UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
DEPARTAMENTO DE PROJETO MECÂNICO

Implementação de um Supervisor de Controle para Robôs Industriais

Carlos Henrique Dias

Orientador : Prof. Dr. João Maurício Rosário

Dissertação apresentada à Faculdade de Engenharia Mecânica da Universidade Estadual de Campinas, como parte dos requisitos exigidos para obtenção do título de mestre em engenharia mecânica.

Campinas - SP
junho de 1993

UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA MECÂNICA

ESTE EXEMPLAR CORRESPONDE A REDAÇÃO FINAL DA TESE DEFENDIDA POR CARLOS HENRIQUE DIAS E APROVADA PELA COMISSÃO JULGADORA EM 25 / 06 / 93.

ORIENTADOR

Tese de : Mestrado

Título da Tese: Implementação Experimental de um Supervisor de Controle para Robôs Industriais

Autor: Carlos Henrique Dias

Orientador: Prof. Doutor João Maurício Rosário

Aprovado por:

Prof. Dr. João Maurício Rosário, Presidente

Prof. Dr. Edson Moschim

Prof. Dr. Paulo Roberto Gardel Kurka

Campinas, 25 de junho de 1993.

AGRADECIMENTOS

O árduo porém gratificante desafio de se concluir um trabalho de Mestrado é resultado do empenho de muitas pessoas, que, em tempos e de maneiras diferentes, contribuíram para que os objetivos fossem alcançados. Citar o nome de todos que estiveram juntos nesta jornada seria uma tarefa difícil, mas gostaria de destacar dentre estas pessoas, aquelas que se associaram ao trabalho de um modo mais intenso.

Assim sendo, faço uma homenagem e agradecimento especiais ao amigo e orientador, o professor Doutor João Maurício Rosário, por sua dedicação e profissionalismo, sempre disposto, ensinando e animando mesmo nas situações mais adversas.

Também ao amigo Benedito Luís Fayan, por sua dedicação e amizade, presentes em cada uma das etapas deste trabalho.

Ao amigo Antônio Paulo Ismael, por sua inestimável ajuda em importantes testes.

Ao CPqD - Telebrás, pelos recursos laboratoriais e, principalmente, pelas pessoas que me apoiaram e contribuíram para a conclusão deste trabalho.

Ao pessoal do GEPROM, pelo prestativo auxílio e constante interesse em ajudar.

A Márcio Batista, um incansável colaborador em todas as etapas.

À Fapesp que através do financiamento, sempre acreditou no sucesso deste projeto.

DEDICO ESTA TESE PARA:

MINHA FAMÍLIA E MEUS AMIGOS

Pela amizade, carinho e ajuda.

MINHA ESPOSA - LÉDINA

Pela amizade, compreensão, companheirismo e carinho.

"Assim diz o SENHOR: Não se glorie o sábio na sua sabedoria, nem o forte na sua força, nem o rico nas suas riquezas; mas o que se gloriar, glorie-se nisto: em me conhecer e saber que eu sou o SENHOR, e faço misericórdia, juízo e justiça na terra; porque destas coisas me agrado, diz o SENHOR". (Jeremias 9:23,24).

RESUMO

O objetivo principal deste trabalho é o de fazer uma análise prática do desempenho de uma arquitetura distribuída baseada na técnica ATGS, aplicada à supervisão e controle de robôs industriais. Inicialmente é feita uma discussão a respeito de arquiteturas de controladores, sendo feita uma comparação entre arquiteturas centralizadas e distribuídas.

Em seguida, a partir de uma arquitetura distribuída, é feita uma descrição de uma solução completa para um supervisor de controle, com ênfase ao controle de juntas, para o qual são apresentadas duas soluções de implementação.

São descritas os módulos *hardware* e *software* que foram implementados, assim como os programas aplicativos utilizados no desenvolvimento do protótipo. Para as montagens experimentais são mostrados o ambiente e procedimento de testes juntamente com os resultados e comentários.

ÍNDICE

INTRODUÇÃO

A utilização de robôs industriais teve seu início por volta dos anos 50, tendo experimentado desde então uma grande evolução tecnológica. Suas principais aplicações estão no auxílio ao trabalho do homem em tarefas repetitivas, perigosas, insalubres e, aliado à informática, na melhoria da produtividade e qualidade dos produtos e serviços.

Dentre estas aplicações, algumas têm se destacado por propiciar ao homem a realização de atividades em ambientes totalmente desfavoráveis, tais como no fundo do mar, onde tarefas de apoio à prospecção de petróleo em plataformas submarinas passaram a ser feitas por manipuladores e robôs industriais adaptados para este uso.

O fundo do mar caracteriza-se por apresentar uma alta pressão associada às correntes oceânicas o que traduz um ambiente com grande variação de parâmetros. Isso faz com que o sistema utilizado no controle desses manipuladores tenha que adaptar-se a essas variações.

Os controladores comerciais, normalmente utilizados no controle dos manipuladores, oferecem poucas possibilidades de alteração de seus modelos e algoritmos de modo a facilitar o processo de adaptação a um meio tão adverso.

Esta realidade levou ao estabelecimento de um programa de cooperação científica na área de Robótica entre o Instituto Tecnológico de Geestacht - GKSS da República Federal da Alemanha, CENPES-PETROBRAS e UNICAMP , no sentido de desenvolver metodologias para intervenções submarinas automatizadas em águas profundas. Dentro desse programa de cooperação foram cedidos à UNICAMP o robô industrial MANUTEC r3, fabricado pela Siemens, e o manipulador submarino KRAFT-GRIPS.

Neste programa foram estabelecidas as seguintes linhas de pesquisa:

i) Utilização de um Manipulador Mecânico para intervenções em águas profundas a partir de um controle à distância (telemanipuladores);

ii) Adaptação do *hardware* de Controle - Supervisor de Controle;

iii) Elaboração de um *software* de visualização do robô no seu ambiente de trabalho a partir do modelo geométrico do robô e meio ambiente;

iv) Confecção de um *software* completo de Geração Autonôma de trajetórias;

v) Estudo do comportamento dinâmico de um manipulador no fundo do mar, e elaboração de um *software* completo de geração autonôma de trajetórias para o robô.

Dentre estas linhas de pesquisa, as atividades ligadas à adaptação do hardware de controle constituíram-se nos grandes motivadores para este trabalho de tese, onde optou-se pelo desenvolvimento de um supervisor de controle com as características necessárias à operação sob condições adversas.

Este trabalho constitui-se no complemento ao trabalho de *Fayan* [1], e seus principais objetivos foram os seguintes:

- Avaliação prática do desempenho de uma arquitetura distribuída e hierarquizada para controle de robôs;

- Implementação de um controlador baseado nessa arquitetura. Esta implementação é orientada à avaliação de desempenho, sem o compromisso inicial de chegar-se a um produto propriamente;

- Montagem de uma plataforma para desenvolvimento de outros controladores.

Para este supervisor foi adotada uma arquitetura distribuída e hierarquizada, baseada na técnica ATGS (Autonomous Trajectory Generating Servomechanism) que hierarquiza o controle de movimentos de robôs manipuladores em dois níveis.

É definido um nível responsável pelo controle dos servomecanismos das juntas e outro, hierarquicamente mais elevado, responsável pelo controle e gerenciamento das juntas do manipulador.

Para abordar os principais aspectos que envolvem uma arquitetura distribuída para controladores de robôs, a partir dos resultados obtidos em montagens experimentais de um protótipo, este trabalho foi estruturado em quatro capítulos.

No Capítulo 1 é feita inicialmente uma discussão a respeito de arquiteturas centralizadas e distribuídas para sistemas de controle. Em seguida, descreve-se a arquitetura hierarquizada, destacando-se os aspectos relevantes da solução adotada.

No Capítulo 2 são descritas as implementações *hardware* e *software* para a referida arquitetura.

O Capítulo 3 conceitua o controle de juntas, enfatizando o tipo de controle adotado e apresentando duas soluções para sua implementação no sistema.

No Capítulo 4 são apresentados os ambientes experimentais utilizados para a obtenção dos resultados, sendo feita também a descrição do robô empregado nas montagens e dos mecanismos utilizados na aquisição e tratamento dos dados. Para cada experimento são apresentados os resultados obtidos seguidos de comentários que enfocam as possíveis implicações no comportamento dessa arquitetura.

Finalmente são apresentados os comentários gerais, as perspectivas e as conclusões desse trabalho.

CAPÍTULO 1

ARQUITETURA DE SUPERVISÃO E CONTROLE DE UM ROBÔ INDUSTRIAL

1.1. INTRODUÇÃO

Este capítulo descreve uma arquitetura hierarquizada a partir da qual ocorre a implementação de uma estrutura de controle, objeto central desta tese. Inicialmente é feita uma conceituação sobre robôs, destacando-se suas partes constituintes, com ênfase às diferentes estruturas de supervisor de controle.

Em seguida, a descrição da arquitetura é feita de maneira a apresentá-la numa abordagem *top-down* através de seus aspectos funcionais, com ênfase em algumas peculiaridades de implementação.

1.2. CARACTERÍSTICAS GERAIS DE UM ROBÔ

1.2.1. Definições básicas

São encontradas na literatura diversas definições para um robô, dentre as quais destacamos:

a) Definição do ponto de vista da concepção [2]:

Um robô manipulador é uma estrutura mecânica constituída de um conjunto de corpos ou elementos (*links*), interligados por meio de conexões denominadas juntas, com o objetivo de sustentar, posicionar e orientar um corpo denominado elemento terminal ou efetuador.

b) Definição do ponto de vista funcional [2]:

Um robô é um manipulador multifuncional reprogramável com controle de posição automático, tendo vários eixos e sendo capaz de manipular materiais, peças, ferramentas ou dispositivos especializados através de operações variáveis programadas a fim de desempenhar uma variedade de tarefas.

1.2.2- Principais componentes

Um robô é constituído de quatro componentes principais :

a) Manipulador: É a própria estrutura mecânica formada pelos elementos interligados por meio de juntas. Cada junta define um eixo ao longo do qual o elemento ligado a ela se movimenta (movimento de rotação ou translação) e um grau de liberdade, de modo que o número de graus de liberdade seja igual ao número de juntas.

b) Sensores: Os sensores têm por finalidade informar as condições do manipulador, obtendo informações sobre a aceleração, velocidade e posição das juntas e permitindo o controle adequado da estrutura mecânica. Os sensores podem ser visuais ou não visuais.

c) Controlador: O controlador tem a função de assegurar um desempenho rápido e preciso do manipulador na execução de suas tarefas, em qualquer condição de operação. A complexidade do controle dos movimentos e a necessidade de cálculos exaustivos para a determinação precisa da trajetória a ser descrita fizeram com que os controladores passassem a utilizar microprocessadores em suas concepções, organizados em diversos tipos de arquiteturas.

d) Elemento de potência: A função básica do elemento de potência é a de converter e fornecer energia para o acionamento do manipulador, atuando diretamente em suas juntas.

Dependendo do tipo de acionamento do robô, o elemento de potência pode ser um amplificador de potência no caso de motores elétricos, um compressor no caso de acionamento pneumático ou um regulador de pressão para acionamento hidráulico.

Dos componentes definidos para um robô, o controlador terá um enfoque especial neste trabalho.

1.3 ARQUITETURAS DE CONTROLADORES

1.3.1 Arquiteturas centralizadas

A Figura 1.1 ilustra uma arquitetura centralizada de um controlador para manipulador industrial típico. Nesta organização, o controlador (um único processador) é responsável por todas as tarefas de operação, tais como:

- controle de juntas;
- interface com o operador;
- inicialização e programação de tarefas;
- aquisição e armazenamento de dados;
- interface com equipamentos industriais;
- monitoramento do espaço operacional e aspectos de segurança.

No caso de grandes instalações industriais, é comum a existência de redes que interligam vários controladores, que operam, no entanto, de modo *stand-alone* dentro de seu espaço.

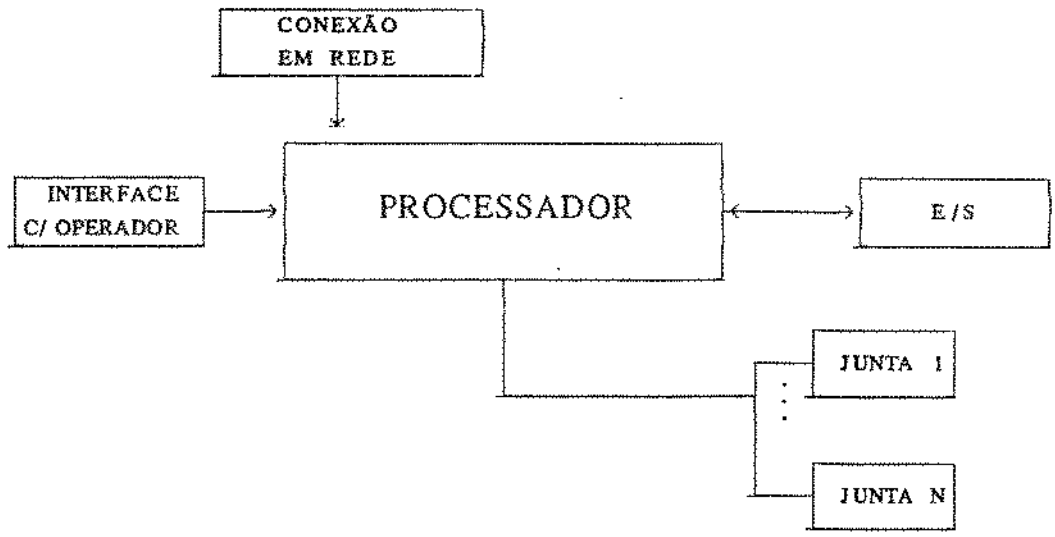


Figura 1.1 - Arquitetura centralizada para um controlador.

O controle centralizado tem como principal vantagem o fato de não apresentar pontos de estrangulamento de comunicação.

Por outro lado, o controle centralizado caracteriza-se por impor ao processador uma sobrecarga de processamento, o que leva muitas vezes a uma queda do desempenho do sistema.

O advento dos microcomputadores pessoais da linha IBM e Mackintosh tornou possível a adoção de plataformas que reúnem capacidade de processamento com uma grande gama de aplicativos que permitem a execução das tarefas alocadas ao controlador.

Nesta solução, a plataforma é estabelecida pela máquina através do *hardware* (*mother-board* e interfaces) e do sistema operacional, sobre a qual podem ser desenvolvidos cartões dedicados para o controle e acionamento das juntas e programas aplicativos que executem os algoritmos de controle, geração de trajetórias e a interface homem- máquina.

As principais vantagens desta implementação são:

• O uso de uma plataforma comercial com tendência de aumento de capacidade de processamento e diminuição do custo;

• Ausência de comunicação entre processadores;

• Alto nível de integração devido ao uso de cartões dedicados para controle e acionamento das juntas que podem interconectar-se diretamente nos *slots* da *mother-board* do Microcomputador.

Em contrapartida a este conjunto de vantagens, pode-se verificar que esta solução possui como desvantagens:

• Necessidade de interação em tempo real entre programas aplicativos, levando ao aumento da complexidade dos programas, com custos adicionais de desenvolvimento e teste;

• Se por um lado esta arquitetura não apresenta o problema de comunicação entre processadores, o fato de existirem diversos processos concorrentes e que trocam mensagens, leva ao surgimento de pontos de estrangulamento na comunicação de processos (troca de mensagens entre regiões de memória);

• Para executar as funções fins do sistema, o processador tem que efetuar ações adicionais para controle de alocação de memória para comunicação entre processos, priorização de programas que estejam em suas regiões críticas de execução, e outras ações pertinentes aos sistemas de *software* em tempo real.

Como exemplos de sistemas implementados segundo uma arquitetura centralizada, temos o controlador do Manipulador 7576 (Tipo SCARA) e o robô servo-hidráulico da SERVUS (SR4) [2].

1.3.2 Arquiteturas distribuídas

Como foi discutido no item 1.3.1, uma das principais desvantagens de um controlador centralizado, que é a sobrecarga do processador, levou à adoção de uma arquitetura onde um processador poderoso efetua uma distribuição de tarefas (rotinas de *software*).

Nesta implementação, apesar de existir um único processador, representado pela própria máquina, estão presentes também as características de uma arquitetura distribuída, uma vez que cada tarefa é vista como um processo lógico em execução. Do ponto de vista do sistema operacional da máquina, cada processo representaria um processador em estado de latência aguardando ativação e a máquina em si seria o conjunto desses processos.

A construção deste sistema necessita de um desenvolvimento significativo de *software* básico, para controlar e priorizar a ativação das tarefas, e de programas aplicativos que as executem. Além disso, a execução das tarefas ocorre em tempo real e de modo concorrente, podendo então surgir *overheads* que podem diminuir significativamente o desempenho do sistema.

Estes fatores motivam em muitos casos a adoção de uma arquitetura distribuída tanto em *hardware* como em *software*. Este processamento distribuído pode ser de duas maneiras:

a) Uma tarefa tem sua execução distribuída em vários processadores a partir da alocação das partes concorrentes aos diversos processadores interligados por barramento ou memória comum.
b) As tarefas são independentes entre si e distribuídas geograficamente nos diversos processadores. Neste caso a configuração assemelha-se à de uma rede de computadores.

O tipo de distribuição do item b adequa-se ao controle de robôs, pois este possui um conjunto de tarefas elementares e de execução independente, tais como controle de juntas, controle de trajetórias, comunicação entre processadores e interface homem-máquina.

As principais vantagens da utilização de controladores com processamento distribuído são:

- Otimização de desempenho através do uso de processadores adequados a cada aplicação;
- Projeto, codificação, teste e depuração do *software* para cada processador podem ser feitos independentemente de outros processadores;
- Sistema modular, com desenvolvimento de controladores específicos a cada aplicação;
- Distribuição das tarefas de modo a facilitar a detecção e possível correção de falhas;

Em contrapartida, podem ser identificados os seguintes pontos negativos para o controle distribuído:

- A comunicação entre os processadores passa a ter um papel importante no desempenho do sistema, podendo ser um fator limitante nos processos em tempo real.

- A existência de módulos separados executando tarefas completas, traz à luz a questão da confiabilidade do sistema, pois a ocorrência de uma falha em um destes módulos pode implicar na interrupção da execução da tarefa. Neste caso, uma solução bastante empregada em diversos sistemas distribuídos envolve o uso de mecanismos de duplicação de módulos, o que pode resultar num aumento significativo no custo do sistema.

- Os diferentes módulos do sistema requerem ambientes de testes específicos e complexos.

A análise destes pontos por meio de procedimentos experimentais é justamente um dos objetivos objetivos deste trabalho, sendo necessário, para isso, a definição de uma arquitetura com estas características, definida no próximo item.

1.4 - ARQUITETURA PROPOSTA

Uma vez definida a opção por uma arquitetura de controle com processamento distribuído, a escolha sobre a topologia, a interação e o possível dimensionamento dos processadores assim como a alocação das tarefas tem que levar em conta os seguintes aspectos operacionais [1] :

a) O movimento da estrutura mecânica se realiza através de movimentos de rotação e translação de suas juntas que devem ser controladas simultaneamente e cujo acoplamento dinâmico dificulta o controle independente das mesmas.

b) O comportamento dinâmico da estrutura articulada, fortemente não-linear e dependente das condições operacionais, deve ser levado em consideração na estratégia de controle escolhida, para que possa haver acesso a elementos tais como parâmetros de controle, modelamento, interface com meio exterior, calibração, etc, levando assim a uma perfeita adequação do robô às suas aplicações.

c) A falta de conhecimentos e a dificuldade de acesso à estrutura interna dos supervisores de controle utilizados em robôs industriais obriga o usuário a adquirir pacotes computacionais e interfaces especiais onerosos para cada nova aplicação desejada.

Em função destes aspectos, foi concebida uma estrutura completa para este supervisor, mostrada na Figura 1.2. Deve-se salientar a utilização de algum tipo de interface homem-máquina e de um *software* de visualização do robô que, atuando de um modo conjunto, possibilitam a geração das trajetórias, posteriormente transmitidas à estrutura de controle propriamente dita.

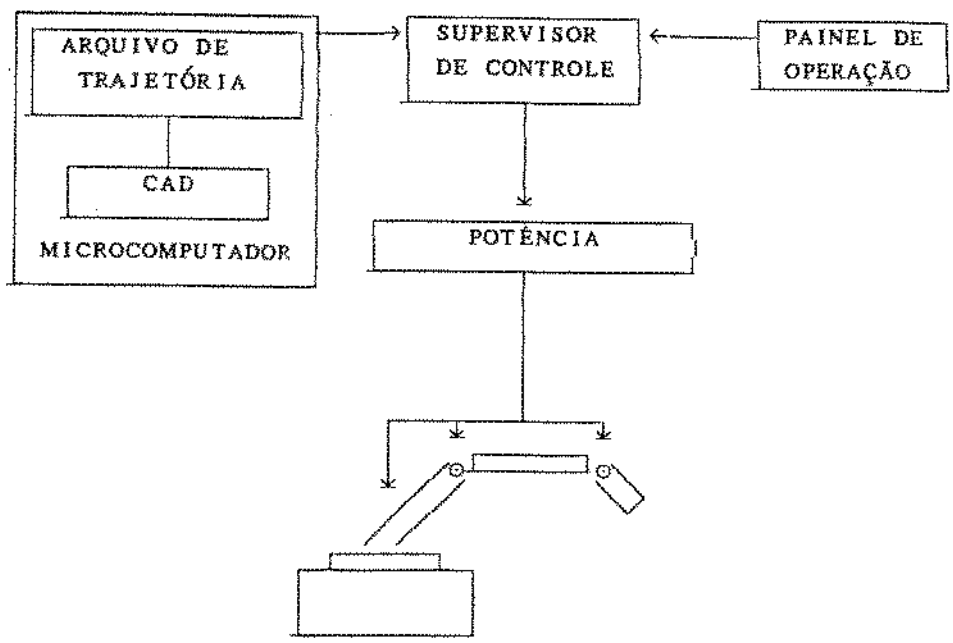


Figura 1.2 - Supervisor de Controle

A concepção desta arquitetura está baseada na técnica de controle ATGS (Autonomous Trajectory Generating Servomechanism), segundo a qual o controle de movimentos de robôs manipuladores é hierarquizado em 2 níveis: o nível de controle dos servo-mecanismos das juntas e o de controle de trajetórias.

Nessa estrutura, os algoritmos de controle dos servo-mecanismos das juntas são executados por microprocessadores dedicados para cada junta enquanto que a supervisão e o controle de trajetórias são executados em outro microprocessador dedicado a esta função.

A partir da técnica ATGS, foi estabelecido um modelo com uma estrutura hierárquica onde identificam-se os seguintes níveis:

• Nível 1: implementa os algoritmos de controle para cada junta do robô.

• Nível 2: atua, a partir do modelo cinemático inverso, na geração de trajetórias e no controle do elemento terminal. Possui comunicação com todas as unidades de nível 1 assim como com o ambiente externo.

• Nível 3: atua como gerador de trajetórias *off-line*, oferecendo também uma interface homem-máquina para visualização dos movimentos.

Deve-se observar o acréscimo de um nível (nível 3) em relação ao modelo original, que propicia, por um lado, a geração de trajetórias segundo perfis adequados a cada aplicação e, por outro lado, estabelece uma interface homem-máquina amigável ao operador do sistema, com recursos de visualização para acompanhamento dos movimentos.

É importante destacar que esta arquitetura, sendo distribuída, tem como um dos seus pontos mais fortes o processamento distribuído, o que possibilita a divisão de tarefas e otimização de recursos de cada processador envolvido. No entanto, este fator positivo determina um dos pontos críticos desta arquitetura, ou seja, a problemática da comunicação entre os processadores, onde podem ser identificados pontos de estrangulamento no sistema.

Estes aspectos foram levados em conta na implementação e nas montagens experimentais do protótipo, sendo enfatizados os tempos envolvidos no processamento e na comunicação entre os processadores.

Na Figura 1.3 pode-se observar a estrutura hierarquizada concebida para o supervisor.

As funções definidas para cada um dos níveis são desempenhadas pelos módulos que constituem os níveis. Estes módulos são implementações de *hardware* e *software* e são objeto de estudo dentro deste capítulo.

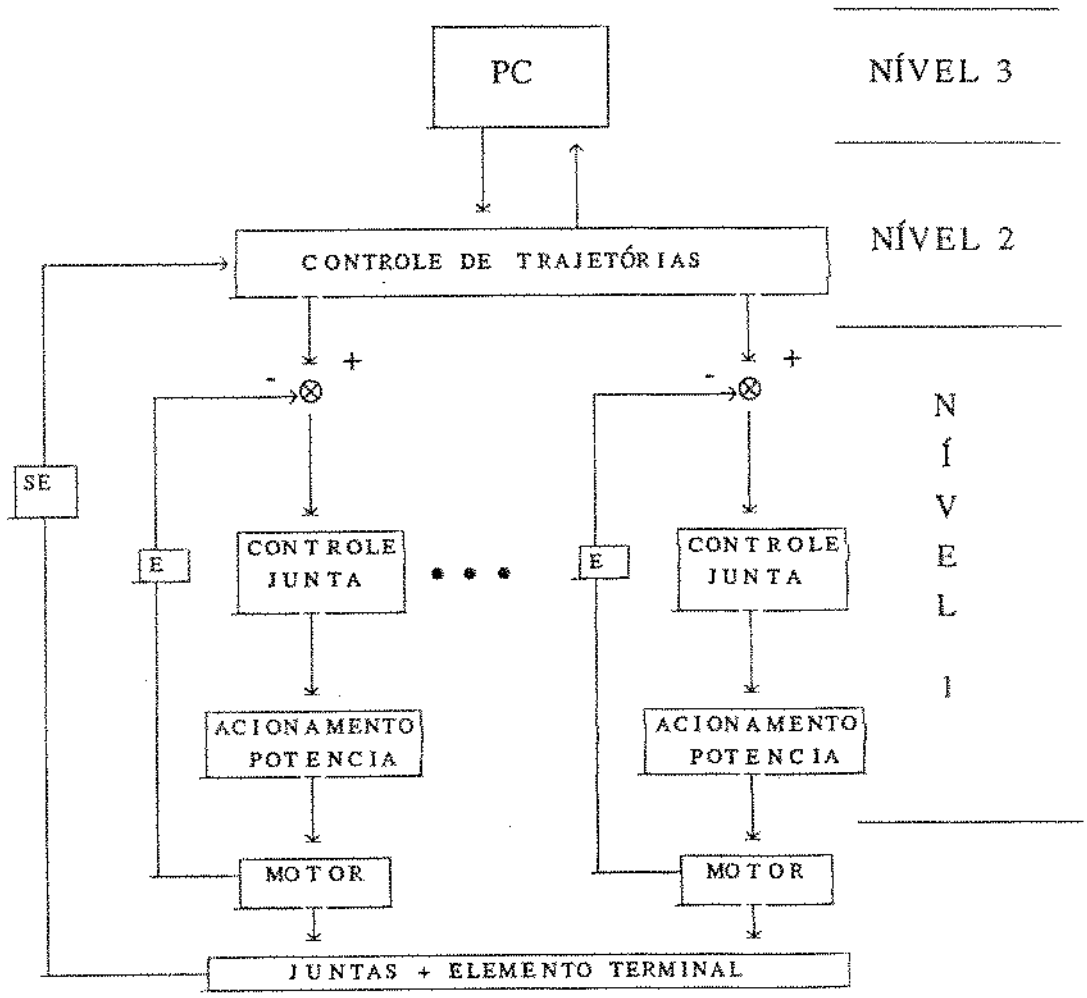


Figura 1.3 - Estrutura do Supervisor de Controle.

Na Figura 1.3 são identificados os seguites elementos:

• **PC** : Provê uma interface homem-máquina para geração de trajetórias através de programas CAD, desenvolvidos a partir do modelo geométrico do robô e ambiente de atuação. No protótipo, as funções pertinentes a este nível são executadas por microcomputador compatível com a linha IBM-PC.

• **Controle de trajetórias**: Neste módulo são efetuados o controle de trajetória assim como a execução dos modelos geométricos. Estas funções são desempenhadas por uma placa com microprocessador executando os programas destinados à execução das mesmas.

• **Controle de juntas**: Esta função é implementada por *n* placas com microprocessador, uma para cada junta do robô, executando o *software* de controle de junta.

• **Acionamento/Potência**: Provê o acionamento do motor a partir da ação de controle resultante da execução do *software* de controle.

• **E - Encoder:** É o dispositivo eletro-mecânico que permite a transdução de posição angular para sinais elétricos que são tratados nas placas.

• **M - Motor:** Este módulo efetua o deslocamento angular da junta.

• **Elemento terminal**: Este módulo representa a ferramenta ou a garra que executa a tarefa específica do robô.

• **SE - Sensor de esforço:** É o elemento que faz a transdução de força para deslocamento linear e de momento angular para deslocamento angular.

1.5 - APLICAÇÃO DA ESTRUTURA PROPOSTA EM UM ROBÔ INDUSTRIAL

A partir da descrição da arquitetura de um supevisor de controle no item 1.3, é proposto um estudo para efetuar o controle para um robô industrial com 6 graus de liberdade.

O algoritmo utilizado para a geração de trajetórias consiste em calcular a velocidade máxima proporcionalmente ao deslocamento imposto a cada junta, de modo que haja simultaneidade de movimentos entre elas. O sistema também permite a utilização de um arquivo de pontos contendo a trajetória completa a ser executada pelo robô.

O sistema constituiu-se numa plataforma *hardware-software*, formada pelos módulos funcionais ilustrados na Figura 1.2, onde aplicativos de controle de juntas e de trajetórias realizam o controle de posicionamento e movimentação do robô, a partir de seus modelos geométrico/cinemático direto e inverso. O objetivo é o de utilizar uma solução aberta que possibilite implementações do mesmo adequadas para diferentes tipos de manipuladores.

A seguir, é apresentada a descrição dos módulos funcionais, onde se enfatizam os aspectos importantes de *hardware* e *software* dentro dos blocos que constituem os módulos.

1.5.1. Módulo PC

Dentro da linha de utilização de plataformas abertas, uma solução adequada para este módulo é o uso de um microcomputador compatível com a linha IBM PC-AT em ambiente DOS, podendo evoluir para ambiente WINDOWS, e linguagem de programação C. Este módulo é constituído pelos blocos ilustrados na Figura 1.4.

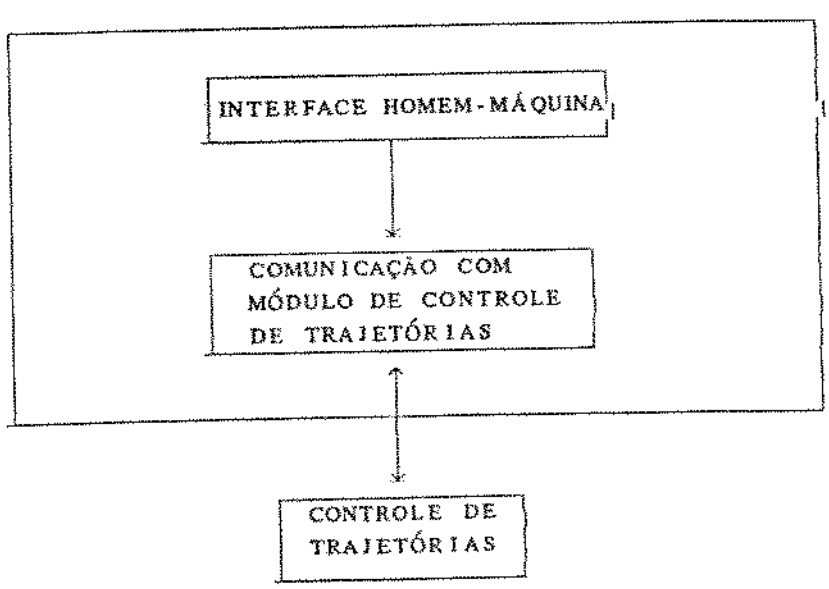


Figura 1.4 - Diagrama de blocos do módulo PC.

a - **Interface homem-máquina.** Este recurso fornece telas interativas com menus, possibilitando que o usuário estabeleça a trajetória desejada através de pontos:

- obtidos a partir do modelo inverso [1];
- inseridos pelo teclado [6];
- obtidos pela execução de trajetórias em CAD [7].

Neste caso, um recurso de grande utilidade tanto no desenvolvimento de algoritmos como na utilização destes em aplicações práticas é um *software* de visualização que possibilite, a partir do modelo do robô, a verificação da evolução de uma trajetória com acompanhamento gráfico. O Capítulo 2 faz uma referência a um *software* com esta finalidade.

b - **Comunicação com módulo de controle de trajetória.** A comunicação é realizada por uma interface serial, executando um protocolo assíncrono sobre uma interface física RS-232, onde são definidas mensagens de comando, controle, sequenciamento e alarme que garantem a integridade dos pontos referentes a cada trajetória enviada. Este bloco comunica-se com o bloco **interface com PC**, pertencente ao **módulo de controle de trajetórias**.

1.5.2. Módulo de Controle de Trajetórias

O PC envia a este módulo os pontos referentes a uma dada trajetória, para que, a partir do modelo inverso do robô, os deslocamentos angulares referentes a cada junta sejam calculados e enviados cada um dos controladores.

A modelagem cinemática inversa é empregada pois, na maioria das aplicações em robótica industrial é requerido o conhecimento exato do posicionamento do manipulador dentro de um volume de trabalho (controle de esforços, controle no espaço cartesiano, identificação de parâmetros).

Na Figura 1.5 são identificados os blocos funcionais que constituem este módulo.

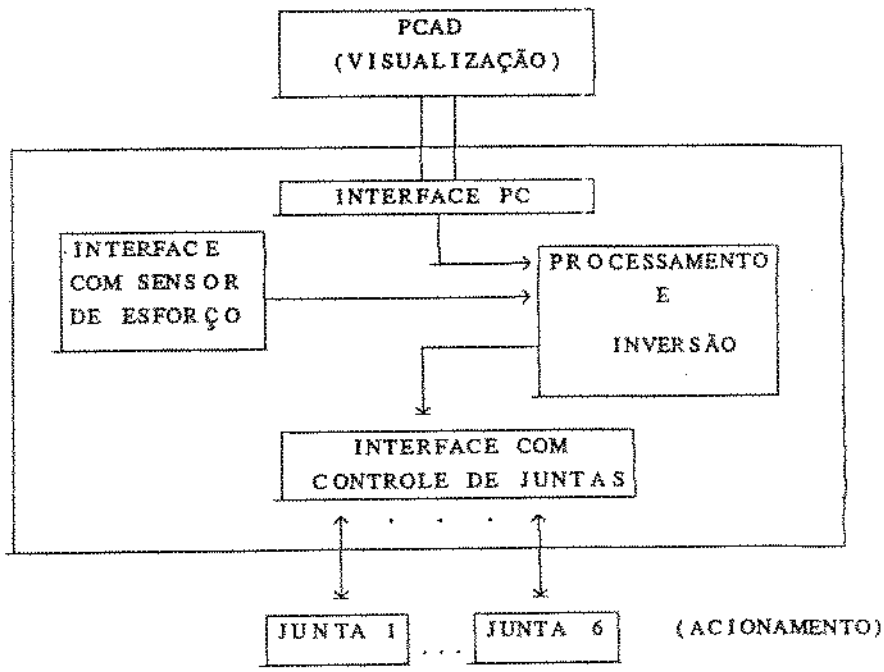


Figura 1.5 - Diagrama de blocos do módulo de controle de trajetórias

a - **Bloco de interface PC**: bloco que se comunica bidirecionalmente com o PC através de uma interface serial, recebendo os pontos referentes à trajetória e enviando ao PC informações a respeito da posição real do elemento terminal.

Nesta interface as mensagens trocadas apresentam a estrutura disposta na Figura 1.6:

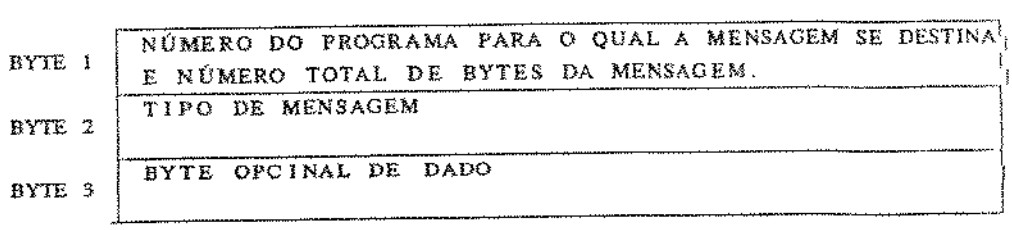


Figura 1.6 - Estrutura da mensagem da interface de comunicação

b- **Bloco de processamento e inversão**: constitui-se basicamente de um microprocessador comercial que recebe os pontos do **bloco de interface** e as leituras do sensor de esforço e, através do modelo cinemático inverso do robô realizado a partir de algoritmos de inversão matricial, obtém os incrementos angulares juntas que poderão ser enviados em tempo real ao manipulador.

c- **Bloco de interface de controle de juntas**: é formado por interfaces seriais, uma para cada módulo de controle de junta. As mensagens de controle são trocadas entre o módulo de controle de trajetória e os **módulos de controle de juntas** segundo um mecanismo de interrupção. Este mecanismo possibilita um interfaceamento simples, se for comparado com um arbitrador ou alocador de mensagens, mas eficiente se levadas em conta as constantes de tempo envolvidas.

Neste bloco, deve-se destacar o microprocessador que faz o tratamento dos pedidos de interrupção e implementa o protocolo de mensagens trocadas entre os módulos, além do controlador programável de interrupção, que atende até 8 níveis de pedido. Os tipos de mensagens trocadas são mostrados na Tabela 1.1.

TIPO DE MENSAGEM	BYTES	SENTIDO O FLUXO
1. DADOS	2	UCS ⇒ UCJ
2. SUPERVISÃO	2	
a. início de trajetória		UCS ⇒ UCJ
b. posição atual		UCS ⇒ UCJ
c. estado inicial		UCJ ⇒ UCS
d. alinhamento		UCS ⇒ UCJ
e. operacional		UCJ ⇒ UCS
3. ALARME (HALT)	1	UCS ⇔ UCJ

Tabela 1.1 - Tipos de mensagens trocadas na interface com omódulode controle de juntas.

1.5.3- **Módulo de Controle de Juntas**

A trajetória de um robô é o resultado da combinação dos movimentos angulares das juntas que o compõem. Após a obtenção dos valores dos deslocamentos angulares no **módulo de controle de trajetórias** segundo uma curva de aceleração pré-determinada, cada **módulo de controle de junta** executa o algoritmo de controle de acordo com a aplicação.

A estrutura de controle reproduzida pelo módulo de controle de juntas pode ser representada pela Figura 1.7 enquanto que seus módulos funcionais são mostrados na Figura 1.8.

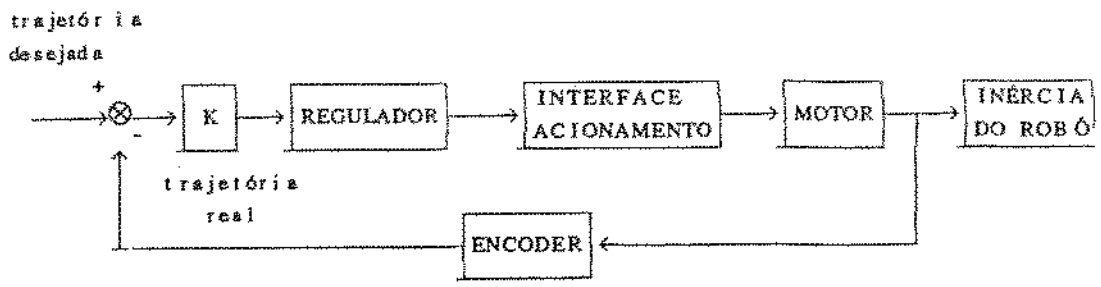


Figura 1.7 - Estrutura de Controle

A Figura 1.7 apresenta uma estrutura de controle de cada junta, onde a trajetória desejada é proveniente do **módulo de controle de trajetória** e a trajetória real é obtida a partir dos transdutores de posição (*encoders*).

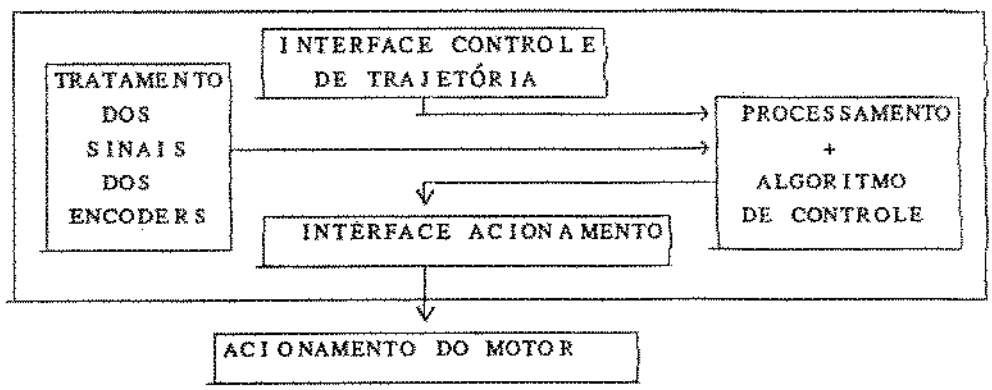


Fig. 1.8 - Diagrama de blocos do módulo de controle de juntas

a - Bloco de interface com controlador de trajetória: este bloco tem uma interface física e lógica que recebe do **módulo de controle de trajetórias** o deslocamento angular desejado relativo à junta que está sendo controlada.

Por sua vez, a interface utiliza um protocolo de trocas de sinais para as trocas de mensagens que são controladas por um mecanismo de interrupção que garante, não obstante sua simplicidade, a integridade dos dados trocados. Este protocolo obedece à descrição das mensagens mostradas na Tabela 1.1.

b- Bloco de tratamento de sinais de encoder: a posição real de uma junta é dada pelos sinais oriundos do elemento que efetua a leitura no eixo do motor. No caso deste elemento ser um *encoder*, são enviados 3 sinais, sendo 2 pistas lógicas de igual período e defasadas de 90° e outra pista que sinaliza a localização de um pulso de início de curso.

Para prover um tratamento que transforme estes sinais do *encoder* na posição real da junta, de modo que a mesma possa ser comparada ao posicionamento desejado, uma representação simplificada do bloco de tratamento de sinais de *encoders* é mostrada na Figura 1.9.

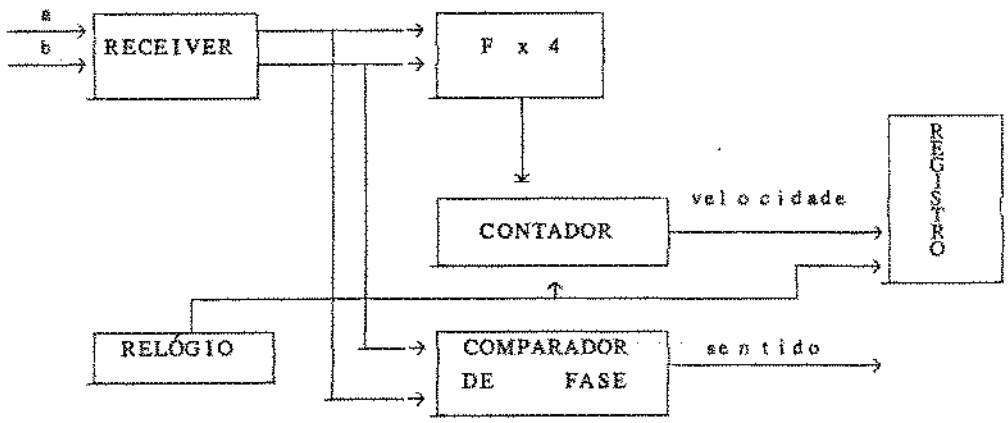


Figura 1.7 - Diagrama do bloco de tratamento dos encoders

A partir deste tratamento, as informações a respeito da posição real da junta estarão disponíveis para o processador, que efetua a comparação com a posição desejada e executa a ação de controle apropriada. Essa seqüência de eventos é objeto de análise do Capítulo 3.

c- **Bloco processamento e algoritmo de controle:** conforme descrito no item b, as informações a respeito da posição real são lidas pelo processador e comparadas com a posição desejada, para que, neste mesmo processador seja executado o algoritmo de controle. Em seguida, a ação de controle resultante da execução do algoritmo, é enviada ao **bloco de interface de acionamento**.

Os eventos que envolvem a priorização e o tratamento das interrupções também são executados no âmbito deste bloco.

1.5.4. **Acionamento/Potência**

O módulo de Acionamento/Potência tem a função básica de converter o sinal resultante da execução do algoritmo de controle pelo **módulo de controle de juntas** e disponível no **bloco de interface de acionamento**, em um sinal de características adequadas ao tipo de acionamento utilizado no robô.

1.5.5. **Motor**

A partir do sinal enviado pela interface de potência o motor efetua o deslocamento angular. Como foi dito anteriormente, a natureza deste motor depende da aplicação a que se destina, podendo ser :

- elétrico (corrente contínua, alternada ou motor de passo)
- hidráulico
- pneumático

1.5.6. **Elemento terminal e sensor de esforço**

No elemento terminal do robô geralmente é desenvolvido um sistema de preensão universal onde acopla-se uma ferramenta que efetua a tarefa para a qual o robô está sendo empregado. Neste elemento terminal, que retrata a posição espacial real do manipulador, pode ser acoplado um sensor de esforço. O sensor tem características de transdução, transformando o momento angular em deslocamento angular e força em deslocamento linear. Estas grandezas são realimentadas diretamente no **módulo de controle de trajetórias** de maneira a compor, juntamente com os pontos enviados pelo PC, uma trajetória mais suave.

1.6. CONCLUSÃO

Este capítulo apresentou uma descrição completa de um robô industrial a partir de seus elementos constituintes e diferentes interfaces. Também foram abordados aspectos acerca de arquiteturas de controladores de robôs, com destaque à arquitetura hierarquizada baseada na técnica ATGS, definida para a implementação do supervisor de controle.

A arquitetura proposta foi descrita através de seus módulos e blocos constituintes, onde são salientados alguns pontos importantes tais como as lógicas e protocolos que foram definidos visando-se a implementação, que passa a ser descrita no Capítulo 2.

CAPÍTULO 2

IMPLEMENTAÇÃO DO SUPERVISOR DE CONTROLE

2.1 - INTRODUÇÃO

Este capítulo apresenta uma descrição detalhada do *hardware* e do *software* que foram concebidos para validar a arquitetura descrita no capítulo anterior.

Inicialmente tem-se uma descrição funcional do *hardware*, onde detalhes importantes da implementação são mostrados no decorrer do texto. Os esquemas completos são apresentados no Anexo I.

O *software* implementado é descrito na seqüência, com alguns detalhes importantes sendo mostrados durante o texto, enquanto que as listagens constam no Anexo II.

2.2 - DESCRIÇÃO DO PROTÓTIPO

Conforme exposto no Capítulo 1, a arquitetura de controle na qual este trabalho foi baseado, tem como características principais a natureza hierárquica aliada a um alto grau de modularidade.

Com o intuito de conceber um protótipo que fosse simples de ser implementado e permitisse a validação da estrutura de controle, partiu-se para uma abordagem na qual considera-se a estrutura de controle do ponto de vista de seu elemento básico, ou seja, o controle de juntas.

Como elemento motivador para esta escolha, deve-se salientar o fato de que o *hardware* da UCJ (Unidade de Controle de Juntas) tem cerca de 90% de sua implementação em comum com o *hardware* da UCS (Unidade de Controle e Supervisão), sendo inclusive a mesma PCI (placa de circuito impresso).

As considerações feitas a respeito do *hardware* também valem para o *software* onde encontraremos uma grande superposição entre a UCI e a UCS, pois tanto o programa de alocação e ativação de tarefas (Monitor) como alguns aplicativos são comuns a ambas as placas.

Este protótipo provê uma plataforma *hardware* e *software* sobre a qual aplicativos de controle são executados, atuando em conjunto com um PC-AT para efetuar tarefas num robô de 3 graus de liberdade que será descrito no Capítulo 4.

Partindo-se do modelo hierárquico descrito no Capítulo 1, pode-se mapear as principais funções de cada nível segundo sua implementação no protótipo, conforme mostra a Tabela 2.1.

MODELO HIERÁRQUICO/ FUNÇÃO DESEMPENHADA	IMPLEMENTAÇÃO NO PROTÓTIPO
NÍVEL 1 -Efetua controle de junta e acionamento do motor	UCI/UAM (unidade de controle de juntas/unidade de acionamento de motor)
NÍVEL 2 -Envia pontos segundo uma trajetória controlada por algoritmo inverso	COMPUTADOR PC
NÍVEL 3 -Oferece interface homem-máquina e gera trajetórias	COMPUTADOR PC

Tabela 2.1 - Mapeamento das funções segundo a implementação no protótipo

O protótipo é constituído dois tipos de elementos fundamentais: módulo e bloco. Um módulo é um conjunto *hardware-software* que encerra o elenco de funções relativas a cada nível da estrutura hierárquica. O bloco, por sua vez, é a entidade que, associada a outros blocos contitui módulo, existindo em forma de *hardware* (circuito, fiação e peças mecânicas) ou *software* (programas básicos ou aplicativos).

Deste modo, o protótipo é formado pelos seguintes módulos: um módulo PC que atua com os módulos de controle de junta, um para cada junta, para controlar os movimentos do robô. Cada módulo de controle de junta envia comandos a um estágio de acionamento que efetua separadamente a movimentação das juntas. A Figura 2.1 ilustra o sistema.

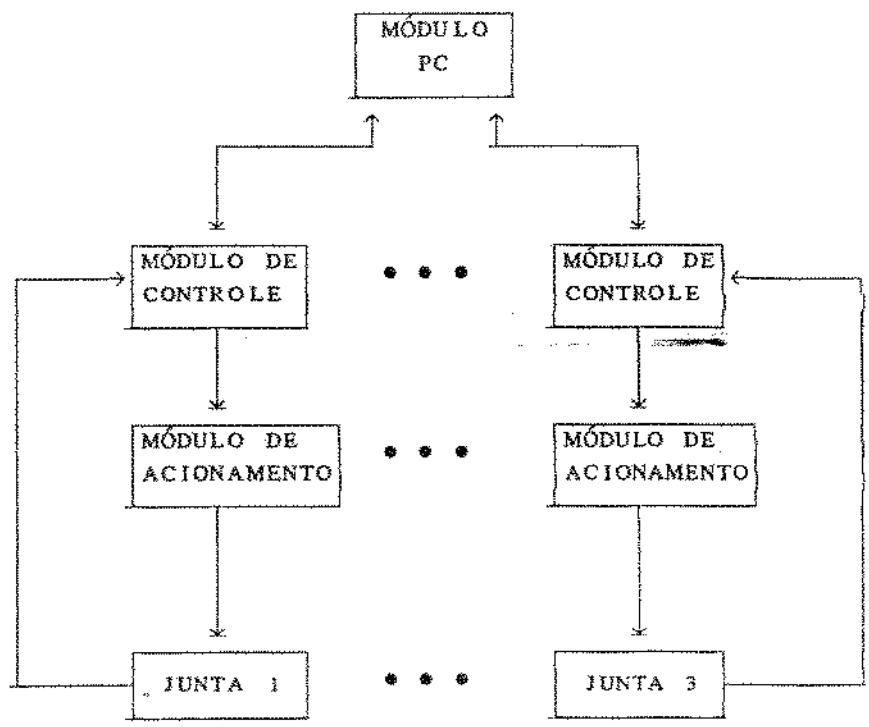


Figura 2.1 - Estrutura do Protótipo

2.3 - Descrição do *HARDWARE*

Conforme descrito no item 2.2, o protótipo é constituído pelo módulo PC, módulo de controle de junta e módulo de acionamento. A cada um destes módulos são associadas as implementações físicas de *hardware* (placas) e *software* (programas). O *software* será discutido no item 2.4.

O *hardware* de cada módulo está implementado da seguinta maneira:

a) Módulo PC: implementado em um equipamento IBM-PC, com uma de suas interfaces seriais comunicando-se com o módulo de controle de junta.

b) Módulo de controle de juntas: implementado em um *rack* onde residem as placas que efetivamente realizam o controle das juntas. Estas placas são chamadas Unidades de Controle de Junta, ou simplesmente UCJ.

c) Módulo de acionamento de junta: implementado numa placa que gera os sinais apropriados para o acionamento do motor da junta. Esta placa é a Unidade de Acionamento do Motor (UAM).

No protótipo em questão, o robô tem suas juntas acionadas pelo movimento de motores de passo. Desse modo, a placa que efetua o acionamento do motor tem uma natureza transcodificadora, transformando comandos gerados pela UCJ em pulsos que movimentam os motores.

É importante identificar o aspecto modular do protótipo no que diz respeito ao tipo de motor do robô a ser controlado, pois no caso de termos um acionamento por motor de CC, o módulo de acionamento seria implementado a partir de um conversor analógico-digital. E assim, para cada tipo de acionamento, pode-se prever uma UAM diferente.

A Figura 2.1 pode ser refeita com as unidades respectivas a cada módulo, conforme indica a Figura 2.2. Nesta figura podem ser observadas também as interfaces existentes entre os módulos.

Na seqüência, serão apresentadas as descrições detalhadas das unidades que foram desenvolvidas para a concepção do protótipo (UCJ e UAM).

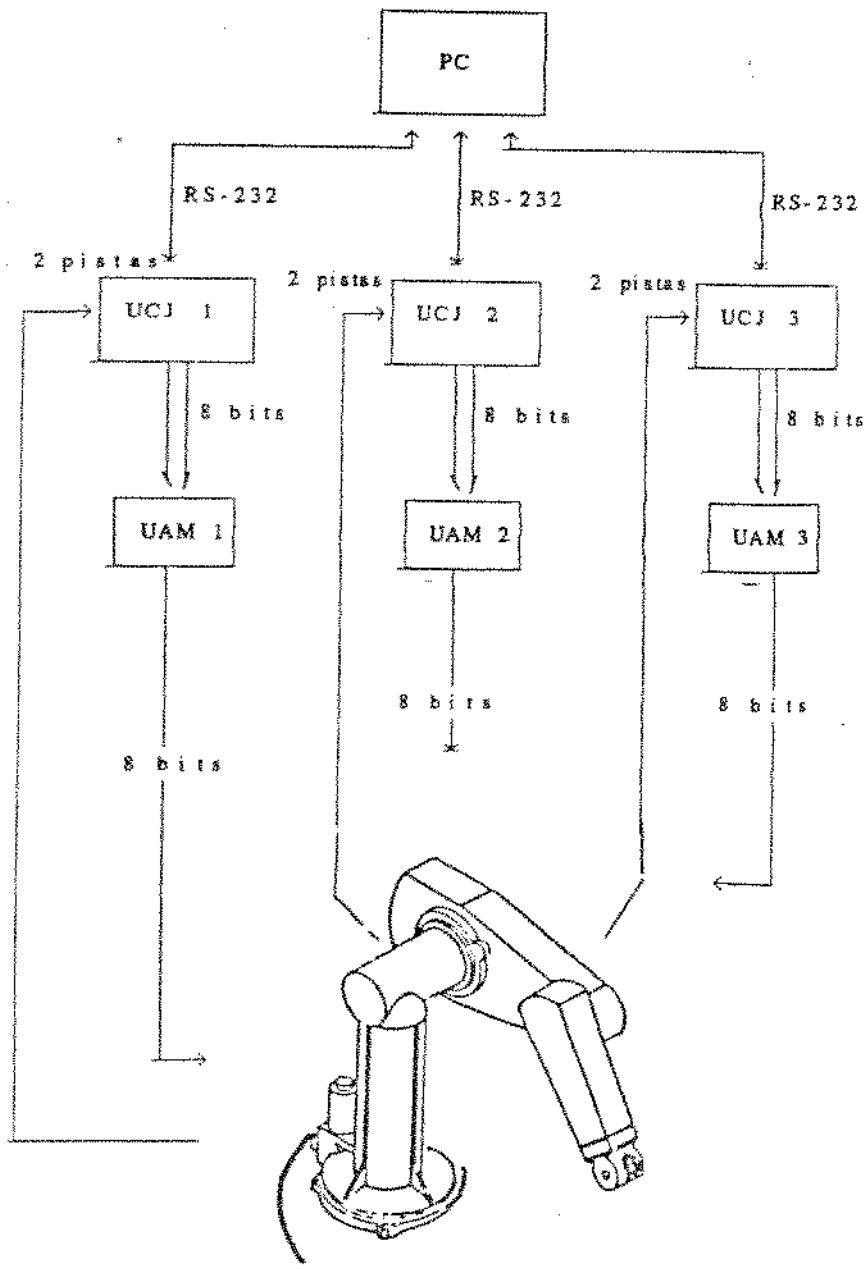


Figura 2.2. - Estrutura completa do protótipo

As descrições das unidades podem ser acompanhadas pela consulta ao Anexo I, onde estão os esquemas elétricos e o diagrama de blocos.

2.3.1. UCJ - Unidade de Controle de Junta

2.3.1.1- Características

A Unidade de Controle de Junta tem a função de efetuar o controle da posição e do movimento de uma junta, possuindo as seguintes características:

- Microprocessador NEC V-20. Este microprocessador é compatível com o microprocessador INTEL 8088 tanto quanto à pinagem como quanto ao *set* de instruções, constituindo-se, devido à duplicação de seu barramento interno, num *upgrade* deste último, possuindo um desempenho pelo menos 30% maior para um mesmo relógio;

- Memória RAM estática com 32 Kbytes de capacidade;

- Memória EPROM com 64 Kbytes de capacidade;

- Controle de 8 níveis programáveis de interrupção concentrados num componente VLSI, o 8259;

- Controle de linha serial feito pelo controlador programável 8530, bastante utilizado em PC;

- Mecanismo que permite a execução de um autoteste que valida as principais lógicas da unidade.

Além dessas características, a UCJ possui as seguintes interfaces externas:

- Interface serial assíncrona com nível elétrico TTL estabelecendo uma conexão com o **módulo PC** no protótipo atual. A próxima etapa de desenvolvimento do protótipo, em andamento, estabelecerá a conexão deste módulo com o **módulo de controle de trajetórias**, implementado na UCS.

• Interface serial assíncrona padrão RS - 232, para conexão com um terminal de vídeo;

• Interface paralela de 8 bits conectando-se com o Módulo de Acionamento de Junta, implementado pela UCM;

• Interface serial de nível elétrico TTL com a função de receber os sinais do *encoder*.

A família lógica dos demais componentes (portas lógicas e registradores) é a HCMOS, devido ao seu consumo menor e maior imunidade a ruído.

2.3.1.2. Descrição Funcional dos principais elementos

• Processador

Constituído pelo microprocessador NEC-V20 [9], compatível com o componente 8088 da INTEL [8], possuindo barramento de dados de 8 bits e barramento de endereço de 20 bits (capacidade de endereçamento de 1 M byte).

O endereçamento do NEC-V20 é do tipo indireto, ou seja, o endereço físico é resultado de uma combinação entre o endereço de segmento (SSSS) e endereço de *off-set* (OOOO). Dessa forma, todas as referências feitas a endereços neste texto, serão da forma SSSS:OOOO.

O relógio é de 8 MHz em onda quadrada com zero *wait-state* devido à operação com RAM estática e EPROM de alta velocidade.

• Relógio

Este circuito oscila a 8 MHz gerando uma onda quadrada que é a base para todos os relógios utilizados na placa (4 MHz para periféricos, 1KHz para interrupção e outros de uso geral).

• Reset

Este circuito gera um pulso positivo (ativo em nível alto), de duração de 240 ms, a partir do acionamento de uma chave do tipo *push-button*.

A duração deste pulso garante que a CPU entre em ciclo de *reset*, onde os sinais de controle são colocados em estado inativo e o barramento de endereços aponta para o endereço F000:FFF0H (endereço físico FFFF0H).

• Barramento de dados

Esta circuito constitui-se num barramento bidirecional, controlado pelos sinais NENDATA e PDIRTNR. O sinal NENDATA é gerado na CPU e ativo em nível baixo, habilitando o fluxo de dados de/para a memória e dispositivos de entrada a saída. A direção do fluxo é dada pelo sinal PDIRTNR que, em nível alto direciona o fluxo para escrita enquanto que em nível baixo, direciona o fluxo para leitura.

As linhas de dados têm sua capacidade de corrente aumentada por rede resistiva.

• Barramento de endereços

Este circuito garante que os endereços permaneçam estáveis durante o ciclo de acesso. O sinal PALE, gerado pela CPU, funciona como um relógio, gravando as linhas de endereço, que permanecem estáveis até o próximo ciclo de acesso da CPU.

• Memória

A memória da está dividida em 2 blocos: o de RAM, com 32K x 8 bits e o de EPROM com 32K x 8 bits. Os 20 bits de endereço da CPU podem endereçar 1 Mbytes, numa faixa compreendida entre 00000H e FFFFFH, sendo que a faixa reservada para RAM vai de 00000H a 07FFFH e a faixa reservada para a EPROM ocupa as últimas posições de F8000H a FFFFFH.

O banco de RAM está implementado numa RAM estática 62256, com tempo de acesso de 100 ns. A EPROM utilizada é a 27256 com 250 ns de tempo de acesso.

• Decodificador de endereços de memória e periféricos

Esta lógica transforma os sinais de endereço originados na CPU em sinais de seleção de memória e de dispositivos de entrada e saída. A lógica se concentra em uma PAL (Programmable Array Logic), onde, a partir das equações que descrevem a geração dos sinais de seleção, são estabelecidos os circuitos combinacionais que geram estes sinais.

O mapeamento completo de endereços encontra-se descrito no item 2.3.1.3.

• Linha serial

Este circuito é formado pelo SCC (Serial Communication Controller) 8530 [12] e os *drivers* de interface. O controlador serial 8530 da AMD possui dois canais programáveis, independentes e bidirecionais que efetuam a conversão serial/paralela e paralela/serial entre o barramento de dados e as interfaces seriais.

O canal A implementa uma interface assíncrona TTL para comunicação com a Unidade Central de Supervisão (UCS), enquanto que o canal B implementa uma interface assíncrona RS-232 [4] para comunicação com o PC.

Interface com a CPU é efetuada por um sinal de interrupção que faz um pedido à lógica de controle de interrupção a cada evento cuja ocorrência esteja programada para gerar interrupção.

• Controle de interrupções

Esta lógica concentra-se no PIC (Programmable Interrupt Controller) 8259 [13], que é um controlador programável de interrupção com 8 níveis de pedido.

A partir da ativação de uma linha de pedido de interrupção, o controlador faz o pedido à CPU, recebendo como resposta 2 pulsos de leitura (sinal NINTA), que carregam o vetor que aponta o endereço de execução da rotina de tratamento referente ao pedido feito.

O controlador resolve a prioridade quando da ativação simultânea de mais de 2 pedidos de interrupção a partir da prioridade programada nos procedimentos iniciais para a operação da placa.

- **Tratamento dos sinais do encoder**

Os sinais do *encoder* são duas pistas defasadas de 90° em média que, após passarem pelo *receiver*, são tratadas de maneira a colocar à disposição da CPU as informações sobre a velocidade (variação) e sentido de rotação do motor.

O sentido é obtido através da comparação sincronizada entre as pistas. A variação é obtida a partir da duplicação da freqüência (mecanismo que aumenta a precisão do sistema), sincronizada pelo mesmo relógio utilizado na sincronização do sentido.

A sincronização é empregada para evitar perda de pontos quando da transição de movimentos.

Uma descrição mais detalhada deste mecanismo encontra-se no trabalho de *Fayan* [1].

- **Contador**

A resultante da duplicação dos sinais do *encoder* é o relógio de contagem do contador. O sentido da contagem é dado pelo sinal que indica o sentido de rotação da junta.

O valor da contagem é lido periodicamente pela CPU em um registro de leitura.

• **Acionamento do motor**

Este circuito atua como uma porta paralela por onde o valor resultante da execução do algoritmo de controle. O valor, representado por um byte, é utilizado para carregar a lógica de acionamento do motor.

2.3.1.3. Dispositivos programáveis

A UCJ possui como recursos programáveis a EPROM (Erasable Programmable Read Only Memory) e a PAL (Programmable Array Logic) [14]. Na EPROM residem os programas que executarão os algoritmos de controle e que serão os objetos de estudo do item 3.4, enquanto que a PAL implementa a lógica de geração dos sinais de seleção da memória e dos dispositivos de entrada e saída.

O tipo de PAL utilizado é a PAL16L8, que possui 10 portas de entrada e 8 portas bidirecionais. Conforme constante no Anexo I, as entradas e saídas são associadas em equações que geram um *fuse-map*, a partir das quais a PAL é gravada, utilizando-se o equipamento DATA IO 2900 [10].

A Tabela 2.2 apresenta a relação de equações que descrevem a geração dos sinais de seleção da memória (RAM e ROM) e dos dispositivos programáveis da UCJ. Nestas equações, a variável à esquerda do sinal " = " é programada como saída na PAL, sendo seu valor determinado pela combinação das variáveis à direita. A variável à esquerda do sinal " = " está definida em lógica negativa, ou seja, o sinal de seleção gerado está em sua forma barrada ($\overline{X}$), por uma imposição da lógica da PAL.

Na própria Tabela 2.2 está definida também a simbologia lógica adotada na descrição das equações apresentadas neste trabalho.

```
/NCSCONT = /NIOPM * /PA7 * PA6 * /PA5 * /PA4 * /NIORD
/NLOAD = /NIOPM * /PA7 * /PA6 * PA5 * /PA4 * /NIOWR
/NCSPIC = /NIOPM * PA7 * /PA6 * /PA5 * /PA4
/NCSROM = NIOPM * /PA19
/NCSRAM = NIOPM * /PA19
/NLDCONTR = /NIOPM * /PA7 * /PA6 * PA5 * PA4 * /NIOWR
```

Tabela 2.2 - Equações lógicas implementadas na PAL.

As seguintes observações podem ser efetuadas a partir da Tabela 2.2.

• os sinais PA0, PA1, PA4, PA5, PA6, PA7 e PA19 são entradas ligadas ao barramento de endereços da CPU, definindo o mapeamento de memória (RAM e EPROM) e de periféricos;

• o sinal NIOPM indica se o acesso é feito à memória (nível lógico 1) ou a um periférico (nível lógico 0);

• o sinal NIORD indica um acesso de leitura em periférico;

• os sinais de formato NCSXXXXX, indicam acessos à memória (RAM e EPROM separadamente) e aos seguintes periféricos: controlador de interrupção (PIC), controlador de linha serial (8530) e registrador de leitura de estado do contador dos sinais do *encoder*;

• o sinal NLOAD é responsável pelo carregamento do valor inicial do contador de sinais do *encoder*;
• o sinal NLDCONTR é responsável pela escrita na porta de interface com a UAM.

A Tabela 2.3 mostra os componentes que podem ser acessados pelo processador, indicando os respectivos endereços físicos.

SINAL	ENDEREÇO FÍSICO	DISPOSITIVO ENDEREÇADO
RDCONT	40H	REGISTRO DE LEITURA DO CONTADOR
WRPIC	80H	CONTROLADOR DE INTERRUPÇÕES
WRSIO	A0H	CONTROLADOR DE LINHA SERIAL
ROMCS	F8000H-FFFFFH	MEMÓRIA EPROM
RAMCS	00000H-07FFFH	MEMÓRIA RAM
WLDCONTR	30H	REGISTRO DE COMANDO PARA ACIONAMENTO DO MOTOR
WLOAD	20H	REGISTRO DE ESCRITA DO CONTADOR

Tabela 2.3 - Endereçamento físico dos componentes da UCJ.

2.3.2. **Unidade de Acionamento de Motor (UAM)**

2.3.2.1. Características

A função específica desta unidade é a de converter a informação resultante da execução do algoritmo de controle enviada pela UCJ em sinais elétricos para efetuar a movimentação do motor correspondente à junta controlada.

No robô do protótipo são empregados motores de passo acionados por impulsos de corrente cadenciados por um relógio, que pode ser ajustado em função das características do sistema. Como a UCJ envia uma palavra paralela correspondente à correção da trajetória, a UAM serializa essa palavra através de um contador, enviando ao motor um número de pulsos igual ao valor decimal representado na palavra. Na palavra enviada, o bit de sinal (bit6) determina o sentido de rotação do motor.

2.3.2.2. Descrição Funcional

A UAM constitui-se basicamente de um contador onde o valor inicial da contagem é carregado pela UCJ. Neste mesmo instante, inicia-se uma contagem decrescente. Quando o contador atinge o valor zero, a contagem é inibida, reiniciando-se novamente quando um novo valor é carregado.

2.4. DESCRIÇÃO DO *SOFTWARE*

Até este ponto os módulos existentes no protótipo foram descritos do ponto de vista do *hardware*, de um modo independente do *software* que é executado em cada módulo. Neste item será apresentada uma descrição dos programas existentes, com a apresentação do mapeamento que posiciona cada programa em seu respectivo módulo. A descrição dos programas pode ser acompanhada da consulta ao Anexo II, onde se encontram as listagens dos mesmos.

2.4.1. Organização dos programas

A Figura 2.2 apresentou a estrutura do protótipo em função de seus módulos constituintes. Assim como foi feito um mapeamento dos elementos *hardware* sobre essa estrutura no item 2.3, será feito o mesmo com o *software*, identificando-se os programas que atuam de maneira a executar as funções do supervisor.

Na Tabela 2.4 são identificados os programas principais, segundo uma divisão que os separa em *software* básico, referindo-se aos programas que, juntamente com o *hardware*, formam uma plataforma e *software* aplicativo, que são os programas executados sobre esta plataforma.

Os programas são identificados também segundo o módulo da estrutura em que está localizado.

MÓDULO	SOFTWARE BÁSICO	SOFTWARE APLICATIVO
PC	SISTEMA OPERACIONAL : DOS PROGRAMA DE EMULAÇÃO DE TERMINAIS (PCPLOT)	1- PROGRAMA GERADOR DE TRAJETÓRIAS 2- PROGRAMA DE GERAÇÃO DE PONTOS PARA A UCI
CONTROLE	MONITOR	1- CONTROLE_ON_OFF - ALGORITMO DE CONTROLE 2- SERIAL - PROGRAMA DE COMUNICAÇÃO SERIAL

Tabela 2.4 - Mapeamanto dos programas nos módulos.

Além desses programas existe uma série de programas de teste que foram utilizados nas montagens experimentais e que serão apresentados no Capítulo 4 e descritos no Anexo III.

2.4.2. *Software* de Geração e Visualização de Trajetórias.

O PC atua no protótipo tanto de modo iterativo, emulando um terminal que se comunica com a UCJ, como de modo *off-line*, quando são executados os programas que têm por finalidade a geração e visualização das trajetórias.

Existem duas estratégias para a obtenção dos pontos da trajetória a ser descrita pelo robô: na primeira, os pontos são obtidos a partir da execução de um programa em Pascal e a evolução da trajetória pode ser acompanhada por um programa de visualização. Na outra estratégia, adotada para o protótipo, os pontos são obtidos pela execução de um programa que define os pontos a serem deslocados e de outro programa que define o perfil de velocidade a ser seguido pela junta.

Estratégia 1: Os pontos da trajetória a ser descrita são obtidos a partir de um *software* [10], elaborado em Pascal, que permite calcular o posicionamento de cada junta, a partir da visualização do robô.

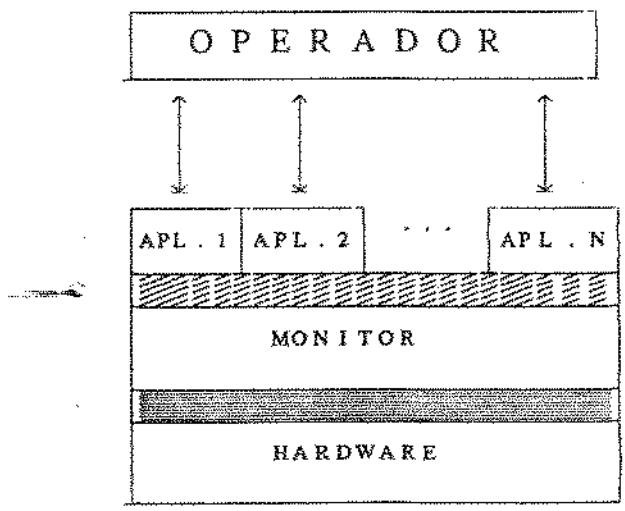
Este programa gera um arquivo de dados que é tratado por um *software* de interpolação e filtragem, de maneira a definir os pontos que serão enviados às UCJ.

Após a definição dos pontos da trajetória, outro programa, o VISUROBO, em que se estabelece um ambiente integrado entre o operador e o robô, apresenta uma visualização gráfica de três vistas do robô. Com isso é possível verificar a evolução de uma trajetória, viabilizando os testes de colisão.

Estratégia 2: Os pontos da trajetória são obtidos pela execução do programa DESLOC e o perfil de velocidade é definido a partir do programa VELOC. Em seguida, o PC executa o programa PCPLOT, para emular um terminal de modo que se possa interagir com a UCJ, através do programa Monitor. Os programas DESLOC e VELOC assim com a trajetória descrita estão no Anexo IV.

3.4.3. Estrutura de *software* do módulo de controle

A partir dos pontos da trajetória a ser executada, obtidos dos programas executados no módulo PC, o módulo de controle deve prover mecanismos para controlar o movimento de cada junta. Para tanto, além do *hardware*, descrito no item 2.3, é definida uma estrutura de *software*. Esta estrutura é modular, conforme mostra a Figura 2.3.



Interface monitor-hardware

Interface aplicativo-monitor

Figura 2.3 - Estrutura do software no módulo de controle.

Na Figura 2.3 pode-se observar que o programa Monitor opera como um elemento de medição entre o *hardware* e os aplicativos, possuindo ele mesmo duas interfaces: uma com o elementos que constituem o *hardware* e outra com os programas aplicativos.

Na interface com o *hardware*, o Monitor provê um conjunto de rotinas gerenciam os recursos físicos da máquina tais como espaço disponível em memória, leitura e escrita nos registros e nos dispositivos programáveis da placa.

A interface com os aplicativos estabelece um conjunto de rotinas que efetuam a ativação e interconexão entre os módulos aplicativos necessários para a execução das tarefas especificadas. Além disso, é oferecido ao operador um conjunto básico de comandos que permitem a interação do mesmo com o sistema.

A seguir são descritos os programas referenciados na Tabela 2.4.

2.4.4. Programa Monitor

O programa Monitor executa as seguintes tarefas:

a) gerenciamento dos recursos do *hardware*;

b) ativação e interconexão dos programas aplicativos;

c) estabelecimento de uma interface de comandos ao operador do sistema.

Em função destes tarefas, o Monitor pode ser dividido em 3 partes:

* gerenciador de recursos *hardware*;

* gerenciador de aplicativos;

* interface com operador.

Cada uma destas partes será descrita a seguir, destacando-se seus principais aspectos funcionais, operacionais e de implementação.

2.4.4.1. Gerenciador de recursos *hardware*

Este bloco reúne um conjunto de rotinas escritas em Assembler 8086/88 desenvolvidos a partir do sistema de desenvolvimento da INTEL, o SDK-86 [5]. Esta rotinas utilizam o primeiro segmento de 1 kbytes da memória RAM para preenchimento dos vetores de interrupção e para alocação do conteúdo de sua pilha de instruções.

Também são efetuados os procedimentos de inicialização do sistema através da configuração dos dispositivos programáveis. Desse modo, cria-se um ambiente operacional que permite a execução das rotinas do gerenciador de aplicativos e da interface de operador.

Este bloco está totalmente disponível e em funcionamento no protótipo.

2.4.4.2. Gerenciador de aplicativos

Este bloco provê mecanismos de alocação, ativação e interconexão entre os aplicativos do sistema. Para tanto, utiliza-se uma estrutura de comunicação baseada na troca de mensagens. Esta troca é feita por máquina de eventos, onde a cada mensagem gerada ou recebida por um módulo *software*, deve-se gerar uma interrupção especificada, fazendo com que uma seqüência de atividades se processe.

Esta mensagens são padronizadas, sendo definidos o número de bytes, os campos de endereço, controle e dados, assim como a especificação dos possíveis valores desses campos.

A função do Monitor é, a partir da existência de uma mensagem, analisar seu conteúdo e tomar as decisões necessárias a seu encaminhamento, que pode ser, por exemplo, a ativação de algum outro módulo da mesma unidade ou a transmissão de uma mensagem a outra unidade do sistema.

Este bloco encontra-se em desenvolvimento em linguagem C, com algumas das funcionalidades já testadas e outras em fase de especificação.

2.4.4.3. Interface com operador

A interface com o operador possibilita mecanismos de interação com o sistema para ativação de programas e acesso a dispositivos programáveis e memórias através de comandos básicos que podem ser combinados em instruções mais complexas conforme a necessidade.

Os comandos desta interface utilizam as rotinas do bloco Gerenciador de Recursos Hardware para acessar os recursos da placa e possuem seu formato especificado pela INTEL. Os comandos são os seguintes:

* D (Display) - Mostra o conteúdo de uma faixa de endereço de memória.

* G (Go) - Executa um programa residente em memória. Este comando é utilizado na ativação de um programa.

* I (Input) - Lê o conteúdo de um registro da placa.

* M (Move) - Transfere o conteúdo de um bloco de memória de uma posição para outra especificada.

* N (Next) - Permite a execução de um programa passo-a-passo. Este comando é bastante útil na fase de depuração de um programa.

* O (Output) - Escreve um dado em um determinado registro de escrita da placa.

* R (Read) - Lê o conteúdo da porta serial conectada ao terminal ou PC. Este comando é utilizado na carga de programas.

* S (Substitute) - Altera o conteúdo de uma determinada posição de memória.

* X - Mostra o conteúdo de todos os registradores da CPU.

2.4.5 - Algoritmo de Controle

Conforme mostrado anteriormente, o controle de juntas é efetuado pela execução de um algoritmo de controle na UCJ. Levando-se em conta fatores como simplicidade de implementação e larga escala de uso, a escolha do tipo de algoritmo recaiu sobre o controlador *on-off*.

Neste tipo de controlador, o sinal de controle é enviado apenas quando um limiar de erro máximo de posicionamento é ultrapassado. Este erro resulta da comparação entre os pontos da trajetória real e os pontos da trajetória desejada. Uma discussão mais aprofundada a respeito deste controlador é feita no Capítulo 3.

O programa que implementa este algoritmo é o CONTROLE_ON_OFF. Este programa, executado a partir da ativação do monitor, realiza basicamente as seguintes funções:

a) Leitura dos pontos referentes à trajetória desejada;

b) Leitura dos pontos referentes à trajetória real;

c) Comparação entre ambos para a determinação do erro;

d) Envio de sinal de controle ao módulo de acionamento.

A descrição completa deste programa está no Capítulo 3 e a listagem do mesmo encontra-se no Anexo II.

2.4.6. Programa de Comunicação Serial (SERIAL)

A função deste programa é realizar a interface entre a UCS e a UCJ. Esta interface opera através de um protocolo assíncrono por interrupção byte a byte.

Protocolo de trocas de mensagens

O Capítulo 1 descreve um protocolo baseado na troca de mensagens, onde sinais da interface são empregados para sinalizar mensagens a serem transmitidas e gerar interrupção.

A interrupção, que ocorre tanto na transmissão como na recepção de dados, está associada com a transição do sinal Request To Send [4] da unidade transmissora, indicando que esta terminou a transmissão de uma dada mensagem. Assim o dado enviado já pode ser analisado pela unidade receptora.

Existem três tipos básicos de mensagens que trafegam na interface entre UCS e UCJ: mensagem de dados, mensagem de supervisão e mensagem de alarme. As mensagens de dados e de supervisão possuem dois bytes, sendo que o 1º indica o tipo de mensagem e o 2º traz o conteúdo referente à mensagem que está sendo enviada. Já a mensagem de alarme possui apenas um byte, sendo tratada como urgente pelo protocolo.

• Mensagem de dados

Este tipo de mensagem ocorre quando a UCS envia pontos referentes a uma dada trajetória para uma UCJ. Seu 2º byte indica, num valor até 255, o deslocamento desejado para a junta.

• Mensagem de supervisão

As mensagens de supervisão são trocadas entre UCJ e UCS com a finalidade de *handshaking*, ou seja, iniciar e finalizar trajetórias, verificar condições de operação fornecer informações necessárias ao funcionamento normal do sistema.

Para a supervisão são definidas as seguintes mensagens:

a) Mensagem de início de trajetória

Mensagem originada na UCS com a finalidade de indicar às UCJ o início e um ciclo de transmissão de dados referentes a uma trajetória;

b) Mensagem de solicitação à UCJ de posição atual

Mensagem originada na UCS solicitando a cada UCJ sua posição atual;

c) Mensagem de indicação de junta na posição inicial (*top* 0)

Mensagem que a UCJ envia à UCS para indicar a ocorrência de interrupção originada pelo pista *top* 0 do *encoder*, indicando que a junta atingiu sua posição de equilíbrio.

d) Mensagem de alinhamento

Solicitação que a UCS faz às UCJ para que estas se preparem para o início da sequência de alinhamento para a posição inicial.

e) Mensagem de unidade operacional

Mensagem bidirecional, onde a unidade transmissora indica à unidade receptora que está pronta para operar.

• Mensagem de alarme

Apenas a mensagem de HALT, de um byte, bidirecional, e que provoca uma interrupção de prioridade máxima no processamento está definida para a fase atual do protocolo.

No caso da UCJ, esta envia uma mensagem de HALT à UCS quando da ocorrência de um erro crítico. Conforme será detalhado no Capítulo 3, o erro crítico é uma proteção *software* que ocorre antes da violação do volume de trabalho da junta.

O protocolo acima descrito encontra-se especificado e em fase de testes, sendo necessária a utilização de uma UCS para sua completa validação.

A UCS, por sua vez encontra-se também em fase final de testes, não sendo possível, portanto, sua utilização nos testes do protótipo. Deste modo, foi necessária uma adaptação do protocolo de comunicação para que os ensaios e medidas de desempenho pudessam ser efetuados.

Protocolo simplificado de trocas de mensagens

A adaptação do protocolo de trocas de mensagens para funcionamento apenas na UCJ levou a uma simplificação da estrutura das mensagens. Neste protocolo as mensagens possuem apenas um byte de extensão e a diferenciação entre os tipos de mensagens é através do bit mais significativo.

Isto acarreta uma sobrecarga de processamento pois as interrupções passam ocorrer a cada byte recebido ou transmitido e o processador tem que fazer uma análise a nível de bits para tomar decisões. Isto não invalida no entanto a análise dos resultados, pelo contrário, pois os tempos obtidos passam a ser o *upper-bound* do sistema. O protocolo completo seria executado em tempos menores do que os tempos envolvidos na execução do protocolo simplificado.

São definidas as mensagens de dados e supervisão, sendo que a mensagem de HALT passa a ser um tipo especial de mensagem de supervisão.

- Mensagem de dados

A mensagem de dados possui o formato dado pela Figura 2.4

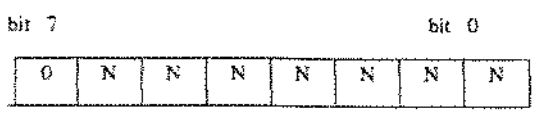


Figura 2.4 - Formato da mensagem de dados

Onde o bit 7 em nível 0 indica o tipo de mensagem e os demais bits indicam o valor do ponto da trajetória.

Nas mensagens de supervisão o bit 7 permanece em 1, e os tipos de mensagem são mostrados na Figura 2.5.

NNNN=00H - consulta
01H - início trajetória
02H - operacional
03H - solicita posição
04H - posição atual
FFH - Halt

Figura 2.5 - Tipos de mensagens de supervisão no protocolo simplificado.

Programa SERIAL

O programa SERIAL é ativado a partir do programa Monitor e possui uma sequência de funcionamento segundo as etapas descritas a seguir:

a) Inicialização

Nos procedimentos iniciais é feita a programação do controlador serial (SCC-8530), do controlador de interrupções (PIC-8259) e são inicializados os *buffers* de transmissão e recepção de mensagens.

Em seguida, são preenchidos os vetores de interrupção, que contêm os endereços de execução das rotinas de tratamento de interrupção.

b) Habilita interrupção

Após os procedimentos iniciais, a interrupção é habilitada e o processador fica aguardando a ocorrência de alguma transmissão ou recepção de dados que provoque a interrupção do mesmo.

c) Ocorre interrupção

O tratamento de uma interrupção envolve duas fases de eventos. A primeira acontece através de trocas de sinais de *hardware*, e é descrita a seguir.

Uma interrupção na linha serial pode ser causada pela transmissão de um dado pelo controlador de linha serial, quando seu *buffer* interno de transmissão fica vazio, ou ser de recepção, quando o *buffer* de recepção do controlador 8530 está com dado disponível ou ainda quando ocorre alteração nos sinais de interface RTS e CTS.

Uma transição no sinal RTS (sinal de saída) da unidade transmissora indica que esta terminou a transmissão de uma dada mensagem. Do lado da unidade receptora, a transição do sinal CTS (sinal de entrada) indica que esta pode transmitir sua mensagem.

Em qualquer um desses casos, o 8259 (PIC) gera um pedido ao processador, que reconhece o pedido enviando ao 8259 um sinal que faz a leitura do vetor de interrupção.

Aqui inicia-se a fase de eventos *software*. De posse do vetor de interrupção, o processador identifica na memória o endereço físico de execução da rotina de tratamento de interrupção. Em seguida, o processador guarda na pilha o endereço de execução corrente e passa a executar as instruções contidas na rotina de interrupção.

d) Tratamento de interrupção

Inicialmente o processador lê um registro do 8530 para identificar o tipo de interrupção ocorrida. Em seguida é feito o tratamento como se segue:

i) Interrupção de transmissão: o processador limpa a interrupção, acessa o *buffer* de transmissão que contém as outras mensagens a serem transmitidas e envia o próximo byte pela linha serial.

ii) Interrupção de recepção: o processador limpa a interrupção de reecepção, lê o byte disponível, identifica o tipo de mensagem recebida para tomar uma decisão sobre a necessidade de uma resposta a esta mensagem e escreve o valor recebido em um *buffer* de recepção.

iii) Interrupção de mudança de sinais de interface: neste caso o processador sinaliza a ocorrência alterando o valor de uma variável e limpa a interrupção.

Como pode-se observar o programa SERIAL trabalha basicamente em função da ocorrência de interrupções, permanecendo em estado de espera enquanto nenhuma interrupção ocorre.

Existe no protótipo uma versão otimizada deste programa com as principais funcionalidades, cujo resultado é analisado no Capítulo 4, onde discute-se o impacto da comunicação serial no desempenho da arquitetura distribuída. No Anexo II são apresentadas as rotinas de tratamento de interrupção.

2.5 - CONCLUSÃO

Neste capítulo apresentou uma descrição detalhada da implementação do supervisor de controle, enfocando inicialmente o *hardware* do mesmo. Foram descritas a UCJ e a UAM, apresentando-se suas características e peculiaridades. Os esquemas elétricos estão no Anexo I.

Em seguida foi feita uma descrição da estrutura do *software*, onde foi destacada a importância do programa Monitor como gerenciador e alocador de aplicativos. Os programas aplicativos de controle e comunicação serial foram descritos na sequência, sendo que o *software* de controle terá uma descrição completa no Capítulo 3.

CAPITULO 3

ESTUDO E CONCEPÇÃO DE UM CONTROLADOR DE POSIÇÃO IMPLEMENT NA UCJ

3.1. INTRODUÇÃO

Este capítulo faz uma abordagem do controle de juntas para a arquitetura utilizada, enfocando o emprego do controlador do tipo *on-off* na implementação do protótipo descrito no Capítulo 2.

Para esta análise são apresentadas duas soluções para a implementação do controlador. O item 3.4 descreve uma solução em *hardware* utilizando componentes programáveis enquanto que o item 3.5 descreve uma solução em *software*, conforme implementado na UCJ.

3.2 - CONTROLE DE JUNTAS

A arquitetura hierarquizada para o supervisor prevê a utilização de uma estrutura de controle para cada junta a ser controlada, realizando as seguintes funções:

a) comparação entre a posição real e a trajetória desejada e teste de erro mínimo (variável);
b) tratamento do sinal de erro através de um algoritmo específico que executa a correção do posicionamento dentro de uma margem de tolerância especificada;
c) envio de um sinal de controle para a estrutura de acionamento do motor em três estados (horário, anti-horário e repouso).

A estrutura de controle é representada na Fig. 3.1.

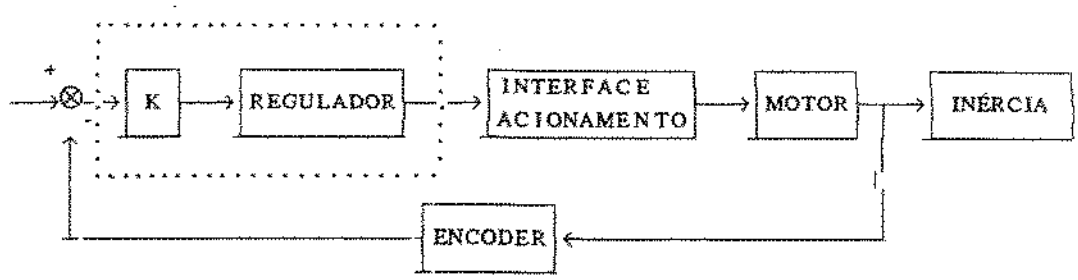


Figura 4.1 - Estrutura de Controle de 1 junta.

Onde:

K: constante do sistema;

Regulador: determina a ação de controle em função do erro de posicionamento;

Interface de acionamento: comanda o acionamento da junta a partir do sinal de controle;

Motor: representa o motor acoplado com a transmissão;

I - Inércia: representa os acoplamentos dinâmicos das demais juntas com a junta em questão;

Encoder: elemento de transdução que informa através de sinais elétricos a posição real da junta no sistema.

A linha pontilhada indica os blocos que são objetos de implementação dos itens 3.4 e 3.5.

3.3 - ALGORITMOS DE CONTROLE

3.3.1. Conceito

O controle adequado de um robô requer uma mecânica de precisão, um mecanismo de acionamento específico à aplicação do robô, a definição correta do volume de trabalho e um algoritmo que garanta movimentos suaves e estabilidade ao robô. Dentre os reguladores que apresentam grande eficiência e implementação relativamente simples, destaca-se o controle não-linear em três estados do tipo *on-off*. Tais controladores são frequentemente utilizados em automação industrial.

Tendo em vista que o objetivo deste trabalho é o de avaliar em termos práticos o desempenho de uma arquitetura de controle e não o de avaliar o comportamento de diversos algoritmos numa dada estrutura, e ainda, levando-se em conta sua simplicidade de implementação, optou-se pelo algoritmo do tipo *on-off*. Além disso, esta escolha possibilita uma comparação de desempenho entre diferentes processadores para um mesmo algoritmo, aumentando as possibilidades de escolha de processadores para a implementação de futuros supervisores de controle.

A seguir são discutidos algus dos pricipais aspectos deste tipo de controlador.

3.3.2 Controle do tipo *on-off*

O princípio de funcionamento do controlador do tipo *on-off* pode ser visualizado através da Figura 3.2, onde uma haste rígida girando em torno de um eixo fixo tem o comportamento análogo ao de um braço de um robô girando em torno de sua junta, sem considerarmos os acoplamentos mecânicos envolvidos e o atrito existente.

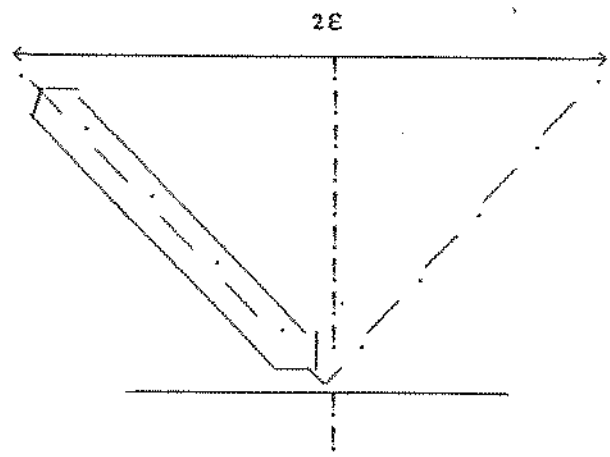


Figura 3.2 - Haste em equilíbrio instável.

Para manter a haste em equilíbrio, estabelece-se uma região de erro máximo (denominada zona morta), que não deve ser ultrapassada. Se for ultrapassada, uma força em sentido contrário ao do movimento é aplicada para tentar restituir a haste à sua posição de equilíbrio. No caso de utilizar-se um controlador analógico clássico a intensidade desta força seria proporcional ao deslocamento apresentado pela haste em relação ao eixo de rotação.

Na estratégia de controle adotada, no entanto, o objetivo é utilizar-se de um controlador não-linear em três estados, de tal forma que o uso de soluções total ou parcialmente implementadas em circuitos digitais possa ser viabilizada. Desse modo, a intensidade da força aplicada em sentido contrário ao deslocamento da haste é constante, caracterizando o controlador não-linear do tipo *on-off* ou *bang-bang*.

O diagrama de blocos de um controlador *on-off* ideal atuando numa junta está representado na Figura 3.3.

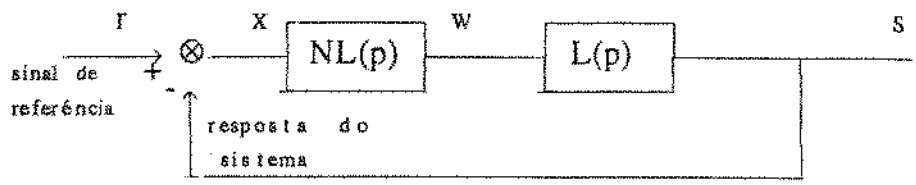


Figura 3.3 - Diagrama de blocos do controlador *on-off*.

Onde NL(p) representa a parte não-linear e L(p) representa a parte linear (motor).

A função **x** é o erro associado à diferença existente entre os sinais de referência **r** e a resposta do sistema **s**. O sinal **x** atua junto à parte não-linear do regulador, de modo que este responda através do sinal **w** com uma ação de correção. O sinal **w**, por sua vez, atua junto à parte linear do sistema (motor), cuja resposta é sinal **s**.

A definição do regulador está justamente na definição da função **w**, que no caso do controlador implementado no protótipo, possui três estados.

Estudos demonstram que a solução de utilizar-se uma zona morta tem o objetivo de eliminar um fenômeno característico de controladores sem zona morta, segundo o qual o sistema oscila indefinidamente em torno de um valor médio ao invés de tender a um valor final [15].

4.3.3. Implementação do algoritmo

Com base nos estudos desenvolvidos para o controlador não-linear do tipo *on-off*, foi implementado um algoritmo de controle com histerese conforme exposto na Figura 3.4.

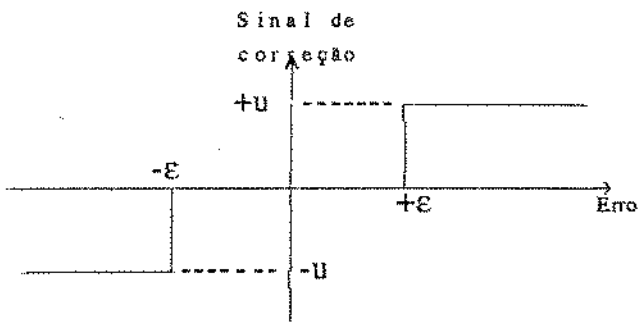


Figura 3.4 - Curva de comportamento do algoritmo *on-off*.

Na Figura 4.4 podemos identificar que, quando o erro de posicionamento tem um valor superior a +ε ou inferior a -ε, o sistema deve responder com o envio de um sinal de correção +u ou -u respectivamente. No sistema da haste girando em torno de um eixo fixo, estes eventos equivalem a dizer que a haste ultrapassou a zona-morta.

No caso em que o erro de posicionamento calculado pelo sistema possui um valor entre -ε e +ε, o sistema deve responder com um sinal nulo, sem ação de correção de trajetória.

Para cada um desses três valores possíveis para a resposta do sistema, é feita uma correspondência com a ação de controle a ser tomada conforme especificado na Tabela 3.1.

É importante ressaltar que a característica do sinal de controle depende não apenas do tipo de acionamento empregado na junta sob controle, como também das constantes de tempo envolvidas. Como será visto posteriormente, no robô utilizado nos testes, foi necessário o envio constante de muitos sinais de correção durante um tempo para que o sistema reagisse de um modo significativo.

VALOR DO ERRO	SINAL DE CORREÇÃO	AÇÃO DE CONTROLE
ERRO > +Ɛ	+U	FORÇA ROTAÇÃO EM SENTIDO HORÁRIO
ERRO < -Ɛ	-U	FORÇA ROTAÇÃO EM SENTIDO ANTI-HORÁRIO
-Ɛ<ERRO<+Ɛ	0	MANTÉM POSIÇÃO

Tabela 4.1 - Ações de controle.

Neste trabalho implementaremos o algoritmo do tipo *on-off*, sendo propostas duas estratégias para a implementação do mesmo. A primeira estratégia faz um tratamento totalmente digital e em *hardware*, onde tanto o tratamento dos sinais do *encoder* quanto o cálculo do erro e o envio do sinal são executados através de uma lógica residente em componente programável do tipo EPLD (Eraseble Programmable Logic Device). Esta proposta é abordada como uma solução teórica no item 3.4.

A segunda faz tratamento do erro através de um *software* que lê através de um contador o valor relativo à posição real ocupada pela junta, calcula o erro de posicionamento em relação à posição desejada e envia o sinal relativo à ação de controle que corrige este erro. Este tratamento foi adotado no protótipo sendo objeto dos procedimentos experimentais descritos no Capítulo 4.

4.4. CONTROLE POR *HARDWARE* [15]

No diagrama de blocos da estrutura de controle para uma junta, ilustrado na Figura 3.5, é definido um conjunto que envolve os blocos de Controle e Tratamento de Erro e que será objeto de estudo para uma solução em *hardware* digital.

As entradas deste conjunto são os sinais de encoder que retratam a trajetória real da junta e sinais que representam a trajetória desejada enquanto que as saídas são as ações de controle resultantes da execução do algoritmo *on-off*.

É importante observar que o equacionamento apresentado para os blocos possibilita que esta solução possa ser implementada de diversas maneiras, seja utilizando dispositivos programáveis, seja através do uso de portas lógicas comerciais.

Deste modo, é definido uma diagrama de blocos para esse conjunto, conforme ilustra a Figura 3.5.

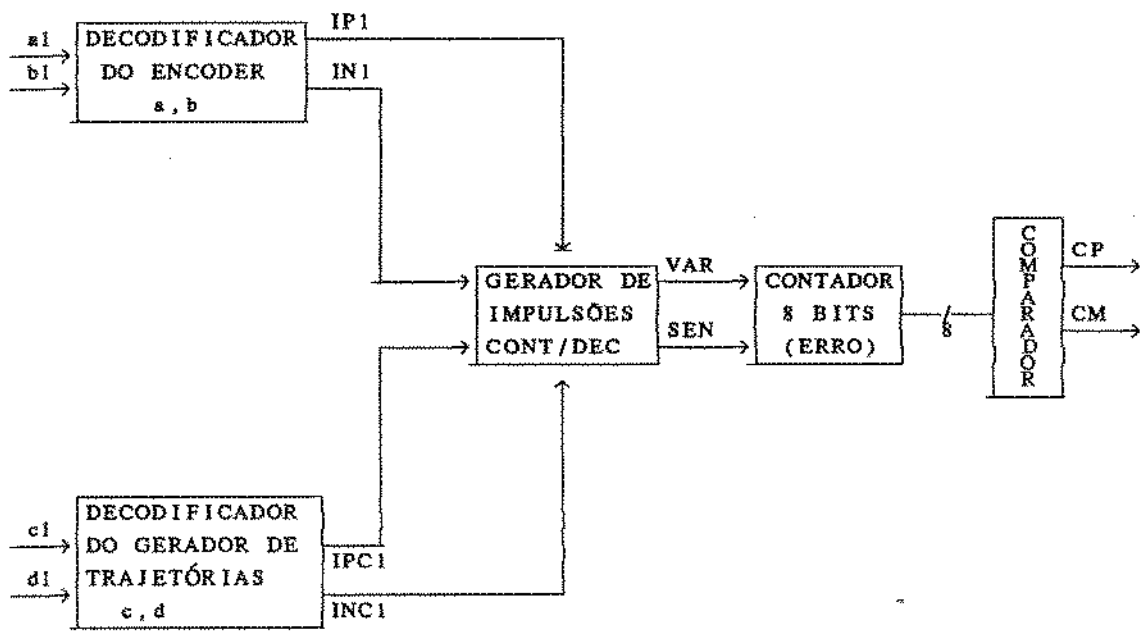


Figura 4.5 - Diagrama de blocos do controlador digital.

A seguir os elementos apresentados na Figura 4.5 são descritos em forma de seus blocos constituintes.

I- Bloco decodificador do sinal do *encoder*.

Neste bloco, os sinais oriundos do *encoder* são transformados em dois sinais de impulsão IP1 e IN1 representando, respectivamente, os sentidos de rotação positivo e negativo.

A informação sobre o total deslocado é obtida através da inserção de um sinal de relógio que efetua a amostragem.

II- Bloco decodificador de gerador de trajetória.

Neste bloco, a trajetória desejada proveniente da UCS e representada por duas pistas (c1 e d1), é transformada em dois sinais de impulsão IPC1 e INC1. Do mesmo modo que no bloco I, IPC1 representa o sentido de rotação positivo enquanto que INC1 representa o sentido de rotação negativo. Assim como no bloco I, existe também um relógio possibilita a obtenção do valor total do deslocamento.

III- Bloco gerador de pulsos de contagem e decontagem.

Os decodificadores dos sinais a,b e c,d fornecem, após tratamento lógico, quatro informações:

- IP1 e IN1 representando a posição real
- IPC1 e INC1 representando a posição desejada

A partir destas informações pode-se determinar o erro de posicionamento (posição desejada - posição real), considerando-se que:

- IP1 corresponde a uma rotação de +1/4 de volta.

- IN1 corresponde a uma rotação de -1/4 de volta.

- IPC1 corresponde a uma rotação da posição desejada de +1/4 de volta.

- INC1 corresponde a uma rotação da posição desejada de -1/4 de volta.

É interessante estabelecer relações lógicas entre as variáveis IP1, IN1, IPC1 e INC1, uma vez que a implementação dessas relações se dará em componentes programáveis que utilizam equações lógicas como arquivos fonte para montagem de seus circuitos.

Pode-se mostrar que as relações entre as variáveis IP1, IN1, IPC1, INC1 são representadas pelas Equações 3.1 a 3.4:

$$A = IP1 * INC1 \quad (3.1)$$

$$B = \overline{IP1} * IN1 * INC1 + IP1 * \overline{IPC1} * \overline{INC1} \quad (3.2)$$

$$C = \overline{IP1} * IN1 * IPC1 + IN1 * \overline{IPC1} * \overline{INC1} \quad (3.3)$$

$$D = \overline{IN1} * IPC1 \quad (3.4)$$

Onde :

- A ação A diminui o contador de erro de 2.
- A ação B diminui o contador de erro de 1.
- A ação C incrementa o contador de erro de 2.
- A ação D incrementa o contador de erro de 1.

IV- Bloco contador de erro

Este bloco pode ser implementado através de um circuito integrado comercial, por questão de simplicidade e por se tratar de uma solução em *hardware*.

O contador é incrementado ou decrementado segundo os valores das variáveis IPC1, INC1 IP1 e IN1. Para acioná-lo é necessário converter estes valores em direção e sentido. Para incrementar o contador de 1 deve ser enviado 1 impulsão e para incrementar de 2 basta enviar 2 impulsões.

A duração das impulsões deve ser de 1 período de relógio. Para incrementar de 2, devem ser enviados 2 períodos de relógio, segundo o circuito da Figura 3.6.

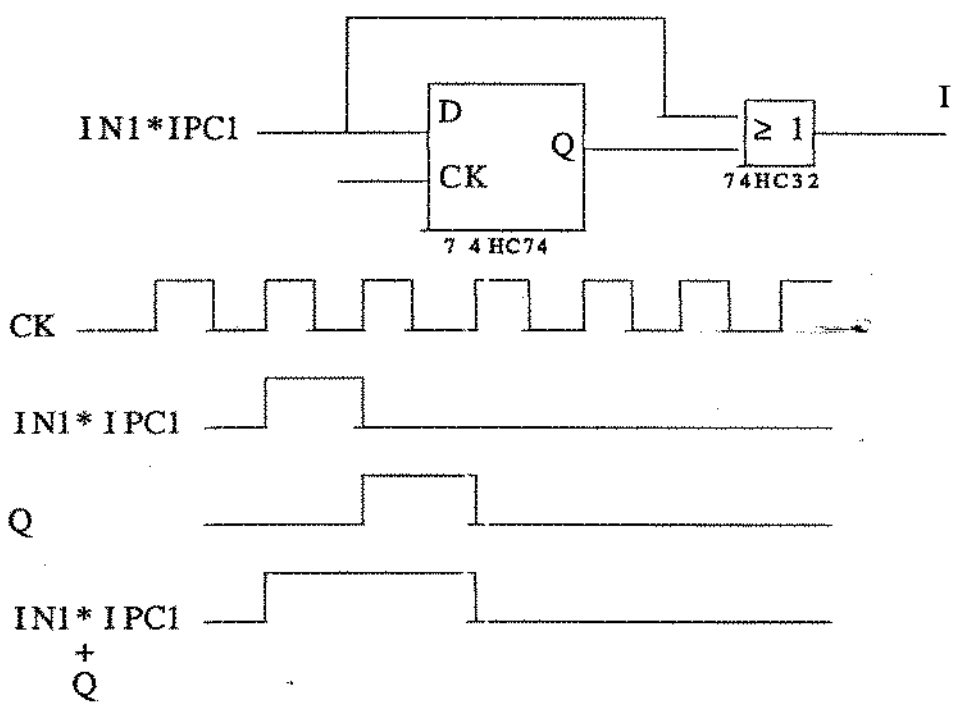


Figura 3.6 - Diagrama de tempo do circuito contador de erro.

V- Bloco Comparador

O controle de erro é feito de tal maneira que se a posição da junta ultrapassa um valor $+\varepsilon_{max}$ ou $-\varepsilon_{max}$, uma ação é tomada para a manutenção do equilíbrio. Estas relações estão representadas pela Equação 3.5:

$$\begin{aligned} &\varepsilon > + \varepsilon_{max} \text{ , executa ação 1.} \\ &\varepsilon < - \varepsilon_{max} \text{ , executa ação 2.} \\ &- \varepsilon_{max} < \varepsilon < + \varepsilon_{max} \text{ , não executa ação.} \end{aligned} \qquad (3.5)$$

O dimensionamento e a implementação deste bloco deve levar em conta as características do *encoder* utilizado no robô a ser controlado, uma vez que o erro máximo resultante da comparação entre a leitura do *encoder* e o ponto da trajetória desejada geralmente é tomado com um percentual do total de pontos deslocados pela junta.

Este erro pode ser escrito de forma binária através de uma palavra, conforme indica a Equação 3.6.

$$\varepsilon = a_{n-1} a_{n-2} \cdots a_1 a_0 = \sum_{i=0}^{n-1} a_i 2^i \qquad (3.6)$$

De posse das especificações do *encoder* e admitindo-se um determinado erro máximo, pode-se estabelecer as relações entre os coeficientes da palavra binária que representa o erro e as ações de controle a serem tomadas em função deste erro.

Para ilustra esta análise, podemos supor o uso de um *encoder* de 256 pontos e admitir um erro máximo associado de 3% (8 pontos).

Deste modo, o erro pode ser representado por 4 bits, sendo 1 de sinal e 3 referentes ao módulo, conforme representado na Equação 3.7.

$$\varepsilon = cba = c*2^2 + b*2^1 + a*2^0 \qquad (3.7)$$

Esta equação permite montar a Tabela 3.2a com os possíveis valores para o erro em função dos coeficientes cba e a Tabela 3.2b, que mostra o módulo do erro ($|\ \varepsilon\ |$) em função dos coeficientes kji. A partir destas tabelas, poder-se estabelecer as relações lógicas entre as variáveis de entrada que fornecem o valor do erro e as variáveis de saída que especificam as ações de controle tomadas.

ε	sinal	c	b	a
7	0	1	1	1
6	0	1	1	0
5	0	1	0	1
4	0	1	0	0
3	0	0	1	1
2	0	0	1	0
1	0	0	0	1
0	0	0	0	0
-1	1	1	1	1
-2	1	1	1	0
-3	1	1	0	1
-4	1	1	0	0
-5	1	0	1	1
-6	1	0	1	0
-7	1	0	0	1
-8	1	0	0	0

Tabela 4.2.a

sinal	k	j	i	$\lvert\varepsilon\rvert$
1	0	0	1	1
1	0	1	0	2
1	0	1	1	3
1	1	0	0	4
1	1	0	1	5
1	1	1	0	6
1	1	1	1	7
1	0	0	0	8

Tabela 4.2.b

As possibilidades contempladas pelas Tabelas 4.2a e 4.2b podem ser separadas para valores positivos e negativos do erro, como se segue nas Eq. 3.9 a 3.12. Nestas equações, além da simbologia já adotada ao longo deste trabalho são acrescentados os seguintes símbolos:

⊕ : indica a operação OU-EXCLUSIVO.

+ : indica a operação OU.

1º caso: erro positivo (ε = cba; sinal = 0; ε_{max} = ywz)

- $\varepsilon > \varepsilon_{max}$ (Ação 1) se:

$$\begin{aligned} &- (c=y)*(b=x)*(a>w) \quad \text{ou} \\ &- (c=y)*(b>x) \quad \text{ou} \\ &- (c>y) \end{aligned} \tag{3.9}$$

Estas relações podem ser escritas de outra maneira, conforme se segue:

$$\begin{aligned} \text{Ação 1} = &\overline{(\,c \oplus y\,)} * \overline{(\,b \oplus x\,)} * (\,a * \bar{w}\,) + \\ &+ \overline{(\,c \oplus y\,)} * (\,b * \bar{x}\,) + \\ &+ (\,c * \bar{y}\,) \end{aligned} \tag{3.10}$$

2º caso: erro negativo (sinal = 1)

ε = kji

- $\varepsilon < -\varepsilon_{max}$ (Ação 2) equivale a $|\varepsilon| > \varepsilon_{max}$, para sinal=1. Com isso pode-se escrever, analogamente ao 1º caso:

$$\begin{aligned} \text{Ação 2} = &\overline{(\,k \oplus y\,)} * \overline{(\,j \oplus x\,)} * (\,i * \bar{w}\,) + \\ &+ \overline{(\,k \oplus y\,)} * (\,b * x\,) + \\ &+ (\,k * \bar{y}\,) \end{aligned} \tag{3.11}$$

Em complemento de 2 pode-se escrever:

$$\begin{aligned} i &= a \\ j &= \bar{b} \oplus \bar{a} \\ k &= \bar{c} \oplus \overline{(\,b + a\,)} \end{aligned} \tag{3.12}$$

As Equações 3.9 a 3.12 são somente combinacionais, podendo ser implementadas em lógica programável numa PAL16L8, dando à solução um alto grau de integração.

Para a implementação de outro tipo de algoritmo, basta reescrever as equações e, se necessário, utilizar-se outro tipo de componente programável.

Uma outra possibilidade é a de utilizar-se um componente do tipo EPLD (Erasable Programmable Logic Device), de alta integração e com capacidade de executar inclusive lógicas sequenciais.

Este tipo de componente pode implementar todos os blocos descritos nessa solução, reduzindo muito a velocidade de execução dos algoritmos e aumentando a confiabilidade com uma integração maior.

4.5- CONTROLE POR *SOFTWARE*

No item 4.4 foi apresentada uma solução onde o algoritmo é executado totalmente em *hardware*. Apesar de possuir um elevado grau de portabilidade, pois existiria um conjunto de equações para cada tipo de algoritmo, esta solução pode implicar em grandes sistemas de equações e necessidade de utilização de componentes programáveis com maior capacidade, principalmente para algoritmos complexos.

Uma solução que reune portabilidade e simplicidade de implementação consiste na execução do algoritmo em *software* na UCJ. Neste caso com uma troca de EPROM pode-se alterar o controlador. Dentro desta filosofia foi elaborado um programa escrito em Assembler 8088, denominado CONTROLE_ON_OFF, que efetua o controle de uma junta segundo a técnica de controle *on-off*.

O CONTROLE_ON_OFF atua de maneira integrada ao *hardware* da UCJ, trocando informações com o mesmo por meio de *buffers* de memória e de portas de entrada e saída. A ativação do programa ocorre a partir do programa Monitor.

Na Figura 3.8 pode-se observar o diagrama que ilustra os blocos de implementação *hardware* e *software* que executam as funções do controlador.

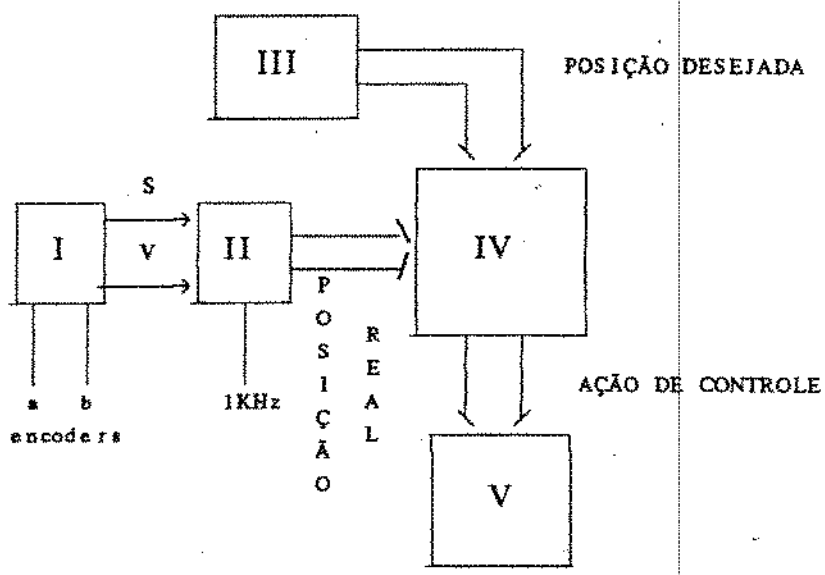


Figura 3.8 - Diagrama de blocos para a solução em *software*.

Neste diagrama os blocos I, II, III e V representam implementações no *hardware* da UCJ enquanto que o bloco IV é implementado em *software*. Os detalhes da implementação, tais como os componentes utilizados já foram apresentados no Capítulo 2.

• Bloco I: Este bloco é responsável pelo tratamento dos sinais de *encoder*, desde filtragem de ruídos, sincronização e transformação destes em dois sinais que informam diretamente ao Bloco II o sentido de rotação (s) e a variação (v).

• Bloco II: Este bloco recebe as informações de sentido e variação do Bloco I, e efetua a contagem dos pontos da posição real da junta. Um relógio de 1KHz faz a leitura do valor do contador para que este seja utilizado pelo programa no cálculo do erro de posicionamento.

• Bloco III: As informações a respeito da posição desejada estão armazenadas nos *buffers* de memória que este bloco representa. À medida de sua necessidade, o programa lê o valor referente à posição desejada para que possa ser comparado com o real.

• Bloco IV: Este bloco representa o programa sendo executado na CPU, que controla também os mecanismos de interrupção, acesso à memória e acesso aos dispositivos de entrada e saída. Os dados gerados pelos blocos II e III são lidos e a execução do algoritmo determina a ação de controle a ser tomada.

• Bloco V: A ação de controle é enviada ao mecanismo de acesso, que no caso do protótipo é a UAM (Unidade de Acionamento de Motor).

3.5.1- Princípio de funcionamento

Conforme descrito no item 4.3, o controlador *on-off* caracteriza-se por executar três tipos de ação de controle em função do resultado da comparação entre o sinal desejado e o sinal da posição real. Esta comparação ocorre periodicamente em função das constantes de tempo do sistema. A função erro, resultante da comparação, obedece o comportamento segundo a Figura 3.9.

No gráfico da Figura 3.9 pode-se observar a existência de uma zona morta de tamanho $2\varepsilon_{máx}$, em torno da qual nenhuma ação de controle será tomada. São definidos os limites $+\varepsilon_{máx}$ e $-\varepsilon_{máx}$ para a execução de ações de controle e os limites $+\varepsilon_{crít}$ e $-\varepsilon_{crít}$ para a ativação de alarme de fim de curso.

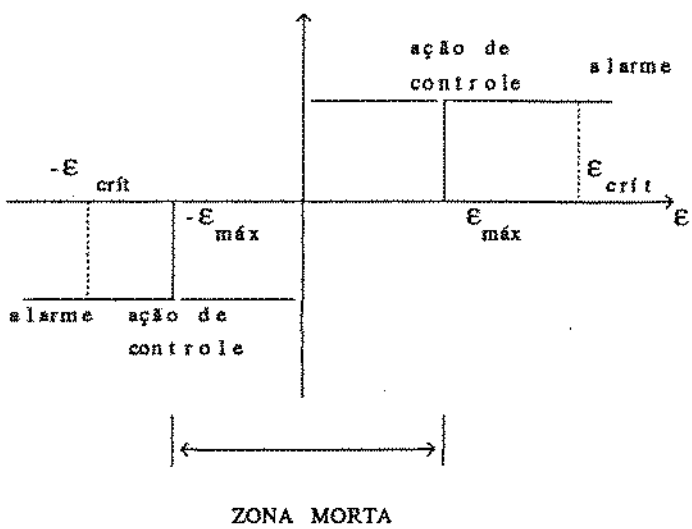


Figura 3.9 - Curva do algoritmo *on-off* implementado.

Tendo em vista a discrepância existente os tempos do processador V20, onde uma instrução é executada em média em 2 μs, e os tempos do sistema a ser controlado, da ordem de dezenas de milisegundos, torna-se necessária a adoção de um mecanismo que armazene as ações de controle enviadas pela UCJ, para que o sistema possa ter tempo para reagir.

Desse modo, foi incorporada uma temporização no controlador, que estabelece o número de vezes que o sinal de controle é enviado para a UAM dentro de uma mesma comparação. No programa CONTROLE_ON_OFF este número é um parâmetro do sistema.

O comportamento da função erro fica então descrito pelo seguinte sistema de inequações:

a) $-\varepsilon_{máx} < \varepsilon < +\varepsilon_{máx}$, mantém posição;

b) $\varepsilon < -\varepsilon_{máx}$, envia ação de correção no sentido contrário ao sentido indicado pelo sinal negativo;

c) $\varepsilon > +\varepsilon_{crit}$, envia ação de correção no sentido contrário ao sentido indicado pelo sinal positivo;

d) $\varepsilon > +\varepsilon_{crit}$, envia alarme de proteção para evitar o fim de curso.

ou

$\varepsilon < -\varepsilon_{crit}$

O CONTROLE_ON_OFF tem sua execução ativada a partir de um comando do monitor já descrito no Capítulo 3, permanecendo ativo enquanto o controlador permanece ligado. As operações executadas pelo programa são as seguintes:

a) leitura dos pontos referentes à trajetória desejada;

b) aquisição dos pontos referentes à posição real ocupada pela junta;

c) cálculo do erro de posicionamento a tratamento do mesmo de acordo com o gráfico da Figura 3.9;

d) envio de um comando de correção de trajetória ou alarme de proteção de fim de curso;

e) envio dos pontos referentes à posição real para um *buffer*, colocando-os à disposição da UCS para que esta efetue o controle de trajetórias;

3.5.2. Implementação e execução

Implementação

Um programa que implemente o algoritmo de controle de juntas de um supervisor deve ser executado em tempo real, de modo que aos atrasos inerentes ao sistema não sejam adicionados atrasos relativos à execução do algoritmo.

Além disso, no caso da UCJ, devem ser feitos acessos constantes aos dispositivos programáveis da placa, fato que provocaria, no caso de utilizar-se uma linguagem de alto nível, atrasos pelo uso adicional de primitivas.

Tendo em vista estes fatores, aliado à disponibilidade de ambiente de desenvolvimento, o programa CONTROLE_ON_OFF foi desenvolvido em Assembler 8088. Para um melhor entendimento, o CONTROLE_ON_OFF pode ser dividido em três partes:

- Inicialização
- Tratamento de interrupção
- Espera da próxima interrupção

a) Inicialização: O programa inicia com a interrupção da CPU desabilitada. O controlador de interrupções (8259) é programado com uma máscara que permite apenas o tratamento da interrupção do contador de 1KHz e da interrupção que aponta a ativação da pista do encoder de *top* 0. Em seguida o vetor de interrupção é preenchido.

Na sequência, as variáveis referentes ao número de pontos da trajetória a ser executada são carregadas nos respectivos apontadores.

Conforme veremos na execução, tanto estas variáveis como os pontos da trajetória desejada são inicializados antes da ativação do programa.

O valor referente ao 1º ponto da trajetória desejada é lido de um *buffer* de memória e enviado para a UAM através da porta de endereço 30H . Com estes procedimentos, a interrupção é habilitada e o programa passa a aguardar sua ocorrência.

b) Tratamento de interrupção

Quando o contador de 1KHz gera um pedido de interrupção ao 8259, inicia-se um procedimento idêntico àquele descrito no programa SERIAL entre a CPU e o controlador. Dentro da rotina de tratamento, o processador lê o conteúdo do registrador que informa a posição real ocupada pela junta. Este valor é comparado com valor desejado já lido do *buffer*.

Na comparação o processador verifica se o erro (a diferença entre os valores) vai provocar alguma ação de controle ou não. No caso do erro ultrapassar o valor crítico, é gerada uma interrupção não-mascarável no processador, de prioridade máxima.

No caso de se gerar uma ação de controle ($|\varepsilon_{máx}| < |\varepsilon| < |\varepsilon_{crít}|$), o valor de correção, cujo módulo é igual a $\varepsilon_{máx}$ e sinal é oposto ao sinal resultante da comparação, é enviado pela porta de endereço 30H tantas vezes quantas foram definidas inicialmente, para a UAM.

Após estes procedimentos o processador retorna ao programa principal.

c) Espera da próxima interrupção

Com a interrupção habilitada, o processador fica aguardando a próxima ocorrência até que todos os pontos da trajetória tenham sido lidos do *buffer*, enviados à UAM e comparados com as leituras dos *encoders*.

Caso o sistema a ser controlado possua uma constante de tempo incompatível com o tempo de 1 ms entre as interrupções, o programa provê uma parâmetro que define o número de interrupções que devem ocorrer entre duas leituras do *encoder*.

No caso de não haver haja mais pontos, o programa retorna ao programa Monitor.

Execução

O programa CONTROLE_ON_OFF pode ser executado de modo automático ou manual. O modo automático requer a integração com outros programas, taïs como o de geração automática de trajetórias [7] e o de modelamento cinemático inverso [1], e se encontra em fase de testes. Por este motivo, mostraremos a execução do programa em seu modo manual, já pronto e integrado ao Monitor, e cujos resultados serão mostrados no Capítulo 4.

Os parâmetros que podem ser definidos por substituição na memória são os seguintes:

- Números de pontos da trajetória a ser executada
- Pontos da trajetória a ser executada

- Valor de erro máximo
- Valor de erro crítico
- Valor inicial (no caso de não se dispor da 3ª pista de *encoder*)
- Número de interrupções de 1 KHz que devem ocorrer entre duas leituras
- Número de vezes que o sinal de controle é enviado para a UAM

Estes parâmetros são definidos a partir do comando S (substituição), antes da execução do programa CONTROLE_ON_OFF.

Em seguida, utilizando-se o comando G (Go) do Monitor, executa-se o programa. As ações de controle resultantes da execução assim como os pontos da trajetória real podem ser visualizados na memória ou ainda através de curvas, conforme pode ser visto no Capítulo 4.

3.6. CONCLUSÃO

Neste capítulo discutiu-se sobre o controle de juntas, com ênfase ao controlador do tipo *on-off*. Foram apresentadas também duas propostas para implementação do controlador na arquitetura distribuída.

A primeira proposta apresentou uma implementação totalmente em *hardware* , utilizando dispositivos programáveis (PAL), onde inclusive foi sugerida a utilização de uma EPLD que poderia implementar toda a solução em um único componente.

A segunda proposta apresentou uma implementação através de um *software* integrado à UCJ, sendo a solução adotada para o protótipo. Foram apresentados os programas desenvolvidos na implementação desta proposta, com destaque para o programa Monitor que propicia uma interface entre o operador do sistema e o equipamento de supervisão além do programa CONTROLE_ON_OFF, que efetua o controle da junta.

No Capítulo 4 são apresentados os ambientes de teste do protótipo, definidos para a solução adotada.

CAPÍTULO 4

APARATO EXPERIMENTAL E TESTES

4.1. INTRODUÇÃO

O objetivo deste capítulo é apresentar o aparato experimental montado para os testes de avaliação da arquitetura. Algumas das etapas de teste utilizaram um robô montado para fins didáticos, o PROTOROB1, que é descrito segundo seus aspectos construtivos e operacionais.

Para cada teste executado, são apresentados os seguintes itens:

- Objetivo
- Ambiente do teste
- Execução
- Comentários

4.2. MONTAGEM EXPERIMENTAL

Utilizando-se uma placa UCJ, uma placa UAM, dois microcomputadores da linha PC e um robô didático, o PROTOROB1, que será descrito no item 4.3, foram montados diversos ambientes de teste com os seguintes objetivos:

a) Verificação do comportamento da arquitetura de controle utilizada, identificando na prática seus pontos altos (simplicidade de concepção, modularidade e caráter distribuído) e principalmente suas limitações (possíveis "gargalos" e *overheads).*

b) Avaliação do desempenho do algoritmo de controle *on-off* atuando no controle do robô, sendo executado em um microprocessador da linha NEC V.20 8088, permitindo a comparação com o desempenho do microprocessador Z.80.

c) Avaliação da atuação do programa de geração de trajetórias em PC assim como o comportamento do robô PROTOROB1, visando sua utilização como robô didático.

Deve-se observar que, por ocasião da montagem dos ambientes de teste, a Unidade Central de Supervisão encontrava-se em fase final de desenvolvimento, não sendo possível sua utilização no protótipo. No entanto, as arquiteturas *hardware* e *software* da UCJ e da UCS são bastante semelhantes entre si e a interface de comunicação entre UCJ e UCS é testada e validada em uma das montagens.

Estes fatos permitem afirmar que os resultados obtidos pelo protótipo atual estarão bem próximos daqueles obtidos quando a UCS for integrada ao sistema.

Os testes estão estruturados de maneira que as lógicas básicas do sistema são validadas uma a uma de modo independente. Em seguida, são executados os testes mais complexos, que integram estas lógicas básicas, permitindo uma avaliação do comportamento do sistema como um todo.

Para a visualização *off-line* dos resultados dos testes são utilizados periféricos (terminais de vídeo e impressoras). Nas montagens em que são necessários aquisição e tratamento dos sinais que traduzem os deslocamentos das juntas do robô, foi utilizado um procedimento específico para este fim, baseado no trabalho de Souza [16]. Este procedimento será descrito no item 4.2.2.

A Figura 4.1 mostra o esquema completo da bancada de testes montada para os procedimentos experimentais.

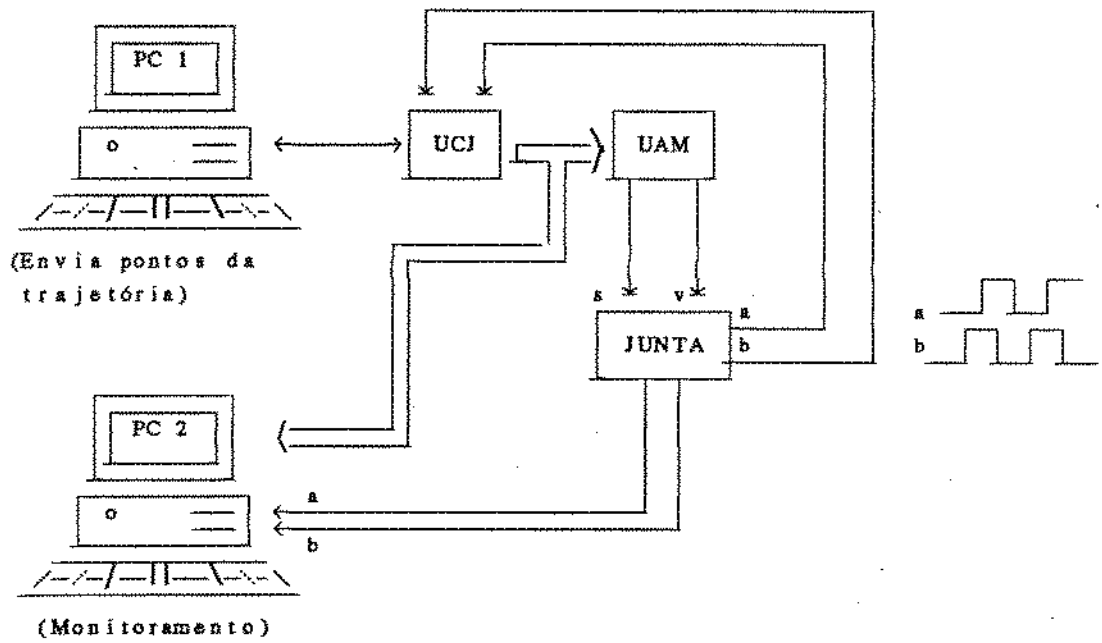


Figura 4.1 - Esquema de montagem da bancada

4.2.1. Descrição do robô PROTOROB1

4.2.1.1. Construção

O objetivo principal da utilização de um robô nos ensaios de laboratório foi o de possibilitar que a estrutura de controle em estudo fosse testada no acionamento de um mecanismo que representasse, guardadas as limitações inerentes ao protótipo, as condições de operação normal de um manipulador. Desse modo pode-se verificar a influência dos acoplamentos entre juntas no desempenho do algoritmo de controle.

Diante do fato de que os robôs comerciais em geral não permitem intervenções em seus algoritmos de controle de juntas, decidiu-se pela utilização de um robô desenvolvido no Laboratório de Automação e Robótica da Faculdade de Engenharia Mecânica da Unicamp.

Este robô, o PROTOROB 1, foi projetado com 3 graus de liberdade e interfaces de controle abertas de maneira a permitir testes com diferentes tipos de algoritmos e diferentes valores para seus parâmetros.

O PROTOROB1 foi concebido segundo uma filosofia de baixo custo, simplicidade de construção e características de acionamento e controle que permitem atuar em cada junta.

O PROTOROB1 possui uma base de alumínio e três juntas, cada qual acionada por um motor de passo. Os braços são construídos em tubos de PVC. A Figura 5.2 representa um croqui do robô.

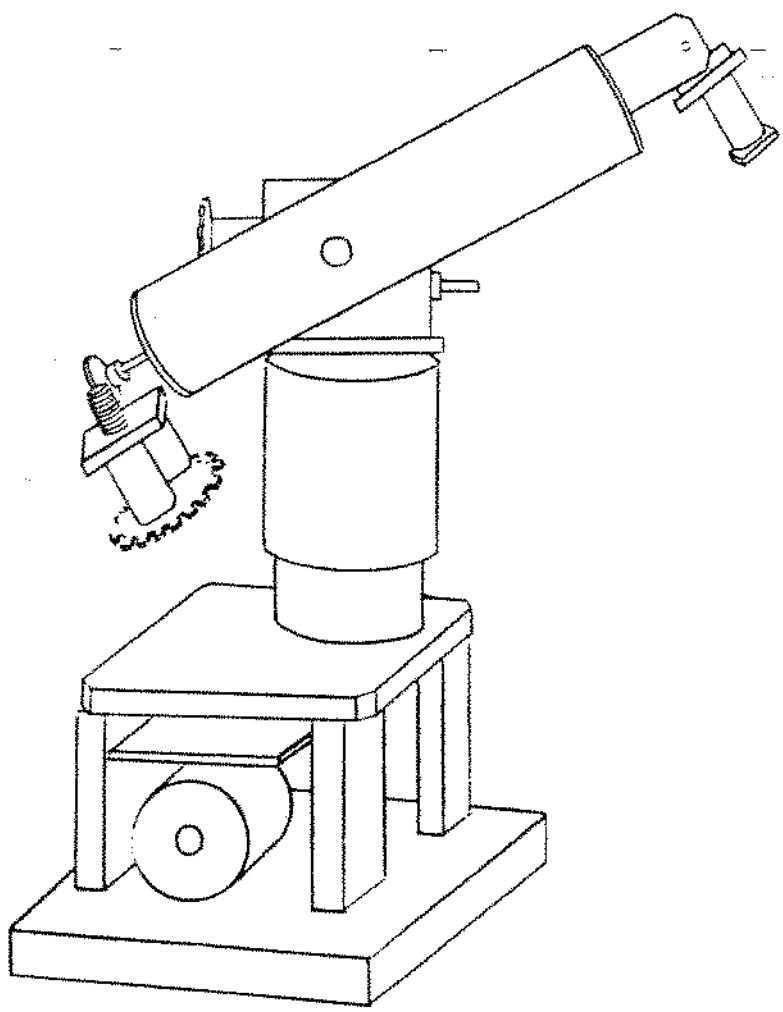


Figura 4 .2 - Robô PROTOROB1

4.2.1.2. Acionamento

O acionamento de cada junta do PROTOROB1 é feito por motor de passo, cujos parâmetros podem ser visualizadas na Tabela 4.1.

	JUNTA		
PARÂMETROS	1	2	3
POLOS	UNIPOLAR	BIPOLAR	BIPOLAR
FASES	4	2	2
ALIMENTAÇÃO	+ 12 V	+ 12 V	+ 12 V
RELAÇÃO DE PASSO	1,8°/PASSO	1°/PASSO	1°/PASSO
FREQUÊNCIA DE OPERAÇÃO	300 Hz	550 Hz	550 Hz

Tabela 4.1 - Parâmetros dos motores do PROTOROB1

O volume de trabalho é definido pelos cursos máximos de operação de cada junta, conforme a Tabela 4.2.

JUNTA	CURSO MÁXIMO
1	270 graus
2	270 graus
3	90 graus

Tabela 4.2 - Cursos máximos das juntas do PROTOROB1

4.2.1.3. Transdução

Segundo a estratégia de soluções de baixo custo, foi criado um dispositivo de transdução que atua de modo análogo ao de um encoder, empregado nas juntas 2 e 3.

O dispositivo (Figura 5.3) é composto por 2 conversores eletro-ópticos acoplados a um disco ranhurado e efetua a transformação da posição ocupada pela junta em dois sinais elétricos (em nível lógico TTL) defasados de 90°.

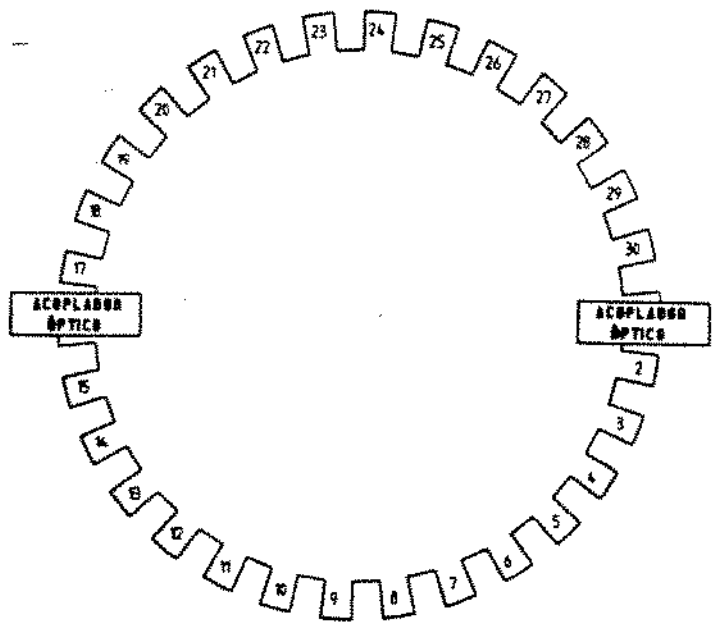


Figura 4.3 - Dispositivo de transdução.

O disco ranhurado possui 30 dentes, existindo uma relação de 12°/ dente.

4.2.1.4. Operação

O ponto da trajetória de referência é enviado pela UCJ à UAM para que esta transforme a palavra original nos sinais de sentido e variação. Estes sinais são enviados à junta do robô e os sinais dos *encoders* são lidos para que seja efetuada a comparação.

O mecanismo de leitura dos *encoders* ocorre da seguinte maneira: A saída de cada acoplador eletro-óptico é um sinal elétrico do tipo trem de pulsos, existindo uma diferença de fase de 90° entre as pistas. Esta diferença é necessária para que as saídas dos acopladores possuam um comportamento análogo ao de duas pistas de *encoder*, sendo obtida a partir da posição em que cada acoplador é fixado em relação ao disco ranhurado. As duas saídas são ligadas aos *drivers* de recepção da UCJ.

A partir dos sinais, a UCJ inicia a comparação entre a posição real e posição existente no arquivo de pontos de trajetória. Desta comparação e da execução do algoritmo de controle resulta a ação de controle.

A ação de controle pode assumir três valores (rotação de *n* pontos no sentido horário ou *n* pontos no sentido anti-horário ou manutenção da posição), sendo escrita no registrador de saída paralela da UCJ.

4.2.1.5. Modelos do Robô

Para definir-se uma trajetória para o robô, é necessário o cálculo do vetor de posição angular em diferentes configurações do mesmo. Torna-se então necessária a modelagem do robô, através dos modelos direto e cinemático inverso, definindo-se os vetores de posição e as matrizes de orientação.

O modelo direto tem por objetivo obter a posição e orientação de cada referencial local em relação ao referencial fixo, a partir da imposição dos ângulos de cada junta.

$$\underline{X} = f (\underline{\theta}) \qquad (4.1)$$

onde

$\underline{\theta} = (\theta_1, \theta_2, \theta_3)$: vetor 3 x 1 posições articulares

$\underline{X} = (x , y , z)$: vetor 3 x 1 posição da extremidade da junta 3.

O modelo cinemático inverso, por sua vez, consiste na obtenção dos ângulos de cada junta a partir da posição e orientação do referencial terminal, que contém um ponto qualquer da ferramenta em relação ao referencial fixo.

$$\underline{\theta} = f^{-1}(\underline{X}) \qquad (4.2)$$

Na Figura 5.4, os referenciais locais são orientados a partir do referencial fixo (inercial). Assim, não há necessidade de reorientação do eixo z de cada junta, simplificando a obtenção da posição e orientação dos vetores de cada referencial.

Para efeito de modelagem, são consideradas as juntas 2 e 3, uma vez que a junta 1 está na base do robô. Desse modo são definidos os seguintes elementos:

θ_i - Ângulo do eixo de rotação no referencial *i*;

v_i - Vetor local da posição da origem do referencial em relação ao referencial *i*;

$[R]_i$ - Matriz de orientação da rotação local do referencial *i*;

$[P]_i$ - Matriz de orientação do referencial *i* em relação ao referencial fixo.

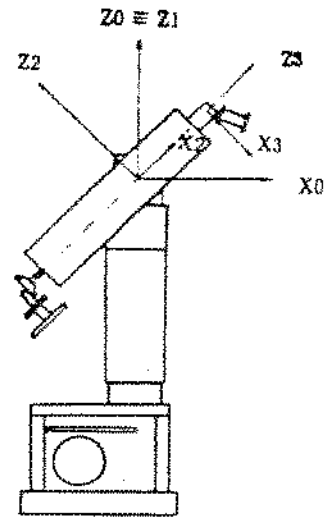


Figura 4.4 - Sistema de referência.

A partir da definição acima, pode-se obter o seguinte conjunto de vetores locais:

$$v_o = [\ 0\ 0\ 0\ ]^T$$

$$v_1 = [\ 0\ 1\ 0\ ]^T \qquad (4.3)$$

$$v_2 = [\ 1\ 0\ 0\ ]^T$$

Verifica-se que v_o define a posição da origem do primeiro referencial em relação ao referencial fixo, v_1 define a posição da origem do segundo referencial em relação ao primeiro e v_2 define a posição da origem do terceiro referencial em relação ao segundo.

Uma vez definidos os vetores locais de posição de cada referencial, é necessário estabelecer as matrizes de orientação local desses referenciais. Como no PROTOROB1 são consideradas as juntas 2 e 3, pode-se verificar que o referencial do primeiro corpo móvel (acoplado à junta 2) tem possibilidade de rotação em relação a z_1 enquanto que os referencial do segundo corpo móvel (acoplado à junta 3) tem a possibilidade de rotação em relação a y_2. Desse modo, pode-se escrever:

$$[\ R\ ]_1 = \begin{bmatrix} c_1 & -s_1 & 0 \\ s_1 & c_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$[\ R\ ]_2 = \begin{bmatrix} c_2 & 0 & s_2 \\ 0 & 1 & 0 \\ -s_2 & 0 & c_2 \end{bmatrix} \qquad (4.4)$$

$$[\,R\,]_3 = \begin{bmatrix} c_3 & 0 & s_3 \\ 0 & 1 & 0 \\ -s_3 & 0 & c_3 \end{bmatrix}$$

Onde $c_i = \cos\theta_i$ e $s_i = \text{sen}\,\theta_i$.

Definindo-se as matrizes de orientação de cada referencial móvel em relação ao referencial fixo, pode-se denotá-las como matrizes de passagem de cada referencial e representá-las da seguinte maneira:

$$[\,P\,]_i = [\,P\,]_{i-1} * [\,R\,]_i \quad i = 1, \tag{4.5}$$

Observe-se que, em relação ao ambiente de trabalho, foi considerado que a ação da gravidade está na direção do eixo z_o do referencial fixo e este é o referencial absoluto nesse ambiente.

De posse da posição e orientação de cada referencial, pode-se montar o modelo direto do manipulador. Assim:

$$\begin{aligned} O_0 &= [\,0\;0\;0\,]^T \\ O_{i+1} &= O_i + [\,P\,]_i * v_i \end{aligned} \tag{4.6}$$

Dessa forma, fica estabelecido o modelo direto do robô, pois, a partir de ângulos de junta pré-fixados, obtém-se a configuração final do mesmo.

Para a obtenção do modelo cinemático inverso, várias estratégias diferentes têm sido amplamente utilizadas. Adotando-se a mesma estratégia de *Fayan* [1], partiu-se para a construção do JACOBIANO da estrutura. Inicialmente estabelecem-se os vetores que qualificam a orientação da rotação local de cada referencial da seguinte maneira:

$$wl_i = [\,0\;0\;0\,]^T \tag{4.7}$$

$$wl_2 = [\ 0\ 1\ 0\ ]^T$$

$$wl_3 = [\ 0\ 1\ 0\ ]^T$$

Dessa forma, obtém-se os vetores que quantificam a orientação da rotação em relação ao referencial absoluto. Então:

$$we_i = [\ P\ ]_i * wl_i \qquad i = 1,2,3. \qquad (4.8)$$

O passo seguinte consiste em estabelecer os vetores de translação em relação ao referencial absoluto, definidos por:

$$te_i = we_i \times (O_{pf} - O_i) \qquad i = 1,2,3. \qquad (4.9)$$

O_{pf} - vetor posição do ponto onde pode ser acoplada uma ferramenta (extremidade do braço 3).

De posse de we_i e te_i, pode-se construir a matriz de transferência associada (JACOBIANO), com a seguinte configuração:

$$[\ J\ ] = \begin{bmatrix} te_{11} & te_{12} & te_{13} \\ te_{21} & te_{22} & te_{23} \\ te_{31} & te_{32} & te_{33} \\ we_{11} & we_{12} & we_{13} \\ we_{21} & we_{22} & we_{23} \\ we_{31} & we_{32} & we_{33} \end{bmatrix} \qquad (4.10)$$

Tendo-se pré-estabelecido a posição desejada da extremidade do braço 3, onde pode ser acoplada uma ferramenta, e o erro mínimo de posicionamento, executa-se um procedimento iterativo para obtenção dos ângulos das juntas.

Genericamente, este procedimento deve fazer uma comparação inicial entre a posição atual e a posição desejada. Caso não esteja dentro do erro previsto, calculam-se os vetores de rotação e translação em relação ao referencial absoluto, forma-se a matriz JACOBIANA, resolve-se o sistema resultante, obtendo-se novos valores angulares das articulações.

Com os novos valores dos ângulos, pode-se calcular os novos valores das matrizes de rotação, matrizes de passagem e posição de cada referencial para nova comparação da posição atual com a posição desejada, e assim, sucessivamente, até que seja satisfeita a condição de convergência imposta.

São definidos vários algoritmos para o procedimento de inversão cuja escolha deve levar em conta a aplicação final do sistema a ser controlado. Se o objetivo for a identificação de parâmetros, é mais apropriado utilizar-se algoritmos de convergência rápida. Para controle no espaço cartesiano, deve-se prever uma trajetória que minimize os deslocamentos das juntas. Se a aplicação for controle de esforços, o algoritmo deve diminuir o erro de posicionamento do elemento terminal do manipulador.

Dentre os principais algoritmos que podem contemplar estas aplicações, podemos citar [1]:

- Gauss
- Greville
- Miss V.1, Miss V.2 e Miss suavizado
- Moore-Penrose

A sequência iterativa para a obtenção do modelo geométrico direto e do modelo cinemático inverso pode ser visualizada através do fluxograma ilustrado na Figura 4.5.

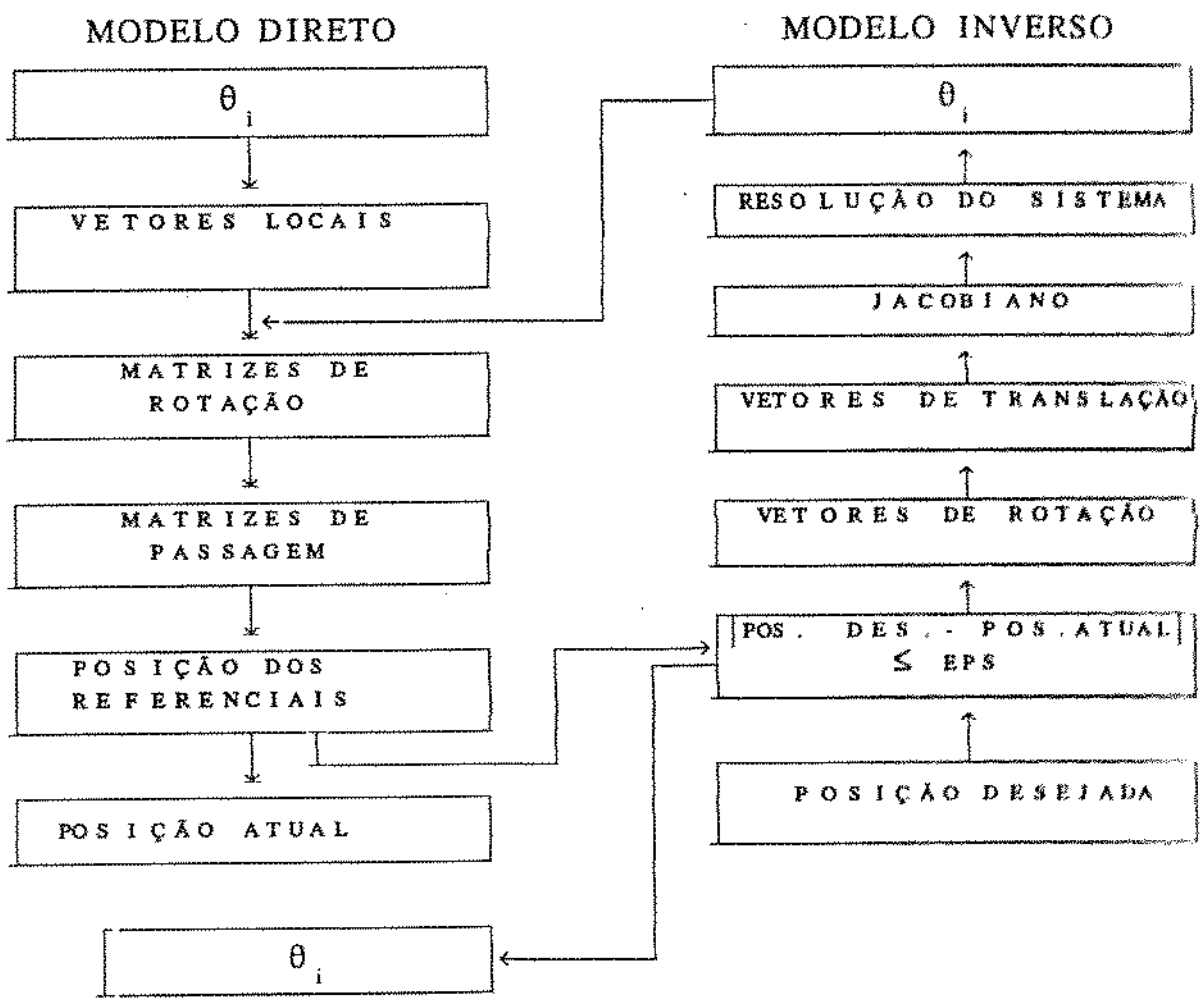


Figura 4.5 - Fluxograma integrado.

4.2.2. Procedimento para Aquisição e Tratamento dos Sinais [16]

O PROTOROB1 possui um mecanismo que gera sinais que indicam a posição ocupada por uma junta. Estes sinais têm o mesmo comportamento dos sinais de *encoder*, ou seja, são duas pistas em nível lógico TTL com uma defasagem nominal de 90° entre si.

Após amplificação e filtragem de ruídos efetuada pelo *driver* da UCJ, é feita a aquisição dos dados a partir de uma interface paralela de entrada e saída conectada diretamente ao barramento de dados de um microcomputador PC.

A interface é de 8 bits, sendo que os 6 menos significativos são reservados para 3 *encoders*, o bit 6 para controle de leitura e o bit mais significativo (bit 7) fornece uma base de tempo externa (Figura 5.6).

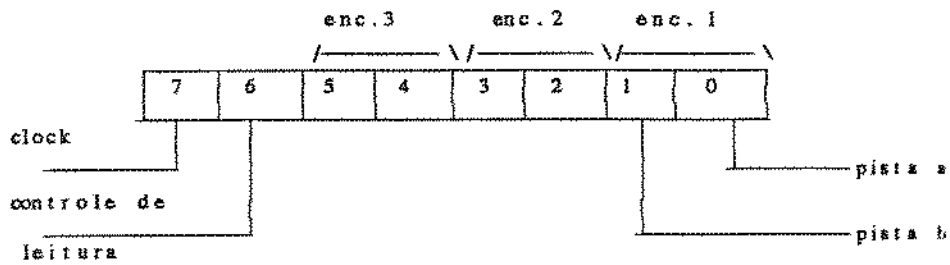


Figura 4.6 - Palavra de aquisição e tratamento dos sinais de encoder

A seqüência de aquisição e tratamento dos sinais é a seguinte:

a) os sinais são enviados ao PC via interface paralela;

b) um programa escrito em Assembly residente no PC lê os estados dos bits na borda de subida do *clock*, armazenando-os em uma região de memória;

c) após a aquisição, este mesmo arquivo é tratado por uma rotina que extrai a variação e o sentido de rotação de junta;

d) um outro programa escrito em linguagem C gera as curvas relativas ao movimento realizado.

4.3. TESTES

Nesta seção são apresentados os testes que foram executados para a validação da arquitetura. Para cada tipo de montagem são especificados o ambiente de teste assim como os resultados obtidos e uma análise sobre o impacto do resultado no comportamento do sistema como um todo.

4.3.1. Teste do algoritmo de controle

A Figura 4.7 representa esquematicamente a montagem do teste do algoritmo de controle.

• OBJETIVO: O objetivo desta etapa foi o de obter resultados que permitissem a avaliação do comportamento do programa de controle para o algoritmo do tipo *on-off* escrito e executado no ambiente do microprocessador NEC V.20.

• AMBIENTE: Nesta etapa utilizou-se uma UCJ acoplada a um PC executando um programa que emula terminal (Pcplot).A UCJ é equipada com uma EPROM onde foram gravados os programas Monitor e CONTROLE_ON_OFF cujas descrições detalhadas estão no Capítulo 3. Para medida do tempo de execução do algoritmo foi utilizado um Analisador Lógico HP 1650A. (Figura 4.7).

• EXECUÇÃO: Dois arquivos de pontos armazenados em RAM definem os dados utilizados na execução do algoritmo. O primeiro simula os pontos da trajetória real descrita pela junta, enquanto que o segundo traduz os pontos da trajetória desejada.

A trajetória desejada obedece um perfil de deslocamento e velocidade definidos pelos programas DESLOC e VELOC respectivamente.

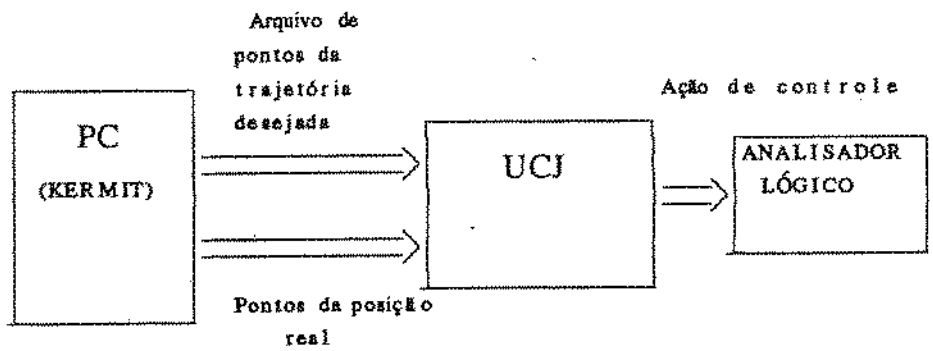


Figura 4.7 - Esquema do teste de algoritmo de controle

Através do comando de substituição S do Monitor, os pontos referentes à trajetória desejada foram preenchidos a partir da posição 0:1000H enquanto que os pontos que simulam a trajetória desejada foram colocados a partir da posição 0:1100H. As variáveis que determinam o erro máximo e o erro crítico foram inicializadas a partir da posição 0:1400H.

O comando G (Go) do Monitor inicia a execução do programa na posição F000:8000H. No início da execução, é enviado um comando de escrita no registro do contador dos *encoders* que funciona como um marcador de tempo, pois, no final do programa, outro comando é enviado, permitindo que o Analisador Lógico meça o tempo existente entre estes dois comandos de escrita, o que corresponde ao intervalo em que o processador permaneceu na execução do algoritmo.

As ações de controle tomadas em função do algoritmo são escritas na posição 0:1200H.

O parâmetro mais importante que foi obtido a partir desta montagem simples foi o tempo de execução do algoritmo, que pode ser visto na Tabela 4.3 para diferentes arquivos de trajetórias.

Apesar de não ter sido feito nesta etapa, o procedimento de aquisição e tratamento de pontos poderia ser usado para efeito de visualização de resultados.

NÚMERO DE PONTOS	TEMPO DE EXECUÇÃO
16	16.04 ms
32	32.15 ms
64	64.37 ms
128	129.02 ms

Tabela 4.3 - Tempo de execução do algoritmo

A partir destes resultados pode-se obter uma relação entre o número de pontos de uma trajetória o tempo de execução do algoritmo, para o programa CONTROLE_ON_OFF e para o processador NEC V.20 @ 8MHz.

$$\text{Tempo de execução (ms)} = n \times \text{número de pontos} \qquad (4.11)$$

Levando-se em conta o fato de que a interrupção cadencia o envio de pontos à UAM numa frequência de 1 KHz, portanto a cada 1 ms, este resultado permitiu concluir que o intervalo de tempo entre duas interrupções consecutivas é suficiente para o tratamento dos pontos segundo o algoritmo implementado.

As medidas até aqui obtidas não seriam totalmente conclusivas uma vez que não trazem a informação sobre quanto tempo o processador leva para efetuar o tratamento de um ponto segundo o comportamento do algoritmo *on-off*.

Partindo-se da estrutura do programa CONTROLE_ON_OFF, pode-se observar que o mesmo é dividido em procedimentos iniciais e rotina de atendimento de interrupção, onde o controle é efetivamente executado.

O tempo total de tratamento de um ponto da trajetória é dado pela soma dos tempos da inicialização e do tratamento da interrupção.

$$T_{total} = T_{inic} + T_{int} \qquad (4.13)$$

O tempo de inicialização é levado em conta apenas quando do tratamento do primeiro ponto assim que o programa é ativado, e pode ser facilmente obtido, medindo-se o tempo existente entre o marcador inicial do programa e o pedido de interrupção.

Este tempo tem valor fixo para um mesmo relógio e vale 56 μs. Assim:

$$T_{inic} = 56\ \mu s$$

Para obter T_{int}, bastou medir o intervalo existente entre o pedido de interupção e saída da rotina, quando é enviado um comando específico (EOI - End Of Interrupt) ao Controlador de Interrupções (PIC). O valor medido é de 73 μs para o tratamento do ponto, incluindo o envio de sinal de controle à UAM. Desse modo:

$$T_{int} = 73\ \mu s$$

Assim:

$$T_{total} = 129\ \mu s$$

Tendo-se disponível 1 ms para o tratamento, conclui-se que o processador dedica apenas 13% do seu processamento para tratar um ponto segundo o algoritmo *on-off*.

• COMENTÁRIOS

A partir dos resultados obtidos, pode-se concluir que o processador tem capacidade, com grande folga inclusive, de tratar os pontos oriundos dos *encoders* segundo o algoritmo implementado pelo programa CONTROLE_ON_OFF.

Para algoritmos mais complexos, haveria ainda uma reserva de tempo que permitiria sua execução neste mesmo sistema.

É importante observar também que a execução do algoritmo simula em memória os pontos obtidos das leituras dos *encoders*, e que, portanto, não foram levados em conta as constantes de tempo do sistema. Como será constatado, estas constantes incorporam atrasos ao sistema, tornando o tempo de processamento do algoritmo muito pequeno em relação ao controle como um todo.

4.3.2. Teste de acionamento do PROTOROB1

• OBJETIVO: Verificar o funcionamento do sistema de acionamento da junta de um robô com motor de passo a partir da UAM (Unidade de Acionamento doMotor). O teste também permitiu a validação da interface de acionamento da UCJ, de modo que com a mudança do tipo de acionemento da junta, a simples substituição da UAM permita a movimentação da mesma.

• AMBIENTE: Nesta montagem foi feito o acionamento em malha aberta do robô pela UAM (Unidade de Acionamento de Motor), a partir de um conjunto de pontos enviados pelo PC1, que representam uma trajetória (Figura 5.6). O PC2 faz a captura dos pontos dos *encoders* do robô, efetua o tratamento adequado e, de modo *off-line*, gera as curvas referentes às trajetórias executadas. A trajetória enviada à UAM também é tratada pelo PC2, permitindo uma comparação e análise do comportamento do sistema.

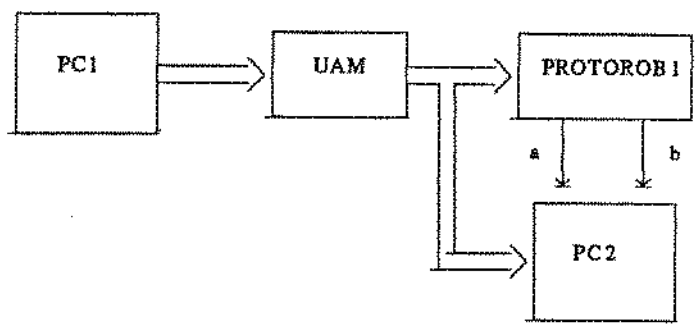


Figura 4.6 - Esquema do teste de acionamento do PROTOROB1

• EXECUÇÃO: Um programa escrito em BASIC executado no PC1 envia comandos de controle iguais ao máximo permitido e à metade do máximo permitidos pela interface do PC1, ou seja 3EH e 1FH e, no sentido contrário, 7EH e 5FH. Estes pontos são enviados em *loop*, com 3 frequências diferentes. A Tabela 4.4 indica as condições de cada ensaio.

ENSAIO	FREQ. AQUISIÇÃO	VALOR
1	20 KHz	7EH
2	10 KHz	7EH
3	5 KHz	7EH
4	20 KHz	3EH
5	5 KHz	3EH
6	20 KHz	5FH
7	5 KHz	5FH
8	20 KHz	1FH
9	5 KHz	1FH
10	10 KHz	7EH
11	20 KHz	7EH
12	5 KHz	3EH
13	10 KHz	3EH

Tabela 4.4 - Condições dos ensaios do acionamento do robô.

O deslocamento da junta é medido por um contador acoplado na saída do disco ranhurado e visualizado pelo *software de* geração de curvas [16]. As curvas da Figura 4.7 mostram as trajetórias obtidas nos ensaios nº 11 e 12 medidas nos sinais dos *encoders*.

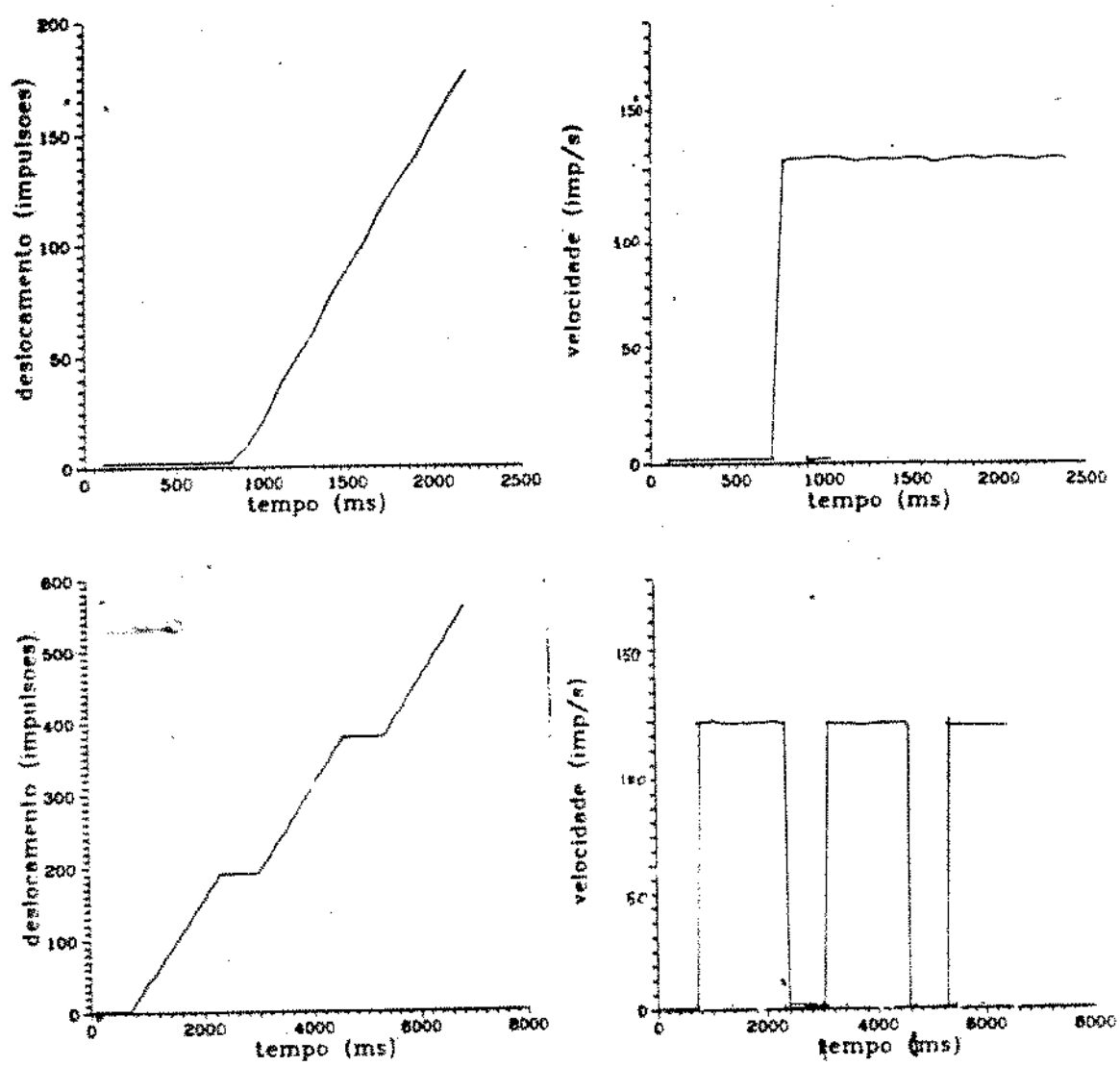


Figura 4.7 - Curvas das trajetórias executadas em malha aberta.

• COMENTÁRIOS: No ensaio nº 11 pode-se observar que o sistema está inicialmente parado, movimentando-se, em seguida, segundo uma trajetória crescente e linear, com velocidade constante. No ensaio nº 12, observa-se a trajetória crescente com dois patamares de deslocamento, onde o sistema permanece parado. Nos demais ensaios, foi observado que o sistema comportou-se satisfatoriamente, executando as mesmas trajetórias impostas pela UAM.

Como o objetivo deste experimento é apenas avaliar o sistema de acionamento do robô, os resultados podem ser considerados satisfatórios.

4.3.3. Teste do protocolo de comunicação UCJ-UCS

• OBJETIVO: Verificar o funcionamento do protocolo de comunicação simplificado definido a partir do protocolo completo de troca de mensagens especificado para a interface UCJ-UCS, enfocando principalmente os tempos envolvidos nas trocas de mensagens. Este teste trouxe um resultado real a respeito da existência de um ponto de estrangulamento na comunicação entre o nível 1 (UCJ) e o nível 2 (UCS) da arquitetura.

• AMBIENTE: Tendo em vista a impossibilidade de utilizar-se a UCS nas montagens experimentais, o teste de protocolo é executado apenas na UCJ com o estabelecimento de um *loop* remoto no canal A do controlador de comunicação serial (8530). O teste é realizado no canal A, uma vez que este é o canal especificado para a comunicação com a UCS e também porque o canal B que estabelece uma interface serial assíncrona com o PC pode ser utilizado para acompanhamento dos eventos através de leituras em memória e registradores da placa.

Assim são definidos dois *buffers* para o canal A, e as mensagens recebidas e transmitidas são lidas e escritas no *buffer* de transmissão e de recepção respectivamente.

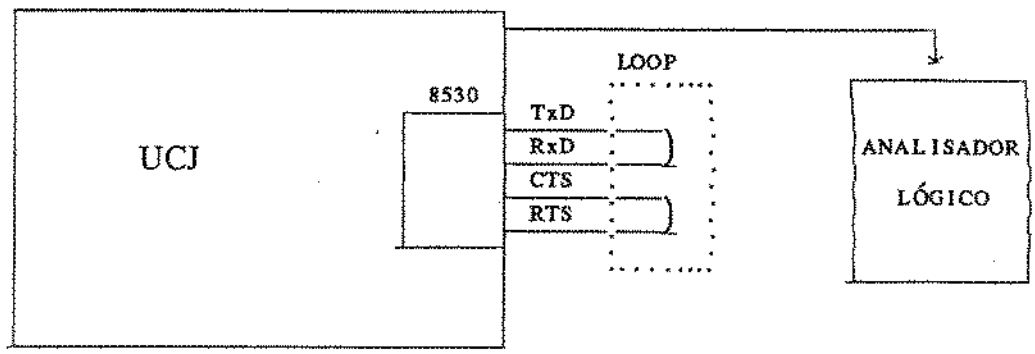


Figura 4.8 - Esquema do teste do protocolo de comunicação

O *loop* remoto foi estabelecido nos sinais de dados (TxD e RxD) e nos sinais de controle de fluxo RTS e CTS. O Analisador Lógico também foi utilizado para medir o tempo envolvido na troca de mensagens (Figura 4.8).

• EXECUÇÃO: O programa utilizado neste teste é essencialmente o mesmo programa SERIAL, descrito no Capítulo 3, adaptado para a execução em *loop* remoto e transmitindo a uma velocidade de 2400 bps. São colocados no *buffer* de transmissão os dados referentes a uma sequência de eventos que simula a interação entre a UCJ e a UCS. A sequência é a seguinte:

• UCS envia mensagem de supervisão requisitando a condição da UCJ. Valor da mensagem: 80H;

• UCJ responde enviando uma mensagem de condição OK. Valor da mensagem: 82H;

• UCS transmite à UCJ uma mensagem de supervisão indicando início de trajetória. Valor da mensagem: 81H;

• UCS transmite os pontos referentes à trajetória desejada. Valor da mensagem: 84H.

Assim como no teste descrito no ítem 4.3.1, foram colocados apontadores no programa que possibilitassem a medida do tempo de transmissão e tratamento das mensagens. A tabela 4.4 mostra alguns dos tempos medidos para alguns conjuntos de mensagens diferentes.

NÚMERO DE MENSAGENS	TEMPO DE TRATAMENTO
2	9 ms
5	22,7 ms
10	46 ms

Tabela 4.4 - Tempos medidos para o protocolo de comunicação.

É imediato concluir-se que o tempo para transmitir-se uma mensagem de 1 byte é de 4,5 ms. Como a velocidade de transmissão é de 2400 bps, o tempo para a transmissão de uma mensagem de 1 byte é:

$$T = \frac{8 \text{ bits} \times 1 \text{ s}}{2400 \text{ bits/s}} = 3{,}3 \text{ ms}$$

Ou seja, dos 4,5 ms medidos entre duas mensagens transmitidas/recebidas, 3,3 ms estão reservados para a taxa de transmissão/recepção.

• COMENTÁRIOS: Baseado nos resultados encontrados, vimos que dos 4,5 ms medidos entre duas mensagens, 73% do tempo (3,3 ms) referem-se à taxa de transmissão de 2400 bps. Os 27 % restantes (1,2 ms) referem-se ao atendimento de interrupções e processamento em geral.

Para taxas de transmissão maiores, o tempo decairia significativamente. Utilizando-se ainda um protocolo assíncrono, as taxas poderiam atingir até 9600 bps (este é um valor típico para este tipo de interface). Neste caso, o tempo de uma mensagem fica:

$$T = \frac{8 \text{ bits} \times 1 \text{ s}}{9600 \text{ bps}} = 830 \ \mu s$$

Neste caso o tempo entre mensagens seria aproximadamente 2 ms. Como as trocas de mensagens são esporádicas, este evento não ocuparia significativamente o processador.

Além disso, no programa SERIAL, um mesmo processador e um mesmo canal do 8530 são responsáveis pela transmissão, recepção e tratamento de todas as mensagens e interrupções decorrentes do tráfego das mesmas. Na operação integrada do sistema, este processamento passa a ser distribuído entre os processadores da UCJ e da UCS.

Como foi dito anteriormente, o protocolo simplificado acrescenta um *overhead* no tratamento, pois impõe ao processador que este analise as mensagens byte a byte, examinando os bits mais sgnificativos, para definir o tipo de mensagem trocada.

Diante de todos estes fatores, pode-se concluir que o grande contribuição deste teste foi a de definir o limitante de pior caso para a operação do sistema no que diz respeito à comunicação entre os processadores.

4.3.4. Controle do robô

• OBJETIVO: Verificar a atuação da UCJ no controle de juntas assim como avaliar o desempenho do programa CONTROLE_ON_OFF na tarefa de controlar a junta segundo um algoritmo do tipo *on-off*.

• AMBIENTE: O ambiente de execução desse teste é uma expansão do ambiente utilizado no teste de acionamento em malha aberta descrito no item 4.3.2, com a inclusão da UCJ para o fechamento da malha e a execução do programa CONTROLE_ON_OFF (Figura 4.9).

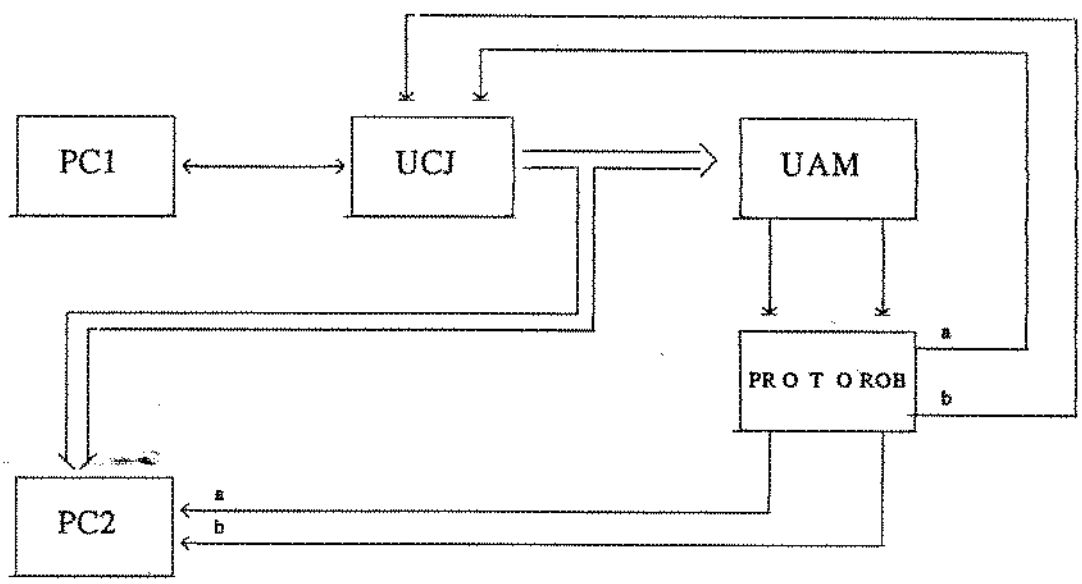


Figura 4.9 - Esquema de teste do controle do robô.

• EXECUÇÃO: Para a execução desse teste, foram impostas diferentes trajetórias com diferentes combinações de parâmetros. Na Tabela 4.5 pode-se verificar os pontos impostos e os valores dos parâmetros.

Os pontos da trajetória assim como os parâmetros iniciais são definidos através de preenchimento na memória.

A trajetória nº 1 foi definida como sendo crescente e linear, apresentando dois patamares onde o sistema permanece parado. Esta trajetória é similar ao ensaio nº 12 do teste de acionamento em malha aberta.

A trajetória nº 2 foi definida a partir do programa DESLOC, com um perfil de velocidade estabelecido pelo programa VELOC, ambos descritos no Anexo IV. Esta trajetória está dividida em três partes. As leis que regem o deslocamento nos três trechos estão na Equação 4.14.

$$\text{Trecho I:} \quad s = \frac{1}{2} \times a \times t^2$$

$$\text{Trecho II:} \quad s = V \times t \qquad (4.14)$$

$$\text{Trecho III:} \quad s = -\frac{1}{2} \times a \times t^2$$

No trecho I o deslocamento se dá segundo uma parábola, com aceleração constante e velocidade crescente. No trecho II, a velocidade é constante e, no trecho III a desaceleração é constante e a velocidade decresce com o tempo.

TRAJETÓRIA	PONTOS DEFINIDOS	PARÂMETROS
1	00,02,04,06,08, 0A,0C,...,8C, 8E,90,90,90,90, 90,90,90,90,90, 90,94,98,9C,A0, A4,...,BA,BC, BE,C0,C0,C0,C0, C0,...C0.	$\varepsilon_{máx} = 1$ $\varepsilon_{crit} = 5$ Nº de sinais de controle enviados = 32
2	00,00,00,00,00, 01,01,01,02,03, 04,04,05,06,07, 09,0A,0B,0C,0E, 10,13,15,17,19, 1B,1D,1F,21,23, ...,AF,B2,B4, B6,B7,B9,BA,BB, BD,BE,C0,C1,C2, C2,C3,C4,C4,C5	$\varepsilon_{max} = 1$ $\varepsilon_{crit} = 5$ Nº de sinais de controle enviados = 1

Tabela 4.5 - Pontos e parâmetros das trajetórias em malha fechada.

Os gráficos nas Figuras 4.10 e 4.11 mostram as curvas de posição e velocidade em função do tempo para os ensaios 1 e 2.

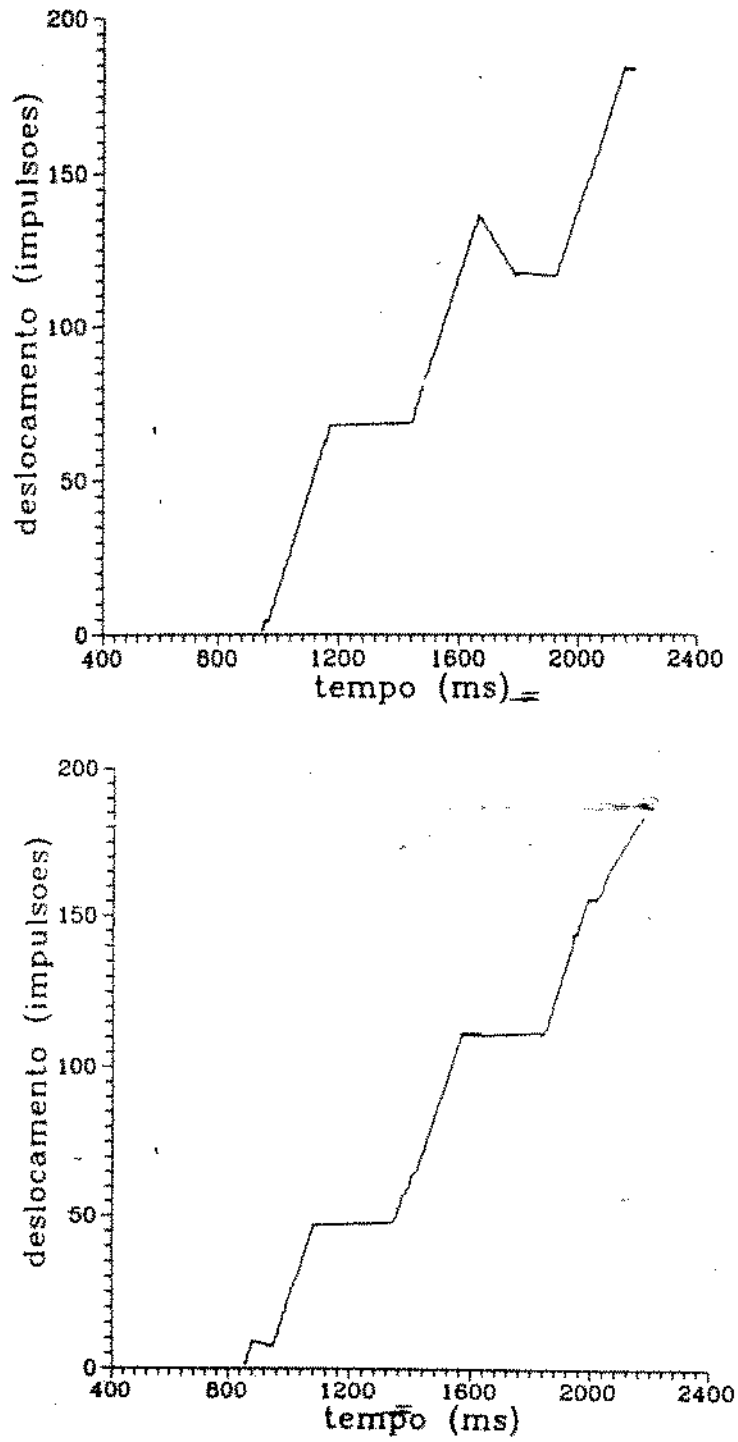


Figura 4.10 - Trajetórias impostas ao robô.

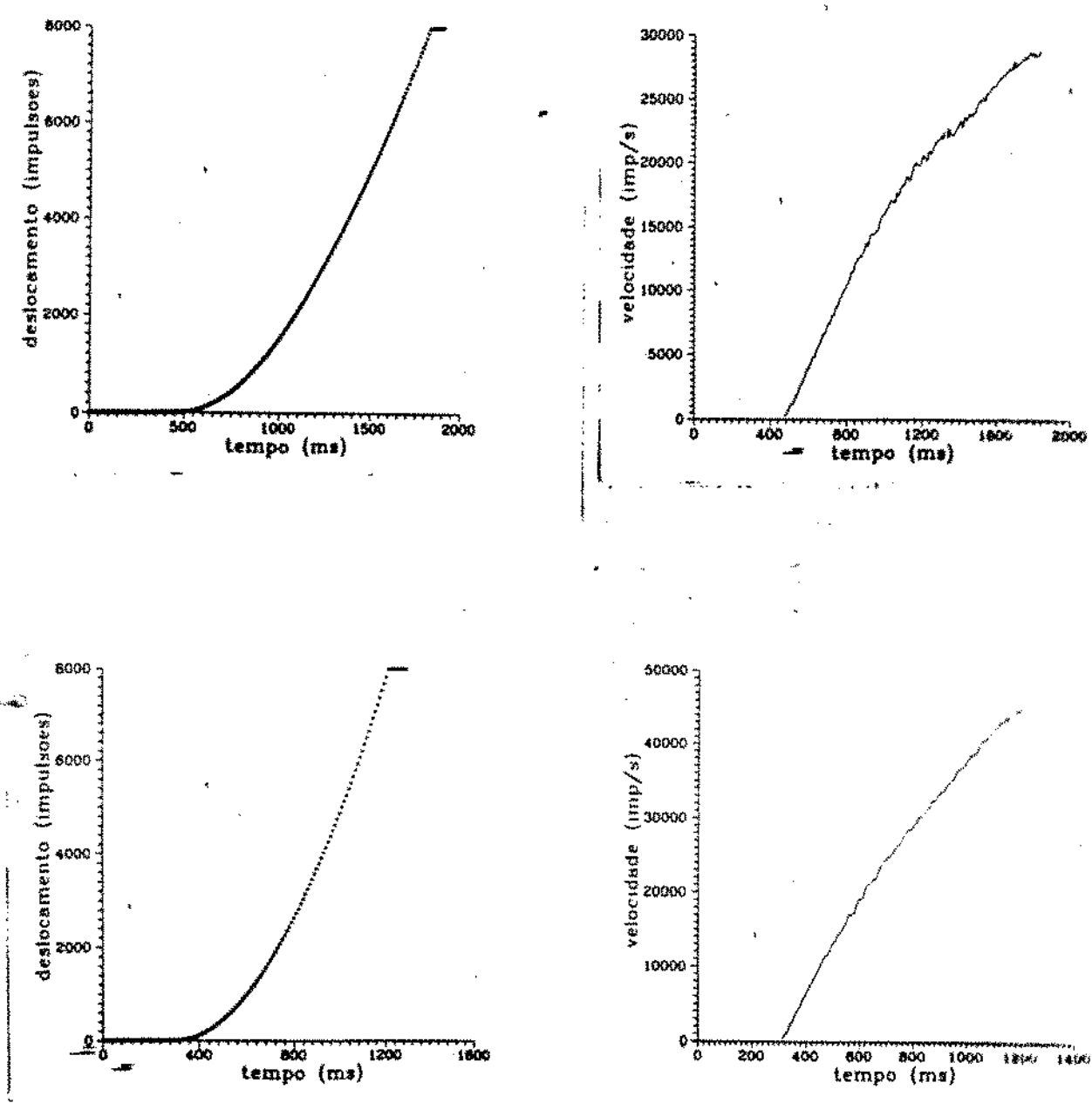


Figura 4.11 - Curvas de posição e velocidade imipostas ao robô.

A partir execução da trajetória, podem ser observados os gráficos comparativos das trajetórias realizadas por uma junta do robô, conforme mostra a Figura 4.12.

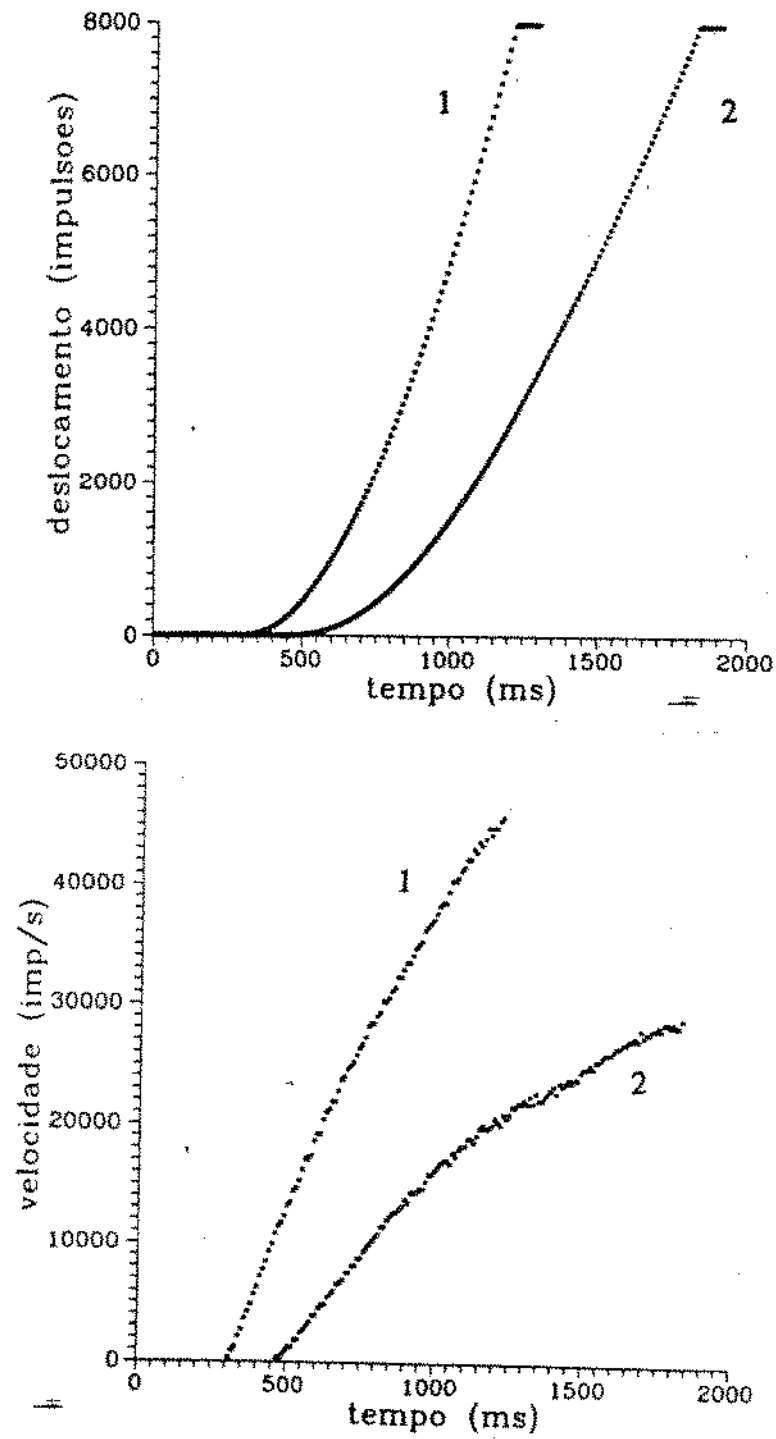


Figura 4.12 - Gráfico comparativo das trajetórias realizadas.

• COMENTÁRIOS: Durante a fase de aquisição de dados, pode-se ter uma idéia mais real das constantes de tempo envolvidas no sistema. Tomando-se, por exemplo o parâmetro que define o número de vezes que o sinal de correção vai ser enviado à UAM, observou-se que, para um número pequeno, o sistema reagiu perdendo muitos pontos, como ficou constatado nas curvas levantados.

Também o parâmetro que estabelece o número de interrupções de 1 KHz que devem ocorrer entre duas comparações foi bastante mudado para que o sistema pudesse responder adequadamente.

A conclusão inicial é de que o sistema possui tempos que são ordens de grandeza maiores que os tempos do processador e mesmo tratando-se sistemas comerciais, o protótipo poderia ser utilizado.

Quanto aos parâmetros do sistema, estes possibilitaram grande gama de combinações, sendo que para alguns valores do erro máximo o sistema não se comportou satisfatoriamente. Este fato se deve à pouca precisão do mecanismo de transdução associado à limitação do número máximo de pontos que podem ser definidos para um trajetória.

As curvas obtidas mostram que a junta sob controle executou trajetórias relativamente próximas ao perfil definido, com os desvios inerentes ao próprio algoritmo (regiões que apresentam pontos bastante fora da trajetória definida).

4.4. CONCLUSÃO

Este capítulo concentrou-se no ambiente experimental montado para a avaliação da arquitetura concebida para a implementação do protótipo.

Foram levantados aspectos construtivos da plataforma de testes, destacando-se o robô PROTOROB1 para fins didáticos. Em seguida foi feita uma descrição sobre a estratégia de testes adotada assim como a descrição do ambiente, procedimento e resultados de cada teste realizado.

Destacam-se os resultados obtidos para a interface de comunicação e o desempenho do algoritmo *on-off* no controle das juntas do robô, assim como o seu impacto no processamento.

CAPÍTULO 5

COMENTÁRIOS E CONCLUSÕES

Este capítulo apresenta uma análise global dos resultados obtidos nos procedimentos experimentais, destacando os aspectos de maior impacto no desempenho da arquitetura na qual o protótipo do supervisor foi baseado.

Em seguida são propostas sugestões para melhoria do desempenho do supervisor implementado segundo a mesma arquitetura. São discutidas também as implicações envolvidas na adoção de cada uma das soluções.

Por fim é feita uma análise a respeito das perspectivas futuras para a evolução do sistema.

5.1. ANÁLISE DO DESEMPENHO

No trabalho de *Fayan* [1], foi apresentada uma análise teórica do desempenho do supervisor onde dividiu-se os trechos para análise em 4 partes:

- Processamento da UCJ
- Protocolo Serial Assíncrono PC-UCS (TTL)
- Processamento da UCS
- Protocolo Serial Assíncrono UCS-UCJ (RS-232)

As montagens experimentais realizadas permitem uma análise direta a partir dos resultados obtidos do processamento na UCJ e do protocolo serial RS-232. Os resultados permitem também uma análise indireta do protocolo serial entre UCS e PC.

5.1.1. Processamento na UCJ

O processamento da UCJ deve prover tratamento de eventos simultâneos de tráfego na linha serial e de execução do algoritmo de controle.

Os resultados do item 4.3.1 mostram que o processador executa o tratamento de um ponto da trajetória em aproximadamente 130 μs. Para um intervalo de 1 ms, isto representa 87 % do tempo disponível para que o processador faça o tratamento da linha serial.

Para um sistema onde as constantes de tempo estabeleçam uma frequência de operação de 4 ms, valor este que encontra-se no conjunto de requisitos iniciais para o supervisor em estudo, este percentual aumentaria para cerca de 96 % de disponibilidade.

Os resultados encontrados para o PROTOROB1, por este ser implementado com motor de passo e possuir um dispositivo de transdução de relativa precisão, mostraram que as constantes do sistema neste caso são pelo menos uma ordem de grandeza maior. Neste caso, o processador da UCJ estaria praticamente em *idle* quanto à execução do algoritmo de controle, pois o percentual de tempo envolvido na execução seria 0,5 % no máximo.

Numa situação em que o processador fosse mais exigido no controle de um sistema com constante de tempo menor e executando um algoritmo de controle mais complexo, poder-se-ia manter o uso do microprocessador NEC V20. Uma análise mais detalhada poderia levar à adoção das seguintes melhorias:

- Aumento do relógio do processador, uma vez que o NEC V20 pode ser encontrado comercialmente em até 16 MHz. Neste caso seria necessário alterar a velocidade de acesso das memórias.

- Troca de processador, mantendo-se a linha INTEL - lembrar que o NEC V20 é um *upgrade* do INTEL 8088. Neste caso, com a migração para um microprocessador 80286 ou mesmo 80386, e com as adaptações necessárias, a UCJ e a UCS estariam em condições de controlar até mesmo sistemas mais complexos, sem perder as vantagens inerentes da arquitetura distribuída e as características de portabilidade presentes na solução adotada.

5.1.2. Protocolo de comunicação

Os procedimentos experimentais para o teste da comunicação entre os processadores mostraram uma situação de sobrecarga do processador, uma vez que este simulou a comunicação entre UCI e UCS, desempenhando os papéis dos processadores de ambas as placas. O protocolo simplificado também contribuiu para o aumento do *overhead* da comunicação.

No entanto, os resultados mostraram que do tempo total para transmissão de uma mensagem de um byte uma parcela fixa de 1,2 ms está associada com o tratamento da mensagem, inclusive atendimento de interupções de diversas fontes. Como pode-se observar nas listagens que contém as rotinas de tratamento de interrupção no Anexo II, existem temporizações não otimizadas e que podem ser adaptadas para uma melhoria de desempenho do protocolo.

A taxa de transmissão, que para um relógio de 2400 bps representou 73% do tempo total, pode ser aumentada para até 9600 bps dentro do mesmo protocolo, passando a ter um papel insignificante no tempo de comunicação.

A operação com o protocolo completo, com mensagens com até 3 bytes de comprimento, mas que utiliza sinais de *hardware* [RST e CTS] para indicar mensagens transmitidas, traz uma melhoria em relacão ao protocolo simplificado, pois as interrupções passam a ocorrer a cada mensagem, e não byte a byte. Desse modo há uma diminuição no tempo de tratamento de mensagens.

Outra soluções para a otimização da via de comunicação entre processadores são apresentadas no item 5.2 (Propostas de melhorias).

5.1.3. Algoritmo de controle

O algoritmo de controle *on-off* executado num processador NEC V20 mostrou-se bastante rápido, sendo necessário inclusive uma compensação de tempo para que o robô pudesse responder aos sinais de controle.

O tempo de execução, da ordem de 130 μs para um ponto da trajetória, pode ainda ser diminuído com as melhorias sugeridas no item 5.1.1.

As limitações inerentes a este tipo de protocolo tais como oscilação e erro significativo também foram observadas. Porém a adoção de outro tipo de algoritmo, dentro da filosofia de modularidade e portabilidade que caracterizam o supervisor, permitem a alteração das características do controlador.

5.2. PROPOSTAS DE MELHORIAS

Para a arquitetura distribuída segundo a qual o supervisor foi concebido, propõe-se melhorias que têm por objetivo um aumento no desempenho da mesma, levando-se em conta os resultados obtidos.

Cada proposta representa um avanço tecnológico, com um custo de desenvolvimento associado.

a) Troca do processador: Colocando-se outro processador da linha INTEL, como 80286 ou 80386, pode-se executar os mesmos programas Monitor e CONTROLE_ON_OFF, configurando-se a CPU no modo real.

Para um aproveitamento maior das potencialidades do processador, seria pertinente a inclusão de mecanismos tais como memória cache e co-processador numérico no caso da UCS, onde cálculos complexos de inversão matricial são efetuados.

No entanto, o sistema não perderia suas características de modularidade e portabilidade e assim se converteria numa plataforma apropriada e eficiente para futuros desenvolvimentos e aplicações.

b) Protocolo de comunicação síncrono: Atuando em outra frente, propõe-se a adoção de um protocolo síncrono do tipo HDLC (High-Level Data Link Control), com confiabilidade bem maior do que o protocolo assíncrono possibilidade de taxas de até 1,5 Mbps, desde que para pequenas distâncias, como é o caso.

Há um custo associado ao desenvolvimento deste protocolo e à integração do mesmo com o sistema. O chip de controle de linha serial 8530 continuaria a ser utilizado na implementação da camada 1 do protocolo.

c) Utilização de memória compartilhada: Para a comunicação entre os processadores pode-se adotar uma solução que utilize memória compartilhada, onde cada processador, seja da UCS ou da UCJ, possui sua memória local e uma memória que pode ser acessada por outros processadores.

Este mecanismo traz consigo a vantagem de diminuir bastante o tempo de comunicação entre os processadores. É necessário, contudo, desenvolver um circuito arbitrador que defina qual processador deve fazer o acesso à memória compartilhada.

Esta solução é empregada por *Le Van* [17], onde cada elemento controlador de junta usa uma área de memória compartilhada para comunicação com o outro controlador assim como para comunicação com um nível de supervisão.

5.3. PERSPECTIVAS E CONCLUSÕES

Este trabalho teve como objetivo principal a implementação de um supervisor de controle de modo a possibilitar uma análise do desempenho da arquitetura distribuída em que este supervisor foi baseado. Foi feita uma implementação parcial, enfocando-se basicamente o controle de juntas e os fluxos de infromações entre os níveis de controle.

As atividades laboratoriais foram bastante intensas, possibilitando uma análise prática dos diversos aspectos envolvidos no controle de robôs. Além disso, foi criado um ambiente de testes permanente que pode ser utilizado em futuras implementações, empregando-se outros algoritmos no controle de mecanismos com outros tipos de acionamento, bastando para isso, desenvolver-se outra UAM.

Quanto às arquiteturas distribuídas, acreditamos que sejam um caminho irreversível para o controle de sistemas complexos, submetidos a condições adversas. Esta idéia se solidifica em recentes aplicações de redes neurais [18], que podemos considerar uma evolução do conceito de arquitetura distribuída.

A perspectiva agora é de que possa ser implementado um protótipo completo, com seis UCJ integradas a 1 UAM, e todos os programas aplicativos, de modo a possibilitar o controle do robô Manutec r3, abrindo assim o caminho para um domínio amplo do conceito de arquiteturas distribuídas aplicadas na supervisão e controle de robôs.

BIBLIOGRAFIA

[1] Fayan, Benedito Luís; "Estudo e Especificação de um supervisoor de Controle para um Robô Industrial", Tese de Mestrado, Faculdade de Engenharia Mecânica, UNICAMP, (1992).

[2] Dias, Marcus Aguiar; "Controlador Programável a Multiprocessadores para Controle Hierárquico de Robôs", Tese de Mestrado, Faculdade de Engenharia Elétrica, UNICAMP, (1991).

[3] Miranda, Márcio Fantini; "Controle de um servomecanismo por um microcomputador dedicado: Uma contribuição ao estudo de controladores para robôs industriais", Tese de Mestrado, Faculdade de Engenharia Mecânica, UNICAMP, (1992).

[4] V.24, International Standard; CCITT; Blue Book, (1988).

[5] SDK-86 - System Design Kit - INTEL

[6] Menezes, J. ; "Elaboração de Software utilizando técnicas de Inteligência Artificial para tratamento de colisões em tempo real para implementação num manipulador industrial".

[7] Lau, I. F. ; "Elaboração de software para visualização e geração automática de trajetórias para robôs manipuladores em intervenções submarinas em águas profundas".

[8] 8088 - 8 bit Processor - Microprocessor and Peripheral Handbook - Vol. I - Microprocessor - INTEL - 1989.

[9] Manual da NEC - V20.

[10] Manual do DATA/IO 2900.

[11] Rosário, J. M.; "A systematic methode to the kinematic analyisis of industrial robots and manutec robo R-15", publicação interna GKSS, Geesthacht, (1990).

[12] AMD - Z85C30 - Enhanced Serial Communications Controller - AMD - ISDN - 1989/90 - Data Book.

[13] PIC - Programmable Interrupt Controller - Intel - Microprocessor and Peripheral Handbook - 8259A - Vol. I - 1989.

[14] Programmable Array Logic - AMD - 1984 - PAL.

[15] Rosário, J. M.; Verdin L.; "Etude de faisabilité d'un asservissement de position par commande tout-ou-rien", ISMCM, (1987).

[16] Souza, J. P. ; "Procedimento Automático para Aquisição e Tratamento do Movimento de um Robo - Uma contribuição ao Estudo de Controladores Não Lineares"; UNICAMP; (1992).

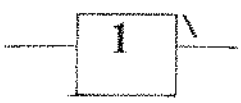
[17] Le Van, D. O.; "Multiple processor architecture for high performance control of robotic structures", IEEE Computer Society Press, (1986).

[18] Walter, Jörg A.; Schulten, K. J.; "Implementation of self-organizing neural networks for visuo-motor control of an industrial robot", IEEE Transactions on Neural Networks, Vol. 4, (1993).

ANEXO I

DIAGRAMAS DE BLOCOS, ESQUEMAS ELÉTRICOS E TABELAS

Este anexo apresenta os diagramas de blocos e esquemas elétricos das unidades UCJ e UAM. Para os esquemáticos, utilizou-se a notação internacional do IEEE (Institute of Electrical and Eletronics Engineers). No entanto, por não ser tão divulgado fora da área da Engenharia Eletrônica, julgamos pertinente elaborar a legenda abaixo, onde são apresentadas as notações empregadas pelo IEEE para as portas lógicas básicas E, OU, OU-EXCLUSIVO e INVERSOR, de maneira a facilitar o entendimento do esquema elétrico.



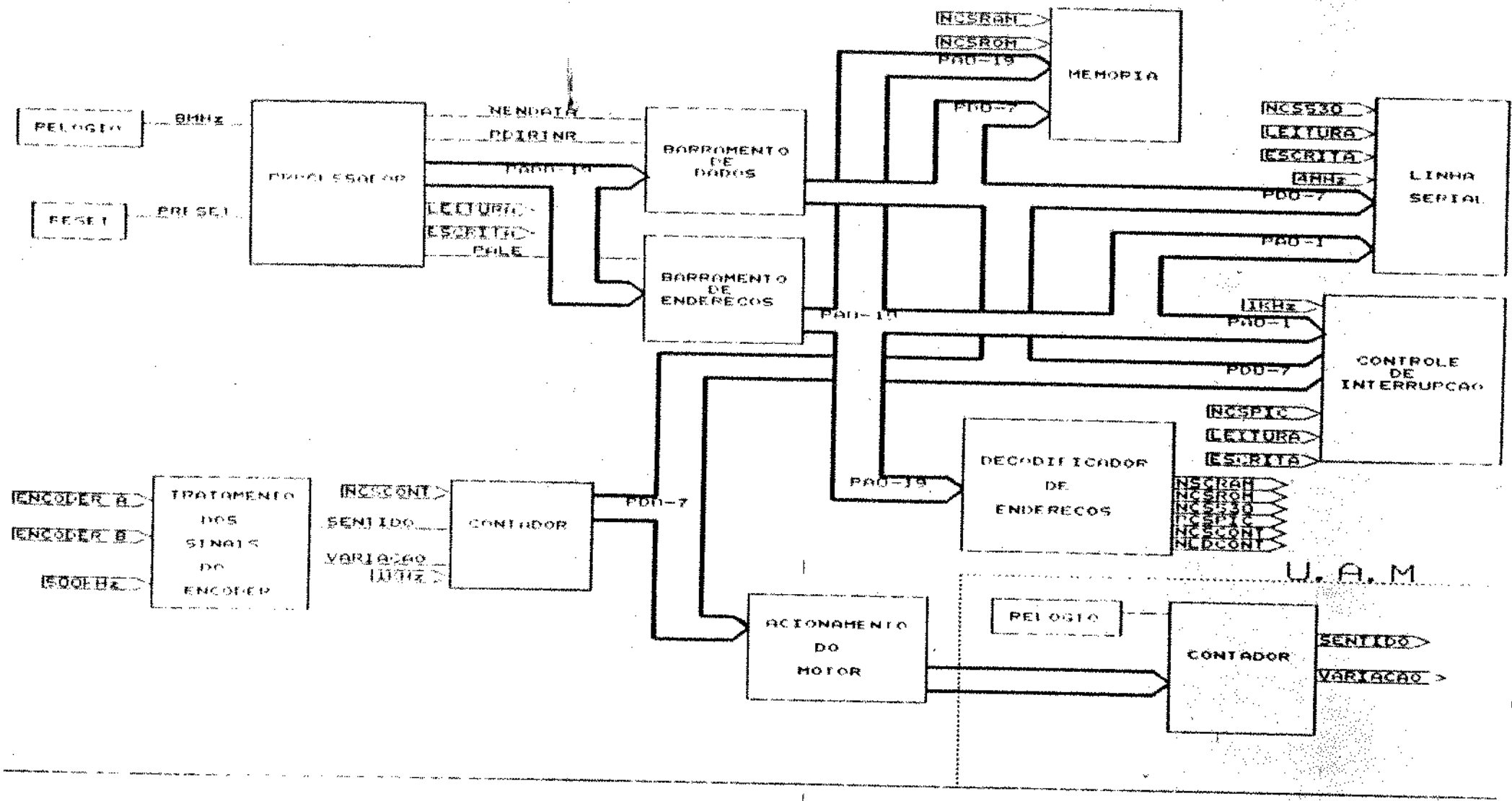
≡ INVERSOR

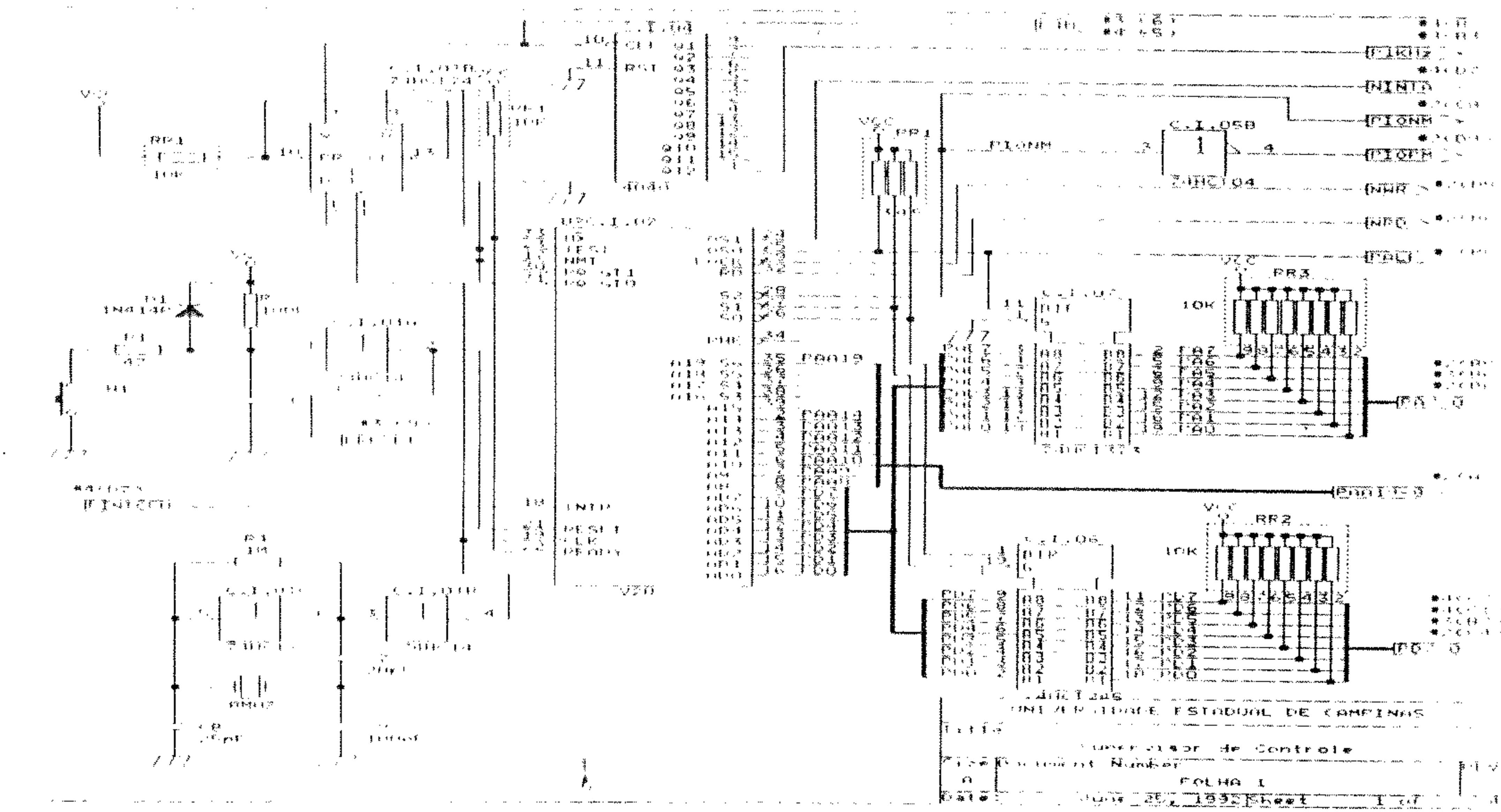
≡ OU

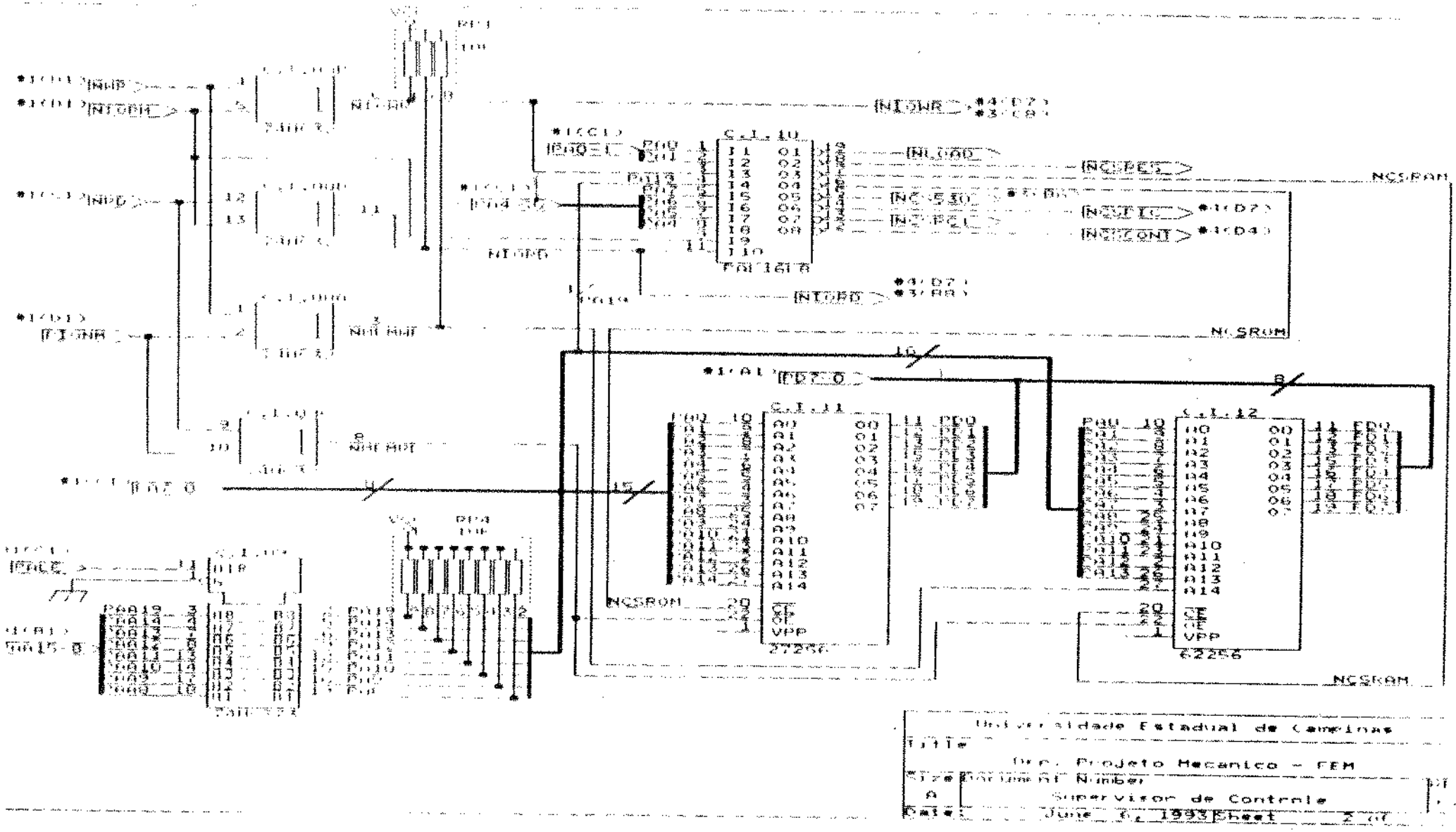
≡ E

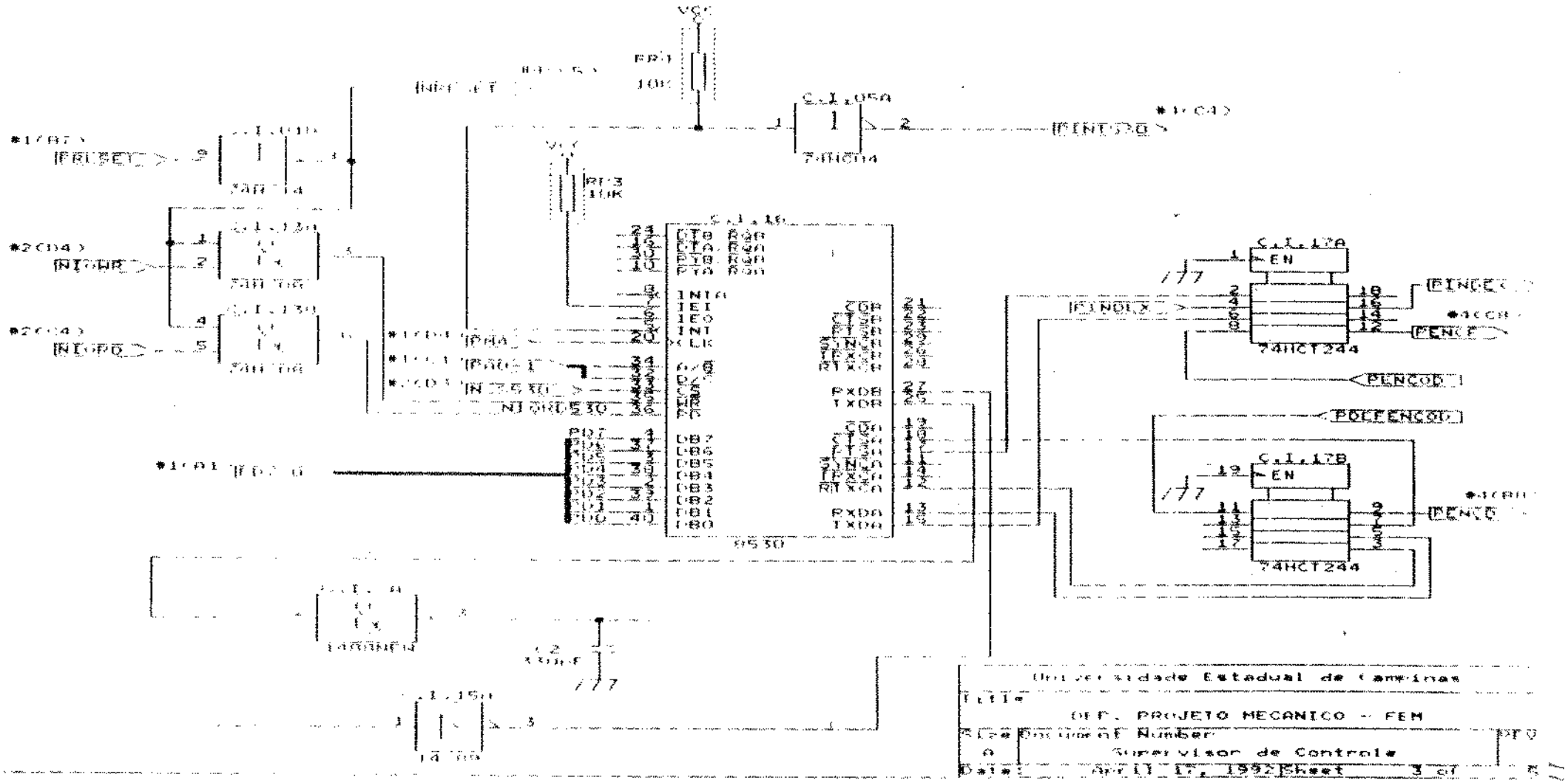
≡ OU-EXCLUSIVO

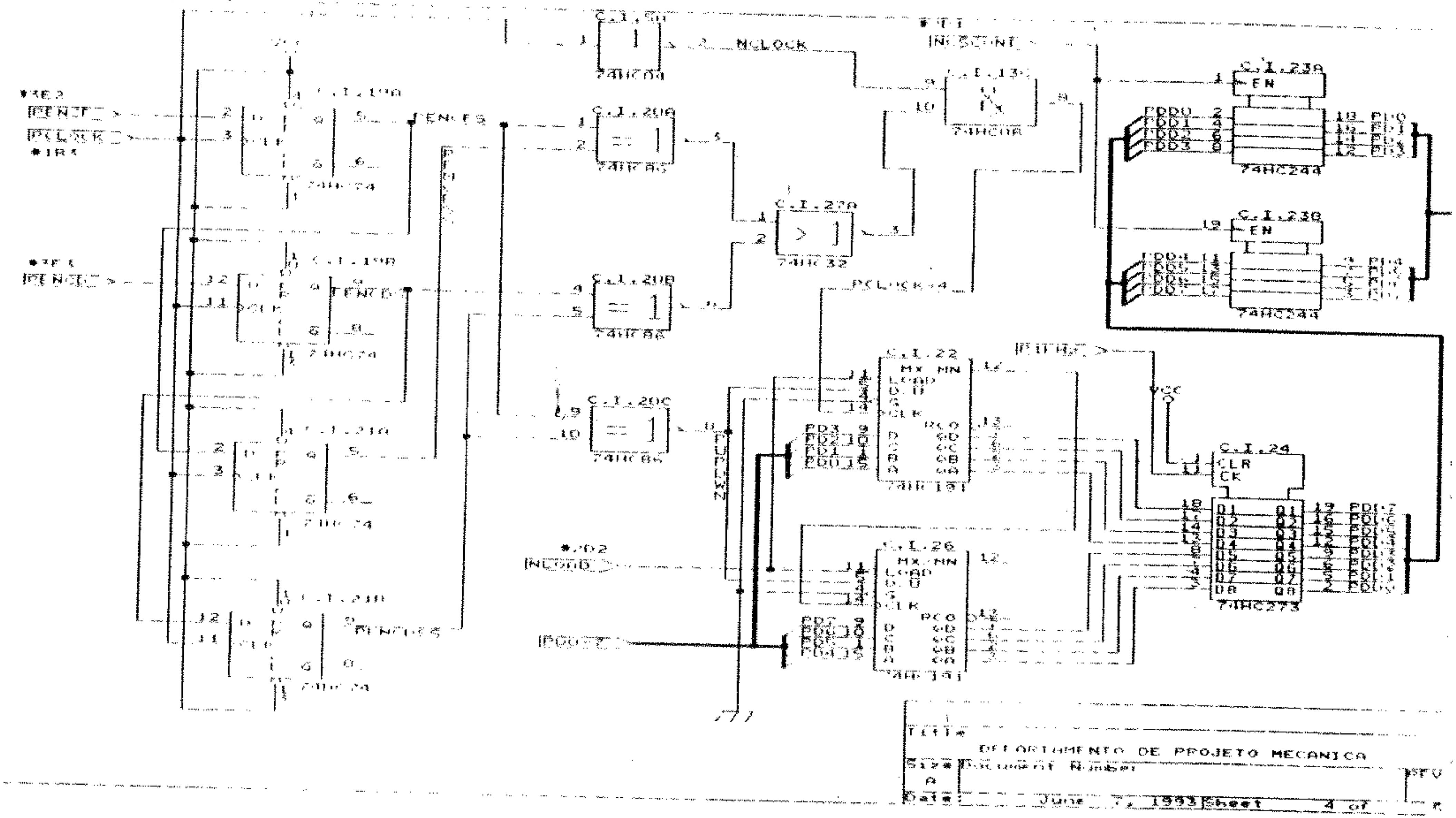
DIAGRAMA DE BLOCOS - U.C.J. + U.A.M.

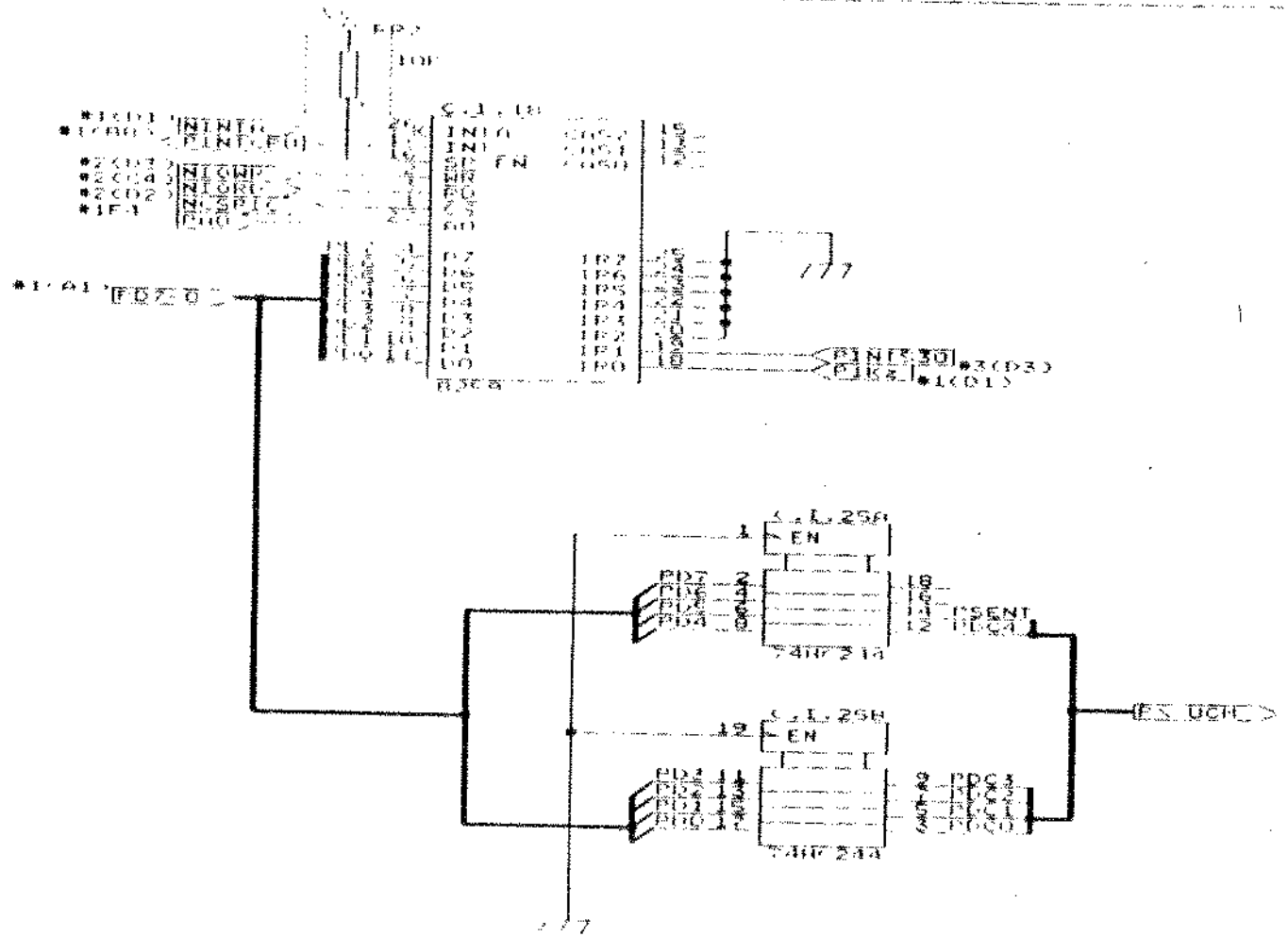



UNIVERSIDADE ESTADUAL DE CAMPINAS
Supervisor de Controle
FOLHA 1
Sheet 1
C.I.05B
74HC04
C.I.06
74HC245
PR1
RR2
RR3
10K
VCC
PIONM
PIOFM
NWR
NRD
NINTA
1N4148
V20


NCSRAM
NCSROM
27256
62256
Universidade Estadual de Campinas
Dep. Projeto Mecanico - FEM
Supervisor de Controle
June 3, 1993


Universidade Estadual de Campinas
DEF. PROJETO MECANICO - FEM
Supervisor de Controle
74HCT244
74HCT244
8530


NCLOCK
C.I.19A
C.I.19B
C.I.21A
C.I.21B
C.I.20A
C.I.20B
C.I.20C
C.I.27A
C.I.23A
C.I.23B
C.I.22
C.I.26
C.I.24
74HC04
74HC86
74HC32
74HC08
74HC244
74HC191
74HC273
PCLOCK
VCC
CLR
CK
EN
DEPARTAMENTO DE PROJETO MECANICA
Title
Size
Document Number
A
Date
1993
Sheet
4 of



Universidade Estadual de Campinas	
Title	Dep. de Projeto Mecanico - FEM
Size: A	Document Number: Supervisor de Controle
Date:	March 11, 1993 Sheet 5 of

COMPONENTES UTILIZADOS NA IMPLEMENTAÇÃO DOS BLOCOS FUNCIONAIS

A - UCJ

BLOCO FUNCIONAL	IDENTIFICAÇÃO DOS COMPONENTES	TIPO DE COMPONENTE
Processador	CI 02	NEC V20
Relógio	CI 01 CI 03 CI 04	74HCT14 74HCT74 74HCT4040
Barramento de Dados	CI 07	74HCT245
Barramento de Endereços	CI 06 e CI 09	74HCT373
Decodificador de Endereços de Memória e Periféricos	CI 10	PAL16L8A
Linha Serial	CI 14 CI 15 CI 16 CI 17	14C88 14C89 8530 74HCT244
Controle de Interrupções	CI 18	8259
Tratamento dos sinais do encoder	CI 17 CI 19 e CI 21 CI 20 CI 122	74HCT244 74HC74 74HC86 74HC32
Contador	CI 22 e CI 126	74HC191
Acionamento do Motor	CI 127	74LS244 (*)

(*) - Como exceção, foi utilizado este componente da família lógica LS, pois apenas esta tecnologia oferece este componente de interface TTL com histerese, apresentando, portanto, maior imunidade a ruídos.

B - UAM

BLOCO FUNCIONAL	IDENTIFICAÇÃO DOS COMPONENTES	TIPO DE COMPOBNENTE
Contador	CI 01 e CI 02	74HC191
Detector de Sentido	CI 04	74HC74
Habilitador / Desabilitador de Contagem	CI 03	74HC00

ANEXO II

LISTAGENS DOS PROGRAMAS

Este anexo apresenta as listagens dos principais programas desenvolvidos na implementação do protótipo. Inicialmente é apresentado o programa CONTROLE_ON_OFF e em seguida são apresentadas as rotinas de tratamento de interrupção do programa SERIAL.

Ambos foram desenvolvidos em Assembler 8088 no ambiente de desenvolvimento HP 64000 e utilizado na implementação do protótipo.

PROGRAMA CONTROLE_ON_OFF

```
"8088"

;-----------------------------------------------
; PROGRAMA DE CONTROLE DE JUNTA DE ROBO
; ELABORADO POR: CARLOS HENRIQUE DIAS
; DATA: 13/04/1993
;-----------------------------------------------

                ASSUME      CS:PROG,DS:DATA

;-----------------------------------------------
; CONSTANTES USADAS NO PROGRAMA
;-----------------------------------------------

;-----------------------------------------------
; USADAS NO CONTROLE DA JUNTA
;-----------------------------------------------
```

```
RWPO20      EQU     20H     ;REG. ESCRITA POS. JUNTA
RWPO30      EQU     30H     ;REG. ESCRITA ACION. JUNTA
RWPO40      EQU     40H     ;REG. LEITURA CONTADOR ENCODER

;-----------------------------------------
; REFERENTES AO CONTROLADOR DE INTERRUPCOES
;-----------------------------------------

ICW1        EQU     80H     ;COMANDO DE INICIACAO 1
ICW2_4      EQU     81H     ;COMANDO DE INICIACAO DE 2 A 4
OCW1        EQU     81H     ;COMANDO DE CONTROLE 1
OCW2        EQU     80H     ;COMANDO DE CONTROLE 2
EOI_ALL     EQU     20H     ;FIM DE INTERRUPCAO

;-----------------------------------------
; SEGMENTO DE DADOS
;-----------------------------------------
            DATA
;-----------------------------------------
; VARIAVEIS UTILIZADAS NO PROGRAMA
;-----------------------------------------
        ORG 00001000H
NUM_PONTOS:     DW      0FFH
BUF_VAL_DES:    DB      0FFH(00)    ;RESERVA 256 POSICOES
                                    ;P/ TRAJ. DESEJADA
        ORG 00001100H
BUF_VAL_LIDO:   DB      0FFH(00)    ;RESERVA 256 POSICOES
                                    ;P/ TRAJ. REAL
        ORG 00001200H
BUF_ACAO:       DB      0FFH(00)    ;RESERVA 256 POSICOES
                                    ;P/ ACAO DE CONTROLE
        ORG 00001300H
BUF_SINAL:      DB      0FFH(00)    ;RESERVA 256 POSICOES
                                    ;PARA SINAL DA ACAO
```

```
                ORG 00001400H
VAL_INI:        DB      0
CONT_INT:       DB      5
EMAX:           DB      6
ECRIT:          DB      12
FLAG:           DB      0
SOMA:           DB      0
TEMPO_1         DB      1
TEMPO_2         DB      1

;-----------------------------------------------
; SEGMENTO DE CODIGO
;-----------------------------------------------

                PROG
;-----------------------------------------------
; PROGRAMA PRINCIPAL
;-----------------------------------------------
PRINC:
                CLI
                MOV     AX,CS           ;INICIA ES=CS
                MOV     ES,CS
                MOV     AX,0H           ;INICIA DS=0H
                MOV     DS,AX

;-----------------------------------------------
; PROGRAMACAO DO CONTROLADOR DE INTERRUPCOES
;-----------------------------------------------
                MOV     DX,ICW1         ;DX CONTEM O END. ICW1
                MOV     AL,17H          ;PROGRAMA ICW1
                OUT     DX,AL           ;INT. BORDA, INTERVALO=4
                                        ;SINGLE MODE
                INC     DX              ;DX CONTEM END. ICW2_4
                MOV     AL,80H          PROGRAMA ICW2
                OUT     DX,AL

                MOV     AL,01H          ;PROGRAMA ICW4
                OUT     DX,AL           ;MODO 8088, MODO NORMAL
```

```
;-----------------------------------------
; PROGRAMACAO DO VETOR DE INTERRUPCAO
;-----------------------------------------
                MOV     DX,OFFSET INT_1K  ;OFFSET ROTINA INT.
                MOV     BX,SEG INT_1K     ;SEG. ROTINA INT.
                MOV     SI,200H           ;APONTA P/ END. INT. 1K
                MOV     DS:[SI],DX        ;ESCREVE OFFSET DA ROTINA
                INC     SI
                INC     SI
                MOV     DS:[SI],BX        ;ESCREVE SEG. DA ROTINA

;-----------------------------------------
; INICIA VARIAVEIS
;-----------------------------------------

INICIO:
                MOV     AL,DS:VAL_INI     ;CARREGA VALOR INICIAL
                OUT     RWPO20,AL

                MOV     BX,O              ;INDEXADOR P/ BUFFERS

                MOV     DI,OFFSET BUF_VAL_LIDO;INICIA BUFFERS C/
                MOV     SI,OFFSET  BUF_VAL_DES ;VAL. DES. E LIDOS

                CALL    EN_INT            ;HAB. INT. DE 1K
                STI

                MOV     DS:FLAG,2         ;INICIA VAR. FLAG
WAIT_1K:
                CMP     DS:FLAG,1         ;OCORREU 1a INT. DE 1K
                JNE     WAIT_1K           ;AGUARDA INT 1K

                MOV     CX,DS:NUM_PONTOS;CONTROLE P/ PONTOS TRAJ.

TOMA_DADOS: NOP                           ;AGUARDA ULTIMA INT. 1K

                MOV     AL,DS:CONT_INT    ;ESPERA N INT.
                MOV     DS:FLAG,AL        ;INICIA VAR. CONT. INT.
WAIT_1K:
                CMP     DS:FLAG,O         ;OCORREU INT. 1K
                JNE     WAIT_1K           ;AGUARDA INT. 1K

                LOOP    TOMA_DADOS        ;OCORRERAM N INT.

                CLI
                CALL    DIS_INT           ;DESABILITA INT.
```

```
;-----------------------------------------
; SUBROTINA: INT_1K
;-----------------------------------------

INT_1K:
        PUSH    AX
        PUSH    CX
        CMP     DS:FLAG,1           ;SE FOR 1a INT RETORNA
        JNE     REOI_1              ;RETORNA

        MOV     AH,DS:[SI+BX]       ;LE BUFFER VAL. DES.
        IN      AL,RRPO40           ;LE REG. POS. ENCODER

        MOV     DS:[DI+BX],AL       ;PREENCHE BUFFER VAL.
                                    ;LIDOS C/ CONT. REG.
        XCHG    AL,AH               ;TROCA AL E AH
        CALL    SUBT                ;COMPARA DOIS VALORES

        PUSH    DI
        PUSH    SI

        MOV     DI,OFFSET BUF_ACAO  ;INICIA BUFFERS DE
        MOV     SI,OFFSET BUF_SINAL ;ACAO E SINAL

        CMP     AL,DS:ECRIT         ;E MAIOR QUE ERRO CRIT.?
        JG      ALARME1             ;SE FOR ALARMA

        CMP     AL,DS:EMAX          ;E MAIOR ERRO MAX.?
        JG      ATIVA1              ;SE FOR ATIVA CONTROLE

        MOV     DS:[SI+BX],55H      ;SINAL S/ FUNCAO
        MOV     DS:[DI+BX],00H      ;NAO EXECUTA ACAO
        JMP     END1
ATIVA1:
        CMP     AH,1
        JE      ATIVA2
        MOV     DS:[SI+BX],0H       ;SINAL +
        MOV     AL,DS:EMAX
        JMP     ATIVA3
ATIVA2:
        MOV     DS:[SI+BX],0FFH     ;SINAL -
        MOV     AL,DS:EMAX
        OR      AL,40H              ;MASCARO CONTEUDO AL
ATIVA3:
        MOV     DS:[DI+BX],0CH      ;ACAO DE CONTROLE
        ADD     AL,SOMA             ;AUMENTA VALOR DO SINAL
        MOV     CX,TEMPO_1          ;NUM. VEZES ACAO E ENV.
```

```
ESPERA1:    NOP
            OUT     RWPO30,AL       ;ENVIA ACAO DE CONTROLE
            LOOP    ESPERA1         ; N VEZES
            JMP     END1

ALARME1:
            MOV     DS:[SI+BX],55H  ;SINAL S/ FUNCAO
            MOV     DS:[SI+BX],0AH  ;VALOR DO ALARME
END1:
            POP     SI
            POP     DI
            INC     BX
REOI_1:
            DEC     DS:FLAG
            MOV     AL,EOI_ALL      ;FIM DE INT. P/ 8259
            NOP
            MOV     CX,TEMPO_2
ESPERA2:    NOP
            LOOP    ESPERA2
            OUT     OCW2,AL
            STI

            POP     CX
            POP     AX
            IRET

;-----------------------------------------------
; SUBROTINA: SUBT
;-----------------------------------------------

;FUNCAO:SUBTRACAO DE DOIS NUMEROS

SUBT:
            PUSH    BX

            MOV     BX,AX           ;VALOR DE AX EM BX
            SUB     AL,AH           ;SUBTRAI AH DE AL
            JNC     SEM_SINAL       ;RESULTADO EM AL

            MOV     AX,BX           ;RECUPERA VALOR DE AX
            XCHG    AL,AH           ;TROCA POS. AL E AH

            SUB     AL,AH

            MOV     AH,01           ;NEGATIVO, AH=1
            JMP     END_SUB

SEM_SINAL:
            XOR     AH,AH           ;RESULTADO POSITIVO
```

```
END_SUB:
                POP     BX
                RET

;-----------------------------------------
; ROTINA QUE HABILITA INTERRUPCAO
;-----------------------------------------
EN_INT:
                PUSH    AX
                MOV     AL,0FEH         ;HABILITA INTERRUPCAO 0
                OUT     OCW1,AL
                POP     AX
                RET

;-----------------------------------------
; ROTINA QUE DESABILITA INTERRUPCAO
;-----------------------------------------

DIS_INT:
                PUSH    AX
                MOV     AL,0FFH         ;DESABILITA INTERRUPCAO 0
                OUT     OCW1,AL
                POP     AX
                RET

;-----------------------------------------
; VETOR DE PROGRAMACAO DOS VETORES DE INTERRUPCAO
;-----------------------------------------

INT_VECTORS: DW         200H,204H

                END
```

ROTINAS DE INTERRUPÇÃO DO PROGRAMA SERIAL

Conforme descrito na Capítulo 3, o programa que implementa o protocolo simplificado de comunicação entre UCJ e UCS é constituído das etapas de inicalização, espera e tratamento de interrupções. Destas etapas, o tratamento das interrupções constitui-se no cerne do programa. Assim sendo, para dar um destaque maior, estas rotinas são apresentadas a seguir.

```
;-----------------------------------------
; TRATAMENTO DA INTERRUPÇÃO DE UMA LINHA NO MODO ASSINCRONO
;-----------------------------------------

INT_AS:     PUSH    DX          ;DX = END. DA LINHA
            MOV     DX,00H      ;REG. COMANDO DO 8530
            MOV     AL,02H      ;APONTA RR2
            OUT     DX,AL
            IN      AL,DX       ;LE RR2
            POP     DX
T_RR2:      AND     AL,06H      ;IDENTIFICA INT.
            CMP     AL,00H
            JZ      TX_INT      ;INT. DE TRANSMISSAO
            CMP     AL,02H
            JZ      EXT_INT     ;INT. DE EXTERNAL STATUS
            CMP     AL,04H
            JZ      RX_INT      ;INT RECEPCAO

;-----------------------------------------
; SPECIAL RECEIVE CONDITION
;-----------------------------------------

ESP_INT:    MOV     AL,30H      ;RESET ERRO
            OUT     DX,AL
            MOV     CX,0FFFFH   ;TEMPORIZA

;-----------------------------------------
; FIM DE INTERRUPCAO PARA O 8259
;-----------------------------------------

EOI_L:      MOV     AL,38H      ;EOI P/ 8530
            PUSH    DX
            MOV     DX,00H      ;REG. EOI
            OUT     DX,AL
            MOV     AL,20H      ;EOI P/ 8259
            OUT     80,AL
```

```
FIM_INT     POP     DX
            IRET

;-----------------------------------------
; INTERRUPCAO DE EXTERNAL STATUS
;-----------------------------------------
EXT_INT:    IN      AL,DX       ;LE RR0
            MOV     AL,10H      ;RESET EXTERNAL
            OUT     DX,AL
            JMP     EOI_L

;-----------------------------------------
; BUFFER DE TRANSMISSAO VAZIO
;-----------------------------------------

TX_INT      OR      AH,03H      ;FLAG INT. TRANSMISSAO
            AND     AH,0FBH     ;RESET FLAG INT. RECEPCAO
            INC     SI
            MOV     AL,CS:[SI]  ;LE PROXIMO DADO
            CMP     AL,0FFH     ;VERIFICA SE E ULTIMO BYTE
            JZ      CMM_TX
            CALL    TRANSMITE   ;ENVIA PROXIMO DADO
            JMP     EOI_L

;-----------------------------------------
; CARACTER DISPONIVEL NA RECEPCAO
;-----------------------------------------
RX_INT:     OR      AH,05H      ;FLAG INT RECEPCAO
            AND     AH,0FDH     ;RESET FLAG INT TRANSMISSAO
            INC     DI
            MOV     CL,CS:[DI]
            CMP     CL,0FFH
            JNZ     NO_FRX
            OR      AH,08H      ;FLAG DE FIM DE RECEPCAO
NO_FRX:     DEC     DI
            MOV     CL,CS[DI]
            CALL    RECEBE      ;LE DADO
            INC     DI
            CMP     AL,CL
            JZ      EOI_L
            OR      AH,01H      ;FLAG ERRO CARACTER
            JMP     EOI_L

;-----------------------------------------
; ROTINA QUE ENVIA UM DADO PARA A LINHA SERIAL
;-----------------------------------------
TRANSMITE:  PUSH    DX          ;AL=DADO
            MOV     DX,REG_DADO
            OUT     DX,AL
            POP     DX
            RET
```

```
;-----------------------------------------------
; LEITURA DOS DADOS NO CONTROLADOR SERIAL
;-----------------------------------------------

RECEBE:     PUSH    DX              ;AL=DADO
            MOV     DX,REG_DADO
            IN      AL,DX
            POP     DX
            RET
```

ANEXO III - PROGRAMAS UTILITARIOS

Além dos programas aplicativos que executam as principais funções do supervisor de controle, tais como controle de juntas, modelamento cinemático e comunicação entre placas, foram desenvolvidos e utilizados outros programas utilitários tanto na fase de desenvolvimento do protótipo como na experimental. Este anexo descreve estes programas, e, para alguns apresenta uma listagem dos mesmos.

A - Programas utilizados na fase de desenvolvimento

A.1 - Teste de Barramento de Endereços (BARRA)

O BARRA é um programa bastante simples gerado em Assembler 8088 e auxilia o desenvolvimento de placas que utilizam microprocessador 8088 durante a fase inicial de testes.

Seu princípio de funcionamento é o de efetuar a varredura dos 16 bits mais significativos do barramento de endereço do processador (bits 4 a 19). O 1º acesso é feito na posição 0FFFEH, onde todos os bits estão em nível 1, exceto o bit 0, que fica em nível 0. No próximo acesso, é feita uma rotação dos bits, sendo acessada a posição 0FFFDH, e assim por diante, até que todas as linhas sejam testadas. Este algoritmo permite a detecção de curto circuito entre as linhas do barramento de endereço.

A ativação do programa se dá pelo sinal de RESET, uma vez que na fase inicial de testes não pode ser garantido a operação das funções mais complexas da placa tais como acesso à memória e ao Controlador de linha Serial. Por este mesmo motivo, é necessário o uso de um Analisador Lógico que possibilite a visualização dos sinais ao longo do tempo. O número de linhas testadas está limitado ao número máximo de sinais que podem ser observados no Analisador Lógico.

Abaixo apresentamos o código fonte do programa:

```
"8088"
;BARRA - PROGRAMA DE TESTE DE BARRAMENTO
        ORG     0FFFF0000H              ;ENDERECO DE RESET
;
INICIO: MOV     BX,00H                  ;POSICAO INICIAL
        MOV     SI,0FFFEH
BACK:   MOV     DS,SI
        MOV     [BX],AL
        ROL     SI,1                    ;ROTACAO DOS BITS
        JMP     BACK
```

A.2 - Teste de Barramento de Dados (DADOS)

O programa DADOS efetua o teste de todas os bits de dados do processador, possibilitando a verificação de curto circuitos entre os sinais do barramento. Como no processador 8088 as linhas de endereço menos significativos (bits 0 a 7) são multiplexadas com as linhas de dados, este teste atua de maneira a complementar o programa BARRA, testando os bits 0 a 7.

O princípio de funcionamento é o de varrer todas as combinações de dados, colocando todos os bits em nível 1 e apenas um deles em nível 0 a cada ciclo de escrita. O programa fonte é mostrado abaixo:

```
"8088"
;DADOS - TESTE DE BARRAMENTO DE DADOS
        ORG     0FFFF0000H
        MOV     AL,0111
        MOV     BX,0011
BACK:   MOV     [BX],AL
        ROL     AL,01
        JMP     BACK
```

B - Programas utilizados na fase de testes do protótipo

B.1 - Programa de geração de pontos de trajetória em malha aberta

Este *software*, desenvolvido em BASIC, faz a geração dos pontos que definem uma trajetória para que sejam enviados a uma junta do robô. No caso do protótipo em teste, a saída do programa são enviados em forma de uma palavra pela porta paralela do PC, para atuar na UAM, de modo que esta envie os sinais referentes à direção e ao sentido do acionamento.

Este programa foi utilizada na etapa de teste acionamento do PROTOROB1, em malha aberta, descrito no Capítulo 4, item 4.3.2.

B.2 - Programa de tratamento dos sinais dos *encoders*.

Este *software* atua de maneira conjunta ao programa de aquisição de pontos descrito no Capítulo 4, item 4.2.2, fazendo a conversão dos pontos lidos a partir dos *encoders* para sinais de sentido e variação que possam ser tratados pelo *software* de visualização gráfica (GRAPHER).

ANEXO IV

PROGRAMAS DE GERAÇÃO AUTOMATICA DE TRAJETORIAS

A - DESLOCAMENTO

```
program    desloc;

uses   crt  ;
     var
     RESPOSTA : char;
     V1,  V2,  V3,  i,  j,  var_aux  :  word;
     VEL2, VEL3  :  real;
     VELOC_MAX, PORVEL, DELT2, DELT3 : real ;
     AC2, AC3, TAC2, TAC3, TVEL2, TVEL3 : real ;
     S2,  S3,  S21,  S31,  S22,  S32,  S23,  S33  :  real  ;
     VARIA, FATOR, NB_TOT2, NB_TOT3, CONT : real;
     nom  :  string  [  5  ]  ;
     nom1:   string   [15];
     f   :   text;

begin

     RESPOSTA :='s';
     clrscr;
     writeln('*************************************************************');
     writeln('*********** PROGRAMA Tratamento de Trajetorias  ********');
     writeln('********       geracao de um arquivo de pontos ASCII   ******');
     writeln('*************************************************************');
     writeln;
     writeln;
     writeln;
     writeln;
     write('entre  o  nome  do  arquivo  a  ser  tratado  :  ');readln(nom);
     nom1  :=  nom  +  '2'  +  '.TRA'  ;
     Assign(f,  nom1   );
     rewrite     (f);
```

```
clrscr;
writeln('*************************************************');
writeln('*************** Parametros do robo ***************');
writeln('*************************************************');
writeln;
writeln;
writeln;
writeln;

writeln('---- juntas ----          --- 2 ---          --- 3 --- ');
writeln;
writeln('Tempo de acel. (ms)        250                250      ');
writeln('Vel. Maxima (I/s)          200                200      ');
writeln;
PORVEL := 1;
VELOC_MAX := 200;
VELOC_MAX := VELOC_MAX*PORVEL;
TAC2 := 0.25;
TAC3 := 0.25;

write('Deslocamento desejado da junta 2 (impulsos) : ');
readln(DELT2);
while (DELT2 < 0) OR (DELT2 > 2000) do
begin
  write('dado incorreto !!!          junta 2 (impulsos) : ');
  readln(DELT2);
end;
write('Deslocamento desejado da junta 3 (impulsos) : ');
readln(DELT3);
while (DELT3 < 0) OR (DELT3 > 2000) do
begin
  write('dado incorreto !!!          junta 3 (impulsos) : ');
  readln(DELT3);
end;
```

```
if DELT3 >= DELT2 then
  begin
    VEL3 := VELOC_MAX ;
    S31 := TAC2 * VEL3 / 2 ;

    S32 := DELT3 - 2 * S31 ;
    S33 := S31 ;
    TVEL3 := (S32 / VEL3) ;
    TVEL2 := TVEL3 ;
    VEL2 := DELT2 / ( TAC2 + TVEL2 );
    S21 := TAC2 * VEL2 / 2 ;
    S22 := DELT2 - 2 * S21 ;
    S23 := S21 ;
end ;

if DELT3 < DELT2 then
 begin
    VEL2 := VELOC_MAX ;
    S21 := TAC3 * VEL2 / 2 ;
    S22 := DELT2 - 2 * S21 ;
    S23 := S21 ;
    TVEL2 := (S22 / VEL2) ;
    TVEL3 := TVEL2 ;
    VEL3 := DELT3 / ( TAC3 + TVEL3 );
    S31 := TAC3 * VEL3 / 2 ;
    S32 := DELT3 - 2 * S31 ;
    S33 := S31 ;
 end ;

    Nb_tot2 := S21 + S22 + S23 ;
    Nb_tot3 := S31 + S32 + S33 ;
    AC2 := (VEL2/TAC2);
    AC3 := (VEL3/TAC3);
```

```
if RESPOSTA  = 's' then
   begin
    clrscr;
    writeln('*************************************************');
    writeln('***************** RESULTADOS ********************');
    writeln('*************************************************');
    writeln;
    writeln('----- axes -----          --- 2 ---          --- 3 --- ');
    writeln;
    writeln('Tempo aceleracao (ms)  ', TAC2 * 1000:7:0 , TAC3 * 1000:18:0 );
    writeln('Tempo velocidade (ms)  ', TVEL2 * 1000:7:0 , TVEL3 * 1000:18:0 );
    writeln('Tempo frenagem (ms)    ', TAC2 * 1000:7:0 , TAC3 * 1000:18:0 );
    writeln('Velocidade Maxima (I/s)', VEL2:7:0 , VEL3:18:0 );
    writeln('Aceleracao (I/s2)      ', AC2:7:0 , AC3:18:0 );
    writeln('No Impulsoes aceleracao', S21:7:0 , S31:18:0 );
    writeln('No Impulsoes velocidade', S22:7:0 , S32:18:0 );
    writeln('No Impulsoes frenagem  ', S23:7:0 , S33:18:0 );
    writeln('No Impulsoes total     ', Nb_tot2:7:0 , Nb_tot3:18:0 );
    writeln;
 end;

if RESPOSTA  = 's' then
    begin
    write('Criacao do arquivo trajetoria). ? (S/N) ');
    readln (RESPOSTA );
end;
```

```
if (RESPOSTA  = 'S') OR (RESPOSTA  = 's') then
  begin
  clrscr;
  writeln;
  write('No de pontos do arquivo : ');
  readln (FATOR);
  writeln;
  writeln;
  writeln;
  writeln('*****************************************************************');
  writeln('*********************  juntas 2,3  ***************************');
  writeln('*****************************************************************');
  writeln;
  writeln;
  writeln('criacao do arquivo trajetoria em andamento ...');
  writeln;
  writeln;
  CONT := 0 ;
  S2 := 0 ;
  S3 := 0 ;
  var_aux := trunc ( TAC2 * 1000 / FATOR ) ;

  for j := 0 to var_aux  do
   begin
      VARIA := CONT/1000 ;
      S2 := AC2 * VARIA*VARIA/2 ;
      S3 := AC3 * VARIA*VARIA/2 ;
{       writeln ( CONT , S2 , S3 ) ; }
      V1 := trunc (CONT) ;
      V2 := trunc (S2) ;
      V3 := trunc (S3) ;
      write ( f , V1 );
      write ( f , #9 ) ;
      write ( f , V2 ) ;
      write ( f , #9 ) ;
```

```
      write ( f , V3 ) ;
      write ( f , #9 ) ;
      writeln ( f ) ;
      CONT := (j + 1) * FATOR;
    end ;

var_aux := trunc ( TVEL2 * 1000 / FATOR) ;

 CONT := 0 ;
 for j := 0 to var_aux - 1 do
  begin
    CONT := ( j + 1 ) * FATOR ;
    VARIA := CONT/1000 ;
    S2 := VEL2 * VARIA ;
    S3 := VEL3 * VARIA ;
    V1 := trunc (CONT + TAC2 * 1000) ;
    V2 := trunc (S2 + S21) ;
    V3 := trunc (S3 + S31) ;
    write ( f , V1 ) ;
    write ( f , #9 ) ;
    write ( f , V2 ) ;
    write ( f , #9 ) ;
    write ( f , V3 ) ;
    write ( f , #9 ) ;
    writeln ( f ) ;
end ;
var_aux := trunc ( TAC2 * 1000 / FATOR) ;
for j := 0 to var_aux - 1 do
 begin
   CONT := ( j + 1 ) * FATOR ;
   VARIA := CONT / 1000 ;
   S2 := VEL2 * VARIA - AC2 * VARIA*VARIA/2 ;
   S3 := VEL3 * VARIA - AC3 * VARIA*VARIA/2 ;
   V1 := trunc ( CONT + (TVEL2 + TAC2) * 1000) ;
   V2 := trunc ( S2 + S21 + S22 ) ;
```

```
        V3 := trunc ( S3 + S31 + S32 ) ;
        write ( f , V1 ) ;
        write ( f , #9 ) ;
        write ( f , V2 ) ;
        write ( f , #9 ) ;
        write ( f , V3 ) ;
        write ( f , #9 ) ;
        writeln ( f ) ;
    end ;
end ;
clrscr;

    close    (f);
     writeln;
     writeln('fim    do    programa');
     writeln;
end .
```

B - VELOCIDADE

```
program   veloc;

uses  crt  ;
     var
     RESPOSTA : char;
     V1, V2, V3, i, j, var_aux  :  word;
     VEL2, VEL3  :  real;
     VELOC_MAX, PORVEL, DELT2, DELT3 : real ;
     AC2, AC3, TAC2, TAC3, TVEL2, TVEL3 : real ;
     VELOC2, VELOC3, S21, S31, S22, S32, S23, S33 : real ;
     VARIA, FATOR, NB_TOT2, NB_TOT3, CONT : real;
     nom  :  string  [  5  ]  ;
     nom1:   string   [15];
     f   :   text;

begin

     RESPOSTA :='s';
     clrscr;
     writeln('*************************************************************');
     writeln('*********** PROGRAMA Tratamento de Velocidades  ********');
     writeln('********       geracao de um arquivo de pontos ASCII   ******');
     writeln('*************************************************************');
     writeln;
     writeln;
     writeln;
     writeln;
     write('entre  o  nome  do  arquivo  a  ser  tratado  :  ');readln(nom);
     nom1 := nom + '2' + '.VEL' ;
     Assign(f,  nom1   );
     rewrite     (f);
```

```
clrscr;
writeln('************************************************');
writeln('*************** Parametros do robo ***************');
writeln('************************************************');
writeln;
writeln;
writeln;
writeln;
writeln('---- juntas ----          --- 2 ---          --- 3 --- ');
writeln;
writeln('Tempo de acel. (ms)        250                250      ');
writeln('Vel. Maxima (I/s)          200                200      ');
writeln;

PORVEL := 1;
VELOC_MAX := 200;
VELOC_MAX := VELOC_MAX*PORVEL;
TAC2 := 0.25;
TAC3 := 0.25;

write('Deslocamento desejado da junta 2 (impulsos) : ');
readln(DELT2);
while (DELT2 < 0) OR (DELT2 > 2000) do
begin
  write('dado incorreto !!!          junta 2 (impulsos) : ');
  readln(DELT2);
end;
write('Deslocamento desejado da junta 3 (impulsos) : ');
readln(DELT3);
while (DELT3 < 0) OR (DELT3 > 2000) do
begin
  write('dado incorreto !!!          junta 3 (impulsos) : ');
  readln(DELT3);
end;
```

```
if DELT3 >= DELT2 then
   begin
      VEL3 := VELOC_MAX ;
      S31 := TAC2 * VEL3 / 2 ;
      S32 := DELT3 - 2 * S31 ;
      S33 := S31 ;
      TVEL3 := (S32 / VEL3) ;
      TVEL2 := TVEL3 ;
      VEL2 := DELT2 / ( TAC2 + TVEL2 );
      S21 := TAC2 * VEL2 / 2 ;
      S22 := DELT2 - 2 * S21 ;
      S23 := S21 ;
end ;

if DELT3 < DELT2 then
 begin
      VEL2 := VELOC_MAX ;
      S21 := TAC3 * VEL2 / 2 ;
      S22 := DELT2 - 2 * S21 ;
      S23 := S21 ;
      TVEL2 := (S22 / VEL2) ;
      TVEL3 := TVEL2 ;
      VEL3 := DELT3 / ( TAC3 + TVEL3 );
      S31 := TAC3 * VEL3 / 2 ;
      S32 := DELT3 - 2 * S31 ;
      S33 := S31 ;
 end ;

      Nb_tot2 := S21 + S22 + S23 ;
      Nb_tot3 := S31 + S32 + S33 ;
      AC2 := (VEL2/TAC2);
      AC3 := (VEL3/TAC3);

 if RESPOSTA = 's' then
   begin
```

```
    clrscr;
    writeln('*************************************************');
    writeln('****************** RESULTADOS ********************');
    writeln('*************************************************');
    writeln;
    writeln('----- axes -----          --- 2 ---          --- 3 --- ');
    writeln;
    writeln('Tempo aceleracao (ms)  ', TAC2 * 1000:7:0 , TAC3 * 1000:18:0 );
    writeln('Tempo velocidade (ms)  ', TVEL2 * 1000:7:0 , TVEL3 * 1000:18:0 );
    writeln('Tempo frenagem (ms)    ', TAC2 * 1000:7:0 , TAC3 * 1000:18:0 );
    writeln('Velocidade Maxima (I/s)', VEL2:7:0 , VEL3:18:0 );
    writeln('Aceleracao (I/s2)      ', AC2:7:0 , AC3:18:0 );
    writeln('No Impulsoes aceleracao', S21:7:0 , S31:18:0 );
    writeln('No Impulsoes velocidade', S22:7:0 , S32:18:0 );
    writeln('No Impulsoes frenagem  ', S23:7:0 , S33:18:0 );
    writeln('No Impulsoes total     ', Nb_tot2:7:0 , Nb_tot3:18:0 );
    writeln;
end;

if RESPOSTA = 's' then
    begin
    write('Criacao do arquivo trajetoria) ? (S/N) ');
    readln (RESPOSTA );
end;

if (RESPOSTA = 'S') OR (RESPOSTA = 's') then
  begin
  clrscr;
  writeln;
  write('No de pontos do arquivo : ');
  readln (FATOR);
  writeln;
  writeln;
  writeln;
```

```
writeln('*************************************************************');
writeln('*********************** juntas 2,3 ***************************');
writeln('*************************************************************');
writeln;
writeln;
writeln('criacao do arquivo trajetoria em andamento ...');
writeln;
writeln;
CONT := 0 ;

var_aux := trunc ( TAC2 * 1000 / FATOR ) ;

for j := 0 to var_aux  do
 begin
     VARIA := CONT/1000 ;
     VELOC2 := AC2 * VARIA ;
     VELOC3 := AC3 * VARIA ;
     V1 := trunc (CONT) ;
     V2 := trunc (VELOC2) ;
     V3 := trunc (VELOC3) ;
     write ( f , V1 );
     write ( f , #9 ) ;
     write ( f , V2 ) ;
     write ( f , #9 ) ;
     write ( f , V3 ) ;
     write ( f , #9 ) ;
     writeln ( f ) ;
     CONT := (j + 1) * FATOR;
  end ;

var_aux := trunc ( TVEL2 * 1000 / FATOR) ;

 CONT := 0 ;
 for j := 0 to var_aux - 1 do
  begin
    CONT := ( j + 1 ) * FATOR ;
```

```
        VARIA := CONT/1000 ;
        VELOC2 := VEL2 ;
        VELOC3 := VEL3 ;
        V1 := trunc (CONT + TAC2 * 1000) ;
        V2 := trunc (VELOC2 ) ;
        V3 := trunc (VELOC3 ) ;
        write ( f , V1 ) ;
        write ( f , #9 ) ;
        write ( f , V2 ) ;
        write ( f , #9 ) ;
        write ( f , V3 ) ;
        write ( f , #9 ) ;
        writeln ( f ) ;
    end ;

    var_aux := trunc ( TAC2 * 1000 / FATOR) ;
    for j := 0 to var_aux - 1 do
     begin
        CONT := ( j + 1 ) * FATOR ;
        VARIA := CONT / 1000 ;
        VELOC2 := VEL2 - AC2 * VARIA ;
        VELOC3 := VEL3 - AC3 * VARIA ;
        V1 := trunc ( CONT + (TVEL2 + TAC2) * 1000) ;
        V2 := trunc ( VELOC2 ) ;
        V3 := trunc ( VELOC3 ) ;
        write ( f , V1 ) ;
        write ( f , #9 ) ;
        write ( f , V2 ) ;
        write ( f , #9 ) ;
        write ( f , V3 ) ;
        write ( f , #9 ) ;
        writeln ( f ) ;
    end ;
end ;
clrscr;
```

```
  close     (f);
    writeln;
    writeln('fim    do    programa');
    writeln;
end .
```