PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE PROCESSOS
MESTRADO PROFISSIONAL

DESENVOLVIMENTO DE UMA FERRAMENTA COMPUTACIONAL DE SINTONIA DE CONTROLADOR DO TIPO PROPORCIONAL, INTEGRAL E DERIVATIVO (PID) BASEADO NO MÉTODO RELÉ. ESTUDO DE CASO: CONTROLE DE TEMPERATURA DA ÁGUA EM UM RESERVATÓRIO.

MÁRIO JORGE DA SILVA MACIEL

Dissertação (Exame de Qualificação) de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia de Processos – Mestrado Profissional, PPGEP/ITEC, da Universidade Federal do Pará, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Processos.

Orientador: Prof. Dr. Edilson Marques Magalhães.
Co-Orientador: Prof. Dr. Tirso Lorenzo Reyes Carvajal.

Belém
FEVEREIRO de 2017

DESENVOLVIMENTO DE UMA FERRAMENTA COMPUTACIONAL DE SINTONIA DE CONTROLADOR DO TIPO PROPORCIONAL, INTEGRAL E DERIVATIVO (PID) BASEADO NO MÉTODO RELÉ. ESTUDO DE CASO: CONTROLE DE TEMPERATURA DA ÁGUA EM UM RESERVATÓRIO.

MÁRIO JORGE DA SILVA MACIEL

DISSERTAÇÃO (EXAME DE QUALIFICAÇÃO) SUBMETIDA (O) AO CORPO DOCENTE DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA PROCESSOS – MESTRADO PROFISSIONAL (PPGEP/ITEC) DA UNIVERSIDADE FEDERAL DO PARÁ COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA DE PROCESSOS.

Examinada (o) por:

_____

Prof. Edilson Marques Magalhães, Dr.
(PPGEP/ITEC/UFPA-Orientador)

_____

Prof. Tirso Lorenzo Reyes Carvajal, Dr.
(PPGEP/ITEC/UFPA-Co-orientador)

_____

Prof. , Dr.
(PPGEP/ITEC/UFPA)

_____

Prof. César Alberto Chagoyén Méndez, Dr.
(UEA)

BELÉM, PA - BRASIL

FEVEREIRO DE 2017

Dados Internacionais de Catalogação-na-Publicação (CIP)
Sistema de Bibliotecas da UFPA

Maciel, Mário Jorge da Silva, 1970-

Desenvolvimento de uma ferramenta computacional de sintonia de controlador do tipo proporcional, integral e derivativo (PID) baseado no método relé. Estudo de caso: controle de temperatura da água em um reservatório. / Mário Jorge da Silva Maciel. - 2017.

Orientador: Edilson Marques Carvajal.
Coorientador: Tirso Lorenzo Reyes Carvajal.
Dissertação (Mestrado) - Universidade Federal do Pará, Instituto de Tecnologia, Programa de Pós-Graduação em Engenharia de Processos, Belém, 2016.

1. Controlador- PID. 2. Método Relé sintonia. 3. Relé Ideal e Histerese. I. Título.

CDD 22. ed. 621.56

Este trabalho é dedicado aos amigos, colegas, familiares que sempre apoiaram, incentivaram e de alguma forma contribuíram para a elaboração do mesmo. Em particular à minha namorada, pai, mãe e irmãos pelo incentivo e parceria.

AGRADECIMENTOS

Primeiramente a Deus que permitiu que tudo isso acontecesse, ao longo de minha vida, e não somente nestes anos como universitário, mas que em todos os momentos é o maior mestre que alguém pode conhecer.

Ao Professor Dr. Edilson Marques Magalhães pela orientação, incentivo e ajuda na elaboração e composição do trabalho.

Ao Professor Dr. Tirso Lorenzo Reyes Carvajal pela ajuda, incentivo e colaboração para o trabalho.

Aos colegas do PPGEP pelo incentivo e cooperação.

Obrigado aos meus familiares, que nos momentos de minha ausência dedicados ao estudo superior, sempre fizeram entender que o futuro é feito a partir da constante dedicação no presente!

Agradeço ao meu pai Manoel Braga Maciel e minha mãe Creuza da Silva Maciel, por ter nos criado com dignidade, caráter e com principio ético, mostrando sempre que o caminho é a educação para que busque o mundo melhor. Heroína que me deu apoio, incentivo nas horas difíceis, de desânimo, cansaço e permitido chegar até aqui.

Em especial, à minha amiga Maria Joicy César Oliveira, pela ajuda com componentes eletrônicos necessários e também com informações técnicas e à colega e professora Msc Fátima Geisa da Ulbra/Manaus, pela ajuda como coordenadora do curso de Engenharia Elétrica que muito me incentivou e viabilizou o término deste curso de mestrado.

A todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

Resumo da Dissertação apresentada ao PPGEP/UFPA como parte dos requisitos necessários para a obtenção do grau de Mestre em Engenharia de Processos (M.Eng.)

DESENVOLVIMENTO DE UMA FERRAMENTA COMPUTACIONAL DE SINTONIA DE CONTROLADOR DO TIPO PROPORCIONAL, INTEGRAL E DERIVATIVO (PID) BASEADO NO MÉTODO RELÉ. ESTUDO DE CASO: CONTROLE DE TEMPERATURA DA ÁGUA EM UM RESERVATÓRIO.

RESUMO

Os processos de produção industrial quase sempre são automatizados ou envolvem algum tipo de automatização. Muitos desses processos automatizados envolvem o projeto de controladores, esses são responsáveis pela articulação e sincronia entre os atuadores e sensores de forma a automatizar um determinado sistema. Entretanto, o projeto de um controlador elaborado usando o método clássico, baseado modelo matemático do sistema, envolve muitos passos e especialistas acerca da planta. Normalmente, o projeto de controladores na indústria envolve equipamentos que não temos acesso ao modelo matemático, sendo assim, seria extremamente complicado obtê-lo usando o método clássico. Existe uma alternativa aos métodos clássicos, que são os métodos empíricos de construção de projetos de controladores que se baseiam na resposta do sistema, sendo assim, será possível injetar alguns sinais na planta e analisar a resposta da mesma e então sintonizar (ou calibrar) o controlador sem a necessidade de realizarmos a modelagem matemática. Nesta dissertação foi desenvolvida uma ferramenta computacional para automatizar o método de sintonia do relé ideal e com histerese, usando como estudo de caso o controle de temperatura da água num reservatório, estes apresentaram como resultados as saídas com boa aproximação dos sinais senoidais periódicos como resposta a um sinal de relé durante o processo de sintonia, conforme a metodologia do relé. E ainda foi possível executar o controlador na planta com o sistema, resultando, como esperado, no controle de temperatura da água com um erro de menos de um grau para mais ou para menos. Como esperado também, o método do relé com histerese resultou em uma sintonia mais precisa, no entanto, com tempo de sintonia mais longo que o do método do relé ideal.

PALAVRAS-CHAVE: PID, Método-Relé, Sintonia.

Abstract of Dissertation presented to PPGEP/UFPA as a partial fulfillment of the requirements for the degree of Master in Process Engineering (M.Eng.)

DEVELOPMENT OF A COMPUTATIONAL TOOL FOR THE CONTROLLER OF THE PROPORTIONAL, INTEGRAL AND DERIVATIVE TYPE (PID) BASED ON THE RELAY METHOD. CASE STUDY: WATER TEMPERATURE CONTROL IN A RESERVOIR.

ABSTRACT

Industrial production processes are almost always automated or involve some kind of automation. Many of these automated processes involve controllers, these are responsible for the articulation and synchronization between actuators and sensors to automate a particular system. However, the design of a controller elaborated using the classical method, based mathematical model of the system, involves many steps and experts about the plant. Typically, the controllers in the industry involves equipment that we do not have access to mathematical, so it would be extremely complicated to get it using the classical method. There is an alternative to classical methods, which are the empirical methods of building controller designs that are based on the system response, so it will be possible to injecting some signals into the plant and analyzing the plant's response and then tune (or calibrate) the controller without the need to perform mathematical modeling. This dissertation was developed a computational tool to automate the method of relay tuning with hysteresis, using as the case study the temperature reservoirs, these showed as results the outputs with good approximation of the signals periodic sine waves in response to a relay signal during the tuning process, according to the relay methodology. And it was still possible to run the controller on the plant with the system, resulting, as expected, in water temperature control with an error of less than a degree for more or less. As expected too, the relay method with hysteresis has resulted in a more accurate tuning, however, with longer tuning time long than that of the ideal relay method.

KEYWORDS - PID, Relay Method, Tune.

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

NOMENCLATURA

K - Ganho estático do processo.

K_d - Ganho Derivativo.

K_i - Ganho Integral.

K_p - Ganho Proporcional.

K_u - Ganho crítico.

P_u – Período de oscilação.

ΔMV - Variação da variável manipulada.

ΔT - Variação do tempo.

ΔU - Variação do sinal do controle.

ΔY - Variação da saída do processo.

ε - Valor da histerese.

ω_n - Frequência natural.

ω_u - Frequência Crítica.

τ - Constante de tempo.

θ - Atraso de Transporte ou tempo morto.

a - Largura de saída do processo.

e - erro.

e(k) - Erro no instante k – discreto.

e(t) ou erro(t) - Erro no instante t – contínuo.

h - Amplitude de saída do relé.

r(t) - referência de processo (SP) do sistema dinâmico.

u - sinal de controle.

u(t) - Sinal de controle (MV)- Contínuo.

y(t) - Variável de processo (PV) do sistema dinâmico.

FOPDT - First Order Plus Dead Time.

IAE - Integral Absolute Error.

IE - Integral of Error.

IMC - Internal Model Control.

SP - Setpoint (Variável de referência).

Erro - Diferença entre o setpoint e a variável de processo.

N(a) - Função descritiva do relé.

MV - Maninulated Variable.

OLE - Multiple Input and Multiple Output.

OPC - OLE for Process Control.

PI - Controlador Proporcional-Integral.

PID - Controlador Proporcional-Integral-Derivativo.

PV - Process Variable.

SISO - Single Input and Single Output.

SOPDT - Second Order Plus Dead Time.

ZN - Ziegler-Nichols.

IDE – Interface Developer Enviroment (Interface de Ambiente de Desenvolvimento).

CPU – Unidade Central de Processamento.

SUMÁRIO

CAPÍTULO 1

INTRODUÇÃO

Do ponto de vista técnico, automatizar sistemas significa substituir tarefas humanas repetitivas sujeitas a erros de execução por máquinas dotadas de sistemas de controle que baseado no modelo da planta do sistema pode realizar ajustes de forma precisa e se adaptar aos ruídos e transitórios que porventura surgirem na resposta do sistema(ASTROM e HAGGLUND, 2006). Essa busca de automatizar vem ao longo do tempo se consolidando, cujas primeiras teorias de modelagem e controle se deram com o nome de "teoria clássica de controle"(OGATA, 1985).

A teoria clássica, contribuiu e vem contribuindo com processo de automatização, mas, em vista das rápidas mudanças de especificações de processos automatizados, essa teoria, possui alguns empecilhos, tais como: falta de agilidade na definição da planta do sistema e também no projeto do controlador(ASTRÖM e HÄGGLUND, 2001). O processo de definição é sinônimo de modelagem do sistema, isto é, o projetista antes de iniciar o processo de automatização, que significa construir o controlador, precisa fazer um estudo dos requisitos e parâmetros de entrada e saída do sistema a ser projetado, então de posse desses resultados experimentais e ajustes realizados, aí sim passa para a fase de projeto do controlador que exigirá do projetista mais alguma manipulação matemática e teorias clássicas para, enfim, novamente simular o sistema e colocar definitivamente em funcionamento(ASTROM e HAGGLUND, 2005). Perceba que todos esses procedimentos são difíceis de serem praticados em um ambiente puramente industrial, já que há bastante informações de resultados acadêmicos e que exigem um certo tempo de simulação em laboratório(COLOGNI, 2008).

É citado que 96% dos controladores utilizados são do tipo PID (RUBAAI, SITIRICHE e OFOLI, 2008). Mais de 90% das malhas de controle encontradas em processos industriais operavam com controladores PI/PID atingindo uma larga faixa de aplicações: controle de processos, drivers para motores, indústria automobilística, controladores de voo, pilotos automáticos, instrumentação, entre outros(ASTRÖM e HÄGGLUND, 2001).

A utilidade dos controladores PID está na sua aplicabilidade geral à maioria dos sistemas de controle(OGATA, 2010). Em particular, quando o modelo matemático da planta não é conhecido e os métodos de projeto analítico não podem ser utilizados, controles PID se mostram os mais úteis. Mesmo com o surgimento de novas técnicas de controle (utilizando lógica fuzzy, sistemas adaptativos, preditivos) percebe-se que, ainda hoje, é predominante o uso deste tipo de controlador nas malhas industriais. Em aplicações industriais, o controle PID é uma estratégia muito popular devido a sua arquitetura simples e sua sintonia ser realizada por métodos igualmente simples e consolidados. Mesmo com o seu grande uso, e possuindo uma grande história e bibliografia, a sintonia de controladores PID ainda é uma área ativa de pesquisa, tanto acadêmica quanto industrial (CONG e LIANG, 2009).

O método empírico de sintonia do relé de controladores PID, também chamado de autos sintonia (autotunning), introduzido por Aström e Hagglund é objeto de estudo desta dissertação, uma vez que se assemelha às ações de calibração de um controlador de um ser humano de forma automática(ASTROM e HAGGLUND, 1995).

A ideia básica da autossintonia é que uma vez formalizada as etapas de sintonia do controlador, o processo ocorre de forma automatizada, isso torna desnecessário a participação de um especialista em controle(ASTROM e HAGGLUND, 2006).

Diante do exposto, neste trabalho será proposto a construção de um software com aplicação em tempo real que gera sintonias de controladores PID a partir do ensaio do método do relé ideal e com histerese. Além disto, a validação do software se dará através de um ensaio que controla a temperatura de um pequeno reservatório com água. Este ensaio permitirá que o projetista do controlador informe alguns parâmetros e mostre o comportamento da saída da planta e do relé.

1.1 - JUSTIFICATIVA DA PROPOSTA DE DISSERTAÇÃO

Normalmente, o projeto de controladores na indústria envolve equipamentos que não temos acesso ao modelo matemático dinâmico, sendo assim, seria extremamente complicado obtê-lo usando o método clássico (OGATA, 2010).

A modelagem de sistemas através dos métodos clássicos, realizando o levantamento do modelo dinâmico, resultam em equações diferenciais de qualquer ordem, sendo, portanto, mais genéricos(VANDORE, 2006). Na prática, a grande maioria dos processos industriais

pode ser modelada por funções de transferência de primeira ou segunda ordem com atraso de transporte(ASTROM e HAGGLUND, 2006).

O método relé de identificação vem sendo empregado para a estimativa de modelos matemáticos de baixa ordem com ou sem atraso de transporte para aplicações em processos industriais(ASTROM e HAGGLUND, 2005). Para plantas de primeira ordem do tipo:

$$G_p(s) = \frac{K_p e^{-\theta_1 s}}{\tau_1 s + 1}$$

O método de autossintonia a ser implementado através de software deverá ser visto como uma ferramenta de sintonia (calibração) à disposição do utilizador que além de auxiliar o processo de sintonia poderá otimizá-lo em menos tempo.

Considera-se que sem limite de tempo, o especialista em controle, devido à sua capacidade de análise, obterá os melhores resultados, mas, a um custo que poderá ser demasiadamente elevado para o projeto.

Como exposto acima, o desenvolvimento da ferramenta computacional a ser desenvolvida nesta dissertação servirá como referência para outros projetos, além do que, tem como intuito evitar a contratação de mão de obra muito especializada e cara, quando se tratar de projetos de pequeno porte, comuns em ambientes industriais.

1.2 - OBJETIVOS

1.2.1 - Objetivo geral

Criar uma ferramenta computacional de referência que automatize a sintonia de controladores PID usando o método relé, com opções de uso do algoritmo do relé ideal ou por histerese. A ferramenta computacional deverá também permitir executar o controlador a partir dos parâmetros proporcional, integral e derivativo gerados pela mesma.

1.2.2 - Objetivos específicos

- O software deverá ter uma camada dedicada ao comando dos componentes do sistema, como os sensores e atuadores.

- Também existirá uma outra camada de software dedicada à interação com o usuário e execução do algoritmo de auto sintonia.
- O sistema deverá apresentar uma interface de usuário que permita ao mesmo informar parâmetros necessários para a execução da sintonia do controlador e que possa mostrar o andamento e conclusão da sintonia.
- O sistema deverá permitir gravar e recuperar dados de parametrização dos controladores PID, bem como executar o mesmo.
- Permitir realizar o controle de temperatura de um recipiente com água, que servirá como estudo de caso do sistema computacional tanto para o momento de sintonia do controlador como para execução do controlador sintonizado.

CAPÍTULO 2

REVISÃO DA LITERATURA

2.1 - CONTROLADORES PID

Desde os primórdios do controle baseado em retroação, que as ações, proporcional, integral e derivativa estavam presentes, implícita ou explicitamente. No entanto, terá sido apenas no trabalho desenvolvido por Minorsky sobre controle de direção de navios publicado em 1922 (MINORSKY, 1922), que o rigor teórico e matemático foi atribuído ao controlador PID (MURRAY, ASTRÖM, *et al.*, 2003).

O controlador PID é constituído por três ações, respectivamente, proporcional (P), integral (I) e derivativa (D).

Apesar da abundância de ferramentas sofisticadas, incluindo controladores avançados, onde 96% dos controladores utilizados é do tipo PID e ainda em utilizado em mais de 90% na malha de controle industrial (RUBAAI, SITIRICHE e OFOLI, 2008), em diversas aplicações como: controle de processos, driver de motores, indústria automobilística, controladores de vôo, pilotos automáticos e outros (ASTROM e HAGGLUND, 1995).

Apesar da sua grande aplicação, não há um consenso na definição do algoritmo de controle PID. De acordo com Astrom e Hagglund (1995), a versão acadêmica do controlador PID tem a seguinte forma:

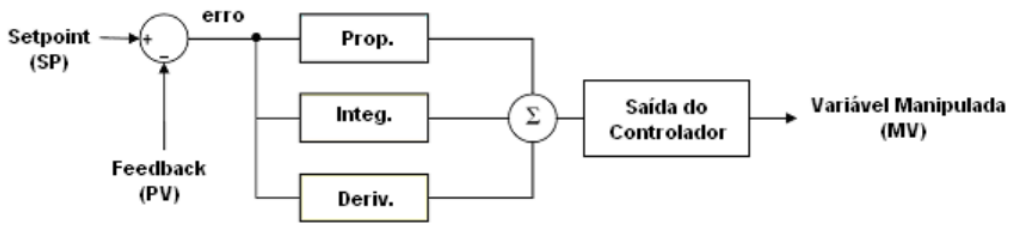
$$u(t) = k_p\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)\,d\tau + T_d\frac{\partial e}{\partial t}\right) \qquad \textbf{(II.1)}$$

onde $u(t)$ representa a variável de controle e o erro é definido por e = SP - PV onde SP é o valor de referência (setpoint), e PV o valor de saída do processo.

Como se pode verificar pela fórmula, a saída do controlador PID é composta pelas três ações e baseada no erro de medida do processo a ser controlado. Assim, se o loop de regulação funcionar corretamente, qualquer variação no erro, causada por mudança de setpoint ou perturbação no processo, será eliminada rapidamente por uma combinação dos

três fatores P, I e D. O diagrama de blocos do controlador PID é mostrada na figura II-1 (ASTRÖM e HÄGGLUND, 1995).

Figura 2-1: Diagrama de blocos do controlador PID.



Fonte: (ASTRÖM e HÄGGLUND, 1995)

2.1.1 - Componente Proporcional

De acordo com (ASTROM e HAGGLUND, 1995), temos que:

O fator proporcional resulta do produto entre o ganho e o erro de medida. Assim, quanto maior o ganho ou o erro, maior será a saída do fator proporcional. Colocando um ganho proporcional, demasiadamente elevado levará o controlador a exceder o setpoint e pode colocar o sistema em oscilação.

A componente proporcional mostra-se insuficiente quando o erro se torna muito pequeno e a saída do controlador se torna muito pequena.

Assim, quanto maior o ganho proporcional menor será o erro estacionário que não deixa de existir e mais próximo se está de um loop instável.

2.1.2 - Componente Integral

O fator integral pode ser explicado como um acumulador de erro que tanto pode aumentar a componente integral quando o erro for positivo, como diminuir quando este for negativo. Quando o controlador está desempenhando bem a sua tarefa, o valor de integral deve ser praticamente nulo (ASTROM e HAGGLUND, 1995).

Mesmo quando for muito pequeno para a componente proporcional, o integral está acumulando até que seja suficiente para atuar. Assim, uma das tarefas da componente integral é eliminar o erro em regime estacionário (OGATA, 1985). O ponto fraco da componente integral é que pode contribuir para a sobre-elevação se o seu valor for muito elevado ao se

aproximar do setpoint. Quanto menor o tempo de integral, mais agressivo será o efeito desta componente na supressão de erro (OGATA, 2010).

2.1.3 - Componente Derivativa

A componente derivativa compara o erro atual com o erro da última verificação. Esta depende da taxa de variação do erro. Quanto maior o ganho de derivativa ou maior a variação de erro, maior será a componente derivativa. O efeito da derivativa é contrapor a possível sobre-elevação provocada pelas componentes proporcional e integral (ASTROM e HAGGLUND, 1995).

Uma derivativa bem calibrada possibilita a utilização de componentes proporcional e integral mais agressivas. Quanto maior for o tempo de derivativa, maior será a sua agressividade (ASTRÖM e WITTENMARK, 1988).

2.2 - SINTONIA DE CONTROLADORES

Atualmente existem métodos sofisticados de sintonizar um controlador que cumpra as especificações, em regime estacionário e transitório enquanto segue referências e rejeita perturbações (ASTROM e HAGGLUND, 2006). Estes métodos requerem do projetista um conhecimento do modelo dinâmico do processo, na forma de equações ou um conhecimento detalhado da resposta em frequência numa gama alargada. Qualquer uma destas informações pode ser muito difícil de obter em meio industrial, o que levou ao desenvolvimento de técnicas sofisticadas de identificação de sistemas (CHENG, 2006).

Os dois primeiros métodos de sintonia apresentados, são métodos bastante conhecidos e utilizados em calibração de controladores PID há mais de 60 anos. Por estas razões e por ter considerado o Método Relé (Relay-Method) como o de maior potencial para as aplicações que se pretende, estes serão apenas brevemente apresentados (CARDOSO, 2002).

2.2.1 - Método de malha fechada de Ziegler-Nichols

O primeiro método largamente utilizado para sintonia de controladores PID foi publicado por Ziegler e Nichols em 1942 (ZIEGLER e NICHOLS, 1942). Neste método o critério de ajuste de parâmetros baseia-se na avaliação da amplitude e frequência das oscilações do sistema no seu limite de estabilidade (CARDOSO, 2002).

Este procedimento é constituído pelos seguintes passos:

- Ativar o controlador em modo proporcional com o ganho muito baixo;
- Aumentar o ganho até que o loop comece a oscilar. A oscilação deverá ser linear e a sua detecção deve ser realizada à saída do controlador;
- Registrar o valor do ganho crítico K_u e o período de oscilação P_u;
- Ajustar os parâmetros do controlador de acordo com a tabela II-1, mostrada abaixo:

Tabela 2-1: Tabela de Ziegler-Nichols para o método de malha fechada.

Controlador	$\mathbf{K_p}$	$\mathbf{T_i}$	$\mathbf{T_d}$
P	$0{,}5K_u$	-	-
PI	$0{,}4K_u$	$0{,}8P_u$	-
PID	$0{,}6\ K_u$	$0{,}5P_u$	$0{,}125P_u$

Fonte: (ZIEGLER e NICHOLS, 1942)

Esta técnica, não é muito utilizada atualmente devido ao seu comportamento oscilatório.

No entanto, é estudada com interesse, pois a experiência mostra que a sua utilização no ajuste de controladores proporciona a muitos sistemas, respostas em malha fechada aceitáveis e serve de base às modernas técnicas de autossintonia (auto-tuning) (ASTRÖM e HÄGGLUND, 1995).

2.2.2 - Método de malha aberta de Ziegler-Nichols

Ziegler e Nichols reconheceram também em 1942 que a resposta de um grande número de sistemas de controle a um passo, manifesta-se sob uma curva de reação. Quando o processo a controlar não envolve integradores nem pólos dominantes complexos-conjugados, a curva de resposta ao degrau unitário pode-se assemelhar a uma curva em forma de S. Se a resposta não apresentar esta curva, o método não se aplica (ZIEGLER e NICHOLS, 1942).

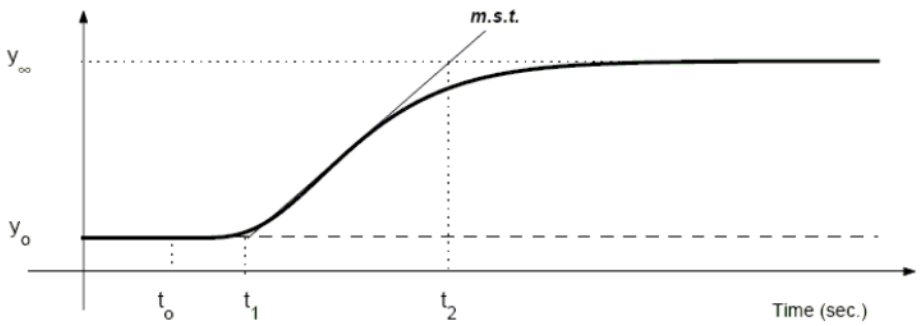
Uma versão linearizada de um sistema simples, pode ser obtida pela experiência em malha aberta, utilizando o seguinte procedimento (ASTROM e HAGGLUND, 2005):

Com o sistema em malha aberta, conforme mostrado na figura II-2, deverá ser levado manualmente até ao seu ponto de operação normal. Se o sistema estabilizar em y(t) = y_0 para uma atuação constante de u(t) = u_0 :

- Em, t_0, aplicar um degrau de variação na atuação de u_0 para u_∞ (variação entre 5 e 20% da gama disponível);
- Registrar a variável de processo até que esta estabilize;
- Traça-se a tangente de declive máximo e calculam-se os parâmetros:

Figura 2-2: Curva de Reação usando ZN

$$K_0 = \frac{y_\infty - y_0}{u_\infty - u_0} \qquad \tau_0 = t_1 - t_0 \qquad \nu_0 = t_2 - t_1$$



Fonte: (ZIEGLER e NICHOLS, 1942)

- Recorre-se então à tabela II-2 de Ziegler-Nichols (ZN) para a curva de reação e calculam-se os parâmetros do controlador PID:

Tabela 2-2: Tuning de Ziegler-Nichols utilizando a curva de reação

Controlador	**K_p**	**T_i**	**T_d**
P	$\frac{V_0}{K_0\tau_0}$		
PI	$\frac{0{,}9V_0}{K_0\tau_0}$	$3\tau_0$	
PID	$\frac{1{,}2V_0}{K_0\tau_0}$	$2\tau_0$	$0{,}5\tau_0$

Fonte: (ZIEGLER e NICHOLS, 1942)

2.3 - MÉTODO DO RELÉ

Aprofundaremos o estudo do método do Relé, introduzido por Astrom e Hagglund, em 1994. Na sequência de uma busca por técnicas de autossintonia adequadas à aplicação na maioria dos sistemas de automação industrial.

As ações de um calibrador automático devem assemelhar-se às de um humano ao sintonizar um controlador. Esta pode-se representar nos seguintes passos (HANG, ASTRÖM e WANG, 2002):

- Observar o comportamento do processo, com maior ou menor intervenção na sua excitação;
- Retirar da experiência uma descrição do comportamento do processo;
- Converter os dados, em como o sistema em malha fechada deveria funcionar, respeitando as limitações do processo observado;
- Determinar os parâmetros do controlador, de acordo com o comportamento pretendido.

Ao formalizar estes passos, estabelece-se um procedimento, que é reconhecido por sintonia de controladores. A autossintonia (autotuning) pode ser visto como um passo a frente, na automação em que o equipamento automatiza as ações do especialista de controle (HANG, ASTRÖM e WANG, 2002).

Apesar da grande maioria das aplicações de autossintonia ser utilizada em controladores simples, este também é muito útil em controladores mais complexos. De fato, este é um pré-requisito para a utilização da maioria dos algoritmos de controle avançado. Um mecanismo de autossintonia é muitas vezes necessário para encontrar a escala de tempo ou os valores iniciais para um controlador mais complexo (JOHNSON e MORADI, 2005).

2.3.1 - Sintonia pelo Método Relé (Relay Method)

A identificação do sistema é parte fundamental da sintonia automática de controladores PID. Os métodos de identificação podem ser classificados como abordagens no domínio da frequência ou no domínio do tempo (ASTRÖM e HÄGGLUND, 1995).

A experiência de projeto de Ziegler-Nichols é uma contribuição de sucesso, principalmente na fase de identificação, na forma como determina as características fundamentais do processo, ganho limite K_u e frequência limite ω_u. Trata-se de uma abordagem simples e de confiança para a identificação das suas características fundamentais (ASTRÖM e HÄGGLUND, 2004).

O método Relay é apresentado como alternativa na identificação do modelo de sistemas. Como vantagem apresenta-se capaz de gerar e manter uma oscilação controlada (a magnitude da oscilação pode ser definida). O sucesso deste método deve-se à simplicidade dos mecanismos de identificação e calibração, e também à sua aplicabilidade em processos lentos ou altamente não lineares (ASTROM e HAGGLUND, 2006).

Como se pode verificar a maioria ainda assenta em métodos de identificação baseados na resposta a um escalão. Astrom e Hagglund (ASTOM e HAGGLUND, 1995), sugeriram a utilização do método Relé para gerar uma oscilação sustentada como uma alternativa ao método por tentativa convencional. Este é bastante eficiente na determinação do ganho crítico e frequência crítica.

Posteriormente, Luyben popularizou o método Relé ao utilizá-lo em colunas de destilação industrial, e chama a este método ATV (Auto-Tune Variation). A coluna de destilação é uma unidade importante de processos químicos industriais. Obter um modelo em função de transferência linear, para colunas altamente não lineares utilizando testes de degrau ou impulso, é uma tarefa muito complicada. De acordo com os estudos de Luyben, sistemas deste tipo apresentam constantes de tempo extremamente longas, como por exemplo $\tau \approx 870h$ (LUYBEN, 1987). Estudos indicam que para se obter um modelo linear preciso, neste tipo de processos apenas se deve infringir variações de magnitude inferiores a 0,01%. Esta limitação incentivou a utilização do método relé, que Luyben transformou numa prática padrão em processos químicos de controle, ao mostrar que este permite obter de uma forma eficiente, modelos lineares para sistemas deste tipo (LUYBEN, 2001).

Vantagens do método Relé (ASTROM e HAGGLUND, 1995) e (ASTRÖM e WITTENMARK, 1988):

- Requer pouco processamento matemático;
- Identifica as características do modelo em torno da sua importante, frequência crítica (frequência onde a margem de fase é $-\pi$);
- Adequa-se a diferentes processos industriais;

– Aplicação não requer conhecimento do modelo matemático do sistema;

– A sintonia em produção, o processo não foge ao sem ponto nominal, pois a perturbação a impor é limitada pelos seus parâmetros;

– Baixa sensibilidade a perturbações, por ser implementado em malha fechada;

– Para processos com uma constante de tempo muito elevada, é mais eficiente em termos de tempo do que os métodos convencionais de degrau ou impulso;

– Evita o fastidioso, procedimento de tentativa e erro na determinação do ganho crítico. Como o que ocorre em Ziegler-Nichols.

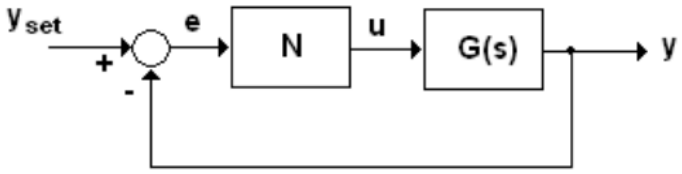
Desvantagens do método Relé (ASTROM e HAGGLUND, 1995):

– Não pode ser aplicado em sistemas muito ruidosos.

2.3.2 - Análise do Relé Ideal (Liga-Desliga)

A sintonia baseada no método relé pode ser analisado partindo do diagrama de blocos da figura II-3, onde G(s) é uma função de transferência linear e N é um elemento não linear (ASTROM e HAGGLUND, 1995).

Figura 2-3: Relé Ideal com realimentação



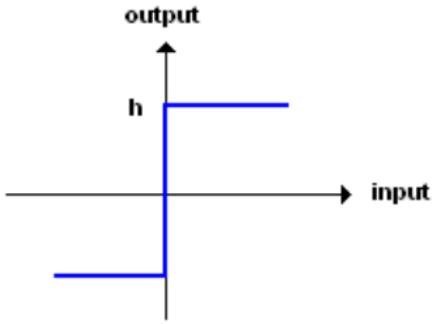
Fonte: (OGATA, 1985)

Se o sinal de entrada no elemento não linear for uma onda senoidal

$$e(t) = a \sin \omega t \qquad \textbf{(II.2)}$$

onde a , representa a magnitude da onda senoidal, então o sinal de saída u(t) do elemento não linear é uma onda retangular como mostrado na figura II-4.

Figura 2-4 - Amplitude h saída do Relé Ideal



Fonte: (ASTROM e HAGGLUND, 2006)

Como a maioria das análises de sistemas de controle se baseiam em teoria linear, a transformada de Fourier é útil neste ponto (OGATA, 1985). A saída do sistema não linear pode ser representada como, assim temos (ASTROM e HAGGLUND, 1995):

$$u(t) = A_0 + \sum_{n=1}^{\infty} A_n \cos(n\omega t) + B_n sen(n\omega t) \qquad \textbf{(II.3)}$$

Onde

$$A_0 = \frac{1}{2\pi}\int_0^{2\pi} u(t)\,d(wt) \qquad \textbf{(II.4)}$$

$$A_n = \frac{1}{\pi}\int_0^{2\pi} u(t)\cos(n\omega t)\,d(\omega t) \qquad \textbf{(II.5)}$$

$$B_n = \frac{1}{\pi}\int_0^{2\pi} u(t)\,sen(n\omega t)\,d(\omega t) \qquad \textbf{(II.6)}$$

Pelo fato de u(t) ser uma função ímpar, os coeficientes $A_0 = 0$ e $A_n = 0$, $\forall n$. A equação, (2.2) torna-se,

$$u(t) = \sum_{n=1}^{n} B_n sen(n\omega t) \qquad \textbf{(II.7)}$$

Se um relé ideal for aplicado, então os coeficientes de B_n tornam-se,

$$B_n = \begin{cases} \frac{1}{n}\frac{4h}{\pi} & n = 1,3,5\ldots \\ 0 & n = 2,4,6\ldots \end{cases} \quad \textbf{(II.8)}$$

A função descritiva é utilizada como ferramenta para análise deste sistema não linear, no domínio da frequência. Apenas a harmônica principal é utilizada na equivalência linear. Isto é, apenas o primeiro coeficiente de Fourier é utilizado para a análise no domínio da frequência. A função descritiva fica,

$$N(a) = \frac{B_1 + jA_1}{a} \quad \textbf{(II.9)}$$

Para o relé ideal, como $A_1 = 0$,

$$N(a) = \frac{4h}{\pi a} \quad \textbf{(II.10)}$$

Assim que uma oscilação sustentada é gerada pelo relé na malha de retroação, a frequência de oscilação corresponde ao limite de estabilidade, isto é

$$1 + G(j\omega_u)N(a) = 0 \quad \textbf{(II.11)}$$

E o ganho crítico K_u,

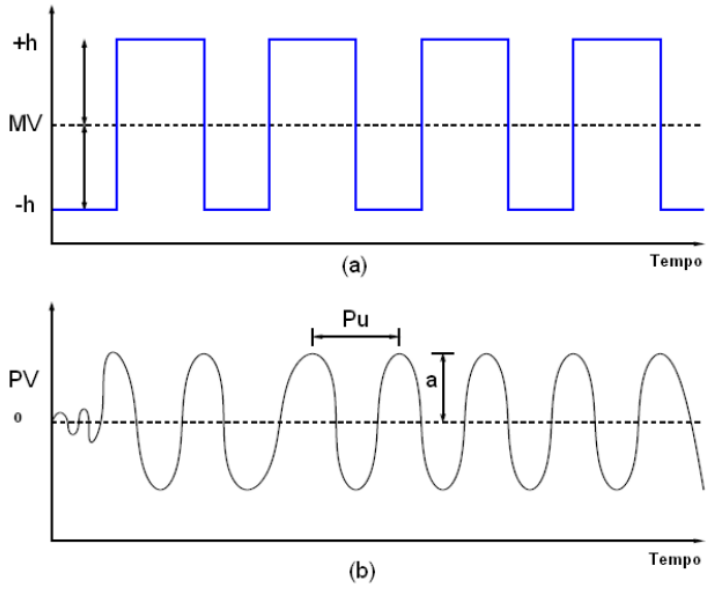
$$K_u = -\frac{1}{G(j\omega_u)} = \frac{4h}{\pi a} \quad \textbf{(II.12)}$$

O sistema de relé realimentado, de Astrom-Hagglund é baseado na observação: quando a saída está em atraso -π em relação à entrada, o sistema em malha fechada oscila com período P_u (ASTROM e HAGGLUND, 1995).

$$\omega_u = \frac{2\pi}{P_u} \quad \textbf{(II.13)}$$

A ação de um relé (figura II-5a), bem como a resposta de um sistema na figura II-5b:

Figura 2-5: (a) Entrada do sistema. (b) Saída do sistema.



Fonte: (ASTROM e HAGGLUND, 1995)

Ao iniciar o relé, este passa para o patamar u = ***h*** e passado um período D (atraso) a saída y começa a aumentar. Quando passa o setpoint o relé comuta para u = ***-h*** e assim sucessivamente (ASTROM e HAGGLUND, 1995).

O seu procedimento, determina o ganho que introduz um atraso de meio ciclo quando operando em realimentação. Trata-se do ganho crítico, K_u que está relacionado com o ponto onde a curva de Nyquist primeiro cruza o eixo real. Parte do sucesso da identificação deste método vem do fato de K_u e ω_u serem retirados diretamente dos resultados experimentais, pois muito raramente se conhece o modelo do sistema (ASTROM e HAGGLUND, 1995).

O procedimento é relativamente simples e eficiente. Fisicamente implica movimentar a variável manipulada contra o processo. Considerando um sistema de ganho estático positivo. Quando se aumenta a entrada (atuação), a saída y do sistema tende a aumentar. Quando se verifica a passagem de y por SP, muda-se a entrada para o sentido oposto. Como consequência temos uma oscilação do sistema, mas a sua amplitude pode ser controlada ao ajustar h (ASTROM e HAGGLUND, 2005).

Através do método relé determinam-se valores característicos que permitem estimar a função de transferência do sistema. Por exemplo, para um sistema de primeira ordem com atraso mostrado na equação II.14 (FOPDT) (CHIEN, HRONES e RESWICK, 1952),

$$G(s) = \frac{K_p e^{-Ds}}{\tau s + 1} \qquad \textbf{(II.14)}$$

Onde, K_p corresponde ao ganho estático, D ao atraso e τ à constante de tempo do sistema. O atraso D determina-se facilmente na parte inicial do teste com o relé, sendo simplesmente o tempo que o sistema demora a responder a uma mudança de setpoint (ASTRÖM e WITTENMARK, 1988). Em teoria, o ganho estático (equação II.15) pode ser obtido comparando a entrada e saída em dois pontos de operação,

$$K_p = \frac{\Delta PV}{\Delta MV} \qquad \textbf{(II.15)}$$

Em sistemas, altamente não lineares, esta variação em MV deve ser tão pequena quanto 10^{-3} %, da gama completa, de forma a que os resultados representem o ganho linearizado. Obter uma estimativa confiável do ganho estático partindo destes dados, é geralmente impraticável (ASTROM e HAGGLUND, 1995).

Sabendo a forma da função de transferência que melhor representa o sistema, pode-se determinar os parâmetros do mesmo com o auxilio das equações de ganho crítico e período crítico (ASTROM e HAGGLUND, 2006).

Para um sistema FOPDT, é apenas necessário *D* ou K_p para resolver as equações para a constante de tempo. Por exemplo, se o teste com o relé determinar D , pode-se calcular τ . Em seguida, determina-se K_p resolvendo (II-14), temos todos os parâmetros para obter a função de transferência do sistema, substituindo em (II-15) chegamos à equação II.16 e II.17 (CHENG, 2006).

$$\tau = \frac{\tan(\pi - D\omega_u)}{\omega_u} \qquad \textbf{(II.16)}$$

$$\tau = \frac{\sqrt{(K_p K_u)^2 - 1}}{\omega_u} \qquad \textbf{(II.17)}$$

Para vários tipos de função de transferência. Assim, partindo de um modelo desconhecido, o método relé permite estimar um modelo para o sistema. Este modelo pode, por exemplo, ser utilizado no desenvolvimento de uma plataforma de simulação (MELLO, 1996).

(LUYBEN, 2001) desenvolveu uma análise da forma de onda PV à excitação com o método relé. Concluiu que através da análise das formas de onda de resposta pode-se identificar algumas características do sistema, como, por exemplo, determinar se trata-se de um sistema de primeira ou segunda ordem e se o atraso é significativo (OGATA, 1985).

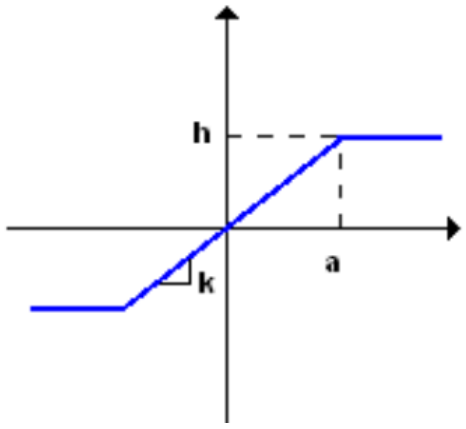
Nos pontos seguintes, apresentam-se melhorias ao Relé ideal.

2.3.3 - Relé com saturação

Como a saída do relé ideal, é uma onda retangular, e se utiliza uma onda senoidal na linearização, existe um erro de 5-20% na estimação do ganho critico, K_u. Aqui a harmônica principal é utilizada para aproximar a onda senoidal (CARDOSO, 2002).

De forma a melhorar a estimação poderiam incluir-se termos de ordem superior na análise de linearização, mas matematicamente esse processo é complicado. Outra solução para melhorar a estimação é modificar a forma do relé. Luyben estudou a aplicação de Relé com saturação (ver figura II-6), em colunas de destilação. Com este método conseguiu melhorar a estimação de K_u em 2-7% (LUYBEN, 1987).

Figura 2-6 - Relé com saturação.



Fonte: (ASTROM e HAGGLUND, 2006)

No entanto esta alteração ao Relé ideal não é atingida sem potenciais problemas. Um dos problemas que se levanta é a estimação do declive da reta de saturação. Esta ao ter declive muito acentuado irá a afetar a estimação com o ruído de processo, caso contrário, se o declive

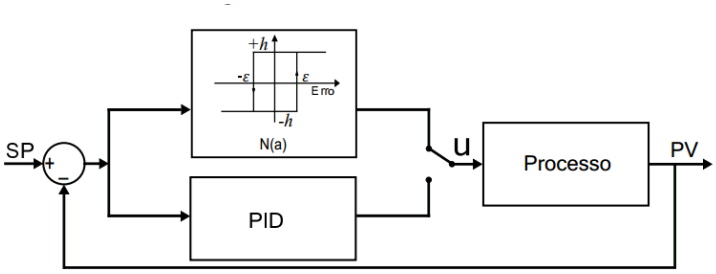
for pouco acentuado, este pode não ser capaz de gerar a oscilação pretendida (LUYBEN, 1987).

Pelos motivos referidos, este método apresenta-se mais adequado a sistemas lentos, como é o caso das colunas de destilação (LUYBEN, 1987).

2.3.4 - Relé com histerese

Buscando contornar o problema de chaveamentos indevidos do relé devido ao ruído presentes nos sinais de campo, propuseram a utilização de histerese no relé, conforme ilustra a Figura II-7 (ASTRÖM e HÄGGLUND, 1984).

Figura 2-7: Relé com saturação com histerese.



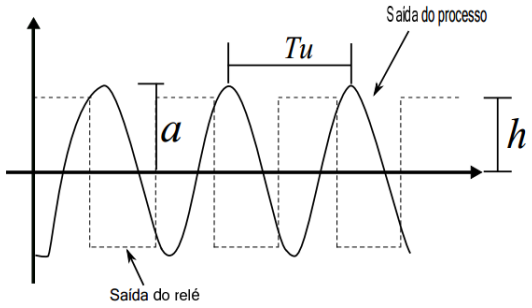
Fonte: (ASTRÖM e HÄGGLUND, 1984)

A comutação do relé é regida pela seguinte regra modificada, onde ε é a largura da histerese:

- Se [erro(t) >= ε], então u(t) = h
- Se [erro(t) < - ε], então u(t) = -h
- Se [- ε < erro(t) < ε], então u(t) = u(t-1)

Na Figura II-8 é possível observar o efeito do relé com histerese sobre a saída do processo.

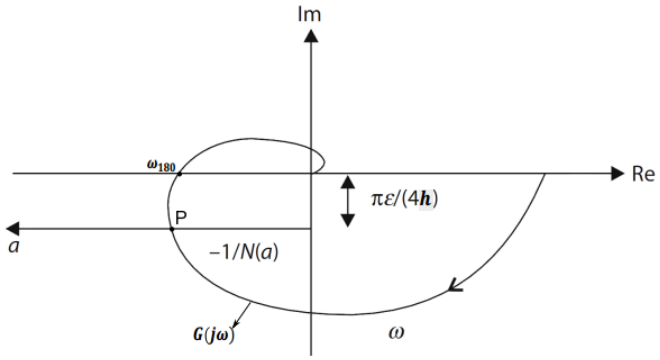
Figura 2-8 - Saída do processo sobre o ensaio do relé com histerese



Fonte: (ASTRÖM e HÄGGLUND, 1984)

Na Figura II-9 é possíel identificar, no diagrama de Nyquist, o ponto crítico, onde a defasagem entre a entrada e saída do processo atinge $-\pi$ rad . Este diagrama seria para um relé com histerese, para o relé sem histerese a função descritiva inversa ($-1/N(a)$) estaria sobre o eixo real (OGATA, 2010).

Figura 2-9: Função descritiva do relé com histerese no Diagrama de Nyquist



Fonte: (OGATA, 2010).

O nível de histerese do relé deve ser tão grande quanto necessário para reduzir significativamente a influência do ruído no ensaio(CARDOSO, 2002). Em aplicações práticas, a histerese deve ser selecionada com base na amplitude do ruído, por exemplo, duas vezes maior que a amplitude do ruído(HANG, LEE e HO, 1993), (COELHO e COELHO, 2004) e (CAMPOS e TEIXEIRA, 2006) visando o estabelecimento do ciclo limite. (WANG, DESARMO e CLUETT, 1999) sugere que o cálculo do nível do ruído seja feita com o sistema em regime por um certo tempo, depois calcular o desvio padrão (σ_{ruido}) do ruído e determinar a histerese como $\varepsilon = 3\sigma_{ruido}$.

.

A introdução de histerese no relé implica em alteração no cálculo do ganho crítico. Para o relé com histerese temos,

$$K_u = \frac{4\Delta}{\pi\sqrt{a^2 + \varepsilon^2}} \qquad \textbf{(II.18)}$$

Da mesma forma que no relé ideal o período crítico (P_u) obtém-se diretamente dos resultados experimentais.

2.4 - MICROCONTROLADORES E A PLATAFORMA ARDUINO

A principal diferença entre um microcontrolador e um computador convencional (PC) é que, além de ter menor porte (tanto no tamanho quanto no poder de processamento), o microcontrolador utiliza dispositivos diferentes para entrada e saída em geral. Por exemplo: em um PC utilizamos teclado e mouse como dispositivos de entrada e monitores e impressoras como dispositivos de saída, já em microcontroladores os dispositivos de entrada e saída são circuitos elétricos e ou eletrônicos.

O Arduino (ver figura II-9) além de ser um microcontrolador é também uma plataforma de prototipagem eletrônica criado com o objetivo de permitir o desenvolvimento de controle de sistemas interativos, de baixo custo e acessível a todos. Além disso, todo material (software, bibliotecas, hardware) é open-source, ou seja, pode ser reproduzido e usado por todos sem a necessidade de pagamento de direitos autorais. Sua plataforma é composta essencialmente de duas partes: O Hardware e o Software.

2.4.1 - A Plataforma Arduino

Como a interface do Arduino (ARDUINO, 2016) com outros dispositivos está mais perto do meio físico que a de um PC, podemos ler dados de sensores (temperatura, luz, pressão etc.) e controlar outros circuitos (lâmpadas, motores, eletrodomésticos etc.), dentre outras coisas que não conseguiríamos diretamente com um PC. A grande diferença com relação ao uso desses dispositivos, no caso do Arduino, é que, na maior parte das vezes, nós mesmos construímos os circuitos que são utilizados, ou seja, não estamos limitados apenas a produtos existentes no mercado: o limite é dado por nosso conhecimento e criatividade.

O melhor de tudo nesse projeto é que seu software, hardware e documentação são abertos. O software é livre (GNU GPL, 2016), o hardware é totalmente especificado (basta entrar no site e baixar os esquemas) e a documentação está disponível em (CREATIVE

COMMONS, 2016) - os usuários podem colaborar (seja escrevendo documentação, seja escrevendo documentação, seja traduzindo) através da wiki.

Figura 2-10: Imagem da placa do Arduino UNO

Fonte: Elaborado pelo próprio autor.

2.4.2 - O Hardware do Arduino

O hardware do Arduino (ARDUINO, 2016) é muito simples (ver figura 8), porém muito eficiente. A seguir é exposto o hardware do Arduino UNO. Esse hardware é composto das seguintes partes:

Fonte de Alimentação

Responsável por receber a energia de alimentação externa, que pode ter uma tensão de no mínimo 7 Volts e no máximo 35 Volts e uma corrente mínima de 300mA. A fonte filtra e depois regula a tensão de entrada para duas saídas: 5 Volts e 3,3 Volts. O requisito deste bloco é entregar as tensões de 5 e 3,3 Volts para que a CPU e os demais circuitos funcionem.

Núcleo CPU

O núcleo de processamento de uma placa Arduino é um microcontrolador, uma CPU, um computador completo, com memória RAM, memória de programa (ROM), uma unidade de processamento de aritmética e os dispositivos de entrada e saída. Tudo em um chip só. E é esse chip que possui todo hardware para obter dados externos, processar esses dados e devolver para o mundo externo. Os desenvolvedores do Arduino optaram em usar a linha de micro controladores da empresa ATMEL. A linha utilizada é a ATMega. Existem placas Arduino oficiais com diversos modelos desta linha, mas os mais comuns são as placas com os chips ATMega8, ATMega162 e ATMega328p. Esses modelos diferem na quantidade de memória de programa (ROM) e na configuração dos módulos de entrada e saída disponíveis.

Entradas e Saídas

Comentamos acima que basicamente toda eletrônica ativa está dentro do chip micro controlador. Para entender o que seria essa eletrônica, vamos considerar o chip mais simples usado no Arduino: o ATMega8.

O chip ATMega8 possui 28 pinos de conexões elétricas, 14 de cada lado. É através desses pinos que podemos acessar as funções do microcontrolador, enviar dados para dentro de sua memória e acionar dispositivos externos.

No Arduino UNO, os 28 pinos (veja figura 2.8) deste micro controlador são divididos da seguinte maneira:

- 14 pinos digitais de entrada ou saída (programáveis)
- 6 pinos de entrada analógica ou entrada/saída digital (programáveis)
- 5 pinos de alimentação (gnd, 5V, ref analógica)
- 1 pino de reset
- 2 pinos para conectar o cristal oscilador

Os dois primeiros itens da lista são os pinos úteis, disponíveis para o usuário utilizar. Através destes pinos que o Arduino é acoplado à eletrônica externa. Entre os 14 pinos de entrada/saída digitais temos 2 pinos que correspondem ao módulo de comunicação serial USART. Esse módulo permite comunicação entre um computador (por exemplo) e o chip.

Todos os pinos digitais e os analógicos possuem mais de uma função. Os pinos podem ser de entrada ou de saída, alguns podem servir para leituras analógicas e também como entrada digital. As funções são escolhidas pelo programador, quando escreve um programa para a sua placa.

Na placa do Arduino, os pinos úteis do micro controlador são expostos ao usuário através de conectores fêmea (com furinhos) onde podem ser encaixados conectores para construir o circuito externo à placa do Arduino.

Entradas Digitais

No total temos disponíveis 20 pinos que podem ser utilizados como entradas digitais. Os 14 pinos digitais mais os 6 pinos analógicos, podem ser programados para serem entradas digitais. Quando um pino é programado para funcionar como entrada digital, através do programa que escrevemos colocamos um comando que ao ser executado efetua a "leitura" da

tensão aplicada ao pino que está sendo lido. Então, após a execução deste comando, sabemos se o pino encontra-se em um estado "alto" ou "baixo".

Na prática, o programa pode saber se um pino está alimentado com 0 (zero) ou 5 Volts. Essa função é utilizada geralmente para identificar se um botão está pressionado, ou um sensor está "sentindo" alguma coisa no mundo externo.

Note que a função de entrada digital apenas entrega 0 ou 1, sem tensão ou com tensão. Não é possível saber quanta tensão está sendo aplicada no pino. Para isso usamos as entradas analógicas.

Entradas Analógicas

Temos disponíveis no Arduino Uno, 6 entradas analógicas. Ao contrário de uma entrada digital, que nos informa apenas se existe ou não uma tensão aplicada em seu pino, a entrada analógica é capaz de medir a tensão aplicada. Através da entrada analógica, conseguimos utilizar sensores que convertem alguma grandeza física em um valor de tensão que depois é lido pela entrada analógica.

Saídas Digitais

Com uma saída digital podemos fazer com que um pino libere 0 volts ou 5 volts. Com um pino programado como saída digital, podemos acender um led, ligar um relé, acionar um motor, dentre diversas outras coisas. Podemos programar o Arduino para no máximo 20 saídas digitais, porém podemos utilizar um ou mais pinos para controlar um bloco de pinos.

Pinos com funções especiais

Existem pinos do Arduino que possuem características especiais, que podem ser usadas efetuando as configurações adequadas através da programação. São eles:

- *PWM***:** Tratado como saída analógica, na verdade é uma saída digital que gera um sinal alternado (0 e 1) onde o tempo que o pino fica em nível 1 (ligado) é controlado. É usado para controlar velocidade de motores, ou gerar tensões com valores controlados pelo programa. Pinos 3, 5, 6, 9, 10 e 11.
- *Porta Serial USART*: Podemos usar um pino para transmitir e um pino para receber dados no formato serial assíncrono (USART). Podemos conectar um módulo de transmissão de dados via bluetooth por exemplo e nos comunicarmos com o Arduino remotamente. Pinos 0 (rx recebe dados) e pino 1 (tx envia dados).

- *Comparador analógico:* Podemos usar dois pinos para comparar duas tensões externas, sem precisar fazer um programa que leia essas tensões e as compare. Essa é uma forma muito rápida de comparar tensões e é feita pelo hardware sem envolver programação. Pinos 6 e 7.
- *Interrupção Externa:* Podemos programar um pino para avisar o software sobre alguma mudança em seu estado. Podemos ligar um botão a esse pino, por exemplo, e cada vez que alguém pressiona esse botão o programa rodando dentro da placa é desviado para um bloco que você escolheu. Usado para detectar eventos externos à placa. Pinos 2 e 3.
- *Porta SPI:* É um padrão de comunicação serial síncrono, bem mais rápido que a USART. É nessa porta que conectamos cartões de memória (SD) e muitas outras coisas. Pinos 10 (SS), 11 (MOSI), 12 (MISO) e 13 (SCK).

Firmware

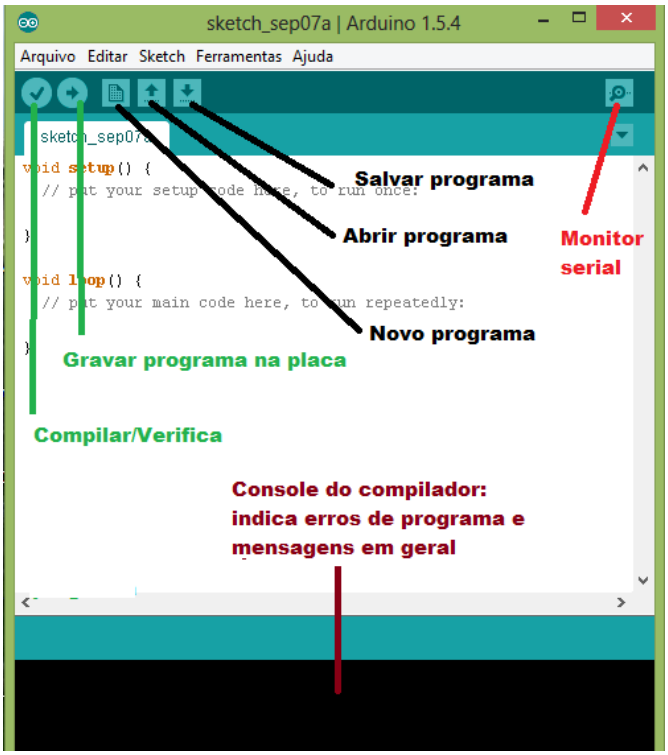
É simplesmente um software que é carregado dentro da memória do microcontrolador. Tecnicamente o firmware é a combinação de uma memória ROM, somente para leitura, e um programa que fica gravado neste tipo de memória. E esse é o caso do microcontrolador que a placa Arduino usa.

2.4.3 - O Software do Arduino

Quando tratamos de software na plataforma do Arduino (Arduino Playground - Java), podemos referir-nos: ao ambiente de desenvolvimento integrado do Arduino e ao software desenvolvido por nós para enviar para a nossa placa. O ambiente de desenvolvimento do Arduino é um compilador gcc (C e C++) que usa uma interface gráfica construída em Java. Basicamente se resume a um programa IDE muito simples de se utilizar e de estender com bibliotecas que podem ser facilmente encontradas. As funções da IDE do Arduino são basicamente duas: Permitir o desenvolvimento de um software e enviá-lo à placa para que possa ser executado.

Para realizar o download do software basta ir até a página oficial do Arduino (http://www.arduino.cc/), escolher o seu SO (existe pra Linux, Mac e Windows) e baixá-lo. Obviamente, por ser open source, é gratuito. Depois de baixado não necessita de nenhuma instalação, é só abrir o IDE e começar a utilizar (Arduino Playground - Java). A interface de programação é mostrada na figura II-10.

Figura 2-11: Interface de programação do Arduino



Fonte: Elaborado pelo próprio autor.

Para começar a utilizar, devemos escolher qual placa estamos utilizando, assim vamos em Tools > Board e à escolhemos. O IDE possui também uma quantidade imensa de exemplos. Para utilizá-los basta ir a File > Examples e escolher qual deles você quer testar, de acordo com sua necessidade.

Estrutura de um Programa

Aproveitando os exemplos que o IDE fornece, o primeiro programa (código fonte abaixo) vai ser acender um led.

```
/*  Blink
Turns on an LED on for one second, then off for one second, repeatedly.
  This example code is in the public domain.
 */
// Pin 13 has an LED connected on most Arduino boards.
// give it a name:int led = 13;
// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}
// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
```

```
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

Aproveitando os exemplos que o IDE fornece, nosso primeiro programa vai ser acender um led. O programa para o Arduino é dividido em duas partes principais: Setup e Loop, como indicam as setas na imagem.

- *A função setup*: serve para inicialização da placa e do programa. Esta sessão é executada uma vez quando a placa é ligada ou resetada através do botão. Aqui, informamos para o hardware da placa o que vamos utilizar dele. No exemplo, vamos informar para a placa que o pino 13 será uma saída digital onde está conectado um LED (no Arduino UNO o pino 13 possui um led integrado).
- *A função loop*: é como se fosse a main () da placa. O programa escrito dentro da função loop é executado indefinidamente, ou seja, ao terminar a execução da última linha desta função, o programa inicia novamente a partir da primeira linha da função loop e continua a executar até que a placa seja desligada ou o botão de reset seja pressionado.

Analisando o resto do programa, o comando digitalWrite escreve na saída do pino 13 o nível de tensão HIGH (5v), acendendo o Led. O comando delay é apenas para o programa aguardar 1000 milésimos. Em seguida, o nível de tensão é alterado para LOW (0 v) e o Led apaga. E assim é repetido infinitamente, até ser desligado.

Com o programa feito, compilamos o mesmo para verificarmos se não existe nem um erro. Caso não contenha erro, agora temos que enviá-lo para placa através do botão de upload (os botões estão especificados na figura II-10). Após o envio os Led's RX e TX deverão piscar, informando que o código está sendo carregado. Logo após o Arduino começará a executar o programa que lhe foi enviado.

2.5 - TIRISTORES

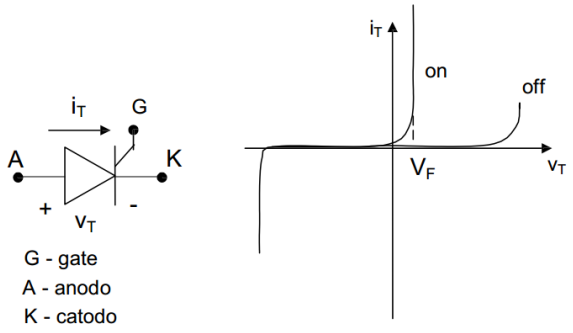
O termo tiristor engloba uma família de dispositivos semicondutores que operam em regime chaveado, tendo em comum uma estrutura de 4 camadas semicondutoras numa sequência p-n-p-n, apresentando um funcionamento biestável. O tiristor de uso mais difundido é o SCR (Retificador Controlado de Silício), usualmente chamado simplesmente de tiristor. Outros componentes, no entanto, possuem basicamente uma mesma estrutura:

LASCR (SCR ativado por luz), também chamado de LTT (Light Triggered Thyristor), TRIAC (tiristor triodo bidirecional), DIAC (tiristor diodo bidirecional), GTO (tiristor comutável pela porta), MCT (Tiristor controlado por MOS)(ALMEIDA, 1996).

2.5.1 - SCR (Retificador Controlado de Silício)

A Fig. II-1 mostra o símbolo do SCR e suas características de operação através da curva **v** x **i**.

Figura 2-12: Tiristor - símbolo e característica de operação do SCR



Fonte: (ALMEIDA, 1996)

Quando o SCR está diretamente polarizado ($v_T > 0$) e é aplicado um pulso positivo de corrente de seu gate (G) para o catodo (K), este dispositivo entra em condução permitindo circulação da corrente I_T entre anodo e catodo. Uma vez em condução, o pulso de gate pode ser removido e o SCR continua em condução como um diodo, ou seja, não pode ser comandado a bloquear. Para que o tal deixe de conduzir é necessário que a corrente I_T caia abaixo do valor mínimo de manutenção (I_H), desta forma o SCR entra novamente na região de corte. Quando o SCR está reversamente polarizado ($v_T < 0$) ele não conduz (BARBI, 1986).

2.5.1.1 - Formas de disparar um SCR

A seguir, segundo (BARBI, 1986), são apresentadas as formas de disparo de um SCR:

a) Disparo por pulso de gatilho

Esta é a forma usual de disparo. Como já foi dito, quando o SCR está diretamente polarizado e recebe um pulso positivo de corrente de gate para catodo, ele entra em condução. O componente se manterá em condução desde que, após o processo de

entrada em condução, a corrente de anodo tenha atingido um valor superior ao limite IL (corrente de "latching"). Sendo assim, a duração do sinal de disparo deve ser tal que permita à corrente atingir o valor IL antes que o sinal de disparo seja retirado.

b) *Disparo por sobretensão*

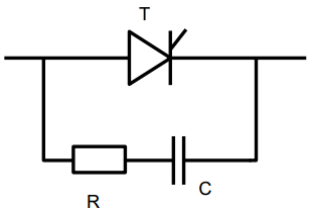
À medida que se aumenta a tensão entre anodo e catodo (diretamente polarizado), é possível iniciar o processo de condução mesmo sem corrente no gate. Este procedimento, nem sempre destrutivo, raramente é utilizado na prática.

c) *Disparo por taxa de crescimento da tensão direta*

Uma vez que o SCR esteja diretamente polarizado, mesmo sem corrente de gate, pode haver a entrada em condução devido à taxa de crescimento da tensão entre anodo e catodo. Se esta taxa for suficientemente elevada (a tensão crescer rapidamente), o SCR entra em condução.

Este disparo, normalmente não desejado, é evitado pela ação de um circuito de proteção conhecido como *snubber*, que se trata de um circuito RC em paralelo com o tiristor. Este circuito é mostrado na figura II-12.

Figura 2-13: Tiristor com circuito *snubber*.



Fonte: (BARBI, 1986)

2.5.1.2 - Métodos de comutação de um SCR

Se por um lado é fácil a entrada em condução de um SCR, o mesmo não ocorre para o seu bloqueio. A condição para o bloqueio é que a corrente de anodo fique abaixo do valor I_H (corrente de manutenção) cujo valor é estabelecido pelo fabricante. Existem duas formas básicas de bloqueio de um SCR (ALMEIDA, 1996):

a) Comutação Natural
Em um circuito CA, a corrente normalmente passa por zero em algum instante levando o SCR ao bloqueio. Este tipo de comutação é chamado comutação pela rede. Em circuitos CC, onde a comutação depende da característica da própria carga, a comutação é definida como comutação pela carga.

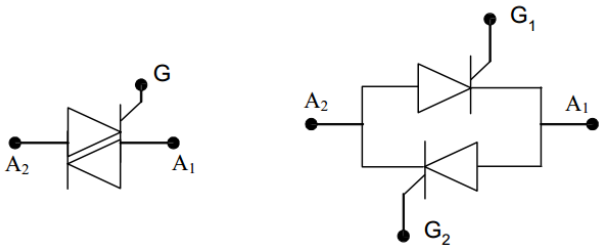
b) Comutação Forçada
É utilizada em circuitos CC onde não é possível a reversão da corrente de anodo. Sendo assim, deve-se oferecer um caminho alternativo para a corrente, enquanto se aplica uma tensão reversa sobre o SCR. Normalmente é utilizado um capacitor carregado previamente com uma tensão reversa, em relação aos terminais do SCR.

No instante desejado para o corte, coloca-se o capacitor em paralelo com o SCR aplicando sobre ele uma tensão reversa. Um exemplo de circuito que utiliza deste tipo de comutação são os inversores.

2.5.2 - TRIAC

O TRIAC é um tiristor que permite a condução de corrente nos dois sentidos, entrando em condução e bloqueando de modo análogo ao SCR. Uma visão simplificada do TRIAC, é a de uma associação de dois SCR's conectados em antiparalelo. Entretanto, note que no caso de dois SCR's é necessário dois terminais de gatilho. A Figura II-13 mostra o símbolo do TRIAC e a comparação com dois SCR's. Como é bidirecional, os termos anodo e catodo ficam sem sentido, assim, os terminais do TRIAC são chamados anodo 1 (A1), anodo 2 (A2) e gatilho (G). Além de conduzir nos dois sentidos, o TRIAC pode ser disparado tanto com pulso positivo como por pulso negativo de corrente aplicado entre o gate (G) e o anodo1 (A1)(ALMEIDA, 1996).

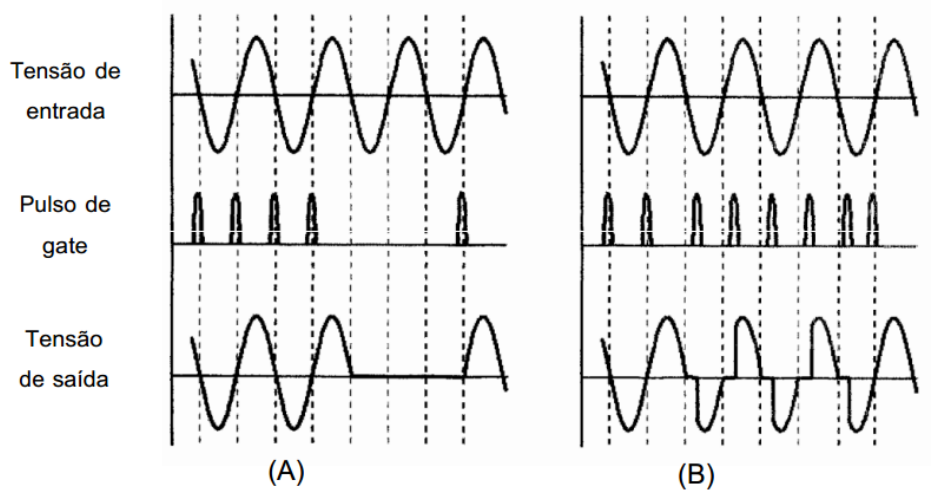
Figura 2-14: Símbolo do TRIAC e comparação com dois SCR’s em antiparalelo.



Fonte: (ALMEIDA, 1996)

O TRIAC é um dispositivo utilizado em baixos níveis de potência quando comparado com o SCR. Um exemplo de aplicação é o controle do fluxo de corrente alternada. Este controle pode ser feito de duas formas: (A) Controle por ciclos inteiros e (B) Controle do ângulo de fase. Conforme mostra a Figura II-14.

Figura 2-15: Controle do fluxo de potência por TRIAC’s. (A) Controle por ciclos inteiros, (B) Controle do ângulo de fase.

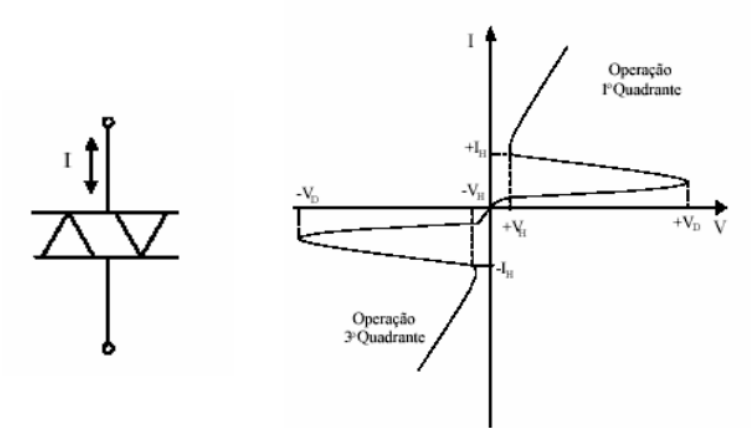


Fonte:(ALMEIDA, 1986)

2.5.3 - O DIAC

Assim como o TRIAC, o DIAC é um dispositivo que permite condução nos dois sentidos tendo aplicações em baixos níveis de potência. Entretanto, a entrada em condução não ocorre devido a um pulso de corrente no gate, mas a partir de uma tensão de disparo aplicada entre seus terminais. A Figura II-15 mostra a característica tensão x corrente e o símbolo comumente utilizado para a representação do DIAC (ALMEIDA, 1986).

Figura 2-16: Símbolo e característica do DIAC.



Fonte: (ALMEIDA, 1986)

Quando o DIAC está submetido a uma tensão inferior a V_D (tensão de disparo), o mesmo não conduz. Depois de atingido o valor da tensão de disparo, o DIAC entra em condução, mantendo uma pequena tensão entre seus terminais. Para o seu bloqueio é necessário que a corrente assuma valor inferior a I_H (corrente de manutenção) (BARBI, 1986).

CAPÍTULO 3

MATERIAIS E MÉTODOS

3.1 - MÉTODOS

A obtenção do modelo matemático do sistema é uma das maiores dificuldades encontradas pelo projetista de um sistema de controle. Existem diferentes métodos para se obter o modelo, sendo que os mais conhecidos são : a resposta transitória temporal e o método de resposta em frequência.

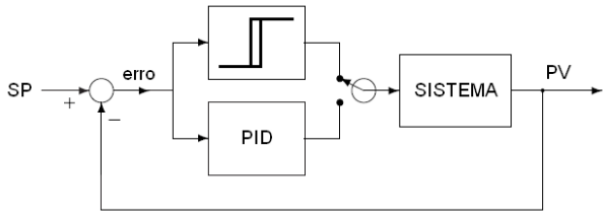
Os métodos tradicionais de levantamento da resposta em frequência de um processo a ser controlado utiliza a medição da resposta estacionária de um processo sujeito a respostas senoidais, como por exemplo, o segundo método de Ziegler e Nichols. Na sintonia baseada no método relé, a frequência de interesse é identificada de forma automática, sendo esta chamada de frequência crítica (Åström e Hägglund, 2001).

O relé por realimentação, como mostra a figura III-1, é um sistema que no primeiro momento, através de um chaveamento, substitui o controlador e atua no processo a partir da saída (u) do controlador que corresponde a variável manipulada (MV). O erro é representado pela diferença entre a saída do processo (PV) e a especificação de setpoint (SP).

O estudo de caso deste trabalho utiliza-se de um sistema de controle de temperatura de um reservatório com água usando o método relé para realizar a sintonia de forma automática. Esta variável de controle ou variável do processo (PV) diz respeito temperatura da água no reservatório que será mensurado através de um sensor de temperatura, sendo que essa medida será processada por um microcontrolador (ATMega), este por sua vez, dispara um TRIAC num determinado ângulo gerando uma potência (MV) que aciona uma resistência de aquecimento fazendo esta aumentar ou diminuir a temperatura. O sistema está conectado a um computador via porta serial cujos dados de entrada do sensor de temperatura é transmitido e processado num programa desenvolvido em Java que mostra esses dados em um gráfico de saída do sistema e que calcula e apresenta os parâmetros do controlador.

Após realizar a sintonia o próprio sistema computacional poderá indicar os parâmetros e em seguida poderá executar o sistema já com o controlador, chaveando do modo de sintonia para o modo de execução do sistema de controle. Vale salientar que o controlador aqui referido trata-se de um controlador PID, como mostrado na figura III-1.

Figura 3-1: Sistema de Sintonia usando Método Relé



Fonte: (COLOGNI, 2008)

3.1.1 – Etapas para Automatização da Técnica de Auto Sintonia usando o Método Relé

– O processo de automatização começa com a necessidade do sistema alcançar o estado permanente, assim, faz-se necessário criar inicialmente um processo P1 para colocar o sistema *em estado permanente*, onde a temperatura deverá crescer até atingir o setpoint (SP). Ver figura III-2.

– Uma vez que o sistema estará em estado permanente, dependendo da escolha do usuário do sistema, poderá optar pelo algoritmo do relé ideal ou relé por histerese (ver figura III-3).

– Foi necessário criar um circuito que receberá os comandos de aumento e diminuição da potência da resistência de elétrica de aquecimento.

– Também foi construída uma placa que se conecta via soquete aos pinos do arduino, uma vez que através destes pinos é que são enviados os comandos tanto para a resistência de aquecimento como para ter acesso ao valor lido pelo sensor de temperatura.

– O programa que acionará a resistência de aquecimento e que faz a leitura do sensor serão escritos em linguagem C++, e serão codificados no ambiente de desenvolvimento do arduino.

– Será construída uma interface em linguagem Java para ser utilizada pelo usuário onde poderão ser informados: o tempo de amostragem, o setpoint, o nome do projeto do controlador. Sendo que essas informações poderão ser salvas em banco de dados e recuperada

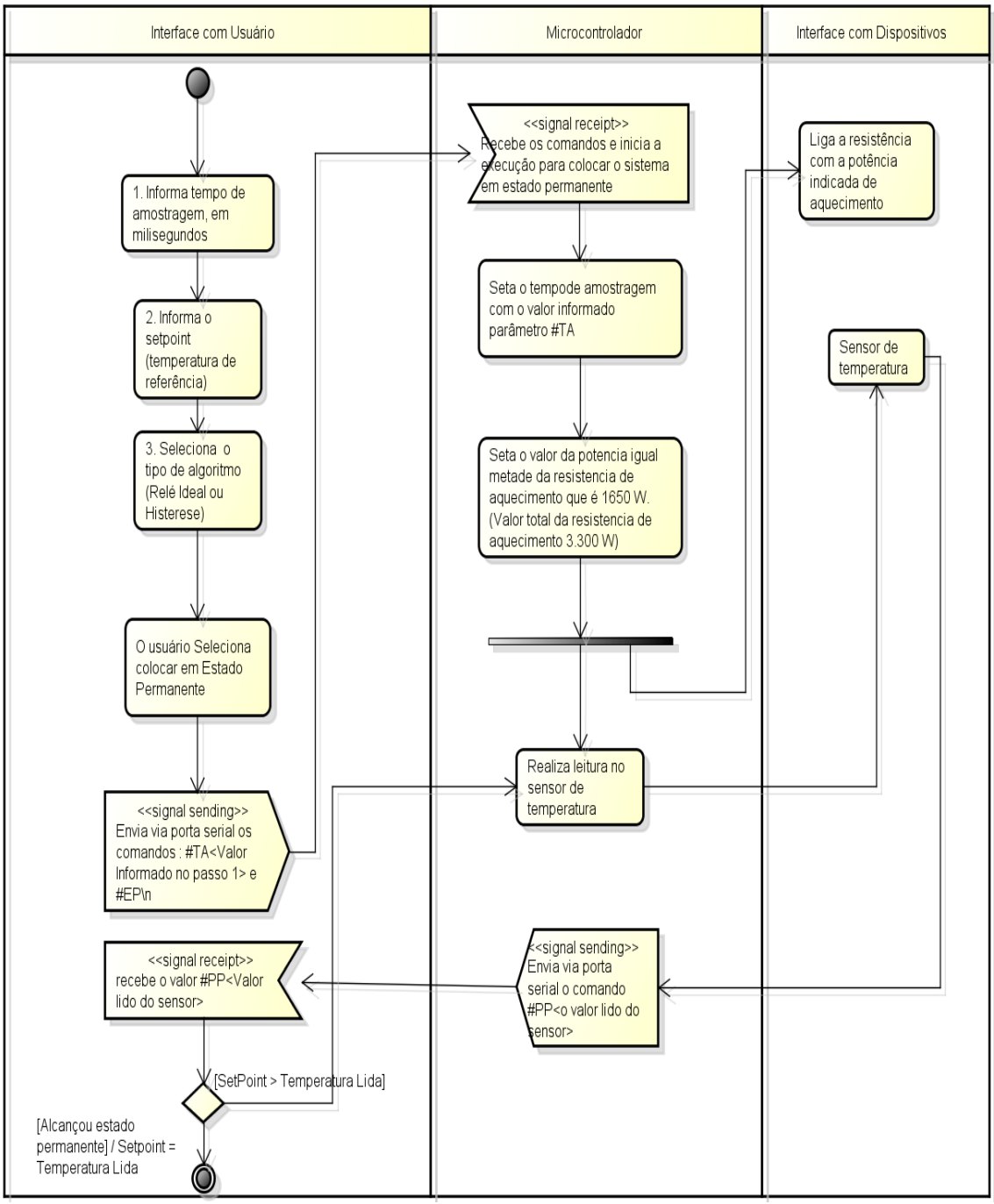
ou removidas posteriormente. A interface deverá permitir também realizar a sintonia do controlador bem como executar a planta com o controlador projetado.

3.1.2 - Cálculo do Desvio Padrão do Ruído (σ_{ruido})

Conforme descrito na seção 2.3 que descreve o método do relé e inclusive por histerese, nesse caso o valor coletado do sensor, deve passar por algumas etapas de filtro para eliminar os ruídos e são essas:

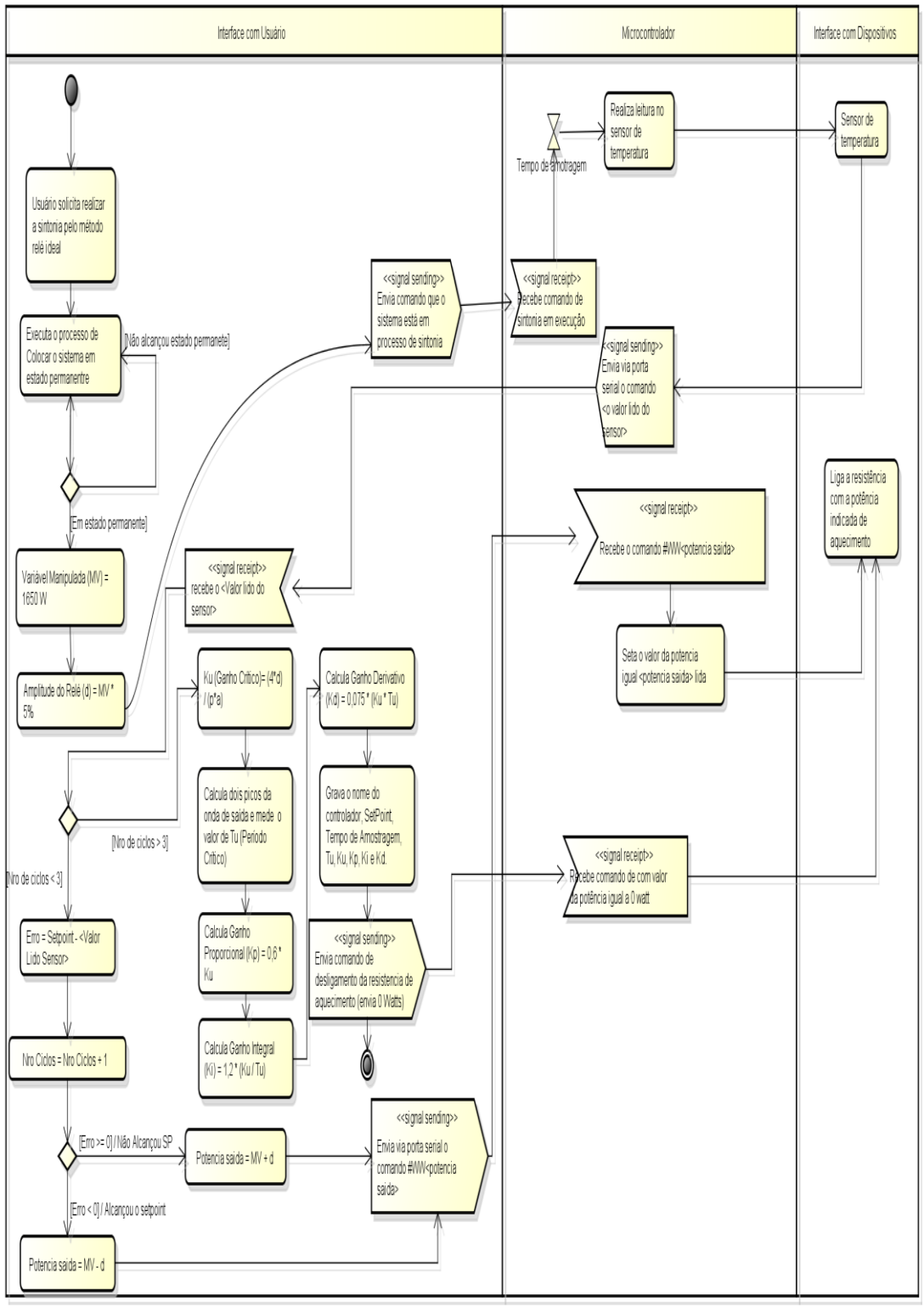
- Os três primeiros picos da senóide do sinal de saída serão desconsiderados, uma vez que o sistema encontra-se em processo de estabilização (isto vale para relé ideal e por histerese).
- No caso do relé histerese, é necessário calcular a partir do quarto pico até o sétimo pico a média $\bar{y}$.
- Em seguida calcular a somatoria da variância das amostras para cada valor de pico lido pelo sensor de temperatura(y). Ou seja, calcula-se variância (s) onde $s = \frac{\sum_{i=4}^{7} (y_i - \bar{y})^2)}{4}$.
- O cálculo do desvio padrão do ruído (σ_{ruido}) é $\sqrt{s}$.

Figura 3-2: O sistema computacional deverá implementar o processo P1 mostrado abaixo para colocar o sistema em estado permanente antes de iniciar a sintonia.



Fonte: Elaborado pelo próprio autor.

Figura 3-3: O sistema computacional deverá implementar o processo P2 mostrado para que o sistema execute o método do relé ideal como uma opção.



Fonte: Elaborado pelo próprio autor.

3.2 - DISPOSITIVOS UTILIZADOS

3.2.1 - Sensor Temperatura

O sensor de tempertura DS18B20 (mostrado na figura III-5) foi desenvolvido para ser utilizado pelo arduino e pode ler temperaturas na faixa de -127 ºC a 85ºC, possui três cabos para ser conectado no arduino: o *cabo vermelho* a ser ligado no +Vcc do arduino, o *cabo preto* a ser ligado no pino GND do arduino e o *cabo amarelo* a ser ligado no pino de dados do arduino, nesse caso deverá estar ligado num pino analógico, no caso selecionei neste projeto o pino 7 do arduino.(Temperatuur meten de Arduino en een DS18B20, 2014)

Figura 3-4: Sensor de Temperatura DS18B20

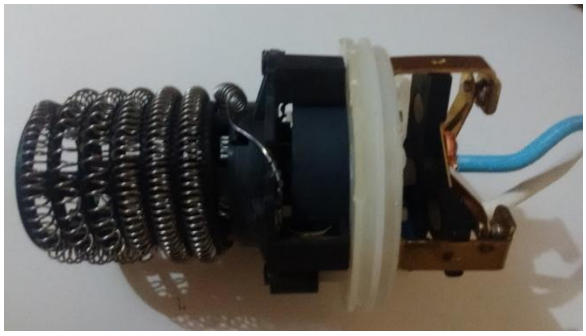
Fonte: Elaborado próprio autor.

No processo será utilizado para medir a temperartura da água no reservatório, essa temperatura estará disponível no arduino e será comparada com a temperatura de referência que desejamos controlar. Foram, lidos artigos acerca desse dispositivos e a sua folha de dados.

3.2.2 - Resistência Elétrica de Aquecimento

Foi utilizada uma resistência elétrica de um chuveiro elétrico de 3.300 W da marca *Fame* com tensão de 127 V (mostrado na figura III-6). Para acionar essa resistência foi utilizado um circuito eletrônico de potência com um TRIAC que permite aumentar ou reduzir a potência do mesmo.

Figura 3-5: Resistência elétrica de aquecimento de um chuveiro com potência de 3.300W usado para aquecer o líquido do recipiente com água usado nesse sistema.



Fonte: Elaborado pelo próprio autor.

3.2.4 - Placa do Arduíno com microcontrolador ATMega

É utilizada uma placa protótipo arduino UNO (*ATMega168*) para executar o programa de acesso ao sensor de temperatura *DS18B20* e a comunicação serial que enviará conforme indicado no tempo de amostragem da interface em Java, o valor instantâneo da temperatura.

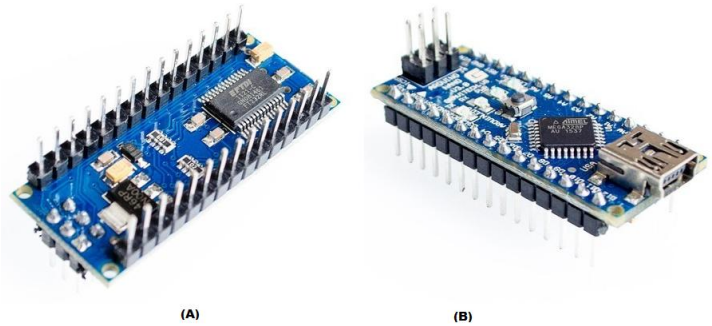
Figura 3-6: Placa do Arduino Uno ATMega168

Fonte: Elaborado pelo próprio autor

É usada uma segunda placa protótipo arduino NANO (ATMega328), veja a figura III-8, cuja função é detectar a passagem de zeros da onda senoidal fornecida pela rede elétrica e servir de referência para setar o ângulo de disparo do TRIAC. Assim, está placa também é necessária para realizar comunicação serial e também atuar no sentido de aumentar ou

diminuir o valor da potência injetada na resistência elétrica que provoca o aquecimento (figura III-6). Este é então o circuito atuador do sistema, cujo valor setado será a MV (Variável Modificada). Por fim, este circuito também dispara um ventilador quando a temperatura instantânea exceder a temperatura de referência (SetPoint) de maneira a trazer a temperatura de volta para o valor de referência.

Figura 3-7: Arduino NANO : (A) Imagem da face inferior (B) Imagem pinos face superior.



Fonte: Elaborado pelo próprio autor.

3.2.5 - Reservatório de Armazenamento de Líquido

É utilizado um reservatório de 10 cm x 40 cm x 45 cm de vidro para armazenar a água a ser controlada a temperatura, como mostrado na figura III-9. Neste serão adaptados o sensor de temperatura (figura III-5), a resistência de aquecimento (figura III-6) e um ventilador (figura III-10).

Figura 3-8: Tanque que manterá a água aquecida, nesta imagem é mostrado sem a resistência de aquecimento e o sensor de temperatura.

Fonte: Elaborado pelo próprio autor.

Figura 3-9: Ventilador de 110 Volts AC.

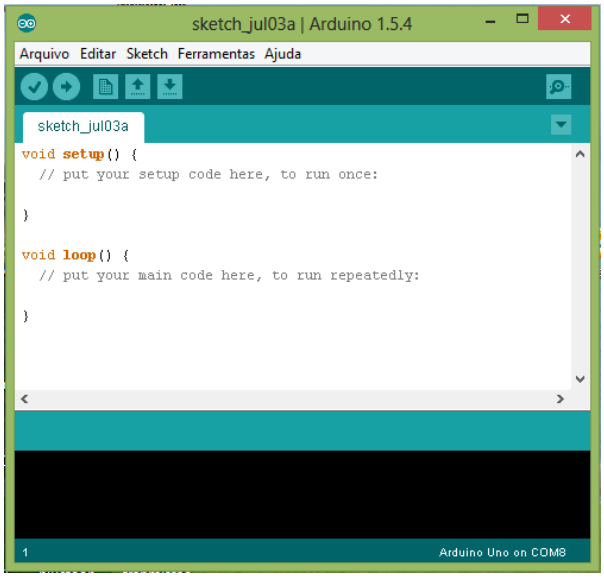
Fonte: Elaborado pelo próprio autor.

3.3 - SOFTWARE PARA DESENVOLVIMENTO DE PROGRAMAS

3.3.1 - Arduino IDE

É a interface gráfica que permite o desenvolvimento dos programas arduino. A ferramenta permite digitar código em linguagem em C e C++, depurar, compilar e transferir o programa via porta USB para a placa do arduino.

Figura 3-10: Tela da IDE do Arduino



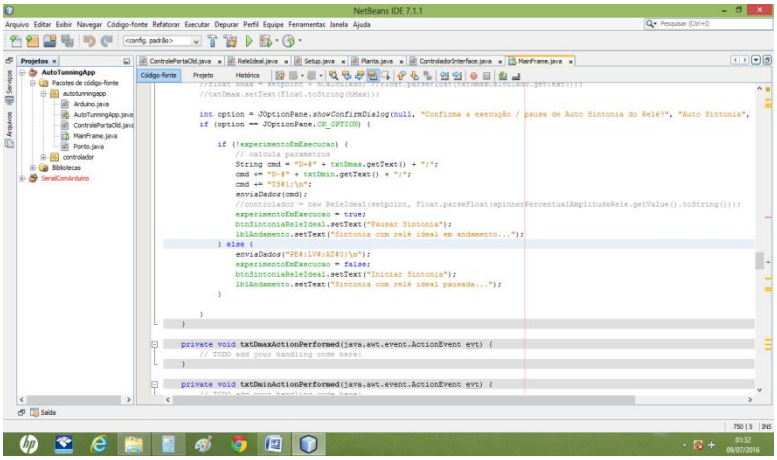
Fonte: Elaborado pelo próprio autor

O programa desenvolvido na IDE do arduino irá interagir com o sensor de temperatura, a resistência de aquecimento e o ventilador. Nesse programa ainda é a realizada a comunicação serial com a aplicação Java, que gerencia a interface com o usuário atualizando o gráfico de saída (temperatura x tempo) e processa os dados recebidos pelo sensor de temperatura que serve como entrada para o PID que processa o algoritmo que implementa o método de sintonia do relé.

3.3.2 - NetBeans IDE

É a interface gráfica utilizada para desenvolver as aplicações na linguagem Java, como mostrado na figura III-12. Nesse ambiente será construído os programas que irão interagir com a placa do arduino através da porta USB no modo serial. Trocando informações com a placa do arduino de maneira a produzir o gráfico de saída do processo.

Figura 3-11: Tela da IDE do NetBeans 7



Fonte: Elaborado pelo próprio autor.

CAPÍTULO 4

ESTUDO DE CASO – CARACTERIZAÇÃO

4.1 - CONTROLE DE TEMPERATURA DA ÁGUA DO RESERVATÓRIO

O desenvolvimento da ferramenta de software foi baseado num ensaio, onde existe um recipiente com certo volume de água que sofrerá aquecimento através de uma resistência elétrica de aquecimento (mesma resistência usada em chuveiro elétrico). A temperatura da água é continuamente monitorada através de um sensor de temperatura, cuja leitura é enviada a placa do microcontrolador arduino, que deverá controlar a temperatura próxima do valor de referência (Setpoint) informado e que serviu de parâmetro para realizar a sintonia. O contexto desse ensaio é mostrado na figura IV-1.

Figura 4-1: Tanque que manterá a água aquecida, nesta imagem é mostrado a resistência de aquecimento e o sensor de temperatura.

4.2 – A PLACA DE INTERFACE DESENVOLVIDA

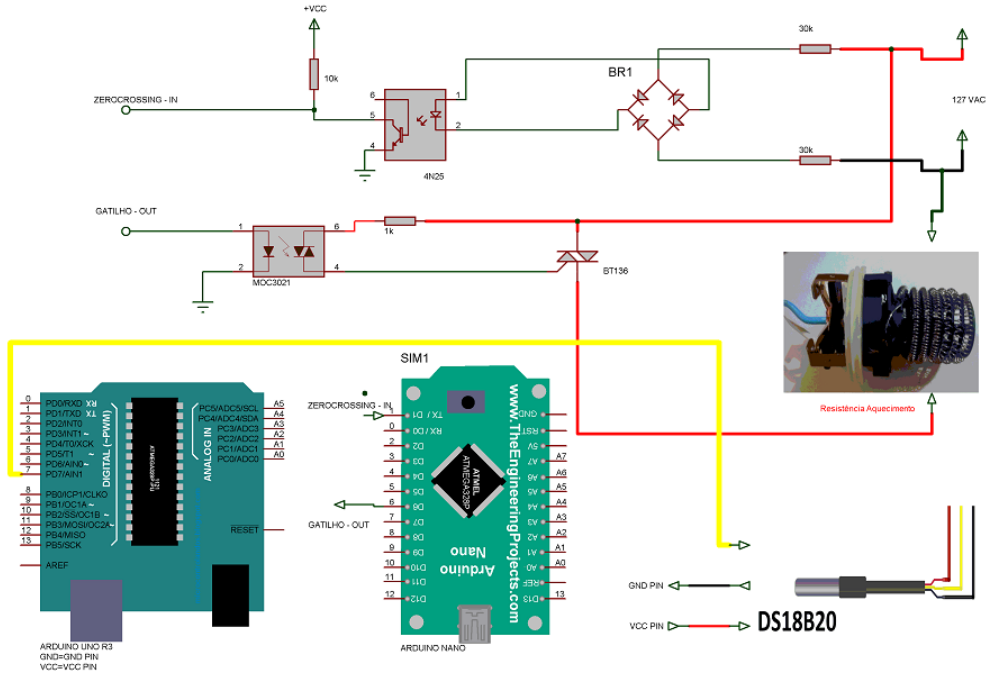
A placa funciona basicamente através do programa armazenado na placa do arduino, sendo este programa responsável pela recepção de comandos da interface de software em Java, da resistência elétrica de aquecimento e do sensor temperatura.

O arduino, como já foi mostrado, trata-se de uma placa de hardware livre e que se utiliza do microcontrolador ATMega. Assim, o arduino tem a função de armazenar e executar o programa. A placa de interface que foi desenvolvida deve ser conectada ao arduino através de pinos soldados à placa.

A placa de interface possui basicamente um regulador de potência da resistência elétrica de aquecimento e um sensor de temperatura que é ligado diretamente ao arduino, através de pinos de conexão compatíveis com o arduíno.

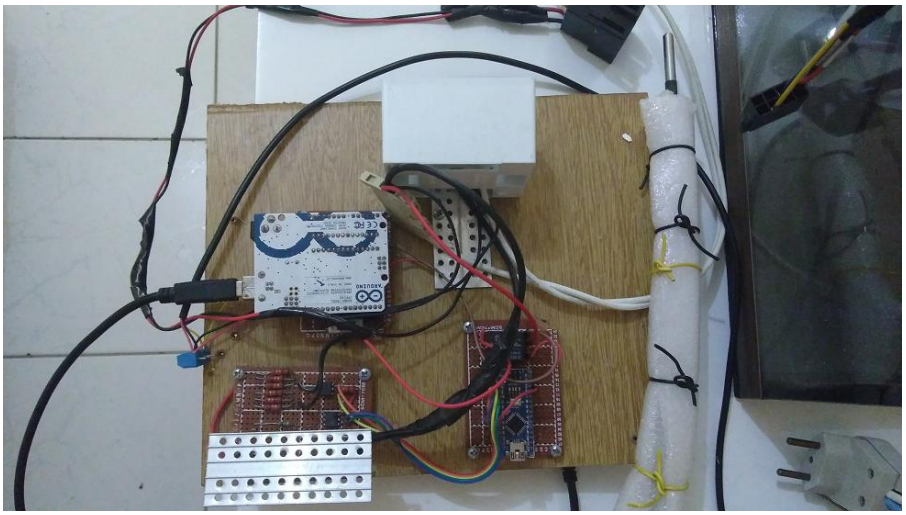
O esquema elétrico resultante é mostrado na Figura IV-2 e a placa física resultante deste esquema é mostrada na figura IV-3.

Figura 4-2: Esquema elétrico da placa de interface e arduino acoplado.



Fonte: Elaborado pelo próprio autor.

Figura 4-3: Imagem da placa física de interface e arduino acoplado.



Fonte: Elaborado pelo próprio autor.

4.2.1 - Programa em Linguagem “C” Embarcado na Placa do Arduino

O comando da resistência de aquecimento e sensor de temperatura do sistema será realizado através de um programa em linguagem “C” implantado no arduino. Esse programa, além de comandar os dispositivos e sensor também transfere o valor lido do sensor de temperatura para a interface do usuário, através da porta serial. Além de comandar via porta USB a transmissão e recepção de dados e parâmetros. Os parâmetros são comandos que permitem a comunicação com o a interface de software.

O programa será implantado no arduino e comandará via comandos da interface Java os acionamentos através da placar de interface de hardware. O programa foi codificado e implantado no arduino usando a interface de desenvolvimento apresentado na seção 3.3.1. O código fonte do programa encontra-se no anexo I e II.

4.3 - INTERFACE JAVA DE INTERAÇÃO COM O USUÁRIO

A interface com o usuário foi desenvolvida em linguagem Java, neste trabalho foi utilizado o ambiente de desenvolvimento do NetBeans. A telas (ou interfaces com o usuário) foram desenvolvidas em Java Swing. Os dados de parâmetros do controlador são armazenados em arquivos de texto. A tela resultante é mostrada na figura IV-4. E o código fonte referente à essa interface é descrito no anexo III e IV.

4.3.1 - A interface com o Usuário

A interface permite executar as seguintes funcionalidades:

1) Conforme a figura IV-4, a tela possui duas áreas reservadas para plotar gráficos. Uma a esquerda que mostra a saída do sistema e a mais a da direita que mostra o sinal de entrada do controlador. No momento da sintonia o gráfico mais a esquerda apresentará o sinal de relé, uma vez que nesse momento é chaveado do sinal de controle para o sinal de relé.
2) Poderá abrir um projeto de controlador, ao clicar no botão *Abrir* o sistema mostra uma tela com o sistema de diretórios que permite ao usuário selecionar os projetos gravados.
3) O sistema como descrito nos objetivos poderá realizar sintonia do relé ideal ou por histerese. Em ambos os casos será necessário preencher a variação de amplitude da altura do sinal de relé, o campo *Variação H(%),* para isto é necessário preencher o campo *Variável Manipulada MV*, isso equivale as amplitudes do relé. Os campos *Tempo de amostra* e o *Setpoint* são campos obrigatórios.
4) O sistema pode realizar a sintonia do Relé Ideal e do Relé Histerese, para realizar o último basta completar as informações, marcando o campo *com histerese* e o o campo *% histerese*. Em seguida, clicando no botão *Iniciar PID* começamos o processo de sintonia. A primeira etapa é colocar o sistema em estado permanente, isto será mostrado na barra de status e não haverá qualquer atualização dos gráficos.
5) Após colocar em estado permanente, os painéis gráficos começam a ser atualizados, como o andamento, o gráfico plotando a saída do sistema e o gráfico de saída do relé. Também após o término do processo de sincronia, mostra os valores dos parâmetros do controlador PID, ou seja, os valores do ganho crítico, período crítico, amplitude da senóide de saída, a largura de histerese, e os parâmetros do controlar proporcional, integral e derivativo.
6) Se a opção de histerese for selecionada os sete primeiros ciclos são executados com relé ideal, sendo que os três primeiros ciclos são descartados e a partir do quarto pico até o sétimo pico é então calculado o desvio padrão. A seguir é mostrado trecho do código do programa em linguagem Java que realiza esse cálculo:

```
private double calculaDesvioPadrao() {
    Ponto pico7 = picos.get(qtdePicos-1);
    Ponto pico6 = picos.get(qtdePicos-2);
```

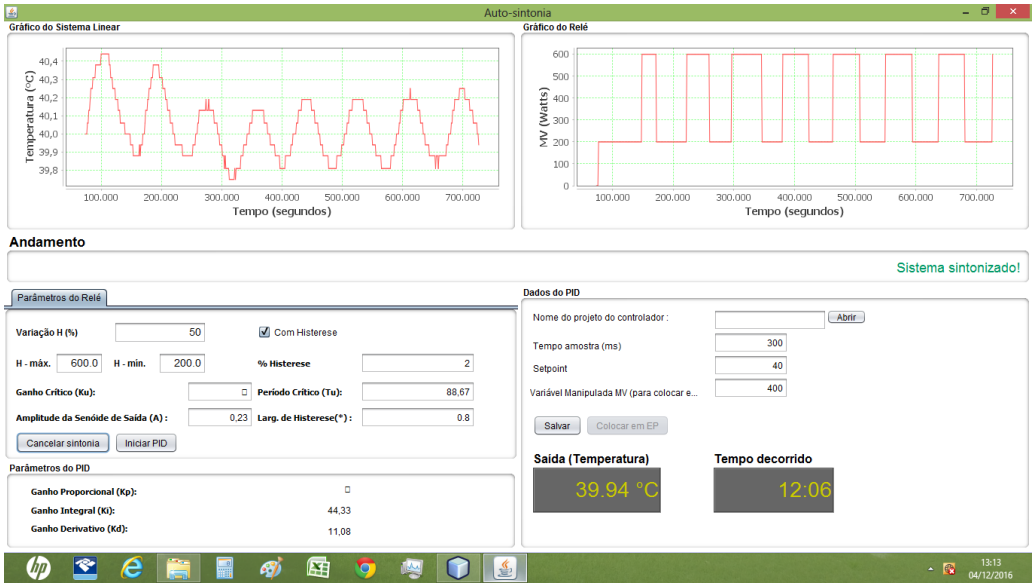
```
        Ponto pico5 = picos.get(qtdePicos-3);
        Ponto pico4 = picos.get(qtdePicos-4);

        float mediaPicos = (pico7.getValor() + pico6.getValor() + pico5.getValor() + pico4.getValor()) / 4;

        double v1 = Math.pow( ((pico7.getValor() - mediaPicos)) , 2); // (valor lido 7 – media)^2
        double v2 = Math.pow( ((pico6.getValor() - mediaPicos)) , 2); // (valor lido 6 – media)^2
        double v3 = Math.pow( ((pico5.getValor() - mediaPicos)) , 2); // (valor lido 5 – media)^2
        double v4 = Math.pow( ((pico4.getValor() - mediaPicos)) , 2); // (valor lido 4 – media)^2
        double variancia = (v1 + v2 + v3 + v4) / 4;
        double desvioPadrao = Math.sqrt(variancia);

        return desvioPadrao;
    }
```

7) Em ambos os casos após o final da coleta de dados é realizado os cálculos dos parâmetros do controlador são eles: K_u, T_u, K_p, K_d e K_i, isto foi implementado no método com código em linguagem Java mostrado abaixo:

```
private void gerarParametrosPID() {

    /* pega os 4 ultimos picos*/
    Ponto pico7 = picos.get(qtdePicos-1);
    Ponto pico6 = picos.get(qtdePicos-2);
    Ponto pico5 = picos.get(qtdePicos-3);


    float mediaPicos = (pico7.getValor() + pico6.getValor() + pico5.getValor()) / 3;
    A = mediaPicos - Planta.setPoint;
    // calcula a média
    Tu = ( ( pico7.getTempo() - pico6.getTempo() ) + ( pico6.getTempo() - pico5.getTempo() ) ) / 2;

    Tu = Tu / 1000; // transfroma de milisegundos para segundos
    float d = (hmax - hmin) / 2;
```

```
// calculo do ganho critico (Ku)
    if (tipoRele == 0) // sem histerese
    {
       Ku = (4 * d) / (Math.PI * A);
    } else { // com histerese
       Ku = (4 * d) / ( Math.PI * Math.sqrt( Math.pow(A, 2) - Math.pow(larguraHisterese, 2) ) );
    }


    // calculo dos parametros do controlador PID
    Kp = 0.6 * Ku;
    Ki = 0.5 * Tu;
    Kd = 0.125 * Tu;

}
```

Figura 4-4: Interface do Sistema Java Swing para usuário.



Fonte: Elaborado pelo próprio autor.

4.3.2 - Resultados da Sintonia ao Utilizar a Ferramenta

Um dos objetivos da ferramenta é auxiliar o projetista do controlador no processo de sintonia do mesmo, usando o método do relé. Algumas execuções foram realizadas e apresentaram resultados nem sempre esperados, mas que permitem auxiliar na escolha correta dos parâmetros no processo de sintonia do controlador. Alguns pontos de escolha de

parâmetros são essenciais para se realizar a sintonia do controlador, assim, alguns desses pontos de seleção serão descritos a seguir, baseado nas observações e experiências adquiridas durante a construção deste trabalho.

4.3.2.1 - Parâmetros que Influenciam o Resultado no Processo de Sintonia

Existem três parâmetros que influenciam diretamente nos resultados de sintonia do controlador: o tempo de amostra, o valor central da amplitude do relé e a variação da amplitude do relé. Esses são descritos abaixo:

a) Tempo de amostra

   O tempo de amostragem tem bastante influência e é um parâmetro configurável, já que dependendo do computador sendo utilizado o tempo de amotra deve aumentar ou diminuir. Se o computador for lento a tendência é que esse tempo seja maior, ao passo que se o computador for mais rápido, o tempo de amostragem será menor.

b) Valor central da amplitude do relé

   Este valor influencia na amplitude e pode permitir gerar ou não senoidal durante o processo de sintonia, pois como sabemos que isso é indispensável. Esse valor equivale a variável de manipulada que correponde ao valor que será aplicado ao atuador, que no caso deste trabalho é a potência aplicada à resistência térmica.

c) Variação da amplitude do relé

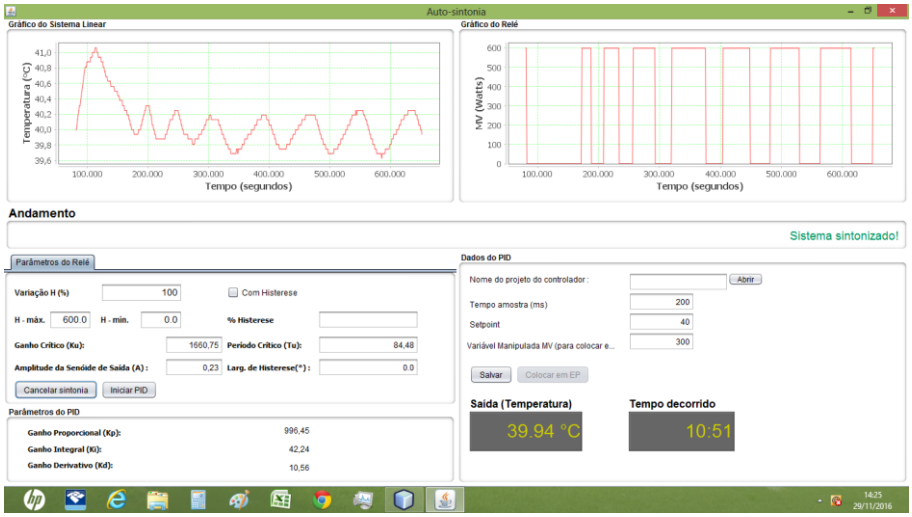
   Esse parâmetro foram um dos pontos mais críticos nos primeiros momentos em que iniciou-se a utilização da ferramenta, pois o método exige que façamos uma escolha do valor da variável manipulada, esse valor exige que tenhamos algum conhecimento do sistema. No caso deste sistema, a variável manipulada que equivale ao valor da potência no atuador, no caso a resistência elétrica de aquecimento terá valor máximo de 1000 Watts de potência. Assim, desejamos uma amplitude de 500 W, devemos selecionar o valor central de 500 W e uma variação percentual de 100%, assim, os valores de MV irão de 0 a 1000 W. Um valor menor também poderá ser selecionado, mas sempre pensando no valor central. Na figura IV-4 é mostrada essa situação.

4.3.2.2 – Resultados Comparação do Método do Relé Ideal e Histerese

Como pode-se comparar nas figuras IV-4 e IV-5 os resultados da sincronia para temperaturas ambas de 40º C para o valor do setpoint, tempos de amostragem diferentes, observe que na figura IV-5 a resolução do gráfico é melhor, mas ambos descrevem saídas senoidais como esperado. Nesse caso também o valor da variável manipulada é diferente.

Observa-se que o resultado é uma senóide com valor de saída com picos aproximadamente simétricos a partir do quarto pico, como já foi previsto na revisão bibliográfica para este tipo de método.

Figura 4-5: Sintonia usando o relé ideal.

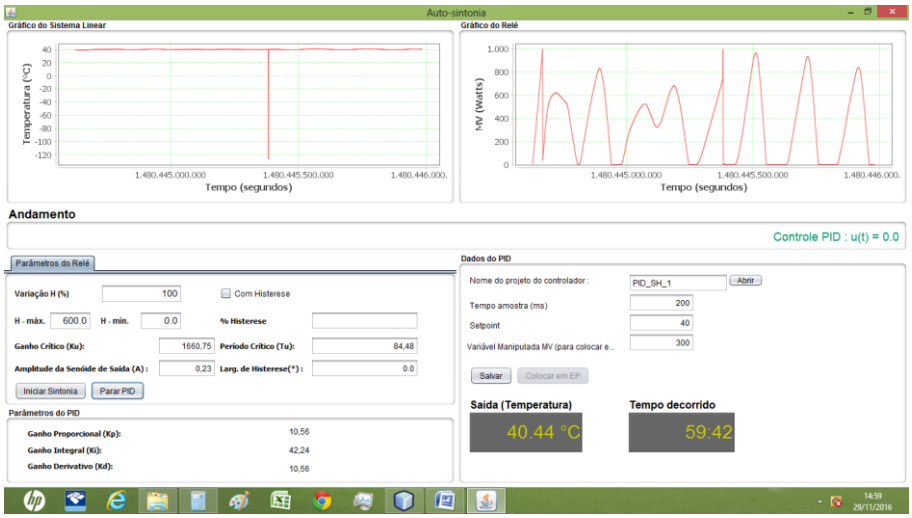


Fonte: Elaborado pelo próprio autor.

Mas, observa-se também que apesar de estabilizar a partir do quarto pico, ainda há alguns valores de pico de ruído, assim, como também é previsto na teoria acerca desse método isso ocorre, pois este método é muito sensível a ruídos. Assim, esse filtro do ruído será melhorado com o método do relé histerese. Então, a ferramenta permite evidenciar esses detalhes que são previsto no seu modelo teórico. Observa-se que o tempo decorrido de sintonia está na casa de 10 a 12 minutos.

Uma vez que o sistema foi sintonizado, é possível ainda dentro da ferramenta, realizar o controle do sistema, como mostrado o gráfico IV-6 onde é mostrado o gráfico desde o início até o momento de estabilização. Nesse caso erro em relação a temperatura de referencia foi de +/- 1º C.

Figura 4-6: Execução do PID após sintonia.

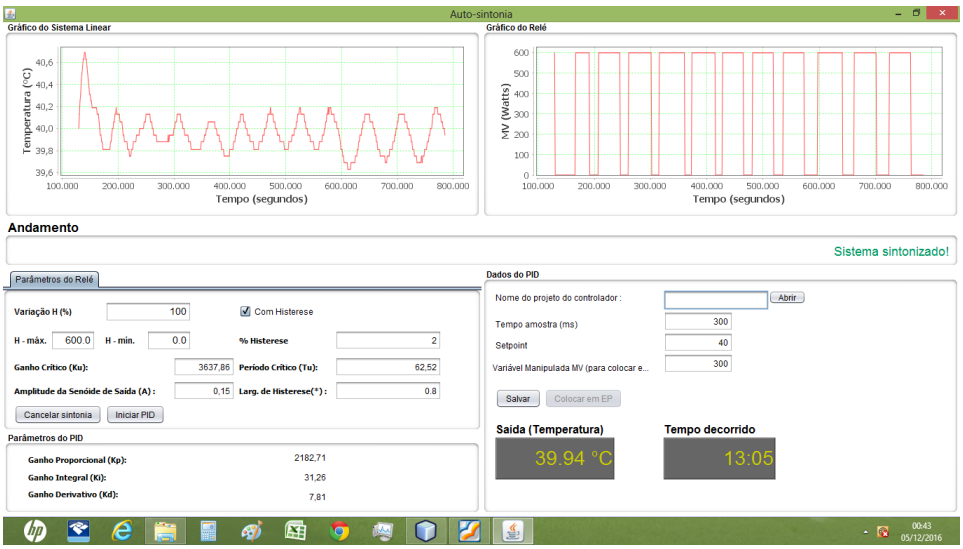


Fonte: Elaborado pelo próprio autor.

Observe que o gráfico da direita mostra a variável de manipulada sobre grande variação, ou seja, o atuador muda quase como uma senóide para manter a temperatura estável. Observe que isso depois de decorridos quase 1 minuto.

Agora na figura IV-7 é mostrado a sincronia de um controlador com histerese. Nesse caso observa-se que o número de ciclos é bem maior, então são descartados os três primeiros picos, depois a partir do quarto pico até o sétimo pico é calculado o desvio padrão. O tempo de processamento aumentou cerca de 2 minutos em relação ao relé ideal.

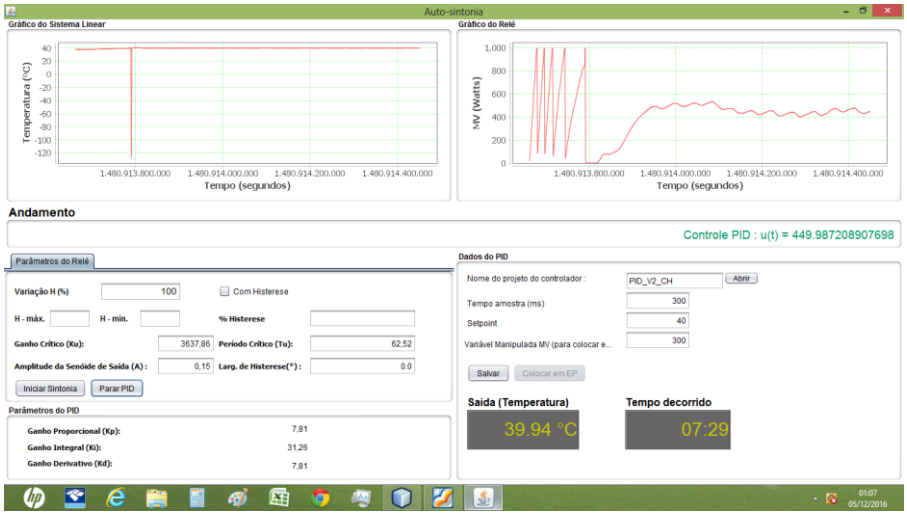
Figura 4-7: Execução do método relé com histerese.



Fonte: Elaborado pelo próprio autor.

Após a sintonia do controlador pelo método histerese, a execução do controlador através do sistema resultou no seguinte gráfico da figura IV-8.

Figura 4-8: Resultado do controlador com os parâmetros inseridos resultantes da sintonia.



Fonte: Elaborado pelo próprio autor.

Neste caso, a temperatura teve variação máxima de 0,15º C, bem abaixo da variação de quando o controlador foi sintonizado pelo método do relé ideal. Isso já era de se esperar, apenas o tempo de sintonia pelo relé histerese foi um pouco maior. Além disso, as variações na variável manipulada foi também foi bem menos variante que a do relé ideal.

CAPÍTULO 5

CONCLUSÕES

A ferramenta desenvolvida foi elaborada baseada no experimento da seção 4.1 ilustrado na figura IV-1, apesar de estar embasada neste experimento, o sistema poderia ser facilmente adapatada para outro tipo de aplicação, uma vez que os parâmetros são os definidos formalmente pela engenharia de controle. Esses parâmetros são, por exemplo, setpoint, variável de controle, variável de processo, erro e variável manipulada que serão sempre presentes em qualquer seja o sistema controlado.

Também não houve precocupação em modelar o sistema para comparar o resultado simulado com o resultado real, por dois motivos: o primeiro é que o propósito do método de sintonia do relé é exatamente não necessitar de modelagem matemática, tendo em vista que fazer a modelagem matemática poderia ser custoso e demorado. Em segundo lugar o próprio resultado da sintonia já mostra que houve sucesso nos resultados, pois percebemos o sistema comportando-se de maneira controlada.

O sistema poderia facilmente incluir mais métodos de sintonia, para isso, bastaria criar novas classes que implementassem o método adicionado e disponibilizar uma aba com os parâmetros de entrada para o usuário informar. Certamente, que isso não é uma tarefa tão trivial, mas a ferramenta tem bastante flexibilidade neste sentido.

Outro fato importante a ser acrescido é que o método do relé, não se aplica a toda a gama de sistema de controle, mas poderia ser aplicado a uma grande maioria de sistemas industriais que é o foco deste trabalho. Até mesmo, a ferramenta pode servir de base de conhecimento para a elaboração de ferramentas mais sofisticadas. A ideia desta dissertação é mostrar a implementação de uma metodologia de sintonia bastante difundida, de maneira a permitir que a comunidade científica e não científica possa vir a observar a teoria sobre o assunto e então poder visualizar a sua implementação na prática.

Também foi objetivo do trabalho a implementação do controlador PID com materiais de custo baixo e fácil acesso, como, por exemplo, o uso do arduino que é um microcontrolador de baixo custo e de fácil acesso. Isto então possibilita aos meios acadêmicos e aqueles que pretendem desenvolver controladores e ainda são iniciantes na área, ter acesso à prática de construção dos mesmos, pois é comum o desenvolvimento de controladores utilizando-se

equipamentos de grande porte e na grande maioria das vezes de custo muito alto, além de que o projeto do controlador é realizado por empresas especializadas e cuja metologia de implementação não é aberta a comunidade técnica.

Por fim, conclui-se que os resultados foram satisfatórios e os objetivos traçados para o desenvolvimento desta dissertação foram alcançados.

BIBLIOGRAFIA

AG, S. White Paper. **SIMATIC PCS 7 APC-Portfolio**, 2008.

ALMEIDA, J. L. A. **Eletrônica de Potência**. São Paulo: Editora Érica, 1986.

ALMEIDA, J. L. A. **Eletrônica Industrial**. São Paulo: Editora Érica, 1991.

ALMEIDA, J. L. A. **Dispositivos Semicondutores:** Tiristores. São Paulo: Editora Érica, 1996.

ARDUINO Playground - Java. Disponivel em: <http://playground.arduino.cc/Interfacing/Java>. Acesso em: 25 outubro 2015.

ARRUDA, L. et al. Um método evolucionário para sintonia de controladores pi/pid em processos multivariáveis. **Revista Controle e Automação**, p. 19(1), 1–16, 2008.

ASTRÖM, K. J.; HÄGGLUND, T. Automatic tuning of simple regulators with specifications on phase and amplitude margins, Automatica, p. 20(5), 645–651, 1984.

ASTROM, K. J.; HAGGLUND, T. PID Controllers. In: _____ **Theory, Design, and Tuning.International**. [S.l.]: [s.n.], 1995.

ASTRÖM, K. J.; HÄGGLUND, T. PID Controllers: Theory, Design, and Tunning. **Instrument Society of America**, Durham, NC, EUA, n. 2a, 1995.

ASTRÖM, K. J.; HÄGGLUND, T. The future of pid control. In: _____ **Control Engineering**. [S.l.]: Practice, 2001. p. 9(11), 1163–1175.

ASTRÖM, K. J.; HÄGGLUND, T. Revisiting the ziegler-nichols step response. **Journal of process control**, 2004. 14(6), 635–650.

ASTROM, K. J.; HAGGLUND, T. **Advanced PID Control. ISA**. [S.l.]: [s.n.], 2005.

ASTROM, K. J.; HAGGLUND, T. Advanced PID Control, Instrument Society of America, Durham, NC, EUA, 2006.

ASTROM, K. J.; HAGGLUND, T. H. **New Tuning Methods for PID controllers**. Proc. 3ª European Control Conference. [S.l.]: [s.1995. p. p.2456-62.

ASTRÖM, K. J.; WITTENMARK, B. Automatic tuning of PID Controllers, ISA, 1988.

BARBI, I. **Eletrônica de Potência**: Editora da UFSC, 1986.

BERGER, H. **Automating with Simatic**: Siemens. 2006.

CAMPOS, M. C. M. M. D.; TEIXEIRA, H. C. G. **Controles típicos de equipamentos e processos industriais**. São Paulo: Editora Edgard Blücher, 2006.

CARDOSO, P. M. **Estudo, proposta e avaliação de novas metodologias de sintonia automática de controladores pid baseadas no ensaio do relê realimentado, automática de controladores pid baseadas no ensaio do relê realimentado, Dissertação de mestrado**. Universidade Federal de Uberlândia, Uberlândia: [s.n.], 2002.

CATE – Centro de Aplicações de Tecnologias Eficientes : Guia Operacional de Acionamentos Eletrônicos, Versão 1.0, PROCEL, CEPEL, ELETROBRAS. [S.l.]. 1998.

CHENG, C. Y. **Autotuning of PID controllers:** a Relay Feedback Approach. 2nd. ed. [S.l.]: [s.n.], 2006.

CHIEN, K. L.; HRONES, A.; RESWICK, J. B. On the automatic control of generalized passive systems. In: _____ **Transactions ASME**. [S.l.]: [s.n.], 1952. p. 74, 175–185.

COELHO, A. A. R.; COELHO, L. D. S. Identificação de Sistemas Dinâmicos Lineares. Santa Catarina, SC: Editora da UFSC, 2004.

COHEN, G. H.; COON, G. A. Theoretical considerations of retarded control. In: _____ **Transactions ASME**. [S.l.]: [s.n.], 1953. p. 75, 827–837.

COLOGNI, M. A. **Estudo e avaliação de metodologias de auto-sintonia de controladores pid visando uma implementação em controlador industrial, Dissertação de mestrado**. Universidade Federal de Santa Catarina. [S.l.]. 2008.

COMUNICAÇÃO Serial Java Arduino - Embarcados. Disponivel em: <http://www.embarcados.com.br/comunicacao-serial-java-arduino>. Acesso em: 23 novembro 2015.

CONG, S.; LIANG, Y. Pid-like neural network nonlinear adaptive control for uncertain multivariable motion control systems, IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS. [S.l.]: [s.n.], 2009. p. 56, 3872–3879.

CONTROL an Arduino from Java _ Dr Dobb's. Disponivel em: <http://www.drdobbs.com/jvm/control-an-arduino-from-java/240163864>. Acesso em: 23 novembro 2015.

CORRIPIO, A. B. Tuning of Industrial Control Systems, Instrumentation Systems, NC, USA, 1996.

DILLENBURG, M. R.; FERRAZ, G. M. Implementação de Controle PID Auto-Adaptativo utilizando o mínimo de recursos em microcontroladores de baixo custo. **4º Congresso Inst. de Automação, Sistemas e Instrumentação**, São Paulo, 2004.

ENDER, D. B. Process control performance: not as good as you think, Control Engineering, 1993.

FLORES T., A. Relay Feedback Auto Tuning of PID Controllers. Mexico: Iberoamericana, Universidad, 2007.

FRANKLIN, G. F.; POWELL, J. D.; EMAMI-NAEINI, A. **Feedback Control of Dynamic Systems**. 4ª. ed. [S.l.]: Prentice Hall, 2002.

FRIMAN, M.; WALLER, K. V. A two-channel relay for autotuning, Ind. Eng. Chem Res. 36, 2662–2671, 1997.

GARCIA, C. E.; MORARI, M. Internal model control - 1, Unifying Review and Some New Results, p. 21, 308–323, 1982.

GRÉGROIRE, M.; DESBIENS, A.; RICHARD, É. **Development of an Auto-tuning PID and Applications to the Pulp and Paper Industry**. In Third International Conference on Industrial Automation. Montréal, Canada: [s.n.]. 1999. p. pages 3.5-3.8.

GUDE, J. J.; KAHORAHO, E.; ETXANIZ, J. Practical aspects of pid controllers: An industrial experience, em ‘IEEE Conference on Emerging Technologies and Factory Automation , ETFA’06’, p. pp. 870–878, 2006.

GYONGY, I. J.; CLARKE, D. W. "On the automatic tuning and adaptation of pid controllers", Control Engineering Practice , p. 4, 364–380, 2005.

HANG, C. C.; ASTRÖM, K. J.; WANG, Q. G. Relay feedback auto-tuning of process controllers – a tutorial review. **Journal of process control**, 2002. 12(1), 143–162.

HANG, C. C.; LEE, T. H.; HO, W. K. Adaptive Control, Instrument Society of America, 1993.

HO, W. K.; HANG, C. C.; CAO, L. S. Tuning of pi controllers based on gain and phase margin specifications, em "Industrial Electronic , 1992., Proceedings of the IEEE International Symposium on", Singapore, p. pp. 879–882 vol.2, 1992.

JENG, J. C.; HUANG, H. P. "Performance assessment and controller design for unknown systems based on gain and phase margins using modified relay feedback", p. pp. 1501–1507, 2006.

JOHNSON, M. A.; MORADI, M. PID Control - New Identification and Design Methods, Springer-Verlag, London, 2005.

LEE, Y. I. **Development of an auto-tuning scheme for discrete PID controller**. ISL Winter workshop. Korea, Seoul: [s.n.]. 1988.

LEVA, A.; COX, C.; RUANO, A. Hands-on PID autotuning: a guide to better utilization. **IFAC Professional Brief**, 2003.

LEVINE, W. S. The Control Handbook, CRC Press , Boca Raton, FL, 1996.

LJUNG, L. System Identification: Theory for User, Prentice-Hall, Upper Saddle River, NJ, 1999.

LOPEZ, A. M. et al. "Tuning controllers with error-integral criteria", Instrumentation Technology, p. 14, 57–62, 1967.

LOPEZ, A. M.; SMITH, C. L.; MURRILL, P. W. Tuning pi and pid digital controllers, Instrumentation Technology, p. 42, 89–95, 1969.

LUO, R.; QIN, S. J.; CHEN, D. "A new approach to closed loop autotuning for pidcontrollers", Ind. Eng. Chem. Res , p. 37, 2462–2468, 1998.

LUYBEN, W. Derivation of transfer functions for highly nonlinear distillation columns. Ind. Eng. Chem. Res., 1987.

LUYBEN, W. Getting more information from relay feedback tests. Ind. Chem. Res., 2001.

LUYBEN, W. L. "'Derivation of transfer functions for highly nonlinear distillation columns", Ind. Eng. Chem. Res, p. 26, 2490, 1987.

MAJHI, S.; LITZ, L. "On-line tuning of pid controllers", Proceedings of the American Control Conference, p. 6, 5003–5004, 2003.

MELLO, L. F. P. **Análise e Projeto de Fontes Chaveadas**. São Paulo: Editora Érica, 1996.

MING, D. M.; XIN, J. Z. "'Performance assessment and controller design based on modified relay feedback", Ind. Eng. Chem. Res., p. 44, 3538–3546, 2005.

MINORSKY. Directional stability of automatically steered bodies. In: _____ **J. Am. Soc. Naval Eng.** [S.l.]: [s.n.], 1922. p. 284.

MUHIDIN, L. PID Controllers in Nineties. Corning Incorporated, Science and Technology Division. New York: [s.n.], 1999.

MULLER, J. **Controlling with Simatic**. Siemens. [S.l.]. 2005.

MURRAY, R. M. et al. "'Future directions in control in an information-rich world", IEEE Control Systems Magazine, p. 23(1), 20–33, 2003.

NI. **"Ziegler-nichols autotuning method"**, 2012. Disponivel em: <http://zone.ni.com/reference/en-XX/help/370401J-01/lvpidmain/ziegnichforms/>.

NOVAIS, J. **Programação de Autómatos, 3ª Ed**. Fund. Calouste Gulbenkian. [S.l.]. 2004.

O'DWYER, A. Pi and pid controllers for time delay processes: a sum-mary, em "Technical Report of Dublin Institute of Technology", Dublin, Ireland., 2000. Disponivel em: <http://www.docsee.kst.ie/aodweb/.>.

O'DWYER, A. PI and PID Controller Tuning Rules, Imperial College Press, Covent Garden, London, 2006.

OGATA, K. **Engenharia de Controle Moderno**. Rio de Janeiro: Pearson Prentice Hall do Brasil, 1985.

OGATA, K. **Engenharia de Controle Moderno**. 5a. ed. São Paulo: Pearson Prentice Hall, 2010.

OVIEDO, J. J.; BOELEN, T.; OVERSCHEE, P. V. "Robust advanced pid control (rapid):Pid tuning based on engineering specifications", IEEE Control Systems Magazine. [S.l.]: [s.n.], 2006. p. 26(1), 15–19.

PETER, J. M. **Characteristics of Power Semiconductors, Application Notes - ST Microelectronics**. [S.l.]: [s.n.], 1999.

POWER electronics. **Interactive Power Electronics on Line Text**, 2016. Disponivel em: <www.utexas.edu/world/lecture –.>. Acesso em: 09 setembro outubro.

RASHID, M. H. **Power Electronics:** Circuits, Devices and Applications. [S.l.]: Prentice Hall International Edition, 1988.

RIVERA, D. E.; MORARI, M.; SKOGESTAD, S. "Internal model control,4", PID Controller Design 4', PID Controller Design. [S.l.]: [s.n.], 1986. p. 25(1), 252–265.

ROVIRA, A.; MURRIL, P. W.; SMITH, C. L. "Tuning controllers for setpoint changes",Instrumentation and Control Systems. [S.l.]: [s.n.], 1969. p. 42, 67–69.

RUBAAI, A.; SITIRICHE, M. J. C.; OFOLI, A. R. "Design and implementation of parallel fuzzy pid controller for high-performance brushless motor drives: An integrated en-vironment for rapid control prototyping", IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS. [S.l.]: [s.n.], 2008. p. 44, 1090–1098.

SKOGESTAD, S. "Simple analytic rules for model reduction and pid controller tuning", Modeling, Identification and Control. [S.l.]: [s.n.], 2004. p. 25(2), 85–120.

TAN, K. K.; LEE, T. H.; JIANG, X. "Robust on-line relay automatic tuning of pid control systems", ISA Transactions. [S.l.]: [s.n.], 2000. p. 39, 219–232.

TAN, K. K.; LEE, T. H.; JIANG, X. "On-line relay identification, assessment and tuning of pid controller", Jornal of Process Control. [S.l.]: [s.n.], 2001. p. 11(1), 483–496.

TEMPERATUUR meten de Arduino en een DS18B20, 2014. Disponivel em: <http://www.tweaking4All.nl/hardware/arduino/arduino-ds18b20-temperatuur-sensor/>. Acesso em: 13 Novembro 2016.

TRIAC com arduino. Disponivel em: <http://automatobr.blogspot.com/2013/05/controle-de-potencia-em-corrente_18.html>. Acesso em: 09 novembro 2016.

VANDORE, V. Auto-tuning control using ziegler nichols, em "9th IEEE/IAS International Conference on Industry Applications – INDUSCON". From de pages of Control Enginnering, 2006. Disponivel em: <http://www.controleng.com/article/CA6378136.html>.

VITECEK, A.; VÍTECKOVA, M. Plant Identification by Relay Method, En-gineering the Future, Laszlo Dudas, Rijeka, Croatia., 2010.

WANG, L.; DESARMO, M. L.; CLUETT, W. R. "‘Real-time estimation of process frequency response and step response from relay feedback experiments", Automatica. [S.l.]: [s.n.], 1999. p. 35, 1427–1436.

WANG, Q.-G.; HANG, C.-C.; ZOU, B. "Low-order modeling fromrelay feedback", Ind. Eng. Chem. Res. [S.l.]: [s.n.], 1997. p. 36, 375–381.

YU, C.-C. **Autotuning of PID Controllers:** A Relay Feedback Approach. 2. ed. London: Springer, 2006.

ZIEGLER, J. B.; NICHOLS, N. B. "Optimum settings for automatic controls", Transactions ASME. [S.l.]: [s.n.], 1942. p. 64, 759–768.

ZIEGLER, J. B.; NICHOLS, N. B. "‘Process lags in automatic control circuits", Transactions ASME. [S.l.]: [s.n.], 1943. p. 65, 433–444.

ZIEGLER, J. G.; NICHOLS, N. B. Optimum Settings for Automatic Controllers. In: _____ **Trans. ASME**. [S.l.]: [s.n.], 1942; 70. p. pp. 759-768.

ANEXOS

ANEXO I - Código fonte do programa a ser embarcado no arduino UNO R3 para controle e aquisição de dados do sensor de temperatura.

```
#include <OneWire.h>
#include <DallasTemperature.h>
/*
 created 12 Feb 2016
 By Mario Jorge Maciel
 modified 16 Abr 2016
 By Mario Jorge Maciel
 modified 05 Nov 2016
 By Mario Jorge Maciel
 */
// Data wire is plugged into port 7 on the Arduino
#define ONE_WIRE_BUS 7
#define VENTILATOR 4
#define TEMPERATURE_PRECISION 12
// Setup a oneWire instance to communicate with any OneWire devices (not just Maxim/Dallas temperature ICs)
OneWire oneWire(ONE_WIRE_BUS);

// Pass our oneWire reference to Dallas Temperature.
DallasTemperature sensors(&oneWire);

DeviceAddress tempDeviceAddress; // We'll use this variable to store a found device address// variables:
int status = 0; // sem ação
float saida = 0;// temperatura atual - Varíavel de controle
String inputString = "";         // a string to hold incoming data
boolean stringComplete = false;
boolean ventilatorOn;
float temperaturaDesligaVetilator;
// parametros recebidos da interface
float tempoAmostra = 50; // inicializa com 50 ms
float setpoint;
void setup() {
```

```
  Serial.begin(9600);//frequência da porta serial
  // configura pinos do sensor e válvula solenoide
  inputString.reserve(200);
  // Start up the library
  sensors.begin();
  inputString.reserve(200);
  // Grab a count of devices on the wire
  int numberOfDevices = sensors.getDeviceCount();
  if (numberOfDevices == 1)
   {
    if(sensors.getAddress(tempDeviceAddress, 0))
     {
       sensors.setResolution(tempDeviceAddress, TEMPERATURE_PRECISION);
     }
   }
   pinMode(VENTILATOR, OUTPUT);
   ventilatorOn = false;
}

void loop() {
  // print the string when a newline arrives:
  if ( stringComplete) {
    readCommands(inputString);
    // clear the string:
    inputString = "";
    stringComplete = false;
  }
  if (status == 1) {
    sensors.requestTemperatures(); // Send the command to get temperatures
    float tempC = sensors.getTempC(tempDeviceAddress);
    delay(20);
    Serial.println((String)tempC);
    delay(50);
    if ( tempC > setpoint ) { // então liga motor
      if (!ventilatorOn) { // se estiver desligado então liga
        digitalWrite(VENTILATOR, HIGH);
```

```
        ventilatorOn = true;
      }
    } else { // desliga
      if (tempC <= temperaturaDesligaVetilator) {
        if (ventilatorOn) { // se estiver ligado então desliga
          digitalWrite(VENTILATOR, LOW);
          ventilatorOn = false;
        }
      }
    }
    delay(tempoAmostra);
  }
}
// read commands
void readCommands(String stringCommands) {
  boolean hasCommand = true;
  int startIndex = 0;
  int indexNow = 0;
  String command = "";
  //teste = "";
  while (hasCommand) {
     indexNow = stringCommands.indexOf(';', startIndex);
      if (indexNow > 0) {
        command = stringCommands.substring(startIndex, indexNow);
        //teste += command + "\n";
        startIndex = indexNow + 1;
    if (command.startsWith("SP#")) { // pega o setpoint
          setpoint = command.substring(3).toFloat();
          //Serial.println(inputString.substring(3));
        }
        if (command.startsWith("TD#")) { // pega a temperatura minima de desligamento
          temperaturaDesligaVetilator = command.substring(3).toFloat();
          //Serial.println(inputString.substring(3));
        }
        if (command.startsWith("PA#")) { // periodo de amostragem
          tempoAmostra = command.substring(3).toInt();
```

```
        //Serial.println(inputString.substring(3));
      }
      if (command.startsWith("LV#")) { // ligar ventilação
        if (!ventilatorOn) { // se estiver desligado então liga
          digitalWrite(VENTILATOR, HIGH);
          ventilatorOn = true;
        }
      }
      if (command.startsWith("DV#")) { // desligar ventilação
        if (ventilatorOn) { // se estiver ligado então desliga
          digitalWrite(VENTILATOR, LOW);
          ventilatorOn = false;
        }
      }
      if (command.startsWith("PE#")) { // pausara experimentacao
        status = 0;
        digitalWrite(VENTILATOR, LOW);
        ventilatorOn = false;
      }
      if (command.startsWith("IE#")) { // iniciar experimentacao
        status = 1;
      }
     } else {
      hasCommand = false;
     }
  }
}// recebe os comandos da porta serial, cada comando finaliza por \n
void serialEvent() {
  while (Serial.available()) {
    // get the new byte:
    char inChar = (char)Serial.read();
    // if the incoming character is a newline, set a flag
    // so the main loop can do something about it:
    if (inChar == '\n') {
      stringComplete = true;
    } else {
```

```
      // add it to the inputString:
      inputString += inChar;
    }
  }
}
```

ANEXO II - **Código fonte do programa a ser embarcado no arduino NANO para o controle de atuação na resistência elétrica de aquecimento.**

```
#include <OneWire.h>
#include <DallasTemperature.h>
/*
 created 12 Feb 2016
 By Mario Jorge Maciel
 modified 16 Abr 2016
 By Mario Jorge Maciel
 modified 15 Nov 2016
 By Mario Jorge Maciel
 */

// Data wire is plugged into port 7 on the Arduino
#define AQUECEDOR 6
// variables:
float potenciaMax = 2000; // potencia máxima
float powerPulse = 0; // potencia máxima
int status = 0; // sem ação
float saida = 0;// temperatura atual - Varíavel de controle
String inputString = "";         // a string to hold incoming data
boolean stringComplete = false;

// parametros recebidos da interface
volatile int power = 0; // é o ponto de operação - MV
volatile boolean detectouZero = false;
void zero_cross_init() {
    detectouZero = true;
}

void setup() {
  Serial.begin(9600);//frequência da porta serial
  // configura pinos do sensor e válvula solenoide
  inputString.reserve(200);
```

```
  // configura pino de controle de aquecimento
  powerPulse = (8333.33 - 8.33) / potenciaMax;
  pinMode(AQUECEDOR, OUTPUT);
  attachInterrupt(1, zero_cross_init, RISING);
}

void loop() {
  //noInterrupts();
  if (status == 1)
  {
    if (detectouZero) {
      int powertime = (powerPulse * (potenciaMax - power));
      delayMicroseconds(powertime);
      // liga o pulso
      digitalWrite(AQUECEDOR, HIGH);
      delayMicroseconds(8.33);
      // Desliga o pulso
      digitalWrite(AQUECEDOR, LOW);
      detectouZero = false;
    }
  }

  noInterrupts();

  if (stringComplete) {
      stringComplete = false;
      if (inputString.startsWith("#")) {
        status = 0;
      }
      if (inputString.startsWith("I")) {
        power = inputString.substring(1).toInt(); // valor PWM calculado no programa java
        status = 1;
      }
      inputString = "";
  }
```

```
  interrupts();
}

// read commands
void readCommands(String stringCommands) {

  boolean hasCommand = true;
  int startIndex = 0;
  int indexNow = 0;
  String command = "";
  while (hasCommand) {

     indexNow = stringCommands.indexOf(';', startIndex);
     if (indexNow > 0) {
       command = stringCommands.substring(startIndex, indexNow);
       startIndex = indexNow + 1;

      if (command.startsWith("MV#")) { // seta valor do atuador
         power = command.substring(3).toInt();
      }

      if (command.startsWith("PE#")) { // pausara experimentacao
         status = 0;
      }
      if (command.startsWith("IE#")) { // iniciar experimentacao
         status = 1;
      }

     } else {
       hasCommand = false;
     }
  }
}
```

```
// recebe os comandos da porta serial, cada comando finaliza por \n
void serialEvent() {
  //noInterrupts();
  while (Serial.available()) {
    // get the new byte:
    char inChar = (char)Serial.read();
    // if the incoming character is a newline, set a flag
    // so the main loop can do something about it:
    if (inChar == '\n') {
      stringComplete = true;
    } else {
      // add it to the inputString:
      inputString += inChar;

    }
  }
 }
```

ANEXO III - Código fonte da classe Java que executa o método do relé.

```
package controlador;

import autotunningapp.Arduino;
import autotunningapp.MainFrame;
import autotunningapp.Ponto;
import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author Mário Jorge da Silva Maciel
 */
public class Rele  {

    public static float erro;
    public float saida, saida_old, saida_old_negativa;
    public static float setPoint;
    public static float h, hmax, hmin;
    public static boolean ligarVentilacao;
    public static float u_t, u_t_1;
    //parametros calculados
    //public static float Tu, Ku, A;
    public static double Tu1, Tu2, Tu, Ku, A;
    public static double Kp, Ki, Kd;
    public static float MV;
    public static int qtdePicos;
    public static List<Ponto> picos;
    public static List<Ponto> pontos;
    public static boolean alteraLarguraHisterese;

    long  tempoPico;
    private boolean detectouPico;
    private boolean experimentacoes;
    private double picoNegativo; // detecta o ultimo pico negativo
```

```
private int tipoRele;
public static double larguraHisterese;
private int multiplicadorHisterese;



private static float u_t_old;

public Rele() {
    inicializa();
    this.tipoRele = 0;
    tempoPico = 0;
}
public Rele(int multiplicadorHisterese) {
    inicializa();
    experimentacoes = true;
    this.multiplicadorHisterese = multiplicadorHisterese;
    alteraLarguraHisterese = false;
    this.tipoRele = 1;
    tempoPico = 0;
}
private void inicializa() {
    // inicializa valores
    u_t = 0;
    u_t_1 = hmax;
    u_t_old = 0;
    qtdePicos = 0;
    saida_old = Planta.setPoint;
    picos = new ArrayList();
    pontos = new ArrayList();
}
public float sintonizar(long tempo, float saida) {

    erro = Planta.setPoint - saida;

    if (tipoRele == 0 || experimentacoes) { // rele ideal
        if (erro >= 0) {
```

```
            u_t = hmax;

        } else {
            u_t = hmin;
        }
    }
    if (tipoRele == 1 && !experimentacoes) { // rele com histerese
        if (erro >= larguraHisterese) {
            u_t = hmax;

        }
        if (erro < ( (-1) * larguraHisterese) ) {
            u_t = hmin;
        }
        if (erro > ((-1) * larguraHisterese) && erro < larguraHisterese) {
            u_t = u_t_1; // usa u(t-1)
        }

        u_t_1 = u_t; // u(t-1) recebe u(t)
    }
    if (detectaPico(new Ponto(tempo, saida)) > 7 && experimentacoes && tipoRele == 1) {
        double desvio = calculaDesvioPadrao();
        larguraHisterese = multiplicadorHisterese * desvio;
        experimentacoes = false;
        alteraLarguraHisterese = true;
    }
    if (detectaPico(new Ponto(tempo, saida)) > 11 && !experimentacoes && tipoRele == 1) {
        gerarParametrosPID();
        Planta.sintonizando = false;
        Planta.sintonizado = true;
    }

    if (detectaPico(new Ponto(tempo, saida)) > 7 && tipoRele == 0) {
        gerarParametrosPID();
        Planta.sintonizando = false;
        Planta.sintonizado = true;
```

```
    }
    return u_t;
}
private int detectaPico(Ponto ponto) {
    float saida = ponto.getValor();
    long tempo = ponto.getTempo();

    // so pega os ciclos positivos
    if (saida >= Planta.setPoint) {
        if (saida > saida_old) {
            saida_old = saida;
            tempoPico = tempo;
            detectouPico = true;
        }
    } else {
        if (detectouPico) {
            picos.add(new Ponto(tempoPico, saida_old));
            detectouPico = false;
            saida_old = Planta.setPoint;
            qtdePicos += 1;
        }
    }
    return qtdePicos;
}
private double calculaDesvioPadrao() {
    Ponto pico7 = picos.get(qtdePicos-1);
    Ponto pico6 = picos.get(qtdePicos-2);
    Ponto pico5 = picos.get(qtdePicos-3);
    Ponto pico4 = picos.get(qtdePicos-4);
    float mediaPicos = (pico7.getValor() + pico6.getValor() + pico5.getValor() + pico4.getValor()) / 4;

    double v1 = Math.pow( ((pico7.getValor() - mediaPicos)) , 2);
    double v2 = Math.pow( ((pico6.getValor() - mediaPicos)) , 2);
    double v3 = Math.pow( ((pico5.getValor() - mediaPicos)) , 2);
    double v4 = Math.pow( ((pico4.getValor() - mediaPicos)) , 2);
```

```
        double variancia = (v1 + v2 + v3 + v4) / 4;
        double desvioPadrao = Math.sqrt(variancia);

        return desvioPadrao;
    }

    private void gerarParametrosPID() {

        /* pega os 4 ultimos picos*/
        Ponto pico7 = picos.get(qtdePicos-1);
        Ponto pico6 = picos.get(qtdePicos-2);
        Ponto pico5 = picos.get(qtdePicos-3);


        float mediaPicos = (pico7.getValor() + pico6.getValor() + pico5.getValor()) / 3;
        A = mediaPicos - Planta.setPoint;

        Tu = ( ( pico7.getTempo() - pico6.getTempo() ) + ( pico6.getTempo() - pico5.getTempo() ) ) / 2; // tira a
media

        Tu = Tu / 1000; // transfroma de milisegundos para segundos
        float d = (hmax - hmin) / 2;

        // calculo do ganho critico (Ku)
        if (tipoRele == 0) // sem histerese
        {
            Ku = (4 * d) / (Math.PI * A);
        } else { // com histerese
            Ku = (4 * d) / ( Math.PI * Math.sqrt( Math.pow(A, 2) - Math.pow(larguraHisterese, 2) ) );
        }


        // calculo dos parametros do controlador PID
        Kp = 0.6 * Ku;
        Ki = 0.5 * Tu;
        Kd = 0.125 * Tu;
```

```
    }
    private int ocorrenciaAmplitude(float Ai) {
        int i = 0;
        float A;
        // busca os valores que mais ocorreram
        for (Ponto pico : picos) {
            A = pico.getValor() - Planta.setPoint;
            //result = Math.max(A, Ai) - Math.min(A, Ai); // diminui o maior valor do menor
            //if ( result <= 0.1 ) { // considera igual até 0.1
            if (A == Ai) {
                i++;
            }
        }
        return i;
    }
    private int ocorrenciaPeriodo(float Tui) {
        int i = 0, count=0;
        float Tu, tempoPrev, result;
        // busca os valores que mais ocorreram
        for (Ponto pico : picos) {
            tempoPrev = picos.get(count).getTempo(); // pega o ponto anterior
            Tu = pico.getTempo() - tempoPrev;
            if ( Tu == Tui ) { // considera igual até 0.1
                i++;
            }
        }
        return i;
    }
}
```

ANEXO IV - Código fonte do programa da interface com usuário desenvolvido em Java.

```
package autotunningapp;

import Util.IOFile;
import Util.PIDSetup;
import com.sun.xml.internal.ws.message.RelatesToHeader;
import controlador.ControladorInterface;
import controlador.Planta;
import controlador.Rele;
import controlador.ServicoPID;
import gnu.io.*;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Graphics;
import java.io.*;
import java.text.DecimalFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.axis.ValueAxis;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
import org.jfree.data.xy.XYDataset;
import org.jfree.data.xy.XYSeries;
import org.jfree.data.xy.XYSeriesCollection;
import sun.org.mozilla.javascript.internal.ContextFactory;

/**
 *
 * @author Mário Jorge da Silva Maciel
```

```
 */
public class MainFrame extends javax.swing.JFrame implements SerialPortEventListener {

    static final XYSeries series1 = new XYSeries("Temperatura lida");
    static final XYSeries series2 = new XYSeries("Temperatura lida");
    XYPlot plotSistema;
    private long acumuladorTempo;
    private BufferedReader saida;
    private SerialPort portSensor;
    private SerialPort portResistencia;
    private static OutputStream serialOutSensor;
    private static OutputStream serialOutResistencia;
    private String portaCOMSensor;
    private String portaCOMResistencia;
    private int taxa;
    private int metodoSintonia;
    private int RELE_IDEAL = 1;
    private int RELE_HISTERESE = 2;
    CommPortIdentifier portId = null;
    static Rele controladorRI;
    public Planta planta = null;
    private String y = "0";
    public volatile static Fila fila;
    public volatile static boolean sistemaEmExecucao;
    private Thread thFila;
    private long horaInicio;
    final JFileChooser fc = new JFileChooser();
    ServicoPID myPID;
    private boolean pidEmExecucao;


    /**
     * Creates new form NewJFrame
     */
    public MainFrame() {
        //arduino = new ControlePorta("COM8",9600);

        initComponents();

        planta = new Planta();
```

```
    fila = new Fila();
    sistemaEmExecucao = false;
    pidEmExecucao = false;

    this.portaCOMSensor = "COM8";
    this.portaCOMResistencia = "COM29";
    this.taxa = 9600;//Integer.parseInt(taxaEdt.getText());
    inicializar();
    initialize();
}

private void inicializar() {
    //Graphics g;
    //acumuladorTempo = 0;
    // create a dataset...
    //super.paintComponents();
    XYDataset dataset1 = createDataset1();
    XYDataset dataset2 = createDataset2();
    final JFreeChart chart1 = createChart1(dataset1);
    final JFreeChart chart2 = createChart2(dataset2);
    ChartPanel myChart1 = new ChartPanel(chart1);
    ChartPanel myChart2 = new ChartPanel(chart2);

    graficoSistema.removeAll();
    graficoSistema.setLayout(new java.awt.BorderLayout());
    graficoSistema.add(myChart1, BorderLayout.CENTER);
    graficoSistema.repaint();
    graficoSistema.revalidate();

    graficoRele.removeAll();
    graficoRele.setLayout(new java.awt.BorderLayout());
    graficoRele.add(myChart2, BorderLayout.CENTER);
    graficoSistema.repaint();
    graficoRele.revalidate();

}

public static void plota(String tempo, String saida) {
    series1.add(Long.parseLong(tempo), Float.parseFloat(saida));
    graficoSistema.invalidate();
```

```
    }

    public static void plotaRele(String tempo, String mv) {
        series2.add(Long.parseLong(tempo), Float.parseFloat(mv));
        graficoRele.invalidate();
    }

    private JFreeChart createChart1(final XYDataset dataset) {

        // create the chart...
        final JFreeChart chart = ChartFactory.createXYLineChart(
                null, // chart title
                "Tempo (segundos)", // x axis label
                "Temperatura (\u00b0" + "C)", // y axis label
                dataset, // data
                PlotOrientation.VERTICAL,
                false, // include legend
                false, // tooltips
                false // urls
                );

        // NOW DO SOME OPTIONAL CUSTOMISATION OF THE CHART...
        chart.setBackgroundPaint(Color.white);


//        final StandardLegend legend = (StandardLegend) chart.getLegend();
        //      legend.setDisplaySeriesShapes(true);
        // get a reference to the plot for further customisation...
        plotSistema = chart.getXYPlot();
        plotSistema.setDomainZeroBaselinePaint(Color.green);
        NumberAxis yAxis = (NumberAxis) plotSistema.getRangeAxis();
        yAxis.setAutoRangeIncludesZero(false);

         plotSistema.setBackgroundPaint(Color.WHITE);
         plotSistema.setDomainZeroBaselinePaint(Color.black); //
         //plot.setAxisOffset(new Spacer(Spacer.ABSOLUTE, 5.0, 5.0, 5.0, 5.0));
         plotSistema.setDomainGridlinePaint(Color.green);
         plotSistema.setRangeGridlinePaint(Color.green);
         plotSistema.setRangeMinorGridlinePaint(Color.GREEN);
         plotSistema.setRangeCrosshairPaint(Color.GREEN);
```

```
     //XYLineAndShapeRenderer renderer = new XYLineAndShapeRenderer();
     // renderer.setSeriesLinesVisible(1, false);
     // renderer.setSeriesShapesVisible(1, false);
      //plot.setRenderer(renderer);


    // change the auto tick unit selection to integer units only...
    //final NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
    //rangeAxis.setStandardTickUnits(NumberAxis.createIntegerTickUnits());
    // OPTIONAL CUSTOMISATION COMPLETED.

    return chart;

}

private JFreeChart createChart2(final XYDataset dataset) {

    // create the chart...
    final JFreeChart chart = ChartFactory.createXYLineChart(
            null, // chart title
            "Tempo (segundos)", // x axis label
            "MV (Watts)", // y axis label
            dataset, // data
            PlotOrientation.VERTICAL,
            false, // include legend
            false, // tooltips
            false // urls
            );

    chart.setBackgroundPaint(Color.white);


    // NOW DO SOME OPTIONAL CUSTOMISATION OF THE CHART...

    //final StandardLegend legend = (StandardLegend) chart.getLegend();
    //      legend.setDisplaySeriesShapes(true);
    // get a reference to the plot for further customisation...
    final XYPlot plot = chart.getXYPlot();
    plot.setDomainZeroBaselinePaint(Color.green);
    NumberAxis yAxis = (NumberAxis) plot.getRangeAxis();
```

```
    yAxis.setAutoRangeIncludesZero(false);
    plot.setBackgroundPaint(Color.WHITE);
     plot.setDomainZeroBaselinePaint(Color.black); //
     //plot.setAxisOffset(new Spacer(Spacer.ABSOLUTE, 5.0, 5.0, 5.0, 5.0));
     plot.setDomainGridlinePaint(Color.green);
     plot.setRangeGridlinePaint(Color.green);
     plot.setRangeMinorGridlinePaint(Color.GREEN);
     plot.setRangeCrosshairPaint(Color.GREEN);

     /* final XYLineAndShapeRenderer renderer = new XYLineAndShapeRenderer();
     * renderer.setSeriesLinesVisible(1, false);
     * renderer.setSeriesShapesVisible(1, false);
     * plot.setRenderer(renderer);
     *
     * // change the auto tick unit selection to integer units only...
     * final NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
     * rangeAxis.setStandardTickUnits(NumberAxis.createIntegerTickUnits());
     */
    // OPTIONAL CUSTOMISATION COMPLETED.

    return chart;

}

private XYDataset createDataset1() {
    final XYSeriesCollection dataset = new XYSeriesCollection();
    dataset.addSeries(series1);
    return dataset;

}

private XYDataset createDataset2() {

    final XYSeriesCollection dataset = new XYSeriesCollection();
    dataset.addSeries(series2);
    return dataset;

}

/**
```

```
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    jMenu1 = new javax.swing.JMenu();
    MasterPainel = new javax.swing.JPanel();
    painelMain = new javax.swing.JPanel();
    painelEsquerdo = new javax.swing.JPanel();
    painelGraficos = new javax.swing.JPanel();
    graficoSistema = new javax.swing.JPanel();
    graficoRele = new javax.swing.JPanel();
    panelAndamento = new javax.swing.JPanel();
    lblAndamento = new javax.swing.JTextField();
    PainelEsquerdoMain = new javax.swing.JPanel();
    painelDireito = new javax.swing.JPanel();
    panelGrafRelePainelPID = new javax.swing.JPanel();
    PainelSelecionarSintonia = new javax.swing.JTabbedPane();
    jPanel4 = new javax.swing.JPanel();
    jLabel43 = new javax.swing.JLabel();
    txtHmax = new javax.swing.JTextField();
    jLabel44 = new javax.swing.JLabel();
    txtHmin = new javax.swing.JTextField();
    jLabel48 = new javax.swing.JLabel();
    jLabel49 = new javax.swing.JLabel();
    jLabel50 = new javax.swing.JLabel();
    btnSintoniaReleIdeal = new javax.swing.JButton();
    txtA = new javax.swing.JTextField();
    txtKu = new javax.swing.JTextField();
    txtTu = new javax.swing.JTextField();
    txtHperc = new javax.swing.JTextField();
    jLabel45 = new javax.swing.JLabel();
    btnGoReleIdeal = new javax.swing.JButton();
    chkHisterese = new javax.swing.JCheckBox();
    jLabel51 = new javax.swing.JLabel();
    txtLarguraHisterese = new javax.swing.JTextField();
    jLabel4 = new javax.swing.JLabel();
```

```
txtPercHisterese = new javax.swing.JTextField();
PainelPID = new javax.swing.JPanel();
lblKp = new javax.swing.JLabel();
lblKi = new javax.swing.JLabel();
jLabel16 = new javax.swing.JLabel();
lblKd = new javax.swing.JLabel();
jLabel17 = new javax.swing.JLabel();
jLabel19 = new javax.swing.JLabel();
painelConfiguracao = new javax.swing.JPanel();
txtNomeControlador = new javax.swing.JTextField();
jLabel1 = new javax.swing.JLabel();
jLabel3 = new javax.swing.JLabel();
txtPeriodoAmostra = new javax.swing.JTextField();
btnGravar = new javax.swing.JButton();
btnColocarEP = new javax.swing.JButton();
btnAbrirControlador = new javax.swing.JButton();
jLabel21 = new javax.swing.JLabel();
txtSetPoint = new javax.swing.JTextField();
jLabel22 = new javax.swing.JLabel();
txtMV = new javax.swing.JTextField();
painelTemperatura = new javax.swing.JPanel();
jLabel2 = new javax.swing.JLabel();
lblSaida = new javax.swing.JTextField();
jLabel5 = new javax.swing.JLabel();
lblTempoDecorrido = new javax.swing.JTextField();

jMenu1.setText("jMenu1");

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
setTitle("Auto-sintonia");
setAlwaysOnTop(true);
setBackground(new java.awt.Color(255, 255, 255));

MasterPainel.setBackground(new java.awt.Color(153, 153, 153));
MasterPainel.setFocusCycleRoot(true);
MasterPainel.setPreferredSize(new java.awt.Dimension(800, 600));
MasterPainel.setLayout(new java.awt.BorderLayout());

painelMain.setBackground(new java.awt.Color(204, 204, 255));
painelMain.setAutoscrolls(true);
```

```
        painelMain.setMinimumSize(new java.awt.Dimension(800, 600));
        painelMain.setLayout(new java.awt.GridLayout(2, 1));

        painelEsquerdo.setFocusTraversalPolicyProvider(true);
        painelEsquerdo.setLayout(new java.awt.BorderLayout());

        painelGraficos.setBackground(new java.awt.Color(255, 255, 255));
        painelGraficos.setLayout(new java.awt.GridLayout(1, 2));

        graficoSistema.setBackground(new java.awt.Color(255, 255, 255));
        graficoSistema.setBorder(javax.swing.BorderFactory.createTitledBorder(null, "Gráfico do Sistema Linear",
0, 0, new java.awt.Font("Arial", 1, 12))); // NOI18N
        graficoSistema.setMinimumSize(new java.awt.Dimension(983, 24));
        graficoSistema.setPreferredSize(new java.awt.Dimension(983, 461));
        graficoSistema.setLayout(new java.awt.BorderLayout());
        painelGraficos.add(graficoSistema);

        graficoRele.setBackground(new java.awt.Color(255, 255, 255));
        graficoRele.setBorder(javax.swing.BorderFactory.createTitledBorder(null, "Gráfico do Relé", 0, 0, new
java.awt.Font("Arial", 1, 12))); // NOI18N
        graficoRele.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());
        painelGraficos.add(graficoRele);

        painelEsquerdo.add(painelGraficos, java.awt.BorderLayout.CENTER);

        panelAndamento.setBackground(new java.awt.Color(255, 255, 255));
        panelAndamento.setLayout(new java.awt.GridLayout(1, 2));

        lblAndamento.setColumns(2);
        lblAndamento.setEditable(false);
        lblAndamento.setFont(new java.awt.Font("Arial", 0, 18)); // NOI18N
        lblAndamento.setForeground(new java.awt.Color(0, 153, 102));
        lblAndamento.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
        lblAndamento.setText("Aguardando comando...");
        lblAndamento.setBorder(javax.swing.BorderFactory.createTitledBorder(null, "Andamento", 0, 0, new
java.awt.Font("Arial", 1, 18))); // NOI18N
        lblAndamento.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                lblAndamentoActionPerformed(evt);
            }
```

```
});
panelAndamento.add(lblAndamento);

painelEsquerdo.add(panelAndamento, java.awt.BorderLayout.PAGE_END);

painelMain.add(painelEsquerdo);

PainelEsquerdoMain.setLayout(new java.awt.BorderLayout());

painelDireito.setLayout(new java.awt.GridLayout(1, 2));

panelGrafRelePainelPID.setBackground(new java.awt.Color(255, 255, 255));
panelGrafRelePainelPID.setLayout(new java.awt.BorderLayout());

PainelSelecionarSintonia.setBackground(new java.awt.Color(255, 255, 255));

jPanel4.setBackground(new java.awt.Color(255, 255, 255));
jPanel4.setBorder(javax.swing.BorderFactory.createTitledBorder(""));

jLabel43.setFont(new java.awt.Font("Arial", 1, 12)); // NOI18N
jLabel43.setText("H - máx.");

txtHmax.setEditable(false);
txtHmax.setFont(new java.awt.Font("Arial", 0, 14)); // NOI18N
txtHmax.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
txtHmax.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      txtHmaxActionPerformed(evt);
   }
});

jLabel44.setFont(new java.awt.Font("Arial", 1, 12)); // NOI18N
jLabel44.setText("H - mín.");

txtHmin.setEditable(false);
txtHmin.setFont(new java.awt.Font("Arial", 0, 14)); // NOI18N
txtHmin.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
txtHmin.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      txtHminActionPerformed(evt);
```

```
    }
});

jLabel48.setFont(new java.awt.Font("Tahoma", 1, 11)); // NOI18N
jLabel48.setText("Amplitude da Senóide de Saída (A) : ");

jLabel49.setFont(new java.awt.Font("Tahoma", 1, 11)); // NOI18N
jLabel49.setText("Ganho Crítico (Ku):");

jLabel50.setFont(new java.awt.Font("Tahoma", 1, 11)); // NOI18N
jLabel50.setText("Período Crítico (Tu):");

btnSintoniaReleIdeal.setText("Iniciar Sintonia");
btnSintoniaReleIdeal.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        btnSintoniaReleIdealActionPerformed(evt);
    }
});

txtA.setEditable(false);
txtA.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
txtA.setText("0.0");
txtA.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        txtAActionPerformed(evt);
    }
});

txtKu.setEditable(false);
txtKu.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
txtKu.setText("0.0");
txtKu.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        txtKuActionPerformed(evt);
    }
});

txtTu.setEditable(false);
txtTu.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
txtTu.setText("0.0");
```

```
txtTu.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        txtTuActionPerformed(evt);
    }
});

txtHperc.setFont(new java.awt.Font("Arial", 0, 14)); // NOI18N
txtHperc.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
txtHperc.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        txtHpercActionPerformed(evt);
    }
});

jLabel45.setFont(new java.awt.Font("Arial", 1, 12)); // NOI18N
jLabel45.setText("Variação H (%)");

btnGoReleIdeal.setText("Iniciar PID");
btnGoReleIdeal.setEnabled(false);
btnGoReleIdeal.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        btnGoReleIdealActionPerformed(evt);
    }
});

chkHisterese.setText("Com Histerese ");

jLabel51.setFont(new java.awt.Font("Tahoma", 1, 11)); // NOI18N
jLabel51.setText("Larg. de Histerese(*) :");

txtLarguraHisterese.setEditable(false);
txtLarguraHisterese.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
txtLarguraHisterese.setText("0.0");
txtLarguraHisterese.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        txtLarguraHistereseActionPerformed(evt);
    }
});

jLabel4.setFont(new java.awt.Font("Tahoma", 1, 11)); // NOI18N
```

```
        jLabel4.setText("% Histerese");

        txtPercHisterese.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
        txtPercHisterese.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                txtPercHistereseActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout jPanel4Layout = new javax.swing.GroupLayout(jPanel4);
        jPanel4.setLayout(jPanel4Layout);
        jPanel4Layout.setHorizontalGroup(
            jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel4Layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(jPanel4Layout.createSequentialGroup()
                        .addGroup(jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addGroup(jPanel4Layout.createSequentialGroup()
                                .addGap(52, 52, 52)
                                .addComponent(txtHmax, javax.swing.GroupLayout.PREFERRED_SIZE, 64,
javax.swing.GroupLayout.PREFERRED_SIZE))
                            .addComponent(jLabel45, javax.swing.GroupLayout.PREFERRED_SIZE, 100,
javax.swing.GroupLayout.PREFERRED_SIZE))
                        .addGap(14, 14, 14)
                        .addGroup(jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING,
false)
                            .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
jPanel4Layout.createSequentialGroup()
                                .addComponent(jLabel44, javax.swing.GroupLayout.PREFERRED_SIZE, 53,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                .addComponent(txtHmin, javax.swing.GroupLayout.PREFERRED_SIZE, 64,
javax.swing.GroupLayout.PREFERRED_SIZE))
                            .addComponent(txtHperc)))
                    .addComponent(jLabel43)
                    .addGroup(jPanel4Layout.createSequentialGroup()
                        .addGroup(jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addComponent(jLabel48)
                            .addComponent(jLabel49)
```

```
                    .addGroup(jPanel4Layout.createSequentialGroup()
                        .addComponent(btnSintoniaReleIdeal)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(btnGoReleIdeal)))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addGroup(jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(txtA, javax.swing.GroupLayout.PREFERRED_SIZE, 88, javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(txtKu, javax.swing.GroupLayout.PREFERRED_SIZE, 88, javax.swing.GroupLayout.PREFERRED_SIZE))))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(chkHisterese, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addGroup(jPanel4Layout.createSequentialGroup()
                    .addGroup(jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addComponent(jLabel51)
                        .addComponent(jLabel50)
                        .addComponent(jLabel4))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                    .addGroup(jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addComponent(txtPercHisterese)
                        .addComponent(txtTu, javax.swing.GroupLayout.DEFAULT_SIZE, 80, Short.MAX_VALUE)
                        .addComponent(txtLarguraHisterese, javax.swing.GroupLayout.PREFERRED_SIZE, 1, Short.MAX_VALUE))))
            .addGap(47, 47, 47))
    );
    jPanel4Layout.setVerticalGroup(
        jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(jPanel4Layout.createSequentialGroup()
            .addContainerGap()
            .addGroup(jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(jLabel45, javax.swing.GroupLayout.PREFERRED_SIZE, 20, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(txtHperc, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(chkHisterese))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
            .addGroup(jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
```

```
                .addComponent(jLabel43, javax.swing.GroupLayout.PREFERRED_SIZE, 20,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(txtHmax, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(jLabel44)
                .addComponent(txtHmin)
                .addComponent(jLabel4)
                .addComponent(txtPercHisterese, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(9, 9, 9)
            .addGroup(jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(jLabel49, javax.swing.GroupLayout.PREFERRED_SIZE, 20,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(jLabel50)
                .addComponent(txtTu, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(txtKu, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(jLabel48)
                .addComponent(jLabel51)
                .addComponent(txtLarguraHisterese, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(txtA, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(jPanel4Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(btnGoReleIdeal)
                .addComponent(btnSintoniaReleIdeal))
            .addGap(23, 23, 23))
        );

        PainelSelecionarSintonia.addTab("Parâmetros do Relé", jPanel4);

        panelGrafRelePainelPID.add(PainelSelecionarSintonia, java.awt.BorderLayout.CENTER);

        PainelPID.setBackground(new java.awt.Color(255, 255, 255));
        PainelPID.setBorder(javax.swing.BorderFactory.createTitledBorder("Parâmetros do PID"));
```

```
        lblKp.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        lblKp.setText("0.0");

        lblKi.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        lblKi.setText("0.0");

        jLabel16.setFont(new java.awt.Font("Tahoma", 1, 11)); // NOI18N
        jLabel16.setText("Ganho Proporcional (Kp):");

        lblKd.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        lblKd.setText("0.0");

        jLabel17.setFont(new java.awt.Font("Tahoma", 1, 11)); // NOI18N
        jLabel17.setText("Ganho Integral (Ki):");

        jLabel19.setFont(new java.awt.Font("Tahoma", 1, 11)); // NOI18N
        jLabel19.setText("Ganho Derivativo (Kd):");

        javax.swing.GroupLayout PainelPIDLayout = new javax.swing.GroupLayout(PainelPID);
        PainelPID.setLayout(PainelPIDLayout);
        PainelPIDLayout.setHorizontalGroup(
            PainelPIDLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(PainelPIDLayout.createSequentialGroup()
                .addGap(22, 22, 22)
                .addGroup(PainelPIDLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                    .addComponent(jLabel17, javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jLabel19, javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(PainelPIDLayout.createSequentialGroup()
                        .addComponent(jLabel16)
                        .addGap(63, 63, 63)))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 0,
Short.MAX_VALUE)
                .addGroup(PainelPIDLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
false)
                    .addComponent(lblKp, javax.swing.GroupLayout.DEFAULT_SIZE, 99, Short.MAX_VALUE)
                    .addComponent(lblKi, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addComponent(lblKd, javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
```

```
            .addGap(208, 208, 208))
        );
        PainelPIDLayout.setVerticalGroup(
            PainelPIDLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(PainelPIDLayout.createSequentialGroup()
                .addGap(5, 5, 5)
                .addComponent(jLabel16)
                .addGap(7, 7, 7)
                .addComponent(jLabel17, javax.swing.GroupLayout.PREFERRED_SIZE, 22,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jLabel19))
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, PainelPIDLayout.createSequentialGroup()
                .addContainerGap()
                .addComponent(lblKp)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(lblKi)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(lblKd)
                .addContainerGap())
        );

        panelGrafRelePainelPID.add(PainelPID, java.awt.BorderLayout.SOUTH);

        painelDireito.add(panelGrafRelePainelPID);

        painelConfiguracao.setBackground(new java.awt.Color(255, 255, 255));
        painelConfiguracao.setBorder(javax.swing.BorderFactory.createTitledBorder(null, "Dados do PID", 0, 0,
new java.awt.Font("Arial", 1, 12))); // NOI18N
        painelConfiguracao.setPreferredSize(new java.awt.Dimension(336, 234));
        painelConfiguracao.setRequestFocusEnabled(false);
        painelConfiguracao.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        txtNomeControlador.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                txtNomeControladorActionPerformed(evt);
            }
        });
        painelConfiguracao.add(txtNomeControlador, new org.netbeans.lib.awtextra.AbsoluteConstraints(260, 30,
150, -1));
```

```
        jLabel1.setText("Nome do projeto do controlador :");
        painelConfiguracao.add(jLabel1, new org.netbeans.lib.awtextra.AbsoluteConstraints(20, 30, -1, 20));

        jLabel3.setText("Tempo amostra (ms)");
        painelConfiguracao.add(jLabel3, new org.netbeans.lib.awtextra.AbsoluteConstraints(20, 70, -1, -1));

        txtPeriodoAmostra.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
        txtPeriodoAmostra.setText("100");
        txtPeriodoAmostra.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                txtPeriodoAmostraActionPerformed(evt);
            }
        });
        painelConfiguracao.add(txtPeriodoAmostra, new org.netbeans.lib.awtextra.AbsoluteConstraints(260, 60,
99, -1));

        btnGravar.setText("Salvar");
        btnGravar.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                btnGravarActionPerformed(evt);
            }
        });
        painelConfiguracao.add(btnGravar, new org.netbeans.lib.awtextra.AbsoluteConstraints(20, 170, 65, -1));

        btnColocarEP.setText("Colocar em EP");
        btnColocarEP.setEnabled(false);
        btnColocarEP.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                btnColocarEPActionPerformed(evt);
            }
        });
        painelConfiguracao.add(btnColocarEP, new org.netbeans.lib.awtextra.AbsoluteConstraints(90, 170, -1, -
1));

        btnAbrirControlador.setText("Abrir");
        btnAbrirControlador.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                btnAbrirControladorActionPerformed(evt);
            }
```

```
        });
        painelConfiguracao.add(btnAbrirControlador, new org.netbeans.lib.awtextra.AbsoluteConstraints(410, 30, -
1, 20));

        jLabel21.setText("Setpoint");
        painelConfiguracao.add(jLabel21, new org.netbeans.lib.awtextra.AbsoluteConstraints(20, 100, 160, -1));

        txtSetPoint.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
        txtSetPoint.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                txtSetPointActionPerformed(evt);
            }
        });
        painelConfiguracao.add(txtSetPoint, new org.netbeans.lib.awtextra.AbsoluteConstraints(260, 90, 99, -1));

        jLabel22.setText("Variável Manipulada MV (para colocar em EP)");
        painelConfiguracao.add(jLabel22, new org.netbeans.lib.awtextra.AbsoluteConstraints(16, 132, 230, -1));

        txtMV.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
        txtMV.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                txtMVActionPerformed(evt);
            }
        });
        painelConfiguracao.add(txtMV, new org.netbeans.lib.awtextra.AbsoluteConstraints(260, 120, 99, -1));

        painelTemperatura.setLayout(new java.awt.GridLayout(2, 1));
        painelConfiguracao.add(painelTemperatura, new org.netbeans.lib.awtextra.AbsoluteConstraints(0, 0, -1, -
1));

        jLabel2.setFont(new java.awt.Font("Arial", 1, 16)); // NOI18N
        jLabel2.setText("Saída (Temperatura)");
        painelConfiguracao.add(jLabel2, new org.netbeans.lib.awtextra.AbsoluteConstraints(20, 210, -1, 37));

        lblSaida.setBackground(new java.awt.Color(102, 102, 102));
        lblSaida.setFont(new java.awt.Font("Arial", 0, 28)); // NOI18N
        lblSaida.setForeground(new java.awt.Color(204, 204, 0));
        lblSaida.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
        lblSaida.setText("0.0");
        lblSaida.setBorder(javax.swing.BorderFactory.createEtchedBorder());
```

```
        painelConfiguracao.add(lblSaida, new org.netbeans.lib.awtextra.AbsoluteConstraints(20, 240, 170, 60));

        jLabel5.setFont(new java.awt.Font("Arial", 1, 16)); // NOI18N
        jLabel5.setText("Tempo decorrido");
        painelConfiguracao.add(jLabel5, new org.netbeans.lib.awtextra.AbsoluteConstraints(260, 210, 180, 37));

        lblTempoDecorrido.setBackground(new java.awt.Color(102, 102, 102));
        lblTempoDecorrido.setFont(new java.awt.Font("Arial", 0, 28)); // NOI18N
        lblTempoDecorrido.setForeground(new java.awt.Color(204, 204, 0));
        lblTempoDecorrido.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
        lblTempoDecorrido.setBorder(javax.swing.BorderFactory.createEtchedBorder());
        lblTempoDecorrido.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                lblTempoDecorridoActionPerformed(evt);
            }
        });
        painelConfiguracao.add(lblTempoDecorrido, new org.netbeans.lib.awtextra.AbsoluteConstraints(260, 240,
160, 60));

        painelDireito.add(painelConfiguracao);

        PainelEsquerdoMain.add(painelDireito, java.awt.BorderLayout.CENTER);

        painelMain.add(PainelEsquerdoMain);

        MasterPainel.add(painelMain, java.awt.BorderLayout.CENTER);

        getContentPane().add(MasterPainel, java.awt.BorderLayout.CENTER);

        pack();
    }// </editor-fold>

    private void txtPeriodoAmostraActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void btnGravarActionPerformed(java.awt.event.ActionEvent evt) {
        fc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
        int ret = fc.showOpenDialog(MainFrame.this);
```

```
    if (ret == JFileChooser.APPROVE_OPTION) {
        File file = fc.getCurrentDirectory();
        // seta os dados da gravação
        PIDSetup setup = new PIDSetup();
        setup.nome = txtNomeControlador.getText();
        setup.tempoAmostra = txtPeriodoAmostra.getText();
        setup.setPoint = txtSetPoint.getText();
        setup.pontoOperacaoMV = txtMV.getText();
        setup.percMV = txtHperc.getText();
        setup.A = txtA.getText();
        setup.Ku = txtKu.getText();
        setup.Tu = txtTu.getText();
        setup.Kp = lblKd.getText();
        setup.Ki = lblKi.getText();
        setup.Kd = lblKd.getText();
        setup.tipoSintonia = metodoSintonia;
        setup.caminho = file;

        IOFile io = new IOFile();
        io.salvar(setup);

        //
    }

}

private void btnColocarEPActionPerformed(java.awt.event.ActionEvent evt) {
}

private void txtSetPointActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

private void txtTuActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

private void txtKuActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}
```

```
    private void txtAActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void btnSintoniaReleIdealActionPerformed(java.awt.event.ActionEvent evt) {

        //inicializar();

        float setpoint = Float.parseFloat(txtSetPoint.getText());

        int option = JOptionPane.showConfirmDialog(MainFrame.this, "Confirma a execução / pausa de Auto Sintonia do Relé?", "Auto Sintonia", JOptionPane.OK_CANCEL_OPTION);
        if (option == JOptionPane.OK_OPTION) {
            if (btnSintoniaReleIdeal.getText().startsWith("Iniciar")) {
                //JOptionPane.showMessageDialog(null, "Teste.","Set OK", JOptionPane.PLAIN_MESSAGE);
                //initialize();
                inicializar();

                sistemaEmExecucao = true;
                planta.emEstadoPermanente = false;
                planta.colocandoEstadoPermanente = true;
                planta.tempoAmostra = Float.parseFloat(txtPeriodoAmostra.getText());
                planta.variavelManipulada = Float.parseFloat(txtMV.getText());
                planta.setPoint = Float.parseFloat(txtSetPoint.getText());
                controladorRI.hmax = Float.parseFloat(txtHmax.getText());
                controladorRI.hmin = Float.parseFloat(txtHmin.getText());

                // setpoint de operação da planta
                float sp = Float.parseFloat(txtSetPoint.getText());
                float td = sp;
                if (chkHisterese.isSelected()) {
                    metodoSintonia = RELE_HISTERESE;
                    int multiplicadorHisterese = Integer.parseInt(txtPercHisterese.getText().trim());
                    //sp += largHisterese;
                    //td -= largHisterese;
                    controladorRI = new Rele(multiplicadorHisterese);
                } else {
                    metodoSintonia = RELE_IDEAL;
                    controladorRI = new Rele();
```

```
            }
            // aciona planat
            String cmd = "PA#" + txtPeriodoAmostra.getText() + ";";  // define o tempo de amostragem
            cmd += "SP#" + sp + ";\n";  // diz qual o ponto de operação para sintonia
            cmd += "TD#" + td + ";\n";  // diz qual a temperatura de desligamento do ventilador
            cmd += "IE#;\n";  // diz qual o ponto de operação para sintonia
            /// Envia comando p/ parar execução
            enviaDadosSensor(cmd);
            // cmd = "I"+txtMV.getText()+"\n";
            cmd = "I1000\n";
            enviaDadosResistencia(cmd);
            horaInicio = System.currentTimeMillis();
            // parametros rele ideal

            lblAndamento.setText("Colocando sistema em estado permanente...");
            btnSintoniaReleIdeal.setText("Cancelar sintonia");
            btnSintoniaReleIdeal.setEnabled(true);

            // cria serviçde leitura da fila
            thFila = new Thread(new ProcessaFila());
            thFila.start();
        } else { // cancelar processo
            String cmd = "";
            //close();
            // Envia comando p/ parar execução
            cmd += "PE#;\n"; //supende a execucao e para a resistência
            enviaDadosSensor(cmd);
            cmd = "#\n;";
            enviaDadosResistencia(cmd);

            btnSintoniaReleIdeal.setText("Iniciar Sintonia");
        }
    }

}

private void txtHminActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}
```

```
    private void txtHmaxActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void txtMVActionPerformed(java.awt.event.ActionEvent evt) {
    }

    private void txtHpercActionPerformed(java.awt.event.ActionEvent evt) {

        float Hmax = (1 + Float.parseFloat(txtHperc.getText()) / 100) * Float.parseFloat(txtMV.getText());
        float Hmin = (1 - Float.parseFloat(txtHperc.getText()) / 100) * Float.parseFloat(txtMV.getText());
        txtHmax.setText(Float.toString(Hmax));
        txtHmin.setText(Float.toString(Hmin));
    }

    private void btnGoReleIdealActionPerformed(java.awt.event.ActionEvent evt) {
        if (btnGoReleIdeal.getText().startsWith("Iniciar")) {
            int option = JOptionPane.showConfirmDialog(MainFrame.this, "Confirma a execução do PID?",
"Execução PID", JOptionPane.OK_CANCEL_OPTION);
            if (option == JOptionPane.OK_OPTION) {
                //initialize();
                float kp = Float.parseFloat(lblKp.getText().replace(",", "."));
                float ki = Float.parseFloat(lblKi.getText().replace(",", "."));
                float kd = Float.parseFloat(lblKd.getText().replace(",", "."));
                float sp = Float.parseFloat(txtSetPoint.getText().replace(",", "."));
                float saidaLimites = 1000F;
                myPID =  new ServicoPID(kp, ki, kd);
                myPID.setSetPoint(sp);
                myPID.setOutputLimits(saidaLimites);
                pidEmExecucao = true;
                sistemaEmExecucao = true;
                planta.emEstadoPermanente = false;
                planta.colocandoEstadoPermanente = false;
                planta.tempoAmostra = Float.parseFloat(txtPeriodoAmostra.getText());
                planta.setPoint = Float.parseFloat(txtSetPoint.getText());
                inicializar();
                btnGoReleIdeal.setText("Parar PID");

                String cmd = "PA#" + txtPeriodoAmostra.getText() + ";";  // define o tempo de amostragem
                cmd += "SP#" + txtSetPoint.getText() + ";\n";  // diz qual o ponto de operação para sintonia
```

```
            cmd += "TD#" + txtSetPoint.getText() + ";\n";  // diz qual o ponto de operação para sintonia
            cmd += "IE#;\n";  // diz qual o ponto de operação para sintonia
            /// Envia comando p/ parar execução
            enviaDadosSensor(cmd);

            // cria serviçde leitura da fila
            thFila = new Thread(new ProcessaFila());
            thFila.start();
          }
        } else {
          int option = JOptionPane.showConfirmDialog(MainFrame.this, "Confirma cancelamento da execução do
PID?", "Execução PID", JOptionPane.OK_CANCEL_OPTION);
          if (option == JOptionPane.OK_OPTION) {
            String cmd = "";
            //close();
            // Envia comando p/ parar execução
            cmd += "PE#;\n"; //supende a execucao e para a resistência
            enviaDadosSensor(cmd);
            cmd = "#\n;";
            enviaDadosResistencia(cmd);
            pidEmExecucao = false;
            sistemaEmExecucao = false;
            btnGoReleIdeal.setText("Iniciar PID");
          }
        }
    }

    private void lblAndamentoActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void txtNomeControladorActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void btnAbrirControladorActionPerformed(java.awt.event.ActionEvent evt) {
        fc.setFileSelectionMode(JFileChooser.FILES_ONLY);
        int ret = fc.showOpenDialog(MainFrame.this);

        if (ret == JFileChooser.APPROVE_OPTION) {
```

```
            File file = fc.getSelectedFile();
            // seta os dados para recuperar
            IOFile io = new IOFile();
            System.out.println(file.getAbsolutePath());
            PIDSetup setup = io.abrir(file);

             txtNomeControlador.setText(setup.nome);
             txtPeriodoAmostra.setText(setup.tempoAmostra);
             txtSetPoint.setText(setup.setPoint);
             txtMV.setText(setup.pontoOperacaoMV);
             txtHperc.setText(setup.percMV);
             txtA.setText(setup.A);
             txtKu.setText(setup.Ku);
             txtTu.setText(setup.Tu);
             lblKp.setText(setup.Kp);
             lblKi.setText(setup.Ki);
             lblKd.setText(setup.Kd);
             metodoSintonia = setup.tipoSintonia;

             if (lblKp.getText().length() > 0 &&
                 lblKi.getText().length() > 0 &&
                 lblKd.getText().length() > 0) {

               btnGoReleIdeal.setEnabled(true);

             }

        }
    }

    private void txtLarguraHistereseActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void txtPercHistereseActionPerformed(java.awt.event.ActionEvent evt) {
        float largHisterese = Float.parseFloat(txtPercHisterese.getText().trim()) *
Float.parseFloat(txtSetPoint.getText()) / 100;
        txtLarguraHisterese.setText(Float.toString(largHisterese));
    }
```

```
private void lblTempoDecorridoActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

/**
 * @param args the command line arguments
 */
/**
 * Médoto que verifica se a comunicação com a porta serial está ok
 */
private void initialize() {
    try {
        //Define uma variável portId do tipo CommPortIdentifier para realizar a comunicação serial
        CommPortIdentifier portId = null;
        try {
            //Tenta verificar se a porta COM informada existe
            portId = CommPortIdentifier.getPortIdentifier(this.portaCOMSensor);
        } catch (NoSuchPortException npe) {
            //Caso a porta COM não exista será exibido um erro
            JOptionPane.showMessageDialog(null, "Porta COM não encontrada.",
                    "Porta COM Sensor", JOptionPane.PLAIN_MESSAGE);
        }
        //Abre a porta COM
        portSensor = (SerialPort) portId.open("Comunicação serial", this.taxa);
        serialOutSensor = portSensor.getOutputStream();
        portSensor.setSerialPortParams(this.taxa, //taxa de transferência da porta serial
                SerialPort.DATABITS_8, //taxa de 10 bits 8 (envio)
                SerialPort.STOPBITS_1, //taxa de 10 bits 1 (recebimento)
                SerialPort.PARITY_NONE); //receber e enviar dados
        // add event listeners
        portSensor.addEventListener(this);
        portSensor.notifyOnDataAvailable(true);
        portSensor.disableReceiveTimeout();
        portSensor.enableReceiveThreshold(1);
    } catch (Exception e) {
        e.printStackTrace();
    }

    // inicializa serial da resistencia
    try {
```

```
        //Define uma variável portId do tipo CommPortIdentifier para realizar a comunicação serial
        CommPortIdentifier portId = null;
        try {
           //Tenta verificar se a porta COM informada existe
           portId = CommPortIdentifier.getPortIdentifier(this.portaCOMResistencia);
        } catch (NoSuchPortException npe) {
           //Caso a porta COM não exista será exibido um erro
           JOptionPane.showMessageDialog(null, "Porta COM não encontrada.",
                 "Porta COM Resistencia", JOptionPane.PLAIN_MESSAGE);
        }
        //Abre a porta COM
        portResistencia = (SerialPort) portId.open("Comunicação serial", this.taxa);
        serialOutResistencia = portResistencia.getOutputStream();
        portResistencia.setSerialPortParams(this.taxa, //taxa de transferência da porta serial
              SerialPort.DATABITS_8, //taxa de 10 bits 8 (envio)
              SerialPort.STOPBITS_1, //taxa de 10 bits 1 (recebimento)
              SerialPort.PARITY_NONE); //receber e enviar dados
        // add event listeners
        // portResistencia.addEventListener(this);
        portResistencia.notifyOnDataAvailable(true);
        portResistencia.disableReceiveTimeout();
        portResistencia.enableReceiveThreshold(1);
     } catch (Exception e) {
        e.printStackTrace();
     }

}

public void enviaDadosResistencia(String cmd) {
   try {
      serialOutResistencia.write(cmd.getBytes());//escreve o valor na porta serial para ser enviado
   } catch (IOException ex) {
      JOptionPane.showMessageDialog(null, "Não foi possível enviar o dado. ",
            "Enviar dados", JOptionPane.PLAIN_MESSAGE);
   }
}

public void enviaDadosSensor(String cmd) {
   try {
      serialOutSensor.write(cmd.getBytes());//escreve o valor na porta serial para ser enviado
```

```
    } catch (IOException ex) {
        JOptionPane.showMessageDialog(null, "Não foi possível enviar o dado. ",
                "Enviar dados", JOptionPane.PLAIN_MESSAGE);
    }
}

/**
 * Método que fecha a comunicação com a porta serial
 */
public void close() {
    try {
        //if (port != null) {
        portSensor.removeEventListener();
        portResistencia.removeEventListener();
        //port.close();
        System.out.println("Fechou a porta COM8 e COM29");
        //}
        serialOutSensor.close();
        serialOutResistencia.close();
    } catch (IOException e) {
        JOptionPane.showMessageDialog(null, "Não foi possível fechar porta COM.",
                "Fechar porta COM", JOptionPane.PLAIN_MESSAGE);
    }
}

// evento serial
@Override
public void serialEvent (SerialPortEvent oEvent) {
    //System.out.println("Event received: " + oEvent.toString());
    // se o experimento nao estiver em execução sai
    if (sistemaEmExecucao) {

        switch (oEvent.getEventType()) {
            case SerialPortEvent.DATA_AVAILABLE:
                try {
                    readSerial();
                } catch (IOException e) {
                    System.out.println(e.getMessage());
                }
                break;
```

```
        }
    }
}

private void readSerial() throws IOException {
    if (portSensor.getInputStream().available() > 0) {
        saida = new BufferedReader(
                new InputStreamReader(
                portSensor.getInputStream()));

            y = (String) saida.readLine(); // saida da planta

            if (y != null) {
                if  (y.length() > 0) {
                    fila.insere(new Ponto(System.currentTimeMillis() - horaInicio, Float.parseFloat(y)));
                    lblTempoDecorrido.setText(formataHora(System.currentTimeMillis() - horaInicio));

                }

            }

    }
}



/* Processa a fila de valores lidos do sensor de tempertura
 *
 */

class ProcessaFila implements Runnable {

    private String cmd;
    private double u_t_pid, u_t_pid_old;
    @Override
    public void run() {
        u_t_pid_old = 0;
        while (sistemaEmExecucao) {
            //System.out.print("size fila = "+fila.vazia());
            if (!fila.vazia()) {
```

```
Ponto y = fila.remove();
//System.out.println("tirou da fila y = "+y);
planta.valorLido = y.getValor();

if (planta.colocandoEstadoPermanente) {
   if (planta.verificaSeAlcancouEstadoPermanente()) {
      System.out.println("alcançou o estado permanente!");
      // btnSintoniaReleIdeal.setEnabled(true);
      lblAndamento.setText("Sistema em estado permenente!");
      // btnColocarEP.setText("Colocar em EP");
      //controlador = new ReleIdeal();
      inicializar();
   }
}


lblSaida.setText(y.getValor() + " \u00b0C");
//acumuladorTempo = acumuladorTempo + (long)planta.tempoAmostra;
//acumuladorTempo = acumuladorTempo + (System.currentTimeMillis() - horaInicio);
//System.out.println("Tempo = " + y.getTempo());
if (!planta.colocandoEstadoPermanente) {
   plota(Long.toString(y.getTempo()), Float.toString(y.getValor()) );
}
//Planta.pontos.add(new Ponto(acumuladorTempo, valorY));

if (pidEmExecucao) {

   u_t_pid = myPID.getSaida(y.getValor());
   plotaRele(Long.toString(y.getTempo()), Double.toString(u_t_pid));

   if (u_t_pid != u_t_pid_old) {
      cmd = "I" + u_t_pid + "\n";
      enviaDadosResistencia(cmd);
      u_t_pid_old= u_t_pid;
   }

   lblAndamento.setText("Controle PID : u(t) = " + u_t_pid );
   try {
      Thread.sleep(100);
   } catch (InterruptedException ex) {
```

```
                Logger.getLogger(MainFrame.class.getName()).log(Level.SEVERE, null, ex);
            }


        }

        if ((metodoSintonia == RELE_IDEAL || metodoSintonia == RELE_HISTERESE ) && planta.emEstadoPermanente) {
            if (planta.sintonizando) {
                float u_t = controladorRI.sintonizar(y.getTempo(), y.getValor());
                plotaRele(Long.toString(y.getTempo()), Float.toString(u_t));
                cmd = "I" + u_t + "\n";
                enviaDadosResistencia(cmd);
                if (Rele.alteraLarguraHisterese) {
                    double sp = planta.setPoint + Rele.larguraHisterese;
                    double td = planta.setPoint - Rele.larguraHisterese;
                    cmd += "SP#" + sp + ";\n";  // diz qual o ponto de operação para sintonia
                    cmd += "TD#" + td + ";\n";  // diz qual a temperatura de desligamento do ventilador
                    enviaDadosSensor(cmd);
                    Rele.alteraLarguraHisterese = false;
                }



                lblAndamento.setText("Sintonizando sistema : u(t) = " + u_t + " erro =" + controladorRI.erro + " qtde de picos = " + controladorRI.qtdePicos);

            }

            if (planta.sintonizado) {
                lblAndamento.setText("Sistema sintonizado!");
                cmd = "";
                // Envia comando p/ parar execução
                cmd = "DV#;\n"; //para a execucao e para a resistência
                cmd += "PE#;\n"; //para a execucao e para a resistência
                enviaDadosResistencia("#\n");
                enviaDadosSensor(cmd);

                txtA.setText(formata(controladorRI.A));
                txtKu.setText(formata(controladorRI.Ku));
```

```
                txtTu.setText(formata(controladorRI.Tu));
                lblKp.setText(formata(controladorRI.Kp));
                lblKi.setText(formata(controladorRI.Ki));
                lblKd.setText(formata(controladorRI.Kd));

                btnGoReleIdeal.setEnabled(true);
                //close();
                // plota picos
                //XYLineAndShapeRenderer renderer = new XYLineAndShapeRenderer();
                //renderer.setSeriesLinesVisible(1, false);
                //renderer.setSeriesShapesVisible(1, false);
               // plotSistema.setRenderer(renderer);
                /*List<Ponto> picos = Rele.picos;
                for (Ponto ponto : picos) {
                   //System.out.println(ponto.getTempo() + " - " + ponto.getValor());

                   plota(Long.toString(ponto.getTempo()), Float.toString(ponto.getValor()));
                }*/

              }
            }
          }

      }
      System.out.print("Saiu do looping!! ");

   }
}

private String formata(double number) {
   DecimalFormat df = new DecimalFormat("###.##");
   return (df.format(number));
}
private String formataHora(long tempo) {

   SimpleDateFormat df = new SimpleDateFormat("mm:ss");
   Date tempoDecorrido = new Date(tempo);
   return (df.format(tempoDecorrido));
}
```

```
    public static void main(String args[]) {
        /*
         * Set the Nimbus look and feel
         */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /*
         * If Nimbus (introduced in Java SE 6) is not available, stay with the
         * default look and feel. For details see
         * http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {
            java.util.logging.Logger.getLogger(MainFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex) {
            java.util.logging.Logger.getLogger(MainFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {
            java.util.logging.Logger.getLogger(MainFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            java.util.logging.Logger.getLogger(MainFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>

        /*
         * Create and display the form
         */
        java.awt.EventQueue.invokeLater(new Runnable() {

            public void run() {
                new MainFrame().setVisible(true);
```

```
        }
    });
}
// Variables declaration - do not modify
private javax.swing.JPanel MasterPainel;
private javax.swing.JPanel PainelEsquerdoMain;
private javax.swing.JPanel PainelPID;
private javax.swing.JTabbedPane PainelSelecionarSintonia;
private javax.swing.JButton btnAbrirControlador;
private javax.swing.JButton btnColocarEP;
private javax.swing.JButton btnGoReleIdeal;
private javax.swing.JButton btnGravar;
private javax.swing.JButton btnSintoniaReleIdeal;
private javax.swing.JCheckBox chkHisterese;
private static javax.swing.JPanel graficoRele;
private static javax.swing.JPanel graficoSistema;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel16;
private javax.swing.JLabel jLabel17;
private javax.swing.JLabel jLabel19;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel21;
private javax.swing.JLabel jLabel22;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel43;
private javax.swing.JLabel jLabel44;
private javax.swing.JLabel jLabel45;
private javax.swing.JLabel jLabel48;
private javax.swing.JLabel jLabel49;
private javax.swing.JLabel jLabel5;
private javax.swing.JLabel jLabel50;
private javax.swing.JLabel jLabel51;
private javax.swing.JMenu jMenu1;
private javax.swing.JPanel jPanel4;
public static javax.swing.JTextField lblAndamento;
private javax.swing.JLabel lblKd;
private javax.swing.JLabel lblKi;
private javax.swing.JLabel lblKp;
private javax.swing.JTextField lblSaida;
```

```
    private javax.swing.JTextField lblTempoDecorrido;
    private javax.swing.JPanel painelConfiguracao;
    private javax.swing.JPanel painelDireito;
    private javax.swing.JPanel painelEsquerdo;
    private javax.swing.JPanel painelGraficos;
    private javax.swing.JPanel painelMain;
    private javax.swing.JPanel painelTemperatura;
    private javax.swing.JPanel panelAndamento;
    private javax.swing.JPanel panelGrafRelePainelPID;
    private javax.swing.JTextField txtA;
    private javax.swing.JTextField txtHmax;
    private javax.swing.JTextField txtHmin;
    private javax.swing.JTextField txtHperc;
    private javax.swing.JTextField txtKu;
    private javax.swing.JTextField txtLarguraHisterese;
    private javax.swing.JTextField txtMV;
    private javax.swing.JTextField txtNomeControlador;
    private javax.swing.JTextField txtPercHisterese;
    private javax.swing.JTextField txtPeriodoAmostra;
    private javax.swing.JTextField txtSetPoint;
    private javax.swing.JTextField txtTu;
    // End of variables declaration
    /*
     * To change this license header, choose License Headers in Project
     * Properties. To change this template file, choose Tools | Templates and
     * open the template in the editor.
     */
}
```