UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA
DEPARTAMENTO DE COMPUTAÇÃO E AUTOMAÇÃO

IMPLEMENTAÇÃO DE UM CONTROLADOR PID AUTO-AJUSTÁVEL

AUTOR: Eng. Alvaro Alberto Brito Llamosas.
ORIENTADOR: Prof. Dr. Luis Gimeno Latre.

Tese apresentada à Comissão de Pós-Graduação da Faculdade de Engenharia Elétrica , como parte dos requisitos para a obtenção do Título de Mestre em Engenharia Elétrica.

Abril de 1988.

Este exemplar corresponde à redação final da tese defendida por Alvaro Alberto Brito Llamosas e aprovada pela Comissão Julgadora em 23 de maio de 1988

RESUMO

Este trabalho apresenta uma implementação em tempo real, de um controlador PID auto-ajustável visando solucionar o problema servo e/ou regulador.

A implementação utiliza os serviços do Núcleo de Tempo Real NTR-80 (desenvolvido para o microprocessador Z-80) através de uma interface ASSEMBLY Z-80/PASCAL-Z.

Dois microcomputadores de processo CAMAÇARI-CALCON servem de base para o controlador PID auto-ajustável e a simulação de um processo de 2.ª ordem.

A implementação é constituída de módulos concorrentes, entre os quais estão um módulo de identificação (estimação) e módulos de comunicação com o meio externo - operador.

ABSTRACT

This work presents a real time implementation of a PID adaptive controller well suited for obtaining good regulator and servo response.

The implementation utilizes the NTR-80 real time kernel through commands in PASCAL-Z and an interface PASCAL-Z/ASSEMBLY Z-80.

The controller and the simulated process run in two CAMAÇARI-CALCON process microcomputers.

The implementation consists on concurrent tasks: identification, control and operator interface.

À minha família.

Este trabalho teve o apoio financeiro do Conselho Nacional de Pesquisa (CNPq) e da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP).

AGRADECIMENTOS

Ao Prof. Luis Gimeno Latre, sem cujo auxílio e boa vontade este trabalho não teria se concretizado;

Ao Prof. Wagner Caradori do Amaral, pela ajuda prestada nos Capítulos 2 e 3;

Ao Prof. Maurício Ferreira Magalhães, pelas críticas, correções e sugestões ao Capítulo 3;

Aos demais Professores, Funcionários e Alunos da UNICAMP que de alguma forma contribuíram com este trabalho;

Ao Centro Tecnológico para Informática (CTI), cujas instalações foram gentilmente franqueadas para a realização da implementação. Aos seus funcionários, que prestaram valioso e indispensável auxílio.

Minha gratidão.

ÍNDICE

CAPÍTULO 1.

Introdução 1

1.1 - Controle auto-ajustável 1

1.2 - Objetivo do trabalho 3

CAPÍTULO 2.

Fundamentos Teóricos 5

2.1 - Introdução 5

2.2 - Simulação do processo 6

2.2.1 - Processos SISO 6

2.2.2 - Gerador de ruído branco 8

2.3 - Algoritmo PID 10

2.4 - Algoritmo recursivo de estimação 15

2.5 - A alocação de polos na solução do problema SERVO 18

2.6 - A alocação de polos na solução do problema REGULADOR 25

CAPÍTULO 3.

Implementação do Controlador 29

3.1 - Introdução 29

3.2 - Características gerais da implementação 30

3.3 - Descrição sucinta das principais ferramentas empregadas 32

3.3.1 - Microcomputador Camaçari 32

3.3.2 - Núcleo de tempo real 33

3.3.2.1 - Introdução 33

3.3.2.2 - Estados 33

3.3.2.3 - Semáforos 35

3.3.2.4 - Características operacionais 37

3.3.2.5 - Funções 39

3.4 - Estruturação do "software" 42

3.4.1 - Introdução 42

3.4.2 - Tarefas Iniciação 1 e 2 45

3.4.3 - Tarefas Relógio 1 e 2 47
3.4.4 - Tarefa Simulação 48
3.4.4.1 - Corpo da tarefa 48
3.4.4.2 - Gerador de ruído branco 49
3.4.5 - Tarefa PID 50
3.4.6 - Tarefa Estimação/Alocação de Polos 52
3.4.7 - Tarefa Variância 55
3.4.8 - Tarefa Display 56
3.4.8.1 - Introdução 56
3.4.8.2 - Implementação 56
3.4.9 - Tarefa Operação 59
3.4.9.1 - Introdução 59
3.4.9.2 - Implementação 59
3.5 - Características operacionais da implementação 62
3.5.1 - Introdução 62
3.5.2 - Atraso de transporte adicional 62
3.5.3 - Estrutura da tarefa PID. Localização da região crítica 67
3.5.4 - Interface com o operador 69

CAPÍTULO 4.
Testes e Resultados Experimentais 75
4.1 - Introdução 75
4.2 - Processo simulado 75
4.3 - Especificação do comportamento servo em malha fechada 77
4.4 - Ruído do canal 81
4.5 - Resultados experimentais 83

CAPÍTULO 5.
Conclusões 97

REFERÊNCIAS BIBLIOGRÁFICAS 99

Apêndice A: Processos de segunda ordem 102
Apêndice B: Implementação do gerador de ruído branco 107
ANEXO A: Programas da implementação (em PASCAL).

CAPITULO 1

INTRODUÇÃO

1.1 - Controle Auto-ajustável.

Antes do advento dos controladores auto-ajustáveis existiam dois métodos de projeto de controladores: i) modelamento do processo e síntese do controle (demorado e de alto custo); ii) ajuste dos parâmetros de uma estrutura de controle definida a-priori (p. ex. a PID - este ajuste leva geralmente a um desempenho apenas razoável).

Um controlador auto-ajustável é aquele que possui a capacidade de reconhecer um processo e suas variações no tempo, ajustando ou alterando seus próprios parâmetros (segundo uma determinada estratégia) de forma a atender as especificações do usuário. É constituído então, em sua forma mais simples, de um módulo de estimação, outro onde é realizada a síntese de controle e o controlador propriamente dito, como mostrado na figura 1.1 (note-se que esta divisão é lógica, não necessariamente física).

O sinal de controle, u(t), e a saída do processo, y(t), são amostrados a cada instante de tempo T e utilizados para a estimação dos parâmetros do sistema. Através destes parâmetros (do sistema) e das especificações de desempenho do usuário, o módulo de síntese do controle determina os parâmetros ótimos para o controlador.

No caso de um processo variante no tempo, o módulo de estimação deve permanecer ativo de modo a reajustar o controlador.

Os controladores auto-ajustáveis caracterizam-se, dentro do contexto de controle adaptativo, pela imposição da separação entre

estimação e controle, como indica a figura 1.1 ([1]). Foram propostos inicialmente em [2], e a partir desta data tem havido um grande número de propostas de diferentes algoritmos consoante os critérios utilizados para a síntese de controle; variância mínima generalizada [3], posicionamento de polos [4], estrutura PID [5, 6], etc... A referência [6] contém um histórico dos controladores auto-ajustáveis.

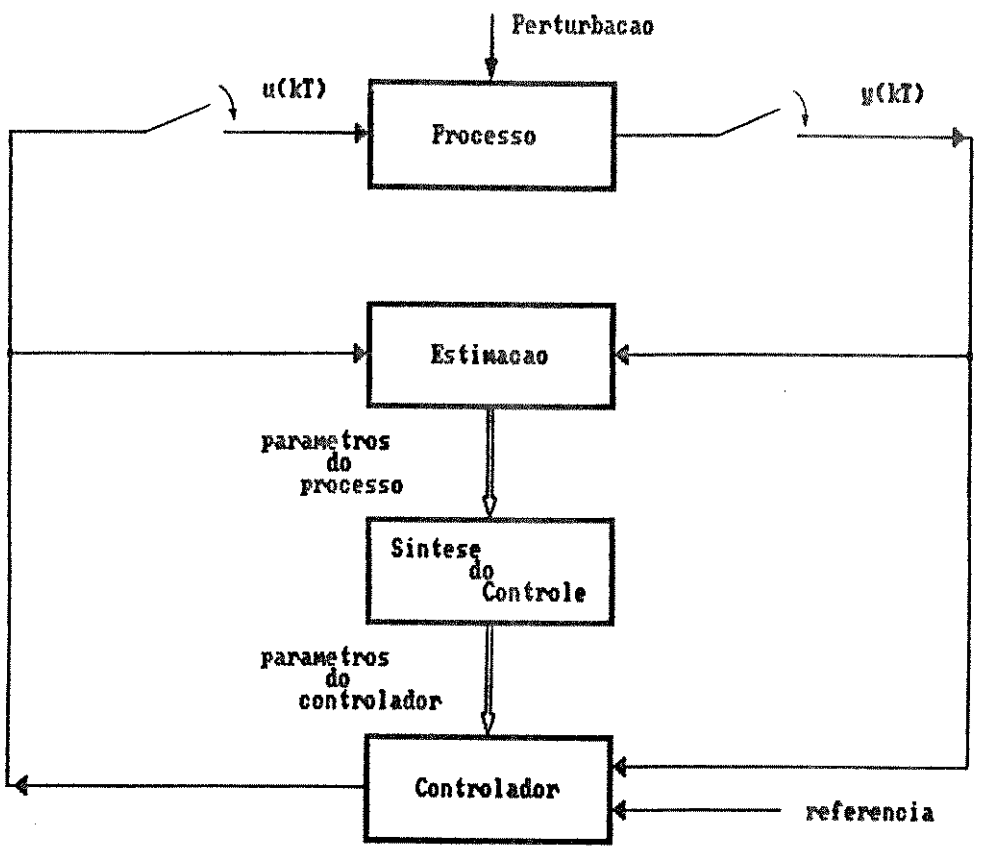

Figura 1.1: Controlador auto-ajustável na malha de controle.

Desde o início do desenvolvimento de algoritmos de controle auto-ajustáveis, a sua implementação baseada em microprocessadores tem sido um dos aspectos estudados. Este fato é devido a que a complexidade relativamente pequena do algoritmo permite pensar em implementações simples. Assim já em [3] é apresentada uma

implementação num microprocessador 8080 e uma versão multivariável foi implementada em [7] sobre um microprocessador 8086.

O objetivo de tornar os controladores auto-ajustáveis controladores de uso geral tem esbarrado na prática no problema de ajuste de parâmetros do mecanismo de estimação e de parâmetros relacionados com o critério utilizado para a síntese do controlador. Estes ajustes implicam num conhecimento do comportamento do algoritmo que o usuário industrial na operação do processo normalmente não tem. Isto tem levado recentemente a uma mudança de orientação passando da adaptação dos parâmetros baseada na estimação do modelo para a adaptação dos parâmetros baseada em regras heurísticas, [8].

O algoritmo apresentado em [6] caracteriza-se por uma interface simples com o usuário na medida em que, no que se refere a parâmetros para a síntese do controlador, apenas exige dele o comportamento tipo segunda ordem ("overshoot" e tempo de resposta), características estas que o usuário normalmente especifica sem maiores problemas.

1.2 - Objetivo do trabalho.

Este trabalho consiste na implementação, em tempo real, do controlador PID auto-ajustável desenvolvido em [6].

Assim, como em [6], utiliza básicamente a técnica de alocação de polos para determinar os parâmetros ótimos do algoritmo PID que desempenha o papel de controlador.

Um módulo alternativo permite que se ajuste os parâmetros do controlador PID de forma a satisfazer-se um compromisso entre a

variância do sinal de controle e o sinal de saída do processo.

Assim, para um determinado "hardware" (microcomputadores de controle de processos industriais) e "software" de apoio (Núcleo de Tempo Real e interface) pode-se verificar a exequibilidade e desempenho de um controlador não-comercial (protótipo) desenvolvido em linguagem de alto nível.

O conteúdo dos capítulos seguintes é:

- Capítulo 2: expõe resumidamente os conceitos teóricos matemáticos sobre os quais a implementação está assentada;

- Capítulo 3: é composto de duas partes: descrição das ferramentas utilizadas (hardware e software) na implementação e, a descrição detalhada de cada módulo (tarefa) do controlador, bem como da simulação do processo;

- Capítulo 4: mostra alguns resultados com o fito de ilustrar o funcionamento do controlador;

- Capítulo 5: conclusões e sugestões para trabalhos futuros.

CAPITULO 2

FUNDAMENTOS TEÓRICOS

2.1 - Introdução.

Neste capítulo estão expostos, de forma resumida, os conceitos teóricos (matemáticos e físicos) sobre os quais a implementação foi realizada.

Assim, a seção 2.2 introduz a representação matemática de processos SISO. Também nesta seção é apresentado o princípio de funcionamento do gerador de ruído branco utilizado.

Na seção 2.3 situa-se uma breve dissertação sobre o algoritmo PID e uma variante sua (a qual é utilizada neste trabalho).

A seção 2.4 consiste de uma ligeira explanação sobre o algoritmo recursivo de estimação empregado (Mínimos Quadrados Estendido).

A seção 2.5 apresenta, de forma compacta, a estratégia de alocação de polos na solução do problema Servo (seguimento de referência).

Por fim, na seção 2.6, fala-se sobre a alocação de polos na solução do problema Regulador. Tenta-se, nesta solução, atingir um compromisso satisfatório entre a variância do sinal de saída e de controle.

Os conceitos são expostos, por vezes, de maneira realmente sucinta (p. ex., a seção 2.4); mormente quando os resultados são bastante gerais. As referências citadas devem ser, na maior parte dos casos, suficientes para dirimir eventuais dúvidas.

2.2 - Simulação do Processo.

2.2.1. Processos SISO.

Um processo SISO (Single Input, Single Output), sem atraso de transporte e sujeito a perturbações estocásticas com densidade espectral racional, pode ser descrito no domínio da variável complexa Z por:

$$A(z^{-1})\ Y(z) = z^{-1}\ B(z^{-1})\ U(z) + C(z^{-1})\ E(z) \qquad (2.1)$$

onde E(z) = transformada Z do ruído branco;

U(z) = transformada Z da entrada do sistema (sinal de controle)

Y(z) = transformada Z da saída do sistema;

conforme ilustrado na figura abaixo:

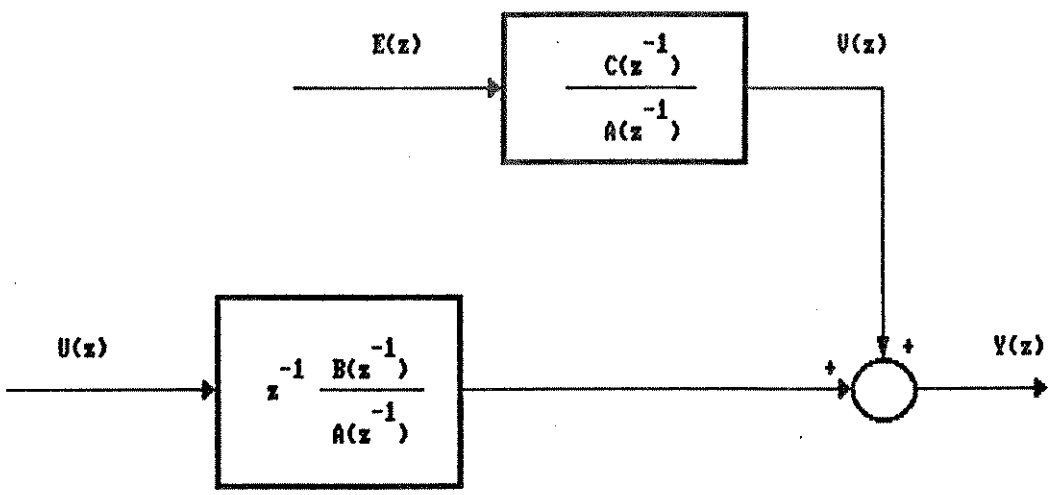

Figura 2.1: Modelo do processo e perturbação.

$A(z^{-1})$, $B(z^{-1})$ e $C(z^{-1})$ são polinômios em Z obtidos a partir da discretização do modelo dinâmico contínuo e da inclusão de um filtro linear atuando sobre o ruído branco de acordo com o teorema da representação, [1].

Os resultados experimentais apresentados no Capítulo 5 são obtidos sobre processos simulados de segunda ordem com a seguinte estrutura para os polinômios $A(z^{-1})$, $B(z^{-1})$ e $C(z^{-1})$:

$$A(z^{-1}) = 1 + a_1 z^{-1} + a_2 z^{-2} \qquad (2.2)$$

$$B(z^{-1}) = b_0 + b_1 z^{-1} \qquad (2.3)$$

$$C(z^{-1}) = 1 + c_1 z^{-1} + c_2 z^{-1} \qquad (2.4)$$

Interpretando-se z^{-1} como o operador atraso unitário a equação 2.1 pode ser escrita na forma de uma equação à diferenças:

$$y(t) = -a_1 y(t-T) - a_2 y(t-2T) + b_0 u(t-T) + b_1 u(t-2T) + e(t) + c_1 e(t-T) + c_2 e(t-2T). \qquad (2.5)$$

Esta equação é a utilizada para simular o processo a ser controlado.

Uma forma adequada para implementar a equação em computador é armazenar as constantes e as variáveis em dois vetores distintos:

$$y(t) = |a_1\ a_2\ b_0\ b_1\ c_1\ c_2| \begin{vmatrix} -y(t-T) \\ -y(t-2T) \\ u(t-T) \\ u(t-2T) \\ e(t-T) \\ e(t-2T) \end{vmatrix} + e(t). \qquad (2.6)$$

ou, em forma compacta:

$$y(t) = \theta^T(t) * \varphi(t) + e(t); \qquad (2.7)$$

onde

$$\theta^T(t) = |a_1\ a_2\ b_0\ b_1\ c_1\ c_2| \qquad \text{e}$$

$$\varphi^T(t) = |-y(t-T)\ -y(t-2T)\ u(t-T)\ u(t-2T)\ e(t-T)\ e(t-2T)|.$$

2.2.2. Gerador de Ruído Branco.

Na escolha de um gerador de números (pseudo) aleatórios, devida atenção deve ser dada as características que determinam sua qualidade ([9]).

Para gerar ruído branco de distribuição Normal necessita-se de um gerador de números aleatórios uniformemente distribuídos. O gerador utilizado neste trabalho emprega um método misto congruente, o qual, através de multiplicação e adição, gera uma sequência (N_i) de inteiros positivos menores que algum valor positivo m. Neste método cada número aleatório N_{i+1} é obtido por:

$$N_{i+1} = (a * N_i + c) \text{ MOD } m \qquad (2.8)$$

A escolha das constantes inteiras a, c e m é importante dado que os valores escolhidos afetarão as propriedades estatísticas da sequência aleatória tais como o período, ou seja, o comprimento da sequência antes que ela volva a repetir.

Dado que é conveniente operar com números aleatórios uniformemente distribuídos no intervalo (0, 1), o algoritmo final será:

$$N_{i+1} = (a * N_i + c) \text{ MOD } m$$
$$r = N_{i+1} / m. \qquad (2.9)$$

Os critérios utilizados para que a escolha destas constantes de forma que a semente (No) possa ser escolhida arbitrariamente são ([9], [10] e [11]):

1. $a = 8t \pm 5$ (t um inteiro positivo);
2. $m/100 < a < m - \sqrt{m}$;

3. os dígitos binários de a não possuem um padrão óbvio;
4. c deve ser um inteiro ímpar com $c/m \cong 0.21132$.

Testes sobre a performance de um gerador assim obtido podem ser encontrados em [11].

2.3 - Algoritmo PID.

O desejo de manter a saída do processo y(t), em algum determinado nível, leva à utilização de uma estratégia de controle. Uma estratégia de controle é um algoritmo ou equação que determina o valor do sinal que deve ser aplicado ao processo (para que ele produza o resultado esperado), como uma função do erro medido (presente e passado). Assim, um diagrama de blocos básico de controle por realimentação é:

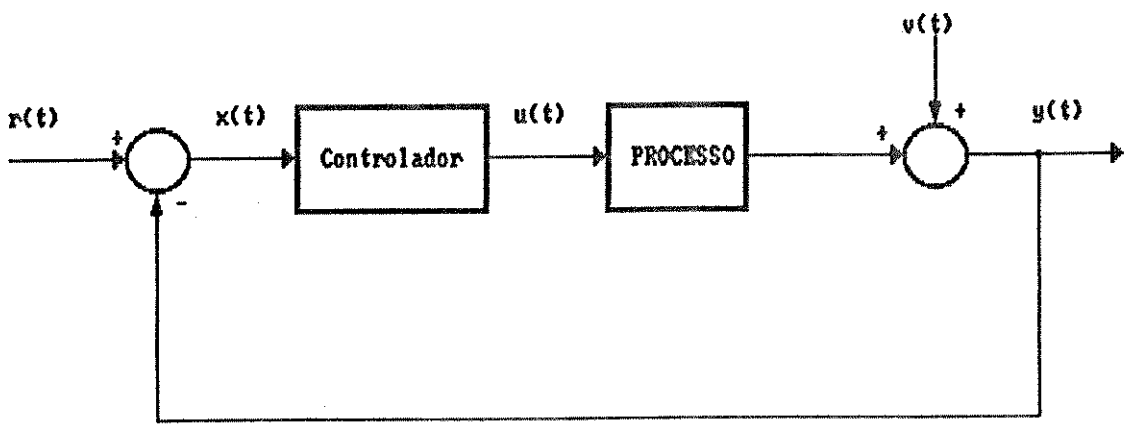

Figura 2.2: Controle por realimentação.

Um algoritmo muito utilizado em controle convencional por realimentação é o controlador de três modos (PID), [12].

O algoritmo PID, em sua forma teórica é constituído por:

$$u(t) = u_o + K_p \left[x(t) + 1/T_i \int_0^t x(\tau)d\tau + T_d \, dx(t)/dt \right] \qquad (2.10)$$

onde u(t) é a saída do controlador e x(t) o erro - ou seja, a diferença entre o "set-point" (referência) e a saída do processo, y(t).

A primeira parcela, u_o, é o valor de regime.

A segunda parcela, $u_p(t) = K_p \cdot x(t)$, corresponde à ação proporcional à diferença entre a saída do processo e o "set-point".

A terceira parcela, $u_i(t) = K_p/T_i \int_o^t x(\tau)\, d\tau$, exercerá uma função integradora sobre o erro $x(t)$, reduzindo-o eventualmente à zero em função do tipo de entrada.

A quarta parcela, $u_d(t) = K_p\ T_d\ dx(t)/dt$, atua proporcionalmente à derivada primeira do erro, isto é, quanto mais brusca for a evolução temporal do erro, maior efeito a ação derivativa terá.

Os parâmetros K_p, T_i e T_d devem ser ajustados de forma a alcançar-se a resposta desejada em malha fechada.

Para obter-se o equivalente digital do controlador PID, os termos integral e derivativo são numéricamente aproximados (utilizando-se integração retangular e derivação triangular, [6]) por:

$$u_n = u_o + K_p \left[x_n + T/T_i \sum_{i=o}^{n} x_i + T_d/T\ (x_n - x_{n-1})\right] \qquad (2.11)$$

onde:

- u_n = saída do controlador no n-ésimo instante de amostragem;
- x_n = erro no n-ésimo instante de amostragem;
- u_o = o "off-set" já mencionado.

A equação 2.11 é conhecida como a forma posição do algoritmo PID, devido a que a saída real do controlador é computada. Uma forma alternativa, a "velocidade", é obtida escrevendo-se inicialmente a

expressão para a saída do controlador no (n-1)-ésimo instante de amostragem

$$u_{n-1} = u_o + K_p \left[x_{n-1} + T/T_i \sum_{i=0}^{n-1} x_i + T_d/T \left(x_{n-1} - x_{n-2}\right)\right] \quad (2.12)$$

e em seguida subtraindo-a da equação 2.11, resultando:

$$u_n = u_{n-1} + K_p \left[(x_n - x_{n-1}) + T/T_i\, x_n + T_d/T \left(x_n - 2x_{n-1} + x_{n-2}\right)\right] \quad (2.13)$$

A forma "velocidade" (equação 2.13) do controlador PID computa a saída incremental ao invés da saída real do controlador e provê alguma proteção contra "reset windup", porque não incorpora somas de sequências de erros.

Uma forma modificada de implementação do algoritmo PID que tem a vantagem de não introduzir zeros adicionais na função de transferência entre o "set-point" e a saída, no sistema em malha fechada ([61]), é a mostrada na seguinte figura:

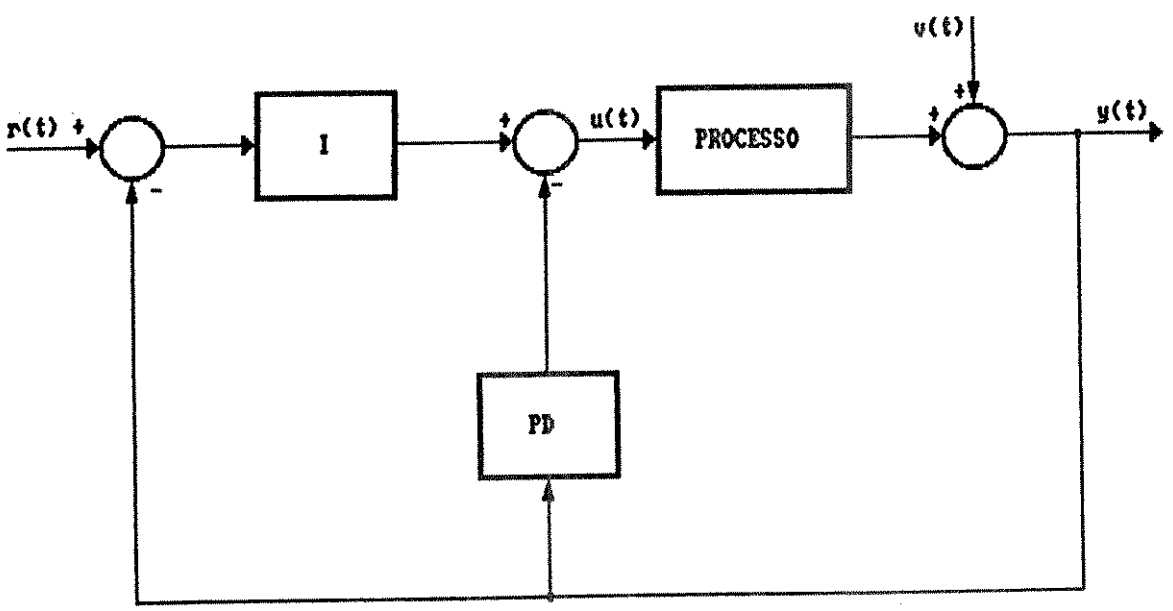

Figura 2.3: Implementação alternativa do controle PID.

A forma "posição" desta implementação é:

$$u(t) = [-y(t)\ (Td+T) + Kp\ Td/T\ y(t-T) + Kp\ T/Ti \sum_{\tau=0}^{t} [r(\tau) - y(\tau)] \qquad (2.14)$$

E a forma velocidade:

$$u(t) = u(t-T) + Kp\ T/Ti\ r(t) - go\ y(t) - g1\ y(t-T) - g2\ y(t-2T) \qquad (2.15)$$

onde:

$$go = Kp\ (1 + Td/T + T/Ti); \qquad (2.16)$$

$$g1 = Kp\ (-2\ Td/T - 1); \qquad (2.17)$$

$$g2 = Kp\ Td/T. \qquad (2.18)$$

Neste trabalho utiliza-se esta última forma (equação 2.15), a qual possui por transformada Z:

$$U(z) = \frac{Kp\ T\ Ti}{(1 - z^{-1})} R(z) - \frac{G(z^{-1})}{(1 - z^{-1})} Y(z) \qquad (2.19)$$

com $$G(z^{-1}) = go + g1\ z^{-1} + g2\ z^{-2}. \qquad (2.20)$$

A representação da saída em função do sinal de referência e da perturbação estocástica é dada por:

$$Y(z) = \frac{z^{-1}\ B(z^{-1})\ Kp\ T/Ti}{A(z^{-1})\ (1 - z^{-1}) + z^{-1}\ B(z^{-1})\ G(z^{-1})} R(z) + \frac{A(z^{-1})\ (1 - z^{-1})}{A(z^{-1})\ (1 - z^{-1}) + z^{-1}\ B(z^{-1})\ G(z^{-1})} V(z). \qquad (2.21)$$

$$\text{onde} \quad V(z) = \frac{G(z^{-1})}{A(z^{-1})} \; E(z). \qquad (2.22)$$

2.4 - Algoritmo Recursivo de Estimação.

O modelo ARMAX utilizado tem a forma (ver equação 2.5):

$$y(t) + a_1\, y(t-1) + \ldots + a_n\, y(t-n) = b_1\, u(t-1) + \ldots + b_m\, u(t-m) + e(t) + c_1\, e(t-1) + \ldots + c_r\, e(t-r);$$

e pode ser representado em forma vetorial por (ver equação 2.7):

$$y(t) = \theta^T(t)\ \varphi(t) + e(t),$$

como foi indicado na seção 2.2.1.

Os algoritmos recursivos de estimação atualizam o vetor $\theta^T(t)$ através do mecanismo descrito à continuação utilizando os valores de entrada e saída contidos no vetor $\varphi(t)$. O esquema geral é o representado na figura 2.4:

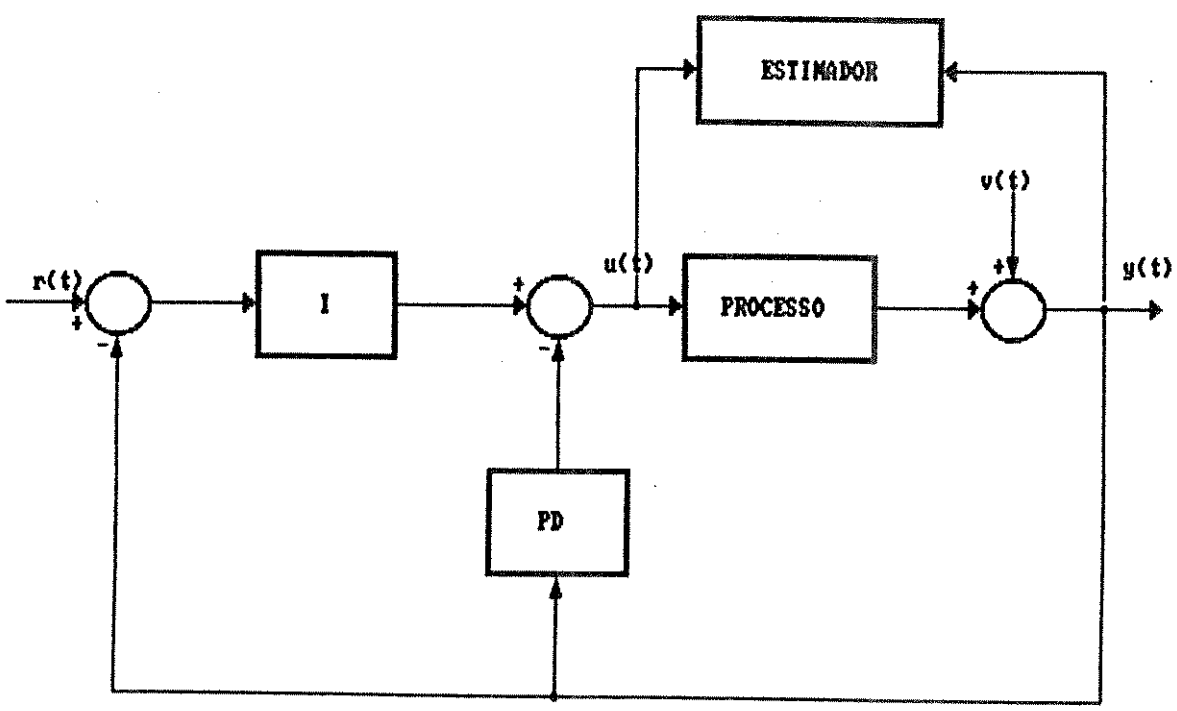

Figura 2.4: Estimação de um processo desconhecido.

Neste trabalho foi utilizado o algoritmo dos Mínimos Quadrados Estendido. A sua versão recursiva é ([14]):

$$\hat{\underline{\theta}}(t) = \hat{\underline{\theta}}(t-1) + \underline{L}(t)\,[y(t) - \underline{\varphi}^T(t)\,\hat{\underline{\theta}}(t-1)] \qquad (2.23)$$

$$\underline{L}(t) = \frac{P(t-1)\;\underline{\varphi}(t)}{\lambda(t) + \underline{\varphi}^T(t)\;P(t-1)\;\underline{\varphi}(t)} \qquad (2.24)$$

$$P(t) = \frac{1}{\lambda(t)}\left[P(t-1) - \frac{P(t-1)\;\underline{\varphi}(t)\;\underline{\varphi}^T(t)\;P(t-1)}{\lambda(t) + \underline{\varphi}^T(t)\;P(t-1)\;\underline{\varphi}(t)}\right] \qquad (2.25)$$

A inicialização da matriz P, dos vetores $\varphi(t)$, L(t) e $\hat{\theta}(t)$, bem como a convergência do algoritmo é discutida em [14].

O fator de esquecimento, $\lambda(t)$, permite que o algoritmo rastreie parâmetros do processo que sejam variantes no tempo.

Uma acentuada melhoria nas qualidades numéricas deste algoritmo pode ser obtida se a matriz **P** é fatorada no produto de duas ou três matrizes, e estas matrizes são atualizadas no lugar de **P**.

Utilizou-se aqui a fatorização conhecida como **U-D**, [8]. Nela P(t) é escrita como

$$P(t) = U(t)\,D(t)\,U^T(t) \qquad (2.26)$$

onde U(t) é uma matriz triangular superior com todos os elementos da diagonal iguais a 1 e D(t) é uma matriz diagonal.

A fatorização garante que P(t) permanece definida positiva e além disto, experimentos demonstraram que a fatorização possui boa

estabilidade numérica e que os erros de arredondamento não afetam significantemente a solução ([14]).

O algoritmo utilizado está descrito em [14] e foi originalmente publicado em [15].

2.5 - A alocação de polos na solução do problema SERVO.

Como foi visto na seção 2.3 a componente determinística da saída em malha fechada na implementação proposta é:

$$Yd(z) = \frac{z^{-1}\ B(z^{-1})\ Kp\ \ T/Ti}{A(z^{-1})\ (1 - z^{-1}) + z^{-1}\ B(z^{-1})\ G(z^{-1})}\ R(z)$$

Dada a ordem dos polinômios **A**, **B** e **G** sabe-se que a ordem do denominador é igual a quatro.

A especificação do comportamento em malha fechada pelo usuário será feita como se o sistema a controlar fosse de 2.ª ordem, simplificação esta bem frequente na prática industrial. Um sistema de segunda ordem

$$H(s) = \frac{wn^2}{s^2 + 2\ \zeta\ wn + wn^2} \qquad (2.28)$$

acrescido de um segurador de ordem zero, possui função de transferência em Z igual a

$$H(z) = \frac{fo\ z^{-1} + f1\ z^{-2}}{1 + d1\ z^{-1} + d2\ z^{-2}} \qquad (2.29)$$

Assim, a estratégia seguida é a de alocar-se dois polos da expressão

$$A(z^{-1})\ (1 - z^{-1}) + z^{-1}\ B(z^{-1})\ G(z^{-1});$$

de modo a satisfazer o comportamento desejado em malha fechada, enquanto os dois polos restantes são alocados o mais próximo possível

da origem, de forma a minimizar sua influência. Note-se que não se procura o cancelamento dos zeros, evitando-se eventuais problemas de instabilidade.

Definindo $D(z^{-1}) = 1 + d_1 z^{-1} + d_2 z^{-2}$ como o polinômio que contém os polos dominantes e sendo $\gamma(z^{-1}) = 1 + \gamma_1 z^{-1} + \gamma_2 z^{-2}$ o polinômio que define os outros dois polos, pode-se escrever:

$$A(z^{-1})(1 - z^{-1}) + z^{-1} B(z^{-1}) G(z^{-1}) =$$

$$(1 + d_1 z^{-1} + d_2 z^{-2})(1 + \gamma_1 z^{-1} + \gamma_2 z^{-2}) \qquad (2.30)$$

Para a definição de d_1, d_2, γ_1 e γ_2 estão disponíveis apenas os coeficientes do polinômio $G(z^{-1})$; g_0, g_1 e g_2. Resta então, apenas um grau de liberdade para a determinação dos parâmetros γ_1 e γ_2.

Igualando-se os termos de mesma potência da equação 2.30 ([6]), obtém-se:

$$g_0 = \frac{\gamma_1}{b_0} + \frac{d_1 - a_1 + 1}{b_0} \qquad (2.31)$$

$$g_1 = \frac{d_2 \gamma_1 + d_1 \gamma_2 + a_2}{b_1} - \frac{b_0 d_2 \gamma_2}{b_1^2} \qquad (2.32)$$

$$g_2 = \frac{d_2 \gamma_2}{b_1} \qquad (2.33)$$

$$\gamma_2 = k_1 \gamma_1 + k_2 \qquad (2.34)$$

$$k_1 = b_1/b_o \qquad \text{e} \qquad (2.35)$$

$$k_2 = - \frac{\left[a_2 - a_1 + \dfrac{b_o\, a_2}{b_1} + \dfrac{b_1}{b_o}\,(d_1 - a_1 + 1) - d_2\right]}{\dfrac{b_o\, d_1}{b_1} - \dfrac{b_o^2\, d_2}{b_1^2} - 1} \qquad (2.36)$$

Determinados d_1 e d_2 através das especificações do usuário para o sistema em malha fechada (ζ e w_n), γ_1 e γ_2 devem ser escolhidos de tal forma que os polos de $(1 + \gamma_1 z^{-1} + \gamma_2 z^{-2})$ tenham o menor módulo possível. O único grau de liberdade existente para a definição de γ_1 e γ_2 apenas permite que estes coeficientes satisfaçam a equação 2.34.

De [6] obtém-se a seguinte ilustração:

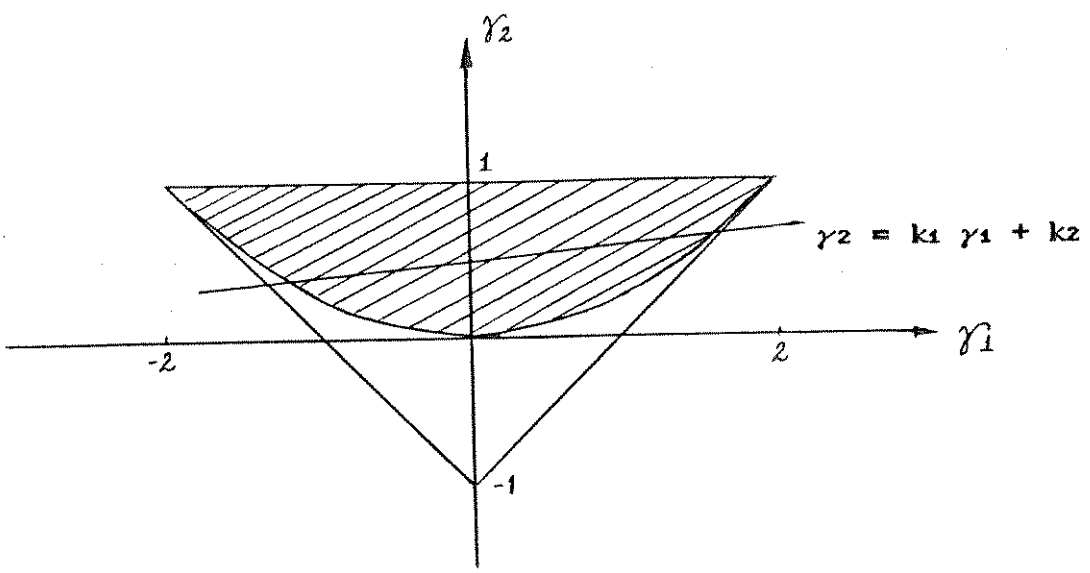

Figura 2.5: **Região de estabilidade para um polinômio de segunda ordem.**

A área formada pelo triângulo restringe os valores de γ_1 e γ_2 para os quais tem-se os polos de $(1 + \gamma_1 z^{-1} + \gamma_2 z^{-2})$ dentro do círculo unitário (estabilidade - ver [16]). A área hachurada indica a região de polos complexos. A reta corresponde à relação linear entre γ_1 e γ_2, isto é $\gamma_2 = k_1 \gamma_1 + k_2$.

Os polos da expressão $(1 + \gamma_1 z^{-1} + \gamma_2 z^{-2})$ são:

$$P_1 = -\frac{\gamma_1}{2} + \frac{\sqrt{\gamma_1^2 - 4\gamma_2}}{2} \qquad (2.37)$$

$$P_2 = -\frac{\gamma_1}{2} - \frac{\sqrt{\gamma_1^2 - 4\gamma_2}}{2} \qquad (2.38)$$

Pode-se provar que ([6]):

1. Se a reta cruza a região correspondente à polos complexos, então a intersecção da reta com a parabóla que possuir menor ordenada (γ_2) será o ponto procurado.

Nas intersecções, $\gamma_1^2 - 4\gamma_2 = 0$ e $\gamma_2 = k_1 \gamma_1 + k_2$,

então $\gamma_1' = 2k_1 + 2\sqrt{k_1^2 + k_2}$ e $\gamma_1'' = 2k_1 - 2\sqrt{k_1^2 + k_2}$;

fornecendo $P_1' = P_2' = -\gamma_1' / 2$ ou (2.39)

$P_1'' = P_2'' = -\gamma_1'' / 2.$ (2.40)

Para escolher-se a intersecção correta basta verificar-se quem

possui menor módulo , γ_1' ou γ_1''.

2. Se a reta não cruza a região mencionada, o ponto desejado será aquele que possuir abscissa (γ_1) igual à zero. Daí, vem:

$$P_1''' = P_2''' = \sqrt{-k_2} \; . \qquad (2.41)$$

Note-se que se os polos são reais então

$$\gamma_1^2 - 4\,\gamma_2 > 0.$$

Usando-se $\gamma_2 = k_1\,\gamma_1 + k_2$, vem que:

$$\gamma_1^2 - 4\,k_1\,\gamma_1 - 4\,k_2 > 0.$$

Esta última inequação representa uma parabóla que não cruza o eixo real γ_1. Seu discriminante é:

$$\Delta = 16\,k_1^2 + 16\,k_2 < 0 \quad \Longrightarrow \quad k_1^2 + k_2 < 0.$$

Estes resultados foram utilizados para a elaboração do seguinte algoritmo , o qual fornece os parâmetros do PID através das especificações do usuário:

1. Faça

$k_1 = b_1/b_0$ e

$$k_2 = - \frac{\left[a_2 - a_1 + \frac{b_0\, a_2}{b_1} + \frac{b_1}{b_0}(d_1 - a_1 + 1) - d_2\right]}{\frac{b_0\, d_1}{b_1} - \frac{b_0^2\, d_2}{b_1^2} - 1}$$

2. Se $k_1^2 + k_2 < 0$, faça $\gamma_1 = 0$ e vá para 4, se não, prossiga:

3. $\gamma_1' = 2\,k_1 + 2\sqrt{k_1^2 + k_2}$; $\gamma_1'' = 2\,k_1 - 2\sqrt{k_1^2 + k_2}$

Se $|\gamma_1'| < |\gamma_1''|$, vá para 4, se não, $\gamma_1' = \gamma_1''$.

4. $\gamma_2 = k_1\,\gamma_1' + k_2$; $\gamma_1 = \gamma_1'$.

5. Faça

$$g_0 = \frac{\gamma_1}{b_0} + \frac{d_1 - a_1 + 1}{b_0}$$

$$g_1 = \frac{d_2\,\gamma_1 + d_1\,\gamma_2 + a_2}{b_1} - \frac{b_0\, d_2\,\gamma_2}{b_1^2}$$

$$g_2 = \frac{d_2\,\gamma_2}{b_1}$$

$$T_d = \frac{g_2\, T}{-2\,g_2 - g_1} \qquad (2.42)$$

$$K_p = \frac{g_1}{-2\ T_d/T - 1} \qquad (2.43)$$

$$T_i = \frac{T}{g_o/K_p - 1 - T_d/T} \qquad (2.44)$$

Sendo que as equações 2.42, 2.43 e 2.44 foram obtidas a partir das equações 2.16, 2.17 e 2.18.

2.6 - A alocação de polos na solução do problema REGULADOR.

A estratégia de alocação de polos exposta na seção 2.5 não leva em consideração o fato de que as variâncias dos sinais de saída e controle podem ser excessivas.

Há duas possibilidades de contornar-se este problema. Uma é a alocação experimental (talvez heurística) dos coeficientes γ_1 e γ_2 até obter-se variâncias razoáveis. A outra é a proposta em [6], em que, dada a ordem baixa do modelo (segunda), obtém-se as expressões analíticas para as variâncias dos sinais de saída e controle. Estas expressões, embora exijam grande quantidade de cálculo, podem ser utilizadas para a previsão das variâncias que resultarão de um determinado ajuste dos parâmetros do controlador. Note-se contudo que, as equações obtidas, dadas a seguir, utilizam o polinômio $C(z^{-1})$, o que implica em resultados pouco precisos.

As equações 2.45 e 2.46 fornecem as componentes estocásticas do sinal de controle e da saída do processo:

$$Y_p(z) = \frac{C(z^{-1})\,(1 - z^{-1})}{D(z^{-1})\quad \gamma(z^{-1})}\quad E(z) \qquad (2.45)$$

$$U_p(z) = \frac{-\,C(z^{-1})\,G(z^{-1})}{D(z^{-1})\quad \gamma(z^{-1})}\quad E(z) \qquad (2.46)$$

As variâncias de $Y_p(t)$ e $U_p(t)$ foram obtidas em [6], em função de γ_1 e γ_2 e definidas como:

$$\text{Var } Y_p(t) = \sigma_e^2\, f_1(\gamma_1, \gamma_2) \qquad (2.47)$$

$$\text{Var } U_p(t) = \sigma_e^2\, f_2(\gamma_1, \gamma_2) \qquad (2.48)$$

onde σ_e^2 é a variância do ruído "branco estacionário" (a média do ruído é assumida ser igual à zero). Também:

$$fi = \frac{-t_1 m_8 - t_2 m_9 - t_3 m_{12} - t_4 m_{14} + \alpha_0 m_1 + \alpha_1 m_2 + \alpha_2 m_3 + \alpha_3 m_4 + \alpha_4 m_5}{1 + t_1 m_6 + t_2 m_9 + t_3 m_{11} + t_4 m_{13}}$$

$m_1 = \alpha_0;$

$m_2 = -t_1 m_1 + \alpha_1;$

$m_3 = -t_1 m_2 - t_2 m_1 + \alpha_2;$

$m_4 = -t_1 m_3 - t_2 m_2 - t_3 m_1 + \alpha_3;$

$m_5 = -t_1 m_4 - t_2 m_3 - t_3 m_2 - t_4 m_1 + \alpha_4;$

$m_6 = (-t_1 t_4 + t_3 - t_2 / (1 + t_4) + t_4 t_3 - t_1) / m_7;$

$m_7 = 1 + t_2 - t_4 (t_2 + t_4) + (t_1 t_4 - t_3) (t_1 + t_3) / (1 + t_4);$

$m_8 = [(t_1 t_4 - t_3) (\alpha_2 m_1 + \alpha_3 m_2 + \alpha_4 m_3) / (1 + t_4) -$

$-t_4 (\alpha_3 m_1 + \alpha_4 m_2) + \alpha_1 m_1 + \alpha_2 m_2 + \alpha_3 m_3 + \alpha_4 m_4] / m_7;$

$m_9 = [-(t_1 + t_3) m_6 - t_2] / (1 + t_4);$

$m_{10} = [-(t_1 + t_3) m_8 + \alpha_2 m_1 + \alpha_3 m_2 + \alpha_4 m_3] / (1 + t_4);$

$m_{11} = -t_1 m_9 - (t_2 + t_4) m_6 - t_3;$

$m_{12} = -t_1 m_{10} - (t_2 + t_4) m_8 + \alpha_3 m_1 + \alpha_4 m_2;$

$m_{13} = -t_1 m_{11} - t_2 m_9 - t_3 m_6 - t_4;$

$m_{14} = -t_1 m_{12} - t_2 m_{10} - t_3 m_8 + m_6;$

$t_1 = d_1 + \gamma_1;$

$t_2 = d_2 + \gamma_2 + d_1 \gamma_1;$

$t_3 = d_2 \gamma_1 + d_1 \gamma_2;$

$t_4 = d_2 \gamma_2;$

Para $f_i = f_2$ $\alpha_0 = g_0$

$\alpha_1 = g_0 c_1 + g_1$

$\alpha_2 = g_2 + g_0 c_2 + g_1 c_1$

$\alpha_3 = g_1 c_2 + g_2 c_1$

$\alpha_4 = g_2 c_2$

Sendo os coeficientes g_0, g_1 e g_2 obtidos das equações 2.31, 2.32 e 2.33, repetidas aqui:

$$g_0 = \frac{\gamma_1}{b_0} + \frac{d_1 - a_1 + 1}{b_0}$$

$$g_1 = \frac{d_2 \gamma_1 + d_1 \gamma_2 + a_2}{b_1} - \frac{b_0 d_2 \gamma_2}{b_1^2}$$

$$g_2 = \frac{d_2 \gamma_2}{b_1}$$

Para $f_i = f_1$ $\alpha_0 = 1$

$$\alpha_1 = c_1 - 1$$

$$\alpha_2 = c_2 - c_1$$

$$\alpha_3 = -c_2$$

$$\alpha_4 = 0$$

Os parâmetros do PID, para um determinado polinômio $\gamma(z^{-1})$, são obtidos através das equações 2.31, 2.32, 2.33, 2.42, 2.43 e 2.44.

Concluindo: as equações 2.47 e 2.48 fornecem as variâncias dos sinais de saída e controle do processo para diferentes polinômios $\gamma(z^{-1})$. Para tanto utilizam também o vetor $\hat{\theta}(t)$, a variância do ruído e os coeficientes d_1 e d_2 (determinados pelas especificações do usuário). É possível então definir os coeficientes γ_2 e γ_2 que resultem em variâncias adequadas. Dado que esta escolha será provavelmente diferente daquela obtida pela estratégia descrita na seção 2.5, ou seja, interferirá no problema servo, um filtro poderá ser adicionado ao sinal de referência, anulando a influência do polinômio $\gamma(z^{-1})$ na parcela determinística da equação 2.21 (ver [6]).

CAPITULO 3

IMPLEMENTAÇÃO DO CONTROLADOR.

3.1 - Introdução.

Este capítulo descreve a implementação e as ferramentas por ela empregadas.

Então, a seção 3.2 cita as características gerais da implementação, tais como os recursos de "hardware" e "software" utilizados.

A seção 3.3 engloba a descrição do "hardware" final da implementação e o "software" especializado que viabiliza as operações em tempo real: o Núcleo de tempo Real.

Na seção 3.4 são descritos, detalhadamente, os módulos que compõem a implementação:

Tarefas de Iniciação;

Tarefas Relógio;

Tarefa Simulação;

Tarefa PID;

Tarefa Estimação/Alocação de Polos;

Tarefa Variância;

Tarefa Display;

Tarefa Operação;

Por fim, a seção 3.5 comenta algumas características operacionais da implementação.

3.2 - Características Gerais da Implementação.

A implementação foi realizada em dois microcomputadores industriais de controle de processos: um para simulação e o outro para controle.

A implementação caracteriza-se por:

- utilização de ferramentas de "software" para aplicações em tempo real: núcleo de Sistema Operacional tempo real;

- utilização de conversores A/D e D/A para comunicação entre os dois micros;

- interface com o operador através de tarefas interativas "on-line";

A implementação permite modificar com grande facilidade a dinâmica do processo simulado e a simulação de processos variantes no tempo.

De um modo geral este trabalho de implementação permitiu avaliar:

- as dificuldades de implementação de um algoritmo de controle como o descrito no capítulo anterior;

- o desempenho do algoritmo face às limitações do processador e portanto as capacidades de um microcomputador do tipo utilizado;

- a praticabilidade da utilização de linguagens de alto nível (PASCAL).

Os algoritmos foram desenvolvidos inicialmente em um VAX 11/785, e posteriormente transportados para um I-7000 (ITAUTEC), devido a utilização de um compilador disponível para o microprocessador Z-80, o PASCAL-Z, [17]. Este compilador possui uma biblioteca de características reentrantes, o que é desejável para aplicação em tempo real.

Assim, os módulos (tarefas), são editados, compilados pelo PASCAL-Z, "assemblados" e "linkados" com rotinas de acesso às portas A/D e D/A e também com uma tabela que faz a conexão com um Núcleo de Tempo Real (ver seção 3.2.2).

Por fim, o programa final (em hexadecimal) é transmitido via RS-232 (porta serial) para o microcomputador de processos, onde um Núcleo de Tempo Real encontra-se armazenado em EPROM.

Para a manipulação dos resultados optou-se, por razões de padronização, pela utilização do pacote gráfico TURBO-GRAPHIX, o qual opera baseado em um XT2002 (Personal Computer). Então, os dados são enviados pelo microcomputador de processos via RS-232 para o XT2002, em tempo real, onde serão manipulados "off-line".

O minicomputador VAX foi utilizado, também, através de terminal gráfico colorido TEKTRONIX , para cálculo e avaliação de modelos, utilizando-se os pacotes CADCLA (Computer Aided Design - técnicas CLÁssicas) e CADPID (Computer Aided Design - PID), desenvolvidos no CTI ([18]).

3.3 - Descrição sucinta das principais ferramentas empregadas.

3.3.1. Microcomputador Camaçari.

Os microcomputadores para controle de processos industriais CAMAÇARI, são fabricados pela CALCON Tecnologia SA. Utilizam como CPU o microprocessador Z-80 (8 bits) e são constituídos de circuitos impressos inseridos em um "rack", de tal forma que o usuário pode, com grande facilidade, alterar sua configuração. Para este trabalho, foram configurados da seguinte forma:

i. Uma placa de CPU, contendo EPROM, comunicação serial e interface para vídeo.

ii. Uma placa de memória RAM dinâmica, com 64 K de capacidade.

iii. Uma placa para conversão Digital/Analógica.

iv. Uma placa para conversão Analógica/Digital.

Os microcomputadores CAMAÇARI permitem a utilização ainda de cartão de saída para relés, cartão de entradas digitais, cartões de memória estática, cartão para controle de discos flexíveis (FDC), etc... Para uma introdução à computadores de processos industriais, veja [19].

3.3.2. Núcleo de Tempo Real.

3.3.2.1. Introdução.

Nesta seção descrever-se-á sucintamente o Núcleo de Tempo Real NTR-80 ([20]), desenvolvido no Centro Tecnológico de Informática/UNICAMP, utilizado na implementação. Um Núcleo de Tempo Real tem por finalidade fornecer um ambiente adequado à Programação Concorrente ([21], [22]). Para utilização do NTR-80 por uma linguagem de alto nível tornam-se necessários os serviços de uma interface que torne transparentes certos aspectos inerentes a uma linguagem Assembly, tais como a utilização de registradores, "flags", etc... A interface entre o NTR-80 e o Pascal-Z, [17], tornou-se possível graças à característica de reentrância da biblioteca deste compilador, como já mencionado. O NTR-80 é exposto aqui sob uma ótica de linguagem de baixo nível, e a sua utilização empregando como linguagem de programação o Pascal-Z será abordada subsequentemente (seção 3.3).

3.3.2.2. Estados.

Como mostra a figura 3.2, uma tarefa sob o gerenciamento do NTR-80 pode estar em quatro estados distintos:

No estado Dormindo, apesar da existência do código da tarefa na memória, a mesma virtualmente não existe para o Núcleo. Quando o sistema vai ser iniciado todas as tarefas acham-se neste estado.

No estado Executando, a tarefa detém o controle do processador. Note-se aqui que, como o NTR-80 é um núcleo preemptivo, uma tarefa que esteja no estado Executando pode perder o processador caso uma outra que possua maior prioridade passe para o estado Pronto, devido à eventos externos ou internos (temporização, por exemplo).

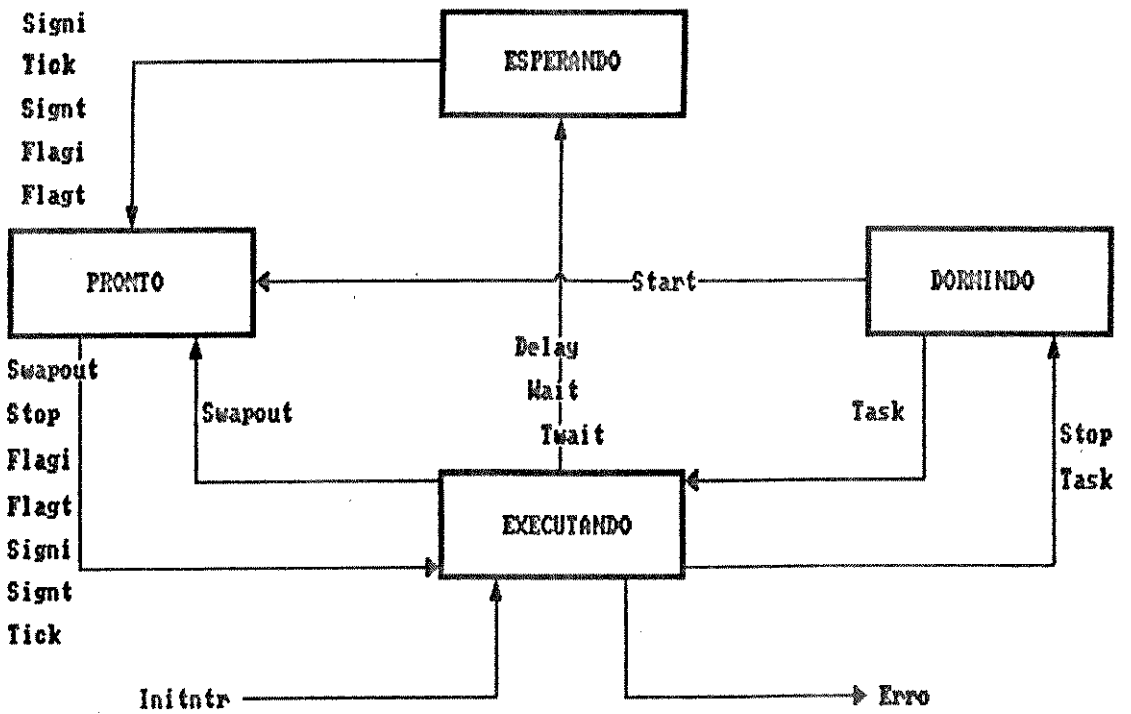

Figura 3.2: Estados das tarefas no NTR-80.

Uma tarefa no estado Esperando aguarda que expire algum prazo ou que algum evento ocorra para que ela possa mudar de estado.

Do estado anterior a tarefa vem para o estado Pronto, no qual a tarefa fica em uma das filas (seção 3.3.2.4 - B), aquela correspondente a sua prioridade, aguardando que o processador seja desocupado. Caso a tarefa possua prioridade superior a da tarefa de posse do processador (no estado Executando), ela substitui a segunda, a qual passa para o estado Pronto.

No diagrama vê-se também as funções do NTR-80 que são utilizadas pelas tarefas para as mudanças de estado. Explicar-se-á posteriormente o papel de tais funções.

3.3.2.3. Semáforos.

"Um semáforo é uma variável protegida, inteira, e não negativa, sobre a qual são definidas duas operações: P e V" ([21], [22]).

A operação P sobre um semáforo x tem o seguinte efeito:

```
IF x > 0  THEN
      x := x - 1
ELSE
      (espera na fila de x);
```

enquanto a operação V faz o "inverso":

```
IF (há tarefas na fila de x) THEN
      (libera uma das tarefas)
ELSE
      x := x + 1;
```

Estas operações são protegidas de forma a que somente uma de cada vez pode ser realizada sobre um semáforo x, seja P ou V.

O exemplo clássico para ilustrar a utilização de semáforos é o emprego dos mesmos no controle de acesso a recursos compartilhados: supõe-se que duas tarefas, completamente independentes, desejam imprimir relatórios utilizando uma mesma impressora. Óbviamente uma das duas terá de esperar; o qual poderia ser feito através do seguinte programa:

```
program One
      var  sema1: semáforo;

procedure T1;
      begin
            repeat
                  .
                  .
                  .
                  P(sema1);
                  imprimir relatório1;
                  V(sema1);
                  .
                  .
                  .
            forever;
      end;

procedure T2;
      begin
            repeat;
                  .
                  .
                  .
                  P(sema1);
                  imprimir relatório2;
                  V(sema1);
                  .
                  .
            forever;
      end;

      begin             (programa principal)
            inicia-semaf(sema1,1);
            cobegin
                  T1, T2;
            coend
      end.
```

Sendo que os comandos **cobegin** e **coend** indicam que os procedures **T1** e **T2** são executados concorrentemente. Como também é visto, ao semáforo **sema1** é atribuído um valor inicial (através de uma função, dado que é uma variável protegida).

O NTR-80 provê semáforos como descrito acima. O semáforo, como mostrado na figura abaixo, possui 3 "bytes", 1 para o contador (utilizado para registrar a ocorrência de operações **P** e **V** sobre o semáforo, ou seja, eventos) e 2 "bytes" para o "link" entre as tarefas que aguardam naquele semáforo a sinalização de eventos.

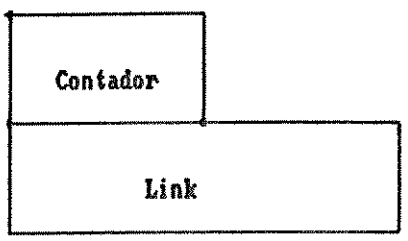

Figura 3.2: Semáforo no NTR-80.

3.3.2.4. Características Operacionais.

Algumas características operacionais do NTR-80 devem ser conhecidas pelo usuário de forma a possibilitar sua utilização. Aqui são mencionadas aquelas relacionadas com este trabalho.

A. BLOCO DE CONTROLE DE TAREFA.

Cada tarefa do sistema que não se encontra no estado Dormindo possui um Bloco de Controle de Tarefa (BCT):

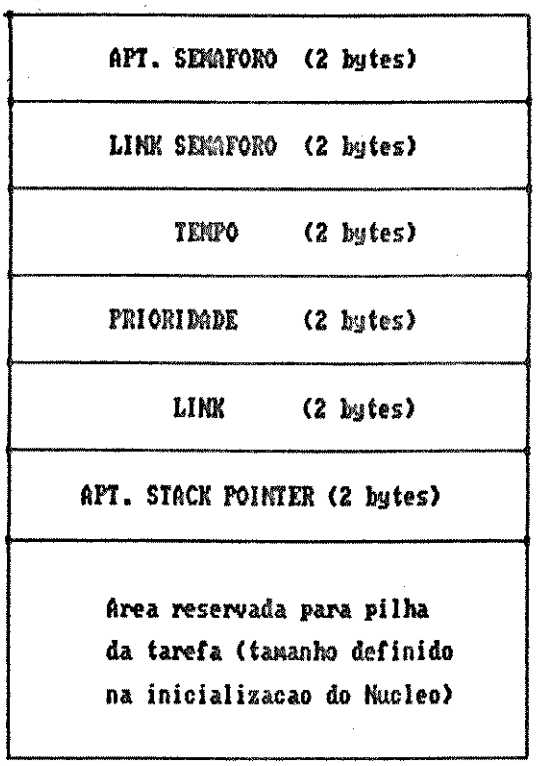

Figura 3.3: Bloco de controle de tarefa.

O BCT comporta, além de uma área reservada para a pilha da tarefa, uma série de campos que são utilizados para armazenar a prioridade da tarefa, o tempo durante o qual a mesma deve ser suspensa, "links" para filas de estado, etc... (maiores detalhes em [20]).

B. FILAS DO NÚCLEO.

No NTR-80 existem várias filas através das quais tarefas e estruturas de dados são organizadas. Obviamente o estado Executando não necessita de fila, nem o estado Dormindo, embora os BCT's disponíveis (define-se o número total deles na inicialização do sistema) estejam em uma fila para eventual utilização. Os

semáforos possuem também um campo de "link" através do qual as tarefas que esperam a ocorrência de eventos relacionados com o mesmo estão unidas. Existe uma fila para cada prioridade para as tarefas que estão no estado Pronto, facilitando assim o reescalonamento das tarefas. As filas são do tipo FIFO, com exceção da fila do estado Esperando, na qual as tarefas são inseridas na ordem crescente do tempo que as mesmas devem esperar.

C. INICIAÇÃO.

Para utilizar-se o NTR-80 uma ampla operação de iniciação deve ser realizada. Contudo estes aspectos operacionais são de relativa simplicidade. A iniciação deve abranger as estruturas de dados utilizadas pelo Núcleo, os semáforos e outras não descritas aqui (se interessarem ao usuário), dado que não utilizadas no trabalho.

3.3.2.5. Funções.

São descritas aqui as funções essenciais para este trabalho. Para um tratamento mais completo do tema veja [20].

INITNTR - Deve ser a primeira função a ser chamada pela rotina de iniciação. Cria a estrutura de dados do Núcleo e inicia os apontadores de fila da seguinte forma:

Corrente - aponta para o BCT atribuído à rotina de iniciação que passa a ser a primeira tarefa do Núcleo, com prioridade zero (mais alta).

Disponível - aponta para a fila de BCT's disponíveis.

Pronto(0) - aponta para NIL.

.

.

.

Pronto(n-1) - aponta para NIL.

Esperando - aponta para NIL.

INITSEM - Parâmetros: end_semáforo, valor. Inicia os semáforos, definindo o valor inicial e zerando o campo de "link" que, como foi dito, é utilizado para formar a fila relacionada com aquele semáforo. Inicia tanto semáforos binários (valor entre zero e um), como semáforos contadores ("qualquer" valor inteiro positivo).

START - Parâmetros: end_tarefa, prioridade. Com esta função transfere-se a tarefa do estado Dormindo para o estado Pronto. Como parâmetro passa-se a prioridade desejada para a tarefa e o seu endereço.

STOP - Esta função passa a tarefa que a chamou do estado Executando para o estado Dormindo, sendo que a tarefa da fila de mais alta prioridade do estado Pronto ocupa o processador.

WAIT - Parâmetro: end_semáforo. Esta função avalia o contador do semáforo indicado e suspende a tarefa corrente (que a chamou) desde que o valor do contador seja igual à zero. Tal acontecendo, a tarefa perde o processador, passando para a fila do semáforo; caso contrário, o contador é decrementado e a tarefa prossegue. Como vê-se, esta função equivale a operação P descrita na seção 3.3.1.3.

SIGNT - Parâmetro: end_semáforo. Através desta função sinaliza-se a ocorrência de um evento, permitindo então que uma tarefa da fila do semáforo indicado (se houver alguma) vá para a fila do estado Pronto que a sua prioridade indicar, ou mesmo que vá para o estado Executando, se a prioridade for superior a da tarefa corrente (escalonador preemptivo). Caso não haja nenhuma tarefa na fila do semáforo, o contador do mesmo é incrementado. Esta função equivaleria portanto, a operação V descrita na seção 3.3.1.3. Esta função é utilizada para sinalizar semáforos contadores (valores entre 0 e 255), e deve ser chamada quando o evento for sinalizado por uma tarefa do sistema. De resto, existem funções similares à função Signt que são utilizadas para semáforos binários e/ou quando o evento for sinalizado por uma rotina de tratamento de interrupção.

DELAY - Parâmetro: ticks. Esta função suspende a tarefa que estiver executando por um certo período de tempo. A tarefa de mais alta prioridade do estado Pronto ocupa o processador. Como parâmetro passa-se o número de ticks que a tarefa deve ser suspensa sendo que cada "tick" corresponde à 0.0164 seg. no CAMAÇARI.

Várias outras funções existentes no NTR-80, aqui não descritas, aumentam a eficiência e flexibilidade em sua utilização.

3.4 - Estruturação do Software.

3.4.1. Introdução.

Nesta seção descreve-se a estrutura do "software" e as tarefas implementadas.

Como foi visto na seção 3.2, em um microcomputador CAMAÇARI é implementada a simulação de um processo e em um outro ficam localizadas as funções de controle e estimação do processo mencionado. É mostrada na figura 3.4 a localização das diferentes tarefas nos dois processadores:

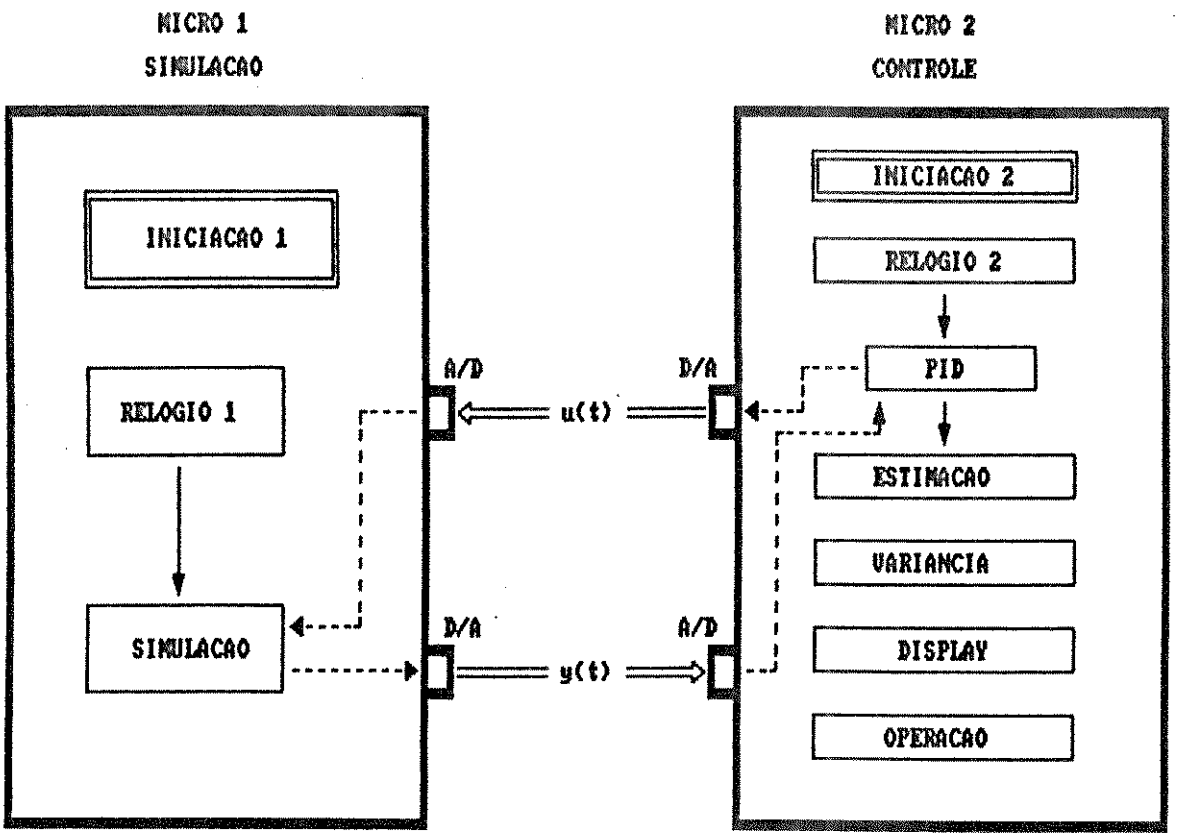

Figura 3.4: Distribuição das tarefas nos microcomputadores.

Assim, tem-se no microcomputador onde situa-se a simulação:

1. Tarefa de Iniciação 1, que define e inicia as variáveis globais, as estruturas de tempo real e, dispara as tarefas. Esta tarefa é processada somente no início da simulação.

2. Tarefa Relógio 1, a qual informa à tarefa Simulação que é tempo de alterar a saída do processo. Esta tarefa é processada uma vez a cada intervalo de discretização do processo simulado.

3. Tarefa Simulação, na qual efetivamente é feita a simulação do processo. Como a anterior ocupa o processador apenas uma vez a cada intervalo de discretização. Após ler o sinal de controle, calcular o valor da saída do processo e escreve-lo na saída D/A, esta tarefa espera ser reativada pela tarefa Relógio.

No microcomputador onde situa-se o controle:

1. Tarefa de Iniciação 2, com a mesma função da anterior.

2. Tarefa Relógio 2, a qual assinalará à tarefa PID o momento no qual esta deve fazer uma nova leitura da saída do processo.

3. Tarefa PID, encarregada de controlar o processo.

4. Tarefa Estimação/Alocação de Polos, a qual estima os parâmetros do processo e define os parâmetros do PID de acordo com o algoritmo de alocação de polos (seções 2.4 e 2.5). Esta tarefa está sincronizada com a tarefa PID, isto é, seu processamento está condicionado ao

processamento da tarefa PID.

5. Tarefa Variância, a qual encarrega-se de determinar os parâmetros do PID através de um compromisso entre as variâncias da entrada e da saída do processo (problema regulador). Dessemelhantemente das últimas três tarefas, que são processadas apenas uma vez a cada intervalo de amostragem, a tarefa Variância tem uma frequência de processamento arbitrária, sendo ativada pelo usuário.

6. Tarefa Display, cuja função é exibir na tela os valores de algumas variáveis globais, essenciais para o operador do sistema. Esta tarefa também possui uma frequência de processamento não fixa, a ser ajustada pelo usuário. Note-se entretanto, que o valor das variáveis é alterado apenas uma vez a cada intervalo de amostragem, não fazendo sentido portanto uma frequência de processamento muito alta para esta tarefa, dado que isto resultará apenas na repetição de valores.

7. Tarefa Operação, pela qual o operador poderá alterar o valor de variáveis do sistema. Acionada através de interrupção, a tarefa Operacão ocupa o processador quando o operador a ativa via teclado.

A seguir é detalhado o funcionamento de cada tarefa.

3.4.2. Tarefas Iniciação 1 e 2.

Apesar do emprego de uma interface NTR-80/PASCAL-Z que permite que o usuário programe em alto nível, o modo de utilização do NTR-80 permanece inalterado em sua essência. Por conseguinte, mostra-se necessária a existência de uma tarefa que realize uma ampla operação de iniciação que abranja as estruturas do Núcleo, as variáveis globais e as tarefas concorrentes. Detalhando, esta tarefa permite:

I. Definir-se (declarar-se) estruturas de dados, tais como canais de comunicação, semáforos, estruturas auxiliares, etc...:

- Semáforos: são definidos declarando-se um semáforo como um "array[1..2] of integer". Como em PASCAL-Z um inteiro é representado por 2 "bytes", tem-se 4 "bytes" para o semáforo, embora ele apenas utilize 3.

- Variáveis Globais: é necessário declarar-se as variáveis que serão comuns ao sistema (sistema aqui significa o conjunto dos programas concorrentes instalados em um microcomputador) tais como: saída do processo (PV), sinal de controle (VO), "set-point" (SP), etc...

II. Declarar-se as funções do NTR-80 como "PROCEDURES" externos e definir-se os seus parâmetros.

- "Procedure" INITNTR (nbct, tambct, nivpri, stack, roterr: integer); external; Apesar da função INITNTR original não possuir nenhum parâmetro, a de alto nível os possui por conveniência, simplificando a definição de uma série de valores:

nbct: informará o número de BCT's necessários ao NTR-80;
tambct: define o tamanho dos BCT's;
roterr: endereço da rotina de erro do Núcleo (nesta implementação não foi criada uma tal rotina, mas ela tem sua utilidade - ver [11]);
nivpri: número de níveis de prioridade desejados;
stack: define o início da área de dados do Núcleo que deve ser calculada da seguinte forma:

NBCT * TAMBCT + 2 * NIVPRI + 20.

- "Procedures" auxiliares devem também ser declaradas e elas possuem finalidades variadas:

a: "procedures" de interface - necessários para a "interface" entre o NTR-80 e o PASCAL-Z, são eles: START1, START2,...,STARTn (n é o número de tarefas).

b: "procedures" de Comunicação - servem para comunicação com o processo (entrada A/D e saída D/A) e são: ESCDA, LEAD.

c: "procedures" Genéricas - neste caso se encaixam duas "procedures" auxiliares: a **Igs**, que possibilita I/O por interrupção (a tarefa Operação trabalha desta forma) e a **Tabela**, que fornece os endereços das funções do Núcleo.

III. Iniciar-se o NTR-80, os canais, semáforos, etc... Neste ponto deve-se iniciar também as variáveis globais mencionadas anteriormente com os valores que julga-se adequados.

IV. "Disparar-se" as tarefas.

3.4.3. Tarefas Relógio 1 e 2.

A utilização de semáforos pelas tarefas introduz a impossibilidade de garantir-se que uma tarefa levará sempre o mesmo intervalo de tempo para ser processada (ver seção 3.5.3). Por conseguinte, o uso direto por estas tarefas da função DELAY do NTR-80 não permite uma frequência rígida de execução. Assim, há a necessidade de uma tarefa independente que forneça a base de tempo para as tarefas que devem ser processadas em intervalos precisos. Para isto foi criada a tarefa Relógio, que possui a particularidade de não realizar nenhuma chamada da função WAIT do NTR-80; não incorrendo assim em nenhum atraso aleatório.

Como pode ser visto no anexo A, a tarefa Relógio é implementada de forma extremamente simples. Sua iniciação consiste apenas na definição do número de Ticks que correspondem ao intervalo de amostragem desejado, e o "loop" infinito repete apenas dois comandos. Tem-se então:

1. Iniciação, onde o tempo de retardo (delay) é definido. Então aqui define-se o intervalo de amostragem, no caso do microcomputador de controle, e o intervalo no qual o processo é discretizado, no caso do microcomputador de simulação.

2. Sinalização (através da função SIGNT do NTR-80) do semáforo ao qual associou-se o PID ou a simulação.

3. Chamada da função DELAY do NTR-80. Como parâmetro passa-se o tempo durante o qual a tarefa estará no estado Esperando do NTR-80. Ao ser reativada, a tarefa vai para o passo 2.

3.4.4. Tarefa Simulação.

3.4.4.1. Corpo da Tarefa.

A tarefa de Simulação possui, como a maioria das tarefas do sistema, duas partes distintas. Uma se incumbe da iniciação de variáveis e é processada apenas uma vez. A outra é repetida continuamente em um "loop" infinito, em uma frequência determinada. Então, tem-se:

1. Iniciação do vetor dos parâmetros do processo, do vetor das medidas do processo (com valores iguais à zero), e dos parâmetros do ruído branco (média, desvio padrão e a "semente" do gerador).

2. Espera (função WAIT do NTR-80) em um semáforo até que a tarefa Relógio sinalize o mesmo, indicando com isto que deve ser alterado o valor da saída do processo (transcorreu o intervalo de discretização).

3. Geração de um sinal de ruído branco, através da utilização do gerador que está descrito na seção 3.4.4.2.

4. Geração de y(t) pelo emprego da equação 2.7:

$$y(t) = \theta^T(t) \quad \varphi(t) + e(t).$$

5. Escrita no canal 0, da saída D/A, do valor de y(t).

6. Leitura do valor de u(t) no canal 0 da entrada A/D.

7. Atualização do vetor de medidas. Volta para 2.

3.3.4.2. Gerador de Ruído Branco.

O apêndice B contém uma implementação do Gerador de Ruído Branco descrito na seção 2.2.2. Esta implementação gera números aleatórios uniformemente distribuídos entre (0, 1) e ela é utilizada para obter números com distribuição gaussiana.

Para gerar uma variável normalmente distribuída utilizam-se doze valores (ver [11], para outros métodos), uniformemente distribuídos entre (0, 1), segundo a equação

$$x = [\sum_{i=1}^{12} N_i - 6\ ]\ *\ S + M, \quad [10];$$

onde S e M são o desvio padrão e a média requeridos respectivamente.

Na "procedure" NORMAL (anexo A) está implementada a equação acima.

3.4.5. Tarefa PID.

A implementação do algoritmo PID deve ser feita de tal forma que a equação 2.15 esteja o mais correta possível, em termos de tempo real. Ou seja, para executar-se esta equação três passos devem ser efetuados, no menor intervalo de tempo possível:

a. ler y(t) na porta A/D.

b. calcular u(t).

c. escrever u(t) na porta D/A.

Assim, a implementação consiste dos seguintes passos:

1. Iniciação de parâmetros e variáveis. Esta iniciação é feita somente uma vez.

2. Realização de uma operação **P** (ver seção 3.3.2.3) no semáforo **acess0**. Isto causa uma espera que dura até que a tarefa Relógio sinalize o mesmo semáforo, indicando que transcorreu o intervalo de amostragem desde a última sinalização (ou seja, a tarefa Relógio realiza uma operação **V** sobre o semáforo **acess0**).

3. Leitura do valor de y(t) na porta A/D.

4. Definição de u(t) pelo operador, caso o PID esteja na opção **manual** (o operador tem completa liberdade dentro da faixa permitida).

5. Cálculo de u(t) segundo a equação 2.15 se, ao contrário, o PID estiver na opção automático:

$$u(t) = u(t-T) + K_p\, T/Ti\; r(t) - g_0\, y(t) - g_1\, y(t-T) - g_2\, y(t-2T).$$

6. Adição (opcional) ao u(t) definido em 3 ou 4, de um PRBS (Pseudo Random Binary Signal), cujo módulo é estabelecido pelo operador.

7. Restrição do valor de u(t) calculado pelos passos anteriores, caso sejam excedidos os limites permitidos (VO_{max} e VO_{min} - fixados pelo operador), a estes mesmos limites.

8. Escrita na porta D/A do valor de u(t).

9. Acesso à base de dados, sempre através de semáforos. Neste acesso, além da transmissão dos valores de u(t) e y(t), ocorre a leitura dos valores de vários parâmetros, entre os quais, K_p, T_i e T_d (que serão utilizados no próximo passo). Estes últimos podem ser definidos de três formas: pelo operador, segundo sua experiência; pela alocação de polos buscando um comportamento de 2.ª ordem em malha fechada (seguimento de referência); ou através de um compromisso entre o seguimento de referência e as variâncias máximas permitidas para o elemento atuador e o sinal de saída. Esta última opção necessita, na presente implementação, da intervenção do operador.

10. Armazenamento dos valores y(t), y(t-T) e u(t), os quais serão utilizados no próximo passo, na área de dados local.

11. Atualização do valor do "set-point": cálculo de r(t+1).

12. Geração do PRBS para o instante (t+1), quando será utilizado no passo 6.

13. Sinalização do semáforo de sincronização, em cuja fila a tarefa Estimação/Alocação de Polos aguarda permissão para ser processada.

3.4.6. Tarefa Estimação/Alocação de Polos.

Neste item são descritos os aspectos relacionados com a implementação do algoritmo dos Mínimos Quadrados Estendido.

A tarefa Estimação/Alocação de Polos, na qual o algoritmo dos Mínimos Quadrados Estendido e o cálculo dos parâmetros do PID por alocação de polos estão inseridos, tem, como todas as outras tarefas, uma iniciação e um "loop" infinito.

Na iniciação da tarefa foi inserida a iniciação do algoritmo recursivo de estimação.

Os passos que constituem esta tarefa são:

1. Iniciação, onde as especificações do usuário quanto ao comportamento em malha fechada do processo são inseridas, o valor inicial do fator de esquecimento é estabelecido, a matriz P é iniciada, como também os vetores $\hat{\theta}$, φ e L (seção 2.4). Note-se que o vetor $\hat{\theta}(t)$ não é iniciado com zeros - como usual - devido a ser ele utilizado no cálculo dos parâmetros do PID desde o início. Outros parâmetros auxiliares também são iniciados.

2. Espera no semáforo de sincronização. Como descrito anteriormente, a cada amostragem do processo a tarefa PID sinaliza o semáforo no qual a tarefa Estimação está esperando.

3. Acesso à área de dados. Todo acesso à área de dados global é feito, como explicado anteriormente, através de semáforo; mantendo assim um padrão de integridade de dados.

4. Desativação da estimação, com o intermédio de um parâmetro denominado **Onoff** (de tipo caracter), aliviando assim o processador consideravelmente. Se o operador faz **Onoff** ter valor igual à n (no), a tarefa vai para o passo 2.

5. Cálculo do erro da predição:

$$\text{erro} = y(t) - \hat{\theta}^T(t-1)\,\varphi(t);$$

onde $\varphi^T(t) = [-y(t-1)\ -y(t-2)\ u(t-1)\ u(t-2)\ \hat{e}(t-1)\ \hat{e}(t-2)]$

$\hat{\theta}^T(t-1) = [\hat{a}_1(t-1)\ \hat{a}_2(t-1)\ \hat{b}_0(t-1)\ \hat{b}_1(t-1)\ \hat{c}_1(t-1)\ \hat{c}_2(t-1)]$.

6. Atualização das estimativas dos parâmetros:

$$\hat{\theta}(t) = \hat{\theta}(t-1) + L(t) * \text{erro}.$$

7. Cálculo do resíduo:

$$\text{resíduo} = y(t) - \hat{\theta}(t) * \varphi^T(t).$$

8. Atualização do vetor φ [obter $\varphi(t+1)$]:

$$\begin{aligned}
\varphi[2] &= \varphi[1] \\
\varphi[1] &= -y(t) \\
\varphi[4] &= \varphi[3] \\
\varphi[3] &= u(t) \\
\varphi[6] &= \varphi[5] \\
\varphi[5] &= \text{resíduo}.
\end{aligned}$$

9. Atualização do vetor de ganho L e da matriz P. Desta forma calcula-se L(t+1) e P(t+1) que serão usados no próximo passo. Esta atualização é feita segundo [15], e armazena a matriz triangular superior **U** e a matriz diagonal **D** na própria matriz P ($P = U^TDU$).

10. Computação do traço ponderado da matriz P:

$$T_P = \sum_{i=1}^{6} \frac{P[i,\ i]}{\hat{\theta}^2[i]} .$$

11. Cálculo da variância do resíduo para um certo número de amostras (janela).

12. Alteração do valor de lambda (através do parâmetro **lambdo**, o operador define o valor final de lambda, ao qual lambda tenderá exponencialmente), segundo a equação ([14]):

lambda = 0.99 * lambda + lambdo.

13. Cálculo de K_P, T_i e T_d segundo a estratégia de alocação de polos (seção 2.5).

14. Transmissão (eventual) de y(t), u(t), r(t), $T_P(t)$ e do vetor $\hat{\theta}(t)$, para um microcomputador XT2002 para processamento gráfico. A transmissão de dados é feita "on-line", mas a manipulação dos mesmos é feita "off-line".

15. Acesso à área global de dados para armazenar os valores estimados dos parâmetros, a variância do resíduo (utilizada pela tarefa Variância) e outros dados. Novamente o acesso à área global de dados é feito com o intermédio de semáforos, garantindo a exclusão mútua. Isto feito, volve ao passo 2.

3.4.7. Tarefa Variância.

Como foi visto na seção 2.6, o cálculo das variâncias dos sinais de saída e de controle utiliza os coeficientes estimados dos polinômios A, B e $C(z^{-1})$. Isto implica em que esta tarefa deve ser acionada somente quando julgar-se que a estimação dos parâmetros está correta.

A tarefa é constituída, a grosso modo, das seguintes partes:

1. Iniciação - onde variáveis específicas são iniciadas.

2. Acesso à área global de dados, feito utilizando-se semáforo para garantir exclusão mútua. Neste acesso são feitas cópias dos parâmetros estimados do processo, da variância estimada do ruído, de k_1 e k_2 (parâmetros que relacionam os coeficientes γ_1 e γ_2 através da equação 2.34), do intervalo de amostragem e de um parâmetro que indica se o resto da tarefa deve ser processada ou não. Se este último parâmetro indicar uma negativa vai-se para o passo 5.

3. Cálculo de K_p, T_i, T_d, a variância de y e de u.

4. Escrita dos resultados na área global de dados utilizando-se semáforo. Estes resultados serão exibidos no vídeo pela tarefa Display. O operador pode, através do console, alterar o valor de γ_1 e assim ir "procurando" as variâncias de y e u que lhe convém. A esses valores de variâncias corresponderão determinados valores para K_p, T_i e T_d, os quais o operador enviará para a tarefa PID.

5. Chamada do procedimento DELAY do NTR-80 que adormece a tarefa pelo tempo desejado. Quando novamente de posse do processador a tarefa vai para 2.

3.4.8. Tarefa Display.

3.4.8.1. Introdução.

A função desta tarefa é unicamente a de exibir na tela os valores "atuais" de parâmetros e variáveis de tal modo que o operador possa fazer escolhas, observar o comportamento do processo, a estimação dos parâmetros, etc...

3.4.8.2. Implementação.

Nesta implementação foram utilizados dois vídeos E-800 fabricados pela T.D.A. - Indústria de Produtos Eletrônicos Ltda. Para seu manejo, ver [23].

Como podem existir outras tarefas que utilizam o vídeo (e efetivamente há uma outra, no nosso caso), diz-se que o mesmo é um recurso compartilhado. Se existem mais de uma tarefa querendo acessar apenas um recurso, então utilizam-se semáforos com a finalidade de exclusão mútua.

As tarefas que utilizam o vídeo (tarefa Display e tarefa Operação) o consideram como uma matriz de caracteres (no caso de um vídeo TDA, uma matriz de dimensão 24 * 80) e assim, para facilitar a manipulação do mesmo dividem-no em duas regiões: uma trabalha com um certo número de linhas e a outra com as restantes. Nada impede contudo, que esta divisão seja mais sofisticada, desde que se adote a seguinte estratégia: o cursor é posicionado em qualquer ponto da matriz mediante o emprego de caracteres de controle adequados; a tela é mantida estática pela não utilização dos caracteres de controle LF (line feed) e CR (carriage return), os quais adicionam uma nova linha a matriz (tela) e posicionam o cursor no início da linha,

respectivamente.

Enfim, esta tarefa consiste dos seguintes passos:

1. Iniciação, na qual a tarefa perfaz a identificação das regiões da tela, escrita de mensagens, exibição de valores iniciais (ou "default") de algumas variáveis, fornecendo um resultado semelhante ao da figura 3.5.

Para isso deve a tarefa obter acesso exclusivo ao vídeo (recurso compartilhado) através de um semáforo enquanto um outro garantir-lhe-á exclusão mútua no que se refere à área global de dados.

2. Leitura de valores na área global de dados e escrita na tela, utilizando os semáforos acima mencionados. Alguns valores são escritos condicionalmente, como pode ser visto no anexo A, visto que se as tarefas que os geram estão desativadas, não interessa exibi-los na tela. Os valores lidos são alterados por outras tarefas, esta apenas os lê.

3. Perda do processador através da utilização da função DELAY do NTR-80. Quando acordar, a tarefa irá para o passo 2. Como foi mencionado na seção 3.4.1, caso as variáveis exibidas na tela não tenham seu valor alterado mais do que uma vez por intervalo de amostragem, não é necessário que esta tarefa seja processada várias vezes por intervalo de amostragem; isto provocará redundância e má utilização do processador.

PV	VO	SP	AM	L/D	PARest
32.345	54.898	40.000	a	y	n

K_p	T_i	Td	TimeAmost	Lambdo	ParPID
3.156	30.777	10.00	6.0	0.00995	o

Theta[1]	Theta[2]	Theta[3]	Theta[4]	Theta[5]	Theta[6]

Contador	Kpest	$T_i$est	Tdest	Traço

PRBS	Varu	Vary	Varr	Sigma1	Transm.
5.0					

$K_p$var	$T_i$var	Tdvar

As opções são: VO(0), SP(1), K_p(2), T_i(3), Td(4), TA(5), VO_{max}(6), Lambdo(7), Sigma1(8), |PRBS|(9), Transm(10), MA(11), LD(12), PE(13), ParPID(14).

Figura 3.5: Utilização de um vídeo como uma matriz estática.

3.4.9. Tarefa Operação.

3.4.9.1. Introdução.

A tarefa Operação é encarregada da interface com o operador. Esta tarefa pode ter um alto grau de sofisticação, no entanto isto acarretará um dispêndio considerável de RAM.

Assim, optou-se por uma tarefa simples, mas que cumpre os objetivos desejados. Não obstante, chegou-se a construir uma versão com vários menus reentrantes, a qual foi abandonada pela razão acima mencionada.

3.4.9.2. Implementação.

Esta tarefa escreve no vídeo, lê no console e escreve na área global de dados. Portanto, ela deverá utilizar dois semáforos: um que lhe dê acesso ao vídeo e outro que lhe permita alterar a área global de dados.

A maneira pela qual o processador é alocado a esta tarefa é diferente das demais. Como as operações de entrada e saída do NTR-80 são através de interrupção, a tarefa Operação ao executar um comando **read** espera até que a tecla RETURN seja apertada (ou seja, o operador digita sua "mensagem" e logo após comprime a tecla RETURN). Enquanto a tecla RETURN não for ativada a tarefa Operação fica no estado Dormindo do NTR-80, não utilizando a CPU. Ao acionamento da tecla RETURN, por interrupção, a tarefa passa ao estado Pronto e volta a concorrer à CPU.

Um detalhe importante que ocorre nesta solução é que o comando **read** da tarefa Operação não pode ser circundado por semáforos, ou seja, não deve haver a seguinte sequência de comandos:

```
Wait(video);
.
.
.
readln(x);
.
.
.
Signt(video);
```

pois desta forma a tarefa Display ficaria aguardando no semáforo vídeo até que o operador comprimisse a tecla RETURN. Tal acontecimento é indesejável obviamente; a tarefa Display deve continuar a atualizar os dados na tela (vídeo) mesmo que o operador não intervenha durante um bom espaço de tempo.

A questão enfim, é que o comando read (ou readln) ecoa na tela os caracteres que foram pressionados antes da tecla RETURN e isto pode acarretar o seguinte problema: suponha que a tarefa Display está escrevendo na tela uma sequência de valores e é interrompida antes de exibi-los todos. Se o operador digita alguns caracteres e pressiona a tecla RETURN pode ser que a tarefa Operação consiga o domínio do processador antes da tarefa Display, e assim ecoará a entrada do operador no local errado.

A solução é simples: dando-se à tarefa Display prioridade maior que à tarefa Operação, quando a primeira perder o processador o recuperará antes da última, pois ficará em uma fila no estado Pronto de prioridade superior. Então a tarefa Operação só entrará na posse da CPU após que a tarefa Display tenha percorrido toda a área da tela que lhe cabe.

Os passos abaixo compõem esta tarefa:

1. Iniciação de algumas variáveis.

2. Escrita no vídeo de uma mensagem ao operador: 'SELECIONE UMA OPÇÃO'; e subsequente espera, no estado Dormindo do NTR-80, que o operador digite sua instrução.

3. Leitura da opção escolhida pelo operador, que acarretará a escrita no vídeo da mensagem: 'DEFINA UM VALOR'. Isto feito, a tarefa fica a aguardar a ação do operador novamente no estado Dormindo, ou seja, perde o processador.

4. Leitura do valor desejado pelo operador para a variável selecionada anteriormente (no passo 2). Para tal acontecer, o operador deve digitar o valor, e a tecla RETURN. Então, a tarefa retorna eventualmente ao processador, agora para efetivamente alterar a variável indicada pelo operador, tanto na área global de dados (memória), como no vídeo, se for o caso. Isto cumprido a tarefa vai para 2.

3.5 - Características Operacionais da Implementação.

3.5.1. Introdução.

Nesta seção apresentaremos algumas características operacionais da implementação, umas das quais terão um aspecto teórico, enquanto que outras possuirão mais a qualidade de "guia do usuário".

3.5.2. Atraso de transporte adicional.

Como foi visto no Capítulo 2, todo o desenvolvimento teórico deste trabalho foi voltado para o controle de processos sem atraso de transporte, porém foi encontrado que, com um processador de velocidade finita, não é possível implementar-se tais processos com precisão absoluta.

Tal fato deriva de que se é desejado que uma simulação em tempo real de um processo físico represente fielmente tal processo, sua resposta no instante (t+T) a um estímulo aplicado no instante (t) deve ser exatamente a mesma que o processo real apresentaria sob idêntica excitação.

Apesar da afirmação anterior parecer ser (e efetivamente é) completamente óbvia, ela tem uma importante consequência. Como foi visto na seção 3.4.4, a única "comunicação" da tarefa Simulação com o mundo externo é através das portas A/D [lendo o sinal VO ou u(t)] e D/A [escrevendo o sinal PV ou y(t)]; contudo, esta "comunicação" não é feita continuamente, mas em intervalos discretos. Ora se é desejado que a tarefa sinta uma mudança no sinal VO [(u(t)], instantâneamente, deve-se fazer com que ela acesse a porta A/D com a maior frequência possível (consequentemente acessará também a porta D/A - ver anexo A).

Esta frequência tem um limite, contudo. Este limite é imposto pelo tempo gasto pela CPU para processar a tarefa, sendo que, no caso da tarefa Simulação, o tempo medido ficou por volta de 170 mseg.

Portanto, incluindo-se uma margem de segurança, tem-se uma tarefa de simulação que será processada a cada 200 mseg. e uma tarefa de controle (PID) que amostrará em intervalos de 6.0 seg. (devido à lentidão da transmissão de dados entre o XT2002 [PC] e o CAMAÇARI - em torno de 1.8 seg. - e também ao tempo de CPU gasto pela tarefa Estimação/Alocação de Polos - por volta de 3.5 seg.). Note-se que, como mencionado anteriormente, a tarefa PID e a Estimação/Alocação de Polos apesar de serem tarefas independentes, estão sincronizadas, isto é, ativado o "flag" adequado, a segunda será processada assim que a primeira abandonar o processador.

Por conseguinte, o fato do sinal de controle não ser lido pela simulação do processo no mesmo instante em que é aplicado (na porta D/A do microcomputador onde está sediada a tarefa Simulação), traduz-se em um atraso de transporte de valor igual à taxa de discretização da tarefa Simulação, como é ilustrado a seguir.

Na análise seguinte assume-se que, na tarefa PID, o intervalo de tempo gasto pelo processador entre a leitura de y(t) na porta A/D e a escrita de u(t) na porta D/A é desprezível. Também assume-se que, na tarefa Simulação, o intervalo de tempo entre o acesso das portas A/D e D/A é nulo.

O esquema abaixo mostra como, nos respectivos microcomputadores, as tarefas Simulação e PID comunicam entre si, se forem processadas com a mesma frequência. As duas tarefas, óbviamente, não são sincronizadas.

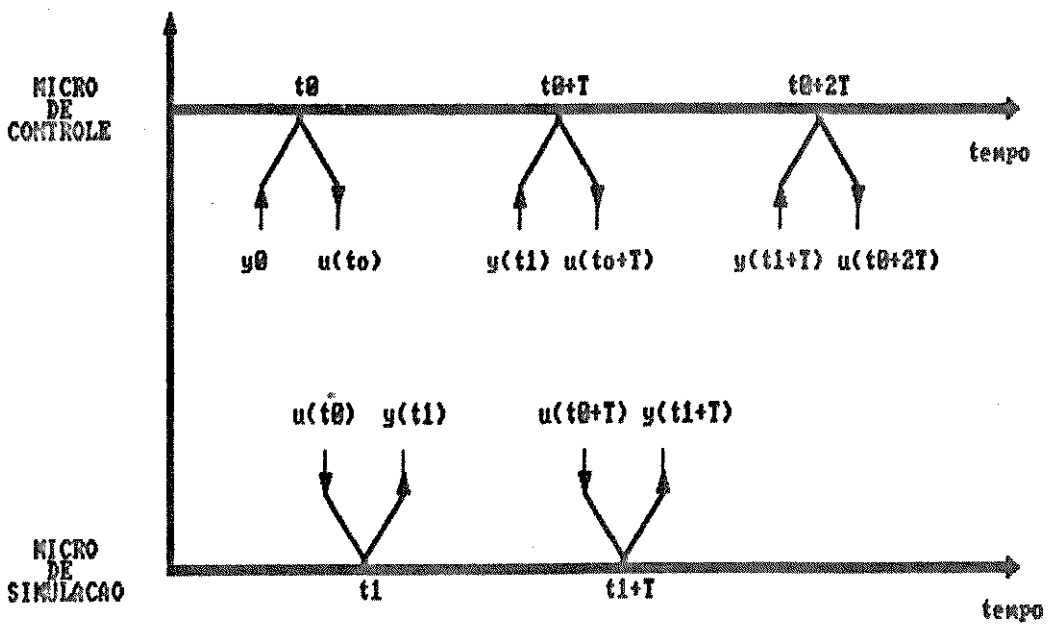

Figura 3.6: **Atraso de transporte igual a um intervalo de amostragem.**

a) em to a tarefa PID lê o sinal yo, calcula e aplica o sinal de controle u(to).

b) em t_1 a tarefa Simulação lê o sinal u(to) aplicado em to e escreve y(t_1) - o qual é calculado utilizando-se a fórmula 2.6, ou seja, não utiliza u(to).

c) em to+T a tarefa PID lê o sinal y(t_1), calcula e aplica o sinal de controle u(to+T).

d) em t_1+T a tarefa Simulação lê o sinal u(to+T) aplicado em to+T, calcula e escreve y(t_1+T), o qual é calculado utilizando u(to).

e) em to+2T a tarefa PID lê o sinal y(ti+T) e aplica o sinal de controle u(to+2T).

Então, no instante to+2T, a tarefa PID está lendo o valor da saída do processo que deveria ter lido no instante to+T - dado que a saída é função de u(to) - caracterizando-se assim um processo com um atraso de transporte igual a um intervalo de amostragem.

O esquema seguinte ilustra a sequência que ocorrerá quando a tarefa Simulação for discretizada com uma frequência duas vezes maior do que a tarefa PID.

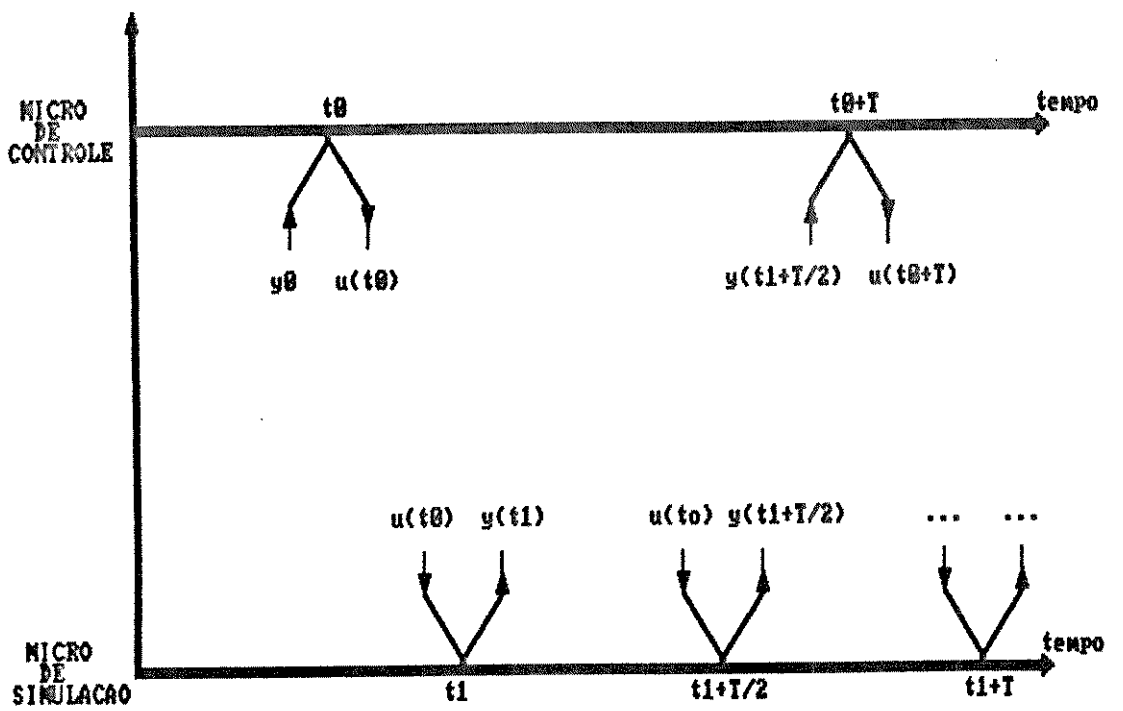

Figura 3.7: **Atraso de transporte igual a 1/2 intervalo de amostragem.**

a) em to a tarefa PID lê o sinal yo e aplica um sinal de controle u(to).

b) em t_1 a tarefa Simulação lê o sinal u(to) aplicado em to e escreve y(t_1), o qual não utiliza o sinal u(to) - equação 2.6.

c) em t_1+T/2 a tarefa simulação lê o mesmo valor u(to) e aplica y(t_1+T/2) - que é função de u(to).

d) em to+T a tarefa PID lê o sinal y(t_1+T/2) e aplica o sinal de controle u(to+T).

O valor de y(t_1+T/2) lido no instante to+T pela tarefa PID corresponde ao valor que seria lido na saída do processo real equivalente no instante to+T/2 se a ele fosse aplicado o mesmo sinal de controle no instante to. Ou seja, tem-se um processo simulado que apresenta um atraso de transporte igual a T/2.

Pode-se concluir então que se a tarefa Simulação for discretizada em uma taxa de 500 mseg. será estimado um processo com atraso de transporte de 500 mseg. e assim sucessivamente.

Como a taxa máxima de processamento da tarefa Simulação é 200 mseg., pode-se concluir que o processo simulado se comportará como se possuísse um atraso de transporte desta magnitude. Em consequência, o processo estimado pode ser representado pelas equações A.21,...,A.26 (apêndice A). Note-se que pequenas imprecisões são possíveis dado que a leitura e escrita nas portas A/D e D/A não são simultâneas, como foi assumido.

Se deseja-se eliminar o "atraso de transporte" pode-se utilizar o esquema da figura 3.6 (atraso igual a um intervalo de amostragem), e fornecer ao estimador os valores de y(t) e u(t-T) - ao

invés de y(t) e u(t) - de modo que possam ser estimados corretamente os parâmetros.

Note-se que quando o atraso de transporte é igual a um intervalo de amostragem as equações A.23 e A.24 reduzem-se às equações A.16 e A.17 respectivamente, enquanto que a equação A.22 reduz-se à zero.

Na seção 4.6 são feitos alguns ensaios que permitem avaliar a influência de tal "atraso de transporte".

3.5.3. Estrutura da tarefa PID. Localização da região crítica.

A equação 2.15, implementada na tarefa PID, faz duas exigências em termos de tempo real:

1. deve ser processada em intervalos precisos de tempo;
2. deve ser processada o mais rápido possível.

Quanto ao segundo item, pouco pode ser feito, dado que a velocidade limite é a velocidade do processador (Z-80, neste caso), mas é importante notar que a equação fornece o valor de u(t) como uma função de y(t), ou seja, assume que o tempo gasto no cálculo de u(t) é desprezível. Para tanto, no início da tarefa procede-se ao cálculo de u(t) e após isto são realizadas as operações que preparam o cálculo de u(t+1) no próximo instante de amostragem.

O primeiro item é responsável pela existência da tarefa Relógio, bem como pela exigência de uma organização dos passos da tarefa PID que não comprometa o desempenho da tarefa, como será explicado a seguir:

A tarefa PID possui um tempo de processamento variável, por duas razões:

i. existem ramificações na tarefa (caminhos diversos); contudo estas variações são relativamente pequenas.

ii. ela possui uma região crítica (região protegida por semáforo). Assim, apesar da tarefa PID desalojar uma outra tarefa do processador por conta de possuir prioridade superior, se a tarefa desalojada perder o processador enquanto estiver sendo percorrida uma região crítica protegida por um semáforo compartilhado com a tarefa PID, esta última perderá o processador ao chamar a função WAIT do NTR-80 (na tentativa de penetrar em sua própria região crítica), só o recuperando quando a outra processar toda sua região crítica (e então sinalizar o semáforo compartilhado - o que acarretará a devolução do processador para a tarefa PID). Ora, este tempo é aleatório, só podendo-se estabelecer o pior caso, acarretando assim a variabilidade do tempo de processamento da tarefa PID.

Então, a tarefa PID possui um tempo de processamento variável (podendo contudo determinar-se o tempo máximo). Assim sendo, utilizar a função DELAY do NTR-80 dentro da tarefa PID não fornecerá uma taxa de amostragem precisa, dado que a soma do tempo de processamento da tarefa PID com o tempo que ela passará no estado Dormindo (que é conhecido - parâmetro da função DELAY) é variável.

A solução encontrada foi a utilização da tarefa Relógio, a qual garante que em intervalos precisos de tempo à tarefa PID será fornecido o processador. É o quanto basta para assegurar que a equação 2.15 seja processada regularmente (em intervalos exatos de tempo). Deve-se cuidar cuidar apenas de que a região crítica (que possui tempo de processamento variável) não situe-se entre o ponto onde a tarefa PID reave o processador e a implementação da equação

2.15. Então, como pode ser visto no anexo A, a tarefa PID reave o processador no início do "loop" infinito [Wait(acess0)], e a equação 2.15 é implementada logo a seguir. A região crítica da tarefa está posterior à escrita do sinal de controle na porta D/A.

3.5.4. Interface com o operador.

Após carregada a implementação no microcomputador CAMAÇARI, o comando de execução da aplicação fará com que a tarefa Iniciação seja processada e então todas as demais tarefas sejam ativadas.

Durante a operação da implementação, o vídeo TDA apresentará a aparência mostrada na figura 3.8 (os valores exibidos servem apenas para ilustrar melhor o aspecto da tela).

O significado de cada campo é o seguinte:

PV - sinal de saída do processo, y(t).

VO - sinal do controlador (entrada do processo), u(t).

SP - sinal de referência, r(t).

AM - chave seletora de PID automático/manual. Se em automático o algoritmo PID calcula a saída do controlador, em manual o operador define o valor de u(t).

L/D - se em "y" (yes), a tarefa Estimação/Alocação de Polos é ativada.

PV	VO	SP	AM	L/D	PARest
32.345	54.898	40.000	a	y	n

K_p	Ti	Td	TimeAmost	Lambdo	ParPID
3.156	30.777	10.00	6.0	0.00995	o

Theta[1]	Theta[2]	Theta[3]	Theta[4]	Theta[5]	Theta[6]
-1.6016	0.6376	0.0194	0.0166	-1.005	0.456

Contador	Kpest	Tiest	Tdest	Traço
88	3.124	32.458	12.335	0.0033

PRBS	Varu	Vary	Varr	Sigma1	Transm.
5.0					150.0

$K_p$var	Tivar	Tdvar

As opções são: VO(0), SP(1), K_p(2), Ti(3), Td(4), TA(5), VO_{max}(6), Lambdo(7), Sigma1(8), |PRBS|(9), Transm(10), MA(11), LD(12), PE(13), ParPID(14).

Figura 3.8: Aspecto da tela durante a operação do controlador.

Parest - este parâmetro deve ser feito igual a "y" (yes) quando pretender-se ajustar os parâmetros do PID através da análise das variâncias do sinal de saída e controle do processo. Estes cálculos dependem da convergência de todos os elementos do vetor $\hat{\theta}(t)$. Ao ser acionado, os valores correspondentes aos campos VarU, VarY, VarR, SIGMA1, K_pVar, T_iVar e T_dVar começam a ser exibidos na tela.

K_p - parâmetro do algoritmo PID, definido pelo operador (opção manual).

T_i - idem.

T_d - idem.

TimeAmost - taxa de amostragem do controlador. Ao ser alterada, deve-se alterar também a tarefa Relógio.

Lambdo - define o valor final do parâmetro **lambda** (fator de esquecimento), através da equação:

$$lambda = 0.99 * lambda + lambdo.$$

ParPID - através deste parâmetro o operador define se o algoritmo PID utilizará os parâmetros K_p, T_i e T_d indicados pelo operador ("o"), pela tarefa Estimação/Alocação de Polos ("e") ou pela tarefa Variância ("v").

Theta[1]..

.....

Theta[6] - estes campos mostram o andamento da estimação, e são atualizados quando o campo L/D indica "y".

Contador - informa há quantos períodos de amostragem o estimador está acionado.

Kpest - parâmetro do PID conforme calculado pela tarefa Estimação/Alocação de Polos.

Tiest - idem.

Tdest - idem.

Traço - indica o valor atual do traço da matriz de covariância.

|PRBS| - indica o valor do módulo do PRBS adicionado à u(t).

Varu - variância do sinal de controle calculado pela tarefa Variância.

Vary - variância do sinal de saída do processo calculada pela tarefa Variância.

Varr - variância do ruído (residual fornecido pela tarefa Estimação/Alocação de Polos) calculada pela tarefa Variância.

Sigma1 - através desta opção, o operador altera o polinômio $\gamma(z^{-1})$ de forma a obter as variâncias que acha adequadas para y(t) e u(t).

Transm - o operador escolhe em que instante de amostragem (indicado pelo campo Contador) a transmissão de dados entre o CAMAÇARI e o PC iniciará.

Kpvar - parâmetro do PID resultante da alteração pelo operador do valor de Sigma1 inicialmente calculado pela tarefa

Estimação/Alocação de Polos.

Tivar - idem.

Tdvar - idem.

Para selecionar uma opção o operador deve pressionar o inteiro correspondente e então a tecla RETURN - então, para alterar K_p, o inteiro 2. A opção 0 (VO) permite ao operador definir o valor da saída do controlador quando o PID estiver na opção manual. Através da opção VO_{max} (6) o operador pode escolher o valor máximo do módulo do sinal de controle. A opção 5 (TA) refere-se ao campo TimeAmost. A opção 11 (MA) ao campo AM. A opção 12 (LD) relaciona-se ao campo L/D e a opção 13 (PE) ao campo PARest.

Note-se que as opções 11 à 14 só aceitam caracteres alfabéticos minúsculos.

Ao acionar da implementação, muitos campos da tela serão preenchidos com os respectivos valores iniciais (ver listagem da tarefa Iniciação no anexo A). Um exemplo de um procedimento rotineiro de operacão seria:

1. Determinar os parâmetros K_p, T_i e T_d.

2. Estabelecer o valor da referência.

3. Por o PID na opção automático.

4. Indicar ao PID que os parâmetros a serem utilizados devem ser os definidos pelo operador.

5. Determinar o valor final de lambda.

6. Determinar quando começará a transmissão de dados do CAMAÇARI para o PC.

7. Acionar a tarefa Estimação/Alocação de Polos.

8. etc...

CAPITULO 4

TESTES E RESULTADOS EXPERIMENTAIS.

4.1 - Introdução.

O objetivo deste capítulo é ilustrar os resultados experimentais obtidos utilizando a implementação descrita no Capítulo 3 sobre processos simulados simples

A seção 4.2 trata do processo simulado utilizado para testar a implementação. Nas seções 4.2 e 4.3 o comportamento servo desejado pelo usuário e a solução obtida pelo algoritmo da seção 2.5 são apresentados. A seção 4.5 comenta as características do ruído causado pelo "hardware" e; por fim, na seção 4.6 alguns resultados selecionados entre os vários ensaios realizados são exibidos.

4.2 - Processo Simulado.

O processo simulado tem a função de transferência entre controle e saída indicada na equação 4.1 (veja A.15 - apêndice A):

$$G(s) = \frac{1}{(1 + 20\ s)(1 + 40\ s)} = \frac{0.00125}{(s + 0.05)(s + 0.025)} \qquad (4.1)$$

As constantes de tempo foram escolhidas de forma a que um tempo de amostragem de 6.0 seg. (limite imposto pela implementação - seção 3.4) seja compatível com a dinâmica do processo. Segundo [24],

é desejável que

$$T * a \leq 0.5;$$

onde a é o maior polo do processo.

Para o processo escolhido (a = 0.05 e b = 0.025):

$$T * a = 0.3.$$

Para estes valores de a e b tem-se que (veja A.1 e A.15):

$$\omega_n = 0.0353553 \quad \text{e} \quad \zeta = 1.0606602.$$

Então, o processo escolhido é super-amortecido, sem atraso de transporte, sendo sua resposta a um degrau unitário:

$$y(t) = 1 + e^{-0.05\,t} - 2\,e^{-0.025\,t};$$

como mostrado na figura abaixo (note-se que o eixo das abscissas de todas as figuras representa instantes de amostragem):

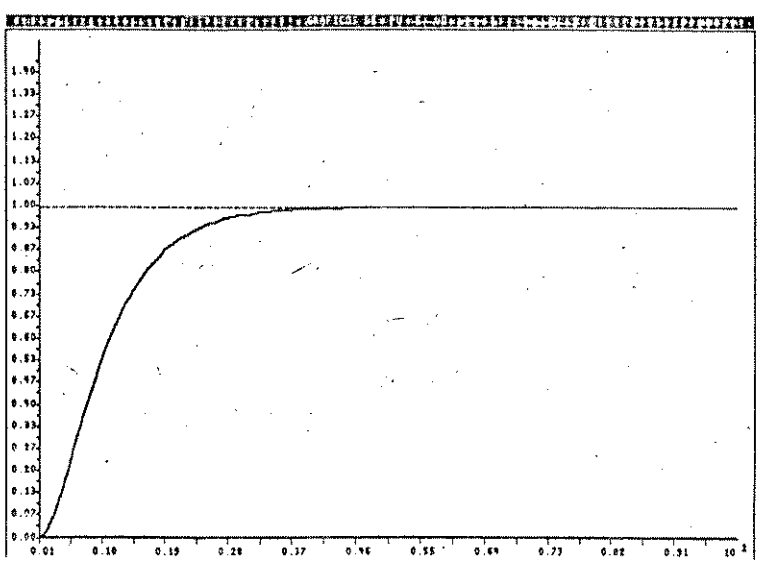

Figura 4.1: Resposta do processo simulado a um degrau unitário.

Define-se o tempo de estabilização a 1% de um processo como o tempo gasto para que a envoltória da oscilação da resposta à um degrau unitário caia a 1% do seu valor final ([13]):

$$e^{-\zeta\ \omega_n\ t_e} \leq 0.01 \quad \therefore \quad t_e = \frac{4.6}{\zeta\ \omega_n}$$

4.3 - Especificação do comportamento servo em malha fechada.

Supõe-se que o usuário deseja um comportamento servo em malha fechada, tal que $\zeta = 0.7$ (ou seja, um "overshoot" máximo de 4.599%) e $t_e = 150$ seg. Isto corresponde a $\omega_n = 0.0438$ e t_p (tempo de "overshoot") igual à 100 seg.

A figura 4.2 mostra o comportamento servo desejado pelo usuário para o processo controlado:

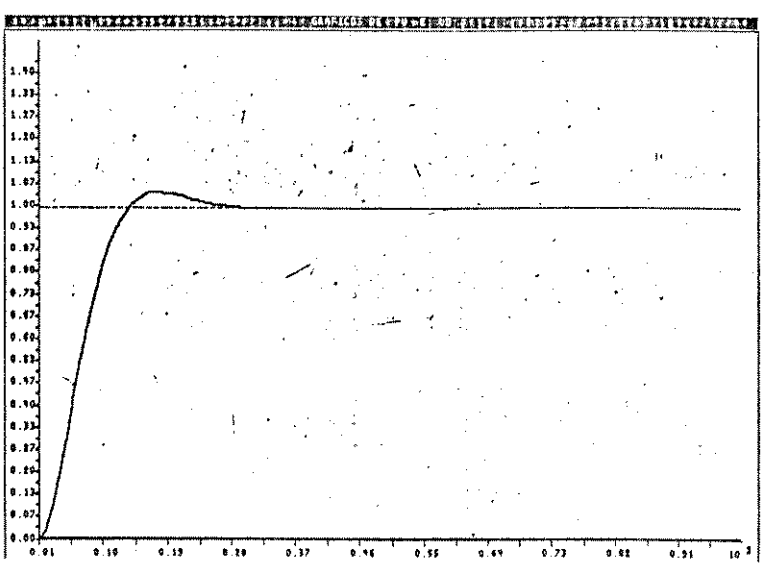

Figura 4.2: **Comportamento desejado para o processo, em malha fechada.**

O modelo discreto (T = 6.0 seg.) do processo é obtido através das equações A.11,...,A.14 (apêndice A):

$$U(z) \longrightarrow \boxed{z^{-1}\,\frac{0.019402 + 0.016699\,z^{-1}}{1 - 1.6015\,z^{-1} + 0.6376\,z^{-2}}} \longrightarrow Y(z)$$

O comportamento servo em malha fechada especificado em 4.3 permite obter o polinômio $D(z^{-1})$ (seção 2.5):

$$D(z^{-1}) = 1 + d_1\,z^{-1} + d_2\,z^{-2}\ .$$

Os coeficientes d_1 e d_2 são obtidos através das equações A.5 e A.6:

$$D(z^{-1}) = 1 - 1.6346\,z^{-1} + 0.6921\,z^{-2}\ .$$

Os polos correspondentes são: $P_{1,2} = 0.8173 \pm j\ 0.1553$.

Os polos não dominantes correspondentes ao polinômio $\gamma(z^{-1})$ são obtidos como descrito na seção 2.5. Obtém-se

$$\gamma(z^{-1}) = 1 - 0.7724\,z^{-1} + 0.1491\,z^{-2}$$

com polos dados por $P_3 = 0.3933$ e $P_4 = 0.3791$.

O polinômio $G(z^{-1})$ do controlador resultante (equações 2.31, 2.32 e 2.33) é dado por:

$$G(z^{-1}) = 10.06 - 15.6\,z^{-1} + 6.18\,z^{-2}$$

com raízes em $G_{1,2} = 0.77 \pm j\ 0.11$.

Os coeficientes do controlador PID (equações 2.42, 2.43 e 2.44) resultante são:

$K_p = 3.2454$

$T_i = 32.4631$

$T_d = 11.4267$

Através da equação 2.19 vê-se que o diagrama de blocos da figura 2.3 pode ser rearranjado da forma ilustrada pela figura 4.3:

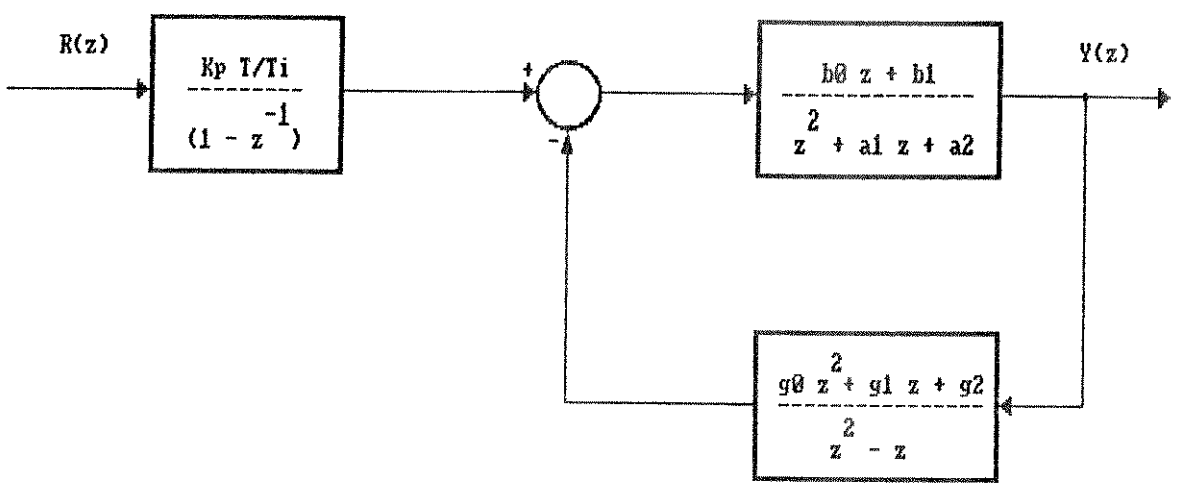

Figura 4.3: Diagrama de blocos do sistema em malha fechada.

A figura 4.4 mostra o lugar das raízes correspondente e a situação dos polos em malha fechada.

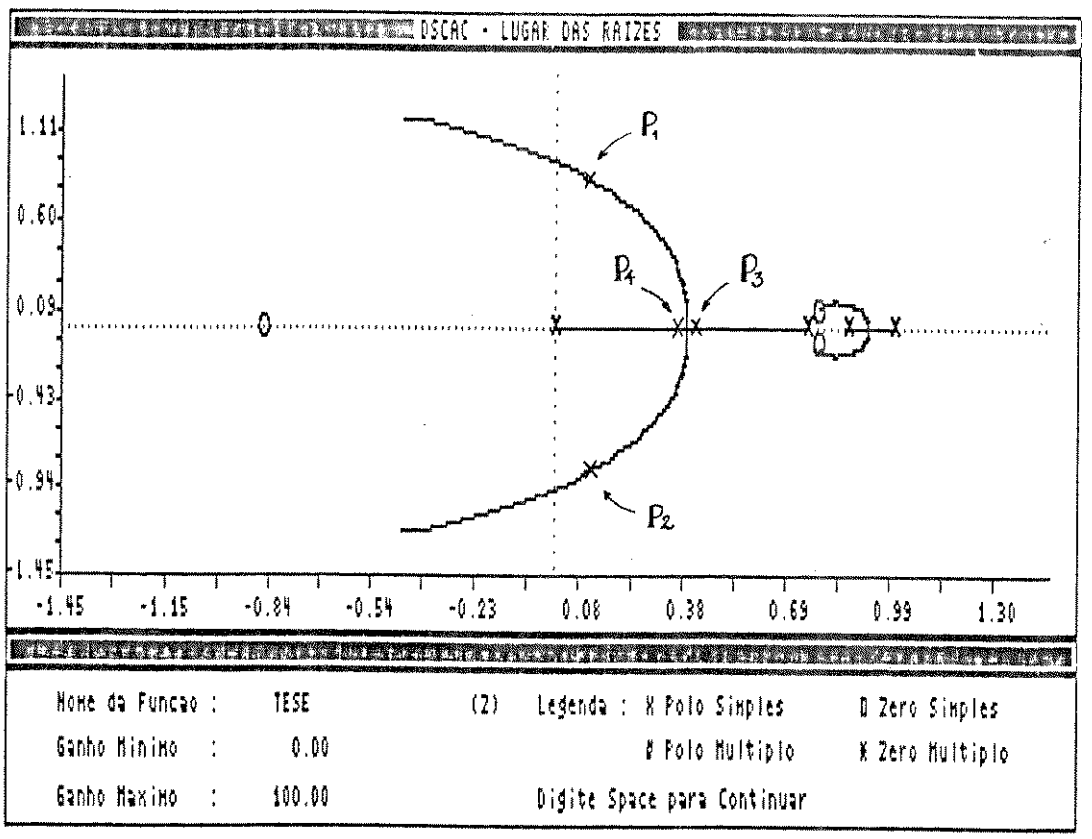

Figura 4.4: **"Root locus" do sistema simulado. Os polos do sistema em malha fechada como determinados pela estratégia de alocação de polos estão indicados na figura.**

4.5 - Ruído do canal.

Além do ruído produzido pelo gerador descrito nas seções 2.2.2 e 3.3.4.2, existe também aquele criado pelas placas de conversão A/D e D/A. Este ruído pode ser minimizado (através do emprego de fontes adequadas, um "terra" eficiente), mas não eliminado, e ademais, sua influência não pode ser desprezada, dado que ele sozinho é suficiente para o funcionamento do estimador.

Assim, vários experimentos que serão exibidos a seguir foram realizados com o emprego apenas do ruído dos canais, mantendo-se o gerador desativado.

A figura abaixo mostra uma realização do ruído do canal. A análise estatística do ruído permite caracteriza-lo como:

1. Branco.
2. Variância < 2.0, suficientemente constante.
3. Média = 0.0.

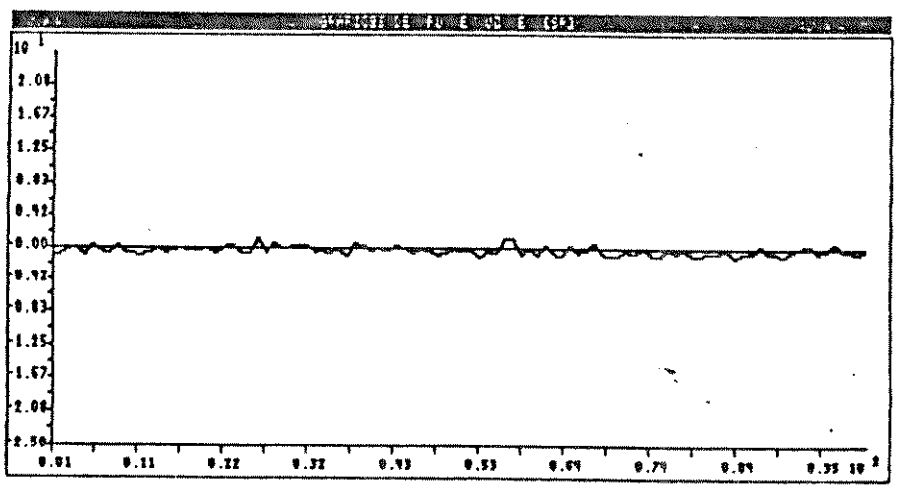

Figura 4.5: **Nível do ruído dos canais de comunicação entre os microcomputadores.**

O modelo completo do sinal de saída é dado então pela equação:

$$Y(z) = \frac{z^{-1}\, B(z^{-1})\, U(z)}{A(z^{-1})} + \frac{z^{-1}\, B(z^{-1})\, R_1(z)}{A(z^{-1})} + \frac{C(z^{-1})\, E(z)}{A(z^{-1})} + R_2(z).$$

onde $R_1(z)$ é a transformada Z do ruído que é adicionado ao sinal de controle enviado pela tarefa PID. Já $R_2(z)$ é a transformada Z do ruído que é adicionado à saída do processo, conforme ilustra o seguinte esquema:

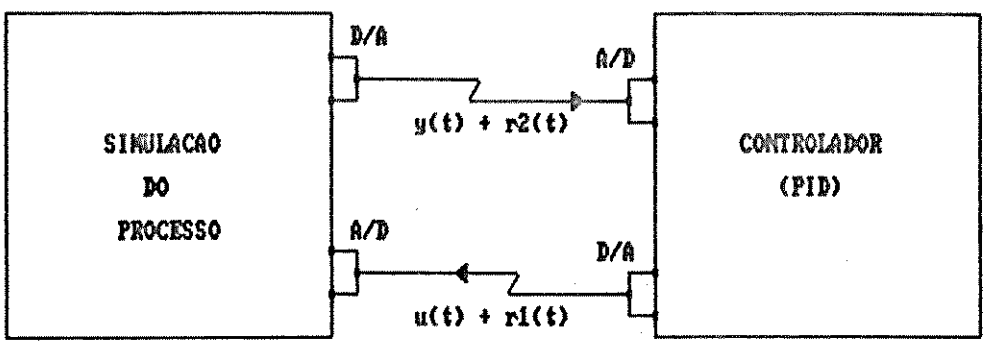

Figura 4.6: **Ruídos causados por Hardware.**

4.6 - Resultados experimentais.

Nesta seção são apresentados resultados experimentais obtidos com a implementação descrita no Capítulo 3. Mostra-se primeiramente o desempenho do estimador em duas situações: malha aberta e malha fechada, a seguir são apresentados os resultados quando se utiliza no controle e na simulação a mesma taxa de amostragem e finalmente apresentam-se os resultados obtidos com o controlador adaptativo controlando um processo com ganho variável.

A. ESTIMAÇÃO DOS PARÂMETROS EM MALHA ABERTA UTILIZANDO COMO SINAL DE ENTRADA UM PRBS.

A figura 4.7 mostra a evolução do sinal de saída quando o sinal de entrada é do tipo PRBS:

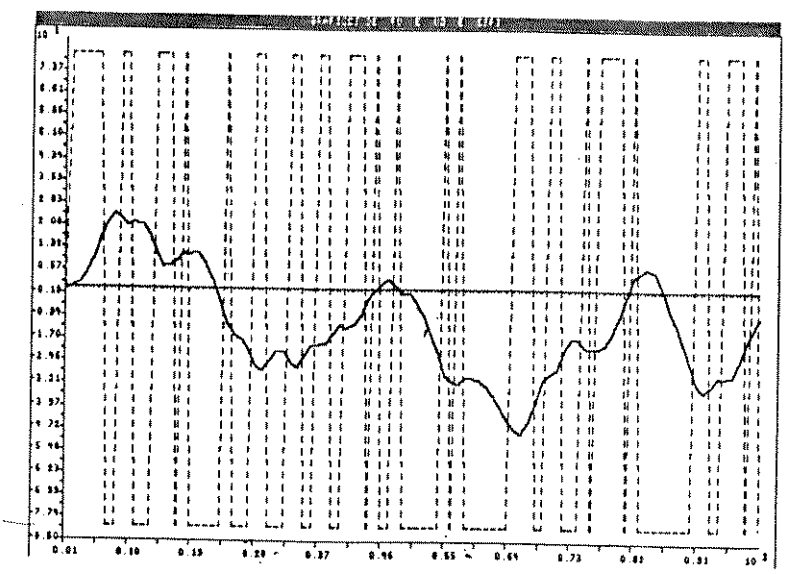

Figura 4.7: Referência (traço-ponto), sinal de saída (contínua) e sinal de controle (tracejada).

As figuras 4.8 e 4.9 mostram a evolução dos parâmetros estimados para os polinômios $A(z^{-1})$ e $B(z^{-1})$:

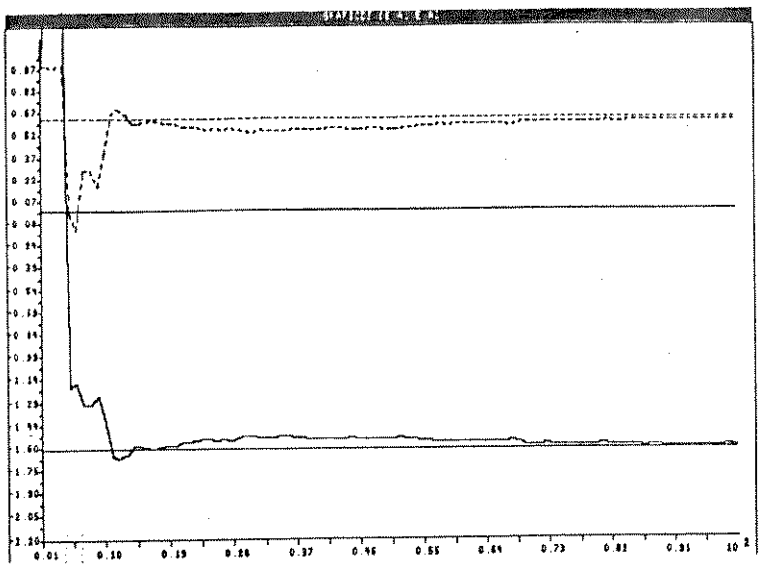

Figura 4.8: Estimação de a_1 (linha cheia) e a_2 (linha tracejada).

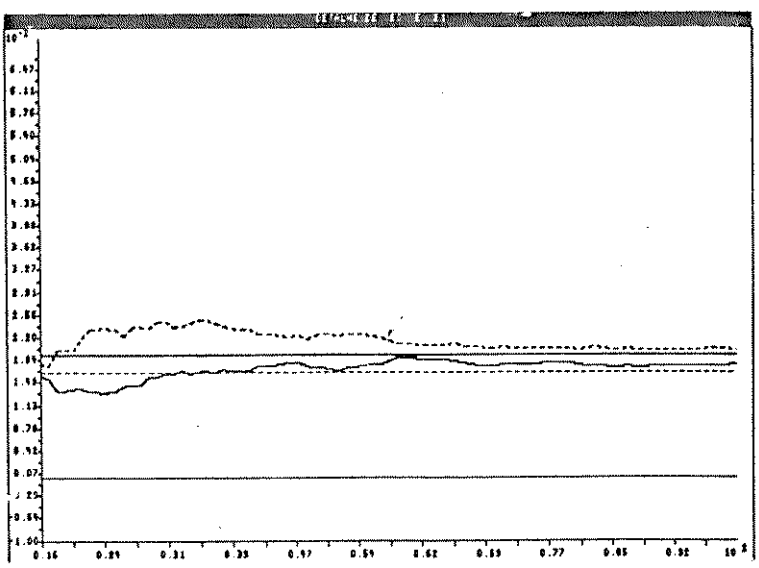

Figura 4.9: Estimação de b_0 (linha cheia) e b_1 (linha tracejada).

Estes resultados foram obtidos utilizando o algoritmo da Matriz Estendida. A utilização do algoritmo dos Mínimos Quadrados levou a resultados semelhantes.

A figura 4.10 representa a evolução do traço da matriz de covariância decrescendo continuamente na medida que os parâmetros estimados se estabilizam.

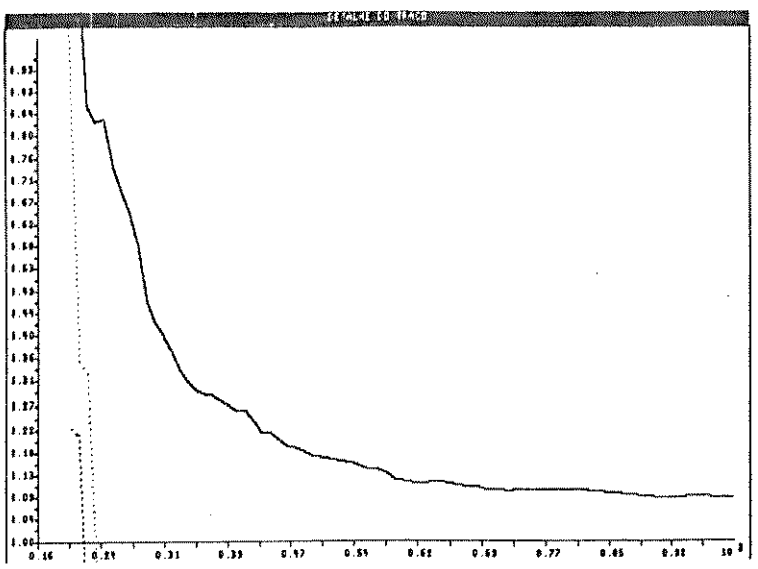

Figura 4.10: **Traço da matriz de covariância (P).**

A figura 4.11 representa a evolução dos parâmetros calculados para o controlador PID utilizando a estratégia de alocação de polos. Os parâmetros K_p, T_i e T_d são calculados a partir dos parâmetros estimados. Na figura estão também representados os valores exatos obtidos a partir do modelo.

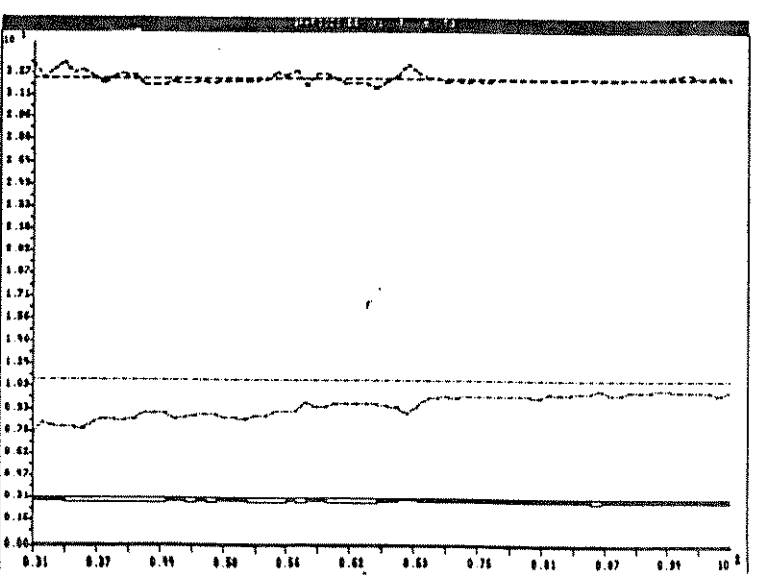

Figura 4.11: **Estimação de K_p (linha cheia), T_i (linha tracejada) e T_d (linha traço-ponto).**

Nota-se na evolução do parâmetro Td calculado uma convergência mais lenta para o valor teórico, mesmo nesta situação de excitação ideal (PRBS). Este aspecto não foi analisado mais detalhadamente.

B. ESTIMAÇÃO DOS PARÂMETROS EM MALHA FECHADA COM CONTROLADOR PID FIXO E ADIÇÃO DE UM PRBS.

Nesta experiência utiliza-se o controlador PID com os seus parâmetros fixos nos valores nominais obtidos na seção 4.4. Para aumentar a excitação com o objetivo de melhorar a estimação acrescenta-se ao sinal de controle um PRBS (Pseudo Random Binary Signal) de módulo 10. A figura 4.12 mostra a evolução da saída e do controle.

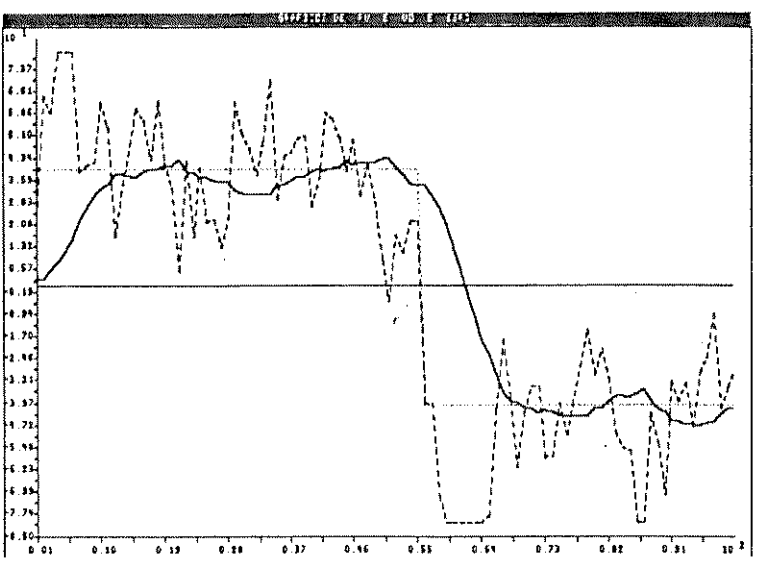

Figura 4.12: **Referência (traço-ponto), sinal de saída (contínua) e sinal de controle (tracejada).**

A influência do sinal PRBS pode ser vista comparando a figura anterior com a figura 4.13 que representa a evolução da saída sem o PRBS.

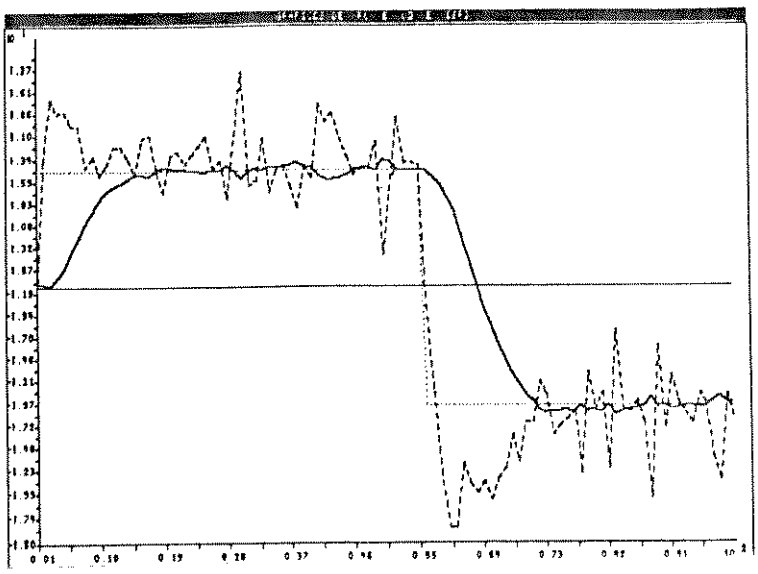

Figura 4.13: Referência (traço-ponto), sinal de saída (contínua) e sinal de controle (tracejada).

As figuras 4.14 e 4.15 mostram a evolução dos parâmetros estimados utilizando o algoritmo dos Mínimos Quadrados e a figura 4.16 o traço da matriz de covariância.

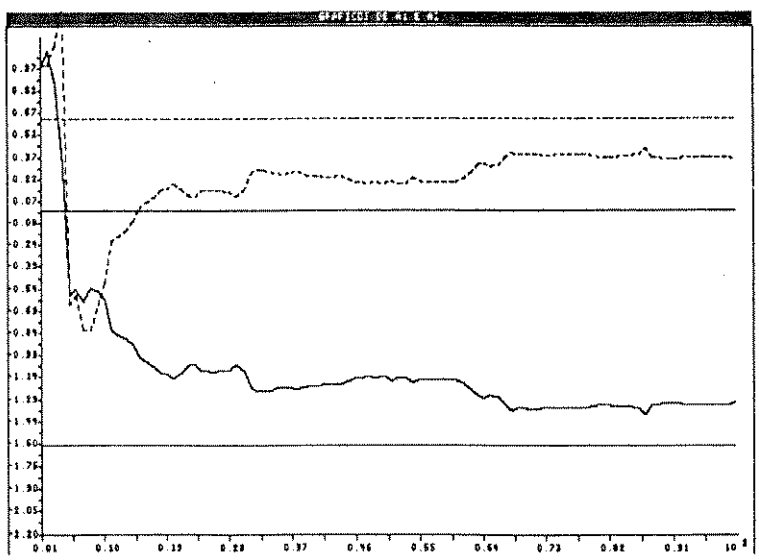

Figura 4.14: Estimação de a_1 (linha cheia) e a_2 (linha tracejada).

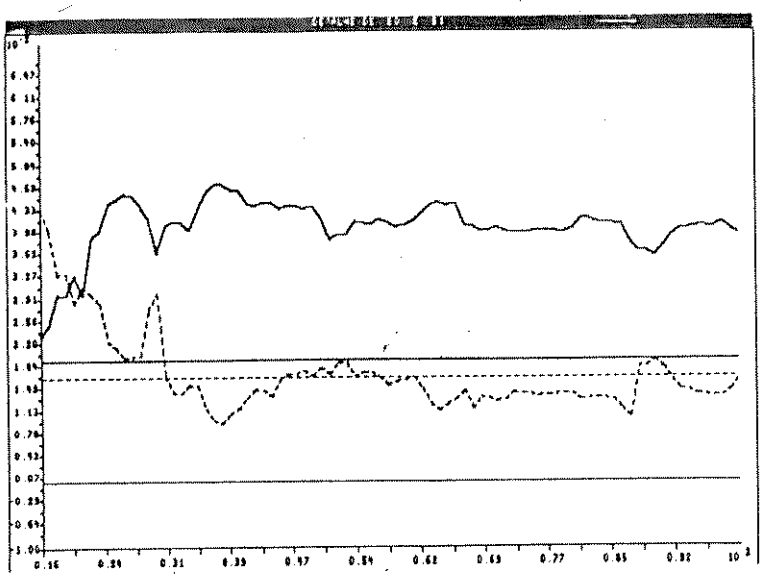

Figura 4.15: Estimação de bo (linha cheia) e b1 (linha tracejada).

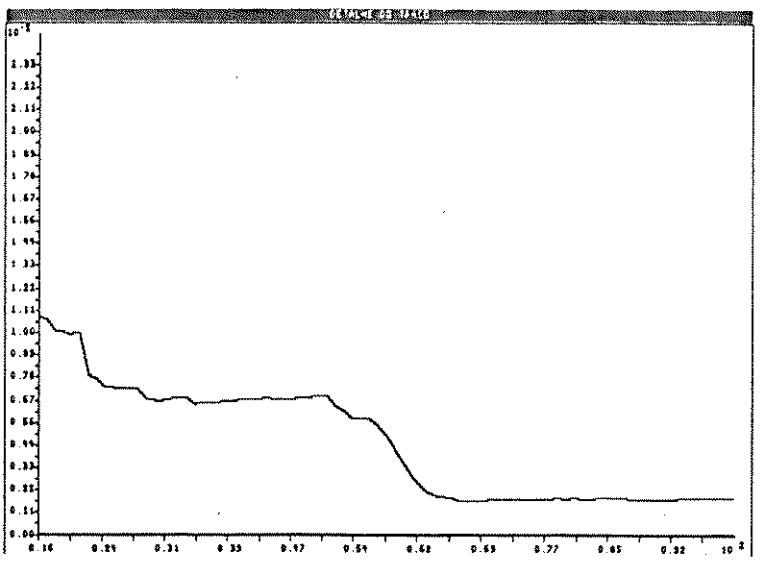

Figura 4.16: Traço da matriz de covariância (P).

Os resultados mostram uma convergência ruim no intervalo de tempo representado e falta de excitação, visivel na grande mudança no traço da matriz de covariância quando da mudança no sinal de referência. A figura 4.17 mostra os parâmetros do controlador PID calculados a partir dos parâmetros estimados.

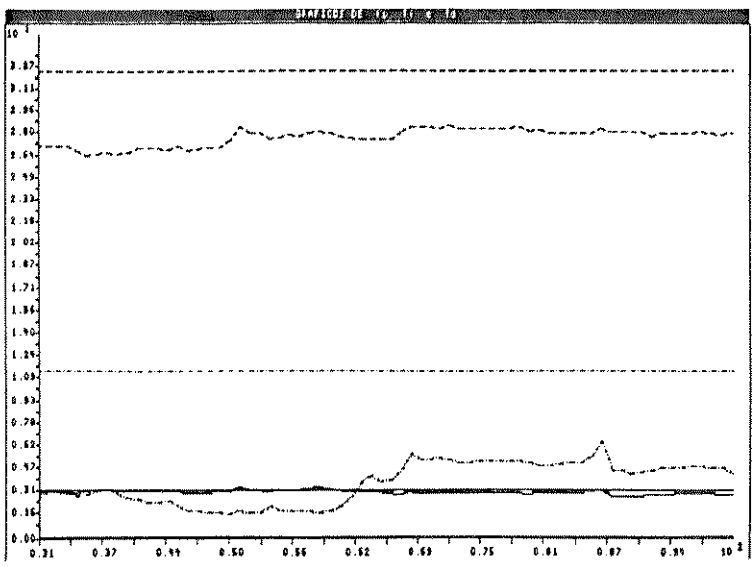

Figura 4.17: Estimação de K_P (linha cheia), T_i (linha tracejada) e T_d (linha traço-ponto).

C. ESTIMAÇÃO DOS PARÂMETROS EM MALHA FECHADA COM CONTROLADOR PID FIXO E ADIÇÃO DE UM SINAL PRBS. TAXAS DE DISCRETIZAÇÃO IGUAIS NOS DOIS PROCESSADORES.

Como foi analisado na seção 3.5 a implementação utilizando taxas de discretização de 6.0 seg. e 0.2 seg. no controle e na simulação respectivamente introduz um atraso suplementar de 0.2 seg. A influência de atrasos fracionários na estimação foi analisada em [25]. Com o objetivo de eliminar esta influência os resultados aqui apresentados foram obtidos com a mesma taxa de discretização nos dois processadores e utilizando no vetor $\varphi(t)$ (equação 2.23) os valores corretos em cada instante de tempo.

O sinal de referência utilizado é um degrau e a evolução da saída e do controle utilizando o PID fixo estão representadas na figura 4.18.

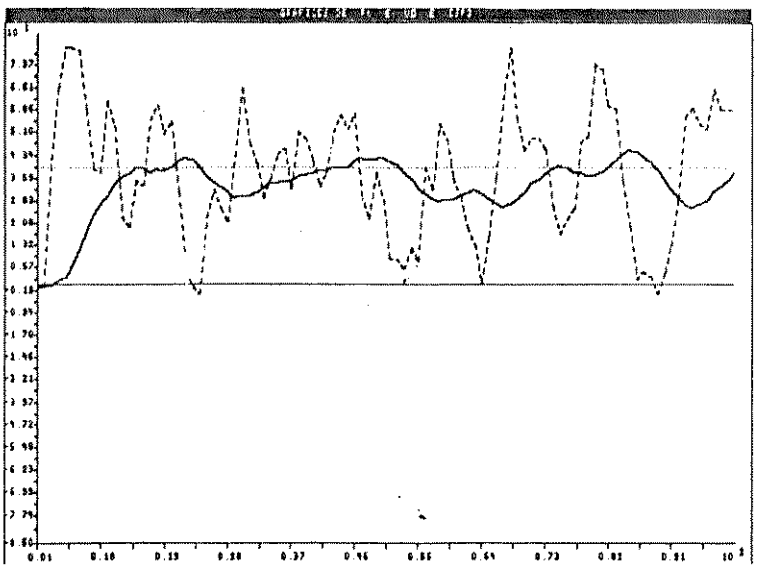

Figura 4.18: Referência (traço-ponto), sinal de saída (contínua) e sinal de controle (tracejada).

A evolução dos parâmetros estimados e o traço da matriz de covariância estão representados nas figuras 4.19, 4.20 e 4.21. A figura 4.22 apresenta a evolução do cálculo recursivo dos parâmetros do PID.

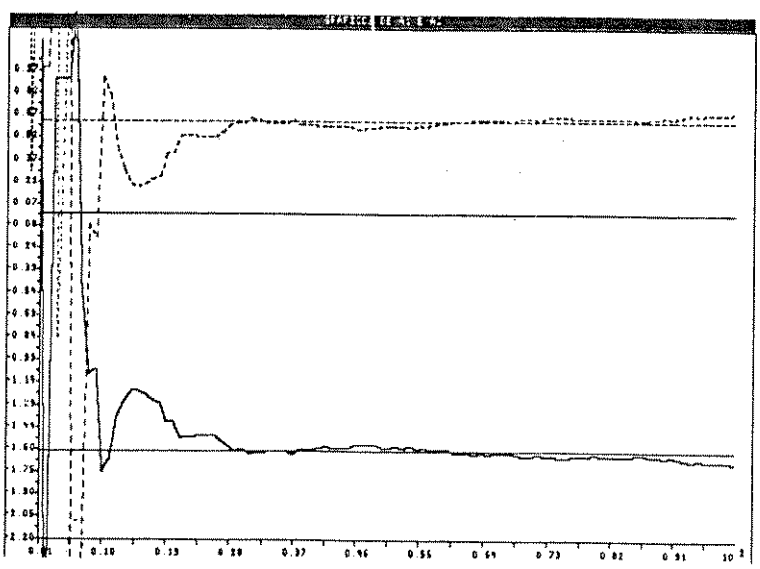

Figura 4.19: Estimação de a_1 (linha cheia) e a_2 (linha tracejada).

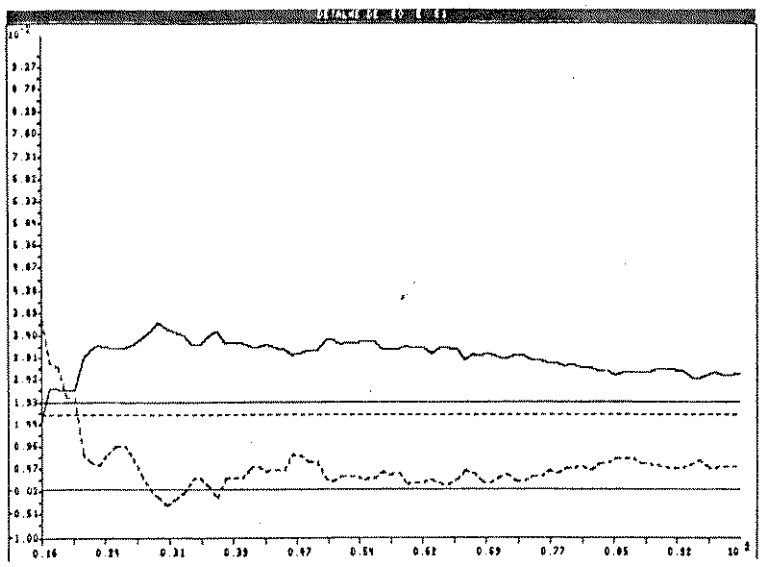

Figura 4.20: Estimação de b_0 (linha cheia) e b_1 (linha tracejada).

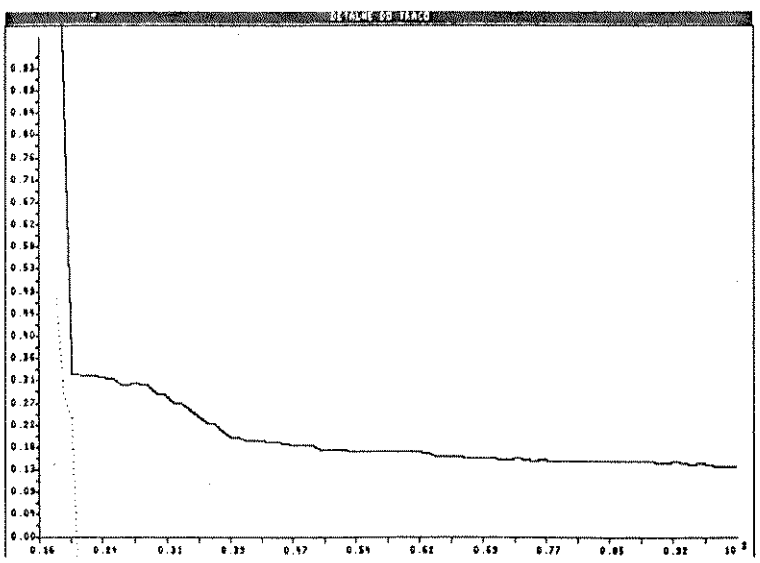

Figura 4.21: Traço da matriz de covariância (P).

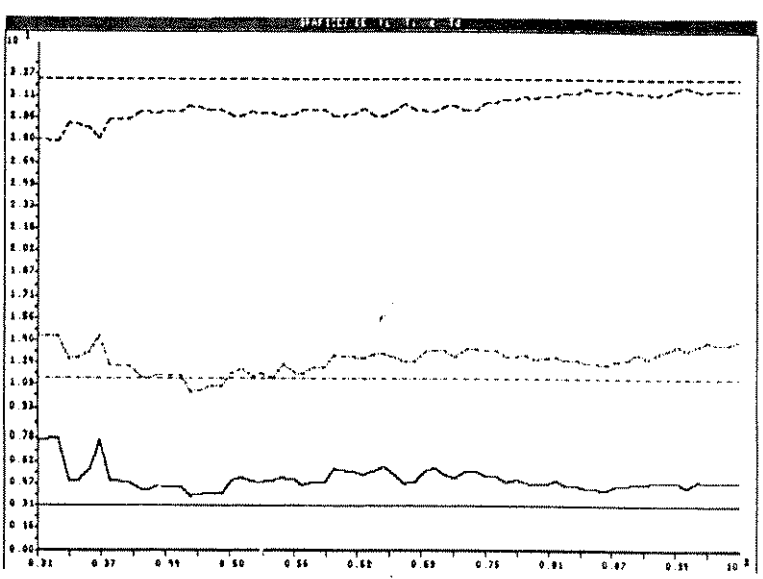

Figura 4.22: Estimação de K_p (linha cheia), T_i (linha tracejada) e T_d (linha traço-ponto).

A convergência nesta realização é melhor do que no caso anterior, permitindo supor que a interferência do atraso fracionário é importante. Não foi feita uma análise mais aprofundada por não estar dentro do escopo deste trabalho. Para o desenvolvimento da implementação realizada o processador de simulação pode utilizar a taxa de 6.0 seg. dado que é compatível com a dinâmica do processo. Numa situação real onde o processo não fosse simulado este problema não se colocaria.

D. CONTROLE ADAPTATIVO DO PROCESSO COM GANHO VARIAVEL.

Nesta experiência utiliza-se o mesmo processo simulado que nos casos anteriores com uma variação linear no ganho entre k = 20 e k = 60 (k = instantes de amostragem). O ganho varia na relação de um para três.

Utiliza-se a taxa de discretização de 6.0 seg. nos dois processadores e o sinal de referência é constante.

O algoritmo de estimação utilizado é o da Matriz Estendida com uma fator de esquecimento (λ) que passa para 0.95 durante a fase de variação do ganho da simulação, retornando a 0.99 após a variação. Supõe-se a situação em que os parâmetros estimados convergiram antes da mudança no ganho do processo, o que corresponde a situação real de ativar o controlador adaptativo quando da ocorrência de mudanças no comportamento do processo.

A figura 4.23 mostra o comportamento do sistema quando se mantém em funcionamento o PID fixo - o sistema em malha fechada torna-se instável dada a mudança no ganho. A figura 4.24 mostra o comportamento do controlador adaptativo mantendo a saída controlada através da adaptação dos parâmetros do controlador.

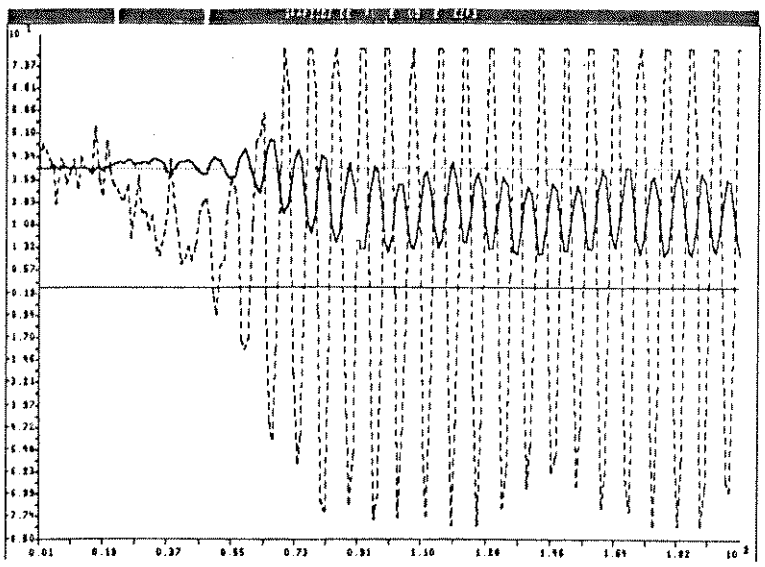

Figura 4.23: Referência (traço-ponto), sinal de saída (contínua) e sinal de controle (tracejada).

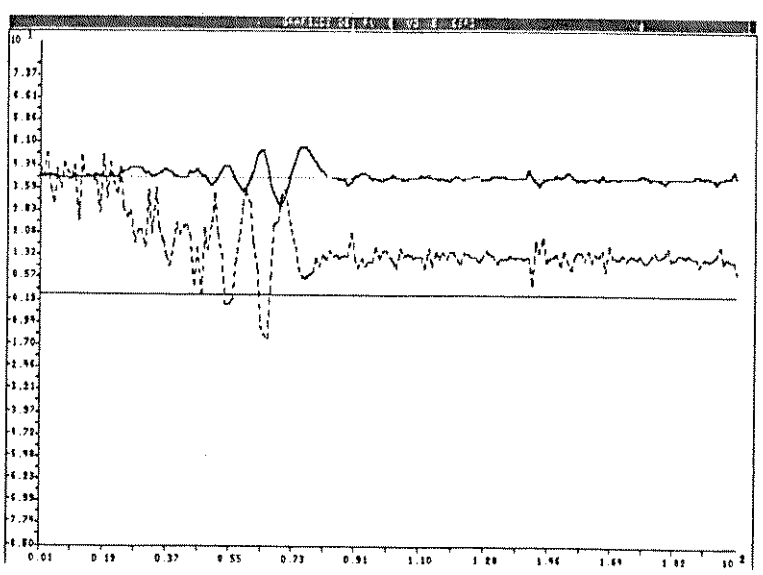

Figura 4.24: Referência (traço-ponto), sinal de saída (contínua) e sinal de controle (tracejada).

A evolução dos parâmetros estimados está representada nas figuras 4.25 e 4.26. Não ocorrem mudanças nos parâmetros do polinômio $A(z^{-1})$, a variação dos parâmetros a_1 e a_2 sendo devida à mudança no fator de esquecimento. Os parâmetros do polinômio $B(z^{-1})$ são modificados (para produzir a mudança no ganho) e na figura 4.26 observa-se que os parâmetros estimados acompanham esta mudança.

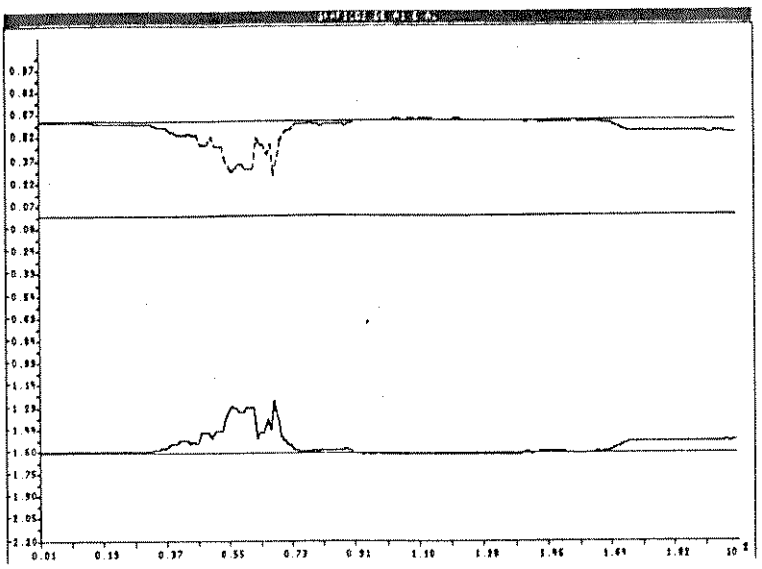

Figura 4.25: Estimação de a1 (linha cheia) e a2 (linha tracejada).

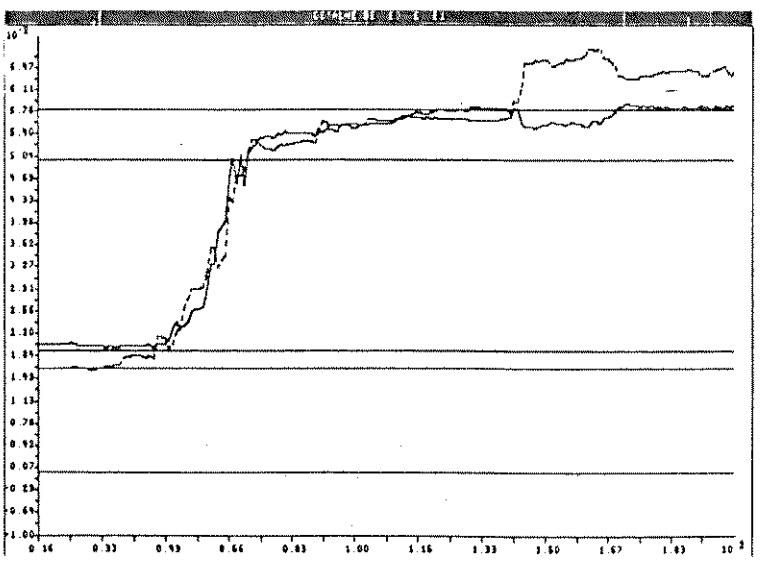

Figura 4.26: Estimação de bo (linha cheia) e b1 (linha tracejada).

A evolução do traço da matriz de covariância está representada na figura 4.27 e a dos parâmetros do PID em 4.28.

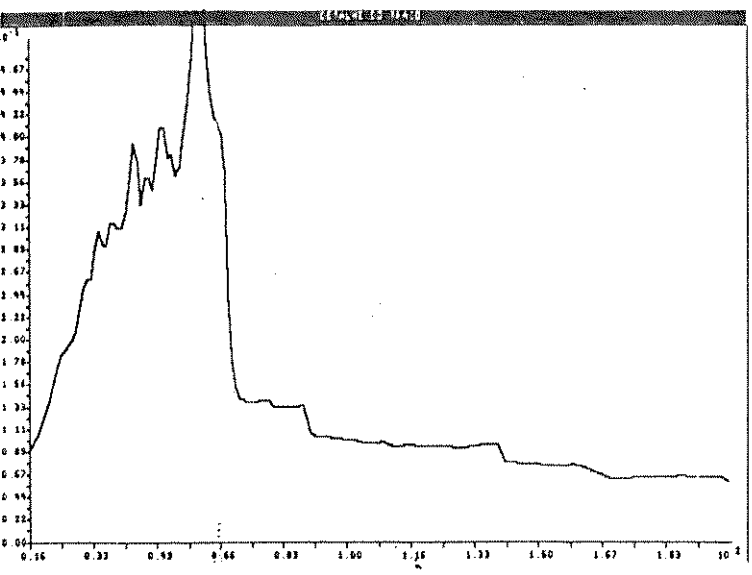

Figura 4.27: Traço da matriz de covariância (P).

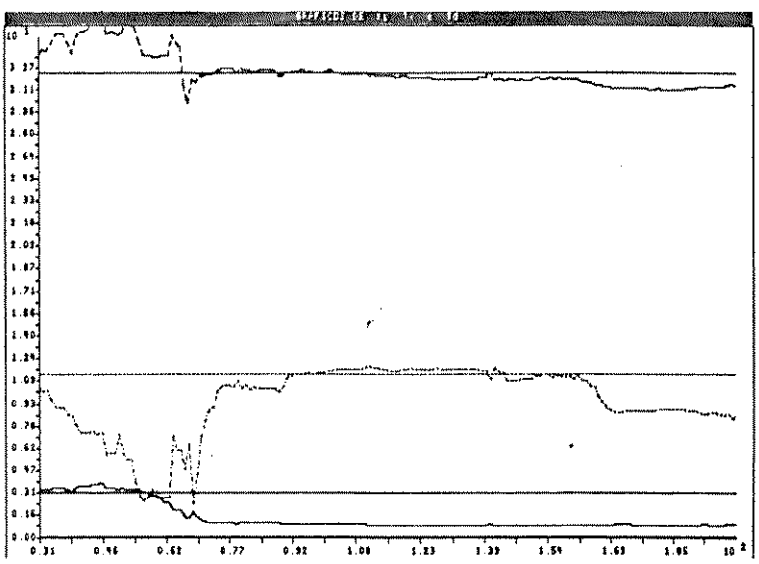

Figura 4.28: Estimação de K_p (linha cheia), T_i (linha tracejada) e T_d (linha traço-ponto).

Através da figura 4.24 pode-se ver que apesar da alteração do ganho do processo (e consequentemente dos parâmetros b_0 e b_1), o algoritmo PID adaptativo continua desempenhando satisfatoriamente suas funções de controle.

A média do sinal de controle passa para 1/3 do sinal de referência, dado que o valor do ganho é agora 3.

Pela figura 4.28 vê-se perfeitamente a evolução correta dos parâmetros do PID. Enquanto K_P cai a um terço do seu valor original, T_i e T_d permanecem praticamente constantes. É importante notar-se que nos últimos vinte instantes de amostragem a estimação principia a divergir dos valores corretos. Tal fato pode ser atribuído à ausência de excitação.

CAPITULO 5

CONCLUSÕES:

Neste trabalho apresentou-se a implementação do controlador PID auto-ajustável desenvolvido em [6].

Tal implementação foi realizada em PASCAL, e utilizou os serviços de um Núcleo de Tempo Real, através de uma interface adequada.

A forma de atuação básica do controlador é uma estratégia de alocação de polos que permite ao usuário tratar o processo em malha fechada como um processo de 2.ª ordem no qual o fator de amortecimento (ζ) e a frequência natural (ω_n) são determinados por ele.

Possui este controlador ainda um módulo opcional que permite estabelecer-se os parâmetros do PID através do exame das variâncias dos sinais de controle e saída do processo. Devido à imprecisão da estimação do polinômio $C(z^{-1})$ e também ao fato deste módulo ter sido exaustivamente estudado em [6], foi o mesmo aqui relegado a segundo plano.

Os experimentos realizados foram então dedicados a ilustrar o comportamento do controlador.

A correção e justeza do controlador foi amplamente verificada através de ensaios feitos sob diversas condições:

a) Parâmetros fixos e PRBS (malha aberta);

b) Parâmetros fixos e PID fixo (mais um PRBS);

c) Parâmetros variáveis e PID fixo;

d) Parâmetros variáveis e PID adaptativo.

O principal produto do trabalho é uma implementação didática e de fácil compreensão (graças à linguagem de alto nível), que permite uma visão clara e abrangente dos problemas característicos do "software" para tempo real (sincronização, temporização, proteção de memória, etc...). O controlador implementado mantém a estrutura simples dos controladores PID e permite, através da adaptação dos seus parâmetros, a auto-sintonização quando ocorrem mudanças nos parâmetros do processo ou no ajuste inicial quando os parâmetros são constantes mas desconhecidos.

Como sugestão para trabalhos futuros fica a possibilidade de aproveitar a implementação para diversas variantes e extensões, tais como a utilização de algoritmos mais sofisticados de estimação (Máxima Verossimilhança), e a introdução de atrasos de transporte (e algoritmos de controle adequados para esta situação).

REFERÊNCIAS BIBLIOGRÁFICAS

[1] - Astrom, K.J.: *Introduction to stochastic control theory*. Academic Press, New York, USA, 1970.

[2] - Astrom, K.J. e Wittenmark, B.: "*On self-tuning regulators*". Automatica, vol. 9, págs. 185-199, 1973.

[3] - Clark, D.W. e aux.: "*Feasibility study of the application of microprocessors to self-tuning controllers*". Report n.° 1137. University of Oxford, England, 1975.

[4 - Wellstead, P.E. e aux.: "*Self-tuning pole/zero assignment regulators*". Int. J. Control, vol. 30, n.° 1, págs. 27-36, 1979.

[5] - Wittenmark, B.: "*Self-tuning PID controllers based on pole placement*". Report TFRT-7179. Lund Institute of Technology, Lund, Sweeden, 1979.

[6] - Batista, J.C.: *Controlador PID auto-ajustável*. Tese de Mestrado . Faculdade de Engenharia Elétrica (UNICAMP), Campinas, Brasil, 1985.

[7] - Lotufo, R.A.: *Implementação de algoritmos de controle adaptativo em microprocessadores*. Tese de Mestrado. Faculdade de Engenharia Elétrica (UNICAMP), Campinas, Brasil, 1981.

[8] - Silva, M.A.; Gomide, F.A.C. e Amaral, W.C.: "*Um sistema especialista para a autosintonia de controladores industriais do tipo PID*". Anais do 4.° Simpósio Brasileiro de Inteligência Artificial, pág. 437. Universidade Federal de Uberlândia, Uberlândia, Brasil, 1987.

[9] - Forsythe, G.E.; Malcolm, M.A. e Moler, C.B.: *Computer Methods for Mathematical Computations.* Prentice Hall, Englewood Cliffs, N.J., USA, 1977.

[10] - Dyck, V.A.; Lawson, J.D.; Smith, J.A. e Beach, R.J.: *COMPUTING - An introduction to structured problem solving using PASCAL.* Reston Publishing Co., Reston, Virginia, USA.

[11] - Knuth, D.E.: *The art of computer programming, vol. 2: Semi-numerical algorithms.* Addison-Wesley, Reading, Mass., USA, 1969.

[12] - Deshpande, P.B. e Ash, R.H.: *Elements of COMPUTER PROCESS CONTROL with Advanced Control Aplications.* Prentice Hall, Englewood Cliffs, N.J., USA, 1983.

[13] - Franklin, G.F. e Powell, J.D.: *Digital Control of Dynamic Systems.* Addison-Wesley, Reading, Mass., USA, 1980.

[14] - Ljung, L. e Söderstrom, T.: *Theory and Practice of Recursive Identification.* The MIT Press, Cambridge, Mass., USA, 1983.

[15] - Thornton, C.L. e Bierman, G.J.: *"UDU covariance factorization for Kalman filtering".* C.T. Leondes, ed: Control and Dynamic Systems, vol. 16, Academic Press, New York, USA, 1980.

[16] - Isermann, R.: *Digital control systems.* Springer-Verlag, Berlin, Germany, 1981.

[17] - *PASCAL/Z Implementation manual - Versão 4.0.* Ithaca Intersystems Inc. Distributed by Lifeboat Associates, 1651, Third Avenue, New York, USA.

[18] - Os sistemas *CADCLA e CADPID* são parte integrante do projeto *DSCAC*. Divisão de Metodologia Aplicada - Instituto de Automação do Centro Tecnológico para Informática (CTI), Campinas, Brasil.

[19] - Gomide, F.A.C. e Netto, M.L.A.: *Introdução à Automação Industrial Informatizada*. Editorial Kapelusz S.A. - EBAI, Buenos Aires, Argentina, 1987.

[20] - Junior, H.J.A. e Ribeiro, I.M.C.: *"NTR-80: Manual do Usuário - Versão 3.1"*. Instituto de Automação (DCP/ETR) do Centro Tecnológico para Informática (CTI), Campinas, Brasil, 1986.

[21] - Magalhães, M.F.: *Software para tempo real*. Editora da Unicamp - EBAI, Campinas, Brasil, 1986.

[22] - Ben-Ari, M.: *Principles of Concurrent Programming*. Prentice Hall, Englewood Cliffs, N.J., USA, 1982.

[23] - *Manual de Operação do Terminal de Video E-800*. Ref. TDA E804.800X00. T.D.A. - Indústria de Produtos Eletrônicos Ltda., Brasil, 1985.

[24] - Sinha, N.K. e Puthenpura, S.: *"Choice of the sampling interval for the identification of continuous-time systems from samples of input/output data"*. IEE Proceedings, vol. 132, pt. D, n.° 6, novembro de 1982.

[25] - Wellstead, P.E.: *"Techniques of Self-Tuning"*. Optimal Cont. App. Math., vol. 3, pág. 305, 1982.

[26] - Shinners, S.M.: *Modern control system theory and aplication*. Addison-Wesley, Reading, Mass., USA, 1972.

APÊNDICE A

A.1 - Processos de 2.ª ordem.

Como foi mencionado na seção 2.5, um processo de 2.ª ordem (na forma padrão - [26]):

$$G(s) = K \frac{\omega_n^2}{s^2 + 2\zeta\,\omega_n\, s + \omega_n^2} \qquad (A.1)$$

possui função de transferência em Z igual à:

$$G(z) = K \frac{z^{-1}\,(b_o + b_1\, z^{-1})}{1 + a_1\, z^{-1} + a_2\, z^{-2}} \qquad (A.2)$$

As expressões que fornecem o valor dos coeficientes b_o, b_1, a_1 e a_2 são (para $\zeta < 1.0$):

$$b_o = 1 - e^{-T\,\omega_n\,\zeta}\cos\left(T\,\omega_n\sqrt{1-\zeta^2}\right) - e^{-T\,\omega_n\,\zeta}\,\frac{\zeta}{(1-\zeta^2)^{1/2}}\,\mathrm{sen}\left(T\,\omega_n\sqrt{1-\zeta^2}\right) \qquad (A.3)$$

$$b_1 = e^{-2\,T\,\omega_n\,\zeta} - e^{-T\,\omega_n\,\zeta}\cos\left(T\,\omega_n\sqrt{1-\zeta^2}\right) + e^{-T\,\omega_n\,\zeta}\,\frac{\zeta}{(1-\zeta^2)^{1/2}}\,\mathrm{sen}\left(T\,\omega_n\sqrt{1-\zeta^2}\right) \qquad (A.4)$$

$$a_1 = -2\, e^{-T\,\omega n\,\zeta}\, cos(T\,\omega n \sqrt{1-\zeta^2}) \qquad (A.5)$$

$$a_2 = e^{-2\,T\,\omega n\,\zeta} \qquad (A.6)$$

Para $\zeta = 1.0$ tem-se:

$$b_0 = 1 - e^{-\omega n\,T} - \omega n\, e^{-\omega n\,T} \qquad (A.7)$$

$$b_1 = e^{-2\,\omega n\,T} - e^{-\omega n\,T} + \omega n\, e^{-\omega n\,T} \qquad (A.8)$$

$$a_1 = -2\, e^{-\omega n\,T} \qquad (A.9)$$

$$a_2 = e^{-2\,\omega n\,T} \qquad (A.10)$$

Para $\zeta > 1.0$ tem-se:

$$b_0 = 1 - e^{-\zeta\,\omega n\,T}\left(e^{T\,\omega n\,(\zeta^2-1)^{1/2}} + e^{-T\,\omega n\,(\zeta^2-1)^{1/2}}\right) -$$

$$- \frac{e^{-\zeta\,\omega n\,T}}{2}\left[\frac{e^{T\,\omega n\,(\zeta^2-1)^{1/2}}\,(\zeta^2-1-\zeta\sqrt{\zeta^2-1})}{1-\zeta^2} + \right.$$

$$\left. + \frac{e^{-T\,\omega n\,(\zeta^2-1)^{1/2}}\,(\zeta^2-1-\zeta\sqrt{\zeta^2-1})}{1-\zeta^2}\right] \qquad (A.11)$$

$$b_1 = e^{-2\zeta\omega_n T} +$$

$$+ \frac{e^{-\zeta\omega_n T}}{2}\left[\frac{e^{T\omega_n(\zeta^2-1)^{1/2}}(\zeta^2-1-\zeta\sqrt{\zeta^2-1})}{1-\zeta^2} + \right.$$

$$\left. + \frac{e^{-T\omega_n(\zeta^2-1)^{1/2}}(\zeta^2-1-\zeta\sqrt{\zeta^2-1})}{1-\zeta^2}\right] \qquad (A.12)$$

$$a_1 = - e^{-\zeta\omega_n T}(e^{T\omega_n(\zeta^2-1)^{1/2}} + e^{-T\omega_n(\zeta^2-1)^{1/2}}) \qquad (A.13)$$

$$a_2 = e^{-2\zeta\omega_n T} \qquad (A.14)$$

Note-se que a equação A.1 pode ser escrita (para $\zeta > 1.0$) na seguinte forma:

$$G(s) = \frac{a\ b}{(s+a)(s+b)} \qquad (A.15)$$

onde: $a\ b = \omega_n^2$ e $a + b = 2\zeta\omega_n$.

Se a e b são reais e $a \neq b$, então $\zeta > 1.0$ e b_o, b_1, a_1 e a_2 da equação A.2 podem ser obtidos das equações A.11,...,A.14 ou através das seguintes equações:

$$b_o = \frac{a(1-\beta) - b(1-\alpha)}{\gamma} \qquad (A.16)$$

$$b_1 = \frac{b\,\beta\,(1-\alpha) - a\,\alpha\,(1-\beta)}{\gamma} \qquad \text{(A.17)}$$

$$a_1 = -\alpha - \beta \qquad \text{(A.18)}$$

$$a_2 = \alpha\,\beta \qquad \text{(A.19)}$$

onde

$$\alpha = e^{-a\,T}, \qquad \beta = e^{-b\,T} \quad e \quad \gamma = a - b.$$

Se ao processo representado pela equação A.15 é adicionado um atraso de transporte, sua nova representação será:

$$G(s) = \frac{a\,b\,e^{-\tau d}}{(s+a)\,(s+b)} \qquad \text{(A.20)}$$

onde τd não é necessariamente um múltiplo inteiro do período de amostragem. Supondo novamente que $\zeta > 1.0$, ter-se-á para a função de transferência em Z:

$$G(z) = \frac{1}{z^1}\,\frac{b_0 + b_1\,z^{-1} + b_2\,z^{-2}}{1 + a_1\,z^{-1} + a_2\,z^{-2}} \qquad \text{(A.21)}$$

$$b_0 = 1 + j_1\,\alpha' + j_2\,\beta' \qquad \text{(A.22)}$$

$$b_1 = -(\alpha + \beta + j_1\,\alpha'\,\beta + j_1\,\alpha' + j_2\,\beta'\,\alpha + j_2\,\beta') \qquad \text{(A.23)}$$

$$b_2 = \alpha\,\beta + j_1\,\alpha'\,\beta + j_2\,\beta'\,\alpha \qquad (A.24)$$

$$a_1 = -\alpha - \beta \qquad (A.25)$$

$$a_2 = \alpha\,\beta \qquad (A.26)$$

onde $\alpha = e^{-a\,T}$, $\beta = e^{-b\,T}$, $\alpha' = e^{-a\,m\,T}$,

$\beta' = e^{-b\,m\,T}$, $j_1 = \dfrac{b}{a-b}$, $j_2 = \dfrac{a}{b-a}$,

$\tau_d = l\,T - m\,T$; $0 < m < 1$ e l um inteiro positivo.

APÊNDICE B

B.1 - Implementação do gerador de ruído branco.

Pode-se implementar o gerador descrito na seção 2.2.2 em PASCAL, para uma máquina de 8 "bits", da seguinte forma:

```
PROCEDURE  Random(var n: integer; {previous random number
                                   (initially the seed)}
                  var u: real); {random number where u is [0, 1)}
BEGIN      {Random}
     n = n * 12869;
     IF (n > 25843) THEN          {prevent overflow on add}
          n = n - 32767 - 1;
     n = n + 6925;
     IF (n < 0) THEN         {handle overflow by adding modulus}
          n = n + 32767 + 1;
     u = n * 3.0517578E-05;
END;       {Random}
```

O parâmetro m é escolhido como $m = 2^{b-1}$, onde b é o número de "bits" da representação dos números inteiros (a qual depende do compilador utilizado). Assim, m = 32768, a = 12869 e c = 6925. Para entender o funcionamento do procedure, há que notar-se o seguinte:

- o comando n = n * a; pode causar um "overflow". Se assim o for e o número resultante for positivo, este número representa [n * a] MOD m, e deve-se a seguir adicioná-lo à c. Caso contrário, (n.º negativo), deve-se alterar seu sinal, bastando para isso adicionar-lhe |m|.

- há a possibilidade de "overflow" na adição com c, por isso testa-se se o número é maior que m - c.

- este "procedure" funcionará independentemente da representação do número inteiro pela máquina (sinal-e-magnitude, "two-complement", "one-complement").

ANEXO A

1. MICROCOMPUTADOR DE SIMULACAO.

O anexo A divide-se em duas partes: a primeira contém os programas referentes ao micro de simulação, e a segunda os referentes ao microcomputador de controle.

As próximas cinco folhas contém:

Program Tese2(0) ——————→ Tarefa Iniciação 1;

Procedure Task1 ——————→ Tarefa Relógio 1;

Procedure Task2 ——————→ Tarefa Simulação.

Os "Procedures" Task3 e Task4 estão sobrando (reserva).

Após estes programas (referentes à simulação) encontra-se outra nota explicativa, a qual antecede os programas referentes ao microcomputador de controle. Esta distribuição tenta evitar equívocos na localização das tarefas.

```
PROGRAM Tese2(O);

      Type  semaforo = array[1..2] of integer;

      Var   acess1, acess2, video: semaforo;
            nbct, tambct, nivpri, stack, roterr: integer;
            endtsk, endt1, endt2, endt3, endt4: integer;
            val, priori: integer;
            Ct, GPV, GVO, Ga1, Ga2, GbO, Gb1, Gc1, Gc2, Gvar: real;

   Procedure Inintr(nbct,tambct,nivpri,stack,roterr: integer);external;
   Procedure Inisem(var sema1:semaforo; valor:integer);external;
   Procedure Wait(var sema2:semaforo);external;
   Procedure Signt(var sema3:semaforo);external;
   Procedure Delay(ticks:integer);external;
   Procedure Stop;external;
   Procedure Tabela;external;
   Procedure Igs;external;
   Procedure Start1(var endt1:integer);external;
   Procedure Start2(var endt2:integer);external;
   Procedure Start3(var endt3:integer);external;
   Procedure Start4(var endt4:integer);external;
   Procedure Start(endtsk,priori:integer);external;
   Procedure Task(endt5,pri:integer);external;
   Procedure Escda(valda, cda: integer);external;
   Procedure Lead(var valad:integer; cad: integer);external;

   BEGIN
{          Inicio do Programa      }
      val := 0;
      Inisem(acess1,val);
      val := 1;
      Inisem(acess2,val);
      val := 1;
      Inisem(video,val);
{       Inicializamos os Semaforos       }
      nbct := 5;
      tambct := 1024;
      nivpri := 4;
      stack := 17408;
      roterr := 0;
      Inintr(nbct,tambct,nivpri,stack,roterr);
{       Inicializamos o Nucleo           }
      Igs;
      Start1(endt1);
      Start2(endt2);
      Start3(endt3);
```

```
        Start4(endt4);
{                       Obtivemos os enderecos das tarefas          }
        Ct  := 0.5;
        Ca1 := 0.0;
        Ca2 := 0.0;
        Cb0 := 0.0;
        Cb1 := 0.0;
        Cc1 := 1.0;
        Cc2 := 0.13;
        Cvar := 0.0012;
        CV0 := 0.0;
        CPV := 0.0;
{                       Inicializamos variaveis do processo                 }
        priori := 1;
        endtsk := endt1;
        Start(endtsk, priori);
        priori := 1;
        endtsk := endt2;
        Start(endtsk, priori);
        priori := 2;
        endtsk := endt3;
        Start(endtsk, priori);
        priori := 3;
        endtsk := endt4;
        Start(endtsk, priori);
{                       Disparamos todas as tarefas                         }
        Stop;
        End.
```

```
EXTERNAL TESE2::KIT47(1);


Procedure Task1;

     Var loop: boolean;
         aa: integer;

     BEGIN
             loop := false;
             aa := ROUND(0.2/0.0164);
             REPEAT
                     Signt(acess1);
                     Delay(aa);
             UNTIL loop;
     END;



Procedure Task2;          {$f- ,m- ,r- ,u- desable  overflow errors }

      Var Theta, Phi: array[1..6] of real;
          ruido, desvio, media, PV, PVaux, VO: real;
          i, semente,  valda, valad, cad, cda: integer;
          a, b, T, alfa, beta, sigma: real;
          loop: boolean;

 PROCEDURE random(var n:integer; var u: real);

      BEGIN {random}
           n := n * 12869;
           IF (n > 25043) THEN {prevent overflow on add}
               BEGIN
                    n := n - 32767 - 1;
               END; {if}
           n := n + 6925;
           IF (n < 0) THEN     {handle overflow by adding modulus}
               n := n + 32767 + 1;
           u := n * 3.0517578e-05;
      END;  {random}


 {Procedure for normally distributed random numbers}

 PROCEDURE normal(var seed: integer; mean,stdev: real; var xn: real);

      VAR u:    real;   {uniformly distributed random number}
          sum:  real;   {sum of the uniform random numbers}
          i: integer;   {loop counter}
```

```
    BEGIN  (normal)
         sum := 0.0;
         FOR i := 1 TO 12 DO
             BEGIN
                  random(seed, u);
                  sum := sum + u;
             END  (for);
         xn := mean + (sum-6) * stdev;
    END    (normal);


BEGIN  ( Task2 )
     a := 0.05;
     b := 0.025;
     T := 0.2;
     alfa := EXP(-a * T);
     beta := EXP(-b * T);
     sigma := a - b;
     Theta[1] := -alfa - beta;
     Theta[2] := alfa * beta;
     Theta[3] := (a - a*beta - b + b*alfa) / sigma;
     Theta[4] := (b*beta - b*beta*alfa - a*alfa + a*alfa*beta)/sigma;
     Theta[5] := 0.0;
     Theta[6] := 0.0;

     desvio := SQRT(0.000012);
     media := 0.0;
     semente := 345;
     PV := 0.0;
     VO := 0.0;
     ruido := 0.0;

     FOR i := 1 TO 6 DO
           Phi[i] := 0.0;
    loop := false;

   REPEAT
      Begin
         Wait(acess1);

         valda := 2048;
         cda := 2;
         Escda(valda,cda);

         ( Atualizacao )
         Phi[2] := Phi[1];
         Phi[1] := -PV;
         Phi[4] := Phi[3];
         Phi[3] := VO;
         Phi[6] := Phi[5];
{         Phi[5] := ruido;                    }
         Phi[5] := 0.0;

   ( Esta parte do programa gera o ruido branco )
```

```
        Normal(semente, media, desvio, ruido);

      ( Ate aqui geramos o ruido branco  )

            PV := 0.0;
            FOR i:= 1 TO 6 DO
                PV := PV + Phi[i] * Theta[i];
             PV := PV + ruido;

      (  Ler  o  sinal  de  controle  na  entrada  A/D   )
            cad := 0;
            Lead(valad, cad);
            VO := valad * 170.0 / 4095.0;
            VO := VO - 85.0;

      (  Escrever  a  variavel  do  processo  na  saida  D/A   )
            PVaux := PV + 85.0;
            PVaux := PVaux * 4095.0 / 170.0;
            valda := ROUND(PVaux);
            cda := 0;
            Escda(valda,cda);


            valda := 0;
            cda := 2;
            Escda(valda,cda);

         End;
      UNTIL loop;
END;  (  de Task2 )



PROCEDURE  TASK3;

     BEGIN
              Stop;
      END;

PROCEDURE TASK4;

     BEGIN
              Stop;
      END;
```

2. MICROCOMPUTADOR DE CONTROLE.

As próximas 18 folhas contém:

Program Tese3(0)	⟶	Tarefa Iniciação 2;
Procedure Task2	⟶	Tarefa Relógio 2;
Procedure Task3	⟶	Tarefa PID;
Procedure Task4	⟶	Tarefa Estimação/Alocação de Polos;
Procedure Task5	⟶	Tarefa Variância;
Procedure Task6	⟶	Tarefa Display;
Procedure Task7	⟶	Tarefa Operação.

```
PROGRAM Tese3(O);

     Type  semaforo = array[1..2] of integer;

     Var   acessO, acess1, acess2, video: semaforo;
           nbct, tambct, nivpri, stack, roterr: integer;
           endtsk, endt2, endt3, endt4, endt5, endt6, endt7,Gpont: integer;
           val, priori: integer;
           Gvoman, Gkp, Gti, Gtd, GPV, GVO, GSP, Gt, Gvomax, Gvomin: real;
           Gmanaut, Gparest, Gonoff, Gpid, Gonps: char;
           GEkp, GEti, GEtd, Gtraco, Ga1, Ga2, GbO, Gb1, Gc1, Gc2: real;
           Gpsrb, Gvarr, Gkk1, Gkk2, Gsigma1, Gvaru, Gvary, GaSP: real;
           GVkp, GVti, GVtd, Glamb: real;

Procedure Inintr(nbct,tambct,nivpri,stack,roterr: integer);external;
Procedure Inisem(var sema1:semaforo; valor:integer);external;
Procedure Wait(var sema2:semaforo);external;
Procedure Signt(var sema3:semaforo);external;
Procedure Delay(ticks:integer);external;
Procedure Stop;external;
Procedure Tabela;external;
Procedure Igs;external;
Procedure Start2(var endt2:integer);external;
Procedure Start3(var endt3:integer);external;
Procedure Start4(var endt4:integer);external;
Procedure Start5(var endt5:integer);external;
Procedure Start6(var endt6:integer);external;
Procedure Start7(var endt7:integer);external;
Procedure Start(endtsk,priori:integer);external;
Procedure Task(endt5,pri:integer);external;
Procedure Escda(valda, cda: integer);external;
Procedure Lead(var valad:integer; cad: integer);external;

BEGIN
      Inicio do Programa      }
     val := 0;
     Inisem(acessO,val);
     val := 0;
     Inisem(acess1,val);
     val := 1;
     Inisem(acess2,val);
     val := 1;
     Inisem(video,val);
      Inicializamos os Semaforos        }
     nbct := 7;
     tambct := 1024;
     nivpri := 4;
     stack := -20480;
     stack := -17408;        }
     roterr := 0;
     Inintr(nbct,tambct,nivpri,stack,roterr);
      Inicializamos o Nucleo            }
```

```
Igs;
Start2(endt2);
Start3(endt3);
Start4(endt4);
Start5(endt5);
Start6(endt6);
Start7(endt7);
{                    Obtivemos os enderecos das tarefas          }
GSP := 0.0;
GaSP := 0.0;
GPV := 0.0;
GVO := 0.0;
Gmanaut := 'm';
Gvomax := 80.0;
Gvomin := -80.0;
Gpont := 0;
Gvoman := 0.0;
Gkp := 3.2454;
Gti := 32.4631;
Gtd := 11.4267;
Gt := 6.0;
Gonoff := 'n';
Gparest := 'n';
Gpid := 'o';
GEkp := 3.2454;
GEti := 32.4631;
GEtd := 11.4267;
Gtraco := 0.0;
Glamb := 0.00995;
Ga1 := -1.5;
Ga2 := 0.7;
Gb0 := 0.2;
Gb1 := 0.02;
Gc1 := 1.0;
Gc2 := 0.13;
Gpsrb := 0.0;
Gonps := 'n';
Gvarr := 0.12;
Gkk1 := 0.1;
Gkk2 := 0.14510;
Gsigma1 := -0.58766;

                  Inicializamos variaveis do processo          }
priori := 1;
endtsk := endt2;
Start(endtsk, priori);
priori := 1;
endtsk := endt3;
Start(endtsk, priori);
priori := 1;
endtsk := endt4;
Start(endtsk, priori);
priori := 3;
endtsk := endt5;
Start(endtsk, priori);
```

```
priori := 2;
endtsk := endt6;
Start(endtsk, priori);
priori := 3;
endtsk := endt7;
Start(endtsk, priori);
                Disparamos todas as tarefas                    )
Stop;
End.
```

```
EXTERNAL TESE3::SUBT44(4);



Procedure Task2;

    Var loop: boolean;
        aa: integer;

    BEGIN
            loop := false;
            aa := ROUND(6.0/0.0164);
            REPEAT
                    Signt(acess0);
                    Delay(aa);
            UNTIL loop;
    END;
```

```
                        END;
                'v':    BEGIN
                                Kp := GVkp;
                                Ti := GVti;
                                Td := GVtd;
                        END;
                END;
                Signt(acess2);

                g0 := kp * ( (td/T) + (T/ti) + 1.0 );
                g1 := -kp * (( 2 * td / T ) + 1.0);
                g2 := kp * td / T;

                PV2t := PV1t;
                PV1t := PV;
                VO1t := VO;

                IF (Manaut = 'a') AND (OnOff = 'y') THEN
                   BEGIN
                        auxquad := SIN(0.0571199*pont);
                        pont := pont +1;
                        IF (auxquad >= 0.0) THEN
                           SP := aSP
                        ELSE
                           SP := -aSP;
                   END
                ELSE
                   SP := aSP;
{                  SP := 0.0;                   }

{ Esta parte do programa gera o ruido  }


IF (OnOff = 'y') THEN
        Begin
                RANDOM(semente, uu);
                IF (uu > 0.5) THEN
                        psrb := modps
                ELSE
                        psrb := -modps;
        End;

                SIGNT(acess1);


        End;
      UNTIL loop;

 END; {   de task3   }
```

```
   WAIT(acessO);

valda := 2048;
cda := 2;
Escda(valda, cda);

cad := 0;
Lead(valad, cad);
PV := valad * 170.0 / 4095.0;
PV := PV - 85.0;

IF (ManAut = 'm') THEN
        VO := VOman
ELSE
     VO := VO1t + kp*T/ti*SP - g0*PV - g1*PV1t - g2*PV2t;
     VO := VO + psrb;

IF (VO > VOmax) THEN
        VO := VOmax;
IF (VO < -VOmax) THEN
        VO := -VOmax;

VOp := VO + 85.0;
VOp := VOp * 4095.0 / 170.0;
valda := ROUND(VOp);
cda := 0;
Escda(valda,cda);

valda := 0;
cda := 2;
Escda(valda, cda);

Wait(acess2);
T := Gt;
VOmax := GVOmax;
OnOff := Gonoff;
OnPsrb := Gonps;
aSP := GSP;
GaSP := SP;
modps := Gpsrb;
ManAut := Gmanaut;
CPV := PV;
GVO := VO;
VOman := Gvoman;
Par := Gpid;
CASE Par OF
'o':        BEGIN
                    Kp := Gkp;
                    Ti := Gti;
                    Td := Gtd;
            END;
'e':        BEGIN
                    Kp := GEkp;
                    Ti := GEti;
                    Td := GEtd;
```

```
EXTERNAL TESE3::SUBT66(1);


Procedure Task3;            {$f- ,m- ,r- ,u- desable  overflow errors }

     Var  loop: boolean;
          PV, VO, SP, aSP, VOman, Kp, Ti, Td, VOmax, VOp: real;
          ManAut, Par, OnOff, OnPsrb: char;
          modps, psrb, T, PV1t, PV2t, VO1t, g0, g1, g2, auxquad, uu: real;
          valda, cda, cad, valad, pont, semente:  integer;

PROCEDURE random(var n:integer; var u: real);

     BEGIN {random}
          n := n * 12869;
          IF (n > 25843) THEN {prevent overflow on add}
              BEGIN
                   n := n - 32767 - 1;
              END; {if}
          n := n + 6925;
          IF (n < 0) THEN     {handle overflow by adding modulus}
              n := n + 32767 + 1;
          u := n * 3.0517578e-05;
     END;  {random}

     BEGIN   (  de task3  )

          Wait(acess2);
          SP := GSP;
          VOmax := Gvomax;
          T := Gt;
          Kp := Gkp;
          Td := Gtd;
          Ti := Gti;
          ManAut := Gmanaut;
          OnOff := Gonoff;
          VOman := Gvoman;
          Signt(acess2);

          g0 := kp * ( (td/T) + (T/ti) + 1.0 );
          g1 :=  -kp * (( 2 * td / T ) + 1.0);
          g2 := kp * td / T;
          semente := 345;
          psrb := 0.0;
          modps := 0.0;
          PV1t := 0.0;
          PV2t := 0.0;
          VO1t := 0.0;
          pont := 1;
          loop := false;
          REPEAT
            Begin
```

```
ERNAL  TESE3::SUBT88(3);

CEDURE  TASK4;

 LABEL 999, 995, 993, 990, 950;

 VAR  P, Timer1, Timer2: array[1..4,1..4] of real;
      temp1, temp2, temp3, erro, eps1, vareps1, varaux, Po, lambda: real;
      Theta, Phi, L, G, F: array[1..4] of real;
      sigma1, sigma2, g0, g1, g2, Kp, Ti, Td, Kt, T, kk1, kk2, tracop: real;
      aux1, aux2, aux3, aux4, PV, VO, SP, d1, d2, omega, Ts, kisi: real;
      i, j, pont, aa, valad, cad, valda, cda: integer;
      t1, t2, t3, t4, doublim,liminf, medtra, lambdo, medaux, medeps1: real;
      onoff, parest, onpsrb: char;
      loop : boolean;
      ni, ist, j1, kk, auxii: integer;
      auxtra1, auxtra2, alfa, beta, gamma, dd, s, al, v: real;
      atext,btext,ctext,dtext,etext,ftext,gtext,htext,itext,jtext: text;


      BEGIN
( Devemos inicializar (ou o default de) vetores, matrizes e parametros  )
            Ts := 150.0;
            kisi := 0.7;
            omega := 4.6/(Ts * kisi); (erro a 1%  em 150 seg.)
            varaux := 0.0;
            medaux := 0.0;
            lambda := 0.99;
            Po := 1000.0;
            liminf := 4000.0;
            onpsrb := 'n';
            t1 := 0.0;
            t2 := 0.0;
            t3 := 0.0;
            t4 := 0.0;
            auxtra2 := 0.0;
            auxtra1 := 0.0;

            FOR i := 1 TO 4 DO
                  BEGIN
                        Theta[i] := 1.0;
                        Phi[i] := 0.0;
                        L[i] := 0.0;
                        FOR j := 1 TO 4 DO
                              BEGIN
                                    Timer1[i,j] := 0.0;
                                    Timer2[i,j] := 0.0;
                                    IF i = j THEN
                                          P[i,j] := Po
                                    ELSE
                                          P[i,j] := 0.0;
                              END
                  END;
            pont := 0;
            loop := false;
```

```
        REPEAT

                WAIT(acess1);
                valda := 2048;                                  }
                cda := 2;                                       }
                Escda(valda, cda);                              }

                Wait(acess2);
                SP := GaSP;
                PV := GPV;
                VO := GVO;
                t := Gt;
                lambdo := Glamb;
                onoff := Gonoff;
                Signt(acess2);

                IF (onoff = 'n') THEN
                        goto 999;

  {     Calcular o erro da predicao     }

                erro := PV;
                FOR i := 1 TO 4 DO
                        erro := erro - Phi[i] * Theta[i];

Atualizar as estimativas dos parametros, calcular o residual       }

                eps1 := PV;
                FOR i := 1 TO 4 DO
                BEGIN
                   Theta[i] := Theta[i] + L[i] * erro;
                   eps1 := eps1 - Phi[i] * Theta[i];
                END;

  {       Atualizar       o       vetor       Phi           }

                Phi[2] := Phi[1];
                Phi[1] := -PV;
                Phi[4] := Phi[3];
                Phi[3] := VO;


  {     Computar o vetor de ganho L, atualizar P e v       }

              FOR i := 2 TO 4 DO
                   BEGIN
                        j := 4 + 2 - i;
                        alfa := Phi[j];
                        j1 := j - 1;
                        FOR kk := 1 TO j1 DO
                            alfa := alfa + P[kk,j] * Phi[kk];
                        F[j] := alfa;
                        G[j] := P[j,j] * alfa;
                   END;
```

```
                C[1] := P[1,1] * Phi[1];
                F[1] := Phi[1];

                alfa := lambda + F[1] * C[1];
                gamma := 0;
                IF (alfa > 0.0) THEN
                    gamma := 1/alfa;
                IF (C[1] <> 0.0) THEN
                    P[1,1] := gamma * P[1,1];

    Aqui comeca o loop   1                )

                        j := 2;
                        993:

                        beta := alfa;
                        dd := C[j];
                        alfa := alfa + dd * F[j];

                        IF (alfa = 0.0) THEN
                                goto 995;

                        al :=  F[j] * gamma;
                        j1 := j - 1;
                        FOR i := 1 TO j1 DO
                            BEGIN
                                s := P[i,j];
                                P[i,j] := s + al * C[i];
                                C[i] := C[i] + dd * s;
                            END;
                        gamma := 1.0/alfa;
                        P[j,j] := beta*gamma*P[j,j]/lambda;

                        995:
                        j := j + 1;
                        IF (j < 5) THEN    (   loop  1   )
                                goto 993;


   (   *     atualizar o vetor de ganho, L[ ].     *   )
(              Computar o traco da matriz P[ ]              )
                  tracop := 0.0;

  (             Aqui   comeca    o     loop     2          )
                  i := 1;
                  990:

                       L[i] := G[i]/alfa;
             (.        tracop := tracop + P[i,i]/( Theta[i] * Theta[i]  );)
                       tracop := tracop + P[i,i];

                       i := i + 1;
                       IF (i < 5) THEN      (   loop   2    )
                            goto 990;
```

```
{     Ligar ou nao o sinal PSBR adicionado ao controle    }

                IF (pont < 20) THEN
                        goto 950;
                medtra :=  (t1 + t2 + t3 + t4 + tracop)/5.0;
                IF (medtra > doublim) THEN
                        onpsrb := 'y'
                ELSE
                        onpsrb := 'n';

                950:
                t4 := t3;
                t3 := t2;
                t2 := t1;
                t1 := tracop;

                IF (liminf > tracop) THEN
                        liminf := tracop;
                doublim := 2.0 * liminf;

                IF (onpsrb = 'y') THEN
                        auxtra1 := 777.0
                ELSE
                        auxtra1 := 0.0;

{          Calcular a media e a variancia de Eps1          }

                medaux := medaux + eps1;
                varaux := varaux + eps1*eps1;
                pont := pont + 1;
                medeps1 := medaux / pont;
                vareps1 := varaux / pont - medeps1 * medeps1;

{       Alterar       o       valor       de       lambda       }

        lambda := 0.99 * lambda + lambdo;

{       Transmitir parametros, testar permissoes, etc...       }

       d1 := 2.0*EXP(-kisi*omega*T)*COS(omega*T*SQRT(1-kisi*kisi));
       d2 := EXP(-2.0 * kisi * omega * T);

       kk1 := Theta[4]/Theta[3];
       aux1 := Theta[3] * Theta[2] / Theta[4];
       aux2 := Theta[4] / Theta[3] * (d1 - Theta[1] + 1);
       aux3 := Theta[3] * d1 / Theta[4];
       aux4 := (Theta[3] * Theta[3] * d2) / (Theta[4] * Theta[4]);
       kk2 := -(Theta[2]-Theta[1]+aux1+aux2-d2)/(aux3-aux4-1);

       if (kk1*kk1+kk2) < 0
             then sigma1 := 0.0;
       if (kk1*kk1+kk2) >= 0 then
                 begin
                      aux3 := sqrt(kk1*kk1+kk2);
```

```
                                aux1 := 2*kk1 - 2*(aux3);
                                aux2 := 2*kk1 + 2*(aux3);
                                aux3 := abs(aux1);
                                aux4 := abs(aux2);
                                if aux3 < aux4
                                   then sigma1 := aux1
                                   else sigma1 := aux2;
                             end;
                sigma2 := kk1*sigma1 + kk2;
                g0 := (d1 + sigma1 - Theta[1] + 1)/Theta[3];
                aux3 := d2 * sigma1 + d1 * sigma2 + Theta[2];
                aux4 := (Theta[3] * d2 * sigma2) / (Theta[4] * Theta[4]);
                g1 := aux3 / Theta[4] - aux4;
                g2 := (d2 * sigma2)/Theta[4];
{       Parametros para  PID com  integrador trapezoidal          }
{               td :=   (2*g2*t)/(g0 - g1 - 3*g2);                }
{               kp :=   (g0 - g1 - 3*g2)/2;                       }
{               ti :=   (t*(g0 - g1 - 3*g2))/(2*(g0 + g1 + g2)); }

                td := g2 * t / (-2.0 * g2 - g1);
                kp := g1 / (-2.0 * td/t - 1.0);
                ti := t / (g0/kp - td/t - 1.0);
                kt := 1/(1 + sigma1 + sigma2);

  REWRITE('LST:', atext);
writeln(atext,theta[1]);

  REWRITE('LST:', btext);
writeln(btext,theta[2]);

 REWRITE('LST:', ctext);
writeln(ctext,theta[3]);

 REWRITE('LST:', dtext);
writeln(dtext,theta[4]);

 REWRITE('LST:', etext);
{       writeln(etext,theta[5]);        }
writeln(etext,auxtra1);

 REWRITE('LST:', ftext);
{       writeln(ftext,theta[6]);        }
writeln(ftext,auxtra2);

 REWRITE('LST:', gtext);
writeln(gtext,PV);

 REWRITE('LST:', htext);
writeln(htext,VO);

 REWRITE('LST:', itext);
writeln(itext,SP);

 REWRITE('LST:', jtext);
writeln(jtext,tracop);
```

```
        IF (onoff =  'y')  THEN
                   BEGIN
                   Wait(acess2);
                   GEKp := Kp;
                   GETd := Td;
                   GETi := Ti;
                   Ca1 := Theta[1];
                   Ca2 := Theta[2];
                   Cb0 := Theta[3];
                   Cb1 := Theta[4];
{                  Cc1 := Theta[5];                          }
{                  Cc2 := Theta[6];                          }
                   Cc1 := auxtra1;
                   Cc2 := auxtra2;
                   Conps := onpsrb;
                   Cvarr := vareps1;
                   Ckk1 := kk1;
                   Ckk2 := kk2;
                   Csigma1 := sigma1;
                   Ctraco := tracop;
                   Cpont := pont;
                   Signt(acess2);
                   END;

                   999:
                   valda := 0;                                 }
                   cda := 2;                                   }
                   Escda(valda, cda);                          }

          UNTIL loop;
 END;                    {   da tarefa de Estimacao   }
```

```
EXTERNAL TESE3::SUBT99(2);

PROCEDURE TASK5;


LABEL  998;

VAR  sigma1,sigma2,d1,d2,t1,t2,t3,t4,a1,a2,b0,b1,c1,c2,g0,g1,g2 :real;
     vary,varu,varr,kk1,kk2,t,te,kisi,omega,kp,ti,td: real;
     mt,m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12,m13,m14,m15,m16,m17:real;
     m18,m19,m20,m21,m22,m23,x1,x2,x3,alfa0,alfa1,alfa2,alfa3,alfa4:real;
     l,aa: integer;
     onoff, parvar: char;
     loop: boolean;


     BEGIN

          te:= 150.0;
          kisi := 0.7;
          omega := 4.6/(te*kisi);

          loop := false;
          REPEAT
                    Wait(acess2);
                    parvar := Cparest;
                    a1 := Ca1;
                    a2 := Ca2;
                    b0 := Cb0;
                    b1 := Cb1;
                    c1 := Cc1;
                    c2 := Cc2;
                    varr := Cvarr;
                    kk1 := Ckk1;
                    kk2 := Ckk2;
                    sigma1 := Csigma1;
                    t := Ct;
                    Signt(acess2);

                    IF  (parvar = 'n') THEN
                          goto 998;

               d1 := -2*exp(-kisi*omega*t)*cos(omega*t*sqrt(1-kisi*kisi));
                    d2 := exp(-2*kisi*omega*t);
                    sigma2 := kk1*sigma1 + kk2;
                    t1 := d1 + sigma1;
                    t2 := d2 + sigma2 + d1*sigma1;
                    t3 := d2*sigma1 + d1*sigma2;
                    t4 := d2*sigma2;

                    g0 := (d1 + sigma1 - a1 + 1)/b0;
          g1 := (d2*sigma1 + d1*sigma2 + a2)/b1 - (b0*d2*sigma2)/(b1*b1);
```

```
                g2 := (d2*sigma2)/b1;

{       Parametros para PID provido com integrador trapezoidal      }
{               TD := (2*g2*t)/(g0 - g1 - 3*g2);                    }
{               KP := (g0 - g1 - 3*g2)/2;                           }
{               TI := (t*(g0 - g1 - 3*g2))/(2*(g0 + g1 + g2));      }

                td := g2 * t / ( 2.0 * g2 - g1);
                kp := g1 / ( 2.0 * td/t - 1.0);
                ti := t / (g0/kp - td/t - 1.0);

                x1 := 1;
                x2 := -1;
                x3 := 0;
                for L := 1 to 2 do
                        begin
                                if L = 2
                                then
                                        begin
                                                x1 := g0;
                                                x2 := g1;
                                               x3 := g2;
                                        end;
                                alfa0 := x1;
                                alfa1 := x1*c1 + x2;
                                alfa2 := x3 + x1*c2 + x2*c1;
                                alfa3 := x2*c2 + x3*c1;
                                alfa4 := x3*c2;
                                m1 := alfa0;
                                m2 := -t1*alfa0 + alfa1;
                                m3 := -t1*m2 - t2*m1 + alfa2;
                                m4 := -t1*m3 - t2*m2 - t3*m1 + alfa3;
                        m5 := -t1*m4 - t2*m3 - t3*m2 - t4*m1 + alfa4;
                                m6 := alfa4*m1;
                                m7 := alfa3*m1 + alfa4*m2;
                                m8 := alfa2*m1 + alfa3*m2 + alfa4*m3;
                m9 := alfa1*m1 + alfa2*m2 + alfa3*m3 + alfa4*m4;
                                m10 := t4*t3 - t1;
                                m11 := -t4*m7 + m9;
                                m12 := 1 + t2 - t4*(t2 + t4);
                                m13 := t1*t4 - t3;
                                mt := m12 + (m13*(t1 + t3))/(1 + t4);
                                m14 := (( -m13*t2)/(1 + t4) + m10)/mt;
                                m15 := ((m13*m8)/(1 + t4) + m11)/mt;
                                m16 := ( -(t1 + t3)*m14 - t2)/(1 + t4);
                                m17 := (- (t1 + t3)*m15 + m8)/(1 + t4);
                                m18 := -t1*m16 - (t2 + t4)*m14 - t3;
                                m19 := -t1*m17 - (t2 + t4)*m15 + m7;
                                m20 := -t1*m18 - t2*m16 - t3*m14 - t4;
                                m21 := -t1*m19 - t2*m17 - t3*m15 + m6;
                    m22 := -t1*m15 - t2*m17 - t3*m19 - t4*m21 + alfa0*m1;
                m22 := m22 + alfa1*m2 + alfa2*m3 + alfa3*m4 + alfa4*m5;
                        m23 := 1 + t1*m14 + t2*m16 + t3*m18 + t4*m20;

                                if L = 1
```

```
                                                then vary := (m22/m23)*varr

                                        else varu := (m22/m23)*varr;
                                end;

                                        Wait(acess2);
                                        Cvaru := varu;
                                        Cvary := vary;
                                        CVkp := kp;
                                        CVti := ti;
                                        CVtd := td;
                                        Signt(acess2);

                        998:
                        aa := ROUND(t/0.0164);
                        Delay(aa);
                UNTIL loop;

    END;                        (  da tarefa Variancia  )



    PROCEDURE  TASK6;

        VAR loop: boolean;
            e:char;

        BEGIN
                e := chr(27);
                Wait(video);
                write(e,'[2J');
                write(e,'[1;2H');
write('·····PV····· ·····VO····· ·····SP····· ·····AM····· ·····L/D····');
                write('  ··PARest···');
                write(e,'[4;2H');
write('·····KP····· ·····TI····· ·····TD····· ·Time Amost· ···Lambdo···');
                write('  ··ParPid···');
                write(e,'[7;2H');
write('··Theta[1]·· ··Theta[2]·· ··Theta[3]·· ··Theta[4]·· ··Theta[5]··');
                write('  ··Theta[6]··');
                write(e,'[10;2H');
write('··Contador·· ···KPest····· ···TIest····· ···TDest····· ···Traco····  ');
                write(e,'[13;2H');
write('···[PSRB]··· ····Varu····· ····Vary····· ····Varr····· ···Sigma1····  ');
                write(e,'[16;14H');
                write('···KPvar····· ···TIvar····· ···TDvar····· ···Sigma1····  ');
                write(e,'[19;8H');
   write('As opcoes sao: VO(0), SP(1), KP(2), TI(3), TD(4), TA(5), VOmax(6),'
                write(e,'[20;8H');
 write('Lambdo(7), Sigma1(8), [PSRB](9), MA(10), LD(11), PE(12), ParPid(13).'
                Wait(acess2);
                write(e,'[2;26H',GSP:9:4);
                write(e,'[2;46H',Cmanaut);
                write(e,'[2;59H',Gonoff);
```

```
                write(e,'[2;72H',Cparest);
                write(e,'[5;39H',Ct:9:4);
                write(e,'[5;52H',Clamb:9:4);
                write(e,'[5;72H',Cpid);
                write(e,'[14;1H',Cpsrb:9:4);
                Signt(acess2);
                Signt(video);
                loop := false;
                REPEAT
                                Wait(video);
                                write(e,'7');
                                write(e,'[2;1H');
                                Wait(acess2);
                                write(GPV:9:4,'      ',GVO:9:4);
                        write(e,'[5;1H',Ckp:9:4,'      ',Cti:9:4,'      ',Ctd:9:4),
                                IF (Gonoff = 'y')  THEN
                                        BEGIN
write(e,'[8;1H',Ca1:9:4,'     ',Ca2:9:4,'     ',Cb0:10:5,'     ',Cb1:10:5);
                                write('     ',Gc1:9:4,'     ',Gc2:9:4);
write(e,'[11;1H',Cpont:3,'     ',CEkp:9:4,'      ',CEti:9:4,'      ',CEtd:9:4);
                                write('     ',Gtraco);
                                        END;
                                IF  (Gparest = 'y') THEN
                                        BEGIN
        write(e,'[14;13H',Gvaru:9:4,'     ',Gvary:9:4,'      ',Gvarr:9:4,'     ');
        write(Gsigma1:9:4);
write(e,'[17;12H',GVkp:9:4,'     ',GVti:9:4,'      ',GVtd:9:4,'    ',Gsigma1:9:4)
                                        END;
                                Signt(acess2);
                                write(e,'8');
                                Signt(video);
                                Delay(61);
              UNTIL loop;
         END;

    PROCEDURE TASK7;

    VAR  loop: boolean;
         e, aux2: char;
         help: integer;
         cons1: text;
         aux1: real;

        BEGIN    (  de Task5   )
                 e := chr(27);
                 reset('CON:',cons1);
                 loop := false;
                 REPEAT
                         Wait(video);
                         write(e,'[23;15H',e,'[2K');
                         write('SELECIONE UMA OPCAO :');
                         Signt(video);
                         readln(cons1,help);
                         Wait(video);
                         write(e,'[23;15H',e,'[2K');
```

```
                write('DEFINA UM VALOR          :');
                Signt(video);
                IF help < 10 THEN
                        readln(cons1,aux1)
                ELSE
                        readln(cons1,aux2);
                Wait(acess2);
                CASE help OF
                  0:    CVOman := aux1;
                  1:    GSP := aux1;
                  2:    CKp := aux1;
                  3:    GTi := aux1;
                  4:    CTd := aux1;
                  5:    GT := aux1;
                  6:    CVOmax := aux1;
                  7:    Glamb := aux1;
                  8:    Csigma1 := aux1;
                  9:    Cpsrb := aux1;
                 10:    Cmanaut := aux2;
                 11:    Gonoff := aux2;
                 12:    Cparest := aux2;
                 13:    Cpid := aux2;
                END;
                Signt(acess2);

                Wait(video);
                CASE help OF
                  1:  write(e,'[2;26H',aux1:9:4);
                  5:  write(e,'[5;39H',aux1:9:4);
                  7:  write(e,'[5;52H',aux1:9:4);
                  9:  write(e,'[14;1H',aux1:9:4);
                 10:  write(e,'[2;46H',aux2);
                 11:  write(e,'[2;59H',aux2);
                 12:  write(e,'[2;72H',aux2);
                 13:  write(e,'[5;72H',aux2);
                END;
                Signt(video);

        UNTIL loop;
END; ( de task5 )
```