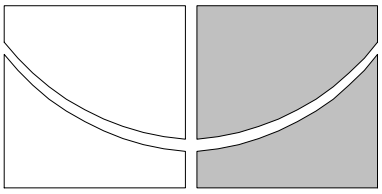
**UNIVERSIDADE DE BRASÍLIA**
**FACULDADE DE TECNOLOGIA**
**DEPTO. DE ENGENHARIA ELÉTRICA**



# CONTROLE AXIAL DE JUNTAS ROTACIONAIS EM MANIPULADORES ROBÓTICOS: IMPLEMENTAÇÃO COM MICROCONTROLADORES

***Rudi Henri van Els***

Dissertação apresentada ao Departamento de Engenharia Elétrica da Universidade de Brasília, como requisito parcial à obtenção do grau de Mestre em Engenharia Elétrica.
Orientador: Prof. Dr. Victor Hugo Casanova Alcalde.

Brasília
1994

**UNIVERSIDADE DE BRASÍLIA**
**FACULDADE DE TECNOLOGIA**
**DEPTO. DE ENGENHARIA ELÉTRICA**

# CONTROLE AXIAL DE JUNTAS ROTACIONAIS EM MANIPULADORES ROBÓTICOS: IMPLEMENTAÇÃO COM MICROCONTROLADORES

***Rudi Henri van Els***

Dissertação apresentada ao Departamento de Engenharia Elétrica da Universidade de Brasília, como requisito parcial à obtenção do grau de Mestre em Engenharia Elétrica.
Orientador: Prof. Dr. Victor Hugo Casanova Alcalde.

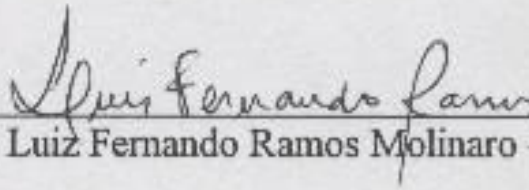
Brasília
1994

Dissertação defendida e aprovada pela banca examinadora constituída pelos professores:

**BANCA EXAMINADORA:**

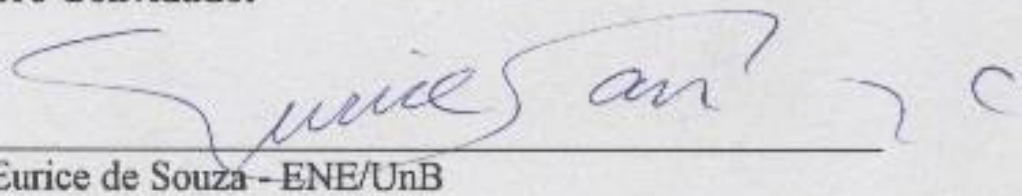
Prof. Dr. Victor Hugo Casanova Alcalde - ENE/UnB (Presidente)

Prof. Dr. Sadek Crisóstomo Absi Alfaro - ENE/UnB (Membro)



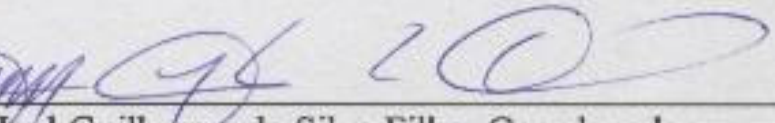
Prof. Dr. Luiz Fernando Ramos Molinaro - ENE/UnB (Membro)

**Membro Convidado:**



Prof. Eurice de Souza - ENE/UnB

Vista e permitida a impressão:



Prof. Dr. Joel Guilherme da Silva Filho, Coordenador
Curso de Pós-graduação de Engenharia Elétrica

Brasília, Setembro de 1994

# Agradecimentos

Agradeço

- ao prof. V.H. Casanova pela orientação.
- aos colegas do Grupo de Automação e Controle em Processos de Fabricação
- à C&D Informática, pela confecção das placas de circuito impresso.
- à CAPES pelo apoio financeiro durante o curso.
- à todos que de alguma forma contribuiram para a realização deste trabalho.
- a minha esposa e meus filhos, a quem dedico este trabalho.

# Dedicatória

A Kika, por acreditar e incentivar, a Allen e Pedro Paulo, pelos momentos de alegria e distração.

# Lista de figuras

# Lista de tabelas



# Lista de algoritmos

# Resumo

O controle de junta de manipuladores robóticos usando estrutura hierárquica de controle, executado por um computador supervisor e microcontrolador de junta, é estudado neste trabalho. O problema de controle global do robô, o planejamento de trajetórias e o controle de movimento, assim como a dinâmica do robô são discutidas e as atribuições do computador supervisor e controlador de junta definidas. Um sistema de desenvolvimento para o microcontrolador MC8031 (Intel) foi construído para facilitar o projeto do controlador de junta. Para estudar problema de controle, um manipulador de um grau de liberdade foi construído, com a montagem de um braço mecânico num servomotor educacional. O estudo prosegue com a discussão das características do servomotor e sua dinâmica, chegando à construção de uma unidade não-linear para compensar uma das características não-lineares do motor. Quatro estratégias de controle (PD, PID, PD com torque computado e Controle por estrutura variável), são projetos e implementados numa linguagem de alto nível. No final, os resultados das diferentes estratégias de controle, obtidos por simulação em computador e realização física são discutidos. É mostrado que o *hardware* proposto é adequado como controlador de junta.

# Abstract

The joint control of robot manipulators using a hierarchical control scheme carried out by a supervisor computer and joint micro controllers, is studied in this dissertation. The global robot control problem, the trajectory planning and motion control, as well as robot dynamics are discussed and the attributions of the supervisor computer and joint controllers defined. A development system for Intel's micro controller MC8031 was built in order to ease the design the joint controller. For the study of the control problem, a manipulator with one degree of freedom was built, with the assembling of a mechanical arm on an educational servo motor. The study continues with the discussion of the servo motor characteristics and its dynamics, leading to the construction of a nonlinear unit to compensate some of the motor's nonlinear characteristics. Four control strategies (PD, PID, Computed Torque PD and Variable Structure Control), were designed and implemented in a high level computer language. At last, the result of the different control strategies, obtained by computer simulation and real world implementation are discussed. It is shown that the *hardware* proposed is suitable as a joint controller.

# Índice

# CAPÍTULO 1. INTRODUÇÃO

O estudo e desenvolvimento de mecanismos robóticos teve o seu primeiro registro significativo nos anos quarenta, quando os primeiros teleoperadores foram fabricados para manipular materiais radioativos. O primeiro robô comercial controlado por computador foi introduzido por Unimation Inc. no final dos anos cinqüenta (Fu e Gonzales,1987).

Os primeiros robôs eram controlados por computadores de processos ou minicomputadores onde todo o controle era feito de modo centralizado. O uso de microprocessadores ou microcontroladores nestas configurações era limitado para ser uma extensão do sistema central. Por exemplo, o controle do robô PUMA 560 (Fu e Gonzales,1986) é realizado por um minicomputador que comanda seis servo controladores, um para cada junta. O servo controlador de junta é implementado por microprocessadores (ROCKWELL 6503) que tinham como principal função compatibilizar os sinais digitais com os sinais analógicos.

Com o desenvolvimento de microprocessadores mais velozes e com mais recursos foi possível descentralizar o controle do robô. O robô IRB2000 do Asea Brown Boveri (ABB, 1993) tem um sistema de controle formado por 3 microprocessadores de 16 bits (MOTOROLA 68000) chamados de computador principal, servo computador e computador de entrada e saída, e um processador digital de sinais (Texas DSP TMS320C25) denominado computador axial. O servo computador implementa o controle dos seis motores de corrente alternada do robô em malha fechada. Para isto, ele conta com o computador axial, que tem por função processar os sinais de controle e gerar os sinais de referência para os acionadores dos seis motores. Outra função do computador

axial é processar os sinais de realimentação para o servo computador. O computador de entrada e saída fica encarregado de fazer a interface com o mundo externo e o computador principal faz o gerenciamento de todo o sistema.

Aqui pode-se constatar uma estrutura hierárquica entre o controle do manipulador efetuado pelo servo computador e a supervisão efetuado pelo computador principal. Ressalva-se ainda que o controle de todos os seis graus de liberdade do manipulador é efetuado por um microprocessador.

O desenvolvimento de novas estratégias de controle possibilitará a descentralização do controle. Assim será possível utilizar para cada junta um microprocessador ou microcontrolador e a coordenação dos chamados controladores de junta ficará a cargo de um supervisor. A primeira vista, não parece uma alternativa vantajosa por aumentar o número de microprocessadores, mas, ao analisar, por exemplo, o robô IRB2000, observa-se que apesar de o servo computador controlar todos os seis motores, o acionamento dos mesmos é efetuado por um controlador proporcional integral discreto usando modulação por largura de pulsos. A vantagem da nova alternativa é que o acionamento e controle do motor também pode ser efetuado diretamente pelo controlador de junta.

Neste trabalho, trata-se do controle do robô de modo descentralizado, dividindo o sistema de controle em dois níveis hierárquicos. Propõe-se sua implementação com um computador supervisor e microcontroladores para o controle axial de junta.

Durante a pesquisa bibliográfica foi constatado que há poucos trabalhos relacionados à implementação de *Hardware* de controle de manipuladores robóticos. O trabalho de Medeleck e Zampanie (1990) tem uma estrutura na mesma linha adotada aqui, eles descrevem um sistema supervisor desenvolvido num micro computador PC-AT ligado a placas controladores de juntas baseados no microprocessador 8085. A transferência de dados para as placas até o PC-AT é

efetuada através de um barramento de dados usando a facilidade de acesso direto à memória "DMA". Um outro trabalho (Madrid e Palhares, 1988) descreve um sistema digital para movimentar manipuladores mecânicos baseado num microprocessador Z80 usando técnicas de acionamento MAP (Modulação por Amplitude de Pulso) e MLP (Modulação por Largura de Pulso).

Quanto ao algoritmo de controle, os trabalhos pesquisados limitam-se a simulações de novas estratégias de controle. O trabalho de Guenther (1990) compara várias estratégias de controle (PD, PID, FeedForward, Adaptativa e Estrutura Variável) para um manipulador de dois graus de liberdade. Hsia (1991) propõe um algoritmo de fácil implementação para o controle independente de juntas de robôs industriais e simula o sistema para o robô PUMA 560.

Sobre a comunicação entre o sistema central e os controladores locais não foi possível encontrar material específico para nosso propósito. O tipo de comunicação e o protocolo usado determinarão o desempenho geral do sistema de controle.

O objetivo deste trabalho é o desenvolvimento de uma estrutura de *Hardware* para o controlador de junta e a avaliação do desempenho deste *Hardware* para algumas estratégias de controle, aplicado ao caso de um braço manipulador robótico de um grau de liberdade. Os princípios que seguimos na elaboração do sistema foram os seguintes:

- *Hardware* de fácil aquisição: (micro controladores, conversores)
- O sistema deve ter um *Hardware* universal, ou seja, o projeto do sistema deve ser feito tendo em mente a possibilidade de controlar também outros tipos de processos sem maiores modificações.

- Deve ter as características de um ambiente de trabalho para o projeto de controladores digitais, diferentemente de um produto acabado.
- Os algoritmos de controle e os programas de suporte devem ser implementados numa linguagem de alto nível, para facilitar posteriormente a portabilidade para controladores mais elaborados.
- E, principalmente, construir o protótipo e verificar o seu funcionamento.

No capítulo 2 apresenta-se o problema de controle global de um manipulador robótico. A estrutura do sistema de controle a ser usada no trabalho é apresentada e o controle de trajetória e movimento são abordados. O sistema de desenvolvimento para microcontroladores que foi projetado para poder implementar o controlador de junta é descrito no capítulo 3. O *Hardware* e *Software* que compõem este sistema são detalhadas. No final deste capítulo são colocadas algumas considerações quanto ao *Software* de desenvolvimento usado para implementar o controlador de junta. O capítulo 4 faz uma análise do acionamento eletromecânico para juntas rotacionais de manipuladores robóticos. O motor de corrente contínua usado no trabalho é apresentado e seu modelo matemático formulado. A interligação do motor com o *Hardware* do controlador é discutida a seguir e no final do capítulo é apresentado o manipulador de um grau de liberdade com seu respectivo modelo matemático. O capítulo 5 trata do desenvolvimento e implementação do controlador de junta robótica. Neste capítulo, quatro estratégias de controle são abordadas e implementadas. Os resultados experimentais obtidos com os controladores são apresentados no capítulo 6 e no final do mesmo apresentadas as conclusões finais.

# CAPÍTULO 2. PROBLEMA DE CONTROLE GLOBAL DE UM MANIPULADOR ROBÓTICO

Um robô pode ser definido, (Schilling, 1990) como um dispositivo mecânico controlado por *software*, que usa sensores para orientar uma ou mais ferramentas através de movimentos programados que objetivam manipular objetos físicos dentre de um espaço. Do ponto de vista da sua construção mecânica, ele pode ser representado como uma cadeia de elos rígidos interligados por juntas flexíveis. Estas juntas podem ser rotacionais ou prismáticas.

Muitos dos robôs atualmente em uso são antropomórficos, no sentido que se assemelham a um braço humano. Conseqüentemente, por analogia, ao analisar a sua estrutura, pode-se denominar os elos do robô, de tronco, braço, antebraço e mão, e denominar as juntas flexíveis, de ombro, cotovelo e pulso como mostrado na figura 2.1.

## 2.1 DESCRIÇÃO DO PROBLEMA DE CONTROLE

O robô pode ser dividido, segundo Mendes e Camanho (1989), em quatro subsistemas funcionais:

- O subsistema de manipulação formado pelo manipulador e a ferramenta.
- O subsistema de sensoreamento externo (sensores externos e sistema de comunicação.
- O subsistema de interface homem-máquina (sistema de programação e controle de tarefas).

- O subsistema de (servo)controle (atuador, sensores de realimentação das juntas, mecanismos de transmissão de energia, controlador e *software* de controle de trajetória e movimento.

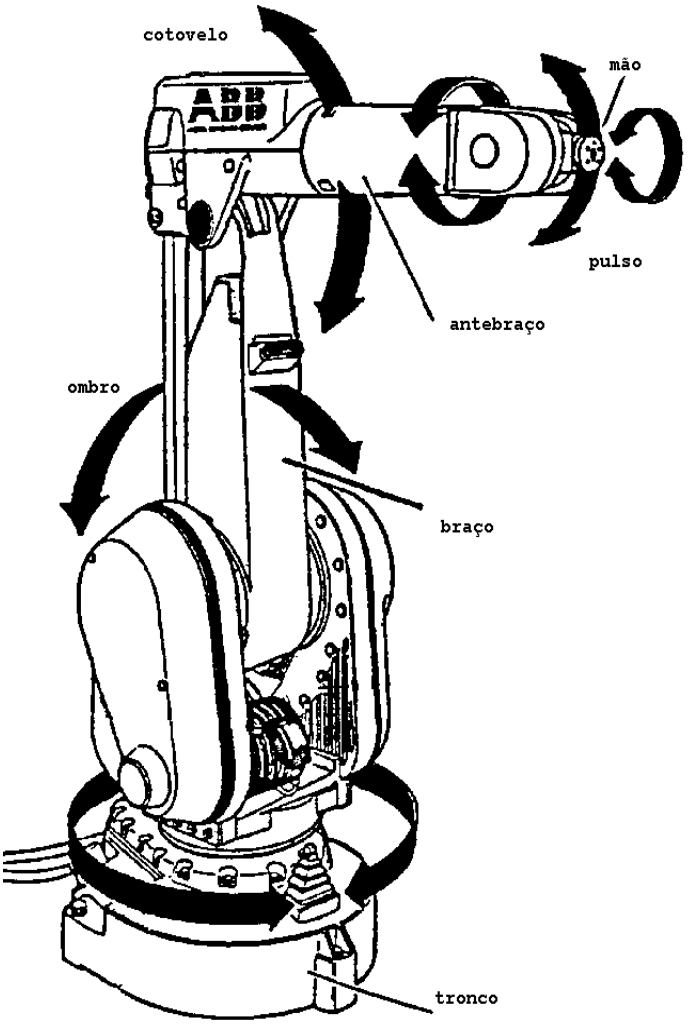



*Figura 2.1. Robô Articulado*

O subsistema de manipulação é determinado essencialmente pela arquitetura do robô e pelo número e tipo dos eixos de movimento. O robô é manipulado no espaço de trabalho tridimensional. Para posicionar a ferramenta em qualquer posição e com qualquer orientação, são necessários seis graus de liberdade. Os primeiros três graus de

liberdade são necessários para posicionar o pulso e os últimos três para orientar a ferramenta no espaço.

O subsistema de sensoreamento externo dos robôs inclui os sensores usados pelo robô para monitorar o seu ambiente de trabalho, como: cameras de TV, visão laser, etc., bem como as interfaces de comunicação entre o robô e outras máquinas ou outros computadores.

O subsistema de interface Homem-máquina permite de forma eficiente a programação, codificação, monitoramento e controle do robô pelo operador. A programação do robô pode ser dividido em três categorias (Schilling, 1990).

- O método "teach-in", "replay" ou "teach-pendant".
- A programação "off-line".
- A programação a nível de tarefas.

O método "teach-in" geralmente é realizado com um "teach-in box" que consiste de um painel-caixa equipado com alavancas e/ou botões com o qual o operador comanda o movimento das juntas. O operador comanda o robô até conseguir a seqüência de movimentos desejada, e depois esta sequencia é armazenada. Quando deseja-se executar o movimento, basta chamar a seqüência e executá-la.

A programação "off-line" do robô utiliza uma linguagem de programação de robôs que possui comandos específicos para controlar o movimento do robô e monitorar ele e/ou seu ambiente. Toda seqüência de movimentos é editada e depois compilada. O programa depois de depurado é executado pelo robô.

A programação do robô com uma linguagem a nível de tarefas difere do método anterior no sentido que o programador não necessita especificar a seqüência de operações que o robô deve executar. A especificação da tarefa independe do tipo do robô. O operador especifica a tarefa a ser executada e o planejador de tarefas ("task-

planner") gera um conjunto de instruções para o robô com base num banco de dados onde estão armazenados a caracterização completa do robô, do ambiente de trabalho, e da tarefa.

O subsistema de (servo)controle de um manipulador pode ser subseqüentemente separado em duas atividades (Fu e Gonzales,1987).

- Controle ou Planejamento de trajetórias.
- Controle de movimento (controle a nível de junta).

O controle de trajetória se refere ao procedimento desenvolvido no sentido de gerar as referências àos controladores de juntas do manipulador, fazendo com que o movimento individual das juntas, executado simultaneamente, desloque adequadamente a ferramenta do robô.

O movimento individual nas juntas é determinado pelo controle de movimento ou controle a nível de junta. Em geral o controle de movimento consiste em: (1) obter o modelo dinâmico do manipulador; e (2) usar este modelo para determinar algoritmos ou estratégias de controle para obter o desempenho dinâmico desejado.

## 2.2 ESTRUTURA DO SISTEMA DE CONTROLE

Ao projetar o sistema de controle de manipuladores robóticos, deve-se levar em conta que os robôs têm uma dinâmica multivariável, não-linear e com acoplamento entre as variáveis. Pode-se considerar o modelo dinâmico completo do Robô função da posição, da velocidade e da aceleração de cada grau de liberdade da estrutura mecânica. Assim sendo, o controle desta estrutura se torna complexo. O cálculo do modelo dinâmico do sistema completo em tempo real nem sempre é possível porque envolve uma grande quantidade de cálculo numérico.

A própria natureza do cálculo do modelo dinâmico sugere que ele deve ser centralizado num computador supervisor. O modelo obtido pelo supervisor é utilizado por controladores nas juntas para implementar os algoritmos de controle de movimento. Uma outra consideração importante é que, por causa da atualização constante do modelo pelo supervisor, deve haver uma comunicação eficiente e permanente entre o supervisor e os controladores de junta.

Neste trabalho pretende-se implementar o controle a nível de junta por microcontroladores (**MCn)** e o controle de trajetória por um microcomputador supervisor (**PC-AT),** tal como representados na figura 2.2.

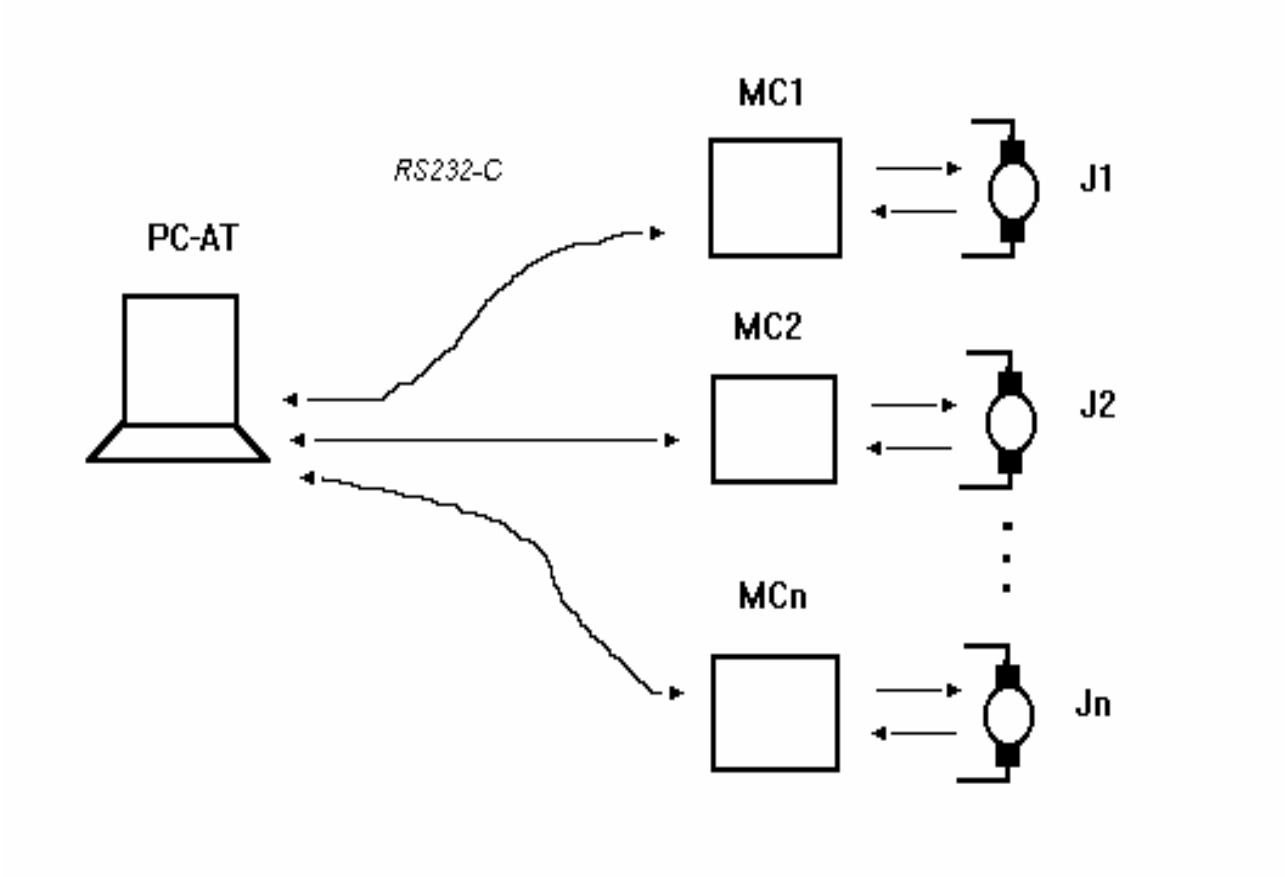



*Figura 2.2. Estrutura do Sistema de Controle de Manipuladores Robóticos*

As funções básicas do supervisor são:

a) interface com o usuário
b) geração de trajetórias
c) coordenação e sincronismo do trabalho dos micro controladores de junta.
d) cálculo do modelo dinâmico do manipulador.

As funções básicas do micro controlador de junta são:
a) implementação do algoritmo de controle e interpolação do movimento a nível de junta
b) acionamento do motor
c) monitoração de posição e velocidade.

Existem vários fatores que influenciam na implementação do sistema de controle com a configuração proposta: complicações tanto a nível do supervisor, quanto a nível do controlador de junta. Há limitações por causa da capacidade de processamento do microprocessador em uso, da quantidade de memória disponível, da linguagem de programação e do tamanho do código executável gerado, e conseqüentemente do tempo de execução dos algoritmos. Outra complicação vem da distribuição das tarefas e funções entre o computador supervisor e o controlador local e do tipo de protocolo de comunicação entre eles.

## 2.3. CONTROLE DE TRAJETÓRIA

Esta seção pretende ampliar a discussão sobre o subsistema de servocontrole do manipulador robótico analisando o controle ou planejamento de trajetórias para efetuar determinada tarefa. O controle de trajetória é nitidamente uma função do computador supervisor. Na literatura não foi encontrada uma definição (ou nomenclatura) única. O termo controle de trajetória também é chamada de controle espacial, planejamento de trajetória ou coordenação do movimento (Alves, 1988).

O controle de trajetória se refere ao procedimento para gerar referências para as juntas a partir de um caminho desejado, levando em consideração fatores relacionados à própria estrutura do manipulador e fatores relacionados ao ambiente de trabalho do manipulador como por exemplo obstáculos no espaço de trabalho.

Um método sistemático para abordar o controle de trajetória é considera-lo como uma caixa preta onde entram o caminho desejado, geralmente em coordenadas cartesianas, as limitações do caminho e as limitações dinâmicas do manipulador (Fu e Gonzales, 1987). A saída dessa caixa preta é uma tabela com posição, velocidade e aceleração desejados, geralmente em coordenadas de juntas. (figura 2.3)

Existem basicamente dois modos de controle de trajetória ou controle espacial: Controle espacial implícito e explícito (Alves, 1988). No primeiro modo, há necessidade de fazer o planejamento da trajetória explicitamente, enquanto no segundo modo, a equação do segmento de curva é embutida no algoritmo de geração de trajetória, tornando-o autônomo.

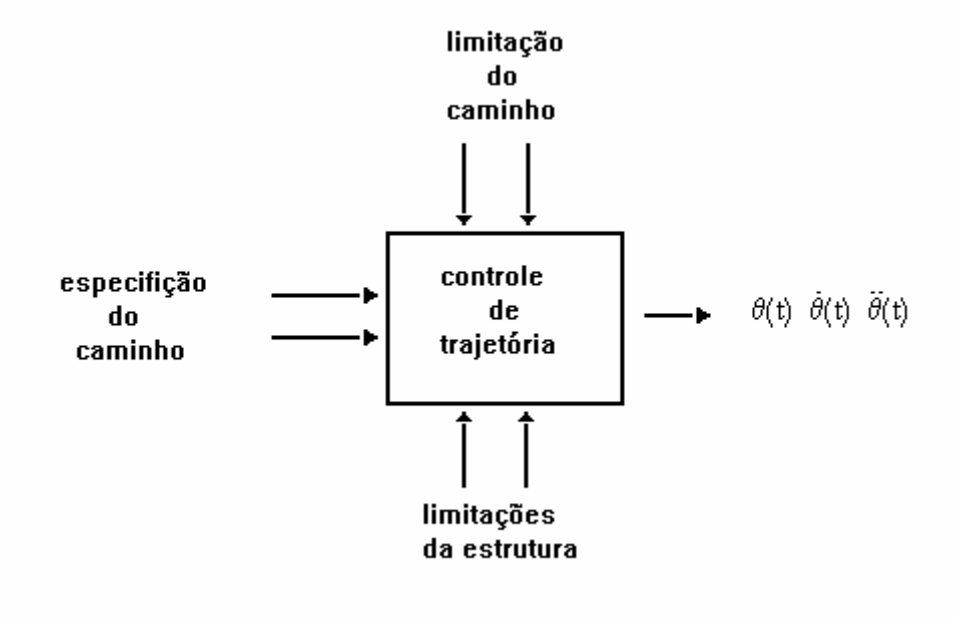



*Figura 2.3. Geração de Trajetórias*

No planejamento explícito da trajetória, é efetuado um cálculo a priori do conjunto de pontos por onde o manipulador deve passar para executar determinada tarefa. Este cálculo geralmente é efetuado com interpolações numéricas. O caminho desejado é aproximado (interpolado) por um conjunto de funções polinomiais e gera-se uma seqüência de pontos de referência no tempo. Existem várias maneiras de efetuar a interpolação. A maneira mais simples é a interpolação linear. Existe a interpolação quadrática onde o caminho é aproximado por funções polinomiais de segunda ordem. Outros tipos de interpolação são a interpolação mista (3-4-3) e a interpolação cúbica ("spline") (Fu e Gonzales, 1987).

A interpolação numérica pode ser realizada tanto no computador supervisor quanto no controlador de junta. A vantagem de efetuar a interpolação pelo supervisor é que a capacidade computacional não será uma limitação na escolha do algoritmo de interpolação. Quando a interpolação é efetuada pelos micro controladores de junta, deve-se levar em consideração a capacidade computacional dos mesmos. Infelizmente, os microcontroladores ainda não tem suporte suficiente para efetuar volumosos cálculos numéricos. A

vantagem da segunda opção é que todo o cálculo é efetuado na placa controladora de junta. O supervisor, neste caso, somente forneceria parâmetros de um determinado movimento a ser cumprido por cada junta.

### 2.3.1 Interpolação quadrática

A interpolação quadrática é um dos métodos mais simples para gerar trajetórias. O algoritmo proposto (algoritmo 1) a seguir gera a trajetória para o caso onde as velocidades final e inicial são nulas.

Algoritmo:

1 Divisão da trajetória em 3 segmentos, aceleração, velocidade constante, freagem.
2 Calcula os parâmetros correspondentes, velocidade, aceleração.
3 Gera a base de tempo
4 Calcula a posição para cada instante de tempo.

```
Algoritmo 1:  Geração de trajetória
%
% Programa para gerar trajetória
%
tmp=256*input('duração em blocos de 256 * T ');
xin=input('ponto inicial >> ');
xfn=input('ponto final   >> ');
%
% cálculo parâmetros
%
deltx=xfn-xin;                % delta posicao
vll=deltx/tmp;                % velocidade linear
vlm=1.5*vll;                  % velocidade aumentada
delt=deltx/2-(vlm*0.25*tmp);  % termino parábola
alfa=delt/((0.25*tmp-1)^2);   % coeficiente da parábola
%
% escala de tempo
%
t1=0:tmp/4-1;
t2=0:tmp/2;
t4=tmp/4-1:-1:0;
%
% gerando trajetória
%
y1=alfa*(t1 .^ 2)+xin;
y23=(vlm*t2)+y1(tmp/4);
```

```
y4=-alfa*(t4 .^ 2)+y23(tmp/2+1)+delt;
t=(0:tmp-2);
y=[y1(1:(tmp/4-1)) y23  y4(2:(tmp/4))];
plot(t,y);
```

A figura 2.4 mostra a trajetória gerada partindo da posição 0 até a posiçao 20 com 256 pontos.

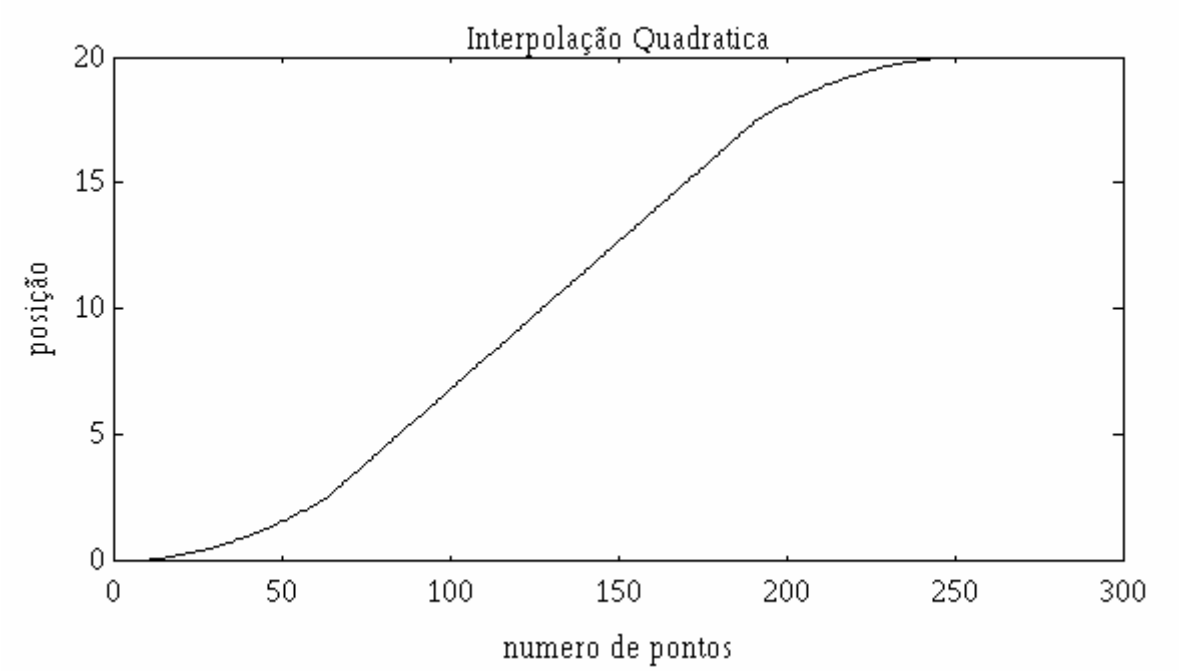



*Figura 2.4. Interpolação quadrática*

Com poucas alterações, pode-se elaborar um algoritmo para condições iniciais gerais, i.e. velocidades iniciais e finais não nulas. A vantagem desta abordagem é que a trajetória pode ser gerada antes de se iniciar o movimento ou durante o movimento.

No algoritmo proposto o problema é o tamanho de memória disponível para armazenar todos os pontos. Aqui o parâmetro crítico é o tempo necessário para gerar cada ponto depois de cada amostragem. Uma solução poderia ser usar uma combinação das duas abordagens, usando uma memória auxiliar de tipo circular (ring buffer).

## 2.4 CONTROLE DE MOVIMENTO

O movimento individual nas juntas é comandado pelo controle de movimento ou controle a nível de junta. A primeira etapa do controle de movimento é determinar o modelo dinâmico do manipulador. Com base neste modelo pode-se determinar a estratégia e o algoritmo de controle a serem usados. Nesta seção será abordada a determinação do modelo dinâmico do manipulador. No capítulo 5, serão tratados o desenvolvimento da estratégia de controle e sua implementação. O cálculo do modelo dinâmico da estrutura robótica é uma função do computador supervisor. Apresentaremos então um método sistemático para calcular o modelo dinâmico de um manipulador robótico.

### 2.4.1 Modelo Dinâmico de um manipulador

O modelo dinâmico de um robô pode ser obtido usando as abordagens de Lagrange-Euler (L-E) ou de Newton-Euler (N-E). Ambas as abordagens fornecem um procedimento sistemático para chegar ao modelo dinâmico do manipulador (Fu e Gonzales, 1987). O cálculo do modelo dinâmico usando L-E faz uso de uma matriz (4x4) de transformação homogênea para representar a cinemática do manipulador. As equações de movimento são um conjunto de equações diferenciais não-lineares de segunda ordem na forma de equação de estado.

O cálculo do modelo dinâmico usando N-E envolve um conjunto de equações recorrentes diretas e equações recorrentes inversas. As equações diretas de N-E (Craig, 1989) para juntas rotacionais, descritas na tabela 1, propagam informação cinemática como velocidade linear e angular, aceleração linear e angular do centro de massa de cada junta, da base da estrutura até a ponta do manipulador. Assim tendo:

$v$ – velocidade linear
$\theta,\omega$ – posição e velocidade angulares
$R$ – matriz de rotação
$P$ – ponto de rotação em relação à base
$Pc$ – distância do centro de massa em relação à base
${}^{c}I$ – matriz tensor de inércia
F,f – força exercida
N,n – momento de inércia
$\tau$ – torque
Z – base de coordenados
$i$ – grau de liberdade

*Tabela 4. Equações Diretas N-E*

$${}^{i+1}w_{i+1} = {}^{i+1}_{i}R \cdot {}^{i}w_i + \dot{\theta}_{i+1} \cdot {}^{i+1}\hat{Z}_{i+1}$$
$${}^{i+1}\dot{w}_{i+1} = {}^{i+1}_{i}R \cdot {}^{i}w_i + {}^{i+1}_{i}R \cdot {}^{i}w_i \times \dot{\theta}_{i+1} \cdot {}^{i+1}\hat{Z}_{i+1} + \ddot{\theta}_{i+1} \cdot {}^{i+1}\hat{Z}_{i+1}$$
$${}^{i+1}\dot{v}_{i+1} = {}^{i+1}_{i}R \cdot ({}^{i}w_i \times {}^{i}P_{i+1} + {}^{i}w_i \times ({}^{i}w_i \times {}^{i}P_{i+1}) + {}^{i}\dot{v}_i)$$
$${}^{i+1}\dot{v}c_{i+1} = {}^{i+1}\dot{w}_{i+1} \times {}^{i+1}Pc_{i+1} + {}^{i+1}w_{i+1} \times ({}^{i+1}w_{i+1} \times {}^{i+1}Pc_{i+1}) + {}^{i+1}\dot{v}_{i+1}$$
$${}^{i+1}F_{i+1} = m_{i+1} \cdot {}^{i+1}\dot{v}c_{i+1}$$
$${}^{i+1}N_{i+1} = {}^{C_{i+1}}I_{i+1} \cdot {}^{i+1}\dot{w}_{i+1} + {}^{i+1}w_{i+1} \times {}^{C_{i+1}}I_{i+1} \cdot {}^{i+1}w_{i+1}$$

As equações recorrentes inversas de N-E para juntas rotacionais, apresentadas na tabela 2, propagam as forças e os momentos exercidos sobre cada junta da ponta do manipulador de volta à base do robô.

*Tabela 5. Equações Inversas N-E*

$${}^{i}f_i = {}^{i}_{i+1}R \cdot {}^{i+1}f_{i+1} + {}^{i}F_i$$
$${}^{i}n_i = {}^{i}N_i + {}^{i}_{i+1}R \cdot {}^{i+1}n_{i+1} + {}^{i}Pc_i \times {}^{i}F_i + {}^{i}P_{i+1} \times {}^{i}_{i+1}R \cdot {}^{i+1}f_{i+1}$$
$$\tau = {}^{i}n_i^{T} \cdot {}^{i}\hat{Z}_i$$

A própria natureza das equações recursivas de N-E sugere que sejam executados por um programa de computador. Por isso, as equações recorrentes foram implementados por

um programa de cálculo simbólico, tendo como parâmetros de entrada a geometria do manipulador, matriz de rotação, matriz tensor de inércia e os momentos e forças atuando na ponta do robô, como mostrado na tabela 3. O programa gera as equações de velocidade, aceleração e os momentos de inércia e torques para cada junta. (apêndice 1)

*Tabela6. tabela de entrada programa cálculo equações N-E*

| | |
|---|---|
| $R$ | – matriz de rotação |
| $P$ | – ponto de rotação em relação à base |
| $P_C$ | – distância do centro de massa em relação à base |
| ${}^{C}I$ | – matriz tensor de inércia |
| $wv$ | – velocidades da base |
| $\dot{w}\dot{v}$ | – aceleração da base |
| $f$ | – forças atuando na ponta |
| $n$ | – momento atuando na ponta |

A solução completa fechada do torque obtido pelo procedimento recorrente pode ser reescrita na sua forma matricial chamada de equação de estado

$$\tau = M(\theta)\ddot{\theta} + V(\theta,\dot{\theta}) + G(\theta) \qquad 2.1$$

Onde:

$M(\theta)$ matriz nxn com os termos inerciais do manipulador

$V(\theta,\dot{\theta})$ vetor nx1 com os termos de forças centrífugas e de Coriolis

$G(\theta)$ vetor nx1 com os termos de forças gravitacionais

$\tau$ vetor das torques

Cada elemento de $M(\theta)$ e $G(\theta)$ é uma função que depende de $\theta$, o vetor de variáveis de junta do manipulador. Cada elemento de $V(\theta,\dot{\theta})$ é uma função da velocidade e posição de todas as juntas do manipulador.

O esforço computacional para calcular a solução fechada do torque usando o método de N-E requer 126n-99

multiplicações e 106n–92 adições (Craig, 1989), onde n representa o número de graus de liberdade da estrutura. Quando o cálculo é realizado simbolicamente, também tem que ser levado em consideração o tamanho da memória alocada para acumular os termos. O programa de cálculo simbólico usado neste trabalho foi desenvolvido com uma estrutura fixa de dados, e por isto tem limitações quanto ao tamanho dos termos e conseqüentemente à complexidade do manipulador; contudo, para os objetivos deste trabalho, funcionou satisfatoriamente.

# CAPÍTULO 3. SISTEMA DE DESENVOLVIMENTO PARA MICROCONTROLADORES BASEADO NO 8031, 8051 E 8751 : KIT-31

Por razões de disponibilidade e suporte, utilizar-se-á o microcontrolador 8031 da INTEL. Em torno dele foi projetado e implementado um sistema de desenvolvimento, chamado KIT-31. Este constitui uma ferramenta de baixo custo para projetar sistemas baseados nos controladores de oito bits 8031, 8051 e 8751 do fabricante INTEL. Suas características tornam-no um sistema adequado para instituições de ensino e pesquisa, bem como para empresas que necessitem de uma ferramenta de desenvolvimento simples e completa. O KIT-31 requer, na sua configuração mínima, um microcomputador IBM-PC-XT com pelo menos 256 Kbytes de RAM, uma interface serial RS232-C, um drive de 5 1/4 " - 360 Kbytes e o sistema operacional MSDOS 2.0 ou versões posteriores.

## 3.1 - ARQUITETURA DO SISTEMA

Existem vários tipos de sistemas de desenvolvimento para microcontroladores ou microprocessadores, todos eles fornecendo facilidades para desenvolver programas aplicativos. Os mais elaborados (chamados in-circuit emulator) são ferramentas que emulam o microprocessador na placa de circuito impresso do sistema que está sendo desenvolvido.

O KIT-31 não chega a tal elaboração, ele é composto pelo *hardware* e *software* mínimos necessários em torno dos quais uma aplicação pode ser desenvolvida. Todo programa é editado dentro do ambiente do IBM-PC usando as facilidades de edição e armazenamento de arquivos. Um "cross-compiler" ou "cross-assembler" gera o código em linguagem de máquina do processador final. Depois do programa pronto, o código em linguagem de máquina é transferido ("download") para a placa que contém o microcontrolador e armazenado numa memória volátil aqui chamada de memória para programa usuário. A partir do IBM-PC é possível mandar executar o programa aplicativo.

O KIT-31, mostrado na figura 3.1, é composto de duas placas de circuito impresso e um programa denominado SD31. A placa principal contém o microcontrolador e a memória do sistema. A placa secundária, também chamada de interface de aquisição e distribuição, contém os circuitos que interligam o controlador aos processos externos.

O controle do KIT-31 é efetuado com o programa SD31 que roda no IBM-PC. Este programa contém um cross-assembler, suporte para depurar o código gerado, e suporte para fazer o transferência do programa gerado para a placa principal do KIT-31. As seções a seguir detalham o circuito das placas principal e secundária, e o programa SD31

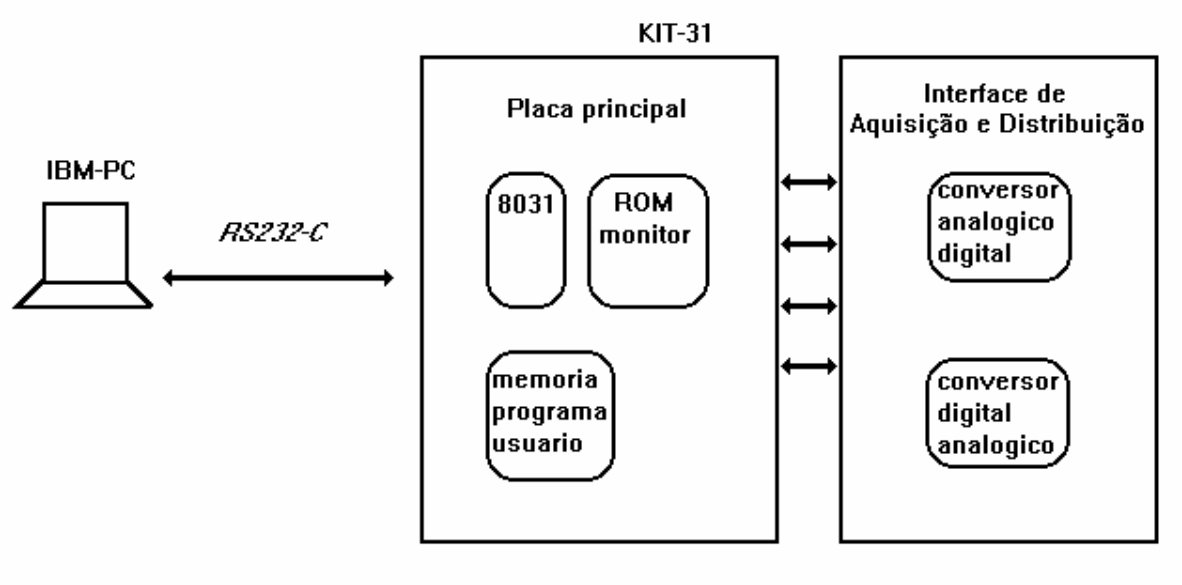



*Figura 3.1. KIT-31*

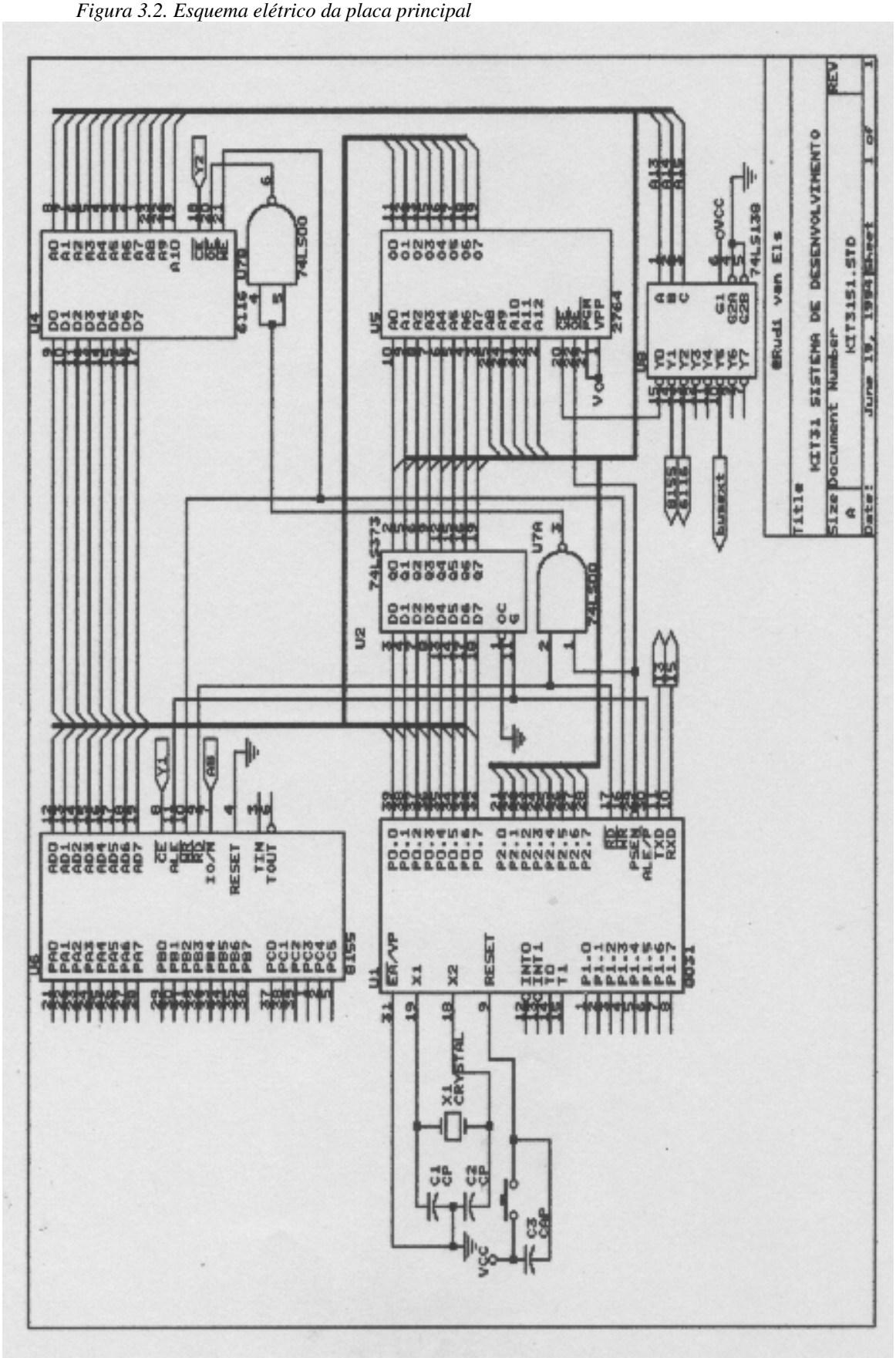
## 3.2 PLACA PRINCIPAL

A Placa principal do KIT-31 contém as seguintes unidades e facilidades:

- micro controlador 8031
- memória EPROM de 8 Kbytes com o programa monitor.
- memória para programa usuário de 8 Kbytes
- memória RAM de 256 bytes
- interface serial RS232-C
- temporizador e três portas paralelas tipo Intel PPI-8155
- acesso ao barramento de dados e endereços através de conectores.
- *software* do programa monitor aberto e documentado

O esquema elétrico da placa principal é mostrado na figura 3.2.

*Figura 3.2. Esquema elétrico da placa principal*



A mapa de memória na figura 3.3 mostra a organização das mesmas e os circuitos de entrada e saída. Convém lembrar que os circuitos de entrada e saída estão mapeados em memória já que o MC8031 não tem um barramento específico de entrada e saída.

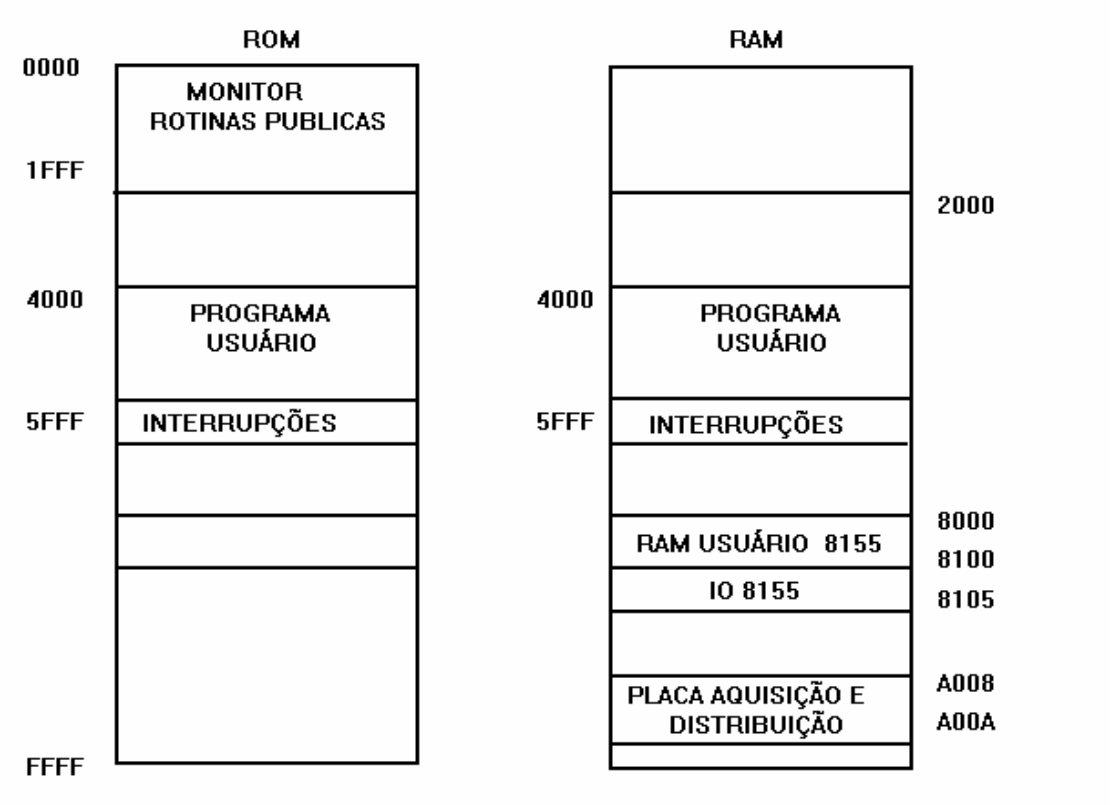



*Figura 3.3. Mapa memória do KIT-31*

O acesso à porta de entrada/saída do MC8031 e às portas da PPI-8155 do KIT-31 é realizado através de conectores na placa principal. O usuário também tem acesso ao barramento de endereços, de dados e de controle através de conectores separados.

## 3.2.1. Programa monitor

O Programa monitor do Sistema de Desenvolvimento KIT-31 se encontra gravado em EPROM e gerencia a operação da placa

principal e a comunicação com o microcomputador IBM-PC. As rotinas do programa monitor (tabela 4) estão disponíveis e documentados para que o usuário possa utilizá-las no seu próprio programa.

*Tabela 4. Rotinas públicas monitor*

| | |
|---|---|
| iniser() | decisegundos() |
| getchar() | segundos() |
| putchar() | outword() |
| outhex() | inword() |
| inhex() | autotest() |
| write() | |

### 3.2.2 Interrupções do sistema

As interrupções do micro controlador 8031 foram desviados para a memória para programa usuário de tal modo que o usuário possa, através de uma instrução de escrita, programar o endereço da subrotina de tratamento de interrupções sem ficar restrito aos endereços físicos do micro controlador. As interrupções externas dos contadores e porta serial que se encontram no início do programa monitor apontam para uma área de memória disponível ao usuário, e para cada vetor é reservado um espaço de 3 bytes, facilitando assim a colocação de uma instrução de desvio incondicional (LJMP ENDEREÇO). A tabela 5 mostra os interrupções do MC8031 e os seu respectivos endereços na memória para programa usuário.

*Tabela 5 Interrupções desviadas do MC8031.*

| NOME | INTERRUPÇÃO | ENDEREÇO |
|---|---|---|
| INT0 | Interrupção externo 0 | 5FF0 |
| TIM0 | Interrupção do temporizador 0 | 5FF3 |
| INT1 | Interrupção externo 1 | 5FF6 |
| TIM1 | Interrupção do temporizador 1 | 5FF9 |
| SERI | Interrupção de comunicação serial | 5FFC |

## 3.3 PLACA DE AQUISIÇÃO E DISTRIBUIÇÃO DE DADOS

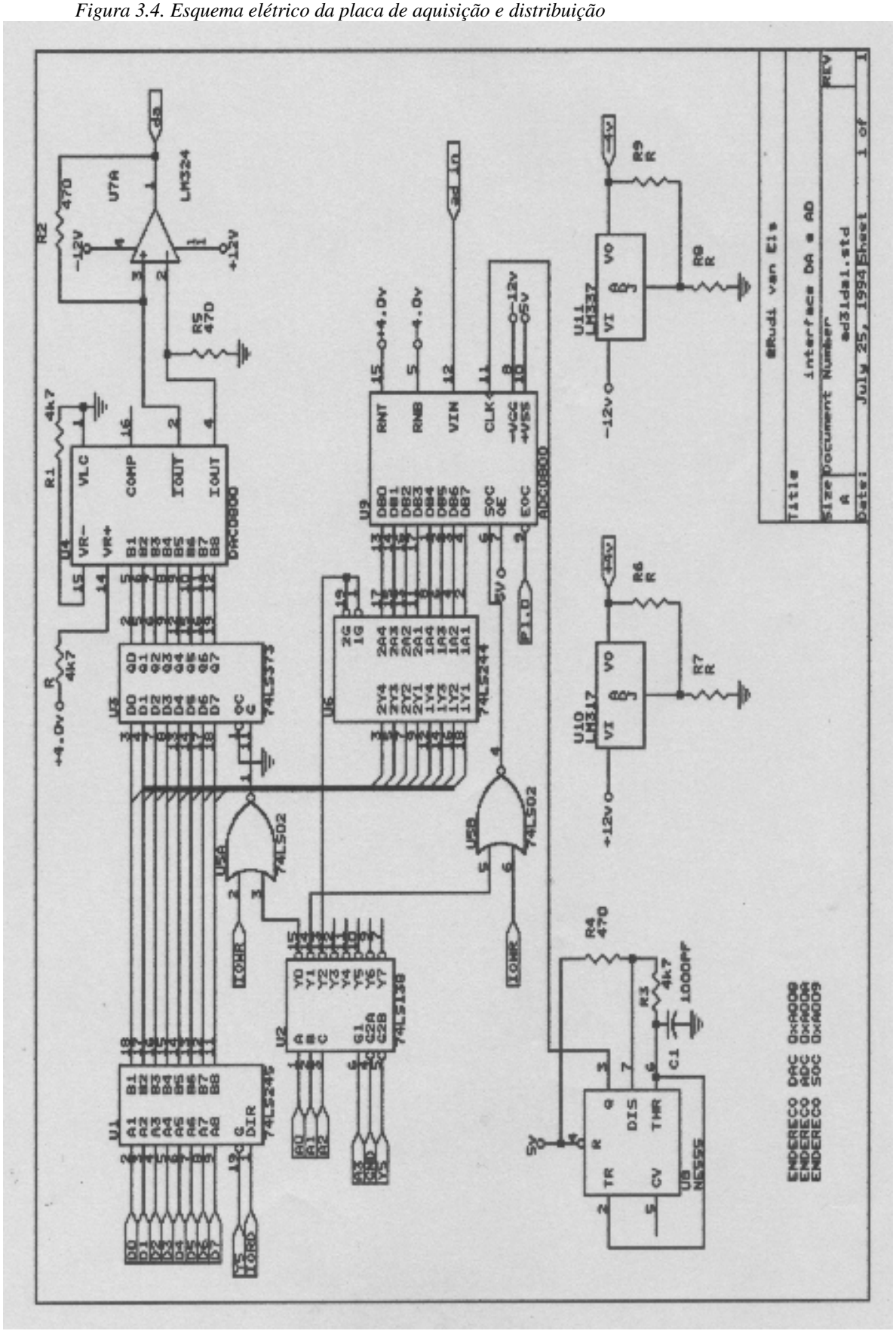
A Placa de Aquisição e Distribuição de Dados mostrada na figura 3.4, consiste de:

- Um conversor analógico/digital de 8 bits
- Um conversor digital/analógico de 8 bits

O conversor analógico/digital é composto de: 1) um circuito integrado ADC0800 que converte o sinal analógico de entrada na faixa de −4 a +4 volts em sinal digital e; 2) um circuito astável para gerar o relógio para a conversão. O erro de fim de escala é de ±1 LSB, e o tempo de conversão é de ±1ms.

O conversor digital/analógico é composto de: 1) um circuito integrado DAC0800 que converte o sinal digital em sinal de corrente proporcional, e; 2) um conversor corrente/tensão baseado no amplificador operacional LM741. O sinal analógico de saída é +0.4 volts para o byte 0FFH, e −0.4 volts para o byte 00H. O tempo de conversao é de aproximadamente 300ns.

*Figura 3.4. Esquema elétrico da placa de aquisição e distribuição*



## 3.4 PROGRAMA SD31

O Programa SD31 controla a placa principal a partir do microcomputador IBM-PC. O programa fornece um ambiente interativo para o usuário com menus para facilitar sua operação. O programa SD31 é composto por dois arquivos "SD31.EXE, XCDATA.COD" que gerenciam todo o sistema, e pode ser dividido em cinco módulos:

- Montador assembler
- Filtro para entrada arquivo de vários formatos
- Editor Hexadecimal
- "Downloader" para comunicação com o KIT-31
- Interface de comunicação serial

O programa fonte do usuário pode ser escrito com qualquer editor de texto ASCII com o mnemônico do micro controlador 8031, usando diretivas básicas do assembler e endereçamento simbólico ou absoluto. Este programa é montado pelo módulo **Montador Assembler** e gravado num arquivo com formato próprio. O montador gera uma listagem na tela e em arquivo no formato ASCII com indicação de erro e mapa de endereços, para facilitar a depuração.

O **Filtro** de arquivo aceita programas montados por outros montadores ou compiladores que geram código executável nos formatos INTEL-STANDARD e ASCII-HEX e pode gerar saídas no formato proprietário ou ASCII-HEX.

O **Editor** Hexadecimal é uma ferramenta rápida para depurar código executável. A comunicação com a placa principal KIT-31, a inicialização da placa e a transferência do código executável para a placa é realizado pelo módulo **Downloader**. Depois de ter carregado o código na memória do KIT-31 o Downloader transfere o comando para o MC 8031. O interface de **Comunicação** possibilita ao usuário verificar o andamento do programa em teste e se comunicar com o KIT-31 usando comunicação serial em formato ASCII.

## 3.5 SOFTWARE DE DESENVOLVIMENTO

Um dos princípios que orientaram este trabalho foi o de implementar os algoritmos de controle e programas de suporte numa linguagem de alto nível, para facilitar a manutenção e a portabilidade das rotinas. Assim, escolheu-se a linguagem C, o compilador usado foi o "Archimedes C-51 Cross-Compiler Kit for 8051 microcontroller Development". Entretanto, foi necessário adaptar o compilador C-51 às limitações do KIT-31.

As rotinas de entrada e saída que acompanham o compilador C-51 possuem um código fonte relativamente grande e assim não puderam ser usados no KIT-31, já que ele somente dispõe de 8 Kbytes de memória de programa. Por isto, foi desenvolvida uma biblioteca de rotinas para entrada e saída de dados, para conversão de dados de hexadecimal para decimal e para conversão de dados em ASCII para hexadecimal, em linguagem assembler do MC8031.

Na construção dos programas do microcontrolador, procurou-se efetuar os algoritmos de controle sempre por interrupções, e manter assim o programa principal disponível para a tarefa de supervisão. O tratamento das interrupções também foi todo ele escrito em linguagem C, sempre tentando otimizar o código para reduzir o tempo de execução e, somente em último caso, fazendo uso de rotinas específicas em Assembler.

O C-51 também fornece suporte para operações de ponto flutuante (float) usando o formato de 32 bits do IEEE de precisão simples, e tem uma biblioteca de funções matemáticas trigonométricas básicas. Entretanto, o cálculo do algoritmo de controle foi efetuado usando aritmética de ponto fixo usando variáveis do tipo inteiro com sinal formado por 16 bits, porque a demanda computacional para utilizar aritmética de ponto flutuante é muito grande. A multiplicação de dois números do tipo float com o KIT-31

usando um cristal de 6.144Mhz demora aproximadamente 8 ms. Utilizou-se então aritmética de ponto fixo e tabelas de referência ("look-up tables").

# CAPÍTULO 4. ANÁLISE DINÂMICA DE ACIONAMENTO ELETROMECÂNICO PARA JUNTAS ROTACIONAIS DE MANIPULADORES ROBÓTICOS.

Nos robôs com acionamento eletromecânico, a movimentação das juntas pode ser efetuada por motores de corrente contínua (CC), motores de passo, ou motores de corrente alternada. Neste capítulo, será analisada a dinâmica de um motor CC visando obter um modelo matemático, e então estudar-se-á a sua aplicação como acionador de juntas robóticas.

## 4.1 ACIONAMENTO POR MOTOR DE CORRENTE CONTÍNUA

O controle do motor CC pode ser efetuado por manipulação das variáveis elétricas de campo ou de armadura. E as variáveis controladas são geralmente a posição do rotor ou o torque de acionamento. Sabe-se que o torque desenvolvido pelo motor é função do fluxo e da corrente na armadura. A tensão contra-eletromotriz é função do fluxo e da velocidade angular.

A manipulação da corrente de armadura é mais apropriada em aplicações onde deseja-se controlar o torque. E a manipulação da tensão, é conveniente quando a variável controlada é a velocidade ou a posição angular. No segundo caso, o torque de carga é tratado como uma perturbação externa atuando no eixo do motor( Koren, 1985).

Neste trabalho foi utilizado um servomotor, parte de um servomecanismo educacional da Feedback Ltd. (Feedback DC Modular Servo) MS150 (Feedback, 1980), mostrado na figura 4.1. Este sistema é utilizado no Laboratório de Controle por Computador. O conjunto consiste de um servoamplificador, um motor, redução por engrenagens, um tacômetro acoplado no eixo principal do motor e um potenciômetro indicador de posição conectado ao eixo de baixa velocidade.

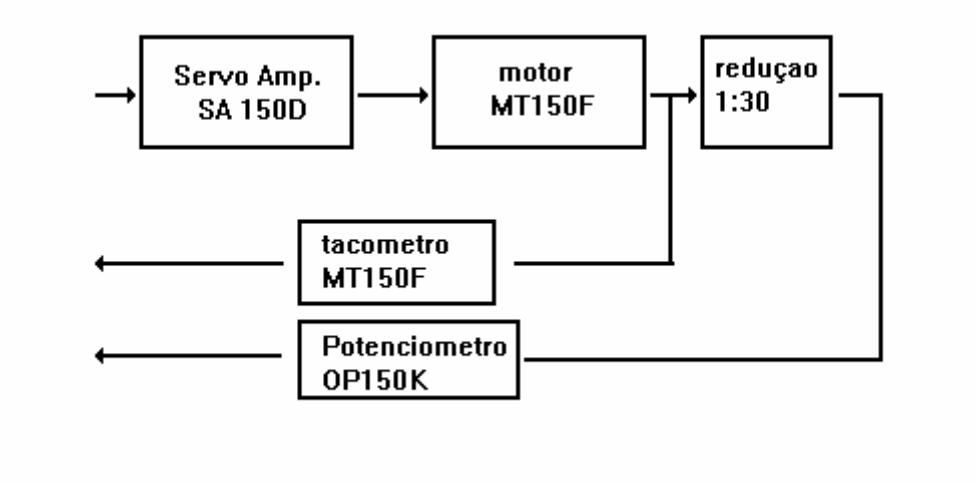



*Figura 4.1. Servomotor Feedback MS150*

## 4.2 MODELO MATEMÁTICO E LINEARIZAÇÃO

O modelo matemático de um sistema pode ser obtido por dois métodos. O primeiro método usa as leis da física para determinar o modelo do sistema. O segundo método usa técnicas de identificação de processos.

O primeiro método tem como objetivo deduzir a partir das leis da física as equações que regem o sistema. Depois, os parametros são calculados ou medidos.

Ao analisar experimentalmente o conjunto servoamplificador e motor considerado, constatou-se que a velocidade desenvolvida pelo motor, quando excitado pela tensão aplicada a partir do servoamplificador, apresenta uma característica não-linear, tipo faixa morta, relativamente grande. O torque desenvolvido pela armadura do motor tem de vencer os atritos estático e seco do motor para colocar o rotor em movimento. A figura 4.2 mostra a faixa morta obtida em testes de laboratório de caracterização automática de dispositivos (Soares Jr, 1990).

O servomecanismo educacional da Feedback inclui uma unidade de compensação linear que compensa a faixa morta. Através de realimentação de velocidade e controle do motor pelo campo. Com o uso desta unidade o conjunto do motor e unidade de compensação apresentam características lineares. (Casanova, 1980)

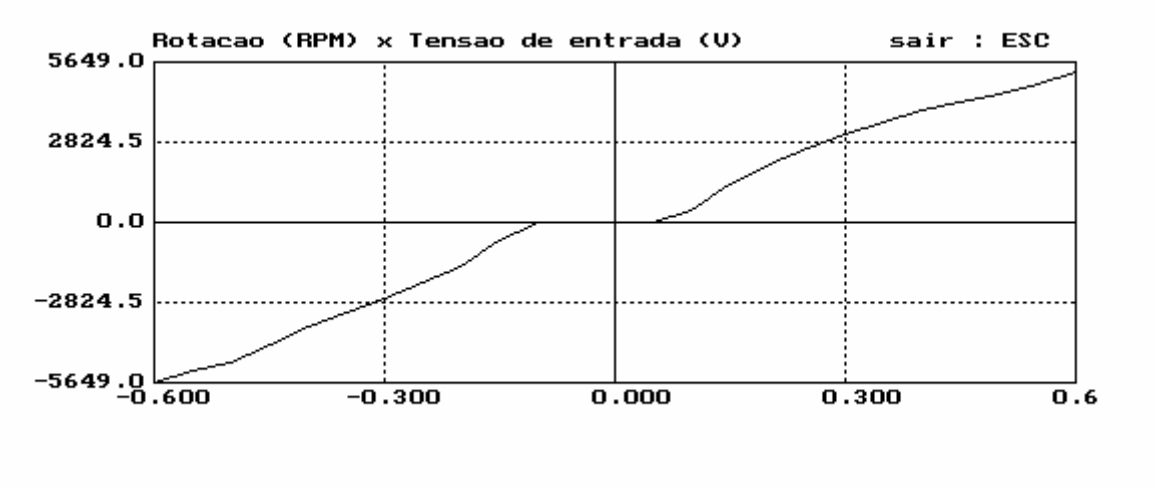



*Figura 4.2. Característica Estática Velocidade-Tensão (Soares Jr, 1990).*

Neste trabalho optou-se por uma técnica que compensa a faixa morta (Fu-Juay Chang, 1990). Ela consiste no cancelamento da faixa morta com uma unidade não-linear aplicada na entrada do servoamplificador conforme mostrado na figura 4.3.

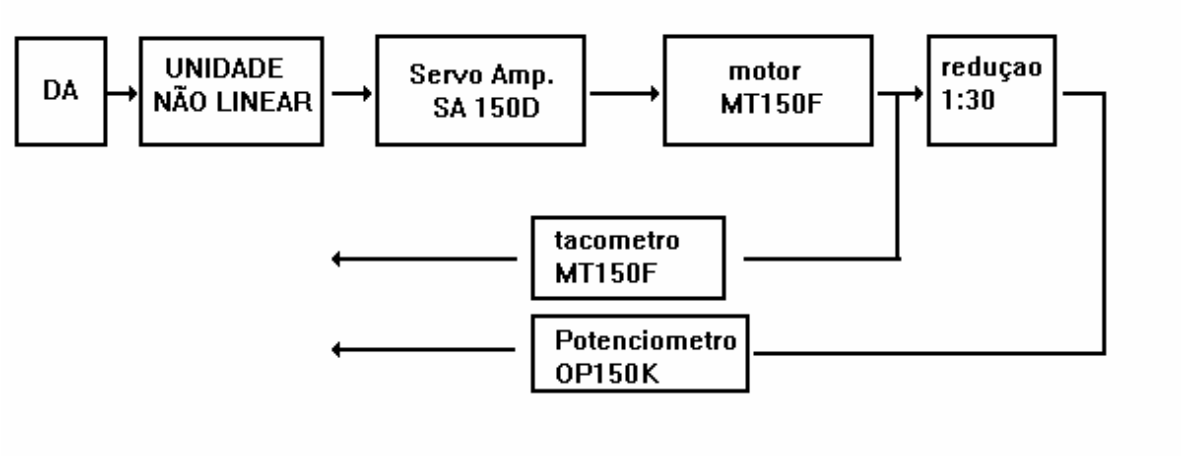



*Figura 4.3. Compensador não-linear para compensar a faixa morta*

O conversor digital-analógico (DA) da interface de aquisição e distribuição do KIT-31 é conectado à unidade não-linear que tem por função compensar a saída do DA e acionar o motor a partir do seu servoamplificador.

A figura 4.4 mostra a característica não-linear utilizada para cancelar a faixa morta. À tensão de saída do conversor DA (Vdac) é somada uma tensão vd+ ou vd- conforme a polaridade de Vdac. Os valores de vd+ e vd- são as tensões mínimas para colocar o motor em movimento, e são obtidos experimentalmente.

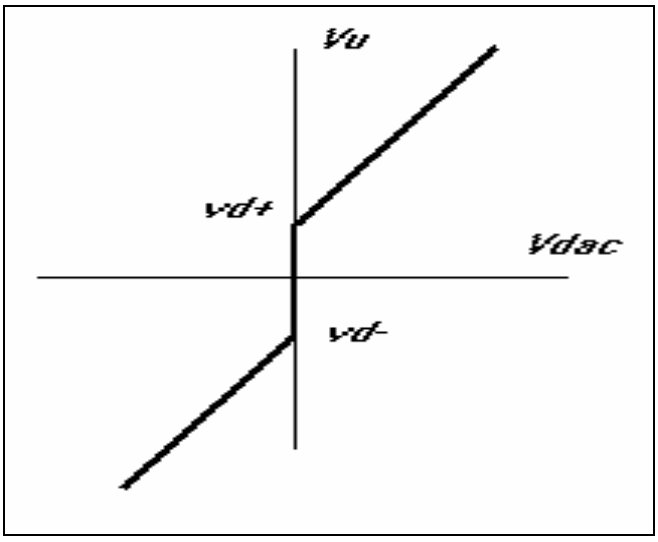



*Figura 4.4. Característica não-linear do compensador*

A figura 4.5 mostra o circuito de compensação. A polaridade do sinal Vdac é fornecido pelo próprio micro controlador através dos pinos VPU_sel e VPU_0.

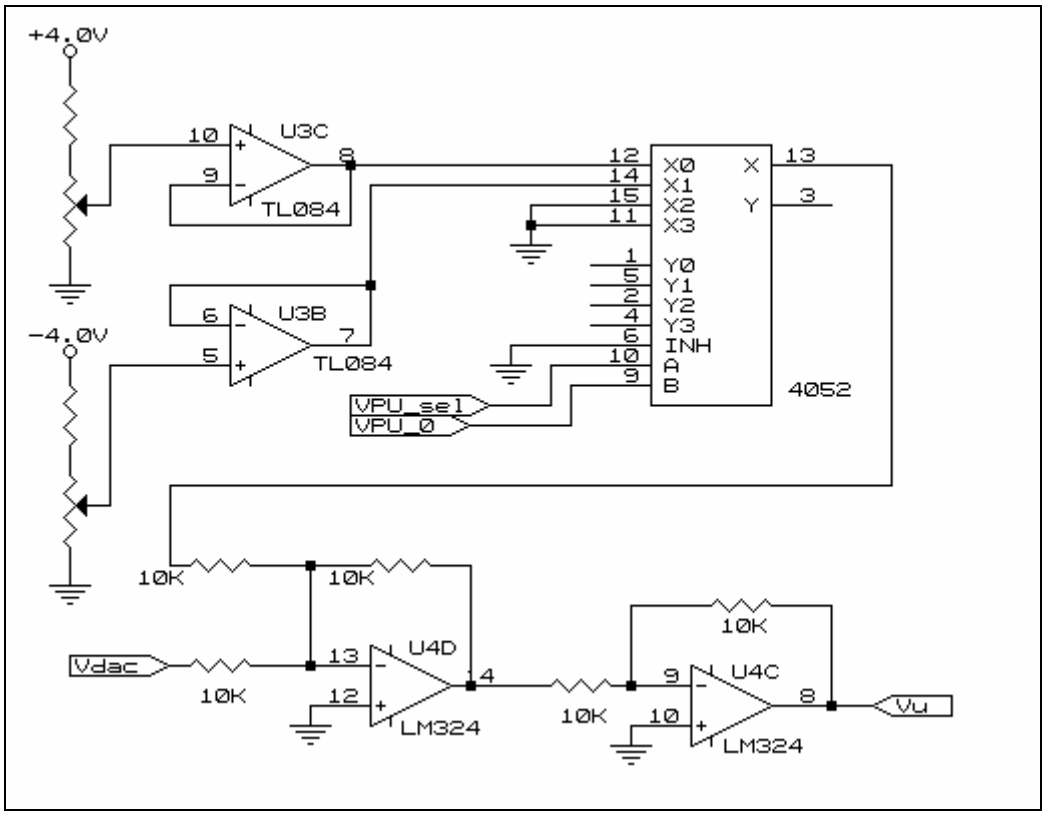



*Figura 4.5. Cicuito de compensação não-linear*

A figura 4.6 mostra a velocidade do conjunto motor, servo e unidade não-linear a partir do conversor DA. A linha tracejada mostra o comportamento sem a unidade não-linear enquanto a linha cheia mostra o efeito de compensação.

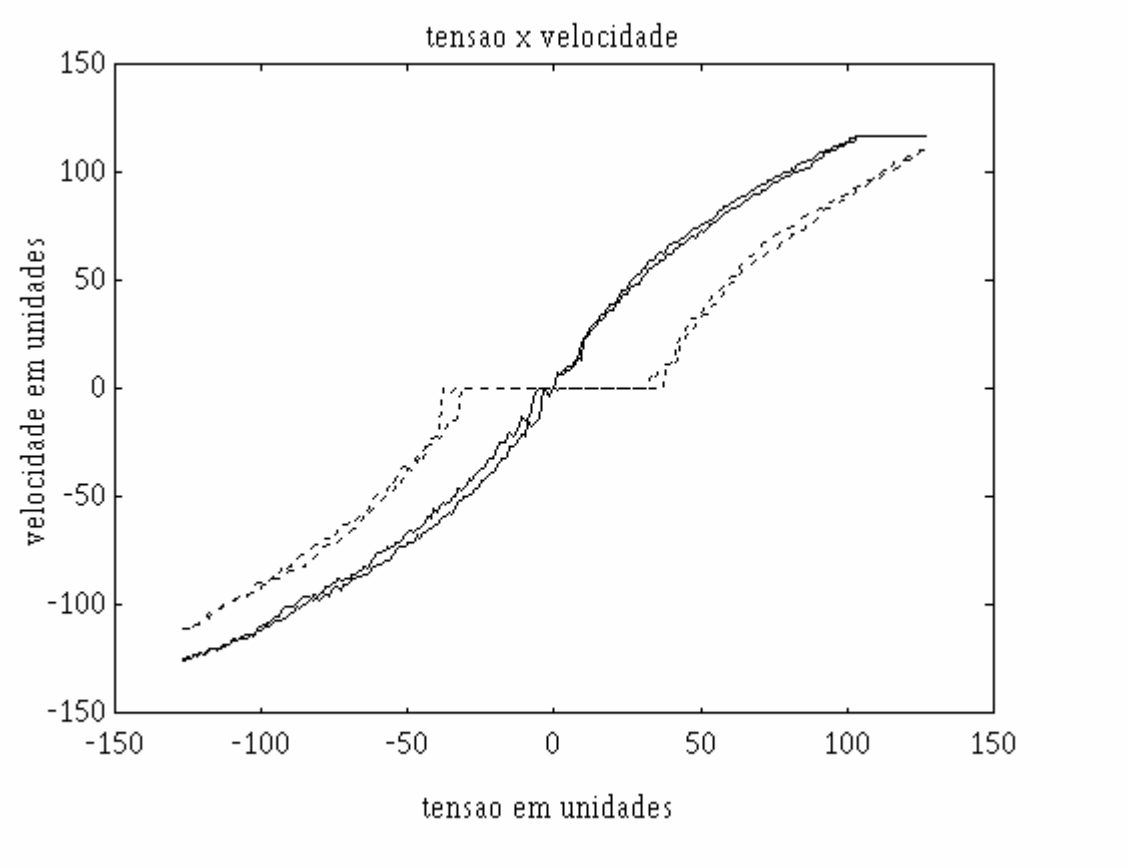



*Figura 4.6. Relação Velocidade x Tensão: Não Compensada e Compensada.*

Havendo-se compensado a faixa morta, proceder-se-á caracterização matemática do conjunto.

### 4.2.1 Modelamento

Um motor de corrente contínua (CC) controlado pela armadura, tendo como variável controlada a posição do rotor, é representado pela figura 4.7.

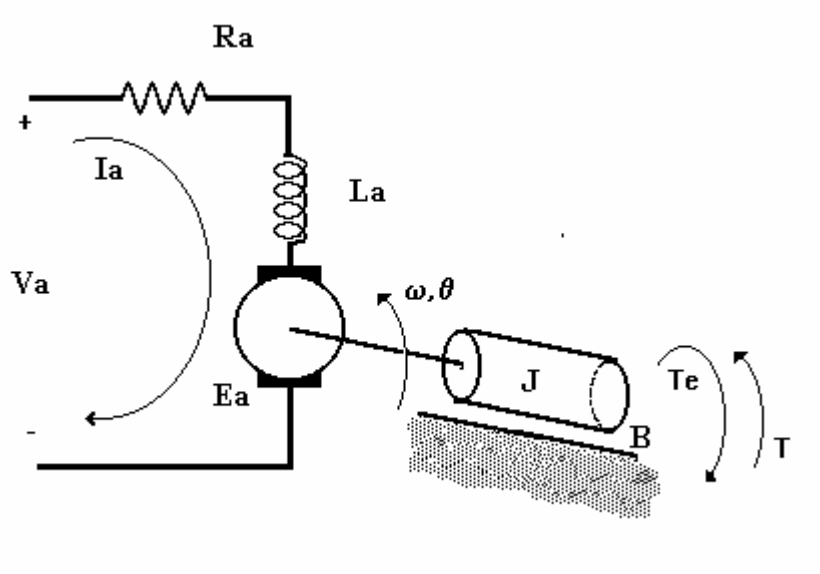



*figura 4.7. Motor de Corrente Continua controlado pela Armadura*

onde:

J – momento de inércia do rotor e carga
B – atrito viscoso do rotor e carga
T – torque fornecido ao rotor
Te – torque externo
Ia – corrente da armadura
Ra – resistência da armadura
La – Indutância da armadura
Va – tensão aplicada à armadura
Ea – tensão contra eletromotriz
ω – velocidade angular
θ – posição angular

No domínio-s as equações que regem o sistema são:

Balanço mecânico

$$Js^2\theta(s) + Bs\theta(s) = T(s) + T_e(s)$$
$$T(s) = K_e \cdot \Phi \cdot I_a(s) \qquad 4.1$$

Balanço elétrico

$$sL_aI_a(s) + R_aI_a(s) + E_a(s) = V_a(s)$$
$$E_a(s) = K_e \cdot \Phi \cdot \omega(s) \qquad 4.2$$

No controle pela armadura de posição ou velocidade de servomotor CC, o torque externo entra no sistema como uma variável de perturbação. Em geral, a constante de tempo elétrica ($\tau_e = L_a / R_a$) é bem menor que a constante de tempo mecânica ($\tau_m = J / B$); assim, pode-se simplificar o modelo. A representação das equações do sistema em forma de diagrama de blocos é mostrada na figura 4.8.

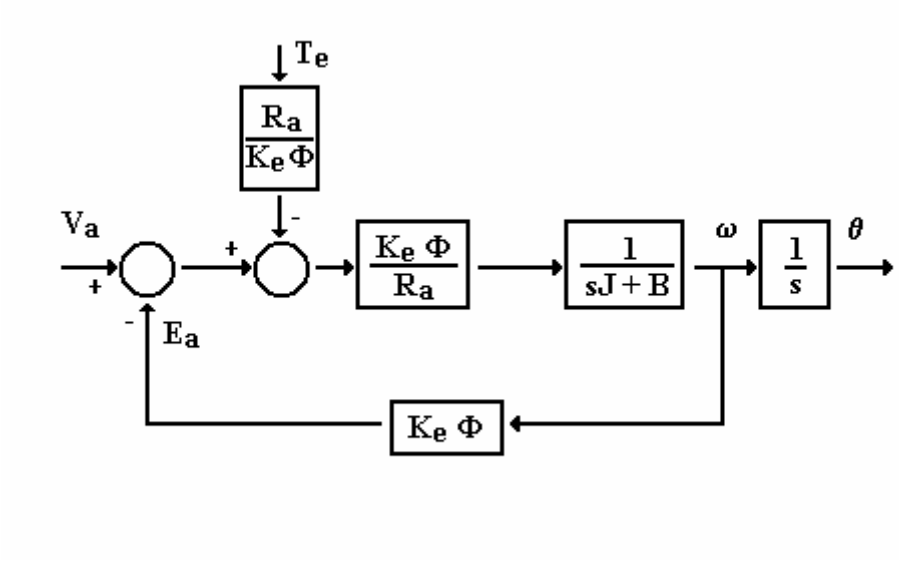



*Figura 4.8 . Diagrama de Blocos de um Motor CC Controlada pela armadura*

Para o motor CC tendo como variável controlada a velocidade de rotação e supondo o torque externo zero, a função de transferência de primeira ordem com:

Constante de tempo $$\tau = \frac{R_a \cdot J}{R_a \cdot B + K_e^{\,2} \cdot \Phi^2} \qquad 4.3$$

e ganho de malha aberta dado por

$$K = \frac{K_e \cdot \Phi}{R_a \cdot B + K_e^{\,2} \cdot \Phi^2}$$

O servomecanismo educacional da FEEDBACK pode ser configurado para controle por campo ou por armadura. O servoamplificador do conjunto quando configurado para trabalhar com controle por armadura, na verdade, não alimenta o enrolamento do campo em separado. O enrolamento do campo é colocado em série com o enrolamento da armadura.

A figura 4.9 mostra o modelo do motor usado para a identificação de parâmetros. Onde **τ** é o constante de tempo do sistema e o ganho em malha aberta é **K.** A redução de velocidade é **n,** e **AD** e **DA** são respectivamente os conversores analógico-digital e digital-analógico da interface de aquisição e distribuição do KIT-31. A

velocidade é medida com um tacômetro e a posição com um potenciômetro denominados respectivamente **Taco** e **Pot.** O torque externo é acoplado ao sistema a partir da constante de torque **Kt.**

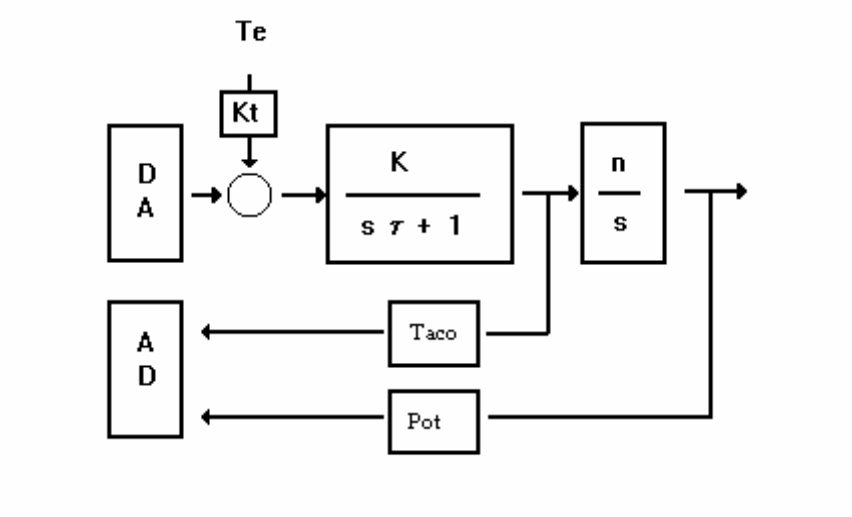



*Figura 4.9 . Modelo Dinâmico completo*

## 4.2.2 Determinação dos parâmetros

Como pretendia-se identificar o processo a partir do próprio micro controlador e sua interface de aquisição e distribuição, foi elaborado um programa em linguagem-C51 para o KIT-31 que cria todo um ambiente para acionar o motor a partir da interface DA. As velocidades e posições são obtidos a partir do conversor analógico-digital da interface de distribuição e aquisição do KIT-31 e guardados na sua memória. Estes dados podem ser acessados a partir de um microcomputador IBM-PC ligado através da interface serial com o KIT-31. Um outro programa executado no PC fornece uma interface entre o usuário e o KIT-31 e transfere os dados com informação de velocidade e posição para o PC, que depois podem ser gravados em arquivo no formato MATLAB.

O ponto de referência para identificação foi o de descrever o sistema de posicionamento em unidades de um bit do micro controlador e os conversores analógico-digital (AD) e digital-analógico (DA) numa escala de -128 a 128 unidades de posição e velocidade. O conversor AD e os respectivos ganhos relacionados com ele foram projetados de tal modo a casar com os valores máximos do tacômetro e o potenciômetro sensor de posicão. Igualmente o conversor DA e a unidade não linear também foram projetados para excitar toda entrada do servo pré-amplificador.

Na figura 4.10 está esquematizada parte da interface de aquisição de dados. Os amplificadores operacionais U1 e U2 ajustam os sinais provenientes do tacômetro e potenciômetro respectivamente e estes são depois amostrados e multiplexados no canal do conversor AD pelo circuito formado por U3, U4 e U5a. Vale ressaltar que os sinais são amostrados simultaneamente nos seus respectivos circuitos até ser quantificados pelo conversor AD.

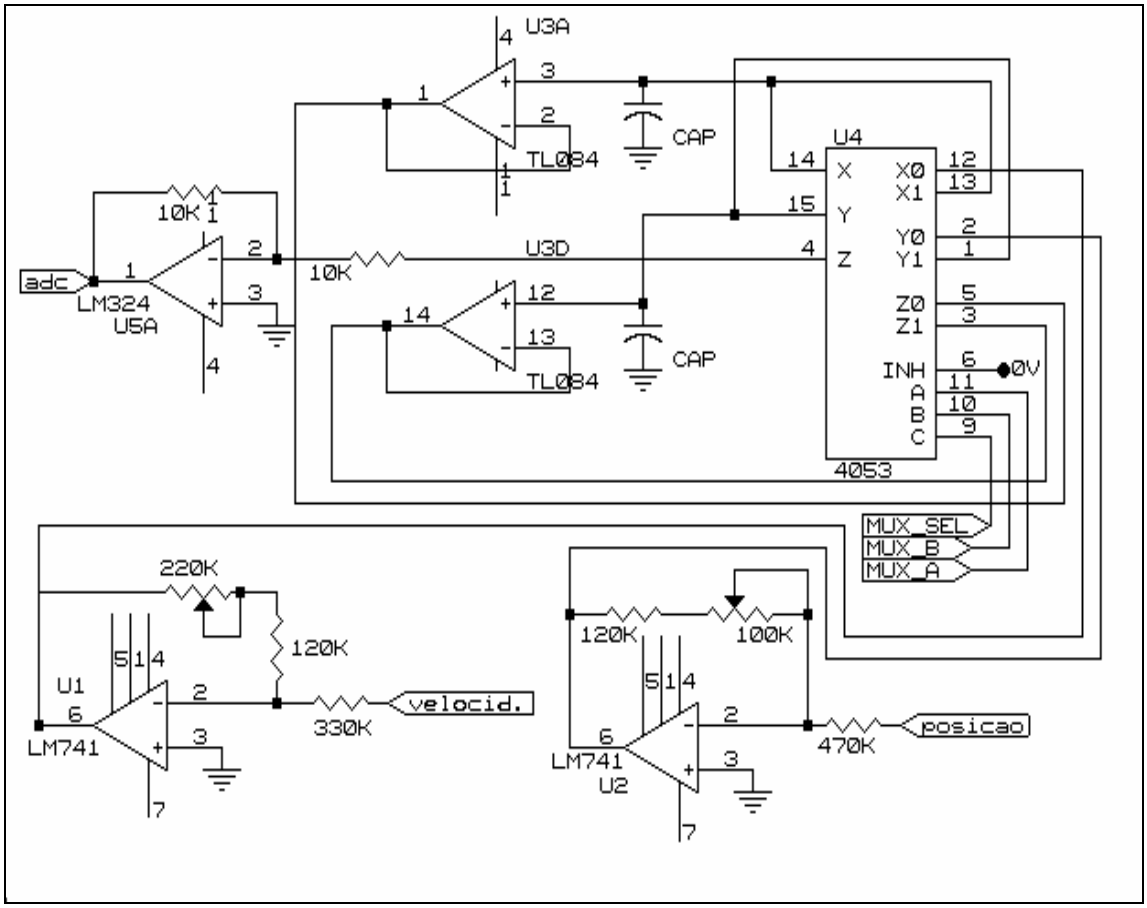



*Figura 4.10. Amostrador-retentor e multiplexador*

A caracterização estática dos conversores AD e DA e sensores de posição e velocidade é:

- Potenciômetro de posição (POT)

O ganho do amplificador operacional U2 foi ajustado para ter uma excursão máxima de +4 −4 volts para a excursão máxima do potenciômetro de +170$^{o}$ e −170$^{o}$.
POT = 4 volts / ((170$^{o}$/180$^{o}$) * $\pi$) rad = 1.348 V/rad

- Tacômetro no eixo principal (TACO)

O ganho do amplificador operacional U1 foi ajustado para ter uma excursão máxima de +4 −4 volts para a velocidade máxima do motor com o potenciômetro acoplado no eixo.
TACO = 0.006 Vs/rad

- Conversor analógico digital (AD)

O conversor ADC0800, parte da interface de aquisição e distribuição do KIT-31 (figura 3.4), utiliza palavras de 8 bits, e realiza a conversão por aproximações sucessivas. O conversor ADC0800 converte o sinal para um byte no formato complemento de um, representando 4 Volts por 00H e −4 Volts por FFH. Deste byte é subtraído 80H para mudar o formato para complemento de dois, que é o formato que o MC8031 do KIT-31 usa internamente para as operações aritméticas
AD = 256 unid. / (4V + 4V) = 32 unid. /Volts

- Conversor digital analógico (DA)

O conversor DAC1408 é um conversor digital-analógico de 8 bits com saída em corrente e encontra-se na placa de aquisição e distribuição do KIT-31 (figura 3.4). Um amplificador operacional faz a conversão corrente/tensão, para conexão com a unidade de compensação não-linear. Na figura 4.2 podemos ver que o pré-amplificador satura para tensões de −0.4 Volts e +0.4 Volts e que a faixa morta é de aproximadamente 10% desses valores. O conversor/corrente tensão foi ajustado para ter um excursão máxima igual à tensão de saturação.

DA = (0.4V + 0.4V) / 256 unid. = 0.003125 Volts/unid.

- Redução de engrenagens (n)

n = 1/30

### 4.2.3 Resultados

As características dinâmicas do motor foram determinadas com ensaios da resposta do sistema excitado por uma entrada degrau fornecida pelo microcontrolador e compensada pela unidade não linear.

Supondo inicialmente o torque da carga zero e monitorando a velocidade no eixo principal, observa-se que o sistema tem um comportamento de um sistema de primeira ordem com um ganho **K** e uma constante de tempo de **τ** :

**K** = 0.75 unid

**τ** = 0.3 seg

Depois foi determinada a constante de torque do motor utilizando um dispositivo de freio magnético sobre um disco no eixo do servomecanismo FEEDBACK, Com a velocidade fixada em 1000 rpm (17 unid.), foi medida a tensão na entrada para vários valores de torques de carga (freio), obtendo-se assim a relação torque-tensão :

**Kt** = 1650 Nm/unid.

# 4.3 - MANIPULADOR DE UM GRAU DE LIBERDADE

Para estudar o movimento das juntas robóticas foi montado um braço no eixo de baixa velocidade do motor do conjunto MS150, implementando assim um manipulador de um grau de liberdade. O braço é de alumínio tendo 50 gramas de peso e 30cm de comprimento, e assume-se a massa concentrada na ponta.

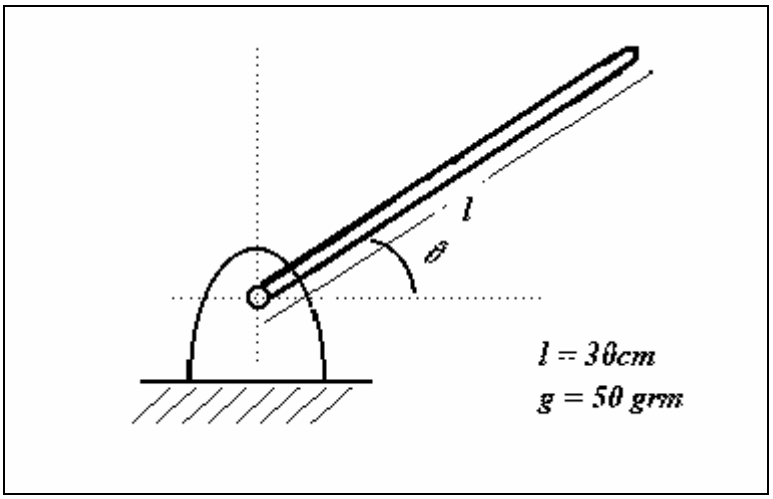



*Figura 4.11 . manipulador de 1 grau de liberdade*

A dinâmica do braço é dada pela equação de torque:

$$\tau = M(\theta)\ddot{\theta} + V(\theta,\dot{\theta}) + G(\theta) \qquad 2.1$$

Onde:

$M(\theta)$ matriz nxn com os termos inerciais do manipulador

$V(\theta,\dot{\theta})$ vetor nx1 com os termos de forças centrífugas e de coriolis.

$G(\theta)$ vetor nx1 com os termos de forças gravitacionais.

$\tau$ vetor dos torques.

O modelo foi obtido com o programa de cálculo simbólico mencionado no item 2.4.1 tendo como entrada os seguintes dados:

Matriz de rotação

$$ {}^{0}_{1}R = \begin{bmatrix} c1 & -s1 & 0 \\ s1 & c1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad {}^{1}_{0}R = \begin{bmatrix} c1 & s1 & 0 \\ -s1 & c1 & 0 \\ 0 & 0 & 1 \end{bmatrix} $$

matriz tensor de inércia (massa centralizada)

$$ {}^{c_i}T_j = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} $$

vetor ponto rotação em relação à base

$$ {}^{i}P_j = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} $$

distância do centro de massa em relação à base

$$ {}^{i}P_{c_j} = \begin{bmatrix} l \\ 0 \\ 0 \end{bmatrix} $$

$V, \omega$ - velocidades da base = 0

$\dot{V}, \dot{\omega}$ - aceleração da base = 0

*f* - forças atuando na ponta = 0

*n* - momento atuando na ponta = 0

O resultado do programa é a equação de torque do braço de um grau de liberdade :

$$ \tau = ml^2\ddot{\theta} + lmg\cos(\theta) \qquad 4.4 $$

Assim, o modelo dinâmico do manipulador acionado pelo servomotor é mostrado na figura 4.12

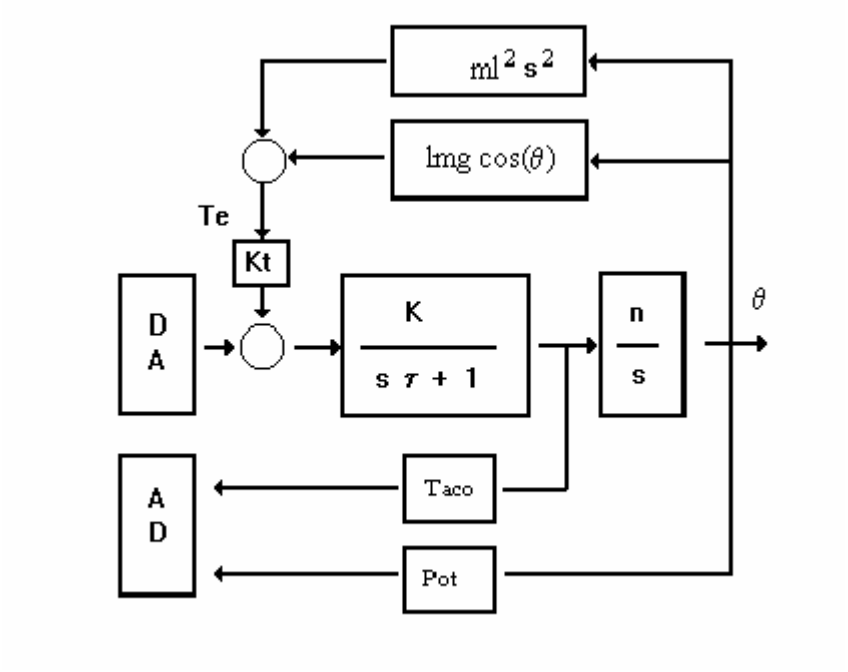



*Figura 4.12 . Modelo manipulador de 1 grau de liberdade*

# CAPÍTULO 5. DESENVOLVIMENTO E IMPLEMENTAÇÃO DE UM CONTROLADOR LOCAL DE JUNTA ROBÓTICA

O projeto do controlador baseado na equação 2.1 é geralmente complicado por causa da presença das características não-lineares e do acoplamento dinâmico entre as variáveis.

$$\tau = M(\theta)\ddot{\theta} + V(\theta,\dot{\theta}) + G(\theta) \qquad 2.1$$

Uma outra abordagem é determinar as entradas dos atuadores que acionam as juntas. Os atuadores são conectados às juntas através de engrenagens, onde a dinâmica do braço aparece como carga. O torque de carga é reduzido por um fator igual ao da redução de engrenagens e os efeitos das não-linearidades de acoplamento são reduzidos. Este fato nos permite projetar controladores independentes baseados nas equações 4.1 e 4.2 sendo o torque externo ($\tau$) tratado como perturbação ao sistema.

Neste capítulo são abordados os controladores Proporcionais-Derivativos, Proporcionais-Integrais-Derivativos sem levar em consideração a influência do torque, e em seguida levando o torque em consideração através da técnica de compensação computada. Uma nova técnica chamada controle por estrutura variável é discutida e implementada para comparar o seu desempenho em relação aos métodos tradicionais.

# 5.1 - CONTROLADOR PROPORCIONAL DERIVATIVO

O controlador proporcional derivativo (PD) é a implementação mais simples a ser usada como controlador de posição para motor. O termo proporcional corresponde a realimentação de posição e o termo derivativo corresponde a realimentação de velocidade.

A realimentação de velocidade atua no fator de amortecimento e a realimentação de posição na frequência natural não amortecida. Num robô manipulador o valor do fator de amortecimento requerido depende da aplicação. Em manuseio de peças, solda, etc., o fator de amortecimento deve ser maior ou igual à unidade (1) para garantir um movimento super-amortecido. São poucas as aplicações em robótica que aceitam movimento sub-amortecido.

Existem duas maneiras para implementar o algoritmo de controle proporcional-derivativo. Na primeira (Tipo A), a velocidade é realimentada com um ganho Kv.

Na segunda implementação (Tipo B) supõe-se que não seja possível medir o valor da velocidade, então é necessário derivá-la da posição medida.

## 5.1.1 Controlador PD: Implementação tipo A

A figura 5.1 mostra a realização do Tipo A no plano-S.

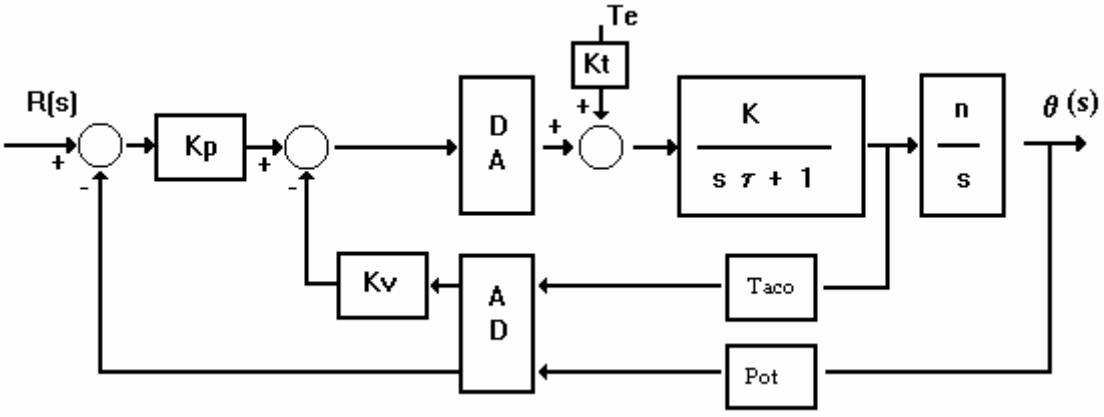



*Figura 5.1. Controlador PD Tipo A*

A função de transferência resulta em:

$$\frac{\theta(s)}{R(s)} = \frac{K \cdot DA \cdot n \cdot Kp}{\tau \cdot s^2 + (1 + K \cdot Kv \cdot AD \cdot Taco \cdot DA)s + AD \cdot K \cdot n \cdot pot \cdot DA \cdot Kp} \qquad 5.1$$

O trecho do programa para efetuar o algoritmo de controle em linguagem C-51 é mostrado a seguir.

*Algoritmo 2. Controlador PD tipo A*

```
void timer_0_int(void)
{
 output(TH0,TH);                          /* recarrega timer */
 output(TL0,TL);
 sample();                                /* sample hold     */
 posi=0x80-posi_atual();                  /* leia posicao    */
 velo=0x80-velo_atual();                  /* leia velocidade */
 acion=linear((ref-posi)*KP-velo*KV);     /* passa pelo VPU  */
 *(char *)DACend=acion;
}
```

## 5.1.2 Controlador PD: Implementação tipo B

A figura 5.2 mostra a implementação do Tipo B no plano-s.

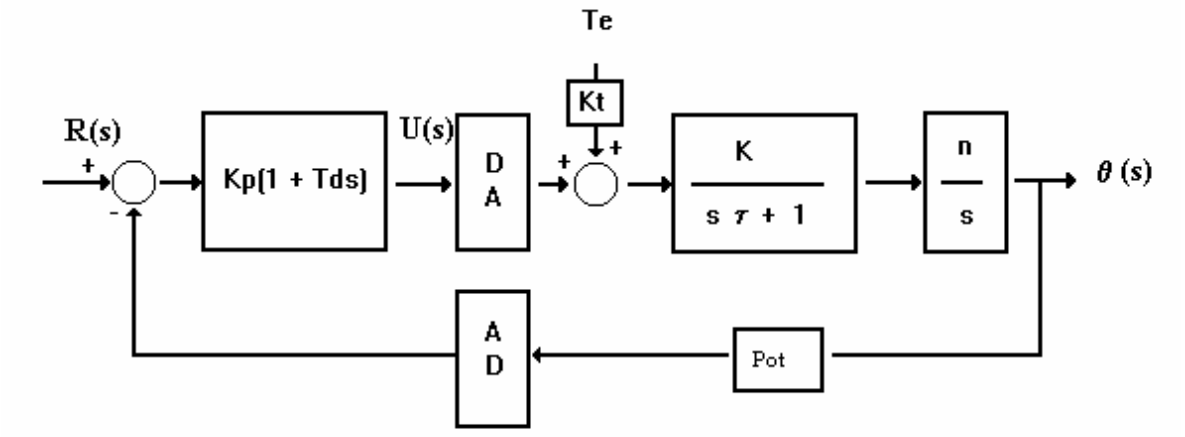



*Figura 5.2. Controlador PD Tipo B*

A função de transferência resulta em:

$$\frac{\theta(s)}{R(s)} = \frac{Kp \cdot K \cdot DA \cdot n(1+Td \cdot s)}{\tau \cdot s^2 + s + (1+Td \cdot s)Kp \cdot K \cdot DA \cdot n \cdot AD \cdot pot} \qquad 5.2$$

O controlador no plano-S é dado por:

$$\frac{U(s)}{E(s)} = Kp(1+Td \cdot s) \qquad 5.3$$

A implementação deste controlador é realizada pelo seu equivalente discretizado usando o método da aproximação retangular regressiva substituindo-se para ***s***:

$$s = \frac{z-1}{T \cdot z} \qquad 5.4$$

Resolvendo o sistema e tomando a transformada inversa chega-se a seguinte algoritmo de controle:

U(k)=q0*E(k) - q1*E(k-1) 5.5

onde q0=Kp*(1+Td/T) e

q1=Kp*Td/T

A implementação deste algoritmo de controle é mostrada no trecho de programa em linguagem C-51 a seguir.

*Algoritmo 3. Controlador PD tipo B*

```
void timer_0_int(void)
{
 output(TH0,TH);                    /* recarrega timer */
 output(TL0,TL);
 sample();                          /* sample hold     */
 posi=0x80-posi_atual();            /* leia posicao    */
 erro=ref-posi;
 acion=linear(q0*erro-q1*erro1));
 *(char *)DACend=acion;
 erro1=erro;
}
```

# 5.2 - CONTROLADOR PROPORCIONAL INTEGRAL DERIVATIVO

Na secçao anterior não foi levado em consideração o efeito do torque externo. Este torque tende a provocar uma diferença entre a posição real e posição desejada, e o controlador PD não tem meios para eliminar este erro. A eliminação do erro pode ser obtida adicionando ação integral ao controlador PD. A figura 5.3 mostra a implementação do controlador PID.

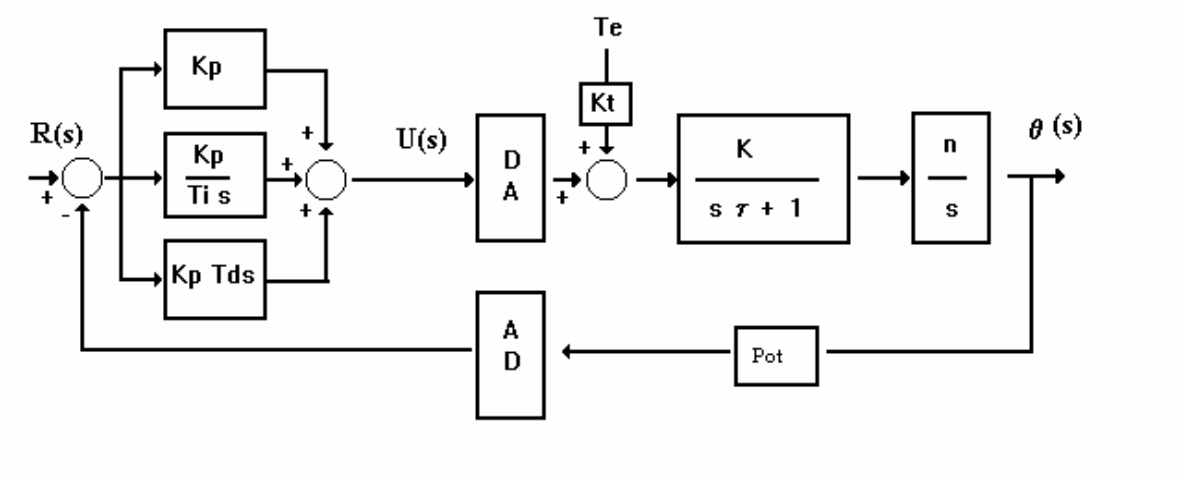



*Figura 5.3. controlador PID*

O controlador PID tem a forma de :

$$U(s) = Kp \cdot E(s) \cdot \left(1 + \frac{1}{Ti \cdot s} + Td \cdot s\right) \qquad 5.6$$

Para implementação do controlador pelo seu equivalente discretizado utiliza-se o método da aproximação retangular progressiva para o integrador e o método da aproximaçao retangular regressiva para o controlador derivativo.
Para o controlador integral substitui-se o ***s*** por:

$$s = \frac{z-1}{T} \qquad 5.7$$

Para o controlador derivativo substitui-se o ***s*** por:

$$s = \frac{z-1}{T \cdot z} \qquad 5.8$$

Substituindo o ***s*** tirando a transformada inversa chega-se ao seguinte algoritmo de controle:

$$U(k)=U(k-1)+q0*E(k)+q1*E(k-1)+q2*E(k-2) \qquad 5.9$$

onde q0= Kp(1+Td/T)
q1= Kp(-1+T/Ti-2Td/T)
q2= Kp*Td/T

A implementação deste algoritmo de controle é mostrada no trecho de programa em linguagem C-51.

*Algoritmo 4. Controlador PID*

```
void timer_0_int(void)
{
 output(TH0,TH);                        /* recarrega timer */
 output(TL0,TL);
 sample();                              /* sample hold     */
 posi=0x80-posi_atual();                /* leia posicao    */
 velo=0x80-velo_atual();                /* leia velocidade */
 erro=ref-posi;
 u=u1+q0*erro+q1*erro1+q2*erro_2;
 acion=linear(u);                       /* passa pelo VPU  */
 *(char *)DACend=acion;
 u1=u;
 erro1=erro;
}
```

# 5.3 - CONTROLE COM TORQUE COMPUTADO

Os controladores descritos nas seções anteriores consideraram o torque do braço como um distúrbio, o efeito do qual tente-se minimizar pela escolha conveniente dos parâmetros do controlador. Uma maneira direta de cancelar o seu efeito é com a técnica de compensação por torque computado (Craig, 1989).

Em geral, a técnica de compensação por torque computado calcula o valor do torque em cima do motor para cada posição do manipulador e conseqüentemente para cada grau de liberdade. Este cálculo é realizado "online" pelo computador supervisor, e pode empregar o algoritmo de Newton-Euler ou de Lagrange-Euler. A forma genérica das equações de Newton-Euler na sua forma matricial é:

$$\tau = M(\theta)\ddot{\theta} + V(\theta,\dot{\theta}) + G(\theta) \qquad 2.1$$

Essa forma também, é chamada de equação de estado porque o termo V depende das velocidades e da posições.

Uma outra alternativa para montar as equações é chamada de equação no espaço de configuração ("configuration space equation"), re-escevendo $V(\theta,\dot{\theta})$

$$\tau = M(\theta)\ddot{\theta} + B(\theta)[\dot{\theta}\dot{\theta}] + C(\theta)[\dot{\theta}^2] + G(\theta) \qquad 5.9$$

onde:

$B(\theta)$ matriz nxn(n-1)/2 dos coeficientes de coriolis

$[\dot{\theta}\dot{\theta}]$ vetor n(n-1)/2x1 de velocidades

$C(\theta)$ matriz nxn de coeficientes centrífugas

$[\dot{\theta}^2]$ vetor nx1 com o quadrado das velocidades

e **n** é o número juntas.

A vantagem desta formulação é que os parâmetros só aparecem em função da posição do manipulador $\theta$. Os parâmetros $M(\theta)$,$B(\theta)$,$C(\theta)$ e $G(\theta)$ podem ser atualizados à velocidade com que o manipulador está se movendo (Craig, 1989).

A principal dificuldade encontrada para implementar o controle por torque computado é a capacidade computacional do controlador. O controlador deve ser capaz de calcular em tempo real o torque que o manipulador desenvolve em cada junta da estrutura.
Quer dizer, para cada amostragem deve-se:
1- Atualizar os $M(\theta)$,$B(\theta)$,$C(\theta)$ e $G(\theta)$
2- Calcular ou medir velocidade e aceleração

3- Calcular a equação do torque

## 5.3.1 PD com torque computado

Nesta secção, é desenvolvido um controlador PD com compensação do torque, o qual é computado em tempo real. Para o sistema em estudo a implementação é mostrada na figura 5.4.

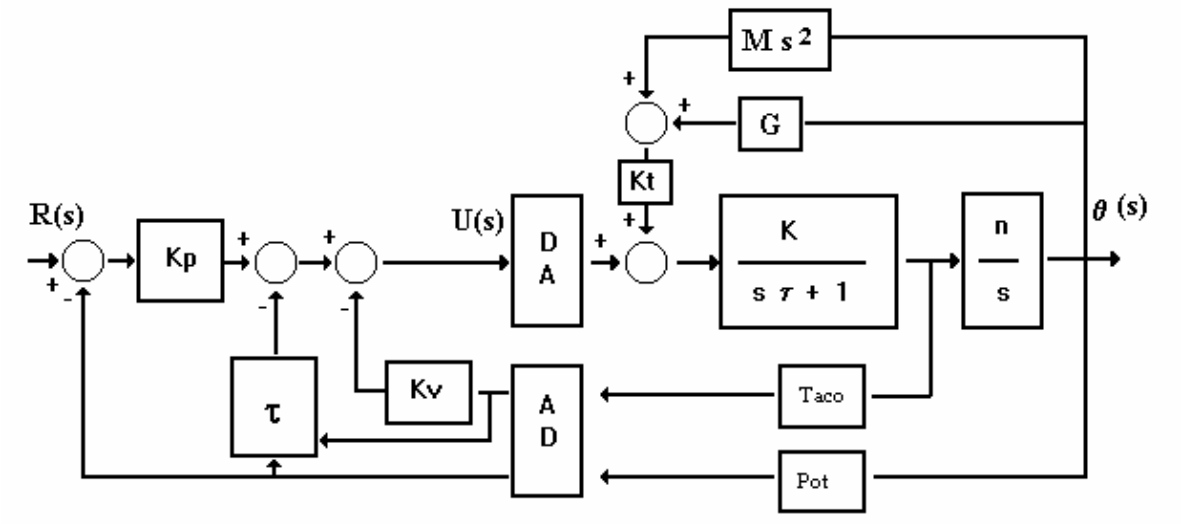


*Figura 5.4. Controlador PD com torque computado*

O controlador calcula para cada movimento do manipulador o torque $\tau$ correspondente a partir dos sinais de posição e velocidade, e fornece este valor para a placa controladora de junta que agrega este valor ao acionamento.

A equação do torque, já obtida na seção 4.3 deste trabalho é:

$$\tau = ml^2 \ddot{\theta} + lmg\cos(\theta) \qquad 4.4$$

O trecho de programa em C-51 que efetua o controle é mostrado na listagem a seguir:

*Algoritmo 5. Controlador PD com torque computado*

```
#define p=1/(DA*POT)
static float acel,Mteta=massa*comprim^2,mlg=massa*comprim*9.8;
static int torque,posi,velo,velo_1;

void timer_0_int(void)
{
 output(TH0,TH);                                /*recarrega timer   */
 output(TL0,TL);
 sample();                                       /* sample hold      */
 posi=0x80-posiatual();                          /* leia posicao     */
 velo=0x80-veloatual();
 acel=(velo-velo1)/T;
 torque=(int)(Mteta*acel+mlg*cos(π-posi*p)); /* cálculo do torque */
 erro=ref-posi;
 u=Kp*erro-velo*Kv+torque;
 acion=linear(u);                                /* passa pelo VPU    */
 *(char *)DACend=acion;
 velo1=velo;
}
```

Na seção 3.5 foi mencionado que a utilização de aritmética de ponto flutuante seria substituída por aritmética de ponto fixo e tabelas de referência ("look-up tables"). Uma tabela com os valores do torque para cada unidade de posição do braço foi computado e armazenada na memória do programa. A partir de simulações foi verificado que a força inercial é pequena em relação à força gravitacional.

O algoritmo 6 mostra a rotina simplificada utilizado para implementar o controlador.

*Algoritmo 6. Controlador PD simplificado com torque computado*

```
static int torque,posi,velo;
extern float  seno[128];   /* tabela com os valores précalculados */

void timer_0_int(void)
{
 output(TH0,TH);                              /*recarrega timer*/
 output(TL0,TL);
 sample();                                    /* sample hold   */
 posi=0x80-posiatual();                       /* leia posicao  */
 velo=0x80-veloatual();
 torque=(int)(seno[posi<0 ? -posi:posi]);
 erro=ref-posi;
 u=Kp*erro-velo*Kv+torque;
 acion=linear(u);                            /* passa pelo VPU */
 *(char *)DACend=acion;
 erro1=erro;
 velo1=velo;
}
```

# 5.4 - ESTRUTURA VARIÁVEL

O controle por torque computado depende do conhecimento exato dos parâmetros do braço manipulador, sendo portanto sensível a erros na estimação destes parâmetros. Uma maneira de eliminar esta dependência do modelo dinâmico do braço, é de mudar bruscamente a estrutura do controlador em função do estado do sistema a ser controlado (Hung,1993)(Fu-Juay Chang, 1990). Controladores deste tipo são chamados sistemas de estrutura variável.

Para controle por estrutura variável, não é necessário conhecer os parâmetros exatos do braço manipulador. Precisa-se somente dos valores máximos e mínimos. Controladores de estrutura variável são robustos no sentido que não são sensíveis a erros na estimativa do valor dos parâmetros.

Para o sistema eletromecânico apresentado na figura 4.9 com o torque externo zero, definindo Y como o sinal elétrico de posição e U como o sinal de controle entrando no conversor DA, a função de transferência do sistema é dado pela equação:

$$\frac{Y(s)}{U(s)} = \frac{DA \cdot K \cdot n \cdot pot \cdot AD}{\tau \cdot s^2 + s} \qquad 5.10$$

sendo a equação diferencial correspondente:

$$\tau \cdot \ddot{Y} + \dot{Y} = K_0 \cdot U \qquad 5.11$$

onde $K_0 = DA \cdot K \cdot n \cdot pot \cdot AD$

Realimentando a variável controlada **Y** com ganho unitário, sendo ***R*** a entrada de referência e $x_1 = R - Y$ o sinal de erro, pode-se modelar o sistema em relação a $x_1$ :

Escolhendo o sinal de erro como variável de estado:

$$x_1 = R - Y \qquad 5.12$$

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -(1/\tau)\cdot x_2 + (\ddot{R} + \dot{R}/\tau) - K_0 \cdot u \end{cases} \qquad 5.13$$

rescrevendo :

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -(1/\tau)x_2 - K_0 \cdot u + f_R(R,t) \end{cases} \qquad 5.14$$

onde $f_R(R,t)$ é uma função de $R$ .

para implementar controle por estrutura variável o controlador tem a estrutura genérica de (Utkin, 1993):

$$u = f_C(x,s) \quad \text{, onde} \qquad 5.15$$

$$s(x) = cx_1 + x_2 \qquad 5.16$$

A função $s(x)$ é que define a mudança da estrutura do controlador, e o sinal de controle $u$ é uma função de $s(x)$ e do estado $x$. A linha $s(x)= 0$ , também chamada de linha de comutação ("switching line") é um sistema linear e independe das características do sistema a controlar. Somente a escolha de $c$ determina a velocidade com que a estrutura do controlador é mudada.

Quando o sistema está operando na linha de comutação, diz-se que está no modo deslizante ("sliding mode"). Substituindo 5.14 em 5.15 obtém-se a equação 5.17

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -(1/\tau)x_2 - K_0 \cdot f_C(x,s) + f_R(R,t) \end{cases} \qquad 5.17$$

O problema do projeto é encontrar uma função de controle de modo que as trajetórias no plano de fase tendam para a linha $s(x)=0$.

Escolhendo um controlador mostrado na figura 5.5 e a função de controle da forma:

$$u = f_C(x,s) = \begin{cases} \varphi_1 \cdot x_1 \Rightarrow s(x) \cdot x_1 > 0 \\ \\ \varphi_2 \cdot x_1 \Rightarrow s(x) \cdot x_1 < 0 \end{cases}$$

5.18

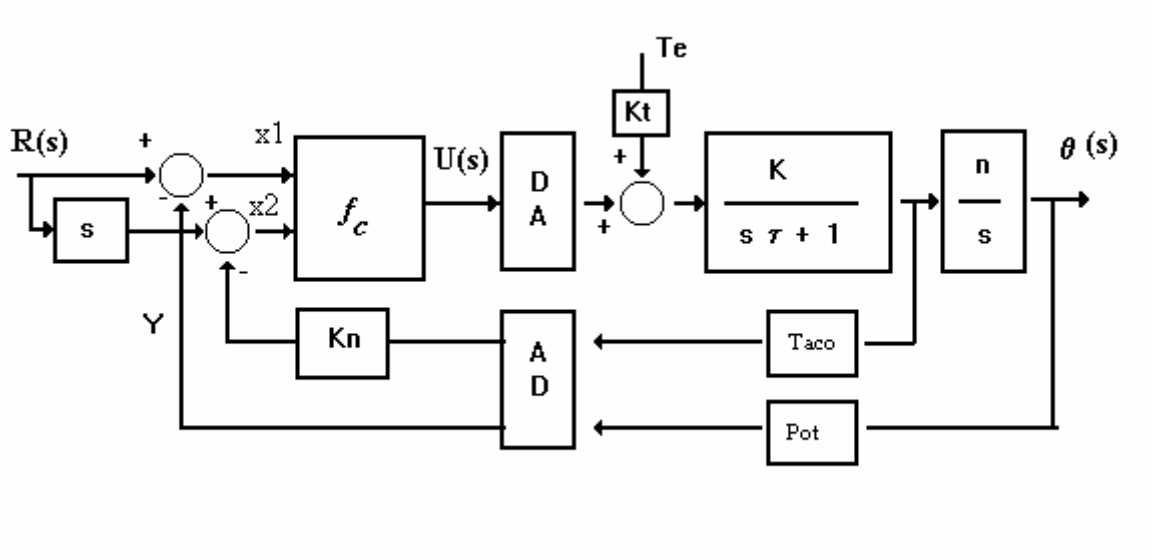



*Figura 5.5. controle por estrutura variável*

A velocidade $x_2$ na implementação do controlador não é medida no eixo de baixa velocidade. Ela é medida pelo tacômetro e multiplicada por um fator de escala Kn, como representado na figura 5.5

O plano de fase do sistema mostra como é o comportamento do controlador. Ele pode ser do tipo "ponto de sela" ou "foco estável" em função da escolha de ***f1*** e ***f2***, como mostrado na figura 5.6.

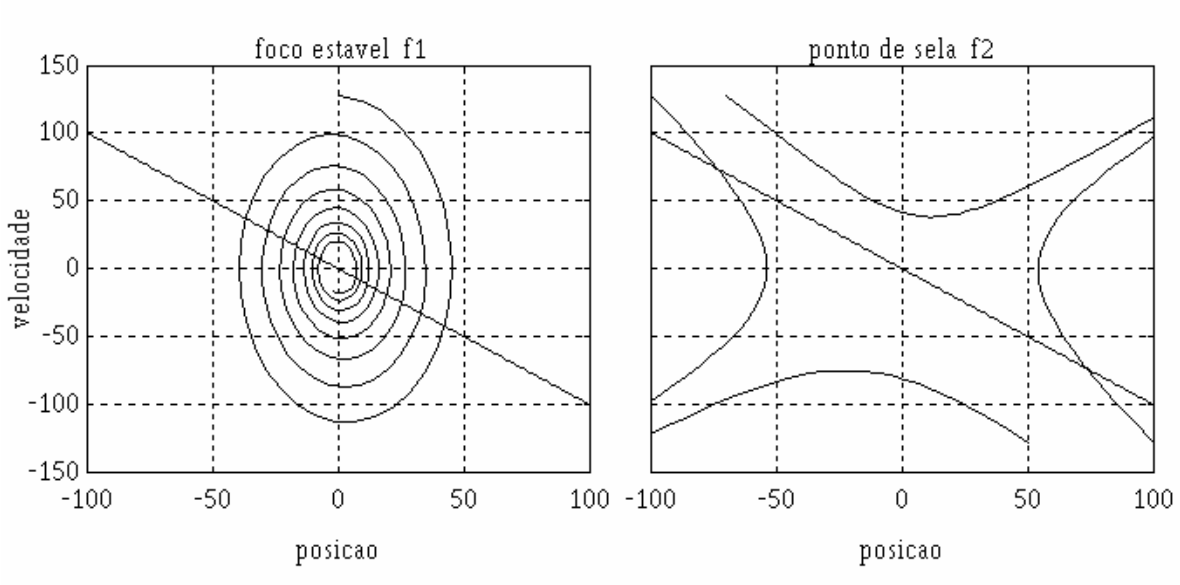



*Figura 5.6 Plano de fase*

A escolha dos parâmetros do controlador consiste em escolher valores de f1, f2 e c, de tal modo que as variáveis de posição $(x_1)$ e velocidade $(x_2)$ praticamente deslizam em cima da linha ***s(x)=0.*** , como mostrado na figura 5.7.

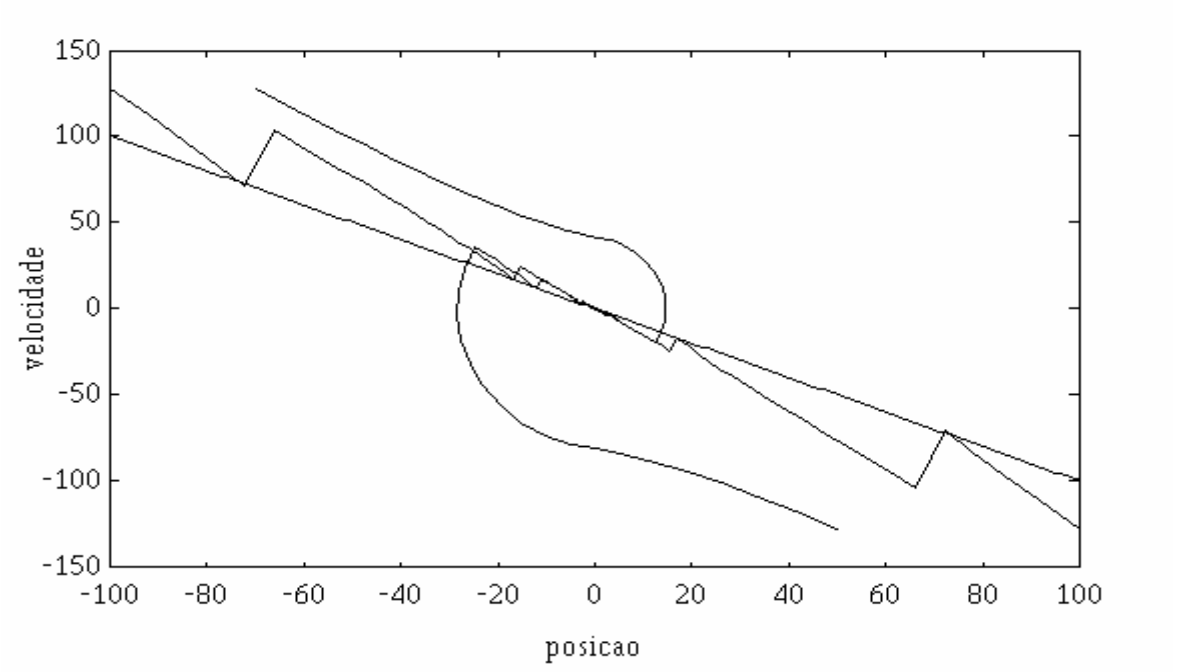



*Figura 5.7 Modo Deslizante*

A implementação deste algoritmo de controle é mostrado no trecho de programa em linguagem C-51.

*Algoritmo 7. Controle por estrutura variável*

```
/*  norm=pot*nx/taco=7.06;        */
/*  c=1  fi1=-20  fi2=5           */

void timer_0_int(void)
{
 output(TH0,TH);                          /* recarrega timer */
 output(TL0,TL);
 sample();                                /* sample hold     */
 posi=0x80-posi_atual();                  /* leia posicao    */
 velo=norm*(0x80-velo_atual());           /* leia velocidade e normaliza
*/
 dref=(ref-ref_1)/T;
 x1=ref-posi;
 x2=dref-velo;
 if (((c*x1 + x2) > 0) xor (x1 > 0)
  u=fi2*x1; else u=fi1*x1;
 acion=linear(u);                         /* passa pelo VPU  */
 *(char *)DACend=acion;
 ref_1=ref;
}
```

A escolha dos parâmetros de todos os controladores apresentados neste capítulo foram basicamente efetuados por tentativa e erro. Com a simulação do sistema chegou-se a valores aproximados e estes depois eram verificados através de ensaios.

No capítulo 6, serão apresentados os parâmetros usados para cada controlador e os resultados obtidos nas simulações e na implementação.

# CAPÍTULO 6. RESULTADOS E CONCLUSÕES

Neste capítulo serão apresentados resultados de ensaios efetuados com os diferentes controladores aplicados ao manipulador em estudo. Foram efetuados ensaios para cada tipo de controlador sem o braço acoplado e com o braço acoplado. O sistema deve seguir uma trajetória de referência definida e armazenada na memória.

A trajetória desejada é gerada pelo computador supervisor em forma de tabela sendo depois transferida para o controlador de junta através da interface serial. O programa de controle executado pelo KIT-31 aceita cinco comandos do computador supervisor.

- receber uma tabela com pontos de referência
- ajustar os parâmetros do controlador
- ligar o controlador e armazenar num buffer interno os primeiros 256 valores inteiros da posição, velocidade e sinal de controle.
- retornar ao supervisor o conteúdo do buffer de posição, velocidade e sinal de controle.
- desligar o controlador

O algoritmo de controle é executado através de uma interrupção temporizada com tempo definido pela frequência de amostragem. O tempo de amostragem para todos os ensaios foi 10 milisegundos. A listagem completa dos programas se encontra no apêndice.

O sinal de referência usado nos ensaios é uma trajetória que varia de 0 a 100 unidades, gerada com o algoritmo de interpolação quadrática apresentado na seção

2.3.1. Nos ensaios avalia-se como o sistema acompanha o sinal de referência.

A escolha dos parâmetros do controlador foi basicamente efetuada por tentativa e erro, primeiro, simulando e analisando o comportamento do sistema e depois procedendo à sintonia fina de parametros com o sistema montado.

Na seqüência a seguir serão mostradas as respostas simuladas e reais de cada tipo de controlador.

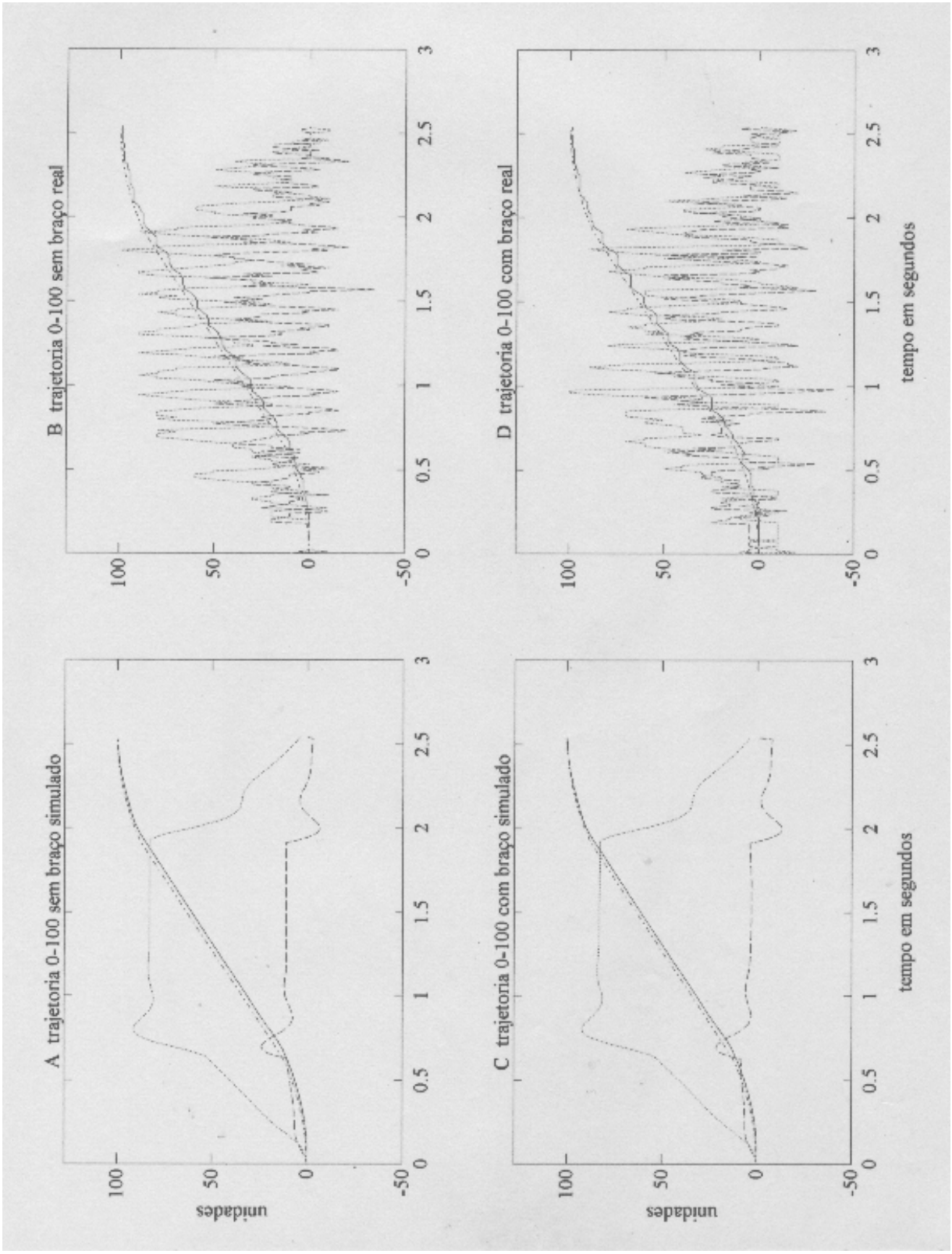
## 6.1 - CONTROLADOR PROPORCIONAL DERIVATIVO: IMPLEMENTAÇÃO TIPO A

O controlador proporcional derivativo tipo A foi programado para trabalhar com ganhos de Kp=20 e Kv=5 (fig.5.1). Na figura 6.1.a e figura 6.1.c são apresentadas as simulações do comportamento do sistema sem o braço e com o braço acoplado respectivamente. A figura 6.1.b e figura 6.1.c mostram o comportamento real sem e com o braço acoplado ao sistema.

Em todas as figuras foi convencionado que o sinal de posição é representado por uma linha cheia. O sinal de velocidade é representado por uma linha pontilhada. O sinal de referência por uma linha com pontos e traços intercalados e, finalmente, o sinal de controle por uma linha tracejada. Somente o sinal de velocidade foi plotado com uma escala de 10:1 para facilitar a comparação.

Podemos observar nas figuras 6.1.a e b que existe um erro de posição. Quando a entrada aumenta linearmente com o tempo, o erro permanece constante. Distinguimos claramente os pontos de aceleração e desaceleração pelos picos do sinal de controle. O efeito do braço pode ser notado pela diminuição do sinal de controle na figura 6.1.b. O braço inicialmente na posição vertical (0) gira até aproximadamente $133^{o}$ (100) em relação ao eixo vertical.

*Figura 6.1. Controlador PD Tipo A*



O sinal de posição e controle nas figuras 6.1.b e d têm oscilações significativas. Acredita-se que essas oscilações são consequências de ruídos provenientes do tacômetro e potenciômetro ou provocados pelo dispositivo de compensação não-linear. Percebe-se que o sinal de controle é o erro multiplicado pelo ganho Kp (20).

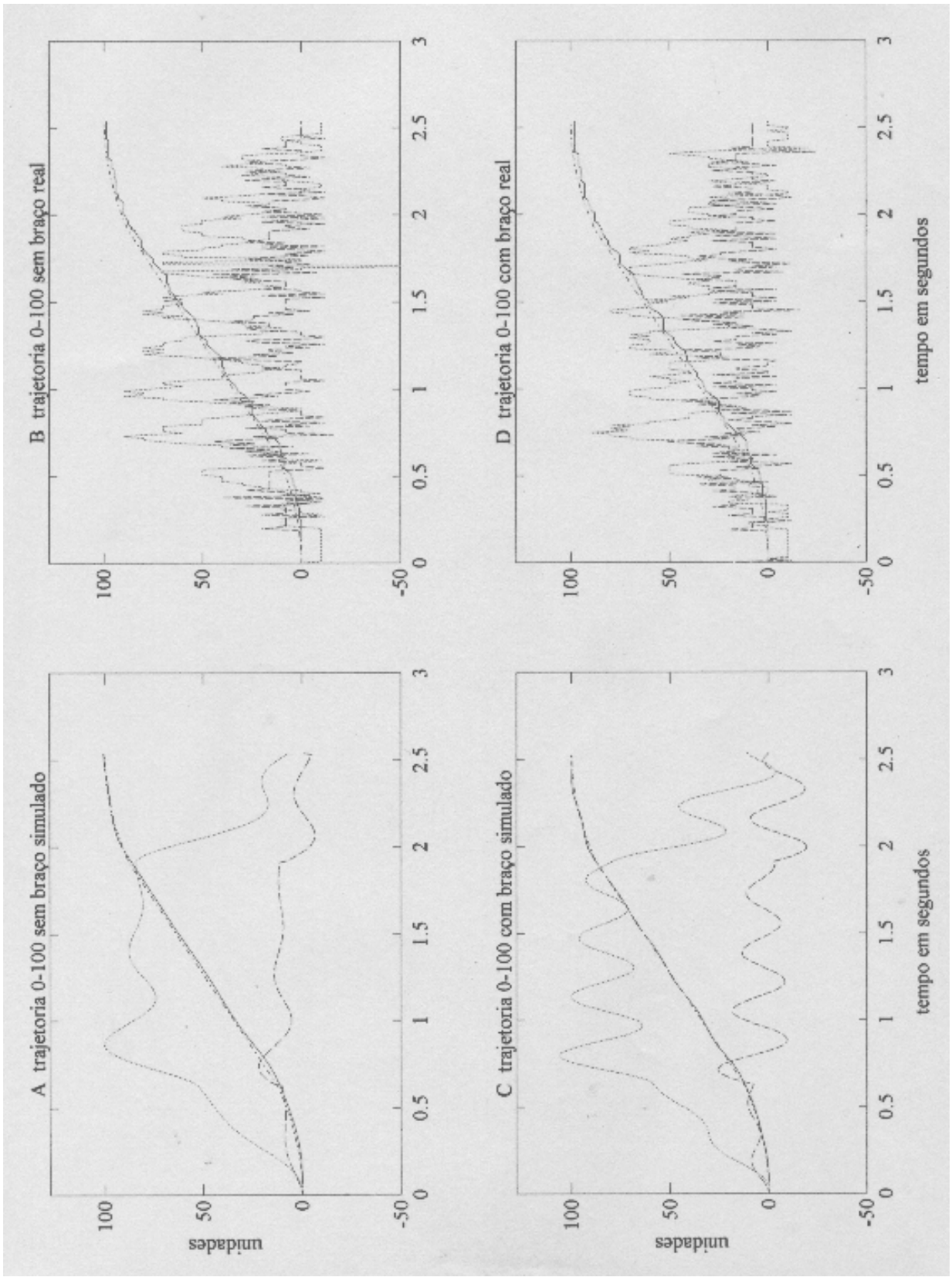
## 6.2 - CONTROLADOR PROPORCIONAL DERIVATIVO: IMPLEMENTAÇÃO TIPO B

O controlador proporcional derivativo tipo B foi programado para ganhos de q0=20 e q1=15. O ganho proporcional q0 foi mantido propositadamente igual a Kp do ensaio anterior, enquanto o q1 foi obtido através de tentativas e erros. Na figura 6.2 a velocidade representada pela linha pontilhada é a velocidade do eixo de alta velocidade.

O erro no acompanhamento da trajetória nas simulações é menor que o sistema da seção anterior, mas, em contra partida, o sistema se mostra mas sensível à ação do braço: O sinal de controle de 6.2.c oscila mais em relação à fig. 6.2.a.

O comportamento oscilatório do sistema real pode provalvelmente ser atribuido ao fato que o cancelamento da não-linearidade (faixa morta) não é perfeito.

*Figura 6.2. Controlador PD Tipo B*



## 6.3 - CONTROLADOR PROPORCIONAL INTEGRAL DERIVATIVO

A escolha dos parâmetros dos controladores proporcionais derivativos foram efetuados pelo método de tentativa e erro. Ao aplicar o mesmo método para determinar os parâmetros do controlador PID o método não mostrou resultados satisfatórios. O uso das regras de sintonia de Ziegler-Nichols (Franklin, 1990) tampouco levou a um resultado satisfatório. O sistema mostrou-se instável ou não conseguiu acompanhar a trajetória desejada.

A única maneira de conseguir um comportamento razoável foi com o uso de um programa de otimização numérico para parâmetros de controladores PID desenvolvido por Huppes (1992) no laboratório de controle de processos por computador. Este programa otimiza parâmetros de controladores PID para garantir ao sistema um comportamento desejado quando o excitado por uma entrada tipo degrau. O programa fornece vários critérios para a otimização.

Para o objetivo proposto neste trabalho, escolheu-se a otimização numérica em relação a um modelo de referência. O modelo utilizado foi um sistema sub-amortecido com um sobre-passo a uma entrada degrau de 10% e um tempo de pico de 0.5 segundos.

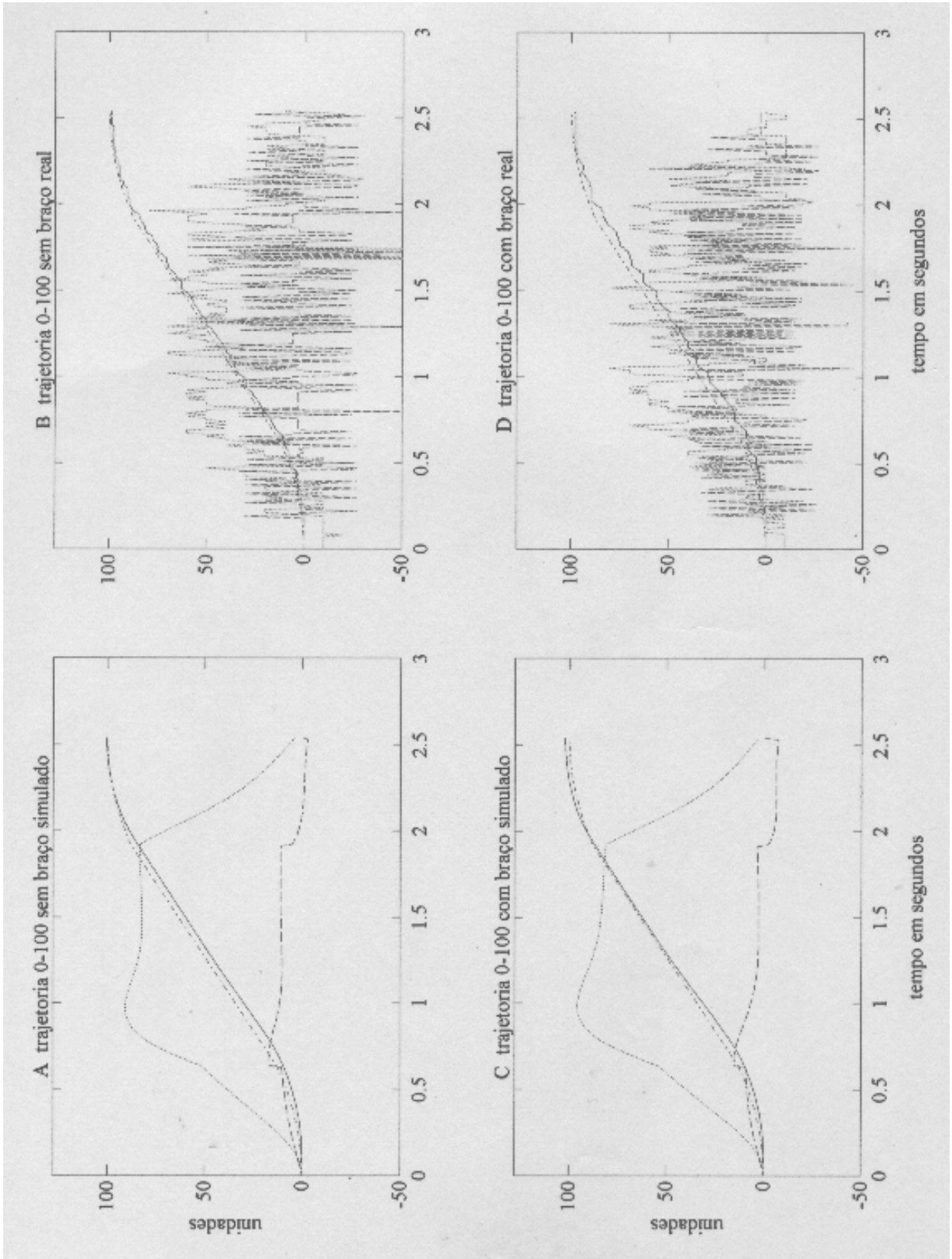
A escolha deste modelo foi baseado no comportamento do mesmo sistema com o controlador PD da secção anterior quando excitado por uma entrada tipo degrau e os valores dos parâmetros obtidos foram q0=30 q1=-57 q2=27.

Entretanto o algoritmo de controle não mostrou-se muito robusto, no sentido de que uma variação de 5% nos parâmetros, pode deteriorar o desempenho. Há necessidade de investigar outros métodos de projeto de controladores PID para este caso.

As figuras 6.3 A e C mostram as simulações sem braço e com braço respectivamente. As figuras 6.3.B e D mostram o comportamento real do sistema quando seguindo a trajetória proposta, sem e com o braço respectivamente.

Observe-se nas simulações que o sistema não consegue parar no ponto final com o braço acoplado, e o efeito do braço também fica clara na magnitude do sinal de controle. O erro na trajetória é significativo com o braço na figura 6.3.d

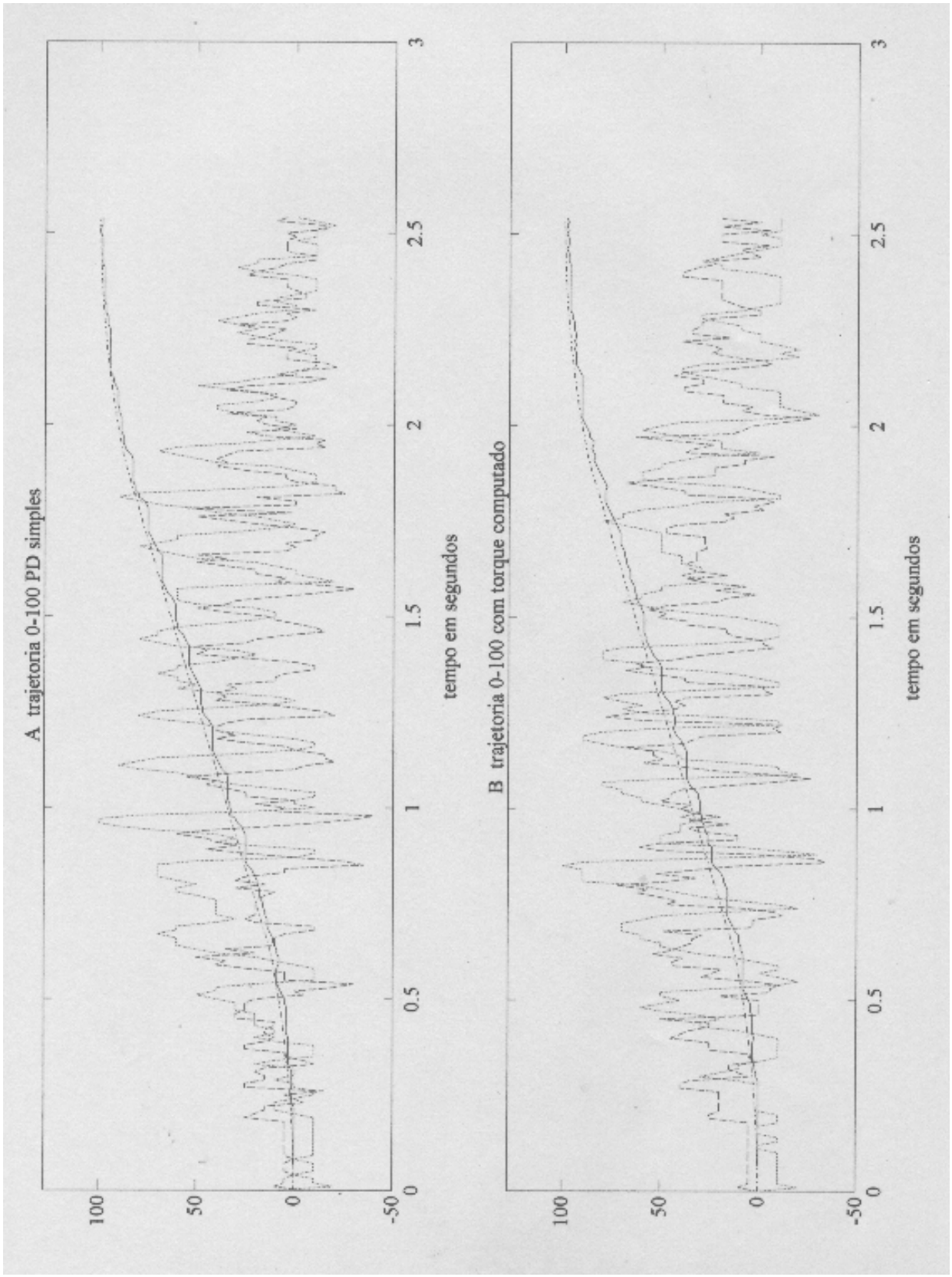
*Figura 6.3. Controlador PID*



## 6.4 - CONTROLE PD COM TORQUE COMPUTADO

Os parâmetros Kp e Kv utilizado neste ensaio são os mesmos utilizados no ensaio descrito na seção 6.1 (Kp=20, Kv=5). A figuras 6.4.A mostra o comportamento do controlador PD tipo A com os mesmos parâmetros e com o braço acoplado e a figura 6.4.B mostra o controlador PD com torque computado.

*Figura 6.4. Controlador PD com torque computado*



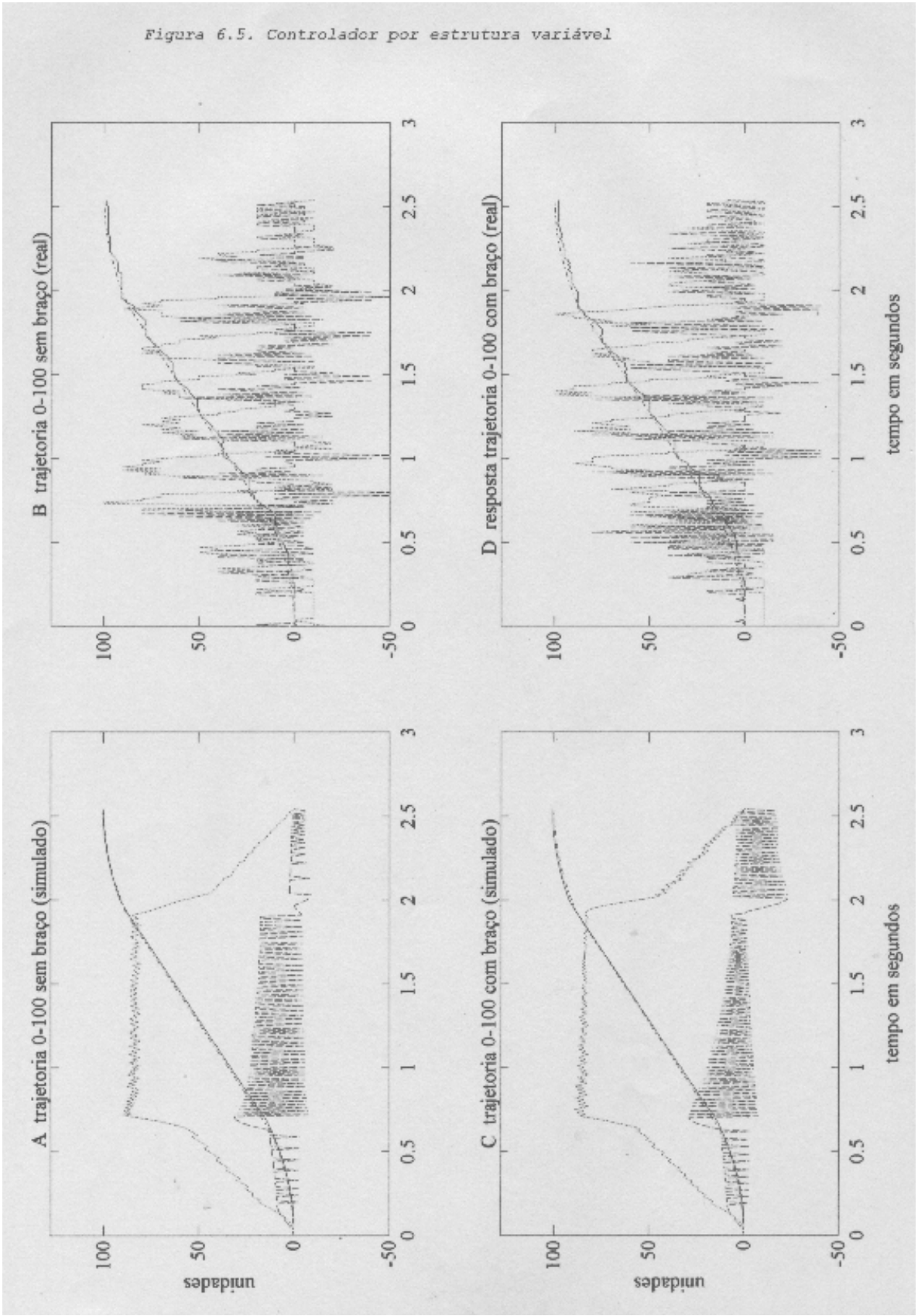
# 6.5 - CONTROLE POR ESTRUTURA VARIÁVEL

A escolha dos parâmetros do controlador por estrutura variável mostrou-se a mais complicada por envolver 3 variáveis interdependentes (f1, f2, c). Primeiramente escolheram-se valores arbitrários para f1 e f2 e traçou-se o plano de fase do sistema como mostrado na figura 5.6. Fixado f1 e f2, traçou-se no plano de fase as linha $s(x) = cx_1 + x_2 = 0$ e $x_1 = 0$ de tal modo para dividir o plano de fase em quatro áreas. A área definido por $s(x) \cdot x_1 > 0$ na figura 5.6.b (ponto de sela) representa um sistema instável, já que as variáveis de fase tendem a crescer assintoticamente. O ganho do controlador neste caso é dado por f1.

Os parâmetros f1 e f2 escolhidos por tentativa e erro são -20 e 5 respectivamente e a linha de comutação é dado por c=1. Os planos de fase do sistema são mostrados na figura 5.6.

A figura 6.5.a e c mostram o sistema simulado. A influência do braço é notado pela diminuição do sinal de controle na figura 6.5.c. Nas figuras 6.5.b e d está mostrado o comportamento do sistema real. Podemos notar que o sistema apresenta um sobrepasso de posição. Isto poderia ser explicado pelo fato que o controlador de estrutura variável ser um controlador proporcional com ganho chaveado.

*Figura 6.5. Controlador por estrutura variável*



# 6.6 - CONCLUSÕES FINAIS

Procuramos neste trabalho desenvolver uma estrutura de *hardware* para o controle axial de juntas rotacionais em manipuladores robóticos e avaliar o desempenho deste para algumas estratégias de controle. Para isto descrevemos o problema de controle global de um manipulador robótico, e utilizamos uma estrutura hierárquica de controle, composto por um computador supervisor (IBM-PC) e um microcontrolador de junta (MC8031), para implementá-lo. Após determinada as atribuições dos níveis hierárquicos, partiu-se para o estudo da dinâmica de um manipulador robótico, que resultou num programa de cálculo simbólico das equações recorrentes de Newton-Euler.

Um sistema de desenvolvimento para trabalhar com o MC8031 foi construído para facilitar o projeto do controlador de junta. Para estudar o movimento das juntas robóticas foi montado um braço mecânico no servomecanismo MS150, implementando assim um manipulador de um grau de liberdade, e procedeu-se a formulação do seu modelo matemático e projeto do controlador.

As dificuldades encontradas durante o projeto do controlador foram: 1) a falta de um modelo confiável e a existência de não-linearidades no sistema; e 2) problemas com o acoplamento do braço ao servo mecanismo e seu sistema de redução de engrenagens.

As consequências das não-linearidades no sistema foram minimizadas com o projeto de uma unidade de compensação não-linear que compensou o efeito da faixa morta do servomotor MS150 e, criando assim, condições para modelar o sistema.

O motor do servomecanismo MS150 não mostrou-se suficientemente adequado para suportar um braço mecânico,

assim ficamos obrigados a manter o braço leve e o comprimento do braço pequeno. Por isto, os efeitos da força inercial e gravitacional não foram muito apreciáveis no desempenho dos controladores com o braço.

Como contribuições deste trabalho colocamos:

1) O projeto e construção de um sistema de desenvolvimento para microcontroladores KIT-31.
2) A implementação de um controlador de manipulador robótico de modo descentralizado, usando um computador supervisor para o controle de trajetória e como controlador de junta o microcontrolador MC8031.
3) Um programa de cálculo simbólico para as equações recorrentes de Newton-Euler.
4) A implementação de um compensador não-linear para o MS150.
5) O projeto de cinco algoritmos de controle em uma linguagem de alto nível e a sua realização física.

Todos os algoritmos de controle implementados apresentaram um comportamento dentro do esperado e das limitações do dispositivo. O desempenho dos controladores no futuro pode ser melhorado com o uso de sensores de maior resolução. Sugerimos neste caso o uso de sensores a base de opto-codificadores e também técnicas de acionamento mais apropriados para sistemas microprogramados como por exemplo técnicas de modulação por largura de pulso ("PWM").

Não foi objeto da pesquisa fazer um estudo comparativo para identificar o melhor controlador, mesmo porque o braço era limitado. Somente mostrou-se que todos os controladores se comportaram dentro do esperado. Cada configuração tem suas limitações e facilidades. Entretanto, sugerimos uma continuação no estudo do controlador por estrutura variável, primeiro, pela facilidade com que ele é implementado por um sistema microprogramado, e, segundo, porque infelizmente neste trabalho não foi possível explorar as propriedades que o tornam robusto à variação de

parâmetros, o que é uma propriedade desejável principalmente em sistemas robóticos.

Como continuação da pesquisa sugerimos o projeto e construção mecânica de um manipulador robótico com dois ou três graus de liberdade, e a utilização do exposto nesta dissertação para controlá-lo.

# REFERÊNCIAS BIBLIOGRÁFICAS

ALVES, João Bosco da Mota, Controle de Robô. Campinas, SP Cartgraf, 1988.

CRAIG, J. Introduction to Robotics - Mechanism and Control, 2nd Edition, Addison-Wesley,1989

FU-JUAY Chang, Position control of DC motors via variable structure system control: A chattering alleviation approach. IEEE transactions on industrial electronics. vol 37 no 6. Dec. 1990.

FU, K.S. GONZALES, R.C LEE,C.Robotics: control, sensing, vision and inteligence. Mc Graw-Hill, 1987

GUENTHER R., LIZARRALDE F., Cunha J., Estudo comparativo de estratégias de controle de manipuladores. In:9. CBA - UFES - Vitoria. pag.201.

HSIA, Steve, LASKY A., ZHENYU GUO, Robust idependant joint controller design for industrial robot manipulators, IEEE transactions on industrial electronics vol.38 no.1 Feb. 1991.

HUNG, jonh Y. Variable structure control: A survey. IEEE transactions on industrial electronics, vol 40. no. 1 Feb. 1993

KOREN,Y. Robotics for Engineers. Mc Graw-Hill, New York, 1985.

MADRID, M. e PALHARES, A. Controle de posicão e velocidade de robos manipuladores, desempenho com juntas em malha aberta e malha fechada DSCE-FEE-UNICAMP in: 8 CBA, p.506-512, Belém PA. 1990

MADRID, M. PALHARES, A. Um sistema de controle digital para movimentar manipuladores mecânicos. In:7, CBA p.588-594, São José dos Campos SP. 1988

MEDELECK A. e ZAMPANIE E.E., Desenvolvimento de um controle supervisor para um manipulador antromórfico com 6 graus de liberdade, UNICAMP in: 1 SAI 1990 CEFET PR

Modular servo system MS150, book 1. Feedback Instruments Ltd.

HSIA, T. C., LASKY, T. A., ZHENYU GUO. Robust independent joint controller design for industrial robot manipulators. In: IEEE transactions on industrial electronics, p.21-25,vol. 38,n.1,1991

PAUL, Richard P. Robot manipulators, mathematics, programming and control, MIT 1981.

SERVICE MANUAL S3 3HAB 009-25 M93A ABB ROBOTICS PRODUCTS 1993, ASEA BROWN BOVERI

SCHILLING, R. J., Fundamentals of Robotics - Analysis and Control. Prentice-Hall, New Yersey, 1990

UTKIN, Vadim I. Sliding mode control design principles and application to electrical drives. IEEE transactions on industrial electronics. p.23-35,vol 40 no. 1. feb. 1993

CASANOVA, V. H. Apostilhas do laboratóriode Controle e Servomecanismos, Universidade de Brasília; 1980.

# APÊNDICE

## Listagem dos programas

# Programa de cálculo simbólico Newton-Euler

```
{=========================================================}
{ programa para calculo simbolico das equacoes recursivas                }
{ de Newton-Euler
                }
{ programa   : NEWEU3.PAS    exp: Craigh p.202                           }
{ biblioteca : NEUBIB.PAS                                                }
{                                                                        }
{ entrada de dados : matriz de rotacao                                   }
{                    tensor de inercia                                   }
{                    geometria da estrutura                              }
{                    momentos de inercia na ponta                        }
{                    forcas na ponta                                     }
{ saida na tela : velocidades, aceleracao e torque                       }
{                                                                        }
{ autor : Rudi van Els nov. 1992       revisao : jan. 1994               }
{=========================================================}
program neweu;
uses neubib,crt;
type str5=string[5];
type vetor=array[1..3] of string;

procedure ini_vet(a,b,c:str5;var Z:vetor);
begin
 z[1]:=a;
 z[2]:=b;
 z[3]:=c;
end;

procedure ini_mat(a,b,c,d,e,f,g,h,i:str5;var m:matriz);
begin
 m[1,1]:=a;m[1,2]:=b;m[1,3]:=c;
 m[2,1]:=d;m[2,2]:=e;m[2,3]:=f;
 m[3,1]:=g;m[3,2]:=h;m[3,3]:=i;
end;

procedure pr_esc_vet(a:string;b:vetor;var c:vetor);
var i:byte;
    s:string;
begin
 for i:=1 to 3 do
  begin
   mult(a,b[i],s);
   c[i]:=s;
  end;
end;

procedure pr_mat_vet(m:matriz;a:vetor;var b:vetor);
var x1,x2,x3,x:string;
    i:byte;
begin
 for i:=1 to 3 do
 begin
  x:='';x1:='';x2:='';x3:='';
  mult(m[i,1],a[1],x1);
  mult(m[i,2],a[2],x2);
  mult(m[i,3],a[3],x3);
  soma(x1,x2,x);
  soma(x,x3,b[i]);
 end;
end;

procedure pr_vet_vet(a,b:vetor;var c:vetor);
var x1,x2:string;
begin
 mult(a[2],b[3],x1);mult(a[3],b[2],x2);subtr(x1,x2,c[1]);
 mult(a[3],b[1],x1);mult(a[1],b[3],x2);subtr(x1,x2,c[2]);
 mult(a[1],b[2],x1);mult(a[2],b[1],x2);subtr(x1,x2,c[3]);
end;

procedure som_vet(a,b:vetor;var c:vetor);
var i:byte;
begin for i:=1 to 3 do soma(a[i],b[i],c[i]); end;

procedure mostr_vet(a:vetor);
var i:byte;
begin for i:=1 to 3 do writeln('vet[',i,']= ',a[i]); end;

const grau=4;

var Z,P,Pc:array [1..grau]  of vetor;
    Rd,Ri,CI:array[0..grau-1]  of matriz;
    omg,domg,dvel,dvelc,Forc,N,fi,ni:array[0..grau-1] of vetor;
    dteta,d2tet,massa:array[1..grau] of str5;
    i:byte;
    x1,x2,x3,x4:vetor;

procedure entrada;
begin
 {   matriz rotacao  linha por linha }
 ini_mat('c1',  's1',  '0',  '-s1',  'c1',  '0',  '0',  '0',  '1', Rd[0]);
 ini_mat('0',  '0',  '0',  '0',  '0',  '0',  '0',  '0',  '1', Rd[1]);

 ini_mat('c1',  '-s1',  '0',  's1',  'c1',  '0',  '0',  '0',  '1', Ri[0]);
 ini_mat('0',  '0',  '0',  '0',  '0',  '0',  '0',  '0',  '1', Ri[1]);

 {   matriz tensor de inercia  }
 ini_mat('x','0','0','0','y','0','0','0','z',CI[1]);
 ini_mat('0','0','0','0','0','0','0','0','0',CI[2]);

 {   constante vetor z }
 ini_vet('0','0','1',Z[1]);
 ini_vet('0','0','1',Z[2]);

 {   vetor iPi+1  ponto  de rotacao em relacao a base  }
 ini_vet('0','0','0',P[1]);
 ini_vet('l1','0','0',P[2]);

 {   distancia centro de massa em relacao a  base  }
 ini_vet('l1','0','0',Pc[1]);
 ini_vet('l2','0','0',Pc[2]);

 {   simbolos velocidade aceleracao e massa }
 dteta[1]:='d_1';dteta[2]:='0';
 d2tet[1]:='d²_1';d2tet[2]:='0';
 massa[1]:='m1' ;massa[2]:='0';

 {  inicializacao das variaveis do sistema  }
 ini_vet('0','0','0',omg[0]);
 ini_vet('0','0','0',domg[0]);
 ini_vet('0','g','0',dvel[0]);

 {  momentos e forcas atuando na ponta }
 ini_vet('0','0','0',fi[3]);
 ini_vet('0','0','0',ni[3]);
end;

const maxgrau=1;

begin
 clrscr;
 entrada;

{   velocidade angular   }

 for i:=0 to maxgrau-1 do
  begin
   pr_esc_vet(dteta[i+1],Z[i+1],x1);
   pr_mat_vet(Rd[i],omg[i],x2);
   som_vet(x1,x2,omg[i+1]);
   writeln(' omega ',i+1:2);mostr_vet(omg[i+1]);
   readln;
  end;

 {   aceleracao angular   }

 for i:=0 to maxgrau-1 do
  begin
   pr_esc_vet(d2tet[i+1],Z[i+1],x1);
   pr_esc_vet(dteta[i+1],Z[i+1],x2);
   pr_vet_vet(omg[i+1],x2,x3);
   pr_mat_vet(Rd[i],x3,x2);
   pr_mat_vet(Rd[i],domg[i],x3);
   som_vet(x1,x2,x4);
   som_vet(x3,x4,domg[i+1]);
   writeln(' derivada omega ',i+1:2);mostr_vet(domg[i+1]);
   readln;
  end;
 clrscr;
 {   aceleracao linear   }

 for i:=0 to maxgrau-1 do
  begin
```

```
  pr_vet_vet(omg[i],P[i+1],x1);
  pr_vet_vet(omg[i],x1,x2);
  pr_vet_vet(domg[i],P[i+1],x3);
  som_vet(dvel[i],x2,x4);
  som_vet(x3,x4,x1);
  pr_mat_vet(Rd[i],x1,dvel[i+1]);
  writeln(' acelaracao linear ',i+1:2);mostr_vet(dvel[i+1]);
  readln;
 end;

{   aceleracao linear no centro   }

for i:=0 to maxgrau-1 do
 begin
  pr_vet_vet(omg[i+1],Pc[i+1],x1);
  pr_vet_vet(omg[i+1],x1,x2);
  pr_vet_vet(domg[i+1],Pc[i+1],x3);
  som_vet(dvel[i+1],x2,x4);
  som_vet(x3,x4,dvelc[i+1]);
  writeln(' acelaracao linear no centro ',i+1:2);mostr_vet(dvelc[i+1]);
  readln;
 end;
 clrscr;
  {   forca    }

for i:=0 to maxgrau-1 do
 begin
  pr_esc_vet(massa[i+1],dvelc[i+1],Forc[i+1]);
  writeln(' forca outward ',i+1:2);mostr_vet(Forc[i+1]);
  readln;
 end;

  {   momento de inercia outward   }

for i:=0 to maxgrau-1 do
 begin
  pr_mat_vet(CI[i+1],omg[i+1],x1);
  pr_vet_vet(omg[i+1],x1,x2);
  pr_mat_vet(CI[i+1],domg[i+1],x1);
  som_vet(x1,x2,N[i+1]);
  writeln(' momento outward ',i+1:2);mostr_vet(N[i+1]);
  readln;
 end;
clrscr;
  {   forca inward iterarion   }

for i:=maxgrau downto 1 do
 begin
  pr_mat_vet(Ri[i-1],fi[i+1],x1);
  som_vet(x1,Forc[i],fi[i]);
  writeln(' forca inward ',i:2);mostr_vet(fi[i]);
  readln;
 end;

  {   momento de inercia inward  }
for i:=maxgrau downto 1 do
 begin
  pr_mat_vet(Ri[i-1],fi[i+1],x1);
  pr_vet_vet(P[i],x1,x2);
  pr_vet_vet(Pc[i],Forc[i],x1);
  pr_mat_vet(Ri[i-1],ni[i+1],x3);
  som_vet(N[i],x3,x3);
  som_vet(x3,x1,x1);
  som_vet(x1,x2,ni[i]);
  writeln(' momento de inercia inward ',i:2);mostr_vet(ni[i]);
  readln;
 end;
clrscr;
writeln(' torque de cada junta ');
writeln;
writeln('torque [1]=');writeln(ni[1][3]);
writeln;
writeln('torque [2]=');writeln(ni[2][3]);
readln;

end.


{=========================================}
{ biblioteca de suporte para calcula simbolico das  }
{ equacoes de Newton Euler                          }
{ programa: NEUBIB.PAS                              }
{ principal : NEWEU?.PAS                            }
{                                                   }
{ rotinas implementados                             }
{     mult      multiplicacao simbolica de strings  }
{     soma      soma          simbolica de strings  }
{     subtr     subtracao     simbolica de strings  }
{     multmat   multiplica de matrices              }
{                                                   }
{ autor : Rudi van Els  nov. 1992   revisao: jan. 1994 }
{=========================================}
unit neubib;
interface
type str4=string[80];
     matriz=array[1..3,1..3] of string;

procedure mult(x,y:string;var z:string);
procedure soma(x,y:string;var z:string);
procedure subtr(x,y:string;var z:string);
procedure multmat(a,b:matriz;var c:matriz);
implementation

end.
```

# Programas de Identificação

```
/*************************************************************/
/*  program  controlador : idn001.c                          */
/*  program  PC-AT       : idn001.pas                        */
/*  descricao  : programa tese de mestrado                   */
/*               identificacao dos transitório ou regime     */
/*               entrada arquivo matlab de 255 bytes         */
/*                                                           */
/*  data       : agosto 1993                                 */
/*  revisao    : abril 94                       @ Rudi   */
/*************************************************************/


#include <incl\io51.h>
#include <incl\rot_c.c>
#include <adda_supp.c>

#define ligado    0xFF
#define desligado 0
#define TH        0xEB
#define TL   0xFF

#define Buf1_ini  0x5200
#define Buf1_fim  0x52FF
#define Buf2_ini  0x5300
#define Buf2_fim  0x53FF
#define dump_ini  0x5400

#define dump_fin  0x54FF

static unsigned char erro=0,buff_flag;
static int ref,velo,posi,buff1,buff2,buff;

extern void segundos(unsigned char);
extern void milisegu(unsigned char);

void inicializa(void)
{ /*  timer 0 clock MODO 1 interupcao  */
 output(TMOD,0x21);
 output(TH0,TH); output(TL0,TL);
 output(IE,0); output(IP,0);
 set_bit(ET0_bit); set_bit(PT0_bit);
 set_bit(IT0_bit); set_bit(TR0_bit);
 *(char *)DACend=linear(0);}

void timer_0_int(void)
{
 output(TH0,TH);
 output(TL0,TL);
 sample();
 posi=0x80-posi_atual();
 velo=0x80-velo_atual();
 *(char *)DACend=linear(0x80-read_XDATA(buff++));
 if (buff_flag==ligado)
 {
  write_XDATA(buff1++,int_byte(posi));
  write_XDATA(buff2++,int_byte(velo));
 }
 if (buff >= dump_fin) buff=dump_ini;
}

void extrn_0_int(void) {}

void main()
{
 int a;
 inicializa();
 while(1)
  { writeln("1=Reset   2=Inicia  3=PARA SISTEMA");
    writeln("4=Carrega_buffer    5=up_pos 6=up_vel ");
    switch(inchar())
    {
     case '1':clear_bit(EA_bit);
              break;
     case '2':writeln(" inicio amostragem");
              buff_flag=ligado;
              buff1=Buf1_ini;
              buff2=Buf2_ini;
              buff =dump_ini;
              set_bit(EA_bit);
              while (buff1 <= Buf1_fim) ;
              buff_flag=desligado;
              break;
     case '3':writeln("sistema parado");
              *(char *)DACend=linear(0);
              clear_bit(EA_bit);
              buff_flag=desligado;
              break;
     case '4':buff=dump_ini;
              while(buff != dump_fin) write_XDATA(buff++,getchar());
              writeln("preencheu");
              break;
     case '5':for (a=Buf1_ini;a<=Buf1_fim;a++) putchar(read_XDATA(a));
              writeln("buf1");
              break;
     case '6':for (a=Buf2_ini;a<=Buf2_fim;a++) putchar(read_XDATA(a));
                                               writeln("buf2");
                                               break;
      default:break;
    }
 }
}
```

```
{*************************************************************
 *  Program       : idn001.pas   programa para identificar o MOTOR    *
 *  controlador   : idn001.c     entrada arquivo matlab 256 bytes     *
 *                :              usando ad e da do kit31              *
 *  units         : interface serial                                  *
 *                : interface placa AD plc810                         *
 *                : interface arquivo entrada e saida MATLAB          *
 *                                                                    *
 *  Date     : 02/09/93                                              *
 *  Revisao : 6 abril 94                    @ Rudi van Els        *
 *************************************************************}

program test;

uses crt,intadda0,intser,intarqu0;

const tambuf=256;

var
   ser:tela_ser;
   darq:matarquivo;
   ddat:data_buffer;
   i,am:integer;
   x:char;
   fim:boolean;
   s:string;

begin
 fim:=false;
 ser.config(2,2400);
 repeat
  writeln(' Fim  Serial  Trigger  doWnload  Reset  Upload');
  x:=upcase(readkey);
  case x of 'F':fim:=true;
            'S':begin
                  ser.mostra;
                  writeln;
                end;
            'T':begin
                  writeln(' trigger ');
                  ser.outstring('2');
                  writeln(ser.linha);
                end;
            'W':begin
                  write('nome arquivo .mat >');readln(s);
                  if pos('.',s)=0 then s:=s+'.mat';
                  writeln(' downloading   ',s+'.mat');
                  darq.inicia;
                  darq.abre(s);
                  darq.carrega(ddat);
                  darq.fecha;
                  ser.outstring('4');
                  delay(100);
                  for i:=0 to tambuf-1 do ser.outbyte(ddat[i]);
                  writeln(ser.linha);
                end;
            'U':begin
                  write('nome arquivo .mat >');readln(s);
                  if pos('.',s)<>0 then s:=copy(s,1,pos('.',s)-1);
                  write('Pos ou Vel');
                  if upcase(readkey)='P' then ser.outstring('5')
```

```
                 else ser.outstring('6');
                ddat[0]:=ser.getbyte;
                ddat[0]:=ser.getbyte;
                for i:=0 to tambuf-1 do ddat[i]:=ser.getbyte;
                writeln(ser.linha);
                darq.inicia;
                darq.cria(s+'.mat');
                darq.grava(s,ddat,tambuf);
                darq.fecha;
               end;
            'R':begin
                ser.outstring('3');
                writeln(ser.linha);
               end;
            '0':ser.outbyte(0);
  end;
 until fim;
end.
```

# Identificação torque

```
/*************************************************************/
/*  program  controlador : idtorque.c                        */
/*  program  PC-AT       : comit                             */
/*  descricao  : programa parte do tese de mestrado          */
/*               identificacao curva torque tensao           */
/*               utilizando freio magnetico                  */
/*                                                           */
/*  data       : fevereiro 1994                              */
/*  Revisao    : marco 1994                    @ Rudi   */
/*************************************************************/

#include <incl\io51.h>
#include <incl\rot_c.c>
#include <adda_sup.c>

#define TH       0xEB
#define TL       0xFF

static int ref,velo,posi;

extern void segundos(unsigned char);
extern void milisegu(unsigned char);
extern unsigned char lo(int i);


void inicializa(void)
{
 /*  timer 0 clock MODO 1 interupcao  */
 output(TMOD,0x21);
 output(TH0,TH);
 output(TL0,TL);
 output(IE,0);
 output(IP,0);
 set_bit(ET0_bit);
 set_bit(PT0_bit);
 set_bit(IT0_bit);
 set_bit(TR0_bit);
}

void reseta(void)
{
 sample();
 posi=0x80-posi_atual();                /* leia posicao              */
 velo=0x80-velo_atual();                /* leia velocidade           */
 linear(0);
}

void timer_0_int(void)
{
 output(TH0,TH);
 output(TL0,TL);
 sample();
 posi=0x80-posi_atual();                /* leia posicao              */
 velo=0x80-velo_atual();                /* leia velocidade           */
 *(char *)DACend=linear(ref);
}

void extrn_0_int(void)
{}

void main()
{
 char c,s[10];
 int i;
 inicializa();
 reseta();
 while(1)
  { writeln("1=Reset   3=status ");
    writeln("4=degrau  E=ensaio");
    switch(inchar())
    {
     case '1':reseta();
              writeln("<");
              clear_bit(EA_bit);
              break;
     case '3':write("referencia ");outword(ref);putchar(CR);
              write("velocidade ");outword(velo);putchar(CR);
              break;
     case '4':clear_bit(EA_bit);
              write("valor degau ");readln(s);ref=toint(s);
              writeln("<");
              c=getchar();
              set_bit(EA_bit);
              break;
     case '+':ref++;
              write("referencia ");outword(ref);putchar(CR);
              write("velocidade ");outword(velo);putchar(CR);
              break;
     case '-':ref--;
              write("referencia ");outword(ref);putchar(CR);
              write("velocidade ");outword(velo);putchar(CR);
              break;
     case 'E':writeln("Ensaio +");
              for (i=0;i<=127;i++)
              { ref=i;segundos(3);outword(ref);write("  ");
                outword(velo);writeln("  ");
               }
              writeln(" ");
              writeln("Ensaio -");
              for (i=127;i>=-127;i--)
              { ref=i;segundos(3);outword(ref);write("  ");
                outword(velo);writeln("  ");
               }
              writeln(" ");
              writeln("Ensaio 0");
              for (i=-127;i<=0;i++)
              { ref=i;segundos(3);outword(ref);write("  ");
                outword(velo);writeln("  ");
               }
              writeln(" ");
              writeln("Fim");
              break;
     default:break;
    }
  }
}
```

# Programas de controle
# Rotinas de suporte

```
/*================================*/
/*    programa suporte ADC e CDA   */
/*    arquivo : adda_sup.c         */
/*    dez. 93                      */
/*================================*/


#define VPU_sel   P1_5_bit
#define VPU_0     P1_6_bit

#define MUX_A     P1_0_bit
#define MUX_B     P1_1_bit
#define MUX_sel   P1_2_bit

#define DACend    0xA008
#define ADCend    0xA00A
#define SOC       0xA009
#define EOC       P1_7_bit

unsigned char linear(int a)
{
 if (a>0) { clear_bit(VPU_sel); clear_bit(VPU_0); } else
  if (a<0) { set_bit(VPU_sel); clear_bit(VPU_0); } else
   set_bit(VPU_0);
 if (a>128) return(0xFF); else
  if (a<-128) return(0x00); else
  return(a+0x80);
}

void sample(void)
{
 clear_bit(MUX_A);clear_bit(MUX_B);
 set_bit(MUX_A);set_bit(MUX_B);
}

unsigned char posi_atual(void)
{
 set_bit(MUX_sel);
 *(char *)SOC=0xFF;                  /* start of conversion posicao */
 *(char *)SOC=0;
 while(read_bit(EOC)==0);            /* espera end of conversion */
 return(read_XDATA(ADCend));
}

unsigned char velo_atual(void)
{
  clear_bit(MUX_sel);
  *(char *)SOC=0xFF;
  *(char *)SOC=0;
  while(read_bit(EOC)==0);
  return(read_XDATA(ADCend));
}

unsigned char int_byte(int i)
/* retorna um byte do word com saturacao  */
{ if (i>128) return(0xFF); else if (i<-128) return(0); else return(i + 0x80);}
```

# Controlador PD tipo A

```
/*************************************************************/
/*  program  controlador : pd1impl.c                         */
/*  program  PC-AT       : super.pas                         */
/*  descricao  : programa parte do tese de mestrado          */
/*               realimentacao de velocidade e posicao       */
/*               controlador PD 1. implemnetacao             */
/*                                                           */
/*  data       : janeiro  1994                               */
/*  revisao    : 31 marco 1994                  @ Rudi   */
/*************************************************************/
#include <incl\io51.h>
#include <incl\rot_c.c>
#include <adda_sup.c>

#define TH       0xEB
#define TL       0xFF
#define TRAC     0x20
#define tam      0xFF

extern int getint(void);
static int KP,KV,ref,velo,posi,trajec,acio;
static int baffer1[tam+2],baffer2[tam+2],baffer3[tam+2],baffer4[tam+2];

void inicializa(void)
{
 /*  timer 0 clock MODO 1 interupcao  */
 output(TMOD,0x21);
 output(TH0,TH);
 output(TL0,TL);
 output(IE,0);
 output(IP,0);
 set_bit(ET0_bit); set_bit(PT0_bit);
 set_bit(IT0_bit); set_bit(TR0_bit);
}

void reseta(void)
{
 sample();
 posi=0x80-posi_atual();                 /* leia posicao                */
 velo=0x80-velo_atual();                 /* leia velocidade             */
 *(char *)DACend=linear(0);
 clear_bit(TRAC);
}

void timer_0_int(void)
{
 output(TH0,TH);
 output(TL0,TL);
 sample();
 posi=0x80-posi_atual();                    /* leia posicao       */
 velo=0x80-velo_atual();                    /* leia velocidade    */
 acio=(ref-posi)*KP-velo*KV;
 *(char *)DACend=linear(acio);           /* (ref-posi)*KP-velo*KV); */
 if (read_bit(TRAC))
  { ref=baffer1[trajec];
    baffer2[trajec]=posi;
    baffer3[trajec]=velo;
    baffer4[trajec]=acio;
    trajec++;
    if (trajec==tam) clear_bit(TRAC);
  }
}

void extrn_0_int(void) {}

void main()
{
 char c,s[10]; int a;
 inicializa();
 reseta();
 while(1)
  { writeln("1=Reset   2=Parametros   3=status ");
    writeln("4=degrau  5=download     6=tracking  7=Upload");
    switch(inchar())
    {
     case '1':clear_bit(EA_bit);
              reseta();
              writeln("<");
              break;
     case '2':write("KP = ");readln(s);KP=toint(s);
              write("KV = ");readln(s);KV=-toint(s);
              break;
     case '3':reseta();
              outword(posi);putchar(CR);
              outword(velo);putchar(CR);
              break;
     case '4':clear_bit(EA_bit);
              write("valor degau ");readln(s);ref=toint(s);
              for (a=0;a<=tam;a++) baffer1[a]=ref;
              break;
     case '5':clear_bit(EA_bit);
              writeln("PgDwn");
              for (a=0;a<=tam;a++) baffer1[a]=getint();
              writeln(" OK ");
              break;
     case '6':set_bit(TRAC);
              trajec=0;
              ref=baffer1[0];
              writeln("<");
              c=getchar();
              set_bit(EA_bit);
              break;
     case '7':clear_bit(EA_bit);
              writeln("PgUp");
              c=getchar();
              for (a=0;a<=tam;a++) {outword(baffer2[a]);putchar(CR);c=getchar();}
              c=getchar();
              for (a=0;a<=tam;a++) {outword(baffer3[a]);putchar(CR);c=getchar();}
              c=getchar();
              for (a=0;a<=tam;a++) {outword(baffer4[a]);putchar(CR);c=getchar();}
              break;
      default:break;
    }
  }
}
```

# Controlador PD tipo B

```
/*************************************************************/
/*  program  controlador : pdtipo2.c                         */
/*  program  PC-AT       : super.pas                         */
/*  descricao  : programa parte do tese de mestrado          */
/*               realimentacao e posicao                     */
/*               controlador PD 2. implementacao  q0 e q1    */
/*                                                           */
/*  data       : janeiro  1994                 @ Rudi   */
/*  revisao    : 31 marco 1994                             */
/*************************************************************/

#include <incl\io51.h>
#include <incl\rot_c.c>
#include <adda_sup.c>


#define TH       0xEB
#define TL       0xFF
#define TRAC     0x20

#define tam      0xFF

static unsigned char trac_flag;
static int q0,q1,ref,velo,posi,trajec,erro,erro_1,acio;
static int baffer1[tam+2],baffer2[tam+2],baffer3[tam+2],baffer4[tam+2];

extern void  putint(int);
extern int  getint(void);

void inicializa(void)
{
 /*  timer 0 clock MODO 1 interupcao  */
 output(TMOD,0x21);
 output(TH0,TH);
 output(TL0,TL);
 output(IE,0);
 output(IP,0);
 set_bit(ET0_bit);
 set_bit(PT0_bit);
 set_bit(IT0_bit);
 set_bit(TR0_bit);
}

void reseta(void)
{
 sample();
 posi=0x80-posi_atual();                /* leia posicao                */
 *(char *)DACend=linear(0);
 clear_bit(TRAC);
}

void timer_0_int(void)
{
 output(TH0,TH);
 output(TL0,TL);
 sample();
 posi=0x80-posi_atual();                /* leia posicao                */
 velo=0x80-velo_atual();
 erro=ref-posi;
 acio=q0*erro-q1*erro_1;
 *(char *)DACend=linear(acio);
 if (read_bit(TRAC))
  { ref=baffer1[trajec];
    baffer2[trajec]=posi;
    baffer3[trajec]=velo;
    baffer4[trajec]=acio;
    trajec++;
    if (trajec==tam) clear_bit(TRAC);
  }
 erro_1=erro;
}

void extrn_0_int(void)
{}

void main()
{
 char c,s[10];
 int a;
 inicializa();
 reseta();
 while(1)
 { writeln("1=Reset   2=Parametros   3=status ");
  writeln("4=degrau  5=Download  6=tracking  7=Upload");
  switch(inchar())
  {
   case '1':clear_bit(EA_bit);
            reseta();
            writeln("<");
            break;
   case '2':write("q0 = ");readln(s);q0=toint(s);
              write("hex=");outword(q0);
              writeln;
            write("q1 = ");readln(s);q1=toint(s);write("hex=");outword(q1);writeln;
            break;
   case '3':reseta();
            outword(posi);putchar(CR);
            outword(erro);putchar(CR);
            break;
   case '4':clear_bit(EA_bit);
            write("valor degau ");readln(s);ref=toint(s);
            for (a=0;a<=tam;a++) baffer1[a]=ref;
            break;
   case '5':clear_bit(EA_bit);
            writeln("PgDwn");
            for (a=0;a<=tam;a++) baffer1[a]=getint();
            writeln(" OK ");
            break;
   case '6':set_bit(TRAC);
            trajec=0;
            ref=baffer1[0];
            writeln("<");
            c=getchar();
            set_bit(EA_bit);
            break;
   case '7':clear_bit(EA_bit);
            writeln("PgUp");
            c=getchar();
            for (a=0;a<=tam;a++) {outword(baffer2[a]);putchar(CR);c=getchar();}
            c=getchar();
            for (a=0;a<=tam;a++) {outword(baffer3[a]);putchar(CR);c=getchar();}
            c=getchar();
            for (a=0;a<=tam;a++) {outword(baffer4[a]);putchar(CR);c=getchar();}
            break;
    default:break;
   }
 }
}
```

# Controlador PID

```
/*************************************************************/
/*  program  controlador : pidnrom1.c                        */
/*  program  PC-AT       : super.pas                         */
/*  descricao  : programa parte do tese de mestrado          */
/*                                                           */
/*               controlador PID normal                      */
/*                                                           */
/*  data       : marco 1994                                  */
/*  revisao    : julho 1994                    @ Rudi   */
/*************************************************************/

#include <incl\io51.h>
#include <incl\rot_c.c>
#include <adda_sup.c>

#define TH       0xEB
#define TL       0xFF
#define TRAC     0x20
#define tam      0xFF

extern int getint(void);

static int   q0,q1,q2,ref,velo,posi,trajec,u,u1,erro,erro1,erro2;
static int   baffer1[tam+2],baffer2[tam+2],baffer3[tam+2],baffer4[tam+2];

void inicializa(void)
{
 /*  timer 0 clock MODO 1 interupcao  */
 output(TMOD,0x21);
 output(TH0,TH);
 output(TL0,TL);
 output(IE,0);
 output(IP,0);
 set_bit(ET0_bit);
 set_bit(PT0_bit);
 set_bit(IT0_bit);
 set_bit(TR0_bit);
}

void reseta(void)
{
 sample();
 posi=0x80-posi_atual();                /* leia posicao              */
 velo=0x80-velo_atual();                /* leia velocidade           */
 *(char *)DACend=linear(0);
 clear_bit(TRAC);
}

void timer_0_int(void)
{
 output(TH0,TH);
 output(TL0,TL);
 sample();
 posi=0x80-posi_atual();                /* leia posicao              */
 velo=0x80-velo_atual();                /* leia velocidade           */
 erro=ref-posi;
 u=u1+q0*erro+q1*erro1+q2*erro2;
 *(char *)DACend=linear(u);
 if (read_bit(TRAC))
  { ref=baffer1[trajec];
    baffer2[trajec]=posi;
    baffer3[trajec]=velo;
    baffer4[trajec]=u;
    trajec++;
    if (trajec==tam) clear_bit(TRAC);
  }
 u1=u;
 erro2=erro1;
 erro1=erro;
}

void extrn_0_int(void)
{ }

void main()
{
 char c,s[10];
 int a;
 inicializa();
 reseta();
 while(1)
  { writeln("1=Reset   2=Parametros   3=status ");
    writeln("4=degrau  5=tracking     6=tracking ");
    switch(inchar())
    {
     case '1':clear_bit(EA_bit);
              reseta();
              writeln("<");
              break;
     case '2':write("q0 = ");readln(s);q0=toint(s);
              write("q1 = ");readln(s);q1=toint(s);
              write("q2 = ");readln(s);q2=toint(s);
              erro1=0;erro2=0;u1=0;
              break;
     case '3':reseta();
              outword(posi);putchar(CR);
              outword(velo);putchar(CR);
              break;
     case '4':clear_bit(EA_bit);
              write("valor degau ");readln(s);ref=toint(s);
              for (a=0;a<=tam;a++) baffer1[a]=ref;
              erro1=0;erro2=0;u1=0;
              break;
     case '5':clear_bit(EA_bit);
              writeln("PgDwn");
              for (a=0;a<=tam;a++) baffer1[a]=getint();
              writeln("OK");
              break;
     case '6':set_bit(TRAC);
              erro1=0;erro2=0;u1=0;
              trajec=0;
              ref=baffer1[0];
              writeln("<");
              c=getchar();
              set_bit(EA_bit);
              break;
     case '7':clear_bit(EA_bit);
              writeln("PgUp");
              c=getchar();
              for (a=0;a<=tam;a++) {outword(baffer2[a]);putchar(CR);c=getchar();}
              c=getchar();
              for (a=0;a<=tam;a++) {outword(baffer3[a]);putchar(CR);c=getchar();}
              c=getchar();
              for (a=0;a<=tam;a++) {outword(baffer4[a]);putchar(CR);c=getchar();}
              break;
      default:break;
    }
  }
}
```

# Controlador PD torque computado

```
/*************************************************************/
/*  program  controlador : pdtq.c                            */
/*  program  PC-AT       : super.pas                         */
/*  descricao  : programa parte do tese de mestrado          */
/*                                                           */
/*               controlador PDTQ                            */
/*                                                           */
/*  data       : marco 1994                                  */
/*  revisao    : julho 1994                    @ Rudi        */
/*************************************************************/

#include <incl\io51.h>
#include <incl\rot_c.c>
#include <adda_sup.c>

#define TH       0xEB
#define TL       0xFF
#define TRAC     0x20
#define tam      0xFF

extern int getint(void);

static int   KP,KV,ref,velo,posi,trajec,u,torque;
static int   baffer1[tam+2],baffer2[tam+2],baffer3[tam+2],baffer4[tam+2];
extern float  seno[128];

void reseta(void)
{
 sample();
 posi=0x80-posi_atual();                /* leia posicao              */
 velo=0x80-velo_atual();                /* leia velocidade           */
 *(char *)DACend=linear(0);
 clear_bit(TRAC);
}

void timer_0_int(void)
{
 output(TH0,TH);
 output(TL0,TL);
 sample();
 posi=0x80-posi_atual();                /* leia posicao              */
 velo=0x80-velo_atual();                /* leia velocidade           */
 torque=(int)(seno[posi<0 ? -posi:posi]);
 u=(ref-posi)*KP-velo*KV-torque;
 *(char *)DACend=linear(u);
 if (read_bit(TRAC))
  { ref=baffer1[trajec];
    baffer2[trajec]=posi;
    baffer3[trajec]=velo;
    baffer4[trajec]=u;
    trajec++;
    if (trajec==tam) clear_bit(TRAC);
  }
}

void extrn_0_int(void)
{}

void main()
{
 char c,s[10];
 int a;
 output(TMOD,0x21);
 output(TH0,TH);
 output(TL0,TL);
 output(IE,0);
 output(IP,0);
 set_bit(ET0_bit);
 set_bit(PT0_bit);
 set_bit(IT0_bit);
 set_bit(TR0_bit);
 reseta();
 while(1)
  { writeln("1234567");
    switch(inchar())
     {
     case '1':clear_bit(EA_bit);
              reseta();
              writeln("<");
              break;
     case '2':write("Kp=");readln(s);KP=toint(s);
              write("Kv=");readln(s);KV=-toint(s);
              break;
     case '3':reseta();
              outword(posi);putchar(CR);
              outword(velo);putchar(CR);
              break;
     case '4':clear_bit(EA_bit);
              write("dergau");readln(s);ref=toint(s);
              for (a=0;a<=tam;a++) baffer1[a]=ref;
              break;
     case '5':clear_bit(EA_bit);
              writeln("PgDwn");

              for (a=0;a<=tam;a++) baffer1[a]=getint();
              writeln("OK");
              break;
     case '6':set_bit(TRAC);
              trajec=0;
              ref=baffer1[0];
              writeln("<");
              c=getchar();
              set_bit(EA_bit);
              break;
     case '7':clear_bit(EA_bit);
              writeln("PgUp");
              c=getchar();
              for (a=0;a<=tam;a++) {outword(baffer2[a]);putchar(CR);c=getchar();}
              c=getchar();
              for (a=0;a<=tam;a++) {outword(baffer3[a]);putchar(CR);c=getchar();}
              c=getchar();
              for (a=0;a<=tam;a++) {outword(baffer4[a]);putchar(CR);c=getchar();}
              break;
      default:break;
     }
  }
}
```

# Controlador Estrutura Variável

```
/*************************************************************/
/*  program  controlador : vsscont.c                         */
/*  program  PC-AT       : super.pas                         */
/*  descricao  : programa parte do tese de mestrado          */
/*               controle por estrutura variavel REGULADOR   */
/*                                                           */
/*  data       : janeiro  1994                               */
/*  revisao    : 13 julho 1994                  @ Rudi   */
/*************************************************************/

#include <incl\io51.h>
#include <incl\rot_c.c>
#include <adda_sup.c>

#define ligado    0xFF
#define desligado 0

/* #define Buf1_ini  0x5200  */
/* #define Buf1_fim  0x52FF  */

extern int getint(void);

#define TH        0xEB
#define TL        0xFF
#define TRAC      0x20
#define tam       0xFF

static int fi1,fi2,ccc,u,dref,ref,ref_1,velo,posi,trajec,x1,x2;
static int baffer1[tam+2],baffer2[tam+2],baffer3[tam+2],baffer4[tam+2];


void inicializa(void)
{
 /*  timer 0 clock MODO 1 interupcao  */
 output(TMOD,0x21);
 output(TH0,TH);
 output(TL0,TL);
 output(IE,0);
 output(IP,0);
 set_bit(ET0_bit); set_bit(PT0_bit);
 set_bit(IT0_bit); set_bit(TR0_bit);
}

void reseta(void)
{
 sample();
 posi=0x80-posi_atual();                /* leia posicao               */
 velo=0x80-velo_atual();                /* leia velocidade             */
 *(char *)DACend=linear(0);
 clear_bit(TRAC);
}

void timer_0_int(void)
{
 output(TH0,TH);
 output(TL0,TL);
 sample();
 posi=0x80-posi_atual();                /* leia posicao               */
 velo=(0x80-velo_atual())*-7;           /* leia velocidade             */
 dref=(ref-ref_1)*100;
 x1=posi-ref;
 x2=velo-dref;
 if (((ccc*x1+x2) > 0)  ^ (x1>0))
  u=fi2*x1; else u=fi1*x1;
 *(char *)DACend=linear(u);
 ref_1=ref;
 if (read_bit(TRAC))
  {
    baffer2[trajec]=posi;
    ref=baffer1[trajec];
    baffer3[trajec]=velo;
    baffer4[trajec]=u;
    trajec++;
    if (trajec==tam) clear_bit(TRAC);
  }

}

void extrn_0_int(void)
{}
```

```
void main()
{
 char c,s[10];
 int a;
 inicializa();
 reseta();
 while(1)
  { writeln("1=Reset   2=Parametros   3=status ");
    writeln("4=degrau  5=download    6=tracking");
    switch(inchar())
    {
     case '1':clear_bit(EA_bit);
              reseta();
              writeln("<");
              break;
     case '2':write("fi1  = ");readln(s);fi1=toint(s);
              write("fi2  = ");readln(s);fi2=toint(s);
              write("c    = ");readln(s);ccc=toint(s);
              break;
     case '3':reseta();
              outword(posi);putchar(CR);
              outword(velo);putchar(CR);
              break;
     case '4':clear_bit(EA_bit);
              write("valor degau ");readln(s);ref=toint(s);
              for (a=0;a<=tam;a++) baffer1[a]=ref;
              break;
     case '5':clear_bit(EA_bit);
              writeln("PgDwn");
              for (a=0;a<=tam;a++) baffer1[a]=getint();
              writeln(" OK");
              break;
     case '6':set_bit(TRAC);
              trajec=0;
              ref=baffer1[0];
              writeln("<");
              c=getchar();
              set_bit(EA_bit);
              break;
     case '7':clear_bit(EA_bit);
              writeln("PgUp");
              c=getchar();
              for (a=0;a<=tam;a++) {outword(baffer2[a]);putchar(CR);c=getchar();}
              c=getchar();
              for (a=0;a<=tam;a++) {outword(baffer3[a]);putchar(CR);c=getchar();}
              c=getchar();
              for (a=0;a<=tam;a++) {outword(baffer4[a]);putchar(CR);c=getchar();}
              break;
      default:break;
    }
  }
}
```

•

# Programa supervisor no IBM-PC

```
{************************************************************
 *  Program       : super.pas   programa supervisor do MOTOR       *
 *  controlador   : pd e pid   vss  pdtq                          *
 *  units         : interface serial                              *
 *                  : interface arquivo entrada e saida MATLAB          *
 *                                                                *
 *  Date     : 16/10/93                    @ Rudi van Els         *
 *  Revisao  : 10/06/94                                           *
 ************************************************************}

program super;


uses crt,tipos,intser,intarqu2;

var
   ser:tela_ser;
   darq:matarquivo;
   ddat:data_buffer;
   i,am,x,code:integer;
   s,si,s1:string;

const tambuff=$FF;

procedure gravacanal;
begin
 write('nome arquivo .mat >');readln(s);
 if pos('.',s)<>0 then s:=copy(s,1,pos('.',s)-1);
 darq.inicia;
 darq.cria(s+'.mat');
 darq.grava(s,ddat,$FF);
 darq.fecha;
end;

begin
 ser.config(2,2400);
 ser.defina(1,1,81,23);
 repeat
   ser.mostra;
   x:=ser.funcao;
   case x of 81:begin   { pgdwn }
                write('nome arquivo .mat >');readln(s);
                if pos('.',s)=0 then s:=s+'.mat';
                writeln(' downloading   ',s);
                darq.inicia;
                darq.abre(s);
                darq.carrega(ddat);
                darq.fecha;
                for i:=0 to $FF do begin ser.outword(ddat[i]); delay(10); end;
                writeln(ser.linha);
               end;
             73:begin          { PGup}
                ser.outbyte(0);
                for  i:=0  to  $FF  do  begin  s1:=ser.linha;  val('$'+s1,ddat[i],code);
ser.outbyte(0); end;
                writeln(' canal 1 ');
                gravacanal;
                ser.outbyte(0);
                for  i:=0  to  $FF  do  begin  s1:=ser.linha;  val('$'+s1,ddat[i],code);
ser.outbyte(0); end;
                writeln(' canal 2 ');
                gravacanal;
                ser.outbyte(0);
                for  i:=0  to  $FF  do  begin  s1:=ser.linha;  val('$'+s1,ddat[i],code);
ser.outbyte(0); end;
                writeln(' canal 3 ');
                gravacanal;
               end;
             82:ser.outbyte(2);  { insert }
   end;
 until x=0;
end.
```

# Programas de simulação

```
%==============================
% arquivo dados.m  dados para simulacao
% da  resposta transitorio  simulacao no
% espaco de estados   @Rudi 2/94
%
%  gpot  = ganho potenciometro
%  gtaco = ganho tacometro
%  nx    = reducao engrenagem
%  dac   = conversor DA
%  adc   = conversor AD
%  a     = tao
%  Kt    = constante de torque
%  n     = numero de amostras
%
% x1 = posicao digital
% x2 = velocidade digital
%  ganho em malha aberta motor = 0.75
%  constante de tempo motor    = 0.3
%
% revisao:   06 abril 1994
%==============================
a=1/0.3;
nx=1/30;
adc=32;
gpot=adc*1.27;
gtaco=adc*0.006;
dac=0.003125;
Ka=0.75*a/(gtaco*dac);
Kt=1.65e3;
n=255;
%  torque externo : caracteristicas do braco
ml2=0.050*.30^2;
mlg=.050*.30*9.8;
%  equacao de estado sistema
A=[0  gpot*nx/gtaco;     0            -a];
B=[0    dac*Ka*gtaco]';
C=[1    0];
[AA,BB]=c2d(A,B,0.01);
```

| | |
|---|---|
| PDTIPO1.M | simulação PD tipo A |
| PDTIPO2.M | simulação PD tipo B |
| PID.M | simulação PID |
| PDTQ.M | simulação PD torque computado |
| VSSCONT.M | simulação Estrutura Variavel |
| VSSFASE.M | plano de fase |
| VSSFASE1.M | modo deslizante |