Relatório final de atividades
Iniciação Científica I - F590

"Automatização de um sistema de preparação de amostras via interfaceamento computadorizado e PID digital"

Aluno: Jean-Yves Roulet
RA: 170315 - Engenharia Física
j170315(arroba)dac.unicamp.br

Orientador: Prof. Dr. Abner de Siervo
asiervo(arroba)ifi.unicamp.br

Grupo de Física de Superfícies - GFS
Departamento de Física Aplicada, Instituto de Física "Gleb Wataghin"
UNICAMP

Resumo do projeto

Este projeto visa automatização e controle de um sistema de *sputtering-annealing* existente no Laboratório de Física de Superfícies pela implementação de um programa realizado em *Visual C#* que interfaceia as eletrônicas de cinco equipamentos (um canhão de íons de argônio, uma fonte de alta tensão, uma placa de controle para chaveamento de alta tensão (*sputtering)*, uma fonte de corrente controlada por computador (*annealing)* e um pirômetro óptico). O controle da temperatura da amostra é realizado via um sistema de retroalimentação e atuação na fonte de corrente utilizando a leitura digital da temperatura no pirômetro e uma rotina de controle PID (proporcional-integral-derivativo). Simplificadamente o objetivo deste projeto é a elaboração de um programa que poderá se comunicar com os equipamentos desejados a fim de controlar os ciclos de bombardeamento de íons inertes de argônio (*sputtering)* e os ciclos de aquecimento (*annealing*) da amostra em diferentes rampas de aquecimento e resfriamento. Usa-se a teoria de controle PID, pois é um algoritmo conveniente para que a temperatura da amostra seja controlada com precisão. Com a instrumentação existente, espera-se em um primeiro momento controlar a temperatura da amostra com precisão melhor que 0.5 C. Com relação ao realizado neste projeto, foi efetuada a comunicação entre o computador e cada um dos cinco equipamentos mencionados assegurando o bom funcionamento futuro do controle (leituras e outputs) que dependerá da intercomunicação entre todos os equipamentos e o computador. Também foram realizados diversos programas em *Visual C#* que visavam à aprendizagem desta linguagem de programação e comunicação com portas seriais RS232, como um programa de rampeamento de corrente/tempo e leitura de dados de uma interface Arduino. A seguir foi feito um programa que possibilitou um controle PID completo da temperatura de dois sistemas testes. Por último foi sendo desenvolvido o programa final (algoritmo) de controle de ciclos *sputtering-annealing* e sua interface gráfica. Esta etapa final é bastante delicada e lenta, pois requer o máximo de testes para encontrar o controle mais eficaz; e bastante cuidado para o desenvolvimento do programa como um todo para que atenda a todas as necessidades do laboratório e utilização diária pelos envolvidos.

1- Introdução

A limpeza e preparação de superfícies através de ciclos de bombardeamento de íons inertes de argônio (*sputtering*) e aquecimento (*annealing*) é uma das etapas cruciais em experimentos que envolvem a análise de superfícies, por exemplo, a determinação da estrutura eletrônica e atômica, ou crescimento de nanodepósitos e filmes finos. O principal requisito para conseguir reconstruir a superfície em grandes terraços monoatômicos devidamente ordenados não reside somente na temperatura final de aquecimento, mas essencialmente nas velocidades de aquecimento e resfriamento da amostra.

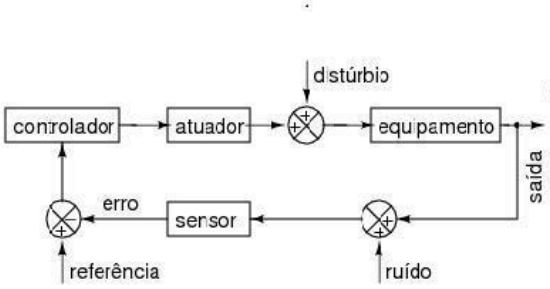
Tipicamente são necessários 5 a 10 ciclos de *sputtering-annealing* para se obter uma superfície vicinal própria para estudos. Detalhadamente temos que cada ciclo de *sputtering-annealing* deve ser realizado seguindo exigências precisas: *sputtering* com íons de argônio acelerados a uma ddp de 600V por um período de tempo de meia hora, logo em seguida a amostra deve ser aquecida até 800K a uma taxa constante de 1K por segundo, mantida a temperatura máxima por dez minutos e resfriada a uma outra taxa constante de aproximadamente 1/3K por segundo. Portanto, para uma amostra típica, cada ciclo demora aproximadamente uma hora e a sua preparação pode durar até dez horas. Este tipo de preparação de amostras é demorado e quando realizado manualmente é sujeito a erros que acarretarão na maioria das vezes a necessidade de se recomeçar a preparação com uma nova amostra. Por isso é útil a automação deste processo a fim de eliminar possíveis erros ao longo deste, e possibilitar

a preparação de amostras em períodos mais convenientes, como durante a noite. Reiterando-se que este projeto visa justamente o aperfeiçoamento da preparação de amostras através de sua automação com embasamento na teoria de controle PID escolhida por ser precisa, sensível e adaptável aos mais diversos tipos de sistemas.

2- Atividades realizadas

2a) Estudo do problema e da literatura existente

O problema a ser resolvido envolve a automação de um processo de malha fechada,ou seja, um controle que tem como resposta uma função da leitura de entrada (output em função de input) que determina as leituras subsequentes, atuando em ciclos repetidos programados (loops do algoritmo). No caso específico desta IC, o que se quer é controlar a temperatura de uma amostra seguindo passos pré-estabelecidos; rampa de aquecimento, manutenção na temperatura máxima e rampa de resfriamento; sabendo que a temperatura real da amostra somente pode diferir de mais ou menos meio grau da temperatura desejada (setada como alvo: setPoint). Por isso o algoritmo PID é requisitado pela sua adaptabilidade a esse tipo de sistema de controle de temperatura e eficácia sob os mais diversos aspectos: rapidez, precisão e relativa simplicidade. Um estudo aprofundado sobre controle PID foi realizado, e este pode ser ilustrado pelas figuras 1 e 2.



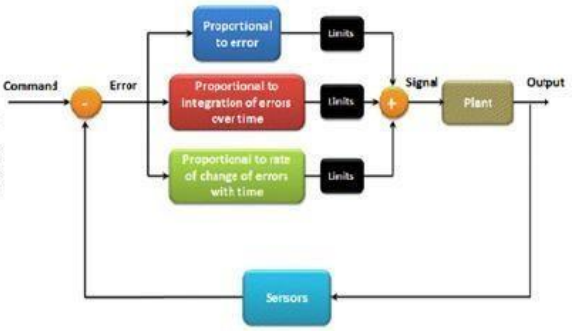


Figura1: Esquema de um controle de malha fechada. ***Figura2: Diagrama de blocos-algoritmo PID.***

Podemos compreender pela ilustração acima o uso do termo “sistema de controle de malha fechada” já que visualmente percebe-se que a leitura dos sensores fornece um *feedback* constante que controla a saída de um atuador para dar algum comando a um agente do sistema (no caso deste projeto um aquecedor/fonte de corrente) e reajustar a variável controlada para que chegue ao valor almejado: setPoint. O que diferencia um simples controle de malha fechada daquele com o algoritmo PID é justamente a forma como é calculada a saída do atuador, neste projeto é o valor de corrente a ser enviada á fonte para esfriar ou aquecer a amostra. Pela figura 2 temos que o sinal de saída é proporcional ao erro, este definido como sendo a diferença entre o ponto de ajuste (setPoint) e a variável do processo (temperatura lida), proporcional à integral do erro desde o início do processo (acúmulo) e proporcional à derivada (instantâneo) do erro. Do termo ‘proporcional’ vem a criação de três constantes deste controle, um para cada relação: kP, kI e kd respectivamente proporcional ao erro, à integral e à derivada do erro. Uma representação um pouco mais matemática do algoritmo PID pode ser dada pela figura 3.

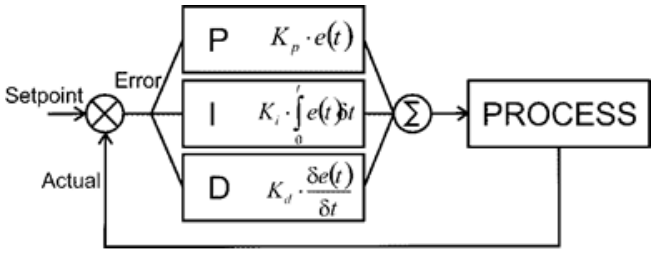
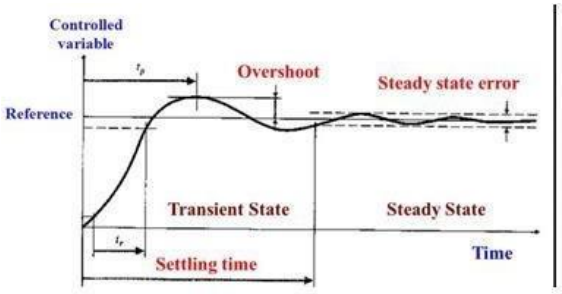


Figura3:Diagrama de blocos do algoritmo PID.

As constantes kP, kD e kI podem ser determinadas de várias maneiras dentre elas as mais recorrentes são: tentativa e erro, método de Ziegler-Nichols e auto-tuning. Vale lembrar que o valor de cada constante pode ser diferente para sistemas diferentes e que dependendo do sistema tratado, existem várias possibilidades para estas constantes. Estas possibilidades representam as diferentes maneiras com as quais a variável de controle se aproxima do setPoint e qual é a tolerância de seu desvio na situação de equilíbrio(steady-state error).Nas figuras a seguir são representadas várias possibilidades com as quais a variável de controle se aproxima do valor desejado e ilustradas algumas nomenclaturas úteis a este tipo de controle:



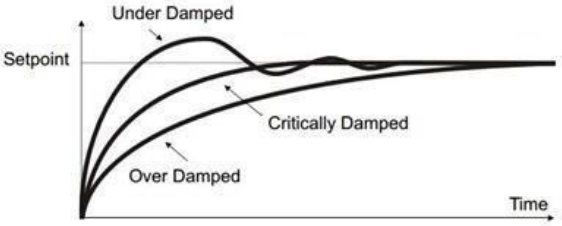


Figura4: Resposta típica de um sistema PID.

Figura5: Possíveis respostas de um sistema PID.

Além disso, para muitos sistemas um controle PID completo pode não ser o mais vantajoso e a implementação de um algoritmo simplesmente P, PI e PD torna-se mais apropriada. Atualmente a calibração de um PID é na maioria das vezes realizada pelo método de Ziegler-Nichols por ser um método simples e que não apresenta muitos riscos de proporcionar um *overshoot* elevado e danificar algum controlador ou atuador do sistema. Neste método as constantes kI e kD são ajustadas como zero inicialmente e a kP é aumentada até o sistema oscilar. Quando isto acontece o valor de kP é dado como kC e o período das oscilações Pc é anotado. Depois as constantes kP, kI e kD são ajustadas segundo a tabela abaixo, onde kP, kI, kD foi substituído por P, I, D respectivamente. Vale lembrar que existem outros métodos para realizar o *tuning* de um sistema com algoritmo PID, mas que este é o mais rápido e preciso. O *auto-tuning* de um sistema também pode ser realizado, mas no geral requer equipamentos caros e/ou algoritmos demasiadamente sofisticados e não será necessário para este projeto.

Control	P	Ti	Td
P	0.5Kc	-	-
PI	0.45Kc	Pc/1.2	-
PID	0.60Kc	0.5Pc	Pc/8

Tabela 1. Ajuste por Ziegler-Nichols, usando o método de oscilação.

Entendido o funcionamento do algoritmo PID, se torna imprescindível a realização de leituras de dados dos equipamentos. Todos estes se comunicam com o computador através de uma comunicação denominada RS232. RS232 (porta de comunicação) é um protocolo para trocas seriais entre um DTE (*Data Terminal equipment*) e um DCE (*Data Communication equipment*), e correntemente usado nas portas seriais dos computadores. Foi estudado, analisado e testado exaustivamente o protocolo de comunicação RS232 em todos os equipamentos garantindo uma boa intercomunicação futura no programa PID final.

Também foi crucial a aprendizagem de uma nova linguagem de programação: o *C Sharp (C#)*, visto a necessidade de se criar uma interface gráfica para acompanhamento da temperatura das amostras preparadas, e dos comandos enviados e recebidos pelas diversas eletrônicas envolvidas. Todos os programas desenvolvidos até o momento presente desta iniciação científica foram feitos em *C#* com o pacote de desenvolvimento de programas *Microsoft Visual Studio 2005-2008-2015* com licença liberada pela Microsoft. Inicialmente foram realizados pequenos programas para assimilação da lógica de programação visual como a confecção de uma calculadora, de um plotador de gráficos, etc... Em seguida foram sendo desenvolvidos programas mais complexos visando o aperfeiçoamento da comunicação serial, da representação de gráficos em tempo real, do controle da fonte, leitura de dados (temperatura) e teste de implementação de *backgroundWorker*, para permitir que os programas que são montados tenham sua parte visual e não visual separadas , o que confere o não travamento da parte visual de um programa mesmo que este esteja sendo usado para realizar outra tarefa independente.

2b) Comunicação com o pirômetro óptico

Esta comunicação foi realizada com a montagem de um programa em *Visual C#* que possibilitou a abertua e fechamento da porta serial ligada ao pirômetro, o recebimento de dados (temperatura, etc...) através do padrão protocolado pelo manual de instruções (*InfraWin5*). Por exemplo, o envio de certo tipo de caracter define o dado a ser enviado ao computador pelo pirômetro, como AA00X (X caracter específico).

2c) Comunicação com a fonte de corrente (annealing)

A fonte de corrente contínua (baixa tensão) da *Icel - modelo PS7000*, também permite a criação de um programa visual para sua comunicação. O programa foi baseado abertura e fechamento de porta serial, e envio e recepção de dados através do protocolo comunicativo presente no manual de instruções. Parte do código do programa da própria Icel para comunicação foi aproveitada e usada como uma espécie de biblioteca na base criada: (*no início do programa - PSServerProj1 foi adicionado às referências*)

```
PSServerProj1.PSServerClass FONTE = new  PSServerProj1.PSServerClass();
```

Segue abaixo duas fotos de programas feitos para comunicação serial com a fonte,e em seguida um pedaço de código de comunicação serial:

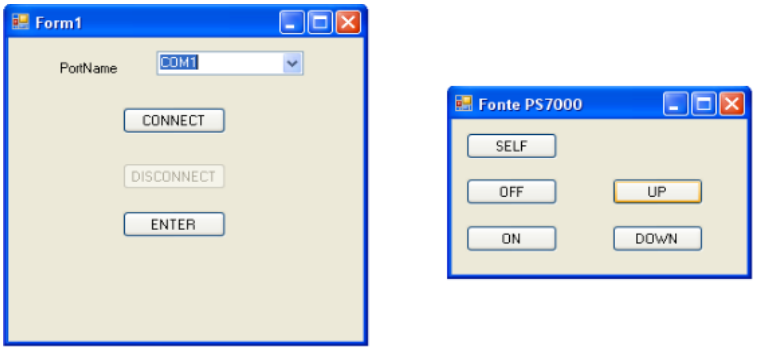


Figura6: Programas de comunicaçaõ com a fonte

```
private void button2_Click(object sender, EventArgs e)//CONNECT
         {
          FONTE.PS_OpenComm(1, 38400, 8, 0, 0);
          FONTE.PS_ControlState(3);
          System.Threading.Thread.Sleep(300);
         }
 Private void button3_Click(object sender, EventArgs e)//DISCONNECT
        {FONTE.PS_CloseComm();}
```

A fonte se comunica através do método acima, graças ao protocolo específico de troca de dados: FONTE.PS_"AÇÃO". Para outros equipamentos (como no caso do pirômetro) foi feita uma comunicação direta via porta serial: ajustando-se Baudrate, DataBits, StopBits, Parity; abrindo e fechando a porta: serialPort.Open() ou serialPort.Close(),enviando dados: serialPort.Write().

2d) Comunicação com a caixa de sputtering/annealing

Comunicação clássica via porta serial com placa *RCOM-Homebee*. Esta comunicação foi testada e se mostrou perfeitamente funcional, garantindo o bom funcionamento da automatização final. Foram desenvolvidos dois programas para o uso desta caixa de controle. Basicamente ela tem a função de ligar e desligar dois relês e duas saídas *DTL (Diode-Transistor Logic)*.Para o controle a ser utilizado somente o segundo relê será usado.O programa foi desenvolvido mapeando os bits de estado da placa segundo o protocolo do manual de instruções.Para controlar a caixa, é preciso saber qual o seu estado, pois, por exemplo, se queremos desligar alguma saída *DTL* sem se conhecer o estado da placa enviando o byte de comunicação *7C (ASCII)* mais o byte de ajustagem *0* (*decimal)* todas as saídas serão desativadas.Logo é necessário saber qual o estado da placa para mudar somente o *bit* de controle da saída em questão. Foi observado uma resposta aleatória do estado da placa, assim o primeiro programa montado embora tivesse um algoritmo perfeito, não funcionava corretamente. Assim foi desnvolvido um segundo programa no qual a placa é ajustada no início e fechamento do programa com estado *0* e o *byte* de controle da placa foi absorvido como sendo o estado da placa. Foi confirmado (analisando o circuito da placa e medindo tensões com um multímetro) que o *byte* enviado para a placa realmente designava o estado da placa. Este segundo programa

funciona correntamente,foi incluído as funcões deste programa no Apêndice B e a interface visual está mostrada a seguir:

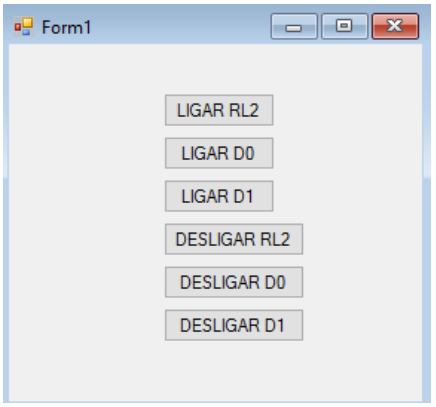


Figura7: Interface do programa de comunicação com a caixa de *sputtering/annealing*

2e) Confecção do programa de controle da fonte

Antes de criar um programa de controle PID que ajusta a temperatura da amostra com a corrente que passa pelo filamento de bombardeamento de elétrons, foi necessário ter um domínio maior sob a fonte. Por isso foi criado um programa que permititiu que se rampeasse tal corrente. Ou seja, o usuário escolhe a maneira com a qual se rampeia a corrente e o programa faz esse rampeamento: a corrente aumenta de zero a um valor máximo a uma taxa constante, e permanece no máximo por um intervalo de tempo e depois diminui à uma certa taxa constante até ser zerada novamente. Segue abaixo um *print-screen* do programa depois de ter efetuado um rampeamento:

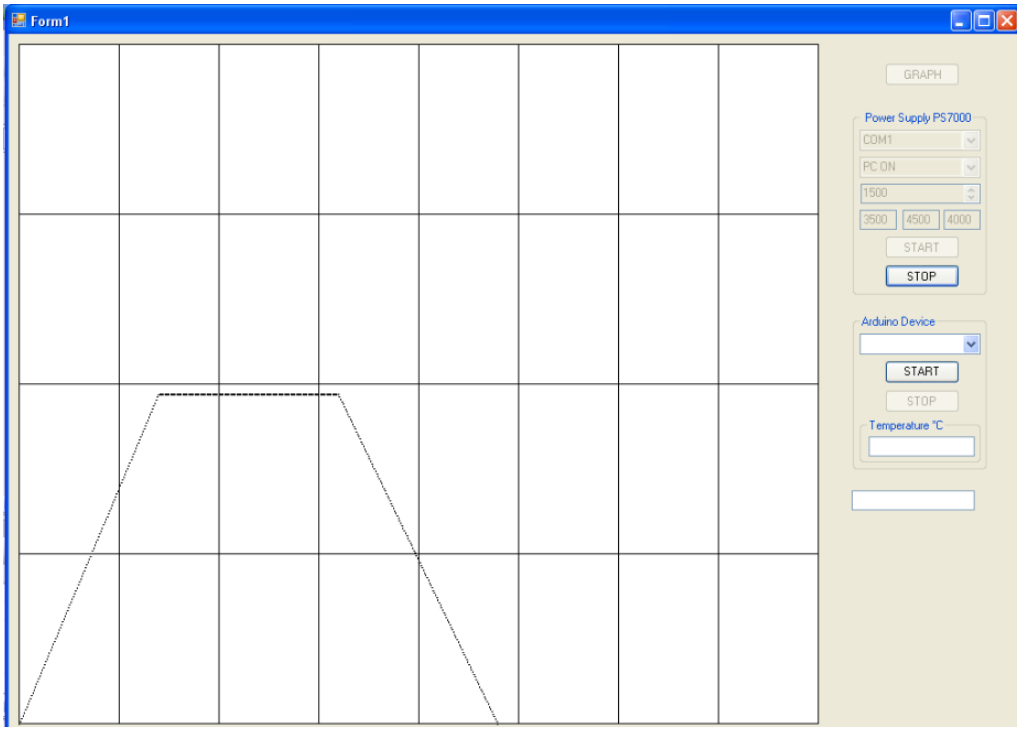


Figura8:Exemplo de um rampeamento de corrente. (Corrente máxima: 1500 mA, tempo de subida: 2500ms, tempo de estabilidade: 4500ms, tempo de descida:4000ms)

Depois de se controlar melhor a fonte, foi montado um programa de controle PID para testes em dois sistemas diferentes. Fotos dos dois sistemas são mostradas abaixo:

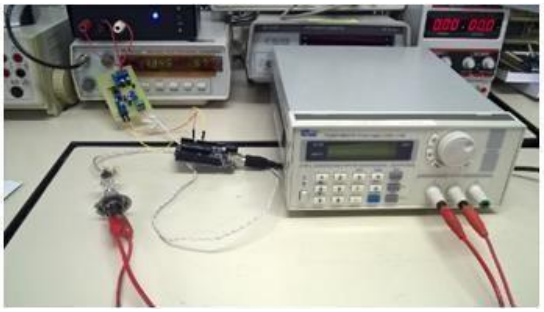


Figura9:Sistema teste com uma lâmpada

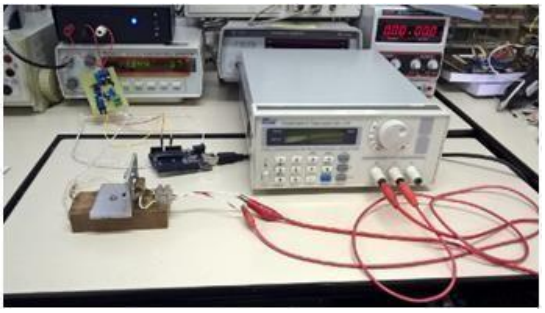


Figura10:Sistema teste com um resistor

Nestes dois sistemas estão presentes a fonte PS700 da Icel, um Arduino, um termopar, uma lâmpada ou resistor, e um circuito auxiliar para leitura de tempeatura. Neste sistema a fonte e o arduino se conectam ao computador, a fonte gera corrente para lâmpada/resistor que aquece o meio e por consequente o termopar em volta deste dispositivo. Portanto é possível simular um sistema de preparação de amostras (*annealing*) e testar algoritmos PID antes de elaborar uma versão final para teste no sistema de preparação de amostras do laboratório. É importante lembrar que o ajuste de um PID e sua implementação correta e segura requerem experiência e por isso é sensato por em prática toda a teoria e ajustar sistemas testes antes de se confrontar a um sistema mais delicado (preparação de amostras). Um programa com algoritmo PID que permite acompanhamento do controle foi elaborado e pode se testar o algoritmo.

Nestes sistemas, a corrente gerada pela fonte atravessa lâmpada/resistor que aquecem o meio; junto com o termopar,o arduino e o circuito permitem a leitura da temperatura em volta da/o lâmpada/resistor. Com leituras de temperatura é possivel realizar um controle desta justamente com o ajuste de corrente gerada.Um pedaço do algoritmo PID é mostrado no apêndice A deste relatório.

2f) Testes das rotinas de controle PID

Com o programa de PID (puro) desenvolvido, foi possível testar o algoritmo e os parâmetros de PID de várias maneira. Por tentativa e erro, obteve-se com a lâmpada um controle bem estável: a temperatura chega ao setPoint com um desvio percentual baixo (0,5%). Obteve-se um resultado semelhante com o método de Ziegler-Nichols. Entretanto houve algumas melhoras: o *overshoot* diminuiu e a estabilidade do sistema aumentou um pouco. Foram comprovadas as expectativas teóricas da teoria de controle PID, a função das constante kP, kI e kD atenderam às expectativas: maiores valores de kP correspondem à oscilações maiores, a kI gera uma integral residual que cria justamente a saída de controle necessária na estabilidade e a kD faz com que a saída diminua se a variável do processo está aumentando rapidamente e aumente caso contrário. Na tentativa e erro os valores de kP, kI e kD foram acertados em 7,3, 0,65 e 0,045 respectivamente. Já com o método de de Ziegler-Nichols chegou-se à 7,1, 0,3 e 0,035 respectivamente. Algumas fotos destes *tunings* estão disponíveis a seguir:

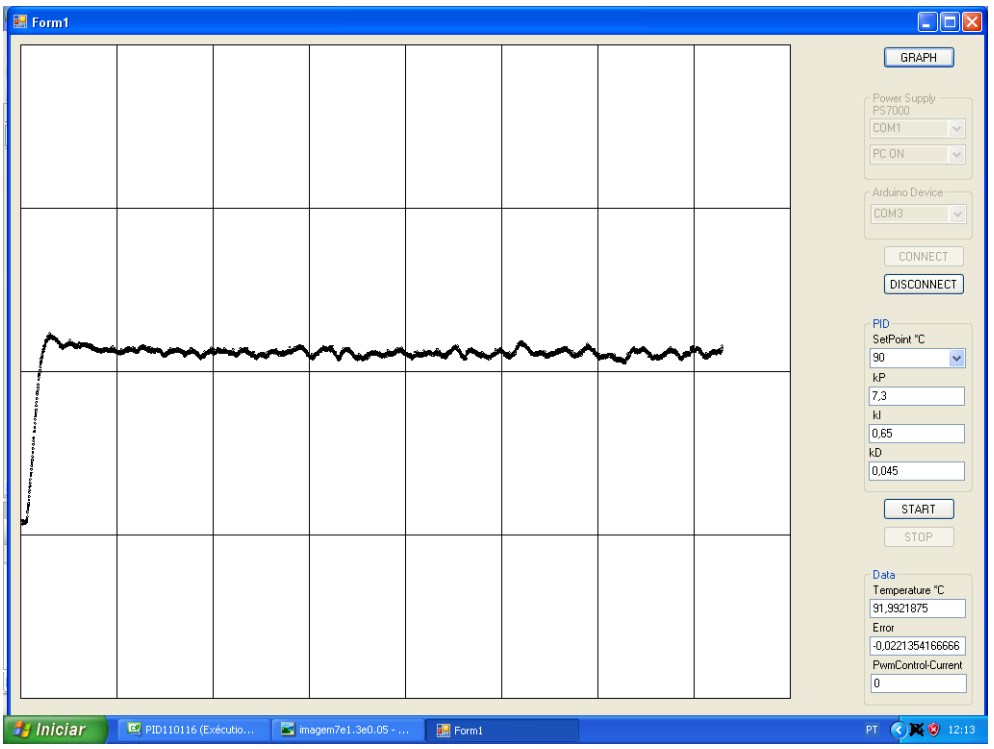


Figura11: Comportamento do sistema teste com *tuning* por tentativa-erro

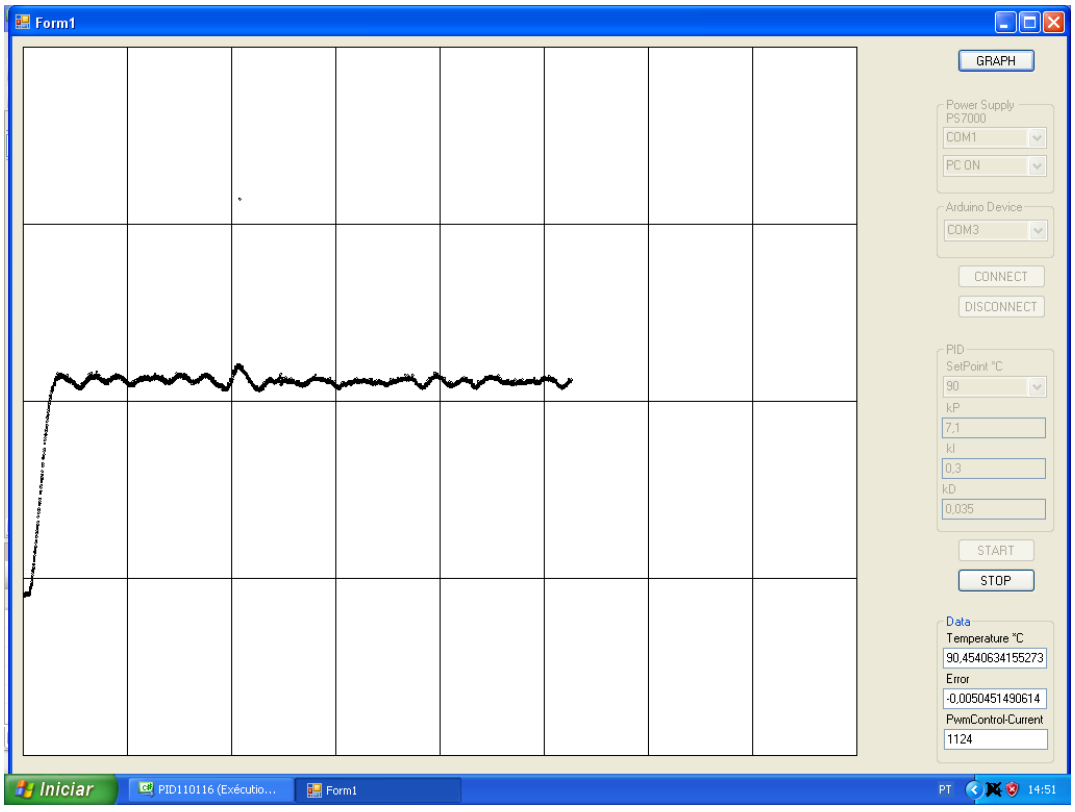


Figura12: Comportamento do sistema com tuneamento pelo método de Ziegler-Nichols

2g) Desenvolvimento e testes do programa final dos ciclos de sputtering/annealing

Após a elaboração de um algoritmo PID (puro) e de programas de comunicação entre o computador e cada eletrônico diretamente controlado (caixa de *sputtering/annealing*, pirômetro óptico, fonte controlada por computador), o programa final da automatização do sistema de amostras foi elaborado. Neste programa, o usuário determina a quantidade de ciclos *sputtering/annealing* a que deseja submeter cada amostra, além de determinar a quantidade de cada processo individual de *sputtering* e de *annealing* em cada um destes ciclos. Por exemplo, o usuário pode definir no programa dez ciclos de *sputtering/annealing* e em cada um destes ciclos haverá um *sputtering* e depois dois *annealings*.

No programa também é possível determinar o tipo de controle desejado durante o *annealing*: utilizando PID ou rampeamento de corrente (sem *feedback*). Para o rampeamento de controle, somente foi incorporado ao programa o programa de rampeamento de corrente já feito anteriormente. O usuário também pode ajustar neste caso o tempo de cada processo do rampeamento: subida,estabilidade e descida; e a corrente na estabilidade. Todas estas opções podem ser definidas num menu incorporado ao programa.Além se o usuário esquecer ou não definiu estas constantes propositadamente, o programa irá realizar um controle *standard* ou *default.* As constantes envolvidas nesta versão *standard* estão ainda a ser definidas com o orientador para que estabelecem o controle de corrente que será mais frequente no dia a dia do laboratório, mas foram definidas para testes no momento.

Para o controle de *annealing* do tipo PID, foram desenvolvidos dois tipos de algoritmos.O primeiro foi uma modificação do PID para o rampeamento: no algoritmo o *setPoint,* temperatura almejada agora é variável segundo a reta 1K/s na subida e 1/3K/s na descida. No segundo o *setPoint* é uma derivada constante: 1 ou 1/3 de Kelvin por segundo. No dois casos o PID foi praticamente mantido inalterado para o controle da temperatura na estabilidade, somente variando a corrente de controle que agora é definida como sendo a corrente da fonte quando a temperatura da amostra atingiu pela primeira vez o *setPoint* , mais o *controlepwm*, este é definido da mesma maneira em relação ao PID puro somente diferindo os valores das constantes Kd,Kp e Ki.

O primeiro algoritmo se mostrou relativamente instável e o segundo bem estável. Ainda devemos testar mais os dois algoritmos e há de se analisar mais profundamente qual é realmente o mais estável e adequado ao dia a dia do laboratório. No Apêndice C são exibidos a parte principal destes dois algoritmos; e a seguir uma foto de um sistema teste, seguido de um *print screen* da tela do computador durante um teste do segundo algoritmo neste sistema teste:

Figura13: Sistema para teste do programa final

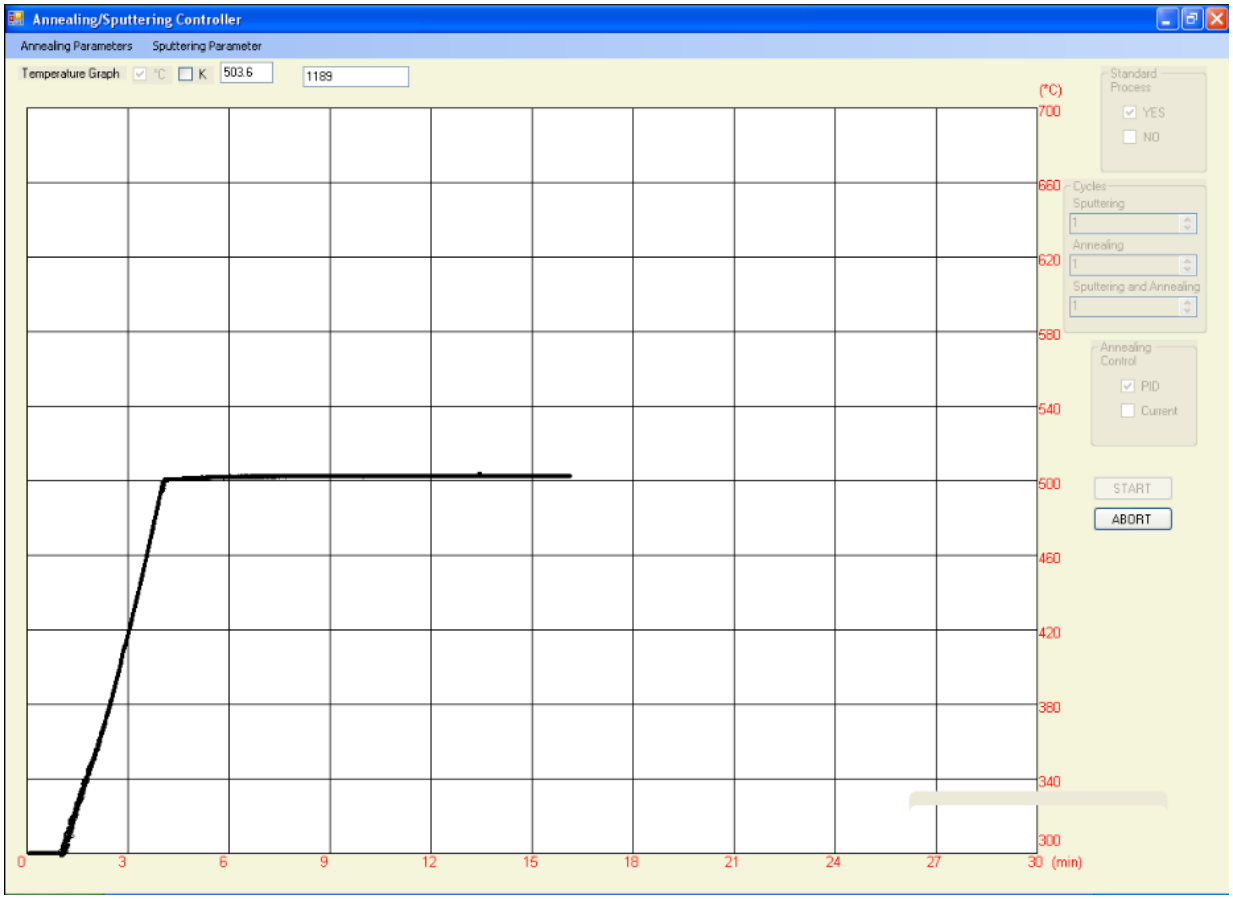


Figura14: Interface gráfica do programa final durante um ciclo de annealing (setPoint=503°C)

Todas as constantes do algoritmo do programa podem ser ajustadas pelo usuário no menu (em azul na fig.14),inclusive as do PID, e uma versão *default* do controle PID também foi criada.Como se pode ver na foto, o sistema teste compreende todos os eletrônicos da automatização que será feita no laboratório, exceto o canhão de íons de argônio e a fonte de alta de tensão, que são os elementos do *sputtering*; uma pequena lâmpada simula o comportamento(temperatura) de uma amostra. Entretanto, como a função da caixa é somente iniciar e parar o *sputtering* ligando e desligando o relê 2 e a saída DTL-0, quando deveria acontecer um *sputtering* só se verificou se a caixa realmente estava no estado mandado verificando com um multímetro as tensões na ponto do fio da caixa.

3-Trabalho de pesquisa científica

Pode ser percebido que este projeto é muito mais amplo que uma simples automação. É de importância crucial para um físico saber criar sua própria intrumentação já que na maioria das vezes, o que está sendo feito tem um caráter pioneiro e inovador e por isso, nunca foi realizado antes. No âmbito da pesquisa científica este tipo de capacitação/aprendizagem, que está além da sala de aula, é muito valorizada já que permite avanços mais rápidos nas próprias pesquisas e os mais diversos sistemas podem ser elaborados pelos próprios pesquisadores ao longo de projetos e possibiltar aperfeiçoamentos talvez inalcançáveis (ou díficeis de se obter por intermédio de auxílio de terceiros,serviços especializados) sem este conhecimento.

Em todo o trabalho realizado ,podemos dizer que foram realizadas pesquisas científicas. Para o desenvolvimento do programa,vários estudos e aprofundamentos foram realizados em busca do melhor desempenho do algoritmo sendo implementado, o que deveria ser usado e de que maneira deveria ser usado para tal ou tal tipo de sistema,por que este sistema se resfria/aquece desta maneira e de que forma isto afetará o controle PID ou PID modificado (valores das constantes no algoritmo principal), este controle funciona para que tipo de sistema,análise de sua estabilidade.O que foi descrito anteriormente pode ser analisado analogamente à um trabaho de pesquisa científica: a realização de um estudo planejado com finalidade de se obter respostas ou resultados para questões e problemas baseado na aplicação do

método científico.

No caso deste projeto, todo um estudo foi sendo desenvolvido com a finalidade de se controlar e optimizar um processo, resolvendo-se o problema de criar o arsenal necessário para isto, sempre em busca do aperfeiçoamento dos resultados através da comprensão dos fenômenos envolvidos. Por exemplo, na busca da obtenção das constantes Kp,Ki e Kd do algoritmo PID a compreensão física do resfriamento e aquecimento de amostras e sistemas: amostras de materiais diferentes se aquecem e resfriam diferentemente, fórmulas famosas descrevem tais comportamentos:

$$(1) Q = m.c.\Delta T;\ \ (2) Q = M.l;\ \ (3) T - T_a = (T_0 - T_a)e^{-kt}$$

foi fundamental; já que não há nenhum método fechado perfeito para isso e saber graças ao conhecimento físico o que de certa maneira esperar é extremamente produtivo e construtivo.O mesmo aconteceu para a criação de algoritmos de PID modificado para rampeamento de temperatura, ressaltando a utilidade do embasamento científico para o bom desenvolvimento do projeto.

Por fim, pudermos perceber o quanto este projeto é rico sob o ponto de vista de pesquisa científica, talvez não da maneira mais tradicional como na busca da compreensão de fenômenos físicos fundamentais, mas no desenvolvimento de um arsenal tecnológico através da própria física.

4-Atividades realizadas

De acordo com o projeto inicial, tinha-se proposto o cronograma abaixo que compreende todos os objetivos deste projeto e o tempo total de sua realização:

Cronograma de atividades para o projeto de iniciação científica I (F590) – Duração de um semestre acadêmico.

1- Estudo da literatura relacionada e familiarização com a instrumentação, o laboratório e com a linguagem *Visual C Sharp(#)*.Programação de pequenas rotinas de leitura/escrita dos equipamentos utilizando as portas de comunicação *RS232*.

2- Construçaõ do programa principal com as rotinas de *sputtering* e *annealing*.

3- Programação do algoritmo PID em *Visual C Sharp (C#)* na rotina de *annealing*. Criação de rampas de aquecimento,temperatura de *annealing*, rampa de refrigeração e de uma interface gráfica para acompanhamento da tempertura.

4- Testes em uma amostra.

5- Trabalho de pesquisa e compreensão dos fenômenos físicos envolvidos em cada processo deste projeto (análises, testes, otimizações,implementações, montagens, aperfeiçoamentos).

Percebe-se que todos os objetivos do projeto inicial foram atingidos, entretanto o programa final de automatização não está completamente finalizado. Precisam ser feitos mais testes para o aperfeiçoamento e escolha dos algoritmos e das constantes, precisa ser revisto e reanalisado o programa inteiro para seu uso definitivo e acrescentados items desejados para o dia a dia do laboratório. Mas ele está basicamente funcionando.

Além disso, naõ foi possível testar o programa numa amostra devido ao fato de que o programa precisa ser analisado minuciosamente antes de entrar em operação permanente; e um sistema semelhante àquele que o programa controlará está sendo preparado para testes antes dos testes no sistema final. Devido ao pouco tempo para o desenvolvimento do projeto, testes não puderam ser feitos em um sistema semelhante ao final; mas um sistema teste mais básico (o último elaborado-fig.13) foi montado para testes de desenvolvimento e ajustagem do programa.

Referências

Livros:

Introduction to PID controllers:theory,tuning and application to frontier areas
C# Programming-Rob Miles' Book
Control system design by Karl Johan Astrom, 2002

Tutoriais , exemplos diversos e imagens:

http://coder-tronics.com/pid-tutorial-c-code-example-pt2/
https://pt.wikipedia.org/wiki/RS-232
https://www.youtube.com/watch?v=txftR4TqKYA
https://gist.github.com/ivanseidel/b1693a3be7bb38ff3b63
https://www.google.com.br/imghp?hl=pt-BR&tab=wi&ei=H_2AWLuvLoekwgTKkrOIBQ&ved=0EKouCBUoAQ

Agradecimentos

Ao meu co-orientador Edson Pedro Cecílio Jr. pelo suporte técnico e grande ajuda ao longo deste projeto. Ao meu orientador, professor Abner de Siervo, pelo importante auxílio e eficientes instruções; e ao professor Richard Landers pelas valiosas sugestões e explições dadas.

Opinião do orientador

"Meu orientador concorda com o expressado neste relatório final e deu a seguinte opinião:

O Sr Jean-Yves é um aluno extremamente dedicado ao projeto, muito inteligente, organizado e sistemático em suas atividades no laboratório.
Realizou todas as etapas de comunicação entre os equipamentos, montagens para testes e os testes em si dos diferentes algoritmos propostos conforme demonstra o seu relatório. O seu projeto de IC atual será fundamental para o desenvolvimento do seu próximo projeto de IC onde irá prepara amostras utilizando a instrumentação aqui desenvolvida. Sem dúvida este trabalho beneficiará inúmeras gerações de estudantes de graduação e pós-graduação e outros usuários do laboratório.
*Apesar de ainda não termos feitos testes finais em uma amostra real (por precaução ao filamento do manipulador da câmara de amostras) não tenho qualquer dúvida que os objetivos iniciais foram completamente atingidos. O aluno está de parabéns pela sua dedicação e êxito no projeto.***"**

Apêndice A – Parte principal do programa de controle PID (puro)

```
bool EVENTO = false; //Definicao do evento para backgroundworker1.

        double PID = 0;
        double error = 0;
        double kP ;
        double kI ;
        double kD ;
        double P = 0;
        double I = 0;
        double D = 0;
        int controlePwm = 0;
        double Temperature;
        double lastTemperature = 25;
        double deltaTempoPID = 0.050;
        int TempoPID = 0;
        double setPoint;

private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs e)
        {
            while (EVENTO == true)
            {
                TempoPID += 50;
                serialPort1.Write("0");
                string TEMPERATURA = serialPort1.ReadLine();
                int CARACTER1 = TEMPERATURA.Length;
                string CARACTER2 = TEMPERATURA.Remove(CARACTER1 - 1);
                int CARACTER3 = Convert.ToInt32(CARACTER2);
                float CARACTER4 = (float)(CARACTER3 * 0.0004887586);
                Temperature = Convert.ToDouble(CARACTER4);
                error = (setPoint - Temperature) / setPoint;
                P = error * kP;
                I += error * kI * deltaTempoPID;
                D = (lastTemperature-Temperature) * kD / deltaTempoPID;
                PID = P + I + D;
                controlePwm =(int)(PID * 2000);
                if (controlePwm > 2500)
                {
                    controlePwm = 2500;
                }
                if (controlePwm < 0)
                {
                    controlePwm = 0;
                }
                FONTE.PS_SetParams((int)(controlePwm), 36000, 10800, 36000, 0);
                lastTemperature = Temperature;
                System.Threading.Thread.Sleep(50);
                backgroundWorker1.ReportProgress(TempoPID);
            }
                if (backgroundWorker1.CancellationPending)
                {
                    e.Cancel = true;
                    backgroundWorker1.ReportProgress(0);
                    return;
                }

        }

        private void backgroundWorker1_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e)
        {
            FONTE.PS_SetParams(0, 0, 10800, 0, 0);
            PID = 0;
            P = 0;
            I = 0;
            D = 0;
```

```
            controlePwm = 0;
            lastTemperature = 25;
            TempoPID = 0;
        }

        private void backgroundWorker1_ProgressChanged(object sender,
ProgressChangedEventArgs e)
        {
            Pen CANETA = new Pen(Color.Black);
            CANETA.Width = 1.5F;
            Graphics IMAGEM = this.CreateGraphics();
            float VARIAVEL1 = (float)((TempoPID * 0.005) + 10);
            float VARIAVEL2 = (float)(690 - (Temperature * 4));
            IMAGEM.DrawEllipse(CANETA, VARIAVEL1, VARIAVEL2,2,2);
            //data
            textBox1.Text = Convert.ToString(Temperature);
            textBox2.Text = Convert.ToString(error);
            textBox3.Text = Convert.ToString(controlePwm);
        }
```

Apêndice B – Funcões do programa de controle da caixa de *sputtering/annealing*

```
        //Definindo a funcao para mandar um byte ESCRITA a placa.
        public void SendByte(int x)
        {   serialPort1.Write(Convert.ToString((Char)123));
            serialPort1.Write(Convert.ToString((Char)x));}
        //Definindo as funcoes para ligar e desligar RL2,D0 e D1.
        public void LigarRL2()
        {   if (Estado == 0){SendByte(2);Estado = 2;}
            if (Estado == 4){SendByte(6);Estado = 6;}
            if (Estado == 8){SendByte(10);Estado = 10;}
            if (Estado == 12){SendByte(14);Estado = 14;}}
        public void DesligarRL2()
        {   if (Estado == 2){SendByte(0);Estado = 0;}
            if (Estado == 6){SendByte(4);Estado = 4;}
            if (Estado == 10){SendByte(8);Estado = 8;}
            if (Estado == 14){SendByte(12);Estado = 12;}}
        public void LigarD0()
        {   if (Estado == 0){SendByte(4);Estado = 4;}
            if (Estado == 2){SendByte(6);Estado = 6;}
            if (Estado == 8){SendByte(12);Estado = 12;}
            if (Estado == 10){SendByte(14);Estado = 14;}}
        public void DesligarD0()
        {   if (Estado == 4){SendByte(0);Estado = 0;}
            if (Estado == 6){SendByte(2);Estado = 2;}
            if (Estado == 12){SendByte(8);Estado = 8;}
            if (Estado == 14){SendByte(10);Estado = 10;}
        }
        public void LigarD1()
        {   if (Estado == 0){SendByte(8); Estado = 8;}
            if (Estado == 2){SendByte(10);Estado = 10;}
            if (Estado == 4){SendByte(12);Estado = 12;}
            if (Estado == 6){SendByte(14);Estado = 14;}}
        public void DesligarD1()
        { if (Estado == 8){SendByte(0);Estado = 0;}
            if (Estado == 10){SendByte(2);Estado = 2;}
            if (Estado == 12){SendByte(4);Estado = 4;}
            if (Estado == 14){SendByte(6);Estado = 6;}}
```

Apêndice C- Parte principal dos dois algoritmos do tipo PID modificado

Primeiro algoritmo:

```
while (EVENT == true)
            {
                TimePID += 50;
                Temperature = GetTemperature();
                if (Temperature >= setPoint) { N += 0.1; }
                if (Temperature >= 300) { M += 0.1; }
                if (Temperature <= 300 && N==0)
                { FONTE.PS_SetParams(((int)TimePID / 10), 36000, 10800, 36000, 0);
```

```
                  if (M==0.1){Current=(int)TimePID / 10)}       }
                if (Temperature >= 300 %% N==0)
                {
                    TimePID += 50;
                    if (N == 0.1) { PIDRiseTime = TimePID; }
                    if (Temperature == setPoint) { N += 0.1; }
                    if (N == 0)
                    {
                setPointVariable = 300 + (TimePID * 0.001);
                error = (setPointVariable - Temperature) / setPointVariable;//Percentual
                        controlePwm += (int)(error * kPRise * 100);
                        if (Current+controlePwm > 2500) Current+controlePwm = 2500; }
                        if (Current+controlePwm < 0) { Current+controlePwm = 0; }
                    FONTE.PS_SetParams((int)( Current+controlePwm),36000,10800, 36000, 0);
                        System.Threading.Thread.Sleep(50);
                        backgroundWorker1.ReportProgress(TimePID);
                      }
                   if (N==0.1) { Current= Current+controlePwm; }
                    if (N != 0 && TimePID < PIDRiseTime + StabilityTime)
                    {

                        error = (setPoint - Temperature) / setPoint;//Percentual
                        P = error * kP;
                        I += error * kI * deltaTimePID;
                        D = (lastTemperature - Temperature) * kD / deltaTimePID;
                        PID = P + I + D;
                        controlePwm = (int)(PID * 1000);
                        if (Current+controlePwm > 2000) { Current+controlePwm = 2000; }
                        if (Current+controlePwm < 0) { Current+controlePwm = 0; }
                      FONTE.PS_SetParams((int)(Current+controlePwm),36000,10800,36000,0);
                        lastTemperature = Temperature;
                        System.Threading.Thread.Sleep(50);
                        backgroundWorker1.ReportProgress(TimePID);
                    }
                    if (TimePID=PIDRiseTime+StabilityTime){Current= Current+controlePwm; }
                    if (TimePID > PIDRiseTime + StabilityTime && N != 0)
                    {
        setPointVariable = setPoint - (TimePID - (PIDRiseTime + StabilityTime)) * 0.003;
                error = (setPointVariable - Temperature) / setPointVariable;//Percentual
                        controlePwm += (int)(error * kPRise * 2500);
                        if (Current+controlePwm > 2500) { Current+controlePwm = 2500; }
                        if (Current+controlePwm < 0) { Current+controlePwm = 0; }
                      FONTE.PS_SetParams((int)(Current+controlePwm),36000,10800,36000, 0);
                        System.Threading.Thread.Sleep(50);
                        backgroundWorker1.ReportProgress(TimePID);
                    }
                }
                if (Temperature < 300 && N != 0)
                { FONTE.PS_SetParams(0, 0, 0, 0, 0); }
            }
            if (backgroundWorker1.CancellationPending)
            { e.Cancel = true; backgroundWorker1.ReportProgress(0); return; }
```

Segundo algoritmo:

```
while (EVENT == true)
            {
                TimePID += 50;
                Temperature = GetTemperature();
                if (Temperature >=setPoint) { N += 0.1; }
                if (N == 0.1) { CurrentRes = (int)(1000+Current2); PIDRiseTime = TimePID; }
                if (N == 0)
                {
                    if (Current1 < 1000)
                    {
                        Current1 += 5;
                        FONTE.PS_SetParams(Current1, 36000, 10800, 36000, 0);
                    }
                    else{setPointVariable=0.05; setPointActual=lastTemperature-Temperature;
                          Current2 +=(int)( setPointVariable/ setPointActual)0.15;
                    FONTE.PS_SetParams((int)(1000+Current2), 36000, 10800, 36000, 0); }
                    System.Threading.Thread.Sleep(50);
                    lastTemperature = Temperature
                    backgroundWorker1.ReportProgress(TimePID);
                }
                if (N != 0 && TimePID <= PIDRiseTime + StabilityTime)
                {
                    error = (setPoint - Temperature) / setPoint;//Percentual
                    P = error * kP;
                    I += error * kI * deltaTimePID;
```

```
            D = (lastTemperature - Temperature) * kD / deltaTimePID;
            PID = P + I + D;
            controlePwm = (int)(PID * 12);
            TotalControl = (int)(controlePwm + CurrentRes);
            if (TotalControl > 2500) { TotalControl = 2500; }
            if (TotalControl < 0) { TotalControl = 0; }
            FONTE.PS_SetParams(TotalControl, 36000, 10800, 36000, 0);
            lastTemperature = Temperature;
            System.Threading.Thread.Sleep(50);
            backgroundWorker1.ReportProgress(TimePID);
        }
        if (TimePID == PIDRiseTime + StabilityTime){CurrentFallI=TotalControl;}
        if (N != 0 && TimePID > PIDRiseTime + StabilityTime)
        {
            if (Temperature>300)
            {
            setPointVariable= 0.05; setPointActual=Temperature-lastTemperature;
            Current3 +=(int)( setPointVariable/ setPointActual)0.15;
            FONTE.PS_SetParams((int)(CurrentFallI-Current3),36000, 10800,36000,0);
            System.Threading.Thread.Sleep(50);
            lastTemperature = Temperature
            backgroundWorker1.ReportProgress(TimePID);
            }
            if (Temperature<=300){
            Current4+=5
            FONTE.PS_SetParams((int)(1000-Current4), 36000, 10800, 36000, 0); }
            System.Threading.Thread.Sleep(50);
            }
    }
    if (backgroundWorker1.CancellationPending)
    {
        e.Cancel = true;
        backgroundWorker1.ReportProgress(0);
        return;
    }
```