ESTE EXEMPLAR CORRESPONDE A REDAÇÃO FINAL DA TESE DEFENDIDA POR André Mendeleck E APROVADA PELA COMISSÃO JULGADORA EM 22 07 91.

Douglas G. Zgui
ORIENTADOR

MENDELECK

UMA LINGUAGEM PARA PROGRAMAÇÃO OFF-LINE DE ROBÔS

013/91

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
DEPARTAMENTO DE MECÂNICA COMPUTACIONAL
JULHO 1991

UMA LINGUAGEM PARA PROGRAMAÇÃO OFF-LINE DE ROBÔS

POR: André Mendeleck
ORIENTADOR: Prof. Dr. Douglas Eduardo Zampieri

Dissertação apresentada à Faculdade de Engenharia Mecânica FEM - UNICAMP como parte dos requisitos exigidos para obtenção do título de MESTRE em MECÂNICA DOS SÓLIDOS.

CAMPINAS - 1991

UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA MECÂNICA

TESE DE: MESTRADO

TÍTULO DA TESE: UMA LINGUAGEM PARA PROGRAMAÇÃO OFF-LINE DE ROBÔS

AUTOR: ANDRÉ MENDELECK

ORIENTADOR: PROF. DR. DOUGLAS EDUARDO ZAMPIERI

APROVADO POR

Douglas E. Zampieri

Prof. Dr. Douglas Eduardo Zampieri ,Presidente
UNICAMP - UNIVERSIDADE ESTADUAL DE CAMPINAS
FEM -Faculdade de Engenharia Mecânica
DMC- Departamento de Mecânica Computacional

Prof. Dr. Paulo Eigi Miyagi
USP - UNIVERSIDADE DE SÃO PAULO
EPUSP - Escola Politécnica da USP
DEM - Departamento de Engenharia Mecânica

Prof. Dr. Alberto Cliquet Jr.
UNICAMP - UNIVERSIDADE ESTADUAL DE CAMPINAS
FEE - Faculdade de Engenharia Elétrica
DEB - Departamento de Engenharia Biomédica

Campinas, 22 de Julho de 1991

Este trabalho contou com o apoio financeiro do

CNPq-CONSELHO NACIONAL DE DESENVOLVIMENTO CIENTÍFICO E TECNOLÓGICO

DEDICO ESTA DISSERTAÇÃO AOS MEUS PAIS

AGRADECIMENTOS:

- à Maria Tereza pela sua amizade e colaboração dispensados durante o período de realização deste trabalho;

- ao colega Carlos Florian Heuberger pela sua amizade, dicas e conversas sobre computação;

- à Maria Elena Pousa Seara pela amizade, apoio logístico e burocrático;

- ao colega Fernando Pinelli Cardoso pelo apoio na manutenção da infra-estrutura computacional;

- e finalmente, ao Prof. Douglas Eduardo Zampieri pela seriedade na conduta da orientação deste trabalho, amizade, respeito e incentivo.

RESUMO

Nesta dissertação é especificado e implementado o protótipo de uma linguagem de programação e um sistema de CAD-Simulação para um robô manipulador antropomórfico com 6 graus de liberdade.

Trata-se de uma linguagem de alto nível destinada a programação de movimentos a nível de manipulador e um sistema de CAD-Simulação que permite a visualização, em "tempo real", dos movimentos do robô em um terminal gráfico, sendo também, implementado um módulo de teach-in e um módulo de simulação apresentando informações sobre a evolução cinemática e dinâmica dos links e garra do robô.

ABSTRACT

This dissertation contains the specification and implementation of a prototype language program and a CAD-Simulation system for an antropomorphic manipulator with six degrees of freedom. The system has a high level language for motion program, a CAD-Simulation system to visualize robot motion in "real time", a teach-in and a simulation sub-system to visualize the kinematic and dynamic evolution of the links and the grasp.

ÍNDICE

RELAÇÃO DE FIGURAS

PARTE A

PARTE B

PARTE C

PARTE D

F43 - EXECUÇÃO
F44 - DESLOCAMENTO DA JUNTA 1
F45 - DESLOCAMENTO DA JUNTA 2
F46 - DESLOCAMENTO DA JUNTA 3
F47 - VELOCIDADE DA JUNTA 1
F48 - VELOCIDADE DA JUNTA 2
F49 - VELOCIDADE DA JUNTA 3
F50 - ACELERAÇÃO DA JUNTA 1
F51 - ACELERAÇÃO DA JUNTA 2
F52 - ACELERAÇÃO DA JUNTA 3
F53 - DESLOCAMENTO DA GARRA NA DIREÇÃO X
F54 - DESLOCAMENTO DA GARRA NA DIREÇÃO Y
F55 - DESLOCAMENTO DA GARRA NA DIREÇÃO Z
F56 - VELOCIDADE DA GARRA NA DIREÇÃO X
F57 - VELOCIDADE DA GARRA NA DIREÇÃO Y
F58 - VELOCIDADE DA GARRA NA DIREÇÃO Z
F59 - ACELERAÇÃO DA GARRA NA DIREÇÃO X
F60 - ACELERAÇÃO DA GARRA NA DIREÇÃO Y
F61 - ACELERAÇÃO DA GARRA NA DIREÇÃO Z
F62 - TORQUE TOTAL NA JUNTA 1
F63 - TORQUE TOTAL NA JUNTA 2
F64 - TORQUE TOTAL NA JUNTA 3
F65 - TORQUE DEVIDO A COMPONENTE DE FORÇA DE GRAVIDADE NA JUNTA 1
F66 - TORQUE DEVIDO A COMPONENTE DE FORÇA DE GRAVIDADE NA JUNTA 2
F67 - TORQUE DEVIDO A COMPONENTE DE FORÇA DE GRAVIDADE NA JUNTA 3
F68 - TORQUE DEVIDO A COMPONENTE DE FORÇA CENTRÍFUGA E CORIOLIS NA JUNTA 1
F69 - TORQUE DEVIDO A COMPONENTE DE FORÇA CENTRÍFUGA E CORIOLIS NA JUNTA 2
F70 - TORQUE DEVIDO A COMPONENTE DE FORÇA CENTRÍFUGA E CORIOLIS NA JUNTA 3
F71 - TORQUE DEVIDO A COMPONENTE DE INÉRCIA NA JUNTA 1
F72 - TORQUE DEVIDO A COMPONENTE DE INÉRCIA NA JUNTA 2
F73 - TORQUE DEVIDO A COMPONENTE DE INÉRCIA NA JUNTA 3

PROLEGÔMENO

OBJETIVOS

Os objetivos principais desta dissertação são: especificação e implementação do protótipo de uma linguagem de programação e um sistema de CAD-Simulação para um robô manipulador antropomórfico com 6 graus de liberdade.

Trata-se de uma linguagem de alto nível destinada a programação de movimentos a nível de manipulador e um sistema de CAD-Simulação que permite a visualização, em "tempo real", dos movimentos do robô em um terminal gráfico, sendo também, implementado um módulo de teach-in e um módulo de simulação, apresentando informações sobre a evolução cinemática e dinâmica dos links e garra do robô. Os seguintes tópicos são abordados:

- modelamento geométrico direto-inverso,
- modelamento dinâmico: Newton-Euler,
- linguagem de programação,
- CAD animação e simulação gráfica dos movimentos do robô.

A motivação por este tema reside no fato de existir uma lacuna entre a utilização de linguagens de programação comerciais e os modelos teóricos. Encontra-se farta documentação sobre as características de linguagens, enquanto que sua implementação não é divulgada. Assim, o desenvolvimento de novas linguagens fica prejudicado, pois, não existe uma "massa crítica" de pessoal com conhecimento adquirido e experiência em implementação.

Este é o primeiro trabalho em linguagens de programação de robôs desenvolvido no Departamento de Mecânica Computacional da Faculdade de Engenharia de Campinas - UNICAMP (Universidade Estadual de Campinas), sendo parte integrante de um projeto para o desenvolvimento de um robô manipulador [MENDELE-90] [ZAMPIER-91].

SISTEMA

O sistema, como mostrado na figura F1, está particionado em dois sub-sistemas:

- linguagem - compilador,
- C.A.D.-simulação.

No sub-sistema compilador, o usuário compila e/ou executa um programa da linguagem de programação. A compilação [AHO-86] consiste em transformar os códigos da linguagem em um código passível de interpretação pela CPU (Unidade Central de Processamento) do computador. A execução de um programa tem a função de permitir ao usuário a verificação e validação da tarefa e gerar uma sequência de dados, sinais de referência, para uso no sub-sistema de CAD-simulação ou pelo próprio usuário (por exemplo, para o hardware eletrônico de controle). No sub-sistema de CAD-simulação o usuário visualiza, no vídeo do computador, a tarefa que está sendo executada, tendo acesso a três módulos:

- CAD-animação
- simulação
- teach-in

No módulo de CAD-animação, o robô é apresentado, em animação gráfica, na forma de esqueleto e/ou sólido no espaço cartesiano tri-dimensional e em projeções nos planos cartesianos: XY, XZ e YZ, onde, a sequência de movimentos pode ser observada por um observador que "passeia" (alterando o ângulo e distância de observação) pela área de trabalho.

No módulo de simulação, o usuário pode observar o desenvolvimento da tarefa utilizando gráficos bi-dimensionais, fornecendo informações sobre a cinemática e dinâmica dos três primeiros graus de liberdade no espaço das juntas e, sobre a cinemática da garra no espaço cartesiano.

No módulo teach-in, utilizando animação gráfica, o usuário conduz o robô para posições-orientações desejadas, formando um conjunto de pontos que podem ser utilizados isoladamente ou em blocos na especificação de tarefas. Os pontos setados podem ser editados e/ou simulados (animação gráfica).

A sequência básica de utilização (passível de alteração) do sistema é a seguinte:

- digitação do programa,
- compilação e execução,
- CAD-animação ou modelamento ou teach-in,

O sistema está implementado em linguagem C [KELLEY-84] [KERNIGH-78] [PURDUM-84] [SCHILDT-87], compondo uma biblioteca de subrotinas de arquitetura aberta, onde o usuário tem acesso aos comandos da linguagem, aos parâmetros internos do sistema, ao fluxo de dados entre os módulos, ao protocolo de comunicação entre os sub-sistemas e aos comandos da linguagem C, podendo criar seus próprios comandos ou modificar os implementados. O hardware eletrônico utilizado é um microcomputador IBM-PC-AT (compatível) e sistema operacinal SISNEplus [ITAUTEC-88].

O acesso aos sub-sistemas é realizado pelos comandos:

- COMP , para compilação
- ANI , para animação.

Estes comandos consistem de arquivos tipo BAT [ITAUTEC-88] contendo os procedimentos necessários para sua utilização.

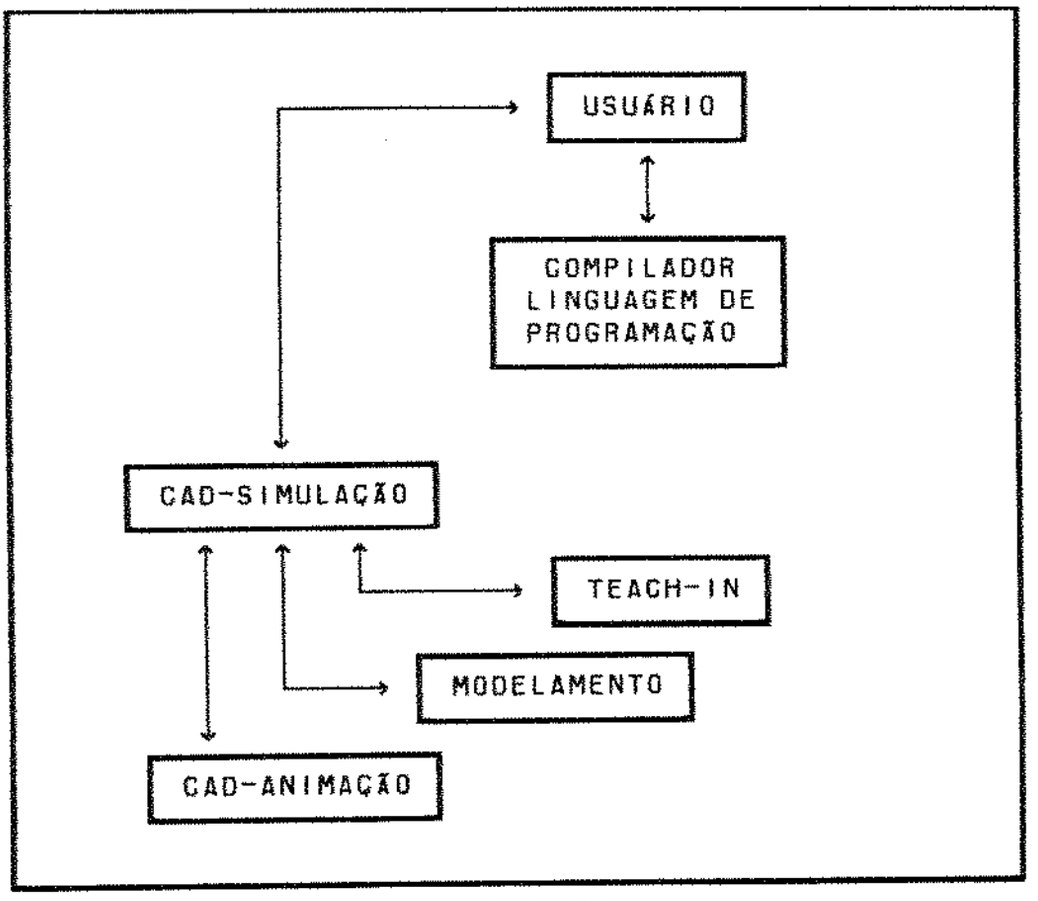

FIGURA F1 - ESTRUTURA DO SISTEMA

TEXTO

A dissertação está dividida em 6 partes. Na parte A é apresentada a descrição da linguagem, sua sintaxe e implementação. Na parte B é desenvolvido o modelamento geométrico direto e inverso, o gerador de trajetória e de movimento utilizado na implementação da linguagem. Na parte C é apresentado o sistema de CAD-Simulação que permite a visualização dos movimentos do robô e da evolução cinemática e dinâmica dos três primeiros links do robô. A parte D contém um programa exemplo utilizando a Linguagem de Programação e o sistema de CAD-Simulação. A parte E contém as conclusões e perspectivas e a parte F as referências.

PARTE A

LINGUAGEM

INTRODUÇÃO

A parte A contém a descrição da linguagem de programação (LP), sendo apresentados os comandos de programação, sua implementação e arquitetura interna. A implementação foi realizada para um robô antropomórfico de 6 graus de liberdade, como mostrado nas figuras F2, F3a e F3b

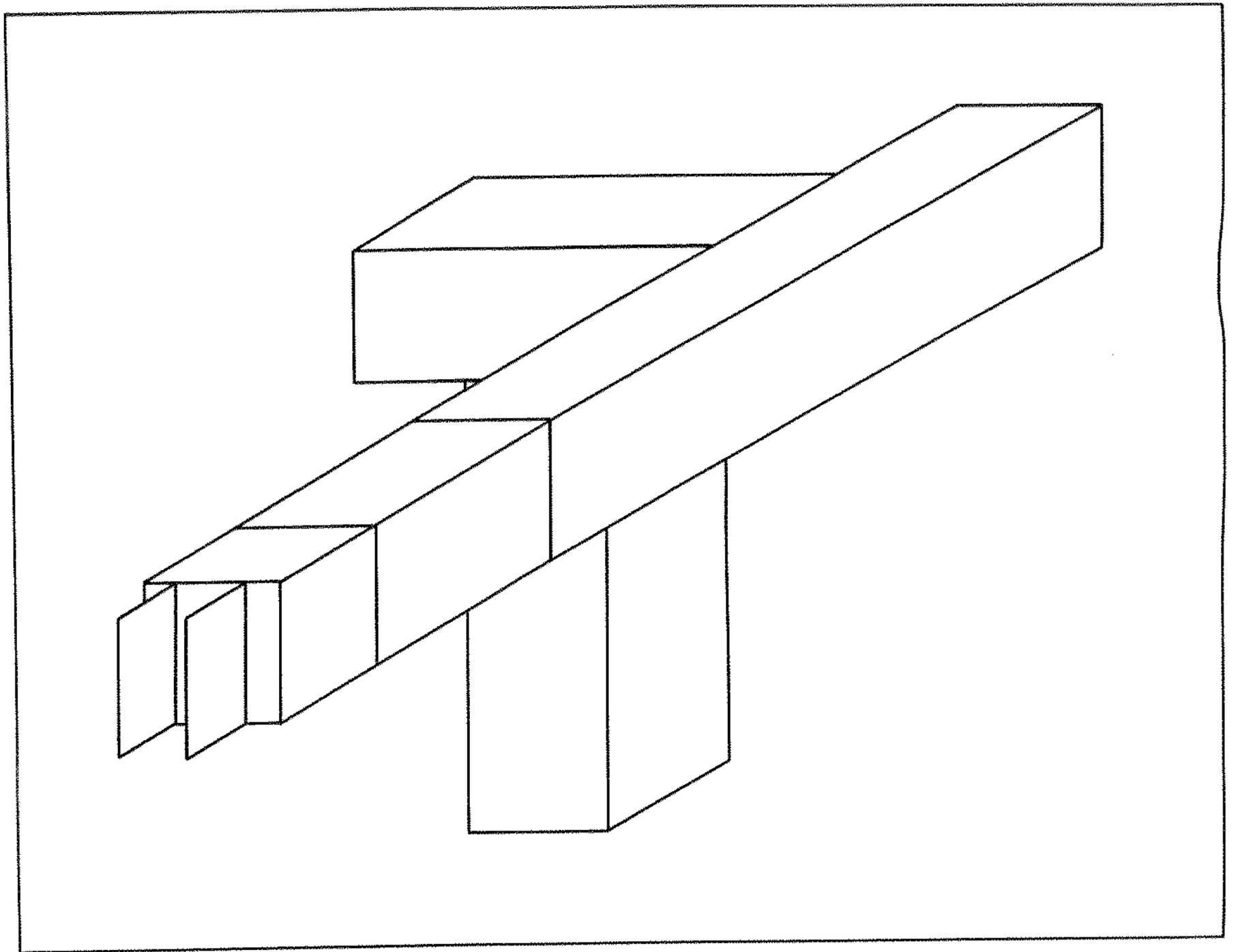

FIGURA F2 - LAYOUT DO ROBÔ

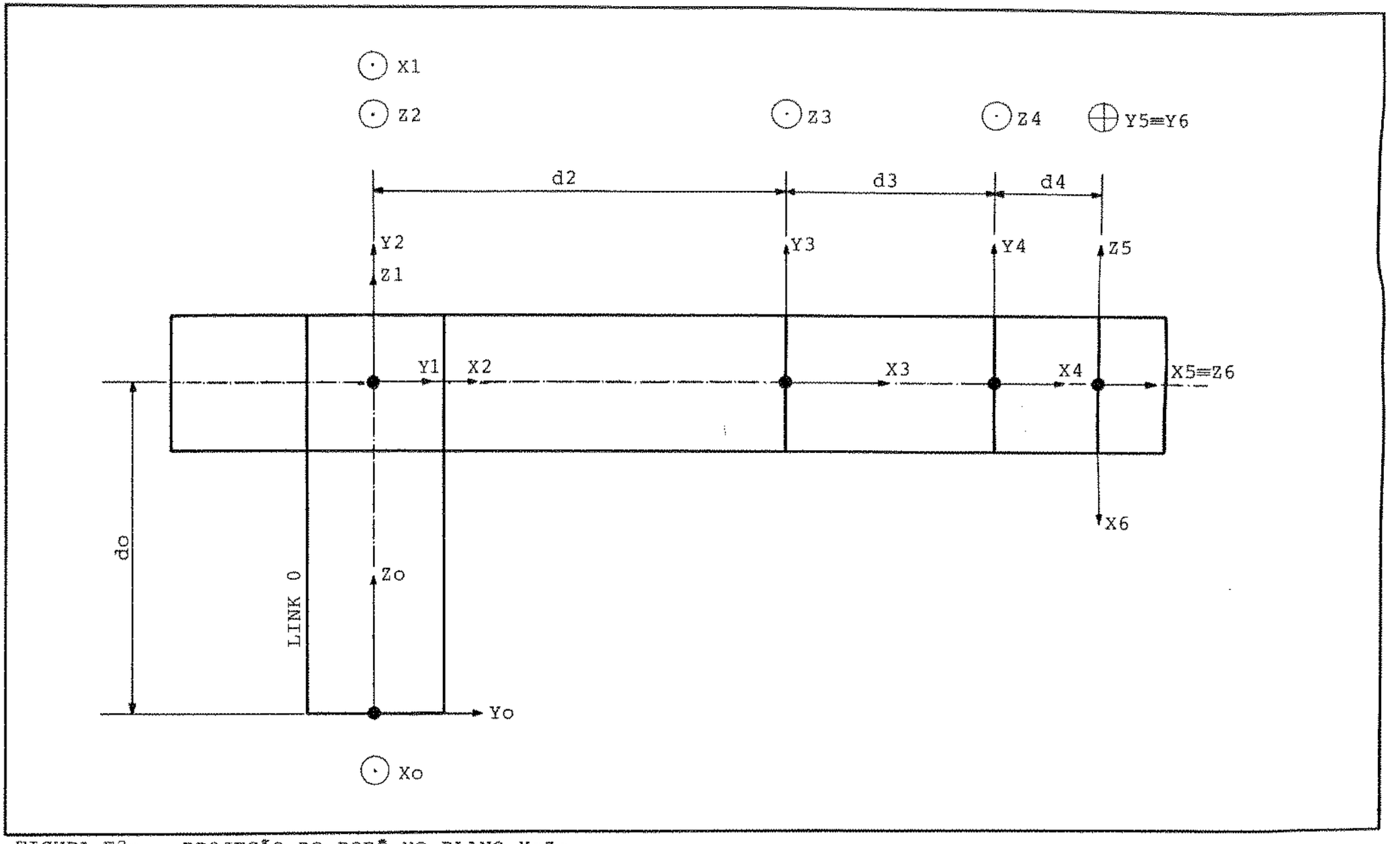

FIGURA F3a - PROJEÇÃO DO ROBÔ NO PLANO YoZo

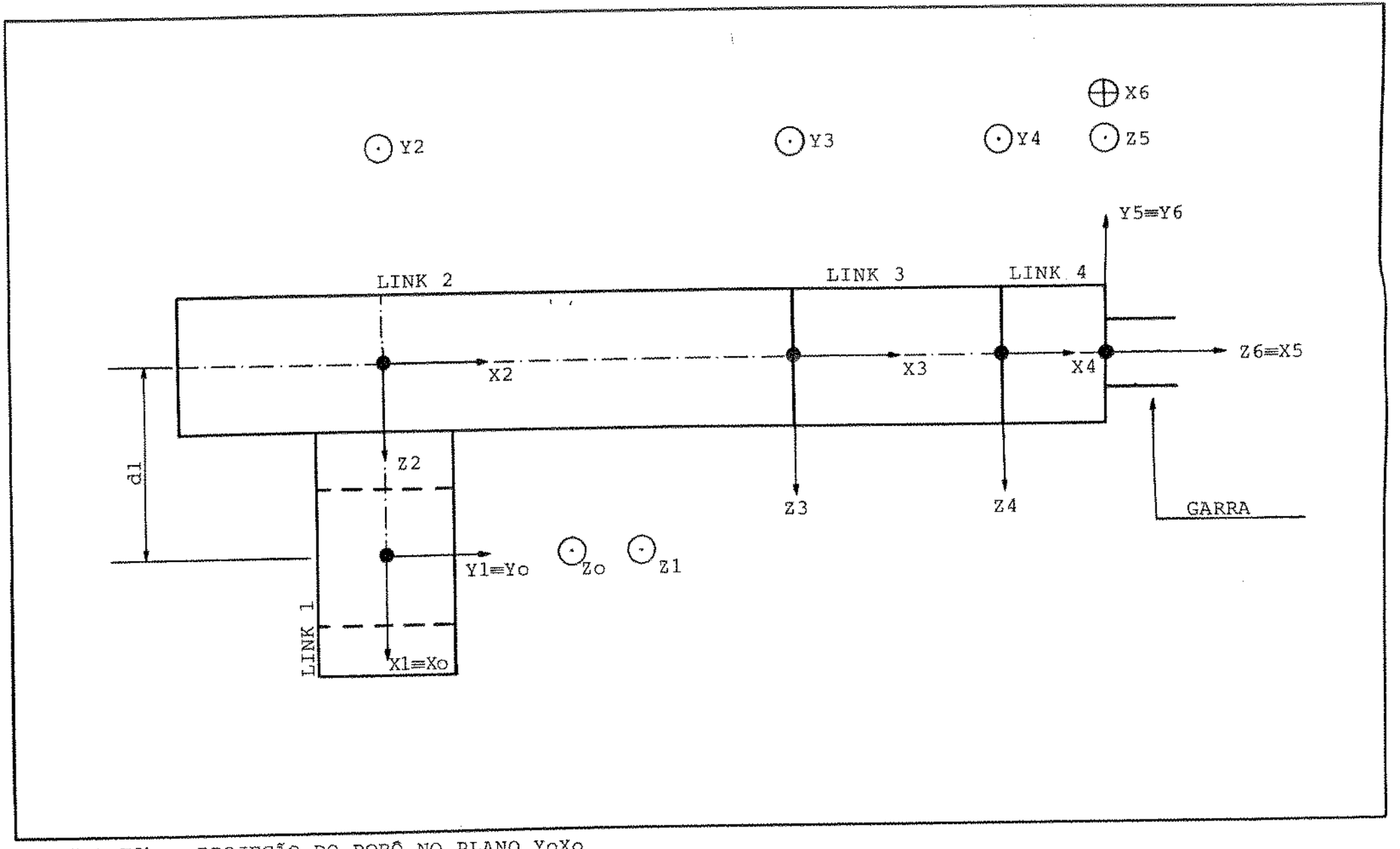

FIGURA F3b - PROJEÇÃO DO ROBÔ NO PLANO YoXo

LINGUAGENS PARA PROGRAMAÇÃO DE ROBÔS

A programação de robôs [AHMAD-88] [REMBOLD-87] [SARIDIS-85] [STORR-87] [LOZANO-83] [DILLMAN-90] [GOMAA-88] pode ser realizada por vários métodos, sendo normalmente utilizados os seguintes:

- seguidores mecânicos
- teclado
- teach-in
- play-back
- programação explícita
- programação implícita
- interação gráfica

O método mais primitivo de programação de robôs é utilizando seguidores mecânicos, onde a evolução de uma trajetória é realizada seguindo guias de referência, sensores de acionamento e fim de operações. Em substituição às guias, utilizam-se teclados (semelhante aos de computador) com capacidade de "memorização" sequenciada de operações. A programação é realizada especificando valores de referência para o hardware eletrônico de controle dos atuadores [NATO-87] [POST-90].

O teach-in consiste de uma "caixa de aprendizado" dotada de joysticks (ou equipamento similar) e teclado, permitindo que o robô seja guiado, pelo usuário, demarcando configurações e estados de referência que o robô deve seguir durante a realização das operações, de tal forma que uma trajetória possa ser interpolada utilizando as configurações fornecidas.

O método de play-back consiste em conduzir o robô manualmente "ensinando" as operações. Durante o procedimento de gravação, suas configurações e estados do efetuador (garra-ferramenta) são amostrados e armazenados. O processo de reprodução consiste em, utilizando os dados armazenados, permitir a reprodução dos movimentos do robô. Este método é semelhante ao utilizado em gravação-reprodução de som de fita magnética. O hardware eletrônico necessário para implementação é simples, não utilizando computadores de uso geral. Os programas são simples sequências de dados, normalmente não havendo possibilidade de loops, desvios condicionados e a sincronização com outros equipamentos.

A programação explícita é um método onde uma operação é implementada utilizando comandos de uma linguagem. Um comando consiste de uma ordem que será interpretada e executada, produzindo uma sequência de dados para o hardware eletrônico de controle, permitindo a execução dos movimentos desejados. As

linguagens de programação de robôs mais utilizadas são [McDONAL-86] [LOZANO-83] [TAYLOR-72] [KUSIAK-88] :

AL (Algorithmic Language) - Stanford University
AML (A Manufacturing Language) - IBM Corporation
APT (Automatically Programmed Tools) - MIT
AR-BASIC (American Robot BASIC) - American Robot Corporation
AUTOPASS (Automatic Parts Assembly System) - IBM Corporation
JARS -Jet Propulsion Laboratory
HELP - General Eletric Company
IRL (Intuitive Robot Language) - MICROBO
LRS (Lenguale para Robots con Sensores) - Politechnic University of Madrid
MAL - MILAN Polytechnic
MAPLE - IBM Corporation
MCL (Manufacturing Control Language) - McDonnell Douglas Corporation
PAL (Purdue Algorithmic Language) - Purdue University
RAIL (Robotic Automatix Incorporated Language) - Automatix, Inc.
ROL (Rhino Operating Language) - State University of New York (SUNY)
RoPL (Robot Programming Language) - E S-I, Inc., and Henderson Industries, Inc.
RPS/RPL (Robot Programming System) - SRI International
SIGLA (Sigma Language) - Olivetti Westinghouse
SRL (Strutured Robot Language) - University of Karlsruhe, Alemanha
TEACH - Bendix Corporation
VAL e VAL II (Versatile Algorithmic Language) - UNIMATION, Inc.

Consideram-se estas linguagens como de alto nível, pois, permitem uma programação "amigável", no entanto, não incorporam um grau mais elevado de "inteligência", como por exemplo, solução automática de tarefa, verificação e desvio de colisão [GOUZENE-84]. A utilização deste método de programação exige do usuário pleno conhecimento da tarefa e dos movimentos do robô, de tal forma, que possa dividi-la em operações mais simples, compondo uma sequência adequada de comandos disponíveis na linguagem. Assim, o usuário é o responsável pela verificação de colisão e integração com outros equipamentos [POLLMAN-87] [PRUENTE-87].

O método de programação implícita é semelhante à programação explícita. O usuário programa uma tarefa através de comandos, contudo, sua implementação é implícita. Um software especialista interpreta um comando gerando automaticamente a sequência adequada de operações, verificando colisão, sincronizando e integrando os equipamentos. Este tipo de linguagem incorpora comandos de altíssimo nível, bibliotecas de soluções, máquinas de estados [NILSSON-80] e uma base de conhecimento e de dados específicos da área de trabalho, dos equipamentos e objetos manipulados que compõe a célula de manufatura. Deve ainda, possuir uma estrutura de software para gerenciamento de comunicação e

interfaceamento com sensores que serão as únicas vias para monitoramento das operações e sincronismo de processos entre tarefas concorrentes, permitindo um controle cooperativo entre os equipamentos (por exemplo vários robôs e pallets) [SARIDIS-83].

Os dados dos equipamentos referem-se às características operacionais de programação e manufatura. Para a área de trabalho é necessário o conhecimento da localização geométrica dos elementos e seu volume de trabalho. Quanto aos objetos, o sistema deve conhecer suas características físicas: massa, densidade, inércia, material, textura, rugosidade, etc, ..., permitindo o melhor desempenho na realização dos movimentos e do interfaceamento com a garra (estabilidade, deformação, etc, ...), suas características geométricas permitindo a avaliação da forma de pega (pegar um tarugo de ferro ou um ovo exige abordagens distintas), relacionamento com outros objetos, avaliando, por exemplo, o grau de deformação e tolerâncias de contato.

As máquinas de estado devem incorporar, na base de conhecimento, estratégias para a abordagem de problemas específicos e novos (aprendizado), gerando, avaliando e selecionando a melhor solução [TAYLOR-72].

Antes da implementação e testes, a tarefa deve ser simulada em terminais gráficos permitindo sua visualização, validação e eventuais correções.

A interação gráfica consiste de um método de programação onde a interface homem-tarefa é realizada por meio de desenhos da peça a ser produzida. Conhecendo as características geométricas da peça em bruto e do produto final, um sistema especialista gera uma sequência intermediária de operações para a manufatura e montagem, cabendo ao usuário "somente" a concepção e projeto do produto. Assim, o sistema especialista deve apresentar algumas características que são fundamentais para a implementação desse método: uma interface homem-máquina que permita a correta definição da tarefa a ser realizada: uma base de procedimentos e métodos para a pesquisa e solução de problemas, sendo normalmente utilizados métodos de inteligência artificial [NILSSON-80]; uma base de conhecimentos específicos para cada equipamento (robôs, máquinas de controle numérico, etc...) permitindo a sua programação: uma base de dados sobre métodos de manufatura , montagem e inspeção: e por fim, uma base de dados sobre os materiais utilizados. Tudo isso integrado produz um software com as sequências de movimentos para que, por exemplo, o(s) robô(s) possa(m) realizar suas tarefas adequadamente [LOZANO-83].

A implementação do método de interação gráfica é muito custosa, pois, exige a utilização de computadores com grande capacidade de processamento, armazenagem e manipulação de dados.

LINGUAGEM DE PROGRAMAÇÃO

A linguagem de programação (LP) é uma linguagem textual, off-line para programação de um robô antropomórfico de 6 graus de liberdade, sendo composta de um conjunto de comandos implementados como subrotinas em linguagem C, de tal forma que um programa em LP é escrito como um programa em C, utilizando todos os recursos desta linguagem e, é neste ambiente de trabalho que o usuário deve implementar uma tarefa. A escolha por este ambiente deve-se ao fato de a linguagem C ser suficientemente poderosa para permitir a implementação de "novos comandos" transparecendo ao usuário tratar-se de uma nova linguagem, facilitando a portabilidade e interfaceamento com o hardware eletrônico.

As linguagens textuais de programação de robôs [AMBER-87] [BLUME-84] [BOBROW-73] [BRAJNIK-90] [CHARNIA-85] [KOZLOWS-89] [LEAKE-89] [LEVAS-89] [POCOCK-89] [PRITSCH-87] [SHEN-85] [SHEN-87] [SPUR-87] podem ser classificadas, segundo o seu "grau de inteligência", como:

- orientada a programação de movimentos,
- orientada a solução e implementação automática de tarefas.

Por não incorporar comandos para a solução e implementação automática de tarefas, a LP é classificada como uma linguagem de alto nível orientada a programação de movimentos, ou seja, uma tarefa é implementada explicitamente descrevendo-se a sequência de movimentos do robô. Contudo, incorpora alguns comandos para manipulação de objetos. Os comandos implementados permitem a programação em alto nível, entretanto, o usuário tem a sua disposição comandos que permitem o acesso a nível de implementação e execução dos movimentos (baixo nível).

As principais características da LP são:

- linguagem textual explícita, estruturada, compilada, com processamento em off-line,
- sintaxe dos comandos semelhante a linguagem C, incorporando todos os tipos de dados e estruturas de controle e fluxo,
- tipos de dados especiais: "frame" com posição-orientação da garra e valor angular das junta e, estrutura para manipulação de objetos,
- comandos de movimento: linear e livre no espaço cartesiano, linear e livre no espaço das juntas e, com orientação fixa,
- comandos de garra: setar um tipo, abrir e fechar,
- permite definir e manipular objetos,
- comandos de baixo nível: permite interferir na geração de trajetória, na geração de sinais de referência para o sistema de simulação (ou

sistema eletrônico de controle) tendo acesso aos parâmetros que geram os movimentos e a base de dados,
- incorpora um modo teach-in.

A figura F4 mostra o fluxo de informações no sistema. O usuário deve editar a tarefa em um editor de texto gerando um arquivo ASC II, compilar como um programa em linguagem C, "linkar" com a biblioteca da LP, gerando um código executável (programa), efetuar a execução do programa gerando uma tabela de sinais de referência (TSR) compatível ao sub-sistema de CAD-simulação. Para uma TSR gerada, o usuário pode utilizar diretamente este sub-sistema.

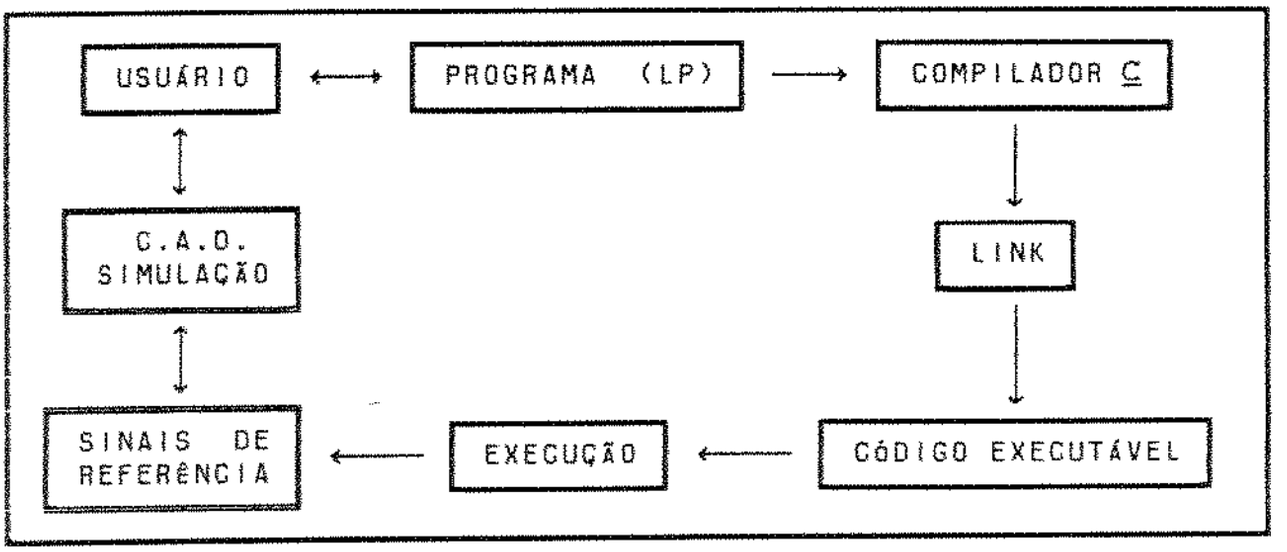

FIGURA F4 - FLUXO DE PROCEDIMENTOS NO SISTEMA

O programa deve estar contido no seguinte cabeçalho:

```
void program(void)
 {
   programa do usuário
 }
```

O comando COMP (arquivo BAT [ITAUTEC-88]) executa automaticamente os processos de compilação, linkagem e execução de um programa. O comando ANI (arquivo BAT) ativa o sistema de CAD-Simulação.

A figura F5 mostra a estrutura da LP, que está dividida em 5 níveis:

- nível 1 - comandos
- nível 2 - interface externa
- nível 3 - controlador interno
- nível 4 - mensagem
- nível 5 - objeto

No nível 1 - comandos, estão definidos os comandos básicos de programação da linguagem. O nível 2 - interface externa consiste de um conjunto de subrotinas de leitura e escrita de dados para o software de CAD-simulação. O nível 3 - controlador interno, tem a função de gerenciar internamente o fluxo de dados durante a execução dos comandos. O nível 4 - mensagem, consiste de um conjunto de subrotinas para o envio e definição de mensagens para e pelo usuário. O nível 5 - objeto é o responsável pelo gerenciamento de dados dos objetos que podem ser manipulados pelo robô.

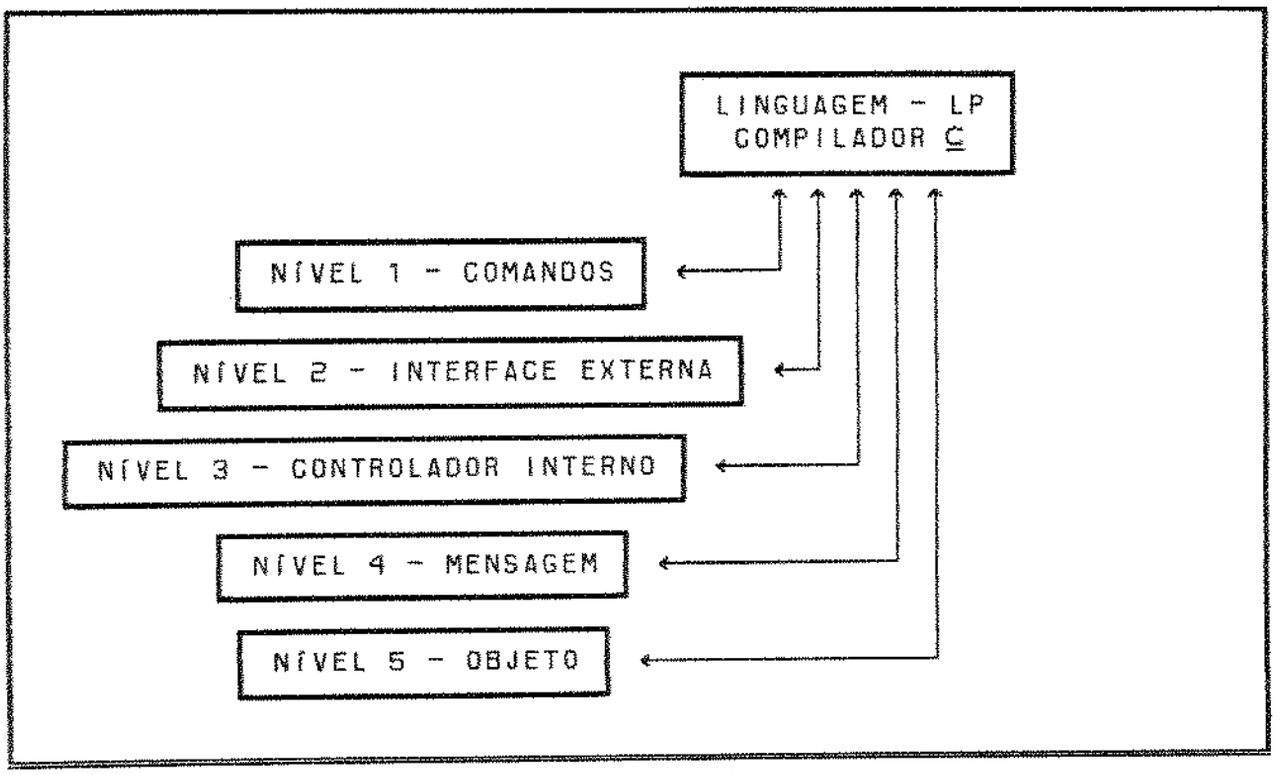

FIGURA F5 - ESTRUTURA DE NÍVEIS DA LINGUAGEM

O quadro a seguir, permite uma comparação das características da LP com algumas linguagens de programação de robôs.

QUADROS COMPARATIVO DE CARACTERÍSTICAS DE ALGUMAS LINGUAGENS [GOUVER-84] [SHIMANO-84]:

LINGUAGENS ⟶ / CARACTERÍSTICAS ↓	VAL	AL	AML	HELP	JARS	MCL	RAIL	RPL	SRL	LP
LINGUAGEM TEXTUAL	X	X	X	X	X	X	X	X	X	X
PROGRAMAÇÃO POR MEMU			X							
TIPO DE LINGUAGEM:										
DEF. POR SUBROTINAS					X					X
DEFINIDA COMO EXTENSÃO DE OUTRA						X				
NOVA LINGUAGEM	X	X	X	X			X	X	X	
TIPO DE DADOS:										
INTEIRO	X	X	X	X	X	X	X	X	X	X
REAL		X	X		X	X	X	X	X	X
FRAME	X	X			X	X	X		X	X
ÂNGULOS DE JUNTAS	X		X		X					X
VETOR		X	X		X	X			X	X
MATRIZ ORIENTAÇÃO	X	X			X	X			X	X
ESPECIFICAÇÃO DE ORIENTAÇÃO:										
MATRIZ ROTAÇÃO		X			X					X
ÂNGULO DE EULER	X	X	X		X		X			
ROLL-PITCH-YAW		X		X		X				
CONTROLE DE UM ROBÔ	X		X		X		X	X	X	X
CONTL. DE MÚLTIPLOS ROBÔS		X	X	X		X				

ESTRUT.DE CONTROLE:										
LABELS	X		X	X	X	X		X	X	X
IF-THEN	X	X	X	X	X	X	X	X	X	X
IF-THEN-ELSE		X	X	X	X	X	X	X	X	X
WHILE-DO		X	X	X	X	X	X	X	X	X
DO-UNTIL		X	X		X		X	X	X	X
CASE		X			X		X	X	X	X
FOR		X	X		X		X	X	X	X
BEGIN-END		X	X		X	X			X	
COBEGIN-COEND		X		X		X			X	
PROC.-FUNCT.-SUBROT.	X	X	X	X	X	X	X	X	X	X
MODOS DE CONTROLE:										
POSIÇÃO	X	X	X	X		X	X	X	X	
FORÇA	X	X							X	
COMPLIANCE		X			X				X	
SENSORES	X	X			X		X	X	X	
CONVEYOR TRACKING				X	X					
TIPOS DE MOVIMENTOS:										
PONTO A PONTO A NÍVEL DE JUNTA	X	X	X	X	X		X	X	X	X
LINHA RETA	X	X			X	X	X	X	X	X
SPLINE	X	X		X	X		X	X	X	
TRAJET. EM TEMPO REAL	X								X	
LINHA DE SINAIS:										
DIGITAL	X		X		X	X	X	X	X	
ANALÓGICO		X				X		X	X	

SUPORTE:										
PROCESSAMENTO PARALELO		X		X					X	
MULTI-TAREFA		X		X					X	
EDITOR DE TEXTO	X	X	X	X	X		X	X	X	
COMPILADOR		X			X	X		X	X	X
INTERPRETADOR	X	X	X	X			X	X		
SIMULADOR		X	X			X			X	X
HELP		X	X						X	
DEBUGGING		X	X	X		X	X	X	X	

NÍVEL 1 - COMANDOS

O nível 1 é o principal, sendo implementado com o auxílio dos outros quatro. Neste nível estão definidos os comandos, a tabela de parâmetros da linguagem (TPL), o modelamento geométrico direto e inverso, o gerenciador de movimento, o gerador de trajetória e os procedimentos de inicialização. A figura F6 mostra a estrutura deste nível.

A execução de um comando é realizada utilizando os dados da tabela de parâmetros da linguagem que contém informações de localização e estado do robô e de controle das rotinas internas da linguagem.

Os procedimentos de inicialização tem a função de inicializar os dados da TPL, sendo acessados pelos comandos de inicialização. O modelo geométrico direto fornece a posição-orientação da garra tomando como entrada o valor angular das juntas. Inversamente, o modelo geométrico inverso fornece o valor angular das juntas a partir da posição-orientação da garra, como mostrado na figura F7 . O gerenciador de movimentos tem a função de coordenar a geração dos sinais de referência angular das juntas, sendo auxiliado por um gerador de trajetória que efetua a discretização do movimento seguindo um perfil de velocidade e por dois módulos: movimento livre e movimento linear, que coordenam a geração destes dois tipos de movimentos.

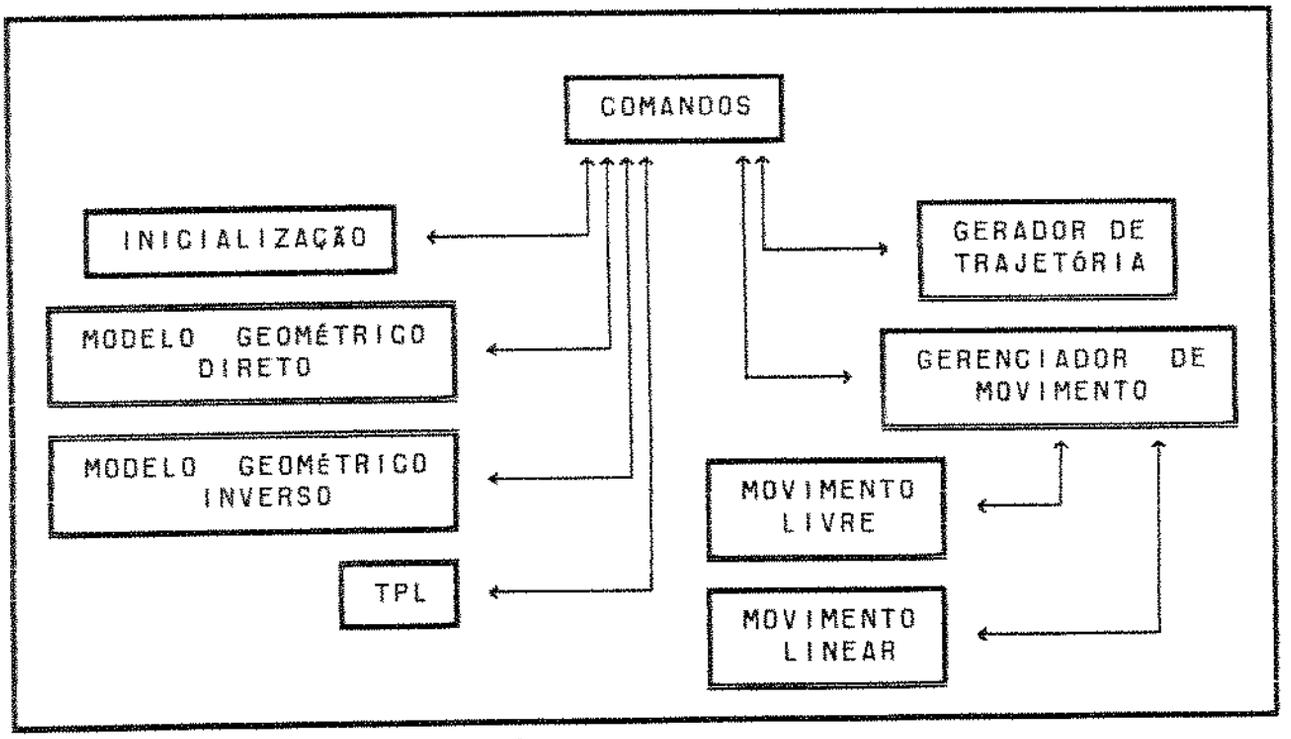

FIGURA F6 - ESTRUTURA DO NÍVEL 1

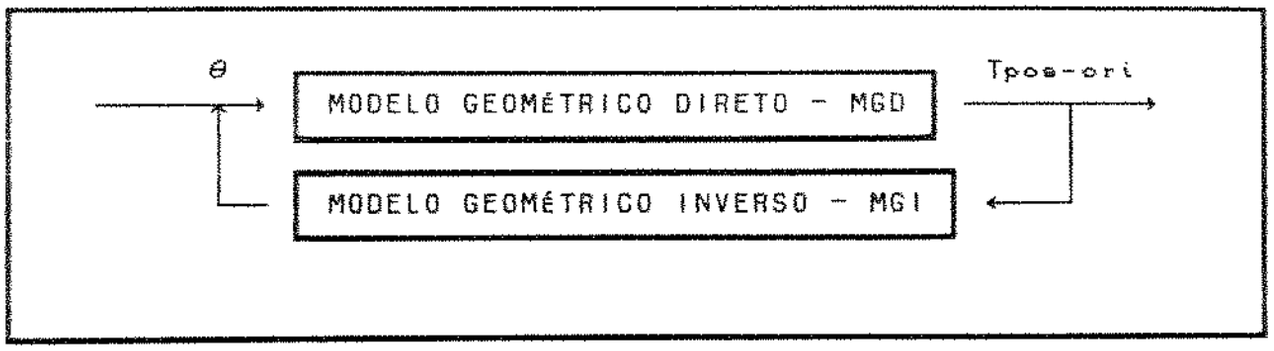

FIGURA F7 - FLUXO DE DADOS PARA MODELOS GEOMÉTRICOS

COMANDOS DA LINGUAGEM

Os comandos da linguagem atuam do sentido de auxiliar o usuário na implementação de uma tarefa, através da geração dos movimentos do robô. Adotou-se os seguintes critérios para a especificação dos comandos:

- permitir a geração e controle de movimentos no espaço das juntas e no espaço cartesiano,
- controlar a orientação e estado da garra,
- permitir o acesso a arquitetura interna da linguagem,
- controlar o tipo de trajetória gerada,
- comunicação com um teach-in,
- permitir a manipulação de objetos.

Assim sendo, selecionou-se um conjunto de comandos para a linguagem, que estão reunidos nos grupos, como mostrado a seguir:

***** MOVIMENTO: responsável pelos movimentos básicos do robô

- MOVER(): gerar um movimento
- MOVERANG(): mover uma junta
- POSICAO(): mover para uma posição no espaço cartesiano
- MOVIMENTO(): setar um tipo de movimento
- SETPRECI(): setar a precisão máxima
- SETVELOMAX(): setar velocidade máxima
- SETNPT(): setar número máximo de pontos de discretização

***** ORIENTAÇÃO: controle de orientação da garra

- ORIFIX(): setar movimento com orientação fixa
- STORIENTA(): setar uma orientação
- MVORI(): setar modo de obtenção da orientação fixa
- RSTORIENTA(): resetar orientação fixa

***** MODELAMENTO: permite o acesso, pelo usuário, dos modelos geométricos utilizados internamente pela linguagem

- MDGEODI(): cálculo do modelo geométrico direto
- MDGEOIN(): cálculo do modelo geométrico inverso

***** TRAJETÓRIA: permite a seleção do perfil de velocidade e o acesso aos dados de discretização da trajetória

- TRACO(): gerar uma trajetória seguindo o perfil de velocidade setado
- VLPERFIL(): setar um perfil de velocidade
- VLSET(): retornar o perfil de velocidade setado

***** COMUNICAÇÃO COM O TEACH-IN: comunicação com o módulo de teach-in do sistema de CAD-Simulação

- LOADD(): carregar dados para teach-in
- SAVED(): salvar em disco dados para teach-in
- STCONTR(): setar valor para parâmetro de controle

***** MENSAGEM: permite o controle e definição de mensagens durante a execução de um programa e o controle do modo de operação em condição de erro de programação

- STERRO(): setar modo de operação em condição de erro
- STMSG(): setar emissão de mensagens
- DFMSG(): definir mensagens
- RSTMSG(): resetar emissão de mensagens

***** INICIALIZAÇÃO: conduz o robô a configurações pré-determinadas, inicializando a tabela de parâmetros da linguagem com valores default

- ZEROMAQ(): conduzir o robô para posição de zero-máquina
- CASA(): conduzir o robô para posição de homem-máquina
- COTOVELO(): setar tipo de configuração do cotovelo

***** GARRA: permite o controle sobre a garra

- ABRIR(): abrir garra
- FECHAR(): fechar garra
- VLGARRA(): retornar o valor corrente da garra
- CODGARRA(): retornar o código corrente da garra
- TROCAGARRA(): trocar garra
- PTGARRA(): setar ponto de troca da garra

***** OBJETO: permite a definição e controle de objetos

- DFOBJ(): definir objeto
- POPXPG(): retornar ponto de pega do objeto
- STOBJETO(): setar objeto
- ATIVOBJ(): retornar código do objeto ativo
- RMOBJ(): zerar dados de um objeto definido
- NOBJDEF(): retornar número de objetos definidos
- OBJCORRENT(): retornar dados do objeto ativo
- RDFOBJ(): redefinir dados dos objetos

***** USO GERAL

- POCORRENT(): retornar a posição cartesiana corrente
- CVGRRD(): converter graus em radianos
- CVRDGR(): converter radianos em graus
- FIXANG(): fixar (travar) uma junta
- LVRANG(): destravar uma junta
- CARGA(): setar um nível de carga
- NPONTOS(): retornar o número de pontos interpolados

SINTAXE DOS COMANDOS

Para utilização dos comandos é necessário conhecer sua sintaxe, ou seja, os tipos de dados (definidos na base de dados) de entrada e saída. A seguir tem-se a listagem dos comandos e sua sintaxe:

```
***** MOVIMENTO
unsigned long int mover(struct t6)
unsigned long int moverang(unsigned long int,double)
unsigned long int posicao(struct t6)
unsigned long int movimento(unsigned long int)
unsigned long int setpreci(unsigned long int,double)
unsigned long int setvelomax(unsigned long int,double)
unsigned long int setnpt(unsigned long int, unsigned long int)


***** ORIENTAÇÃO
struct t6 orifix(void)
unsigned long int storienta(struct t6)
unsigned long int mvori(unsigned long int)
unsigned long int rstorienta(void)


***** MODELAMENTO
struct t6 mdgeodi(struct t6)
struct t6 mdgeoin(struct t6)
```

```
***** TRAJETÓRIA
unsigned long int traco(double,double,double,double,double,
     double*,double*,unsigned long int*,unsigned long int)
unsigned long int viperfil(unsigned long int)
unsigned long int viset(void)


***** COMUNICAÇÃO COM O TEACH-IN
unsigned long int loadd(char*)
unsigned long int saved(char*)
unsigned long int stcontr(unsigned long int,unsigned long int)


***** MENSAGEM
unsigned long int stmsg(unsigned long int)
unsigned long int sterro(unsigned long int)
unsigned long int dfmsg(unsigned long int, char*)
unsigned long int rstmsg(unsigned long int )


***** INICIALIZAÇÃO
unsigned long int casa(void)
unsigned long int zeromaq(void)
unsigned long int cotovelo(unsigned long int)


***** GARRA
unsigned long int abrir(double)
unsigned long int fechar(double)
double vlgarra(void)
unsigned long int cdgarra(void)
unsigned long int trocagarra(unsigned long int )
unsigned long int ptgarra(struct t6)


***** OBJETO
struct objeto dfobj(struct objeto , unsigned long int)
unsigned long int popxpg(unsigned long int,struct t6*)
unsigned long int stobjeto( unsigned long int)
unsigned long int ativobj(void)
unsigned long int rmobj(unsigned long int)
unsigned long int nobjdef(void)
struct objeto objcorrent(void)
unsigned long int rdfobj(struct objeto,unsigned long int,
                                                 unsigned long int)


***** USO GERAL
struct t6 pocorrent(void)
double cvgrrd( double )
double cvrdgr( double )
unsigned long int fixang(unsigned long int)
unsigned long int ivrang(unsigned long int)
unsigned long int carga(unsigned long int)
unsigned long int npontos(unsigned long int)
```

SINTAXE DOS COMANDOS DE USO INTERNO

Internamente, a LP é implementada com o auxílio de comandos especiais que serão referenciados no decorrer do texto. O conhecimento desses comandos permite ao usuário programador interferir (modificar) na execução dos comandos da LP.

```
** NÍVEL 2
   struct t6 transfer(char*,unsigned long int,unsigned long int,
                                                        struct t6)

** NÍVEL 3
   struct t6 trfang(struct t6,unsigned long int)
   struct t6 resang(unsigned long int)
   struct objeto trfobj( struct objeto, unsigned long int)

** NÍVEL 4
   unsigned long int msg(unsigned long int, unsigned long int)
   unsigned long int erro(unsigned long int, unsigned long int)

** NÍVEL 5
   struct objeto trfobj(struct objeto,unsigned long int)
   unsigned long int mcp(double*,double*,double*,double,
                                                 double,double)
   struct objeto rpobjeto(void)
```

O usuário programador que pretender realizar alterações nos comandos da LP ou criar outros, deve conhecer sua implementação. O apêndice 1A contém a descrição dos comandos já implementados na LP.

NÍVEL 2 - INTERFACE EXTERNA

O nível 2 - interface externa consiste de um conjunto de rotinas de comunicação entre a linguagem e o sistema de CAD-Simulação. A figura F8 mostra a estrutura deste nível.

A nível interno da linguagem, tem-se uma rotina, chamada TRANSFER(), que atualiza o arquivo de comunicação com o sistema de CAD-Simulação. Os comandos de movimento, garra, inicialização, orientação e objetos geram dados que implicam na mudança de estado do manipulador ou do sistema, por exemplo, mudança de posição-orientação, valor e código da garra, setar um tipo de cotovelo e ativar-desativar objeto. Estes dados são gerados sequencialmente a medida que os comandos são executados, ou então, através da chamada da rotina de transferência (TRANSFER()) que tem dois modos de operação, acionados pelos códigos CD1 e CD2. O código CD1 é utilizado quando uma sequência de valores de referência são gerados utilizando a estrutura de dados interna da linguagem. O código CD2 é utilizado para a transferência de dados isoladamente. Por exemplo, o comando MOVER() gera uma sequência de pontos utilizando a estrutura de dados interna, acionando a rotina TRANSFER() com o código CD1, fazendo com que os dados sejam transferidos para o arquivo de comunicação. A rotina TRANSFER() tem como parâmetro de entrada uma estrutura do tipo posição-orientação, que com o código CD2, possibilita a transferência dos dados desta estrutura para o arquivo de comunicação. Este procedimento pode ser utilizado pelo usuário na implementação de novos comandos.

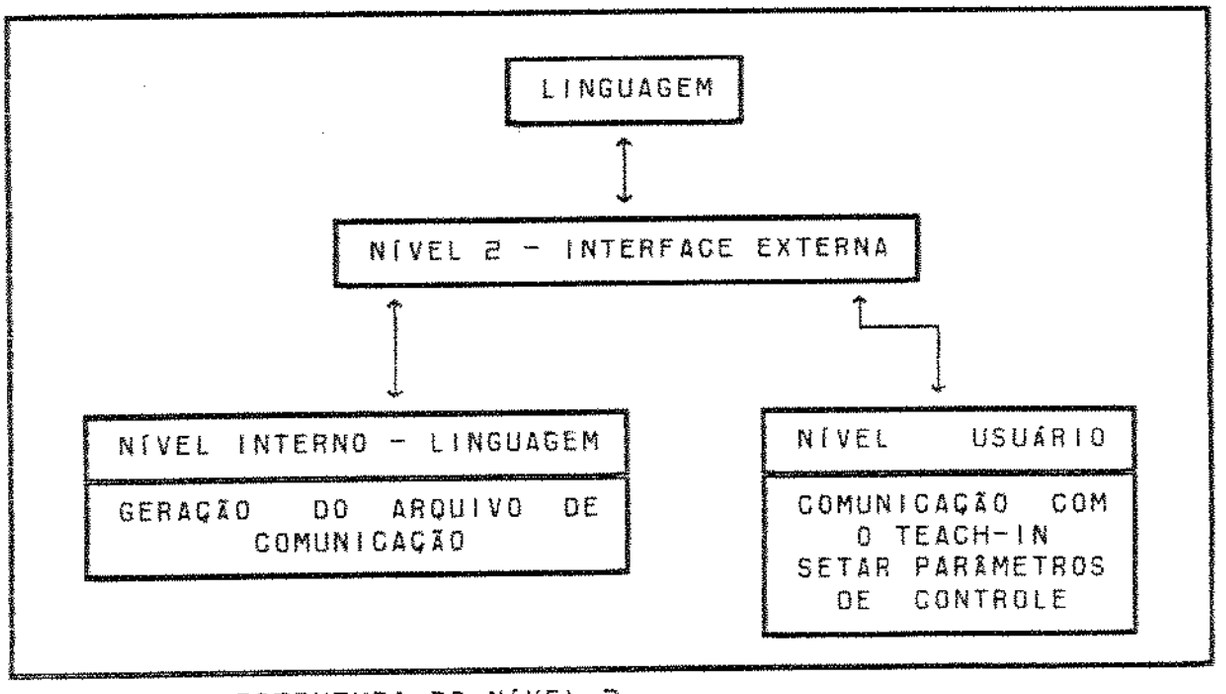

FIGURA F8 - ESTRUTURA DO NÍVEL 2

A rotina TRANSFER() atua no sentido de produzir um arquivo de dados como um protocolo de comunicação. Os dados são organizados em blocos, correspondendo a um comando executado ou uma chamada da rotina. Um bloco contém as seguintes informações:

- nome do comando executado,
- número de pontos de discretização: NP,
- incremento de tempo,
- tempo total de movimento,
- número de objetos,
- para cada objeto tranferir
  - largura, comprimento e altura,
  - coordenada XYZ dos vértices. Se existir um objeto resetado ou não definido transferir o valor ZERO,
- para os pontos discretizados: I=0 até NP transferir:
  - valor angular das juntas,
  - código da garra ativa,
  - código do objeto ativo,
  - parâmetros de controle.

O último bloco executado contém a mensagem: "FIM" indicando fim de tarefa e de arquivo.

A sintaxe e implementação da rotina TRANSFER() é a seguinte:

TRANSFER(n,cd,np,tt)

ENTRADA: n: nome do comando
cd: código do modo de operação
CD1: uso interno
CD2: usuário
np: número de pontos a ser transferido
tt: estrutura TB a ser transferida

SAÍDA: operação correta (VERDADE) ou falsa (FALSO)

DESCRIÇÃO: Efetuar a transferência de dados para o arquivo de comunicação com o sistema de CAD-Simulação. MODO CD1 transfere dados da estrutura interna (Base de Dados). MODO CD2 transfere os dados em tt.

ALGORITMO
- transferir código de operação
- se código CD1 transferir:
  - nome do comando executado
  - número de pontos interpolados: NP
  - incremento de tempo
  - tempo total de movimento
  - número de objetos

- para cada objeto tranferir
  - largura, comprimento e altura
  - coordenada XYZ dos vértices. Se existir um objeto resetado ou não definido transferir o valor ZERO
- para os pontos discretizados: I=0 até NP
  transferir:
  - valor angular das juntas
  - código da garra ativa
  - código do objeto ativo
  - parâmetros de controle

- se código CD2 transferir
  - nome do comando executado
  - número de objetos
  - para cada objeto tranferir
    - largura, comprimento e altura
    - coordenada XYZ dos vértices. Se existir um objeto resetado ou não definido transferir o valor ZERO
  - transferir o valor angular das juntas especificados em tt
  - valor angular das juntas
  - código da garra ativa
  - código do objeto ativo
  - parâmetros de controle

A nível de usuário, tem-se dois comandos de comunicação com o módulo TEACH-IN: LOADD() e SAVED() e um comando de uso geral: STCONTR().

O comando STCONTR() permite ao usuário acessar os parâmetros de controle, que são slots de transferência de dados para o sistema de CAD-Simulação, não tendo função definida na linguagem. O usuário pode utilizar estes slots para implementação de novos comandos, por exemplo, rotina de espera, controle de periféricos e pallets. São 10 os parâmetros de controle, numerados de 0 a 9, incorporando um valor inteiro positivo.

O usuário pode trocar informações com o modo TEACH-IN através dos comandos:

- LOADD() para leitura de dados,

- SAVED() para salvar dados.

A transferência de dados é efetuada através da área de memória destinada aos pontos setados pelo módulo TEACH-IN. O acesso é realizado através do ponteiro estrutura tipo T6: ptt() e da variável VNP que contém o número de pontos setados.

NÍVEL 3 - CONTROLADOR INTERNO

O nível 3 - controlador interno é constituído de um conjunto de subrotinas para o gerenciamento do fluxo de dados entre os comandos, a TPL, a base de dados e o usuário. A figura F9 mostra a estrutura de dados. As subrotinas que compõe este nível são:

- TRFANG(): gerenciar a transferência de dados do tipo posição-orientação,
- RESANG(): gerenciar a montagem de estrutura de dados do tipo posição-orientação a partir de dados dos pontos discretizados,
- TRFOBJ(): gerenciar a transferência de dados do tipo objeto.

Todas 3 são de uso interno do sistema. As funções implementadas são:

SUBROTINA TRFANG():
- atualizar posição-orientação corrente e desejada na TPL
- atualizar orientação fixa na TPL
- fornecer posição-orientação corrente e desejada
- fornecer orientação fixa ativa

SUBROTINA RESANG():
- consultar a base de dados do sistema e fornecer a estrutura de posição-orientação de um ponto discretizado na trajetória

SUBROTINA TRFOBJ():
- atualizar dados dos objetos na base de dados
- fornecer dados de um objeto

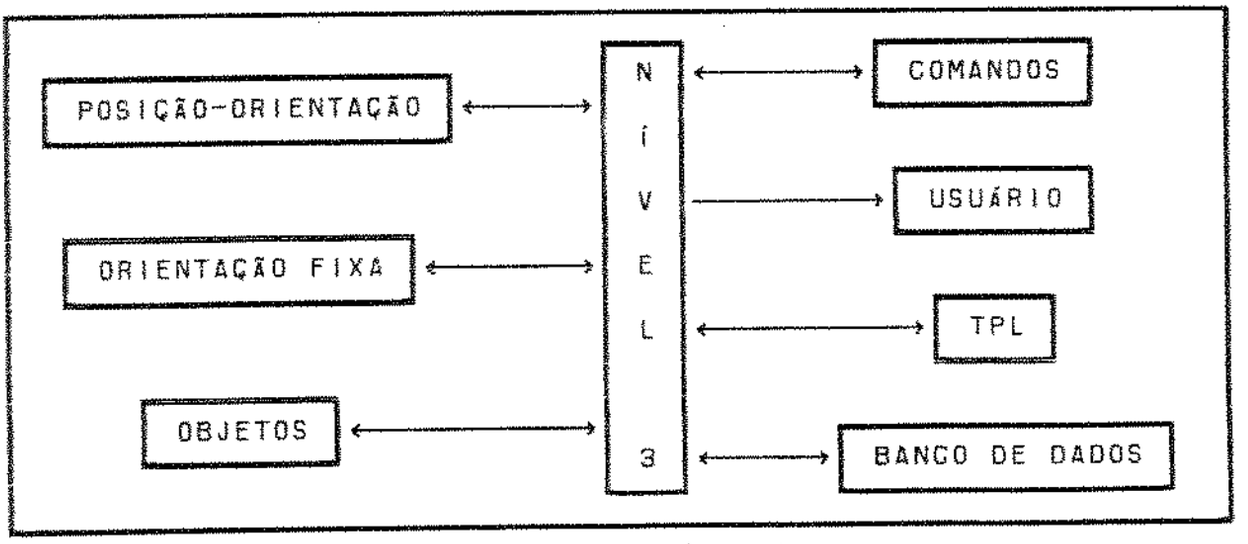

FIGURA F9 - ESTRUTURA DE DADOS DO NÍVEL 3

NÍVEL 4 - MENSAGEM

O nível 4 - mensagem constitui-se de um conjunto de rotinas para o gerenciamento do fluxo de mensagens para o usuário e execução dos comandos pelo sistema. As mensagens são de dois tipos:

- comando

- erro

As mensagens de comando são enviadas antes da execução de um comando indicando seu estado ativo. As mensagens de erro são enviadas sempre que uma condição de erro é detectada pelo sistema, podendo manifestar-se de dois modos:

- uma condição de erro é detectada, a mensagem de erro é enviada ao usuário pelo vídeo, a execução do comando é interrompida e a sequência do programa não é alterada,

- uma condição de erro é detectada, a mensagem de erro é enviada ao usuário pelo vídeo, a execução do comando e do programa são interrompidas.

Estes modos são ativados pelo usuário através do comando: STERRO(DESATIVAR-para setar o primeiro modo; ABORTAR - para setar o segundo modo) ; o primeiro modo é default.
O sistema utiliza dois comandos implementadas neste nível:

- msg() para enviar uma mensagem de comando,

- erro() para enviar uma mensagem de erro.

Estes comandos são de uso interno do sistema. O usuário pode interferir na execução deste nível através dos COMANDOS DE MENSAGEM definidos no nível 1-comandos, de tal forma que possa definir (DFMSG), setar (STMSG) ou resetar (RSTMSG) uma mensagem de comando ou alterando o modo de operação em condição de erro (STERRO).

Implementação dos comandos:

MSG(m1,m2)

ENTRADA: m1: código da primeira parte da mensagem
m2: código da segunda parte da mensagem
SAÍDA: retornar o valor VERDADE se mensagem estiver ativada, FALSO se mensagem estiver desativada
DESCRIÇÃO: enviar ao usuário uma mensagem indicando o comando executado
ALGORITMO
- consultar a TPL verificando se a flag de mensagem está setada, se NÃO => retornar FALSO
- incrementar parâmetro da TPL indicando o número de comandos executados
- consultar a base de dados e obter as mensagens de código: m1 e m2
- enviar as mensagens pelo vídeo

ERRO(m1,m2)

ENTRADA: m1: código da primeira parte da mensagem
m2: código da segunda parte da mensagem
SAÍDA: retorna o valor FALSO
DESCRIÇÃO: enviar ao usuário uma mensagem de erro, interromper a execução do comando retornando o valor FALSO e interromper a execução do programa se flag de condição de erro estiver setada
ALGORITMO
- consultar a TPL verificando se a flag de mensagem está setada, se SIM
  - consultar a base de dados e obter as mensagens de código: m1 e m2
  - enviar as mensagens pelo vídeo
- consultar a TPL verificando se a flag de erro está setada, se SIM abortar a execução do programa
- retornar o valor FALSO

NÍVEL 5 - OBJETO

A garra tem a função de atuar como interface entre o robô e os objetos manipulados, alterando sua posição-orientação (movimento), e ainda, permitir o interfaceamento entre objetos. No nível 5 é o responsável pelo gerenciamento da execução dos procedimentos para manipulação de objetos, sendo que a linguagem incorpora os seguintes comandos:

- dfobj()
- objcorrent()
- popxpg()
- stobjeto()
- ativobj()
- nobjdef()
- rdfobj()
- rmobj()

A função e implementação destes comandos está descrita no nível 1 - Comandos.

Um objeto consiste de um paralelepípedo com dimensões e posição-orientação no espaço cartesiano conhecidas, sendo representado como um "esqueleto", ou seja, o objeto não é um "sólido" com massa, textura, inércia, rugosidade e outras características físicas, e sim, um "ponto" com "área de influência" representada por um paralelepípedo com arestas definidas por vértices conhecidos. Não se utilizou a representação por sólidos, pois, implicaria na implementação de modelos matemáticos elaborados exigindo computadores com maior capacidade de processamento e memória [GROSSMA-78] [REQUICH-80] [SANDOR-85]. O sistema reconhece até 10 objetos, sendo definidos e localizados com os seguintes parâmetros:

- L : largura,
- C : comprimento,
- H : altura,
- matriz de orientação,
- vetor posição e
- nível de carga que o sistema deve assumir durante um movimento quando o objeto está ativo.

A matriz de orientação e o vetor posição compoem uma estrutura do tipo T6 (consultar base de dados) definidos a partir do referencial absoluto, sendo as dimensões especificadas seguindo a figura F10, onde a altura (H) é medida ao longo do eixo Z, a largura (L) ao longo do eixo X e o comprimento (C) ao longo do

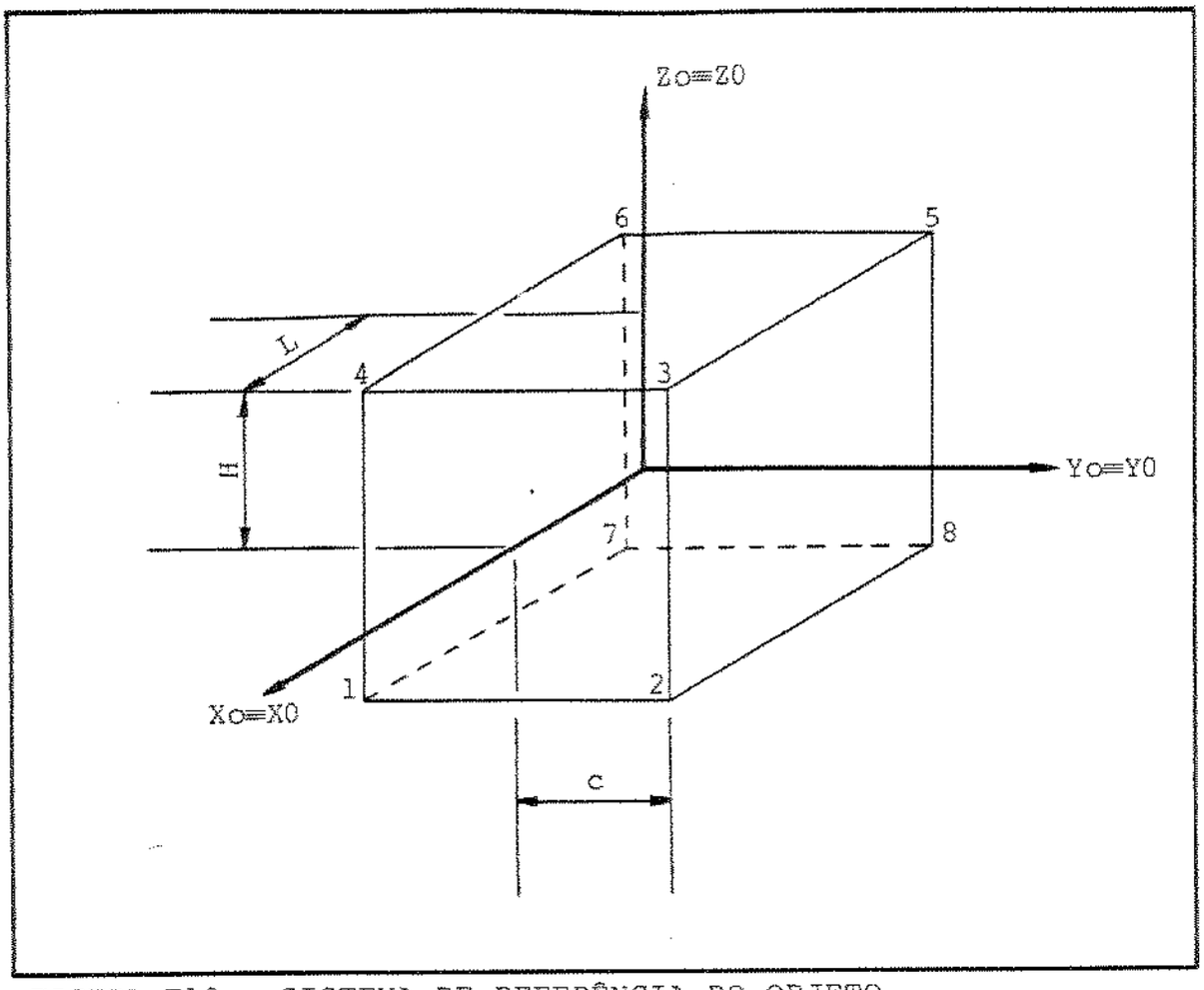

FIGURA F10 - SISTEMA DE REFERÊNCIA DO OBJETO

eixo Y. Um referencial local (Oo) é definido no centro geométrico do paralelepípedo, necessário na atualização da posição-orientação durante um deslocamento. Para a pega de um objeto, a matriz orientação da garra deve estar alinhada com a do objeto, ou seja, as matrizes orientação são idênticas. O objeto é pego pela sua base, medida no plano XY e a uma distância -H no sistema de referência local. Assim, a posição (absoluta) de pega precisa ser corrigida e é dada por:

$$Pxg = Pxo - Kx * H$$
$$Pyg = Pyo - Ky * H$$
$$Pzg = Pzo - Kz * H$$

onde H : altura do objeto,
Pxo, Pyo, Pzo: posição absoluta do centro geométrico do objeto,
Pxg, Pyg, Pzg: posição absoluta do centro geométrico da base do objeto, ou seja, o ponto de pega,
Kx, Ky e Kz: terceira coluna da matriz orientação.

A matriz orientação e vetor posição são definidos a partir do modelo geométrico direto do robô, onde, as juntas são consideradas como "pseudo-juntas". Especificando-se o valor angular das "pseudo-juntas" e calculando o modelo geométrico direto, determina-se a posição-orientação do objeto. Note que o comando MDGEODI() fornece diretamente estes cálculos. Definida a orientação, a posição pode ser alterada, transladando o objeto para outra posição e, vice-versa, definindo a posição, pode-se recalcular a orientação do objeto alterando o valor angular das "pseudo-juntas" e calculando o MGD. Adotou-se este tipo de representação, objetivando a unificação do modelamento geométrico e simplificando a estrutura interna da linguagem, contudo, poder-se-ia utilizar outros métodos, como os ângulos de Euler [PAUL-83]. O nível de carga (NC) especifica a percentagem da velocidade máxima assumida pelo sistema que será aplicada durante sua movimentação se ativo: $1 \leq NC \leq 10$, onde NC = 10 é a máxima velocidade.

O comando DFOBJ() permite ao usuário definir um objeto para o sistema, utilizando um dos três procedimentos, cujos parâmetros de entrada são:

OP1 - matriz orientação, vetor posição, parâmetros (L , H , C) e nível de carga conhecidos,
OP2 - ângulos para orientação, vetor posição, parâmetros (L , H , C) e nível de carga conhecidos,
OP3 - assumir valores default.

Na primeira opção - OP1 o usuário deve definir a matriz orientação e o vetor posição, os parâmetros geométricos: H, L e C e o nível de carga. Na segunda opção - OP2 o usuário define o vetor posição, os parâmetros geométricos, o nível de carga e os ângulos de orientação que internamente permitem o cálculo da matriz orientação. Na terceira opção - OP3, o sistema define o objeto segundo valores default:

- L = H = C = 0.01 unidades de medida de comprimento
- orientação: T1'= T4'= T5'=T6'=0, T2'= PI/4. e T3'= -PI/4 (valor das "pseudo-juntas")
- nível de carga = 5 (50%)

O usuário pode acessar a estrutura de dados dos objetos através do ponteiro estrutura OBJETO: OBJ[número do objeto] (consultar a base de dados).

Uma vez definido, um objeto não pode ser removido da base de dados, no entanto, seus dados podem ser redefinidos através do comando RDFOBJ(), seguindo procedimento análogo ao utilizado no comando DFOBJ() e com as mesmas opções. Para um objeto ser ignorado, deve-se usar o comando RMOBJ() que zera todos os seus parâmetros, atribuindo dimensão, orientação e posição nulas e nível de carga 1 .

O comando STOBJETO() ativa um objeto, ou seja, independentemente da sua posição-orientação o objeto aparecerá, no simulador CAD e na garra do robô. Assim, para que a animação seja "realista", o usuário deve executar o comando POPXPG() movendo a garra para o ponto de pega e então ativar o objeto. Se o objeto de número 0 for ativado, significa que não existe objeto ativo e os dados de posição-orientação do objeto ativo são atualizados na base de dados para a última configuração atingida, e então, objeto setado (ativo) é resetado (desativado).

O sistema reconhece um objeto como um apêndice da garra, ou seja, enquanto um objeto estiver ativo, o modelo geométrico direto fornece a posição-orientação do centro geométrico deste e não da garra. Este tipo de abordagem permite ao usuário definir seu próprio conceito de objeto e sua interface com o manipulador. Craig [CRAIG-86] mostra como desenvolver esta interface aplicando matriz de transformação homogêneas.

Internamente, os comandos de objetos definidos no nível 1 são implementados com o auxílio das seguintes rotinas, de uso exclusivo do nível 5:

- TRFOBJ()
- MCP()
- RPOBJETO()

como mostrado na figura F11.

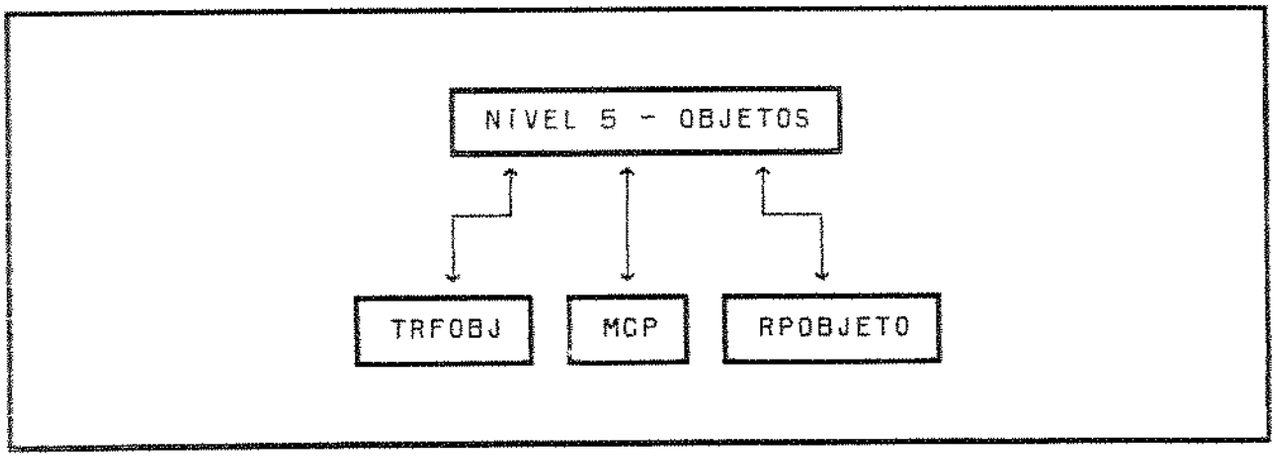

FIGURA F11 - ESTRUTURA DO NÍVEL 5

A rotina TRFOBJ() gerencia o fluxo de dados entre o sistema e a base de dados dos objetos, operando em quatro opções:

OP1 - transferir dados de uma estrutura tipo objeto para a base de dados, indexado pelo ponteiro FC[],
OP2 - transferir dados da base de dados para uma estrutura tipo objeto, indexado pelo ponteiro FC[]
OP3 - transferir dados do objeto ativo para uma estrutura tipo objeto,
OP4 - transferir dados de uma estrutura tipo objeto para o objeto ativo.

A implementação da rotina é a seguinte:

TRFOBJ(tt,cd)

ENTRADA: tt : estrutura objeto
cd : opção
SAÍDA: estrutura objeto
DESCRIÇÃO: gerenciar o fluxo de dados entre o sistema e a base de dados dos objetos
ALGORITMO
- se opção 1, transferir tt para a base de dados do objeto indexado por FC[0]
- se opção 2, consultar a base de dados e transferir os dados do objeto indexado por FC[0]
- se opção 3, consultar a base de dados e transferir os dados do objeto ativo
- se opção 4, transferir tt para a posição do objeto ativo na base de dados

A rotina MCP() prepara a localização dos vértices dos objetos a partir das dimensões: L, H e C seguindo a ordem indicada no quadro abaixo (ver figura F10):

VÉRTICE	X	Y	Z
1	L	-C	-H
2	L	C	-H
3	L	C	H
4	L	-C	H
5	-L	C	H
6	-L	-C	H
7	-L	-C	-H
8	-L	C	-H

Os dados de saída desta rotina são utilizados pela rotina TRANSFER() do nível 2 para transferência da correta localização do objeto no referencial absoluto:

$$[P_v] = [R] * [P_r] + [P_o]$$

onde
- P_v : vetor (x,y,z) da posição absoluta dos vértices do objeto
- R : matriz orientação
- P_r : vetor (x,y,z) da posição relativa dos vértices do objeto no seu referencial local
- P_o : vetor posição absoluta do centro geométrico do objeto

A rotina RPOBJETO() atualiza a posição-orientação do objeto ativo após ser resetado, pois, durante os movimentos estes valores são alterados. A orientação é a mesma da garra, mas o vetor posição absoluta do centro geométrico do objeto deve ser corrigido:

$$[P_o] = [P_{ga}] + H * [k]$$

onde
- P_{ga} : vetor (x,y,z) da posição absoluta de pega do objeto
- H : altura do objeto
- P_o : vetor (x,y,z) da posição absoluta do centro geométrico do objeto
- k : terceira coluna da matriz orientação do objeto

Note que o ponto de pega (e de movimentação na garra) é diferente do centro geométrico, estando deslocado de $-H$ ao longo do eixo Z no referencial local (no objeto). A atualização dos dados é automática após a execução de um comando de movimento ou orientação.

TABELA DE PARÂMETROS DA LINGUAGEM (TPL)

A tabela de parâmetros da linguagem (TPL) é constituída de uma base de dados global, ou seja, o acesso é permitido as rotinas e comandos da linguagem. Sua função é registrar o estado das flags e parâmetros da linguagem permitindo a troca de informações sem a necessidade de chamada de subrotinas.

A TPL está implementada através da alocação de memória do tipo static [KELLY-84], ou seja, durante a execução do programa sua localização (na memória) permanece constante e está dividida em duas áreas:

- tabela de flags
- tabela de parâmetros

A tabela de flags registra o valor das flags da linguagem, sendo acessada através do ponteiro tipo unsigned long int TSI [KELLY-84]:

[0] : uso geral
[1] : indica se uma junta 1 está travada (=1) ou não (=0)
[2] : indica se uma junta 2 está travada (=1) ou não (=0)
[3] : indica se uma junta 3 está travada (=1) ou não (=0)
[4] : indica se uma junta 4 está travada (=1) ou não (=0)
[5] : indica se uma junta 5 está travada (=1) ou não (=0)
[6] : indica se uma junta 6 está travada (=1) ou não (=0)
[7] : flag de orientação: = 1 orientação setada
= 0 orientação resetada
[8] : flag de movimento: = 1 linear
= 0 livre
[9] : flag de velocidade: = 1 perfil trapezoidal
= 0 perfil senoidal
[10] : nível de carga
[11] : uso geral
[12] : flag de tipo de cotovelo: = 0 LIVRE
= 1 UP
= 2 DOWN
[13] : flag de orientação 2: = 1 setar uma orientação preservando a posição original
= 2 setar uma orientação não preservando a posição original
[14] : código da garra

[15] a [34] : uso geral - reservado para implementação de pallets
[35] : código do objeto ativo
[36] : número de pontos interpolados no antepenúltimo movimento
[37] : número de pontos interpolados no penúltimo movimento
[38] : número de pontos interpolados no último movimento
[39] : número total de objetos definidos
[40] a [49] : parâmetros de controle
[50] : flag de mensagem: = 0 mensagem desativada
= 1 mensagem ativada
[51] : flag de erro: = 0 mensagem desativada
= 1 mensagem ativada
[52] : número de comandos processados
[53] : flag de erro: = 0 se erro for detectado imprimir mensagem
= 1 se erro for detectado imprimir mensagem e abortar execução

A tabela de parâmetros registra o valor de parâmetros de uso global, sendo acessada através do ponteiro tipo double tvs[] (KELLY-84):

[0] : uso geral
[1] a [6] : posição angular corrente das juntas
[7] a [9] : posição cartesiana corrente: Px, Py e Pz respectivamente
[10] : tempo total de trajetória
[11] a [16] : posição angular desejada
[17] a [19] : posição cartesiana desejada: Px, Py e Pz respectivamente
[20] : incremento de tempo
[21] a [26] : máxima velocidade angular
[27] a [29] : máxima velocidade cartesiana: Px, Py e Pz respectivamente
[30] : parâmetro de uso geral no perfil de trajetória
[31] a [40] : uso geral
[41] a [46] : precisão para movimento angular das juntas
[47] a [49] : precisão para movimento cartesiano
[50] : precisão para movimento da garra
[51] a [70] : uso geral
[71] a [76] : mínimo intervalo angular das juntas
[77] a [79] : mínimo intervalo cartesiano: Px, Py e Pz
[80] : mínimo intervalo da garra
[81] a [86] : máximo intervalo angular das juntas
[87] a [89] : máximo intervalo cartesiano: Px, Py e Pz
[90] : máximo intervalo da garra
[91] a [99] : matriz orientação corrente

```
        [100] : valor corrente da garra
[101] a [109] : matriz orientação desejada
        [110] : valor desejado da garra
[111] a [119] : matriz orientação fixa
[120] a [132] : uso geral
[133] a [141] : matriz orientação para troca da garra
[142] a [144] : posição cartesiana para troca da garra
        [145] : máxima velocidade da garra
```

A TPL é de uso interno da linguagem. A execução de uma rotina ou comando implica necessariamente na sua consulta e/ou atualização. O usuário pode acessar a TPL indiretamente através dos comandos da linguagem, por exemplo, a posição TS[8] pode ser acessada pelo usuário através dos comandos VLPERFIL() e VLSET(). O acesso também pode ser realizado através dos ponteiros TS[] e TVS[].

Com relação às unidades de medida, adotou-se o seguinte padrão:

```
comprimento-deslocamento: .. metro (m)
angular: ..................... radianos (rad)
tempo: ....................... segundos (s)
massa: ....................... kilograma (Kg)
força: ....................... Newton (N)
```

BASE DE DADOS

Este item mostra como os dados estão estruturados e podem ser acessados internamente no sistema e a nível de usuário, de tal forma que a base de dados [KEMPER-87] [CONNELL-89] possa ser utilizada com maior eficiência. Estas informações são importantes ao projeto, pois, permitem o desenvolvimento e integração, ao sistema, de novos comandos e procedimentos. Contudo, para o usuário programador é irrelevante esse conhecimento, bastando conhecer a estrutura de dados da linguagem.

São quatro os tipos de dados definidos no sistema:

- constantes,
- variável simples (unsigned long int, double, ...),
- ponteiros,
- estruturas (T6 e OBJETO).

No entanto, a linguagem C reconhece outros tipos que o usuário pode utilizar sem prejuízo do desempenho do sistema.

Quanto as estruturas, o sistema reconhece dois tipos:

- struct T6
- struct OBJETO

O tipo struct T6 incorpora informações de valor angular das juntas, posição-orientação, valor da garra e condição de erro do manipulador e área de trabalho, definidos da seguinte forma:

```
struct T6
    {
      double t1,t2,t3,t4,t5,t6 : valor angular das juntas
      double px,py,pz : posição cartesiana (quarta coluna da
                        matriz transfornada homogênea)
      double nx,ny,nz,
             ox,oy,oz,
             kx,ky,kz  :  matriz orientação (três primeiras
                          colunas da matriz transformada homogênea)
      unsigned long int erro: código de condição de erro
                              VERDADEIRO = 1
                              FALSO = 0
     }
```

O tipo struct objeto incorpora informações de dimensão, matriz orientação, vetor posição, ângulos de orientação, nível de carga e condição de erro dos objetos, definidos da seguinte forma:

```
struct objeto
    {
      double ax,ay,az : ângulo de orientação do objeto, indicando
                        rotação sobre os eixos X, Y e Z
                        respectivamente
      struct tB op : estrutura do tipo TB definindo a matriz
                     orientação, posição do centro geométrico e
                     condição de erro do objeto
      double c,h,l : dimensão: comprimento, altura e largura
      unsigned long int fc[0] : ponteiro indexador (uso interno
                                do sistema)
      unsigned long int ncv : nível de carga
    }
```

A figura F12 mostra a estrutura da base de dados da linguagem.

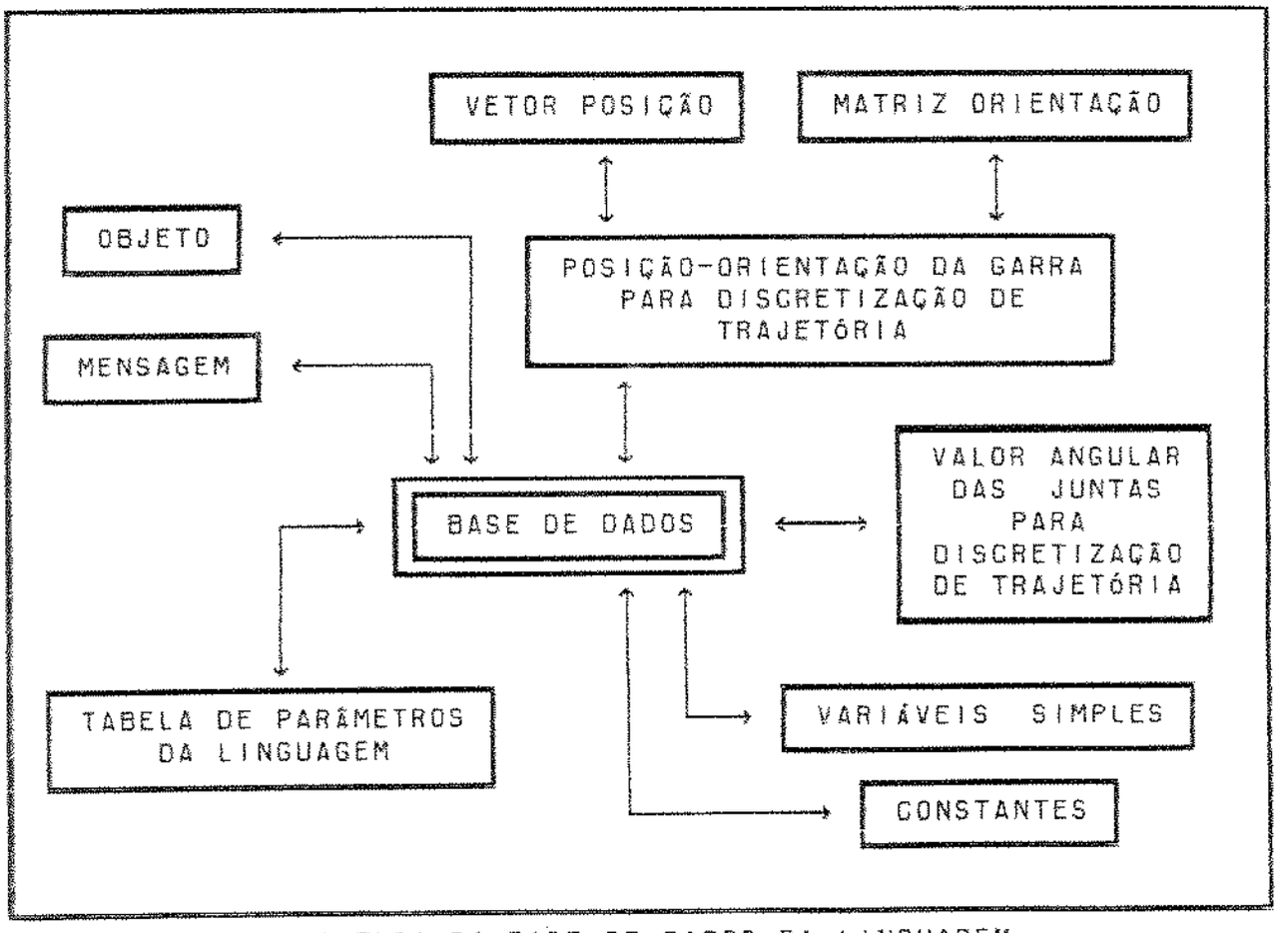

FIGURA F12 - ESTRUTURA DA BASE DE DADOS DA LINGUAGEM

O tipo constante define as variáveis cujo valor permanece inalterado durante a execução do programa, identificando valores pré-definidos. A linguagem reconhece as seguintes constantes:

CONSTANTE	VALOR	
VERDADE	1	
FALSO	0	
OP1	1	indica opção número 1
OP2	2	indica opção número 2
OP3	3	indica opção número 3
MAXOBJ	10	número máximo de objetos
MAXPT	512	número máximo de pontos de discretização
ATIVAR	1	ativar uma opção
DESATIVAR	0	desativar uma opção
UP	1	setar cotovelo UP
DOWN	2	setar cotovelo DOWN
LIVRE	3	setar cotovelo livre
TRAPEZOIDAL	1	setar perfil de velocidade
SENOIDAL	2	setar perfil de velocidade
MTPOS	1	setar orientação fixa mantendo a posição corrente
SMTPOS	2	setar orientação fixa sem manter a posição corrente
PI	3.1415927	constante matemática
ERROMIN	0.0001	valor mínimo para truncamento numérico
DX , DY e DZ	7, 8 e 9	indicam a direção cartesiana
GARRA	10	indica garra
J1, J2, J3, J4, J5 e J6	1, 2, 3, 4, 5, 6	indicam o número da junta

O tipo variável simples é de uso interno, utilizado na implementação das subrotinas e comandos.

O tipo ponteiro contém o endereço de uma outra variável, ou seja, de uma área de memória. Este tipo de variável é utilizada na alocação de memória fora da segmentação padrão da linguagem C. As variáveis referenciadas por ponteiros definem:

USO INTERNO

- TPL : tabela de parâmetro da linguagem
- mensagens do sistema
- pontos discretizados de trajetória: matriz orientação, vetor posição, garra, ângulos das juntas e tempo
- objetos

USUÁRIO:

- posição-orientação dos pontos setados para o modo teach-in, acessado através do ponteiro: ptt[número do ponto]

Os ponteiros de uso interno podem ser acessados pelo usuário por:

- ponteiro char rr[número da mensagem] : mensagens do sistema definidas de 1 a 250
- ponteiro struct objeto obj[número do objeto] : dados dos objetos, obj[0] contém dados do objeto ativo
- ponteiro double vt1[], vt2[], vt3[], vt4[], vt5[], vt6[]: contém o valor angular das juntas durante o procedimento de discretização de trajetória
- ponteiro double vnx[], vny[], vnz[], vox[], voy[], voz[], vkx[], vky[], vkz[] : contém o valor da matriz orientação durante o procedimento de discretização de trajetória
- ponteiro double vpx[], vpy[], vpz[]: contém a posição cartesiana corrente durante o procedimento de discretização de trajetória
- ponteiro double vgr[] : contém o valor de abertura da garra durante o procedimento de discretização de trajetória
- ponteiro double vtp[] : contém o valor de tempo durante o procedimento de discretização de trajetória

APÊNDICE 1A

IMPLEMENTAÇÃO DOS COMANDOS DA LINGUAGEM DE PROGRAMAÇÃO

***** COMANDOS DE MOVIMENTO

MOVER(tt)

ENTRADA: tt: posição-orientação da garra e valor angular das juntas

SAÍDA: operação correta (VERDADE) ou falsa (FALSO)

DESCRIÇÃO: executar o movimento do manipulador e da garra, consultar o item GERENCIADOR DE MOVIMENTO. Gerar sinais de referência angular para as juntas e atualizar a TPL: posição-orientação corrente, valor da garra e nmero de pontos interpolados.

ALGORITMO

- se flag de movimento LIVRE setada e orientação RESETADA faça
  - verificar juntas TRAVADAS e corrigir o seu valor angular
  - calcular modelo geométrico direto e atualizar posição-orientação desejada
  - consultando a TPL, para todas as juntas e para o movimento da garra, calcular o tempo total (TF) de trajetória setando o maior valor
  - calcular o incremento de tempo (DT) considerando o valor de TF
  - calcular o número de pontos interpolados verificando se não ultrapassa o máximo valor permitido
  - chamar rotina para movimento livre
  - chamar rotina para atualizar arquivo geral com os sinais de referência
- se flag de movimento LINEAR setada faça
  - verificar se existe alguma junta TRAVADA, se SIM => ERRO
  - consultando a TPL verificar se flag de orientação esta setada, se SIM: consultar a TPL e obter a orientação constante, tomar a posição desejada (tt) e montar a estrutura com a posição-orientação final da garra
  - calcular modelo geométrico inverso e atualizar posição-orientação e valor angular desejado das juntas
  - consultando a TPL, para as direções cartesianas: PX, PY, PZ e para o movimento da garra, calcular o tempo total (TF) de trajetória setando o maior valor
  - calcular o incremento de tempo (DT) considerando o valor de TF

- calcular o número de pontos interpolados verificando se não ultrapassa o máximo valor permitido
- chamar rotina para movimento LINEAR (ver gerador de movimento)
- chamar rotina para atualizar arquivo geral com os sinais de referência

- se flag de movimento LINEAR resetada e flag de orientação setada => ERRO
- atualizar número de pontos interpolados na TPL

MOVERANG(cd,vl)

ENTRADA: cd: número da junta
vl: valor angular absoluto
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: executar o movimento de uma única junta do manipulador
ALGORITMO
- se flag de movimento LINEAR e orienteção fixa estiverem setada => ERRO
- consultar a TPL e obter o valor angular corrente das juntas
- atualizar o valor angular da junta cd com o valor vl
- chamar a rotina MOVER e executar movimento

POSIÇÃO(tt)

ENTRADA: tt: posição da garra
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: mover a garra para a posição tt mantendo a orientação corrente
ALGORITMO
- consultar a TPL e obter a orientação corrente
- montar uma estrutura com a orientação corrente e posição deseJada
- calcular modelo geométrico inverso e atualizar valor angular das juntas
- chamar a rotina MOVER e executar movimento

MOVIMENTO(op)

ENTRADA: op: opção de movimento
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: setar o tipo de movimento
op = 3 movimento LIVRE
op = 4 movimento LINEAR
ALGORITMO
- setar flag de tipo de movimento com a opção op na TPL

SETPRECI(op,vl)

ENTRADA: op: item setado
op= 1 a 6 número da junta
op= 7 a 9 direção cartesiana PX, PY e PZ
op= 10 garra
vl: valor setado
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: setar valor da precisão para cálculo de trajetória. Valor setado na TPL
ALGORITMO
- verificar se 1 <= cd <=10
- setar na TPL o valor vl seguindo a opção op

SETVELOMAX(op,vl)

ENTRADA: op: item setado
op= 1 a 6 número da junta
op= 7 a 9 direção cartesiana PX, PY e PZ
op= 10 garra
vl: valor setado
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: setar valor da velocidade máxima para cálculo de trajetória. Valor setado na TPL
ALGORITMO
- verificar se 1 <= cd <=10
- setar na TPL o valor vl seguindo a opção op

SETNPT(op,vl)

ENTRADA: opção: OP = 1 setar valor em vl
= 0 assumir cálculos default
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: fixar o número de pontos de discretização para todos os movimentos do robô. O cálculo do incremento de tempo é assumido como: TF/VL, onde, TF:tempo total de movimento e VL: número de pontos de discretização (10 ≤ VL ≤ 512)
ALGORITMO
- se op = 1 setar a flag de número de pontos de discretização e o número de pontos
- se op = 0 resetar flag de nmero de pontos de discretização

***** COMANDOS DE ORIENTAÇÃO

ORIFIX()

ENTRADA:
SAÍDA: orientação fixa ativada
DESCRIÇÃO: retornar a orientação fixa ativa
ALGORITMO
- consultar a TPL verificando se flag de orientação fixa está ativa, se não => erro
- consultar a TPL e retornar a orientação fixa

STORIENTA(tt)

ENTRADA: tt: orientação desejada
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: Efetuar mudança de orientação da garra.
ALGORITMO
- obter a posição-orientação corrente consultando a TPL
- se opção de ORIENTAÇÃO mantendo a posição corrente estiver setada, construir uma STRUCT com a orientação desejada e posição corrente
- se opção de ORIENTAÇÃO não mantendo a posição corrente estiver setada, corrigir o valor angular das juntas:
  θ_5 = asin(KX*cos(θ_1)+KY*sin(θ_1))
  se $\theta_5 \cong \pm 90°$ => erro
  θ_6 = asin(-(cos(θ_1)*NX+sin(θ_1)*NY)/cos(θ_5))
  θ_4 = asin(KZ/cos(θ_5))-θ_2-θ_3
  - calcular o modelo geométrico direto
- calcular o modelo geométrico inverso
- setar movimento LINEAR
- mover para a posição-orientação desejada
- salvar na TPL a orientação constante
- setar flag de orientação constante

MVORI(op)

ENTRADA: op: opção de movimento
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: setar o tipo de movimento para orientação da garra:
op = 0 DESATIVAR flag
op = 1 MTPOS setar orientação da garra mantendo a posição corrente da garra
op = 2 SMTPOS setar orientação da garra sem manter a posição corrente da garra

ALGORITMO
- setar flag de orientação com a opção op na TPL

RSTORIENTA()

ENTRADA:
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: resetar a flag de orientação constante
ALGORITMO
- resetar a flag de orientação constante na TPL

***** COMANDOS PARA MODELAMENTO

MDGEODI(tt)

ENTRADA: tt: valor angular das juntas
SAÍDA: posição orientação da garra
DESCRIÇÃO: Cálculo do modelo geométrico direto seguindo a formulação desenvolvida no item: MODELO GEOMÉTRICO DIRETO
ALGORITMO
- calcular a posição-orientação da garra:

```
Nx= -c1*c5*s6-c6*s1*s234+s1*s6*s5*c234
Ny= -c1*s5*s6*c234+c1*c6*s234-c5*s1*s6
Nz=  -c6*c234-s5*s6*s234
Ox= -c1*c5*c6+s1*s5*c6*c234+s1*s6*s234
Oy= -c1*c6*s5*c234-s6*c1*s234-c5*c6*s1
Oz= s6*c234-s5*c6*s234
Kx= -c1*s5-s1*c5*c234
Ky= c1*c5*c234-s1*s5
Kz= c5*s234
Px= -c1*d1-c2*s1*d2-s1*c23*d3-d4*s1*c234+Kx*H
Py= d4*c1*c234+d3*c1*c23+c1*c2*d2-s1*d1+Ky*H
Pz= d4*s234+s23*d3+s2*d2+d0+Kz*H
```

MDGEOIN(tt)

ENTRADA: tt: posição-orientação da garra
SAÍDA: valor angular das juntas
DESCRIÇÃO: Cálculo do modelo geométrico inverso seguindo a formulação desenvolvida no item: MODELO GEOMÉTRICO INVERSO
ALGORITMO
- se θ_1 destravado faça:

```
ro = sqrt( tf.px*tf.px+tf.py*tf.py)
alf = atang(tf.px,tf.py,1)
```

-cálculo da primeira opção

θ_{1a} = atang(-d1/ro, sqrt(1-d1*d1/(ro*ro)),1) -alf

- cálculo da segunda opção

θ_{1b} = atang(-d1/ro, -sqrt(1-d1*d1/(ro*ro)),1) -alf

- verificar se θ_4 pertence ao intervalo angular, se não ERRO
- verificar o menor deslocamento angular a assumir como θ_1

- se θ_5 destravado faça

```
k1 = c1*tf.nx+s1*tf.ny
k2 = c1*tf.ox+s1*tf.oy
k3 = c1*tf.kx+s1*tf.ky
k4 = sqrt(k1*k1+k2*k2)
```

- cálculo da primeira opção

θ_{5a} = atang(-k3, -k4,2);

- cálculo da segunda opção

θ_{5b} = atang(-k3, k4,2);

- verificar se θ_5 pertence ao intervalo angular, se não ERRO
- verificar o menor deslocamento angular a assumir como θ_5

- se θ_6 destravado faça

θ_6 = asin(-k1/c5)

- verificar se θ_6 pertence ao intervalo angular, se não ERRO

- se θ_3 destravado faça

```
k1 = (c1*tf.py-s1*tf.px)
k2 = (c1*tf.ky-s1*tf.kx)
k3 = tf.pz-d0
k4 = k3-d4*tf.kz/c5
k5 = k1-d4*k2/c5
k24 = d4*s5*s6*(-s1*tf.nx+c1*tf.ny)
      +d4*s5*c6*(-s1*tf.ox+c1*tf.oy)
      -d4*c5*(-s1*tf.kx+c1*tf.ky) -s1*tf.px+c1*tf.py
k34 = d4*s5*s6*tf.nz+d4*s5*c6*tf.oz
      -d4*c5*tf.kz+tf.pz-d0
c3 = (k24*k24+k34*k34-d2*d2-d3*d3)/(2.*d2*d3)
k6 = (k4*k4+k5*k5-d3*d3)/(2.*d2)
s3 = sqrt(1-c3*c3)
```

- cálculo da primeira opção

```
t3a = asin(s3)
```

- cálculo da segunda opção

```
t3b = asin(-s3)
```

- verificar se θ_3 pertence ao intervalo angular, se não ERRO
- verificar o menor deslocamento angular a assumir como θ_3

- se θ_2 destravado faça

```
k5 = k1-k2*d4/c5
k4 = k3-tf.kz*d4/c5
k6 = s3*d3
```

θ_2 = valor corrente - TPL

```
k7 = 1000.
```

- cálculo iterativo de θ_2

```
enquanto |k7-θ2| > ERRO_MÍNIMO faça
  ca = cos(θ2)
  sa = sin(θ2)
  k7 = θ2
  θ2a = (-sqrt(ca*ca*k5*k5+2*ca*ca*k4*k4
        -2*ca*sa*k5*k4-2*ca*k6*k4
        +2*sa*sa*k5*k5+sa*sa*k4*k4+2*sa*k5*k6)
        -ca*k5+ca*θ2*k4
        -sa*k5*θ2-sa*k4)/(ca*k4-sa*k5)
  θ2b = (sqrt(ca*ca*k5*k5+2*ca*ca*k4*k4
        -2*ca*sa*k5*k4-2*ca*k6*k4
        +2*sa*sa*k5*k5+sa*sa*k4*k4+2*sa*k5*k6)
        -ca*k5+ca*θ2*k4
       -sa*k5*θ2-sa*k4)/(ca*k4-sa*k5)
- verificar o menor deslocamento angular a assumir
  como θ2
- verificar se θ2 pertence ao intervalo angular, se
  não ERRO
- cálculo de θ4
  θ4 = atang(-s2*k2+c2*tf.kz,c2*k2+s2*tf.kz,2) - tf.t3
- verificar se θ4 pertence ao intervalo angular, se não
  ERRO
```

***** COMANDOS DE TRAJETÓRIA

TRACO(a,po,pf,tf,dt,*sp,*st,*n,kl)

```
ENTRADA:   a: percentual no tempo de aceleração
          po: ponto inicial
          pf: ponto final
          tf: tempo final
          dt: incremento de tempo
          kl: tipo de perfil de velocidade
                1-TRAPEZOIDAL
                2-SENOIDAL
SAÍDA:    *sp: ponteiro dos pontos interpolados
          *st: ponteiro de tempo para os pontos interpolados
           *n: ponteiro com o número de pontos interpolados
DESCRIÇÃO: Cálculo do perfil de trajetória
ALGORITMO:
  - para KL = TRAPEZOIDAL faça
    - se a > 0.5 => erro
    - calcular tempo de início de aceleração nula: tk=a*tf
    - calcular ponto de início de traletória com
      aceleração nula:  pk = (a*pf+2*po-3*a*po)/(2-2*a)
    - cálculo da velocidade máxima:
                              vm = (pf-2*pk+po)/(tf-2*tf)
```

- cálculo da aceleração máxima:
  am = 2*(pk-po)/(tf*tf)
- cálculo do número de interpolações para
  - primeiro período com aceleração: i1 = tk/dt
  - período de aceleração nula: i2 = (tf-tk)/dt
  - segundo período com aceleração: i3 = tf/dt
  - se i3 > núm. máx. de pontos interpolados => erro
- interpolação para o primeiro trecho acelerado
  - para i = 0 até i1 faça
    - tempo: *(st+i)= i*dt
    - interpolação: *(sp+i)= am*tempo^2/2+po
- interpolação para o trecho de aceleração nula
  - para i = i1+1 até i2 faça
    - tempo: *(st+i)= i*dt
    - interpolação: *(sp+i)= vm*tempo+pk-vm*tk
- interpolação para o segundo trecho acelerado
  - para i = i2+1 até i3 faça
    - tempo: *(st+i)= i*dt
    - interpolação:
      *(sp+i)= am*tf*tempo-am*tempo^2/2 + pf +
      - am*tf^2/2

- para KL = SENOIDAL faça
  - cálculo da aceleração máxima:
    am = PI*PI*(pf-po)/(2*tf^2)
  - cálculo do número de interpolações i3 = tf/dt
    - se i3 > núm. máx. de pontos interpolados => erro
  - interpolação
    - para i = 0 até i3 faça
      - tempo: *(st+i)= i*dt
      - interpolação:
        *(sp+i)= (pf+po)/2 - am*tf^2/(PI^2)*
        *cos(PI*tempo/tf)

VLSETO

ENTRADA:
SAÍDA: perfil de velocidade setado
DESCRIÇÃO: retornar o tipo de perfil de velocidade setado
ALGORITMO
- consultar a TPL e retornar o tipo de perfil de velocidade setado

VLPERFIL(op)

ENTRADA: op: opção de perfil
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: setar o tipo de perfil de velocidade, consultar item: GERADOR DE PERFIL DE VELOCIDADE
  op = 1 perfil trapezoidal
  op = 2 perfil senoidal
ALGORITMO
- setar flag de perfil de velocidade com a opção op na TPL

***** COMUNICAÇÃO COM O "TEACH-IN"

LOADD(nm)

```
ENTRADA:  nm: nome do arquivo de pontos
SAÍDA: ptt[]: vetor tipo struct t6 dos pontos setados no
              modo TEACH-IN
DESCRIÇÃO: Efetuar a leitura dos pontos setados no modo
           TEACH-IN, do arquivo de nome nm para o vetor
           ptt[]
ALGORITMO
  - abrir arquivo nm para leitura
  - ler do arquivo o número de pontos setados: vnp
  - para i = 0 até vnp faça
     - ler do arquivo:
        - valor angular das juntas: T1, T2, T3, T4, T5, T6
        - posição da garra: PX, PY, PZ
        - orientação da garra: NX, NY, NZ,
                               OX, OY, OZ,
                               KX, KY, KZ
        - valor da garra: GR
        - código de erro: ERRO
```

SAVED(nm)

```
ENTRADA:  nm: nome do arquivo de pontos
SAÍDA:  operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: Efetuar  a  escrita  dos  pontos  setados  pelo
           usuário para o modo TEACH-IN, no arquivo de nome
           nm
ALGORITMO
  - abrir arquivo nm para leitura
  - escrever no arquivo o número de pontos setados: vnp
  - para i = 0 até vnp faça
     - escrever no arquivo:
        - valor angular das juntas: T1, T2, T3, T4, T5, T6
        - posição da garra: PX, PY, PZ
        - orientação da garra: NX, NY, NZ,
                               OX, OY, OZ,
                               KX, KY, KZ
        - valor da garra: GR
        - código de erro: ERRO
```

STCONTR(cd,vl)

ENTRADA: cd: número do slot ou parâmetro de controle
vl: valor a ser setado
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: setar o valor vl para o slot ou parâmetro de controle cd
ALGORITMO
- verificar se 0 <= cd <= 9
- setar na TPL, para o código cd o valor vl

***** COMANDOS DE MENSAGEM

STERRO(op)

ENTRADA: op: opção
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: setar modo de operação em condição de erro.
Opção:
op = 1 abortar a execução do programa para a primeira ocorrência de um erro
op = 0 imprimir mensagem de erro e não abortar programa
ALGORITMO
- setar na TPL a opção op de modo de operação em erro

STMSG(op)

ENTRADA: op: opção ATIVAR DESATIVAR
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: ativar ou desativar a emissão de mensagens
ALGORITMO
- se op = ATIVAR setar a flag na TPL ativando a emissão de mensagens
- se op = DESATIVAR setar a flag na TPL desativando a emissão de mensagens

DFMSG(cd,mg)

ENTRADA: cd: número da mensagem 250 < cd < 299
mg: mensagem
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: definir para o sistema uma mensagem de número cd e nome mg pelo usuário
ALGORITMO
- verificar se 250 < cd < 299, se não ERRO
- ativar mensagem de número cd e copiar para a área de memória destinada a mensagens, o texto mg

RSTMSG(cd)

ENTRADA: cd: código da mensagem 250 < cd < 299
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: desativar mensagem de número cd definida pelo usuário
ALGORITMO
- verificar se 250 < cd < 299, se não ERRO
- desativar mensagem de número cd e retirar o texto da área de memória destinada a mensagens

***** COMANDOS PARA INICIALIZAÇÃO

ZEROMAQO

ENTRADA:
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: conduzir o robô para a posição de ZERO-MÁQUINA e resetar os valor da TPL, desativar garra e objetos ativos
ALGORITMO
- setar o valor angular das juntas para a posição de zero-máquina
- calcular modelo geométrico direto
- setar flag de movimento LIVRE, resetar flag de orientação
- executar rotina MOVER() para a posição-orientação de zero-máquina
- executar rotina de inicialização da TPL

CASA()

ENTRADA:
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: conduzir o robô para a posição de CASA e setar os valores da TPL para a nova posição. São três as posições de CASA:
- posição de cotovelo livre
- posição de cotovelo UP
- posição de cotovelo DOWN

ALGORITMO
- consultar a TPL e obter o tipo de cotovelo setado
- executar a rotina COTOVELO() para a opção setada setando os valores da TPL

COTOVELO(op)

ENTRADA: op: tipo de cotovelo
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: posicionar o robô nas posições:
op = 1 cotovelo UP onde $0 \leq \theta_2 \leq PI/2$
$-PI/2 \leq \theta_3 \leq 0$
op = 2 cotovelo DOWN onde $-PI/2 \leq \theta_2 \leq 0$
$0 \leq \theta_3 \leq PI/2$
op = 3 cotovelo LIVRE onde $-PI/2 \leq \theta_2 \leq PI/2$
$-PI/2 \leq \theta_3 \leq PI/2$

ALGORITMO
- setar flag de movimento LIVRE na TPL
- resetar flag de orientação na TPL
- resetar tipo de movimento de orientação
- destravar juntas
- se op = LIVRE faça
  - setar valores angulares: $\theta_1 = \theta_2 = \theta_3 = \theta_4 = \theta_5 = \theta_6 = 0$
  - setar intervalo angular: $-PI/2 \leq \theta_2 \leq PI/2$
    $-PI/2 \leq \theta_3 \leq PI/2$
  - calcular modelo geométrico direto
  - executar rotina MOVER() para a posição-orientação desejada
- se op = UP faça
  - setar valores angulares: $\theta_1 = \theta_4 = \theta_5 = \theta_6 = 0$
    $\theta_2 = PI/2$
    $\theta_3 = -PI/2$
  - setar intervalo angular: $0 \leq \theta_2 \leq PI/2$
    $-PI/2 \leq \theta_3 \leq 0$
  - calcular modelo geométrico direto
  - executar rotina MOVER() para a posição-orientação desejada

- se op = DOWN faça
  - setar valores angulares: $\theta_1 = \theta_4 = \theta_5 = \theta_6 = 0$
    $\theta_2 = -PI/2$
    $\theta_3 = PI/2$
  - setar intervalo angular: $-PI/2 \leq \theta_2 \leq 0$
    $0 \leq \theta_3 \leq PI/2$
  - calcular modelo geométrico direto
  - executar rotina MOVER() para a posição-orientação deselada

***** COMANDOS PARA A GARRA

ABRIR(vl)

ENTRADA: vl: valor absoluto de abertura da garra
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: abrir a garra até o valor vl. Se vl menor que o valor corrente o procedimento não é executado

ALGORITMO
- consultar a TPL e verificar se existe garra ativa, se não => ERRO
- consultar a TPL e verificar se vl é menor que o valor corrente, se sim => ERRO
- setar, na TPL, o valor deselado para a abertura da garra
- chamar a rotina MOVER e executar o movimento

FECHAR(vl)

ENTRADA: vl: valor absoluto de fechamento da garra
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: fechar a garra até o valor vl. Se vl maior que o valor corrente o procedimento não é executado

ALGORITMO
- consultar a TPL e verificar se existe garra ativa, se não => ERRO
- consultar a TPL e verificar se vl é maior que o valor corrente, se sim => ERRO
- setar, na TPL, o valor deselado para fechamento da garra
- chamar a rotina MOVER e executar o movimento

VLGARRA()

ENTRADA:
SAÍDA: valor corrente da garra
DESCRIÇÃO: retornar o valor corrente da garra
ALGORITMO
- consultar a TPL e retornar o valor coreente da garra

CDGARRA()

ENTRADA:
SAÍDA: código da garra ativa
DESCRIÇÃO: retornar o código da garra ativa
ALGORITMO
- consultar a TPL e retornar o código da garra ativa

TROCAGARRA(op)

ENTRADA: op: tipo de garra
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: trocar a garra do manipulador.
op = 0 desativa garra
op = 1 setar garra número 1
ALGORITMO
- consultar a TPL e obter a posição-orientação de troca da garra
- executar movimento
- setar na TPL a garra ativa

PTGARRA(tt)

ENTRADA: tt: posição-orientação para troca da garra
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: setar na TPL a posição-orientação de troca da garra
ALGORITMO
- tomar a posição-orientação de troca da garra: tt e setar a TPL

***** COMANDOS PARA MANIPULAÇÃO DE OBJETOS

DFOBJ(tt,op)

ENTRADA: tt: parâmetros do objeto
op: opção de definição
SAÍDA: parâmetros do objeto
DESCRIÇÃO: definir objeto para o sistema.
Opções:
op = 1 assumir como entrada a posição-orientação do objeto
op = 2 assumir como entrada o valor angular de posição-orientação do objeto
op = 3 assumir posição-orientação default
consultar item: OBJETOS

ALGORITMO
- incrementar número de objetos definidos
- se op = 1 setar na área de memória de objetos os parâmetros definidos em tt
- se op = 2 calcular a posição-orientação do objeto e setar na área de memória de objetos os parâmetros calculados e os definidos em tt
- se op = 3 setar na área de memória de objetos os parâmetros DEFAULT

POPXPG(cd,tt)

ENTRADA: cd: código do objeto
tt: posição-orientação de pega do objeto
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: retornar o ponto de pega do objeto cd
consultar item: OBJETOS

ALGORITMO
- verificar se objeto cd está definido
- consultar área de memória para objetos e retornar a posição-orientação de pega do objeto

STOBJETO(cd)

ENTRADA: cd: código do objeto
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: setar na TPL o objeto cd como ativo
consultar item: OBJETOS

ALGORITMO
- verificar se objeto cd está definido
- setar na TPL a flag de objeto ativo: cd

ATIVOBJ()

ENTRADA:
SAÍDA: código do objeto ativo
DESCRIÇÃO: fornecer o código do objeto ativo.
consultar item: OBJETOS
ALGORITMO
- consultar a TPL e retornar o código do objeto ativo

RMOBJ(cd)

ENTRADA: cd: número do objeto
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: zerar os dados do objeto cd
consultar item: OBJETOS
ALGORITMO
- zerar os parâmetros do objeto cd
- atualizar dados na área de memória para objetos

NOBJDEF()

ENTRADA:
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: retornar o número de objetos definidos
ALGORITMO
- consultar a TPL e retornar o número de objetos definidos

RDFOBJ(tt,op,cd)

ENTRADA: tt: parâmetros do objeto
op: opção de redefinição
cd: número do objeto
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: redefinir objeto para o sistema.
Opções:
op = 1 assumir como entrada a posição-orientação do objeto
op = 2 assumir como entrada o valor angular de posicão-orientação do objeto
op = 3 assumir posição-orientação default
consultar item: OBJETOS
ALGORITMO
- se op = 1 setar na área de memória de objetos, para o objeto cd, os parâmetros definidos em tt

- se op = 2 calcular a posição-orientação do objeto (cd) e setar na área de memória de objetos os parâmetros calculados e os definidos em tt
- se op = 3 setar na área de memória de objetos os parâmetros DEFAULT, para o objeto cd

OBJCORRENTO

ENTRADA:
SAÍDA: dados do objeto ativo
DESCRIÇÃO: retornar os dados do objeto ativo
ALGORITMO
- consultar a TPL verificando se existe um objeto ativo, se não => erro
- retornar os dados do objeto ativo

***** COMANDOS DE USO GERAL

POCORRENTO

ENTRADA:
SAÍDA: posição-orientação corrente
DESCRIÇÃO: retornar a posição-orientação corrente
ALGORITMO
- consultar a TPL e retornar a posição-orientação corrente

CVGRRD(vl)

ENTRADA: vl: valor em graus
SAÍDA: valor em radianos
DESCRIÇÃO: converter graus em radianos
ALGORITMO
- calcular: valor = vl*PI/180

CVRDGR(vl)

ENTRADA: vl: valor em radianos
SAÍDA: valor em graus
DESCRIÇÃO: converter radianos em graus
ALGORITMO
- calcular: valor = vl*180/PI

FIXANG(cd)

ENTRADA: cd: número da junta
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: torna FIXA a junta de número cd
ALGORITMO
- se cd = 0 setar flag na TPL tornando fixa todas as juntas
- se cd = 1 ou 2 ou 3 ou 5 ou 6 setar flag na TPL tornando fixa a junta de número cd

LVRANG(cd)

ENTRADA: cd: número da junta
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: torna LIVRE a junta de número cd
ALGORITMO
- se cd = 0 setar flag na TPL tornando livre todas as juntas
- se cd = 1 ou 2 ou 3 ou 5 ou 6 setar flag na TPL tornando livre a junta de número cd

CARGA(nc)

ENTRADA: nc: nível de carga
SAÍDA: operação correta (VERDADE) ou falsa (FALSO)
DESCRIÇÃO: setar nível de carga
ALGORITMO
- setar o nível de carga na TPL com o valor nc

NPONTOS(op)

ENTRADA: op: opção
SAÍDA: número de pontos interpolados
DESCRIÇÃO: retornar o número de pontos interpolados na execução de um movimento:
op = 1 último movimento
op = 2 penúltimo movimento
op = 3 ante-penúltimo movimento
ALGORITMO
- verificar se 1 <= op <=3, se não ERRO
- consultar a TPL e retornar o número de pontos interpolados segundo a opção op

PARTE B

MODELAMENTO GEOMÉTRICO E TRAJETÓRIA

INTRODUÇÃO

Os modelos são abstrações efetuadas com o intuito de permitir a representação, reprodução e análise de algo, no caso, os movimentos do robô.

Na parte B apresentar-se-á o modelamento geométrico direto e inverso, o gerador de trajetória e o gerenciador de movimentos, responsáveis pela localização teórica e geração dos movimentos da garra.

MODELAMENTO GEOMÉTRICO

MODELO GEOMÉTRICO DIRETO

O modelo geométrico direto (MGD) permite determinar a posição e orientação teórica dos links e da garra do robô dado o valor angular ou deslocamento das juntas [BENNETT-89] [CRAIG-86] [FU-87] [LEE-82] [MOORING-89] [NNAGY-86] [RANKY-85]. Denavit e Hartenberg (DH) [DENAVIT-55] propuseram uma metodologia para representar as relações geométricas entre os links com o menor número de parâmetros (4 parâmetros). No espaço cartesiano tri-dimensional são necessários 6 parâmetros para localizar um corpo: 3 parâmetros de posição e 3 parâmetros angulares para orientação. Segundo DH um parâmetro de posição e um de orientação são fixos. O método proposto consiste em representar esta relação por meio de matrizes de transformações homogêneas, descrevendo a posição e orientação de um link em relação a outro, de tal forma que, com operações matriciais possa-se obter o MGD. Neste trabalho, utilizam-se os princípios propostos por DH, contudo, seguindo a sistemática apresentada a seguir.

A matriz transformada homogênea para o espaço cartesiano tri-dimensional consiste de uma matriz $A^{(4\times4)}$:

$$ {}^{i}A_{i+1} = \left[\begin{array}{c|c} {}^{i}R_{i+1}^{(3\times3)} & {}^{i}P_{i+1}^{(3\times1)} \\ \hline 0 \quad 0 \quad 0 & 1 \end{array}\right] $$

Os índices i e $i+1$ indicam que a matriz A fornece a relação de posição e orientação do sistema de referência $i+1$ em relação ao sistema i. A matriz R fornece a orientação e o vetor P a posição da origem do referencial $i+1$ em relação ao referencial i. Na última linha tem-se [0 0 0 1], onde os três primeiros valores atuam como fator de perspectiva e o quarto como fator de escala, sendo utilizados em computação gráfica [GIOLI-78] . A matriz R é obtida pela composição de rotações dos eixos cartesianos do referencial $i+1$ em relação ao referencial i. O mesmo se aplica ao vetor P, que consiste de uma translação da origem de um referencial em relação ao outro [CRAIG-86] [MALCOLM-88] [PAUL-83] [SNYDER-85].

Para a sua correta localização, fixa-se um sistema de referência em cada link e determina-se a matriz transformação homogênea entre dois links consecutivos, sendo que um deles, normalmente localizado na base do robô, deve apresentar rotação e velocidade linear nulas, assumindo as características de um sistema referencial inercial [MERIAN-81], a partir do qual serão realizadas todas as medidas.

Para a localização dos sistemas de referência adotou-se o seguinte procedimento:

- alinhar o eixo Z_0 (o índice zero indica sistema de referência inercial 0_0) ao longo do eixo de movimento do primeiro link, localizando o sistema de coordenadas inercial. A direção dos eixos X_0 e Y_0 deve ser convenientemente estabelecida, de tal forma, a minimizar o número de parâmetros para a localização do referencial 0_1 em relação ao referencial 0_0, simplificando o modelamento.

- para os links de i=1 a GL, onde GL é o número de graus de liberdade, repita os passos:
  - posicionar o eixo Z_i na direção do movimento da junta i, no link i
  - posicionar a origem do i-ésimo sistema de referência na intersecção dos eixos Z_i e Z_{i-1}, na normal comum aos eixos Z_i e Z_{i-1}, ou então, posicionar a origem do i-ésimo sistema de referência no ponto de intersecção de duas juntas consecutivas
  - alinhar o eixo X_i na mesma direção do eixo X_{i-1}, ou então, de forma a minimizar o número de rotações necessárias para orientar o referencial

- para i=GL até 1 determinar as matrizes de rotação R e os vetores P, estabelecendo as matrizes A.

Este procedimento não é tão automatizado quanto ao proposto por DH, no entanto, propicia maior liberdade na alocação dos sistemas de coordenadas. A aplicação do método de DH é demasiadamente fechada, como uma "caixa preta" [JONES-88], onde o projetista perde a sensibilidade dos procedimentos que produzirão as matrizes de transformação, conduzindo a erros na especificação dos quatros parâmetros de DH. Quanto ao resultado, o produto final é idêntico.

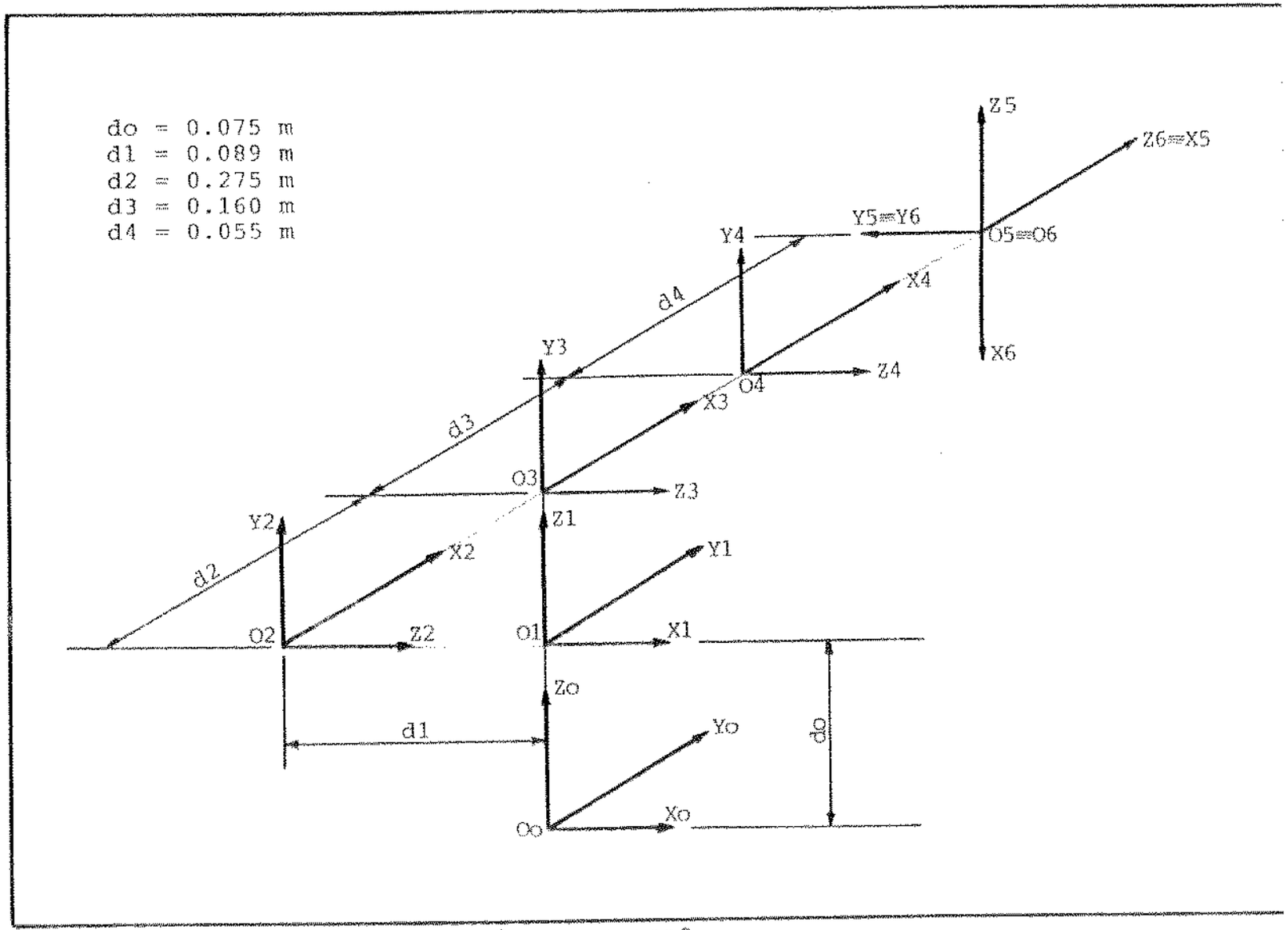

FIGURA F13 - SISTEMAS DE REFERÊNCIA DO ROBÔ

Estabelecidas as matrizes A, a localização da garra em relação ao referencial inercial é dada pelo produto matricial:

$$T_n = {}^{0}A_n = {}^{0}A_1 * {}^{1}A_2 * \ldots * {}^{n-1}A_n = \begin{bmatrix} Nx & Ox & Kx & Px \\ Ny & Oy & Ky & Py \\ Nz & Oz & Kz & Pz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

onde a primeira coluna da matriz resultante $(T)_n$ indica a orientação do versor X do referencial n em relação ao n referencial inercial. Analogamente para a segunda e terceira colunas: versor Y e versor Z. A quarta coluna indica a posição absoluta da origem do n-ésimo sistema de referência, em relação ao referencial inercial.

Na figura F13 tem-se a localização dos sistemas de referência assumidos neste trabalho, produzindo as seguintes matrizes de transformação homogênea:

$${}^{0}A_1 = ROT(Z_1, \theta_1) * TRANS(0,0,d_0) =$$

$$= \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & d_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^{1}A_2 = ROT(X,90) * ROT(Y,90) * ROT(Z_2, \theta_2) * TRANS(-d_1,0,0) =$$

$$= \begin{bmatrix} 0 & 0 & 1 & -d_1 \\ c_2 & -s_2 & 0 & 0 \\ s_2 & c_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^{2}A_3 = ROT(Z_3, \theta_3) * TRANS(d_2,0,0) =$$

$$= \begin{bmatrix} c_3 & -s_3 & 0 & d_2 \\ s_3 & c_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{3}A_{4} = ROT(Z_4, \theta_4) * TRANS(d_3, 0, 0) =$$

$$= \begin{bmatrix} c4 & -s4 & 0 & d3 \\ s4 & c4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{4}A_{5} = ROT(X, -90) * ROT(Z_5, \theta_5) * TRANS(d_4, 0, 0) =$$

$$= \begin{bmatrix} c5 & -s5 & 0 & d4 \\ 0 & 0 & 1 & 0 \\ -s5 & -c5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{5}A_{6} = ROT(Y, 90) * ROT(Z_6, \theta_6) =$$

$$= \begin{bmatrix} 0 & 0 & 1 & 0 \\ s6 & c6 & 0 & 0 \\ -c6 & s6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Localização da garra no sistema inercial:

$$T_6 = {}^{0}A_{6} = {}^{0}A_{1} * {}^{1}A_{2} * {}^{2}A_{3} * {}^{3}A_{4} * {}^{4}A_{5} * {}^{5}A_{6} = \begin{bmatrix} Nx & Ox & Kx & Px \\ Ny & Oy & Ky & Py \\ Nz & Oz & Kz & Pz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

onde

```
Nx= -C1*C5*S6-C6*S1*S234+S1*S6*S5*C234
Ny= -C1*S5*S6*C234+C1*C6*S234-C5*S1*S6
Nz=  -C6*C234-S5*S6*S234
Ox= -C1*C5*C6+S1*S5*C6*C234+S1*S6*S234
Oy= -C1*C6*S5*C234-S6*C1*S234-C5*C6*S1
Oz= S6*C234-S5*C6*S234
Kx= -C1*S5-S1*C5*C234
Ky= C1*C5*C234-S1*S5
Kz= C5*S234
Px= -C1*d1-C2*S1*d2-S1*C23*d3-d4*S1*C234
Py= d4*C1*C234+d3*C1*C23+C1*C2*d2-S1*d1
Pz= d4*S234+S23*d3+S2*d2+do
```

com s, c, d indicando seno, cosseno e distância entre as origens de dois sistemas de referência consecutivos.

O modelo geométrico direto foi calculado simbolicamente utilizando o software REDUCE [MENDELE-89] [HEARN-85] [RAYNA-87].

A localização da garra é determinada utilizando-se a matriz T6 uma vez que está fixa na origem do sexto sistema de referência, como mostrado na figura F14.

Um objeto é localizado em relação ao seu centro geométrico. Enquanto estiver ativo, ou seja, o objeto está acoplado à garra, sua posição absoluta é calculada pós-multiplicando a matriz T6 pela matriz T6obj. Assim, como o objeto é simétrico (consultar nível 5 - objetos) a posição do seu centro geométrico é definida pelo vetor:

$$ {}^{6}P_{obj} = \begin{bmatrix} 0 \\ 0 \\ H \end{bmatrix} $$

onde o índice 6 indica que o objeto está temporariamente definido em relação ao sexto sistema de referência. O valor de H corresponde a metade da sua altura. A orientação é a mesma da garra, pois, estão rigidamente acoplados, produzindo a matriz identidade:

$$ {}^{6}R_{obj} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} $$

Calculando a localização absoluta do objeto por:

$$T6obj = T6 * {}^{6}Tobj$$

onde

$$ {}^{6}Tobj = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & H \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

tem-se

$$ T6obj = \begin{bmatrix} Nx & Ox & Kx & Px+Kx*H \\ Ny & Oy & Ky & Py+Ky*H \\ Nz & Oz & Kz & Pz+Kz*H \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

que conduz a uma correção no vetor posição do MGD, definido por:

$$Px = -c1*d1-c2*s1*d2-s1*c23*d3-d4*s1*c234+Kx*H$$
$$Py = d4*c1*c234+d3*c1*c23+c1*c2*d2-s1*d1+Ky*H$$
$$Pz = d4*s234+s23*d3+s2*d2+d0+Kz*H$$

As figura F15 e F16 mostram a área de trabalho máxima do robô projetada no planos XY e YZ em relação ao sistema de coordenadas absoluto. Como a linguagem de programação permite a manipulação de objetos, a área de trabalho é aumentada em função da dimensão H (altura do objeto).

Adotou-se como intervalo angular para as juntas, os seguinte valores:

$$0 \leq \theta_1 \leq PI$$
$$-PI/2 \leq \theta_2 \leq PI/2$$
$$-PI/2 \leq \theta_3 \leq PI/2$$
$$-PI/2 \leq \theta_4 \leq PI/2$$
$$-PI/2 < \theta_5 < PI/2$$
$$-PI/2 \leq \theta_6 \leq PI/2$$

O intervalo angular das juntas permite uma maior área de trabalho, no entanto, a redução faz-se necessária devido a limitações do modelo geométrico inverso.

A junta 5 não pode assumir os valores ± PI/2 em função de singularidades no modelo geométrico inverso, desenvolvido a seguir.

Assim, pode-se notar que em função da alocação do sistema de coordenadas absoluto, a área de trabalho está contida, na maior parte, no segundo e terceiro quadrantes do plano XY.

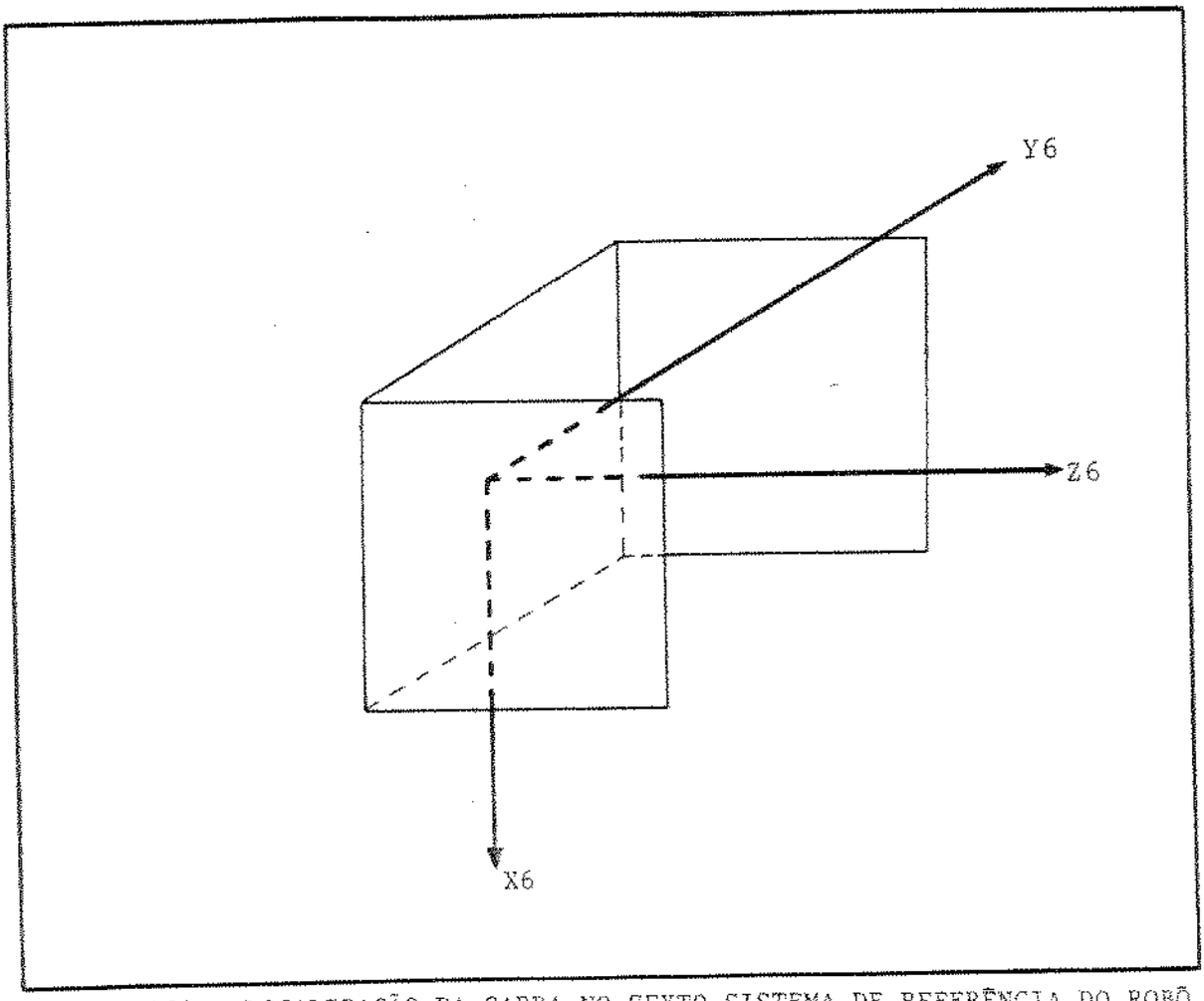

FIGURA FI4 - LOCALIZAÇÃO DA GARRA NO SEXTO SISTEMA DE REFERÊNCIA DO ROBÔ

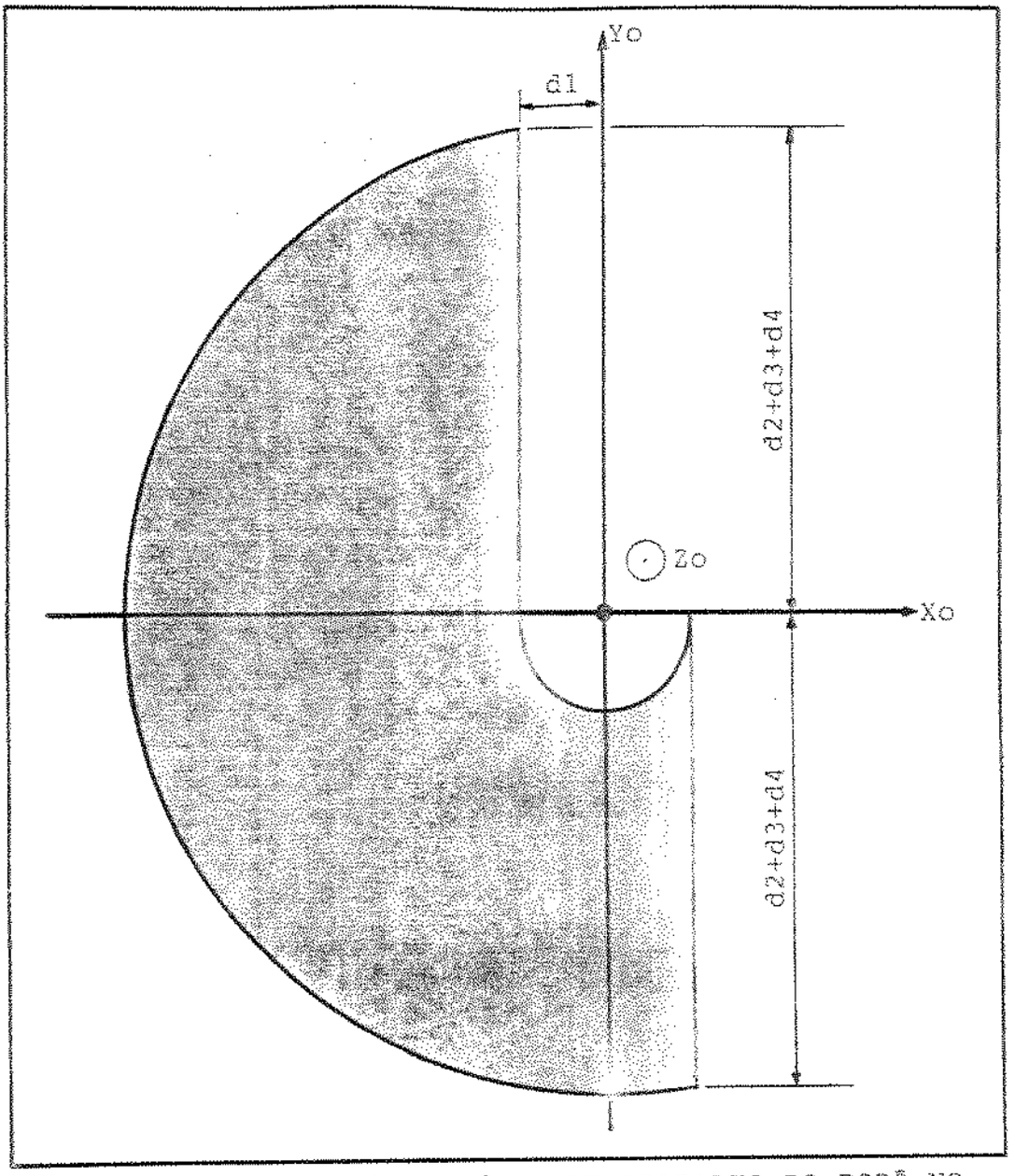

FIGURA F15 - PROJEÇÃO DA ÁREA DE TRABALHO DO ROBÔ NO PLANO XoYo

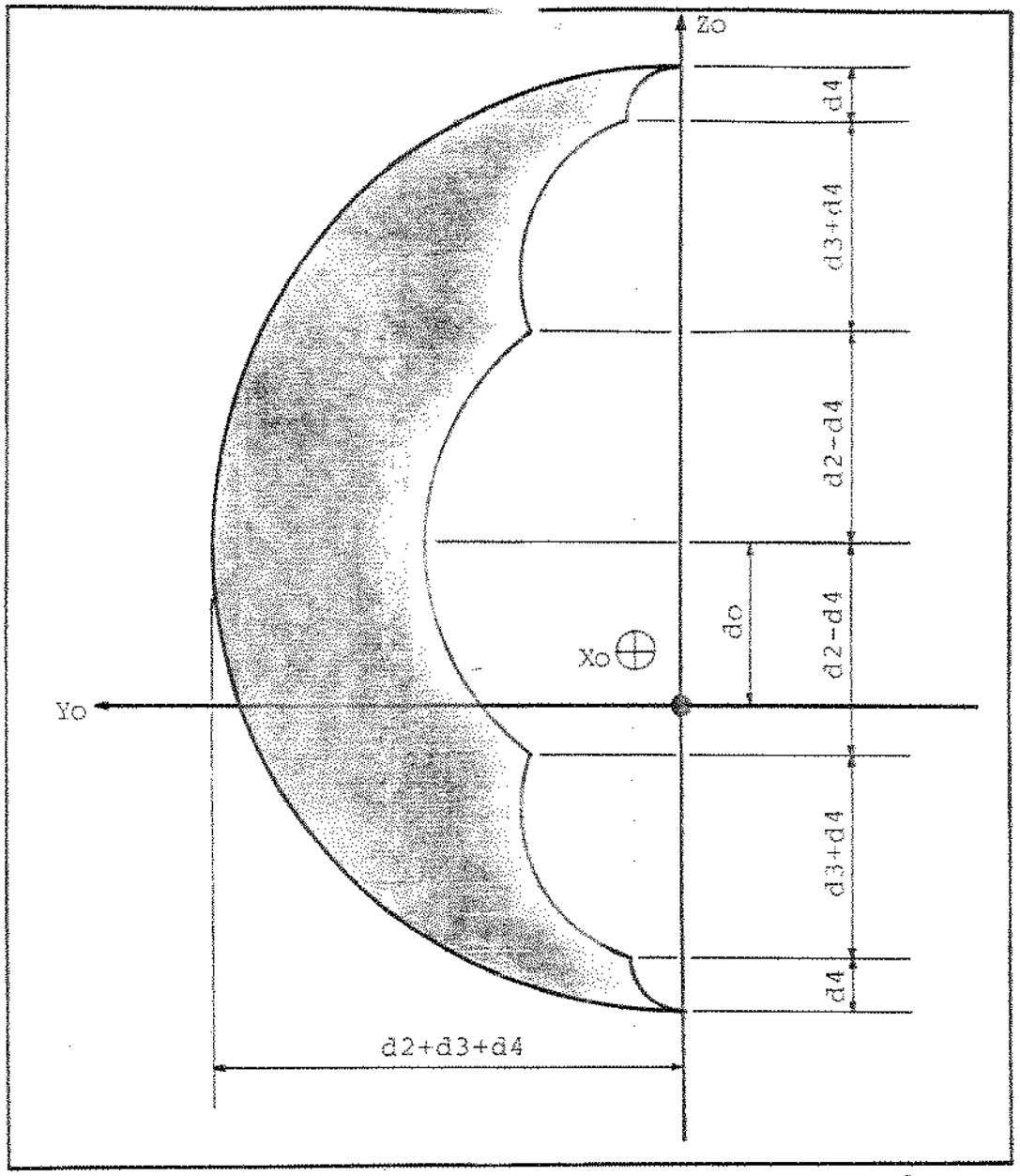

FIGURA F16 - PROJEÇÃO DA ÁREA DE TRABALHO DO ROBÔ NO PLANO YoZo

MODELO GEOMÉTRICO INVERSO

O modelo geométrico inverso (MGI) permite calcular o valor angular teórico das juntas dadas a posição e orientação do último link, ou seja, conhecida a matriz T_n obter θ_i [AKEN-84] [CRAIG-86] [PARKER-89] [PAUL-81] [PAUL-83] [WAMPLER-86] [WAMPLER-89]. Na literatura pode-se encontrar um variado número de soluções para o problema. Utilizam-se basicamente dois métodos: analítico (fórmula algébrica) e método numérico. O método analítico apresenta menor número de operações aritméticas comparado com o numérico, contudo a solução pode ser difícil de ser obtida ou não existir [TOURASS-89], pois, o equacionamento é não linear e se for "mal condicionado" pode incorporar degenerações numéricas [PAUL-83]. Outro aspecto é que as fórmulas algébricas podem ser implementadas, com facilidade, em chips dedicados ou utilizando co-processadores matemáticos [KAMEYAM-89]. O método numérico é amplamente utilizado em robôs redundantes. Este método pode ser sensível as condições iniciais, pois, envolve processo iterativo. No entanto, a sua principal característica é a modularidade e portabilidade. Robôs com características geométricas diferentes podem utilizar o mesmo procedimento de cálculo, com poucas adaptações.

A solução proposta por Paul [PAUL-83], utilizada nesta dissertação, consiste em dada a matriz posição-orientação da garra:

$$T_6 = {}^0A_1 * {}^1A_2 * {}^2A_3 * {}^3A_4 * {}^4A_5 * {}^5A_6 \ ,$$

pré-multiplicá-la pela inversa da matriz A mais a esquerda da equação:

$$({}^0A_1)^{-1} * T_6 = {}^1A_2 * {}^2A_3 * {}^3A_4 * {}^4A_5 * {}^5A_6 \ ,$$

ou então, pós-multiplicar pela inversa da matriz A mais a direita da equação:

$$T_6 * ({}^5A_6)^{-1} = {}^0A_1 * {}^1A_2 * {}^2A_3 * {}^3A_4 * {}^4A_5 \ ,$$

e, para cada igualdade procurar identificar uma relação bi-unívoca entre os elementos da matriz resultante a esquerda e a direita, identificando relações que permitam obter o valor angular de uma junta a partir de dados conhecidos. Este procedimento conduz a multiplas soluções, exigindo do projetista "certa intuição" na

escolha das relações adequadas, pois, uma solução pode ser consistente para uma configuração e não para outra, incorporando singularidades ou produzindo degenerações numéricas [PAI-89] [PAUL-83] [NAKAMUR-88]. Por exemplo, na escolha da relação

$$\theta = \text{atan}(a/b) \ ,$$

na medida que $b \to 0$ o valor de θ incorpora erros numéricos no cálculo do arco-tangente.

Como regra geral, a escolha das relações deve ser tal que os elementos da matriz T_6 (conhecida) estejam distribuídos na solução dos θ_n e o uso da função arco-tangente é preferencial, pois, incorpora o quadrante angular, evitando ambiguidades.

Observa-se que as relações são não lineares, envolvendo a soma e produto de funções trigonométricas, cuja resolução produz múltiplas soluções. Normalmente escolhe-se a que produza menor deslocamento angular e movimento simultâneo. No entanto, na presença de obstáculos a solução pode exigir uma pesquisa por todas as soluções, ou até, a especificação de pontos intermediários e movimento não simultâneo das juntas [BRADY-83] [CHANG-88] [DLABRA-89].

Para este trabalho, utilizando as matrizes A (modelo geométrico direto), as matrizes A^{-1} e, com o auxílio do software REDUCE para cálculo simbólico, determinou-se o modelo geométrico inverso obtendo as seguintes relações:

***** CALCULO PARA θ_1

Da igualdade $({}^0A_1)^{-1} * T_6 = {}^1A_6$, considerando a relação especificada na linha 1 - coluna 4:

$$c_1 * Px + s_1 * Py = d_1$$

obtém-se

$$\theta_1 = \text{atan2}\left(\pm \sqrt{1 - d_1 * d_1 / (r * r)} \ , \ -d_1 / r \right) - \text{atan2}(Py/Px)$$

onde $r = \sqrt{Px*Px + Py*Py}$

atan2: arco tangente duplo (intervalo: -PI a PI)

***** CÁLCULO PARA θ_5

Da igualdade $({}^0A_1)^{-1} * T_6 = {}^1A_6$, considerando as relações:

linha 1 - coluna 1: $K1 = c_1 * N_x + s_1 * N_y = -c_5 * s_6$
linha 1 - coluna 2: $K2 = c_1 * O_x + s_1 * O_y = -c_5 * c_6$
linha 1 - coluna 3: $K4 = c_1 * K_x + s_1 * K_y = -s_5$

obtém-se

$$\theta_5 = \text{atan2}(-K3 , \pm K4)$$

onde

$$k3 = \sqrt{K1*K1 + K2*K2}$$

***** CÁLCULO PARA θ_6

Da igualdade $({}^0A_1)^{-1} * T_6 = {}^1A_6$, considerando as relações:

linha 1 - coluna 1: $K1 = c_1 * N_x + s_1 * N_y = -c_5 * s_6$
linha 1 - coluna 2: $K2 = c_1 * O_x + s_1 * O_y = -c_5 * c_6$

obtém-se

$$\theta_6 = \text{atan2}(K1 , K2)$$

para $\theta_5 \neq \pm 90^\circ$

***** CÁLCULO PARA θ_3

Da igualdade $({}^0A_1)^{-1} * T_6 = {}^1A_6$, considerando as relações:

linha 2 - coluna 4: $c_1 * P_y - s_1 * P_x = d_4 * c_{234} + c_2 * d_2 + c_3 * d_3$
linha 3 - coluna 4: $P_z - d_0 = d_4 * s_{234} + s_{23} * d_3 + s_2 * d_2$

e da igualdade $^{1}A_{4} = (^{0}A_{1})^{-1} * T_{6} * (^{5}A_{6})^{-1} * (^{4}A_{5})^{-1}$

considerando as relações:

linha 2 - coluna 1: $c_{234} = - s_6 * s_5 * [- s_1 * N_x + c_1 * N_y] + - s_5 * c_6 * [- s_1 * O_x + c_1 * O_y] + - c_5 * [- s_1 * K_x + c_1 * K_y]$

linha 3 - coluna 1: $s_{234} = - N_z * s_6 * s_5 - O_z * s_5 * c_6 + K_z * c_5$

obtém-se

$$\theta_3 = \arcsin (\sqrt{1 - c_3 * c_3})$$

onde $c3 = \dfrac{K24*K24 + K34*K34 - d_2 * d_2 - d_3 * d_3}{2 * d_2 * d_3}$

$$K24 = d_4 * s_5 * s_6 * [- s_1 * N_x + c_1 * N_y] + + d_4 * s_5 * c_6 * [- s_1 * O_x + c_1 * O_y] + - d_4 * c_5 * [- s_1 * K_x + c_1 * K_y] - s_1 * P_x + c_1 * P_y$$

$$K34 = d_4 * s_5 * s_6 * N_z + d_4 * s_5 * c_6 * O_z - d_4 * c_5 * K_z + P_z - d_0$$

***** CÁLCULO PARA θ_2

Da igualdade $(^{1}A_{2})^{-1} * (^{0}A_{1})^{-1} * T_{6} = {}^{2}A_{6}$ considerando as relações:

linha 2 - coluna 4: $- c_1 * s_2 * P_y - c_2 * d_0 + c_2 * P_z + s_1 * s_2 * P_x = = s_{34} * d_4 + s_3 * d_3$

linha 2 - coluna 3: $c_5 * s_{34} = - s_2 * [c_1 * K_y - s_1 * K_x] + c_2 * K_z$

obtém-se

$$- s_2 * K5 + c_2 * K4 - s_3 * d_3 = 0$$

onde

$K1 = c_1 * P_y - s_1 * P_x$
$K2 = c_1 * K_y - s_1 * K_x$
$K3 = P_z - d_0$

$$K4 = K3 - \frac{K2*d4}{c5}$$

$$K5 = -\frac{K2*d4}{c5} + K1$$

Nota-se que a equação é não linear para θ_2, pois, é função de s2 e c2. Uma solução seria proceder de forma análoga ao cálculo de θ_1 obtendo uma forma algébrica, no entanto, implementamos a seguinte solução:

- expandir s2 e c2 em série de TAYLOR [KAPLAN-82] até a segunda ordem:

$$s2 = s(\theta i) + c(\theta i) * (\theta 2 - \theta i) - s(\theta i) * \frac{(\theta 2 - \theta i)^2}{2} ,$$

$$c2 = c(\theta i) - s(\theta i) * (\theta 2 - \theta i) - c(\theta i) * \frac{(\theta 2 - \theta i)^2}{2} ,$$

substituir na equação anterior e solucionar a equação do segundo graus resultante. Utilizando o software REDUCE obteve-se as seguintes respostas:

```
θ2a=(-sqrt(ca*ca*k5*k5+2*ca*ca*k4*k4-2*ca*sa*k5*k4-2*ca*k6*k4
     +2*sa*sa*k5*k5+sa*sa*k4*k4+2*sa*k5*k6)-ca*k5+ca*tf.t2*k4
     -sa*k5*tf.t2-sa*k4)/(ca*k4-sa*k5)
θ2b=(sqrt(ca*ca*k5*k5+2*ca*ca*k4*k4-2*ca*sa*k5*k4-2*ca*k6*k4
     +2*sa*sa*k5*k5+sa*sa*k4*k4+2*sa*k5*k6)-ca*k5+ca*tf.t2*k4
     -sa*k5*tf.t2-sa*k4)/(ca*k4-sa*k5)
```

onde ca e sa correspondem ao cosseno e seno respectivamente, do valor corrente de θ_2.

A solução obtida é função de θ_i, ou seja, o erro no valor de θ_2 é função de quão próximo o "chute" inicial (θ_i) está da solução. Para contornar este problema, assume-se como $\theta_i = \theta_{2corrente}$ e executa-se um procedimento iterativo:

```
ENQUANTO ( (θi - θ2calculado) > ERROMINIMO ) FAÇA
  {
   θi = θ2 calculado na iteração anterior
   calcular θ2
  }
```

Em média com 10 iterações o algoritmo converge para um valor de θ_2 com erro na quarta casa decimal.

A escolha por este procedimento e não pela fórmula algébrica deve-se ao fato de que a solução ficou muito "sensível" ao valor do seno de θ_3, principalmente para $s_3 \cong 0$ e $s_3 \cong 1$, o que não ocorre para o cálculo de θ_1, pois, o valor à esquerda da equação é constante: $-d_1$.

***** CÁLCULO PARA θ_4

Da igualdade $({}^1A_2)^{-1} * ({}^0A_1)^{-1} * T_6 = {}^2A_6$ considerando as relações:

linha 1 - coluna 3: $c_1 * c_2 * KY - s_1 * c_2 * KX + s_2 * KZ = c_5 * c_{34}$

linha 2 - coluna 3: $-c_1 * s_2 * KY + s_1 * s_2 * KX + c_2 * KZ = c_5 * s_{34}$

obtém-se

$$\theta_4 = \text{atan2}\left(\frac{-s_2 * K2 + c_2 * KZ}{c_2 * K2 + s_2 * KZ} \right) - \theta_3$$

O MGI foi desenvolvido não considerando a presença de objetos e/ou ferramentas acopladas à garra, pois, as orientações são as mesmas e, a sua posição pode ser removida do MGD sem prejuízo e simplificando os cálculos. Assim, o MGI foi calculado a partir da matriz T6 e não sobre $T6_{obj}$. Esta abordagem permite ao usuário definir seu próprio conceito de objeto independentemente da estrutura do robô.

A escolha pelo método de PAUL deve-se ao fato de se utilizar fórmulas algébricas. Contudo, notou-se que o equacionamento obtido reduz a área de trabalho devido a degenerações numéricas, principalmente na fronteira geométrica da área de trabalho. As equações obtidas incorporam um alto grau de acoplamento angular e são não lineares devido, principalmente, às funções trigonométricas, o que dificulta a análise matemática dos "polos e zeros" nas fórmulas algébricas desenvolvidas. Com a metodologia adotada, esse tipo de problema sempre ocorrerá. Uma solução é efetuar uma análise profunda dos problemas numéricos das funções obtidas e adotar soluções de software para as regiões críticas. O método numérico utilizando o jacobiano, proposto por PAUL [PAUL-83] e CRAIG [CAIG-86] [ORIN-84] [NAGY-87] [KLEIN-83] reduz este problema, no entanto, exige maior consumo de CPU, sendo crítico em trajetórias longas, onde o número de interpolações é grande (mais de 500 pontos), mesmo para linguagens off-line.

GERADOR DE TRAJETÓRIA

Em programação off-line, os movimentos do robô são, normalmente, planejados considerando-se uma configuração inicial e uma final (desejada - destino), a partir da qual determinam-se os caminhos a serem seguidos, que podem ser infinitos se o robô for redundante. A determinação dos caminhos possíveis ou do melhor caminho não são objetivos deste item [FROMMHE-87] [KUNG-89] [LIÉGEOI-87] [NAGATO-73] [NAKAMUR-87]. O interesse é desenvolver o formalismo para definir e descrever a trajetória de um movimento [ANGELES-88], [BANHABI-88], [DLABKA-89], [GOLDENB-87], [KIM-85], [LIN-83a], [RED-87],[SAKAUE-87], [TAN-89] e [TAYLOR-79].

Uma trajetória consiste de uma sequência de configurações que o robô deve assumir em movimento, podendo ser definida no espaço cartesiano ou no espaço das juntas, selecionado em função da aplicação e recursos de programação. O movimento da garra em linha reta, por exemplo, deve ser implementado no espaço cartesiano, pois, apresenta o modelamento matemático adequado. Contudo, para o sistema, a trajetória deve ser especificada no espaço das juntas, pois, é neste que o hardware-software de controle atua. O vínculo entre os dois espaços é dado pelos modelos geométricos direto e inverso como definido anteriormente.

A figura F17 mostra a estrutura de um gerador de trajetória. A trajetória é gerada a partir de um modelo matemático, normalmente, assumido como um ou um conjunto de polinômios. O polinômio tem sido amplamente utilizado, principalmente pela facilidade de implementação em computador e em sistemas de controle (em baixo nível), contudo, outros modelos são adotados, por exemplo, função cossenoidal, onde a trajetória desenvolvida é uma ou uma combinação de cossenóides. Os polinômios utilizados são de baixa ordem, não superior a quinta, evitando um caminho muito ondulado, pois, um polinômio de alta ordem tende a produzir um perfil com oscilações. A experiência mostra que o polinômio de terceira ordem apresenta bom desempenho gerando uma trajetória suave e pequenos "overshoots" a nível de controle de deslocamento das juntas [FU-87]. O polinômio pode ser aplicado diretamente sobre as configurações inicial e final, atuando como uma função geradora, ou então, aplicado sobre um conjunto reduzido de configurações da trajetória completa, atuando como um polinômio interpolador.

A combinação de polinômios faz-se necessária devido ao desempenho físico dos atuadores e do robô. Um motor não inicia o movimento com velocidade máxima e, também, não finaliza abruptamente. Por outro lado, a estrutura do robô não deve estar sujeita a variações repentinas de velocidade e aceleração, o que provoca entre outras consequências, danos ao hardware eletro-mecânico, erro de posicionamento e instabilidade no manuseio de objetos. Assim, utiliza-se um conjunto de polinômios produzindo um modelo adequado, com taxas, de velocidade e

aceleração elevando-se gradualmente no início do movimento e reduzindo-se nas proximidades da configuração desejada.

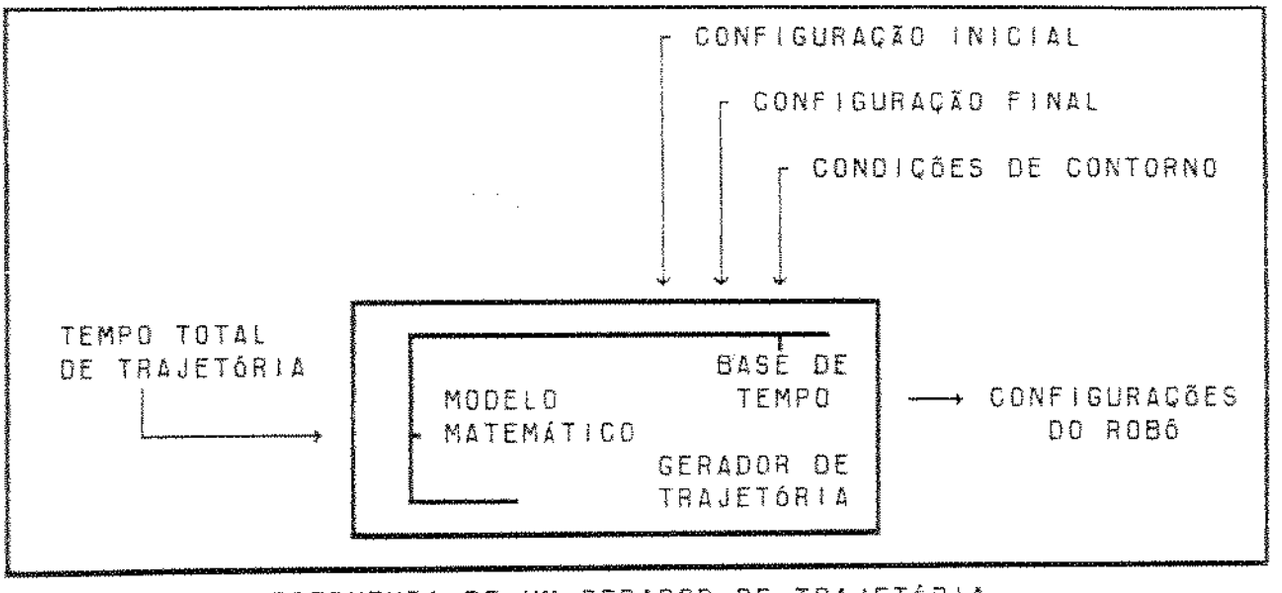

FIGURA F17 - ESTRUTURA DE UM GERADOR DE TRAJETÓRIA

As composições, de polinômios, mais utilizadas são:

- trajetória 2-1-2: A trajetória é dividida em três seguimentos, sendo: primeiro seguimento um polinômio do segundo grau, o segundo seguimento um polinômio do primeiro grau e o terceiro seguimento um polinômio do segundo grau,
- trajetória 3-5-3: A trajetória é dividida em três seguimentos, sendo: primeiro seguimento um polinômio do terceiro grau, o segundo seguimento um polinômio do quinto grau e o terceiro seguimento um polinômio do terceiro grau,
- trajetória 4-3-4: A trajetória é dividida em três seguimentos, sendo: primeiro seguimento um polinômio do quarto grau, o segundo seguimento um polinômio do terceiro grau e o terceiro seguimento um polinômio do quarto grau,
- trajetória 3-3-3-3-3: A trajetória é dividida em cinco seguimentos formados por polinómio do terceiro grau.

Cada composição apresenta suas vantagens em função da aplicação, cujo desenvolvimento matemático pode ser encontrado em Paul [PAUL-83], Craig [CRAIG-86], Fu [FU-87], Lin [LIN-83b] e Luh [LUH-84].

O uso de splines no cálculo de trajetória, também, é amplamente difundido, produzindo-se suaves perfis de velocidade e, em cálculo de trajetória on-line, permite antecipar duas ou três configurações [CHANG-88].

Para a seleção e modelagem do tipo de trajetória, deve-se considerar os seguintes pontos:

- o tempo total de trajetória baliza os valores máximos de velocidade e aceleração desenvolvidos pelos atuadores,
- as configurações corrente e de destino devem ser avaliadas, pois, tem influência direta no torque exigido uma vez que variam diretamente com a inércia da carga,
- a "distância" a ser percorrida especifica o nível de discretização e o perfil adotado,
- o perfil de jerk [KYRIAKO-89] deve ser considerado, pois, pode conduzir a danos ao hardware mecânico, ao objeto manipulado e em caso de operações que envolva contato físico, pode-se danificar a peça contato,
- o erro de trajetória deve ser considerado, pois, o método adotado especifica o erro teórico de trajetória,
- deve-se minimizar o gasto de energia e o tempo de trajetória.

Na prática, todas estas considerações estão relacionadas. O perfil de deslocamento influencia o perfil de velocidade, que influencia o perfil de aceleração e este, o perfil de jerk, e todos tem parcela de contribuição no valor de torque, bem como a geometria da estrutura e configuração determinam à inércia atuando nos motores. Como resultado tem-se desgaste prematuro do hardware mecânico, erro de posicionamento, desvio de trajetória e instabilidade durante o movimento.

Um gerador de trajetória inteligente deve conter um "mapa de conhecimento" de desempenho do robô em função destas considerações, atuando na seleção dos parâmetros adequados.

O gerador de trajetória implementado na LP é um sub-conjunto do que foi apresentado. Implantou-se dois métodos:

- trajetória 2-1-2: método tradicional com perfil de velocidade trapezoidal utilizado em controle de motores,
- trajetória cossenoidal: a trajetória segue um perfil senoidal de velocidade.

A escolha pelo primeiro método deve-se a facilidade de implementação, apresentar um perfil de deslocamento suficientemente amortecido e permitir o direto controle de desvio (erro) de trajetória podendo "simular" o deslocamento em linha reta. Note que forçar a garra a desenvolver um perfil de deslocamento linear é "impossível", pois, existe um período de aceleração e desaceleração que provocam desvios de trajetória, bem como, os desvios provocados pela realimentação em controle em malha fechada.

O perfil senoidal de velocidade foi implementado por produzir um perfil de torque sem pontos de descontinuidade o que não ocorre com o perfil trapezoidal.

Não se efetou nenhum estudo sobre erro de trajetória comparando um caminho pré-estabelecido (por exemplo uma

linha reta) é o desenvolvido pelos dois métodos. Também não implementou-se nenhum critério de avaliação e validação de trajetória. A evolução geométrica, cinemática e dinâmica teórica dos movimentos é apresentada no sistema de CAD-Simulação. A arquitetura da rotina que gera os perfis é aberta e modular, permitindo ao usuário implementar novos métodos de trajetória. O modelo dinâmico não foi utilizado na geração de trajetória, pois a abordagem foi puramente geométrica. Se o modelo do sistema de controle fosse implementado, então, o modelo dinâmico deveria ser considerado.

O gerador de trajetória (GT) do sistema consiste de uma rotina que gera uma trajetória seguindo um perfil de velocidades, ou seja, "interpolar pontos" entre dois conhecidos, a partir de critérios pré-estabelecidos. Assim, o GT permite gerar trajetórias no espaço cartesiano, espaço de juntas e no espaço de orientação (dadas duas matrizes de orientação (R) conhecidas, o GT interpola matrizes (R), elemento por elemento, tomando como parâmetros os valores das matrizes (R) inicial e final), com parametrização em função de um parâmetro t, no caso, o tempo. O usuário tem acesso ao GT através da rotina interna TRACO(), onde estão implementados os geradores de trajetória: trapezoidal e senoidal.

O gerador de trajetória trapezoidal gera uma sequência de "pontos" seguindo um perfil trapezoidal de velocidade, produzindo, no espaço interpolado, um trecho parabólico seguido de um linear e finalizando por outro parabólico, como mostrado na figura F18.

O equacionamento utilizado é o seguinte:

assumindo

TF: tempo total de trajetória
NP: número de pontos interpolados NP >0
DT: incremento de tempo DT = TF /NP
a : percentual de tempo de aceleração
TK: tempo de aceleração TK = a * TF
PO: ponto inicial para discretização
PF: ponto final para discretização
VM: velocidade máxima
AM: aceleração máxima
i : número da discretização 0 ≤ i ≤ NP

perfil de velocidade

TRECHO I: ACELERADO

$V_I(i*DT) = AM * (i*DT)$ $\qquad 0 \leq i \leq a*NP$

TRECHO II: VELOCIDADE CONSTANTE

$V_{II}(i*DT) = VM$ $\qquad a*NP \leq i \leq (1-a)*NP$

TRECHO III: ACELERADO

$V_{III}(i*DT) = AM * TF - AM * (i*DT)$
$\qquad (1-a)*NP \leq i \leq NP$

derivando as expressões em função do parâmetro (ι*DT) e calculando a aceleração:

TRECHO I: ACELERADO

$$A_I(\iota * DT) = AM \qquad 0 \leq \iota \leq a*NP$$

TRECHO II: VELOCIDADE CONSTANTE

$$A_{II}(\iota * DT) = 0 \qquad a*NP \leq \iota \leq (1-a)*NP$$

TRECHO III: ACELERADO

$$A_{III}(\iota * DT) = -AM \qquad (1-a)*NP \leq \iota \leq NP$$

integrando as equações de velocidade em função do parâmetro (ι*DT) e calculando o deslocamento:

TRECHO I: ACELERADO

$$S_I(\iota * DT) = AM * (\iota * DT)^2/2 + PO \qquad 0 \leq \iota \leq a*NP$$

TRECHO II: VELOCIDADE CONSTANTE

$$S_{II}(\iota * DT) = VM * (\iota * DT) + PK - VM * TK$$

$$a*NP \leq \iota \leq (1-a)*NP$$

TRECHO III: ACELERADO

$$S_{III}(\iota * DT) = AM * TF * (\iota * DT) - AM * (\iota * DT)^2/2 + PF - AM * TF^2/2$$

$$(1-a)*NP \leq \iota \leq NP$$

onde

$$VM = \frac{(PF - 2 * PK + PO)}{(TF - 2 * TK)}$$

$$AM = \frac{2 * (PK - PO)}{TK * TK}$$

O valor do parâmetro a deve estar entre: $0 < a \leq 0.5$. Se $a = 0.5$, o trecho de velocidade constante será nulo, correspondendo a um perfil triangular de velocidade.

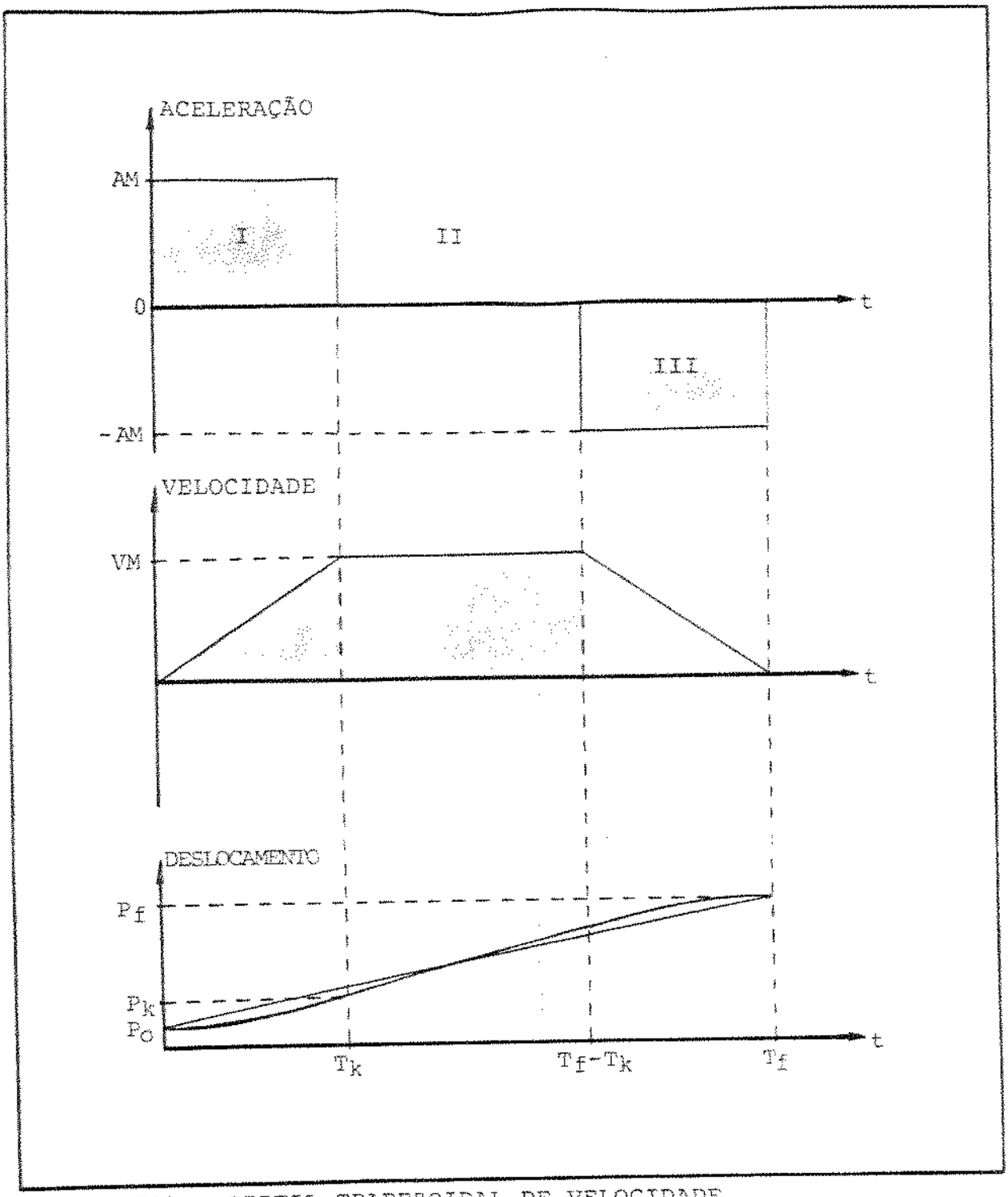

FIGURA F18 - PERFIL TRAPEZOIDAL DE VELOCIDADE

De forma análoga, o gerador de trajetória cosenoidal gera uma sequência de "pontos" seguindo um perfil senoidal de velocidade, produzindo no espaço interpolado, um trecho cossenoidal, como mostrado na figura F19:

perfil de velocidade

$$V(i*DT) = VM * \sin (i * DT * PI / TF)$$

$$0 \leq i \leq NP$$

derivando a expressão em função do parâmetro (i*DT) e calculando a aceleração:

$$Ax(i*DT) = AM * \cos (i * DT * PI / TF)$$

$$0 \leq i \leq NP$$

onde

$$AM = \frac{PI * VM}{TF}$$

integrando a equação de velocidade em função do parâmetro (i*DT) e calculando o deslocamento:

$$Sx(i*DT) = - \frac{TF * VM}{PI} * \cos (i * DT * PI / TF) + PO$$

$$0 \leq i \leq NP$$

onde

$$VM = \frac{PF - PO}{TF} * PI$$

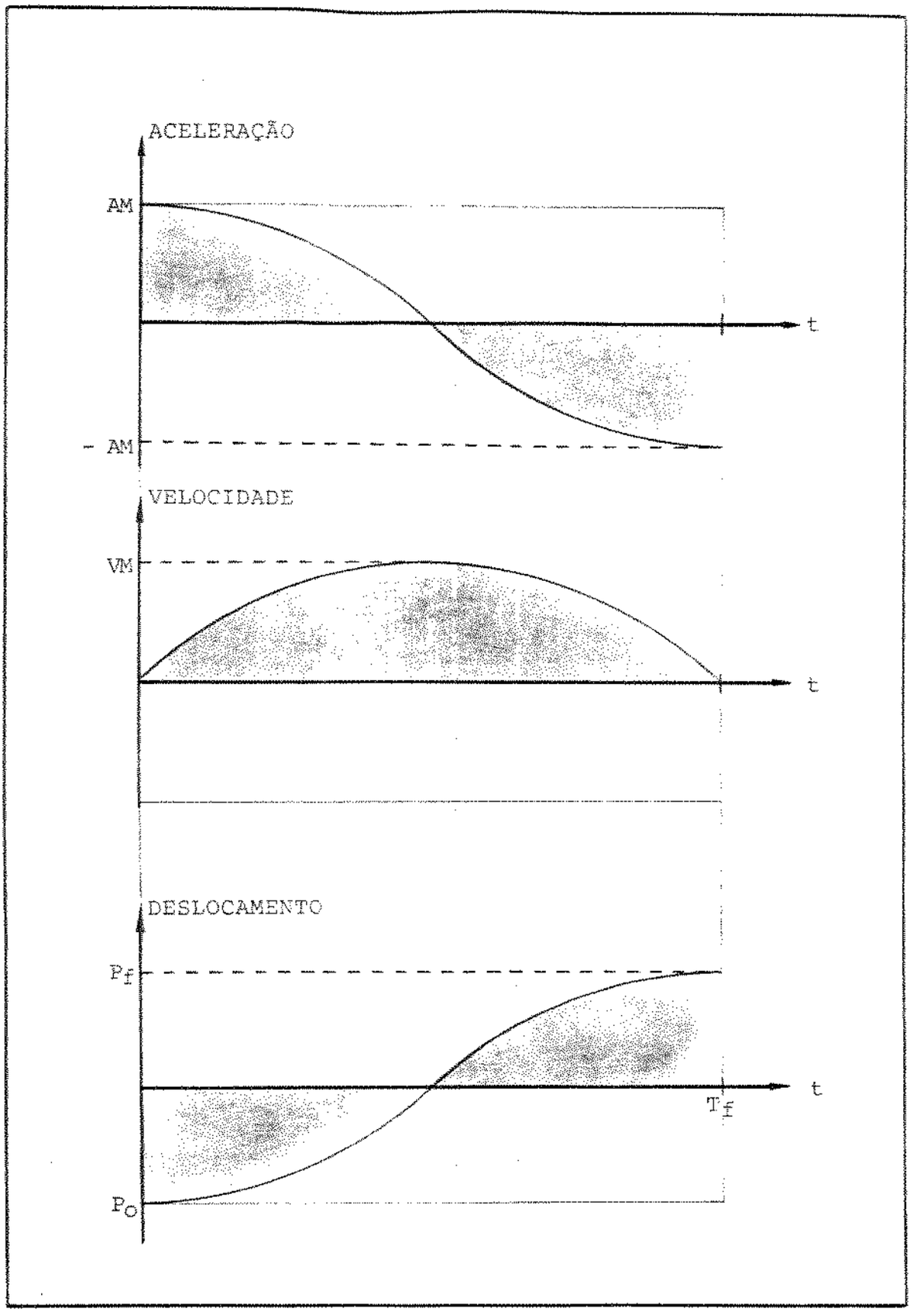

FIGURA F19 -PERFIL SENOIDAL DE VELOCIDADE

GERENCIADOR DE MOVIMENTO

O gerenciador de movimento (GM) tem a função de gerenciar o movimento das juntas do robô, sendo acessado pelos comandos de movimento da linguagem. Suas atribuições são:

- especificar os parâmetro de trajetória:
  - tempo total de trajetória
  - velocidade máxima
  - discretização
  - nível de carga
  - tipo de trajetória

- coordenar a geração dos movimentos:
  - livre
  - linear
  - linear com orientação fixa
  - ajuste de orientação
  - ajuste de posição

O GM é auxiliado pelo gerador de trajetória e por dois módulos: movimento livre - que gera os movimentos livres
movimento linear - que gera os movimentos lineares,
produzindo como resultado, os valores de referência das juntas.

IMPLEMENTAÇÃO

***** MOVIMENTO LIVRE

Os movimentos livres (comandos MOVER e MOVERANG por exemplo) do robô são gerados seguindo o seguinte algoritmo:

- consultar a TPL e obter o valor angular corrente das juntas
- obter o valor angular desejado das juntas
- verificar o intervalo para posição angular desejada
- calcular tempo total de trajetória (TF) e incremento de tempo (DT)
- chamar a rotina TRACO() e gerar o perfil de movimento para as juntas
- atualizar a TPL:
  - posição-orientação corrente
  - número de pontos interpolados
- atualizar arquivo geral de trajetória com os sinais de referência gerados

***** MOVIMENTO LINEAR

O movimento linear do robô é gerado seguindo o seguinte algoritmo:

- consultar a TPL e obter a posição-orientação corrente da garra
- montar a matriz Ac com a posição-orientação corrente
- obter a posição-orientação desejada
- montar a matriz Ad com a posição-orientação desejada
- verificar se existe alguma junta travada,
  se sim => ERRO (não realizar movimento linear com junta travada)
- calcular tempo total (TF) de trajetória considerando o deslocamento no espaço cartesiano e o valor da garra
- calcular o incremento de tempo (DT)
- chamar a rotina TRACO() e gerar o movimento para cada elemento da matriz A, tomando como entrada o valor dos elementos das matrizes Ac e Ad. Por exemplo, para o elemento nx, que na posição corrente assume o valor nxc e na posição desejada assume o valor nxd. A rotina TRACO() deve ser executada tomando como entrada os valores nxc e nxd. Procedendo de forma análoga para os elementos da matriz A, obtém-se como resultado um conjunto de matrizes contendo a posição-orientação intermediária da trajetória. Como a evolução do vetor posição P<PX,PY,PZ> segue um perfil linear (se o tipo de movimento setado for o TRAPEZOIDAL), a trajetória descrita é linear com ajuste de orientação. Se o perfil SENOIDAL estiver ativo o procedimento é idêntico, no entanto, a trajetória será cossenoidal.
- para cada matriz A intermediária calcular o modelo geométrico inverso obtendo o valor angular das juntas
- atualizar a TPL:
  - posição-orientação corrente
  - número de pontos interpolados
- atualizar arquivo geral de trajetória com os sinais de referência gerados

***** MOVIMENTO COM ORIENTAÇÃO FIXA

Os movimentos com orientação fixa do robô são gerados seguindo o seguinte algoritmo:

- especificar a posição desejada: vetor Pd<PXd,PYd,PZd>
- consultar a TPL e obter a orientação fixa
- montar a matriz A com a posição-orientação desejada
- executar algoritmo para MOVIMENTO LINEAR

***** MOVIMENTO COM AJUSTE DE ORIENTAÇÃO

Os movimentos com ajuste de orientação são gerados seguindo o seguinte algoritmo:

- especificar a orientação desejada: matriz Rd
- consultar a TPL e verificar: se flag de posição fixa está setada faça:
  - consultar a TPL e obter a posição corrente, senão assumir posição especificada
- montar a matriz A com a posição-orientação desejada
- atualizar TPL setando a flag de orientação fixa e a orientação desejada
- executar algoritmo para MOVIMENTO LINEAR

***** MOVIMENTO COM AJUSTE DE POSIÇÃO

O ajuste da posição cartesiana do robô segue o seguinte algoritmo:

- consultar a TPL e obter a orientação corrente: matriz Rc
- especificar a posição desejada
- especificar a orientação desejada: matriz Rd
- montar a matriz A com a posição-orientação desejada
- executar algoritmo para MOVIMENTO LINEAR ou LIVRE, consultando a flag de tipo de movimento

***** CÁLCULO DO TEMPO TOTAL DE TRAJETÓRIA E INCREMENTO DE TEMPO

O cálculo do tempo total de trajetória e do incremento de tempo obedece o princípio de que todas as juntas devem iniciar e terminar um movimento simultaneamente, a menos que a junta esteja travada. Os cálculos são realizados a partir do valor da máxima velocidade linear no espaço das juntas ou cartesiano, da precisão e do número máximo de pontos de discretização, especificados na TPL e em função do tipo de movimento. O usuário pode setar o valor da velocidade máxima através do comando SETVELOMAX(), especificando a velocidade máxima no espaço das juntas, da garra no espaço cartesiano ou de abertura da garra. A precisão pode ser setada através do comando SETPRECI(), especificando a precisão do movimento das juntas, da garra no espaço cartesiano ou de abertura da garra.

O cálculo do tempo total de trajetória é função do perfil de trajetória assumido:

- PERFIL TRAPEZOIDAL

consultando a TPL, a velocidade máxima (VM) é especificada por:

$$VM = VM_{TPL} * NÍVEL\ DE\ CARGA$$

onde VM_{TPL}: velocidade máxima especificada na TPL

assim

$$TF = \frac{(PF - PO)}{VM}$$

onde TF: tempo total de trajetória,
PO: posição inicial,
PF: posição final.

- PERFIL SENOIDAL

consultando a TPL, a velocidade máxima (VM) é especificada por:

$$VM = VM_{TPL} * NÍVEL\ DE\ CARGA$$

onde VM_{TPL}: velocidade máxima especificada na TPL

para o perfil senoidal, a velocidade máxima é dada por:

$$Vmax = \frac{(PF - PO) * PI}{2*TF}$$

onde PI: constante matemática,

igualando Vmax = VM

temos

$$TF = \frac{(PF - PO) * PI}{2*VM}$$

Conhecido o valor do tempo total da trajetória, o cálculo do incremento de tempo é dado por:

$$DT = \frac{TF}{NP}$$

onde

$$NP = 10 + (MAXPT - 10) * \frac{(PF - PO)}{\text{max. intervalo}(PF - PO)}$$

fator de correção normalizado

se (PF - PO)/NP > precisão então assumir NP = MAXPT e se a resolução não for suficiente ABORTAR

NP: número de pontos de discretização
MAXPT: número máximo de pontos de discretização assumido pelo sistema

O valor de NP é função do número máximo de pontos de discretização assumido pelo sistema multiplicado por um fator de correção normalizado, de tal forma que trajetórias curtas contenham menor número de pontos de discretização que outra com maior intervalo. O "max intervalo" consiste na maior variação numérica da variável discretizada. Se a precisão for muito apertada, o sistema assume o número máximo de pontos de discretização do sistema NP = MAXPT, se não for suficiente para obter a precisão desejada o processamento é abortado. Note que se o deslocamento for desprezível, o sistema assume um mínimo de 10 pontos de discretização. Este valor mínimo deve ser reduzido ou aumentado em função da resolução dos encoders no caso real.

PARTE C

C.A.D.-SIMULAÇÃO

INTRODUÇÃO

Na parte C apresentar-se-á o sistema de CAD-Simulação, que agrupa 3 módulos: CAD-animação onde os movimentos do robô são visualizados em animação gráfica; TEACH-IN onde o usuário pode conduzir o robô para configurações deseladas ; e Modelamento, onde é possível visualizar, em gráficos, a evolução cinemática e dinâmica dos três primeiros links do robô.

C.A.D. - SIMULAÇÃO

O sistema de CAD-simulação tem a função de permitir ao usuário a análise, a partir de gráficos e animação gráfica, de uma tarefa implementada na linguagem de programação (LP) [FALLSID-89] [KOPACEK-88] [LANGRUD-87] [LATOMBE-84] [MYERS-84] [MILBERG-88] [WECK-87] [WOZNIAK-89] [WU-84]. O sistema está dividido em três módulos, como mostrado na figura F20. Assim, o usuário pode visualizar em animação gráfica os movimentos do robô, garra e objetos, verificando a evolução geométrica, cinemática da garra e dinâmica dos links e, gerar uma sequência de configurações através do módulo de teach-in. A integração com a linguagem de programação realiza-se por meio de um arquivo de dados: CAD.DAT, gerado durante a execução de uma tarefa pela subrotina TRANSFER() (ver nível 2 da LP).

O acesso ao sistema ocorre através do software ANI. Um menu principal é apresentado permitindo a seleção das opções:

- CAD-animação
- teach-in
- editar
- modelamento
- abandonar

O menu principal também tem a função de inicializar os parâmetros do sistema com os valores default.

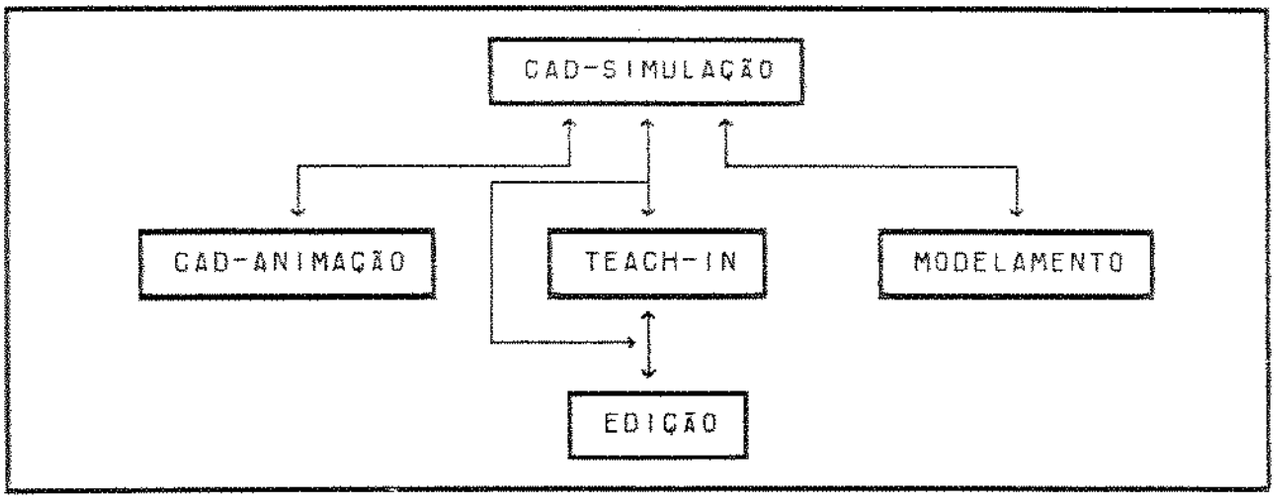

FIGURA F20 - ESTRUTURA DE MÓDULOS DO SISTEMA DE CAD-SIMULAÇÃO

C.A.D. - ANIMAÇÃO

O módulo de CAD-animação [CORITES-81] [WEINSTO-87] é o responsável pelo gerenciamento da animação gráfica [MORGAN-84] de uma tarefa, permitindo a visualização dos movimentos dos links e garra do robô e dos objetos. Os elementos que compõe a área de trabalho do sistema: robô, garra e objetos: são apresentados em um ambiente bi e/ou tri-dimensional [NNAJI-86], sendo observados por um observador que pode "passear" pela área de trabalho alterando seu ângulo de visão. A implementação deste módulo está concretizada na rotina CADANIM() com o auxílio das rotinas auxiliares, estando dividida em duas partes:

- leitura de dados
- animação gráfica

A leitura de dados é realizada tomando como entrada os dados contidos no arquivo cad.dat:

- posição angular das juntas,
- dados dos objetos,
- dados da garra ativa e
- parâmetros de controle,

montando uma estrutura tipo ponto (consultar base de dados) e atualizando a base de dados. A animação gráfica ocorre com a execução da subrotina de visualização. A figura F21 mostra o fluxograma de execução deste módulo. Este processo de leitura e animação gráfica continua até que o usuário interrompa a execução ou até o fim dos dados no arquivo.

Com a execução da subrotina de visualização o usuário tem acesso aos seus comandos internos, de tal forma, que poderá:

- ativar o editor de pontos do modo teach-in
- setar uma nova posição-orientação da garra, que será resetada com o próximo acesso ao arquivo
- ativar-desativar o modo passo a passo
- alterar o ângulo de observação
- setar tipo de apresentação do robô: esqueleto ou sólido
- setar plotagem tri-dimensional ou projeções

Com o interrupção ou finalização da execução da animação, o controle do sistema é devolvido ao menu principal, ver figura F22, mantendo setada a última configuração da área de trabalho, ou seja, a última posição angular das juntas, posição-orientação da garra e dos objetos.

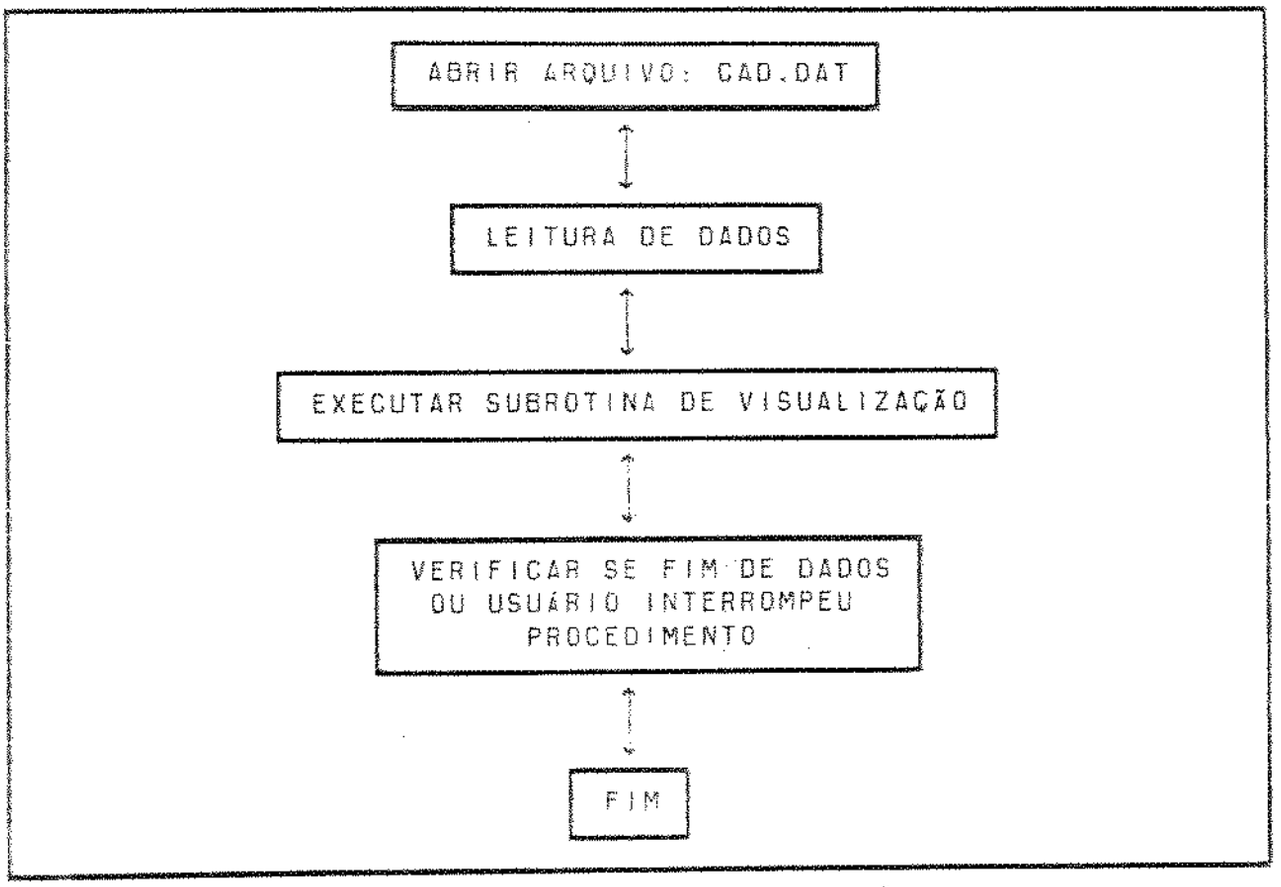

FIGURA F21 - FLUXOGRAMA DO MÓDULO CAD-animação

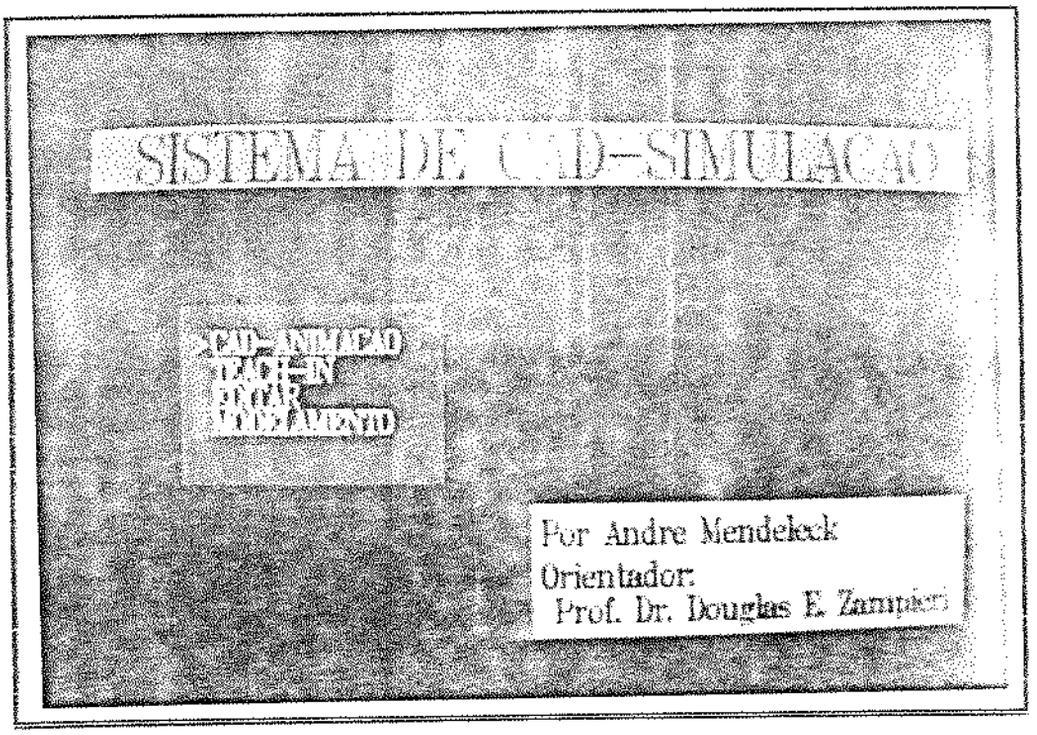

FIGURA F22 - MENU PRINCIPAL

TEACH-IN

O módulo teach-in [CRAIG-86] [GRAHAM-87] [LOZANO-83] permite que o robô seja conduzido pelo usuário a uma posição e orientação no espaço cartesiano, memorizando-a para uso durante a programação de uma tarefa. Normalmente, uma teach-in box (caixa de ensino) é acoplada ao hardware eletrônico, de tal forma que, por meio de teclado, joystick, mouse e/ou comando por voz seta-se uma sequência de pontos compondo a trajetória que deve ser executada pelo robô, sendo referenciadas a posição-orientação, velocidade e aceleração das juntas, força-torque externo na garra e ferramenta manipulada.

O teach-in implementado neste sistema faz referência somente a posição-orientação da garra, no entanto, a estrutura de software do módulo permite a incorporação de outros parâmetros, como velocidade e perfil de trajetória. A implementação é realizada através da rotina TEACHIN(), assumindo como parâmetros de entrada os dados correntes da base de dados, ou seja, é assumido como condições iniciais o estado corrente da área de trabalho. Se uma animação gráfica é interrompida e o módulo teach-in é ativado, a última posição angular das juntas, da garra e dos objetos é assumida como dado de entrada.

A figura F23 mostra o fluxograma da rotina TEACH-IN(), que consiste basicamente de um loop de execução da subrotina de visualização cujo comandos são apresentados no item ROTINAS AUXILIARES. Se o módulo é abortado o controle do sistema é transferido para o menu principal.

O módulo teach-in permite ao usuário editar os pontos setados, ou então, setar uma sequência de pontos utilizando o teclado. Nesta opção, pode-se ter acesso ao valor angular das juntas, a posição-orientação e valor de abertura da garra para os pontos setados. Uma "janela" [MICROSO-86] de tela completa é ativada setando duas páginas, ver figuras F24 e F25:

PÁGINA 1 - mostra o valor angular das juntas e o valor de abertura da garra

PÁGINA 2 - mostra o valor angular das juntas, posição-orientação e valor de abertura da garra

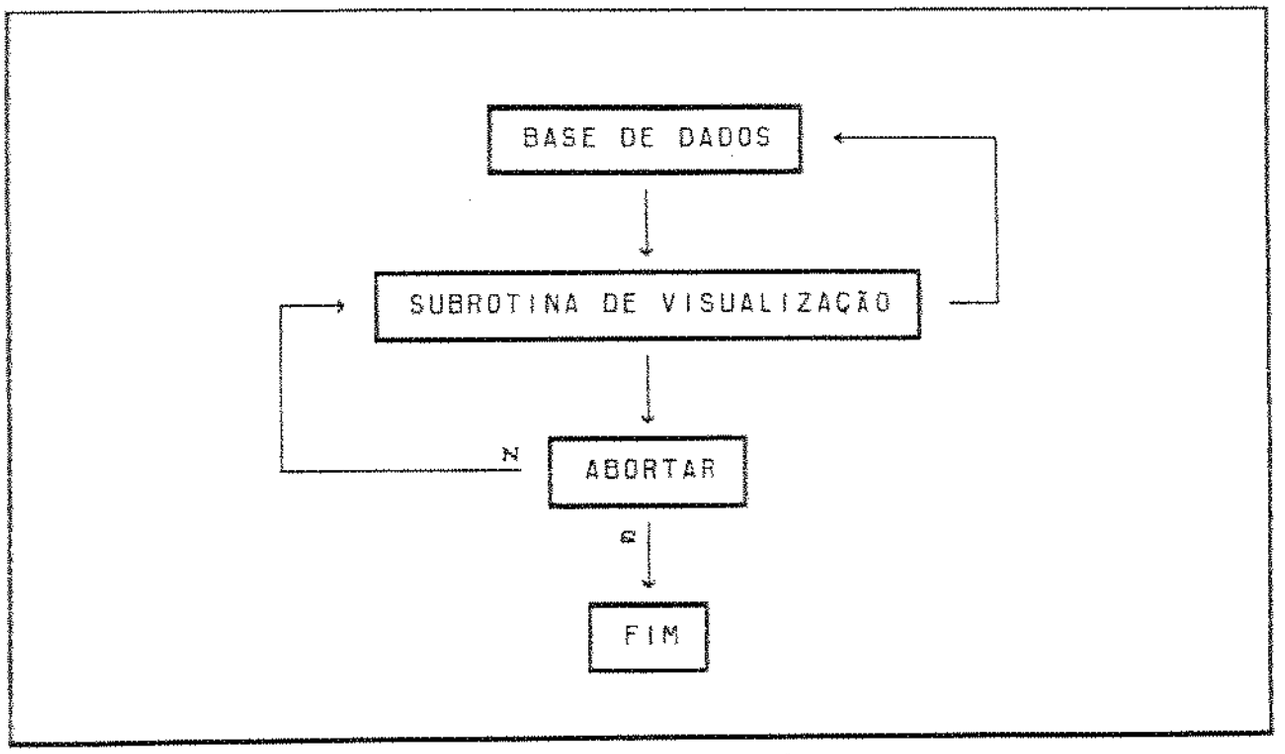

FIGURA F23 - FLUXOGRAMA DE EXECUÇÃO DO MÓDULO TEACH-IN

O usuário tem a sua disposição uma série de comandos de edição, como segue:

TECLA	
p P	ativa a página complementar. Se a página 1 estiver ativa, a página 2 será ativada e vice-versa
3	rolar uma página abaixo
9	rolar uma página acima
2	rolar um ponto abaixo
8	rolar um ponto acima
7	ativar página a partir do primeiro ponto setado
1	ativar página a partir do último ponto setado
c C	"clear": apagar todos os pontos setados

s S "save": salvar em disco os pontos setados
opções: A - salva todos os pontos
B - salva um bloco de pontos

l L "load": carregar pontos setados de um arquivo em disco
opções: L - carrega pontos, deletando os pontos na memória
F - carrega dados a partir do último ponto setados na memória
P - carrega dados a partir de um ponto ⊡ setados na memória

q Q abandonar edição

d D "deletar": apagar um ou um bloco de pontos setados, zerando o valor angular das juntas. Os pontos não são removidos da memória.

b B "bloco": ativar a opção de bloco

"espaço" setar o início e fim de um bloco

i I "inserir": setar novos pontos através do teclado a partir de um ponto especificado
opções: A - "ângulos": especificar o valor angular das juntas e o valor da garra, cabendo ao sistema o cálculo da posição-orientação da garra
P - "posição-orientação": especificar a matriz orientação, o vetor posição e o valor de abertura da garra, cabendo ao sistema o cálculo do valor angular das juntas. Considerar a presença ou não de objeto ativo.

r R "remover": remover um ponto ou um bloco de pontos da memória. A posição absoluta dos pontos a partir do ponto de fim de bloco fica alterada

g G "correção": correção de um ponto setado
opções: idênticas as do comando inserir

a A "acrescentar": acrescentar um novo ponto a partir do último setado

o O "copiar": copiar um bloco a partir de um ponto especificado

O editor é residente no sistema, estando ativo durante a execução do sub-sistema de CAD-simulação.

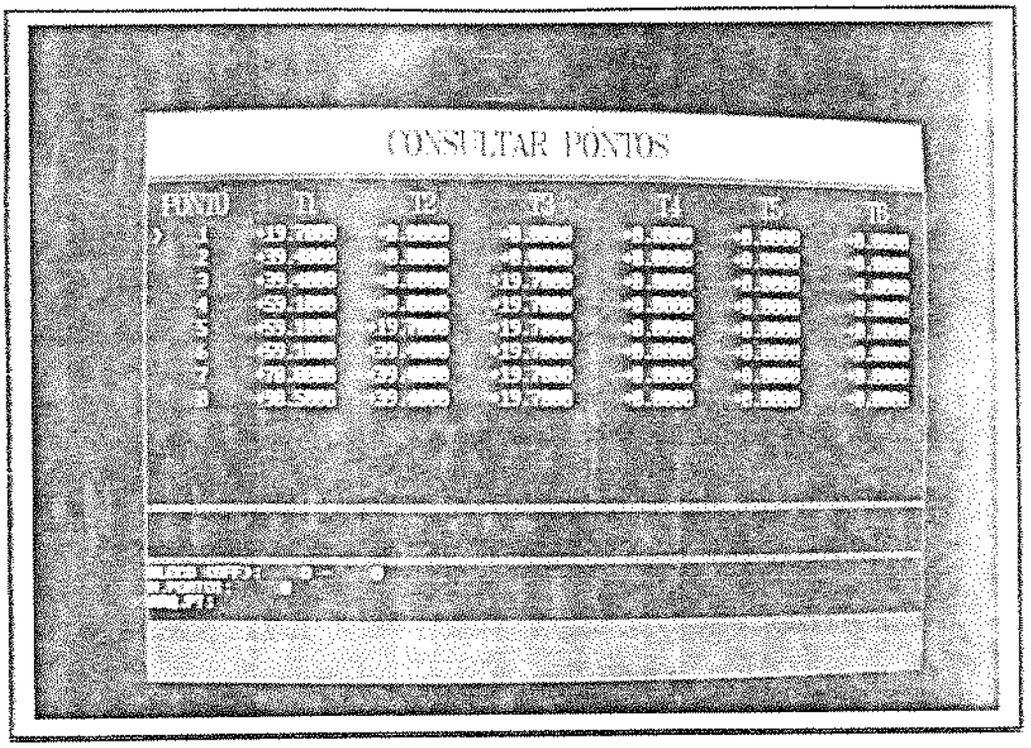

FIGURA F24 - PÁGINA 1

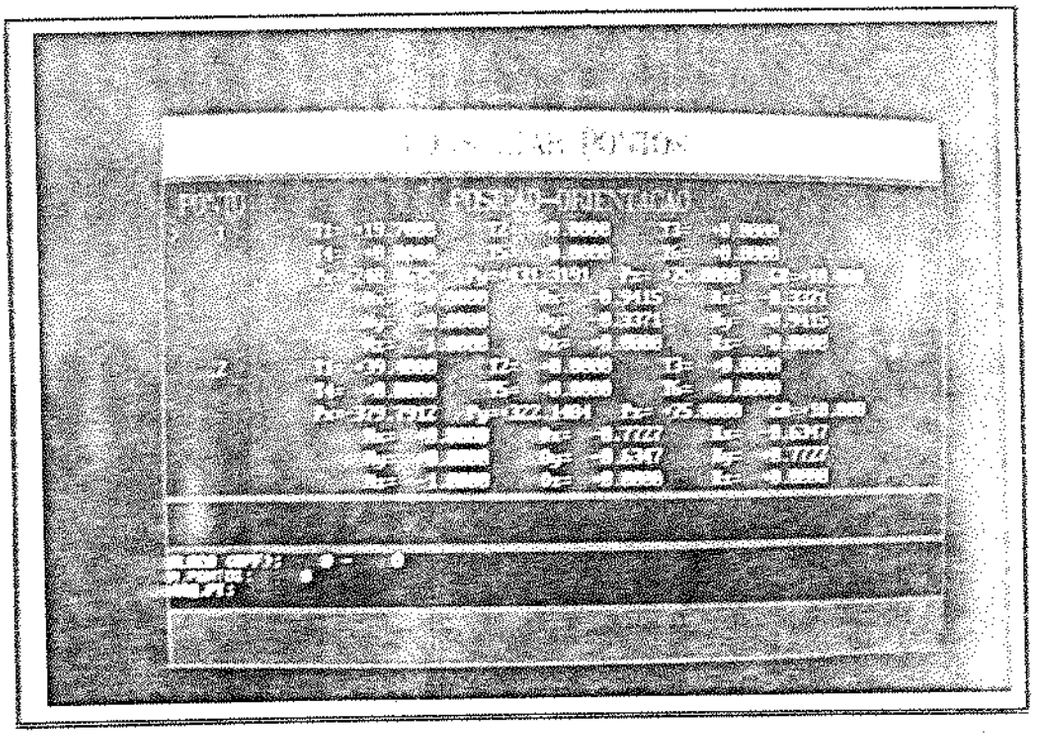

FIGURA F25 - PÁGINA 2

MODELAMENTO

O módulo de modelamento permite ao usuário acompanhar, através de gráficos bi-dimensionais a evolução da cinemática e dinâmica dos três primeiros links do robô (ver figura F26), plotando gráficos de:

- posição, velocidade e aceleração da garra no espaço cartesiano em relação ao referencial absoluto.
- posição, velocidade e aceleração das três primeiras juntas, em relação aos respectivos sistemas de referência relativos
- torque total na juntas
- torque devido a componente de gravidade nas juntas
- torque devido a componente de inércia nas juntas
- torque devido as componentes de força centrífuga e coriolis

A figura F27 mostra o fluxograma do módulo. A simulação toma como dados de entrada os valores de posição angular das juntas do arquivo CAD.DAT gerado pela linguagem de programação. Utilizando o modelo geométrico direto, calculam-se a posição cartesiana, velocidade e aceleração da garra em relação ao referencial absoluto, a seguir, calculam-se a velocidade e aceleração das juntas no referencial relativo. Os cálculos de velocidade e aceleração são realizados a partir do valor de posição do item em análise. Por exemplo, seja {P} um vetor posição, a velocidade {V} é calculada aplicando a definição de derivada:

$$V(t+\Delta t) = \lim_{\Delta t \to 0} \frac{P(t+\Delta t) - P(t)}{\Delta t}$$

considerando Δt conhecido, correspondendo ao intervalo de tempo de discretização, o cálculo da velocidade é dado por:

$$V(i+1) = \frac{P(i+1) - P(i)}{\Delta t}$$

onde i: número da discretização.

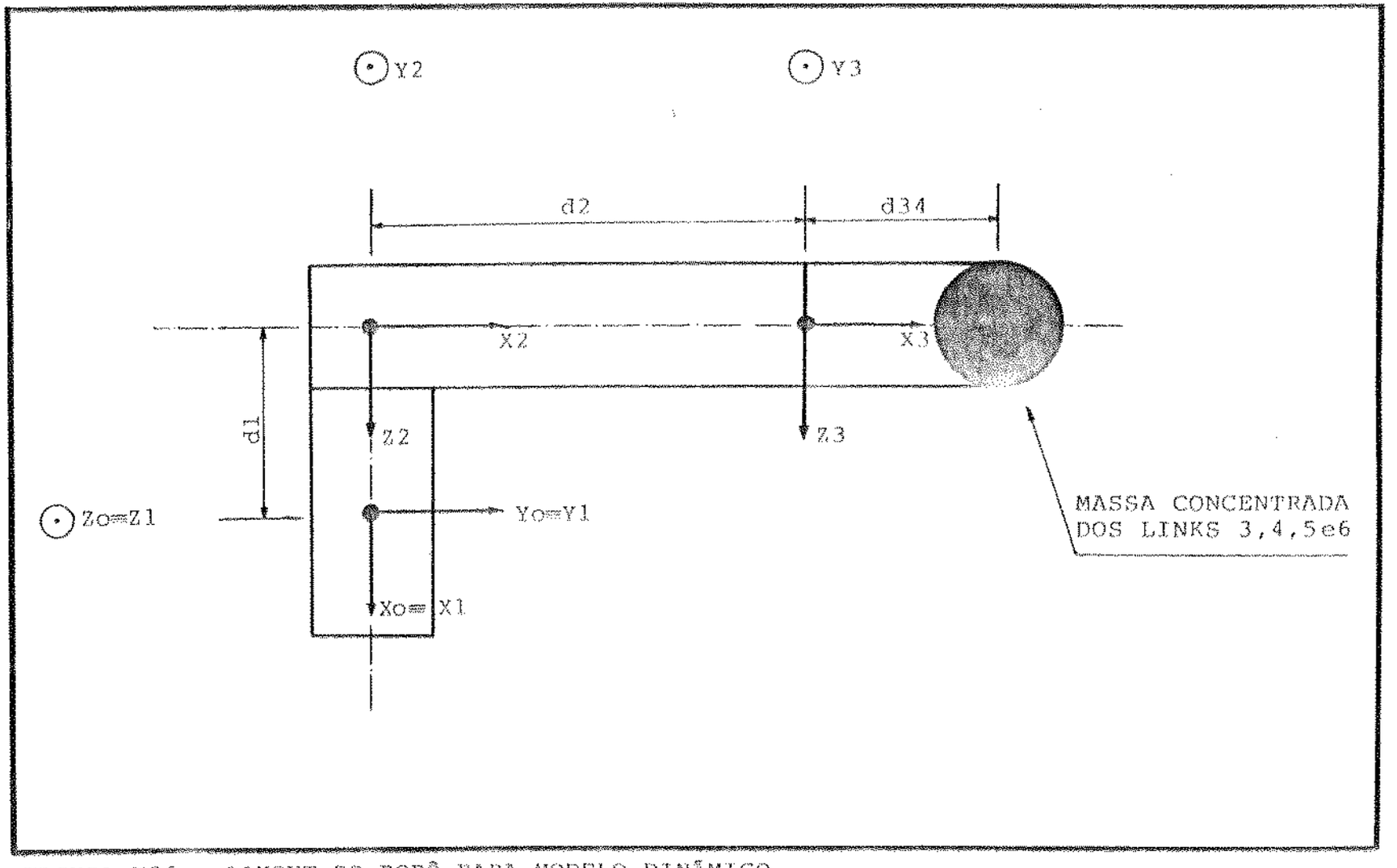

FIGURA F26 - LAYOUT DO ROBÔ PARA MODELO DINÂMICO

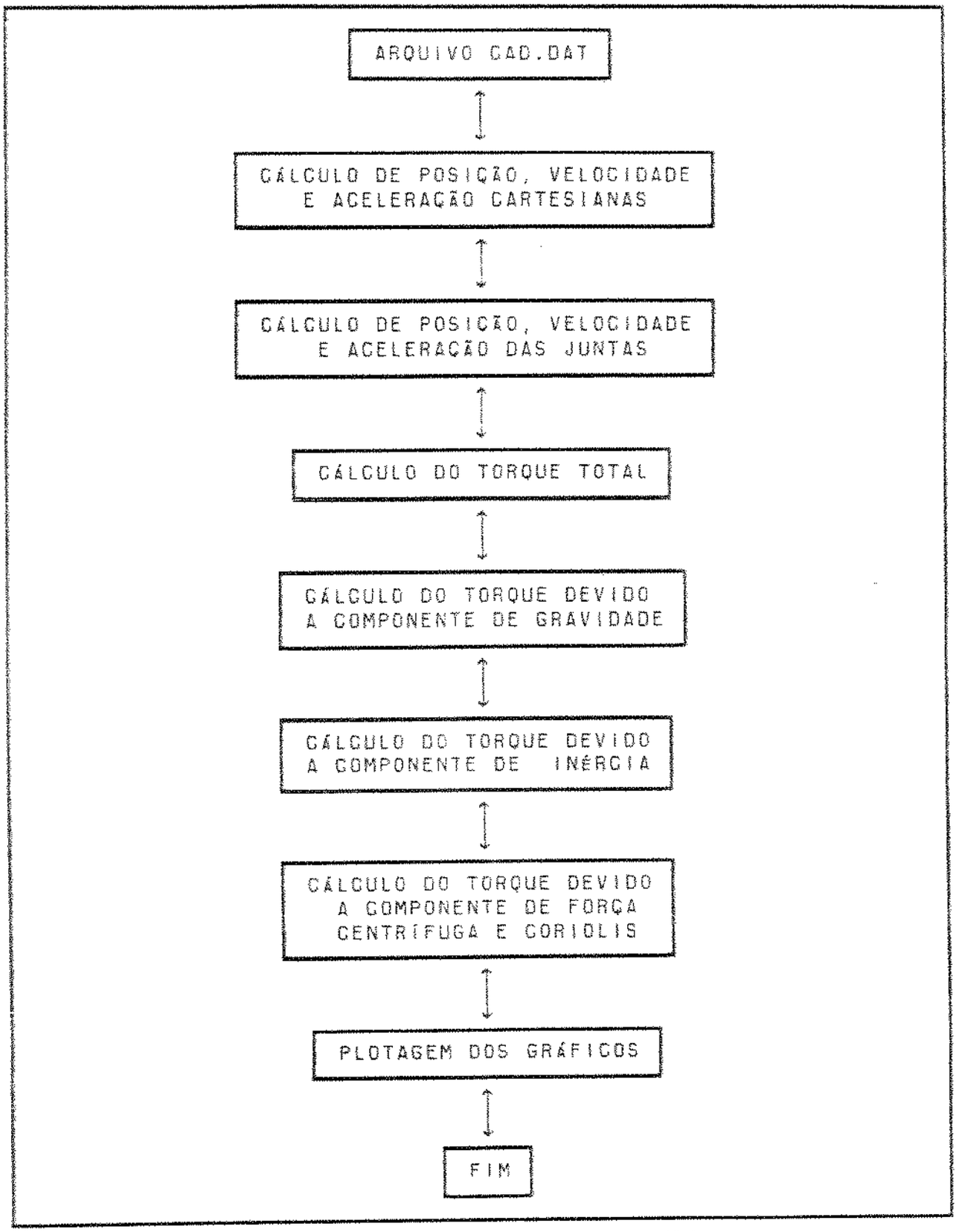

FIGURA F27 - FLUXOGRAMA DO MÓDULO DE MODELAMENTO

O cálculo da aceleração {A} é análogo, sendo aplicado sobre a velocidade:

$$A(i+1) = \frac{V(i+1) - V(i)}{\Delta t}$$

Utilizando este método, perdem-se o primeiro e último valores de velocidade e o primeiro, segundo, penúltimo e último valores de aceleração. No entanto, o sistema assume que o robô tem velocidade nula de início e fim de trajetória, de tal forma, que somente o primeiro e último valores de aceleração ficam desconhecidos; o sistema força esses valores para zero. Outro procedimento para o cálculo de velocidade e trajetória seria utilizar um polinômio interpolador, evitando a perda de informações, no entanto, não foi implementado.

O torque total é calculado a partir do modelo dinâmico, dado por uma expressão do tipo:

$$\mathfrak{T} = M(\theta) * \left[\ddot{\theta} \right] + CR(\theta) * \left[\dot{\theta} \, , \, \dot{\theta} \right] + CF(\theta) * \left[\dot{\theta}^2 \right] + G(\theta)$$

onde $\mathfrak{T}$: torque total

$\left[\ddot{\theta} \, , \, \dot{\theta} \, , \, \theta \right]$: vetores de aceleração, velocidade e posição angular das juntas

$M(\theta)$: matriz dos coeficientes da componente de inércia

$CR(\theta)$: matriz dos coeficientes da componente de força de coriolis

$CF(\theta)$: vetor dos coeficientes da componente de força centrífuga

$G(\theta)$: vetor dos coeficientes da componente de gravidade

O cálculo da componente de gravidade é efetuado a partir da equação de $\mathfrak{T}$ atribuindo aos termos de aceleração e velocidade o valor zero. Analogamente, o cálculo da componente de torque devido a inércia é efetuado a partir da equação de $\mathfrak{T}$ atribuindo o valor zero aos termos de velocidade e subtraindo a componente de gravidade

$$\mathfrak{T}_{inercia} = \mathfrak{T}(\ddot{\theta} \, , \, \dot{\theta} = 0 \, , \, \theta) - G(\theta)$$

O cálculo da componente de torque devido a força centrífuga e coriolis é efetuado a partir da equação de $\mathfrak{T}$ atribuindo o valor zero aos termos de aceleração e subtraindo a componente de gravidade:

$$\mathfrak{T}_{\text{contr. \& coriol}} = \mathfrak{T}(\ddot{\theta} = 0,\ \dot{\theta},\ \theta) - G(\theta)$$

O cálculo do modelo dinâmico para as três primeiras juntas deve-se ao fato de serem as críticas para o dimensionamento dos atuadores e de controle, por apresentarem maior massa e inércia que as três últimas. O modelo dinâmico utilizado foi o de Newton-Euler por apresentar o menor número de operações aritméticas e exigir menor esforço na implementação e de CPU que outros, como o de Lagrange-Euler [ARMSTRO-86] [DRIELS-88] [FEATHER-87] [GOOD-85] [HOLLERB-80] [LEE-82] [LEE-87] [NEUMAN-87] [SILVER-82] [VUKOBR-82a] [VUKOBR-82b] [SILVER-82]. A metodologia utilizada é a desenvolvida por Craig [CRAIG-86], como segue:

N : NÚMERO DE GRAUS DE LIBERDADE

PARA $i = 0$ ATÉ $N - 1$ FAÇA

$$^{i+1}\omega_{i+1} = {}^{i+1}_{i}R^{*}\,{}^{i}\omega_{i} + \dot{\theta}_{i+1}{}^{*}\,{}^{i+1}\hat{Z}_{i+1}$$

$$^{i+1}\dot{\omega}_{i+1} = {}^{i+1}_{i}R^{*}\,{}^{i}\dot{\omega}_{i} + {}^{i+1}_{i}R^{*}\,{}^{i}\omega_{i} \times \dot{\theta}_{i+1}{}^{*}\,{}^{i+1}\hat{Z}_{i+1} + \ddot{\theta}_{i+1}{}^{*}\,{}^{i+1}\hat{Z}_{i+1}$$

$$^{i+1}\dot{V}_{i+1} = {}^{i+1}_{i}R^{*}[\,{}^{i}\dot{\omega}_{i} \times {}^{i}P_{i+1} + {}^{i}\omega_{i} \times ({}^{i}\omega_{i} \times {}^{i}P_{i+1}) + {}^{i}\dot{V}_{i}\,]$$

$$^{i+1}\dot{V}c_{i+1} = {}^{i+1}\dot{V}_{i+1} + {}^{i+1}\dot{\omega}_{i+1} \times {}^{i+1}Pc_{i+1} + {}^{i+1}\omega_{i+1} \times ({}^{i+1}\omega_{i+1} \times {}^{i+1}Pc_{i+1})$$

$$^{i+1}F_{i+1} = m_{i+1}{}^{*}\,{}^{i+1}\dot{V}c_{i+1}$$

$$^{i+1}N_{i+1} = {}^{i+1}_{c}I_{i+1}{}^{*}\,{}^{i+1}\dot{\omega}_{i+1} + {}^{i+1}\omega_{i+1} \times {}^{i+1}_{c}I_{i+1}{}^{*}\,{}^{i+1}\omega_{i+1}$$

PARA $i = N$ ATÉ 1 FAÇA

$$ {}^{i}f_{i} = {}^{i}_{i+1}R \; {}^{i+1}f_{i+1} + {}^{i}F_{i} $$

$$ {}^{i}n_{i} = {}^{i}N_{i} + {}^{i}_{i+1}R \; {}^{i+1}n_{i+1} + {}^{i}Pc_{i} \times {}^{i}F_{i} + {}^{i}P_{i+1} \times {}^{i}_{i+1}R \; {}^{i+1}f_{i+1} $$

$$ \tau_{i} = {}^{i}n_{i}^{T} \; {}^{i}\hat{Z}_{i} $$

onde

${}^{i+1}\omega_{i+1}$: velocidade angular do link $i+1$ em relação ao sistema de referência $i+1$

${}^{i+1}\dot{\omega}_{i+1}$: aceleração angular do link $i+1$ em relação ao sistema de referência $i+1$

${}^{i+1}\dot{v}_{i+1}$: aceleração linear do link $i+1$ em relação ao sistema de referência $i+1$

${}^{i+1}\dot{vc}_{i+1}$: aceleração linear do centro de gravidade do link $i+1$ em relação ao sistema de referência $i+1$

${}^{i+1}F_{i+1}$: força atuando no centro de gravidade do link $i+1$ em relação ao sistema de referência $i+1$

${}^{i+1}N_{i+1}$: torque atuando no centro de gravidade do link $i+1$ em relação ao sistema de referência $i+1$

${}^{i}f_{i}$: força total atuando no link i em relação ao sistema de referência i

${}^{i}n_{i}$: torque total atuando no link i em relação ao sistema de referência i

τ_{i} : torque total na junta i

A escolha por essa metodologia deve-se ao fato de ser a mais "algoritmizada" da literatura e de fácil acompanhamento (debug) dos cálculos intermediários da cinemática dos movimentos.

O modelo dinâmico foi implementado em duas etapas:

I - simbolicamente
II - numericamente

Na primeira etapa, a metodologia recursiva de Craig foi implementada simbolicamente utilizando o software REDUCE produzindo as equações algébricas de torque para as três primeiras juntas, que na segunda etapa foi implementada numericamente no módulo de simulação. A utilização de um software para cálculo simbólico do equacionamento dinâmico de robôs é de fundamental importância, reduzindo o número de operações aritméticas: eliminando operações de multiplicação e soma por zero e agrupando termos comuns. O software REDUCE, apesar de apresentar eses recursos a nível de comandos, mostrou-se ineficiente nas operações de simplificação matemáticas (principalmente trigonométricas) e no gerenciamento de memória.

O resultado simbólico obtido é mostrado a seguir, donde, nota-se que um número razoável de simplificações matemáticas ficaram por serem realizadas, pois, se fosse implementado os procedimentos necessários o software REDUCE "travaria o computador". Contudo, seguramente o número de operações aritméticas é menor que uma implementação numérica.

Equacionamento do torque para as três primeiras juntas do robô, utilizando o software REDUCE:

```
/*  CALCULO DE TORQUE PARA A JUNTA:  3 */
TQ3= -c2*c2*c3*s3*IX3*v1*v1+c2*c2*c3*s3*IY3*v1*v1+c2*c2*c3*s3
     *v1*v1*cd34*cd34*m34+c2*c2*s3*v1*v1*d2*cd34*m34
     -c2*c3*c3*s2*IX3*v1*v1+c2
     *c3*c3*s2*IY3*v1*v1+c2*c3*c3*s2*v1*v1*cd34*cd34*m34
     +c2*c3*s2*v1*v1*d2*cd34*m34+c2*c3*g*cd34*m34+c2*
     s2*s3*s3*IX3*v1*v1-c2*s2*s3*s3*IY3*v1*v1
     -c2*s2*s3*s3*v1*v1*cd34*cd34*m34+c2*s3*a1*d1*cd34*m34
     +c3*s2*s2*s3*IX3*v1*v1-c3*s2*s2*s3*IY3*v1*v1
     -c3*s2*s2*s3*v1*v1*cd34*cd34*m34+c3*s2*a1*d1*cd34*m34
     +c3*a2*d2*cd34*m34-s2*s3*g*cd34*m34+s3*v2*v2*d2
     *cd34*m34+IZ3*a2+IZ3*a3+a2*cd34*cd34*m34+a3*cd34*cd34*m34
```

```
/* CALCULO DE TORQUE PARA A JUNTA:  2 */
TQ2= -c2*c2*c3*s3*IX3*v1*v1+c2*c2*c3*s3*IY3*v1*v1+c2*c2*c3*s3
     *v1*v1*cd34*cd34*m34+c2*c2*s3*v1*v1*d2*cd34*m34
     +c2*c3*c3*c3*s2*v1*v1*d2*cd34*m34-c2*c3*c3*s2*IX3*v1*v1
     +c2*c3*c3*s2*IY3*v1*v1+c2*c3*c3*s2
     *v1*v1*d2*d2*m34+c2*c3*c3*s2*v1*v1*cd34*cd34*m34
     +c2*c3*c3*g*d2*m34+c2*c3*s2*s3*s3*v1*v1*d2*cd34*m34+c2*c3
     *s2*v1*v1*d2*cd34*m34+c2*c3*g*cd34*m34+c2*s2*s3*s3*IX3*v1*v1
     -c2*s2*s3*s3*IY3*v1*v1+c2*s2*s3*s3*v1*v1*d2*d2*m34
     -c2*s2*s3*s3*v1*v1*cd34*cd34*m34-c2*s2*IX2*v1*v1+c2*
     s2*IY2*v1*v1+c2*s2*v1*v1*cd2*d2*m2+c2*s3*s3*g*d2*m34
     +c2*s3*a1*d1*cd34*m34+c2*g*cd2*m2
     -c3*c3*s2*s2*s3*v1*v1*d2*cd34*m34+c3*c3*s2*a1*d1*d2*
     m34+c3*c3*a2*d2*d2*m34+c3*s2*s2*s3*IX3*v1*v1
     -c3*s2*s2*s3*IY3*v1*v1-c3*s2*s2*s3*v1*v1*cd34*cd34*m34
     +c3*s2*a1*d1*cd34*m34+2.*c3*a2*d2*cd34*m34+
     c3*a3*d2*cd34*m34-s2*s2*s3*s3*s3*v1*v1*d2*cd34*m34
     +s2*s3*s3*a1*d1*d2*m34-s2*s3*g*cd34*m34+s2*a1*d1*cd2*m2
     +s3*s3*a2*d2*d2*m34-2.*s3*v2*v3*d2*cd34*m34-s3*v3*v3
     *d2*cd34*m34+IZ2*a2+IZ3*a2+IZ3*a3+a2*cd2*d2*m2+a2
     *cd34*cd34*m34+a3*cd34*cd34*m34
```

```
/*  CALCULO DE TORQUE PARA A JUNTA:  1 */
TQ1= c2*c2*c2*c3*c3*c3*v1*v1*d1*cd34*m34+c2*c2*c2
     *c3*c3*v1*v1*d1*d2*m34+c2*c2*c2*c3*
     s3*s3*v1*v1*d1*cd34*m34+c2*c2*c2*s3*s3*v1*v1*d1*d2*m34
     +c2*c2*c2*v1*v1*d1*cd2*m2-c2*c2*c3*c3*s2*s3*v1*v1*d1*cd34*m34
     +c2*c2*c3*c3*IY3*a1+c2*c2*c3*c3*a1*d1*d1*m34
     +c2*c2*c3*c3*a1*cd34*cd34*m34+2.*c2*c2
     *c3*s3*IX3*v1*v2+2.*c2*c2*c3*s3*IX3*v1*v3
     -2.*c2*c2*c3*s3*IY3*v1*v2-2.*c2*c2*c3*s3*IY3*v1*v3-2.
      *c2*c2*c3*s3*v1*v2*cd34*cd34*m34-2.*c2*c2
      *c3*s3*v1*v3*cd34*cd34*m34+2.*c2*c2*c3*a1*d2*cd34*m34
      -c2*c2*s2*s3*s3*s3*v1*v1*d1*cd34*m34+c2*c2*s3*s3*IX3*a1+
      c2*c2*s3*s3*a1*d1*d1*m34-2.*c2*c2*s3*v1*v2*d2*cd34*m34
      -2.*c2*c2*s3*v1*v3*d2*cd34*m34+c2*c2*IY2*a1
      +c2*c2*a1*d1*d1*m2+c2*c2*a1*d2*d2*m34+c2*c2*a1*cd2*d2*m2
      +c2*c3*c3*c3*s2*s2*v1*v1*d1*cd34*m34+c2
      *c3*c3*s2*s2*v1*v1*d1*d2*m34+2.*c2*c3*c3*s2*IX3*v1*v2
      +2.*c2*c3*c3*s2*IX3*v1*v3-2.*c2*c3*c3*s2*IY3*v1*v2
      -2.*c2*c3*c3*s2*IY3*v1*v3-2.*c2*c3*c3*s2*v1*v2*cd34*cd34*m34
      -2.*c2*c3*c3*s2*v1*v3*cd34*cd34*m34+c2*c3*c3
      *v2*v2*d1*d2*m34+c2*c3*s2*s2*s3*s3*v1*v1*d1*cd34*m34
      +2.*c2*c3*s2*s3*IX3*a1-2.*c2*c3*s2*s3*IY3*a1
      -2.*c2*c3*s2*s3*a1*cd34*cd34*m34
      -4.*c2*c3*s2*v1*v2*d2*cd34*m34-2.*c2*c3*s2*v1*v3*d2*cd34*m34
```

```
-c2*c3*v1*v1*d1*cd34*m34+c2*c3*v2*v2*d1*cd34*m34
+2.*c2*c3*v2*v3*d1*cd34*m34+c2*c3*v3*v3*d1*cd34*m34
+c2*s2*s2*s3*s3*v1*v1*d1*d2*m34+c2*s2*s2*v1*v1*d1*cd2*m2
-2.*c2*s2*s3*s3*IX3*v1*v2-2.*c2*s2*s3*s3*IX3*v1*v3+2.*c2*s2
*s3*s3*IY3*v1*v2+2.*c2*s2*s3*s3*IY3*v1*v3+2.*c2*s2*s3
*s3*v1*v2*cd34*cd34*m34+2.*c2*s2*s3*s3*v1*v3*cd34*cd34
*m34-2.*c2*s2*s3*a1*d2*cd34*m34+2.*c2*s2*IX2*v1*v2
-2.*c2*s2*IY2*v1*v2-2.*c2*s2*v1*v2*d2*d2*m34-2.*c2*
s2*v1*v2*cd2*d2*m2+c2*s3*s3*v2*v2*d1*d2*m34
+c2*s3*a2*d1*cd34*m34+c2*s3*a3*d1*cd34*m34
-c2*v1*v1*d1*d2*m34-c2*v1*v1*d1*cd2*m2+c2*v2*v2*d1*cd2*m2
-c3*c3*s2*s2*s2*s3*v1*v1*d1*cd34*m34+c3*c3*s2*s2
*IX3*a1+c3*c3*s2*s2*a1*d1*d1*m34+c3*c3*s2*a2*d1*d2*m34
-2.*c3*s2*s2*s3*IX3*v1*v2-2.*c3*s2*s2*s3*IX3*v1*v3
+2.*c3*s2*s2*s3*IY3*v1*v2+2.*c3*s2*s2*s3*IY3*v1*v3
+2.*c3*s2*s2*s3*v1*v2*cd34*cd34*m34+2.*c3*s2*s2*s3*v1
*v3*cd34*cd34*m34+c3*s2*a2*d1*cd34*m34+c3*s2*a3*d1*cd34*m34
-s2*s2*s2*s3*s3*s3*v1*v1*d1*cd34*m34+s2*s2*s3*s3*IY3*a1+
s2*s2*s3*s3*a1*d1*d1*m34+s2*s2*s3*s3*a1*cd34*cd34*m34
+2.*s2*s2*s3*v1*v2*d2*cd34*m34+s2*s2*IX2*a1+s2*s2*a1*d1*
d1*m2+s2*s3*s3*a2*d1*d2*m34+s2*s3*v1*v1*d1*cd34*m34
-s2*s3*v2*v2*d1*cd34*m34-2.*s2*s3*v2*v3*d1*cd34*m34
-s2*s3*v3*v3*d1*cd34*m34+s2*a2*d1*cd2*m2+IZ1*a1
```

onde

TQ1, TQ2, TQ3:	torque total nas juntas 1, 2 e 3,
d1, d2:	distância entre as origens dos sistemas de referência,
v1, v2, v3:	velocidade angular das juntas 1, 2 e 3,
a1, a2, a3:	aceleração angular das juntas 1, 2 e 3,
m2 e m34:	massa dos links 2 e 3 (link 3 e carga),
cd2 e cd34:	posição relativa do centro de gravidade dos links 2 e 3 (link 3 e carga),
g:	aceleração da gravidade,
IZ1, IX2, IY2, IZ2, IX3, IY3, IZ3:	termos da matriz de pseudo-inércia [CRAIG-86].

BASE DE DADOS

A base de dados do sistema de CAD-simulação está estruturada com o intuito de armazenar informações pertinentes a garra, objetos, robô e permitir sua manipulação aritmética através de uma área destinada a cálculos. Os módulos que compõe a base de dados são:

- estrutura ponto
- garra
- robô
- objeto
- cálculo

A estrutura ponto é semelhante a estrutura T6, contendo informações sobre o sistema: valor angular das juntas, posição-orientação da garra, localização do observador, escala de projeção, localização do plano de projeção, valor da garra, ângulos de rotação do observador, fator de translação no plano de vídeo e flag de condição de erro.

Os dados de valor angular das juntas e posição-orientação tem função idêntica aos da estrutura T6. A localização do observador e do plano de projeção, o fator de escala de projeção e o fator de translação são utilizados durante o procedimento de projeção de um ponto do espaço tri-dimensional para o plano de projeção. Os ângulos de rotação do observador permitem a observação do manipulador sob vários ângulos. A flag de erro indica a condição de erro detectada durante algum procedimento.

O item garra consiste de uma área de memória contendo informações da geometria da garra. O sistema reconheçe a garra como um apêndice do último link, acoplado ao último sistema de referência. A garra [KATO-82] é reconhecida como um elemento simétrico em relação ao primeiro, segundo, terceiro e quartos quadrantes do sistema de coordenadas cartesiano, permitindo a sua definição, fornecendo-se apenas as dimensões de largura, altura e comprimento em relação ao primeiro quadrante. Desta forma, o sistema identifica somente garras tipo paralela e de sucção, ver figura F29. Para DZ $\neq$ 0 tem-se uma garra paralela com deslocamento ao longo do eixo Y$_G$. Para DZ = 0 tem-se uma garra tipo sucção. A área de garra está dividida em 5 slots, numerados de 0 a 4, contendo os valores de DX, DY, DZ para 4 garras. O slot 0 contém DX = DY = DZ = 0 representando a ausência de garra.

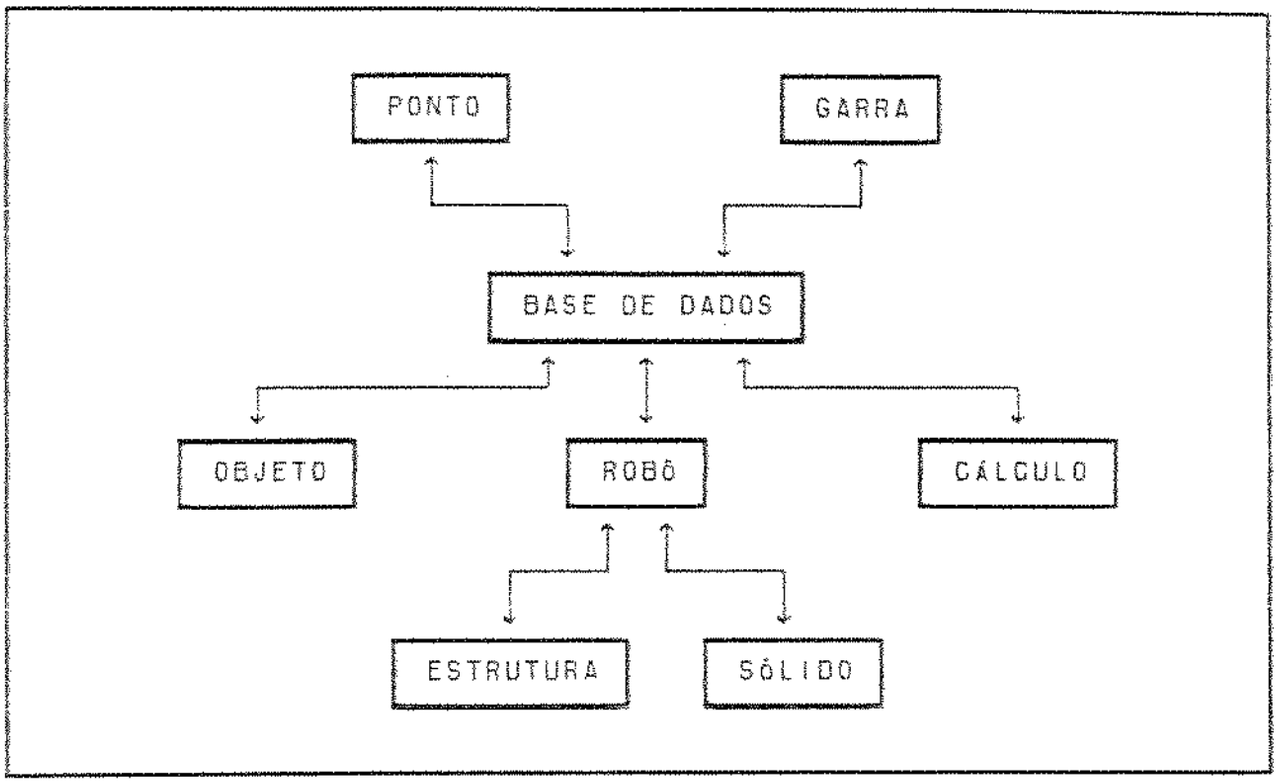

FIGURA F28 - ESTRUTURA DA BASE DE DADOS DO SISTEMA CAD-animação

O item robô consiste de uma área de memória destinada ao armazenamento de dados da representação gráfica do robô, que é considerado como um sub-conjunto de sólidos, sendo cada link modelado como um paralelepípedo, ver figura F30. Esta área de memória está dividida em duas sub-áreas:

- estrutura
- sólido

A sub-área de estrutura contém informações sobre as coordenadas cartesianas (PX,PY,PZ) dos vértices do paralelepípedo em relação ao referencial local e as relações de arestas, como mostrado nas tabelas de vértices e arestas, permitindo a representação gráfica do esqueleto do robô.

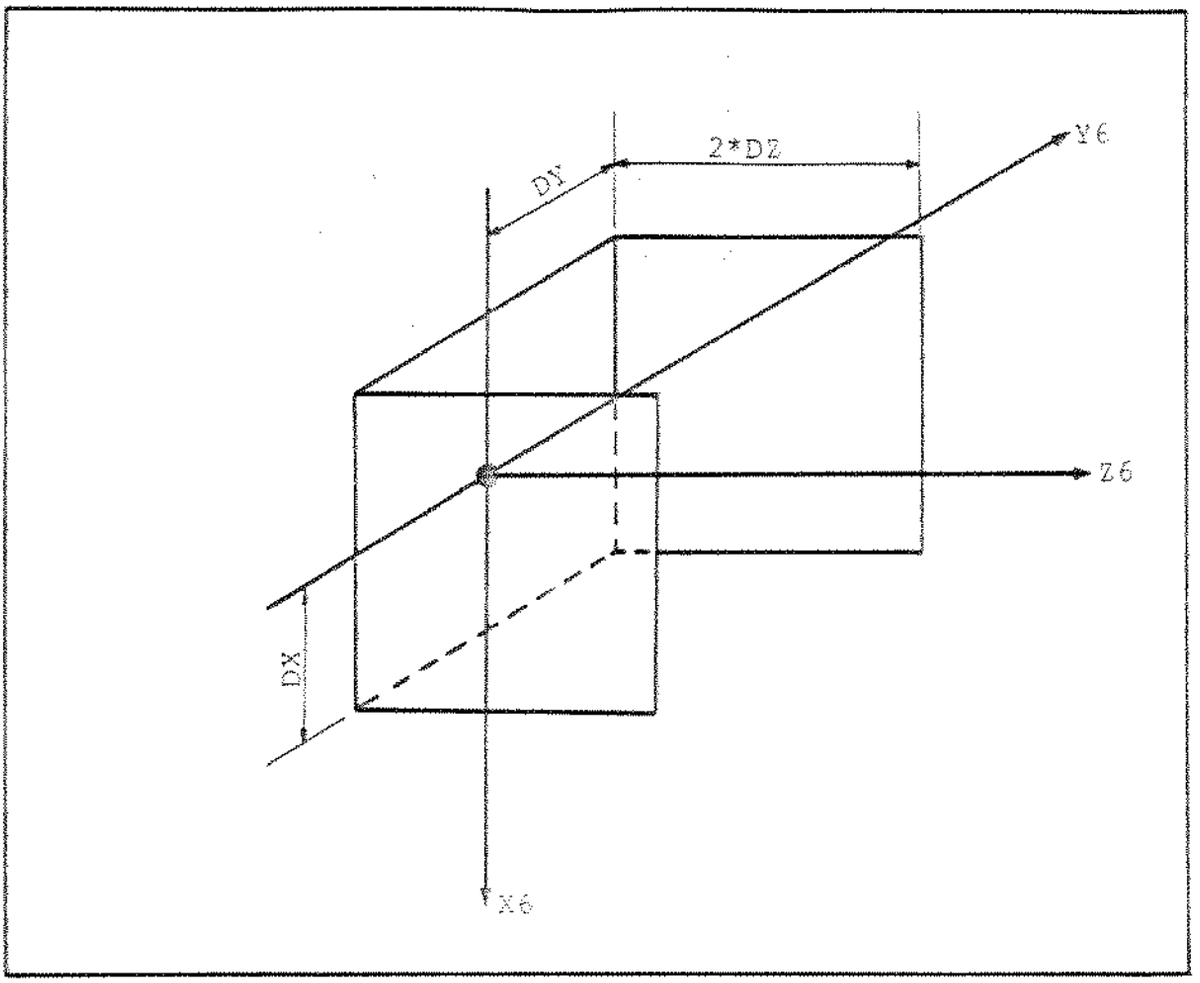

FIGURA F29 - GARRA

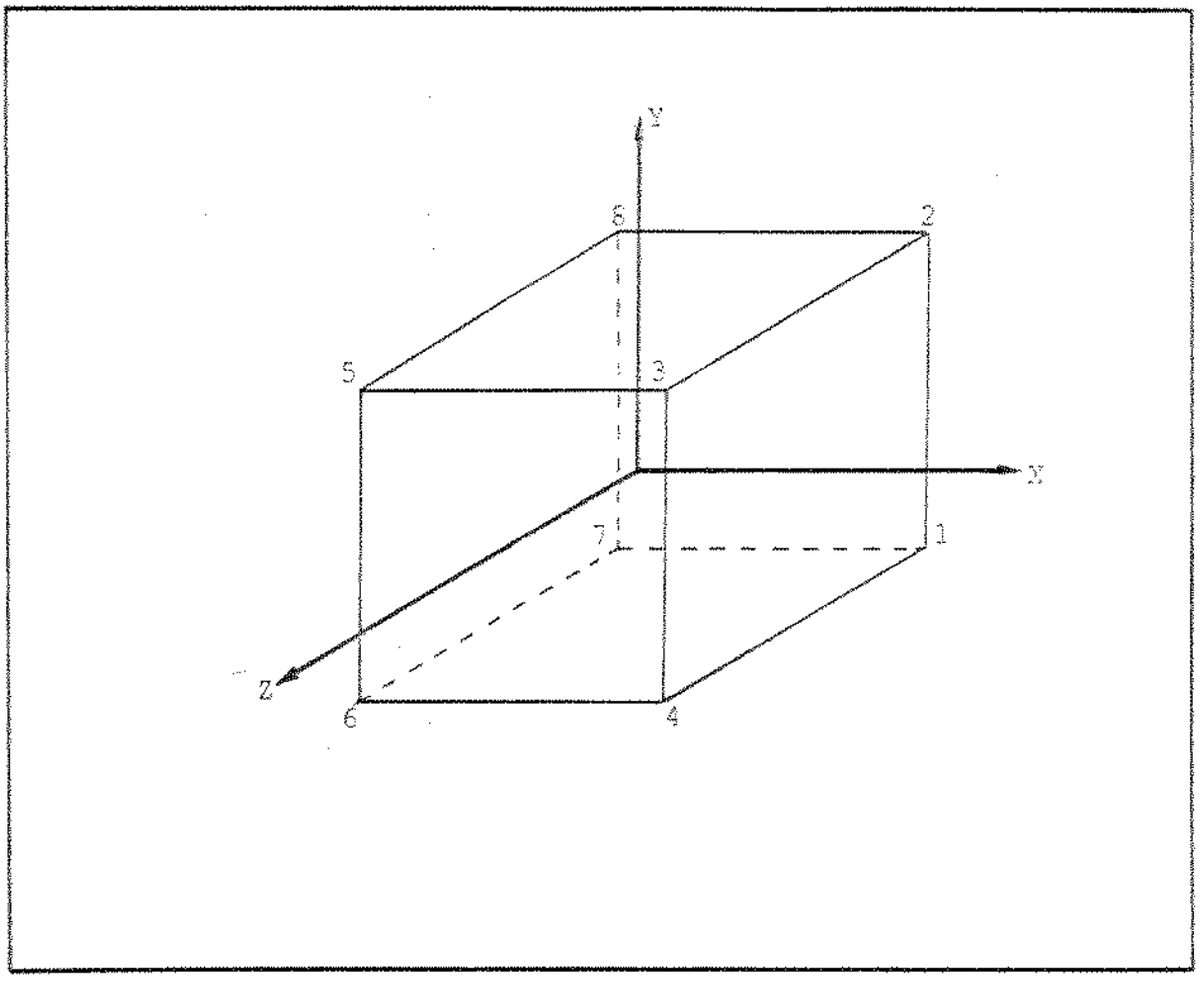

FIGURA F30 - LINK DO ROBÔ

LINK i			
VÉRTICE	COORDENADAS CARTESIANAS		
	PX	PY	PZ
1	....	....	....
.	....	....	....
.	....	....	....
8	....	....	....

TABELA DE VÉRTICES DOS LINKS

LINK i		
ARESTA	RELAÇÃO DE VÉRTICES	
1	1	2
2	2	3
3	1	4
4	3	4
5	2	8
6	1	7
7	3	5
8	4	6
9	5	6
10	5	8
11	6	7
12	7	8

TABELA DE ARESTAS

A sub-área de sólido contém informação sobre a relação de vértices que compõe as faces (planos) dos links. Por exemplo, seguindo o desenho da figura F30 tem-se:

FACE	VÉRTICES			
1	1	2	8	7
2	2	3	5	8
3	3	4	6	5
4	4		7	6
5	8	5	6	7
6	1	4	3	2

que permite a representação gráfica do robô como um sólido.

O item objeto consiste de uma área de memória destinada ao armazenamento de dados da representação gráfica dos objetos, contendo, como para a representação do robô, a tabela de vértices e a tabela de arestas de cada objeto. Esta área de memória é contigua a área do item robô, sendo considerada pelas subrotinas do sistema como um apêndice, processando os cálculos como um membro do robô.

O item cálculo consiste de uma área de memória destinada ao armazenamento de cálculos intermediários.

Todos os itens são de acesso global, sendo que a alocação de espaço físico na memória é realizada durante o procedimento de inicialização do sistema.

ROTINAS AUXILIARES

As rotinas auxiliares consistem de um conjunto de subrotinas de apoio a implementação dos módulos de CAD-animação e teach-in. O conhecimento da estrutura funcional destas rotinas permite o melhor entendimento da implementação e operação destes módulos, ver figura F31. Estas rotinas estão divididas em seis blocos:

- inicialização
- vídeo
- comunicação
- I/O dados de entrada e saída
- modelamento
- gráficos

A descrição dos blocos será feita com a intenção de esclarecer sua funcionalidade.

O bloco de inicialização tem a finalidade de inicializar o sistema gráfico da linguagem C e variáveis de uso interno.

O bloco vídeo consiste de duas subrotinas de auxílio no gerenciamento do envio de mensagens e entrada de dados, pois, a interface homem-máquina está estruturada utilizando o conceito de janelas, que é suficientemente amplo para ser abordado neste trabalho. No entanto, o sistema entende como "janela" uma área destacada na região de vídeo destinada a entrada de dados e envio de mensagens, após o que, terminada sua utilidade, o conteúdo primitivo da área ocupada é restaurado.

O bloco de comunicação tem a função de gerenciar a troca de informações entre o modo teach-in e a linguagem, sendo realizada através de arquivo de dados. Na linguagem de programação o usuário tem a disposição os comandos LOADD() e SAVED() para carregar e salvar dados do e para o módulo teach-in.

O bloco I/O-dados tem a função de interface homem-máquina, enviando mensagens e entrada de dados.

O bloco modelamento tem a função de gerenciar as subrotinas de cálculo matemático dedicado: modelamento geométrico direto, inverso e dinâmico, projeção, cálculo de posição dos links do robô e cálculo do ângulo de observação do observador.

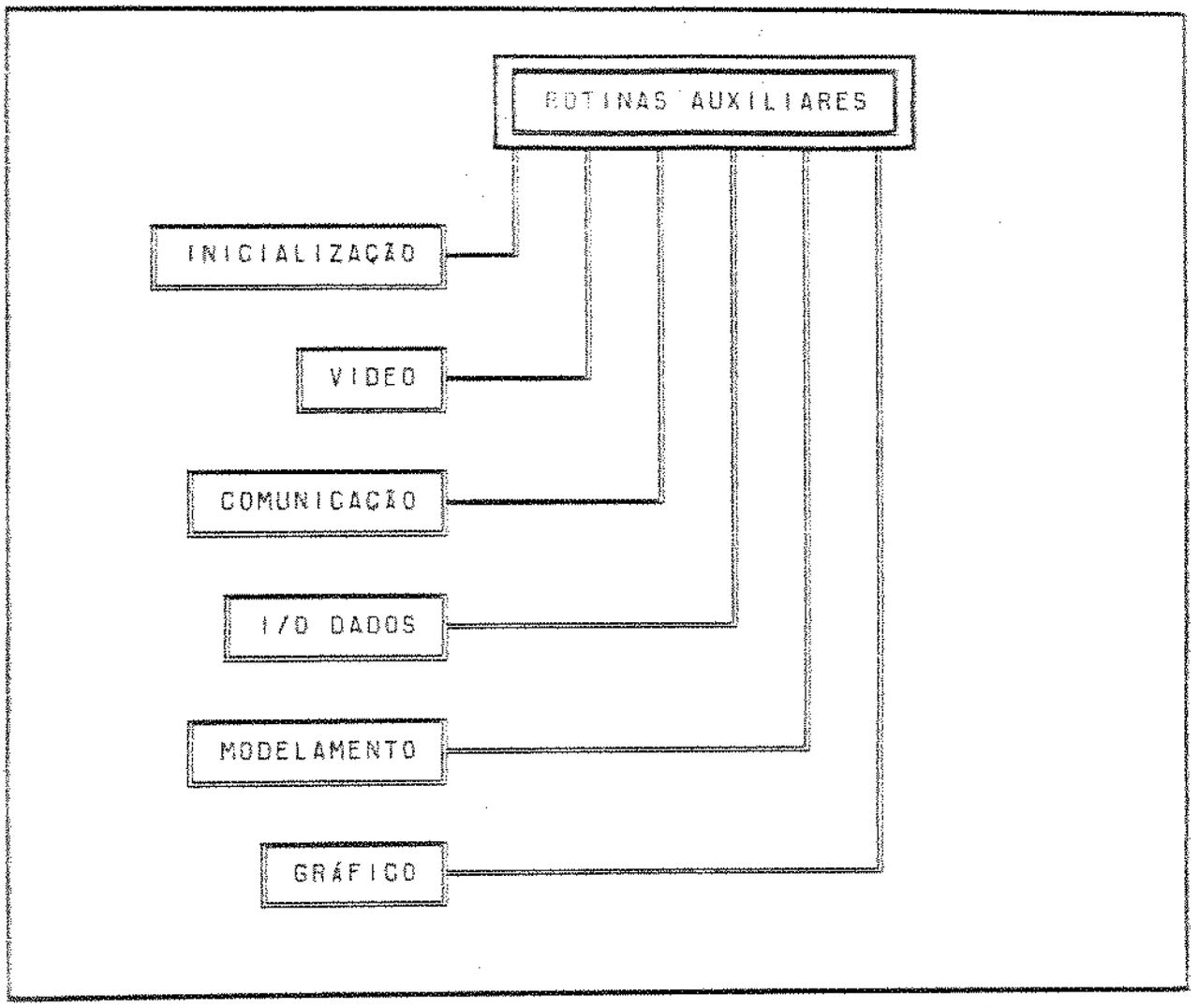

FIGURA F31 - ESTRUTURA FUNCIONAL DAS ROTINAS AUXILIARES

Uma rotina de projeção projeta um ponto no espaço tri-dimensional para o plano de projeção (visualização). A figura F32 mostra a posição do observador (XO), o ponto (P) a ser projetado no plano α e a sua projeção p.

A figura F33 mostra a vista superior (plano XY) do robô na posição de zero-máquina, indicando a posição de um observador e do plano de projeção, bem como, as direções de movimento. O sistema permite o movimento de aproximação e afastamento do observador e do plano de projeção em relação aos elementos da área de trabalho, sobre o eixo X_O, permitindo a visualização de detalhes ou do conjunto. Como os movimentos são sincronizados, ambos são aproximandos ou afastados simultaneamente, produzindo o efeito visual desejado com maior rapidez.

O robô, os objetos e a garra consistem em um conjunto de pontos P_i, sobre os quais, aplicando sucessivas

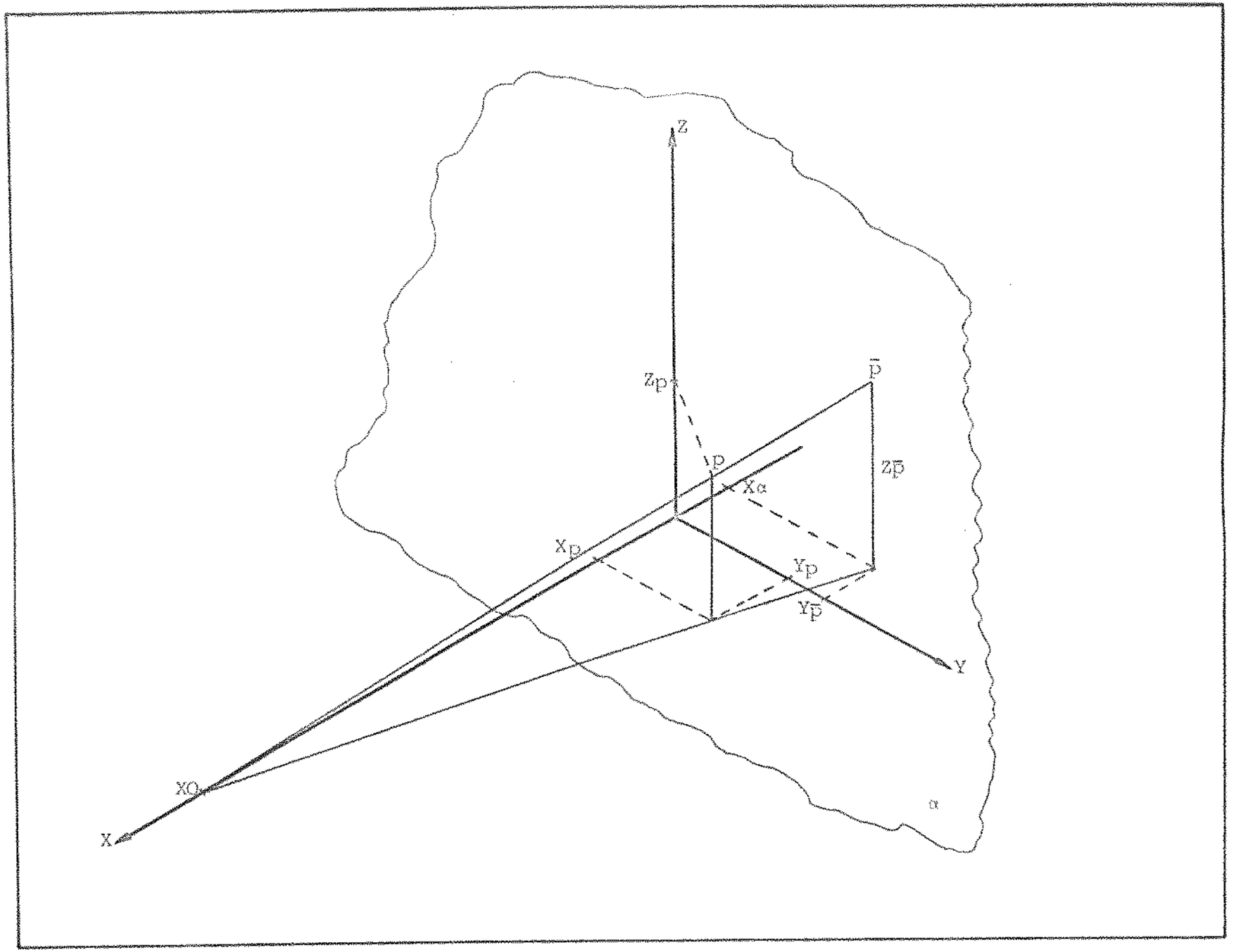

FIGURA F32 - PROJEÇÃO

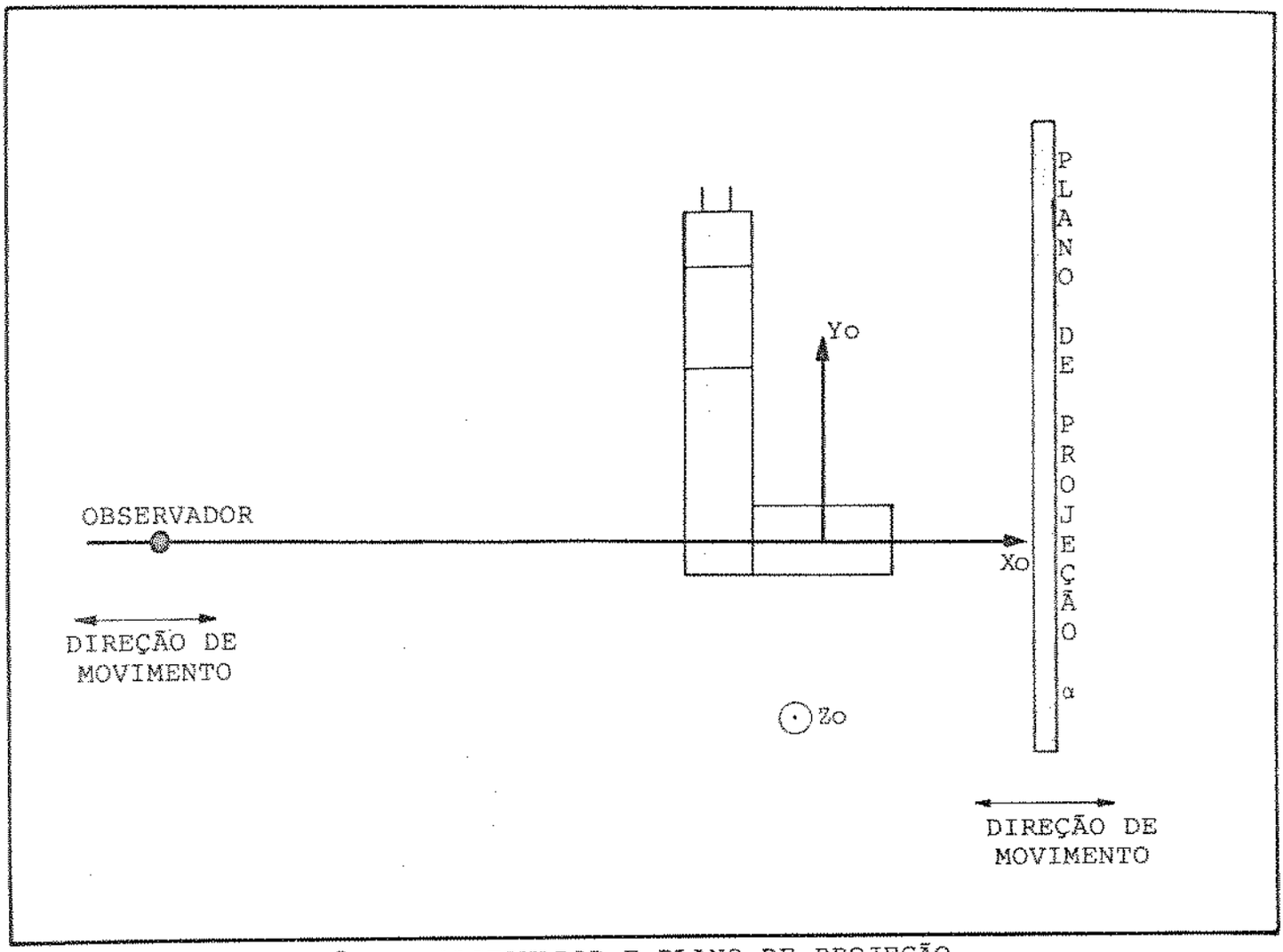

FIGURA F33 - POSIÇÃO DO OBSERVADOR E PLANO DE PROJEÇÃO

rotações (eixo X,Y,Z) pode-se mudar o ângulo de observação, ou seja, produz-se novos pontos P_i' correspondendo a uma nova posição de observação. Com este procedimento, não é o observador que move e sim, toda a imagem da área de trabalho.

A figura F34 mostra o fluxo de procedimentos utilizado para a geração de uma imagem. Aos pontos definidos na base de dados executa-se uma subrotina localizando os vértices da imagem do robô, da garra e objetos no referencial absoluto, a partir da qual executa-se uma rotina alterando o ângulo de observação da área de trabalho. A seguir, executa-se a subrotina de projeção e finalmente a plotagem da imagem na área de vídeo.

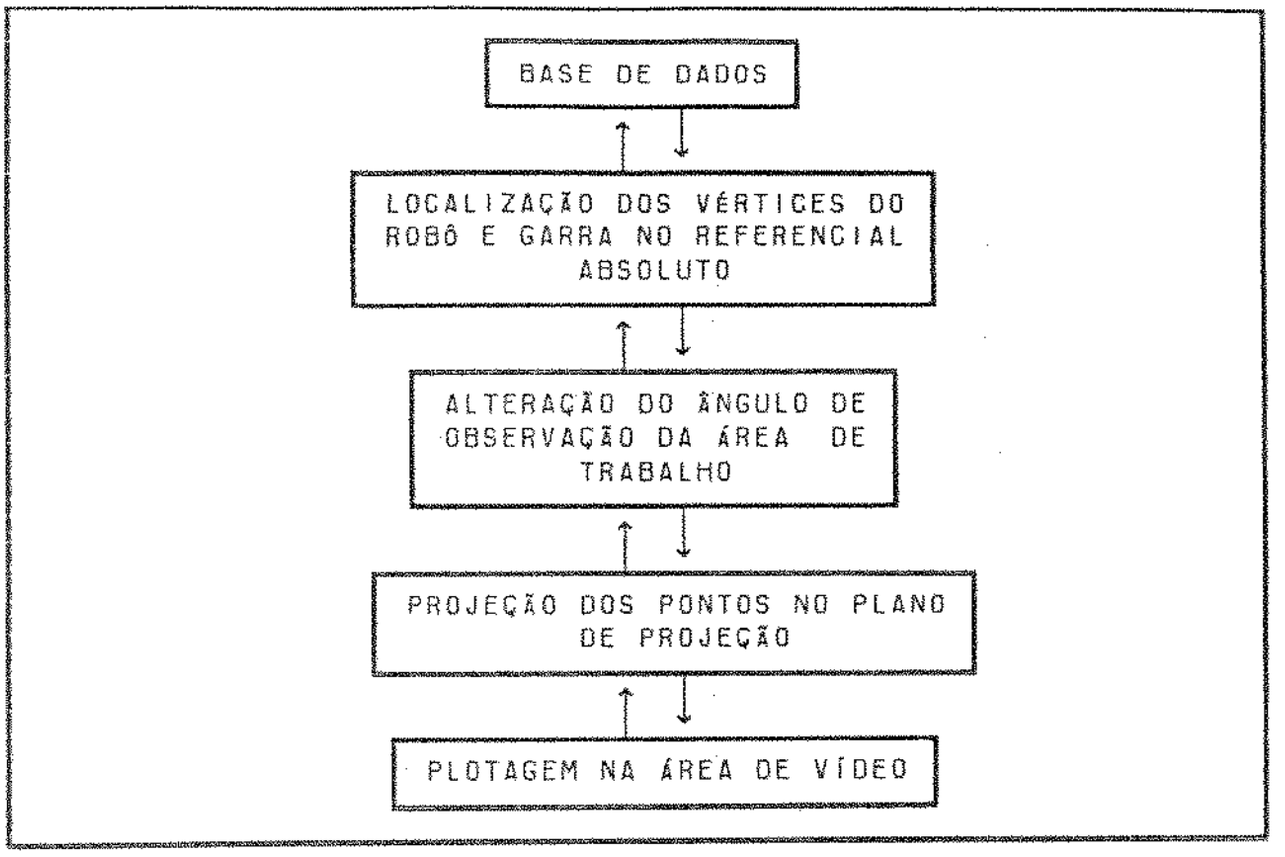

FIGURA F34 - FLUXOGRAMA DOS PROCEDIMENTOS PARA ANIMAÇÃO GRÁFICA

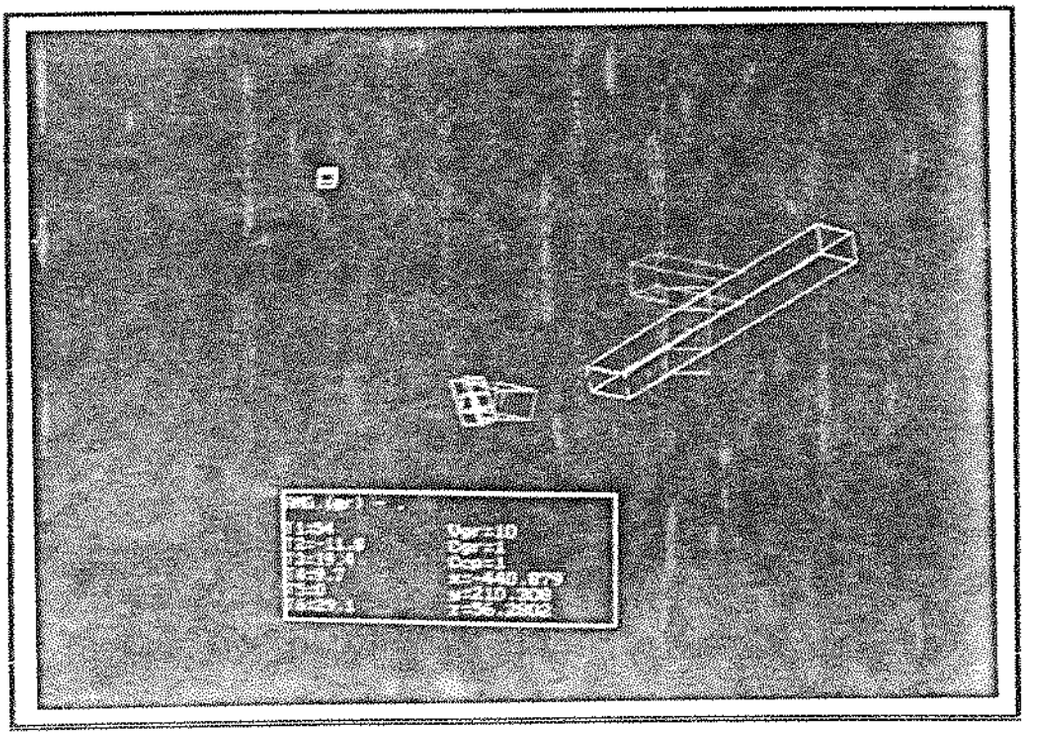

FIGURA F35 - ROBÔ COMO ESQUELETO

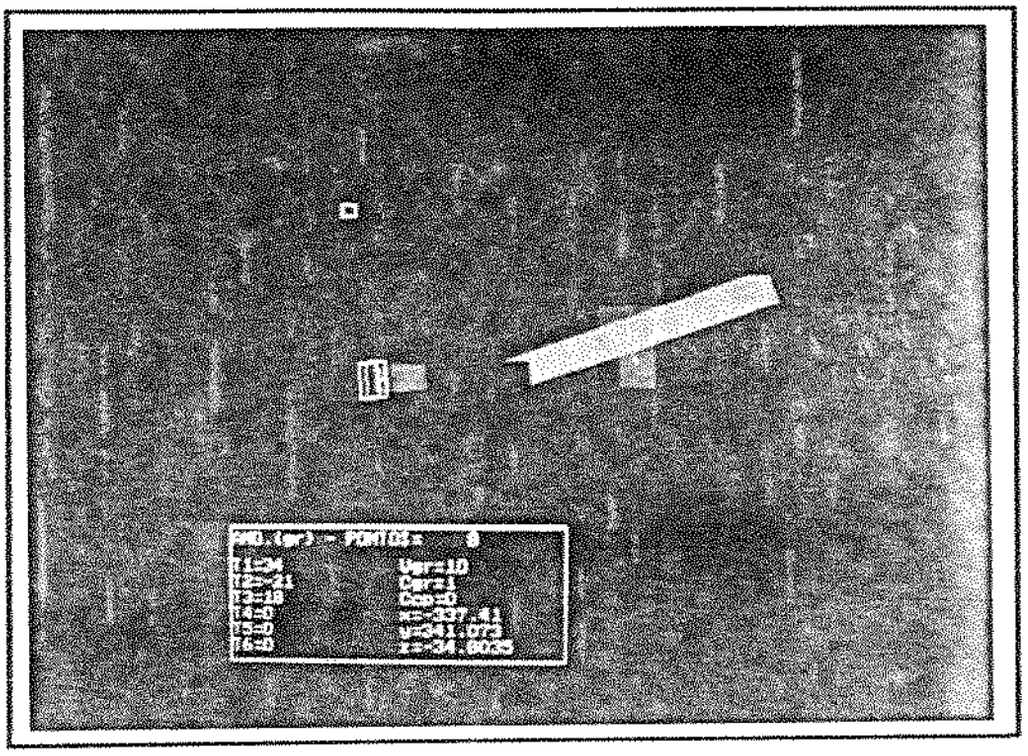

FIGURA F36 - ROBÔ COMO SÓLIDO

O bloco gráfico é composto de um conjunto de subrotinas auxiliares para a implementação da animação gráfica, sendo possível representar o robô graficamente de duas formas:

- esqueleto
- sólido

Na forma de esqueleto é representado o contorno (o esqueleto) do robô, como um "modelo de arame", ou seja, as arestas que formam os links. Na forma de sólido, o robô é apresentado como corpo sólido. As faces que compõe a representação prismática dos links são preenchidas com a cor das arestas. A plotagem é executada a partir da orientação do link em relação ao usuário que não pode ter seu ângulo de observação alterado. O intervalo angular das juntas é dividido em faixas de 45°, de tal forma, que cada combinação de faixas produza uma sequência na ordenação de plotagem dos links. O efeito obtido é o de "linhas escondidas" com um baixo custo computacional, sendo plotadas 2 imagens completas por segundo. Em contrapartida, a garra e os objetos são plotados somente na forma de esqueleto.

Algumas subrotinas auxiliam diretamente na implementação dos modos de CAD-animação e teach-in. Suas funções são: monitorar o teclado, setar os modos de visualização e gerenciar a plotagem das imagens. Durante a execução da animação o usuário

pode alterar os parâmetros e modos de visualização, tendo a disposição os seguintes procedimentos:

TECLA (Uso no TEACH-IN e CAD-Animação)	
o O	tornar ativo um objeto definido pela linguagem de programação
e E	ativar o editor de pontos setados
x X	executar uma sequência de pontos setados
m M	setar uma nova posição-orientação da garra
r R	setar o valor da garra no módulo teach-in
<ENTER>	setar uma nova posição-orientação para módulo teach-in
t T	setar um valor de step para o sistema, alterando a discretização angular de movimento das juntas e ângulo de observação
n	ativar o modo passo a passo. A animação é executada quadro a quadro aguardando que uma tecla seja digitada
N	resetar o modo passo a passo. A animação é executada "continuamente"
q Q	setar uma condição de abandono, abortando a execução da subrotina
[	incrementar step em 10 unidades de medida
]	decrementar step em 10 unidades de medida
{	incrementar step em 1 unidade de medida
}	decrementar step em 1 unidade de medida
P	incrementar step em 0.1 da unidade de medida
p	decrementar step em 0.1 da unidade de medida
+	fixar step em +0.1 da unidade de medida
-	fixar step em -0.1 da unidade de medida
1	deslocamento da junta 1 de um step
2	deslocamento da junta 2 de um step

3	deslocamento da junta 3 de um step
4	deslocamento da junta 4 de um step
5	deslocamento da junta 5 de um step
6	deslocamento da junta 6 de um step
0	afastar o ponto do observador e o plano de projeção da posição da posição corrente
.	aproximar o ponto do observador e o plano de projeção da posição corrente
7	mudança do ângulo do observador na direção X de um step
8	mudança do ângulo do observador na direção Y de um step
9	mudança do ângulo do observador na direção Z de um step
l L	ativar a opção de traçado da trajetória da garra. Uma linha é plotada indicando a trajetória percorrida pela garra
G	setar código da garra ativa
s	setar plotagem do robô na forma de sólido
w	setar plotagem do robô na forma de esqueleto
v	setar plotagem das projeções
f	setar "zoom" gráfico da projeção no plano YZ
g	setar "zoom" gráfico da projeção no plano -XZ
h	setar "zoom" gráfico da projeção no plano YX
/	setar plotagem do robô na forma de sólido e esqueleto
*	conduz o robô a posição de zero-máquina
d D	setar opção de pausa e consulta ao usuário entre dois comandos sucessivos da linguagem

A seguir apresentar-se-á uma série de imagens geradas pelo sistema de CAD-Simulação:

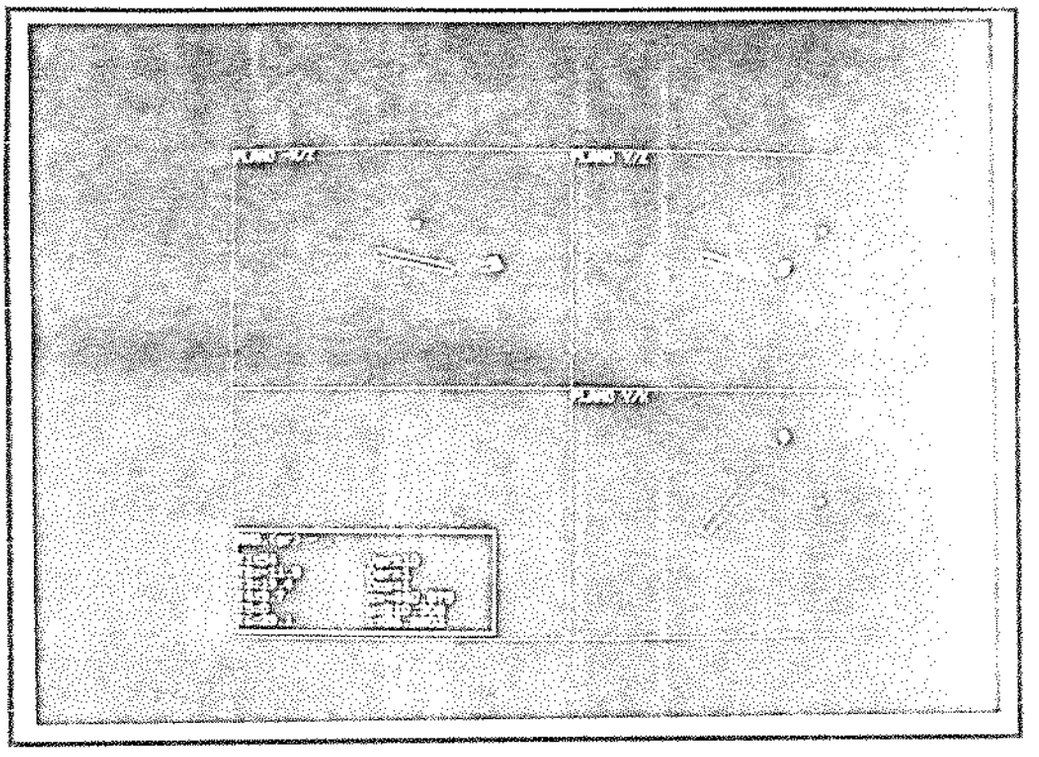

FIGURA F37 - PROJEÇÕES

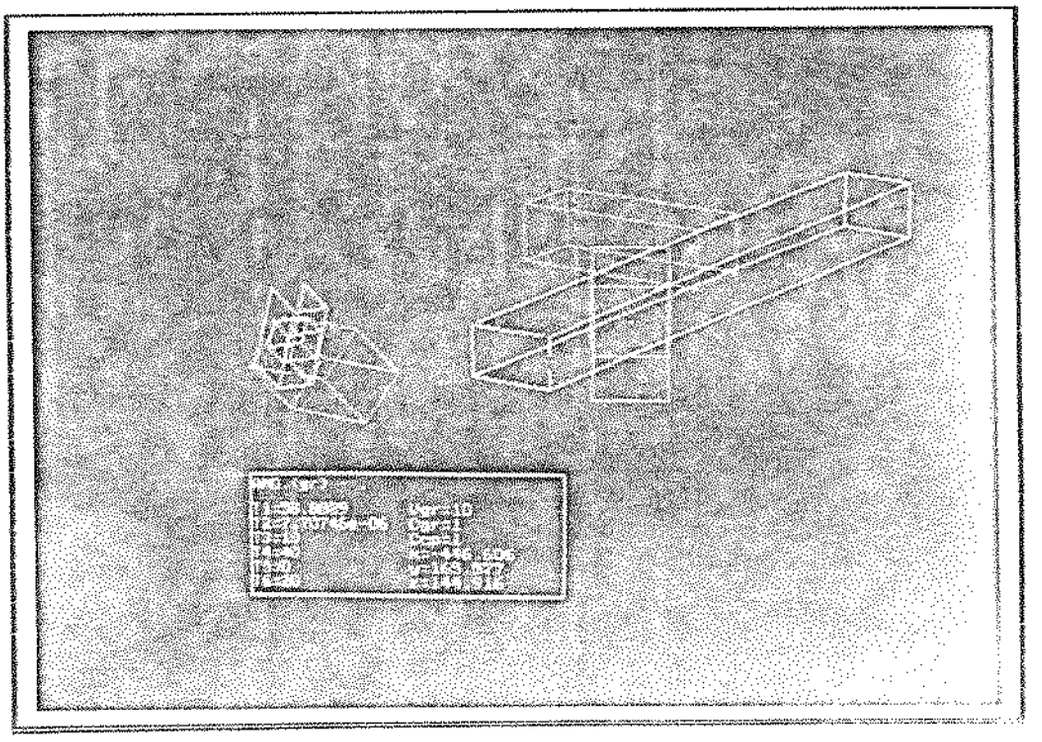

FIGURA F38 - "ZOOM" GRÁFICO

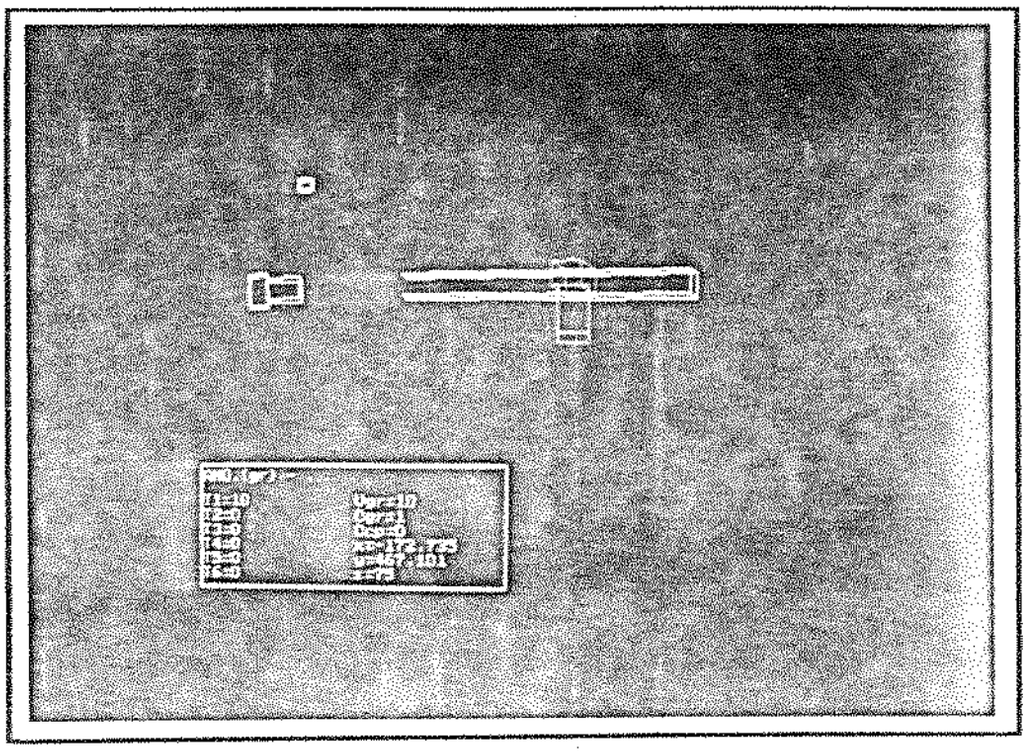

FIGURA F39 - VISTA FRONTAL

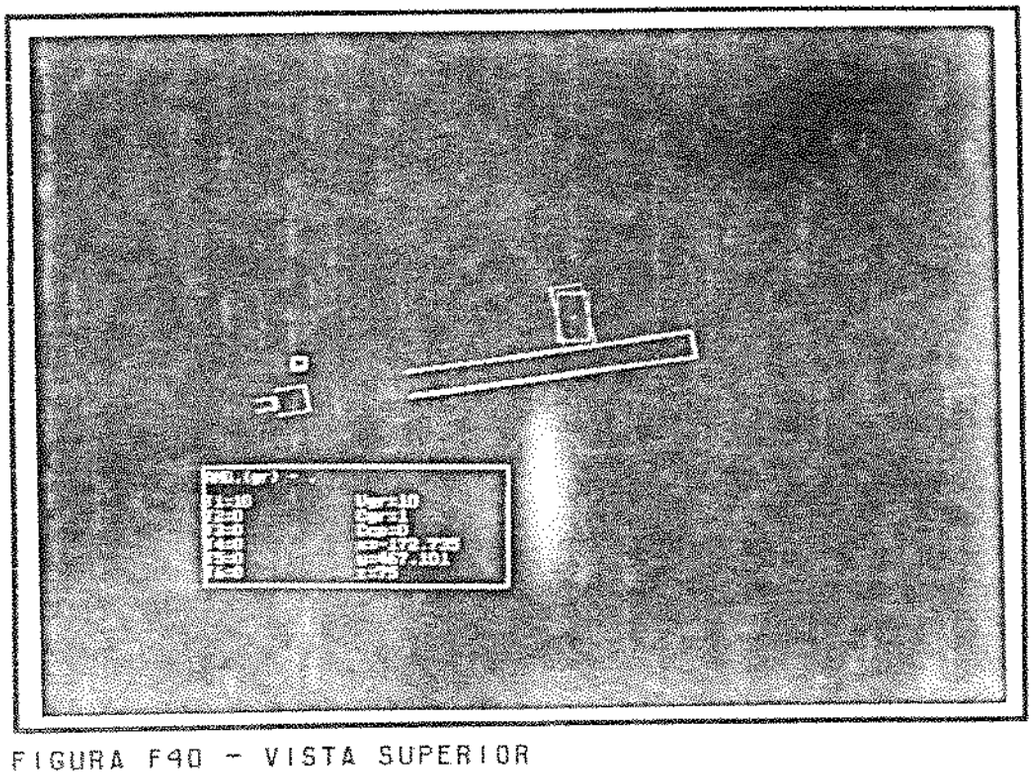

FIGURA F40 - VISTA SUPERIOR

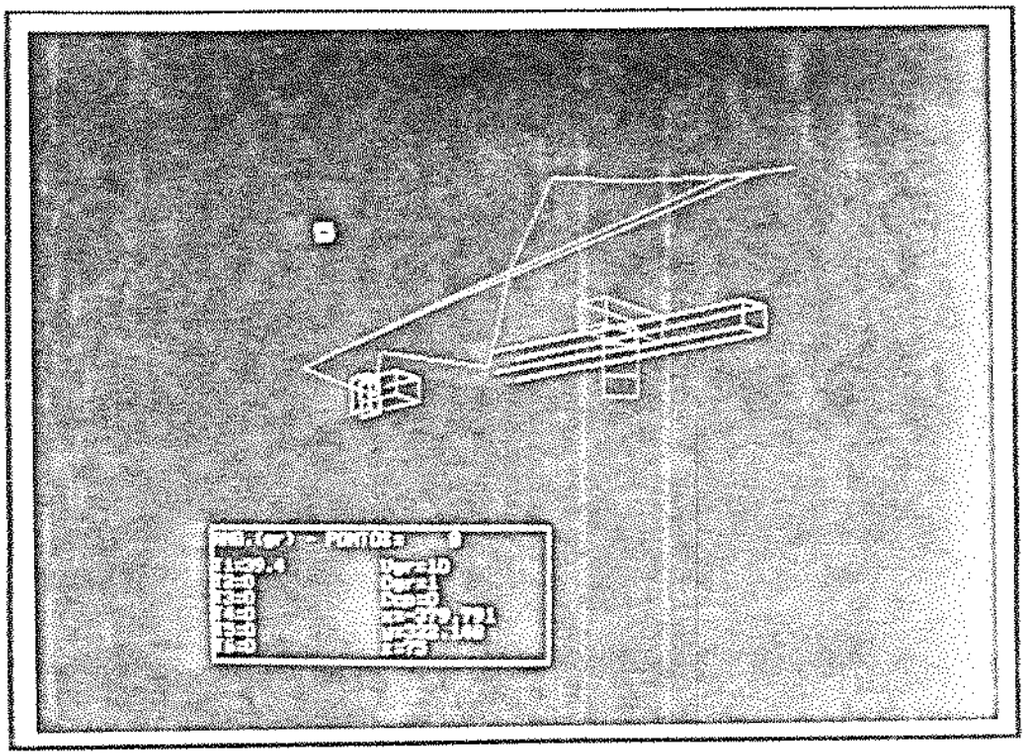

FIGURA F41 - OPÇÃO DE TRAÇADO

A figura F42 mostra o layout do teclado numérico com algumas funções atribuídas as teclas. Para cada ciclo de monitoramento é gerada uma nova imagem em uma área de memória auxiliar, de forma, que o usuário não observa sua plotagem. Com a imagem pronta, a área auxiliar é setada como ativa, permitindo sua visualização e a área que estava ativa torna-se auxiliar. Este procedimento de seleção de áreas de memória (páginas gráficas) é interno da placa de vídeo, sendo acionado por flags na área de I/O do computador.

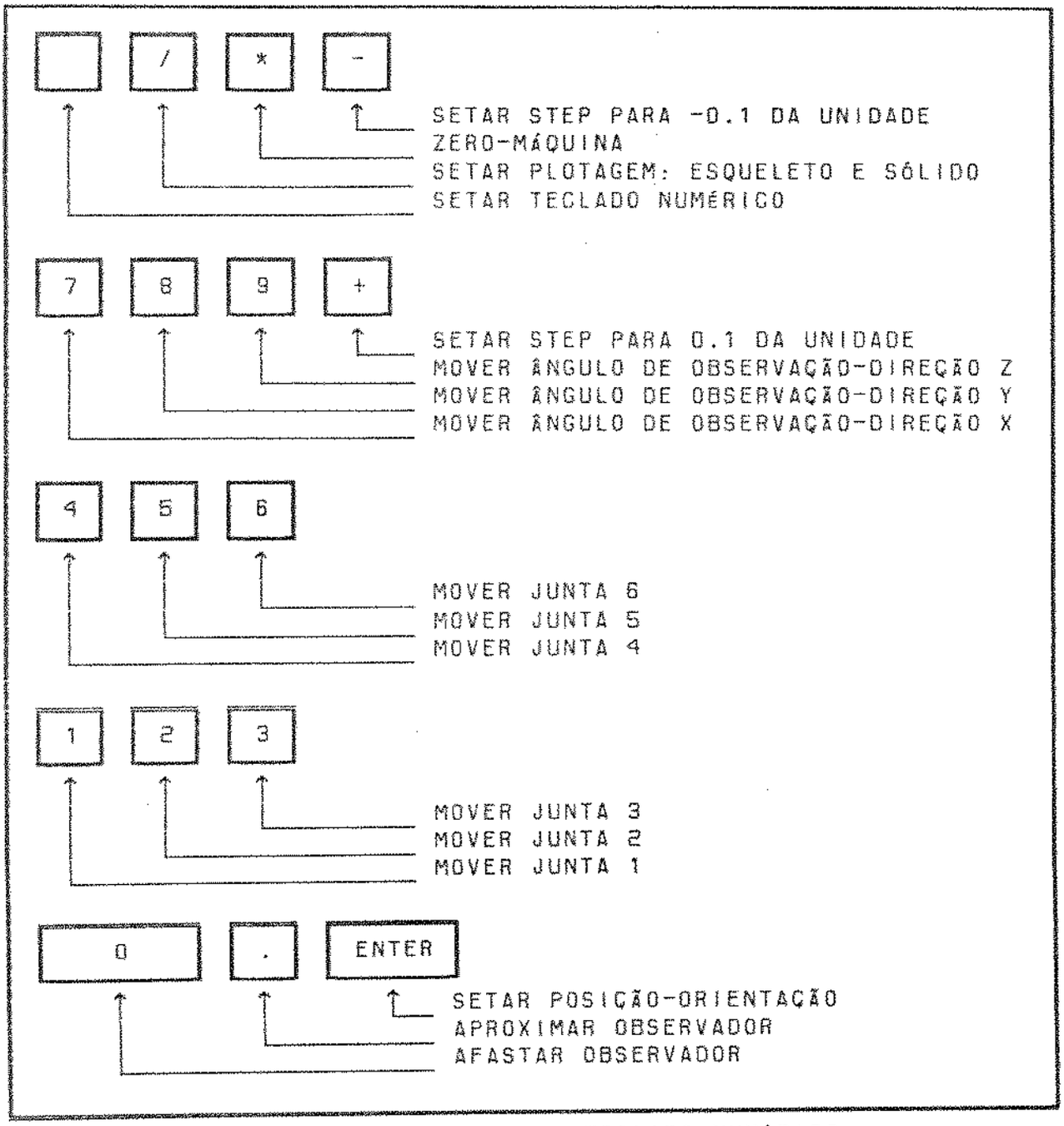

FIGURA F42 - FUNÇÕES ATRIBUIDAS AO TECLADO NUMÉRICO

PARTE D

EXEMPLOS

EXEMPLO

Como exemplo apresentar-se-á a implementação de uma tarefa utilizando alguns comandos básicos da linguagem com o intuito de mostrar:

- definição de tipos de dados,
- definição e manipulação de objetos,
- movimento LIVRE e LINEAR,
- interface com o TEACH-IN,
- execução de movimentos repetitivos,
- utilização de comandos da linguagem C
- execução de uma tarefa,
- simulação.

Outros comandos da linguagem podem ser utilizados seguindo o procedimento desenvolvido no exemplo a seguir.

PROGRAMA

```
/*****************************************************************/
/*                        PROGRAMA - EXEMPLO                     */
/*****************************************************************/
/*                                                               */
/*  Por André Mendeleck                                          */
/*                                                               */
/*  DATA:  14/02/1991                                            */
/*  LOCAL: UNICAMP - Universidade Estadual de Campinas           */
/*         FEM - Faculdade de Engenharia Mecânica                */
/*         DMC - Departamento de mecânica Computational          */
/*****************************************************************/
void program(void)
{
  static struct tB t1,t2;
  struct objeto tt;
  double k;

  unsigned long int i,j;

  /*setar numero de pontos de discretizacao*/
  setnpt(ATIVAR,20);

  /*definir objeto 1 com valores default*/
  dfobj(tt,3);

  /*definir objeto 2 com valores default*/
  dfobj(tt,3);
```

```
/*setar movimento livre*/
movimento(LIVRE);

/*trocar garra*/
trocagarra(1);

/*abrir garra*/
abrir(10.);

/*ler posicao-orientacoao de pega do objeto 1*/
popxpg(1,&t1);

/*mover para o ponto de pega do objeto 1*/
mover(t1);

/*setar objeto 1 como ativo*/
stobjeto(1);

/*carregar dados do teach-in*/
loadd("tp");

/*mover para o primeiro ponto*/
mover(ptt[1]);

/*repetir processo 3 vezes*/
for(i=1;i<=3;i++)
 {
   mover(ptt[2]);
   mover(ptt[3]);
   mover(ptt[1]);
   }

/*setar movimento linear*/
movimento(LINEAR);

/*mover para o segundo ponto do teach-in*/
mover(ptt[2]);

/*mover para o terceiro ponto do teach-in*/
mover(ptt[3]);

/*desativar objeto*/
stobjeto(0);

/*setar movimento livre*/
movimento(LIVRE);

/*mover para o primento ponto do teach-in*/
mover(ptt[1]);

/*ler posicao-orientacoao de pega do objeto 1*/
popxpg(1,&t1);
```

```
  /*mover para o ponto de pega do objeto 1*/
  mover(t1);

  /*setar objeto 1 como ativo*/
  stobjeto(1);

  /*mover para o primento ponto do teach-in*/
  mover(ptt[1]);

  /*mover para a posicao de zero maquina*/
  zeromaq();
}
```

EXECUÇÃO

```
[COMANDO     1]:SETNPT   : SETAR NUMERO DE PONTOS INTERPOLADOS
[COMANDO     2]:DFOBJ    : DEFINIR OBJETO
[COMANDO     3]:DFOBJ    : DEFINIR OBJETO
[COMANDO     4]:MOVIMENTO   : SETAR TIPO DE MOVIMENTO
[COMANDO     5]:TROCAGARRA   : TROCA DA GARRA
[COMANDO     6]:ABRIR    : ABRIR GARRA
[COMANDO     7]:POPXPG   : POS-ORI DO OBJETO
[COMANDO     8]:MOVER    : ...
[COMANDO     9]:STOBJETO   : SETAR OBJETO ATIVO
[COMANDO    10]:LOADP    : LOAD DADOS TEACH-IN
[COMANDO    11]:MOVER    : ...
[COMANDO    12]:MOVER    : ...
[COMANDO    13]:MOVER    : ...
[COMANDO    14]:MOVER    : ...
[COMANDO    15]:MOVER    : ...
[COMANDO    16]:MOVER    : ...
[COMANDO    17]:MOVER    : ...
[COMANDO    18]:MOVER    : ...
[COMANDO    19]:MOVER    : ...
[COMANDO    20]:MOVER    : ...
[COMANDO    21]:MOVIMENTO   : SETAR TIPO DE MOVIMENTO
[COMANDO    22]:MOVER    : ...
[COMANDO    23]:MOVER    : ...
[COMANDO    24]:STOBJETO   : SETAR OBJETO ATIVO
[COMANDO    25]:MOVIMENTO   : SETAR TIPO DE MOVIMENTO
[COMANDO    26]:MOVER    : ...
[COMANDO    27]:POPXPG   : POS-ORI DO OBJETO
[COMANDO    28]:MOVER    : ...
[COMANDO    29]:STOBJETO   : SETAR OBJETO ATIVO
[COMANDO    30]:MOVER    : ...
[COMANDO    31]:ZEROMAQ   : MOVER PARA POSICAO DE ZEROMAQUINA
```

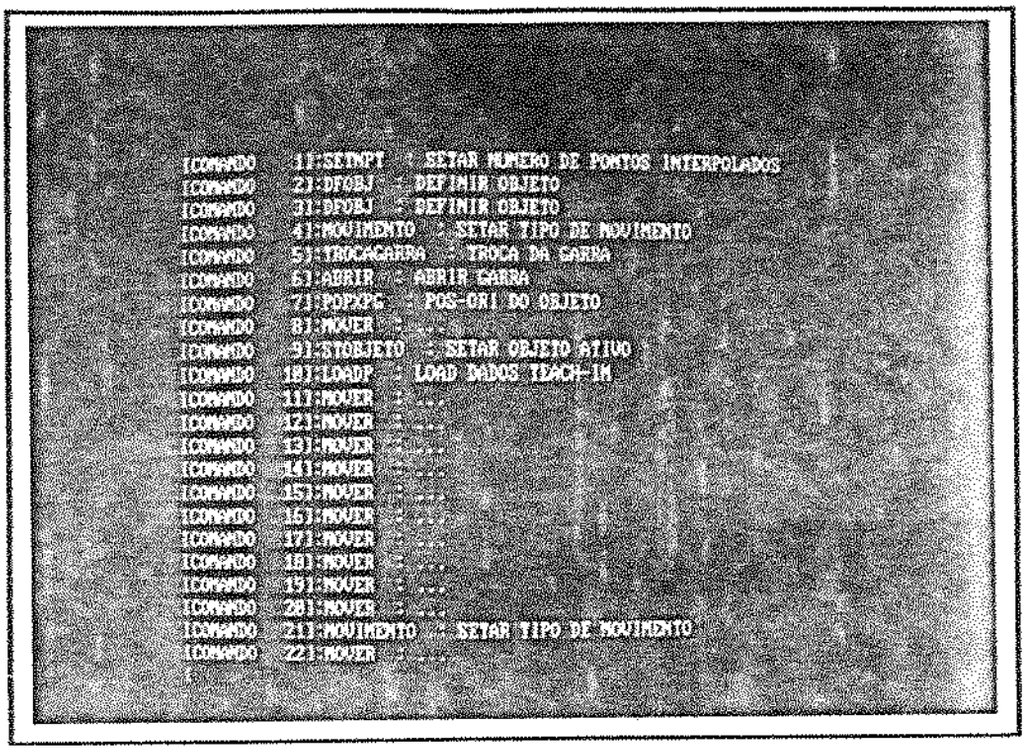

FIGURA F43 - EXECUÇÃO

Como ilustração, selecionou-se um comando (comando 11) de movimento do exemplo e plotou-se os gráficos disponíveis no módulo de MODELAMENTO, como segue nas páginas a seguir. As configurações são:

- inicial: θ_1 = 0.0 rad.
  θ_2 = 0.83422 rad.
  θ_3 = -0.89212 rad.
  θ_4 = 0.05790 rad.
  θ_5 = 0.0 rad.
  θ_6 = 0.0 rad.

- final: θ_1 = 0.34907 rad.
  θ_2 = 0.17453 rad.
  θ_3 = 0.34907 rad.
  θ_4 = 0.0 rad.
  θ_5 = 0.0 rad.
  θ_6 = 0.0 rad.

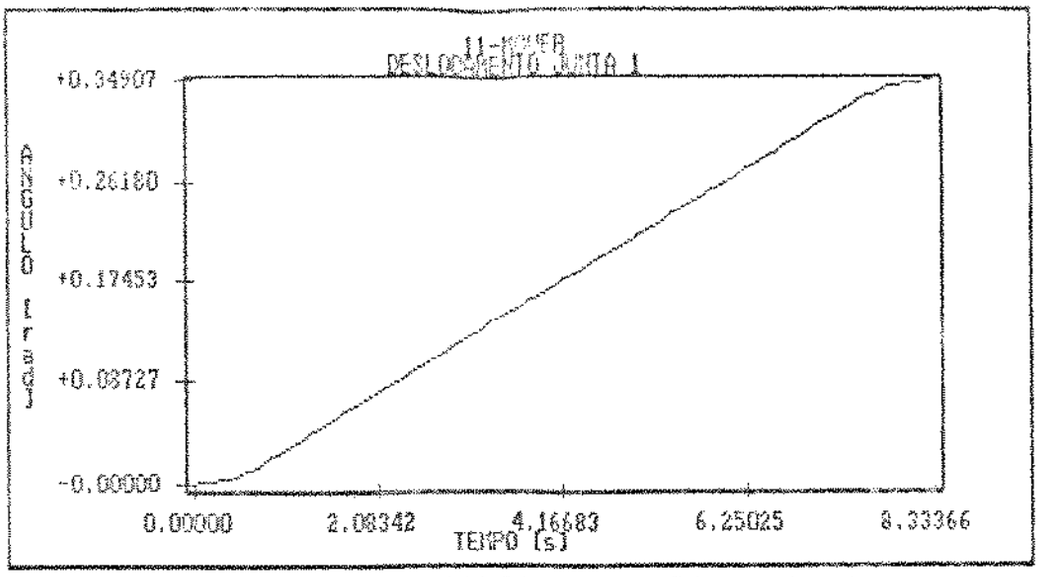

FIGURA 44 - DESLOCAMENTO DA JUNTA 1

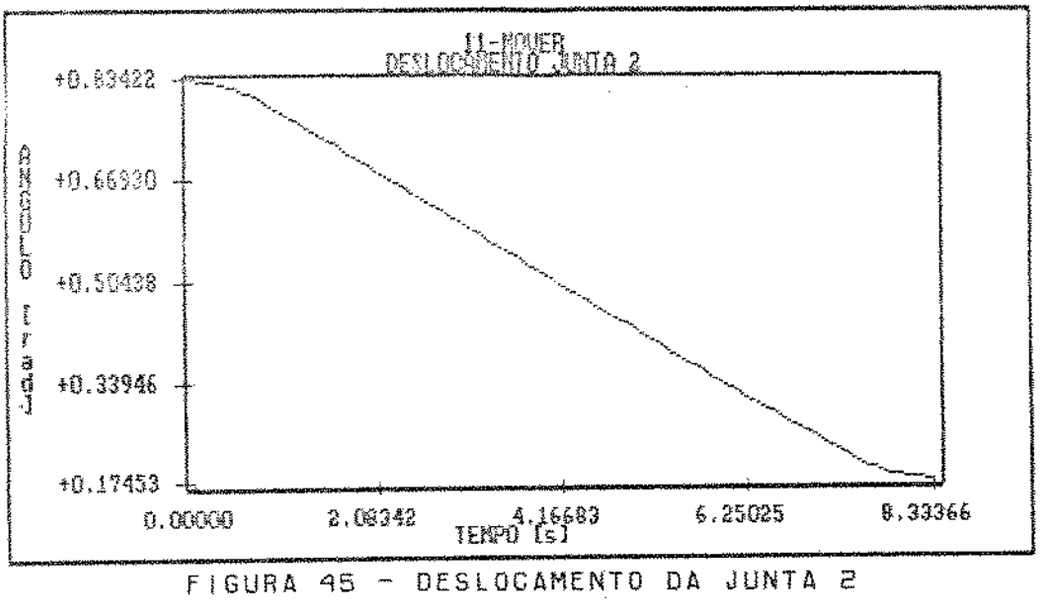

FIGURA 45 - DESLOCAMENTO DA JUNTA 2

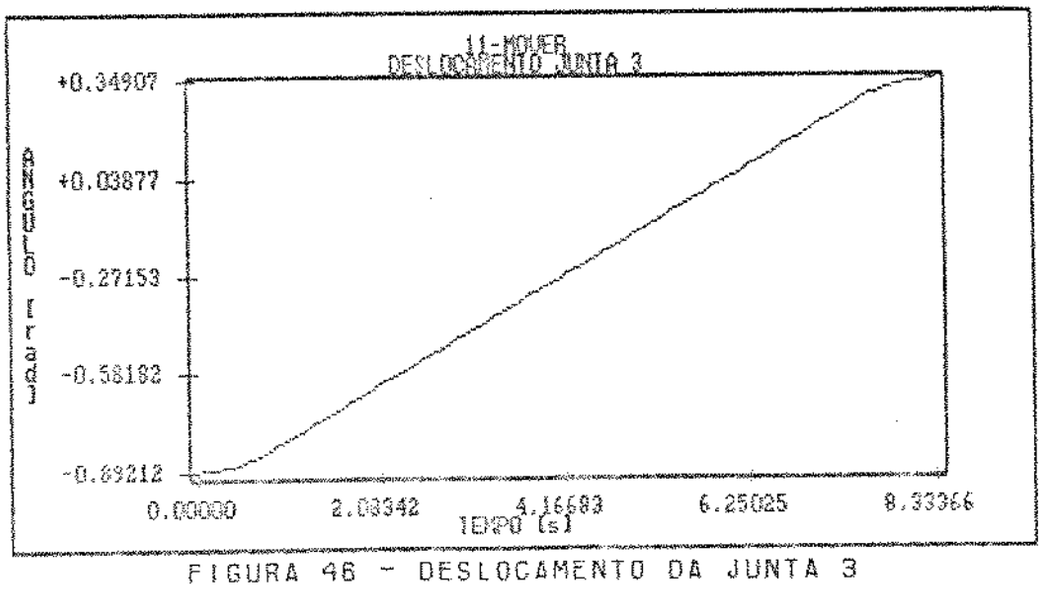

FIGURA 46 - DESLOCAMENTO DA JUNTA 3

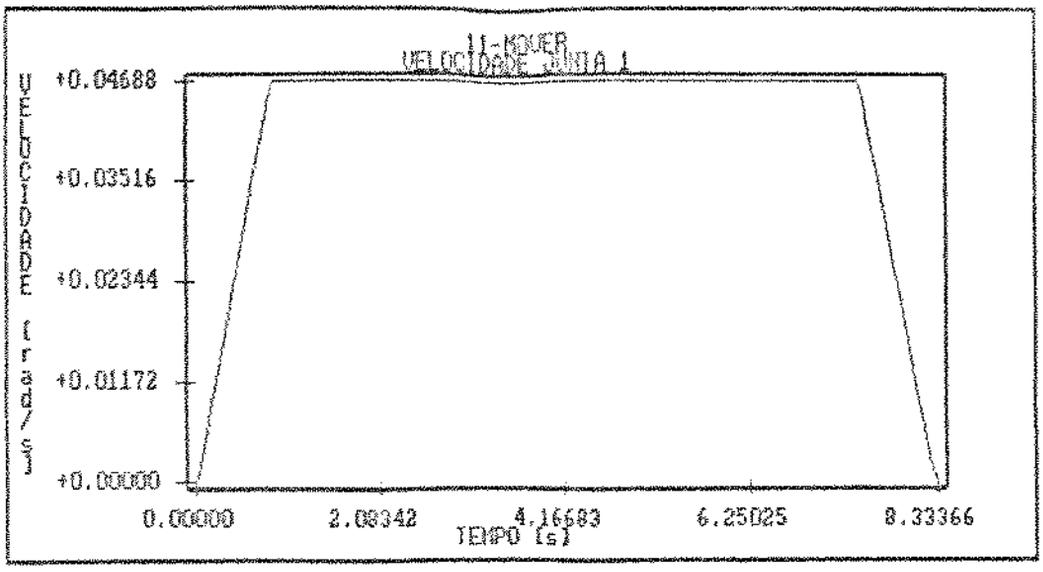

FIGURA 47 - VELOCIDADE DA JUNTA 1

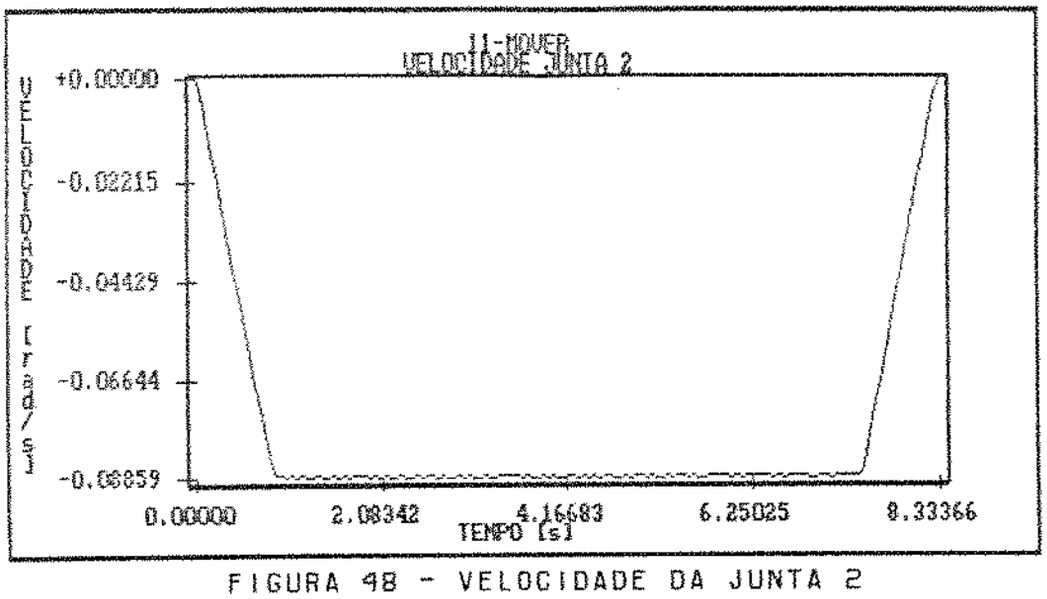

FIGURA 48 - VELOCIDADE DA JUNTA 2

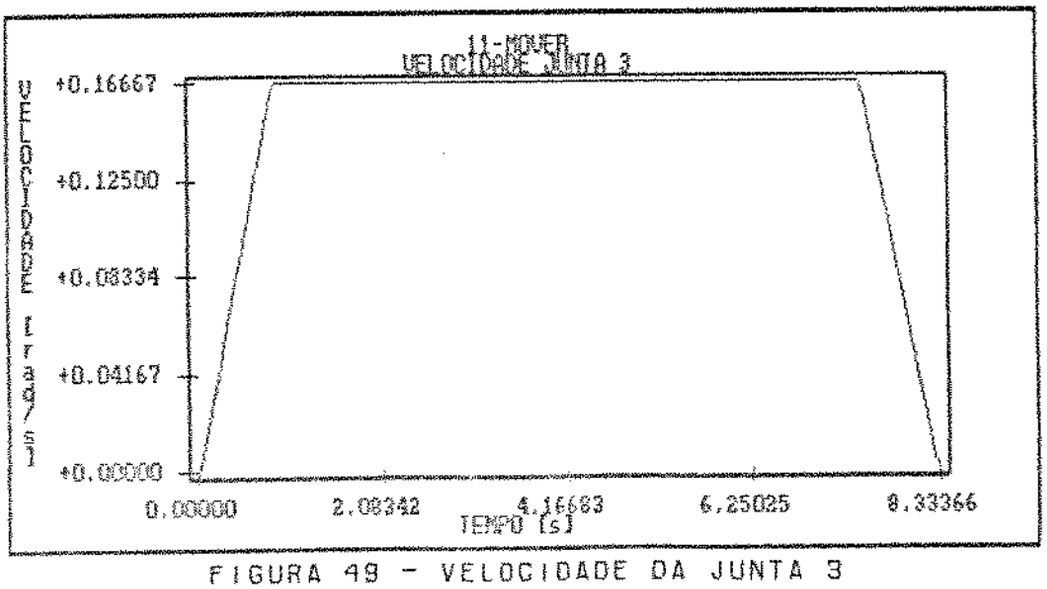

FIGURA 49 - VELOCIDADE DA JUNTA 3

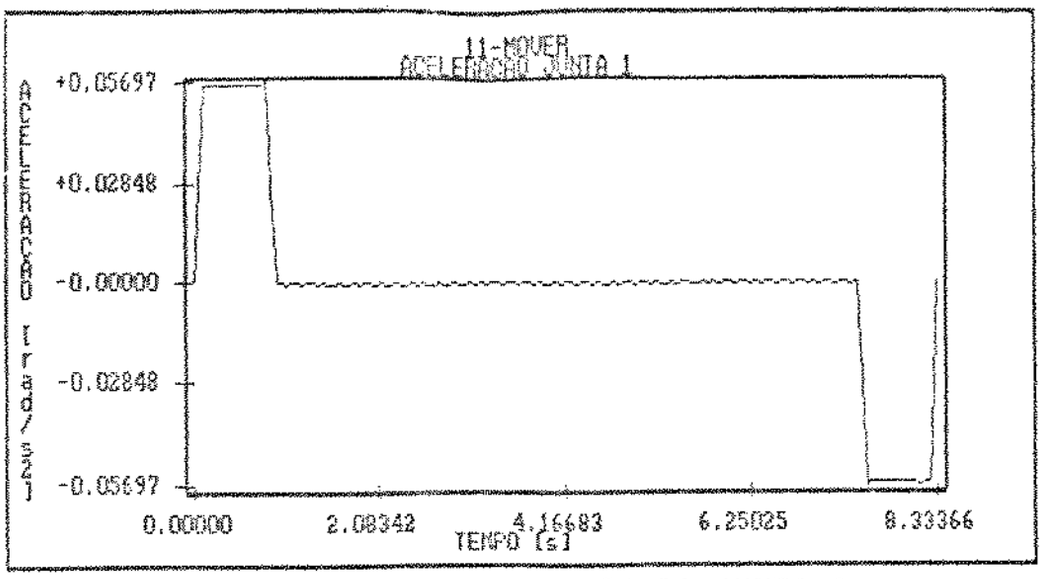

FIGURA 50 - ACALERAÇÃO DA JUNTA 1

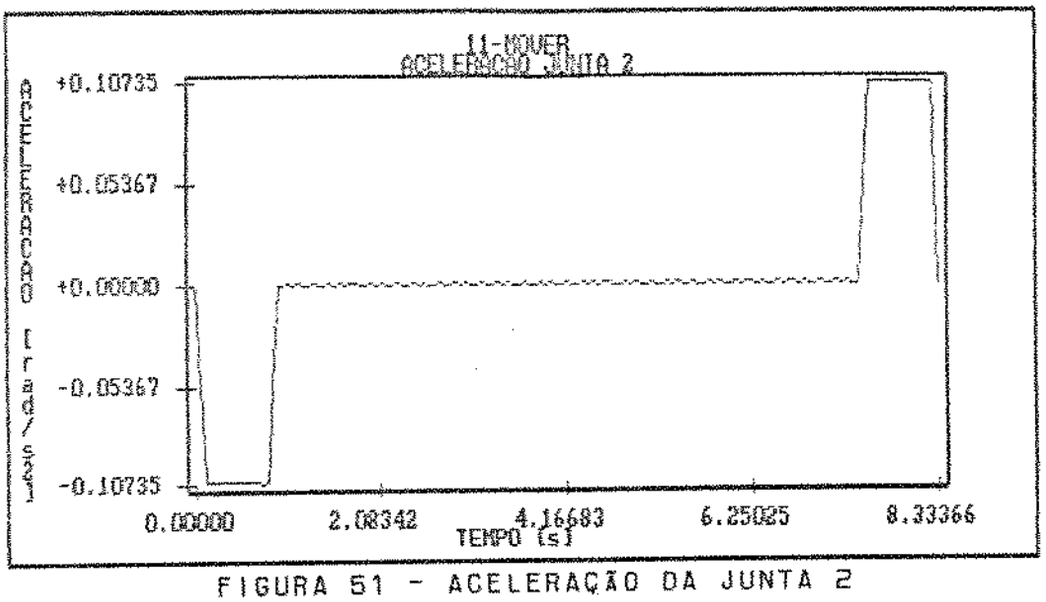

FIGURA 51 - ACELERAÇÃO DA JUNTA 2

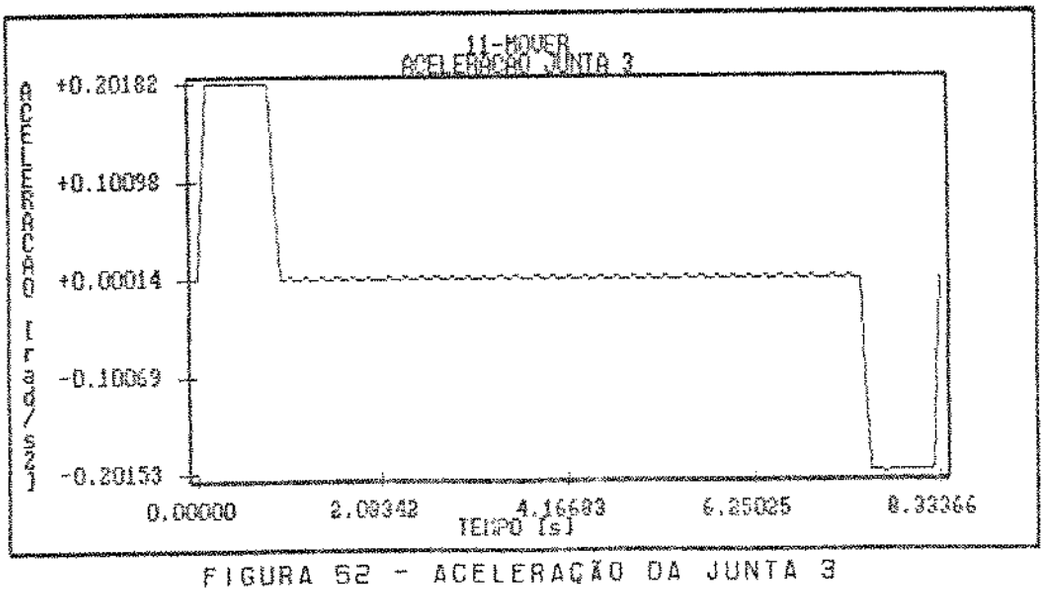

FIGURA 52 - ACELERAÇÃO DA JUNTA 3

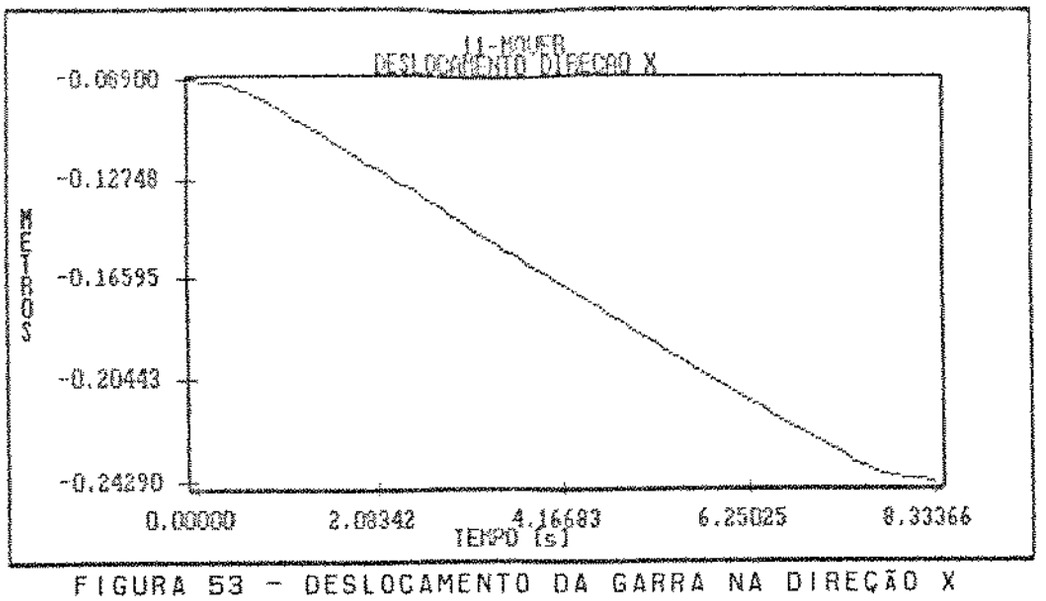

FIGURA 53 - DESLOCAMENTO DA GARRA NA DIREÇÃO X

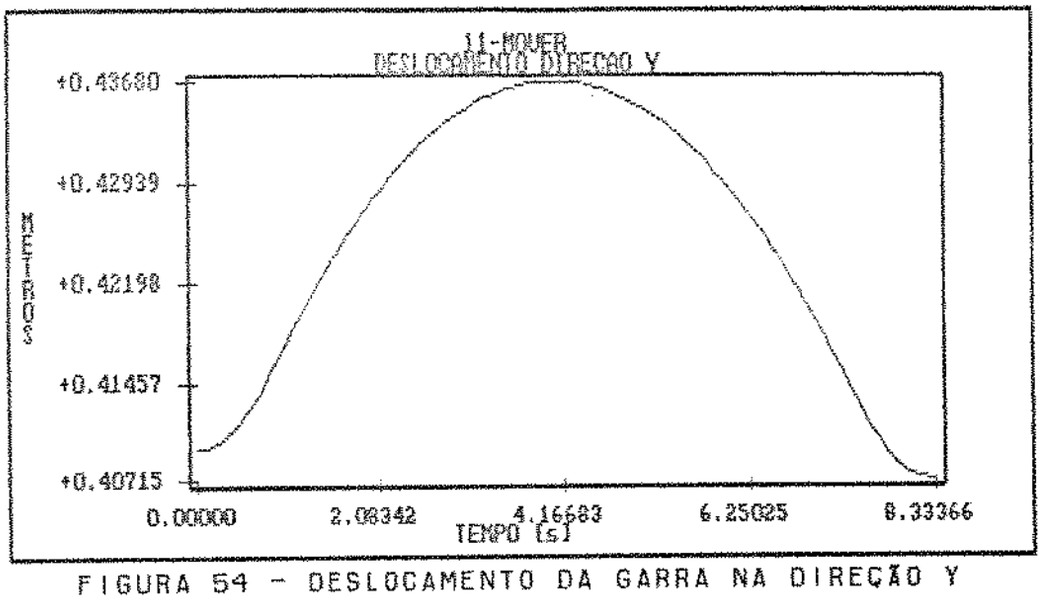

FIGURA 54 - DESLOCAMENTO DA GARRA NA DIREÇÃO Y

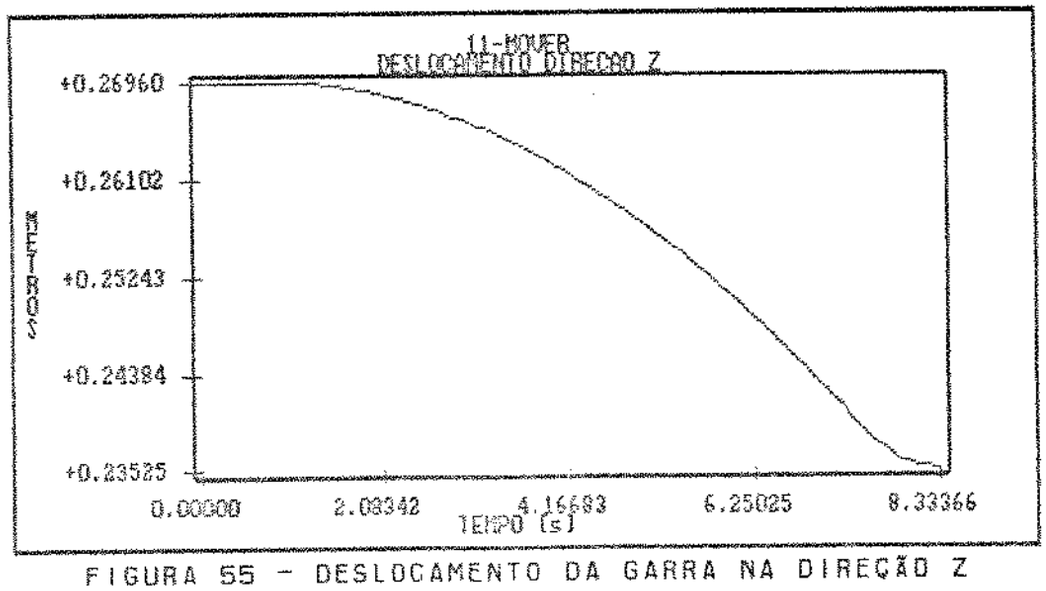

FIGURA 55 - DESLOCAMENTO DA GARRA NA DIREÇÃO Z

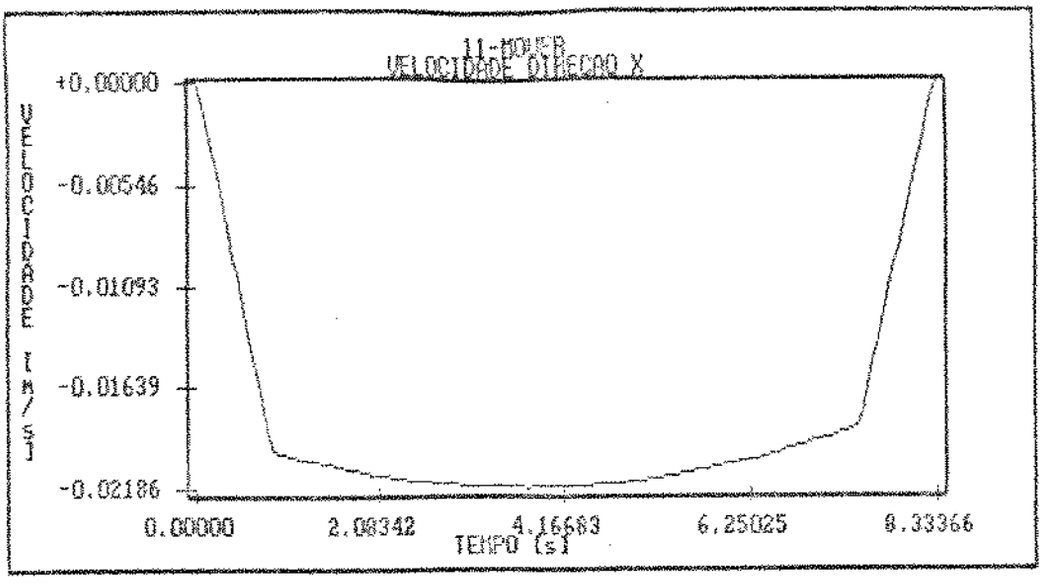

FIGURA 56 - VELOCIDADE DA GARRA NA DIREÇÃO X

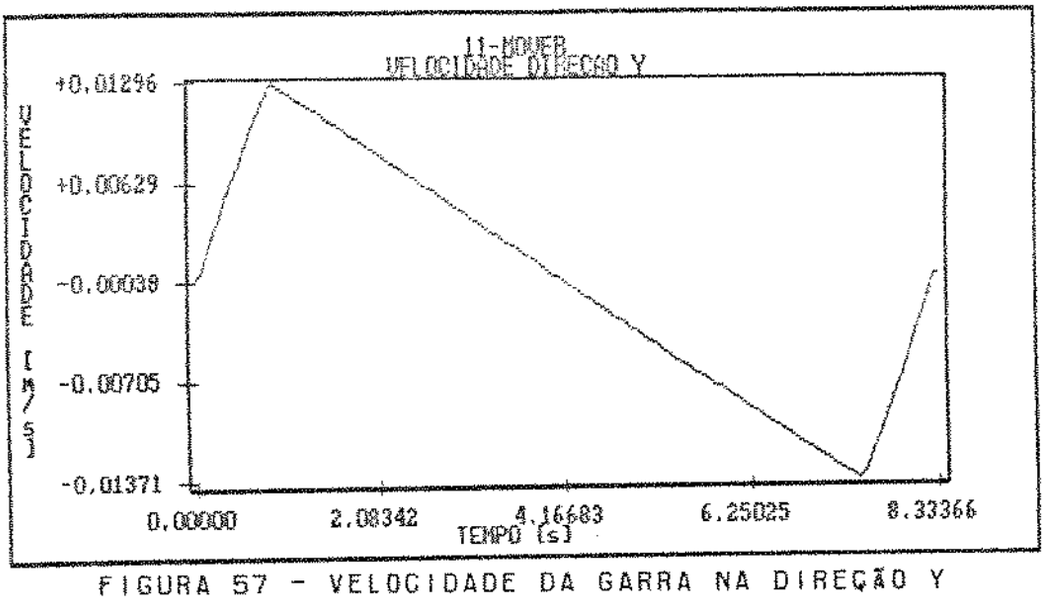

FIGURA 57 - VELOCIDADE DA GARRA NA DIREÇÃO Y

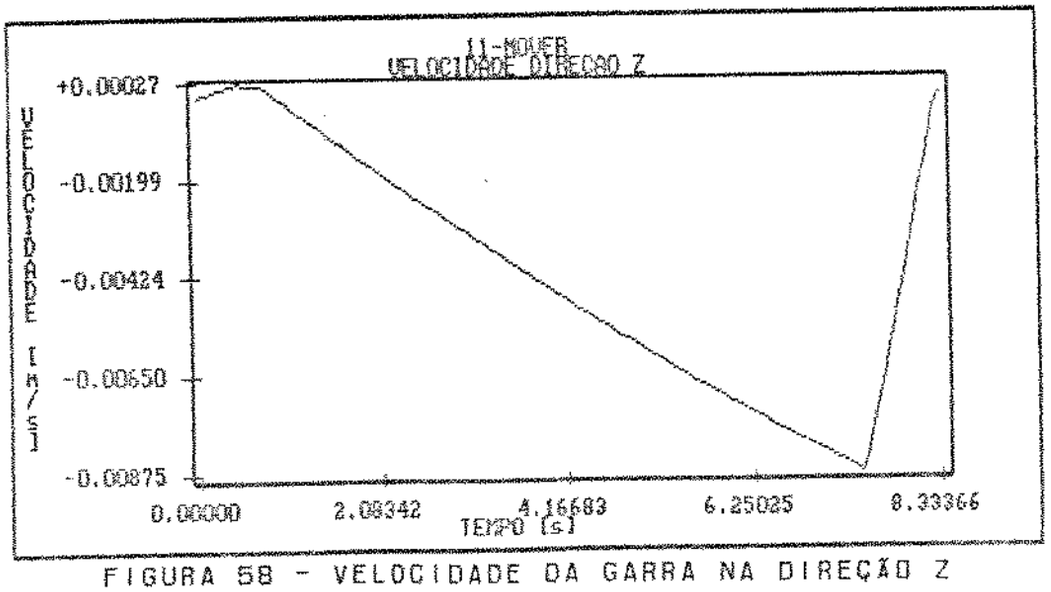

FIGURA 58 - VELOCIDADE DA GARRA NA DIREÇÃO Z

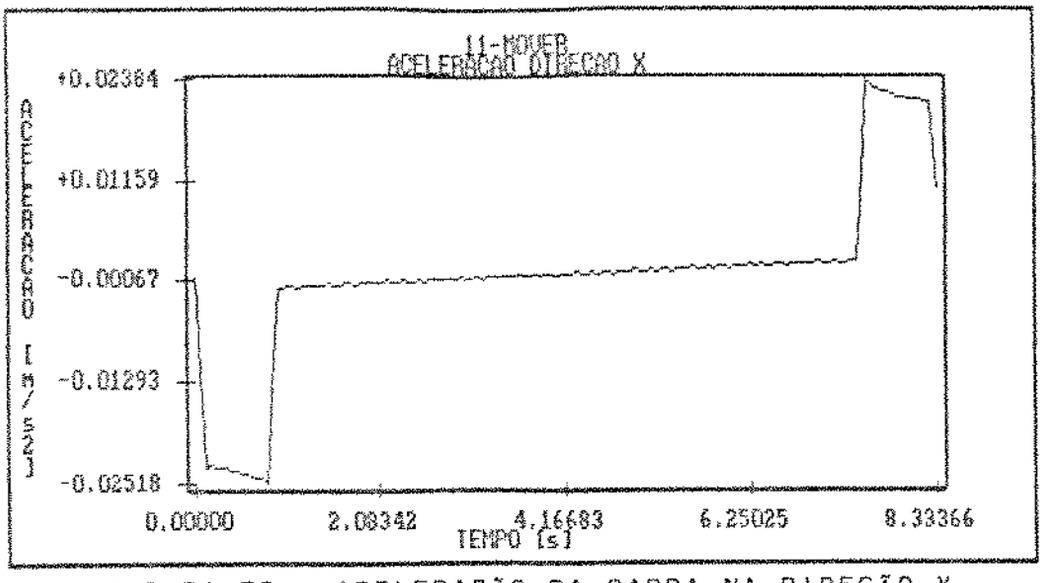

FIGURA 59 - ACELERAÇÃO DA GARRA NA DIREÇÃO X

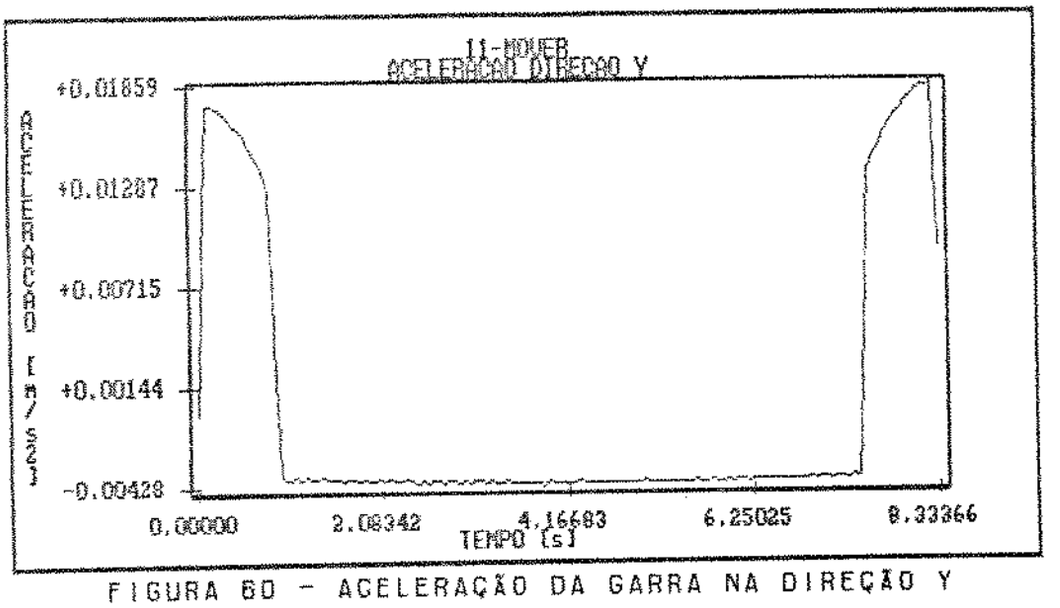

FIGURA 60 - ACELERAÇÃO DA GARRA NA DIREÇÃO Y

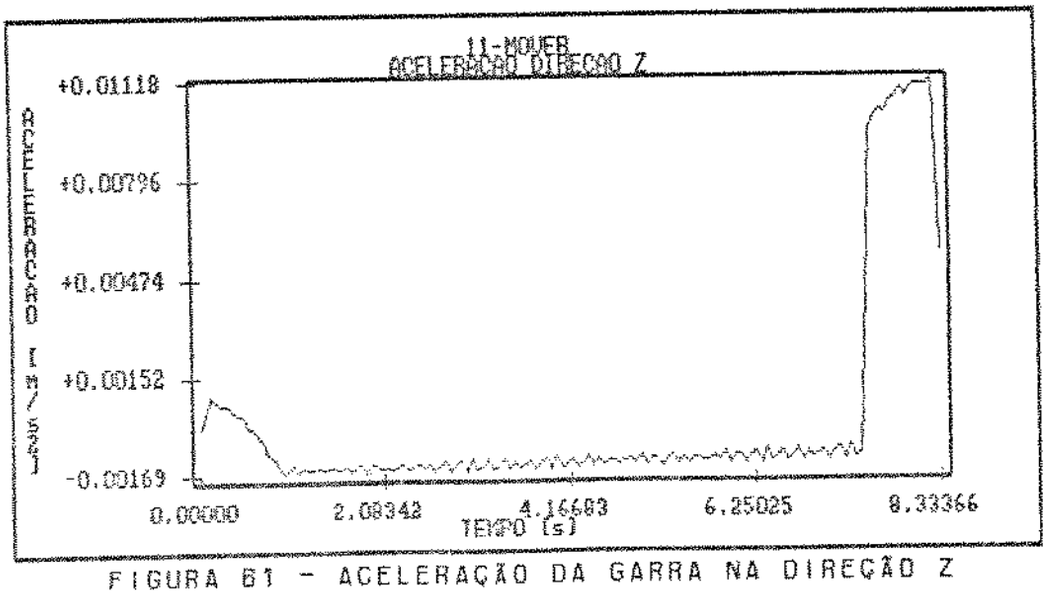

FIGURA 61 - ACELERAÇÃO DA GARRA NA DIREÇÃO Z

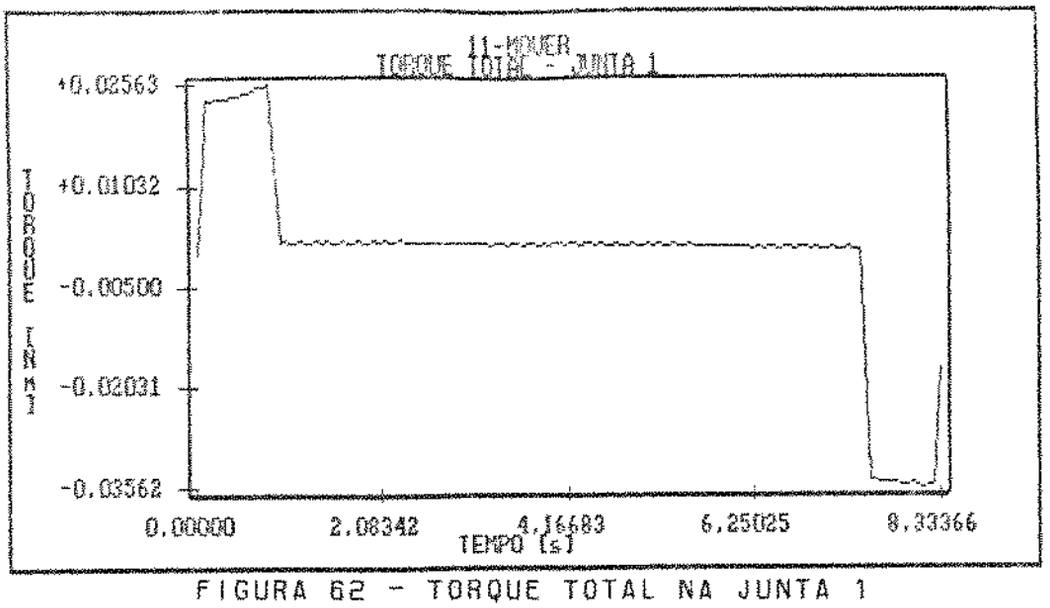

FIGURA 62 - TORQUE TOTAL NA JUNTA 1

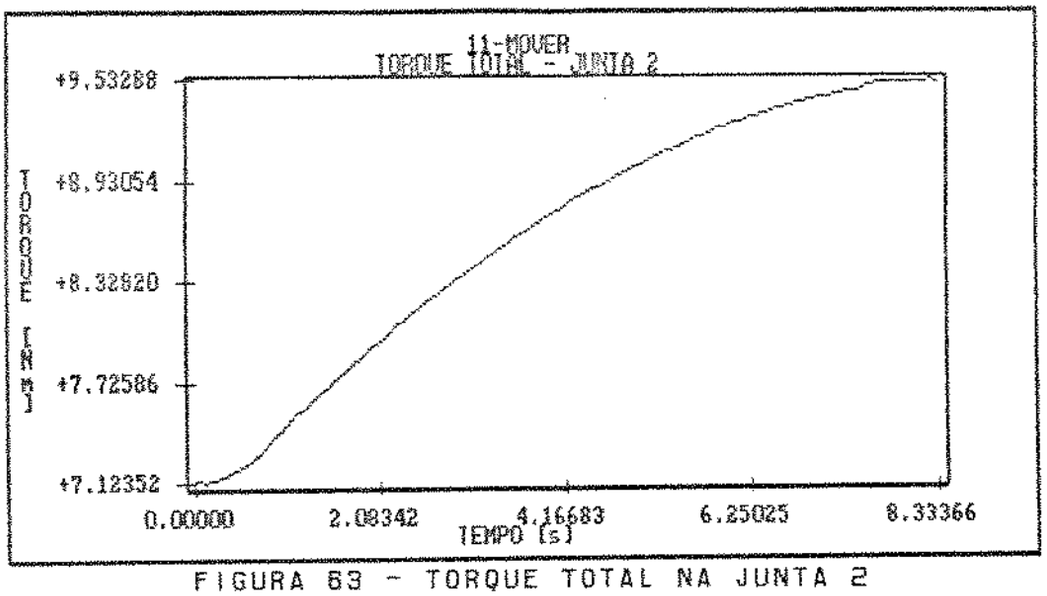

FIGURA 63 - TORQUE TOTAL NA JUNTA 2

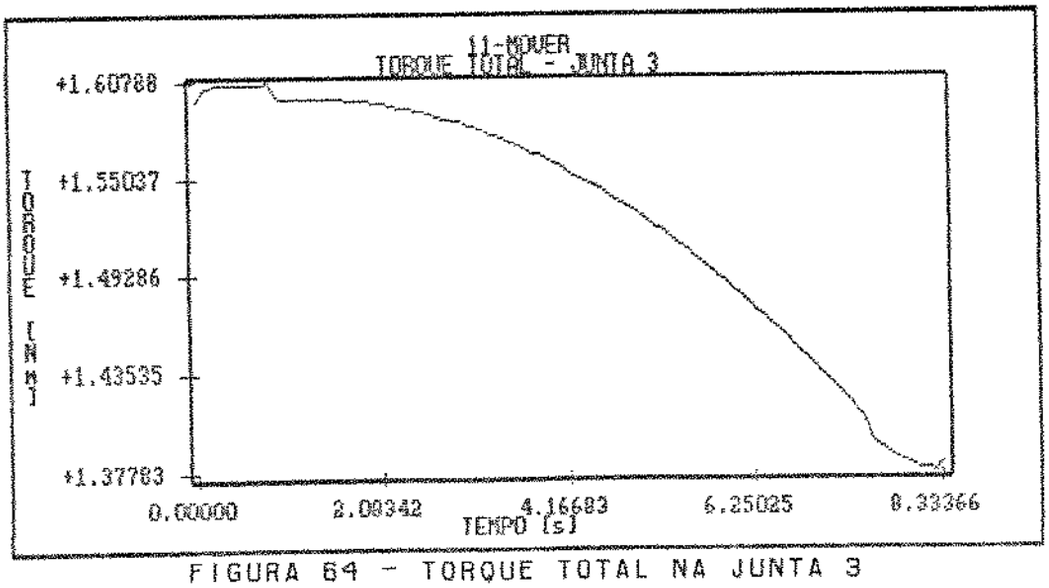

FIGURA 64 - TORQUE TOTAL NA JUNTA 3

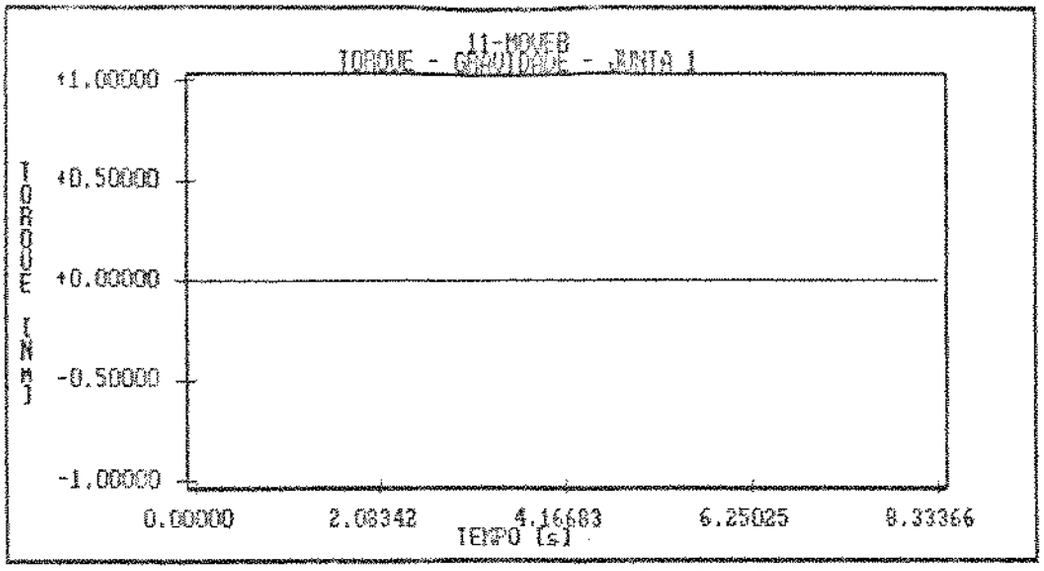

FIGURA 65 - TORQUE DEVIDO A COMPONENTE DE GRAVIDADE NA JUNTA 1

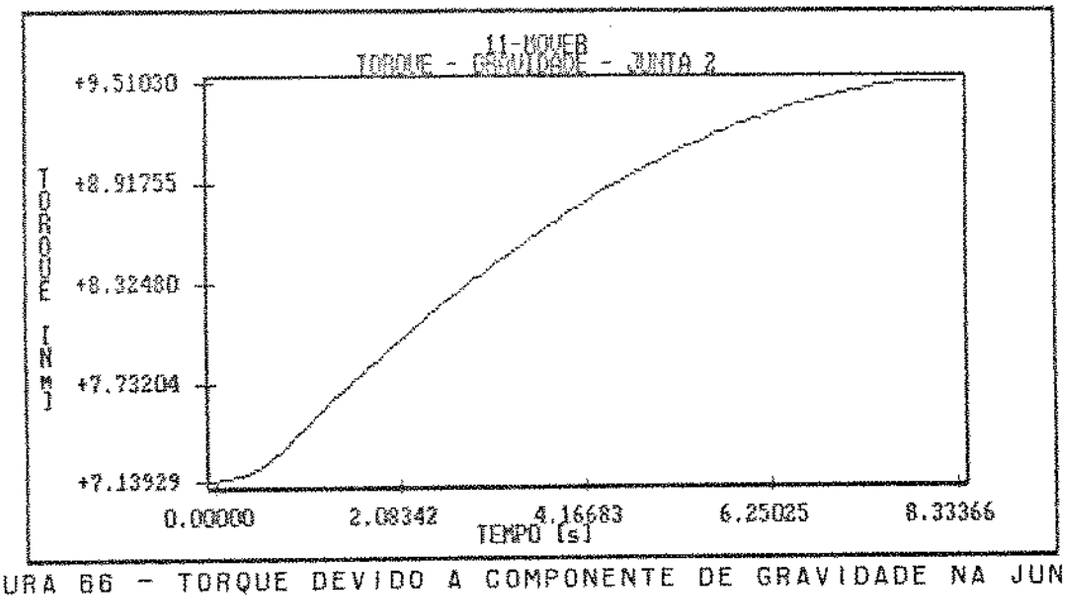

FIGURA 66 - TORQUE DEVIDO A COMPONENTE DE GRAVIDADE NA JUNTA 2

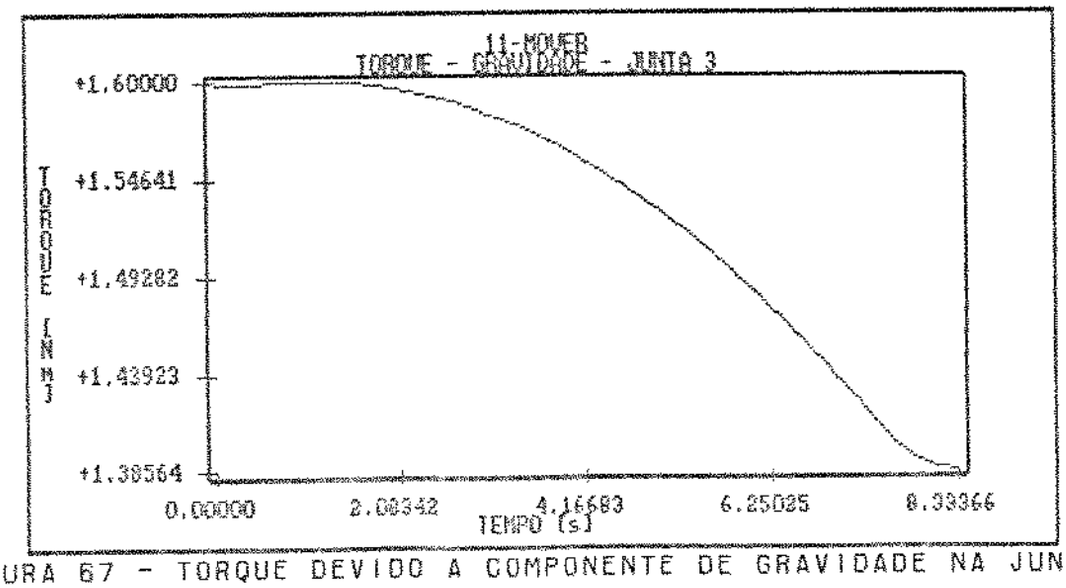

FIGURA 67 - TORQUE DEVIDO A COMPONENTE DE GRAVIDADE NA JUNTA 3

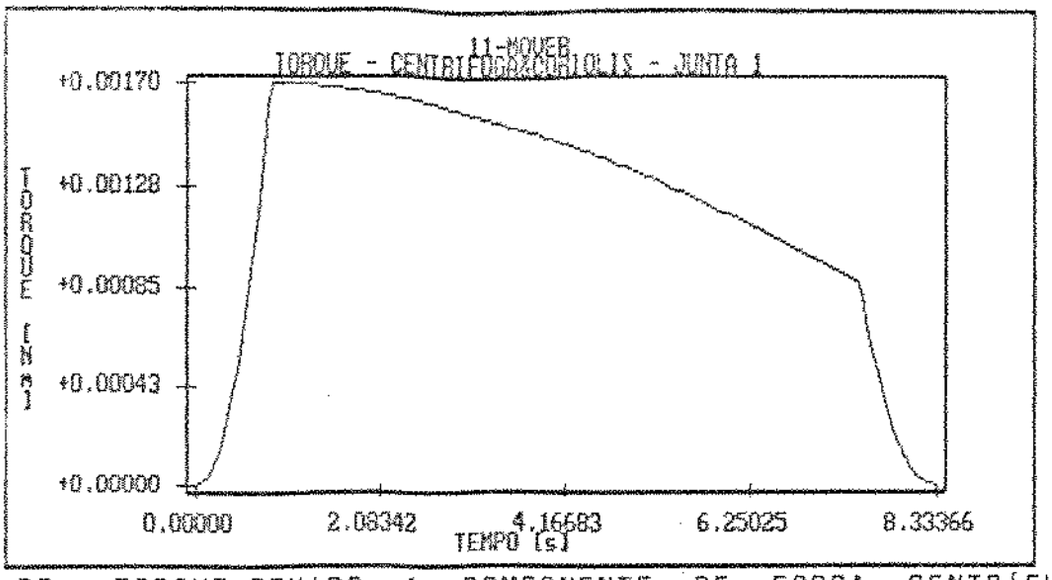

FIGURA 68 - TORQUE DEVIDO A COMPONENTE DE FORÇA CENTRÍFUGA E CORIOLIS NA JUNTA 1

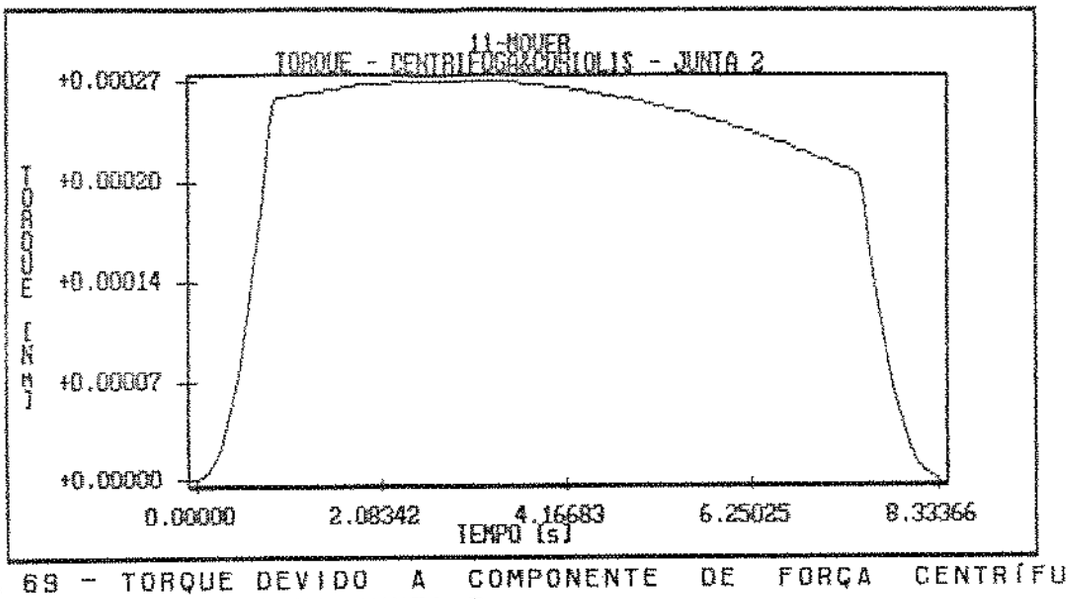

FIGURA 69 - TORQUE DEVIDO A COMPONENTE DE FORÇA CENTRÍFUGA E CORIOLIS NA JUNTA 2

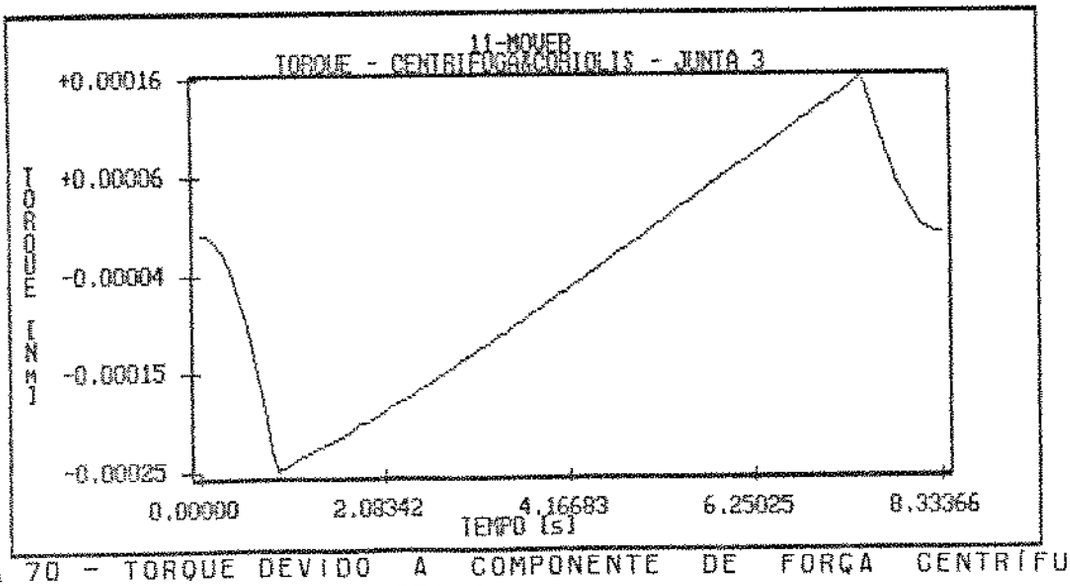

FIGURA 70 - TORQUE DEVIDO A COMPONENTE DE FORÇA CENTRÍFUGA E CORIOLIS NA JUNTA 3

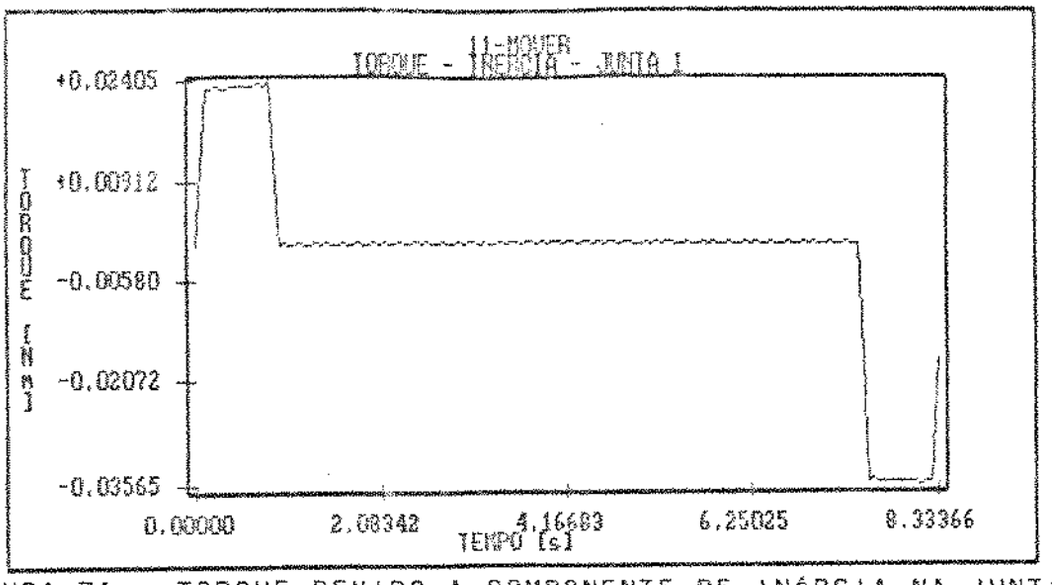

FIGURA 71 - TORQUE DEVIDO A COMPONENTE DE INÉRCIA NA JUNTA 1

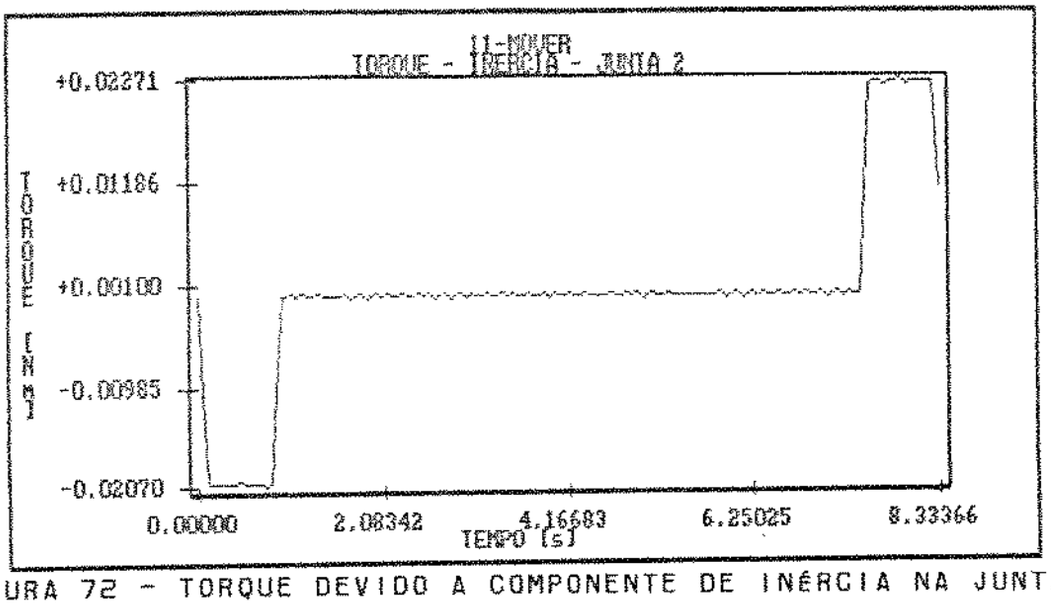

FIGURA 72 - TORQUE DEVIDO A COMPONENTE DE INÉRCIA NA JUNTA 2

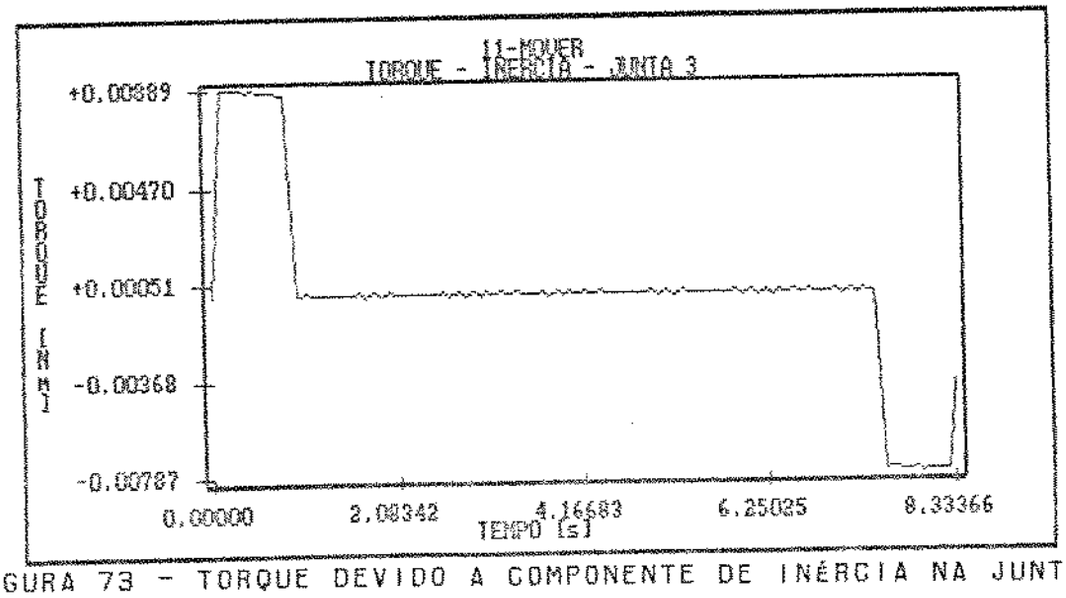

FIGURA 73 - TORQUE DEVIDO A COMPONENTE DE INÉRCIA NA JUNTA 3

PARTE E

CONCLUSÕES E PERSPECTIVAS

CONCLUSÕES E PERSPECTIVAS

O desenvolvimento do protótipo de um sistema de programação off-line para robôs manipuladores, possibilitou-nos a oportunidade de realizar testes com modelos matemáticos disponíveis na literatura e avaliar sua integração com uma interface homem-máquina padrão (linguagem de programação). Não se trata de mais uma tentativa de re-inventar a roda, mas sim, a necessidade de inicializar um trabalho de pesquisa nesta área, tendo como meta, dominar e gerar técnicas de confecção de software para preparação e simulação, off-line, de tarefas para robôs.

A concepção da linguagem de programação, permitindo a implementação de tarefas utilizando comandos de alto nível e por outro lado, dar a possibilidade de testes de novas estratégias através da programação em baixo nível, foi a opção adequada, tornando o acesso ao sistema de controle dos atuadores, transparentes ou não, ao usuário.

Especificou-se um número mínimo de comandos cobrindo um largo espectro de movimentos possíveis de serem realizados pelo robô. Sendo sua sintaxe baseada na linguagem "C" padrão [KERNIGH-78], permite a portabilidade para outro hardware (computador) "sem a necessidade de adaptações".

Apesar de possuir "canais" de realimentação com o sistema de controle, a linguagem não incorpora comandos e não tem arquitetura para interfaceamento com sensores e processamento concorrente em tempo real. Como há possibilidade da aplicação de robôs em tarefas que exija alguma "inteligência" de máquina (com o desenvolvimento de processadores e sensores eficientes e, de técnicas de programação), esses tipos de processamentos devem ser incorporados em futuras implementações. Tratando-se de uma linguagem para programação off-line, tal abordagem poderia ter sido incorporada ao sistema, contudo, optou-se por caminhos mais seguros, tendo em vista que a arquitetura interna da linguagem exigiria uma modelagem matemática e um hardware mais elaborados, além do que,ainda não se tinha experiência com modelos mais simples.

Quanto ao nome dos comandos, notou-se que não existe uma uniformização mínima, cada sistema utiliza seu próprio código. Seguindo a regra, adotou-se uma nomenclatura "nacionalizada" com comandos em português. Comparando os comandos de duas linguagens: VAL e AML, notou-se que apesar dos comandos apresentarem nomes e sintaxe diferentes, alguns aspectos funcionais são idênticos, como por exemplo, os comandos de movimento. Isso significa que existe um elo comum para a especificação de um conjunto básico de comandos com nome, sintaxe e função idênticas. A utilização de um código comum não é fundamental na programação e integração de

múltiplos robôs, contudo, tornaria as interfaces entres sistemas, mais simples.

Modelamento. Utilizou-se basicamente três modelos matemáticos para a implementação do sistema: modelo geométrico direto, modelo geométrico inverso e modelo dinâmico.

O modelo geométrico direto é o mais simples de ser definido e implementado, sendo aplicado na geração de movimentos do robô e no sistema de CAD-animação para a geração de imagens.

O modelo geométrico inverso merece algum comentário a mais que o apresentado anteriormente. O método proposto por PAUL, adotado nesta dissertação, é de fácil compreensão e aplicação, contudo, exige muita atenção na escolha das "equações corretas". Como as equações obtidas são não lineares e incorporam os vínculos físicos entre os links, procurou-se selecionar aquelas que se mostrassem menos sensíveis a degenerações numéricas para as posições e orientações assumidas pela garra. Nota-se que, as equações são do tipo:

$$\theta = f^{-1}(\text{NUM}() \;/\; \text{DEN}())$$

onde

$f^{-1}()$: uma função trigonométrica inversa,

NUM(): equação do numerador,

DEN(): equação do denominador.

Uma análise superficial logo indica que se NUM() → 0 ou NUM() → ±∞ ou DEN() → 0 ou DEN() → ±∞, simultaneamente ou não, podem surgir problemas numéricos para o correto cálculo de $f^{-1}()$. Esse tipo de problema foi encontrado na determinação das expressões para os ângulos θ_2 e θ_3, uma vez que o robô é redundante no plano formado pelas juntas 2, 3 e 4*. Das equações que permitiam esses cálculos, todas incorporavam degenerações numéricas. Assim, em função desses problemas, adotou-se uma solução numérica restrita ao cálculo de θ_2 e θ_3. Selecionou-se duas equações linearmente independentes: $f(\theta_2)$ e $f(\theta_3)$; e realizou-se uma expansão em série de Taylor (2ª ordem) dos termos em seno e cosseno para os ângulos θ_2 e θ_3, cuja solução é conhecida. As soluções resultantes são dependentes de um valor angular inicial, devendo estar próximo da solução desejada, que no caso pode ser desconhecido. Então, adotou-se o seguinte procedimento iterativo: assume-se o valor angular corrente e calcula-se o novo valor de θ, que por sua vez, é assumido como um "novo" valor inicial, e assim, sucessivamente até que se alcance um valor constante. Os resultados foram satisfatórios, reduzindo o problema inicial a áreas próximas da máxima fronteira física da área de trabalho, onde o braço está completamente estendido.

---

* Note que as juntas 2, 3 e 4 estão em um mesmo plano, o que provoca a redundância.

O modelo dinâmico implementado tem por objetivo uma avaliação teórica do perfil de torque das 3 primeiras juntas, a partir dos perfis de velocidade e trajetória selecionados. Aplicando um coeficiente de segurança ao torque calculado, de forma a representar as perdas eletro-mecânicas não incorporadas ao modelo, pode-se verificar se o torque disponível nos atuadores será ou não suficiente para realizar o movimento.

Pelas expressões apresentadas na parte B, conclui-se que o cálculo do modelo dinâmico completo (6 graus de liberdade) exigiria um grande esforço computacional. Após executar uma série de exemplos em condições críticas: máximo momento de inércia, máxima variação de deslocamento e velocidade ($\approx$ 0.1 rad/s compatíveis com a performance desejada para o projeto mecânico), pode-se dizer que para a junta 1, os termos de inércia são preponderantes e, para as juntas 2 e 3, os termos de gravidade são preponderantes. Assim, anulando os termos numericamente desprezíveis, define-se um equacionamento com reduzido número de termos, tornando computacionalmente viável sua implementação em malhas de realimentação para a geração de trajetórias seguindo um perfil de torque.

CAD-Simulação. A principal função do sistema de CAD-Simulação é permitir o acompanhamento da evolução dos movimentos do robô e dos objetos manipulados através de gráficos e animação gráfica. As grandes limitações para esse tipo de sistema são a velocidade de processamento, a capacidade de armazenamento de dados e a resolução gráfica do hardware eletrônico. Neste trabalho utilizou-se um computador IBM-PC-AT-12MHz com 1 Mbyte de memória, o que orientou a escolha dos modelos de esqueleto e sólidos para a representação gráfica do robô. Poder-se-ia ter utilizado técnicas que proporcionassem maior realismo do que o obtido, no entanto, o custo computacional seria muito alto, impossibilitando a geração da animação em "tempo real".

Finalizando, como perspectivas para trabalhos futuros, tem-se os seguintes tópicos:

- desenvolvimento de interface homem-máquina "mais amigável" através de pré-processadores "inteligentes", programação por menus, programação por tarefa e comando por voz,
- normalização de comandos para programação de robôs e protocolos de comunicação,
- desenvolvimento de modelamento geométrico, cinemático e dinâmico, de comandos e estratégias para implementação de tarefas em tempo real com computadores de processamento distribuído e concorrente,
- desenvolvimento de sistemas "inteligentes" para tomada de decisão e sincronização de tarefas entre robôs e máquinas de comando numérico,

PARTE F

REFERÊNCIAS

REFERÊNCIAS

[AHMAD-88] Ahmad, S. I.
"Programming, Task and Motion"
International Encyclopedia of Robotics-Applications and Automation, Editor R. C. Dorf, 1988
Ed. Wiley-Interscience Publication-John Wiley & Sons
Vol. 2, pág. 1260-1268

[AHO-86] Aho, A. V. ; Sethi, R. ; Ullman, J. D.
"Compilers: Principles, Techniques, and Tools"
Ed. Addison-Wesley Publishing Company, Inc. 1986

[AMBER-87] Amber, A. P. ; Cameron, S. A. ; Corner, D. F.
"Augmenting the RAPT Robot Language" in "Languages for Sensor-Based Control in Robotics"
Editores: U. Rembold ; K. Hormann
NATO ASI, Vol. F29, pág. 305-327
Ed. Springer-Verlag Berlin 1987

[ANGELES-88] Angeles, J. ; Alivizatos, A. ; Zsombor-Murray, P. J.
"The Synthesis of Smooth Trajetories for Pick-and-Place Operations"
IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-18, N# 1 , January/February 1988, pág. 173-178

[AKEN-84] Aken, L. V. ; Brussel, H. V.
"Software for Solving The Inverse Kinematic Problem for Robot Manipulators in Real Time"
Advanced Software in Robotics
Editores: A. Danthine and M. Géradin
Ed. Elsevier Science Publishers B. V. (Holland) 1984
pág. 159-174

[ARMSTRO-86] Armstrong, B. ; Khatib, O. ; Burdick, J.
"The Explicit Dynamic Model and Inercial Parameters of The PUMA 560 Arm"
IEEE International Conference on Robotics and Automation Proceeding 1986, pág. 510-518

[BENHABI-88] Benhabib, B. ; Zak, G. ; Tabarah, E.
"Position Control of Two-Arm Manipulators for Coordinated Point-to-Point Motion"
Journal of Robotic Systems, 5 (2), pág.103-124
Ed. John Wiley & Sons, Inc. 1988

[BENNETT-89] Bennett, D. J. ; Hollerbach, J. M.
"Identifying the Kinematics of Robots and Their Tasks"
IEEE International Conference on Robotics and Automation Proceeding 1989, pág. 580-586

[BESANT-83] Besant, C. B.
"Computer-Aided Design and Manufacture"
Ed. Ellis Horwood Ltd. 1983

[BISON-87] Bison, P. ; Pagello, E. ; Priolo, L. ; Zivlani, S.
"Simulation Tools as a Programming Aid for Robot Programming"
NATO ASI, Vol. F29, pág. 343-357
"Languages for Sensor-Based Control in Robotics"
Editores: U. Rembold ; K. Hormann
Ed. Springer-Verlag Berlin 1987

[BLUME-84] Blume, C. ; Jakob, W.
"Design of The Strutured Robot Language (SRL)"
Advanced Software In Robotics
Editores: A. Danthine and M. Géradin
Ed. Elsevier Science Publishers B. V. (Holland)
1984, pág. 127-143

[BOBROW-73] Bobrow, D. G. ; Wegbreit, B.
"A Model for Control Structures for Artificial Intelligence Programming Languages"
Third International Joint Conference on Artificial Intelligence, August 1973, Stanford University, pág. 246-253

[BRADY-83] Editores: Brady, M. ; Hollerbach, J. M. ; et al.
"ROBOT MOTION: Planning and Control"
Ed. MIT Press 1983

[BRAJNIK-90] Brajnik, G ; Guida, G. ; Tasso, C.
"User Modeling in Expert Man-Machine Interfaces: A Case Study in Intelligent Information Retrieval"
IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-20, N# 1 , January/February 1990, pág. 166-185

[CHANG-88] Chang, Y.-H. ; Lee, T.-T. ; Liu, C.-H.
"On-Line Cartesian Path Trajectory Planning for Robot Manipulators"
IEEE International Conference on Robotics and Automation Proceedings 1988, pág. 62-67

[CHARNIA-85] Charniak, E. ; McDermott, D.
"Introduction to Artificial Intelligence"
ED. Addison-Wesley Publishing Company, Inc. 1985

[CONNELL-89] Connell, J. H.
"A Behavior-Based Arm Controller"
IEEE Transactions on Robotics and Automation
Vol. 5, N# 6, December 1989, pág. 784-791

[CRAIG-86] Craig, J. J.
"Introduction to Robotics Mechanics & Control"
Ed. Addison-Wesley Publishing Company - 1986

[DENAVIT-55] Denavit, J. ; Hartenberg, R. S.
"A Kinematic Notation for Lower Pair Mechanisms Based on Matrices"
Journal of Applied Mechanics, June 1985,
pág. 215-221

[DILLMAN-90] Dillmann, R.
"Application of AI Methods in Robotics"
1° SAI - SIMPÓSIO DE AUTOMAÇÃO INTEGRADA - 1990
Curitiba - Paraná - Brasil

[DLABKA-89] Dlabka, M ; Held, J. ; Zhang, Y.
"A Practical Approach for Planning and Realization of Optimal Trajectories for Industrial Robots"
IFAC Robot Control - Proceeding Series 1989
N# 10, pág. 517-522

[DRIELS-88] Driels, M. R. ; Fan, U. J. ; Pathre, U. S.
"The Aplication of Newton-Euler Recursive Methods to The Derivation of Close Form Dynamic Equations"
Journal of Robotic Systems, 5(3), 1988, pág. 229-248

[FALLSID-89] Fallside, F. ; Jahanbin, M. R. ; Marsland, T. P. ; Tabandeh, A. S. ; Wright, M. W.
"A Prototype, Integrated, CAD-Based Robotic Assembly System"
IEEE International Conference on Robotics and Automation Proceedings 1989, pág. 228-234

[FEATHER-87] Featherstone, R.
"Robot Dynamics Algorithms"
Ed. Kluwer Academic Publishers 1987

[FROMMHE-87] Frommherz, B. ; Hoermann, K.
"A Concept for a Robot Action Planning System" in "Languages for Sensor-Based Control in Robotics"
Editores: U. Rembold ; K. Hormann
NATO ASI, Vol. F29, pág. 125-145
Ed. Springer-Verlag Berlin 1987

[FU-87] Fu, K. S. ; Gonzalez, R. C. ; Lee, C. S. G.
"ROBOTICS: Control, Sensing, Vision, and Intelligence"
Ed. McGraw-Hill Book Company 1987

[GIOLI-78] Gioll, W. K.
"Iterative Computer Graphics Data Structures Algorithms Languages"
Ed. Prentice-Hall, Inc. 1978

[GOLDENB-87] Goldenberg, A. A. ; Apkarian, J. A. ; Smith, H. W.
"A New Approach to Kinematic Control of Robot Manipulators"
Transctions of The ASME Journal of Dynamic Systems, Measurement, and Control, Vol. 109, June 1987, pág. 97-103

[GOMAA-88] Gomaa, H.
"Programming of Multiple Robot Systems"
International Encyclopedia of Robotics-Applications and Automation, Editor R. C. Dorf, 1988
Ed. Wiley-Interscience Publication-John Wiley & Sons
Vol. 2, pág. 1250-1260

[GOOD-85] Good, M. C. ; Sweet, L.M. ; Strobel, K. L.
"Dynamic Models for Control System Design of Integrated Robot and Drive-Systems"
Transactions of The ASME Journal of Dynamic Systems, Measurement, and Control, Vol. 107, March 1985, pág. 53-59

[GOUZENE-84] Gouzenes, L.
"Strategies for Solving Collision-Free Trajectories Problems for Mobile and Manipulator Robots"
The International Journal of Robotics Research
Vol. 3, N# 4, Winter 1984, pág. 51-65

[GRAHAM-87] Graham, J. H. Editor
"Computer Arquitectures for Robotics and Automation"
Ed. Gordon and Breach Science Publishers S. A.

[GROSSMA-78] GROSSMAN, D. D. ; Taylor, R. H.
"Interactive Generation of Object Models with a Manipulator"
IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-8, N# 9 , September 1978, pág. 667-678

[GRUVER-84] Gruver, W. A. ; Soroka, B. I. ;
Craig, J. J. ; Turner, T. L.
"Industrial Robot Programming Languages: A Comparative Evaluation"
IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-14, N# 4, July/August 1984, pág. 565-570

[HEARN-85] Hearn, A. C.
"REDUCE User's Manual"
Ed. RAND Publication 1985

[HOLLERB-80] Hollerbach, J. M.
"A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity"
IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-10, N# 11, November 1980, pág. 111-117

[ITAUTEC-88] Itautec Informatica S.A.
"Sistema Operacional SISNE plus - Manual de Referência" Versão 3.30

[JONES-88] Page-Jones, M.
"Projeto Estruturado de Sistemas"
Ed. McGraw-Hill 1988

[KATO-82] Kato, I.
"Mechanical Hands Illustrates"
Editor of English Version: Kuni Sadamoto
Survey Japan, 1982

[KAMEYAM-89] Kameyama, M. ; Matsumoto, T. ; Egami, H. ; Higuchi, T.
"Implementation of a High Performance LSI for Inverse Kinematics Computation"
IEEE International Conference on Robotics and Automation Proceeding 1989, pág. 757-762

[KAPLAN-82] Kaplan, W.
"Cálculo Avançado" Vol. II
Ed. Edgard Blucher Ltda. 1982

[KELLEY-84] Kelley, Al ; Pohl, I.
"A Book on C"
ED. The Benjamin/Cummings Publishing Company, Inc. 1984

[KEMPER-87] Kemper, A. ; Wallrath, M. ; Lockemann, P. C.
"Database Support for Robotics Applications" In "Languages for Sensor-Based Control in Robotics"
Editores: U. Rembold ; K. Hormann
NATO ASI, Vol. F29, pág. 283-303
Ed. Springer-Verlag Berlin 1987

[KERNIGH-78] Kernighan, B. W. ; Ritchie, D. M.
"The C Programming Language"
Ed. Prentice-Hall, New Jersey 1978

[KIM-85] Kim, B. K. ; Kang, G. S.
"Minimum-Time Path Planning for Robot Arms and Their Dynamics"
IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-15, N# 2, March/April 1985, pág. 213-223

[KLEIN-83] Klein, C. A. ; Huang, Ching-Hsiang
"Review of Pseudoinverse Control for Use with Kinematically Redundant Manipulators"
IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-13, N# 3 , March/April 1983, pág. 245-250

[KOPACEK-88] Kopacek, P. ; Troch, I.
"Software for Simulation and Control of Robots and Manipulators"
IFAC Software for Computer Control 1988
Editores: I. M. MacLeod and A. D. Heher
N# 2, pág. 119-126

[KORITES-81] Korites, B. J.
"Graphic Software for Microcomputers"
Ed. Kern Publications 1981

[KOZLOWS-89] Kozlowski, K.
"POLROB-A Manipulator Level Programming Language"
IFAC Robot Control - Proceeding Series 1989
N# 10, pág. 453-458

[KUNG-89] Kung, S.-Y. ; Hwang, J.-N.
"Neural Network Architectures for Robotic Applications"
IEEE Transactions on Robotics and Automation
Vol. 5, N# 5, October 1989, pág. 641-657

[KUSIAK-88] Kusiak, A.
"Programming, Off-Line Languages"
International Encyclopedia of Robotics-Applications and Automation, Editor R. C. Dorf, 1988
Ed. Wiley-Interscience Publication-John Wiley & Sons
Vol. 2, pág. 1235-1250

[KYRIAKO-88] Kyriakopoulos, K. J. ; Saridis, G. N.
"Minimum Jerk Path Generation"
IEEE International Conference on Robotics and Automation Proceeding 1988, pág. 364-369

[LANGRUD-87] Langrudi, S. A. ; Ochs, J. B.
"A Graphic Simulator of The Off-Line Programming Language VAL-II" in "Off-Line Programming of Industrial Robots "
Editores: A. Storr, J. F. McWaters
Ed. Elsevier Science Publishers B. V. (N-Holland)
IFIP 1987, pág. 119-135

[LATOMBE-84] Latombe, J. C.
"Automatic Synthesis of Robot Programs from CAD Specifications"
NATO ASI Series, Vol.F11
Robotics and Artificial Intelligence
Edited by M. Brady et al.
Ed. Springer-Verlag Berlin Heidelberg 1984

[LEAKE-89] Leake, S. ; Green, T. ; Cofer, S. ; Sauerwein, T.
"Hierarchical Ada Robot Programming System (HARPS): A Complete and Working Telerobot Control System Based on The NASREM Model"
IEEE International Conference on Robotics and Automation Proceeding 1989, pág. 1022-1028

[LEE-82] Lee, C. S. G.
"Robot Arm Kinematics Dynamics and Control
IEEE COMPUTER, December 1982, pág. 62-79

[LEE-87] Lee, C. S. G. ; Lee, B. H.
"Development of Generalized d'Alembert Equations of Motion for Robot Manipulators"
IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-17, N# 2, March/April 1987, pág. 311-325

[LEVAS-89] Levas, A. ; Jayaraman, R.
"WADE: An Object-Oriented Environment for Modeling and Simulation of Workcell Applications"
IEEE Transactions on Robotics and Automation, Vol. 5, N# 3, June 1989, pág. 324-336

[LIÉGEOIS-87] Liégeois, A.
"Simulation as a Programming Aid" in "Languages for Sensor-Based Control in Robotics"
Editores: U. Rembold ; K. Hormann
NATO ASI, Vol. F29, pág. 331-341
Ed. Springer-Verlag Berlin 1987

[LIEBERM-77] Lieberman, L. I. ; Wesley, M. A.
"AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly"
IBM J. Res. Develop., July 1977, pág. 321-333

[LIN-83a] Lin, Chun-Shin ; Chang, Po-Rong
"Joint Trajectories of Mechanical Manipulators for Cartesian Path Approximation"
IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-13, N# 6 , November 1983, pág. 1094-1102

[LIN-83b] Lin, Chun-Shin ; Chang, Po-Rong ; Luh, J. Y. S.
"Formulation and Optimization of Cubic Polynomial Joint Trajectories for Industrial Robots"
IEEE Transactions on Automatic Control, Vol. AC-28, N# 13, December 1983, pág. 1066-1073

[LOZANO-83] Lozano-Pérez, T.
"Robot Programming"
Proceedings of the IEEE, Vol. 71, N# 7, July 1983, pág. 821-841

[LUH-83] Luh, J. Y. S.
"Conventional Controller Design for Industrial Robots - A Tutorial"
IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-13, N# 3, May/June 1983, pág. 298-316

[LUH-84] Luh, J. Y. S. ; Lin, C. S.
"Approximate Joint Trajectories for Control of Industrial Robots Along Cartesian Paths"
IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-14, N# 3, May/June 1984, pág. 444-450

[MACIEJE-89] Maciejewski, A.
"Kinetic Limitations on the Use of Redundancy in Robotic Manipulators"
IEEE International Conference on Robotics and Automation Proceeding 1989, pág. 113-118

[MACK-89] Mack, B. ; Bayoumi, M. M.
"Design and Integration of New Software for The Robot Controller Test Station"
IEEE International Conference on Robotics and Automation Proceeding 1989, pág. 866-873

[MALCOLM-88] Malcolm Jr., D. R.
"ROBOTICS: An Introduction"
2° Edição 1988
Ed. Delmar Publishers, Inc.

[McDONAL-86] McDonald, A. C.
"Robot Technology - Theory Design and Applications"
Ed. Prentice - Hall New Jersey 1986

[MENDELE-89] Mendeleck, A. ; Zampieri, D. E.
"Software para Equacionamento Cinemático e Dinâmico Simbólico de Robôs Manipuladores"
ANAIS - X COBEM Congresso Brasileiro de Eng. Mecânica, Dezembro 1989, Rio de Janeiro, Brasil
pág. 455-457

[MENDELE-90] Mendeleck, A. ; Zampieri, D. E.
"Desenvolvimento de um Controle Supervisor para um Manipulador Antropomórfico com 6 graus de Liberdade", pág 170-175
ANAIS - 1° SAI - SIMPÓSIO DE AUTOMAÇÃO INTEGRADA - 1990 - Curitiba - Paraná - Brasil

[MERIAN-81] Meriam, J. L.
"Dynamics"
Ed. John Wiley & Sons 1981

[MICROSO-86] MICROSOFT WINDOWS Guide d'Utilisation
MICROSOFT Corporation 1986

[MILBERG-89] Milberg, J. ; Schrufer ; Tauber, A.
"Requirements for Advanced Graphic Robot Programming System"
IFAC Robot Control - Proceeding Series 1989
N# 10, pág. 487-492

[MOORING-89] Mooring, B. W. ; Padavala, S. S.
"The Effect of Kinematic Model Complexity on Manipulator Accuracy"
IEEE International Conference on Robotics and Automation Proceeding 1989, pág. 593-598

[MORGAN-84] Morgan, C. L.
"Bluebook of Assembly Routines for The IBM PC & XT"
Ed. The Waite Group, Inc. 1984

[MYERS-84] Myers, R. E.
"Microcomputer Graphics for The IBM PC"
Ed. Addison-Wesley Publishing Company, Inc. 1984

[NAGATA-73] Nagata, T. ; Yamazaki, M. ; Tsukamoto, M
"Robot Planning System Based on Problem Solvers"
Third International Joint Conference on Artificial Intelligence, August 1973, Stanford University, pág. 388-395

[NAGY-87] Lovass-Nagy, V. ; Schilling, R. J.
"Control of Kinematically Redundant Robots Using (1)-Inverses"
IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-17, N# 4 , July/August 1987, pág. 644-649

[NAKAMUR-86] Nakamura, Y. ; Hanafusa, H.
"Inverse Kinematic Solutions with Singularity Robustness for Robot Manipulator Control"
Transactions of The ASME Journal of Dynamic Systems, Measurement, and Control, Vol. 108, September 1986, pág. 163-171

[NAKAMUR-87] Nakamura, Y. ; Hanafusa, H.
"3D Autonomous Trajectory Control of Robot Manipulators"
Robotics & Computer-Integrated Manufacturing, Vol. 3, N# 4, 1987, pág. 395-408

[NATO-87] Report of the Working group
"Robot Programming Languages" in "Languages for Sensor-Based Control in Robotics"
Editores: U. Rembold ; K. Hormann
NATO ASI, Vol. F29, pág. 601-611
Ed. Springer-Verlag Berlin 1987

[NEUMAN-87] Neuman, C. P. ; Murray, J. J.
"The Complete Dynamic Model and Customized Algorithms of The Puma Robot"
IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-17, N# 4, July/August 1987, pág. 635-643

[NILSSON-80] Nilsson, N. J.
"Principles of Artificial Intelligence"
Ed. Tioga Publishing Co. 1980

[NNAGY-86] N-Nagy, F. ; Siegler, A.
"Engineering Foundations of Robotics"
Ed. Prentice-Hall International 1986

[NNAJI-86] Nnaji, B. O.
"Computer-Aided Design, Selection and Evaluation of Robots" Vol. 2 1986
Ed. Elsevier Science Publishing Company, Inc.

[NORCIO-89] Norcio, A. F. ; Stanley, J.
"Adaptive Human-Computer Interfaces: A literature Survey and Perspective"
IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-19, N# 2 , March/April 1989, pág. 399-408

[ORIN-84] Orin, D. E. ; Schrader, W. W.
"Efficient Computation of The Jacobian for Robot Manipulators"
The International Journal of Robotics Research Vol. 3, N# 4, Winter 1984, pág. 66-75

[PAI-89] Pai, D. K. ; Leu, M. C.
"Generic Singularities of Robot Manipulators"
IEEE International Conference on Robotics and Automation Proceeding 1989, pág. 738-744

[PARKER-89] Parker, J. K. ; Goldberg, D. E.
"Inverse Kinematics of Redundant Robots Using Genetic Algorithms"
IEEE International Conference on Robotics and Automation Proceeding 1989, pág. 271-276

[PAUL-81] Paul, R. P. ; Shimano, B. ; Mayer, G. E.
"Differencial Kinematic Control Equations for Simple Manipulators"
IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-11, N# 6 , June 1981, pág. 456-460

[PAUL-83] Paul, R. P.
"Robot Manipulators: Matematics, Programming, and Control of Robot Manipulators"
Ed. MIT Press 1983

[POCOCK-89] Pocock, G.
"A Distributed, Real-Time Programming Language for Robotics"
IEEE International Conference on Robotics and Automation Proceeding 1989, pág. 1010-1015

[POLLMAN-87] Pollmann, W. ; Dzembritzki, H.
"Off-Line Programming of Industrial Robots (IR)"
Off-Line Programming of Industrial Robots
Editores: A. Storr, J. F. McWaters
Ed. Elsevier Science Publishers B. V. (N-Holland)
IFIP 1987, pág. 7-15

[POST-90] Post, S. ; Sage, A. P.
"An Overview of Automated Reasoning"
IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-20, N# 1 , January/February 1990, pág. 202-224

[PRITSCH-87] Pritschow, G. ; Storr, A. Gruhler, G. ; Schumacher, H.
"Off-Line Programming System with Geometrical Data Recording by Manually Guided Industrial Robot"
Off-Line Programming of Industrial Robots
Editores: A. Storr, J. F. McWaters
Ed. Elsevier Science Publishers B. V. (N-Holland)
IFIP 1987, pág. 53-68

[PRUENTE-87] Pruente E. A. ; Balaguer C. ; Berrientos A.
"LRS: A High Level Explicit Programming Language for Sensor-Based SCARA Type Robot"
NATO ASI, Vol. F29
"Languages for Sensor-Based Control in Robotics"
Editores: U. Rembold ; K. Hormann
Ed. Springer-Verlag Berlin 1987

[PURDUM-84] Purdum, J. J. ; Leslie, T. C. ; Stegemoller, A. L.
"C Programmer's Library"
QUE Corporation - Indianapolis, 1984

[RANKY-85] Ranky, P. G. ; Ho, C. Y.
"Robot Modelling Control and Applications with Software"
Ed. I.F.S. Ltd 1985

[RAYNA-87] Rayna, G.
"REDUCE Software for Algebraic Computation"
Serie Symbolic Computation - AI
Ed. Springer-Verlag N. Y., Inc. 1987

[RED-87] Red, W. E. ; Troung-Cao, H. V. ; Kim, K. H.
"Robot Path Planning in Three-Dimensions Using The Direct Subspace"
Transactions of The ASME Journal of Dynamic Systems, Measurement, and Control, Vol. 109, September 1987
pág. 238-244

[REMBOLD-87] Rembold, U.
"Programming of Industrial Robots - Today and in the Future" in "Languages for Sensor-Based Control in Robotics"
Editores: U. Rembold ; K. Hormann
NATO ASI, Vol. F29, pág. 1-23
Ed. Springer-Verlag Berlin 1987

[REQUICH-80] Requicha, A. A.
"Representations for Rigid Solids: Theory, Methods, and Systems"
Computing Surveys, Vol. 12, N# 4, December 1980
pág. 437-464

[SAKAUE-87] Sakaue, S. ; Sugimoto, K.
"Path Generation by Rate Control of Manipulators"
Robotics & Computer-Integrated Manufacturing
Vol. 3, N# 4, 1987, pág. 381-387

[SANDOR-85] Sandor, J.
"Octree Data Structures and Perspective Imagery"
Comput. & Graphics, Vol. 9, N# 4, 1985, pág. 393-405

[SARIDIS-83] Saridis, G. N.
"Intelligent Robotic Control"
IEEE Transactions on Automatic Control, Vol. AC-28, N# 5, May 1983, pág. 547-557

[SARIDIS-85] Saridis, G. N. editor
"Advances in Automation and Robotics"
Ed. J.A.I. Press, Inc. 1985

[SCHILDT-87] Schildt, H.
"Artificial Intelligence Using C"
ED. McGraw-Hill, Inc. 1987

[SHEN-85] Shen, H. C. ; Signarowski, G. F. P.
"A Knowledge Representation for Roving Robots"
IEEE 1985 CAIA pág. 621-628

[SHEN-87] Shen, H. C. ; Wong, A. K. C.
"A Model for Robot Programming and Control System"
"Languages for Sensor-Based Control in Robotics"
Editores: U. Rembold ; K. Hormann
NATO ASI, Vol. F29, pág. 45-59
Ed. Springer-Verlag Berlin 1987

[SHIMANO-84] Shimano, B. ; Geschke, C. ; Spalding, C.
"VAL II: A Robot Programming Language and Control System"
SME ROBOTS VIII Conference, Detroit, June 1984

[SILVER-82] Silver, W. M.
"On The Equivalence of Lagrangian and Newton-Euler Dynamics for Manipulators"
The International Journal of Robotics Research
Vol. 1, N# 2, Summer 1982, pág. 118-128

[SNYDER-85] Snyder, W. E.
"Industrial Robots: Computer Interfacing and Control"
Ed. Prentice-Hall, Inc. 1985

[SPUR-87] Spur, G. ; Duelen, G. ; Krause, F.-L.
"An Integrated Approach Toward Off-Line Robot Programming"
Off-Line Programming of Industrial Robots
Editores: A. Storr, J. F. McWaters
Ed. Elsevier Science Publishers B. V. (N-Holland)
IFIP 1987, pág. 181-191

[STORR-87] Storr, I. A. ; Schumacher, H.
"Programming Methods for Industrial Robots"
Off-Line Programming of Industrial Robots
Editores: A. Storr, J. F. McWaters
Ed. Elsevier Science Publishers B. V. (N-Holland)
IFIP 1987, pág. 1-4

[TAN-89] Tan, H. H. ; Potts, R. B.
"A Discrete Trajectory Planner for Robotic Arms with Six Degrees of Freedom"
IEEE Transactions on Robotics and Automation, Vol. 5
N# 5, October 1989, pág. 681-690

[TAYLOR-72] Taylor, R. ; Summers, P. ; Meyer, J.
"AML: A Manufacturing Language"
The International Journal of Robotics Research
Vol. 1, N# 3, 1972

[TAYLOR-79] Taylor, R. H.
"Planning and Execution of Straight Line Manipulator Trajectories"
IBM J. Res. Develp., Vol 23, N# 4, July 1979,
pág. 424-436

[TOURASS-89] Tourassis, V. D. ; Ang Jr., M. H.
"A Modular Architecture for Inverse Robot Kinematics"
IEEE Transactions on Robotics and Automation, Vol. 5
N# 5, October 1989, pág. 555-568

[VUKOBRA-82] Vukobratovic, M. ; Potkonjak, V.
"Applied Dynamics and CAD of Manipulation Robots"
ED. Springer-Verlag Berlin, Heidelberg 1982

[VUKOBRA-82] Vukobratovic, M. ; Potkonjak, V.
"Dynamics of Manipulation Robots - Theory and Application"
ED. Springer-Verlag Berlin, Heidelberg 1982

[WAMPLER-86] Wampler II, C. W.
"Manipulator Inverse Kinematic Solutions Based on Vector Formulations and Damped Least-Squares Methods"
IEEE Transactions on Systems, Man, and Cybernetics,
Vol. SMC-16, N# 1 , January/February 1986,
pág. 93-101

[WAMPLER-89] Wampler II, C. W. ; Agrawal, S. K.
"An Implementation of Inverse Kinematic Functions for Control of a Redundant Wrist"
IEEE International Conference on Robotics and Automation Proceeding 1989, pág. 914-919

[WECK-87] Weck, M. ; Niehaus, Th. ; Osterwinter, M.
"An Interactive Model Based Robot Programming and Simulation Workstation"
Off-Line Programming of Industrial Robots
Editores: A. Storr, J. F. McWaters
Ed. Elsevier Science Publishers B. V. (N-Holland)
IFIP 1987, pág. 41-51

[WEINSTO-87] Weinstock, N.
"Computer Animation"
Ed. Addison-Wesley Publishing Company, Inc. 1987

[WOZNIAK-89] Wozniak, A. ; Warczynski, J.
"Robot Simulation and Programming System"
IFAC Robot Control - Proceeding Series 1988
N# 10, pág. 437-442

[WU-84] Wu, Chi-Haur
"A Kinematic CAD Tool for the Design and Control of a Robot Manipulator"
The International Journal of Robotics Research
Vol. 3, N# 1, Spring 1984, pág. 58-67

[ZAMPIER-91] Zampieri, D. E. ; Rosário, J. M. ; Martins, J. L. Saramago, M. A. P.
"Projeto e Modelamento de um Manipulador"
Congresso de Engenharia Mecânica Norte-Nordeste, Natal, Rio Grande do Norte, Março 1991, pág.. 9-16