

DEPARTAMENTO DE

ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

Desenvolvimento de Controlador Avançado para Sistema Embebido

FACULDADE DE ENGENHARIA

UNIVERSIDADE DO PORTO

Rua dos Bragas, 4099 Porto Codex – PORTUGAL

Desenvolvimento de Controlador Avançado para Sistema Embebido

FACULDADE DE ENGENHARIA DA
UNIVERSIDADE DO PORTO

DESENVOLVIMENTO DE CONTROLADOR AVANÇADO PARA SISTEMA EMBEBIDO

004 (043) SOU/DES
UNIVERSIDADE DO PORTO
Faculdade de Engenharia
BIBLIOTECA
N.º 6388
COD 004(043)
Data ___ ___ ___ ___

Armando Jorge Miranda de Sousa

Lic. Eng. Electrotécnica e de Computadores pela
Faculdade de Engenharia da Universidade do Porto

Dissertação submetida para satisfação parcial dos
requisitos do grau de Mestre em Engenharia
Electrotécnica e de Computadores
(Especialidade de Informática Industrial)
Financiado pelo Programa Praxis XXI (BM2316/94)

Porto, Setembro 1996

«(...) Um documento em, suponhamos, controlo por Lógica Imprecisa, pode ser escrito numa linguagem tão acessível quanto possível (um óptima prática cientifica) ou com tanto simbolismo quanto possível (quanto maior o número de letras gregas, mais impacto científico). (...) Se o autor tiver alguma coisa de substância para dizer, então é provável que ele não o queira ofuscar com excessivo simbolismo»

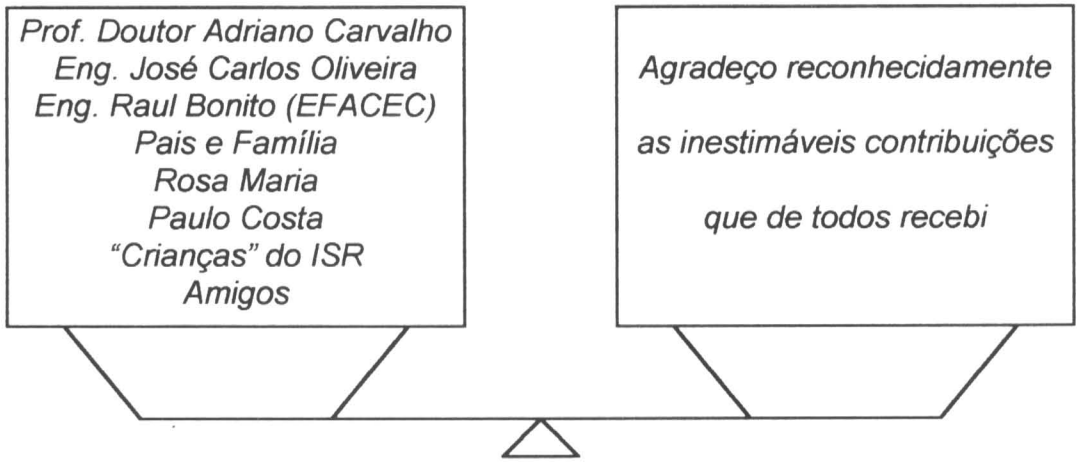
E.H. Mamdani (1993)

Prof. Doutor Adriano Carvalho
Eng. José Carlos Oliveira
Eng. Raul Bonito (EFACEC)
Pais e Família
Rosa Maria
Paulo Costa
"Crianças" do ISR
Amigos

Agradeço reconhecidamente

as inestimáveis contribuições

que de todos recebi



Sumário

Os sistemas embebidos são sistemas fechados, que, por vezes apresentam fortes interacções com componentes mecânicos, podem incluir hardware específico à aplicação e onde o software está sujeito a restrições temporais no âmbito do Tempo Real. O projecto deste tipo de sistemas deve, portanto, ser especialmente cuidado. Devem ser utilizados métodos estruturados para permitir a correcta análise dos requisitos levantados pela aplicação, com o cuidado de separar correctamente análise e implementação. A documentação gerada deverá servir não só para permitir a futura manutenção do sistema mas também para permitir re-analisar rapidamente qualquer nova questão introduzida, p. ex. por dificuldades de implementação. Como este tipo de sistemas é fechado e geralmente de difícil acesso, poderá também ser apropriado prever estruturas próprias para facilitar a procura e eliminação de erros, tanto a nível de hardware como a nível de software.

As aplicações industriais dos sistemas embebidos a nível do controlo de processos são bem conhecidas. Recentemente, tem aparecido várias técnicas de controlo que procuram resolver o controlo de processos industriais com características não lineares e/ou onde a modelização matemática não é, na prática, aplicável. Neste contexto surge a inteligência artificial a preconizar uma simulação não numericamente exacta mas que pretende incorporar o conhecimento existente acerca do processo. Este tipo de simulações pode, depois, ser utilizado para o desenho do controlador na área da I.A.. Dentro desta área destaca-se a metodologia da Lógica Imprecisa (*Fuzzy Logic*), por permitir o controlo inteligente de processos com elevada qualidade e robustez, mantendo ainda assim grande simplicidade e baixo tempo de instalação. De facto, controladores deste tipo são viáveis em ambientes com baixos recursos computacionais e é possível tirar partido das vantagens deste tipo de controlo mesmo sem a modificação de hardware.

Neste trabalho desenvolveu-se um controlador embebido inteligente para o processo industrial de secagem de transformadores. Utilizou-se a Lógica Imprecisa para melhorar o anterior controlador convencional, resultando em benefícios tanto a nível de regime dinâmico como a nível de regime final. Adicionalmente, conseguiu optimizar-se a implantação de código de tal forma que não foi necessária qualquer modificação de hardware.

Palavras Chave

- Sistemas Embebidos
- Tempo Real
- Controlo
- Controlo Robusto
- Controlo Avançado
- Controlo Inteligente
- Inteligência Artificial
- *Fuzzy Logic*
- Lógica Imprecisa
- Lógica Difusa

Lista de Abreviaturas

μ - (μC ou μP)

μC - Microcontrolador

μP - Microprocessador

AGV - *Auto Guided Vehicle*

AP - Autómato Programável

ASIC - *Application Specific Integrated Circuit*

CAD - *Computer Integrated Manufacturing*

CAE - *Computer Aided Engineering*

CAM - *Computer Aided Manufacturing*

CIM - *Computer Integrated Manufacturing*

CIME - CIM *and Engineering*

CISC - *Complex Instruction Set Computer*

CNMA - *Communications Networks for Manufacturing Applications*

CoPSE - Controlo do Processo de Secagem e Enchimento

CPU - *Central Processing Unit*

DDE - *Dynamic Data Exchange*

DFT - *Discrete Fourier Transform*

DOS - *Disk Operating System*

DRAM - *Dynamic* RAM

DSP - *Digital Signal Processor*

E/S - Entrada/Saída

EEPROM ou E2PROM - *Electrically EPROM*

EMC - *Electro Magnetic Compatibility*

EPROM - *Erasable Programmable ROM*

FEUP - Faculdade de Engenharia da Universidade do Porto

FFT - *Fast Fourier Transform*

FPGA- *Field Programmable Gate Array*

GA - *Genetic Algorithm*

HDL - *Hardware Description Language*

I.A. - Inteligência Artificial

ICE - *In-Circuit Emulator*

IGBY - *Insulated Gate Bipolar Transistor*

IPC - *Inter-Process Communication*

ISA - *Industry Standard Architecture*

LASER - *Light Amplification by Stimulated Emission of Radiation*

LCD - *Liquid Crystal Display*

LED - *Light Emitting Diode*

MAP - *Manufacturing Automation Protocol*

MLI - Modulação da Largura dos Impulsos

MMS - *Manufacturing Message Specification*

MTBR - *Mean Time Between Failures*

MTTR - *Mean Time to Repair*

OSI - *Open Systems Interconnection*

OTP - *One Time Programmable*

PAL - *Programmable Array of Logic*

PC - *Personal Computer*

PCB - *Printed Circuit Board*

PWM - *Pulse Width Modulation*

RAM - *Random Access Memory*

RISC - *Reduced Instruction Set Computer*

ROM - *Read Only Memory*

S.E. - Sistema Embebido

S.O. - Sistema Operativo

SISC - *Specific Instruction Set*

SP - *Stack Pointer*

T.R. Tempo Real

VHDL - Virtual HDL

VLSI - *Very Large Scale Integration*

Índice Genérico

Sumário iii
Palavras Chave iv
Lista de Abreviaturas v
Índice Genérico vi
Parte I - Sistemas Embebidos e Tempo Real 1
1. Introdução 2
1.1 Contexto 2
1.2 Motivação 4
1.3 Metodologia 5
2. Sistemas Embebidos 6
2.1 Introdução 6
2.2 Controlo Digital 7
2.3 Tempo Real 9
2.4 Controlo Distribuído 14
2.5 Resumo 17
3. Engenharia embebida 18
3.1 Mecatrónica 18
3.2 Projecto 19
3.3 Software 25
3.4 Hardware 31
3.5 Resumo 54
Parte II - Experimentação e Controlo 55
4. Apresentação do Caso Prático 56
4.1 O processo 56
4.2 Controlo Convencional 60
4.3 Aquisição de dados - a envolvente fabril 65
4.4 Objectivo do Trabalho Prático 69
4.5 Recursos computacionais disponíveis 72
4.6 Compressão do código já existente 73
4.7 Resumo 74
5. Métodos de Controlo Avançado 75
5.1 Utilização de tabela de parâmetros PI(D) 75
5.2 Aquisição dos parâmetros do sistema on-line 76
5.3 Controlo por modo de deslizamento 78
5.4 Filtros de Kalman 79
5.5 Resumo 80

6. Controlo Inteligente **81**
6.1 Introdução ao controlo inteligente *81*
6.2 Metodologias de controlo Inteligente *81*
6.3 Sistemas Periciais *82*
6.4 Redes Neuronais *83*
6.5 Algoritmos Genéticos *85*
6.6 Solução: Fuzzy Logic *86*
6.7 Resumo *94*
7. Simulação do CoPSE **95**
7.1 Identificação *95*
7.2 Modelo *95*
7.3 Ajustes *99*
7.4 Resumo *102*
8. Implementação **103**
8.1 Introdução à Lógica de Controlo *104*
8.2 Regras de Inferência *110*
8.3 Tabela vs. Matemática Imprecisa *113*
8.4 Simulação, Programação e Resultados *115*
8.5 Estabilidade *137*
8.6 Análise de Resultados *138*
8.7 Resumo *139*
9. Conclusões **141**
9.1 Trabalho Desenvolvido *141*
9.2 Trabalho Futuro *142*
Bibliografia **144**
Índices **150**
Índice Pormenorizado *150*
Índice de Figuras *153*
Índice de Tabelas *155*
Anexo A - Comparação de Desempenho em Controlo por Lógica Imprecisa **A1**

PARTE I

Sistemas Embebidos

e

Tempo Real

1. Introdução

1.1 Contexto

De acordo com Bennet (1994, p. 12), um ***computador embebido*** está intimamente ligado com o seu ambiente de onde recebe e para onde envia estímulos variados. Os sistemas embebidos são discretos e quase passam despercebidos, bem arrumados dentro de máquinas complexas, utilizadas no dia a dia. Neste tipo de sistemas o objectivo último não é a computação pura. No dia a dia, profissional ou não, utiliza-se frequentemente sistemas e processadores embebidos a controlar processos, p. ex. a quotidiana máquina de lavar loiça. Ganssle (1995a) afirma que existem cerca de 40 aplicações embebidas numa casa média.

Com o crescente nível de automatização de equipamentos, em casa, no escritório ou na indústria, aparece a necessidade de controlar de forma eficiente os processos do quotidiano.

A ***Engenharia do Controlo*** é o ramo da engenharia que se dedica ao estudo do comportamento de processos industriais, tendo por objectivo melhorar o desempenho dos equipamentos e/ou processos. Um dos métodos mais utilizados é a introdução de malhas de ***realimentação*** e respectivo ***controlador***. A introdução desta filosofia de controlo implica medições de parâmetros reais (de saída) do processo para garantir que os objectivos do controlo são cumpridos tão bem quanto possível.

O ***Controlo por Computador*** é muitas vezes o modo de implementação prática da Engenharia do Controlo, mais abrangente porque se ocupa com o funcionamento quer do equipamento quer do processo: desde o modo de arranque, ao funcionamento normal, um eventual modo de paragem, actuação em caso de falha (levar o sistema para um funcionamento ou paragem seguro), etc.. De acordo com Åström (1990, p.2) o ***controlo de processos*** utilizando computadores remonta à segunda metade dos anos 50: nessa altura os computadores davam instruções aos operadores humanos ou regulavam malhas de controlo analógicas. O ***controlo digital***, com malhas de realimentação totalmente digitais aparece apenas em torno de 1962, como substituição de malhas analógicas. A realimentação totalmente digital significa que o computador lê directamente os ***sensores*** e

altera o decorrer do processo comandando os ***actuadores***, eventualmente utilizando cadeias de conversão entre os dispositivos externos e o computador.

O avanço da engenharia do controlo e do controlo por computador, acompanhado pelo crescente poder de cálculo disponível, permite gerir sistemas cada vez mais complexos, sempre com um crescente nível de fiabilidade, inteligência e autonomia. As características dos sistema convencionais de controlo e respectivas implementações reais são discutidas por vários autores (Hanselmann 1987, Åström 1990, Bennet 1994) e são, actualmente, bastante divulgadas.

No passado, a indústria recorria intensivamente à supervisão humana de muitos processos por falta de sistemas automáticos de controlo apropriados: por ex. o dosear da alimentação de carvão nas locomotivas ferroviárias ou nos barcos a vapor. Num controlo deste tipo, o operador utiliza os seus conhecimentos e a leitura de vários sensores para decidir como deve reagir de maneira a conduzir o processo até ao ponto de funcionamento pretendido.

Actualmente, os avançados mísseis inteligentes incluem a bordo sistemas computadorizados, que se destinam a manobrar o sistema por forma a atingir os seus objectivos. Grande parte das máquinas industriais recentes utiliza algum tipo de processador para gerir, controlar e/ou supervisionar o seu funcionamento. Mesmo nas nossas casas os electrodomésticos, os sistemas de alta fidelidade, os telefones portáteis, etc. utilizam alguma forma de microprocessador para gerir o sistema onde estão inseridos. A todos estes sistemas chamamos ***sistemas embebidos***, S.E. *(Embedded Systems)*. Estes processadores gerem os limites do sistema, as interfaces com o exterior e o seu funcionamento interno.

O sistema embebido deve fazer a ***supervisão*** de todos os mecanismos a ele ligados, por forma a tirar conclusões do estado actual do sistema, tendo em vista a melhor acção de comando possível, *i. e.* aquela que mais rapidamente leve o sistema ao funcionamento pretendido, mantendo especial atenção aos possíveis limites de perigo do sistema.

A noção do tempo é indissociável dos sistemas embebidos, pois neste género de sistemas existem sempre requisitos temporais, evidentes no caso do sistema de comando de um míssil auto-guiado, mas também presente em qualquer outro sistema embebido. Em termos simples, ao conceito da existência de restrições temporais chamamos ***tempo real***, que pretende dizer que os resultados devem ser gerados sempre a par da realidade, tal como referido por Bennet (1994, secção 1.3).

Com o aparecimento de grandes máquinas sofisticadas, surgem novas técnicas de controlo baseadas na interligação de vários sistemas de controlo e assim nasce o ***controlo distribuído***, (Bennet, 1994, secção 2.7), levantando problemas novos a nível de fiabilidade e robustez de funcionamento.

Nesta saga de sofisticação surge a necessidade de optimizar o desempenho dos sistemas complexos onde., ou não é possível ou não é desejável (rentável) obter uma formulação matemática adequada, pelo que se desenvolveram novos métodos de controlo, designados por ***controlo robusto***. Por exemplo, no ***controlo adaptativo***, o sistema de controlo de alguma forma adquire o estado do processo e tende a adaptar-se de modo a que ele próprio seja afinado automaticamente para que a acção de controlo seja a mais apropriada para o sistema tal como ele é no presente. Naturalmente, tal sofisticação será necessária só se o processo mudar de características de forma apreciável (Åström *et al*, 1971, Åström, *et al* 1990).

Leitch (1992) refere que o aparecimento do ***controlo por inteligência artificial*** fornece ao engenheiro de controlo uma importante extensão no seu ramo de trabalho ao permitir a representação do pensamento humano em sistemas sob comando de um computador.

É precisamente nesta interessante área que se insere o presente trabalho.

1.2 Motivação

No contexto anteriormente referido, interessa aprofundar o relacionamento entre os sistemas embebidos e os mais actuais sistemas de controlo. As aplicações para controlo avançado em sistemas embebidos são visíveis tanto no controlo de processos industriais como em sistemas para o grande público. No âmbito do controlo avançado surgem com especial destaque as técnicas baseadas em Inteligência Artificial (I.A.).

De acordo com Kiendl *et al* (1994), os sistemas embebidos baseados em inteligência artificial atingem já muitos sistemas acessíveis ao grande público tal como sistemas automáticos de caixa de velocidade nos automóveis, suspensões activas, etc.. Os mesmos autores avançam que outras aplicações aparecerão brevemente, tanto a nível industrial em situações de controlo não linear como a nível de produtos para o consumidor.

A intenção deste documento é fornecer um guia para o correcto desenvolvimento de sistemas embebidos e respectivos controladores, na área onde o trabalho foi realizado, a lógica imprecisa (*Fuzzy Logic*), um método no âmbito da Inteligência Artificial. Pretende-se

ainda descrever o trabalho realizado no desenvolvimento de uma peculiar aplicação na área do controlo de processos industriais.

1.3 Metodologia

A engenharia do controlo é uma área geralmente considerada complexa, que exige o domínio de muitas tecnologias de suporte: por um lado, exige o conhecimento profundo do processo em causa, seus requisitos temporais, consequências das várias acções de controlo possíveis, etc.; por outro, exige o domínio de todos os componentes do sistema, seus limites, seu funcionamento seguro, tempos de resposta e requisitos temporais, etc..

Este trabalho pode ser entendido como estando dividido em duas partes. A primeira parte engloba os conceitos relativos a sistemas embebidos e os conceitos relacionados com o tempo real. Esta primeira parte engloba os capítulos 2, 3, onde se faz um compilação orientadora dos vários aspectos relacionados com o trabalho em sistemas embebidos.

O capítulo 2 fala das definições, classificações, objectivos, etc. enquanto que no capítulo seguinte se fala da metodologia de desenvolvimento, perante restrições do tipo tempo real.

Pretende-se valorizar o aspecto didáctico deste documento procurando, em especial, sensibilizar o leitor para a necessidade da correcta especificação do desenho de um sistema embebido, sem esquecer a produção de documentação apropriada. A falta destes elementos trará, de futuro, dificuldades e custos acrescidos para realizar mudanças, problema especialmente importante tendo em conta que os S.E. tendem a ser fechados e limitados.

A segunda parte deste documento pretende cobrir conjuntamente o controlo e a parte experimental, realizada sobre um sistema embebido industrial. Estas duas componentes são estudadas em conjunto porque não se pretende alcançar todos os métodos de controlo actualmente existentes, os método estudados fazem sentido perante a sua aplicabilidade para o processo em vista. Não se pretende ser exaustivo num campo tão complexo e prolífero em produções científicas muito recentes como o controlo de processos.

O trabalho prático efectuado demonstra alguns dos problemas de uma aplicação real de controlo. A solução proposta situa-se na área da inteligência artificial, essa a escolha será justificada perante a teoria e os resultados experimentais obtidos.

Finalmente, apresentam-se as principais conclusões e várias linhas de acção para trabalhos futuros, na sequência deste.

2. Sistemas Embebidos

2.1 Introdução

De acordo com **Brown (1994, pág. 1)**, os sistemas embebidos (S. E.) aparecem frequentemente como caixas negras (sistemas fechados), associadas a equipamentos, contendo processadores dedicados principalmente ao controlo do equipamento de que fazem parte integrante. **Levi *et al* (1990 pág. 2)** refere que uma característica própria dos S.E. é que neste tipo de sistemas, não ocorre desenvolvimento de software.

Os S.E. utilizam tipicamente software residente nalgum tipo de ROM[1] (EPROM[2], EEPROM[3], *Flash* EPROM[4], OTP[5], etc.) e utiliza algum do espaço de endereçamento para leitura de Entradas e accionamento de Saídas (E/S).

A falha de um sistema embebido acarreta a falha do equipamento a que está associado, pelo que estão sempre associados a estes sistemas especiais cuidados de robustez, durabilidade, capacidades de recuperação de erros, gestão do relacionamento com os limites do sistema, etc..

Em geral, os S.E. necessitam de hardware próprio e o desenvolvimento de hardware e software é muitas vezes feito pelo menos parcialmente em simultâneo. Como tal, durante a fase de teste e eliminação de erros, é muitas vezes difícil distinguir se um certo erro é devido a uma deficiência de hardware ou de software.

Num sistema embebido, software, hardware e aparência exterior são algo diferentes do PC de secretária (*desktop PC)* normal que é vulgar ter em casa. Na realidade, poucos sistemas embebidos são baseados em sistemas operativos que estão no *top* de vendas, poucos são baseados nos processadores mais conhecidos do público em geral e poucos têm monitor e teclado, pelo menos na configuração que é comum para o utilizador médio de um PC de mesa. O tempo é um factor que está sempre presente num sistema embebido, p. ex.

[1] *Read Only Memory* - Programada na fábrica uma única vez, pode ser lida muitas vezes

[2] *Erasable Programmable* ROM - Programada em aparelhos especiais, só pode ser apagada toda duma vez por exposição a raios Ultra Violetas

[3] *Electrically* EPROM - Também designada de E2PROM, pode ser reprogramada através de impulsos eléctricos. É algo lenta e tem uma capacidade de reprogramação limitada. Pode existir integrada em microcontroladores

[4] *Flash* EPROM - Estes dispositivos podem ser apagados por impulsos eléctricos, na sua totalidade ou em blocos. São mais rápidos e permitem mais reprogramações que as EEPROM's

[5] *One Time Programmable* - Uma EPROM sem janela de UV, logo só pode ser programada uma vez (é caro inserir a janela de UV no processo construtivo dos integrados)

ninguém compraria um simples leitor de Discos Compactos (*Compact Disk*) que demorasse 10 minutos a passar de uma faixa musical para outra: isso seria indicativo de que o S. E. que posiciona o LASER sobre o disco e faz a descodificação da informação é demasiado lento.

Em resumo, um sistema embebido é parte integrante de um sistema mais vasto, forma um sistema computacional com requisitos especiais, com um forte relacionamento entre hardware e software, inserido dentro de um sistema fechado com limites bem estabelecidos com o exterior. Muitas vezes implementam algoritmos de engenharia de controlo e têm sempre requisitos temporais.

Procura-se hoje em dia que nenhum sistema seja completamente fechado ao exterior e também começam a aparecer alguns sistemas embebidos que comunicam com o exterior, mesmo sob o perigo de os tornar susceptíveis a avarias pelo facto de comunicarem. Este assunto será desenvolvido em "2.4.1 Sistemas Distribuídos", pág. 14.

2.2 Controlo Digital

O Controlo Digital ou Controlo por Computador é a função desempenhada por muitos sistemas embebidos. Muitos dos sistemas embebidos industriais são desenhados para o controlo de processos físicos, químicos, termodinâmicos, eléctricos, etc., que precisam, por motivos variados, de ser mantidos na zona desempenho óptimo, entre outros objectivos possíveis.

Segundo **Bennet (1994, pág. 24)** o controlo por computador incluirá:

- Aquisição de dados
- Sequenciação de operações
- Malha de realimentação
- Supervisão
- Análise de dados
- Armazenamento de dados
- Interface com o utilizador[1]

[1] Também conhecido pelos acrónimos anglo-saxónicos: *Human Computer Interface* (HCI) e *Man Machine Interface* (MMI)

O mesmo autor assegura que os objectivos incluirão:

- Eficiência
- Facilidade de operação[1]
- Segurança[2]
- Melhoria da qualidade da produção
- Redução do desperdício
- Reduzido impacto ambiental
- Redução da carga de trabalho directamente associada ao processo

O controlo digital inclui a monitorização de vários tipos de sinais provenientes de sensores e a criação de sinais para comando do processo através de actuadores.

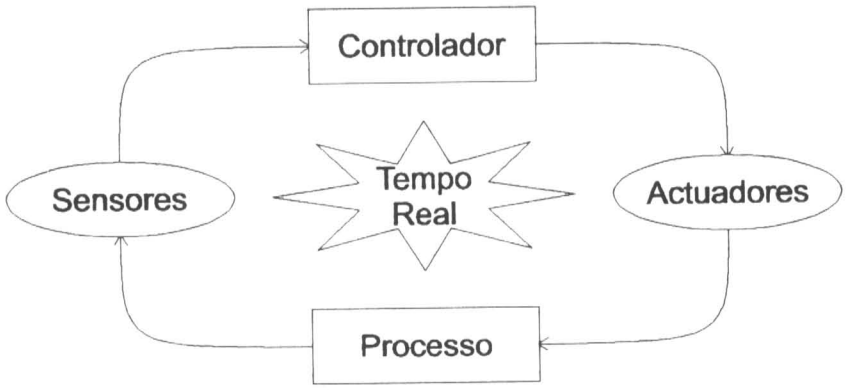


Figura 1: Diagrama de blocos para o controlo por computador (na presença de restrições de tempo real)

Quando o controlo por computador inclui uma malha de realimentação totalmente digital, o que acontece frequentemente, então o diagrama de blocos é o apresentado na Figura 1. Nesta figura o "Controlador" é alguma forma de controlador digital ou, por outras palavras, alguma forma de computador. O bloco "Sensores" representa toda a cadeia de aquisição e condicionamento de sinal necessário para que o "Controlador" consiga medir o que acontece no "Processo". Esta cadeia pode incluir conversores Analógicos Digitais e toda a electrónica necessária. Também o bloco "Actuadores" pode necessitar de electrónica externa, p. ex. para realizar uma conversão Digital Analógica para a entrada de actuadores lineares.

[1] Inclui a inteligência do sistema, seu grau de autonomia e também a facilidade de utilização da interface HCI

[2] Tanto do processo em si como dos utilizadores

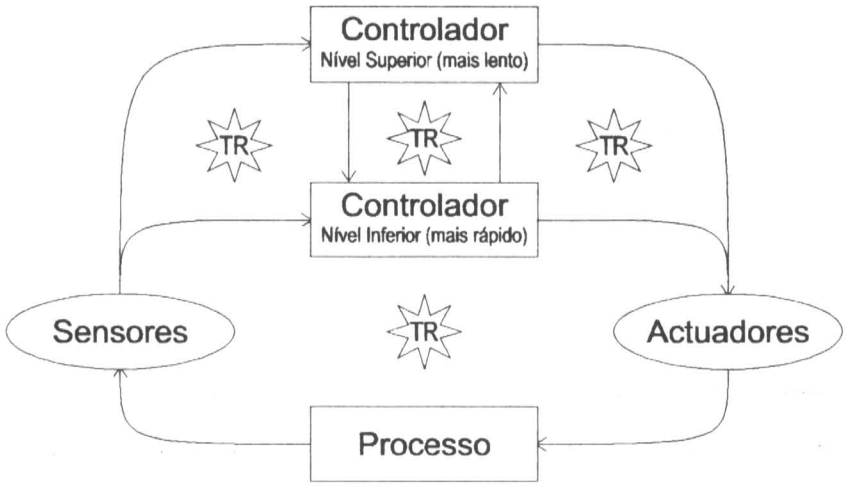


Figura 2: Diagrama de blocos para o controlo hierárquico
(na presença de restrições de tempo real)

Tal como se pode entender pela Figura 2, existe a possibilidade de 2 malhas de controlo conviverem, tornado-se assim o controlo hierárquico. Uma muito rápida, autónoma, de nível hierárquico inferior e outra, mais lenta, que passa pelo controlador mais complexo, que ajustará os parâmetros do outro controlador a um ritmo mais baixo (e/ou actua directamente sobre o processo). O controlador hierarquicamente inferior poderá ser totalmente analógico com o objectivo de ser extremamente rápido. No entanto, ambas as malhas podem ser digitais e até partilhar o mesmo processador, mantendo, no entanto, a malha mais interior uma dinâmica superior à outra. Este tipo de arquitecturas hierárquicas são muito comuns em processos de tipo contínuo, em que o processo tem durações de muitos meses ou mesmo anos e onde pequenos ajustes existem mas são feitos sem parar o processo, daí as vantagens de descentralizar o processo (**Bennet 1994**). Este tipo de estrutura pode ser desenvolvido repetidas vezes em malhas cada vez mais interiores e mais rápidas (hierarquicamente inferiores). Em teoria, é também possível que duas malhas de controlo se cruzem, o que retira o cariz hierárquico do sistema e introduz problemas complexos no tanto no âmbito do controlo como no da Teoria dos Sistemas .

Perante toda esta problemática existe sempre um denominador comum: o Tempo.

2.3 Tempo Real

Existem várias maneiras de definir o conceito do Tempo Real (T.R.), indissociável de qualquer sistema de controlo e qualquer sistema embebido.

Bennet (1994, pp. 10-12) refere algumas definições apropriadas:

- Um sistema T.R. é aquele onde o instante de tempo em que o resultado (certo) é gerado, é relevante, pois existem restrições no tempo de resposta
- Sistemas T.R. devem produzir respostas correctas dentro de determinados períodos de tempo. Se tal não acontecer, ocorrerá degradação do desempenho do sistema e/ou funcionamento erróneo. Este tipo de situação deve também ser previsto.
- Um sistema de tempo real lê e comanda entradas e saídas em limites de tempo impostos externamente, não à medida que o sistema gera os resultados
- Um programa onde o funcionamento correcto implica tanto o resultado lógico da computação efectuada como também o tempo em que os resultados foram produzidos

Tal como referido por todas as definições, um resultado só é correcto se for calculado em tempo útil, doutra maneira talvez os dados em que os cálculos se baseiam já tenham alterado significativamente ou talvez já tenha já tenha ocorrido alguma consequência indesejável porque as respostas não chegaram a tempo. Convém realçar que nenhuma definição de Tempo Real refere exactamente o período de tempo aceitável para a produção de resultados, porque naturalmente, depende do processo em causa. Poderá tratar-se da mais pequena fracção de segundo para o sistema de comando de um míssil inteligente, até ... muitas horas, para processos longos tal como a anodização de peças metálicas[1].

O tempo real é um conceito aplicável a muitos sistemas. Imaginemos um sistema que consegue prever se existirá um sismo nas próximas 24 horas. Esse sistema só será útil se o resultado for gerado significativamente antes de decorrerem as 24 horas. Outro exemplo: o sistema de piloto automático de um avião deve gerar respostas de importância crucial em pequeníssimos tempos sob a ameaça de o avião (começar a) cair porque a resposta demorou demasiado a ser calculada.

Stankovic (1988), no artigo que parece ter introduzido o conceito de computação em tempo real, refere claramente que é preferível um sistema lento mas determinístico[2] a um outro rápido, mas onde os tempos de execução são muito variáveis. O último tipo de sistemas pode eventualmente não respeitar as restrições impostas por algumas vezes, o que pode ser penalizador para sistema. Será, portanto, mais útil para a computação em tempo

[1] Processo de cobertura da superfície exposta de um objecto electricamente condutor por deposição electro-química de uma camada extremamente fina de um óxido electricamente isolador, processo inverso do que ocorre quando uma bateria eléctrica funciona como gerador

[2] Onde é possível determinar o tempo máximo de execução

pode ser penalizador para sistema. Será, portanto, mais útil para a computação em tempo real o sistema um pouco mais lento, onde se pode garantir os tempos máximos de execução no pior caso e assim garantir que as restrições temporais são sempre cumpridas.

2.3.1 Classificação de Estímulos T. R.

2.3.1.1 Cíclicos

Bennet (1994, secção 1.4.1) refere que se o desempenho computacional do sistema for suficiente para permitir que os fenómenos exteriores, tendo uma constante de tempo bem conhecida, sejam amostrados em tempos máximos suficientes, então a sincronização de cada tarefa deverá ser feita em ciclo, por ex. recorrendo ao relógio do dia para correr processos em intervalos regulares de tempo. Este tipo de estímulos são designados por ***cíclicos*** e aparecem frequentemente em sistemas T.R. e S.E. em particular.

A especificação de sistemas deste tipo aparece geralmente sob a forma de "correr o processo N de T em T [tempo]".

2.3.1.2 Aperiódicos

O mesmo autor refere ainda que outras acções deverão ser despoletadas como resposta a estímulos externos imprevisíveis. A resposta a este tipo de estímulos deve ser baseado na ocorrência desse evento (*event based*) e geralmente são utilizadas as interrupções do microprocessador para lançar rotinas apropriadas ao tratamento deste tipo de acontecimentos, designados por ***aperiódicos***.

A especificação destes sistemas relaciona-se com tempos máximos de inicio do processo de atendimento (***tempo de latência***) ou pode ser expressa em termos de tempo máximo para o fim dessa tarefa (fim do tratamento) ou pode ser uma combinação dos dois (Ganssle 1995b).

2.3.1.3 Interactivos

Este tipo de sistemas apresentam restrições do tipo estatístico, por exemplo "tempo médio de resposta menor que X segundos" (Bennet, 1994, secção 1.4.3). São sistemas destinados geralmente a comunicações com seres humanos (lentos!), que podem esperar um pouco mais pelas respostas sem grande prejuízo, isto porque os dados em que se baseiam não mudam com o atraso na resposta. Poderá no entanto haver restrições a nível estatístico, p. ex. média e desvio padrão nos tempos de atendimento.

2.3.2 *Hard Real Time* e *Soft Real Time*

Os Sistemas T.R. devem cumprir limites temporais, mas convém conhecer a penalização pela incapacidade de cumprir essas restrições. Haverá alguma margem de manobra? Se faltar pouco para terminar processo actualmente em execução, dever-se-á, mesmo assim, perder o tempo necessário para o interromper e processar o novo pedido? Tal como referido por Åström (1990, pág. 499), estas perguntas são de resposta complicada e exigem tratamento específico em torno de serviços disponibilizados pelo sistema operativo (S.O.). Este assunto é aprofundado, entre outros autores, por Milenkovic (1992), Bennet (1994, capítulo 6), Kopetz (1991), Tindell (1993) e Levi *et al* (1990, Parte IIII). O tratamento específico deste assunto deve incluir as várias estratégias de comutação de processos, características desejáveis de previsibilidade, como obter medidas dos tempos característicos das tarefas, estratégias de comutação de tarefas que conduzam a garantias temporais, etc..

A questão central é, portanto, qual a penalização devido ao não cumprimento das metas temporais (*deadlines*) ?

No caso de um sistema de navegação de avião, um pequeno atraso para avisar que um embate está iminente será catastrófico... este tipo de sistemas são sistemas ***hard real time***. Ver Figura 3 - *Hard R.T. System* 1. Existem sistemas mais penalizadores que outros, por exemplo um sistema de aquecimento de solda, que deve manter a solda a certa temperatura aceitaria pequenos desvios aos limites temporais pretendidos mas com penalizações a crescerem rapidamente com o tempo em excesso, tal como representado no sistema *Hard Real Time* 2 da Figura 3. No caso das máquina de levantar dinheiro, conhecidas por ATM (*Automatic Teller Machine*), de um pequeno atraso não resultará nada de grave ... evidentemente, se o resultado demorar muito mais que o limite temporal, então a penalização será elevada: o sistema deixa pura e simplesmente de respeitar as restrições de tempo real. Estes sistemas são classificados como ***soft real time***. (ver Figura 3).

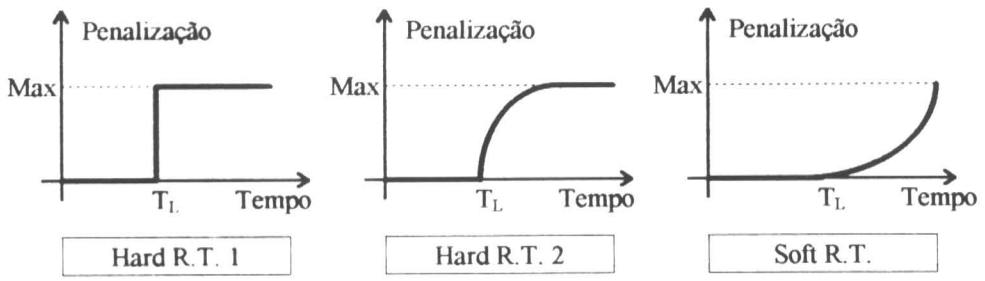


Figura 3: Aspecto gráfico das penalizações por incumprimento das metas temporais T_L - Tempo Limite

Segundo **Bennet (92, pág. 17)** as restrições temporais podem formalizar-se de acordo com a Tabela 1.

Tabela 1 - Formalização das restrições temporais para os vários tipos de sistemas Tempo Real

Sistemas *Hard Real Time*		Sistemas *Soft Real Time*	
Periódico (cíclico)	Aperiódico (eventos)	Periódico (cíclico)	Aperiódico (eventos)
$t_C(i) = t_S \pm a$	$t_E(i) \leq t_L$	$\frac{1}{n}\sum_{i=1}^{n} t_C(i) = t_S \pm a$ $n = T/t_S$	$\frac{1}{n}\sum_{i=1}^{n} t_E(i) \leq t_a$ $n = T/t_S$

$t_C(i)$ - Intervalo entre o ciclo i e o ciclo i-1

t_S - Tempo máximo desejado para o ciclo (pode representar um frequência de amostragem)

$t_E(i)$ - Tempo de resposta (tempo de latência) à ocorrência n.º i do evento E

t_L - Tempo limite de resposta

t_a- Tempo médio de resposta permitido para o evento E durante um certo intervalo T

n - número de ocorrências do evento e dentro do intervalo T ou número de repetições cíclicas dentro do intervalo temporal T

a - pequena tolerância temporal

Esta formalização atribui uma pequena tolerância a acontecimentos cíclicos no caso de sistemas *hard* R.T., obrigando no entanto os eventos esporádicos a serem atendidos dentro de um certo limite de tempo.

Para sistemas mais ligeiros, *soft* R.T., as restrições são do tipo estatístico onde se pretende garantir tempos médios aceitáveis para os acontecimentos cíclicos e também para acontecimentos esporádicos.

É provável que existam combinações dos requisitos *soft* e *hard* para um mesmo sistema real e o desenhador do sistema tentará reduzir ao mínimo o número de actividades críticas, do tipo *Hard* R.T. **(Bennet, 1994, secção 1.5)**.

Mesmo perante restrições do tipo *Soft Real Time*, **Milenkovic (1992)** refere claramente que o atendimento a novas rotinas deve ter um tempo máximo de atendimento garantido, para além do evidentemente desejável tempo médio de atendimento baixo.

2.4 Controlo Distribuído

2.4.1 Sistemas Distribuídos

Os sistemas embebidos distribuídos[1] constituem uma ambiciosa iniciativa dos dias de hoje e são alvo de pesquisa envolvendo quantidades avultadas de dinheiro[2]. Sendo o propósito destes sistemas o ***controlo distribuído***, pode-se compreender o interesse neste assunto. Até os nossos dias, utilizaram-se estruturas centralizadas de controlo, porventura de forma hierárquica, mas raramente distantes no espaço. O controlo, quando era feito de forma distribuída, era sempre feito de forma totalmente autónoma, sem interligação com outros sistemas com interesse. Hoje em dia, no entanto, devido a esforços para a introdução do conceito CIM[3] e consequente quebra de barreiras de comunicações, devido à baixa generalizada no preço do poder de cálculo e devido ao avanço nas redes e respectivos protocolos de comunicações (nomeadamente ao nível das redes de campo - *filed bus*) e ao aparecimento de sensores a actuadores inteligentes, entre outros factores, pensa-se em interligar de facto todos os subsistemas. Esta interligação pode até ser de forma heterárquica (contrário de hierárquica), sem existir um ponto onde toda a informação converge. Este tipo de arquitectura descentralizada e com um certo grau de inteligência distribuída permite uma eficaz supervisão do sistema e poder fornecer um acréscimo de fiabilidade, p. ex. se se conseguir instalar redundância para os sistemas críticos.

Milenkovic (1992, secção 13.1), Bennet (1994, pág. 49) , e Tanenbaum (1992, parte II, pp. 363-336) referem as vantagens (☺) e desvantagens (☹) dos sistemas distribuídos:

- ☺ Carga computacional é dividida pelos vários processadores. Cada um tem um trabalho simples porque analisa poucos sinais simultaneamente, p. ex. apenas um sinal por processador, o que facilita muito o processamento e permite realizar cálculos mais complexos. Outra possibilidade será comunicar os dados já processados para um outro processador que se dedica apenas à malha de realimentação
- ☺ O baixo custo dos recentes microprocessadores (μP) produzidos em série é uma vantagem face a um hipotético computador com poder de cálculo equivalente ao conjunto

[1] Esta designação é utilizada no artigo referido pela nota de rodapé 2

[2] Ver *Electronic Design* 8 Janeiro 1996, pág. 116, "STREAM *Fosters Development of Distributed Embedded Systems*" ou Gergeleit *et al* (1994)

[3] *Computer Integrated Manufacturing*, interligação de todos os recursos dentro de uma empresa dentro duma estrutura de rede de comunicações. Este acrónimo foi substituído por CHIME, *Computer and human integrated manufacturing and Engineering*, pretende-se dizer que a engenharia, a produção, os humanos e os computadores devem estar perfeitamente integrados e em cooperação.

- ☺ Pode nem existir um computador centralizado com poder de cálculo suficiente para substituir todos os outros computadores da rede
- ☺ Partilha e balanceamento de recursos, p. ex. capacidade de armazenamento pode ser partilhada e uma escassez temporária pode ser colmatada por um outro sistema, o que inerentemente acrescenta fiabilidade ao conjunto
- ☺ O sistema pode ser inerentemente distribuído no espaço, logo um sistema distribuído pode ser mais adaptado
- ☺ A flexibilidade e a facilidade de expansão do sistema aumentam, acrescentar uma máquina é fácil para executar novas tarefas
- ☺ Aumento da fiabilidade do sistema, a falha de um sistema não crítico não pára todo o conjunto
- ☺ A arquitectura distribuída pode tornar mais fácil a introdução de redundância no conjunto e facilita também a transferência para estruturas de salvaguarda da parte avariada
- ☺ É mais fácil fazer pequenas mudanças, localizadas numa pequena porção do conjunto
- ☹ Grandes mudanças podem provocar efeitos colaterais indesejáveis
- ☹ A falha da rede de comunicações pode acarretar que muitas máquinas fiquem isoladas (senão todas)
- ☹ As comunicações podem ser demasiado lentas e provocar outros erros (p. ex. erros de transmissão, de saturação da rede, etc.)
- ☹ Segurança: a informação viaja pela rede e poderá eventualmente ser interceptada
- ☹ O software fica mais pesado
- ☹ Pode haver necessidade de gerir a partilha de recursos
- ☹ A administração do sistema é mais complexa

Um projecto de Investigação e Desenvolvimento (I&D), actualmente em curso na F.E.U.P., que incorporará uma filosofia de controlo distribuído é aquele que controla a ligação à rede eléctrica de distribuição de energia de vários geradores eólicos de pequena capacidade. Cada gerador de energia terá o seu controlador local (embebido) e comunicará o seu estado actual a outro controlador que se encarregará de encaminhar devidamente a potência fornecida para posterior fornecimento à rede.

2.4.2 Sistemas Distribuídos em Tempo Real

Tal como nas outras formas de controlo, também o controlo distribuído exige o cumprimento de restrições temporais, com a agravante de que as restrições se passam, agora, entre máquinas ligadas por uma rede de comunicações.

A necessidade de ***protocolos de comunicações determinísticos***, isto é, que tenham tempos máximos de transmissão previsíveis, aparece naturalmente para que o desempenho

das comunicações possa ser contabilizado na caracterização das restrições temporais a impor aos cálculos, face às restrições de tempo real de todo o conjunto (Gergeleit *et al*, 1994).

Actualmente, assiste-se precisamente ao momento em que o poder de cálculo necessário para implementações distribuídas começa a estar disponível de forma economicamente interessante, mesmo perante restrições de tempo real, tal como referido por Tanenbaum (1992, parte II) e Levi *et al* (1990 cap. 15). Gergeleit *et al* (1994) vai mais longe ainda e apresenta um sistema orientado ao objecto, distribuído, em tempo real que utiliza uma série de recursos avançados para tentar cumprir os requisitos de tempo real determinados previamente para o sistema.

2.4.3 Tolerância e Fiabilidade

Os problemas dos sistemas distribuídos aparecem no controlo distribuído com especial pertinência. A rede pode falhar, algum parceiro de comunicação pode falhar, as comunicações podem falhar, os parceiros de comunicação podem ficar dessincronizados, etc..

Estes e outros problemas são tratados pelos autores anteriormente referidos mas a solução ocupa sempre poder de cálculo, o que pode dificultar o cumprimento de todos os requisitos, de tempo real e todos os outros específicos ao sistema. Tal como seria de imaginar, todas as situações devem ser previstas e cada componente do sistema deve, sempre que detectar uma falha do parceiro de comunicações, proceder da maneira prevista (p. ex. lançar um alarme); se a falha for grave, então o sistema deve, de forma autónoma, tomar medidas para trazer a parte do sistema directamente dependente de si até um estado seguro de funcionamento (ou repouso). Será, então, importante estudar quais as consequências de possíveis falhas no sistema. Nesta filosofia de tolerância e robustez de funcionamento, é importante prever uma gradual degradação de desempenho por impossibilidade de cumprir um certo requisito, p. ex. de T.R. (Gergeleit et al, 1994).

Os custos da implementação dos Sistemas Distribuídos são elevados quando se pretende a anunciada robustez, pelo que estes sistemas são apenas utilizados quando a aplicação o exige e é possível suportar os encargos decorrentes da resolução de todos os problemas mencionados.

2.5 Resumo

Durante este capítulo introdutório estudaram-se as características particulares dos S.E.. A sua função primordial é o controlo, controlo esse que se pretende atempado face à realidade, que faz surgir o conceito de T.R.. As várias facetas do T.R. foram estudadas como sendo o tipo de estímulos a que o S.E. está submetido.

O estado da arte na área do controlo inclui ainda a o controlo distribuído no espaço, utilizando redes de computadores com características especiais de fiabilidade e previsibilidade. As características deste tipo de sistemas foram referidas e, sendo verdadeiro que esta área movimenta muita investigação recente, a sua implementação só será interessante para processos complexos por forma a suportar os custos do elevado poder de cálculo necessário à resolução de todos os problemas inerentes à distribuição física dos vários componentes.

3. Engenharia embebida

3.1 Mecatrónica

Segundo Comerford (1994) «***Engenharia Mecatrónica*** é a combinação sinergética da engenharia mecânica, electrónica de controlo e engenharia dos sistemas no desenho de produtos e processos de produção».

A filosofia por trás da mecatrónica é, então, a necessidade de fazer uma análise conjunta dos sistemas electro-mecânicos, de maneira a assegurar o seu desempenho óptimo. Esta optimização do desempenho pode dizer respeito a um sistema em malha aberta, onde se procura os melhores valores para todos os parâmetros envolvidos no projecto, por forma a encontrar o melhor relacionamento possível entre electrónica e mecânica. Os referidos conceitos podem também ser utilizados com vantagem no desenho de sistemas de controlo realimentados, especialmente na zona de interface entre várias tecnologias de suporte diferentes.

O mesmo autor assegura ainda que o conceito da mecatrónica ajusta-se bem às novas técnicas de ***engenharia concorrente***, onde uma equipa de desenhadores, cada um especialista numa certa fase do processo, num certo assunto, está envolvido no desenho quase simultâneo de todas as fases do projecto. Numa abordagem convencional, estes projectistas seriam sucessivamente confrontados com o trabalho do nível de projecto anterior, um trabalho fechado a mudanças, mesmo de pequenos pormenores[1]. Esta limitação pode levar a um acréscimo de trabalho e/ou custos pois pode haver necessidade de trabalho extra para cumprir aquilo que foi anteriormente determinado, quando existiria uma solução mais fácil se a fase anterior de projecto não estivesse fechada/concluída. Com a engenharia concorrente, os vários projectistas podem trocar ideias acerca da melhor solução, a qualquer nível do sistema, pois ele está todo em desenvolvimento quase simultâneo. A eficiência aumenta, a qualidade do projecto é aumentada e os custos de engenharia do projecto são também reduzidos.

A junção das ferramentas de desenho e engenharia assistida por computador (CAD[2]/CAE[3]) da engenharia electrotécnica com as da engenharia mecânica resulta numa

[1] Se existissem mudanças, perder-se-ia alguma parte do trabalho subsequente, que teria de ser completado uma vez mais

[2] *Computer Aided Design*

[3] *Computer Aided Engineering*

poderosa ferramenta para a mecatrónica onde se pode simular em conjunto os aspectos eléctricos, mecânicos e de relacionamento. Ainda Comerford (1994) afirma que um dos programas mais utilizados na área da mecatrónica permite a análise da sensibilidade à variação dos parâmetros de cada componente e também a análise dos limites de funcionamento de cada componente. Tal como seria de esperar, a utilização de tal ferramenta de engenharia assistida por computador permite aumentar a robustez do projecto e a sua qualidade, para além de baixar o custo final do projecto (evita-se o sobredimensionamento de componentes). Poupa-se também tempo e dinheiro na fase de engenharia do projecto porque se evita a construção e teste desapoiado de protótipos.

O tipo de análise preconizado pela Engenharia Mecatrónica é especialmente apropriado para o desenho de sistemas complexos, porventura controladores embebidos onde exista uma componente de interacção com outro tipo de sistemas, p. ex. sistemas mecânicos. Ainda Comerford (1994) refere que as aplicações incluem suspensões activas, sistemas hidráulicos de travagem, sistemas robóticos, entre muitos outros.

3.2 Projecto

O desenho de qualquer projecto complexo deve ser cuidadosamente documentado e o seu planeamento alcançado de forma estruturada mas global, tal como conseguido pela abordagem conjunta de sistemas electro-mecânicos preconizada pela Engenharia Mecatrónica. Quer se trate de um projecto de computadores (hardware ou software), de engenharia civil, ou de carpintaria, o conceito é o mesmo: a especificação estruturada do projecto a empreender vai-se tornando cada vez mais importante à medida que (Dorf *et al* 1993, secção 84):

- o projecto é mais extenso
- tem maior duração prevista
- é mais sofisticado
- mais pessoas trabalham nele
- aumenta a probabilidade do projecto inicial necessitar de mudanças e/ou actualizações

Considerando agora a construção de um sistema embebido, deve referir-se que uma análise estruturada, utilizando método estruturados conceituados permite[1]:

- uma eficaz compreensão de todas as faces do problema
- melhor avaliação dos requisitos internos do sistema
- a abstracção da metodologia de implementação permitirá numa fase final projectar os limites hardware/software do sistema no sítio mais conveniente (por vezes difícil, na prática)
- a geração de documentação normalizada, eventualmente utilizando uma notação gráfica normalizada para uma rápida compreensão das interacções dentro e fora dos limites do sistema
- menor tempo de implementação
- facilidade de trabalho em grupo
- facilidade de fazer manutenção e/ou pequenas mudanças ao projecto no futuro

Nota: Não será recomendado qualquer método mais apropriado para este desenvolvimento global, pelo contrário, pretende-se fornecer linhas de actuação que possam ser utilizadas com qualquer método. Alguns métodos estruturados serão referidos brevemente em "3.3.2-Desenvolvimento em T. R.", pág. 28.

A abstracção de pormenores de implementação deixa que a equipa de projectistas se concentre no essencial, deixando para a parte final os problemas relacionados com a implementação. A implementação pode até requerer a revisão do projecto por forma a torna-lo mais exequível, por exemplo desenhar sistemas específicos para auxílio à eliminação de erros (*debugging*). Tal iteratividade entre projecto e tentativa de implementação é desejável e natural, deve até ser aproveitada para gerar documentação apropriada, clarificando as razões que orientaram o seu desenvolvimento de determinado modo.

A análise completa do problema antes da divisão entre hardware[2] e software revela-se extremamente importante, pois nos sistemas embebidos é, em geral, fácil libertar um para carregar o outro, permitindo assim que o projecto seja o mais apropriado.

---

[1] Os conceitos apresentados são retirados de Ward *et al* 1985, Rumbaugh *et al* 1991 e Dorf *et al* 1993. Estes autores apresentam metodologias de desenvolvimento de software que podem ser extrapoladas para o caso típico de sistemas embebidos onde se desenvolve hardware e software, o que introduz complexidade extra ao sistema. Exemplos da complexidade, potenciais problemas e necessidade de estruturação do desenvolvimento de S.E. podem ser consultadas nas referências Kurpis (1996), Jani *et al* (1996), Ganssle (1990b), que contribuem também para os conceitos apresentados nesta secção

[2] Considerando que algum hardware será desenhado especificamente para a aplicação (em geral assim acontece, nos S.E.)

Durante o desenho do projecto deve haver o cuidado de escolher uma plataforma com o poder de cálculo apropriado para os algoritmos que se pretende implementar, se tal escolha for possível no âmbito do trabalho em curso. Ver "3.2.2 Escolha da Plataforma de Processamento", pág. 23.

Muitos pormenores do projecto estão ainda dependentes de uma escolha: se vai ser utilizado um sistema operativo tempo real ou não. Existe ainda o problema de escolher o S.O. mais apropriado para a aplicação em causa. Ganssle (1994a) indica que 80% dos sistemas embebidos não utilizam S.O. comerciais, no entanto, o autor refere que tendo utilizado um pela primeira vez, nunca mais deixou de o fazer.

A necessidade de incluir um S.O. comercial pode obrigar à escolha de uma outra plataforma computacional diferente, o que pode obrigar reconsiderar alguns pontos anteriormente aceites como encerrados, p. ex. pode ser necessário fazer mais uma iteração e recomeçar a afectação de recursos computacionais. Todo este raciocínio é válido de cada vez que se estuda uma hipotética configuração software/hardware, intimamente ligados nos S.E..

Antes de dar início à implementação deve estar-se certo de que o desenho do projecto é apropriado, pesando todos os compromissos inerentes a cada problema e a cada solução. Após cada decisão, sempre bem fundamentada, deve ser produzida a documentação necessária a uma rápida reavaliação da decisão perante a chegada de novos dados.

Não deve ser minimizada a importância de permitir um *debugging* facilitado do sistema: será útil prever desde logo essa necessidade e eventualmente incluir código e/ou hardware específico para a eliminação e diagnóstico de erros tanto durante o desenvolvimento com durante o funcionamento normal.

Na fase final do desenho do projecto (antes de começar a implementação) deve fazer-se a afectação de todos os recursos computacionais importantes: devem ser considerados temporizadores, interrupções, entradas e saída tanto analógicas como digitais, portas de comunicações, etc.. Será também importante pensar numa estrutura de prioridades para as várias tarefas que devem ser executadas. Deve ainda ser ponderado se será necessário ou conveniente um barramento normalizado (*standard*) para a aplicação e se tal barramento deverá ser apenas digital, se deverá ser analógico também, qual largura de dados e endereços, equipamento disponível no mercado, etc..

Uma vez iniciada a implementação, muitos custos terão já ocorrido mas pode existir a necessidade de voltar a analisar qualquer pormenor que tenha surgido e nessa altura poder-

se-á utilizar a documentação gerada anteriormente para tomar novas decisões, tomando em consideração os novos dados. A Figura 4 procura descrever sob a forma de fluxograma o que se tem vindo a referir nesta secção.

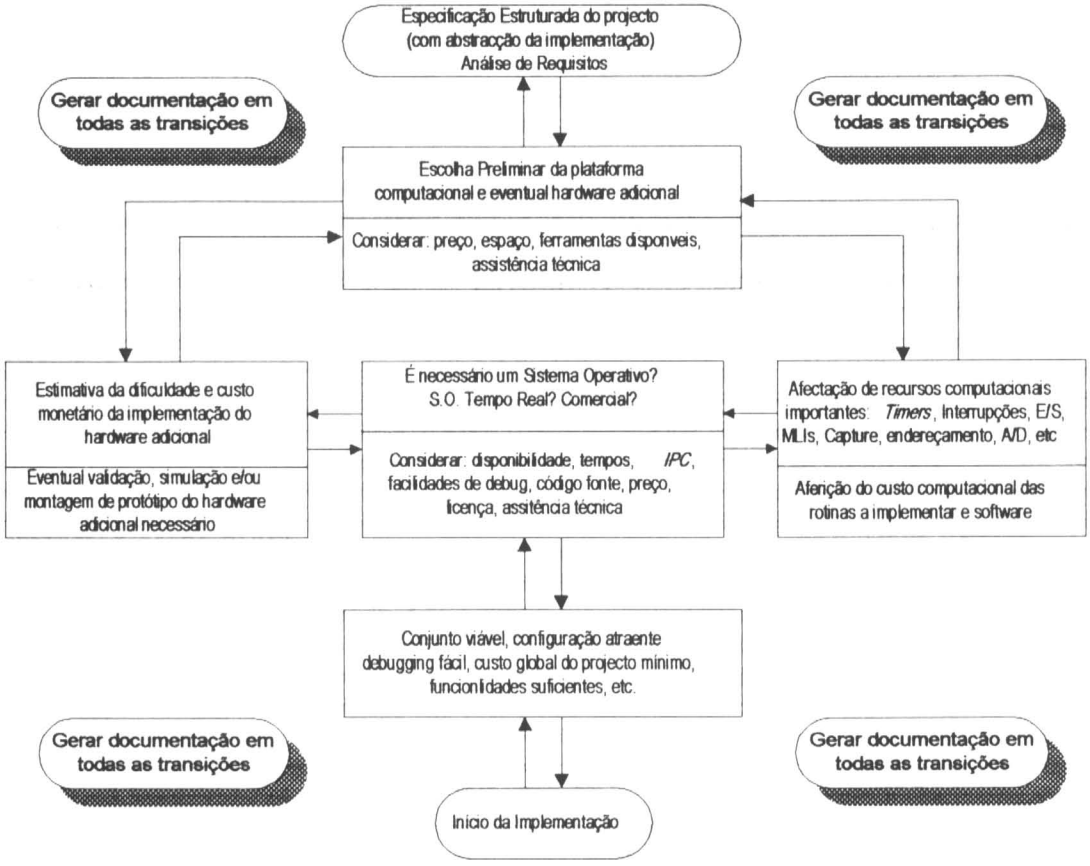


Figura 4: Fluxograma da metodologia proposta para o desenvolvimento global de sistema embebidos

3.2.1 Equilíbrio Software/Hardware

Na secção anterior foi preconizado que a análise inicial de requisitos seja feita com abstracção da implementação, para que essa opção fosse tomada de forma bem fundamentada numa fase posterior do projecto. Será portanto necessário manter um equilíbrio específico a cada tentativa de implementação no balancear entre resolver os problemas por programação ou por hardware específico.

Quando um sistema embebido tem uma componente de hardware fraca, o projecto apresenta as seguinte características (Figura 5):

- ☹ O microcontrolador (μC) ou microprocessador (μP) fica sobrecarregado
  - Conseguirá cumprir todas as restrições temporais?
  - Haverá falta de recursos computacionais, p. Ex.: faltarão temporizadores (*timers*)?
- ☺ Demora-se menos tempo a fazer *debugging* do hardware (exigiria recursos caros)

- ☹ Demora-se mais tempo a resolver problemas de software, o que é, em geral, demorado no caso do software de S. E. e pode também revelar-se caro
- ☺ Em geral o projecto entra em funcionamento (precário) mais cedo
- ☺ O projecto fica com dimensões físicas mais pequenas

Quando um sistema embebido tem uma componente de hardware forte (Figura 5):

- ☹ Eventualmente poderá reduzir a flexibilidade de funcionamento
- ☹ Quase sempre encarece o projecto
- ☺ Simplifica o software
- ☺ Em geral o sistema fica mais rápido e fiável
- ☹ Pode haver necessidade de várias iterações até estabilizar o hardware: pode demorar tempo e atrasar até o desenvolvimento do software também
- ☹ Resolver problemas em hardware requer equipamentos caros: osciloscópios, etc.
- ☹ Podem surgir haver problemas de espaço, espaço para ligações, etc.

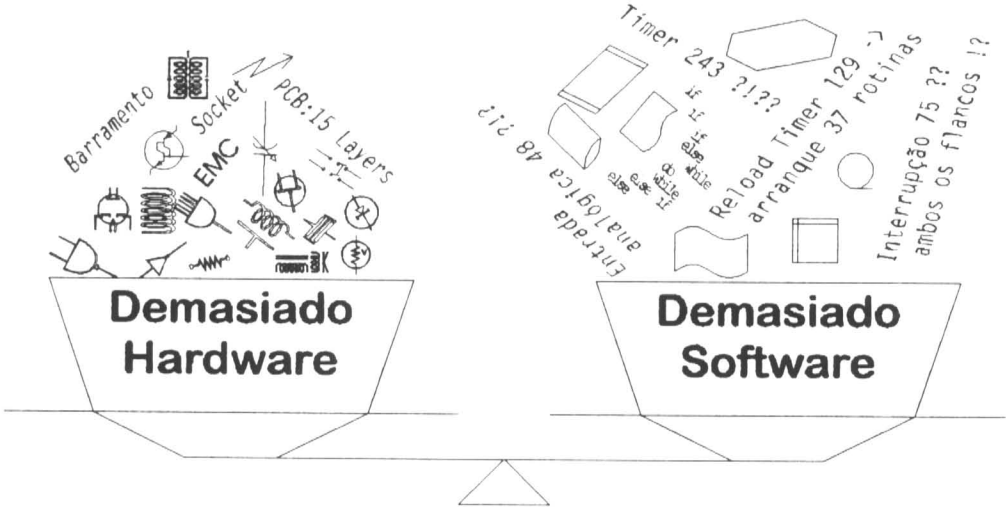


Figura 5:Compromisso entre hardware e software excessivos

3.2.2 Escolha da Plataforma de Processamento

A escolha de um microprocessador (μP) ou microcontrolador (μC) para o sistema que vai ser desenvolvido é, a maior parte das vezes, uma escolha entre os vários descendentes duma mesma família ou a escolha entre uma plataforma de 8 ou 16 bits (Ganssle 1990c). Esta limitação conhece uma variante quando se decide passar a utilizar um novo microcontrolador.

Actualmente, ao desenhar um novo projecto, convirá ponderar com cuidado se se deve investir tempo no desenvolvimento de uma placa para o μC ou μP a utilizar, para além de o escolher a ele mesmo (ver "3.4.9-Ferramentas para desenvolvimento", pág. 51).

De acordo com Ganssle (1990c), muitos factores devem influir a escolha do μC ou μP a utilizar. Muitos são técnicos, outros não:

- Disponibilidade do hardware, do software e da assistência necessários
- Necessidade de rentabilizar os investimentos anteriores, tanto a nível de *stocks* de hardware como a nível de software (reutilizar compiladores, sistemas operativos, etc.)
- Rentabilizar os conhecimentos anteriormente adquiridos
- Se se vai investir num novo μ[1], então há que considerar o preço do conjunto das ferramentas necessárias (incluindo compilador, *debugger*, ICE[2], S.O.T.R., etc.) e estudar também se são ferramentas com futuro, com assistência técnica, etc.. O μ tem capacidades de *debugging* internas?
- Na escolha do processador deve influir a disponibilidade de versões com RAM interna apropriada e, idealmente, também com alguma forma de ROM interna apropriada, ex.: OTP ou EPROM
- Deve ponderar-se a capacidade de endereçamento tanto de memória como (principalmente) de E/S analógicas e digitais
- Não esquecer outros recursos computacionais:
  - PWM
  - Conjuntos de temporização, comparação e captura (*timers*, *compare*, *capture*)
  - número e velocidade das interrupções
  - características das interrupções: prioridades, flanco, etc.
  - capacidades integradas de comunicação
  - poder de cálculo (capacidades de vírgula flutuante, etc.)
- A necessária avaliação de custos não deve ser feito só ao preço do μC mas sim ao preço de toda uma solução viável

A escolha de uma nova plataforma não é trivial e muitas vezes opta-se simplesmente pelo sistema que já é conhecido e escolhe-se o hardware adicional apropriado, com vista a rentabilizar os conhecimentos anteriormente adquiridos, para evitar uma potencialmente demorada aquisição de conhecimentos relativos a uma nova plataforma de processamento.

As considerações acerca da escolha iterativa do conjunto hardware/software feitas em "3.1-Projecto", pág. 18 permanecem válidas.

Depois do rápido estudo da plataforma de processamento apropriada para um projecto embebido, estude-se de seguida o software para o sistema.

[1] μ leia-se como "micro" e entenda-se que refere tanto um μP, microprocessador ou um μC microcontrolador

[2] *In-Circuit Emulator* - ver "3.4.9-Ferramentas para desenvolvimento", pág. 51

3.3 Software

O Software dos S.E. tem-se revelado uma parcela cada vez mais importante do custo total dos sistemas embebidos recentes. Tal deve-se à baixa de preço do hardware e ao aumento de complexidade dos algoritmos (e software em geral) utilizados em S.E.. Tal facto é referido por **Morse *et al* (1996)**, afirmando que o peso relativo entre o hardware e o software tem vindo a mudar-se substancialmente em torno do software nos últimos 20 anos (ver Figura 6).

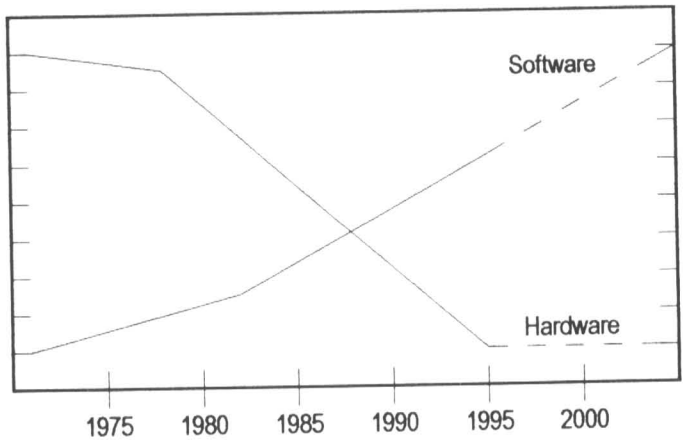


Figura 6: Custos relativos software/hardware
(as curvas são indicativas da ordem de grandeza apenas)
Morse *et al* (1996)

A mesma referência assegura que o software acaba por tornar-se caro porque as aplicações apresentam exigência crescente e os melhoramentos, actualizações e correcções se vão tornando cada vez mais frequentes. O software de comunicações tende a fazer crescer ainda mais aos custos pois ele é geralmente caro e comprado externamente, porventura ao fabricante do hardware de comunicações.

Tal como referido por **Dorf *et al* (1993, secção 84)**, a "crise" do software é caracterizada pela sistemática entrega de trabalhos fora de prazo e pelo ultrapassar dos orçamentos previstos. Segundo a mesma referência, iniciou-se a procura de métodos para o desenvolvimento apoiado (e estruturado) de sistemas de software. A filosofia de desenvolvimento do software mudou dramaticamente com o aparecimento do trabalho **DeMarco (1979)**. Nesse trabalho é proposto que o software seja desenvolvido tal como qualquer outro grande trabalho de engenharia: através da construção de modelos, que permitem a troca de ideias entre as várias partes envolvidas no projecto (ver também 3.3.2 Desenvolvimento em T. R., pág. 28).

Os mais recentes desenvolvimentos na área da engenharia de software conduzem à análise e implementação orientada ao objecto. Um dos métodos mais divulgados é o método

descrito por Rumbaugh *et al* (1991). Este método apresenta uma partição do projecto em 3 vistas ortogonais, o modelo de objectos (estrutura e relacionamento das partes dentro do projecto), modelo dinâmico (sequência de operações) e o modelo funcional (transformação de dados). Este método prevê ainda 3 fases para aplicação do método: a primeira fase, de análise (análise de requisitos, abstracção da implementação), uma segunda fase de desenho do modelo de acordo com o método, e finalmente a modificação do modelo anteriormente conseguido para permitir a sua implementação. O Objectivo de toda esta metodologia é a desmontagem da realidade com vista a permitir um fácil implementação. Para acelerar a implementação de sistemas analisados com o referido método, existem ferramentas CASE (*computer aided software engineering*), que geram parte do código da implementação (p. ex. em C++) a partir do grafismo introduzido pelo método.

Para o futuro próximo espera-se o método unificado para desenvolvimento OO, que está actualmente sob desenvolvimento e acerca do qual existe apenas informação preliminar Booch (1996). Os principais autores do método são, talvez, os mais notáveis autores a nível de tecnologia OO, pelo que se espera muito deste método.

3.3.1 Linguagens de Programação

Ganssle (1995c) estuda várias das linguagens comuns em sistemas embebidos, referindo-se a seguir as conclusões desse trabalho:

- APL é uma linguagem rara, que obriga a raciocínios complexos, o que faz com que seja custoso programar por se tornar demasiado críptica
- ADA é a linguagem da qual se poderá dizer "se compila, então funciona" - essa é uma característica altamente desejável para uma linguagem destinada a sistemas embebidos onde é difícil fazer *debugging*
- A linguagem C é, de facto, a linguagem da esmagadora maioria dos sistemas embebidos actuais
- A linguagem C++ começa a despertar interesse mas ainda poucas aplicações a usam em todas as suas potencialidades

As linguagens de alto nível C e o C++ são extremamente apropriadas para programação de sistemas embebidos porque[1]:

- São linguagens altamente normalizadas, definidas há algum tempo por instituições independentes como a ANSI - *American National Standard Institute.*

[1] Bennet (1994, secção 5.14) descreve algumas características desejáveis em linguagens para a programação de sistemas em Tempo Real

- Permitem programar indiferentemente com funcionalidades próximas da linguagem *assembly* (baixo nível) e logo depois com funcionalidades de alto nível
- Permitem uma interface fácil com a linguagem *assembly* do processador
- São linguagens compiladas e o código gerado pode ser altamente optimizado resultando assim em código compacto e de rápida execução
- São linguagens genéricas, largamente utilizadas na comunidade da engenharia
- Melhoram a legibilidade do código
- Têm estruturas de dados claras, regras de visibilidade das variáveis bem definidas
- As estruturas de controlo e fluxo de execução do programa são claras e poderosas
- Permitem a construção fácil e normalizada de um projecto modular

Opiniões mais conservadoras continuaram a defender que em aplicações embebidas todos os bytes e todos os microsegundos contam e assim defendem a programação em *assembly*. Tal como referido em Ganssle (1995c), esta posição vai-se tornando cada vez mais frágil à medida que o poder de cálculo embebido se torna mais poderoso. De acordo com o referido autor, contar ciclos T de execução[1] é um terrível desperdício da inteligência do programador e deve até ser evitado excepto nas aplicações mais sensíveis.

Será mesmo necessário recorrer à programação em *assembly*? Será que o código gerado pelo compilador é suficientemente bom? Na realidade, com a melhoria da qualidade dos compiladores de C, hoje em dia pode conciliar-se a programação de toda a aplicação em C com uma execução rápida e um código moderadamente compacto. Depois de tudo programado em C, é sempre possível rever o *assembly* gerado e, se necessário, reprogramar directamente em *assembly* as partes de desempenho mais lento e chamadas mais frequentemente. Em Ganssle (1990c) é referido claramente: estudos revelam que 90% do tempo computacional é gasto em 10% do código. Identificar esta porção de código e concentrar as optimizações nessa área será, possivelmente, muito vantajoso. Como é claro, deve existir um cuidado extra no código de interrupções frequentes. Transcrevem-se ainda duas notas de cuidado das referências Ganssle (1995c) e Ganssle (1990c), especialmente pertinentes para S.E.:

1. Um compilador de C (por muito bom que seja) necessita sempre de mais alguma ROM e também mais alguma RAM para executar o programa compilado, necessidades das rotinas próprias do compilador (em tempo de execução)
2. É necessário ponderar se o microcontrolador (μC) que estamos a usar é minimamente apropriado para a execução de código C. Dois exemplos de μCs que

[1] Cada ciclo interno do processador

não são muito apropriados para C: o 6805 e o 8051. Escrever código *assembly* para estes μCs pode trazer uma maior diferença de desempenho que noutras plataformas mais apropriadas a executar código compilado C. Eventualmente até é mais importante o processador em que o código vai correr do que o compilador que produzirá o código.

3.3.2 Desenvolvimento em T. R.

O desenvolvimento de software sob restrições de tempo real é ainda mais difícil que o software sem essa preocupação. Este capítulo será apenas uma breve referência a alguns métodos (não se pretende ser exaustivo), apropriados para o desenvolvimento deste tipo de sistemas, sem no entanto desenvolver excessivamente qualquer deles.

Um dos métodos mais antigos mas ainda hoje bastante apropriado é o método proposto por Ward (1985). O modelo de vários níveis de abstracção, uma representação gráfica bem estudada, com preocupações tanto a nível de movimentação de dados do sistema como da evolução dos vários estados do sistema. É fornecido ainda um método de documentação de todo o sistema. A programação neste método pode ser assistida por uma ferramenta CASE, com pequenas diferenças face ao método original.

De acordo com Freedman *et al* (1995), que pesquisa os modelos de desenvolvimento em T.R., a aplicação de técnicas Orientada ao Objecto (O.O.) para o desenvolvimento de sistemas em T. R. pode ser conseguido através do trabalho Selic (1994). Freedman *et al* (1995) afirmam que este método cobre todo o ciclo de desenvolvimento, desde a especificação à implementação e inclui ainda comunicações entre objectos, com as vantagens adicionais de suportar arquitecturas distribuídas e implementações sob este método serem facilitadas pela utilização de uma ferramenta CASE.

Estes métodos dizem respeito à análise estruturada do problema, mas a implementação deste tipo de análise raramente é simples e necessita de suporte a nível do S.O..

3.3.3 Utilização S. O. Tempo Real

Ganssle (1994a) refere que 80% dos S. O. utilizados em S.E. são sistemas *ad hoc*, desenvolvidos especificamente para o projecto ou reutilizados de outras situações; os 20% restantes do mercado dos S.O. T. R. atingem cerca de 20 milhões de dólares americanos.

Ainda de acordo com a mesma referência, o resultado de se considerar que o S.O. tão pequeno e simples que pode ser desenvolvido num hora ou duas é uma estrutura cheia de erros difíceis de eliminar, que demora demasiado a corrigir e oferece uma reduzida gama de

não são muito apropriados para C: o 6805 e o 8051. Escrever código *assembly* para estes μCs pode trazer uma maior diferença de desempenho que noutras plataformas mais apropriadas a executar código compilado C. Eventualmente até é mais importante o processador em que o código vai correr do que o compilador que produzirá o código.

3.3.2 Desenvolvimento em T. R.

O desenvolvimento de software sob restrições de tempo real é ainda mais difícil que o software sem essa preocupação. Este capítulo será apenas uma breve referência a alguns métodos (não se pretende ser exaustivo), apropriados para o desenvolvimento deste tipo de sistemas, sem no entanto desenvolver excessivamente qualquer deles.

Um dos métodos mais antigos mas ainda hoje bastante apropriado é o método proposto por Ward (1985). O modelo de vários níveis de abstracção, uma representação gráfica bem estudada, com preocupações tanto a nível de movimentação de dados do sistema como da evolução dos vários estados do sistema. É fornecido ainda um método de documentação de todo o sistema. A programação neste método pode ser assistida por uma ferramenta CASE, com pequenas diferenças face ao método original.

De acordo com Freedman *et al* (1995), que pesquisa os modelos de desenvolvimento em T.R., a aplicação de técnicas Orientada ao Objecto (O.O.) para o desenvolvimento de sistemas em T. R. pode ser conseguido através do trabalho Selic (1994). Freedman *et al* (1995) afirmam que este método cobre todo o ciclo de desenvolvimento, desde a especificação à implementação e inclui ainda comunicações entre objectos, com as vantagens adicionais de suportar arquitecturas distribuídas e implementações sob este método serem facilitadas pela utilização de uma ferramenta CASE.

Estes métodos dizem respeito à análise estruturada do problema, mas a implementação deste tipo de análise raramente é simples e necessita de suporte a nível do S.O..

3.3.3 Utilização S. O. Tempo Real

Ganssle (1994a) refere que 80% dos S. O. utilizados em S.E. são sistemas *ad hoc*, desenvolvidos especificamente para o projecto ou reutilizados de outras situações; os 20% restantes do mercado dos S.O. T. R. atingem cerca de 20 milhões de dólares americanos.

Ainda de acordo com a mesma referência, o resultado de se considerar que o S.O. tão pequeno e simples que pode ser desenvolvido num hora ou duas é uma estrutura cheia de erros difíceis de eliminar, que demora demasiado a corrigir e oferece uma reduzida gama de

serviços, para além de complicar muito o porte da aplicação de um processador para outros e reutilização de software.

As características desejáveis de um S.O. T.R. (Ganssle 1994a, Bennet 1994, secção 6.2, Levi *et al* 1990 cap. 11, Tyle 1995a) são:

- Distribuição do poder de cálculo pelas várias tarefas através de escalonamento preemptivo[1] de tarefas
- Gestão de tarefas: criação de tarefas, alteração e consulta de propriedades das tarefas (p. ex. prioridade), reescalonamentos, etc.
- Estratégias de comutação de tarefas: estabelecimento de prioridades, cumprimento garantido das restrições temporais, etc.
- Gestão e disponibilização de recursos: gestão de memória (atribuição dinâmica, libertação e partilha de memória), *device drivers*, ficheiros, chamadas ao S. O., temporizadores (eventualmente com precisão variável), provisão para transferência rápidas de dados, etc.
- Comunicação entre tarefas (IPC - *inter process communication*), permitir métodos para implementação de exclusão mútua, semáforos, mensagens, comunicações eventualmente com prioridades, etc.
- Gestão de excepções e interrupções externas

Ganssle (1994a) refere ainda que os S.O. devem prever o *debugging* de modo fácil e a inclusão do código fonte não deve ser considerado como essencial já que alterar o código fonte do S.O. poderá dar azo a erros de difícil detecção.

Tyle (1995a) e Tyle (1995b) faz uma brevíssima descrição de alguns dos mais recentes S.O. do mercado, especialmente no que diz respeito à aplicação de arquitecturas x86 para S.E., e/ou em T.R..

Gien (1996) descreve brevemente as características desejáveis e inovadoras que será previsível que um S.O. para S.E., e logo, em T.R., tenha, num futuro próximo[2]:

- Tamanho variável de acordo os requisitos da aplicação: camadas sucessivas utilizadas opcionalmente oferecem extensões de capacidades à medida da aplicação
- Previsão para mono-processador bem assim como para processamento paralelo
- Arquitecturas de comunicações previstas no S.O. e disponibilização de meios para distribuição transparente

[1] S.O. preemptivo é aquele onde as tarefas saem de execução por acção do núcleo por oposição aqueles sistemas onde as tarefas se auto-suspendem (S.O. não preemptivos)

[2] Só são referidos aqui os assuntos que não foram ainda mencionados

- Núcleo[1] executivo de alta performance, próprio para T.R., com capacidades adequadas à aplicação
- Interfaces de programação abertas (API - *Application Programming Interface*), cumpridoras de standards de mercado tal como Unix, Posix (T.R. e *multi-thread*[2])
- Ferramentas de desenvolvimento, simulação e teste em estações de trabalho e PCs

Todas estas características são altamente desejáveis para um S.E.:

- A extensibilidade do S.O. para se adaptar aos requisitos da aplicação (o conceito não é novo mas é importante)
- A disponibilização de comunicações a nível do S.O., para permitir a utilização de protocolos comerciais, não desenvolvidos especificamente
- A distribuição transparente, para os S.E. é um novidade que visa a potencialidade de aplicação em distribuídos e possibilitar controlo distribuído de forma fácil
- Também inovador é o conceito de, por cima do mesmo núcleo executivo em T.R., ser utilizada uma interface com a aplicação estandardizada, por forma a ser possível a integração de aplicações não específicas, de fabricantes externos, mantendo a estrutura do núcleo executivo de alta performance
- Dada a dificuldade de trabalho em S.E., é necessário que o S.O. possibilite a utilização de ferramentas de simulação e que forneça métodos para *debugging* rápido

Ganssle (1994a) explica que os principais motivos para a não utilização de S.O. em S.E. são o custo e o desconhecimento. Certamente que, quando os S.O. passarem a dispor de todas as características descritas nesta secção, não haverá lugar a dúvidas: um S.O. é essencial para todos os sistemas embebidos excepto os de tamanho muito, muito reduzido.

Depois de se ter referido a metodologia de desenvolvimento para S.E., depois da discussão relativa ao software, há necessidade de rever as plataformas de processamento e suas características. É o que se faz de seguida.

[1] A camada mais interior do S.O.

[2] Tarefas mais leves para o S.O., de início mais rápido, que partilham o ambiente com o processo pai

3.4 Hardware

3.4.1 O mercado actual

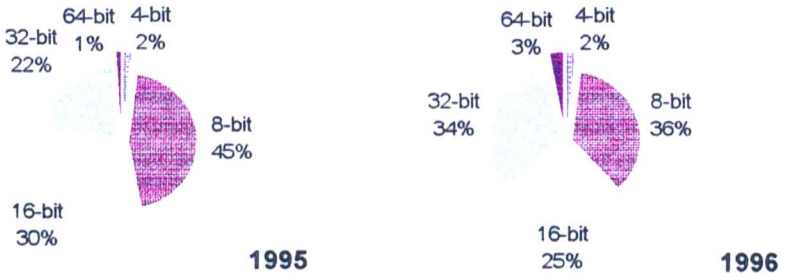


Figura 7: Distribuição do mercado de (μC e μP) utilizados em S.E., em função do tamanho da palavra, durante o ano de 1995 e intenção para 1996 (de acordo com questionário EDN 8/1/96, pp. 53-57)

De acordo com **Bursky (1996a)**, que cita dados (Figura 7) de um inquérito aos leitores da revista "*Electronic Design*", o mercado das aplicações embebidas de 4 bits manterá a sua quota de mercado durante o ano de 1996, que diz respeito às aplicações de menores requisitos computacionais e de maior sensibilidade aos custos. Tendo em conta que as aplicações embebidas tendem a tornar-se, em geral, mais sofisticadas e a necessitar de maior poder de cálculo, é natural que as intenções de trabalho utilizando o tamanho da palavra de 8 e mesmo 16 bits diminui de 1995 para 1996, aumentando as intenções de trabalho em S.E. com os mais recentes processadores e controladores de 32 ou 64 bits.

Ganssle (1995a) refere que se continuará a utilizar o processador apropriado para a aplicação em causa e que as aplicações embebidas que acabam por necessitar de poder de cálculo de 32 bits acabam por ser poucas mas muito badaladas, enquanto que o mercado continua a aplicar continuamente processadores de 4 e 8 bits para uma miríade de aplicações.

De seguida listam-se os factores que afectam o desempenho de uma plataforma embebida e podem levar a escolher um tipo de plataforma em detrimento doutra.

3.4.2 Microcontrolador *vs.* Microprocessador

O microprocessador (μP) dedica-se primordialmente ao processamento, p. ex. fazer cálculos e interagir com o utilizador. Pelo contrário, o microcontrolador (μC) dedica-se a controlar aspectos do seu "ambiente" (ver Figura 8).

Em geral, os sistemas embebidos são comandados por μCs.

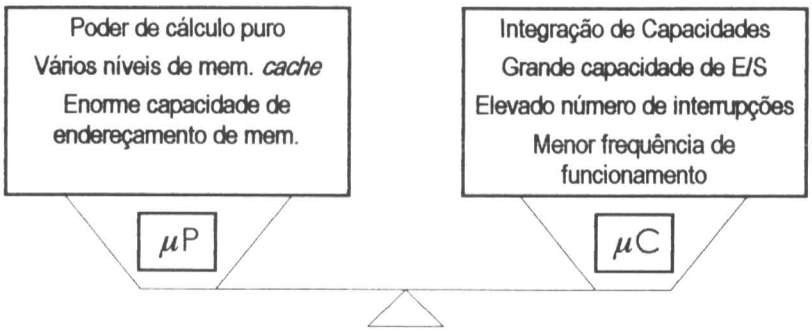


Figura 8: Comparação de características entre μC e μP

De acordo com **(Hersh 1996)**, o μC é dirigido para a integração de capacidades (um só integrado solitário mas poderoso) enquanto que o μP é voltado para o poder de processamento puro, para fazer cálculos e mostrar resultados, tal como necessário no mercado de PCs de secretária (*Desktop* PCs) ou poderosas estações de trabalho (*workstations*) de cálculo. São utilizados em sistemas que geralmente são complexos e muitas vezes utilizam muitos componentes adicionais para melhorar a performance. Os sistemas que utilizam μP são, quase sempre, produzidos em grandes quantidades. Os μCs são construídos a pensar que serão, muito possivelmente, parte de sistemas com hardware construído de propósito para a aplicação em vista. Os sistemas baseados em μCs têm geralmente poucos componentes, podendo ser produzidos em pequenas quantidades ou não. Em geral não necessitam de poder de cálculo em quantidade mas sim de capacidade de gerir os limites do sistema, isto é, controlar.

De acordo com **(Hersh 1996)** os μCs mais comuns incluem, pelo menos, as seguintes capacidades:

- Processador
- Alguma RAM interna
- Possibilidade de ROM/PROM/*Flash*/EPROM interna
- Temporizadores (*Timers*)
- Controladores de interrupções

E muitas vezes incluem muitas outras capacidades:

- Entradas analógicas, eventualmente multiplexadas
- Saídas de MLI, Modulação da Largura dos Impulsos (PWM - *Pulse Width Modulation*)
- Saídas em *Duty Cycle* (depois de um filtro passa baixo ficam saídas analógicas)
- Interrupções programáveis (a qual dos flancos, a qual dos níveis)
- *Watchdog* - dispositivo de segurança que, se não for refrescado periodicamente faz um *reset* ao processador assinalando que o programa entrou em estado inconsistente
- Portas de comunicações

- Capacidades de redução de consumo (*power down*)

Refira-se que é falso que só existem S.E. baseados em μC. De facto, existem muitos S.E. baseados em μP aparente menos apropriados para este tipo de aplicações. A disponibilidade de hardware e também de software, o elevado nível de documentação e o tão baixo preço de sistemas baseados em μP, tal como os IBM PC compatíveis levam a que o seu espectro de utilização se alargue até aos sistemas embebidos (Shear, 1994b).

3.4.3 Arquitecturas Internas de Processamento

3.4.3.1 CISC's

Complex Instruction Set Computer.

Segundo Bayko (1996) não existe, na realidade, um conjunto de características próprias deste tipo de micros (μs). É CISC o que não é nem RISC, nem DSP, etc. (ver Figura 9). Os CISCs têm a arquitectura mais tradicional (antiga). Entre as características que é comum encontrar estão os pequenos conjuntos de registos internos ao processador, a existência de instruções *assembly* com 2 operandos na memória, instruções *assembly* com comprimento (muito) variável e existência de programação em microcódigo[1].

A filosofia de desenho destes processadores é que a linguagem *assembly* deve ser tão poderosa quanto possível, tornando-os muito genéricos (a ideia de que isso traria maior velocidade caiu com o aparecimento dos RISCs). São apropriados para serem programados directamente *assembly* por humanos

Os CISCs da família 80x86 lideram o mercado dos processadores para computadores pessoais desde há algum tempo. O 8086 apresenta 4 registos genéricos de 16 bits, acessíveis como 8 registos de 8 bits. Existe uma segmentação conseguida com 4 registos próprios, escolhidos através do contexto da instrução que está a ser executada. Apresenta memória segmentada e tem um espaço isolado para Entradas e Saídas de 64Kbytes. O 8086 é um bom exemplo da complexidade dos CISC's, com instruções que podem ter comprimentos de 1 a 4 bytes (outros elementos da família apresentam comprimentos ainda maiores).

[1] Código dentro do microprocessador, em hardware, que traduz instruções exteriores em sequência de acções interiores necessárias para implementar a instrução requerida

3.4.3.2 RISC's

Reduced Instruction Set Computer.

Os computadores pessoais estão a experimentar uma mudança em direcção aos μPs RISC. Tal garantia é dada pelo acordo entre Apple e IBM para o desenvolvimento rápido de novos processadores deste tipo. A procura pela maneira de disponibilizar um poder de cálculo cada vez maior, a um custo mínimo assim o exige.

De acordo com **Bayko (1996)**, a IBM foi pioneira das ideias RISC (mas não do acrónimo). Considerada como uma filosofia de desenho, consiste em eliminar os factores que impedem as optimizações de hardware e balancear a favor do desempenho o trabalho de executar tarefas genéricas mas geralmente programadas numa linguagem de alto nível. O processador é pensado desde o início para fazer processamento de várias tarefas pseudo simultaneamente, em regime de multi-tarefa (*multi-tasking*).

A questão subjacente à filosofia RISC está no compromisso de deixar mais trabalho ao compilador da linguagem de alto nível ou, pelo contrário, implementar um elevado número de instruções *assembly*, tornando assim o hardware mais complexo e mais próximo da linguagem de alto nível. A diferença é que é tecnologicamente mais apropriado reduzir o tamanho do hardware e torná-lo assim mais rápido (desta maneira o software ficará também mais rápido). A compilação poderá ser mais complexa mas esse é um preço baixo a pagar para obter uma execução rápida... as desvantagens desta abordagem é que o compilador fica também mais caro e passa a ser extremamente difícil a um humano programar neste género de linguagem *assembly*.

Em resumo, a filosofia RISC diz simplificar para acelerar e assim:

- Um único tamanho para todas as instruções
- Todas as instruções executadas num único ciclo (ou em tempo fixo)
- Recurso a *pipelining*[1]
- Eliminação do microcódigo[2] (elimina-se tempos varáveis de execução)
- Instruções *assembly* sempre baseadas em registos internos e, no máximo, um operando da memória
- Elevado número de registos internos do processador
- Selecção dos registos através de janela deslizante (*windowed registers*)

[1] Características dos microprocessadores irem buscar à memória as instruções e prepararem a sua execução (ir buscar operandos à memória, etc.) para aumentar a velocidade da execução. Não é propriedade apenas dos RISC's

[2] Máquina de estados interna ao processador, utilizada para descodificar instruções complexas

A utilização da selecção de registos através da janela deslizante é extremamente apropriada para permitir rápidas mudança de contexto[1].

Estas características apareceram na tentativa de eliminar os pontos que impediam as arquitecturas CISC de se tornarem ainda mais rápidas na execução de código escrito numa linguagem de alto nível, num ambiente multi-tarefa, necessário à grande maioria das aplicações recentes.

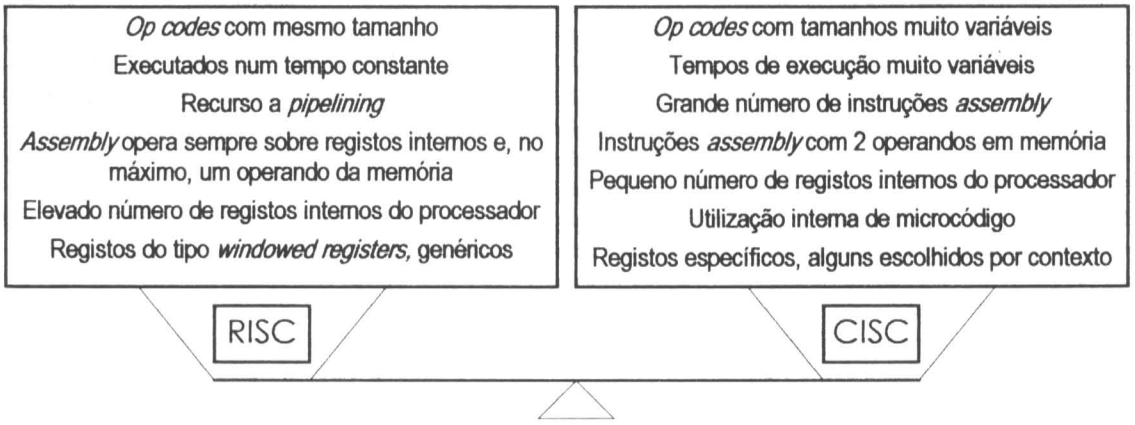


Figura 9: Comparação entre arquitecturas de μs: RISC e CISC

Não existe um conjunto de regras bem definidas para distinguir RISC de outras arquitecturas. Na realidade, algumas das características enunciadas nesta secção começam a surgir em processadores CISC mais recentes e mais poderosos.

Já existem μC RISC[2], que se revelam muito apropriados para sistemas em T.R. em geral e em particular para S.E.. O μC 80C166 da Siemens[3] é exemplo do que se consegue obter através de um desenho cuidadoso dentro da filosofia RISC, onde as interrupções são testadas todos os 50ns, a mudança de contexto é feita num ciclo de 100ns e apresentando o poder de processamento comparado na Figura 10, de acordo com informação testes divulgados pela Siemens sob a forma de publicidade. Adicionalmente, pode ser consultado o Anexo A onde se compara o desempenho de algumas plataformas de processamento e, entre elas, o μC RISC 80C166, o μC 8051 e o μP 486SX.

[1] Quando o processador deixa de executar uma rotina (que ainda não terminou) e passa para outra, diz-se que ocorreu uma mudança de contexto

[2] Bastantes μC são derivados de μP e, nesse caso, em geral o μC aparece no mercado muito mais tarde do que μP que o originou. No caso dos controladores da família x86, o controlador mais recente é um 486 compatível da National (Bursky 1995b), 2 gerações de atraso face ao Pentium Pro (686). A Intel oferece apenas o 386EX, com 3 gerações de atraso.

[3] Não existem μP desta família, ela foi desenhada propositadamente para o controlo

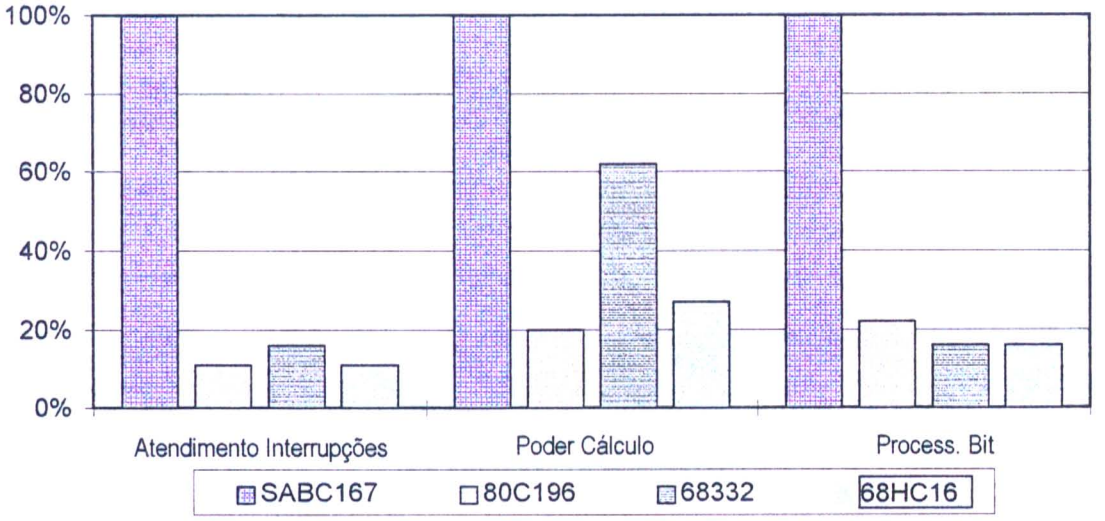


Figura 10: Comparativo do poder de cálculo do μC C167 (RISC) com outros μC não RISC (publicidade da Siemens em várias revistas técnicas)

3.4.3.3 DSP's

Os *Digital Signal Processors* podem ser utilizados como μC genéricos mas dispondo de arquitecturas apropriadas para processamento de sinal (ex. processamento de tabelas). Isto confere-lhes uma arquitectura algo própria, com algumas versões preparadas especialmente para cálculo de FFTs[1], sendo para isso providas de instruções *assembly* especialmente poderosas para este tipo de aplicações.

O artigo (Schafer, 1996) lista as aplicações típicas dos DSPs: Filtragem linear, Análise de espectros (no domínio das frequências) ou tratamento/transformação de sinais.

Este tipo de μC têm geralmente uma limitada gestão de memória, pouco suporte de interrupções, pouca integração de funções - inexistência de vários canais de entrada analógico, inexistência de canais de MLI (Modulação de Largura de Impulsos - *Pulse Width Modulation*), etc.. Tal como referido por Bennet, (1994, secção 3.4.2) estes processadores geralmente utilizam caminhos diferentes para dados e endereços, com diferentes números de bits, de acordo com a arquitectura "*Harvard*".

A família mais conhecida de DSP's é, talvez, a 320CX0 da *Texas Instruments*. Como exemplo, o TMS320C30 é um processador de 16 bits, vírgula flutuante (existem outras versões com vírgula fixa), tem 8 registos de 40 bits para dados, mais 8 registos auxiliares

[1] FFT - *Fast Fourier Transform* - Um algoritmo rápido, muito divulgado para cálculo eficiente de Transformadas de Fourier discretas e análise espectral de sinais

(ponteiros) de 32 bits e ainda 12 registos de controlo. Especialmente curioso é a existência de um registo de repetição de bloco, própria para processamento repetitivo de um bloco.

Gregg_Bennet[1] (1996) refere que é possível que, de futuro, muitas aplicações na área do controlo sejam baseadas em DSP para além do controlo de motores, que segundo refere, seria a área de aplicação tradicional deste tipo de processadores. Ainda o mesmo autor refere este tipo de controladores permite algoritmos mais sofisticados, maior grau de confiança, reduz o número de componentes do conjunto do controlador, permite actualizações e melhoramentos (*upgrades*) fáceis constituindo assim uma melhoria na relação desempenho/preço dos sistemas baseados em DSPs. Esta opinião deve ser entendida com algumas reservas pois parece nascer de uma comparação entre os mais recentes DSPs e os μC mais consagrados, logo mais antigos e menos poderosos.

3.4.3.4 PICs

Os PIC's são pequenos μCs, construídos com o custo em mente, incluindo minimização do n.º de pinos[2], pequeno número de instruções *assembly*, muito bem escolhidas (SISC-*Specific Instruction Set Computer*), elevado número de registos internos bem assim como memórias RAM e *Flash* EPROM (ou OTP[3]) internas (Bayko 1996, Ganssle 1995a).

Para exemplificar como é possível construir pequenos sistemas utilizando PICs, refira-se uma aplicação com dois integrados apenas (Bayko 1996): o processador é um PIC16C56 (18 pinos) que têm um núcleo interpretador de *tokens* de BASIC, comandos que, depois de pré-compilados, serão executados pelo código residente em 512 palavras de ROM interna. O outro integrado é uma EEPROM série (8 pinos), que contém o programa, pré-compilado a partir do BASIC. Ambos os dispositivos são construídos pela MicroChip.

O objectivo destes dispositivos é oferecer um desempenho razoável com um custos baixíssimos (1 dólar americano). Como curiosidade refira-se que a L.A. Gear construiu sapatilhas para criança que piscam um LED[4] a cada passo dado, graças a uma bateria de lítio que alimenta o LED e o PIC durante o tempo de vida esperado da sapatilha (Ganssle 1995a).

---

[1] Gregg Bennet e não Stuart Bennet

[2] O preço de um Circuito Integrado (C. I.) varia não só a área de silício e como também com o tipo de empacotamento, isto é, um C.I. com a mesma área de silício mas com mais pinos será muito mais caro que outro com um menor número de pinos

[3] Um dos factores de sucesso é precisamente a utilização de memória OTP (*One Time Programmable*), basicamente uma EPROM sem a janela para apagar o dispositivo, que encarece muito a produção do integrado

[4] Light Emitting Diode - Díodo emissor de luz, muito utilizado como pequena lâmpada, de menor consumo e maior robustez

A utilização deste tipo de dispositivos atinge tal popularidade, que, por vezes, se substitui electrónica discreta (*gates* lógicas e PALs[1]) por PICs porque são mais programáveis, repetidas vezes, e tão baratos como outras soluções. Podem também ser utilizados como dispositivos de autenticação, colocados propositadamente para dificultar o processo de cópia ilícita de hardware pois existem métodos para encriptar o código programado e assim impedir a leitura do código implementado.

3.4.3.5 Outras Arquitecturas

O acrónimo SISC significa *Specific Instruction Set Computer*, utilizado para referir um processador está de algum modo situado entre os RISC e os CISC, contendo algumas instruções complexas com um objectivo bem definido.

Processadores paralelos, tal como o nome indica, são computadores com vários CPU's em paralelo com o objectivo final de aumentar o poder de cálculo conjunto. Existem várias arquitecturas: *Single Instruction stream Multiple Data stream* [SIMD], *Multiple Instruction Single Data Stream* [MISD], *Multiple Instruction Multiple Data Stream* [MIMD] (Bennet 1994, secção 2.4.1). O grande problema dos computadores paralelos é a gestão dos dados partilhados, de sequências de processamento que originam resultados diferentes quando executadas por ordens diferentes, e a partilha de recursos críticos (Byrd *et al*, 1995). Os Transputers são um dos tipos de processadores paralelos, fazem computação paralela e permitem distribuição dinâmica de processos entre os vários processadores, que, depois de programados adequadamente, podem acrescentar fiabilidade ao sistema, pois a falha de um elemento não implica a falha de todos.

O Intel Pentium Pro, sucessor do Pentium, já vem preparado para processamento paralelo de raiz com o objectivo de execução simultânea de duas rotinas em processadores diferentes.

Mansur (1996) fala da promissora fusão entre DSPs e RISCs para processadores de comunicações. Segundo este autor, a sinergia da junção destas duas arquitecturas permite uma redução do custo, tamanho e consumos dos μC para comunicações portáteis.

[1] *Programmable Addressable Logic* - lógica discreta dentro de um integrado que geralmente só pode ser programado uma vez. Tem a vantagem de ser excepcionalmente rápida

3.4.4 PCs Embebidos

Tal como refere **Busrky (1995b)**, a quantidade esmagadora de software existente para os processadores x86 é uma forte mais valia na construção de qualquer sistema. **Tyle (1995b)** acrescenta que o desenvolvimento de aplicações de PCs embebidos permite tirar partido da proximidade e familiaridade deste tipo de arquitectura.

Apesar da arquitectura IBM PC não ser muito apropriada para aplicações embebidas, a facilidade de arranjar conjuntos que fazem (muito) mais do que o pretendido, a preços inferiores ao que se conseguiria se se tentasse fazer um sistema que fizesse apenas o que aplicação exige, indica que esta arquitectura deve ser considerada em bastantes aplicações.

Da família 80x86 existe o μC Intel 80386EX, derivado do 386 com alguns acréscimos de forma se formar um controlador. **Bursky (1995b)** descreve já um controlador baseado no 486, fabricado pela National Semiconductors. Algumas vantagens da aplicação deste μC são:

- Este tipo de controladores executa o código compilado para os respectivos processadores (que representa uma imensa biblioteca)
- Acrescenta ainda o seu poder de cálculo com integração de funções
- A integração de funções permite diminuir o número de componentes do sistema, o que reduz o seu tamanho
- Consumo reduzido, aplicação de modo de economia de energia a partes do integrado

Muitos fabricantes publicitam arquitecturas totalmente IBM PC compatíveis para controlo embebido[1], que utilizam μPs em vez de μCs: supostamente, o acréscimo de poder de cálculo ao utilizar um novíssimo μP de 32 ou 64 bits, juntamente com algum hardware adicional (extremamente barato) e a utilização de ferramentas de software baratas e muito disponíveis deve contrabalançar o facto de não se utilizar um μC. A pergunta fica: será compensador a troca de um μC de 16 bits por um novíssimo μP de 32 bits?

A oferta de material para PCs embebidos é grande, tanto a nível de hardware como a nível de software. Os preços são extremamente baixos e convidam à "preguiça" de comprar e programar um "familiar" PC para aplicações embebidas **(Shear 1994a, Tyle 1995a)**.

[1] Basicamente placas mãe PC compatível mas planeadas com a preocupação de as tornar mais robustas e duráveis, tanto do ponto de vista mecânico como do eléctrico. Essas placas utilizam os mesmos componentes dos PCs de mesa para tirar proveito dos baixos custos desse tipo de material mas procurando conferir ao conjunto uma aplicabilidade alargada

Muitas vezes utiliza-se o sistema operativo DOS (Shear 1994b) numa arquitectura IBM PC compatível, com o objectivo de utilizar o ambiente de trabalho acolhedor (*user-friendly*) que as actuais ferramentas de trabalho deste S.O. têm. O DOS é um S.O. monoutilizador, não multi-tarefa e pode precisar de muitas modificações para poder ser utilizável em sistemas tempo real. Uma programação apropriada pode, no entanto, contornar todos os problemas do DOS, desde que a aplicação tenha características apropriadas. O mercado disponibiliza vários produtos próprios para suprir estas faltas (ou substituir, mantendo a compatibilidade) dos S.O. mais vulgares como o DOS ou o MS Windows (Tyle 1995a e Tyle 1995b).

A disponibilidade de todo o tipo de acessórios para PC, particularmente nos barramentos standard do PC XT[1] ou PC AT[2], permite poupar muito tempo a desenvolver aquilo que outros possam já estar a comercializar, desde que sirva a aplicação em vista e tenha a dimensão apropriada.

3.4.4.1 ISA (8 e 16 bits)

O barramento que foi denominado ISA (*Industry Standard Architecture*) aparece no IBM PC XT original. É um barramento passivo, de 8 bits e a uma frequência máxima de 4.33 MHz.

Posteriormente, aquando do lançamento do IBM PC AT, esse barramento ganha poder até aos 16 bits, sem perder compatibilidade com o anterior. A velocidade é de cerca de 8 MByte/s (transmissão paralela de 16 bits).

Este barramento continua presente em todos os computadores pessoais pois tornou-se um *standard de facto* (é tão divulgado e utilizado que dificilmente se prescinde dele, exactamente como a IBM o definiu inicialmente).

Devido ao tipo de conexão física utilizada é um barramento que não está preparado para esforços mecânicos, as pistas são muito largas (ocupa muito espaço - para os padrões de hoje) e não está preparado para ser utilizado frequentemente. Há, no entanto, alternativas.

3.4.4.2 PC/104

É um standard que define um novo barramento com posicionamento preciso dentro de placas com dimensões normalizadas, de formato Euro-Card.

[1] Barramento ISA de 8 bits

[2] Barramento ISA de 16 bits

Este barramento foi desenhado para aplicações embebidas, é compacto, passivo, mecanicamente robusto, considera problemas de EMC[1] e é aceite por muitos fabricantes.

Muitos fabricantes de placas mãe para PCs (*motherboards*) industriais começam a incluir nas suas placas barramentos PC/104 para além do ISA.

3.4.4.3 Compact *PCI*

O barramento PCI (*Peripheral Component Interconnect*) é o barramento mais utilizando para trânsito de dados em 64 bits (a nível de PCs de mesa). Todos os IBM PCs e todos os Apple Macintosh produzidos actualmente vêm equipados com este barramento. Este barramento define rígidos parâmetros eléctricos para funcionamento a muito altas frequências com o objectivo de permitir elevada largura de banda no trânsito de informações, para utilização de periféricos com alto desempenho. Dados dos fornecedores de equipamentos asseguram que a performance destes barramentos ronda os 132MB/s em modo de *burst*[2]. O barramento é passivo e existem normas de distâncias físicas e potências de emissão muito exigentes. O espaçamento entre pistas é muito pequeno, o que é positivo porque resulta em pouco espaço ocupado mas torna este barramento pouco robusto a esforços mecânicos.

A variante *compact* PCI redesenha as características mecânicas do barramento e acrescenta cuidados com vista a ser utilizável em ambientes industriais e mesmo em sistemas embebidos. O barramento passa a ter conectores macho e fêmea para ter características mais robustas e passa a existir a possibilidade de trocar placas sem desligar o computador, uma característica muito desejada para todos os barramentos industriais.

Os barramentos PCI e *compact* PCI têm a importante característica de não serem unicamente aplicáveis a arquitecturas x86.

3.4.5 Barramentos VME e VXI

Os barramentos VME e VXI são standards da indústria desde há muitos anos.

O barramento VME é totalmente digital e define vários níveis de sofisticação, não sendo particular a qualquer arquitectura de computador ou processador.

O VXI é um barramento VME acrescido de linhas analógicas. Tudo foi normalizado a pensar na alta velocidade de transmissão de dados, para permitir alta performance na aquisição de dados, de alta qualidade e precisão.

---

[1] EMC - *Electro-Magnetic Compatibility* - Compatibilidade e interferência electromagnética

[2] Transferência em modo rápido

Este tipo de barramentos são caros mas indispensáveis quando se pretende aplicações de alto nível de sofisticação.

3.4.6 Integrados Específicos para a Aplicação

A possibilidade de desenhar circuitos integrados específicos para a aplicação (ASIC[1]) é uma realidade com as novas técnicas de trabalho em VLSI[2] e constitui a resposta à aspiração de resolver um problema de engenharia embebida utilizando um integrado específico para a aplicação em causa.

3.4.6.1 Linguagens de Descrição de Hardware

As linguagens de descrição e simulação de hardware já são utilizadas há muito pelos fabricantes de semicondutores para criarem todos os integrados mais complexos. Estas linguagens de descrição do hardware (HDL - *Hardware Description Language*) permitem simular o funcionamento do hardware, a vários níveis de abstracção. Pode-se assim testar o funcionamento interno e também o funcionamento interligado com outros dispositivos desde que exista a descrição do seu funcionamento na mesma linguagem. Evidentemente, estas ferramentas só se tornam apelativas para fora das empresas que produzem integrados quando se disponibiliza (e eles existem) compiladores de linguagens HDL para hardware.

Actualmente, duas HDLs dominam o mercado: VHDL[3] e Verilog. Existem "compiladores" de qualquer das referidas HDLs para FPGAs (*Field Programmable Gate Array*), dispositivos programáveis, parecidos com as PALs mas bem mais poderosas. Podem ser programadas por métodos automáticos, de acordo com aquilo que foi desenhado na HDL, sem recursos excessivamente dispendiosos (Bursky, 1995a, 1996b). Para atingir as densidades[4] necessárias à obtenção de μC num único integrado, só através dos métodos de fabrico convencionais de silício, que como se sabe, requer recursos muito caros. Este estado da tecnologia é referido por Joshi *et al*(1996) e Dorf *et al* (1993, pág. 1752).

Actualmente já é possível comprar a descrição HDL de alguns μCs, por exemplo da família 80C166 da Siemens. No CD-ROM da Siemens, é referida a existência de uma macrocélula do seu coprocessador Fuzzy, o SAE 81C99A, para integração dentro da

[1] *Application Specific Integrated Circuit*

[2] *Very Large Scale Integration*

[3] *Virtual Hardware Description Language*

[4] Transístores por área de silício ou transístores por integrado. Refira-se um os novíssimos Intel Pentium Pro possuem cerca de 5,5 milhões de transístores no mesmo integrado

arquitectura 166. A política da Siemens é oferecer aos seus clientes a possibilidade de "acrescentos" específicos à aplicação.

3.4.6.2 Futuro Promissor

Extrapolando do que foi anteriormente referido, é possível imaginar a aplicação de uma HDL para o desenho de todo um sistema embebido.

Se fosse possível comprar a descrição HDL de qualquer integrado antes de o comprar (em termos físicos), então seria possível desenhar toda uma placa de circuito impresso (*Printed Circuit Board* - PCB) em ambiente computacional de simulação (CAD[1]/CAE[2]). Obtendo a descrição do μC, da memória, da *Flash* EPROM, dos conversores e da toda a lógica necessária, seria possível montar e simular o conjunto, antecipando muitos problemas que doutra maneira só seriam corrigidos na fase de protótipo. Esta é, realmente, a filosofia que alguns fabricantes empregam para o fabrico interno de placas para os novos processadores[3].

Alargando ainda mais os conceitos anteriores, poder-se-ia, por exemplo, utilizar uma macrocélula[4] do centro de processamento do 8051 e acrescentar o número de temporizadores necessários à aplicação. Qualquer acrescento específico, programado directamente em hardware, seria também possível, talvez até permitindo as quantidades desejadas de RAM e ROM internamente. A biblioteca disponibilizaria os trabalhos anteriores para reutilização e seria também possível comprar "componentes" (macrocélulas) interessantes, de diferentes fornecedores externos. Após uma simulação do conjunto, se se revelasse que um dos blocos era insuficiente, tornar-se-ia possível mudar apenas esse bloco e respectivas ligações, permitindo assim que todo o trabalho anterior fosse aproveitado.

Idealmente, seria até possível juntar dentro do integrado específico as partes analógicas e digitais para obter um projecto embebido dentro de um único integrado.

Para o conjunto ser completo, deveriam existir ferramentas de CAM[5] para a construção do protótipo da placa de circuito impresso (PCB). Deveriam existir compiladores para gerar

---

[1] *Computer Aided Design* - Desenho Assistido por computador

[2] *Computer Aided Engineering* - Engenharia por computador, diz respeito a simulação computacional, etc.

[3] A Siemens, p. ex., tal como referido na pág. 9 do documento publicitário "*High Performance 16-bit µC Family 80C166*", n.º encomenda: B158-H6669-X-X-7400

[4] Conjunto "Caixa Negra" para o qual existe uma descrição das funcionalidades oferecidas

[5] *Computer Aided Manufacturing*, fabrico automático por computador

um ficheiro de construção automática do PCB, que seria depois alterado no Tango[1] ou qualquer outro programa de construção de PCBs para acrescentar conectores, suportes, etc., permitindo depois a construção do protótipo de forma automática.

Para que o conjunto fosse ainda mais poderoso, deveria também ser possível criar uma interface para ferramentas mecatrónicas, de tal forma que todo o projecto físico desde o hardware até à parte mecânica fosse simulada em conjunto, desde o início.

Desta maneira seria fácil, rápido e barato desenhar e construir um sistema com hardware próprio, tal como a maioria dos sistemas embebidos. Claro que a realidade está longe da situação ideal aqui descrita.

3.4.7 Funcionalidades e Poder de Cálculo

3.4.8 Necessidade/Disponibilidade do Poder de Cálculo

A maioria das aplicações embebidas tem um universo limitado de acontecimentos, mas deixou de ser verdade que o poder de cálculo necessário para os gerir seja pequeno, pois a complexidade e a variedade de aplicações aumenta incessantemente (Gien, 1996). É que as imposições de tempo real num sistema controlado por computador obrigam a grandes preocupações de desempenho para garantir o bom funcionamento do sistema a controlar.

Desenvolvem-se sistemas cada vez mais complexos, que necessitam de algoritmos cada vez mais complexos, com temporizações cada vez mais exigentes, multiplicando-se o número de tarefas simultâneas...

De facto, a complexidade dos novos algoritmos de controlo, que pretendem explorar o sistema na sua zona de desempenho máximo, exige poder de cálculo acrescido. Uma das razões porque o controlador PI(D) é muito vulgar é que ele é extremamente leve em termos de peso computacional.

As aplicações que necessitam de grande poder de cálculo existem e o actual baixo preço dos controladores embebidos de elevada performance justifica a construção desses sistemas de controlo, que devem manter o processo na sua zona de performance óptima, o que, por sua vez, resultará em ganhos económicos para o utilizador do processo em causa. Uma das áreas em que o controlo embebido necessita de poder de cálculo é a área da robótica autónoma, onde existe a necessidade de tomada de decisões de forma autónoma, tal como referido por Pereira (1994).

[1] Programa de desenho de PCBs muito conhecido

Apresenta-se, de seguida, algumas características que afectam o desempenho da plataforma de controlo, quer seja um μP ou um μC embebido.

3.4.8.1 Pipelining

Própria dos RISCs e também adoptada pelos CISCs, o *pipelinig* é um passo na execução simultânea de várias instruções dentro do mesmo processador.

O tempo de processamento é dividido em várias tarefas elementares, que serão executadas simultaneamente (Pfeiffer 1996):

1. Busca (*Fetch*): ir buscar à memória um *op-code*
2. Descodificação: identificar a classe de *op-code* e ir buscar parâmetros
3. Execução: cumprir a operação indicada
4. Escrita em atraso (*Write-Back*): escrita do resultado onde apropriado

Havendo lugar a conflitos, a arquitectura tem que resolvê-los.

Quando operações lêem e escrevem ao mesmo tempo, é dada a necessária prioridade à escrita.

O principal problema desta arquitectura são os saltos condicionais: não se sabe à partida qual a próxima instrução a ser executada. Como a condição de salto ainda não foi avaliada, não é possível saber qual o ramo que a execução tomará. Se o processador errar no endereço que foi se foi buscar, então a fila de processamento terá de ser limpa e ter-se-á que esperar para ir buscar o *op-code* ao endereço apropriado, esperar a sua descodificação e só depois continuar a fazer o normal processamento *pipelined*. Alguns processadores mais potentes vão até buscar ambos os ramos do salto condicional, mas o comum é que se opte por ir buscar o endereço para onde se saltou da ultima vez, conseguido-se assim que se a condição mantiver o seu valor lógico não exista perda alguma de performance. Esta maneira de pensar é apropriada para os ciclos das linguagens de alto nível "para", "repetir até" e "enquanto", que saltam repetidas vezes para o mesmo sitio e depois saem para outra zona diferente do programa.

3.4.8.2 Memória

3.4.8.2.1 *Cache* vários níveis

Através de uma estrutura em pirâmide, onde os acessos à memória mais utilizados são mantidos numa memória de elevado desempenho (memória *cache*), conseguem-se acréscimos de velocidade grandes devido ao princípio da localidade dos programas. Este princípio refere um programa raras vezes acede a uma variável sem a reutilizar algumas

vezes de seguida, geralmente contém ciclos e saltos que acabam por levar a que zonas do programa sejam executadas muitas vezes quase de seguida, etc. pelo que se justifica manter os acessos à memória mais utilizados em zonas de *cache*, onde as leituras podem ser feitas rapidamente (e as escritas também), deixando para depois a escrita na mais memória lenta e existente em grande quantidade.

3.4.8.2.2 Dinâmica

A memória dinâmica, conhecida pelo acrónimo DRAM, tem visto o seu preço descer rapidamente devido à explosão do consumo do PC caseiro.

É basicamente constituída por um condensador por bit e esse condensador precisa de ser refrescado periodicamente para colmatar fugas internas.

A utilização de estados de espera (*wait-states*) é necessária para a implementação de sistemas de baixo custo, permitindo a utilização de aparelhos lentos conjuntamente com outros rápidos em estruturas síncronas (tal como a maioria dos processadores).

A memória rápida do tipo *cache* e um elevado número de registos internos, tal como preconizado pelos RISCs, minora os efeitos de estados de espera para a memória.

3.4.8.2.3 Estática

A memória estática é a mais utilizada em sistemas simples, onde não se quer incluir um integrado especial para refrescar as memórias dinâmicas.

Este tipo de memórias é basicamente um flip-flop por bit.

São mais caras, mais rápidas e consomem mais potência eléctrica do que as memórias dinâmicas.

3.4.8.2.4 EDO

Enhanced Data Out.

Só as placas para PC mais modernas apresentam esta particularidade: após uma busca a um certo endereço a memória irá buscar autonomamente o byte seguinte para o caso de ser necessário. Se esse endereço seguinte for realmente pedido, então o ciclo será mais rápido. Doutra maneira tem o mesmo funcionamento que uma DRAM.

Este tipo de memória é pouco mais cara que a memória DRAM.

3.4.8.3 Temporização, Captura, Comparação, MLI

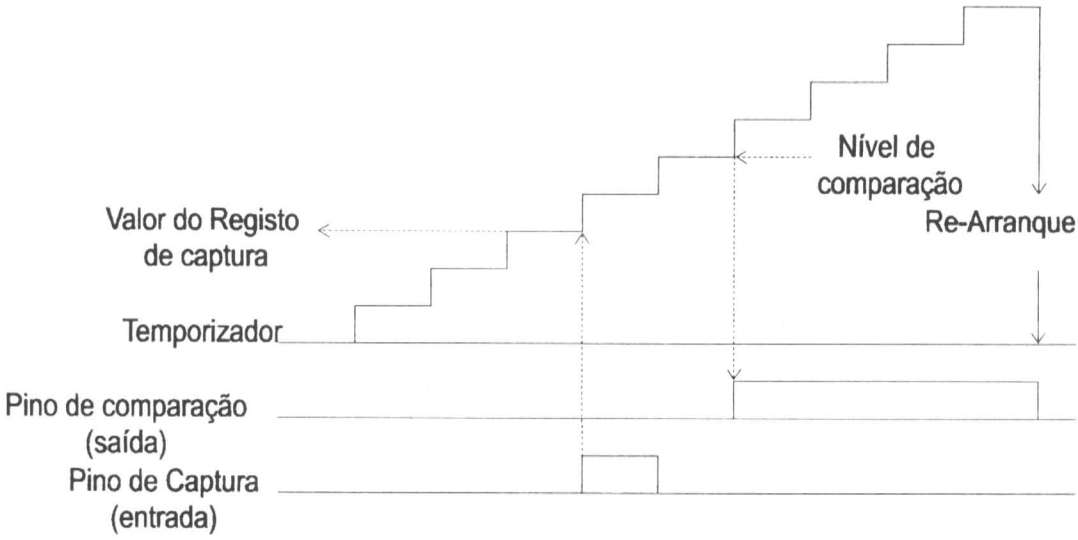


Figura 11: Ilustração gráfica do funcionamento dos modos de comparação e captura de um temporizador

A conhecida estrutura de Temporizador, comparador e captura, dos derivados da família 8051, nomeadamente o 80535, é um exemplo das potencialidade extremamente úteis que se podem incluir dentro de um μC.

Os temporizadores dos vários μC do mercado podem ter muitos modos de funcionamento, tamanhos variados, sentido programável, várias velocidades de contagem, *reload* automático ou não - com ou sem interrupção associada ao *overflow*, etc.

A lógica de comparação associada ao temporizador produz num pino de saída um sinal que é um quando o valor do temporizador é maior que o registo de comparação associado (ver Figura 11). Uma saída de comparação é uma saída em *duty cycle*. Se se aplicar a essa saída um filtro passa baixo, então teremos uma saída analógica (a dinâmica dessa saída depende da impedância de entrada da electrónica a jusante).

A lógica de captura espera um impulso no pino de entrada de captura e quando detecta esse impulso copia o valor instantâneo do temporizador para o registo associado de captura (ver Figura 11). Ao pino de entrada pode estar associado uma interrupção ou não (independentemente da captura).

O funcionamento da lógica de MLI (PWM)[1] é muito semelhante ao de comparação só que neste tipo de dispositivo, o impulso de MLI fica centrado dentro o período de contagem do temporizador. Este impulso é chamado de MLI porque o facto do impulso ficar centrado no tempo permite estratégias de controlo normais em inversores, que, doutra

[1] A téncnica da Modulação da Largura de Impulsos pode ser testemunhada na Figura 17

maneira, seriam conseguidas à custa de lógica externa. Precisamente para evitar essa situação, o 167 oferece 4 canais de PWM, em que se pode fazer um AND interno entre um dos canais e os outros, possibilitando assim um impulso próprio para o comando de inversores, com impulsos centrados e já com "tempo morto[1]".

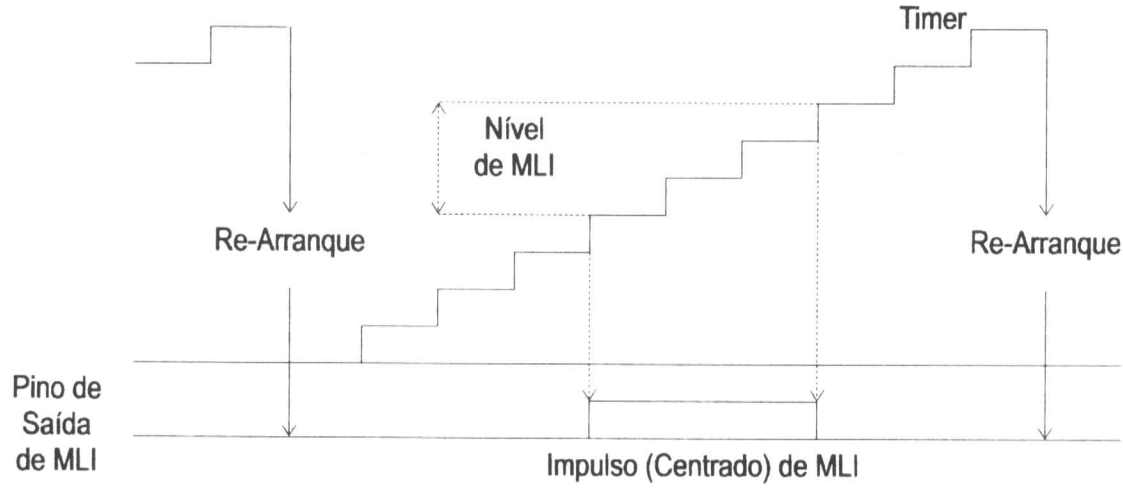


Figura 12: Ilustração gráfica do funcionamento do modo de PWM

3.4.8.4 E/S, analógicas e digitais

Os portos de entrada e saída (E/S) são, porventura, os mais importantes aspectos de um μC, o método para conseguir exercer o controlo.

A maioria dos controladores permite designar a função de, pelo menos, alguns dos seus pinos e programá-los para entrada ou saídas, conforme as necessidades da aplicação.

Geralmente os μC só têm um conversor Analógico Digital (A/D) mas alguns integram também multiplexers analógicos que fazem o agulhamento do canal analógico a ler.

Os conversores Analógico/Digital geralmente integrados nos μC são de aproximações sucessivas e têm referências ajustáveis internamente (por programação), com o objectivo de permitir subir o número de bits da aquisição, à custa de novas aquisições com referências modificadas.

3.4.8.5 Interrupções

Todos os μs permitem saltar para rotinas de interrupção, que lançam em execução rotinas para o tratamento de acontecimentos graves ou assíncronos.

[1] Tempo necessário à saída de condução de um interruptor estático, antes do qual não poderá ser ligado outro interruptor do mesmo ramo, para conversores em forma H

Em sistemas embebidos é comum surgirem muitas fontes de interrupção, pelo que será muito importante programar as suas prioridades, isto é, saber quais são as interrupções que podem interromper as outras.

A gestão de interrupções é um aspecto extremamente importante no tratamento de sistemas com restrições de tempo real muito exigentes e parâmetros tal como o tempo máximo de demora no atendimento da interrupção (Ganssle 1995b), são muitas vezes fulcrais para a escolha de um μ em detrimento de outro (parâmetro também utilizado na selecção de sistemas operativos).

Alguns μCs permitem escolher por software o tipo de interrupção que se pretende utilizar, se ao flanco, e a que flanco, se ao nível e a que nível.

3.4.8.6 Endereçamento e E/S

As arquitectura de endereçamento dividem-se em Harvard ou Von Neuman.

Os DSPs têm em geral, um barramento de dados maior que o barramento de endereços. Tal característica faz sentido quando se pretende poupar silício (baixar o preço) e fazer processamentos repetitivos rápidos com dados, nomeadamente dados em vírgula fixa de elevada precisão e se sabe que a aplicação não será muito grande. No fundo, características próprias dos μC.

Os processadores Harvard endereçam dados e memória de blocos diferentes e têm de existir instruções diferentes para ir buscar dados constantes à memória do programa (que não pode ser escrita nunca). Neste tipo de arquitecturas, o número de bits do barramentos de dados e de endereços pode ser diferente (Bennet 94, secção 3.4.2), tal como acontece nos DSPs e na família 8051.

Os processadores da arquitectura Von Neuman, endereçam apenas um bloco conjunto de memória (dados e programa). Este tipo de arquitectura utiliza o mesmo percurso para dados e endereços.

Ainda outra questão relativa a arquitecturas de endereçamento, é a forma de endereçar E/S. Existem processadores que endereçam separadamente E/S e outros que obrigam a fazer o seu mapeamento em memória, que não pode assim ser aproveitada para dados nem programa. Naturalmente, dotar o μ de uma zona de endereçamento para E/S isolada requer instruções específicas para E/S, que, doutra maneira, seria conseguido com instruções normais de movimentação de dados na memória.

3.4.8.7 Comunicações integradas

Quase todos os μCs actuais incluem portas de comunicação. É essencial comunicar, na programação, na aplicação, para *debugging*, etc..

O protocolo mais utilizado pelo mercado é RS 232, que se transformou num standard, extremamente divulgado e quase universalmente aceite. É um protocolo ponto a ponto mas pode ser facilmente convertido numa rede de campo do tipo 485.

Têm aparecido recentemente vários μC com redes de campo integradas, tal como o caso da Philips com o seu I^2C e da Siemens com o CAN.

O CAN é um protocolo determinístico que está a ter muito sucesso (principalmente na Europa) devido à sua utilização na indústria automóvel. A cablagem no Mercedes classe C foi substituída por uma rede este tipo, com todos os sensores e actuadores ligados à rede, que passa por todos os pontos do carro.

3.4.8.8 Capacidades de Multi-Tarefa

A necessidade de cumprir requisitos de tempo real obriga a frequentes mudanças de contexto do programa principal, que é interrompido frequentes vezes por rotinas com prioridade maior que a sua.

A velocidade de mudança de contexto é afectada por muitos truques de hardware, criados de propósito para acelerar este tipo de mudança:

- A existência de um grupo de registos alternativos (ex.: EXX do Z80)
- A existência de vários grupos de registos genéricos (ex.: tal como no 8051)
- A utilização de parâmetros no *stack* (tal como em alguns DSP's)
- A utilização de registos genéricos sob a forma de *offsets*, face a um apontador de contexto (*context pointer*) – utilizado em nos RISC's (tal como no 80C166)

A abstracção das características do hardware na mudança de contexto pode ser conseguida através da utilização de um S.O. comercial, no entanto o desempenho alcançado está muito limitado pelo hardware, que pode até impor restrições, p. ex. a nível do número máximo de tarefas em execução.

Refira-se adicionalmente que a utilização de demasiadas tarefas a serem executadas pseudo-simultaneamente gasta poder de cálculo e obriga a gastos exagerados a nível de recursos de comunicação entre processos IPC (*Inter process Communication*).

3.4.8.9 Stack Pointer *protegido por hardware*

Saber se ocorreu um erro de *stack* revela-se difícil em software, mas pode ser programado em hardware de forma fácil: depois de cada operação de *stack*, basta comparar o valor do registo *Stack Pointer* (SP) com outros registos que definam o princípio e o fim da zona de *stack*. Esta foi a aproximação que o Siemens 166 tomou para ajudar o programador a eliminar este tipo de erros do sistema. Será ainda importante saber exactamente quando se excedeu os limites do *stack* porque geralmente esses erros são difíceis de detectar e fáceis de confundir com outro tipo de problemas. De facto, esta maneira de agir está relacionada com a filosofia RISC, mais registos internos, mais verificações sobre menos instruções implementadas, etc..

Repare-se como a maneira de pensar característica dos S.E., ao requerer fiabilidade total obrigou a que esta interessante característica aparecesse (do que se conhece) primeiro nos μC que nos μP (geralmente as novidades são introduzidas primeiro nos μP e só depois nos μC).

3.4.9 Ferramentas para desenvolvimento

O desenvolvimento de cartas de processador é muito atraente para aplicações embebidas pois permite a integração de alguma electrónica à medida da aplicação juntamente com a plataforma de processamento. Este desenvolvimento é, no entanto, um processo tão complexo que por vezes acaba por sair mais caro o desenvolvimento que o preço de mercado de uma placa equivalente. Este facto é agravado tendo em conta que os novos, poderosos e complexos μs do mercado têm muitas vezes caixas próprias para montagem superficial, em que os pinos estão muito próximos o que impede quase por completo a sua montagem manual.

A construção de um protótipo em circuito impresso (PCB - *Printed Circuit Board*) exige algum esforço para fazer a montagem de todos os componentes. Dificilmente o conjunto funcionará globalmente à primeira tentativa, mas as partes que funcionam correctamente são um ponto de partida importante para o processo de eliminação dos erros (Ganssle 1993, Blakeslee 1993). Depois da reparação de todo o hardware, será provável acontecer uma pista partida, uma solda fria[1] ou qualquer outro problema, verificando-se assim a famosa lei de Murphy[2].

[1] "Solda Fria" - fenómeno frequente quando se solda erradamente um componente, em que a existência de óxidos (isoladores) entre o componente e a solda leva a que exista um contacto físico mas não eléctrico, que é um problema difícil de detectar

[2] A lei dos pessimistas: se alguma coisa pode correr mal, correrá de certeza nas piores circunstâncias

Para fazer face ao desenvolvimento de hardware existem ferramentas sofisticadas que se revelam essenciais: Os ICE (*In-Circuit-Emulator*) e os analisadores lógicos.

Um analisador lógico é um osciloscópio de memória, digital, com muitos canais de entrada e que permite fazer *debugging* de protocolos de comunicações digitais ou do funcionamento da periferia do processador.

Para testes completos de *debugging* em T.R., com o μ a executar e sem o parar (essencial para os S.E. e para sistemas em T.R.), a ferramenta apropriada é o ICE (Ganssle 1990a, Blakeslee 1993). Apesar de dispendioso, é, de facto, a solução, especialmente se se usa processadores que não têm qualquer tipo de ajuda interna para *debugging* (a maioria deles). O ICE utiliza os sinais da periferia do processador e eventualmente da EPROM para analisar o funcionamento de software do μ. Como seria de esperar, um certo ICE só funciona para uma certa plataforma de processamento.

Debugging de software via software pode ser conseguido utilizando provimentos específicos das arquitecturas que permitem modos de *debugging,* tal como a execução do programa passo a passo e a chamada de monitores em *breakpoints* de hardware[1] (Ganssle, 1993). Esta interessante característica é aproveitada, entre outros programas, pelo *Turbo Remote Debugger* para a família de IBM PC, que executa o software a testar e envia os dados para um outro PC, que serve de analisador de software. Também o *Background Debugging Mode* (BDM) dos μs da Motorola (683xx e 68HC11) permite interessantes características: o porto de BDM é constituído por 3 pinos para comunicações série que levam informação acerca do estado actual do μ embebido para uma plataforma externa, que pode até modificar esse estado (Ganssle 1993).

Programar numa linguagem de alto nível exige ferramentas de *debugging* apropriadas para lidar com outra linguagem que não *assembly*. O mais apropriado é mesmo adquirir o conjunto compilador e *debugger* da linguagem de alto nível simultaneamente.

A utilização de um "cordão umbilical" para fazer a programação e *debugging* de S.E. é um método muito utilizado em PCs embebidos e é uma boa maneira de obter uma ferramenta de *debugging* barata, útil para a fase avançada de eliminação de erros. Naturalmente, é necessário que as comunicações funcionem e que o processador mande informação frequente, para que seja possível saber o que se passa no sistema. Se existirem *breakpoints* de hardware, a tarefa pode ser um pouco mais fácil, mas o esforço do envio de dados será

[1] O Hardware encarrega-se de, ao chegar a um certo endereço, chamar a rotina de monitorização

sempre da aplicação. Por esse mesmo motivo, não será fácil ou pode até ser impossível detectar alguns tipos de erros através deste método.

Tendo em conta que, na generalidade dos casos, não é possível parar um processador embebido, mesmo que momentaneamente, devido à dinâmica do sistema (Tempo REAL), apenas a utilização de hardware específico poderá ajudar ao *debugging*. A solução pode passar pela utilização de um analisador lógico ou ainda um ICE (Blakeslee 1993).

A utilização de simuladores de software (ex.: C-SPY-8051 ou D-SCOPE-166) para retirar erros de plataformas embebidas é útil mas também tem um espectro de aplicação muito limitado. Se o mau funcionamento for proveniente de interacções com hardware externo, então será muito difícil ou mesmo impossível encontrar o erro. Seria diferente se existisse um simulador construído à medida da aplicação, que simulasse o μ e todo o hardware que o rodeia nesta aplicação, compilando o mesmo programa para hardware e para a simulação de todo o conjunto. Tal foi realizado em (Sousa *et al*, 1994), com relativo sucesso e sem demasiado trabalho, à custa da utilização de uma linguagem de alto nível, portável para outras plataformas, como a linguagem C.

Diagnosticar o que se passa num sistema embebido é difícil, particularmente se o problema é relacionado com as restrições de temporais não serem cumpridas. Este problema exige apurado tratamento específico e muitas vezes se fazem módulos do programa só para se fazer *debugging* eficiente. Um ICE conseguirá fornecer importantes pistas acerca do que se passa dentro do processador e ao mesmo tempo pesquisar a sua periferia, juntando informação necessária para relacionar situações complexas sem alterar o funcionamento do sistema no seu conjunto (Blakeslee 1993) e (Ganssle 1990a).

No caso do *debugging* incluir também um S.O. dedicado para a aplicação, então, provavelmente, será necessário ainda mais tempo para retirar todos os erros do S.O., da aplicação e do conjunto.

Em resumo (Blakeslee 1993, Ganssle 1990a, Ganssle 1994b, Ganssle 1993):

- Antes de começar a escrever software para a aplicação, testar bem todo o hardware
- Escrever rotinas de teste exaustivo do sistema, incluindo o teste de actuadores e sensores em toda a sua gama de funcionamento (se possível)
- Manter as rotinas automáticas de teste e diagnóstico do hardware na aplicação final, p. ex. durante o arranque, enviar mensagens de erro compreensíveis
- Se for preciso um S.O., o melhor é comprá-lo. Se não for possível, então testá-lo até se ter a certeza que funciona 100% bem antes de prosseguir

- Incluir na aplicação código a pensar que vai ser necessário fazer *debugging*, talvez implementar mesmo um cordão umbilical. Enviar o estado do sistema e mensagens de erro compreensíveis pelo cordão umbilical ajuda o desenvolvimento da aplicação e principalmente a sua futura manutenção
- Quando falham outros métodos o osciloscópio digital, o analisador lógico e principalmente o ICE são indispensáveis
- Ser metódico e objectivo e científico na procura dos erros (Blakeslee 1993, Ganssle 1994b)

3.5 Resumo

A engenharia mecatrónica apresenta o conceito de um ambiente de simulação integrado para electrónica e mecânica. Para além do conceito, em si, ser interessante para o trabalho em sistemas embebidos com um forte relacionamento com dispositivos mecânicos, a filosofia de que estes assuntos devem ter um estudo global é estendida ao relacionamento próximo que existe entre hardware e software para os S.E.. Foi proposta uma forma de actuação para o desenho de projectos embebidos que é aplicável com qualquer método estruturado de análise. O referido método deve permitir a abstracção da implementação, o que, por seu turno, irá permitir projectos correctamente equilibrados. O planeamento do projecto deve ser cuidadoso e a produção de documentação normalizada é essencial para permitir conclusões rápidas acerca da introdução de novos dados, lançados p. ex. por necessidades da implementação.

O desenvolvimento do software para S.E. é complexo. A necessidade de cumprir as restrições de T.R. exige recursos a nível de S.O., apresentando-se uma lista das características desejáveis para os S.O. de projectos embebidos.

A nível de hardware justificou-se a busca de poder de cálculo acrescido. Foram discutidas as várias cambiantes internas dos processadores mais vulgares, bem como a maioria dos recursos comuns em microcontroladores. Algumas das famílias mais conhecidas de controladores foram mencionadas e também utilização da arquitectura IBM PC em S.E..

A relativa dificuldade da implementação real de hardware específico, necessário a grande parte dos S.E. foi também discutida, mencionando-se ainda as ferramentas necessárias para que tal tarefa possa ser possível de forma fácil.

PARTE II

Experimentação

e

Controlo

4. Apresentação do Caso Prático

4.1 O processo

No âmbito de um anterior projecto de Investigação e Desenvolvimento (I&D), a Faculdade de Engenharia da Universidade do Porto (FEUP) desenvolveu, em conjunto com a EFACEC (Transformadores Industriais) e outros ainda outros parceiros, um complexo e inovador sistema de secagem de transformadores a óleo para distribuição de energia eléctrica (Quintas *et al*, 1994).

Os transformadores a óleo devem ser secos por razões qualidade e fiabilidade de funcionamento.

O sistema que se pretende melhorar é o controlador de temperatura deste processo.

O processo de secagem utiliza um inversor de tensão para introduzir corrente nos enrolamentos do transformador, que tem o secundário em curto-circuito. Deste modo, existe um auto aquecimento dos enrolamentos do transformador, que permite vaporizar os líquidos presentes nas peças construtivas. Para permitir a rápida eliminação dos vapores existentes nas imediações do sistema, o transformador a secar é mantido em vácuo durante o processo de secagem. A frequência de alimentação utilizada no inversor é fixa em 5 Hz para que a tensão induzida no secundário seja baixa e assim existam correntes de aquecimento tanto no(s) enrolamento(s) primário(s) como no(s) secundário(s) do transformador. Doutra maneira, as correntes no primário seriam muito baixas e não causariam auto-aquecimento significativo. Os transformadores de distribuição destas potências são, normalmente, trifásicos mas o sistema funciona também com transformadores monofásicos.

O processo de secagem utiliza 2 cubas de vácuo com capacidade máxima de 5 transformadores. Os 5 inversores que se pretende comandar podem ser ligados a qualquer das duas cubas. O número real de transformadores presente depende das dimensões físicas e do planeamento da produção.

Os parâmetros dos transformadores a construir variam (no limite) peça a peça conforme requisitos do cliente final. Alguns parâmetros são eléctricos, tal como tensão no enrolamento primário e secundário, respectivas correntes que traduzem uma certa potência de funcionamento, tensão de isolamento, etc.. Existem, no entanto, uma série de pormenores construtivos não eléctricos muito importantes (do ponto de vista do

controlador de temperatura), tal como a temperatura máxima a que o transformador deverá funcionar, forma física dos núcleos magnéticos, disposição física do fio condutor, espaços e formato dos canais do interior dos enrolamentos, etc.. Muitos outros pormenores tal como configuração dos pontos de fixação, isoladores pretendidos, ligações eléctricas, etc. não interessam ao controlador de temperatura mas exigem que o trabalho de montagem do conjunto do transformador seja, em grande parte, manual.

Os transformadores em causa têm dimensões muito variáveis que podem ir, em termos de volume comparativo, desde 1 até cerca de 15 m^3. Apresentam componentes em madeira e/ou papel, por razões de custo e funcionalidade. As quantidades destes componentes, que aparecem imprevisivelmente húmidos, varia de transformador para transformador, tal como a sua configuração e colocação exacta. A humidade dos componentes varia com a sua proveniência e com estação seca ou húmida do ano o que pode mesmo reflectir-se no processo de secagem de alguns transformadores.

Os enrolamentos, que podem ser de alumínio ou cobre, variam em forma e dimensão de acordo com os referidos factores construtivos. Os enrolamentos são construídos já com a preocupação de que será necessário secá-los e assim existem canais de formato variado que servem, entre outros propósitos, a necessidade de escoar o vapores formados durante a secagem.

Devido à elevada flexibilidade necessária durante a produção, grande parte dos ajustes e apertos terão sido feitos por trabalhadores, pelo que se admitem diferentes folgas e pequenas diferenças entre peças que poderiam aparentemente ser iguais.

Do ponto de vista termodinâmico e dos fenómenos de transferência envolvidos, a modelização matemática deste sistema apresenta-se algo complexa. Tal modelização obrigaria a estabelecer grande número de simplificações e pressupostos, além de que muitas informações relevantes nem sequer estão disponíveis, tal como os valores dimensionais de todos os parâmetros dos enrolamentos, tendo em conta que um número variável de transformadores são secos em simultâneo, também a temperatura dos outros transformadores que estão na mesma cuba ou ainda o valor da pressão actual. Por outro lado, a quantidade de humidades na madeira e no papel presente no transformador é indeterminada. Tendo em conta todas estas dificuldades concluiu-se que tal formulação matemática seria impraticável, no âmbito deste trabalho. A modelização matemática do processo de secagem de transformadores foi objecto de estudo preliminar anterior e o

grupo de trabalho envolvido concluiu também que tal matematização do sistema não era viável. Não foi possível obter qualquer relatório sobre este estudo.

Durante a secagem, a temperatura do transformador em aquecimento é determinada através do cálculo da resistência do enrolamento, conseguido através da medição de amplitude e diferença de fase da corrente, face à tensão. A EFACEC fornece as seguintes fórmulas para o cálculo da temperatura actual, no caso dos enrolamentos em cobre:

$$T_2 = \frac{R_1}{R_2}(235 + T_1) - 235 \qquad (1)$$

T_1 e T_2 são, respectivamente, as temperaturas inicial e final

R_1 e R_2 são, respectivamente, as resistências inicial e final

Para o caso dos enrolamentos em alumínio:

$$T_2 = \frac{R_1}{R_2}(225 + T_1) - 225 \qquad (2)$$

A grande vantagem deste método de medição de temperatura é eliminar todos os sensores de temperatura. Tal é importante não só por considerações económicas mas também para permitir o enchimento automático de óleo na fase seguinte do processo.

A grande desvantagem do método proposto é que a medição de temperatura depende inteiramente na medida inicial da resistência dos enrolamentos, a uma temperatura que é considerada como conhecida mas é inserida manualmente pelo operador.

A medida da temperatura[1] obtida é o integral de elementos infinitesimais de temperatura, ao longo de cada um dos enrolamentos do transformador. A medida de temperatura, que é monofásica, sucede-se nas fases (se o transformador for trifásico). Assim, a existência de um eventual ponto quente, afectaria decididamente a medida de temperatura obtida. Durante a fase de desenvolvimento inicial do projecto, comprovou-se que a temperatura medida é, na prática, apropriada e o dia a dia demonstra-o também.

O objectivo do sistema é elevar a temperatura dos enrolamentos até cerca de 110°C, mas de forma a nunca ultrapassar os 130°C - tal resultaria, provavelmente, em queimaduras no verniz isolador dos enrolamentos do transformador, o que obrigaria à sua substituição integral. Como indicação, refira-se que o custo mínimo dos transformadores que passam por este equipamento ronda os 5 mil contos.

[1] Fala-se indiferentemente de temperatura ou de resistência

Foi desenvolvido um sistema embebido para comandar esta aplicação, que tem vários requisitos. É necessário cumprir os limites de corrente dos enrolamentos, fazer medidas das temperaturas, comandar o inversor, gerir comunicações com exterior, monitorizar medidas e sua evolução para detectar defeitos e avarias, providenciar informação para o exterior (gerir o monitor de LCD's), tratar entradas (teclado, etc.), gerar referências, etc..

Quando a temperatura do transformador está longe da temperatura final pretendida, 110°C, é aplicada a corrente máxima aos enrolamentos, definida pela EFACEC como 120% da corrente nominal (I_N). Esta limitação no valor máximo da corrente visa proteger os enrolamentos de esforços mecânicos excessivos, que poderiam surgir pela existência de demasiados vapores em ebulição no interior do sistema.

Supondo, por aproximação que os vapores extraídos do transformador são de água, o seu ponto de ebulição varia muito conforme a pressão existente dentro da cuba. A consulta de tabelas apropriadas, tal como em Smit *et al* (1987, tabela C.1) assegura que a água (pura) entra no estado gasoso a temperaturas inferiores a 19°C desde que a pressões seja inferior a 2 kPa (aproximadamente 20 mili bar). Se a pressão na cuba de secagem não atingir os 20 mili bar ao fim de 20 minutos de processo, a secagem é interrompida e o controlador do processo de vazio activa um alarme. Evidentemente, o transformador vai aquecendo de forma a trazer a humidade do interior dos materiais até à sua periferia, de forma a poder ser eliminada pelas bombas de vácuo. Em conclusão, o ritmo a que o vapor de água é eliminado das proximidades do sistema varia com a pressão na cuba e a temperatura do transformador. O valor da pressão na cuba é desconhecido do controlador do processo de aquecimento de transformadores.

Uma das maiores dificuldades do processo que se tem vindo a descrever reside no facto de que, uma vez retirada a humidade das proximidades do sistema, o transformador perdeu alguma massa (foram retirados líquidos). Se se continuar a enviar potência para o sistema ao ritmo anterior, a sua temperatura subirá rapidamente, com o perigo de ultrapassar a temperatura máxima admissível.

A evolução da pressão no interior da cuba é dificilmente previsível já que o arranque das bombas de vácuo é feito manualmente, pelo que podem existir esquecimentos, atrasos, etc. e depende ainda do estado de uso do óleo utilizado nas bombas de vácuo para fazer o isolamento entre baixa e alta pressão. Tal como referido anteriormente, será disparado um alarme se a pressão for superior a 20 mbar passados cerca de 20 minutos do arranque das bombas. O sistema de vazio foi montado, segundo projecto conjunto das partes

envolvidas no projecto inicial, pela Hedrich, (firma alemã que fornece este tipo de equipamentos industriais) e é controlado por um autómato programável (A. P.) Siemens 115. O sistema da Hedrich inclui uma interface gráfica para comando das partes de vazio e permite enviar dados para o sistema de secagem (ver mais à frente "4.3 Aquisição de dados - a envolvente fabril"). Existem duas bombas de vazio primário (uma para cada cuba) e duas bombas extra, que podem ser ligadas a uma das cubas para formar alto-vácuo (Figura 18). O processo de secagem de transformadores demora 3 horas. Depois desse período, a alimentação eléctrica é desligada e as bombas de alto-vácuo são ligadas até à pressão baixar dos 5 mbar. Passado algum tempo, apenas com uma bomba de alto-vácuo ligada, e de acordo com instruções do operador, o processo de enchimento com o tipo de óleo certo e a quantidade apropriada para o transformador em causa inicia-se.

A todo este conjunto chamou-se CoPSE, Controlo do Processo de Secagem e Enchimento (de Transformadores de Distribuição).

4.2 Controlo Convencional

O sistema CoPSE, em funcionamento há cerca de 6 anos, padece de diversas dificuldades a nível de hardware e de software.

O sistema de controlo baseia-se em inúmeras regras *If... Then... Else...* que foram sendo modificadas ao longo dos anos, conforme o autor ia ganhando conhecimentos acerca do comportamento do processo, para lidar com as especificidades do processo em causa. Entre as temperaturas de 80 e 111°C o controlo é conseguido através de um controlador convencional do tipo Proporcional + Integral (PI)[1], implementação de velocidade, tal como referido por **Bennet (1994, secção 4.4.3)**. Fora desta faixa de temperaturas, o sistema é totalmente não linear: para temperaturas abaixo de 80°C são aplicadas correntes que "aquecem qualquer transformador"; acima de 111°C existem vários escalões que devem "arrefecer o transformador". Este sistema sucede a um PI convencional que foi retirado de funcionamento porque apresentava características de instabilidade e/ou embalamento (*wind-up*) quando se secavam transformadores de tamanho muito pequeno ou muito grande (pois o controlador tinha sido afinado para os transformadores de tamanho médio). Um controlador PI(D) não é apropriado para este sistema porque os parâmetros (em termos de Teoria dos Sistemas) que relacionam tempo, potência debitada e temperatura variam muito entre transformadores, variam mesmo entre transformadores

[1] Para aumentar a robustez do algoritmo foi eliminada a componente Derivativa (D) destes controladores

de parâmetros eléctricos iguais e variam ainda para o mesmo transformador durante a secagem. Será, portanto, de esperar que a utilização de controladores convencionais do tipo PI(D) se revele pouco apropriada. Naturalmente, a primeira acção para tornar o controlo mais robusto foi anular a componente derivativa, mas um único sistema de controlo PI será sempre pouco apropriado para um sistema onde a resistência dos enrolamentos varia continuamente entre os vários transformadores numa gama de 0,1 a 1kΩ e onde um certo transformador, durante a secagem, varia a sua resistência na proporção de 1 para 1.3.

Como o processo de aquecimento é baseado em perdas por efeito de Joule, ou seja, o acréscimo de temperatura é proporcional a RI^2, a função de transferência (temperatura, corrente) será não linear[1] com certeza absoluta, mesmo que o valor de R, resistência dos enrolamentos, fosse considerado constante. Tal deve-se ao facto da grandeza de entrada no processo, aquela que se pode controlar, ser a corrente e não a potência. Pode então dizer-se que o processo que se pretende controlar é não linear e variante no tempo.

Um problema adicional resulta da forma como é feita a medida de temperatura no processo. Sendo a medida de temperatura baseada na corrente que circula nos enrolamentos, só é possível fazer medições enquanto se aquece o transformador. Este problema é ainda agravado pelo facto de que, quando o transformador é percorrido por correntes baixas, este entra numa zona de magnetização francamente linear, onde não existem estudos acerca do comportamento do transformador[2]. O funcionamento nesta zona de baixas correntes faz com que passem a existir um número elevado de harmónicos que por sua vez, impedem o cálculo preciso da amplitude e fase da corrente face à tensão, o que impede o cálculo da temperatura. De facto, comprova-se que a temperatura medida com correntes abaixo de metade da corrente nominal do transformador não é fiável. Este processo apresenta, pois, a invulgar característica de não permitir medidas sem uma apreciável actuação sobre o sistema.

Para contornar o problema de não se poder operar com correntes muito baixas, opta-se por iniciar um funcionamento com um *duty-cycle* variável, a fazer um liga/desliga de tal modo que o valor médio da corrente seja o apropriado para manter a temperatura o mais constante possível. Refira-se, por curiosidade, que, na prática, esta solução foi inventada

[1] Um sistema linear é um sistema que pode ser descrito por equações diferenciais lineares, cujos coeficientes são constantes ou apenas funções da variável independente. Só neste tipo de sistemas é que é válido o teorema da sobreposição.

[2] Podem ser retiradas analogias da alimentação a frequência variável, baixa amplitude, do comportamento do motor assíncrono trifásico, que pode ser modelizado como um transformador com o rótor em curto-circuito (tal como é o caso)

pelos operários que impediam a subida excessiva da temperatura desligando e ligando a alimentação da potência do inversor, controlando assim o processo, de acordo com os seus conhecimentos de "peritos". Devido ao modo de funcionamento descrito, o processo não pode ser considerado (no âmbito da Teoria dos Sistemas -T.S.) um sistema discreto no tempo, no sentido em que as medidas não chegam a uma cadência regular nem tão pouco surgem sem actuação sobre o sistema.

Para o arranque do sistema, o operador insere os parâmetros eléctricos do transformador e a temperatura inicial. O sistema de controlo de aquecimento arranca com o cálculo da resistência inicial dos enrolamentos, que é sempre feito a 100% da corrente nominal do transformador (I_N) durante 30 segundos, considerando que não existe auto-aquecimento durante a medida inicial. Posteriormente, é medida a percentagem de desequilíbrio entre as 3 fases do transformador (se ele for trifásico). Se este valor estiver dentro dos limites, o processo continua, senão é lançado um alarme. Na fase inicial, quando a temperatura estiver muito baixa, p. ex. abaixo dos 80°C, então a corrente injectada é a corrente máxima, 120% I_N., pelas razões atrás referidas. Só quando a temperatura está próxima dos 110°C existe a necessidade de um controlador de temperatura. Por razões de segurança existem condições de saturação que limitam, por software, em qualquer altura do programa, a corrente a 220 A ou a 120% I_N, o que for menor. Para além destes limitadores por software, existem protecções de hardware que são ajustadas pelo processador, durante o início do processo, para o mais baixo dos dois valores: 220 A ou 1,3 I_N.

Como já foi referido, o sistema calcula a temperatura com base na amplitude e fase da corrente aos 5 Hz, face à tensão gerada. Tal tarefa não é fácil quando a forma de onda real contém bastantes harmónicos. Para determinar a amplitude e fase do harmónico principal, aos 5 Hz, é utilizada uma DFT (transformada discreta de Fourier – *Discrete Fourier Transform*). Este é o melhor método quando se adquire a corrente em apenas 3 pontos. Assume-se que não existem harmónicos pares e assim considera-se que existe simetria de meia-onda; sabe-se também que os pontos t=0 e t=0,1 segundo da onda são obrigatoriamente nulos, tal como se pode ver na Figura 13 e na Figura 14.

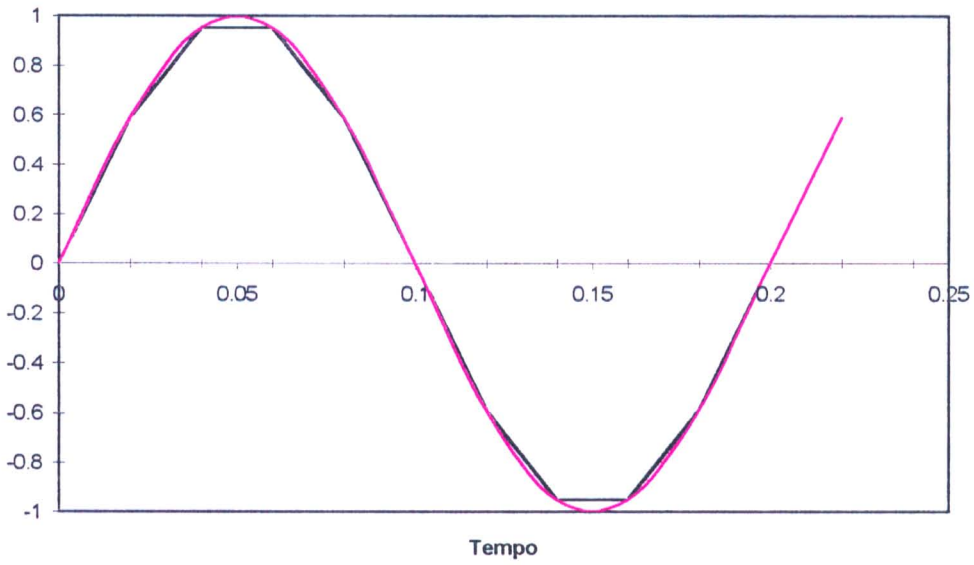


Figura 13: Sinusóide ideal amostrada a 50 Hz

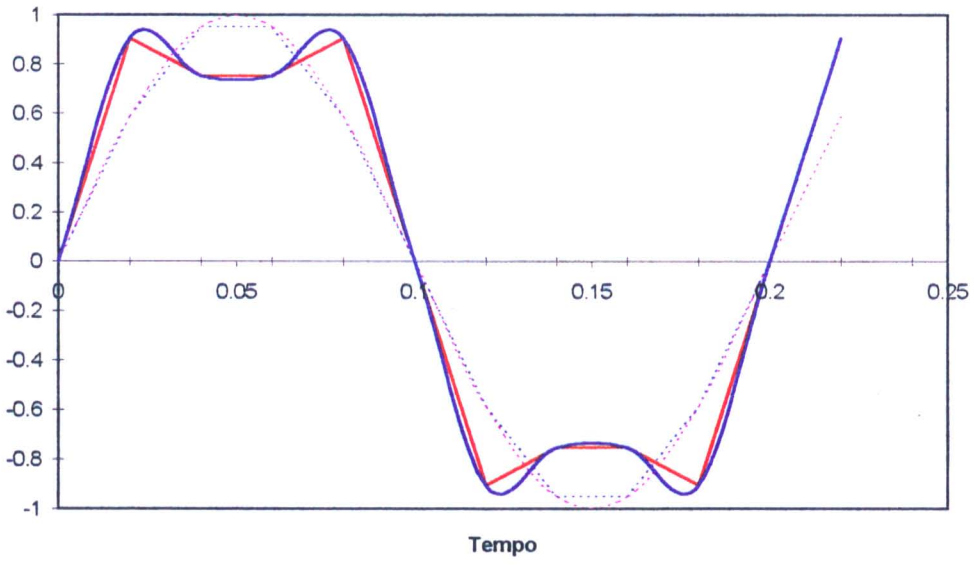


Figura 14: Sinusóide fortemente afectada pelo 3º harmónico, amostrada a 50 Hz

Para o número de pontos em causa, o cálculo duma FFT[1] (*Fast Fourier Transform*) é computacionalmente mais pesado que o da DFT correspondente. Segundo o teorema da amostragem, a frequência máxima que podemos ter com uma amostragem de 50Hz é de 25Hz. Na prática, muitas vezes utilizam-se frequências de amostragem 5 vezes maiores que a frequência máxima a observar. Como se pretende amostrar uma onda de 5 Hz, os limites são cumpridos. A existência de harmónicos de ordem superior à 5ª levaria à ocorrência do fenómeno do *aliasing*, se não existisse um filtro analógico. No entanto, existe um filtro

[1] Algoritmo muito utilizado para o cálculo optimizado de DFT de sinais com muitas amostras

analógico *anti-aliasing*, que garante a atenuação suficiente de frequências superiores a 25 Hz para que a temperatura não venha afectada significativamente.

De referir que, no conjunto, com a manipulação da cadeia de medida e a conversão analógica digital de 10 bits, a medida de corrente é feita com uma precisão global de 12 bits em toda a sua escala de 0 até 220 A_{RMS}, a corrente máxima que os inversores debitam em segurança. O valor do Bit Menos Significativo é de $220/2^{12}=0.053$ A. Como a corrente mínima a medir é de cerca de 0,6 A, então a precisão global do sistema na medida de corrente será +/- 5%. A precisão na medida da temperatura será sempre pior devido ao erro na medição da resistência inicial dos enrolamentos. Adicionalmente, existe o erro na temperatura que o operador introduz no início do processo (equações 1 e 2).

Testes feitos no período de arranque do sistema confirmam que esta cadeia de medida consegue medir a temperatura do transformador de forma apropriada.

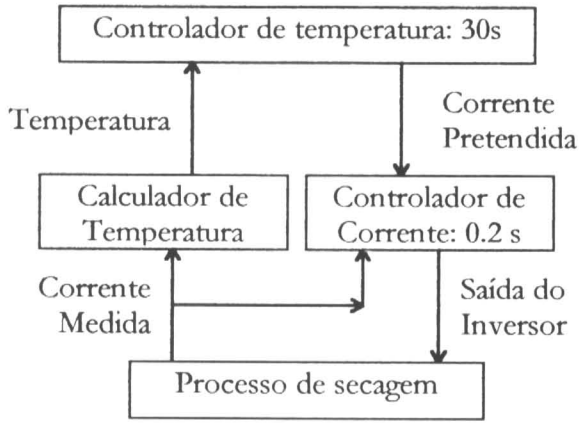


Figura 15: Esquema de princípio da hierarquia de controladores envolvidos no CoPSE (o esquema real é bastante mais complexo)

Tal como referido anteriormente, o CoPSE tem por objectivo controlar a corrente de tal forma que a temperatura seja mantida perto dos 110°C. Devido à utilização de um inversor de tensão, tem que existir uma malha de controlo interior, rápida que se encarregue de estabilizar o valor da corrente rapidamente (este assunto é desenvolvido em 8-Implementação, pág. 103). O Controlador de corrente é chamado à frequência de 5 vezes por segundo, enquanto que o controlador de temperatura é chamado de 30 em 30 segundos, quando existe corrente. O calculador de temperatura é chamado continuamente, para a fase seguinte, sempre que medida anterior está concluída. O ciclo de cálculo da temperatura não é síncrono pois só se aproveitam valores estabilizados de corrente. A amostragem da corrente que serve de base à temperatura é feita à cadência de 50 Hz. O esquema simplificado de funcionamento dos controladores no CoPSE é apresentado na Figura 15.

4.3 Aquisição de dados - a envolvente fabril

Na nave industrial onde este trabalho foi desenvolvido circulam AGVs[1] e existe um armazém automatizado de peças semi fabricadas. A zona é considerada limpa por padrões europeus, onde o uso de protectores para o calçado é obrigatório. Este ambiente já acolheu projectos de âmbito europeu, nomeadamente da série CNMA (*Communication Networks for Manufacturing Applications*), e mais recentemente o projecto Esprit 7096 intitulado "*CIME*[2] *Computing Environment Integrating a Communications Network for Manufacturing Applications*". Estes projectos movimentaram sofisticados recursos informáticos que, no entanto, nem sempre se mantinham depois da sua conclusão.

Por exemplo, verificou-se ser impossível montar de modo fiável a rede de comunicações que permitiria levar os dados do controlador do processo até um PC (ou qualquer outro sistema) onde fosse possível armazenar esses dados para posterior estudo. Felizmente, todos os equipamentos foram preparados para comunicar e assim quebrar as barreiras das várias tecnologias utilizadas, prontos para o funcionamento integrado por rede de computadores, segundo o conceito CIM[3]. As dificuldades de interligação de equipamentos de produção é grande e o conjunto instalado, que está a ser descrito tal como existe efectivamente no quotidiano da fábrica, demonstra como não é fácil interligar os vários tipos de equipamentos, proveniente de diversos fabricantes.

Os armários de electrónica que constituem o sistema de aquecimento, a que se chama fontes, contêm um rectificador controlador a tiristores, um barramento intermédio de tensão contínua e um inversor a IGBTs[4], controlado por um integrado dedicado, gerador de MLI (PWM)[5].

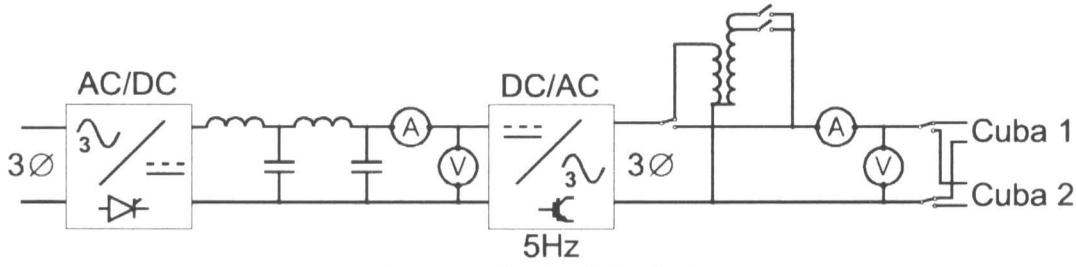


Figura 16: Circuito de Potência

[1] *Automated Guided Vehicle* Veículo Filo Guiado - segue uma guia mas movimenta-se de forma autónoma

[2] *Computer Integrated Manufacturing and Engineering*

[3] *Computer Integrated Manufacturing*

[4] *Insulated Gate Bipolar Transistor* - Transistor bipolar de porta isolada. Um dos tipos de interruptores estáticos de potência

[5] *PWM-Pulse Width Modulation*, em português: Modulação de Largura dos Impulsos. Para o conjunto em causa, este método permite obter uma quase sinusóide de corrente, a partir de impulsos de tensão contínua que alterna de polaridade a alta frequências através da utilização de interruptores estáticos (electrónica de potência), tal como representado na Figura 17

Devido à utilização de um inversor de tensão, a amplitude da corrente na carga (o transformador que se pretende secar) é resultado do índice de modulação (Figura 17) da tensão utilizada no inversor e da tensão contínua do barramento intermédio (Figura 16). Para permitir uma gama ainda maior de excursão na tensão, existe um auto transformador de saída que pode elevar a tensão acima da tensão da tensão de alimentação do inversor (ver Figura 16). A tensão no barramento intermédio depende do ângulo de disparo dos tiristores que constituem os bloco rectificador (Figura 16). Esse ângulo de disparo é gerado por uma malha de realimentação analógica[1], que segue uma referência analógica para o valor da tensão contínua pretendida no barramento intermédio.

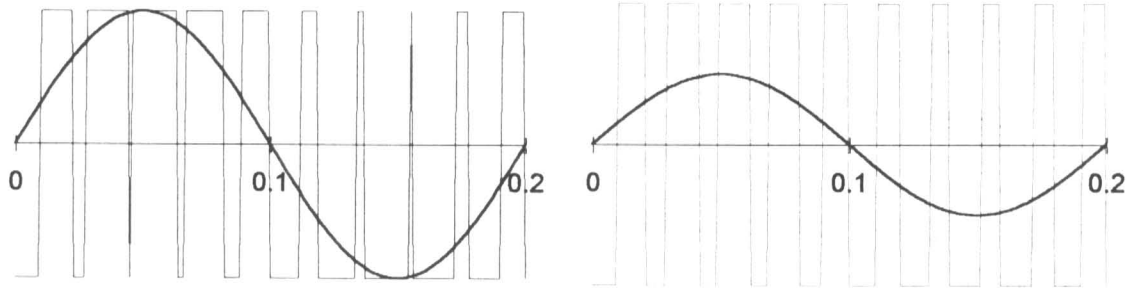


Figura 17: Modulação de Largura de Impulsos (MLI) mostrando na figura da esquerda um Índice de Modulação maior que o da figura da direita. A tensão MLI da esquerda geraria numa hipotética carga uma corrente menor que a tensão MLI da direita

Todo este conjunto, incluindo a aquisição de dados e comando da cadeia de medida, protecções, interface com o utilizador (entrada e visualização de dados), supervisão do sistema, geração de referências, etc., é conseguido através de um sobrecarregado microcontrolador 80535, um derivado da família 8051. Foi programado em C através do compilador IAR versão 3.0. Todas as interrupções foram escritas directamente em *assembly*, utilizando apenas a estrutura de interrupções programáveis do controlador, *i. e.* sem qualquer S.O. de suporte. De facto, este 80535 que consegue, a custo, cumprir as restrições de tempo real do sistema, apenas porque o sistema é muito lento, as frequências utilizadas são de 5 Hz e a temperatura não varia rapidamente.

Em cada uma das 5 fontes (ver Figura 18) existe então um 80535 que pode funcionar em modo local (totalmente autónomo) ou pode ser Escravo[2] de comunicações 485, recebendo comandos e enviando assim os dados de e para uma interface gráfica. Esta interface gráfica é implementada num PC, presente noutro armário da zona de controlo de vazio e

[1] Proveniente do μC, passando através de um conversor Digital/Analógico

[2] No âmbito do paradigma de comunicações Mestre - Escravo (*Master - Slave*)

enchimento de óleo (da responsabilidade da *Hedrich*). Mas o Mestre[1] de comunicações 485 é um A. P. *Siemens* 115, que implementa um *gateway*[2] de comunicações entre os armários fontes de potência para os transformadores e o outro A. P. *Siemens* 115, que gere todo o sistema de vazio, comando das cubas, enchimento de óleo e faz ainda interface com o PC que gere também a interface gráfica, extremamente lento.

As fontes e o Autómato Programável (AP) que implementa a *gateway* de comunicações são da responsabilidade conjunta da FEUP e da EFACEC; o sistema de vazio, comando das cubas, enchimento de óleo, o outro AP (de comando do vazio) e o PC de interface gráfica são da responsabilidade da *Hedrich*. Para as duas partes comunicarem, utilizou-se todas as definições de um dos projectos europeus e utilizaram-se cartas MAP (*Manufacturing Automation Protocol*) para transferir dados de acordo com o protocolo OSI (*Open Systems Interconnection*), entre dois os dois A. P. iguais.

Como já referido, os dados não são enviados para nenhum sistema que permita o seu fácil acesso e registo. Por essa razão optou-se por quebrar as barreiras de comunicação próprias dos APs. A solução inicial era instalar uma carta de comunicações RS232 no autómato, mas não foi possível conseguir o barramento passivo apropriado para se poder utilizar a carta, pelo que a ideia foi abandonada. Considerou-se utilizar uma outra carta de comunicações MMS[3], que permitiria a comunicação fácil com programas em Windows via protocolo DDE[4]. Com essa carta poder-se-ia utilizar programas como o InTouch para conseguir uma conectividade OSI de modo fácil e rápido. Novamente, a ideia foi abandonada devido a dificuldades na sua obtenção, face aos custos envolvidos e à inexistência do barramento anteriormente referido.

Não dispondo doutra maneira de comunicar, a solução final consiste num protocolo de comunicações baseado na actuação de saídas do autómato. Dado que o AP da FEUP serve apenas de *gateway* de comunicações, não haveria problemas de espaço no barramento para instalar uma carta de saídas. Montou-se um sistema de aquisição de dados de 9 bits digitais em paralelo, utilizando apenas a porta paralela de um PC 286. Prosseguiu-se com a programação tanto do lado do AP como do lado do PC. O protocolo é muito simples, baseado na transmissão paralela de 8 bits aquando de um nono sinal de *strobe*. O formato

[1] No âmbito do paradigma de comunicações Mestre - Escravo (*Master - Slave*)

[2] *Gateway* - Processo intermédio, que interpreta a informação recebida num certo protocolo de comunicações e envia a mesma informação noutro protocolo

[3] *Manufacturing Message Specification*

[4] *Dynamic Data Exchange* - Protocolo muito utilizado para troca de dados entre programas Windows

da mensagem começa com um byte de start, que tem todos os bits em 1[1] e continua com os dados de acordo com a ordenação estabelecida. Tendo em conta que o protocolo é totalmente unidireccional, todos tempos envolvidos na emissão e recepção foram cuidadosamente ponderados. Verificou-se que ocasionalmente ocorriam erros de transmissão, um byte era perdido. Descobriu-se que o PC 286 era tão lento a escrever os dados no disco que por vezes ignorava um byte! A solução encontrada consistiu em retirar o *smartdrive* do DOS, reduzir o comando *buffers* do CONFIG.SYS para o valor de 10 e obrigar a que o disco seja escrito sempre e só quando desejado (o que não ocorria antes). Para forçar a escrita no disco foi criado um comando myflush, que cria uma cópia do *file handle* do ficheiro no DOS, fecha a cópia e mantém ainda o ficheiro aberto na linguagem C. Refira-se que este método é o único que faz com que o DOS altere o tamanho do ficheiro, permitindo assim que se a alimentação do PC for cortada, os dados anteriormente recolhidos não se percam. O programa de comunicações está pensado de tal forma que, perante qualquer problema, o funcionamento normal seja sempre retomado mesmo utilizando comunicações unidireccionais. Esta característica é conseguida através do arranque automático do programa de comunicações com o arranque do computador. A sincronização com início das mensagens é feita através do byte especial do início das mensagens e também com a análise da coerência dos dados recebidos, o que permite recuperar de falsos sincronismos quando aparecer um byte de dados igual ao byte de início imediatamente após o arranque do sistema. Consegue-se assim uma sincronização automática entre o (só) emissor e o (só) receptor. Deste modo, se a electricidade faltar, o sistema continuará a aquisição e armazenamento correcto dos dados. O sistema funciona sem monitor e sem teclado até ao momento em que se pretende copiar os dados para a disquete.

Os dados são escritos em formato CSV (*Comma Separated Values* - dados separados por vírgulas), que é um formato geralmente aceite pelas folhas de cálculo. Neste caso, os dados foram tratados através da folha de cálculo MS Excel.

O sistema de registo dos dados foi intensivamente testado e depois transportado para a EFACEC onde entrou em funcionamento com sucesso total, depois de resolvidos alguns problemas de memória no AP. Conseguiu-se assim que, se as fontes enviarem as medidas pela rede 485, então essas medidas são registadas em disco para posterior análise.

[1] Devido ao tipo de dados em transmissão, não é provável que este byte apareça no meio de uma mensagem (se aparecer no meio da mensagem, não existirá erro algum)

O conjunto que se tem vindo a descrever é ilustrado na Figura 18.

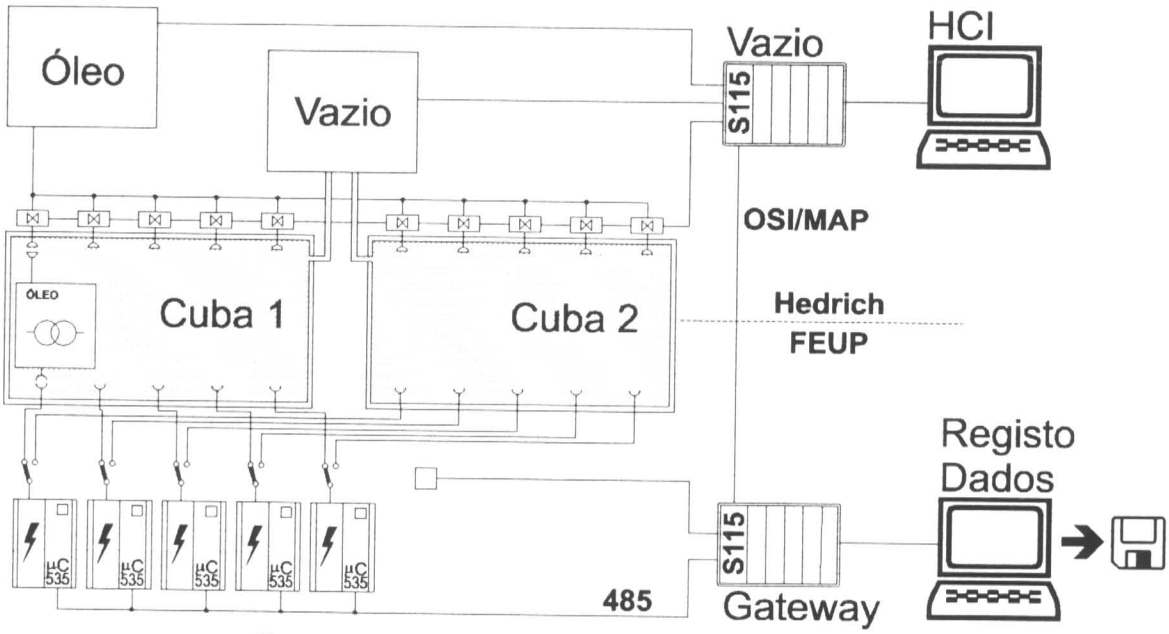


Figura 18:Diagrama dos vários blocos constituintes do CoPSE
(HCI significa *Human Computer Interface*, interface gráfica)

4.4 Objectivo do Trabalho Prático

Em funcionamento há cerca de 6 anos, o CoPSe carece, como já referido, de alguns melhoramentos.

O objectivo deste trabalho consiste em melhorar o sistema de controlo de temperatura por forma a evitar que, no seu regime final, apresente um *ripple*[1] maior que o aceitável.

Numa grande unidade fabril como a EFACEC, as metas de produtividade geram grande pressão sobre os responsáveis e operadores do processo. Uma falha no controlador do processo é um atraso mínimo de 5 horas úteis de produção[2]. Tal atraso representa não só uma quebra de produtividade como também ameaça o cumprimento dos prazos de entrega, já que, nesta fase, o transformador está praticamente concluído. Destinando-se ao mercado de exportação, o que é muito frequente, os custos agravam-se ainda mais se existir necessidade de fazer o reescalonamento de transportes. Por estes motivos, todo o trabalho teve que ser desenvolvido num número mínimo de testes, garantindo que nenhum transformador é danificado e que os atrasos na produção são mínimos.

[1] Flutuações de temperatura depois do sistema se ter aproximado do valor final

[2] O processo de secagem, em si, demora 3 horas mas as cubas permanecem fechadas durante mais cerca de 2 horas, para se fazer o enchimento dos transformadores de óleo

A Figura 19 refere-se ao funcionamento do sistema com o controlador convencional. Tanto para os transformadores grandes, de 1 MVA, como para os pequenos, de 100kVA, pode observar-se um *ripple* de temperatura apreciável, que pode ser melhorado apesar de todas as condicionantes que foram descritas ao longo deste capítulo. Para o processo representado na Figura 19, a temperatura em regime final tomou valores entre 96 e 117 °C, perfazendo assim um *ripple* de quase 21°C.

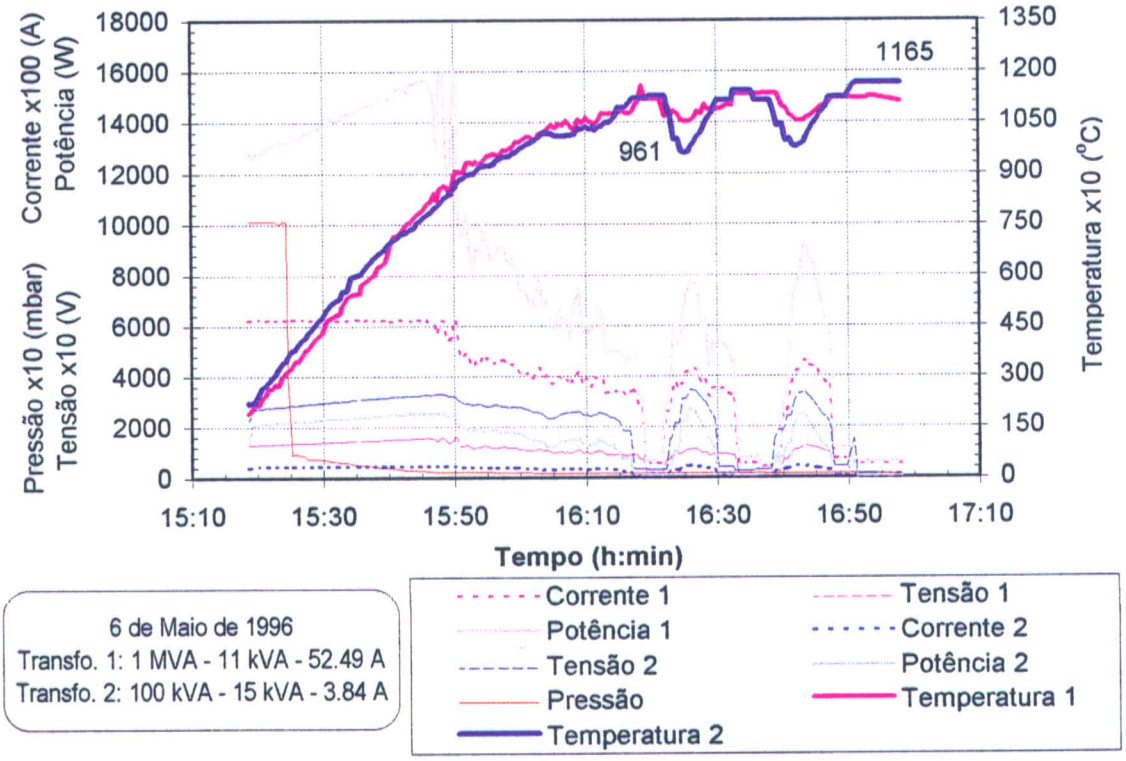


Figura 19: Funcionamento do CoPSE.
Comparação entre a secagem de um transformador
de 1MVA (1) e outro de 100kVA (2)
(sistema de controlo antigo)

A secagem de transformadores de médias dimensões apresenta os mesmos problemas, o *ripple* de temperatura é grande e o sistema não tende para um regime final estável.

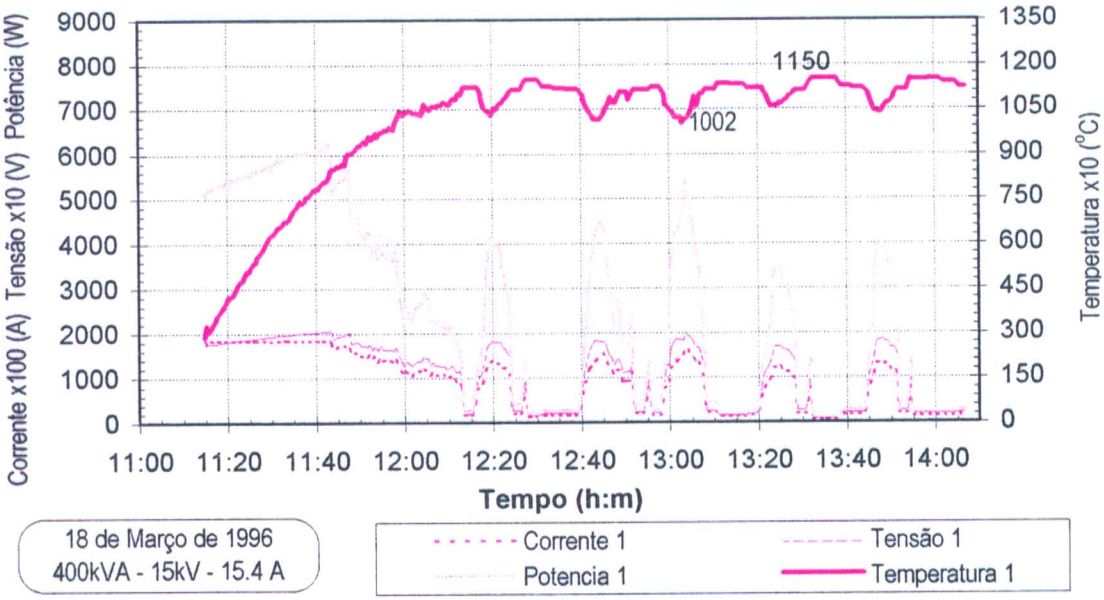


Figura 20: CoPSE em funcionamento sobre um transformador de potência média

Além das restantes funções, o controlador deverá estar preparado para situações pontuais imprevistas, tal como o rearranque do processo após actuação do sistema de paragem. A Figura 21 representa os dados recolhidos num ensaio em que esta situação se verificou[1].

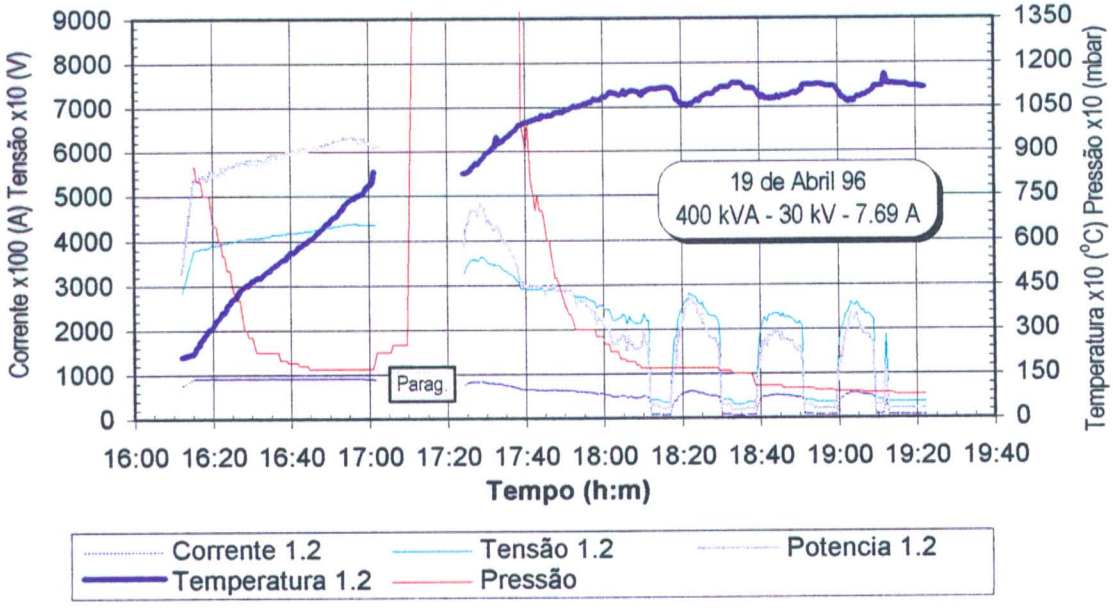


Figura 21: Processo CoPSE interrompido e depois retomado

Repare-se que, depois de ocorrer a paragem de emergência não há medidas. Durante o defeito, a pressão dentro da cuba atinge os valores atmosféricos por razões de segurança.

[1] Acidentalmente, alguém se encostou ao botão de paragem de emergência!

O Processo de secagem é retomado depois de repostas todas as condições de segurança e viabilidade do processo. A pressão na cuba desce apenas depois do processo de secagem ter sido retomado: geralmente, em funcionamento normal manual, as bombas de vazio são postas a funcionar antes de arrancar com o processo de aquecimento.

4.5 Recursos computacionais disponíveis

As dificuldade de trabalho num sistema de tamanho tão grande foram muitas, especialmente porque verificou-se que a documentação do hardware instalado era pobre e por vezes incompleta. A documentação do software anterior resumia-se a comentários ao longo do programa de um extenso programa, com a dificuldade acrescida de existirem saltos frequentes entre as rotinas em linguagem C e as rotinas em *assembly*, sem qualquer suporte de sistema operativo o que dificulta a compreensão do código, que se torna extremamente ligado ao hardware.

Tal como foi referido, o CoPSE é baseado num sobrecarregado mircocontrolador (μC) 80535. O programa instalado anteriormente instalado no μC era tão grande que deixava apenas 2 KB livres de EPROM.

O mapeamento de memória do sistema é baseado no modelo médio de memória[1], que junta memória de dados e de programa no mesmo espaço de endereçamento. De facto, o 80535 pode endereçar 64KB de memória de programa + 64KB de memória de dados mas tal é impossível com o hardware existente[2]. Assim, só é possível endereçar 64KB de memória conjunta para dados, E/S e programa. Importa não alterar hardware para que o sistema antigo continue em funcionamento durante o desenvolvimento deste novo controlador de temperatura.

Tabela 2: Tabela dos blocos de memória utilizados no CoPSE. A descodificação destes blocos é feita por hardware pelo que não podem ser feitas modificações. Para cada bloco mostra-se o espaço (inicialmente) disponível para o controlador de temperatura

Conteúdo	Tamanho	End. Início	End. Fim	Livre
EPROM, (Tabelas PWM)	16 KB	0	3FFFh	-
RAM	8KB	4000h	5FFFh	±5kB
E/S	8KB	6000h	7FFFh	-
EPROM	32KB	8000h	FFFFh	±2kB

[1] Dentro da classificação feita pelo compilador de C, da IAR

[2] A lógica de descodificação é à base de *gates*, pelo que é impossível alterar qualquer pormenor de endereçamento sem operações de alteração de hardware

A Tabela 2 mostra o endereçamento utilizado no CoPSE e as quantidades de memória disponíveis para a rotina do controlador de temperatura, já depois de retirada a anterior rotina de controlo da temperatura.

4.6 Compressão do código já existente

Para conseguir mais espaço livre foi necessário:

- Optimizar a compilação para redução do tamanho do executável
- Retirar código obsoleto, nunca chamado (na versão 3 do compilador da IAR esse trabalho não é automático)
- Retirar do núcleo do compilador (CL8051) muitas rotinas nunca utilizadas, tal como as relativas à atribuição dinâmica de memória, movimentação e entrada de literais (*strings*), tratamento de operações com inteiros longos (nunca eram utilizados), conversões de e para longos, conversões de literais para números, e vice versa, etc.
- Retirar literais nunca utilizados: mensagens de erro desactualizadas, etc.
- Substituição de macro-funções por chamadas a funções

Com estas operações foi possível libertar memória num total de cerca de 8KB.

Felizmente o espaço disponibilizado foi suficiente para a implementação pretendida. Se isso não tivesse acontecido, poder-se-ia recorrer a uma trabalhosa alternativa: substituir funções em C por equivalentes em *assembly*, tentando optimizar o código ao máximo. Como o 80535 não é um μC recente, ele não foi concebido com estruturas que permitam a execução fácil de código C (depois de compilado). A compilação manual de código C para *assembly* leva, em geral, a grandes acréscimos de velocidade e também a grande compactação do código gerado (especialmente na antiga versão do compilador de C utilizado). A rescrita de código seria trabalhosa porque a passagem de elementos (parâmetros de funções e respectivos resultados) do C para o *assembly* é complicada na referida versão do compilador. A referida manipulação de parâmetros e resultados exige chamadas a rotinas apropriadas do núcleo do compilador, que são muito lentas porque se baseiam na memória externa. No programa original, tal problema foi contornado de forma interessante, através da utilização dos bytes de memória RAM internos ao 80535, disponíveis por acesso indirecto à memória com endereços 80h-FFh (ver Figura 22). Claro que o aproveitamento desta zona de memória de 128 bytes era de 100%. A memória que não era utilizada para troca de informação era utilizada para *stack* interno. Alterar o tamanho da *stack* interna seria extremamente arriscado dado que a execução do programa é

muito lenta, existem muitas interrupções simultâneas e existe ainda algum código *assembly* que utiliza ainda a *stack* interna (durante as interrupções).

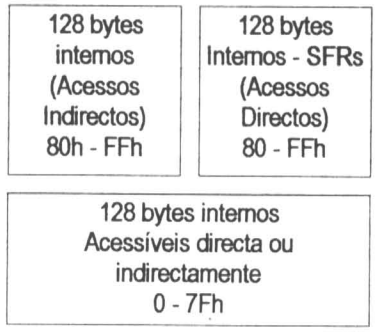


Figura 22: Memória do 80535: Modos de endereçamento
SFR: *Special Function Register* (figura adaptada do manual do μC)

4.7 Resumo

Durante este capítulo, descreveu-se o CoPSE, Controlador do Processo de Secagem e Enchimento de transformadores, em funcionamento na EFACEC - Transformadores Industriais.

Foi referida a envolvente fabril e o protocolo de comunicações que foi montado por forma a permitir o registo automático de dados.

O controlo convencional da temperatura neste processo revela-se pouco apropriado devido à não linearidade e à larga variação dos parâmetros do processo entre secagens e mesmo durante o seu decurso. O traçado da temperatura na fase final do processo apresenta características pobres de estabilidade em torno da referência, devido à natureza do primeiro controlador desenvolvido.

Foi feita uma introdução à implementação e fez-se a listagem dos recursos computacionais livres para a implementação do novo controlador.

5. Métodos de Controlo Avançado

Muitas alternativas para o melhoramento do controlador foram consideradas. O processo de secagem de transformadores acabou por revelar-se bastante complexo, não dispensando atenção a muitos detalhes.

Os métodos que se descrevem de seguida foram estudados como possíveis soluções para o melhoramento de características mas acabaram por ser abandonados pelas razões apresentadas, permitindo porém um melhor conhecimento de todas as características do processo.

5.1 Utilização de tabela de parâmetros PI(D)

Uma solução simples consistiria em fazer um ajuste *off-line*, baseado nos parâmetros eléctricos do sistema[1], utilizando a estrutura do controlador PI já existente. Através da análise de secagens anteriores de transformadores com parâmetros eléctricos similares, poder-se-ia criar um sistema de pesquisa de tabelas que forneceria parâmetros apropriados para o controlo do processo prestes a iniciar-se. Seria ignorada a variação dos parâmetros do sistema durante o decorrer da secagem.

Verificou-se que em produção se encontram com frequência transformadores de dimensões totalmente diferentes mas com os mesmos parâmetros eléctricos. De facto, transformadores de maiores dimensões físicas destinam-se frequentemente a mercados com climas quentes onde se incluem volumosas alhetas dissipadoras de calor e também enrolamentos maiores, de maneira a que o calor gerado possa ser facilmente transferido para o exterior.

Como a razão entre os volumes dos transformadores de igual potência atinge proporções de 1 para 3, admitiu-se que terão comportamentos termodinâmicos diferentes e consequentemente também o PI(D) que controla o processo deveria ser ajustado de forma diferente.

Infelizmente não existem registos de uma secagem onde transformadores de iguais parâmetros eléctricos mas volumes exteriores muito diferentes tenham sido secados simultaneamente, lado a lado. No entanto, os operadores do processo confirmam que se observam diferenças no processo de secagem entre transformadores da mesma potência.

[1] Apenas os parâmetros eléctricos do transformador são introduzidos durante a secagem: Potência, corrente e tensão

5.2 *Aquisição dos parâmetros do sistema* on-line

Tratando-se de um sistema lento, admitindo ainda manter o processo base de controlo PI, seria possível recorrer à estimação dos parâmetros do sistema, durante o seu decurso.

Desta forma conseguir-se-ia um controlador adaptativo.

Tal sistema parte do conhecimento empírico dos limites de variação dos parâmetros a estimar do sistema. A actuação deve ser de tal forma que permita fazer o reconhecimento do estado do sistema sem interromper o processo (estimar *on-line*), devendo ainda permitir a formulação de uma estratégia de controlo com baixo custo computacional.

Para o CoPSE, um método apropriado poderia ser implementado através de um "micro" degrau de potência e posterior registo da correspondente resposta. No entanto, há vários aspectos a considerar acerca deste método:

1. O processo em causa é não linear
2. Teria que se estabilizar o sistema antes de enviar o micro-degrau
3. A quantidade de energia contida no degrau de potência a enviar é muito importante: para outros sistemas poder-se-ia deixar simplesmente de alimentar o sistema e medir a resposta. Neste caso, teríamos que anular a corrente e depois medir a resposta ao degrau quando se passa a alimentar o sistema, esperando que a primeira medida não gere auto-aquecimento significativo
4. A temperatura tem que variar o suficiente para se poder aplicar o método mas não tanto que derrote o objectivo de a manter próxima dos 110°C, o que é difícil porque o ritmo de descida da temperatura com corrente nula varia muito durante a secagem, conforme a quantidade de vapores nas imediações do sistema
5. O número de vezes que se pretenderia aplicar o método teria que ser objecto de estudo cuidadoso
6. Através da resposta ao degrau seria eventualmente possível afinar o PI(D) sem fazer o reconhecimento do sistema, p. ex. através do conhecido método de Ziegler Nichols (Åström, 1984, secção 8.3 e Bennet 1994, secção 4.6).

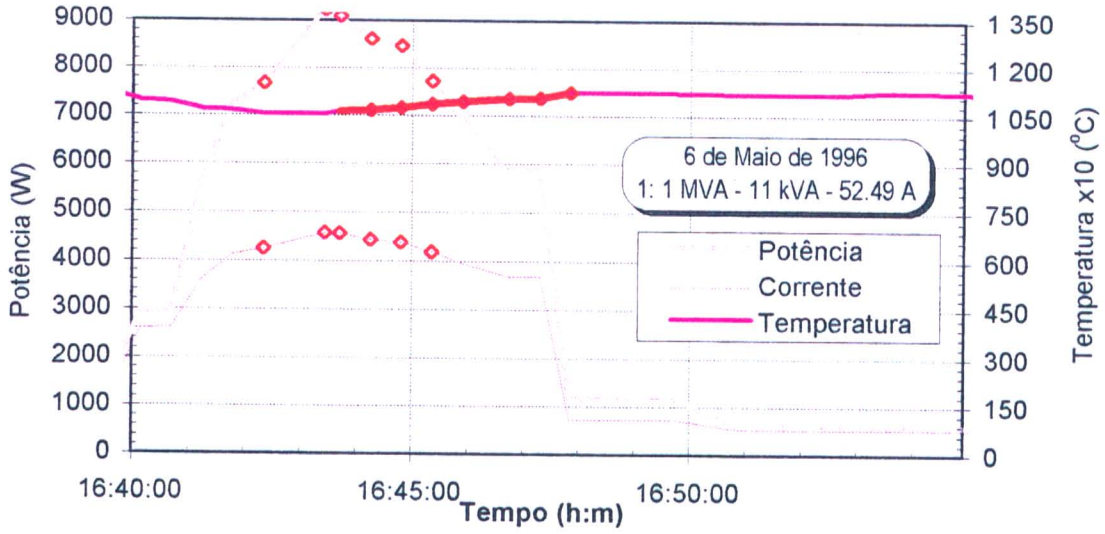


Figura 23: Ampliação nos tempos da Figura 19
(pormenor da temperatura, potência e corrente)

A Figura 23 diz respeito à aplicação de um (quase) pedestal de potência .

Pode-se ver, assinalado a vermelho e com marcas, os pontos máximos de actuação (envio de potência) sobre o sistema. Pode-se ver também a resposta, assinalada sobre a temperatura da mesma maneira. Na verdade, admite-se que não é possível reconhecer o sistema a partir da resposta a este pedestal. A explicação reside no facto de os valores armazenados correspondem a uma filtragem dos valores reais, nomeadamente médias das ultimas medidas. Tal tratamento deve-se à intenção de enviar estes dados para uma interface gráfica, tal como representado na Figura 18.

Observando ainda a Figura 23 repare-se que quando a corrente começa a baixar significa que na realidade ela é nula e as medidas de temperatura cessam imediatamente: a temperatura permanece constante quando deveria eventualmente ter algum atraso antes de ficar constante[1]. As medidas seguintes de temperatura ocorrem em instantes onde há picos de corrente de 50%I_N mas a corrente apresentada é a corrente média, equivalente ao *duty cycle* a que o sistema é obrigado (instantes posteriores aos representados pela Figura 23).

Os dados registados não permitem então um reconhecimento fiável do sistema, pelo que é muito difícil estudar a viabilidade de tal sistema, *i. e.* independentemente de ser possível uma implementação viável, não é fácil desenvolve-la com os dados disponíveis.

[1] Os dados continuam a ser enviados mas a temperatura vêm constante através da acção de um segurador de ordem zero (ZOH-*Zero order Holder*), um processo comum de converter processos discretos no tempo em processos contínuos, que introduz o tipo de limitações referidas (Astrom, 1984, secção 2.4)

5.3 Controlo por modo de deslizamento

Consideraram-se vários algoritmos de controlo não linear, de controlo robusto e de controlo adaptativo. Quase todos são computacionalmente pesados e pouco apropriados para casos em que não existe modelo para o sistema (no âmbito da T.S. convencional).

Mesmo sendo algo inadequado para o controlo do processo em causa, o controlo em modo de deslizamento deve ser realçado pelas suas potencialidades de fácil aplicação prática.

O controlo por modo de deslizamento (Bühler, 1986) apresenta o conceito de uma hipersuperfície de controlo e permite a utilização de estruturas diferentes para o sistema de acordo com a posição relativa do estado do sistema face à referida hipersuperfície de controlo. A actuação do controlador deve ser tal que, por solicitações extremas do actuador, o estado do sistema se aproxime até ao estado desejado para o sistema "escorregando" pela hipersuperfície de controlo.

O Controlo em modo de deslizamento (*sliding mode*) apresenta as seguintes vantagens (☺) e desvantagens (☹) (Bühler, 1986):

- ☺ Extremamente robusto, fornece controlo não linear, por vezes óptimo
- ☺ Imune a alguns tipos de variação nos parâmetros do sistema
- ☺ Fácil de programar em estruturas com vários graus de complexidade
- ☺ Peso computacional bastante menor que outros métodos de controlo não linear
- ☺ Extremamente apropriado a sistemas com actuadores não lineares
- ☺ Apropriado para sistemas muito rápidos
- ☺ Introduz a noção de que se pode comandar no tempo e não apenas nas entradas do sistema
- ☹ Não é aplicável por causa da ausência de medida quando a corrente é zero
- ☹ Exige solicitações extremas do actuador, que é modificado frequentes vezes
- ☹ Não aplicável para sistemas com atraso puro (potencialmente o caso)

O controlo em modo de deslizamento é muitas vezes utilizado para controlo de inversores e/ou motores, quando se pretende tornar o sistema controlado imune à variação de parâmetros de partes do sistema. Este algoritmo de controlo não é apropriado para o CoPSE porque é excessivamente lento, tem uma forte componente integrativa e potencialmente algum atraso puro (ver "7-Simulação do CoPSE", pág. 95).

Coleman *et al* (1994) faz uma comparação de aplicabilidade de métodos de controlo, entre os quais controlo por modo de deslizamento e conclui que este é o melhor método de controlo. Esta conclusão seria de esperar de acordo com Bühler (1986), pois o

sistema considerado pertence à categoria em que o controlo por modo de deslizamento torna o sistema em malha fechada insensível à variação de parâmetros do sistema original.

5.4 Filtros de Kalman

De acordo Cooper *et al* (1994), o filtro de Kalman fornece um algoritmo para o processamento óptimo de dados, utilizando recursividade, que funde o conhecimento do processo e a informação dos sensores para actualizar o estado do sistema. Este tipo de algoritmo é um estimador do estado do sistema[1], que executa um ciclo de previsão/correcção condicionado à informação existente. A evolução do sistema depende por ex. do estado anterior do sistema, da informação recolhida pelos sensores, da confiança associada a cada sensor, de modelos de partes do sistemas ou erros de medida associados a partes do sistema, etc.

De seguida referem-se vantagens ☺ e desvantagens ☹ da aplicação deste método (Cooper *et al* 1994, Åström *et al* 1990):

- ☺ Trata-se de um controlo no âmbito da Teoria dos Sistemas (T.S.) convencional pois o filtro de Kalman acaba por ser um observador para o sistema
- ☹ Necessita que a estrutura e parâmetros aproximados do sistema sejam conhecidos
- ☺ A utilização de funções que estimam o valor dos erros cometidos para cada parâmetro e medida do sistema permite obter o estado do sistema e assim gerar acções de comando óptimas para o estado estimado do sistema
- ☹ Exige descrições matemáticas dos blocos do sistema e respectivos erros associados
- ☺ Técnica muito apropriada para a fusão da informação proveniente de vários sensores, cada um com características bem conhecidas (geralmente é o caso)
- ☺ Muito usado quando o sistema é complexo e composto de várias partes, p. ex. se existem vários sensores para a mesma grandeza, vários actuadores, etc.
- ☹ Computacionalmente muito pesado e gastador de memória devido ao recurso à manipulação permanente de matrizes
- ☹ O sistema necessita de algumas características matemáticas para que o método possa ser aplicável sem restrições

A implementação deste método para o CoPSE permitiria contornar o problema da ausência de medida com a corrente nula: seria possível estimar a evolução da temperatura (e respectivo erro) a partir da reunião de outros dados, pois existe o estado aproximado do sistema. Claro que o método é impraticável num μC 80535 com 8KB de RAM.

[1] Um observador no estado de espaços

Os filtros de Kalman são muito usados pelos veículos autónomos, nomeadamente os veículos autónomos sub-aquáticos como meio de construir o estado global do sistema a partir de muitos sensores e perante a actuação de muitos actuadores num sistema complexo (Fryxell *et al*, 1994).

Os filtros de Kalman são uma sólida ferramenta para permitir a matematização de conhecimentos variados acerca do sistema e respectiva envolvente. Neste aspecto, a Inteligência Artificial (I.A.) e os filtros de Kalman relacionam-se de alguma forma (Kosko, 1993).

5.5 Resumo

No âmbito da aplicação que se tem vindo a descrever, foram considerados vários métodos avançados de controlo. Os métodos aqui apresentados revelaram-se promissores para outros processos, que exijam controlo avançado, mesmo para processos não lineares. O estudo da viabilidade destes métodos revelou características importantes acerca do comportamento do processo de secagem de transformadores, que importa conhecer para a correcta implementação.

6. Controlo Inteligente

6.1 Introdução ao controlo inteligente

Por analogia com o controlo humano, uma solução contemplando um controlador do processo com características inteligentes não pode ser ignorada.

De acordo com Leitch (1992), a utilização de técnicas de Inteligência Artificial (I.A.) aplicadas ao controlo, alarga o leque de ferramentas disponíveis, de modo a abranger os sistemas de alta complexidade e/ou de difícil modelização. O referido autor argumenta que outras técnicas de controlo avançado, baseadas numa rigorosa modelização do sistema, podem trazer resultados sub-óptimos no caso de tal modelização ser inacessível ou devido a suposições feitas durante o processo de modelização. A utilização de técnicas empíricas e modelos simplificados mas correctos no sentido de que «o modelo é correcto se servir o seu propósito» é defendida. As técnicas de I.A. conduzem à utilização de modelos com elevado nível de perceptibilidade, incorporando a compreensão macroscópica existente relativamente ao problema. Se o conhecimento do processo é incerto, vago ou impreciso, então o controlador deve incorporar tais conhecimentos. Ainda o mesmo autor defende as simulações qualitativas (não numéricas) pelas seguintes razões:

1. Para reduzir o peso computacional dos outros algoritmos já existentes, numéricos e precisos (exemplo de um campo de aplicação: a astronomia)
2. Para descrever sistemas onde métodos tradicionais são ineficazes
3. Para obter modelos compreensíveis e que traduzam o comportamento macroscópico do sistema, incorporando conhecimentos heurísticos do processo
4. A utilização de descrições declarativas permite ultrapassar o conhecimento parcial, incompleto ou incerto que porventura exista no sistema

Pelo que foi anteriormente referido podemos concluir que o controlo no campo da I.A. é uma alternativa a ser considerada face às características do processo de secagem de transformadores.

6.2 Metodologias de controlo Inteligente

Segundo (Kosko 1992 - pág. 26) a I.A. fornece ferramentas de controlo para sistemas sem ser necessário reconhecer os seu parâmetros, nem sequer de forma estocástica. Nesta

abordagem, existem 2 tecnologias que se complementam e que têm a aplicabilidade mostrada na Tabela 3.

Tabela 3: Taxonomia dos métodos para trabalho em sistemas sem estimação de parâmetros (adaptada de **Kosko, 1993, pág. 26**)

		Tecnologia do trabalho	
		Simbólico	Numérico
Conhecimento do Processo	Estruturado	Sistemas Periciais em I.A.	Sistemas *Fuzzy*
	Não Estruturado	-----	Sistemas Neuronais

De acordo com a Tabela 3 e tendo em conta que existe conhecimento estruturado da aplicação presente, que requer um controlo do tipo numérico, a tecnologia de trabalho que deve ser aplicada é a dos sistemas *Fuzzy*. **Mamdani (1994)** refere também que esta tecnologia é apropriada para implementação em sistemas com poder de cálculo limitado, o que é necessário para a aplicação em causa.

Para além dos métodos de controlo inteligente referidos por **Kosko (1993)**, referir-se-á também brevemente os Algoritmos Genéticos (*Genetic Algorithms* - GA) e os Sistemas Periciais.

6.3 Sistemas Periciais

Os Sistemas Periciais permitem "armazenar" o conhecimento dos peritos permitindo que um utilizador ingénuo utilize as estratégias do perito para tomar decisões. **Harris *et al* (1993, secção 1.4)** refere que os sistemas periciais utilizam uma colecção de factos, heurísticas, conhecimento proveniente do senso-comum conjuntamente com outros tipos de conhecimento para fazer inferências e assim gerar mais conhecimento. Os sistemas periciais permitem contornar conhecimentos incompletos pois pode-se utilizar eficazmente a informação existente e pode-se também programar linhas de actuação quando a informação não existe. A referência anterior assegura ainda que os sistemas periciais são destinados a duas classes de controlo:

- Aplicações *off-line* - desenho de controladores, escalonamento, planeamento, gestão, e análise de defeitos
- Aplicações *on-line* - gestão e controlo de processos durante o seu decurso

Os sistemas periciais são modulares no sentido em que o conhecimento e as regras de inferência são guardados em espaços diferentes de programação. Assim, acrescentar conhecimento é simples e não altera a estrutura de controlo, formada pelas regras de inferência.

Harris *et al* (1993, secções 1.4 e 1.8) refere ainda, como exemplo, que uma potencial aplicação dos sistemas periciais ao controlo seria no ajuste de controladores desenhados utilizando o método de Ziegler-Nichols (Åström 1986, secção 8.3), que se presta bem para ser afinado perante heurísticas conhecidas por especialistas e que poderiam ser capturadas em sistemas periciais para permitir controlo automático, o que acaba por ter algum relacionamento com o que foi proposto em 5.1-Utilização de tabela de parâmetros PI(D), pág. 75.

Um bom sistema pericial deve ser validado para garantir que as regras foram bem assimiladas pelo programa. A utilização de sistemas periciais como suporte de apoio à decisão é frequente e esse tipo de sistemas deve permitir o estudo das razões que levaram à tomada de certa decisão.

Harris *et al* (1993, secção 1.4) refere que todos os sistemas de controlo baseados em regras sofrem de algumas fraquezas:

- ☹ Assumem que todas as informações estão disponíveis (necessitam que a inexistência de informação seja prevista)
- ☹ Algumas subtilezas temporais podem passar despercebidas
- ☹ O conjunto de regras de controlo pode não ser consistente de perito para perito ou de ciclo para ciclo do processo
- ☹ A informação recolhida pode não estar a fazer o controlo óptimo do processo

A implementação deste tipo de sistemas exige ainda grades capacidades de cálculo não numérico e é gastadora de memória tanto de dados como de programa, logo este tipo de controlo será pouco apropriado para o CoPSE.

6.4 Redes Neuronais

A engenharia do controlo por Redes Neuronais (*Neural Networks*) tenta reproduzir o comportamento biológico de baixo nível dos neurónios (do cérebro) para tentar resolver os problemas mais complexos da engenharia do controlo (Jutten *et al* 1994).

As Redes Neuronais atingem o conhecimento através da criação de sinapses entre os vários pontos da rede. Cada uma dessas sinapses representa conhecimento adquirido

durante a fase de treino, essencial a qualquer rede neuronal. Treinar a rede significa alterá-la por forma a que a sua estrutura reflicta o resultado pretendido (conhecido) com as entradas (conhecidas). Se o conjunto de treino tiver características apropriadas, quando o conjunto for submetido a novas entradas, ele produzirá novas saídas, de acordo com as regras que foram memorizadas.

As vantagens (☺) e desvantagens (☹) das Redes Neuronais (**Jutten *et al* 1994, Harris *et al* 1993**) são resumidas de seguida:

- ☺ Adquire conhecimento sem programação específica (é como um estimador cujos parâmetros são estimados durante o treino)
- ☺ Conseguem generalizar conhecimentos, ou seja, mapear entradas a saídas similares
- ☺ Não necessitam de qualquer formulação matemática
- ☺ Formam mapeamentos contínuos, não lineares, apropriados para o controlo
- ☺ São arquitecturas potencialmente distribuídas e paralelas
- ☺ Têm potencialidades para funcionamento em tempo real
- ☹ Geralmente grandes, de tamanho muito maior que se fosse utilizado um estimador dos parâmetros do sistema no âmbito da T.S. convencional
- ☹ O conjunto de treino precisa de ser bem escolhido: deve ser suficientemente rico para a rede aprender o essencial
- ☹ O tamanho da rede precisa de ser bem ponderado face ao processo e face à capacidade de processamento disponível
- ☹ Uma rede demasiado grande, mal treinada pode vir a responder bem apenas ao conjunto de treino – pode ficar com memória a mais e inteligência a menos!
- ☹ O conhecimento programado não pode, em geral, ser retirado para validação, pelo que se houver alguma sinapse errada só se chegará a essa conclusão através de testes do sistema
- ☺ A correcção pode ser feita com um novo conjunto de treino bem escolhido
- ☺ Em princípio as redes neuronais são estáveis e absorvem mais informação mantendo a anterior (propriedade da plasticidade)
- ☺ Potencialmente adaptativas durante o decurso do processo

O funcionamento deste tipo de sistema é sempre sujeito a muitos testes pelo que não será apropriado para a aplicação em vista, para além de que estes sistemas levam a estruturas maiores e de maior peso computacional que outros métodos.

É de salientar que as redes neuronais têm tido muito sucesso, nomeadamente para produtos destinados ao consumo do grande público, especialmente no Japão, onde as técnicas de I.A. tem feito grande sucesso (ver Tabela 4).

Tabela 4: Produtos para consumo no Japão, provenientes de várias marcas comerciais, utilizando Redes Neuronais (adaptado de **Takagi, 1991**)

Matsushita Electric	ar condicionado, máquina de lavar, aspirador, tacho de arroz, aquecedor utilizando querosene, chaleiras eléctricas, forno de microondas, discos de indução
Sanyo	máquina de lavar, máquina de secar roupa, forno de microondas, escalfeta, carpete eléctrica, ventoinha, aquecedor de mesa, tacho de arroz, aquecedor utilizando querosene
Hitachi	tacho de arroz, aquecedor utilizando querosene, máquina de lavar, aspirador
Sharp	frigorífico, máquina de lavar, tacho de arroz, aquecedor utilizando querosene
Mitsubishi	aquecedor utilizando querosene, disco de indução
Toshiba	aquecedor utilizando querosene, máquinas de lavar
Fujitsu General	aquecedor utilizando querosene
Corona	aquecedor utilizando querosene
Toyotomi	aquecedor utilizando querosene

6.5 Algoritmos Genéticos

A área da I.A. inclui ainda os algoritmos genéticos (*Genetic Algorithm* - GA) também por vezes designados por algoritmos evolucionistas. Este tipo de algoritmos tem tido, porventura, menos aplicação que outros métodos de I.A.. De acordo com **(Passino, 1995)**, o princípio de funcionamento destes algoritmos baseia-se numa analogia com a evolução das espécies, selecção natural e genética retirada da ciência da biologia. O computador simula a "vida" de populações e faz sobreviver e evoluir apenas as mais "aptas". Para o conseguir, os algoritmos genéticos, mantém uma colecção de parâmetros característicos de cada controlador. Essa colecção de informação é trabalhada depois com os operadores genéticos:

1. Reprodução - representa a sobrevivência do mais apto
2. Junção de características - representa acasalamento
3. Mutação de características - introdução de características aleatórias

Estes operadores genéticos são aplicados a cada geração de dados, juntamente com uma função de medida da aptidão, que estará relacionada com os objectivos que se pretendem cumprir. Os resultados da aptidão, geração após geração, ditarão quais as colecções de dados que vingam e quais as que definam e morrem, escolhendo-se assim o controlador mais apropriado para o sistema.

As vantagens (☺) e desvantagens (☹) de controlo utilizando Algorítmos Genéticos são resumidas de seguida:

- ☹ É necessário propor vários controladores minimamente viáveis à partida
- ☺ As características dos vários controladores propostos para o sistema são sucessivamente apuradas
- ☺ O controlador final possivelmente será melhor que qualquer um dos inicialmente programados (resultado de mutações e junção de características)
- ☺ O controlador é extremamente robusto e flexível
- ☹ O apuramento é feito à custa de muita computação: teste da viabilidade sobre cada controlador e posteriormente aplicar os operadores genéticos (Reprodução, Junção de propriedades e Mutação de características)
- ☹ O resultado obtido é estocástico, mas dirigido – em princípio convergirá, mas não se sabe ao fim de quantas iterações o processo estabiliza

Para o sistema considerado neste trabalho, os Algoritmos Genéticos produziriam possivelmente uma solução viável caso a plataforma de processamento não fosse um μC 80535.

6.6 *Solução:* Fuzzy Logic

Tal como preconizado em "6.2-Metodologias de controlo Inteligente", pág. 81, optou-se por implementar o método do controlo por "*Fuzzy logic*". A tradução para português carece ainda de algum consenso pelo que se propõe as expressões "Lógica Imprecisa" ou "Lógica Difusa". Na língua francesa, divulgou-se a expressão "Logique Floue", que poderia traduzir-se também por "Lógica Imprecisa". Durante este trabalho, será também a expressão utilizada.

A aplicação da lógica imprecisa confere graus intermédios de verdade para todas as afirmações. Perante esta perspectiva:

- É recusada a lei do terceiro excluído
- É recusada a lei da não contradição
- As afirmações imprecisas (*Fuzzy*) são classificadas com níveis de verdade intermédios entre absolutamente verdadeiro e completamente falso (que são as classificações extremas, também possíveis)

Um dos exemplos da maneira de pensar da lógica imprecisa (Kosko, 1992, pág. 4), remete para o paradoxo clássico do mentiroso de Creta que diz que todos os habitantes de Creta são mentirosos. Se o que ele diz for verdade, então ele é um mentiroso (mas diz a verdade!). Se ele estiver a mentir, então a frase é verdadeira e ele não é mentiroso (mas acabou de dizer uma mentira!).

Todos os paradoxos tem a mesma forma. A afirmação S e a sua negação "não S" tem o mesmo grau de verdade[1] t(S):

$$t(S) = t\ (\text{não } S) \tag{3}$$

Ambas as afirmações são verdadeiras ou são ambas falsas

Para lógica bivalente[2],

$$t(\text{não } S) = 1\text{-}t(S) \tag{4}$$

E então (3) resulta

$$t(S)=1\text{-}t(S) \tag{5}$$

Se S é verdade, t(S)=1 e fica 1=0. Se S é falsa, t(S) é 0 e vem 0=1. É o referido paradoxo.

Aceitando valores de veracidade intermédios, tal como a lógica imprecisa preconiza, permite achar a solução para a equação 5:

$$2\ t(S) = 1$$

$$t(S) = \tfrac{1}{2} \tag{6}$$

Assim, está resolvido o paradoxo com "meias verdades", que é afinal a forma como se lida com este tipo de problemas no quotidiano.

A lógica imprecisa permite explicitar que "No que respeita à realidade, as leis da matemática não são exactas. Quando o são, não se referem à realidade", como afirma Albert Einstein (**Kosko, 1992, pág. 263**).

De facto, as transições entre conceitos reais não são abruptas. Os conceitos são, muitas vezes, subjectivos e existem zonas de transição entre conceitos diferentes e bem definidos e bem conhecidos. Como ilustração, considere-se a pergunta: A temperatura de 28°C será agradável? Não se trata de saber qual a probabilidade de 28°C ser agradável mas sim do facto de, para algumas pessoas ser agradável e para outras não. A resposta das pessoas não é aleatória, apenas existem opiniões diferentes. Muito provavelmente, 100% dos inquiridos estariam de acordo que -12°C não é uma temperatura agradável. Mas poderiam estar em desacordo parcial se seria uma temperatura fria ou se seria muito fria.

[1] grau de verdade, neste contexto, é sinónimo de valor lógico da afirmação

[2] Bivalente, só 100% verdadeiro ou só 100% falso, lógica booleana ou digital

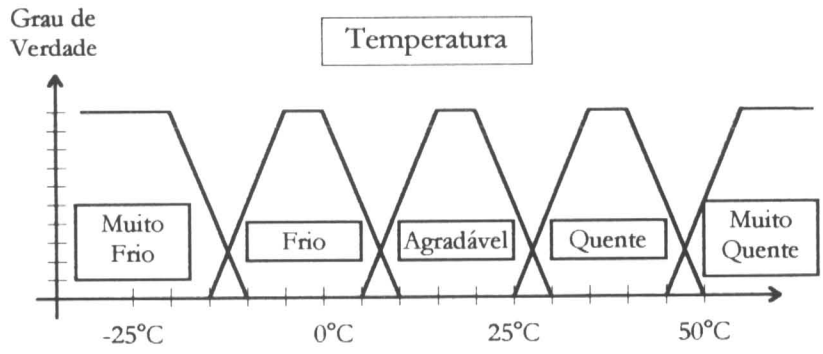


Figura 24: Regiões imprecisas da variável temperatura

O autor do artigo que originou a lógica imprecisa, Lotfi Zedeh refere que «à medida que a complexidade de um certo sistema aumenta, a capacidade para fazer afirmações significativas do seu comportamento diminui até se atingir um ponto de transição, a partir do qual precisão e significado (relevância) se tornam mutuamente exclusivas» (Cox 1994, pág. 2). A lógica imprecisa permite representar conhecimentos imprecisos, até contraditórios, e suporta raciocínios e conclusões baseados nesse tipo de conhecimentos incompletos ou vagos.

Cox (1994) refere ainda que a utilização de modelos imprecisos para o sistema permite ainda[1]:

- ☺ MTBF[2] e MTTR[3] melhorados
- ☺ Capacidade para modelizar simplificadamente sistemas complexos
- ☺ Permitem uma melhorada representação cognitiva dos conhecimentos dos peritos
- ☺ Permitem a modelização de sistemas envolvendo vários peritos que podem ter opiniões diferentes
- ☺ Permite o tratamento de incertezas e possibilidades

De facto, a utilização de lógica imprecisa permite representar de forma agradável o conhecimento do dia a dia através da utilização de variáveis linguísticas: a variável imprecisa temperatura pode ser descrita pelas regiões imprecisas da Figura 24.

Considere-se a construção de um sistema que deve aconselhar um preço de venda no mercado de um novo produto (Cox 1994, pág. 387). O perito da área financeira diz "O preço deve ser alto" e o perito de Marketing diz "O preço deve ser baixo". Utilizando os princípios da Lógica Imprecisa, a região *alto* da variável preço pode ser representada por uma recta crescente no grau de veracidade. Admitindo que os peritos concordam em que

[1] Confirmando o referido em "6.1-Introdução ao controlo inteligente", pág. 81

[2] *Mean Time Between Failures* - Tempo médio entre avarias

[3] *Mean Time to Repair* - Tempo médio de reparação

se atinge a verdade absoluta (o preço é alto) quando o preço de venda é 3 vezes superior ao preço de custo, então a região fica definida tal como mostra a Figura 25. Considerando que (preço) *Baixo* é a negação[1] do preço alto, então t(*Baixo*)=1-t(*Alto*), obtemos uma recta decrescente, que atinge a falsidade total quando o preço de venda é 3 vezes superior ao preço de custo. A representação do conhecimento de "*Preço* deve ser *Alto* **e** *Preço* deve ser *Baixo*" está representado na Figura 25, sendo a conclusão: o preço não deve ser nem demasiado alto nem demasiado baixo.

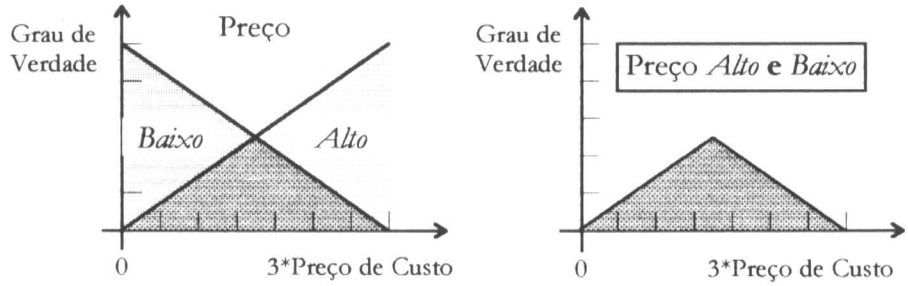


Figura 25: Variável Preço, regiões *Alto*, região *Baixo*
Forma da região (*Alto* **e** *Baixo*)

Se um terceiro perito, da área do fabrico afirmar que o preço de venda deve rondar 2 vezes os custos de fabrico, após a definição da região de preço *2xCusto*, tal como na Figura 26, então pode-se observar na mesma figura a região correspondente à conjunção dos 3 requisitos.

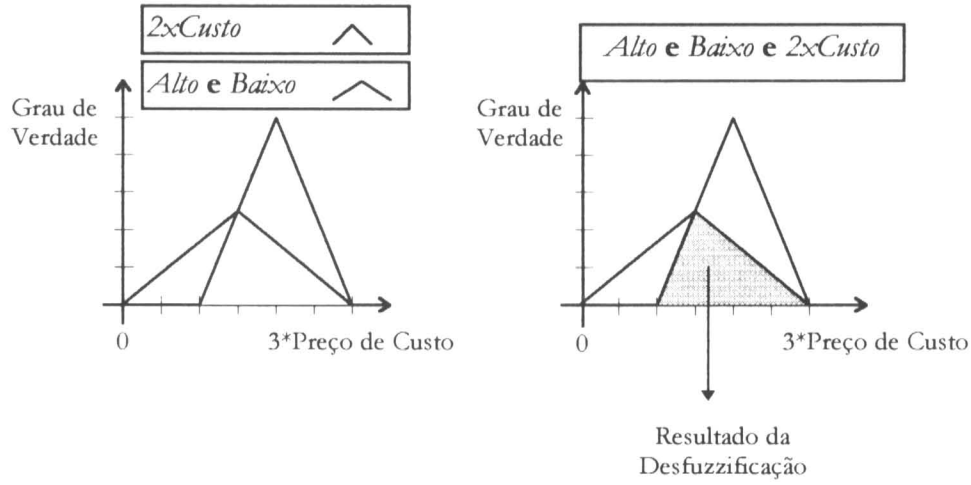


Figura 26: Variável Preço, região (*Alto* **e** *Baixo*), região *2xCusto*, conjunção das duas regiões

O processo de conversão da região imprecisa final num número, neste caso no preço aconselhado é denominado de desfuzzificação[2] e reside simplesmente no cálculo do centro

[1] Esta definição para o operador negação é proposta por Lotfi A. Zadeh

[2] Neologismo que traduz à letra *defuzzification*, eventualmente poder-se-ia também utilizar "*precisificação*" ou "*desimprecisificação*"

de gravidade[1] da região final (também representado na Figura 26) e pode ser conseguido através da expressão (Harris 1993, pág. 78):

$$v = \frac{\int_x x \times t(x)}{\int_x t(x)} \tag{7}$$

v é o valor da desfuzzificação

$t(x)$ é o grau de verdade no ponto x

A estrutura de um controlador genérico de lógica imprecisa necessita de ter a forma descrita na Figura 27.

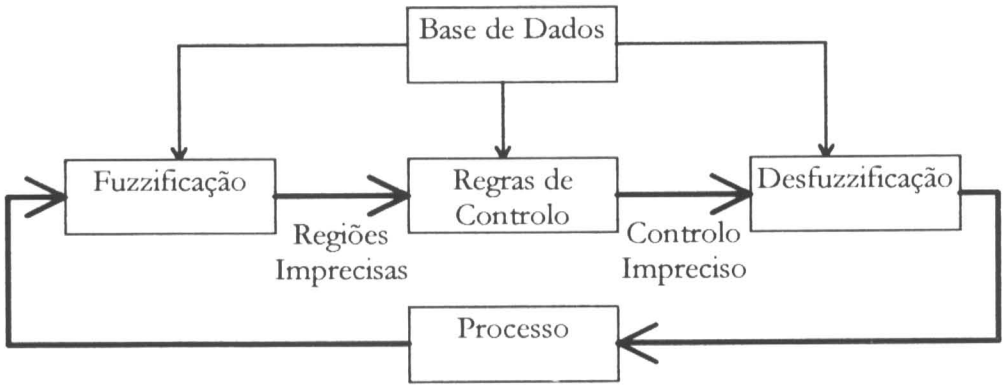


Figura 27:Estrutura de um controlador genérico, baseado em lógica imprecisa (Harris 93, pág. 97)

O processo de fuzzificação[2] trata do mapeamento inicial que tem de existir entre as entradas (valores reais, ditos cristalinos, p. ex. leituras de sensores) e as regiões imprecisas de maneira a que as regiões imprecisas reflictam o estado real do sistema. A base de dados do sistema consiste nas informações relevantes e definições para a discretização do processo, normalização do universo de medidas para o domínio das regiões imprecisas consideradas, etc.

As regras de controlo em lógica imprecisa tomam o aspecto de regras de inferência lógica: "**Se** (antecedente) **Então** (consequente)". O antecedente pode ser formado composto por conjunções e disjunções de condições que têm a forma "(variável de entrada) **é** (região imprecisa)". A aplicação de regras de controlo utiliza um algoritmo de inferência e a informação existente acerca do estado do sistema (o antecedente) para gerar novos conjuntos imprecisos que descrevem o estado pretendido para as saídas. Os consequentes

[1] O método é denominado Centro de Gravidade ou Centróide

[2] Neologismo para traduzir *fuzzification*, poder-se-ia utilizar também "*imprecisificação*"

também tomam a forma "(variável de saída) **é** (região imprecisa)". O processo de desfuzzificação encarrega-se de converter as regiões consequentes para valores reais de comando dos actuadores.

Existem muitos operadores de fuzzificação, desfuzzificação e inferência, que conduzem a que o controlador tenhas características desejadas de não linearidade das saídas.

De acordo com **Harris (1993, exemplo 2.11)**, considere-se um problema de controlo da posição de veio de motor. As regras propostas para o controlador são baseadas no erro e na derivada do erro. Tomando as variáveis Erro *E*, Derivada do Erro *dE*, a saída Velocidade *V* e as regiões definidas na Figura 28, podemos aceitar as seguintes regras de controlo:

R1: **Se** *E* **é** *AZ* **e** *dE* **é** *PS* **então** *V* **é** *NS*

R2: **Se** *E* **é** *AZ* **e** *dE* **é** *AZ* **então** *V* **é** *AZ*

R3: **Se** *E* **é** NS **e** *dE* **é** *NS* **então** *V* **é** *PS*

R4: **Se** *E* **é** NS **e** *dE* **é** *AZ* **então** *V* **é** *PB*

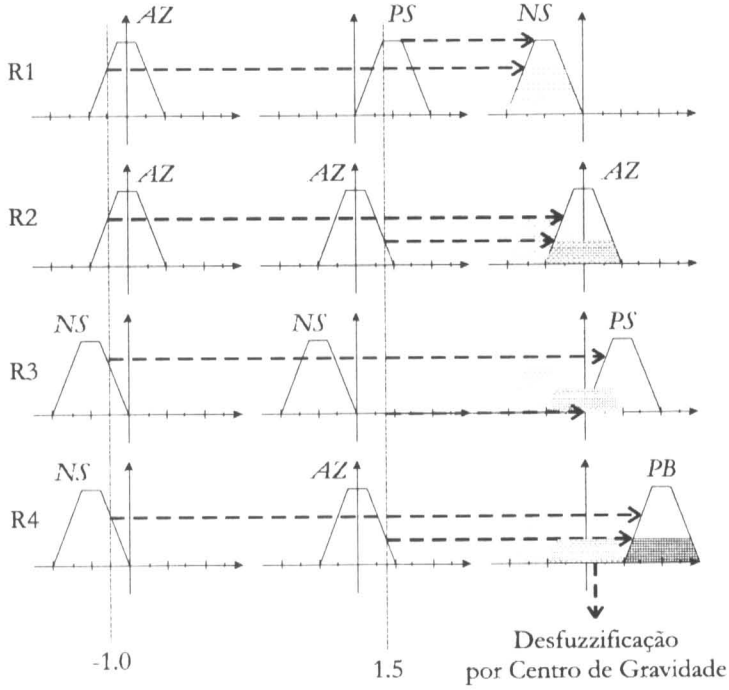


Figura 28: Exemplo de inferência por lógica imprecisa

Considerando um par de entradas reais no sistema de controlo, já após a fuzzificação, de (-1.0, 1.5), observe-se na Figura 28 o método de inferência utilizado pela lógica imprecisa, considerando como operador **mínimo** para a conjunção de condições e para a inferência.

Regra 1: **t**(*E* **é** AZ)=0.7 ; **t**(*dE* **é** *PS*)=0.85

- *V* **é** *NS* com grau de confiança de 0.7[1] e 0.7 é o antecendente da regra – resultado da conjunção das condições -- **mínimo** de (0.7 e 0.85)
- A região de saída desta regra é o **mínimo** (operador de inferência) da confiança no antecedente (0.7) da regra e do grau de verdade a região *NS* de *V*, ao longo de todo o seu domínio[2]

Regra 2: **t**(*E* **é** AZ)=0.7 ; **t**(*dE* **é** *AZ*)=0.45

- *V* **é** *AZ* com grau de confiança 0.45
- Esta regra cria uma região imprecisa de saída através da utilização do operador **mínimo**, de inferência, entre o valor 0.45 e a região *AZ* de *V*
- A região imprecisa de saída resultante das regras R1 e R2 é o **máximo**[3] (operador de agregação) das duas regiões de saída anteriores (Jager, 1995)
- Após a execução destas duas regras, V é parcialmente *NS* e parcialmente *AZ*

Regra 3: **t**(*E* **é** *AZ*)=0.7 ; **t**(*dE* **é** *AZ*)=0.0

- *V* é *PS* com grau de confiança 0.0
- A região resultante não é alterada (operador de agregação é o **máximo**)

Regra 4: **t**(*E* **é** *AZ*)=0.7 ; **t**(*dE* **é** *AZ*)=0.45

- *V* é *PS* com grau de confiança 0.45
- A região resultante é alterada, de acordo com o operador agregação

Após esta passagem pelas 4 regras de controlo, obtemos uma região de saída que precisa de ser desfuzzificada para produzir um valor para a velocidade do motor. Utilizando o método da desfuzzificação por centro de gravidade (equação 7), obtém-se o valor 0.25 para a velocidade do motor.

A lógica imprecisa tem tido grande aplicação, tanto a nível de produtos para o consumidor como para controladores industriais e as inúmeras aplicações incluem áreas tão variadas como (Bosacchi,1993, Kiendl *et al* ,1994, Marks, 1994):

1. Engenharia Química
2. Electrónica
3. Sensores
4. Controlo de (servo) motores
5. Robótica
6. Sistemas para automóveis

[1] Se a conjunção fosse representada pelo operador **produto algébrico**, então o valor final do atecedente seria 0.7*0.85=0.595

[2] Se o operador de inferência fosse o **produto algébrico**, então toda a região *NS* seria multiplicada pelo valor do antecedente da regra

[3] Representa o acumular de conhecimento. O operador de agregação é, quase sempre, o operador **máximo** (Jager, 1995)

7. Indústria Aeroespacial
8. Comunicações
9. Bioengenharia
10. Processamento de imagem
11. Reconhecimento de padrões
12. Gestão, sistemas de apoio à decisão e sistemas periciais

A utilização de técnicas imprecisas tem tido grande aceitação. A comparação de nperformance entre o controlo por técnicas imprecisas e outros métodos tem sido estudado por alguns autores. **Coleman *et al* (1994)** compara a performance do PID clássico, do controlo por lógica imprecisa e pelo método de controlo por deslizamento, quando os parâmetros do processo sob controlo variam. A conclusão a que o autor chega é que controlo por deslizamento é melhor que controlo por lógica imprecisa que por sua vez é melhor que o PID clássico. Este resultado não espanta, pois os teste são feitos na zona de aplicabilidade óptima do controlo em modo de deslizamento, tal como referido em 5.3-Controlo por modo de deslizamento, pág. 78. Também **Dister *et al* (1993)** confirma que o controlo por lógica imprecisa fornece soluções de controlo mais robustas que o PI convencional.

Jager (1995, Anexo B), demonstra que os controladores lineares (convencionais) estão contidos e podem ser "emulados" pela lógica imprecisa. O controlo por lógica imprecisa é, portanto, mais rico e mais flexível que o convencional, daí resultando, em geral, métodos de controlo mais robustos, se o sistema tiver sido desenvolvido com esse objectivo.

Um dos aspectos mais promissores da lógica imprecisa é o seu relacionamento com outras formas de I.A. (**Jager, 1995** e **Harris, 1993**):

- Ajuste *off-line* - as técnicas de treino das redes neuronais podem ser utilizadas para moldar automaticamente a estrutura das regiões imprecisas através da simulação de entradas e saídas desejadas (conjuntos de treino), durante a fase de desenho do controlador
- A lógica imprecisa pode tornar-se adaptativa, sendo as estrutura das regiões imprecisas moldadas durante o próprio processo, através das técnicas de aprendizagem *on-line*, provenientes de outras áreas da I.A.

6.7 Resumo

Ao longo deste capítulo foi analisado como a I.A. fornece meios para o desenvolvimento de controladores robustos. Foi também referido que a I.A. encoraja as simulações que descrevem o funcionamento macroscópico do sistema em vez de atender ao seu funcionamento interno. Estas características tornam-se importantes no desenvolvimento do controlador robusto devido à necessidade de ser implementado com o processo em serviço industrial. Assim, a simulação apropriada do processo e controlador assume primordial importância.

7. Simulação do CoPSE

7.1 Identificação

A identificação de parâmetros para um modelo é pesquisada em (Åström *et al*, 1971), onde aparece uma definição para esta classe de problemas: "Identificação é a determinação, baseada no relacionamento Entrada/Saída, de um sistema dentro de uma certa classe, que é equivalente ao sistema sob teste"[1]. Dentro desta perspectiva, e no âmbito dos modelos utilizáveis em metodologias de I.A., procurou-se obter um modelo que descreva o funcionamento macroscópico do sistema sem atender ao seu funcionamento interno, extremamente complexo e com muitos pormenores desconhecidos. O modelo aqui desenvolvido é apropriado para o fim em vista: permitir um fácil desenvolvimento do controlador para este sistema.

Tal como referido anteriormente, secção "5.2-Aquisição dos parâmetros do sistema *on-line*" pág. 77, não seria muito correcto estimar os parâmetros deste sistema à luz da Teoria dos Sistemas convencional porque alguns dados são filtrados e outros gerados de modo totalmente artificial: durante os períodos em que a corrente é nula, obtém-se valores de temperatura constante, o que não é verdade.

No entanto, foi possível desenvolver um "modelo" para o sistema. Na sua forma, não se aproxima da abordagem tradicional da Teoria dos Sistemas.

7.2 Modelo

Este "modelo" foi gerado tendo em conta os conhecimentos acerca do desenrolar do processo e é constituído por equações simples que relacionam tempo, correntes anteriores e temperatura actual, tal como estas grandezas apareceriam representadas no controlador de temperatura do processo real. Esta abordagem permite a simulação do processo em malha fechada, o que por sua vez permite que o controlador seja desenhado e afinado rapidamente. Esta simulação assume ainda maior importância tendo em conta que o processo é muito demorado e que o sistema é embebido, não sendo por isso possível alterar o programa sem mudar a EPROM que contém o programa.

[1] Citação de L. A. Zadeh, "From Circuit Theory to Systems Theory" Proc. Of IRE 50, 856-865 (1962)

No que diz respeito ao modelo, a corrente é a grandeza de entrada e a temperatura é a saída, de tal forma que o modelo calcule a próxima medida de temperatura, face à história da corrente no sistema, ditada pelo controlador de temperatura. Deste modo será possível simular o processo em malha fechada (o diagrama de princípio do controlador de temperatura está na Figura 15).

A fórmula genérica é obtida empiricamente, por tentativas apoiadas no conhecimento do desenrolar do processo. O número de tentativas necessárias para obter a fórmula final acabou por ser inferior ao esperado. Foram utilizados os dados registados, guardados no Excel, que foram enviados de 30 em 30 segundos, a mesma velocidade com que é chamado o controlador de temperatura no sistema real.

No que toca à evolução macroscópica do processo, sabe-se que o processo acumula de alguma forma a energia que lhe vai sendo transferida, de acordo com a lei de Joule, $P=RI^2$. Sabe-se também que quando a corrente é nula durante algum tempo, a temperatura desce. De acordo com estas heurísticas, a fórmula genérica pesquisada foi:

$$T(t) = T(t-1) \times Perdas + R_{eqv} \times \sum_{j=0}^{mem} \left(I(t-j) \times Peso(j) \right)^{pot} \qquad (8)$$

Em que:

T(t) e *T(t-1)* - representam respectivamente as temperaturas actual e anterior

Perdas - Exponencial de perdas térmicas do transformador ($0<Perdas<1$)

I(t - j) - Corrente no instante de tempo j medidas antes (j=0 refere a corrente actual)

Peso(j) - Peso com que cada amostra anterior de corrente afecta a temperatura actual

pot(i) - Peso não linear das amostras de corrente, suposto igual para todas as amostras

mem - memória do sistema, quantas amostras anteriores é que são necessárias para calcular o acréscimo de temperatura actual

Os primeiros testes realizados revelaram de imediato que esta fórmula descrevia de alguma forma a realidade.

O parâmetro "*Perdas*" decorre da necessidade empírica de fazer descer a temperatura do transformador, quando a corrente é muito baixa ou mesmo nula. Estas perdas ocorrem sempre mas são visíveis apenas quando a corrente é nula ou muito baixa.

Relativamente às perdas por fenómenos de transferência termodinâmicos:

1. Perdas por condução: Aproximadamente nulas porque o sistema está a uma pressão que varia mas é sempre baixa (abaixo dos 20 mbar, ver p. ex. Figura 19)

2. Perdas por convexão: Não há correntes de fluídos em movimento em torno do transformador, excepto o vapor de água que é retirado rapidamente das imediações por acção das bombas de vácuo
3. Perdas por radiação: Ocorrem entre o transformador em causa e os outros transformadores (que têm temperaturas próximas) e principalmente com as paredes da cuba (próxima da temperatura ambiente)
4. A vaporização da líquidos consome alguma energia que tende a retardar o aquecimento do transformador, o ritmo da vaporização é imprevisível perante a natureza dos líquidos extraídos do processo e perante a variação imprevisível da pressão na cuba ao longo do tempo

A determinação das perdas foi desaconselhada devido à sua complexidade e devido ao facto de variarem significativamente ao longo do tempo (especialmente as referidas com o número 4). Como a aproximação a uma exponencial se mostrou razoável, não foi investido mais tempo nesta questão[1]. A evolução no tempo no sentido da diminuição das perdas pode ser traduzida por uma aproximação do parâmetro *Perdas* à unidade, representando o conceito do transformador arrefecer mais devagar. Algumas tentativas de ajuste rapidamente revelaram que o parâmetro *Perdas* deveria ter um valor na ordem de 0.99x.

O cálculo dos *Pesos(i)* e do valor *mem*, da memória do sistema foram os parâmetros de mais difícil determinação. De início, todos os pesos foram considerados unitários, para tentar determinar a memória do sistema. Uma questão importante seria qual o valor do atraso puro no sistema, isto é, será que a corrente actual contribui para a temperatura actual? E a corrente anterior? Tal como seria de esperar verificou-se que *Peso(j=0)*=0. Isto confere algum atraso puro ao sistema. Novas tentativas revelaram que a corrente anterior não influenciava ainda a temperatura actual (atraso puro ainda maior). A Figura 23 parece indicar que o talvez atraso puro do processo seja na ordem dos 2 minutos, a que correspondem 4 amostras. Mais tentativas de ajuste levaram à conclusão que a amostra anterior à ultima já afecta o sistema. A explicação pode ser que na Figura 23 é fácil confundir a memória do sistema com o atraso puro.

A prossecução do trabalho revela que a memória do sistema ronda as 5 amostras (apenas!). A variação do tamanho da memória (*mem*) afecta decididamente a forma do ajuste obtido,

[1] Na realidade também se experimentou a aproximação *(Tj-1)-20)*Perdas)* mas como não se verificaram melhoramentos, optou-se pela simplicidade. Tal deve-se, provavelmente, ao facto do parâmetro *Perdas* tomar importância na zona final do processo, onde a temperatura não varia muito, pelo que as diferenças entre as duas aproximações são compensadas durante o ajuste pelo próprio valor do parâmetro em causa

muito mais que os pesos das amostras, que acabaram por ser fixados tal como indicado na Tabela 5. Os valores escolhidos revelam uma abordagem pessimista pois descrevem o sistema como tendo um atraso importante. Esse tipo de sistemas é considerado geralmente como sendo de difícil controlo.

Tabela 5: Valores finais dos *Pesos* de cada amostra

j	0	1	2	3	4	*mem*=5	...
Peso(j)	0	0	1	1	1.1	1.2	0

Os ajustes feitos revelavam que a simulação acompanhava o andamento geral da temperatura mas as oscilações de temperatura da simulação nunca eram tão bruscas e tão profundas como as da temperatura medida.

O parâmetro *pot* não existia, de início, porque parecia óbvio que tomaria o valor 2 perante a ideia de que a temperatura subiria de acordo com a potência enviada para o sistema, calculada pela lei de Joule $P=RI^2$. Acabou por ser acrescentado posteriormente, para tentar artificialmente (sem grande justificação teórica) fazer com que as oscilações tomassem valores maiores. O parâmetro *pot* poderia tomar o valor 2.001 para melhorar quase imperceptivelmente o aspecto das curvas na zona final da secagem. Por simplicidade e por ser o valor teoricamente correcto, utilizou-se o valor 2.

Os parâmetros *Perdas* e R_{eqv} são os mais importantes. O seu ajuste expedito pode ser conseguido através da característica procura de objectivo "*goal seek*" do Excel, que utiliza um método numérico para levar um ponto do gráfico até ao valor pretendido, por mudança de um célula indicada. Para ajustar o início e comportamento geral da curva, pedir para obter resultados através da mudança de R_{eqv}. Para ajustar a parte final da curva (em que as correntes são muito baixas), alterar *Perdas*.

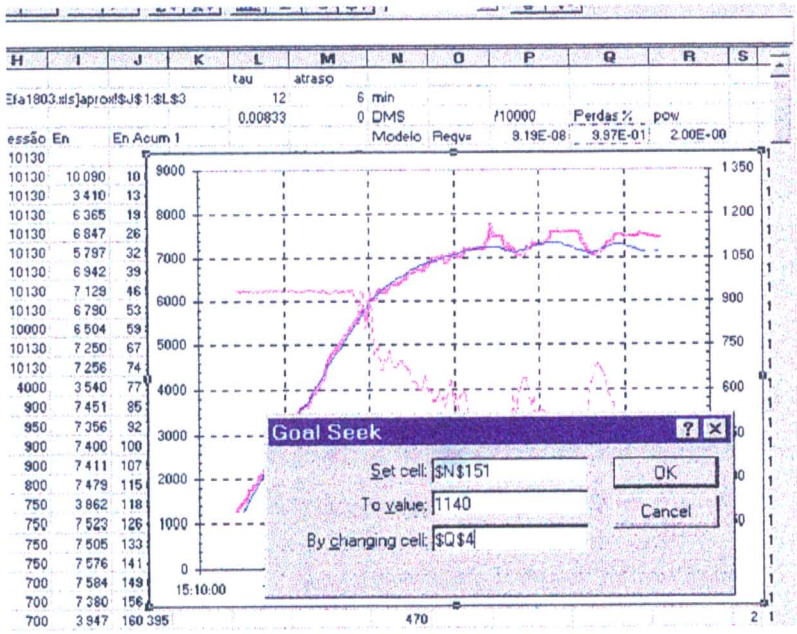


Figura 29: Procura de objectivos no Excel: Essencial para estimar os parâmetros da simulação *ad-hoc*

A equação final da simulação é mais simples que a original (8):

$$T(t) = T(t-1) \times Perdas + R_{eqv} \times \sum_{j=2}^{5} \left(I(t-j)\right)^2 \qquad (9)$$

7.3 Ajustes

Tal como se pode ver pelo traçado "Modelo 1" da Figura 30, a aproximação é bastante boa mas parece genericamente melhor comportada que a realidade (oscilações mais amortecidas na simulação que na realidade).

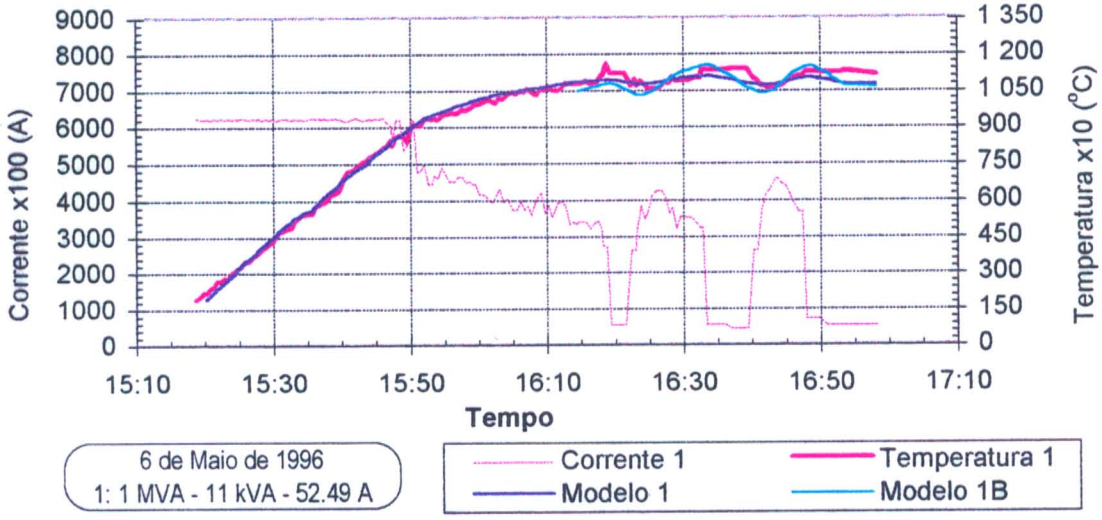


Figura 30:Simulação do comportamento da secagem do transformador através de dois métodos

Tal característica é provavelmente devida à mudança do valor de R_{eqv} durante o processo. Como contornar? Fácil: basta considerar o início do processo mais tarde! Pode-se utilizar a mesma relação (equação 9) para começar a simulação com outros parâmetros (*Perdas* e R_{eqv}) e com uma temperatura inicial retirada de uma fase mais avançada do processo. Veja-se como a curva "Modelo 1B" da Figura 30 se aproxima bastante mais da curva real, especialmente nas subidas de temperatura. As descidas de temperatura são abruptas devido ao repetidamente referido fenómeno da falta de informação quando a corrente é nula: a temperatura é mantida constante durante o tempo em que a corrente é nula e depois desce abruptamente quando finalmente se faz uma medida (não é uma descida em degrau porque as medidas são filtradas antes de serem enviadas).

Tal como se pode ver na Figura 31, a simulação com os parâmetros do transformador 1, de 1 MVA, "Modelo 1" é totalmente inadequada para o transformador 2, de 100kVA. Como as correntes do transformador nº2 são muito mais baixas que na secagem do nº1, a temperatura dessa simulação nunca chega a subir.

Estimando outros parâmetros R_{eqv} e *Perdas* para o transformador 2, pode-se ver que a simulação já se revela apropriada.

Comparando a Figura 31 e a Figura 32 podemos ver que os parâmetros determinados para os transformadores 2 e 3 (transformadores iguais[1]), geram simulações adequadas às respectivas temperaturas reais.

Também a simulação da evolução da temperatura do transformador 2 com a corrente do transformador 3 (Figura 32, "Modelo 2") e o recíproco (Figura 31, "Modelo 3")

[1] Mais do que terem iguais parâmetros eléctricos, estes transformadores têm características construtivas iguais. Claro que subsistem pormenores construtivos diferentes relativos à montagem manual de alguns componentes

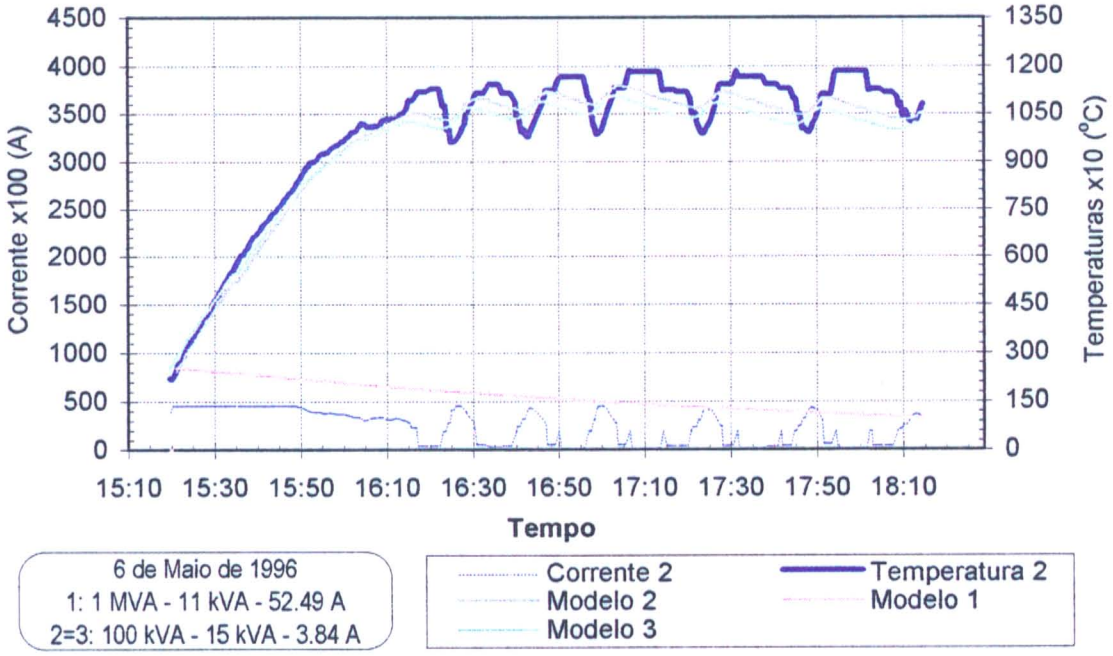


Figura 31: Temperatura durante a secagem do transformador n.º 2
Curvas comparativas das temperaturas estimadas para uma
secagem dos transformadores 1 e 3 com a corrente 2

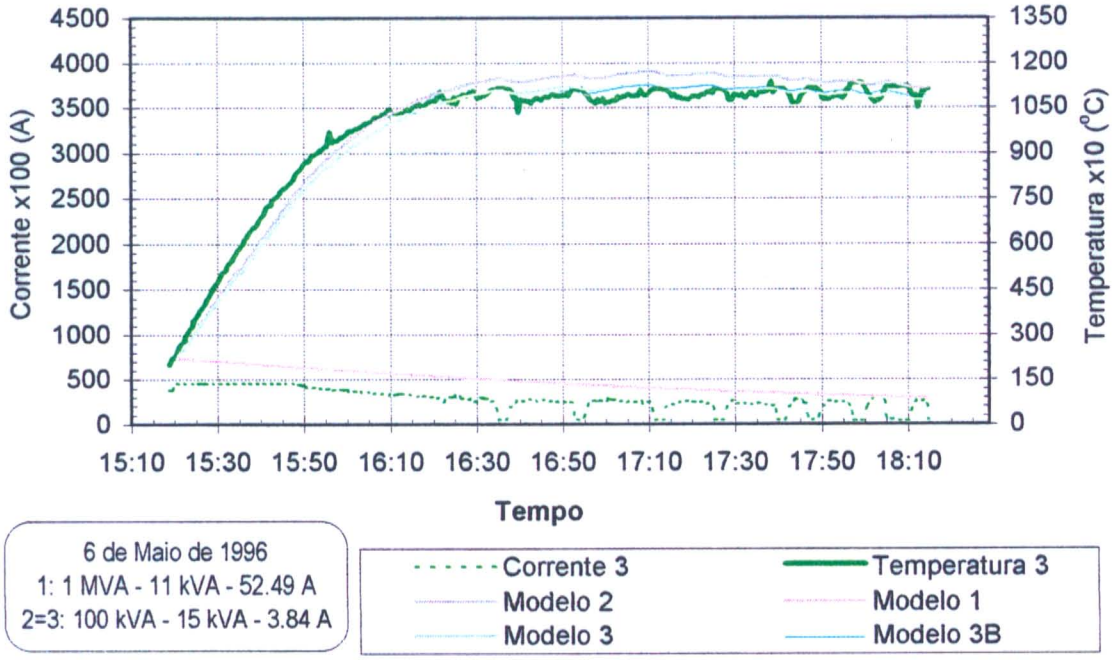


Figura 32: Temperatura durante a secagem do transformador n.º 3
Curvas comparativas das temperaturas estimadas para uma
secagem dos transformadores 1 e 2 com a corrente 3

7.4 Resumo

Neste capítulo apresentou-se uma simulação desenvolvida fora do âmbito da Teoria dos Sistemas convencional. A fórmula na base da simulação foi conseguida por tentativas orientadas pelo conhecimento macroscópico do desenrolar do processo.

O modelo tenta prever a evolução da temperatura, a partir dos estados anteriores. Deste modo é possível realizar a simulação do sistema em malha fechada.

Foram realizadas muitas tentativas de ajuste, que permitiram chegar à conclusão que a estrutura encontrada é suficientemente flexível para descrever a secagem de todos os transformadores, desde que os parâmetros *Reqv* e *Perdas* sejam ajustados para cada secagem. As discrepâncias entre valores reais e os do ajuste, na parte final do processo podem ser atribuídos à variação dos parâmetros durante a secagem e/ou a erros de medida. Na parte final do processo as correntes a medir são sempre de 50% da corrente nominal, existindo assim elevado conteúdo harmónico, que dificulta a medida precisa da temperatura. Para os valores em que existiam dúvidas, optou-se por soluções pessimistas por forma a obrigar a que o controlador seja desenhado para uma robustez máxima.

8. Implementação

Toda a implementação é baseada na alteração da rotina `atende_temperatura`, que corre periodicamente, de 30 em 30 segundos, sempre que existir uma medida de temperatura, ou seja, sempre que a corrente estiver estabilizada (num valor maior que $50\%I_N$), o que pode levar à acumulação de pequenos atrasos no tempo de chamada a esta rotina.

Muitas outras rotinas são executadas por intermédio de interrupções:

- Medida de corrente, com 10 bits de precisão, pelo que são necessárias várias conversões do A/D e respectiva modificação de referências internas do μC (cadência: 50Hz, as várias conversões são executadas sempre que a anterior termina)
- Cálculo da DFT para determinação da fase e amplitude do harmónico principal da corrente (cadência: logo que existam medidas suficientes e estabilizadas)
- Cálculo da resistência dos enrolamentos e nova temperatura, cálculo de medidas intermédias, filtragem de medidas, etc. (cadência: sempre que a operação anterior tiver sido concluída com sucesso)
- Malha de comando da corrente: comando da cadeia de medida da corrente (sempre que se muda o valor de corrente pretendido), alteração dos índices de MLI, eventual utilização do autotransformador de saída, geração de comandos para o hardware, geração de referências, etc. (cadência: 5 Hz)

Nota: Por vezes existe a necessidade de obrigar à existência exclusão mútua entre "processos" que leiam e alterem as mesmas variáveis – como na realidade não se trata de processos mas sim de interrupções, esta tarefa é simples.

A rotina `atende_temperatura` deve alterar a variável[1] `corr_pret` (corrente pretendida), que será lida pela rotina `atende_corrente` (executada de 0,2 em 0,2 segundos), que por sua vez se encarrega de tomas as providências para trazer a corrente para um valor próximo do seu valor pretendido. Como já foi referido, correntes menores que 50% da corrente nominal não podem ser utilizadas pelo que existe uma outra variável chamada `descanso` que dita o tempo que a corrente deverá passar em valor nulo[2] (Figura 33). Durante o período de tempo em que a corrente é nula não há novas medidas de

[1] Na versão 3.0 do IAR a utilização de variáveis volateis é feita na pilha (*stack*) do C, em memória externa, pelo que são extremamente lentas. Para se ganhar velocidade de processamento, todas as variáveis são globais estáticas. Versões posteriores do compilador de C tonam estáticas até as variáveis volateis com o objectivo de tornar os acessos mais velozes.

[2] Após a ordem de anular a corrente, o rectificador é parado. A energia acumulada no barramento intermédio tem ainda que ser consumida, isto é, enviada para a carga, o que demora um tempo variável a acontecer (o inversor continua sempre a funcionar)

temperatura e o controlador de temperatura não é chamado. Só será chamado depois do tempo de descanso e do tempo necessário para fazer mais uma leitura de temperatura.

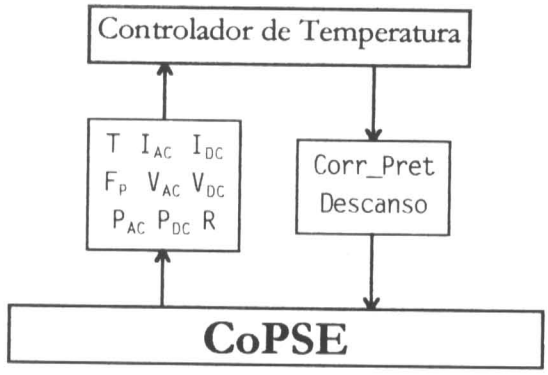


Figura 33: Tráfego de dados no controlador de temperatura
(*T* representa Temperatura, F_P factor de potência)

8.1 Introdução à Lógica de Controlo

Tal como referido, pretende-se fazer a introdução de conhecimento no controlador do CoPSE, de acordo com a técnica da lógica imprecisa (*Fuzzy Logic*).

O SieFuzzy é uma ferramenta computacional de ajuda à engenharia da programação (CASE - *Computer Aided Software Engineering*) e foi utilizada para o desenvolvimento rápido do controlado em vista.

O controlo que se pretende implementar terá uma entrada em temperatura e uma saída em corrente. O controlador deve permitir erro nulo em regime permanente (*steady state error*), ou seja, deve convergir de forma estável no final do processo para a referência. Tal pode ser obtido através das várias estruturas de controlo impreciso análogo ao controlo PID (Harris *et al* 1993, capítulo 4) convencional, mesmo sem a implementação de características adaptativas durante o decurso do processo, desde que o controlador tenha a robustez necessária.

O problema do controlo deste sistema ramifica-se em várias vertentes:

1. A primeira aproximação aos 110°C
2. A estabilidade da temperatura em regime final
3. A variação das características do processo durante o seu decurso
4. A adaptatibilidade do sistema a transformadores de dimensões muito diferentes

O perigo da primeira situação é a existência de uma sobreelevação da temperatura que exceda os limites de segurança, devido ao sistema ser integrativo e estar à algum tempo na corrente máxima de $120\%I_N$.

A segunda vertente diz respeito à qualidade da secagem, qualidade essa que se pretende melhorar com a aproximação e estabilização em torno dos 110°C.

As características 3 e 4 obrigam a que o sistema seja robusto tanto perante a mudança de parâmetros de uma secagem para outra como durante a mesma secagem.

Em primeira instância considerou-se que uma estrutura de lógica imprecisa, do tipo Proporcional + Derivativo (Harris *et al* 1993, capítulo 4), com saída "em velocidade", seria indicada para o controlo do processo. A utilização da saída "em velocidade", significa que a matemática do controlador dita a variação necessária para a saída[1] do sistema em vez de ordenar um novo ponto de funcionamento em termos absolutos.

Como se pretende um sistema robusto, decidiu-se que a componente relativa ao tempo deveria ser eliminada do controlador e assim esta componente, apelidada de derivativa, deveria ser substituída. Surge, então, a pesquisa de qual o parâmetro a utilizar para obter o controlo inteligente/antecipativo ao ponto de reduzir a corrente antes da primeira travessia pelos 110°C, evitando assim a sobreelevação inicial do sistema?

A procura deste novo parâmetro revelou um potencial candidato: a energia acumulada pelo processo. Como é sabido, a energia é o integral da potência ao longo do tempo, pelo que é fácil calcular uma estimativa da energia ao longo do processo: basta calcular o somatório de todas as potências enviadas para obtermos uma estimativa da energia gasta na secagem do transformador. Este somatório produzirá numa energia acumulada e os valores obtidos dizem respeito à unidade "Watt vezes meio minuto", pois a potência é somada à cadência de 2 medidas por minuto, aproximadamente.

Pretende-se que o valor da energia acumulada durante a fase inicial do processo (corrente constante a $120\%I_N$) seja indicativa do que irá acontecer posteriormente. Esta relação deveria existir: a energia acumulada a uma certa temperatura (elevada) inclui os factores potência, tempo e também perdas. Como na fase inicial do processo a corrente é constante, deveria ser possível medir de alguma forma as perdas do transformador durante este período através da história das potências debitadas.

[1] Por simplicidade, considera-se que existe apenas uma saída para o sistema: a corrente desejável para o sistema, que depois será utilizada para calcular `corr_pret` e `descanso`

Foram estudadas as energias acumuladas aos 80°C e aos 100°C, os resultados são resumidos na Tabela 6.

Tabela 6: Quadro resumo das energias acumuladas

	Energia Acumulada (kW * (meio minuto)), no ..., aos ...								
	Transformador (i)			Transformador (ii)			Transformador (iii)		
Ensaio	80°C	100°C	Final	80°C	100°C	Final	80°C	100°C	Final
6/5	1MVA - 11kV			100kVA- 15kV			100kVA - 15kV		
	369 48%	554.3 71%	775.5 100%	63.8 46%	97.3 70%	138.5 100%	61.3 42%	98.1 68%	144.8 100%
18/3	400kVA - 15kV			400kVA - 15kV			400kVA - 15kV		
	133.9 35%	205.8 54%	384.0 100%	137.2 36%	207.7 55%	380.3 100%	153.8 41%	220.9 58%	378.8 100%
19/4 (Cuba 2)	250kVA - 15kV			630kVA - 10kV			50kVA - 15kV		
	163.4 49%	217.8 66%	330.6 100%	118.2 36%	178.9 54%	332.6 100%	209.3 64%	251.7 77%	325.6 100%

Apesar da variação dos valores encontrados, para transformadores de 50kVA até 1000kVA, as relações percentuais para a energia acumulada aos 80°C face à energia acumulada até ao fim da secagem apresentam uma característica em comum: a energia aos 80°C é sempre acima de 33% da energia ao fim do processo; também a energia acumulada até aos 100°C está sempre acima dos 50% da energia total.

Perante tal conclusão, seria de admitir que, depois de ter a energia acumulada aos 80°C, se pudesse prever a (primeira) passagem pelos 100°C através da energia: seria um factor que permitiria antecipar a chegada à zona onde a corrente deve ser baixada para não existir sobreelevação de temperatura.

Pretende-se, pois, construir um controlador do tipo P, utilizando lógica imprecisa, de duas entradas (temperatura e energia acumulada) e onde a saída é a variação pretendida na corrente. Este tipo de controlador é correspondente ao PI convencional, mas alargando o seu conceito para duas entradas. O referido tipo de controlador é representado facilmente no SieFuzzy (Figura 34), como havendo um conjunto de regras que liga duas entradas a uma saída.

No que diz respeito ao SieFuzzy, as duas entradas "ErrTemp" e "ErrEn" e saída "DeltaCorr" são cristalinas, "Crisp SBYTE", constituídas por um inteiro de um byte com sinal. Para o SieFuzzy, uma variável cristalina representa um número e uma variável do tipo *Fuzzy* imprecisa representa um vector de graus de verdade.

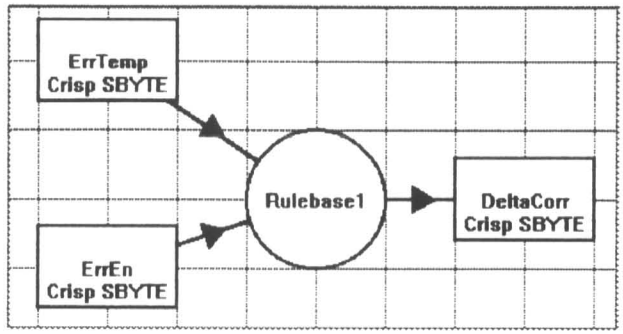


Figura 34: Estrutura do conjunto de regras imprecisas proposto para o novo controlador do CoPSE

À partida, todos os vectores imprecisos foram definidos com forma trapezoidal, por uma questão de facilidade de representação e o tipo de conhecimento apresentado ser apropriado[1].

Tal como preconizado por **Jager (1995)**, devem ser utilizadas regiões imprecisas com forma trapezoidal ou em forma de "S" com o objectivo de evitar variações bruscas na superfície de controlo gerada. De acordo com a mesma fonte, o grau de sobreposição das regiões imprecisas define o "amortecimento" da superfície de controlo obtida. Estes dados foram também confirmados durante o decorrer do trabalho.

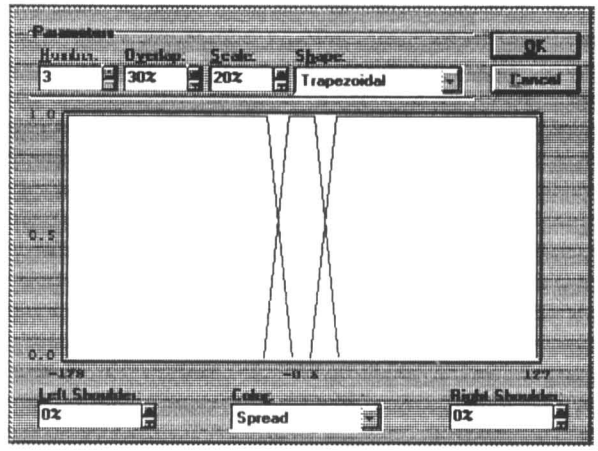


Figura 35: Definição da variável imprecisa (*Fuzzy*) "ErrTemp"

Note-se como a definição da variável "ErrTemp" (Figura 35) classifica como erro Z[2] (nulo) apenas valores muito próximos de zero, o que trará um funcionamento com um pequeno *ripple* de temperatura.

[1] As superfícies de controlo com 50x20 amostras e regiões trapezoidais ou com regiões em forma de S são, aparentemente, iguais

[2] O SieFuzzy obriga aos nomes das varáveis imprecisas (*fuzzy*), pelo que não é possível arranjar outros nomes que não os standard "N", "Z", "P" respectivamente para "negativo", "zero", "positivo"

Nota: O SieFuzzy define os nomes das regiões imprecisas, pelo que não é possível arranjar outros nomes que não os standard "N", "Z", "P" respectivamente para "negativo", "zero", "positivo"

A definição das variáveis imprecisas como sendo vectores de bytes deve-se à necessidade de a implementação ocupar pouco espaço e ser rápida. Como tal, apenas estão disponíveis 256 valores discretos no domínio da variável.

A fuzzificação da variável "ErrTemp" (Figura 36) será apenas a diferença entre a referência de temperatura que se pretende e o valor real medido, em °C. Se não existisse um factor de correcção, α, então a precisão do sistema estaria limitada a 1°C.

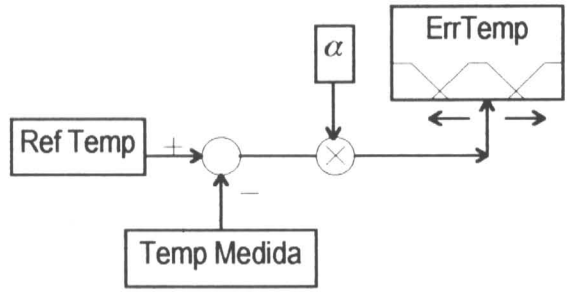


Figura 36: Fuzzificação da variável "ErrTemp"

A definição das regiões imprecisas para a variável erro de energia sugerida pelo SieFuzzy foi aceite pois a manipulação dos dados durante a fuzzificação é suficiente para o efeito pretendido ser atingido.

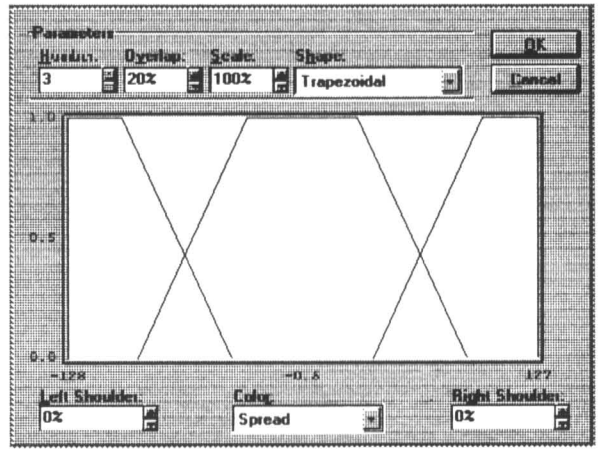


Figura 37: Definição de "ErrEn"

A fuzzificação de "ErrEn" foi escolhida de maneira a saber o que é um erro de energia relevante.

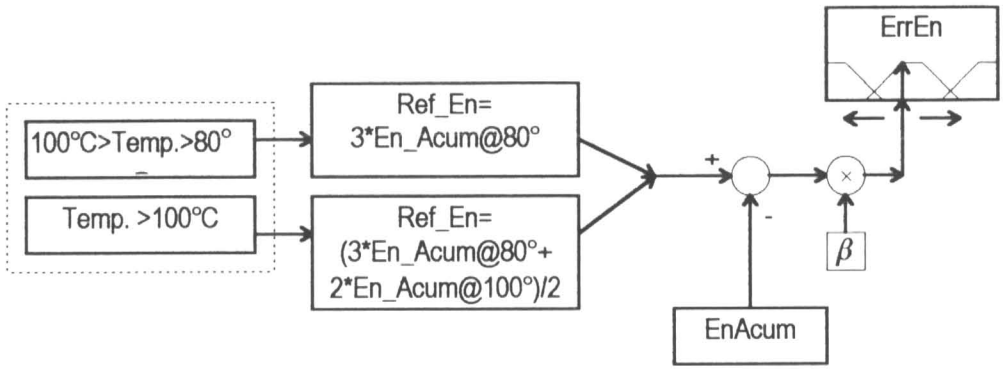


Figura 38:Fuzificação da variável "ErrEn"

Qual o valor do erro de energia que fará descer o valor da corrente antes da temperatura atingir o valor de 110°C? A utilização de um factor $\beta=1/1000$ garante que o erro de energia (na unidade kW*minuto*2) se situa do domínio da variável ErrEn, que é de -128 a +127. O resto do algoritmo de fuzzificação da energia será apresentado durante a discussão das regras propostas para o controlo e é representado na Figura 38.

Para a corrente de saída optou-se por utilizar 5 regiões imprecisas para descrever o domínio, com o objectivo de fazer um comando de precisão, necessário por causa da saída em velocidade (Figura 39).

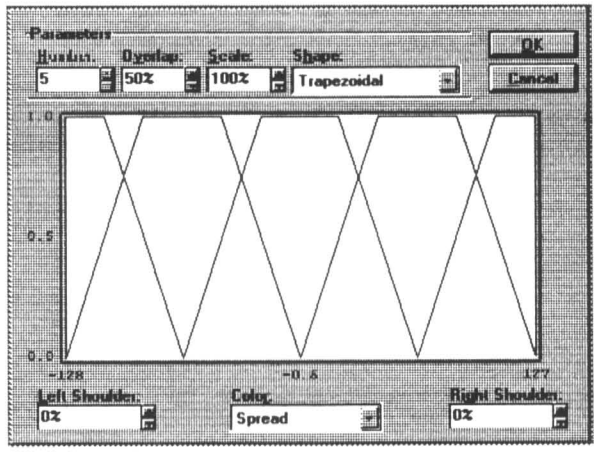


Figura 39: Definição Imprecisa (*Fuzzy*) da variável de saída do controlador, "DeltaCorr"

A actuação do controlador sobre a saída é sujeita a várias operações de ajuste, tal como representado na Figura 40. O valor da desfuzificação da variável de saída imprecisa "DeltaCorr" é multiplicada por uma série de factores correctivos, que pretendem tornar o sistema o mais robusto possível:

- Corrente nominal do transformador a secar: como a parte imprecisa do controlador não tem em conta o tamanho do transformador, a acção deverá ser proporcional à corrente nominal do transformador

- Ganho do controlador: Um parâmetro ajustável, a ser introduzido pelo utilizador, com o propósito de se estudar diferentes performances do controlador
- Ajuste de resistência: como a actuação é feita em corrente, a variação da resistência alteraria o valor da potência injectada no sistema

Estes factores de "adaptação" são importantes porque tornam o sistema real mais controlável e mais próximo da simulação.

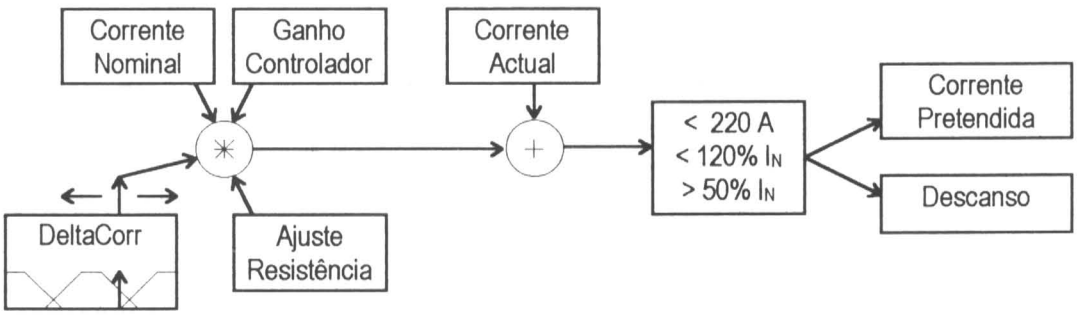
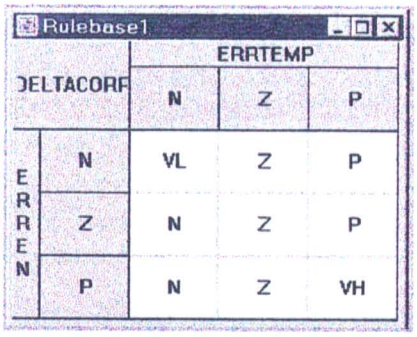


Figura 40: Actuação do controlador sobre a corrente (posterior à desfuzzificação)

8.2 Regras de Inferência

Tendo a estrutura do controlador toda delineada, só falta desenhar o conjunto de regras de inferência. Estas regras devem reflectir um comportamento de realimentação negativa, tendente a estabilizar o sistema em malha fechada, por forma a anular o erro de temperatura e devem permitir que a corrente desça para valores mais pequenos perante a primeira aproximação aos 110°C. Assim, se o erro de temperatura se está a aproximar do zero impreciso com uma energia demasiado grande, deve haver uma forte acção de redução da corrente. Esta forma de pensar foi implementada com as regras mostradas na Tabela 7.

Tabela 7: Tabela de regras do controlador



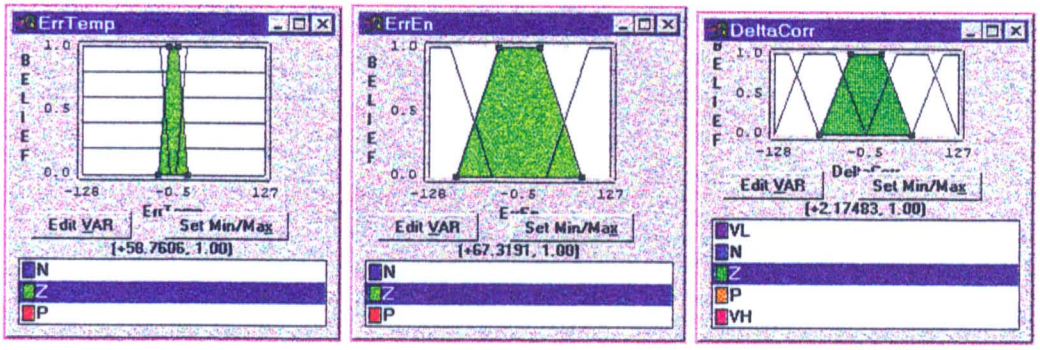


Figura 41: Definição das regiões para cada variável imprecisa
(os nomes das regiões são definidos pelo SieFuzzy)

Quando ErrTemp é P (positivo), então é porque Ref Temp> Temp Medida (ver Figura 36) e é necessário subir o valor da corrente actual: Deltacorr é P (as regiões são definidas na Figura 41). Se o erro de energia for também P (positivo), de acordo com a Figura 38, então estamos longe da zona de perigo no parâmetro da energia e a acção deve ser muito grande (VH - *Very High*). Da maneira como o sistema está construído, quando a energia ultrapassar a sua referência, ErrEn é N (negativo) e o controlador deixa ainda subir a corrente até se ultrapassar a referência de temperatura. Quando tal acontecer, então ErrTemp será N e também ErrEn será N. Existirá, então, uma forte acção (DeltaCorr é *Very Low*, VL) que faz baixar a corrente. Se não se tomassem providências doutra forma, devido ao sistema ser lento e integrativo, existiria sempre uma sobrelevação, pois a forte acção de comando só aparece quando o erro de temperatura é N, depois da temperatura ter ultrapassado a sua referência. Depois disso, acção contrária seria maior que o normal, ou seja, existe o comportamento que se pretendia implementar mas em atraso. Para obrigar a que este funcionamento aconteça antes da chegada aos 110°C, optou-se por fazer baixar a referência de temperatura até aos 100°C durante a primeira aproximação – como tenderá a haver uma sobrelevação controlada pelo controlador, a temperatura excederá sempre, mas não por muito, os 105°C. Quando se ultrapassar os 105°C pela primeira vez, então a referência é mudada para o seu valor definitivo, 110°C. Depois da referência estar no seu valor final, o controlador tratará de gerir a nova aproximação ao valor final, já com

valores de corrente mais próximos dos definitivos, o que levará o controlo a ser bastante mais preciso na aproximação aos 110°C, tornando o sistema com uma sobrelevação que se pretende perfeitamente controlada.

As regras de inferência em conjunto com as definições das variáveis imprecisas geram a superfície de controlo impreciso mostrada na Figura 42, que mostra as acções de comando tomadas perante todas as combinações das entradas[1].

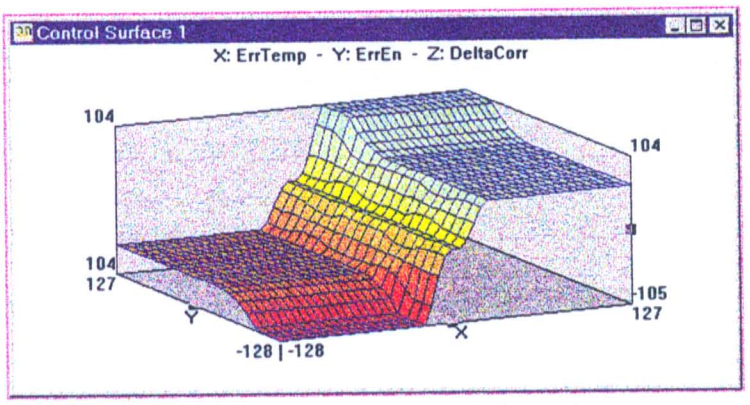


Figura 42: Superfície de controlo impreciso,
(50 amostras ErrTemp + 20 amostras ErrEn)

Este raciocínio será testado utilizando as potencialidades de simulação do SieFuzzy e posteriormente serão introduzidos melhoramentos tanto na simulação como no controlador.

Para fazer um rápido teste à lógica de controlo, basta utilizar a funcionalidade de teste rápido do SieFuzzy, muito prática para ir considerando qual a melhor maneira de converter as entradas e ver quais as saídas, especialmente interessante quando se tem muitas regras que se pretende testar em conjunto pois pode-se ter várias instâncias desta funcionalidade de teste rápido. Este modo de funcionamento é mostrado na Figura 43, para o conjunto de regras apresentado anteriormente.

[1] A lógica da mudança da referência de temperatura, nada tem a ver com a superfície de controlo

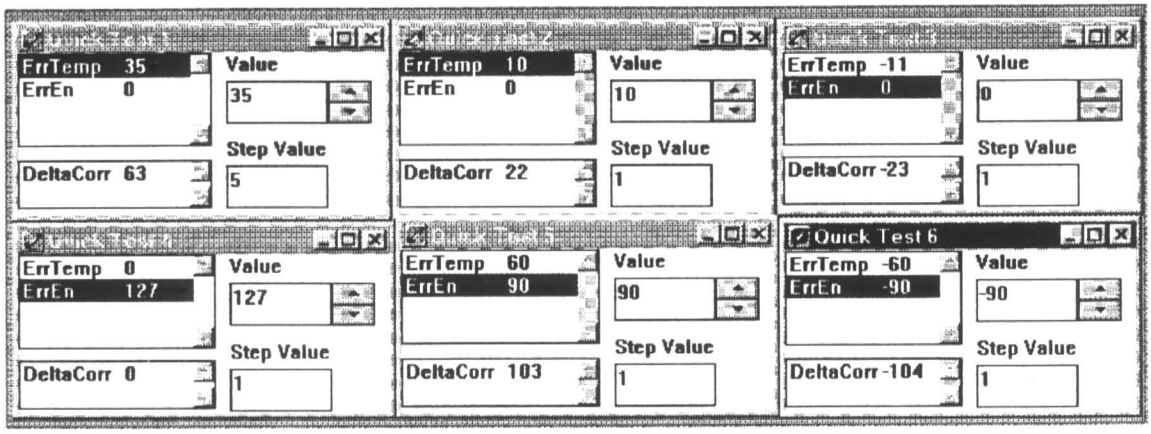


Figura 43:Teste rápido do SieFuzzy

8.3 Tabela vs. Matemática Imprecisa

A implementação do protótipo de controlador apresentado obrigou a que a implementação fosse conseguida perante as apertadas restrições de espaço de memória presentes no CoPSE, apresentadas na Tabela 2.

A matemática *fuzzy* é simples mas muito variada (Jager, 1995). Grande parte dos métodos de trabalho com conjuntos imprecisos são implementados nas livrarias incluídas (em código fonte -- linguagem C) no SieFuzzy. Cada método é implementado separadamente para impedir que o tamanho do código cresça demasiado.

A utilização das rotinas de matemática imprecisa permitem que o código produzido reflicta exactamente as regras de inferência implementadas, conduzindo a resultados exactos. Mas olhando para a Figura 42 podemos ver que é possível fazer previamente as contas da acção de comando resultante de todas as combinações possíveis das entradas.

Não é possível dispor de métricas exactas do tamanho do código executável mas pode-se dizer que o programa com a matemática imprecisa não coube na memória de programa (EPROM) disponível, cerca de 8 KB.

As vantagens (☺) e desvantagens (☹) da utilização de tabelas de pesquisa (LUT - *Look up table*) para a implementação de controladores baseados em lógica imprecisa são as seguintes:

☺ Permitem que as operações de inferência sejam calculadas previamente
☺ Troca de poder de cálculo computacional (tempo durante a execução) por tabelas (espaço de memória)
☺ As tabelas de pesquisa são constantes e ficam em ROM (em geral a ROM é mais abundante que RAM, em sistemas embebidos)

- ☺ Poupa-se RAM: seria necessário armazenar vários vectores temporários para executal os "cálculos" imprecisos
- ☺ Alguns μCs implementam pesquisas rápidas de tabelas directamente em *assembly*[1] pelo que o ganho de performance num μC será maior que num μP, desde que a implementação seja em *assembly* (com o SieFuzzy é em C)
- ☺ A utilização de interpolação linear sobre a tabela de pesquisa permite reduzir ainda mais memória ocupada:
  - Só é bom se a superfície de controlo for bem comportada (em geral é)
  - Gasta-se um pouco mais de poder de cálculo para concluir a interpolação
  - Solução intermédia entre só memória e só matemática
- ☹ A precisão vem deteriorada devido à amostragem
- ☺ A precisão obtida na aproximação é conhecida antecipadamente

Para o CoPSE (talvez para todos os S.E., em geral), a utilização de tabelas de pesquisa com interpolação (linear, para ser rápida) é ideal porque:

- ☺ Poupa muito poder de cálculo: a manipulação de vectores durante a execução das regras de inferência seria muito lenta, pois envolveria movimentação de muitos dados na memória externa, que é muito lenta, no 80535
- ☺ Poupa também a escassa memória disponível (ROM e também RAM)
- ☺ Foi possível descer o número de amostras da tabela de pesquisa até o programa caber na memória disponível sem aumentar o erro na aproximação (como referido, a superfície de controlo é bem comportada)
- ☺ Tendo em conta que o domínio das regiões imprecisas está dividido em 256 partes apenas, a precisão não é muito alterada pela utilização da tabela de pesquisa

Os valores reais, definitivos, para o cálculo da tabela de pesquisa foram (Figura 42):

20 amostras para "ErrEn"

50 amostras para "ErrTemp"

(utilizando um tabela de *char* com interpolação linear)

Vem uma tolerância de 13.2% no valor da saída "DeltaCorr"

Não foi possível obter melhores valores de precisão por forma ao programa caber no espaço disponível. O valores de 13.2% de tolerância em "DeltaCorr" é muito aceitável tendo em conta que o controlador é adaptativo e logo não haverá problema que a superfície de controlo seja interpolada, ou seja, que a superfície de controlo fique ainda mais gradual, com variações mais amortecidas. Não foi possível testar a viabilidade de outra solução pois não era possível sequer compilar a aplicação com a matemática imprecisa, devido à memória de programa não ser suficiente.

[1] O 80535 implementa esta característica, consultar p. ex. Yeralan *et al* (1995, pág. 263)

O tamanho ocupado pela tabela é de 1000 amostras, do tipo *char*, ou seja, apenas um byte.

O tamanho do código gerado pelo SieFuzzy é, no entanto, um pouco maior pois inclui as funções de pesquisa em tabelas n-dimensionais e as rotinas de interpolação. Para além destas rotinas, foi desenvolvido ainda mais código, nomeadamente para a fuzzificação, a desfuzzificação, e interface com pormenores de implementação do resto do sistema (consultar, a este respeito, o Anexo A).

Uma falha do SieFuzzy é que a simulação funciona sempre com matemática imprecisa. Como tal, não é possível simular o sistema com os resultados da tabela de pesquisa, sendo assim impossível avaliar se o número de amostras da tabela de pesquisa é exagerado, apropriado ou insuficiente.

8.4 Simulação, Programação e Resultados

O SieFuzzy inclui um interessante motor de simulação. A simulação é constituída por código numa linguagem de alto nível, própria do SieFuzzy, parecida com linguagem C (segunda afirma a documentação do programa). Para constituir uma simulação básica do problema basta preencher as caixas de iniciação e corpo da simulação com as funções matemáticas envolvidas no sistema (Figura 44).

O SieFuzzy executa a parte de iniciação do modelo da simulação, depois executa o modelo um certo número de vezes especificado e depois actualiza os gráficos especificados.

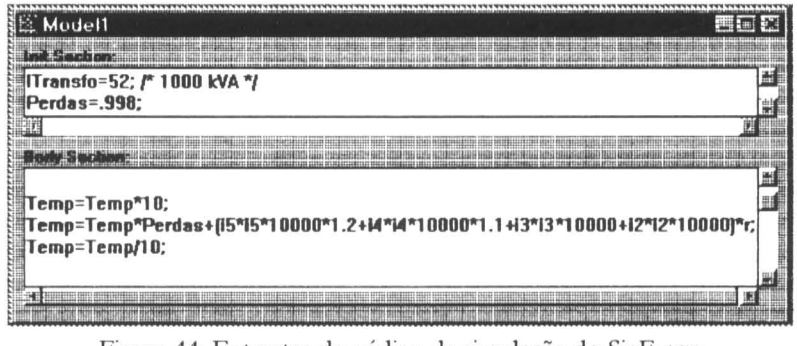


Figura 44: Extractos do código da simulação do SieFuzzy

Pela própria natureza do funcionamento em energia acumulada, que pretende medir as perdas do sistema durante o aquecimento em corrente constante durante a subida inicial da temperatura, não é fácil simular esta parte da lógica de inferência.

A simulação foi baseada no modelo do sistema que se tem vindo a apresentar neste capítulo. Para estudar a afinação do controlador basta utilizar o editor do SieFuzzy para alterar a forma das regiões imprecisas, mesmo durante a execução da simulação.

Utilizando no SieFuzzy os parâmetros encontrados pelos ajustes conseguidos no Excel, a partir dos registos de secagens anteriores, construiu-se uma simulação de forma rápida e poderosa. Os resultados desta primeira versão da simulação estão patentes na Figura 45, onde se simula o sistema em malha fechada, *i. e.* processo e controlador em conjunto. O erro de energia na parte inicial da simulação é mantido na região "*N*", para que a acção inicial seja vigorosa, no sentido de evitar a sobreelevação inicial. Quando se atinge os 100°C, o valor do erro de energia é anulado, região "*Z*", e controlador passa apenas a um controlador proporcional, de saída em velocidade (o que é equivalente a um controlador do tipo PI convencional).

A simulação utiliza os parâmetros conseguidos para os sistema através dos ajustes conseguidos na secção anterior. Tal como referido, cada ponto da simulação corresponde a 30 segundos reais, que é também a cadência a que é chamado o controlador.

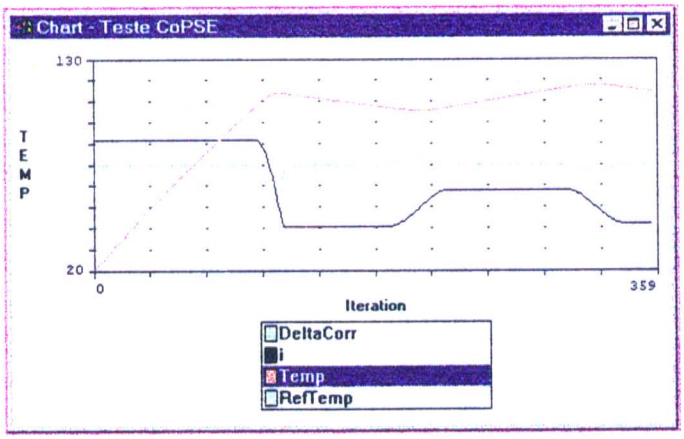


Figura 45: Resultados preliminares da simulação
Escalas: I de 0 a 100 A; Temp e RefTemp de 20 a 130°C
DeltaCorr de -127 a 127. Ganho Controlador=0.002.
Cada ponto corresponde a 30 seg. reais. O tempo representado corresponde a 3 horas, duração normal da secagem

Pode confirmar-se que toda a lógica é coerente, a temperatura desce antes dos 120°C. Parece que não existe uma acção drástica ao chegar aos 110°C, mas a regra n°4, "**se** ErrTemp=*N* **e** ErrEn=*N* **então** DeltaCorr=*VL*" é avaliada como tendo antecedente de valor não nulo 24 vezes, tal como documentado na Figura 47 (a numeração das regras está patente na Figura 46).

Rulebase1

	Name	On	ErrTemp	ErrEn	Weight	DeltaCorr
1	Rule2	AND	N	P	100%	N
2	Rule4	AND	N	N	100%	VI
3	Rule5	AND	P	P	100%	VH
4	Rule7	AND	Z	P	100%	Z
5	Rule8	AND	Z	N	100%	Z
6	Rule9	AND	P	N	100%	P
7	Rule1	AND	Z	Z	100%	Z
8	Rule3	AND	N	Z	100%	N
9	Rule6	AND	P	Z	100%	P
10						

Figura 46:Regras de controlo impreciso (na forma de texto)

Rulebase1 (Tuning)

Name	Alpha	Count	Info
Rule8	0%	124	E W:100% L:0 H:1
Rule4	0%	24	E W:100% L:0 H:0.380696
Rule2	0%	0	E W:100% L:0 H:0
Rule5	0%	0	E W:100% L:0 H:0
Rule7	0%	0	E W:100% L:0 H:0
Rule9	0%	0	E W:100% L:0 H:0
Rule1	100%	227	E W:100% L:0 H:1
Rule3	0%	10	E W:100% L:0 H:0.380314
Rule6	0%	58	E W:100% L:0 H:0.0548761

Figura 47: Contagem do número de vezes que cada regra foi avaliada com antecedente não nulo, para a simulação da Figura 45

Para testar que, de facto, existe uma grande actuação (DeltaCorr=VL) quando a temperatura sobe pela primeira vez acima dos 110°C basta alterar a simulação para a temperatura subir mais rapidamente, p. ex. 5 vezes mais depressa (*Reqv* da simulação 5 vezes maior). Os resultados são apresentados na Figura 48.

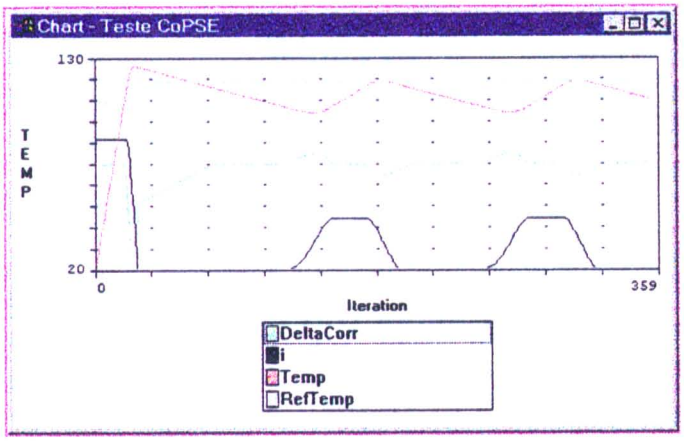


Figura 48:Teste do modo de funcionamento drástico (as escalas são as mesmas da Figura 45) Ganho Controlador=0.002

Na Figura 48 já se pode ver valore de DeltaCorr na região imprecisa VL, pelo que existe de facto alguma segurança confirmada: Se o processo funcionasse 5 vezes mais depressa do que a simulação, então existiria alguma sobreelevação mas sempre abaixo dos 130°C. Repare-se como a vigorosa acção de controlo anula a corrente por forma a evitar a subida da temperatura. A Figura 49 confirma que a regra 4 (Figura 46) foi actuada 9 vezes

(apenas!). Aqui podemos comprovar o raciocínio impreciso: na Figura 45 e correspondente contagem da Figura 47, a regra 4 é chamada mais vezes com menor grau de veracidade e a desfuzzificação acaba por nunca atingir valores na região de `VL` de `DeltaCorr`. Na Figura 48 e Figura 49, onde já é necessária a acção drástica, preconizada pela regra 4, "se ErrTemp=*N* e ErrEn=*N* então DeltaCorr=*VL*", esta regra é chamada menos vezes e com maior grau de confiança, abandonando-se rapidamente esta zona visto que a acção de comando é forte.

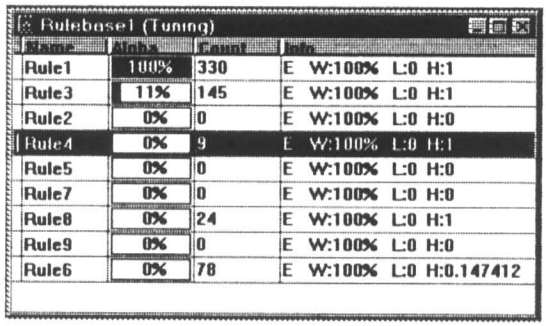


Figura 49:Contagem da actuação de cada
regra na simulação da Figura 48
(numeração das regras em forma de texto na Figura 46)

Como a actuação sobre a corrente de saída é proporcional à corrente nominal do transformador, este comportamento pode ser alargado a todos a todos os transformadores (isso foi confirmado perante muitas simulações). Existe no entanto o parâmetro ganho do controlador que foi mantido até agora no (apropriado) valor de 0.002. Na Figura 50 apresenta-se o plano corrente / temperatura do processo simulado em malha fechada, que se afigura totalmente estável.

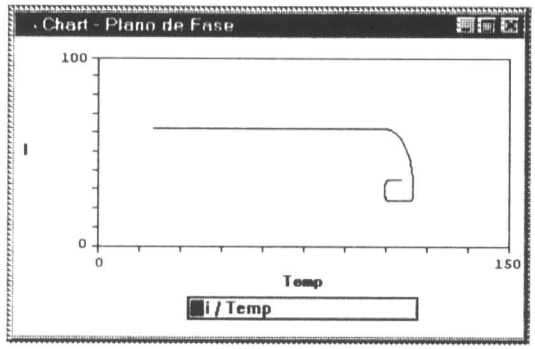


Figura 50: Plano Corrente Temperatura relativos
à simulação da Figura 45

O efeito da variação do ganho do controlador é mostrado nas figuras seguintes, que garantem também que o controlador é estável para todos os valores de ganho aqui testados. Como se pode ver, o sistema é tem sempre um *ripple* na saída (não converge para um regime estacionário), o que não seria de esperar tendo em conta a estrutura de saída em

velocidade do controlador. Na Figura 51 e na Figura 52 pode-se ver que se o ganho do controlador for demasiado pequeno, $2*10^{-4}$, o sistema torna-se mais lento e tem uma sobreelevação inicial maior, que se aproxima perigosamente do perigoso valor de 130°C.

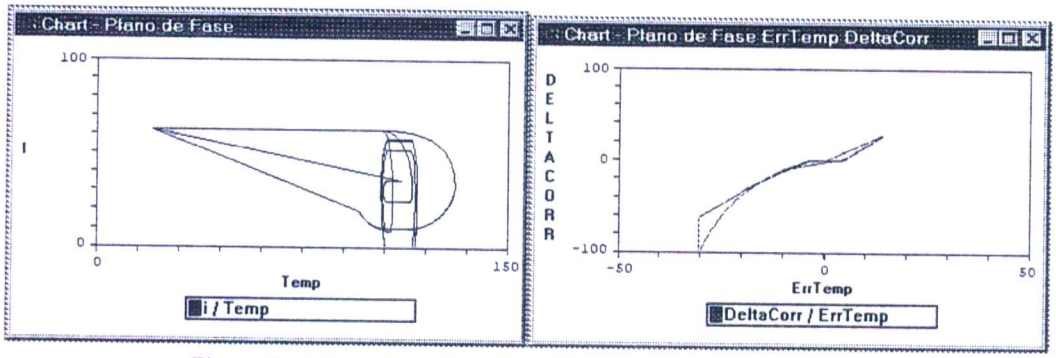


Figura 51: Planos DeltaCorr/DeltaTemp e DeltaCorr/ErrTemp
(para ganhos do controlador de: $2*10^{-4}$ 2, $2*10^{-3}$ e $2*10^{-2}$)

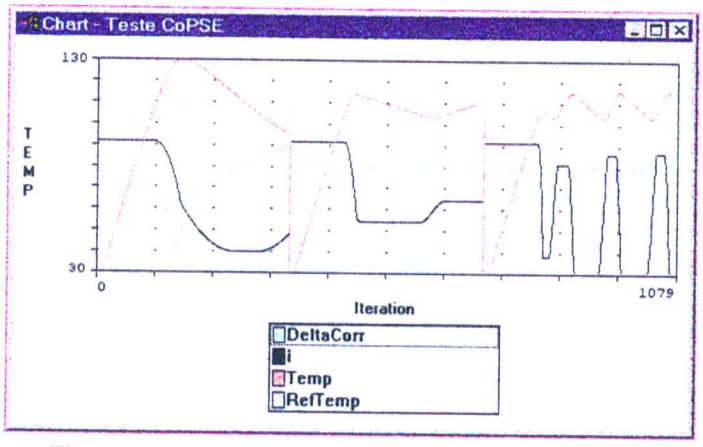


Figura 52:Simulações para ganhos de, respectivamente da esquerda para a direira: $2*10^{-4}$ 2, $2*10^{-3}$ e $2*10^{-2}$
Cada simulação dura 3 horas=360 pontos
I de 0 a 100 A; Temp e RefTemp de 30 a 130°C
DeltaCorr de -127 a 127; Ganho Controlador=0.002

8.4.1 Versão 1

Tal como já foi referido, esta versão utiliza uma tabela de pesquisa em ROM, programada em código C, gerado pelo SieFuzzy e depois compilada com o compilador de C para o 80535 da IAR, versão 3.0. O SieFuzzy permite gerar código C apropriado para vários compiladores, o compilador escolhido foi o Keil 8051, outra versão de compilador de C para a mesma família de μC. Os métodos de inferência e desfuzzificação foram mantidos nos valores sugeridos pelo SieFuzzy: o operador inferência é o **produto**, agregação é o **máximo**, conjunção é o **mínimo**. Este operadores produzem as simulações e a superfície apresentada anteriormente.

O controlador convencional anteriormente instalado baseava o seu funcionamento no envio de 2 minutos de potência para o sistema (tempo fixo) e depois esperar um máximo

de 20 minutos com a potência desligada. Esta situação é pior do que a actual onde se envia 30 segundos de potência e depois se descansa durante um máximo de 5 minutos. As medidas passam a ser mais frequentes e a precisão no comando da temperatura sobe também pois evita-se que a medida faça subir o valor da temperatura (que existiria com o envio de potência durante 2 minutos seguidos). Durante o tempo em que não existe corrente, o controlador fica sem medidas e o transformador fica sujeito não só a perdas desconhecidas e potencialmente variáveis no tempo como também fica sujeito a influências externas, p. ex. provenientes dos outros transformadores em secagem.

Para possibilitar a implementação prática do controlador, criou-se uma variável `corrente_devia` que mantém o valor que o controlador gostaria de impor ao sistema. Se essa variável for maior que 50% I_N, então a `corr_pret=corrente_devia` (e `descanso=0`). Implementar valores de corrente_devia menores inferiores a 50% I_N, o sistema faz um `descanso` múltiplo de 30 segundos, obtido através da "fórmula" (trecho de código):

```
descanso=(int) (0.5*corrente_nominal/corrente_devia+1)      (10)
```

A medida seguinte será efectuada, passado o tempo de descanso, à corrente de 50% I_N.

8.4.1.1 Simulação

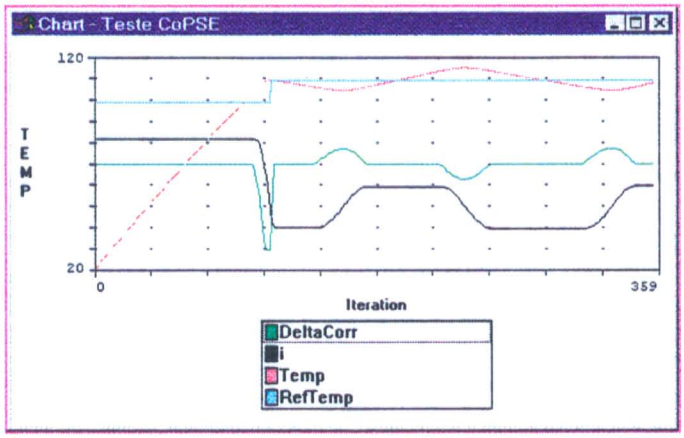


Figura 53: Simulação com os parâmetros utilizados no ensaio 1
G_{CTRL}=0.001 - (1MVA) - I_N=52A - R_{eqv}=5.75E-8 - Perdas=0.998
Escalas: I de 0 a 100 A; Temp e RefTemp de 20 a 120°C
DeltaCorr de -127 a 127. Cada ponto corresponde a 30 seg. reais.
O tempo representado corresponde à duração normal da secagem

A simulação da Figura 53 foi obtida antes do ensaio prático. Devido às característica integrativas do processo e ao facto do controlador só ser antecipativo para a primeira aproximação aos 110°C, a saída em velocidade parece não ser suficiente para fazer com que o processo em malha fechada convirja (sempre) para um valor final estável, apesar de a oscilação prevista ser mais pequena que o controlador convencional permitia.

Simulações com outros parâmetros demonstram que é possível fazer com que a temperatura final estabilizasse num valor final próximo da referência, não sendo porém essa a rega geral. A situação mostrada na Figura 53 é tomada como próxima da situação real possível, demostrando um comportamento que pode ser melhorado ainda. A existência de períodos de descanso foi, nesta primeira simulação, ignorada.

8.4.1.2 Resultados

Por exigências de segurança, a temperatura inicial da fonte 1 foi falseada e em vez de 20 foi inserido o valor 30 °C. Murphy, o pessimista, acabou por ter razão, e, durante este ensaio, o sistema de aquisição automático de dados não funcionou devidamente pelo que o processo da Figura 54 foi acompanhado manualmente. Deste modo, é possível assinalar exactamente cada ponto com nova informação de temperatura, onde o controlador de temperatura é chamado.

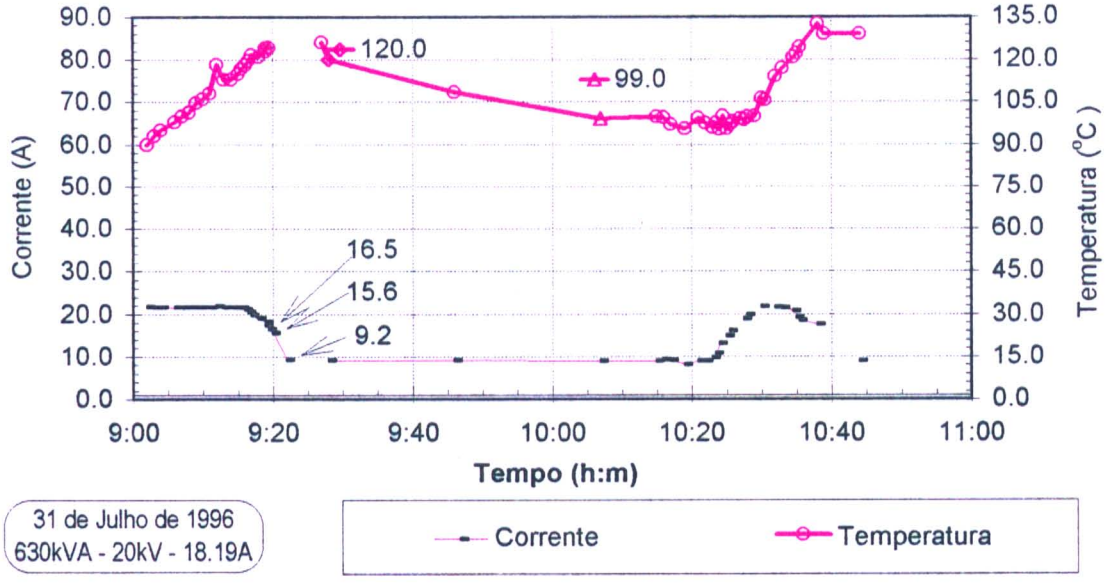


Figura 54:Resultados experimentais do controlador Impreciso v.1

Como foi utilizado um ganho de 0.001, o controlador do processo não conseguiu evitar uma grande sobrelevação e o sistema foi suspenso manualmente cerca das 9:22, aos 125°C, para evitar um disparo por excesso de temperatura. Para além do valor do ganho, esta sobreelevação pode-se ter devido ao algoritmo de energia não ter funcionado a tempo: como a temperatura inicial foi falseada, os 100°C foram atingidos com uma energia mais pequena que o esperado. A regra da energia poderá ter entrado em funcionamento (tardio) no ultimo registo de temperatura, antes do processo ter sido suspenso manualmente: a corrente tinha caído 6,4A e tal variação só é possível com actuações em DeltaCorr na

ordem dos -100 (região *VL*) o que obriga a que a regra 4 (Figura 46) tenha sido avaliada com elevado grau de veracidade.

O principal problema deste processo é revelado claramente entre as 9:30 e as 10:15, onde o controlador de temperatura apenas foi chamado 3 vezes. O grande problema é que, ao contrário de outros sistemas discretos no tempo, a medida não chega em intervalos regulares e o controlador não é chamado regularmente. Como a saída é em velocidade, sendo o controlador chamado poucas vezes, não é possível alterar a derivada da temperatura. O correcto comando do processo é reposto quando a corrente é maior que 50% I_N e o controlador passa a ser chamado de 30 em 30 segundos.

O sistema de segurança do CoPSE acabou por interromper a secagem pouco antes das 10:40 por a temperatura ter atingido os 130°C. Na retoma da secagem, o controlador ordenou a baixa de corrente a tal ponto que a próxima medida seria feita dentro de 30 minutos. Essa acção é exagerada mas não descabida: entre as 9:46 e 10:07, 21 minutos, a temperatura desceu cerca de 12°C. Como o transformador está a ficar cada vez mais seco, as perdas são menores e o transformador arrefece mais devagar. Porventura, o descanso de 30 minutos, levaria a uma descida de temperatura na ordem dos 18°C, o que a deixaria cerca dos 112°C.

A Figura 55 compara dois transformadores iguais, o transformador 1 com controlo por lógica imprecisa (que tem vindo a ser descrito) e o transformador 5, com o anterior controlo. O ganho do controlador é, ainda, de 0.001. Também neste caso foi necessário parar manualmente o processo por causa da tardia actuação da regra relativa à energia, mesmo assim nítida cerca das 12:45. Como as correntes necessárias durante a secagem têm valores percentuais maiores face à intensidade nominal transformador, no caso da referida Figura 55, pelo que o funcionamento do controlador é mais "fácil" e a temperatura não oscila tanto como no caso anterior.

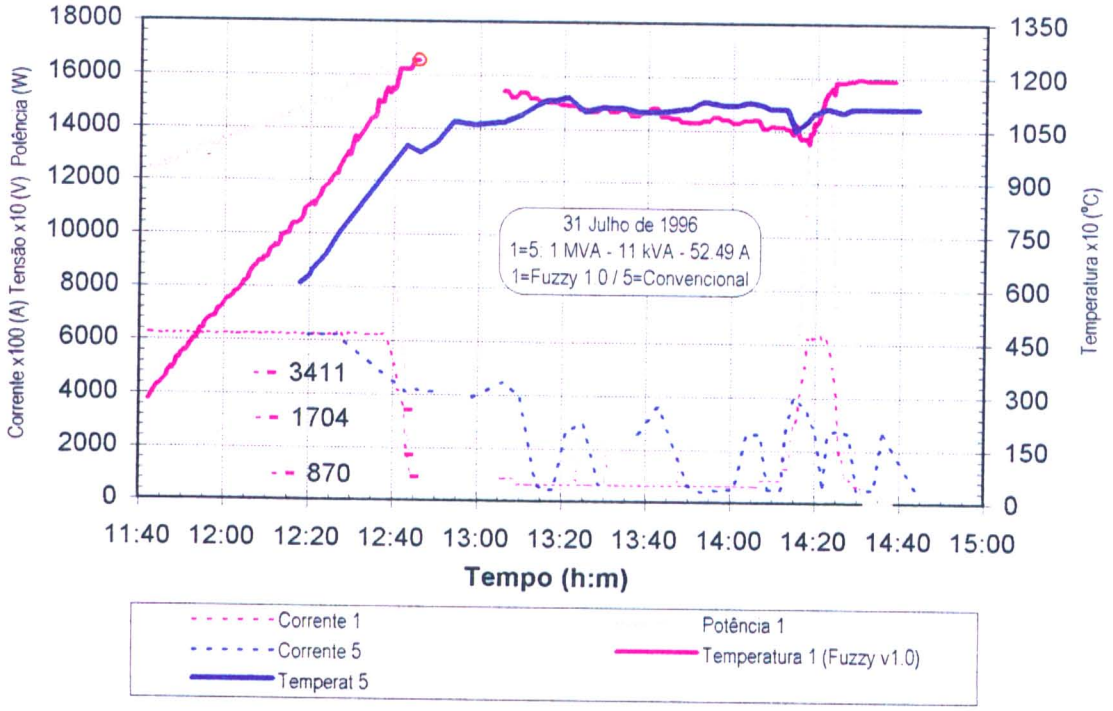


Figura 55: Resultados do controlador Impreciso v.1, comparando com um transformador igual no controlador antigo

A alteração do ganho do controlador para 0,002 torna o controlador muito melhor e já consegue "dominar" a sobrelevação inicial de temperatura. Também o seguimento é apropriado, apenas que no ponto onde a aquisição terminou o controlador tinha acabado de dar ordem para um tempo de descanso de 2 Horas, o que não pode acontecer.

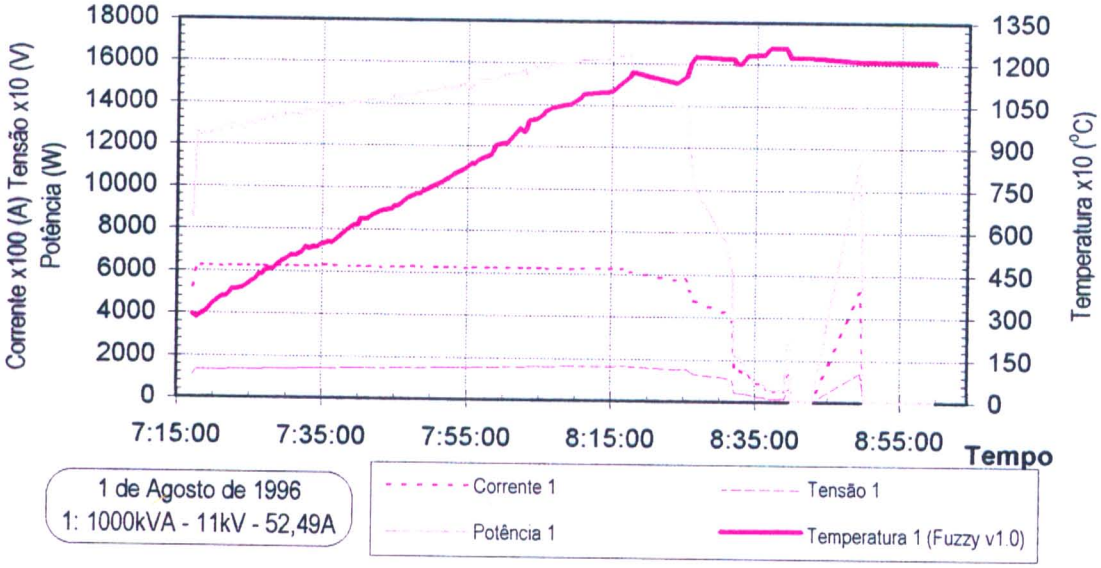


Figura 56: Resultados do controlador Impreciso v1.0 com ganho 0.002

8.4.2 Versão 2

Imperativos de ordem prática obrigaram a que o funcionamento com a acumulação de energia fosse substituído. Perante as conclusões anteriores pode-se verificar que este tipo de funcionamento não é imune a variações na temperatura inicial do processo, o que retira robustez ao controlador, para além de o tornar mais difícil de operar.

A parte da primeira aproximação aos 110°C foi substituída por uma série de decisões cristalinas (*crisp* - não imprecisas, não *fuzzy*) que reduzem progressivamente a corrente de modo a impedir que exista sobreelevação de temperatura, mesmo sem a utilização de qualquer parâmetro externo. Para evitar a sobreelevação inicial, aos 80°C a corrente é reduzida para o seu valor nominal e aos 100°C, ela é reduzida para 70% da corrente nominal do transformador. Os peritos confirmam que com este valor de corrente, todos os transformadores aquecem. O controlador de corrente deve ser apenas activado aos 110°C, doutra maneira tenderá a fazer elevar o valor da corrente (o que derrotaria o princípio de funcionamento que se pretende implementar).

Para dar mais segurança à parte final do processo, foram criadas garantias que a corrente nunca desce abaixo de 1.25% da corrente nominal (implica tempo máximo de descanso de 20 minutos). Também a corrente nunca poderá ser maior que $20\%I_N$, se a temperatura do transformador estiver acima dos 120°C. Esta regra é proveniente do conhecimento do que na fase final do processo de secagem, com a temperatura acima dos 120°C, transformador deve ser forçado a arrefecer e esta corrente obriga a isso mesmo. Estas regras foram implementadas de forma cristalina (*if... then... else...*), para que o tamanho do programa não cresça demasiado face ao pouco espaço disponível.

8.4.2.1 Simulação

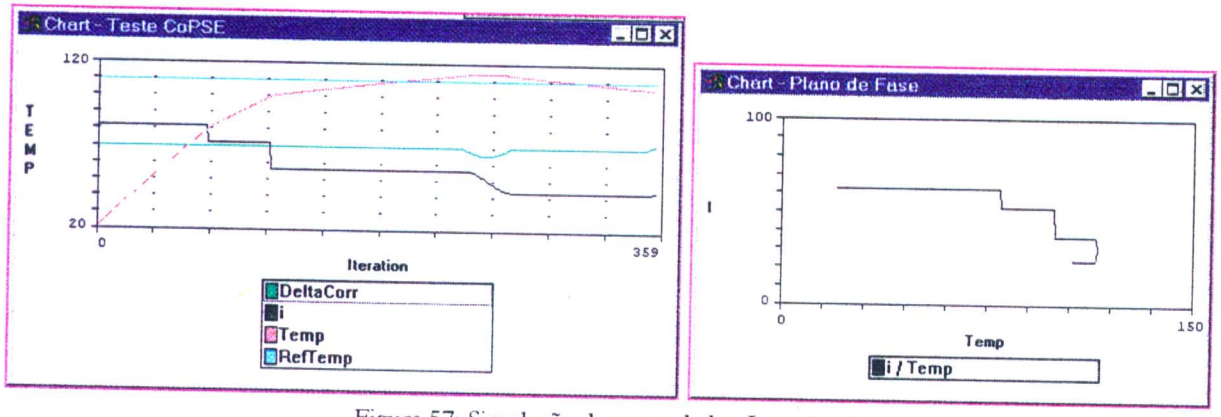


Figura 57: Simulação do controlador *Impreciso* v2

Tal como se pode ver, o controlo é robusto e a sobreelevação é eliminada de certeza, seja qual for a história anterior do sistema porque a aproximação aos 110°C é feita de forma muito lenta. O Controlador apenas entra em funcionamento aos 110°C, para fazer descer a corrente da forma mais correcta possível.

8.4.2.2 Resultados

Infelizmente, não foi possível ensaiar esta metodologia de eliminação da sobreleelevação inicial, devido a uma avaria de hardware na fonte onde estava a decorrer o ensaio[1].

A versão 1.0 apresenta já uma característica de seguimento da temperatura apreciavelmente boa, desde que o ganho do controlador seja 0,002.

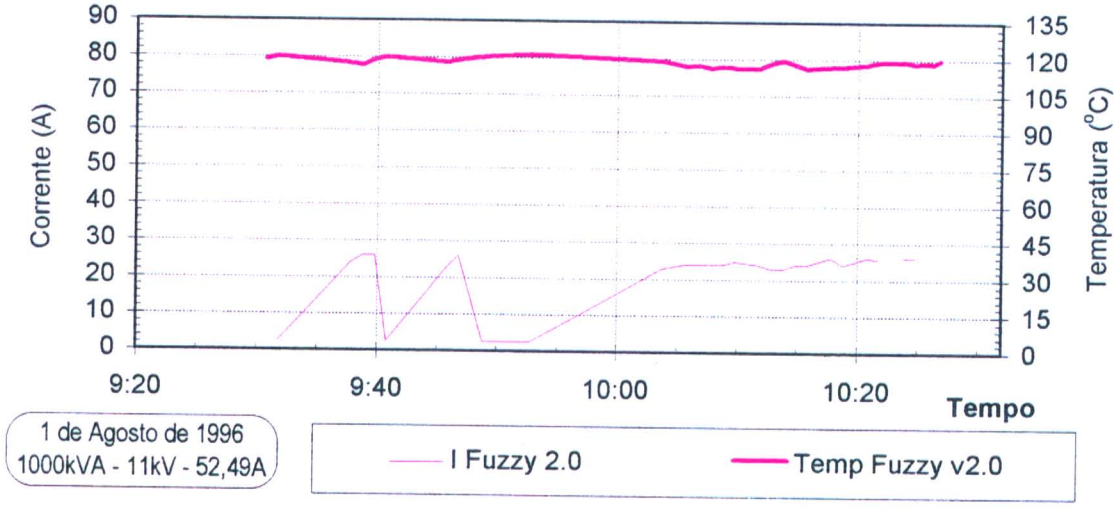


Figura 58: Trecho de actuação do controlador Impreciso 2.0
(ganho controlador=0.002)

A Figura 58 mostra os dados obtidos durante a parte do final do processo, que foi possível retomar depois da avaria.

O seguimento da corrente na parte final do processo é bastante melhor que na versão anterior do controlador. Tal facto deve-se a que a corrente pretendida na anterior versão do programa não era muitas vezes atingida. Esse facto foi contornado rescrevendo o código:

[1] Na realidade, a EPROM onde o programa estava gravado estava danificada e ocorreu um erro no decurso do processo, o que impediu que a aproximação aos 110°C decorresse com a versão apropriada do programa. A situação de erro era definitivamente devido à EPROM pois mudando para outra fonte a mesma EPROM, o processo não prosseguia e uma EPROM com igual programa funcionou bem logo a seguir.

```
corrente_pretendida = corrente_pretendida+…
```

como

```
corrente_pretendida = corrente_real_media+…
```

Esta mudança visa que o controlador de temperatura tome conhecimento de qualquer dificuldade do controlador de corrente em impor a corrente pretendida.

A partir do acompanhamento do processo enquanto ele decorre e também patente na Figura 58, pode-se concluir que a actuação sobre processo não se faz com variações contínuas, suaves. De facto, da forma como foi implementada, a corrente (média) varia de forma não contínua (equação (10)). Mantendo 30 segundos fixos a enviar potência a 50% I_N e sendo os tempos de descaso inteiros, existe um erro de truncatura não desprezável no cálculo da corrente média, tal como documenta a Figura 59. O erro cometido pode chegar a 25% da corrente nominal do transformador, a que correspondem cerca de 18% de erro na potência entregue ao sistema (valores percentuais face à grandeza nominal). Este erro ocorre na zona logo abaixo dos 50% I_N, zona essa que é muito utilizada na parte final do processo. Este factor ganha especial relevo devido à saída em velocidade exigir precisão na actuação, para que se possa convergir para um regime final estável, que se atingirá a maior parte das vezes a correntes inferiores a 50%I_N.

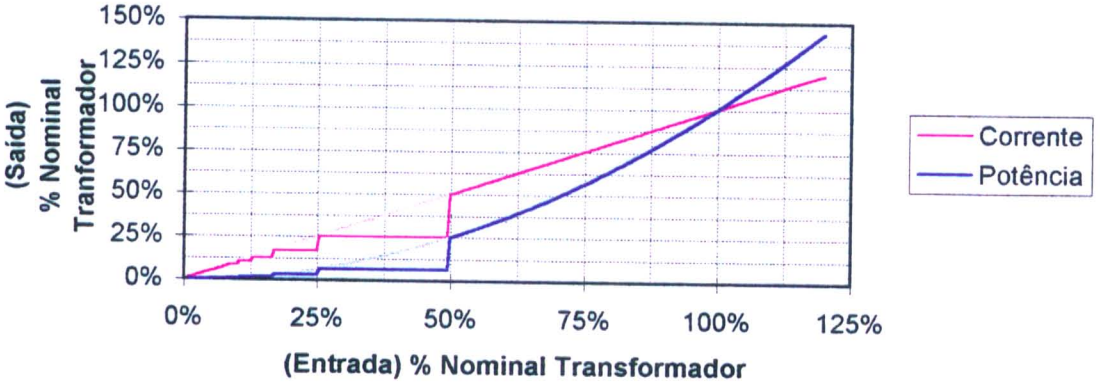


Figura 59: Discrepâncias entre correntes e potências pretendida e obtidas, considerando 30 segundos em 50%I_N e um tempo variável inteiro de descanso (erro de truncatura)
`descanso=ceil(0.5/corrente_devia)`

A Figura 59 realça que o CoPSE consegue debitar de forma apropriada desde a 50% a 120% da corrente nominal, a que correspondem deste 25% da potência nominal a 144% da potência nominal. Para correntes pretendidas abaixo de 50%I_N, entra-se em modo de liga/desliga e não se fornece os valores pretendidos de corrente e potência, devido a um erro que se pode chamar de truncatura, pois é devido ao facto do descanso ser inteiro.

A mudança feita no programa, de modo a basear a próxima corrente desejada pelo controlador na corrente real anterior, tende também a contrariar o erro de truncatura referido pela Figura 59 pois a nova corrente é calculada a partir do valor realmente existente no processo. O traçado da Figura 58 é bom também devido a esta razão.

A versão seguinte do controlador soluciona este problema.

8.4.3 Versão 3

Para o melhoramento do desempenho do controlador estudou-se a forma como se envia potência para o sistema. A formulação matemática aqui apresentada baseia-se na premissa de que o acréscimo de temperatura no sistema é proporcional à potência média debitada no intervalo em causa, o que é válido como aproximação da realidade, para a zona de estabilização de temperatura final.

A forma como se envia potência para o sistema, em pedestal, onde a corrente de $50\%I_N$, é enviada durante 30 segundos, seguida de um certo tempo de descanso, múltiplo de 30 segundos, é descrita pela Figura 60.

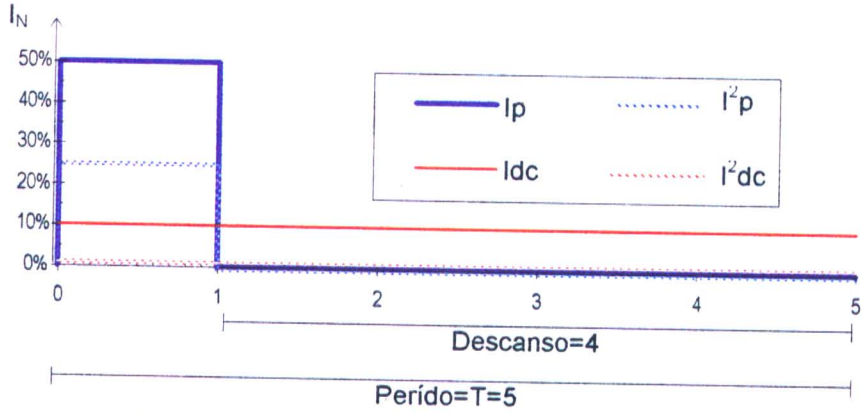


Figura 60: Ilustração do funcionamento em Liga/Desliga
para T=Descanso+1=5

A potência média que se pretende enviar para o sistema é $P_{med}^{I=I_{DC}} = (I_{DC}{}^2 \times T) / T = I_{DC}{}^2$ (ver Figura 60), resultante de uma corrente desejada, constante, ao longo de um período de tempo $T=(descanso+1)$.

Nota: Omite-se o factor *R*, que é irrelevante para a discussão apresentada

Nas versões 1 e 2 do controlador de temperatura, envia-se um pedestal de corrente, $I_P=0.5*I_N$, de duração 1 unidade de tempo (30 segundos), para um período de $(descanso+1)$ unidades de tempo, que fornece a potência média de $P_{med}^{I=I_P} = I_P{}^2 / T$, e assim:

$$P_{med} = \frac{1}{T}\int_0^T I^2 \rightarrow \begin{cases} P_{med}^{I=I_{DC}} = ({I_{DC}}^2 \times T)/T \\ P_{med}^{I=I_P} = ({I_P}^2 \times 1)/T \\ I_{DC} = I_p/T \end{cases} \rightarrow P_{med}^{I=I_P} = T \times P_{med}^{I=I_{DC}} \qquad (11)$$

A equação (11) demonstra que se comete um certo erro, inerente ao enviar a energia para o sistema sob a forma de um pedestal de potência. A potência média e logo o acréscimo de temperatura no transformador será maior que o desejado, na proporção do período da onda.

O estudo dos erros relacionados com a forma como se envia potência para o sistema revela duas fontes de erro:

1. O já referido erro de truncatura (Figura 59)
2. O Erro pelo facto de se enviar um pedestal de potência em vez de se enviar a potência distribuída no tempo, equação (11)

Uma metodologia para cancelar os erros anteriormente descritos seria deixar de obrigar a que o período de descanso seja múltiplo de 30 segundos. Esta mudança iria alterar toda a estrutura de conjunto do software e possivelmente levaria a problemas de hardware em cumprir ordens rápidas de comutação frequente entre o descanso e a alimentação da carga. Por estas razões, optou-se por outra solução mais simples.

O cancelamento dos erros referidos pode ser conseguido se o período em que se envia potência não tiver sempre a corrente em 50% I_N mas sim um outro valor, calculado por forma a que a potência média entregue ao sistema seja a pretendida.

Tal como referido anteriormente, a potência média fornecida ao sistema é $P_{med}^{I=I_P} = {I_P}^2 / T$, em vez de do valor que se pretendia enviar, $P_{med}^{I=I_{DC}} = {I_{DC}}^2$. Assim:

$$P_{med}^{I=I_{DC}} = P_{med}^{I=I_P} \rightarrow \frac{{I_P}^2}{T} = {I_{DC}}^2 \rightarrow T = descanso + 1 = ceil\left(\frac{{I_P}^2}{{I_{DC}}^2}\right) \qquad (12)$$

$$P_{med}^{I=I_{DC}} = P_{med}^{I=I_P} \rightarrow \left(\frac{{I_P}^2}{T}\right) = {I_{DC}}^2 \rightarrow I_P = I_{DC} \times \sqrt{T} \qquad (13)$$

A utilização das relações anteriores permitem o cancelamento dos erros devido à utilização de um pedestal de energia em vez a enviar de forma distribuída, tal como demonstrado também na Figura 61 (comparar face à Figura 59).

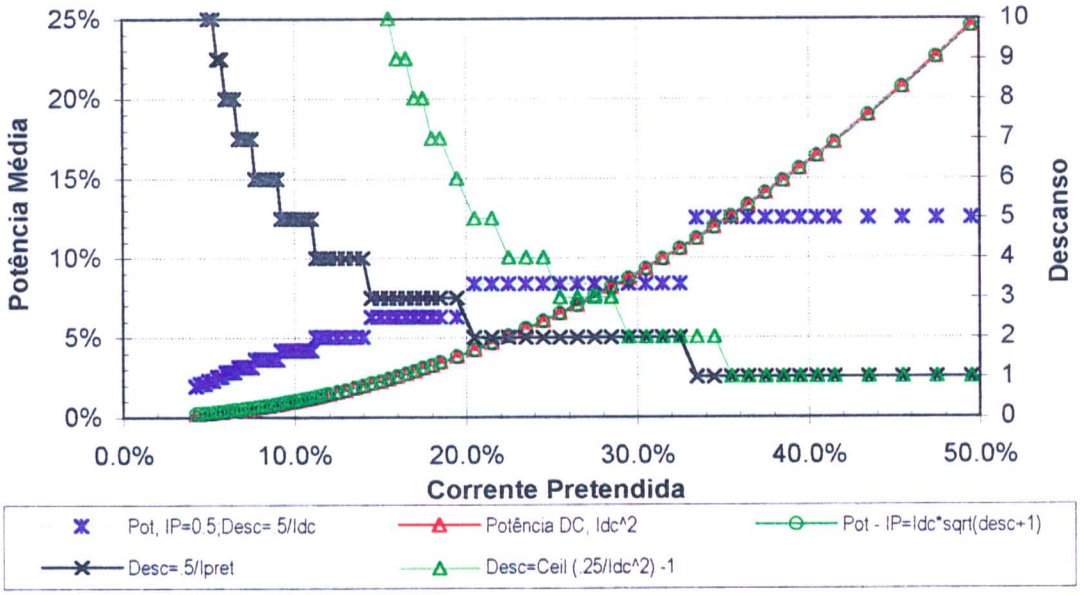


Figura 61: Comparativo do envio de potência em forma de pedestal (I_P) face ao envio da corrente de forma distribuída (I_{DC})

O traçado da corrente real calculada através da corrente média (*i*), junto da corrente real calculada de acordo com o método apresentado nesta secção (*Ip*), face à corrente pretendida pelo controlador, vem na Figura 62, onde se pode confirmar que a corrente permanece sempre acima de $50\%I_N$, tal como necessário.

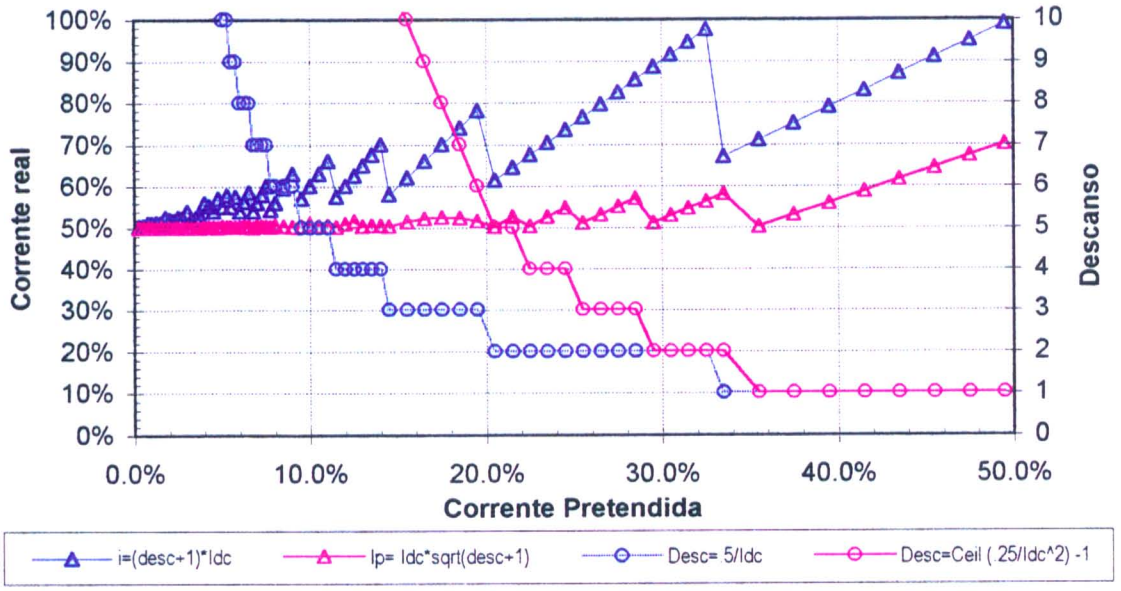


Figura 62: Corrente reais obtidas pelos dois métodos de envio de potência sob a forma de pedestal

8.4.3.1 Simulação

Esta forma de pensar foi testada com o seguinte trecho de simulação:

```
descanso=floor((ITransfo/2)^2/idevia^2+1)-1;
i=idevia*sqrt(descanso+1);
```

A versão 3 introduz um novo conceito para o controlador impreciso, que passa a basear-se no valor absoluto da corrente percentual do transformador para impedir a sobreelevação inicial e assim formar um sistema em malha fechada com características de amortecimento crítico.

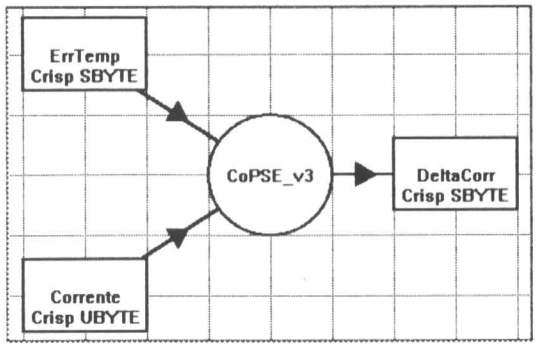


Figura 63: Diagrama de E/S do controlador v.3
(retirado do SieFuzzy)

As regras que regem o sistema são agora as mostradas pela Figura 64.

	Name	Op	ErrTemp	Corrente	Weight	DeltaCorr
1	Rule1	AP	Z	<>HIGH	100%	Z
2	Rule2		P		100%	P
3	Rule3	AP	N	<>HIGH	100%	N
4	Rule4	AP	N	HIGH	100%	VI
5						

Figura 64:Novas regras de controlo impreciso
(superfície de controlo representada na Figura 66)
Operador AP representa o **Produto Algébrico**

Para melhorar a performance do controlador optou-se por passar a utilizar o operador de conjunção AP, *Algebraic Product*, o operador produto algébrico. A fuzzificação do erro de temperatura passou a utilizar um parâmetro α=2, para melhorar a precisão do controlador de 1°C para 0.5°C, ficando no entanto mais sujeito a eventuais problemas de ruído de medida que possam surgir. Para obrigar a um seguimento da referência de elevada qualidade, a região imprecisa Z (Figura 65) da variável ErrTemp foi ainda mais estreitada, para que uma gama de valores ainda mais pequena seja considerada como sendo um erro aceitável. Tal é viável devido à precisão na actuação sobre o sistema conseguida pelo novo algoritmo de cálculo do descanso e respectiva corrente para o envio de potência.

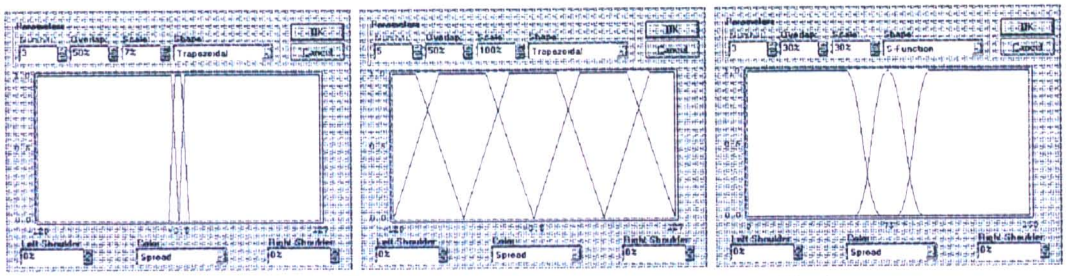


Figura 65: Formas das regiões imprecisas para as variáveis
ErrTemp, DeltaCorr e Corrente,
respectivamente da direita para a esquerda

As regiões imprecisas da variável Corrente têm a forma de S para simbolizar o tipo de conhecimento de que se tem: existe certeza na zona de transição mas mantendo uma razoável sobreposição nas zonas de menor nível de veracidade, conseguindo-se assim a imprecisão nas zonas próximas da transição.

O controlador final, que se tem vindo a descrever é caracterizado pela superfície de controlo da Figura 66.

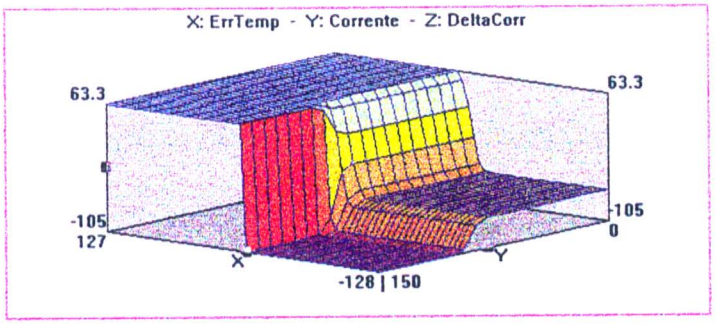
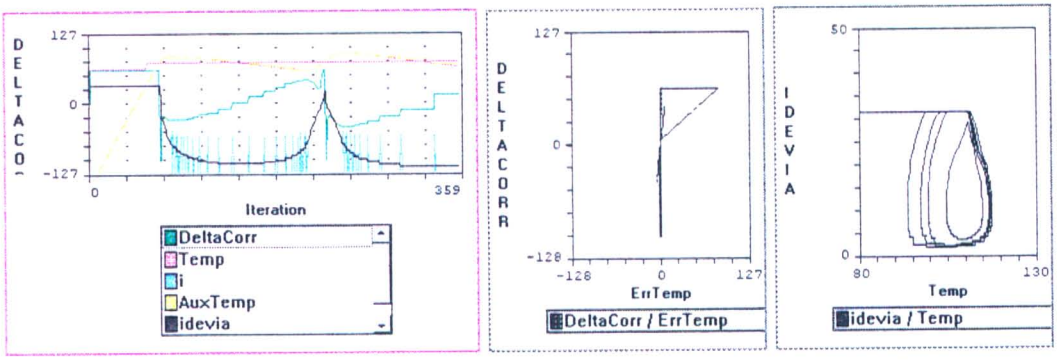


Figura 66: Superfície de controlo impreciso V3. Grelha de 50x20.

A existência de uma grande inclinação na superfície de comando não é indesejável, pelo contrário, é resultado directo da regra n.º 4 (Figura 64), que diz para descer muito o valor da corrente caso o seu valor seja elevado e caso a temperatura esteja demasiado alta.

Para ter um factor de segurança na aproximação aos 110°C, a referência é fixada inicialmente em 105°C e só depois é passada para o valor definitivo de 110°C. Várias tentativas demonstram que a mudança na referência é apropriada entre os 85 e os 95°C.



(a) (b) (c)

Figura 67: Simulação do novo controlador sem antecipação
31/7 - 1A, Ruído=0,Antevisão 0,
Evolução: Não, Aos 85°C: RefTemp 105->110
(a) e (b) após 360 Iterações (3 horas de processo)
(c) Após 1000 iterações - estabiliza a figura

A Figura 67 prova que os melhoramentos introduzidos no sistema beneficiam de facto o traçado da temperatura mas não se consegue ainda que a temperatura estabilize num certo valor final próximo da referência.

O grande problema dos controladores anteriores é que não existe qualquer efeito antecipativo. Uma hipotética solução seria a utilização de mais uma entrada para o sistema, uma nova variável imprecisa, tal como o erro de temperatura da iteração anterior. Se essa variável extra tivesse apenas 20 amostras, então a tabela de pesquisa passaria a ocupar, não os actuais 1 000 bytes, mas sim uns incomportáveis 20 000 de EPROM.

Uma das vantagens dos controladores do tipo impreciso é que existem vários "locais" onde a inteligência pode ser inserida. Através do conhecimento que se tem ganho do processo pode confirmar-se que sistema é integrativo e lento. Dir-se-ia que o sistema possui uma certa inércia, que significa que se o sistema estiva a subir de temperatura, possivelmente ele continuará a subir de temperatura, etc.. Tal consideração é também verdadeira face à saída do controlador, que, sendo em velocidade, faz com que sistema varie lentamente o seu ponto de funcionamento. Atendendo a esta "inércia" do sistema, é possível considerar que a última diferença de temperatura de temperatura se irá manter para a futura medida .

Para obter a desejada actuação antecipativa considera-se uma temperatura auxiliar (AuxTemp) que representa a temperatura actual afectada por um valor baseado na ultima diferença de temperatura. Este conceito de antecipação pode ser estendido multiplicando a última diferença de temperatura por um coeficiente (ANTECIPA) que torna o comportamento do sistema, por comparação aos controladores PID convencionais, mais "derivativo" para valores maiores da variável ANTECIPA.

A simulação passa a conter a seguinte linha de código:

```
AuxTemp=Temp-(TempAnt-Temp)*ANTECIPA
```

Toda a lógica de controlo, que dantes era aplicada à temperatura, passa agora a ser aplicada à temperatura auxiliar. A mudança da referência de 105 para 110°C continua, no entanto, a ser feita através do valor real da temperatura, várias simulações apontam que o valor de 95°C é o valor apropriado para a referida mudança.

Valores maiores de "antecipação" levam a convergências mais perfeitas e à eliminação completa da sobrelevação do sistema quando o processo atinge pela primeira vez a referência, tal como na Figura 68. Esta situação pode ser alargada para todos os transformadores, tanto os mais lentos como os mais rápidos.

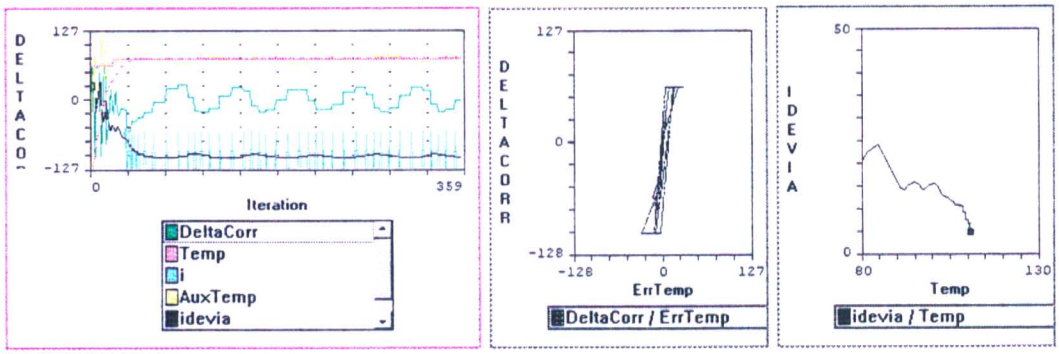


Figura 68:Simulação com elevado ganho antecipativo, sem ruído
31/7 - 1A, Rini*5, Ruído=0,Antevisão 10,
Evolução: Não, RefTemp 105->110

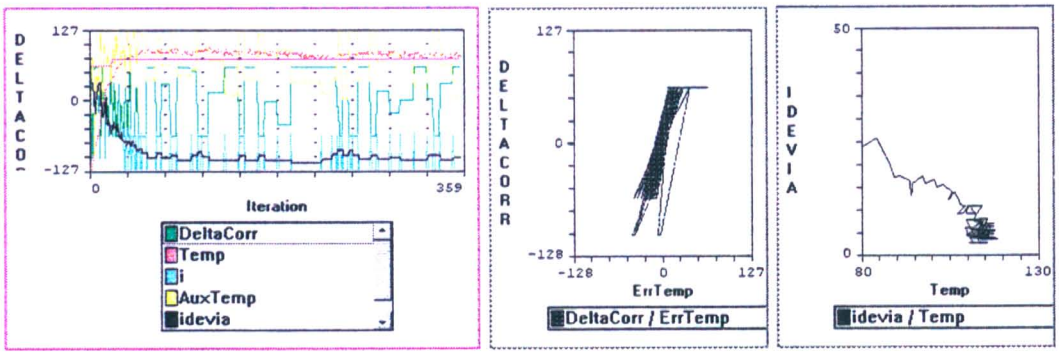


Figura 69 :Simulação com elevado ganho antecipativo, com ruído
31/7 - 1A, Rini*5, Ruído[-2,2],Antevisão 10,
Evolução: Não, RefTemp 105->110

A Figura 69 revela, no entanto, que um controlo tão antecipativo já fica sujeito ao efeito do ruído que possa existir no sistema, pelo que se optou por um valor final de antecipação de 5. O valor do ruído presente na Figura 69 é possivelmente maior que o do sistema real, mas demonstra uma potencial fragilidade do controlo, que se pretende robusto. Este comportamento é análogo ao existente quando se faz variar a componente derivativa dos controladores PID convencionais, *i. e.* uma importante componente derivativa faz o

sistema aproximar-se da instabilidade. Refira-se, no entanto, que, ao contrário do PID convencional, a acção de controlo não depende explicitamente do tempo, ou seja, é provável que a última variação de temperatura do sistema se mantenha, independentemente da ultima medida ter sido feita 30 segundos atrás ou 10 minutos atrás, tendo em conta que a próxima medida será também feita dentro de um tempo na mesma ordem de grandeza.

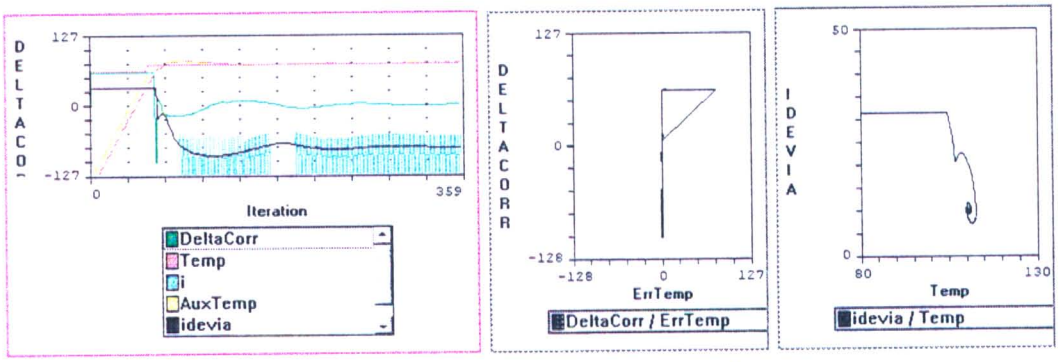


Figura 70 Simulação com antecipa=5
31/7 - 1A, Rini*1, Ruído=0,Antevisão 5, Evolução: Não,
Aos 95:RefTemp 105->110

A Figura 70 apresenta o sistema final, com antecipação de 5. Existe de facto alguma sobrelevação, mas a alteração dos parâmetros do sistema demonstram que a sobrelevação é sempre muito bem controlada, nunca é muito maior que aquela apresentada na Figura 70, sendo muitas vezes muito menor, especialmente no caso do processo ser mais lento.

Para um transformador hipotético, muito rápido, com um R_{eqv} 10 vezes maior que o real, a sobrelevação é sempre muito bem controlada, tal como prova a Figura 71.

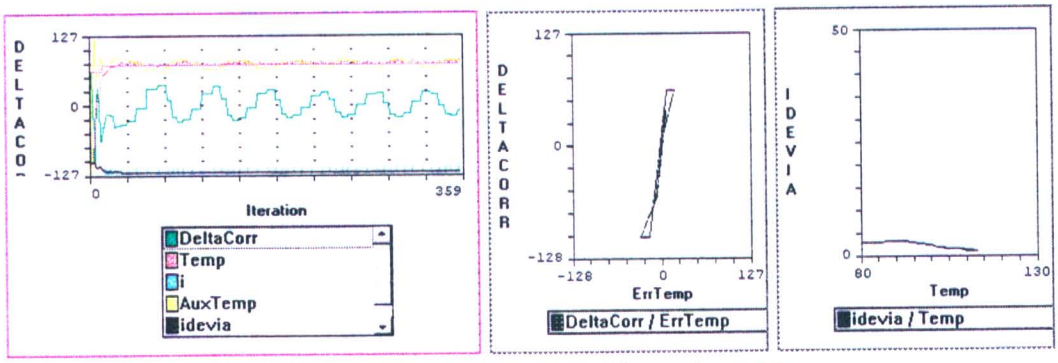


Figura 71: Transformador hipotético muito rápido
Antevisão: *5, Temperaturas: 30-130°C Correntes:0-50A
6/5 - 3A, Rini*=10, Ruído=0,Antevisão 5, Evolução: Não,
Aos 100:RefTemp 105->110

A robustez deste controlador pode ser provada pela Figura 72, onde se mostra a simulação de um transformador onde o R_{eqv} foi multiplicado por 5 logo de início e onde se fez os parâmetros evoluir, *i.e.*, todos os 15 pontos da simulação as perdas do sistema são

reduzidas (o sistema passa a arrefecer mais devagar) e a resistência equivalente do sistema sobe (equivalente à massa do sistema ser menor – por perda de água – o sistema passa a aquecer mais depressa). O trecho de simulação que executa essa operação é transcrito de seguida:

```
if (iteration%15)==0 && iteration< 300 then
    Perdas=Perdas*1.00002;
    r=r*1.15;
end
```

No final do processo, o valor de R_{eqv} é $5*1.15^{15}=81.8$ vezes superior ao valor real e o parâmetro perdas está muito perto da unidade, repare-se como os impulsos de corrente na parte final do processo se encontram muito afastados (muito mais afastados que na realidade) e mesmo assim o controlador consegue manter a temperatura perto da referência, mesmo com um ruído exagerado no sistema.

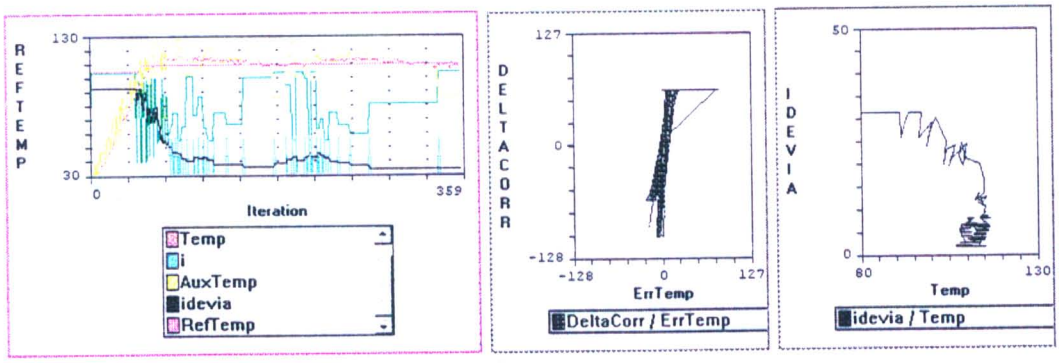


Figura 72: Simulação de transformador evolutivo no tempo
Curva 1a de 31/7, Rini*=5, ruído [-2,+2], Evolução!
Antevisão *5, Temperaturas: 30-130°C, Correntes:0-50A

Perante estes bons resultados em simulação, este sistema parece muito prometedor.

8.4.3.2 Resultados

Murphy, o pessimista, acabou por se revelar realista pois houve mais problemas no registo automático de dados pelo que todos os processos descritos nesta secção são registados manualmente.

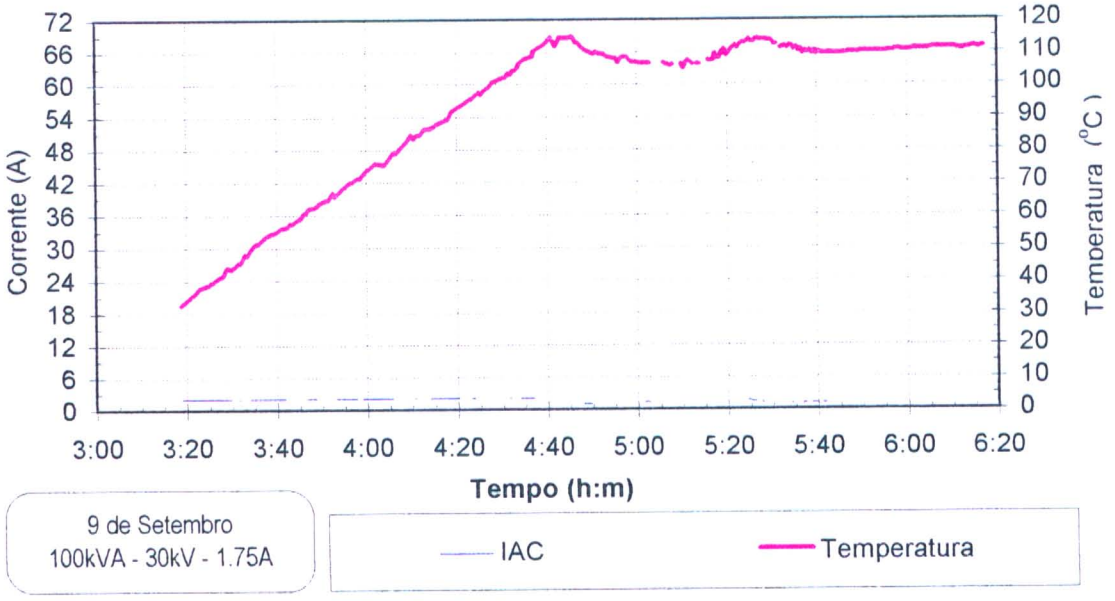


Figura 73:Resultados experimentais para o controlador v.3

O traçado da Figura 73 revela um comportamento bastante melhor do que o conseguido com controladores anteriores, um pouco devido ao facto do transformador ser mais fácil de controlar e necessitar de menos períodos de descanso.

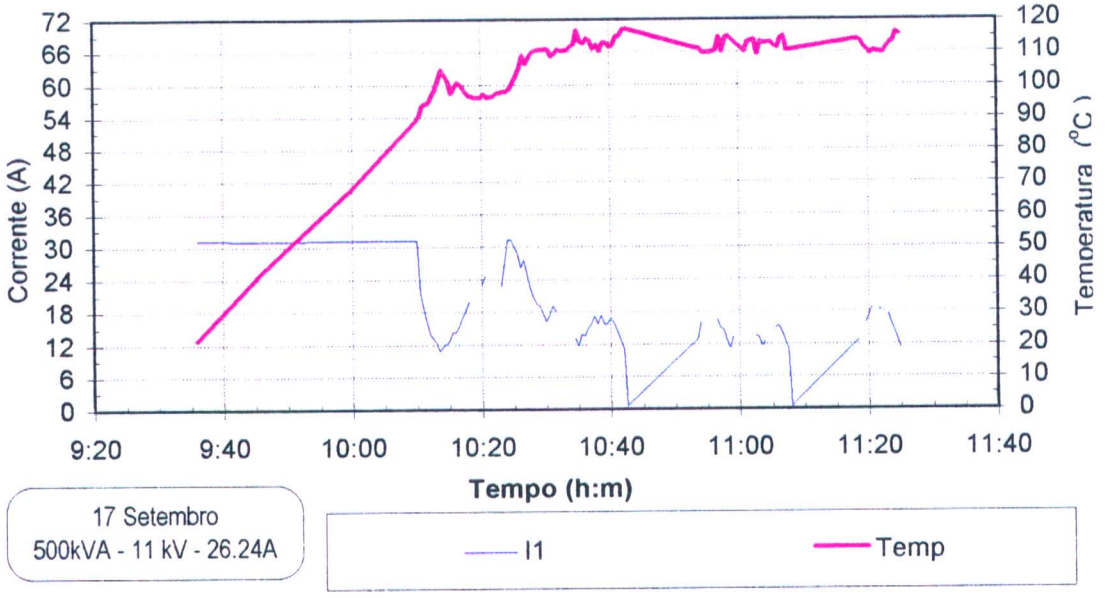


Figura 74: Resultados experimentais para o controlador v.3

A Figura 74 mostra como, de facto, o novo controlo é eficaz, também para transformadores de tamanho médio. Repare-se como são nítidos os tempos de descanso que houve necessidade de aplicar para que o transformador não subisse de temperatura. Mesmo neste difícil controlo, a referência é seguida muito mais de perto que no controlo convencional.

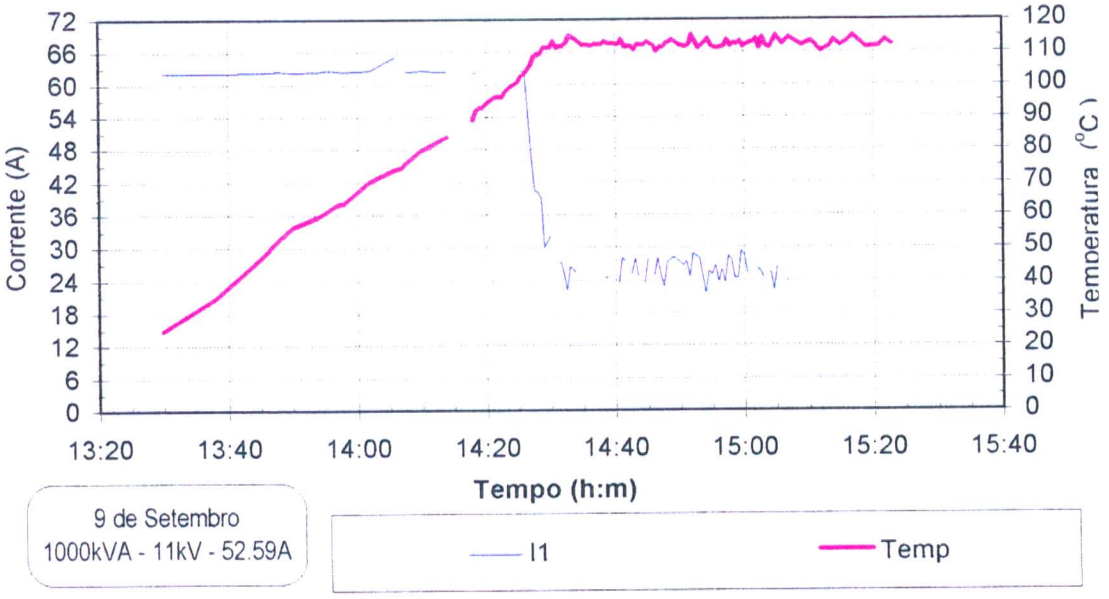


Figura 75: Resultados experimentais para o controlador v.3
(transformador 1MVA)

Também para a Figura 75 a análise dos resultados é boa, o sistema não tem qualquer sobrelevação, a temperatura tem uma aproximação perfeita aos 110°C e o seguimento é também muito melhorado face ao controlo convencional. Uma vez mais, Murphy, o realista (!), teve razão e ainda outra avaria na fonte onde decorria este ensaio impediu que o processo decorresse normalmente.

Como referido, não existem dados comparativos acerca destes ensaios porque os processos foram acompanhados manualmente.

8.5 Estabilidade

Um tema que foi pouco abordado neste trabalho foi a estabilidade do controlador. Tal como referido claramente por **Mamdani 1993**, a lógica imprecisa não contribui para o culto do estudo analítico que tem varrido a teoria dos sistemas. De facto, a lógica imprecisa padece do facto de não permitir a utilização de critérios matemáticos de prova inequívoca de estabilidade. Adicionalmente, assegura que a indústria nunca exigiu a estabilidade matemática como sendo necessária e suficiente para a aceitação de um sistema de controlo bem desenhado. Continua, afirmando que poucos sistemas instalados na indústria cumprem esse requisito "requisito", que teria sido criado nos meios académicos. Assegura ainda que, para a indústria, a estabilidade matemática nunca foi um teste suficiente para a aceitação de um controlador, *i.e.*, nenhum controlador industrial passa sem validação experimental, esse sim, é o teste definitivo.

A estabilidade do controlador foi testada, mesmo para situações exageradas face à realidade: ruído exagerado, transformadores muito lentos, muito rápidos, antecipação exagerada, etc.. e o comportamento foi sempre apropriado, tal como pode ser verificado facilmente pela análise dos traçados dos planos `idevia/temperatura`.

A validação experimental demonstra também que o controlador não só é inerentemente estável como revela características muito boas, pelo que se considera que esta questão está assegurada. Para que a confiança seja, no entanto, completa, admite-se mais alguns testes antes da implantação industrial final.

8.6 Análise de Resultados

A versão executável do controlador, versão final, ocupa 6 a 8KB de EPROM, incluindo as rotinas de fuzzificação, desfuzzificação e interface com o resto do programa. Desta forma foi possível desenvolver o novo controlador sem qualquer alteração de hardware.

Todo o código gerado pelo SieFuzzy, englobando a tabela de controlo, rotina de interpolação e rotina de pesquisa da tabela resulta num código objecto de 5.5 KB, que, compilado, produz cerca de 1.2KB ocupados na EPROM, utilizando 7 bytes de RAM apenas. Devido às características industriais da plataforma de teste, não foi possível medir tempos de execução. Para uma comparação do desempenho do código gerado pelo SieFuzzy, consultar o Anexo A.

Os dados adquiridos nos ensaios possíveis revelam que este novo controlador cumpre os requisitos que tinham sido postos:

- Aproximação inicial aos 110°C
  - Rápida aproximação inicial, mais rápida que no controlador convencional
  - Para alguns transformadores, o cotovelo de temperatura é perfeito
  - Nos piores casos existe uma sobrelevação mínima
- Convergência em torno do valor final
  - Qualquer que seja a forma de aproximação inicial, a temperatura aproxima-se e acaba por estabilizar num valor próximo do pretendido
  - A estabilidade da forma da temperatura obtida pode ser considerada óptima se se considerar que as oscilações no traçado da temperatura perto do final do processo podem ser atribuídas a erros de medida que se podem considerar na ordem dos ±5%
- Robustez do controlo

- O controlo é extremamente robusto e a variação dos parâmetros do processo de secagem não é sequer visível, em malha fechada
- Não existe qualquer parâmetro extra que os operadores necessitem de configurar

- Secagem de transformadores muito diferentes
  - A secagem dos transformadores de todos os tamanhos é assegurada
  - Não é necessário mudar parâmetro algum entre transformadores

Perante estes resultados, confirma-se que a lógica imprecisa fornece um método apropriado para o controlo de processos não lineares e de características difíceis, como o caso apresentado. Os ensaios realizados dizem respeito a todos os tamanhos de transformadores mas admite-se que são ainda necessários mais ensaios para assegurar a performance óptima do sistema de controlo.

Um dos requisitos iniciais era que o controlador devia ser desenvolvido com um número mínimo de iterações, o que foi plenamente conseguido: apenas 3 soluções de controlo foram testadas.

8.7 Resumo

Neste capítulo descreveu-se o desenvolvimento rápido do projecto de um controlador por lógica imprecisa, utilizando a ferramenta SieFuzzy. O trabalho com esta ferramenta revelou-se essencial, na medida em que engloba simulação e controlador, permitindo fazer ajustes nas regiões imprecisas no decurso da simulação, por forma a ser possível concluir rapidamente os resultados das alterações efectuadas. O principal problema do SieFuzzy acabou por ser os erros intermitentes, que levam à perda do trabalho realizado porque este fica gravado de forma inconsistente. Também a incapacidade de a simulação utilizar a tabela de pesquisa que foi realmente utilizada na implementação em linguagem C é um grave entrave a implementações embebidas, onde se pretende minimizar o espaço de memória ocupado pela tabela. O SieFuzzy apresenta a vantagem de gerar código C para um compilador destinado à família 8051, o que representa preocupações a nível da eficiência do código implementado. Naturalmente, é de lamentar que uma ferramenta tão poderosa não possa criar também código *assembly* directo para a família 8051, uma família extremamente divulgada de controladores, ao contrário da ferramenta descrita no anexo A.

Jager (1995) refere que apesar da lógica imprecisa ser extremamente configurável, o desenvolvimento de controladores acaba por ser rápido porque muitos parâmetros nunca chegam a ser alterados. Este trabalho é exemplo disso mesmo. Mandani (1993) refere que grande parte dos controladores industriais utiliza apenas 7 a 15 regras. Este

controlador utiliza apenas 4 regras, e a validação experimental decorreu durante apenas 4 semanas (não contíguas).

Apresentaram-se várias soluções para os problemas específicos da aplicação, que acabaram por levar a uma implementação com características globais muito boas.

A implementação final ocupa apenas 6 a 8KB de memória, compilado com o IAR C, versão 3.0, o que revela como o controlador é extremamente compacto, o que permitiu aliás que não fosse necessário introduzir qualquer alteração ao hardware já em serviço industrial.

A análise de resultados efectuada revela que o controlador impreciso tem um desempenho bastante melhor que o controlador convencional anteriormente instalado.

9. Conclusões

9.1 Trabalho Desenvolvido

Neste trabalho procurou apresentar-se uma abordagem introdutória e orientadora dos sistemas embebidos e problemas associados. A função primordial deste tipo de sistemas é o controlo, que levanta restrições do tipo tempo real. Caracterizou-se o controlo distribuído como sendo uma área onde existe, na actualidade, pesquisa muito promissora e intensa.

Salientou-se o relacionamento entre a engenharia mecatrónica e os sistemas embebidos para referir que o projecto de sistemas complexos deve ser abordado como um todo, de tal forma que os problemas tenham a solução ao nível mais conveniente. De acordo com este princípio, foi apresentado um modo de acção para o projecto de sistemas embebidos, a ser aplicados com qualquer método de análise estruturada de desenvolvimento.

Abordou-se, de seguida, o software dos sistemas embebidos. Referiram-se as condicionantes do desenvolvimento sob o signo do tempo real e que este tipo de desenvolvimento necessita de funcionalidades a nível do sistema operativo, que foram caracterizadas com especial incidência nos pormenores específicos aos sistemas embebidos.

O hardware dos sistemas embebidos foi também estudado. Foram abordadas as arquitecturas dos processadores e as várias funcionalidades que permitem facilitar o controlo. Aspectos tão importantes como o suporte à multi-tarefa e a provisão para uma rápida eliminação de erros foram também estudados. A problemática do desenvolvimento de hardware para sistemas embebidos foi apresentada na perspectiva de que tal desenvolvimento deve ser encarado com cuidado pois pode revestir-se de problemas de complexos.

Seguiu-se a apresentação do CoPSE, o sistema embebido para controlo industrial objecto da componente experimental descrita.

Dentro das peculiares características do controlo deste sistema, foi procurada uma solução no âmbito da inteligência artificial. Para tal, foi desenvolvido uma simulação do comportamento macroscópico do processo, baseada em conhecimentos heurísticos do seu comportamento. A simulação desenvolvida foi extremamente importante para permitir o desenvolvimento do controlador durante o funcionamento industrial.

A solução de controlo encontrada situa-se na área da lógica imprecisa, um ramo da inteligência artificial. Tal escolha foi justificada e apresentou-se uma breve introdução às técnicas utilizadas. Recorreu-se uma ferramenta de ajuda à engenharia da programação para o desenho e afinação rápida do controlador, simulado o processo em malha fechada. Foi possível enquadrar o software desenvolvido nas condicionantes da anterior plataforma de hardware, pelo que não foi necessária qualquer modificação física ao sistema.

Os resultados do novo controlador foram muito satisfatórios pois o controlo da temperatura foi bastante melhorado: a aproximação à referência final é conseguida rapidamente, a temperatura é mantida sempre longe do limite de perigo para o sistema e o seguimento da referência é, agora, muito melhor do que o conseguido com o controlo convencional.

9.2 Trabalho Futuro

A aplicação industrial definitiva carece ainda de mais ensaios para testar se comprovar o efectivo bom funcionamento do controlador.

O melhoramento do controlador de temperatura pode ser conseguido a nível de:

- Aproximação inicial à referência e regime dinâmico
  - Seria interessante conhecer os resultados obtidos pelo aumento da componente antecipativa, por forma a conhecer exactamente qual o melhor valor para este parâmetro, mantendo todas as considerações de segurança
  - Seria também curioso estudar se a capacidade de antecipação real do sistema seria melhorada ou não com a introdução da antecipação a dois níveis, utilizando a primeira e segunda derivada do sistema
  - Seria relevante conhecer, na prática, se existe a necessidade da medida de segurança de alterar a referência durante a aproximação inicial
  - Existiriam possivelmente benefícios em fazer a afinação do controlador *on-line*, isto é, baseado em dados reais[1]
- Precisão em regime estacionário e regime final
  - Deveria seguir-se um estudo profundo do ruído de medida e suas características
  - Deveria estudar-se a influência do valor absoluto da corrente nos erros de medida
  - O ajuste do controlador *on-line* poderia também ser benéfica

[1] Se fosse possível a utilização de um cordão umbilical, poder-se-ia ajustar as formas das regiões imprecisas perante a chegada dos dados reais. Seria também possível ajustar os processos de fuzzificação e de desfuzzificação durante o próprio decurso do processo. A afinação do controlador seria, então, baseada em dados da aplicação real e não da simulação

- O ajuste do controlador *on-line* poderia também ser benéfica
- Seria ainda interessante o estudo do benefício trazido pela utilização de outra ferramenta CASE, que produzisse directamente código *assembly* para o processador. Tal ferramenta possivelmente introduziria grandes benefícios a nível de velocidade de execução e tamanho do código compilado. Será então possível optimizar outras secções do código, permitindo assim que o conjunto fosse mais rápido, o que poderia significar uma maior estabilidade de outros sistemas, tal como a malha da corrente, o que afectará por sua vez a temperatura.

Adicionalmente, seria interessante a aplicação de outros métodos de controlo avançado para que fosse possível comparar o desempenho entre várias técnicas.

Bibliografia

Åström 1971 - "*System Identification - A Survey*"
Karl. J. Åström & P. Eykhoff
Automatica, Vol 7, pp 123-162
Grã-Bretanha, 1971

Åström *et al* 1990 - "*Computer Controlled Systems- Theory and Design, 2nd Ed.*"
Karl J. Åström e Bjorn Wittenmark
Prentice Hall, 1990

Bayko 1996 - "*Great Microprocessor Of The Past & Present*"
John Bayko
bayko@cs.uregina.ca
www.ee.umd.edu/~wwang/Great.CPUs.html

Bennet 1994 - "*Real Time Computer Control - An introduction - 2nd Ed.*"
Stuart Bennet
Prentice Hall International Series in Systems and Control Engeneering, 1994

Blakeslee 1993 - "*Real Time Debugging Techniques*"
Thomas Blakeslee & Jan Liband
Embedded Systems Programming, Volume 8, n.º 4 1993

Booch 1996 - "*Unified Method for Object Oriented Development - Version 0.8*"
Grady Booch e James Rumbaugh
Rational Software Corporaion

Bosacchi *et al* 1993 - "*Fuzzy Logic & the Intelligent Highway Sistem*"
Bruno Bosacchi & Ichiro Masaki
in: **Marks 1994**, pp 13 a 18

Brown 1994 - "*Embedded Systems Programming In C and Assembly*"
John Forrest Brown
Van Nostrand Reinhold" Computer Library - 1994

Bühler 1986 - "*Reglage par Mode Glissement*"
Hansruedi Bühler
Presses Polytehniques de Romandes
Losanne, 1986

Bursky 1995a - "*Improved Array Efficiency lets FPGAs Challenge Gate Arrays*"
Dave Bursky
Electronic Design - 5 Setembro 1995, pp 91-97

Bursky 1995b - "*486 CPU Packs Peripherals for Real Time System Needs*"
Dave Bursky
Electronic Design - 5 Setembro 1995, pp 167-168

Bursky 1996a - "*Embedded Controller Architectures Suit All Needs*"
Dave Bursky
Electronic Design - 8 Janeiro de 1996, pp 53-64

Bursky 1996b - *"Efficient RAM-Based FPGAs Ease System Design"*
Dave Bursky
Electronic Design - 22 Janeiro de 1996, pp 53-62

Byrd *et al* 1995 - *"Multithreaded Processor Architectures"*
Gergory T. Byrd e Mark A. Holliday
IEEE Spectrum, Agosto 1995, pp 38-46

Coleman *et al* 1994 - *"A comparison of Robustness: Fuzzy Logic, PID & Sliding Mode Control"*
Charles Coleman & Datta Godbole
retirado dos arquivos on-line da NASA

Comerford 1994 - *"Mecha...what?"*
Richard Comerford
IEEE Spectrum, Agosto 1994, pp.46-49

Cooper *et al* 1994 - *"A frequency Response Method for Multi-Sensor High Speed Navigation Systems"*
Simon Cooper e Hugh Durrant-Whyte
Documentação do *Course of Industrial Mobile Robotics, Theory and Applications*
COMETT, Dezembro 94

Cox 1994 - *"The Fuzzy logic Handbook"*
Earl Cox
Academic Press, Inc, 1994

DeMarco 1979 - *"Structured Analysis And System Specification"*
T. DeMarco
Prentice Hall, 1979

Dister *et al* 1993 - *"Using Simulink to Compare Fuzzy Logic and PI Motor Speed Control"*
Carl Dister e Semyon Royak
Technical Paper - Industrial Controls Division - Reliance Electric Reasearch Center

Dorf *et al* 1993 - *"The Electrical Engineering Handbook"*
Richard C. Dorf, Martin Bolton, John Oldfield, etc.
CRC Press, 1993

Freedman 1995 - *"Étude Sur Des Techniques, Méthodes Et Outils De Développement Logiciel De Systèmes Temps Réel"*
Paul Freedman, Jean Michel Goutal, Yves Leborgne e Daniel Gaudreau
CRIM, Centre de Recherche Informatique de Motréal, 1995

Fryxell 1990 - *"An Integrated Approach to the Design and Analysis of Navigation, Guidance and Control Systems for Autonomous Underwater Vehicles"*
Daniel Fryxell, Paulo Oliveira, António Pascoal e Carlos Silvestre
Documentação do *Course of Industrial Mobile Robotics, Theory and Applications*

Ganssle 1990a - *"The ICE Blues"*
Jack G. Ganssle
Embedded Systems Programming, Novembro de 1990

Ganssle 1990b - *Microcontroller C Compilers*
Jack G. Ganssle
Electronic Engineering Times, Novembro de 1990

Ganssle 1990c - *"Selecting a CPU"*
Jack G. Ganssle
Embedded Systems Programming, Dezembro de 1990

Ganssle 1993 - *"Built-in Debuggers"*
Jack G. Ganssle
Embedded Systems Programming, Setembro de 1993

Ganssle 1994a - *An OS in a Can*
Jack G. Ganssle
Embedded Systems Programming, Janeiro de 1994

Ganssle 1994b - *"Troublesooting"*
Jack G. Ganssle
EDN, Outubro de 1994

Ganssle 1995a - *"Small is Beautiful"*
Jack G. Ganssle
EDN, Maio de 1995

Ganssle 1995b - *"Interrupt Predictability"*
Jack G. Ganssle
Embedded Systems Programming, May 1995

Ganssle 1995c - *"Embedded Lingos"*
Jack G. Ganssle
Embedded Systems Prog ramming, Junho 1995

Gergeleit *et al* 1994 - *"DIRECT: Towards a Distributed Object Oriented Real Time Control System"*
M Gergeleit, J Kaiser, H Streich
Proceedings of: Workshop on Concurrent Object Based Systems, 26-29 Outubro, 1994

Gien 1996 - "RTOS move to the Open World"
Michel Gien
Electronic Design, 8 de Janeiro 1996, págs. 114 a 117

Greg_Bennet 1996 - *"DSP Chip's Challenge MCUS for control Systems"*
Gregg Bennet
Electronic Design, 8/1/1996, pág. 58

Hanselmann 1987 - *"Implmentation of Digital Controllers - a Survey"*
H. Hanselmann
Automatica - Vol 23, nº 1, pp 7-32

Harris 1993 - *"Intelligent Control - Aspects of Fuzzy Logic and Neural Networks"*
CJ Harris, CG Moore & M Brown
World Scientific Pub. Co.
Wold Scientific Series in Robotics and Automated Systems, 1993

Hersch 1996 - *Embedded Processor & Microcontroller Primer & FAQ*
Russ Hersch, 28 Janeiro de 1996
russ@silicom.co.il
ftp://www.cera2.com/micrxfaq.htm

Jager 1995 - "*Fuzzy Logic in Control*"
René Jager
Tese de Doutoramento da Universidade de Delft, Holanda, 1995

Jani *et al* 1996 - "*Selecting the Right Design Approach designing and developing critical R.T. embedded systems*"
Yashvant Jani e Gabi Leshem
Electronic Design - 22 Janeiro 1996 - pp 111-117

Joshi 1996 - "*Core Technologies lets designers roll custom solutions*"
Tapan Joshi
Electronic Design, 8 de Janeiro 1996, caixa das pág. 56 e 57

Jutten *et al* 1994 - "*Neural Networks: Introduction and Examples*"
Chistian Jutten, Pierre Puget e Laurent Hérault
Documentação da conferência "*Intelligence artificielle et automatique*" - Sessão 10 - Grenoble, 12 a 16 Setembro 1994

Kiendl 1994 - "*Fuzzy Reasoning: Industrial Experience and Research Experience*"
H. Kiendl, Ch. Frenck, J.J. Rüger
Documentação da conferência "*Intelligence artificielle et automatique*" - Sessão 10 - Grenoble, 12 a 16 Setembro 1994

Kopetz 1991 *et al.* - "*The Design of Real Time Systems from specification to implementation and verification*"
H. Kopetz, R. Zainlinger, G. Fohler, H. Kantz, P. Puschner, W. Schütz
IEEE, Software Engineerting Journal, Maio 1991

Kosko 1992 - "*Neural Networks and Fuzzy Systems*"
Bart Kosko
Prentice Hall International, Inc., 1992

Kurpis 1996 - "*Coping with compatibility headaches*"
Perter Kurpis
Electronic Design - 22 Janeiro 1996 - pp 118-120

Leitch 1992 - "*Artificial Intelligence in control: some myths, some fears, plenty of prospects*"
Roy Leitch
Computing & Control Engeneering Journal, Julho 1992
Também: Documentação da conferência "*Intelligence artificielle et automatique*" - Sessão 10 - Grenoble, 12 a 16 Setembro 1994

Levi *et al* 1990 - "*Real Time System Design*"
Shem-Tov Levi e Ashok Agrawala
McGraw Hill, 1990

Mamdani 1994 - "*Twenty Years of Fuzzy Control: Experiences Gained and Lessons Learnt*"
E. H. Mamdani
in: **Marks 1994**, pp 19 a 24

Mansur 1996 - "*Future Communications Processors will Fuse DSP and RISC*"
Dan Mansur
Electronic Design, 8 de Janeiro 1996, págs. 99 a 102

Marks 1994 - "*Fuzzy Logic Technology and Applications*"
Rober Marks II
IEEE Technoly Update Series

Milenkovic 1992 - "*Operating Systems - Concepts And Designs*" - *2nd Ed.*
Milan Milencovic
McGrawHill International Editions , 1992

Morse 1996 - "*The increasing importance of software*"
Jon Morse & Steve Hargrave
Electronic Design, 8 de Janeiro 1996, págs. 109 a 113

Passino 1995 - "*Intelligent Control for Autonomous Systems*"
Kevin Passino
IEEE Spectrum, Junho 95

Pereira 1994 - "*Mobile Robotics Overview*"
Fernando Lobo Pereira
Documentação do *Course of Industrial Mobile Robotics, Theory and Applications*
COMETT, Dezembro 94

Pfeiffer 1996 - "*SAB80C166 RISC for Embedded applications*"
Olaf 'Olu' Pfeiffer
pfeiffer@hitex.com
http://www.cera2.com/micro.htm

Quintas *et al* 1994 - *A New High Performance Controller for Drying and Oil Filling of Distribution Power Transformers*
A. R. Quintas, A. Carvalho, A. Araújo, A. Martins, J. Oliveira
IEE Conference on Power Electronic and Drives, 1994

Rumbaugh *et al* 1991 - "*Object Oriented Modelling and Design*"
James Rumbaugh, Michel Blaha, William Premerlani e William Lorensen
Prentice Hall, 1991

Schafer 1996 - "DSP *Fundamentals*"
Ronald W Schafer
IEEE Spectrum, Maio de 1996

Shear 1994a - "*Embedded Systems Developers Embrace The PC Architecture*"
David Shear
EDN, 3 de Março de 1994, pp 39-44

Shear 1994b - "*Desktop DOS Goes Undercover to Run Embedded Systems*"
David Shear
EDN, 4 Agosto 1994, pp 34-47

Smith et al 1987 - "*Introduction to Chemical Engineering Thermodynamic, 4th Ed.*"
J.M. Smith e H.C. Van Ness
McGraw Hill, 1987

Sousa *et al* 1994 - Veículo Auto-Guiado
A. Sousa, F. Pacheco, I. Praça e L. Ferreira
Seminário ACI, 93/94

Stankovic 1998 - "*Misconceptions About Real Time Computing*"
John A. Stankovic
IEEE Computer, Outubro 88, pp 10-19

Takagi 1991 - "*Applications Of Neural Networks And Fuzzy Logic To Consumer Products*"
Hideyuki Takagi
in: **Marks 1994**, pp 8 a 12

Tanenbaum 1992 - "*Modern Operating Systems*"
Andrew S. Tanenbaum
Prentice Hall International, Inc., 1992

Tindell 1993 - "*An Extendible Approach For Analysing Fixed Priority Hard Real Time Tasks*"
K. W. Tindell
Universidade de York, 1993

Tyle 1995a - "*A designer's Guide to Real Time Operating Systems*"
Sherrie Van Tyle
Electronic Design - 21 Agosto 1995, pp 75-79

Tyle 1995b - "*Embedded Systems Conference Tackles Software, Hardware & Debugging*"
Sherrie Van Tyle
Electronic Design - 5 Setembro 1995, pp 48-63

Ward 1985 *et al*. - "*Structured Development For Real Time Systems*"
P. Ward e S. Mellor
Prentice Hall, 1985

Yeralan 1995 - "*Programming and Interfacing 8051 Micorcontrollers*"
S. Yeralan e A. Ahluwalia
Addison Wesley, 1995

Índices

Índice Pormenorizado

Sumário iii

Palavras Chave iv

Lista de Abreviaturas v

Índice Genérico vi

Parte I - Sistemas Embebidos e Tempo Real 1

1. Introdução 2

1.1 Contexto 2

1.2 Motivação 4

1.3 Metodologia 5

2. Sistemas Embebidos 6

2.1 Introdução 6

2.2 Controlo Digital 7

2.3 Tempo Real 9
2.3.1 Classificação de Estímulos T. R. 11
2.3.1.1 Cíclicos 11
2.3.1.2 Aperiódicos 11
2.3.1.3 Interactivos 11
2.3.2 *Hard Real Time* e *Soft Real Time* 12

2.4 Controlo Distribuído 14
2.4.1 Sistemas Distribuídos 14
2.4.2 Sistemas Distribuídos em Tempo Real 15
2.4.3 Tolerância e Fiabilidade 16

2.5 Resumo 17

3. Engenharia embebida 18

3.1 Mecatrónica 18

3.2 Projecto 19
3.2.1 Equilíbrio Software/Hardware 22
3.2.2 Escolha da Plataforma de Processamento 23

3.3 Software 25
3.3.1 Linguagens de Programação 26
3.3.2 Desenvolvimento em T. R. 28
3.3.3 Utilização S. O. Tempo Real 28

3.4 Hardware 31
3.4.1 O mercado actual 31
3.4.2 Microcontrolador *vs.* Microprocessador 31
3.4.3 Arquitecturas Internas de Processamento 33
3.4.3.1 CISC's 33
3.4.3.2 RISC's 34
3.4.3.3 DSP's 36

3.4.3.4 PICs 37
3.4.3.5 Outras Arquitecturas 38
3.4.4 PCs Embebidos 39
3.4.4.1 ISA (8 e 16 bits) 40
3.4.4.2 PC/104 40
3.4.4.3 Compact PCI 41
3.4.5 Barramentos VME e VXI 41
3.4.6 Integrados Específicos para a Aplicação 42
3.4.6.1 Linguagens de Descrição de Hardware 42
3.4.6.2 Futuro Promissor 43
3.4.7 Funcionalidades e Poder de Cálculo 44
3.4.8 Necessidade/Disponibilidade do Poder de Cálculo 44
3.4.8.1 Pipelining 45
3.4.8.2 Memória 45
3.4.8.2.1 *Cache* vários níveis 45
3.4.8.2.2 Dinâmica 46
3.4.8.2.3 Estática 46
3.4.8.2.4 EDO 46
3.4.8.3 Temporização, Captura, Comparação, MLI 47
3.4.8.4 E/S, analógicas e digitais 48
3.4.8.5 Interrupções 48
3.4.8.6 Endereçamento e E/S 49
3.4.8.7 Comunicações integradas 50
3.4.8.8 Capacidades de Multi-Tarefa 50
3.4.8.9 Stack Pointer protegido por hardware 51
3.4.9 Ferramentas para desenvolvimento 51
3.5 Resumo *54*
Parte II - Experimentação e Controlo **55**
4. Apresentação do Caso Prático **56**
4.1 O processo *56*
4.2 Controlo Convencional *60*
4.3 Aquisição de dados - a envolvente fabril *65*
4.4 Objectivo do Trabalho Prático *69*
4.5 Recursos computacionais disponíveis *72*
4.6 Compressão do código já existente *73*
4.7 Resumo *74*
5. Métodos de Controlo Avançado **75**
5.1 Utilização de tabela de parâmetros PI(D) *75*
5.2 Aquisição dos parâmetros do sistema on-line *76*
5.3 Controlo por modo de deslizamento *78*
5.4 Filtros de Kalman *79*
5.5 Resumo *80*
6. Controlo Inteligente **81**
6.1 Introdução ao controlo inteligente *81*
6.2 Metodologias de controlo Inteligente *81*
6.3 Sistemas Periciais *82*

6.4 Redes Neuronais 83
6.5 Algoritmos Genéticos 85
6.6 Solução: Fuzzy Logic 86
6.7 Resumo 94
7. Simulação do CoPSE **95**
7.1 Identificação 95
7.2 Modelo 95
7.3 Ajustes 99
7.4 Resumo 102
8. Implementação **103**
8.1 Introdução à Lógica de Controlo 104
8.2 Regras de Inferência 110
8.3 Tabela vs. Matemática Imprecisa 113
8.4 Simulação, Programação e Resultados 115
8.4.1 Versão 1 119
8.4.1.1 Simulação 120
8.4.1.2 Resultados 121
8.4.2 Versão 2 124
8.4.2.1 Simulação 124
8.4.2.2 Resultados 125
8.4.3 Versão 3 127
8.4.3.1 Simulação 130
8.4.3.2 Resultados 135
8.5 Estabilidade 137
8.6 Análise de Resultados 138
8.7 Resumo 139
9. Conclusões **141**
9.1 Trabalho Desenvolvido 141
9.2 Trabalho Futuro 142
Bibliografia **144**
Índices **150**
Índice Pormenorizado 150
Índice de Figuras 153
Índice de Tabelas 155
Anexo A - Comparação de Desempenho em Controlo por Lógica Imprecisa **A1**

Índice de Figuras

Figura 1: Diagrama de blocos para o controlo por computador 8
Figura 2: Diagrama de blocos para o controlo hierárquico 9
Figura 3: Aspecto gráfico das penalizações por incumprimento do T.R. 12
Figura 4: Fluxograma do desenvolvimento global de sistema embebidos 22
Figura 5: Compromisso entre hardware e software excessivos 23
Figura 6: Custos relativos software/hardware (curvas indicativas) 25
Figura 7: Mercado dos S.E., em função do tamanho da palavra 31
Figura 8: Comparação de características entre μC e μP 32
Figura 9: Comparação entre arquitecturas de μs: RISC e CISC 35
Figura 10: Comparativo do poder de cálculo da arquitectura RISC 36
Figura 11: Ilustração gráfica da comparação e captura de um temporizador 47
Figura 12: Ilustração gráfica do funcionamento do modo de PWM 48
Figura 13: Sinusóide ideal amostrada a 50 Hz 63
Figura 14: Sinusóide fortemente afectada pelo 3º harmónico, amostrada a 50 Hz 63
Figura 15: Esquema de princípio da hierarquia de controladores envolvidos no CoPSE (o esquema real é bastante mais complexo) 64
Figura 16: Circuito de Potência 65
Figura 17: Efeito do Índice de Modulação em MLI 66
Figura 18: Diagrama dos vários blocos constituintes do CoPSE 69
Figura 19: Controlo covencional, comparação transformadores de 1MVA e 100kVA 70
Figura 20: Controlo convencional, transformador de potência média 71
Figura 21: Processo CoPSE interrompido e depois retomado 71
Figura 22: Memória do 80535: Modos de endereçamento 74
Figura 23: Ampliação nos tempos da Figura 19 88
Figura 25: Variável Preço, *Alto*, *Baixo*, (*Alto* **e** *Baixo*) 89
Figura 26: Variável Preço, região (*Alto* **e** *Baixo*), região *2xCusto*, sua conjunção 90
Figura 28: Exemplo de inferência por lógica imprecisa 91
Figura 29: Procura de objectivos no Excel 99
Figura 30: Simulação da secagem do transformador através de dois métodos 99
Figura 31: Temperatura do transformador n.º 2 Curvas comparativas das temperaturas estimadas para uma secagem dos transformadores 1 e 3 com a corrente 2 101
Figura 32: Temperatura do transformador n.º 3 Curvas comparativas das temperaturas estimadas para uma secagem dos transformadores 1 e 2 com a corrente 3 101
Figura 33: Tráfego de dados no controlador de temperatura 104
Figura 34: Estrutura do conjunto de regras imprecisas proposto para o novo controlador do CoPSE 107
Figura 35: Definição da variável imprecisa (*Fuzzy*) "ErrTemp" 107
Figura 36: Fuzzificação da variável "ErrTemp" 108
Figura 37: Definição de "ErrEn" 108
Figura 38:Fuzificação da variável "ErrEn" 109
Figura 39: Definição Imprecisa (*Fuzzy*) da variável de saída do controlador, "DeltaCorr" . 109
Figura 40: Actuação do controlador sobre a corrente (posterior à desfuzzificação) 110
Figura 41: Definição das regiões para cada variável imprecisa 111
Figura 42: Superfície de controlo impreciso, (50 amostras ErrTemp + 20 amostras ErrEn) 112
Figura 43:Teste rápido do SieFuzzy 113
Figura 44: Extractos do código da simulação do SieFuzzy 115
Figura 45: Resultados preliminares da simulação 116
Figura 46: Regras de controlo impreciso (na forma de texto) 117
Figura 47: Contagem do número de vezes que cada regra foi avaliada com antecedente não nulo, para a simulação da Figura 45 117

Figura 48:Teste do modo de funcionamento drástico 117
Figura 49:Contagem da actuação de cada regra na simulação da Figura 48 118
Figura 50: Plano Corrente Temperatura relativos à simulação da Figura 45 118
Figura 51: Planos DeltaCorr/DeltaTemp e DeltaCorr/ErrTemp (para ganhos do controlador de: $2*10^{-4}$ 2, $2*10^{-3}$ e $2*10^{-2}$) 119
Figura 52: Simulações para ganhos de, respectivamente: $2*10^{-4}$ 2, $2*10^{-3}$ e $2*10^{-2}$ 119
Figura 53: Simulação com os parâmetros utilizados no ensaio 1 G_{CTRL}=0.001, transformador 1MVA 120
Figura 54:Resultados experimentais do controlador Impreciso v.1 121
Figura 55: Resultados do controlador Impreciso v.1, comparando com um transformador igual no controlador antigo 123
Figura 56: Resultados do controlador Impreciso v1.0 com ganho 0.002 123
Figura 57: Simulação do controlador *Impreciso* v2 124
Figura 58: Trecho de actuação do controlador Impreciso 2.0 (ganho controlador=0.002) 125
Figura 59: Discrepâncias entre correntes e potências pretendida e obtidas (erro de truncatura) descanso=ceil(0.5/corrente_devia) 126
Figura 60: Ilustração do funcionamento em Liga/Desliga para T=Descanso+1=5 127
Figura 61: Comparativo do envio de potência em forma de pedestal (I_P) face ao envio da corrente de forma distribuída (I_{DC}) 129
Figura 62: Corrente reais obtidas pelos dois métodos de envio de potência sob a forma de pedestal 129
Figura 63: Diagrama de E/S do controlador v.3 (SieFuzzy) 130
Figura 64: Novas regras de controlo impreciso (superfície de controlo na Figura 66) 130
Figura 65: Formas das regiões imprecisas para as variáveis ErrTemp, DeltaCorr e Corrente, respectivamente da direita para a esquerda 131
Figura 66: Superfície de controlo impreciso V3. Grelha de 50x20 131
Figura 67: Simulação do novo controlador sem antecipação 132
Figura 68:Simulação com elevado ganho antecipativo, sem ruído 133
Figura 69 :Simulação com elevado ganho antecipativo, com ruído 133
Figura 70 Simulação com antecipa=5 134
Figura 71: Transformador hipotético muito rápido (Antevisão: 5) 134
Figura 72: Simulação de transformador evolutivo no tempo 135
Figura 73: Resultados experimentais para o controlador v.3 (100 kVA) 136
Figura 74: Resultados experimentais para o controlador v.3 (500 kVA) 136
Figura 75: Resultados experimentais para o controlador v.3 (1MVA) 137

Índice de Tabelas

Tabela 1: Formalização das restrições temporais de sistemas Tempo Real 13
Tabela 2: Tabela dos blocos de memória utilizados no CoPSE. Espaço (inicialmente) disponível para o controlador de temperatura 72
Tabela 3: Taxonomia dos métodos para trabalho em sistemas sem estimação de parâmetros (adaptada de Kosko, 1993, pág. 26) 82
Tabela 4: Produtos para consumo no Japão, provenientes de várias marcas comerciais, utilizando Redes Neuronais (adaptado de Takagi, 1991) 85
Tabela 5: Valores finais dos *Pesos* de cada amostra 98
Tabela 6: Quadro resumo das energias acumuladas 106
Tabela 7: Tabela de regras do controlador v1 111

Anexo A - Comparação de Desempenho em Controlo por Lógica Imprecisa

A Tabela A1 resume dados de testes acerca do desempenho comparativo de vários processadores em testes com lógica imprecisa. Os resultados são, respectivamente, tempo médio de execução/tempo máximo de execução, tamanho do código objecto gerado e RAM ocupada. Esta tabela é fornecida como publicidade da ferramenta CASE "FuzzyTECH", da Inform[1] e pode ser obtida a partir da *World Wide Web*, no endereço http://www.inform-ac.com.

Para o controlador do CoPSE, alojado num μC da família 8051, o SieFuzzy, gerando código em C, produz 5.5KB de código objecto, que resulta em 1.2KB de programa executável e utiliza 7 bytes de RAM apenas. Recorde-se que a tabela de controlo ocupa 1000 bytes, tem uma precisão de 8 bits, representando 4 regras de inferência (todas com igual prioridade), baseadas em duas entradas e uma saída. Não foi possível registar os tempos de execução.

A Tabela A1 pode servir para comparar o código gerado pela ferramenta fuzzyTECH com o código gerado pelo SieFuzzy, claramente distintos à partida. O fuzzyTECH gera código *assembly* enquanto que o SieFuzzy gera código C. O Siefuzzy ocupa muito espaço com a tabela de controlo em memória de programa e utiliza a manipulação de variáveis do C para conseguir ocupar 7 bytes apenas de memória de dados. A utilização de uma ferramenta de geração de código em directamente em *assembly*, tal como o fuzzyTECH, traria benefícios em termos de velocidade de execução e possivelmente a nível do tamanho do código executável, especialmente devido ao 80535 não ser um μC apropriado para executar código C e também devido a permitir optimização de pesquisa em tabelas (só) por *assembly*. A utilização do C, permite, no entanto, poupar alguns recursos devido às rotinas em *assembly* terem que ser totalmente auto-suficientes, quando o compilador de C disponibiliza rotinas que poderiam eventualmente ser utilizadas partilhadas com todo o programa.

Da análise da Tabela A1, é possível destacar claramente a plataforma 81C99, que é um controlador específico para lógica imprecisa, da Siemens, que apresenta no entanto, a precisão de apenas 8 bits. Este processador específico é bastante mais rápido que o poderoso 486SX a 33 MHz, que, no entanto é programado em C (e não em *assembly*).

[1] Tel: +49-2408-9456-80, Fax: +49-2408-9456-85

Refira-se a curiosa inclusão neste comparativo, de um A. P. com capacidades específicas de lógica imprecisa, também com desempenho assinalável. Outros fabricantes produzem também cartas específicas para lógica imprecisa, tal como a Omrom.

Uma importante ausência é a do integrado NLX220, da Adaptive Logic[1], um coprocessador de lógica imprecisa com E/S analógicas, amostradas internamente em 8 bits. Este processador recebeu um prémio de inovação da revista EDN, 13 de Abril de 1995 e seria interessante comparar o seu desempenho com o μC 81C99[2].

Será ainda possível utilizar a Tabela A1 para comparar o desempenho entre as várias plataformas de processamento, para um certo problema de controlo, não sendo porém, o código gerado para os diferentes testes comparável. Para comparar, p. ex. um μC RISC outro CISC, ambos de 16 bits, compare-se os dados do CISC 196 com o RISC 166, de onde ressalta a clara superioridade do RISC.

Tabela A1:Comparativo de desempenho em vários testes durante a execução de inferência por lógica imprecisa
* Regras com prioridades

μP ou μC	7 Regras 2E/1S (Teste 0)	20 Regras 2E/1S (Teste 1)	20 Regras* 2E/1S (Teste 2)	80 Regras* 3E/1S (Teste 3)	500 Regras* 8E/4S (Teste 4)
PC: fuzzyTECH Precompiler Edition, código C, resolução de 16 bits, dispositivo testado: 80486SX, 33 MHz, MS-C 8.00	0.05/0.07 ms 0.73 KB OBJ 39 Byte RAM	0.05/0.08 ms 0.82 KB OBJ 47 Byte RAM	0.05/0.09 ms 0.89 KB OBJ 47 Byte RAM	0.09/0.12 ms 1.27 KB OBJ 53 Byte RAM	0.35/0.45 ms 3.5 KB OBJ 185 Byte RAM
80C196: fuzzyTECH MCU-96 Edition, código assembly, resolução de 16 bits, dispositivo testado: 80C196KD, ROM interna, 20 MHz	0.19/0.23 ms 0.63 KB ROM 50 Byte RAM	0.24/0.33 ms 0.69 KB ROM 58 Byte RAM	0.26/0.38 ms 0.78 KB ROM 58 Byte RAM	0.44/0.64 ms 1.17 KB ROM 66 Byte RAM	1.87/2.45 ms 3.47 KB ROM 214 Byte RAM
80C166: fuzzyTECH MCU-166 Edition, código assembly, resolução de 16 bits, disp. Testado: 80C166, ROM interna, 20 MHz	0.05/0.06 ms 0.44 KB ROM 30 Byte RAM	0.06/0.08 ms 0.56 KB ROM 30 Byte RAM	0.06/0.09 ms 0.61 KB ROM 30 Byte RAM	0.10/0.14 ms 0.99 KB ROM 36 Byte RAM	0.43/0.56 ms 3.2 KB ROM 194 B RAM
TMS-320: fuzzyTECH MCU-320 Edition, código assembly, resolução de 16 bits, disp. testado: TMS-320C52, ext. ROM, 20 ns inst.	0.02/0.022 ms 0.85 KB ROM 54 Byte RAM	0.023/0.03 ms 0.99 KB ROM 60 Byte RAM	0.024/0.031 ms 1.06 KB ROM 62 Byte RAM	0.039/0.049 ms 1.8 KB ROM 70 Byte RAM	0.146/0.188 ms 5.7 KB ROM 218 B RAM
8051: fuzzyTECH MCU-51 Edition, código assembly, resolução de 8 bits, dispositivo testado: 80C51, ROM interna, 12 MHz	1.0/1.2 ms 0.45 KB ROM 21 Byte RAM	1.4/1.9 ms 0.54 KB ROM 25 Byte RAM	1.5/2.0 ms 0.58 KB ROM 25 Byte RAM	4.4/4.9 ms 1.0 KB ROM 29 Byte RAM	16.7/18.0 ms 3.0 KB ROM 103 B RAM
PIC16C5X: fuzzyTECH MP Edition, código assembly, resolução de 8 bits, dispositivo testado: PIC16C74, ROM interna, 20 MHz	0.4/0.4 ms 0.32 KB ROM 27 Byte RAM	0.7/0.8 ms 0.41 KB ROM 31 Byte RAM	0.9/1.0 ms 0.44 KB ROM 31 Byte RAM	2.5/2.7 ms 0.7 KB ROM 35 Byte RAM	10.3/11.4 ms 2.2 KB ROM 107 Byte RAM
Fuzzy Coprocessor: fuzzyTECH MCU-99 Edition, código binário, resolução de 8 bits, disp. Testado: 81C99, ext. ROM, 20 MHz	-/0.029 ms - -	-/0.031 ms - -	-/0.031 ms - -	-/0.037 ms - -	-/0.066 ms - -
Autómato Programável: fuzzyTECH IA Edition, fuzzy function block, resolução de 16 bits, disp. testado: Klockner-Moeller PS4/401	-/1.6 ms - -	-/1.6 ms - -	-/1.6 ms - -	-/1.6 ms - -	-/3.0 ms - -

[1] Info@adaptivelogic.com, http://www.adaptivelogic.com, Tel: +1-408-383-7200, Fax: +1-408-383-7200

[2] A folha de dados deste coprocessador pode ser obtida através do CDROM da Siemens ou, por Fax: + 49 89 41 44 84 73, nº encomenda:B158-H6892-X-X-7600

FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO
BIBLIOTECA
0000006388