEMP_RESCAB_DIGITS[2]+ '0';
  Serial_Tx_Buffer[15] = TEMP_RESLIN_DIGITS[0]+ '0';
  Serial_Tx_Buffer[16] = TEMP_RESLIN_DIGITS[1]+ '0';
  Serial_Tx_Buffer[17] = TEMP_RESLIN_DIGITS[2]+ '0';
  Serial_Tx_Buffer[18] = TEMP_RESAMB_DIGITS[0]+ '0';
  Serial_Tx_Buffer[19] = TEMP_RESAMB_DIGITS[1]+ '0';
  Serial_Tx_Buffer[20] = TEMP_RESAMB_DIGITS[2]+ '0';
  Serial_Tx_Buffer[21] = '1';
  Serial_Tx_Buffer[22] = '\0';

    Serial2.print("REC_MATLAB");
    delay(1);
    Serial2.print(Serial_Tx_Buffer);
    delay(1);
    Serial2.print("\r\n");
    delay(1);
//

  StrClear(Serial_Tx_Buffer, ENV_BUF_SZ);
}
```

```
// Serial Receive
void SerialController::Serial_Receive(double* SETPOINT_PWM)
{
   if (Serial3.available()>0) {  // got Serial

        accelSerialRead(1);
        //char c = Serial3.read(); // read 1 byte (character) from matlab
        //storeInByte(c);
        //Serial.print(c);

        if(StrContains(Serial_Rx_buffer, "ENV_MATLAB"))
        {
            StrClear(Serial_Rx_buffer, req_index);
            req_index = 0;
            hasCommStarted = true;
            Serial.write("Init Command detected");
        }

        if(StrContains(Serial_Rx_buffer, "VCPT") && hasCommStarted)
        {
          processReceiveData("VCPT",SETPOINT_PWM); // Cabine
          accelSerialRead(4);
        }

        if(StrContains(Serial_Rx_buffer, "VKIX") && hasCommStarted)
        {
          processReceiveData("VKIX",SETPOINT_PWM);  // Liner
Esquerdo/Vermelho
          accelSerialRead(4);
        }

          if(StrContains(Serial_Rx_buffer, "VAZR") && hasCommStarted)
        {
          processReceiveData("VAZR",SETPOINT_PWM); // Liner Direito/Azul
          accelSerialRead(4);
        }
  } // Serial available
} // Serial receive end


// accelerate the Serial read
void SerialController::accelSerialRead(int count)
{
  char inByte;
  int i = 0;
  for( i = 0 ; i < count ; i++)
  {
    if (Serial3.available()>0)
    {
```

```
      inByte = Serial3.read();
      storeInByte(inByte);
    }
  }
}
// clears array
void SerialController::StrClear(char *str, char length)
{
    for (int i = 0; i < length; i++) {
        str[i] = 0;
    }
}

// searches for the string sfind in the string str
// returns 1 if string found
// returns 0 if string not found
char SerialController::StrContains(char *str, char *sfind)
{
    char found = 0;
    char index = 0;
    char len;

    len = strlen(str);

    if (strlen(sfind) > len) {
        return 0;
    }
    while (index < len) {
        if (str[index] == sfind[found]) {
            found++;
            if (strlen(sfind) == found) {
                return 1;
            }
        }
        else {
            found = 0;
        }
        index++;
    }

    return 0;
}

// store the specified inByte
void SerialController::storeInByte(char inByte)
{
 if (req_index < (REQ_BUF_SZ - 1)) {
       // Serial.print(inByte);
    if(inByte != (char)-1)
    {
```

```
    Serial_Rx_buffer[req_index] = inByte;
     req_index++;
   }
 }
}

// Process numeric received data
void SerialController::processReceiveData(char* header, double*
SETPOINT_PWM)
{
  StrClear(Serial_Rx_buffer, req_index);
  req_index = 0;

  boolean messageIsOver= false;
  char inByte = '\0';
  char lastinByte;

  while(!messageIsOver)
  {
    lastinByte = inByte;
    inByte = Serial3.read();
    //Serial.print(inByte);

    if ((inByte == 10 && lastinByte == 13) ||req_index == 2){

      char Valor[req_index+1];

      short u = 0;

      for(u = 0; u < req_index ; u++)
      {
        Valor[u] = Serial_Rx_buffer[u];
      }

      Valor[req_index] = '\0';
      StrClear(Serial_Rx_buffer, req_index);
      req_index = 0;

      if(StrContains(header, "VCPT"))
        SETPOINT_PWM[0] = atoi(Valor);
      else if(StrContains(header,"VKIX"))
        SETPOINT_PWM[1] = atoi(Valor);
      else if(StrContains(header,"VAZR"))
        SETPOINT_PWM[2] = atoi(Valor);
      messageIsOver = true;
   }

    else if ((inByte == '\r') || (inByte == '\n' && lastinByte != '\r'))
    {
         //req_index -= 1; // ignore inbyte
```

```
      }
      else
      {
         if(inByte >= 48 && inByte <=57 && req_index <= 2)
         {
            storeInByte(inByte);
         }
      }
   }
}

// getDigits from a number
void SerialController::getDigits(int Number, int NumSize, int* NumberDigits)
{
  float FNumber = (float)Number;

    for (int i=NumSize-1; i>=0; i--) {
      float y = (float)(pow(10, i));
      float z = FNumber/y;
      float x2 = FNumber/(y*10.0000f);
      int x3 = 10*floor(x2);
      NumberDigits[NumSize-i-1] = floor(z) - x3;
    }
}

//check Serial Communication. If Serial is offline, turn off all devices.
void SerialController::checkSerialCommunication()
{
  if(Serial3.available() <= 0)
     timeout++;
  else
     timeout = 0;

  if(timeout >= 100)
  {
     SETPOINT_PWM[0] = 0;
     SETPOINT_PWM[1] = 0;
     SETPOINT_PWM[2] = 0;
     timeout = 0;
  }
}
```

VentController

```
/*
 * VentController.h
 *
 *  Created on: 22/01/2014
 *      Author: MPN
 */

#ifndef VENTCONTROLLER_H_
#define VENTCONTROLLER_H_

#include <VentHandler.h>
#include <PID_v1.h>
#include <TimerThree.h>

class VentController
{

const static int VENT_NUM = 3;

public:
VentController();
void InterruptWorker();
void Init(int* ventpin_hall, int* ventpin_pwm, double* setpoint_rot);
void calculateRotation();
void setVentilatorPWM();

private:
void _setVentilatorPWM(int vent1PWM, int vent2PWM, int vent3PWM);
void computeVentPID();
void initPIDs();
void getPWMRatioForVent(double* ROT_MATRIX, double *PWM_MATRIX);
void getVentilatorRotation();
int VENTPIN_HALL[VENT_NUM];
int VENTPIN_PWM[VENT_NUM];
double VENT_PWM[VENT_NUM];
double VENT_ROT[VENT_NUM];
double INPUT_PWM[VENT_NUM];
double INPUT_ROT[VENT_NUM];
VentHandler Vent[VENT_NUM];

// PIDs
PID VENT_PIDs[VENT_NUM];

};

#endif /* VENTCONTROLLER_H_ */

/*
```

```
 * VentController.cpp
 *
 *  Created on: 22/01/2014
 *      Author: MPN
 */
#include <Arduino.h>
#include "VentController.h"


void VentController::Init(int* ventpin_hall, int* ventpin_pwm, double*
SETPOINT_PWM)
{
        int pincount = 0;
        for(pincount = 0 ; pincount < VENT_NUM ; pincount++)
        {
                VENTPIN_HALL[pincount] = ventpin_hall[pincount];
                VENTPIN_PWM[pincount]= ventpin_pwm[pincount];
        }

        VentHandler vent1(VENTPIN_PWM[0],VENTPIN_HALL[0]);
        VentHandler vent2(VENTPIN_PWM[1],VENTPIN_HALL[1]);
        VentHandler vent3(VENTPIN_PWM[2],VENTPIN_HALL[2]);

        Vent[0] = vent1;
        Vent[1] = vent2;
        Vent[2] = vent3;

        PID VENT1_PID(&INPUT_PWM[0],&VENT_PWM[0],
&SETPOINT_PWM[0] ,0.4, 0.8, 0.3, DIRECT);
        PID VENT2_PID(&INPUT_PWM[1],&VENT_PWM[1],
&SETPOINT_PWM[1] ,0.4,0.8,0.3, DIRECT);
        PID VENT3_PID(&INPUT_PWM[2],&VENT_PWM[2],
&SETPOINT_PWM[2] ,6, 1.2, 0, DIRECT);

        VENT_PIDs[0] = VENT1_PID;
        VENT_PIDs[1] = VENT2_PID;
        VENT_PIDs[2] = VENT3_PID;

        initPIDs();
        Timer3.initialize(1000);
        Timer3.attachInterrupt(this->InterruptWorker, 1000);
}

void VentController::InterruptWorker()
{
          getVentilatorRotation();
          computeVentPID();
}

void VentController::computeVentPID()
```

```
{
        VENT_PIDs[0].Compute();
        VENT_PIDs[1].Compute();
        VENT_PIDs[2].Compute();
}

void VentController::initPIDs()
{
        VENT_PIDs[0].SetMode(AUTOMATIC);
        VENT_PIDs[0].SetOutputLimits(0,100);
        VENT_PIDs[1].SetMode(AUTOMATIC);
        VENT_PIDs[1].SetOutputLimits(0,100);
        VENT_PIDs[2].SetMode(AUTOMATIC);
        VENT_PIDs[2].SetOutputLimits(0,100);
}

void VentController::getPWMRatioForVent(double* ROT_MATRIX, double
*PWM_MATRIX)
{
 int count = 0;

 for(count = 0 ; count < VENT_NUM ; count++)
 {
   int a = 1;
   int b = 1;
   switch(count)
   {
   case 0:
      a = 16.53;
      b = 679.18;
      break;
   case 1:
     a = 18.99;
     b = 411.96;
     break;
   case 2:
     a = 11.64;
     b = 1113.2;
     break;
   }
   PWM_MATRIX[count] = (ROT_MATRIX[count] - b)/a;
 }
}

void VentController::setVentilatorPWM()
{
        _setVentilatorPWM(VENT_PWM[0], VENT_PWM[1],VENT_PWM[2]);
}
```

```
void VentController::_setVentilatorPWM(int vent1PWM, int vent2PWM, int
vent3PWM)
{
        Vent[0].setPWM(vent1PWM);
        Vent[1].setPWM(vent2PWM);
        Vent[2].setPWM(vent3PWM);
}


void VentController::getVentilatorRotation()
{
        Vent[0].MagnetDetector();
        Vent[1].MagnetDetector();
        Vent[2].MagnetDetector();
}

void VentController::calculateRotation()
{
        Vent[0].calcRotation();
        Vent[1].calcRotation();
        Vent[2].calcRotation();

  INPUT_ROT[0] = Vent[0].getRotation();
  INPUT_ROT[1] = Vent[1].getRotation();
  INPUT_ROT[2] = Vent[2].getRotation();

  getPWMRatioForVent(INPUT_ROT,INPUT_PWM);
}
```

VentHandler

```
#ifndef VENTHANDLER_H
#define VENTHANDLER_H

#include <Arduino.h>

class VentHandler{
  public:
    // constructors:
    VentHandler();
        VentHandler(int dir1PinA, int dir2PinA, int speedPin, int enPin, int SH);
        VentHandler(int speedPin,int SH);
    void enable(bool en);
    void setDirection(bool dir);
    void setPWM(int rate);
    void MagnetDetector();
        long getRotation();
        void calcRotation();
```

```
  private:
        int dir1PinA;
    int dir2PinA;
    int speedPin;
    int enablePin;
    int SensorHallPin;
    int NumberOfMagnets;
    long rpm;
    volatile long rpmcount;
    unsigned long timeold;
    unsigned long time;
    unsigned long interrupt_time;
    unsigned long last_interrupt_time;
    volatile bool _HALLSTATE;
        void setPwmFrequency(int pin, int prescaler);


};

#endif

#include <Arduino.h>
#include <VentHandler.h>

//Construtor

VentHandler::VentHandler()
{
  // Default Constructor
}

VentHandler::VentHandler(int dir1PinA, int dir2PinA, int speedPin, int enPin, int
SH)
{
    this->dir1PinA = dir1PinA;
    this->dir2PinA = dir2PinA;
    this->speedPin = speedPin;
    this->enablePin = enPin;
    this->SensorHallPin = SH;

    pinMode(SensorHallPin,INPUT);
    pinMode(dir1PinA, OUTPUT);
    pinMode(dir2PinA, OUTPUT);
    pinMode(enablePin,OUTPUT);
    pinMode (speedPin, OUTPUT);

    NumberOfMagnets = 2;
    rpm = 0;
    rpmcount = 0;
}
```

```
VentHandler::VentHandler(int speedPin,int SH)
{

    this->speedPin = speedPin;
    this->SensorHallPin = SH;

    pinMode(SensorHallPin,INPUT);
    pinMode(speedPin, OUTPUT);

    NumberOfMagnets = 2;
    rpm = 0;
    rpmcount = 0;

     setPwmFrequency(speedPin, 1);
}

// enable or disable the cooler
void VentHandler::enable(bool en)
{
    digitalWrite(this->enablePin,en);
}

// set the Cooler direction
void VentHandler::setDirection(bool DIR)
{
  if(DIR == true)
  {
          digitalWrite(this->dir1PinA, LOW);
          digitalWrite(this->dir2PinA, HIGH);
  }

  else{
          digitalWrite(this->dir1PinA, HIGH);
          digitalWrite(this->dir2PinA, LOW);
          }
}

// generate a PWM signal with the specified rate
void VentHandler::setPWM(int rate)
{
    int analogInput = (rate*255)/100;
    analogWrite(speedPin, analogInput);
}

// get Rotation
long VentHandler::getRotation()
{
 return (rpm);
}
```

```
// calculate the rotation
void VentHandler::calcRotation()
{
    if(rpmcount >= NumberOfMagnets)
    {
     time = millis();
     unsigned long calc = (rpmcount/NumberOfMagnets)*60000/(time-timeold);
     rpm = calc;
     rpmcount = 0;
     timeold = time;

           Serial.print(millis());
     Serial.print(',');
     Serial.print(rpm);
     Serial.print("\r\n");
    }
}

// detect the magnets using Hall sensor
void VentHandler::MagnetDetector(){

  bool MagnetDetected = LOW;
  bool _LastHallState = _HALLSTATE ;
  unsigned long dif;
  _HALLSTATE = digitalRead(SensorHallPin);

  if (_HALLSTATE == LOW && _LastHallState == HIGH){
    interrupt_time = millis();
    dif = interrupt_time - last_interrupt_time;

    if (dif >= 10) // debounce
    {
      rpmcount++;
     // calcRotation();
    }
    last_interrupt_time = interrupt_time;
  }
 // return MagnetDetected;
}

void VentHandler::setPwmFrequency(int pin, int prescaler) {

  //prescaler = 1 ---> PWM frequency is 31000 Hz
  //prescaler = 2 ---> PWM frequency is 4000 Hz
  //prescaler = 3 ---> PWM frequency is 490 Hz (default value)
  //prescaler = 4 ---> PWM frequency is 120 Hz
  //prescaler = 5 ---> PWM frequency is 30 Hz
  //prescaler = 6 ---> PWM frequency is <20 Hz
```

```
  int Eraser = 7;
  if(pin == 9 || pin == 10) {
    TCCR2B &= ~Eraser;
    TCCR2B |=prescaler;
  }
  else if(pin == 2 || pin == 3 || pin == 5) {
    TCCR3B &= ~Eraser;
    TCCR3B |=prescaler;
  }
  else if(pin == 6 || pin == 7 || pin == 8) {
    TCCR4B &= ~Eraser;
    TCCR4B |=prescaler;
  }
}
```

TempControl

```
/*
 * TempControl.h
 *
 *  Created on: 21/01/2014
 *      Author: MPN
 */

#include <Arduino.h>
#include <DHT.h>
#include <PID_v1.h>
#include <VentControl.h>
#include <TimerThree.h>

#ifndef TEMPCONTROL_H_
#define TEMPCONTROL_H_

#define DHTTYPE 22

class TempControl
{
public:

        TempControl(int dht_num);
        void Init();
        void readTemperatures(float* DHT_TEMP);
private:

        static const int DHT_NUM = 9;
        int DHTPIN[DHT_NUM];
        float DHT_READ[DHT_NUM-4];
        DHT dht_sensors[DHT_NUM];
};
```

```
#endif /* TEMPCONTROL_H_ */

#include <TempControl.h>
#include <DHT.h>
/*
 * TempControl.cpp
 *
 *  Created on: 22/01/2014
 *      Author: MPN
 */

TempControl::TempControl(int dht_num)
{

	//DHT_READ= new int[this->DHT_NUM -4];

	int dht_count;
	int dht_pin = 22;

	for (dht_count = 0 ; dht_count < DHT_NUM ; dht_count++)
	{
		DHTPIN[dht_count] = dht_pin;
		dht_pin++;
	}
}

void TempControl::Init()
{
	DHT dht1(DHTPIN[0], DHTTYPE);
	DHT dht2(DHTPIN[1], DHTTYPE);
	DHT dht3(DHTPIN[2], DHTTYPE);
	DHT dht4(DHTPIN[3], DHTTYPE);
	DHT dht5(DHTPIN[4], DHTTYPE);
	DHT dht6(DHTPIN[5], DHTTYPE);
	DHT dht7(DHTPIN[6], DHTTYPE);
	DHT dht8(DHTPIN[7], DHTTYPE);
	DHT dht9(DHTPIN[8], DHTTYPE);

	dht_sensors[0] = dht1;
	dht_sensors[1] = dht2;
	dht_sensors[2] = dht3;
	dht_sensors[3] = dht4;
	dht_sensors[4] = dht5;
	dht_sensors[5] = dht6;
	dht_sensors[6] = dht7;
	dht_sensors[7] = dht8;
	dht_sensors[8] = dht9;

	int pinCount = 0;
```

```
        for(pinCount = 0; pinCount < DHT_NUM; pinCount++)
        {
                dht_sensors[pinCount].begin();
        }
}
void TempControl::readTemperatures(float* DHT_TEMP)
{
  int i = 0;
  for( i = 0 ; i < DHT_NUM ; i++)
  {
    DHT_TEMP[i] = dht_sensors[i].readTemperature();
  }

  if(!isnan(DHT_TEMP[0]) && !isnan(DHT_TEMP[1]))
  {
                // CABIN TEMPERATURE
  DHT_READ[0] = (DHT_TEMP[0]+DHT_TEMP[1])/2;
  }

  if(!isnan(DHT_TEMP[2]) && !isnan(DHT_TEMP[3]) && !isnan(DHT_TEMP[4])
&& !isnan(DHT_TEMP[5]))
  {
                // LINER TEMPERATURE
  DHT_READ[1] =
(DHT_TEMP[2]+DHT_TEMP[3]+DHT_TEMP[4]+DHT_TEMP[5])/4;
  }

  if(!isnan(DHT_TEMP[6]))
  {
    // RES CABIN TEMPERATURE
    DHT_READ[2] = DHT_TEMP[6];
   }

  if(!isnan(DHT_TEMP[7]))
  {
    // RES LINER TEMPERATURE
    DHT_READ[3] = DHT_TEMP[7];
  }

  if(!isnan(DHT_TEMP[8]))
  {
    // ENVIRONMENT TEMPERATURE
    DHT_READ[4] = DHT_TEMP[8];
  }

}
```

AirFlowController

```
/*
 * AirFlowController.h
 *
 *  Created on: 21/01/2014
 *      Author: MPN
 */

#ifndef AIRFLOWCONTROLLER_H_
#define AIRFLOWCONTROLLER_H_

#include <AirFlowHandler.h>

class AirFlowController
{
	static const int AirFlowSensorNumber = 2;
private:
	int AirFlowPin;
	float AirVelocity;
public:
	void Init(int AF_Cabin_Pin, int AF_Liner_Pin);
	void getAllAirFlowReads(float* AIRFLOW_READ)
	AirFlowHandler AirFlowSensors[AirFlowSensorNumber];
};



#endif /* AIRFLOWCONTROLLER_H_ */

/*
 * AirFlowController.cpp
 *
 *  Created on: 21/01/2014
 *      Author: MPN
 */
#include "AirFlowController.h"

void AirFlowController::Init(int AF_Cabin_Pin, int AF_Liner_Pin)
{
	AirFlowHandler a1(AF_Cabin_Pin, 1.1, 1.6, 2.3, 0.2, 0.13,0.0);
	AirFlowHandler a2(AF_Liner_Pin, 1.3, 1.7, 2.1, 0.4, 0.08,0.01);
	AirFlowSensors[0] = a1;
	AirFlowSensors[1] = a2;
}

void AirFlowController::getAllAirFlowReads(float* AIRFLOW_READ)
{
	AIRFLOW_READ[0] = AirFlowSensors[0].getAirFlow();
	AIRFLOW_READ[1] = AirFlowSensors[1].getAirFlow();
```

```
}
```

AirFlowHandler

```
/*
 * AirFlowHandler.h
 *
 *  Created on: 21/01/2014
 *      Author: MPN
 */

#ifndef AIRFLOWHANDLER_H_
#define AIRFLOWHANDLER_H_

#include<Arduino.h>

class AirFlowHandler
{

private:
        int AirFlowPin;
        float C0;
        float C1;
        float C2;
        float C3;
        float C4;
        float C5;
        float AirVelocity;
public:
        AirFlowHandler();
        AirFlowHandler(int pin, float C0, float C1, float C2,float C3, float C4, float
C5);
        float getAirFlow();

};

#endif /* AIRFLOWHANDLER_H_ */

/*
 * AirFlowHandler.cpp
 *
 *  Created on: 21/01/2014
 *      Author: MPN
 */
#include <AirFlowHandler.h>

void AirFlowHandler::AirFlowHandler(int pin, float C0, float C1, float C2,float C3,
float C4, float C5)
{
```

```
        AirFlowPin = pin;
        pinMode(AirFlowPin,INPUT);

        this->C0 = C0;
        this->C1 = C1;
        this->C2 = C2;
        this->C3 = C3;
        this->C4 = C4;
        this->C5 = C5;
}

float AirFlowHandler::getAirFlow()
{
  float DigitalVolt = analogRead(AirFlowPin);
  float Voltage = 5*(DigitalVolt/1023);
  AirVelocity = C0 + C1*Voltage+ C2*(pow(Voltage,2)) + C3*(pow(Voltage,3))
                + C4*(pow(Voltage,4))+ C5*(pow(Voltage,5));

  return AirVelocity;
}
```

HeaterMain

```
#define REQ_BUF_SZ 500

const int PWMPIN_CABIN = 9;
const int PWMPIN_LINER = 10;
const int ledPin = 13;
unsigned long blinkRate = 100;
char Serial_Rx_buffer [REQ_BUF_SZ] = {'\0'};
int req_index = 0;
boolean hasCommStarted = true;

int CAB_RES_PWM = 0;
int LIN_RES_PWM = 0;

int CAB_RES_PWM_LIMIT = 30;
int LIN_RES_PWM_LIMIT = 30;

void setup()
{

pinMode (PWMPIN_CABIN, OUTPUT);
pinMode (PWMPIN_LINER, OUTPUT);
setPwmFrequency(9, 5);
setPwmFrequency(10, 5);
Serial3.begin(115200);
Serial.begin(9600);
}
```

```
void loop()
{
  Serial_Receive();
  setRESPWM();
  checkSerialCommunication();

  Serial.println(CAB_RES_PWM);
  Serial.println(LIN_RES_PWM);
}

void setPWM(int pin, int rate)
{
 int analogInput = (rate*255)/100;
 analogWrite(pin, analogInput);
}

void setPwmFrequency(int pin, int prescaler) {

  //prescaler = 1 ---> PWM frequency is 31000 Hz
  //prescaler = 2 ---> PWM frequency is 4000 Hz
  //prescaler = 3 ---> PWM frequency is 490 Hz (default value)
  //prescaler = 4 ---> PWM frequency is 120 Hz
  //prescaler = 5 ---> PWM frequency is 30 Hz
  //prescaler = 6 ---> PWM frequency is <20 Hz

  int Eraser = 7;
  if(pin == 9 || pin == 10) {
    TCCR2B &= ~Eraser;
    TCCR2B |=prescaler;
  }
  else if(pin == 2 || pin == 3 || pin == 5) {
    TCCR3B &= ~Eraser;
    TCCR3B |=prescaler;
  }
  else if(pin == 6 || pin == 7 || pin == 8) {
    TCCR4B &= ~Eraser;
    TCCR4B |=prescaler;
  }
}

void Serial_Receive()
{
  //EthernetClient client = server.available();  // try to get client
  //Serial.begin(115200);
  if (Serial3.available()) {  // got Serial

            // client data available to read
    //char c = Serial.read(); // read 1 byte (character) from client
```

```
    //Serial.write("Serial2 available");
    //blink();
    accelSerialRead(1);

    if(StrContains(Serial_Rx_buffer, "ENV_MATLAB"))
    {
      StrClear(Serial_Rx_buffer, req_index);
      req_index = 0;
      hasCommStarted = true;
      Serial.write("Init Command detected");
    }
    if(StrContains(Serial_Rx_buffer, "RTIV") && hasCommStarted)
    {
      processReceiveData("RTIV");
      accelSerialRead(4);
    }

    if(StrContains(Serial_Rx_buffer, "RQWA") && hasCommStarted)
    {
      processReceiveData("RQWA");
      accelSerialRead(4);
    }
  }
}

// Clear an array of char
void StrClear(char *str, char length)
{
    for (int i = 0; i < length; i++) {
        str[i] = 0;
    }
}

// Check if a char array contains a sequence of characters
char StrContains(char *str, char *sfind)
{
    char found = 0;
    char index = 0;
    char len;

    len = strlen(str);

    if (strlen(sfind) > len) {
        return 0;
    }
    while (index < len) {
        if (str[index] == sfind[found]) {
            found++;
            if (strlen(sfind) == found) {
                return 1;
```

```
            }
        }
        else {
            found = 0;
        }
        index++;
    }

    return 0;
}

void blink()
{
  digitalWrite(ledPin,HIGH);
  delay(blinkRate); // delay depends on blinkrate value
  digitalWrite(ledPin,LOW);
  delay(blinkRate);
}


void processReceiveData(char* header)
{
   StrClear(Serial_Rx_buffer, req_index);
   req_index = 0;

   boolean messageIsOver= false;
   char inByte = '\0';
   char lastinByte;

   while(!messageIsOver)
   {
      lastinByte = inByte;
      inByte = Serial3.read();

      if ((inByte == 10 && lastinByte == 13) ||req_index == 2){

         char Valor[req_index+1];

         short u = 0;

         for(u = 0; u < req_index ; u++)
         {
            Valor[u] = Serial_Rx_buffer[u];
         }

         Valor[req_index] = '\0';
         StrClear(Serial_Rx_buffer, req_index);
         req_index = 0;

         if(StrContains(header, "RTIV"))
```

```
            CAB_RES_PWM = atoi(Valor);
          else if(StrContains(header,"RQWA"))
            LIN_RES_PWM = atoi(Valor);

          messageIsOver = true;
          }

          else if ((inByte == '\r') || (inByte == '\n' && lastinByte != '\r'))
          {
            //req_index -= 1; // ignore inbyte
          }

          else
          {
            if(inByte >= 48 && inByte <=57)
            {
                  storeInByte(inByte);
            }
          }
    }
}

// accelerate the Serial read
void accelSerialRead(int count)
{
  char inByte;
  int i = 0;
  for( i = 0 ; i < count ; i++)
  {
    if (Serial3.available()>0)
    {
      inByte = Serial3.read();
      storeInByte(inByte);
    }
  }
}

void storeInByte(char inByte)
{
 if (req_index < (REQ_BUF_SZ - 1)) {
       // Serial.print(inByte);
   if(inByte != (char)-1)
   {
     Serial_Rx_buffer[req_index] = inByte;
     req_index++;
   }
 }
}

// set PWM for Heater Resistors
```

```
void setRESPWM()
{
  if(CAB_RES_PWM > CAB_RES_PWM_LIMIT)
      CAB_RES_PWM = CAB_RES_PWM_LIMIT;

  if(LIN_RES_PWM > LIN_RES_PWM_LIMIT)
      LIN_RES_PWM = LIN_RES_PWM_LIMIT;

  setPWM(PWMPIN_CABIN, CAB_RES_PWM);
  setPWM(PWMPIN_LINER, LIN_RES_PWM);
}

// check if Communication is online,
// if not cut all set points
void checkSerialCommunication()
{
  if(Serial3.available() <= 0)
     timeout++;
  else
     timeout = 0;

  if(timeout >= 100)
  {
      CAB_RES_PWM = 0;
      LIN_RES_PWM = 0;
      timeout = 0;
  }
}
```