

ESTE EXEMPLAR CORRESPONDE A REDAÇÃO FINAL DA
TESE DEFENDIDA POR Fabio Passeto

..... E APROVADA PELA
COMISSÃO JULGADORA EM 01 / 09 / 2000.

João Maurício Rosário
ORIENTADOR

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA

**Desenvolvimento de um Aplicativo para Simulação
e Controle de Manipuladores Robóticos com
Ênfase em Aplicações Didáticas**

Autor : **Fabio Passeto**
Orientador: **João Maurício Rosário**

77/00

UNICAMP
BIBLIOTECA CENTRAL
SEÇÃO CIRCULANTE

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

**Desenvolvimento de um Aplicativo para Simulação
e Controle de Manipuladores Robóticos com
Ênfase em Aplicações Didáticas**

Autor : Fabio Passeto

Orientador: João Maurício Rosário

Curso: Engenharia Mecânica.

Área de concentração: Mecânica dos Sólidos e Projeto Mecânico

Dissertação de mestrado apresentada à comissão de Pós Graduação da Faculdade de Engenharia Mecânica, como requisito para obtenção do título de Mestre em Engenharia Mecânica.

Campinas, 2000
S.P. - Brasil

UNIDADE	Bc
N.º CHAMADA:	TI UNICAMP
	P267d
V.	Ex.
TOMBO BC/	44475
PROC.	16-392/01
C	<input type="checkbox"/>
D	<input checked="" type="checkbox"/>
PREC.	R\$ 11,00
DATA	16/05/01
N.º CPD	

CM-00155189-0

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

P267d Passeto, Fabio
Desenvolvimento de um aplicativo para simulação e controle de manipuladores robóticos com ênfase em aplicações didáticas / Fabio Passeto.--Campinas, SP: [s.n.], 2000.

Orientador: João Maurício Rosário
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Mecânica.

1. Robótica. 2. Robôs – Sistemas de controle. 3. Cinemática das máquinas. 4. Modelos matemáticos. 5. Material didático. 6. Ensino – Meios auxiliares. 7. Linguagem de programação – Robôs. I. Rosário, João Maurício. II. Universidade Estadual de Campinas. Faculdade de Engenharia Mecânica. III. Título.

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

DISSERTAÇÃO DE MESTRADO

**Desenvolvimento de um Aplicativo para Simulação
e Controle de Manipuladores Robóticos com
Ênfase em Aplicações Didáticas**

Autor: Fabio Passeto


Orientador: João Maurício Rosário



**Prof. Dr. João Maurício Rosário, Presidente
DPM – FEM – Unicamp**



**Prof. Dr. Antônio Batocchio
DPM – FEM – Unicamp**



**Prof. Dr. José Armando Valente
NIED – Unicamp**

Campinas, 1 de setembro de 2000

Dedicatória:

Dedico este trabalho todos as pessoas que trabalham no desenvolvimento e difusão do conhecimento como um patrimônio pertencente a toda a humanidade.

Agradecimentos

Este trabalho não poderia ser terminado sem a ajuda de diversas pessoas às quais presto minha singela homenagem:

A meus familiares, pelo apoio e auxílio na elaboração deste trabalho.

A CNPQ pelo auxílio financeiro através de bolsa de estudos oferecida.

A todos os professores e colegas do departamento que auxiliaram de forma direta ou indireta na realização deste trabalho, em especial a João Vilhete pelos conhecimentos na área de robótica educacional, a Carlos Erig pelo auxílio na área de eletrônica e microcontroladores e a Newton Cardoso pelo auxílio na teoria sobre modelagem de manipuladores;

Em especial, ao professor João Maurício Rosário, meu orientador neste trabalho e ao Departamento de Projeto Mecânico da Faculdade de Engenharia Mecânica da Unicamp pelo suporte prestado.

As we progress on any journey, starting with a single step, we observe many wondrous and novel surroundings. With each step, the world takes on a different look as we see new places and new faces. We also see ourselves reflected back from the eyes and faces on those we meet. Such visions portray images of where we have been and how far we have come along our path. In the process, without asking a question, we begin to wonder how far we have actually journeyed and how far we have yet to go. Such wonder may emanate from simple curiosity or from concern about the sufficiency of our supplies, our energy, or our will. The more precise and anxious among us will seek assurance that our efforts are taking us where we want to go.

DiBiella (1998),p.165.

Resumo

PASSETO, Fabio, *Desenvolvimento de um aplicativo computacional para controle de manipuladores robóticos com ênfase em aplicações didáticas*, Campinas,: Faculdade de Engenharia Mecânica, Universidade Estadual de Campinas, 2000. 190 p. Dissertação (Mestrado)

O trabalho investiga a utilização de robôs em atividades de ensino, identificando os recursos disponíveis no momento e os fundamentos que justificam sua utilização. Dentro deste contexto, foi desenvolvido um manipulador robótico de baixo custo e de construção simplificada que pode ser conectado a um computador IBM-PC através da porta serial. Para efetuar seu controle, foi criado um aplicativo computacional dotado de um interpretador de comandos que permite a criação de programas estruturados. O aplicativo também possui um simulador que permite visualizar a movimentação e cálculo da posição e orientação de um modelo de manipulador criado a partir dos parâmetros de Denavit-Hartenberg. O sistema formado pelo aplicativo computacional e a parte construtiva constituem uma ferramenta funcional que pode ser utilizada para desenvolver atividades de ensino ligadas a manipuladores robóticos, oferecendo uma melhor funcionalidade em relação a sistemas semelhantes já desenvolvidos.

Palavras Chave

-Robótica, Programação de Robôs, Didática, Simulação de Robôs, Ferramentas de Ensino

Abstract

PASSETO, Fabio, Development of a computer application to control robotic arms, with emphasis on didactic applications, Campinas,: Faculdade de Engenharia Mecânica, Universidade Estadual de Campinas, 2000. 190 p. Dissertation (Mestrado)

This investigation describes the use of robots on teaching activities, identifying the presently available resources and basis that justify its use. In this context there have been developed a low-cost and simplified construction robotic manipulator that can be connected to an IBM-PC computer by the serial port. To implement its control there has been created a software provided with a command interpreter that allows the creation of structured programs. The software has also a simulator that allows visualize the movement and calculate the position and orientation of a model of a manipulator created by using the Denavit-Hartenberg parameters. The system formed by the computer software and the physical part constitute a functional tool that can be utilized to develop learning activities related to robotic manipulators, offering a better functionality compared to similar systems already existent.

Key Words

-Robotics, Robot Programming, Didactics, Robot Simulation, Tools for Learning

Índice analítico

LISTA DE FIGURAS	XIII
LISTA DE TABELAS	XV
NOMENCLATURA	XVI
CAPÍTULO 1	1
Introdução	1
CAPÍTULO 2	7
Revisão bibliográfica e perspectiva histórica	7
2.1 Perspectiva histórica	8
2.2 Uma abordagem sobre o aprendizado	10
2.3 Perspectiva histórica a respeito de programas didáticos na área de robótica	14
2.4 Perspectiva atual sobre ferramentas para simulação de robôs	17
2.5 Fornecedores na área de robótica educacional	18
2.6 A inserção deste trabalho no contexto atual de ensino em automação e robótica	19
CAPÍTULO 3	23
Descrição do Manipulador Robótico Utilizado	23
3.1 O desenvolvimento do manipulador	26
3.2 Parte operativa	29
3.2.1 Medidas de performance e características de manipuladores	30
3.2.2 Escolha do acionamento do robô	34
3.2.2 A estrutura dos elos do robô	42
3.3 Parte de comando	44
3.3.1 Arquitetura do comando de robôs	45

3.3.2 Estruturação da parte de comando	48
3.3.3 Serial Servo Controller (SSC)	50
CAPÍTULO 4	53
Descrição do aplicativo computacional	53
4.1 Concepção do trabalho	54
4.2 Os módulos que compõem o aplicativo	56
4.2.1 O módulo Console	58
4.2.2 O módulo Programa	61
4.2.3 O módulo <i>Teach-in Box</i>	62
4.2.4 O módulo Simulador	63
4.3 A linguagem de programação do aplicativo	74
4.3.1 Regras gerais para sintaxe da linguagem de programação	74
4.3.2 Comandos	75
4.3.3 Operadores	76
4.3.4 Funções	76
4.3.5 Usando variáveis	77
4.4 Inserindo o modelo do robô no simulador	80
CAPÍTULO 5	85
Conclusões e perspectivas futuras	85
5.1 Desenvolvimentos futuros	89
REFERÊNCIAS BIBLIOGRÁFICAS	91
REFERÊNCIAS A INFORMAÇÕES DISPONÍVEIS PELA INTERNET	93
ANEXO I	95
Estrutura interna do programa	95
A I.1 Formulário MDIForm1	99
A I.2 Formulário frmMain	100
A I.2.1 Estrutura do interpretador de comandos	101

A I.2.2 A rotina String2Array	101
A I.2.3 A rotina Row2Words3	104
AI.2.4 Operador de argumentos, operador de parênteses e operador matemático	106
AI.2.5 A geração de movimentos e simulação multitarefa	108
AI.3 Formulário frmProgram	111
AI.4 Formulário frmTeach	112
AI.5 Formulário frmDHSim	113
A I.5.1 Criando imagens 3-D para ser vistas com óculos coloridos	114
AI.6 Formulário <i>Values</i>	117
AI.7 Módulo Module1	117
 ANEXO II	 119
Referências na Área de Robótica Educacional	119
A II.1 Instituições e ferramentas relacionados à programação de robôs	119
A II.2 Pesquisas e instituições ligados à simulação de robôs	126
A II.3 Fornecedores de sistemas de robótica	132
 ANEXO III	 139
A linguagem de programação	139
 ANEXO IV	 149
Representação cinemática de manipuladores robóticos	149
A IV.1 - Definição do modelo	149
A IV.1.1 Coordenadas do controlador para coordenadas dos servos	150
A IV.1.2 Coordenadas dos servos para coordenadas das juntas	151
A IV.1.3 Coordenadas de juntas para espaço cartesiano	153
A IV.2 Parâmetros de Denavit-Hartenberg	154
A IV.3 Orientação inversa RPY	159
A IV.4 Estudos de caso: Modelagem de diversos manipuladores	164
A IV.4.1 Robô Puma	165
A IV.4.2 Robô Quatro-barras	169
A IV.4.3 Robô Pick-and-place	171

A IV.4.4 Robô Scara	173
A IV.4.5 Teste do algoritmo de orientação inversa	175
ANEXO V	179
Estudo de caso: Construção, modelagem e testes com um manipulador robótico experimental	179
A V.1 A construção das juntas	180
A V.2 Sistemas de transferência de movimento:	181
A V.3 A base do robô	182
A V.4 Garra ou elemento terminal	183
A V.5 Especificações técnicas dos servos utilizados	188
A V.6 Modelagem do robô	189

Lista de figuras

<i>Figura 1.1 : A montagem criada para a Educar'98.</i>	2
<i>Figura 2.1: Duas partes da capacidade de aprendizagem de uma organização.</i>	11
<i>Figura 3.1: foto do robô didático desenvolvido.</i>	23
<i>Figura 3.2: Foto de um Kit da Lynxmotion montado.</i>	27
<i>Figura 3.3: O robô modificado.</i>	28
<i>Figura 3.4: Conjunto motor-redução produzido pela Tamy.</i>	35
<i>Figura 3.5: Mola de tração construída em liga de memória de forma de Ni-Ti.</i>	36
<i>Figura 3.6: Servo em perspectiva.</i>	37
<i>Figura 3.7: Servo desmontado.</i>	38
<i>Figura 3.8: Diversas formas de transmitir o movimento de um servo.</i>	41
<i>Figura 3.9: Estruturação da parte de comando.</i>	49
<i>Figura 4.1: Imagem do aplicativo como um todo.</i>	54
<i>Figura 4.2: Janela do módulo Console, o núcleo do aplicativo.</i>	58
<i>Figura 4.3: Teach-in Box permite mover as juntas do robô e gravar posições para criar programas.</i>	62
<i>Figura 4.4: Desenho do modelo do robô em 3-D.</i>	66
<i>Figura 4.5: Recurso de traçar trajetória do robô.</i>	67
<i>Figura 4.6: O registro da trajetória do grip.</i>	69
<i>Figura 4.7: O efeito de sombra na perspectiva isométrica.</i>	70
<i>Figura 4.8: A garra do robô desenhada no simulador.</i>	71
<i>Figura 4.9: Avaliação da performance do simulador e sugestões para melhorá-la.</i>	72
<i>Figura 4.10: Orientação e posição dos sistemas de coordenadas que compõem o modelo do robô.</i>	73
<i>Figura 5.1: Uma das possíveis utilizações do robô é manipular pequenos objetos dispostos em um ambiente estruturado.</i>	87
<i>Figura 5.2: Robô utilizando um apontador laser para simular uma operação de pintura.</i>	88
<i>Figura 5.3: Robô simulando uma tarefa.</i>	88
<i>Figura A I.1: Os elementos básicos de um DFD.</i>	96

Figura A I.2: O fluxo de dados entre as principais rotinas do aplicativo	98
Figura A I.3: A visão em profundidade.	115
Figura A II.1: Logotipo da LOGO Foundation	119
Figura A II.2: Tela de programação do TC Logo	120
Figura A II.3: Aplicativo Chipwits sendo executado em um emulador para PC	121
Figura A II.4: Programação por ícones no Chipwits	122
Figura A II.5: Tela do aplicativo MM Logic	124
Figura A II.6: Tela do Winlogo, na versão em português.	125
Figura A II.7: Simulador do Robotoy, em Java	126
Figura A II.8: Simulador de robô, em Java	127
Figura A II.9: Simulação de um robô KuKa em VRML	128
Figura A II.10: Um modelo de robô em VRML.	130
Figura A II.11: Simulador de robô para manutenção em águas profundas	131
Figura A II.12: Simulador Kraft_C, desenvolvido para a Petrobrás.	132
Figura A II.13: Logotipo da P.A.R.T.S.	133
Figura A II.14: Kit de robô produzido pela PARTS	134
Figura A II.15: Várias configurações para o Robix	134
Figura A II.16: Perspectiva explodida de um manipulador Robix	135
Figura A II.17: Desenho esquemático da comunicação envolvendo o Robix e o PC.	136
Figura A II.18: Tela do aplicativo Score Base	138
Figura A IV.1: A seqüência de transformadas elementares que formam a transformada de Denavit-Hartenberg	156
Figura A IV.2: Robô com configuração Puma.	165
Figura A IV.3: Janela do simulador exibindo o modelo do robô Puma.	167
Figura A IV.4: Estrutura cinemática de um robô quatro barras.	170
Figura A IV.5: Estrutura cinemática de um robô para paletização.	172
Figura A IV.6: Robô com configuração Scara	174
Figura A IV.7: Modelo do robô Scara no simulador.	175
Figura A IV.8: Modelo criado para verificar o algoritmo de orientação inversa.	177
Figura A V.1: O robô desenvolvido	179
Figura A V.2: As camadas que compõem o rolamento axial.	183
Figura A V.3: Partes do pulso e rotação da garra.	185
Figura A V.4: Partes que compõem a garra.	186
Figura A V.5: o sistema de rotação e fechamento da garra montado.	187
Figura A V.6: a garra fechada.	187
Figura A V.7: dimensões do Futaba S3101.	189

Lista de tabelas

<i>Tabela 3.1: Quadro comparativo: parte de comando e parte operativa de um SAP.....</i>	<i>24</i>
<i>Tabela 3.2: Especificações técnicas de um servo.....</i>	<i>41</i>
<i>Tabela 3.3: Faixa dos valores disponíveis comercialmente.....</i>	<i>42</i>
<i>Tabela A I.1: Correspondências entre desvios "para baixo".....</i>	<i>103</i>
<i>Tabela A I.2: Exemplo de valores atribuídos a variáveis.....</i>	<i>105</i>
<i>Tabela A I.3: Funcionamento da rotina Row2Words3.....</i>	<i>105</i>
<i>Tabela A IV.1: Valores de ângulos e função atan2 associados a intervalos de seno e cosseno de um ângulo.....</i>	<i>164</i>
<i>Tabela A V.1: Especificações técnicas dos servos utilizados nas juntas 1, 2, 3, 6 (acionamento a distância) e garra (acionamento a distância).....</i>	<i>188</i>
<i>Tabela A V.2: Especificações técnicas dos servos utilizados nas juntas 4, e 5.....</i>	<i>188</i>
<i>Tabela A V.3: Parâmetros cinemáticos de um robô tipo Puma.....</i>	<i>190</i>

Nomenclatura

Abreviações

BPS -	<i>Bauds Per Second (taxa de transmissão).</i>
CISC -	<i>Complex Instruction Set Controller.</i>
DPM -	<i>Departamento de Projeto Mecânico da Unicamp</i>
GDL -	<i>Graus De Liberdade.</i>
LIFO -	<i>Last In First Out (tipo de fila).</i>
I/Os -	<i>Entradas e saídas.</i>
LAR -	<i>Laboratório de Automação e Robótica do DPM - Unicamp</i>
NIEd -	<i>Núcleo de Informática Aplicada à Educação</i>
OTPROM -	<i>One Time Programmable Read Only Memory (normalmente, uma EPROM, com encapsulamento opaco, que não permite apagar o conteúdo gravado).</i>
P -	<i>(Controle tipo) Proporcional – Sinal do controlador proporcional ao erro.</i>
PIC -	<i>Programmable Integrated Controller (produzido pela Microchip) .</i>
PID -	<i>(Controle tipo) Proporcional, Integrativo e Derivativo.</i>
PLCC -	<i>Plastic Leader Carrier Cartridge. Tipo de soquete para circuitos integrados.</i>
RAM -	<i>Random Access Memory.</i>
RISC -	<i>Reduced Instruction Set Controller.</i>
SAP -	<i>Sistema Automatizado de Produção.</i>
VRML -	<i>Virtual Reality Modelling Language.</i>

Termos técnicos

<i>array</i> -	Vetor (tipo de variável)
<i>Assembly</i> -	Linguagem de programação, que pode ser obtida a partir da linguagem de máquina.
<i>string</i> -	Seqüência de caracteres de texto.
<i>debugar</i> -	Processo de eliminação de erros em um programa.
<i>encoder</i> -	Codificador de posição.
<i>Overshooting</i> -	Fenômeno comum na área de controle, quando uma variável sendo controlada ultrapassa o valor de referência em uma resposta ao degrau.
<i>teach by doing</i> -	Procedimento que consiste em utilizar um manipulador robótico fisicamente para determinar pontos de um programa.
<i>polling</i> -	Procedimento de um mestre em uma rede perguntar periodicamente aos escravos valores de variáveis.
<i>loop</i> -	Do inglês, laço.
<i>grip</i> -	Garra ou elemento terminal de um robô.
<i>buffer</i> -	Memória acumuladora de dados, usada para troca de dados com dispositivos externos.

Capítulo 1

Introdução

O Laboratório de Automação e Robótica (LAR) da Faculdade de Engenharia Mecânica da Unicamp vem desenvolvendo há algum tempo atividades de ensino e pesquisa envolvendo Sistemas Automatizados de Produção (SAPs).

Em 1998 durante a Feira Internacional de Educação Educar'98 foram apresentadas maquetes didáticas desenvolvidas em conjunto com o Núcleo de Informática Aplicada à Educação (NIED).

A partir da utilização de sistemas LEGO integrados a manipuladores robóticos didáticos RobixTM interfaceados por computador foi desenvolvido um sistema integrado de manufatura que permitiu a visualização de conceitos fundamentais de automação.

Naquela ocasião Fred Martin, pesquisador especialista no desenvolvimento de robôs didáticos do Massachusetts Institute of Technology (MIT), EUA ficou surpreso ao verificar como um manipulador robótico didático, apesar de extremamente simples do ponto de vista construtivo, se mostrava uma ferramenta engenhosa e funcional na manipulação de pequenos objetos.

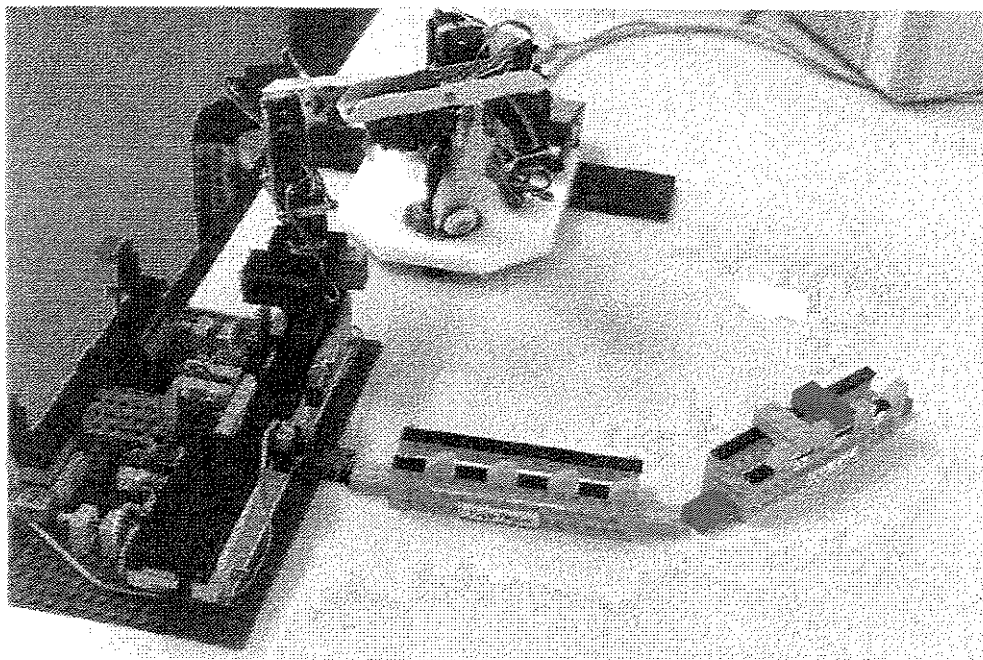


Figura 1.1 : A montagem criada para a Educar'98.

À esquerda, a montagem LEGO simulando uma esteira transportadora e um forno de tratamento térmico. Os suportes para as peças estão dispostos de modo que, alterando a posição do servo que faz a rotação do braço em relação à base, usando a mesma programação para pegar ou depositar peças em um suporte ou no outro.

Conforme constatado na revisão bibliográfica, apesar de ter havido um grande desenvolvimento de robôs móveis para uso em atividades de pesquisa e ensino, inclusive sendo o MIT uma referência nesta área, certamente ainda há muito a se desenvolver na área de manipuladores robóticos a fim de se explorar toda sua potencialidade.

Tanto na construção física do robô, quanto no aplicativo que permite sua programação, existe uma demanda por que permitam a construção de dispositivos funcionais sem requerer grande investimento de recursos na aquisição dos materiais.

O desenvolvimento deste trabalho surgiu da identificação por um lado, da potencialidade do uso de manipuladores robóticos em atividades de ensino e por outro lado, da possibilidade de se criar um sistema em diversos aspectos melhor que os disponíveis comercialmente no

momento.

Este trabalho trata do desenvolvimento de um aplicativo computacional genérico, capaz de realizar toda a parte de controle de alto nível de um manipulador robótico, independente da tecnologia utilizada para seu acionamento ou configuração física, necessitando somente a entrada dos parâmetros cinemáticos adequados sem necessidade de alteração do código-fonte.

Isto juntamente com o desenvolvimento da parte física do robô, traz como resultado um sistema completo formando um manipulador robótico que pode ser controlado e programado a partir de um computador PC.

Este sistema se presta ao ensino na área de robótica, permitindo abordar aspectos de programação, modelagem cinemática e integração com outros dispositivos.

Um conjunto de um ou mais coprocessadores dedicados se encarrega das atividades de controle de baixo nível, sendo que estes são projetados especificamente para controlar um determinado tipo de acionamento utilizado nas juntas do robô (motores de passo, motores CC etc.).

Toda a lógica de controle de alto nível é concentrada no PC, aproveitando sua capacidade de processamento, de armazenamento de dados e de interface com o usuário, enquanto os coprocessadores se encarregam somente de gerenciar o acionamento elétrico do manipulador robótico a partir dos comandos recebidos do PC através da porta serial.

O aplicativo é completo ao ponto de poder utilizado em modo simulação sem a presença fisicamente do manipulador, porém toda sua funcionalidade é alcançada quando a ele se associa uma parte física, composta por um manipulador dotado de um sistema de controle simplificado conectado ao computador por meio da porta serial.

Como será visto no capítulo 2, existem *kits* de robôs didáticos à venda no mercado, mas

nenhum deles apresentou as características desejadas de performance pelo custo, o que levou à necessidade de desenvolvimento nesta área. O robô implementado foi construído a partir de um *kit* comercial totalmente remodelado.

O capítulo 3 aborda conceitos genéricos ligados a manipuladores, que serviram de embasamento teórico no desenvolvimento deste trabalho e mostra os principais aspectos relativos à construção de um manipulador robótico, com o objetivo de servir como referência para futuros desenvolvimentos semelhantes.

O capítulo 4 enfoca o aplicativo computacional desenvolvido, descrevendo seu funcionamento do ponto de vista do usuário, sem se ater a detalhes de seu funcionamento interno.

O capítulo 5 faz a conclusão do trabalho e traz perspectivas para futuros desenvolvimentos na área.

No Anexo I está a documentação do aplicativo computacional desenvolvido, no tocante a aspectos de seu funcionamento interno.

O Anexo II traz o resultado de pesquisas realizadas na área de robótica educacional, identificando em que contexto se encontram os desenvolvimentos nesta área.

O Anexo III é uma tabela de referência da linguagem de programação do aplicativo computacional, descrevendo o funcionamento de cada um de seus comandos.

O Anexo IV trata da modelagem cinemática de manipuladores e traz alguns exemplos de modelagem de configurações cinemáticas mais comumente encontradas.

Finalmente, o Anexo V traz os aspectos construtivos do manipulador robótico desenvolvido neste trabalho. Não houve preocupação em documentar detalhadamente suas características através de desenhos técnicos, porque o manipulador construído representa uma solução

particular, cujo projeto foi adequado especificamente aos componentes empregados na sua construção.

Capítulo 2

Revisão bibliográfica e perspectiva histórica

Este capítulo se inicia com uma perspectiva histórica sobre o desenvolvimento da robótica, trata da fundamentação do desenvolvimento de ferramentas didáticas e identifica instituições e ferramentas relacionados à programação e simulação de robôs e fornecedores de sistemas de robótica.

A robótica é uma área muito extensa, que trata basicamente da aplicação da automação em máquinas. Compreende dispositivos com características bastante distintas, como máquinas-ferramenta, robôs autônomos e manipuladores robóticos.

Este trabalho trata especificamente de manipuladores robóticos, que são manipuladores sobre os quais se implementa um sistema de automação e controle.

Manipuladores são dispositivos com grande flexibilidade de movimento, capazes de realizar de tarefas complexas através da movimentação de suas juntas. Para isto, contam com graus de liberdade cuja coordenação de movimentos leva à orientação e posicionamento do elemento terminal.

Os manipuladores robóticos são bastante adequados à automação flexível pois necessitam apenas reprogramação e adequação do elemento terminal sem que haja necessidade de se

modificar a construção física para se adequar aos mais diversos tipos de tarefas a serem realizadas, diferente de outros sistemas automatizados de produção, que são fisicamente configurados para realizar tarefas específicas.

2.1 Perspectiva histórica

O termo "robô" surgiu da ficção científica da década de 20, e começou a se tornar realidade a partir da década de 40, com a construção de manipuladores teleoperados. Na década de 50, surgiram os primeiros manipuladores robóticos agregando a tecnologia de máquinas de usinagem de Comando Numérico aos manipuladores teleoperados.

Atualmente, o uso de manipuladores robóticos está principalmente ligado às montadoras de veículos, devido à complexidade das tarefas que têm de ser realizadas e ao alto volume de produção que justifica arcar com seu elevado custo. O tipo de aplicação à qual o manipulador se destina tem forte influência sobre suas características. Pode-se destacar os principais tipos de aplicações:

- i) Robôs autônomos: demandam um sistema de comando sofisticado e um complexo sistema de sensores.
- ii) Robôs teleoperados: realizam tarefas especiais como operações a grandes profundidades sob a água, manipulação de material nuclear etc. Neste caso, as tarefas não são repetitivas, sendo necessário um comportamento flexível e um sistema de sensores para auxiliar a operação.
- iii) Aplicações industriais: realizam operações repetitivas e demandam um sistema avançado de programação, ambiente estruturado, sensores e em alguns casos, mobilidade.

Dentro destas, este trabalho enfoca os robôs industriais, que são programados para realizar tarefas repetitivas.

As principais aplicações para manipuladores robóticos na indústria são: soldagem (mais de 50% dos casos), aplicação de cobertura, operações de corte, montagem de produtos, rebarbação e manipulação de produtos, *palletizing*.

As operações de soldagem basicamente podem ser tipo solda-ponto ou solda a arco e

garantem uma qualidade superior à das soldas realizadas por pessoas.

Normalmente, são criadas células de soldagem, com sistemas de transporte ou posicionamento de produtos; em muitos casos, o robô ou a peça se move durante a operação de soldagem, ampliando a capacidade do robô em alcançar os pontos mais distantes.

Um automóvel possui entre 5000 e 6000 pontos de solda, demandando intenso trabalho, que também influi na qualidade final do produto. Normalmente, vários robôs trabalham simultaneamente em um mesmo *chassis*, e em pontos difíceis de serem atingidos.

Nas operações de solda a arco, é importante um sofisticado sistema de controle de trajetória, já que a solda é feita em movimento contínuo. Na grande maioria dos casos, a solda é feita com um movimento linear, porém em alguns casos é necessário interpolar o movimento através de segmentos de reta e arcos de circunferência, ou criar trajetórias contínuas.

Em operações de corte que demandam pouca precisão e alta flexibilidade, braços robóticos são usados em substituição a máquinas-ferramenta. Tipicamente, o robô opera um jato de água, laser de corte, plasma etc. Para realizar cortes com maior precisão, utiliza-se modelos que guiam o elemento terminal durante a operação.

Operações de pintura ou aplicação de selante ou adesivo requerem uma grande flexibilidade de movimentos do robô, e normalmente são programadas através da movimentação manual do robô, que memoriza os movimentos e os repete indefinidamente.

Para montagem de produtos, utiliza-se complacência ativa ou passiva. O robô não deve simplesmente ser utilizado para realizar as mesmas tarefas que antes eram realizadas por pessoas, e sim deve-se fazer uma completa remodelagem do produto e da linha de produção para se adequar à montagem por robôs.

Outro tipo de aplicação bastante freqüente é manipulação de produtos e rebarbação, em que

o robô manipula um produto que é trabalhado por uma máquina fixa, seja alimentando a máquina com peças ou seja segurando a peça enquanto a máquina realiza alguma operação.

Este tipo de aplicação requer a capacidade de integração do manipulador robótico com outros equipamentos. Frequentemente, são utilizadas máquinas ou dispositivos com graus de liberdade controlados pelo próprio robô, denominados "graus de liberdade externos".

Nas operações de paletização, robôs são usados para manipular objetos, colocando-os em uma matriz (um *pallet* ou caixa para acomodar o produto). Este tipo de operação não demanda muita precisão, nem flexibilidade de movimentos. Frequentemente, são necessários apenas 3 graus de liberdade, pois não existe a necessidade de mudar a orientação dos objetos sendo manipulados. Neste tipo de operação, pode-se incluir além da montagem de *pallets*, a montagem de circuitos eletrônicos e manipulação de medicamentos.

2.2 Uma abordagem sobre o aprendizado

Em se tratando de um trabalho que enfatiza aplicações didáticas, é importante fundamentar conceitos sobre aprendizagem e como este trabalho pode contribuir neste sentido.

Segundo DiBiella, (1998), o desenvolvimento da capacidade de aprendizado em organizações depende da integração de **orientações para o aprendizado e de fatores facilitadores**.

O conceito de aprendizado está ligado ao ganho de experiência, construção de competência, e de se evitar a repetição de problemas e erros que desperdiçam recursos.



Figura 2.1: Duas partes da capacidade de aprendizagem de uma organização.
(baseado em DiBiella (1998), p.24)

As orientações para o aprendizado compreendem a forma pela qual o conhecimento é produzido e a informação é transmitida. Representam o estilo adotado por cada organização, ou seja, representa os fatores descritivos. Independente de quais orientações forem adotadas, o aprendizado pode ocorrer de forma eficiente.

O gerenciamento do aprendizado pode ser feito de modo normativo (aprendizado em equipes, visão repartida, pensamento em sistemas, modelos mentais ou maestria pessoal), desenvolvimental (evolucionar ou revolucionar) ou por capacidade (aprendizado contínuo, ligado ao desenvolvimento das capacidades atuais da organização).

Os fatores facilitadores são normativos, ou seja, precisam estar presentes para que o aprendizado ocorra de modo eficiente. São práticas ou condições que promovem o aprendizado dentro de todos os tipos de organização. Estabelecem um potencial de aprendizado, que pode se concretizar ou não, dependendo da orientação adequada. Em contrapartida, se estes fatores não estiverem presentes, o aprendizado será sufocado.

O desenvolvimento deste trabalho se insere no campo educacional fornecendo uma

ferramenta que pode ser aplicada no aprendizado como um fator facilitador, seja como objeto de estudo ou como forma de despertar curiosidade, levando à criação de novas idéias e tecnologias e também como suporte à experimentação, proporcionando um recurso de alta qualidade para o aprendizado.

Gardner, (1995), na página 78 ajuda a esclarecer conceitos importantes, definindo inteligência como “*a capacidade de resolver problemas ou elaborar produtos que sejam valorizados em um ou mais ambientes culturais.*”.

Nas últimas décadas tem havido um crescente reconhecimento da importância da capacidade de utilizar diversos sistemas simbólicos. Os símbolos são característicos da cultura de uma sociedade e permitem a comunicação entre os indivíduos.

As culturas mais avançadas desenvolvem um segundo nível de símbolos, inventados (ou notacionais), como a escrita e os números. Os símbolos de segunda ordem permitem fazer a representação de idéias e permitem o desenvolvimento de ferramentas e conceitos que se sobrepõem uns aos outros.

Um bom exemplo é a matemática, que parte de operações fundamentais como soma e subtração para chegar a operações sucessivamente mais complexas como multiplicação e divisão, exponenciação e logaritmo e assim por diante.

Assim, o indivíduo deixa de ter a necessidade de um “conhecimento prático”, sempre atrelado à realidade, passando a adquirir a capacidade de trabalhar com conceitos abstratos que são representados usando os sistemas simbólicos de segunda ordem.

Isto permite a resolução de problemas de enorme complexidade, muito além da capacidade do que o conhecimento prático jamais permitiria.

No entanto, no ambiente escolar e de ensino superior a aprendizagem se dá de uma forma

descontextualizada e fragmentada. O aluno tem dificuldade em reunir os conhecimentos adquiridos em disciplinas distintas e tem dificuldade de aplicar o que aprendeu.

A resolução de problemas parte inicialmente da identificação do problema, para modelagem através de sistemas de segunda ordem, fazendo determinadas hipóteses e simplificações adequadas, obtenção de resultados através de manipulações abstratas utilizando as ferramentas adequadas e posteriormente interpretação dos resultados obtidos para encontrar e implementar uma solução.

A manipulação de sistemas simbólicos ensinada nas escolas é apenas uma parte deste processo e por isso surgem dificuldades na resolução de problemas práticos, levando os alunos a descreditarem na importância do aprendizado e procurarem aplicar conhecimentos práticos sempre que possível.

Uma caricatura desta realidade é a eterna rivalidade entre técnicos e engenheiros: os técnicos se baseiam em conhecimento prático adquirido em anos de experiência, na criatividade e na habilidade de aplicar metodologias das quais não conhecem os fundamentos.

Por outro lado, os engenheiros procuram aplicar o conhecimento que aprenderam para a resolução de problemas, mas têm dificuldade em fazer a ponte entre teoria e prática, e muitas vezes chegam a soluções absurdas ou impraticáveis.

Gardner (1995) defende que a educação deve ser voltada ao entendimento, ou seja, à “capacidade de aplicar conhecimentos, conceitos ou habilidades (...) adquiridos em algum tipo de ambiente educacional em um novo exemplo ou situação em que este conhecimento é de fato relevante.”

A maioria dos alunos não compreende os conteúdos apresentados e se mostra incapaz de mobilizar os conceitos apropriados que aprenderam, mesmo que tenham sido bons alunos.

Fundamentado nisto, Gardner defende o ensino voltado ao entendimento, que trabalha com a aplicação de conhecimentos que são diversas vezes revisitados durante o curso sob perspectivas diferentes.

Além de permitir uma melhor assimilação do conteúdo, o ensino voltado ao entendimento leva à capacidade de adquirir perícia em uma área profissional.

O desenvolvimento de atividades de laboratório objetivam a exemplificar e ilustrar conceitos teóricos que estão sendo transmitidos. Neste caso, o estudo é desenvolvido pela aplicação da teoria genérica sobre um caso particular, sendo feita a ligação entre teoria e prática.

A aplicação da teoria a um caso particular resulta em melhor fixação do conteúdo e permite verificar se a assimilação da teoria foi completa, uma vez que o aluno deverá sentir naturalmente a falta das partes da matéria que não tenha assimilado e que sejam necessárias ao experimento.

Após a realização de uma ou mais práticas de laboratório, o aluno passa a estar apto a aplicar os conceitos assimilados para a resolução de qualquer problema, mesmo extrapolando para aqueles que não foram exemplificados em aula.

2.3 Perspectiva histórica a respeito de programas didáticos na área de robótica

O surgimento da informática a custos acessíveis a partir da década de 70 levou à criação de ferramentas inovadoras na época, vislumbrando as novas possibilidades que poderiam revolucionar o ensino. Apesar dos computadores disponíveis naquela época serem extremamente primitivos se comparados aos atuais, foram criadas ferramentas com recursos muito interessantes e que até hoje poderiam estar sendo utilizadas.

Num período seguinte, estas ferramentas parecem ter sido um pouco esquecidas e com o tempo, deixaram de ser inovadoras. Isto é citado no *site* da Organização Logo. Segundo consta, por um período considerável de tempo esta linguagem de programação esteve praticamente

abandonada, desatualizando-se em relação às inovações que surgiram em outras linguagens de programação e veio a ser novamente atualizada com a adoção do padrão LISP para os comandos.

Segundo consta no *site* da Logo Foundation

URL:<http://el.www.media.mit.edu/groups/logo-foundation/> EUA: acessado em 03/10/2000,

"By the early 1990's some educators in the United States began to see Logo as old and out of date. The lack of innovation in LogoWriter and the sluggish pace of upgrading of the classic Logos was in sharp contrast to the rapid development of modern, flashy educational software that took advantage of the Macintosh / Windows graphical user interface. There were some Logo drop outs and Logo did not attract its share of interest among the many new computer-using educators in the United States and Canada.

This was not necessarily the case in the rest of the world. In 1988 the Programa Informática Educativa was initiated in Costa Rica by the Omar Dengo Foundation, the Ministry of Public Education and IBM Latin America. This still growing project has put Logo in the hands of 35% - soon to be 50% - of Costa Rica's elementary school students and their teachers. A similar project has been initiated in Costa Rica's secondary schools.

The Costa Rican projects have provided extensive teacher education and support with a strong emphasis on Logo's constructionist educational approach. They have been taken as models for similar endeavors in a dozen other Latin American countries."

Provavelmente, a massificação do uso dos computadores como videogames, processadores de texto e *browsers* de Internet e a adoção marginal destas ferramentas por parte das instituições de ensino, relegou o desenvolvimento de informática aplicada à educação a um segundo plano.

A partir da revisão bibliográfica, no Anexo II pode ser constatado que numa época em que havia necessidade de racionalizar o uso dos recursos computacionais, que eram bastante limitados, foi possível criar ferramentas poderosas para auxílio ao ensino. Os exemplos são o TC

Logo e Chipwits, que podiam ser executados em um computadores de 8 bits com 48 Kb de memória RAM e 1 MHz de *clock* do processador.

Hoje, estas linhas de pesquisa evoluíram para a construção de robôs móveis microprocessados, dispostos de recursos avançados de programação e a custo bem reduzido.

Atualmente, o desenvolvimento da robótica de baixo custo dá uma nova força a estas atividades. As ferramentas atuais são muito mais interessantes pois possibilitam trazer para o mundo real os dispositivos, que antes eram somente simulados no computador.

Existem inúmeras possibilidades de criação de robôs autônomos de baixo custo, com ferramentas funcionais e que podem ser aplicados à educação com custos reduzidos.

Estes robôs não utilizam tecnologia de ponta, nem soluções tecnológicas revolucionárias, mas possibilitam que novas aplicações sejam desenvolvidas a custos bastante acessíveis.

Curiosamente, a partir de inúmeras pesquisas foi verificado que não existe grande desenvolvimento na área de manipuladores robóticos. Algumas soluções permitem construir manipuladores de pequeno porte a um custo muito inferior a dos manipuladores robóticos construídos com a tecnologia usada nos manipuladores industriais, seguindo a tendência dos robôs autônomos. O Anexo II traz referências sobre as opções disponíveis atualmente.

O surgimento de tais ferramentas viabiliza o uso em aulas de laboratório auxiliando no ensino de robótica. Um exemplo são os robôs didáticos RobixTM usados pelo LAR no ensino básico de programação de robôs e para criação de Sistemas Automatizados de Produção (SAPs) em aulas de laboratório.

Porém o software que permite a programação que acompanha o produto somente é capaz de gravar trajetórias compostas por diversos movimentos ponto-a-ponto, sendo que o uso de condicionais deve ser feito criando-se programas em Linguagem C para acessar as portas de

entrada e saída, e acionar a sequência de movimentos desejada. Isso dificulta a criação de programas, pois a criação da estrutura do programa e a programação dos movimentos do robô são feitos em ambientes separados.

A simples possibilidade de desenvolver um aplicativo capaz de aglutinar as duas funções, de programar os movimentos e criar de programas estruturados já representa uma grande melhora em relação à forma com que está estruturado o sistema Robix.

Além disso, o aplicativo desenvolvido oferece diversos outros recursos como o simulador cinemático, que torna sua funcionalidade muito superior à dos outros sistemas disponíveis para ensino atualmente.

Da mesma forma que houve uma racionalização nos robôs autônomos, da mesma forma este trabalho procura desenvolver a possibilidade de se construir manipuladores robóticos com uma concepção racionalizada, o que permite reduzir custos de construção a valores bastante acessíveis às instituições envolvidas no ensino de robótica.

Esta racionalização se fundamenta na possibilidade de construção de uma parte de comando sofisticada para o robô, centralizada em um computador PC. Por outro lado, a construção do manipulador é simplificada ao máximo, limitando sua aplicação a mover pequenas peças ou simular processos industriais.

Com o objetivo de abordagem didática de grande parte da teoria relacionada à robótica não existe a necessidade de utilizar um robô de grande porte ou alta performance.

2.4 Perspectiva atual sobre ferramentas para simulação de robôs

A simulação de robôs pode ser criada a partir de linguagens de programação convencionais, como o Visual Basic, que foi usado neste trabalho, ou o Matlab, largamente usado na FEM; ou baseado em linguagens de realidade virtual como o Java 3-D, ADA ou ainda a VRML. O Anexo

II mostra diversos sistemas desenvolvidos para simular manipuladores robóticos.

Seguramente, as linguagens de mais alto nível com capacidades especialmente desenvolvidas para simulação 3-D resultam em uma solução mais rápida e visualmente mais elaborada. O principal limitante das simulações mais complexas neste sistema é a necessidade de um computador com grande capacidade de processamento, ou de uma placa gráfica com capacidade 3-D.

A VRML demanda a instalação de um *plug-in* para funcionar. Estes são distribuídos gratuitamente por várias empresas na Internet em versões *evaluation*.

Para a finalidade de pesquisa porém, as soluções prontas não permitem abordar alguns dos conceitos básicos, como por exemplo, as transformações homogêneas e além disso não oferecem a flexibilidade de uma linguagem de programação convencional.

Por esta razão, foi escolhido o Visual Basic para criação do simulador cinemático para robôs. Embora esta solução tenha sido mais trabalhosa, foi possível implementar outros recursos, como múltiplas perspectivas, imagens estéreo, traçado de trajetórias que no caso da VRML, somente estariam disponíveis se o programa de visualização os habilitasse.

2.5 Fornecedores na área de robótica educacional

Na área de robôs autônomos microprocessados, a gama de opções é tão vasta que nem vale a pena procurar os principais expoentes. Porém, na área específica de manipuladores robóticos encontra-se basicamente dois tipos de solução: robôs didáticos construídos a partir da mesma tecnologia dos robôs industriais, porém de performance inferior e consequentemente, menor custo; ou robôs construídos a partir de servos para modelos de radiocontrole ou materiais de baixo custo, como motores de passo.

Esta segunda solução possibilita a criação de manipuladores robóticos sempre de pequeno

porte (tipicamente, com uma capacidade de carga de até 100 gramas), a um custo extremamente reduzido.

A partir de uma vasta pesquisa em fornecedores, foram encontrados apenas três que atuam nesta área: Robix, Lynxmotion e PARTS. Veja maiores detalhes no Anexo II.

Certamente, a redução de custo limita a construção do robô a um modelo bastante simples e com algumas características diferentes das de um robô industriais: material de construção, tipo de controlador usado, entre outras. O ideal será sempre simular da melhor forma possível as características de um robô industrial.

Apesar das limitações construtivas, estes robôs têm a vantagem de um custo muito reduzido, o que possibilita a qualquer laboratório adquirir, ou até mesmo construir por conta própria, um grande número deles. A palavra-chave neste caso é **tornar a tecnologia acessível ao aluno**.

Quanto às diferenças existentes entre a parte construtiva de um destes robôs (tanto na parte operativa, quanto na parte de controle), não será propriamente um problema, dependendo do assunto que se pretende abordar.

Obviamente, estes robôs não são adequados para abordar os aspectos nos quais há diferenças significativas dos robôs industriais, como por exemplo, a parte construtiva. Este trabalho procura desenvolver uma solução que resulte ao menor custo possível e capaz de simular da melhor forma possível as características encontradas em um robô industrial.

2.6 A inserção deste trabalho no contexto atual de ensino em automação e robótica

Primeiramente ao tratar de robôs didáticos é importante fundamentar o que se entende por robôs didáticos. Na realidade, não existe uma fronteira clara separando o que ser pode considerado um robô didático. Existe sim um conjunto de características que tornam um robô

adequado a determinada utilização, sendo que uma das possíveis utilizações de robôs é para ensino.

O principal requisito para que um robô se preste a atividades de ensino é o baixo custo. Robôs industriais são concebidos para trabalhar ininterruptamente ao longo de vários anos, com alta confiabilidade e grande precisão. A grande maioria deve ter força suficiente para manipular peças ou ferramentas na realização de tarefas.

Consequentemente, robôs industriais são caros e em geral somente são viáveis economicamente para atividades produtivas de grande complexidade, em geral substituindo o trabalho humano em atividades de outra forma difíceis de serem automatizadas.

Certamente o robô mais adequado para se usar em atividades didáticas seria um robô industrial porém como seu custo se torna proibitivo na maioria dos casos, torna-se necessário desenvolver robôs especificamente concebidos para que seu custo produtivo seja reduzido.

Isso é conseguido reduzindo os requisitos de confiabilidade, capacidade de carga, precisão, vida útil e segurança. Pode parecer um contra-senso reduzir os requisitos de segurança em robôs didáticos, mas o fato é que devido à sua pequena capacidade de carga, estes são praticamente inofensivos, e o que se pode considerar requisitos de segurança são proteções para evitar danos ao próprio equipamento.

Diferente da maioria dos robôs didáticos, que usam a mesma tecnologia dos robôs industriais para criar dispositivos didáticos, neste caso foi usada tecnologia de aeromodelos de controle remoto. São materiais tecnologicamente avançados e de baixo custo, pois são produzidos em larga escala. Além disso, são facilmente encontrados em lojas especializadas e fáceis de se trabalhar, sem necessidade de ferramentas especiais.

O robô didático na forma como foi concebido neste trabalho é um dispositivo de baixo custo, que procura ao máximo simular as características de um robô industrial, mas devido a

diferenças na parte de controle, no acionamento, construção dos elos, capacidade de carga, precisão etc. presta-se principalmente para simular tarefas de um robô industrial, e não necessariamente deve realizá-las.

Para exemplificar, o robô implementado não seria capaz de erguer uma pistola de pintura, mas pode simular uma operação de pintura apontando um feixe de LASER sobre o modelo miniatura de um automóvel. Obviamente, deve-se fazer a transposição necessária entre a tarefa que o robô está realizando e a que estaria realizando em um uso industrial.

Do ponto de vista do aplicativo computacional, não existem diferenças muito grandes entre o que foi desenvolvido e um aplicativo voltado ao controle de um robô industrial. Por um lado alguns pontos tiveram que ser melhor elaborados entre eles a interface com o usuário, elaboração visual e recursos para torná-lo mais amigável.

Por outro lado, o sistema de comunicação pela porta serial da forma como foi implementado não é adequado para controlar um robô industrial pois não possui performance suficiente para controlar um robô de alta precisão e nem oferece um protocolo de comunicação com a segurança que seria exigida.

Não obstante, o aplicativo computacional poderá igualmente ser utilizado para controlar robôs industriais. No entanto, para isto seriam necessárias algumas modificações no sentido de aumentar a segurança e adequar algumas características do aplicativo às necessidades dessa classe de robôs.

Outra característica importante do aplicativo é o fato de este ser genérico, ou seja, capaz de controlar um braço robótico, independente de sua configuração ou forma de acionamento utilizada.

Isto é possível graças a diversas opções de configuração do aplicativo que permitem adequá-lo a virtualmente qualquer manipulador robótico. Coprocessadores são encarregados de

gerar sinais elétricos para acionar o robô a partir das informações transmitidas através da porta serial do computador PC. Estes necessariamente devem ser adequados ao sistema de acionamento do robô.

No robô que foi implementado, usou-se um coprocessador vendido comercialmente, mas eventualmente pode ser necessário projetar um coprocessador com características diferentes. Como o aplicativo computacional concentra toda a parte cálculos mais complexos, o projeto de um coprocessador é uma tarefa relativamente fácil e seu custo de produção, bastante reduzido.

O capítulo seguinte trata com maiores detalhes o manipulador robótico utilizado, abordando suas principais características. O Anexo V também enfoca o manipulador, se concentrando em aspectos do seu projeto mecânico.

Capítulo 3

Descrição do Manipulador Robótico Utilizado

Para possibilitar a verificação do funcionamento do aplicativo desenvolvido, foi construído um robô experimental. A solução adotada para implementação da parte física (braço robótico e coprocessador) representa uma, entre um vasto conjunto de possíveis soluções.

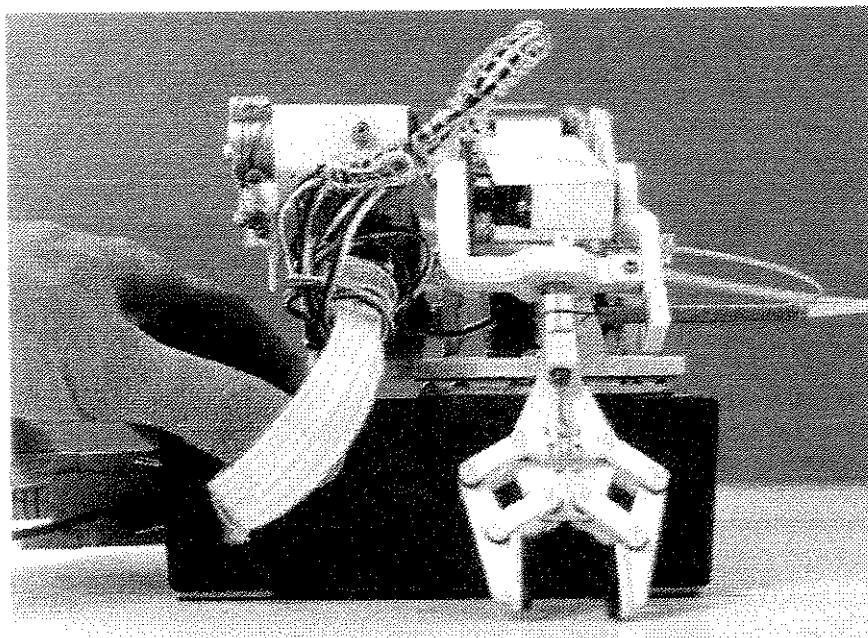


Figura 3.1: foto do robô didático desenvolvido.

Um manipulador robótico deve ser visto do ponto de vista sistêmico, como sendo composto por sub-sistemas principais e estes por sua vez, compostos por outros sub-sistemas e assim

sucessivamente. O próprio manipulador robótico tem por finalidade operar em um sistema mais complexo, digamos, uma linha de produção.

A metodologia geral de análise consiste em separar um problema complexo em partes mais fáceis de solucionar, definindo-se como estas vão interagir. A definição das fronteiras que separam as diversas áreas é feita meramente por conveniência, a fim de facilitar a análise do problema.

A abordagem a respeito de manipuladores robóticos será feita a partir da definição de dois sub-sistemas principais: a Parte de Comando e a Parte Operativa. Pode-se ver na tabela 3.1 as principais características destas partes, descritas genericamente para um Sistema Automatizado de Produção (SAP).

Tabela 3.1: Quadro comparativo: parte de comando e parte operativa de um SAP.

Parte Operativa	Parte de Comando
Manipula matéria	Manipula informações
Age diretamente sobre o produto	Controla o processo e fornece visualizações
Tendência à especialização	Tendência à padronização
Construção complexa	Programação complexa

Por Parte de Comando entende-se tudo aquilo que é voltado a manipular informações e realizar tarefas de controle. No caso deste trabalho, isto inclui o aplicativo computacional desenvolvido, o PC, as placas eletrônicas sensores etc.

Por outro lado, a Parte Operativa tem por objetivo manipular fisicamente os objetos e

realizar transformações de forma e características. Está ligada à parte mecânica que tem por finalidade manipular a matéria.

Esta separação em dois subsistemas principais é conveniente entre outras razões porque a parte de comando tende a ser genérica, enquanto a parte operativa possui uma tendência à especialização.

A parte operativa possui um alto grau de especialização e neste caso devido a necessidades construtivas foi projetada especificamente para utilizar determinados modelos de servos para seu acionamento e observando determinados requisitos funcionais tais como precisão, capacidade de carga etc..

O aplicativo computacional é um programa de computador, que pode ser duplicado quantas vezes for necessário e somente necessita sua instalação em um computador para ser utilizado.

Devido ao fato de ter custo de produção zero, como forma de reduzir o custo de um robô didático o aplicativo concentra para si toda a complexidade do controle de um manipulador robótico, de tal forma que pode até ser utilizado sem a presença do manipulador pois possui um simulador que exibe a movimentação de um modelo matemático de um robô.

O restante da parte de comando é formada por um ou mais coprocessadores, controladores dos servos, sensores etc. e se encarrega de transformar as informações recebidas via porta serial do aplicativo em ações de controle.

Devido ao fato de terem de comandar o acionamento mecânico das juntas, estes elementos são até certo ponto especializados, sendo que seu projeto tem de levar em consideração as necessidades da parte operativa, como tempo de resposta, modelo do sistema físico e até restrições quanto ao tamanho físico da placa, valores de tensão e corrente elétrica entre outros fatores.

Consequentemente, existe uma grande independência entre o aplicativo computacional e a parte física do manipulador robótico (o restante da Parte de Comando e toda a Parte Operativa). Para que ambas as partes operem em conjunto basta uma compatibilização do protocolo de troca de dados e que as limitações por parte do aplicativo, como velocidade de geração de trajetórias, sistema de comunicação para as entradas e saídas etc. sejam adequadas às características demandadas pelo manipulador.

Ambas as partes, parte construtiva e aplicativo puderam ser desenvolvidas de forma independente e simultaneamente devido à independência entre elas. Isto permitiu identificar requisitos funcionais para o aplicativo e também permitiu a evolução da concepção do manipulador através de inúmeros testes e modificações para que seu funcionamento atendesse às expectativas.

3.1 O desenvolvimento do manipulador

Como pode ser visto na revisão bibliográfica (Anexo II), a disponibilidade de manipuladores robóticos de baixo custo no mercado é bastante restrita. Como ponto de partida para o desenvolvimento do manipulador robótico, foi adquirido e montado um kit que inclui os materiais necessários para montar o manipulador robótico e um coprocessador para comandar os servos denominado Serial Servo Controller (SSC), da empresa americana Lynxmotion.

No entanto, este se mostrou pobre em alguns aspectos, que tiveram de ser aprimorados. O resultado foi praticamente uma reconstrução completa do manipulador, aproveitando algumas aspectos do projeto original e modificando os demais. Os maiores inconvenientes identificados foram:

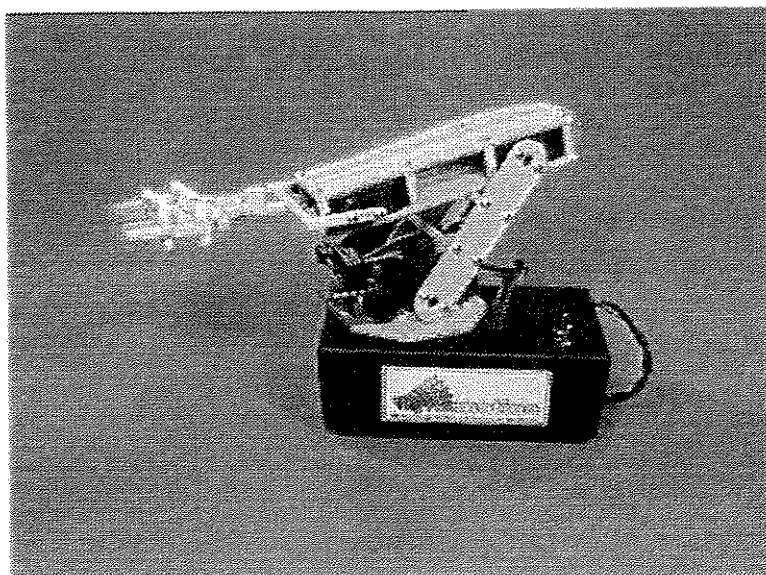


Figura 3.2: Foto de um Kit da Lynxmotion montado.

- O robô possui apenas 4 GDL, enquanto a maioria dos manipuladores opera com 6 GDL.
- A construção dos elos é simplificada demais, sendo que os servos são colados à estrutura, enquanto deveriam possuir uma fixação móvel para possibilitar manutenção.
- A parte de comando não possui controle de entradas e saídas, limitando integração com outros dispositivos.
- Os servos utilizados têm a limitação física de uma amplitude de movimento próxima a 180°, enquanto robôs industriais costumam a operar com amplitudes de movimentos superiores.
- O controlador opera com apenas 8 bits para posicionamento dos servos, resultando em uma precisão abaixo do que a parte operativa é capaz de oferecer.
- O comprimento da garra varia de acordo com sua abertura.
- O *kit* não inclui um *software* para controle do robô, e os disponíveis à venda somente são capazes de memorizar seqüências de movimentos, sem permitir a criação de programas estruturados.

Visto os itens acima relacionados, pode-se perceber a possibilidade e de certa forma a necessidade de desenvolver um manipulador robótico com características melhores.

Atualmente, não existem à venda outros *kits* de robôs com melhores características a um preço acessível e a disponibilidade de *kits* robóticos de baixo custo é realmente muito limitada, o

que levou à necessidade de desenvolver aspectos da parte física do robô.

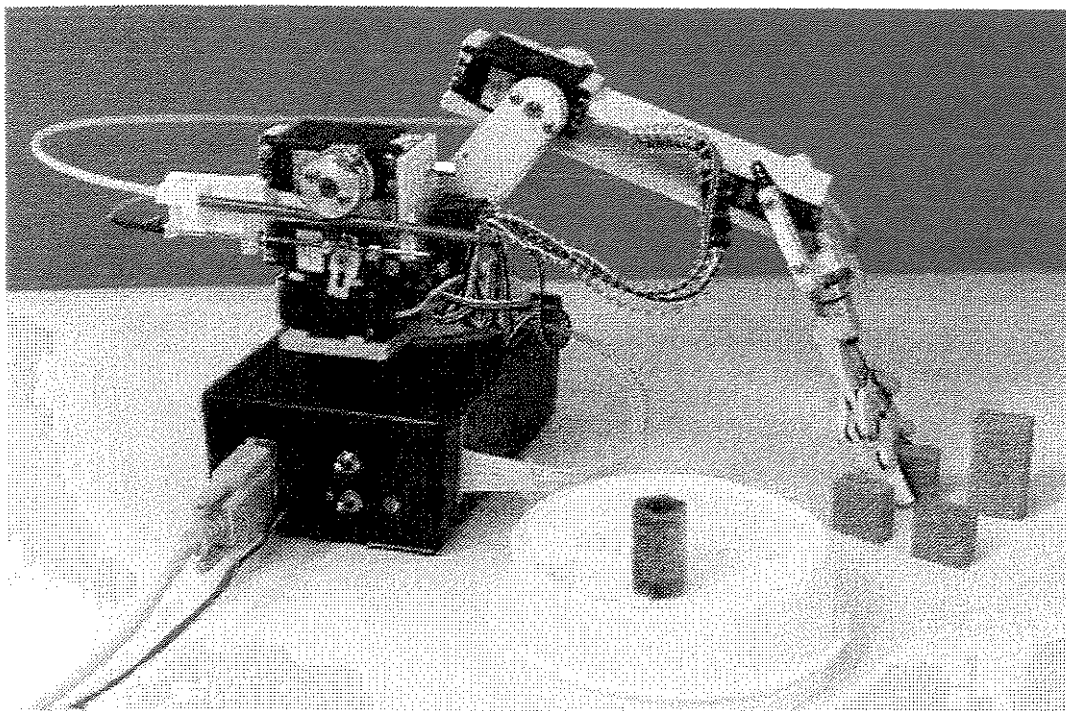


Figura 3.3: O robô modificado.

Como os materiais e tecnologia empregados na parte física do robô são bastante simples, foi possível modificar inteiramente o *kit*, com o intuito de tentar eliminar os problemas acima citados. Foram identificados os seguintes pontos possíveis de serem melhorados:

- Introdução de mais dois graus de liberdade ao robô (rotação do antebraço e rotação da mão), deixando-o com a configuração de um robô Puma.
- Melhora da construção dos elos do robô, permitindo que os servos, que anteriormente eram colados, possam ser removidos para manutenção, além permitir uma fixação mais rígida. O kit apresentou alguns problemas críticos, como a rotação da base que apresentava muito atrito e muita folga, além da construção do elo da mão, que era muito precária.
- Desenvolvimento de sistemas de transferência de movimento que aumentem a amplitude de rotação dos servos, além de permitir que eles sejam montados em locais onde sua massa exerça menos influência sobre o robô.
- Classificação dos tipos de juntas que compõem o robô e desenvolvimento juntas com menor fricção e menor folga, que possam ser aplicadas em projetos semelhantes.

- **Melhora da performance no geral, ampliando a capacidade de carga, velocidade máxima e demais parâmetros de performance.**
- **Projeto de um coprocessador capaz de operar portas de entrada e saída e com melhor precisão para o posicionamento das juntas, possibilitando inclusive uma redução de custo.**

3.2 Parte operativa

Esta seção mostra aspectos construtivos do manipulador que foi desenvolvido, mas sem a preocupação em fazer uma documentação muito detalhada uma vez que, como foi citado anteriormente, a parte operativa é especializada e consequentemente esta é uma solução particular ao problema mais amplo de se criar um manipulador robótico educacional.

O objetivo é demonstrar a factibilidade da construção de um robô funcional e de custo reduzido, atendendo aos propósitos estabelecidos para este trabalho. O anexo V traz com maiores detalhes aspectos construtivos do protótipo construído.

A especificação do robô deve partir do tipo de acionamento escolhido, que irá determinar as características do coprocessador e a forma de construção da estrutura do manipulador.

A parte operativa de um manipulador robótico é essencialmente um conjunto de elos e juntas dispostos em uma determinada configuração.

As juntas são os "músculos" do robô, sendo responsáveis por realizar trabalho e criar os movimentos. Cada junta é um sistema mecânico, formado por um motor e (normalmente) uma redução, um *encoder* e outros elementos mecânicos, como rolamentos, guias etc.. As juntas podem ser rotativas, caso introduzam um grau de liberdade de rotação, ou prismáticas, caso introduzam um grau de liberdade de deslocamento.

Os motores usados para acionamento das juntas de um robô são de alta performance e normalmente são servomotores de diversos tipos, ou motores de passo. Eventualmente, ao invés de motores, são utilizados outros dispositivos de acionamento, como pistões hidráulicos ou a ar

comprimido.

As reduções utilizadas em robótica devem ser isentas de folga, para garantir o correto posicionamento da junta e evitar vibração. Um exemplo é a redução harmônica.

Os *encoders* realizam a leitura de posição da junta e usualmente são de leitura ótica, dividindo a amplitude de movimento em um determinado número de incrementos.

Os elos por sua vez, têm como principal papel oferecer sustentação mecânica às juntas. Normalmente são construídos em alumínio para redução de massa e devem apresentar uma elevada rigidez para garantir a precisão dos movimentos do robô.

3.2.1 Medidas de performance e características de manipuladores

Para realizar análise comparativa entre manipuladores, e também para avaliar a adequação de um manipulador para realizar determinada tarefa, são usadas diversas medidas de performance.

3.2.1.a) Graus de liberdade

O número de graus de liberdade determina a capacidade ou não do robô em combinar orientação e posicionamento de sua extremidade. Um robô que possua menos de 6 GDL sempre possuirá alguma restrição à orientação de sua extremidade; um robô com 6 ou mais GDL poderá ou não possuir restrições à orientação, dependendo da configuração cinemática do robô e do posicionamento relativo ao objeto.

É importante ressaltar que a abertura e fechamento da garra de um robô não é contada como sendo um grau de liberdade. Usa-se robôs com menos de 6 GDL tipicamente em tarefas de movimentação de peças ou paletização, que requerem apenas 3 GDL.

Robôs com 6 GDL são usados em tarefas mais complexas, tipicamente: soldagem, pintura, aplicação de selante, rebarbação e montagem. Eventualmente, robôs com 7 ou mais GDL são utilizados para possibilitar ao robô desviar de obstáculos que poderiam impedi-lo de alcançar um determinado ponto.

Além disso, freqüentemente graus de liberdade externos são adicionados ao sistema, movimentando a peça trabalhada ou o próprio robô, que poderá estar montado sobre um trilho móvel.

Em algumas configurações de robôs, pode haver um grau de liberdade que somente seja comandável para duas posições. Por exemplo, um robô pode ter sua garra comandada apenas para a posição "erguida" ou "abaixada". Isto é considerado um "meio grau de liberdade", pois não é oferecido pleno controle sobre a junta.

3.2.1.b) Volume de trabalho

O volume de trabalho indica o volume dentro do qual o robô poderá posicionar o elemento terminal. É separado em volume destro (no qual o robô é capaz de orientar plenamente a ferramenta) e volume de orientação restrita, no qual haverá restrições para a orientação do elemento terminal.

Na maioria das vezes, o robô opera em um volume com restrições de orientação da ferramenta, de modo que é necessário planejamento cuidadoso das tarefas a se realizar.

3.2.1.c) Velocidade de movimento

A velocidade máxima de movimento é um dos parâmetros de performance importantes, sendo relevante principalmente nos movimentos de grande amplitude, nos quais o robô poderá atingir seu limite de velocidade.

3.2.1.d) Aceleração

A aceleração é um parâmetro importante para movimentos de pequena amplitude, nos quais há constantemente aceleração e desaceleração. Permite determinar a produtividade de um robô, um fator que pesa muito na avaliação econômica de um manipulador robótico.

3.2.1.e) Capacidade de carga

A capacidade de carga determina o porte de um determinado robô, o que está intimamente relacionado com seu custo. A capacidade de carga pode ser ampliada com a adição de contrapesos ou molas, que aliviam o esforço sobre as juntas. Depende também, da condição de posicionamento das juntas na qual o robô está operando.

3.2.1.f) Acuidade de movimentos

A acuidade de movimentos de um manipulador robótico é uma medida importante de performance, indicando sua capacidade para realização de determinadas tarefas ou a maneira pela qual elas serão estruturadas.

Garras auto-centralizadoras, complacência e guias externas para orientar movimentos são meios de lidar com a falta de acuidade de movimento. Uma estruturação de tarefas bem concebida poderá tranquilamente lidar com grandes imprecisões sem demandar o uso de um robô de alta precisão e conseqüentemente, custo elevado.

É conveniente conceituar a diferença entre erros e incertezas nos movimentos de um robô, sendo que os erros ocorrem sempre de uma mesma maneira toda vez que um mesmo movimento é repetido e conseqüentemente podem ser identificados e compensados durante a programação do robô.

Porém, a ocorrência de erros limita a geração de programas a partir de um modelo

matemático, já que sua avaliação depende da presença física do manipulador robótico. Normalmente, um programa gerado a partir de um modelo matemático é primeiramente testado e ajustado para compensar a ocorrência de erros.

Os erros podem ser resultado das tolerâncias de fabricação que resultam em alterações na geometria do robô, em relação ao que foi projetado. Este erro pode ser eliminado através de um procedimento de identificação de parâmetros.

Também podem ser resultado de esforços causados pelo peso próprio do robô, do objeto manipulado e forças resultantes das acelerações. No caso do robô implementado, o controlador utilizado é do tipo proporcional, logo é incapaz de eliminar erros estáticos. Isto somado à elasticidade dos elos leva a erros deste tipo.

Além disso, é preciso ter em conta que o controlador não é capaz de gerar a trajetória desejada com exatidão. Por exemplo, como o controlador que não possui modelo cinemático inverso não é possível gerar trajetórias retilíneas e a diferença entre a trajetória executada e a desejada é um erro de posicionamento.

Erros nos transientes ocorrem devido à excitação causada pelo início ou fim de um movimento e são rapidamente estabilizados. O principal deles é o *overshooting*, um fenômeno que ocorre na maioria dos controles e indica uma perda de precisão de posicionamento num curto período após um movimento, e pode causar uma colisão no final do movimento.

Para se aumentar a produtividade de um robô, grandes deslocamentos que não requerem precisão são realizados a alta velocidade e no final desses, certas precauções devem ser tomadas para que haja tempo suficiente para estabilização do manipulador antes de requerer precisão de movimento.

O tempo de estabilização indica o tempo demandado por certo mecanismo para, após um estímulo, atingir um determinado parâmetro de precisão. Este parâmetro serve para determinar

eventuais tempos de espera na programação dos movimento, para que haja estabilização da posição antes de realizar um movimento no qual se requer grande precisão.

Por outro lado, as incertezas nos movimentos ocorrem de forma aleatória, logo a não é possível compensá-las e somente pode-se avaliar sua intensidade para verificar se estas não irão causar problemas na execução das tarefas.

A perda de repetibilidade ocorre fundamentalmente devido à influência de esforços externos, como vibração e devido a forças internas, como o atrito nas juntas. Este componente normalmente é desconsiderado em robôs industriais, porém nos robôs avaliados, de construção simplificada, torna-se um fator relevante.

Além disso, folgas nas juntas, erros de leitura nos *encoders* (principalmente no caso do robô desenvolvido neste trabalho, que utiliza resistores variáveis) e eventuais variações na função de transferência das juntas (por exemplo, devido a flutuações na tensão de alimentação) e inconsistência na geração de trajetórias, que dependendo do algoritmo utilizado, poderá criar resultados diferentes para a execução de um mesmo movimento, levam a incertezas no movimento e ao comprometimento da repetibilidade do manipulador.

O atrito entre as juntas gera um efeito de histerese, ou seja, o manipulador adquire uma "memória", que irá forçar o servo para fora da posição comandada. Além disto, servos mal fixados, elos demasiadamente elásticos ou juntas com folga ou mal alinhadas contribuem para causar perda de repetibilidade.

3.2.2 Escolha do acionamento do robô

Embora o modelo adquirido já apresentasse uma solução para acionamento das juntas, foram avaliadas outras possibilidades, que eventualmente poderiam ter sido utilizadas em uma ou mais juntas do robô.

O uso de servomotores tipo industrial de pequeno porte representa uma solução factível, porém leva a um custo elevado da parte construtiva do robô, requerendo uma caixa de transmissão robusta e um controlador de boa qualidade.

Outra possibilidade seria a utilização de motores de passo, que podem ser adquiridos no mercado a um baixo custo. É possível criar um circuito de controle bastante simples e que funcione adequadamente.

Neste caso seria necessário fazer uso de algum sistema de transmissão de movimento entre o motor e a junta, uma vez que o torque não é suficientemente elevado para o acionamento direto.

Uma outra possibilidade que foi avaliada é a criação de servos utilizando um conjunto de motor CC e redução, adquirido comercialmente e implementando-se uma malha de controle utilizando um microcontrolador.

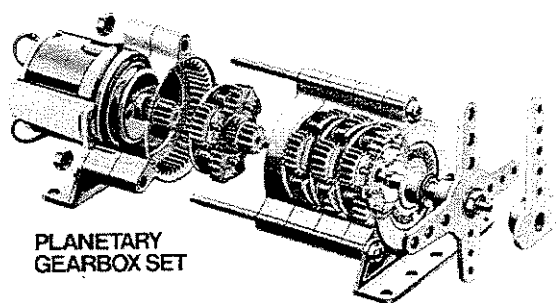


Figura 3.4: Conjunto motor-redução produzido pela Tamy.

Existem várias opções de motores CC com caixas de redução à venda no mercado por preços reduzidos, que poderiam ser utilizados para mover as juntas do manipulador robótico. Em sua maioria, foram desenvolvidos para construção de brinquedos elétricos ou robôs autônomos. Uma possibilidade interessante são os motores usados para acionar vidros elétricos de carros, que possuem peso reduzido e velocidade adequada.

Também foi cogitado o uso de atuadores construídos usando ligas de memória de forma. Foram feitos alguns testes, entre eles, com uma mola de tração, que à temperatura ambiente tem

um comportamento complacente, podendo ser esticada com facilidade, e quando aquecida pela passagem de uma corrente elétrica se contrai, podendo movimentar uma junta do robô.

A figura 3.5 mostra a mola esticada à temperatura ambiente e a recuperação da forma original após aquecimento acima de 70° C.

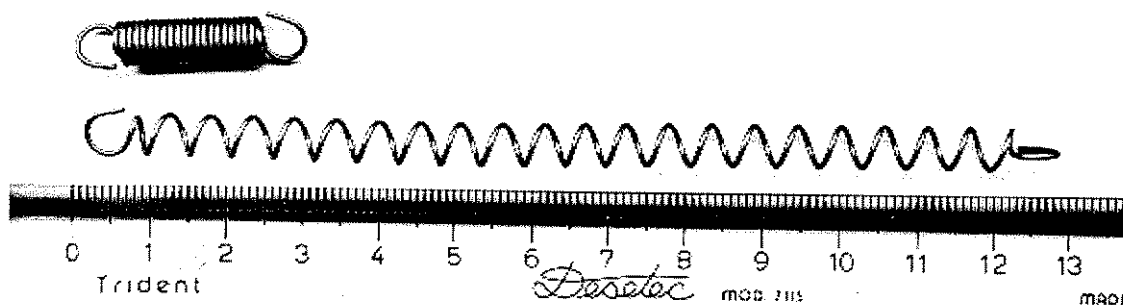


Figura 3.5: Mola de tração construída em liga de memória de forma de Ni-Ti.

Esta é uma possibilidade viável porém o tempo de resposta é lento e seria necessário trabalhar com uma tensão elétrica muito baixa, o que dificultaria a implementação do sistema elétrico, devido a necessidade de ter de lidar com uma corrente grande e problemas com quedas de tensão nos semicondutores e na transmissão.

Além disso, a implementação de um controlador representa um problema complexo e ainda em desenvolvimento. A grande vantagem deste material em relação aos motores elétricos é seu peso extremamente reduzido.

Possivelmente, este material se mostre adequado para fazer o sistema de fechamento da garra, que poderia ser implementado com um controlador muito simples, possivelmente até um controle de malha aberta, visto que não há necessidade de um controle preciso sobre o fechamento da garra.

Além disso, o fato de ser um atuador linear ao invés de rotativo leva a uma simplificação no projeto da garra do manipulador.

Finalmente, a solução utilizada no *kit* adquirido e também que pareceu ser a mais adequada são os servos desenvolvidos para modelos de controle remoto. Neste trabalho, são denominados apenas como "servos", mas é preciso se ter em mente que não são iguais aos servomotores normalmente usados em manipuladores robóticos.

Estes servos foram a princípio desenvolvidos para uso em aeromodelismo, um *hobby* bastante divulgado nos EUA e graças a isso existe uma vasta gama de modelos disponíveis no mercado, que podem ser adquiridos em lojas especializadas no Brasil. O preço é bastante acessível e varia bastante, dependendo das características do modelo escolhido. Os utilizados neste robô foram os mais baratos.

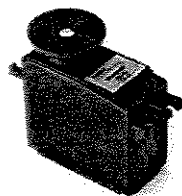


Figura 3.6: Servo em perspectiva.

Cada servo possui uma caixa plástica que abriga um pequeno motor elétrico, uma redução, um *encoder* e um controlador. A caixa plástica, além da função estrutural de permitir a fixação do servo e sustentar seus componentes internos, protege as partes sensíveis contra poeira e outros contaminantes.

Na figura 3.7 pode-se ver o pequeno controlador junto ao fundo da caixa. O conjunto de engrenagens fica protegido pela tampa da caixa (que possui abas de fixação).

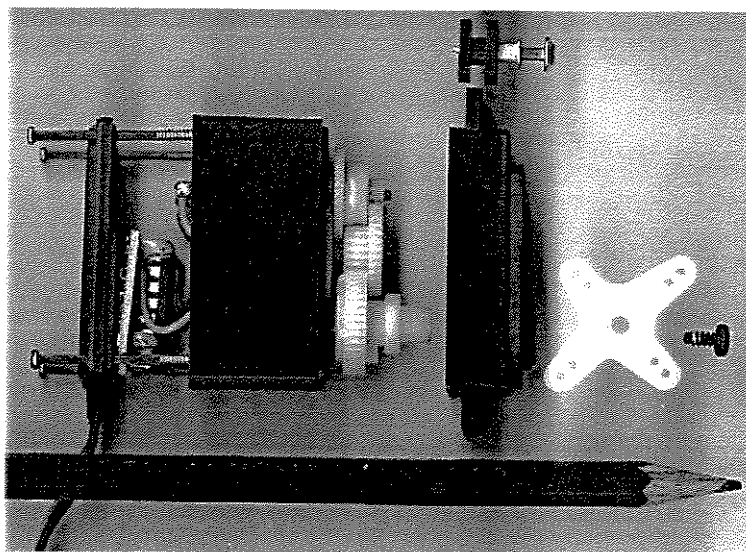


Figura 3.7: Servo desmontado.

Devido ao tipo de aplicação a que se destinam, estes servos possuem um controle muito simples, para ter um baixo custo e tamanho reduzido. O controlador utilizado é analógico, tipo proporcional e discreto no tempo. Um resistor variável ligado ao eixo de saída serve como *encoder*.

O controle de malha fechada é feito internamente no servo, sendo que não há qualquer *feedback* externamente.

Devido ao tipo de controle que possuem (proporcional), estes servos se comportam como se houvesse uma mola forçando o eixo de saída para a posição comandada. Sempre que houver uma força externa atuando, haverá um erro de posicionamento proporcional a esta força.

Isto resulta em excessiva flexibilidade do robô, e imprecisão na posição do elemento terminal, devido à ação da força da gravidade (que, de certa forma pode ser avaliada e compensada) e forças de atrito nas juntas ou forças externas (estas, totalmente imprevisíveis).

Atualmente já existem no mercado modelos de servos que possuem um controlador digital,

porém seu custo ainda é muito elevado, e foi identificado apenas um modelo disponível à venda. Aparentemente, existe certa incompatibilidade com o sinal de comando, em relação aos servos convencionais, o que deveria ser investigado junto ao fabricante antes de se optar por sua utilização.

O servo recebe sinais de comando PWM com nível de tensão entre 0 e +5V, cujo período em estado alto, variando de 1 a 2 milissegundos, determina a posição comandada. Seu controlador não possui memória, logo a ação de controle somente ocorre imediatamente após o envio de um pulso de comando e é preciso repetir periodicamente este sinal para manter o servo na posição.

Devido a estas características, o controlador é discreto no tempo, apesar de ser analógico. Consequentemente, sua função de transferência depende da frequência na qual são enviados os pulsos de comando. Caso o período entre estas repetições seja curto (uma onda quadrada, por exemplo), a rigidez do servo aumenta consideravelmente, melhorando a precisão do robô. Normalmente, o período entre dois pulsos é da ordem de 20 milissegundos.

Além disso, aumentando-se a tensão de alimentação de 4,8 (nominal) para 6,0 volts, por exemplo, é obtido um aumento da ordem de 22% no torque do servo, e uma redução de 15% no tempo de resposta. (fonte: especificações técnicas de um fabricante, no URL <http://www.hobbico.com/accys/hcam1000.html> , em 27/09/99)

Porém, aumentando-se a rigidez, do servo, é reduzida sua complacência, aumentando o risco de haver quebra das engrenagens ou queima do motor em decorrência de algum erro durante a operação do robô.

A fixação à estrutura do respectivo elo foi feita através das abas laterais e também comprimindo a caixa do servo sobre o elo. Porém, é recomendável observar a posição do motor interiormente no servo, para permitir dissipação de calor. Ele é orientado com seu eixo paralelo ao eixo de saída do servo, e montado no lado oposto ao do eixo de saída, pois abaixo deste é

montado o potenciômetro que atua como *encoder*, e consome espaço interno na caixa do servo.

A ligação elétrica é feita através de três fios, um de alimentação de potência, um terra comum e um de sinal. Cada fabricante usa um tipo diferente de terminal, e deve-se tomar cuidado, pois a conexão errada poderá queimar o servo.

O eixo de saída possui um alto torque, de modo que não é necessário o uso de qualquer tipo adicional de redução. Sua amplitude de rotação é da ordem de 180° , variando dependendo do modelo.

O servo de radiocontrole é de fácil utilização, uma vez que já vem com um conjunto de braços e parafusos apropriados. A seguir, vemos várias opções de braços e formas de montagem.

A figura 3.8 mostra diversas formas de transmissão do movimento de um servo. Também pode-se ver um coxim de borracha, montado à aba para fixação e um rolamento que é montado internamente no eixo de saída.

A principal forma de utilização destes servos em aeromodelos é transmitindo o movimento através de cabos de comando flexíveis. Na outra extremidade, estes cabos são ligados a uma alavanca, presa à parte móvel.

Para o robô implementado, foi desenvolvido um sistema de transmissão de movimentos linear, usando um cabo de aço, mais apropriado para uso em robótica (veja no Anexo V).

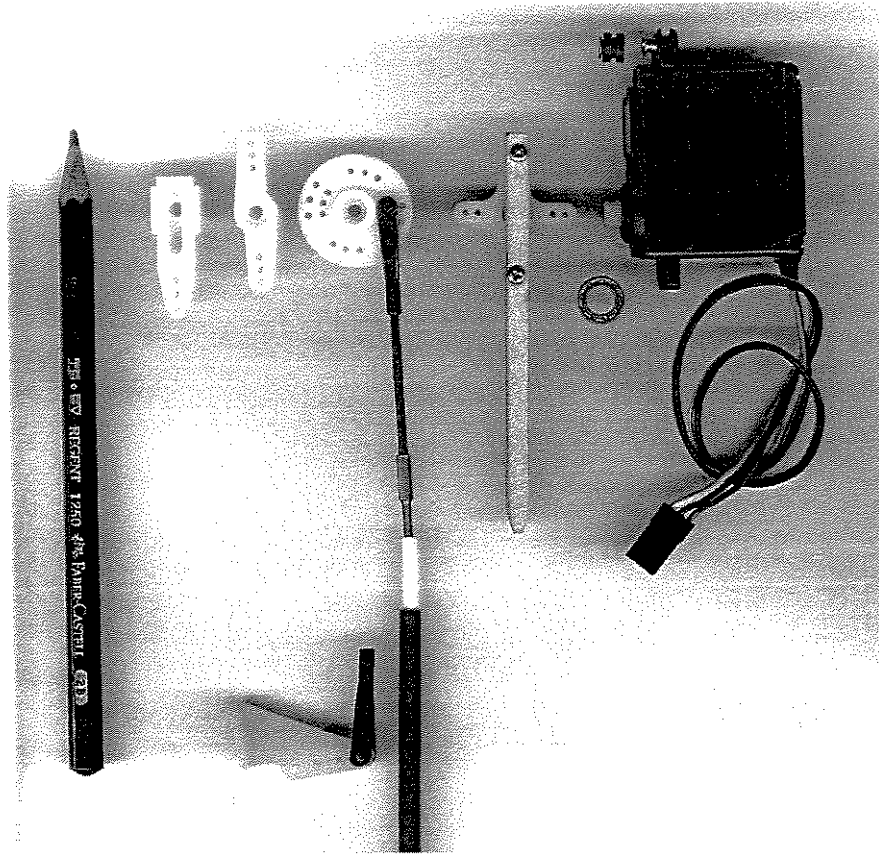


Figura 3.8: Diversas formas de transmitir o movimento de um servo.

Existe uma padronização das características do servo. Por exemplo, o modelo na imagem possui as seguintes características:

Tabela 3.2: Especificações técnicas de um servo.

Fabricante, modelo	Hobbico Stanard Servo
Torque	42 [oz.in]
Tempo de resposta	0.19 sec.@60°
Massa	1.75 [oz]

Este é um modelo convencional quanto ao seu tamanho, velocidade e torque. É também o modelo mais barato produzido pelo fabricante.

O torque é medido em onças x polegada (1 [Oz.in] = 72,008E-3 [Kgf.cm]), quando o servo está parado exercendo força (*stalled*). Existe no mercado servos com torque entre 16 (Hitec HS-60) e 231(Hitec HS-815) Onças x Polegadas. Isto corresponde a 1,15 e 16,63 [Kgf.cm], respectivamente.

O tempo de resposta é o período que o servo leva para realizar uma rotação de 60° (sem torque externo).

A massa é medida em onças (1 [Oz] = 28,350 [g]) e varia de 14 (Hitec HS 60) a 145 (Hitec HS-815) gramas, dependendo do modelo.

Existem modelos especiais com tamanho reduzido, com rolamentos, com engrenagens de metal, com motor *coreless* e de alta velocidade. O preço varia de US\$ 9,99 a US\$ 160,00 (nos EUA, cotação em jan/2000, na loja americana Tower Hobbies), dependendo das características.

Tabela 3.3: Faixa dos valores disponíveis comercialmente.

Preço	US\$9,99 a US\$160,00
Torque	16 a 231 [oz.in]
Tempo de resposta	0.07 a 0,5 sec.@60°
Massa	14 a 145 [oz]

A vida útil das engrenagens é relativamente curta, principalmente se forem de plástico, mas podem ser adquiridas no comércio engrenagens de reposição, a um custo reduzido. De qualquer forma, não é recomendável submeter estes servos a funcionamento ininterrupto e também devem ser evitados esforços desnecessários.

3.2.2 A estrutura dos elos do robô

A estrutura dos elos foi feita em PVC espumado, um material fácil de ser trabalhado, leve e que pode ser colado com cianoacrilato (p/ ex., Super Bonder), devido à sua porosidade. Atualmente, a Tigre, está produzindo tal material no Brasil. Igualmente, poderia ser utilizada

madeira, que possui características semelhantes e possivelmente até superiores.

Foi cogitado o uso alumínio para a construção. Provavelmente, a melhor solução seria o uso de perfis. Uma clara vantagem em relação ao PVC espumado seria melhorar a dissipação de calor, uma vez que o material que foi utilizado é isolante térmico.

A grande desvantagem do alumínio em relação ao PVC é a dificuldade em soldar ou colar as partes. Provavelmente, esta solução se torne vantajosa para a construção de robôs com dimensões maiores que as do robô construído, para os quais o PVC se mostre pouco resistente para a construção dos elos.

É preciso bastante atenção quanto ao peso da estrutura do robô. O simples fato de se colar as partes de PVC que formam o antebraço ao invés de aparafusar, foi responsável pela redução de 6 gramas na estrutura, o que representa 1/3 do peso de um dos micro-servos utilizados. O peso da estrutura do antebraço (sem os servos) ficou em apenas 18 gramas.

O resultado final do aprimoramento do manipulador robótico foi bastante satisfatório, melhorando sua precisão, aumentando os graus de liberdade e reduzindo a massa no antebraço. Por outro lado, houve um aumento da complexidade da construção, que passou a utilizar mais peças, algumas das quais usinadas em metal.

Procurou-se melhorar a performance geral do manipulador através de melhorias na concepção construtiva sem aumentar muito a complexidade do projeto e sem utilizar materiais mais caros. Assim, o resultado final foi um manipulador com características bastante melhoradas em relação ao projeto original e ainda assim, bastante acessível do ponto de vista de custo construtivo.

O Anexo V traz diversos detalhes a respeito da construção do braço robótico, que podem servir como base para futuros desenvolvimentos semelhantes.

3.3 Parte de comando

Robôs industriais podem ser classificados de acordo com seu sistema de controle em quatro tipos básicos: de sequência fixa, de repetição com controle ponto a ponto, de repetição com controle de trajetória contínua e finalmente, robôs inteligentes.

Os de sequência fixa não empregam controladores para determinar a posição das juntas, mas operam baseado em chaves-limite, detetores discretos de posição ou batentes mecânicos. A colocação destes dispositivos determina um número limitado de posições que o robô pode assumir, e o controlador aciona as juntas, fazendo o manipulador passar de uma posição a outra.

O controle independente das juntas oferece a capacidade de controlar posição e velocidade de cada uma das juntas. As forças e momentos decorrentes do acoplamento que existe entre os elos são tratados como perturbações externas. Usualmente, são usados controladores tipo PID.

Outros tipos de controle mais avançados permitem melhorar a performance do manipulador. Alguns exemplos são o controle ótimo, controle adaptativo e controle preditivo.

A Parte de Comando é estruturada de forma hierárquica, sendo composta por subsistemas de controle que atuam em mais baixo nível, que operam em aplicações de tempo real (tempo de resposta pequeno), por exemplo, os controladores das juntas.

Subsistemas que atuam com alto nível de informações possuem uma lógica complexa, e não necessitam de um tempo de resposta tão rápido, como por exemplo, o interpretador de comandos de um robô.

Freqüentemente estes subsistemas são fisicamente implementados em computadores separados, utilizando o tipo de computador mais adequado a cada caso. Esta solução foi utilizada na implementação do manipulador robótico usado para testar o aplicativo: a Parte de Comando está centralizada em um computador PC rodando o aplicativo desenvolvido, mas também é

necessário o uso de um coprocessador para controle de entradas e saídas e para comandar os servos do robô.

3.3.1 Arquitetura do comando de robôs

A funcionalidade de um robô depende, além de sua parte construtiva, da facilidade de programação. Não existe uma linguagem que seja padrão para todos os robôs, porém há grande semelhança entre elas.

Os tipos básicos de linguagens de programação são: Linguagens de programação ponto-a-ponto, linguagens de movimentos básicos no nível Assembler, Linguagens não-estruturadas de alto nível, Linguagens estruturadas de alto nível, linguagens tipo C.N., linguagens orientadas a objetos e linguagens orientadas a tarefas.

3.3.1 a) – Operação por telecomando

A operação por telecomando é a forma menos estruturada de comandar um manipulador robótico. O operador atua diretamente sobre as juntas, fazendo o manipulador realizar alguma tarefa.

Este tipo de utilização é bastante comum nos casos em que o um manipulador é utilizado em substituição direta à aplicação da força humana diretamente. Porém, nestes casos o homem não é substituído na tarefa de controle do manipulador.

3.3.1 b) – Programação por aprendizado e repetição

Para alguns tipos de aplicação, a maneira mais simples de se realizar a programação é "ensinar" o robô a realizar uma seqüência de movimentos, e fazer com que este repita a seqüência linear de movimentos sempre que desejado. Isto é um caso típico para robôs de pintura, que seguem o tipo de programação *"teaching by doing"*.

3.3.1 c) – Programação estruturada

Este tipo de programação faz uso de uma linguagem de programação estruturada, aproveitando seus recursos de criar desvios estruturados, subrotinas e *loops* para controlar o robô na realização de tarefas complexas.

Além de facilitar a construção de um programa, as técnicas de programação estruturada facilitam a tarefa de modificação do mesmo, uma vez que passa a ser necessário modificar somente uma subrotina, ao invés de ser necessário a reedição de todo o programa.

Este tipo de programação permite reagir instantaneamente a mudanças no ambiente de trabalho, através da leitura de sensores. A desvantagem deste processo reside na maior complexidade do código do programa, que requer um programador bem qualificado.

Freqüentemente, a estrutura do programa é criada em *off-line* e depois são inseridas as coordenadas dos pontos usando o robô em modo *teach*, o que é um procedimento relativamente fácil de ser realizado.

Não existe uma padronização nas linguagens de programação estruturada, de um modo geral elas são baseadas em alguma linguagem de programação existente como o Pascal ou Algol, e são acrescentados comandos especiais para o controle de robôs, incluindo: possibilidade de criação de rotinas com execução paralela, cíclica ou limitada pelo tempo, instruções de movimentação do robô, com diferentes tipos de interpolação e parâmetros de movimento; instruções paralelas para dois robôs cooperando, especificação de diferentes trajetórias e instruções para manipulação de sistemas auxiliares, como elemento terminal ou outros acessórios e processamento de sinais de entrada e saída.

As principais linguagens de programação são as seguintes: T3, RPL, VAL, AL, PAL, Funky, Emily, Maple, Autopass, RCL, Help, Anorad, Sigla.

3.3.1 d) – Programação a partir de um modelo

Neste tipo de programação, o uso do manipulador robótico fisicamente para gravar os pontos do programa é substituído pelo uso de um modelo matemático. Isso permite reduzir o tempo improdutivo do robô, deixando-o mais disponível para a produção.

Usualmente, a estrutura principal do programa e os pontos são gravados *off-line*, baseado no modelo matemático, e posteriormente, os pontos são ajustados operando-se diretamente sobre o manipulador, para assegurar o correto funcionamento do programa. O aplicativo desenvolvido neste trabalho insere-se nesta categoria.

3.3.1 e) – Geração automática de programas

O modo mais avançado de gerar programas é através de um sofisticado aplicativo computacional que, a partir de modelos gerados em CAD, permite a simulação de todo o ambiente de trabalho, e a partir dele, fazer a definição de pontos e trajetórias a serem cumpridas. A partir disso, é criado automaticamente o programa que comanda o robô no cumprimento destas etapas.

Este tipo de programação é adequado a tarefas complexas e tem se desenvolvido bastante, com a criação de sofisticados aplicativos que permitem uma grande agilidade na criação de programas, principalmente sendo vantajoso em aplicações de maior complexidade. Operações de solda em peças com geometria complexa ou a coordenação de vários robôs em uma mesma célula de trabalho são alguns exemplos.

Embora nestes casos a geração do programa em si seja facilitada, em contrapartida é preciso que o ambiente de trabalho esteja também modelado em computador e que as tarefas estejam bem estruturadas.

3.3.1 f) – Integração com outros dispositivos

O robô, como parte integrante de um sistema de produção, deve ter capacidade de se integrar facilmente com outros dispositivos, com os quais terá que interagir.

A integração é tanto física (posição de objetos, evitar colisões, ferramentas automatizadas na célula de produção), quanto na estrutura de comando e na troca de informações pelas entradas e saídas, assim como a troca de parâmetros, como a posição do objeto que está sendo trabalhado, que pode estar sendo movido durante a operação.

A forma de integração de um manipulador robótico com outros dispositivos pode ser feita de diversas maneiras, podendo-se citar a mais elementar que é a do controlador do robô realizar periodicamente leituras nas portas de entrada. Este tipo de procedimento é denominado *polling*.

Porém, em algumas situações será necessário que o robô esteja sempre receptivo a algum evento externo durante a operação, o que caracteriza uma programação voltada a eventos. Um caso típico é a implementação de um *loop* de segurança, que se for aberto, deverá causar a parada imediata do robô.

3.3.2 Estruturação da parte de comando

A estrutura de comando implementada representa apenas uma das inúmeras possibilidades de estrutura de comando para controlar o manipulador robótico. O sistema implementado demonstrou ser eficiente para a construção de robôs de pequeno porte e permite a simplificação da parte física do robô, concentrando as atividades de controle no PC, o que reduz sua complexidade.

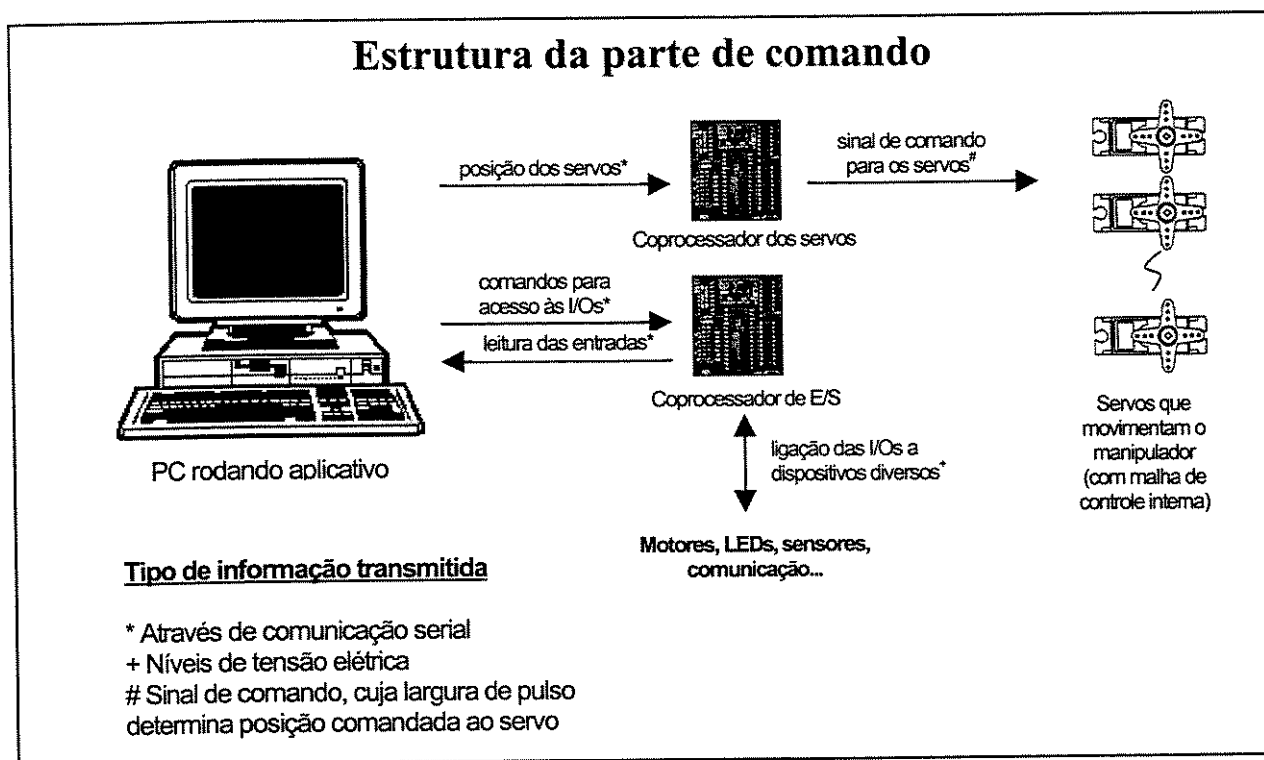


Figura 3.9: Estruturação da parte de comando.

A figura 3.9 mostra como o PC centraliza todas ações de controle, e utiliza coprocessadores para enviar o sinal de comando aos servos e realizar acesso a portas de entrada e saída.

O sistema de controle implementado é centralizado em um PC, que utiliza coprocessadores para tarefas específicas de enviar sinal de comando aos servos e controle das entradas e saídas. A comunicação entre o PC e os coprocessadores é feita através da porta serial.

O coprocessador para comando dos servos recebe pacotes de dados, cada um contendo a posição de um deles, e envia pulsos de comando em intervalos regulares.

Caso se tivesse optado por outro sistema de acionamento das juntas, como servomotores, motores de passo etc., seria necessário projetar um coprocessador que atenda às necessidades

específicas, e possivelmente implementar pequenas modificações no aplicativo computacional.

A concentração da lógica de controle no PC reduz a complexidade do programa do coprocessador, e facilita esta tarefa. O uso de um protocolo de comunicação padronizado permite, inclusive, criar um robô com sistema de acionamento híbrido, composto por juntas com acionadores de diferentes tecnologias.

É possível utilizar um coprocessador para o comando das portas de entrada e saída, e pode ser conectado diretamente a diversos dispositivos, como sensores, motores elétricos, botões, lâmpadas de sinalização.

Ambas as funções de envio de sinal aos servos e controle das portas de entrada e saída podem eventualmente ser realizadas por um único controlador ou por controladores diferentes, conforme a conveniência.

Para implementar a comunicação entre o PC e os coprocessadores, poderia ser usada a porta paralela ou mesmo o barramento do PC. Optou-se pela porta serial, que possui taxa de transmissão de dados suficiente para a aplicação em particular, e normalmente está disponível nos computadores, facilitando a conexão.

3.3.3 Serial Servo Controller (SSC)

Este foi o coprocessador testado para enviar sinal de comando aos servos do robô. O SSC é um produto comercial, baseado em um PIC 16C61 pré-programado pelo fabricante. Este microcontrolador em particular possui 36 *bytes* de memória RAM e 13 pinos de I/O, além de contador e *timer*.

Microcontroladores são dispositivos que incorporam um microprocessador, memória, portas de entrada e saída e circuitos eletrônicos de apoio em um único chip. Este tipo de solução oferece além de simplicidade de projeto, uma solução miniaturizada e de baixo custo para

problemas de controle.

O SSC permite gerar os sinais de comando para 8 servos simultaneamente, e com resolução de 8 bits para o posicionamento. O sistema de endereçamento permite conectar até 16 destes controladores simultaneamente em uma mesma porta serial.

A opção de adquirir um coprocessador disponível no mercado é a maneira mais rápida de conseguir colocar o sistema em funcionamento, ao passo que a opção de desenvolver um coprocessador consome mais tempo e demanda uma certa infraestrutura em equipamentos e programas que permitam desenvolver, debugar e gravar um programa em um microcontrolador.

Porém, adquirir uma solução pronta implica em não ser possível alterar as características do controlador, que se torna uma "caixa preta". O SSC, apesar de funcionar adequadamente, possui alguns problemas, que sugerem que seria mais vantajoso desenvolver por conta própria um controlador com características semelhantes:

O SSC possui resolução de 8 bits para o posicionamento dos servos, para comandar uma amplitude de movimento próxima a 180 graus. Logo, o incremento de posicionamento para um servo é: $180^\circ / 256 = 0,7[\text{graus} / \text{incremento}]$.

Considerando para um caso típico que o manipulador robótico esticado tenha um braço de 30 centímetros, neste caso o incremento angular, traduzido em incremento de posicionamento do elemento terminal é de: $300 \times \pi / 256 = 3,68[\text{mlímetros}]$.

Notadamente, este incremento é exageradamente grande. Verificou-se que a precisão construtiva do manipulador pode garantir um erro de posicionamento da ordem de 1 mm.

A resolução de posicionamento dos servos pelo controlador deveria ser, no mínimo, de 10 bits, o que levaria a um incremento linear um pouco inferior a 1 mm para o caso descrito acima. Apesar de ser um valor muito grande, se comparado a robôs industriais, seria da ordem de

grandeza da incerteza no posicionamento verificada no manipulador construído.

Outra característica que poderia ser melhorada é aumentar a frequência de emissão dos pulsos, que como foi verificado, levaria a um aumento da rigidez dos servos e consequentemente, melhor precisão de movimentos.

Capítulo 4

Descrição do aplicativo computacional

Este capítulo trata do aplicativo computacional desenvolvido, apresentando suas características de utilização, essencialmente o que o usuário final precisa conhecer utilizá-lo.

Neste capítulo não são abordados os detalhes internos de seu funcionamento. Uma descrição mais detalhada a respeito de seus aspectos internos e organização dos formulários e subrotinas que o compõem está no Anexo II.

O aplicativo foi escrito em Visual BasicTM, e fornece um ambiente integrado para a simulação e operação de um robô. Ele oferece uma linguagem de programação interpretada, manipulação direta do robô usando a *Teach Box* e um simulador gráfico da movimentação do robô.

A figura 4.1 mostra as diversas janelas que compõem o aplicativo, dispostas no interior do formulário principal.

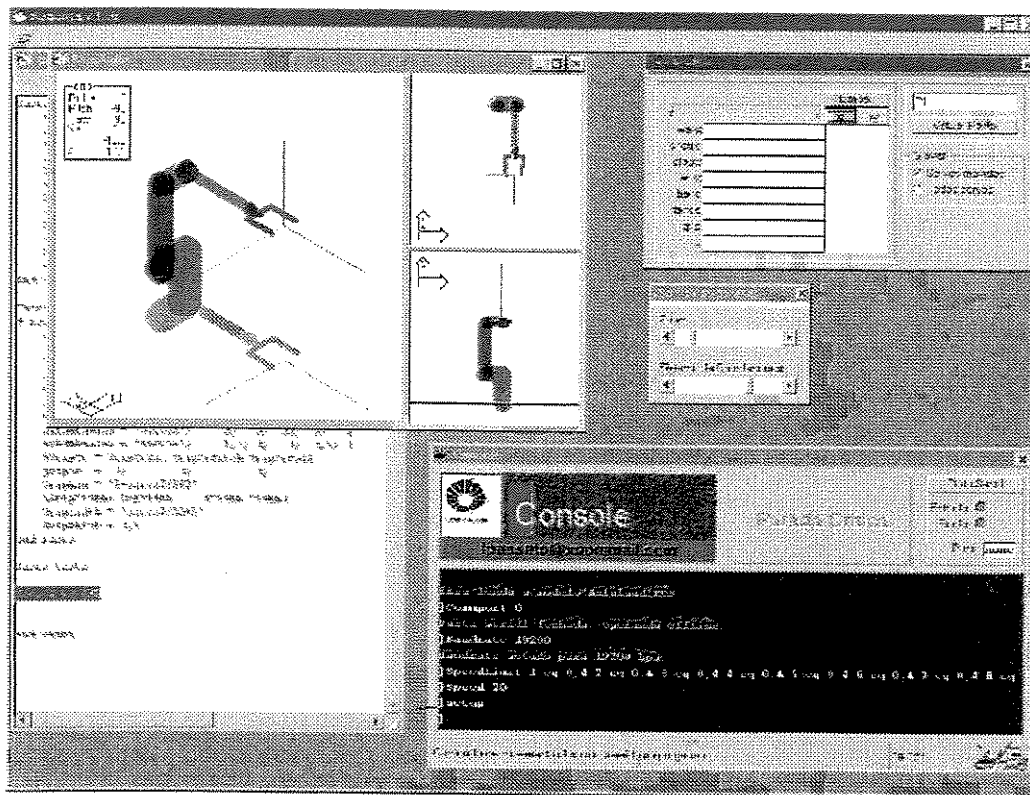


Figura 4.1: Imagem do aplicativo como um todo.

4.1 Concepção do trabalho

No desenvolvimento deste trabalho, vários conceitos foram adotados, para orientar sua concepção. Estes estão enumerados e descritos para que se tornam claros os fundamentos e objetivos que levaram ao resultado obtido.

4.1 a) Interface com o usuário

O programa fornece constantemente mensagens coerentes com que está ocorrendo, mantendo o usuário constantemente informado.

Muitos programas produzidos comercialmente são de funcionamento obscuro ou fornecem mensagens de erro incompreensíveis ou até mesmo incorretas. Quando o usuário se depara com qualquer problema, sua solução torna-se complicada, pela dificuldade em identificar o que

realmente o ocasionou.

O programa possui uma barra de *status* na janela Console, que constantemente fornece mensagens indicando ações do usuário ou eventual ocorrência de erros. Quando a estrutura do programa não permite identificar exatamente o erro, a mensagem emitida será igualmente ambígua.

Além disso, toda a configuração do robô é feita através de comandos especiais da linguagem de programação, eliminando as caixas de entrada, que freqüentemente trazem confusão ao usuário.

4.1 b) Resistência a erros

O mal uso por parte do usuário não deve causar a interrupção do programa, nem resultar em funcionamento inesperado ou inconsistente, que possa danificar o equipamento. O programa verifica as ações do usuário e na medida do possível, procura evitar o uso inadequado.

4.1 c) Modularidade

O programa é composto por diversas janelas, cada uma cumprindo uma função específica. Seu funcionamento é independente e os canais de troca de informação (chamadas de rotinas ou variáveis), claramente definidas. O mesmo ocorre com as subrotinas.

4.1 d) Uso do teclado

O aplicativo combina funções acionadas pelo *mouse* com comandos e navegação via teclado, numa maneira semelhante ao que ocorre nos programas de CAD. Os comandos escritos e configurações podem ser executados via código no próprio programa criado pelo usuário.

4.1 e) Genericidade

O aplicativo é dentro do possível, genérico. Não foi construído para ser aplicado a uma situação específica e sim permitir o máximo de flexibilidade possível, o que possibilita configurar o aplicativo para controlar virtualmente qualquer manipulador robótico que se deseje, sem necessidade de alteração do código-fonte.

4.1 f) Adaptabilidade

Como não se trata de um programa comercial, o código-fonte é aberto, permitindo a fácil adaptação a tarefas que não estejam inicialmente previstas. O uso de uma linguagem de programação bastante divulgada (Visual Basic), a documentação detalhada de seu funcionamento e a estruturação das partes que o compõem tornam esta tarefa relativamente fácil.

4.1 g) Acessibilidade

Tanto o aplicativo quanto a parte operativa utilizados devem ser acessíveis a um baixo custo, permitindo o uso extensivo em atividades educacionais, sem demandar investimentos volumosos na aquisição de equipamento.

Devido à simplicidade da construção, será possível construir robôs de diferentes configurações e usando diferentes tecnologias de acionamento a um baixo custo e sem necessidade de se utilizar equipamentos sofisticados.

A concentração das ações de controle no PC permite a simplificação do coprocessador que controla o robô, facilitando seu projeto e reduzindo seu custo.

4.2 Os módulos que compõem o aplicativo

Nesta parte serão descritos os módulos que compõem o aplicativo. Cada um é representado

por uma janela, quando o aplicativo está sendo executado. O aplicativo computacional oferece um ambiente amigável e possui uma estrutura modular, sendo formado por diversas janelas contidas em uma moldura principal: Console, Programa, *Teach-in Box* e Simulador.

Cada um desses módulos cumpre um papel específico e o usuário deverá conhecer as funções que são inseridas em cada uma delas. Além da modularidade, esta disposição em várias janelas permite que elas se sobreponham, otimizando o uso da área da tela.

A mudança de foco de uma janela para a outra pode ser feita facilmente, pressionando-se as teclas F2, F3, F4, F5.

Console é o núcleo do programa, representando o painel de comando do robô. Assim como ocorre em ambientes de desenvolvimento como o Matlab, o Console possui uma memória que armazena variáveis, que são mantidas mesmo após a execução de um programa e aceita comandos de execução imediata ou executados a partir de uma subrotina. Esta é uma característica das linguagens de programação interpretadas. Já as linguagens baseadas em um compilador apenas executam comandos contidos no código do programa e o valor das variáveis é apagado após a execução de um programa.

O módulo Programa funciona como um editor do programa residente na memória do robô, semelhante aos que compõe as linguagens de programação, possui comandos especiais para indentar o texto.

O módulo *Teach* representa a *Teach-in-box*, reunindo controles para mover o robô e para inserção automática de linhas no programa. Normalmente, a *Teach-in-box* é uma caixa de comandos construída fisicamente, que atua como um *joystick*. Neste caso, para simplificação do projeto, foi criada uma *Teach-in-box* virtual, na qual os comandos são operados pelo *mouse* e pelo teclado.

Finalmente, o módulo Simulador representa uma simulação do robô, permitindo a

utilização do aplicativo sem estar conectado a um manipulador robótico, ou auxiliando na programação de um robô real.

4.2.1 O módulo Console

O Console é o núcleo deste aplicativo e reúne as funções centrais, como o interpretador de comandos e o sistema de controle do robô. Possui uma área denominada Caixa Imediata, que funciona de modo semelhante a um *prompt* de DOS para receber comandos via teclado.

Existe a preocupação de se deixar a operação do robô o mais clara possível ao usuário, de modo que o programa constantemente emite mensagens ao usuário através de um painel, na parte inferior da janela. Estas mensagens foram criadas baseadas na estrutura do programa, de modo que sempre são coerentes com o que realmente está acontecendo.

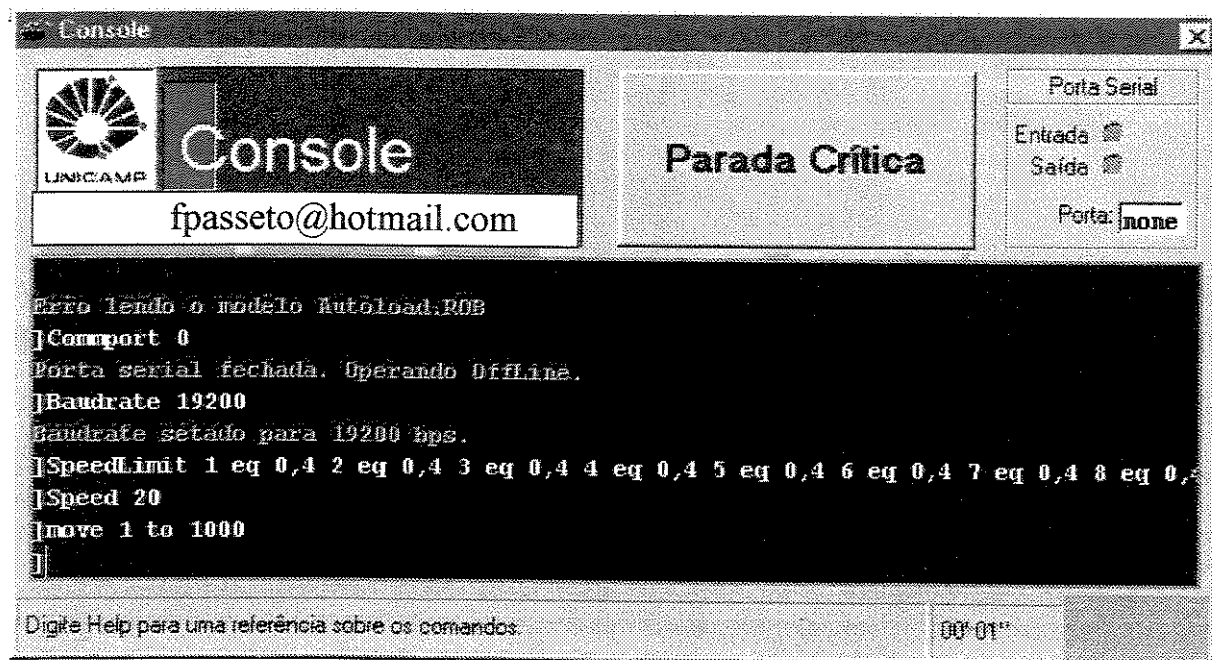


Figura 4.2: Janela do módulo Console, o núcleo do aplicativo.

Toda vez que se pressiona a tecla <Enter>, a linha de comando é executada. Caso o comando seja chamada de uma macro, a estrutura de todo o programa residente na memória é

mapeada. Consequentemente, se houver um erro na estrutura de qualquer parte do programa, será emitida uma mensagem de erro cancelando a execução, mesmo que o erro esteja fora da macro requisitada.

Além de executar os comandos nela digitados, a área imediata devolve mensagens escritas, que registram ocorrências relevantes ao usuário. Estas mensagens são exibidas em verde, para distinguir dos comandos digitados pelo usuário, que aparecem em branco.

Digitando:

```
]help commands  
]help operators  
]help functions  
]help comport
```

Pode-se ver um sistema de *help* simplificado que descreve brevemente o funcionamento dos comandos, operadores e funções.

O comando *Print* exibe uma mensagem na janela imediata, permitindo avaliar o resultado de expressões matemáticas, o valor de variáveis, além de exibir mensagens durante a execução de um programa.

Para simplificar a digitação de um mesmo comando sucessivas vezes, existe implementado um sistema semelhante ao *DosKey*. Pressionando-se a seta para cima, a linha de entrada é preenchida com entradas anteriores.

Na parte inferior da janela existe uma barra de *status*, que constantemente exibe mensagens ao usuário, além de outras informações relevantes, como operação *On-line* ou *Off-line* etc.

Na parte superior da janela são exibidos alguns dados relevantes, como a porta serial utilizada e a taxa de transmissão. Próximo a esta, existe um botão de parada crítica, que

interrompe imediatamente a execução de um programa, caso seja necessário.

Além disto, existem menus que oferecem outras funções ou atalhos a comandos que podem ser digitados na janela imediata. Por exemplo, a porta *Comm* pode ser selecionada via *menu drop-down* ou digitando o comando *Commport*.

4.2.1 a) Porta Serial

Permite escolher a porta *Comm* que será usada para comunicação com o coprocessador. A princípio, nenhuma porta é escolhida. Sempre que for escolhida a opção **none**, a comunicação serial é desligada, e o aplicativo opera somente no modo simulação (mesmo que o robô esteja conectado ao PC).

Quando é selecionada uma porta serial, os coprocessadores são atualizados com a posição atual das juntas e portas de saída do aplicativo.

4.2.1 b) Protocolo

O aplicativo atualmente possui implementado apenas um protocolo para comunicação com o coprocessador SSC (coprocessador utilizado para o robô).

Caso este protocolo não satisfaça as necessidades de uma determinada aplicação, pode-se incluir outros protocolos de comunicação no aplicativo, modificando seu código-fonte.

Protocolo do SSC:

255 + Address + Data ...

Onde: 255 é um *Byte* de sincronismo, *Address* indica o servo cuja posição será modificada, e *Data* é a posição que será atribuída ao servo.

Formado da transmissão: Taxa de 2400 ou 9600 *bps*, sem paridade, 8 bits de dados e 1 *stop bit*.

4.2.1 b) Servos Conectados...

Este comando abre uma caixa de entrada que permite determinar o número de servos que estão conectados ao(s) controlador(es). Os servos não conectados não são habilitados na *teach-in-box* para modificação e nem são referidos quando a *teach-in-box* insere uma linha no programa. São porém acessados se seu endereço for citado no comando *move*.

4.2.2 O módulo Programa

O módulo Programa é um editor de texto para o programa que está sendo criado. Toda vez que o aplicativo é carregado, a janela Programa irá receber o programa contido no arquivo **Autoload.ROB**. Este programa deve ser utilizado para armazenar a configuração inicial do robô, e pode servir de modelo para criação de novos programas.

Para facilitar a compreensão da estrutura do programa que está sendo criado, foi implementado um recurso de indentamento automático ou manual. Pressionando-se <Shift> + <Espaço> é inserido um indentamento e <Shift> + apaga um nível de indentamento.

Para executar uma das linhas do programa, basta dar um duplo clique sobre ela. Isto é bastante útil durante a edição de programas.

Na barra de menus, existem os comandos convencionais para a leitura e gravação de programas, além de comandos para navegação, para trazer para frente uma das outras janelas.

Além disso, este módulo possui os comandos *cut*, *copy* e *paste* usuais dos processadores de texto.

4.2.3 O módulo *Teach-in Box*

O módulo *Teach-in Box* permite operar diretamente o robô (ou o modelo, tanto faz) para realizar a aprendizagem das posições. Uma vez posicionado o robô, ao se pressionar um dos botões (Inserir Ponto), (To) ou (By), a posição será registrada no programa, através da inserção automática de uma linha na posição onde se encontra o cursor.

É considerada como sendo a origem do movimento a posição do robô quando foi pressionado pela última vez o botão (Limpar), (to), (by) ou (Gravar Ponto).

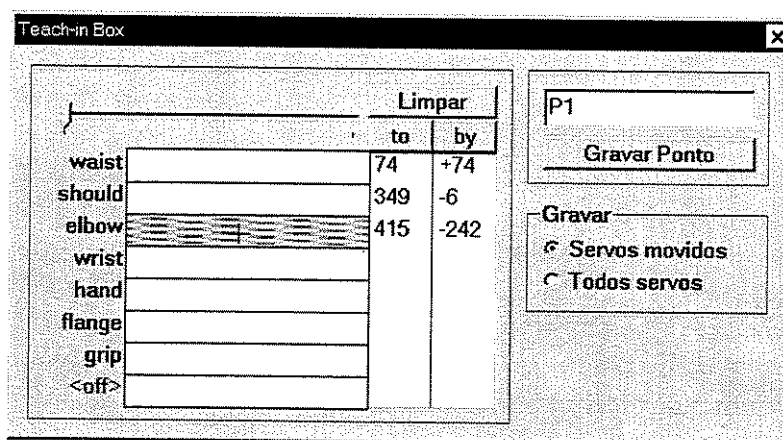


Figura 4.3: *Teach-in Box* permite mover as juntas do robô e gravar posições para criar programas.

Quando a movimentação de uma junta é referida como *to*, será memorizado um movimento absoluto para a posição, e como *by*, será efetuado um movimento relativo à posição anterior do robô (um conjunto de incrementos de posição das juntas). A opção *by* é útil para a criação de programas estruturados. Pode-se misturar as opções *to* e *by* em um mesmo comando Move, sem problemas.

Outra opção é definir um ponto que irá representar o estado das juntas do robô, um outro recurso utilizado na criação de programas estruturados: o nome da variável aparece em uma caixa de texto e possui um número que será incrementado automaticamente quando se faz a inserção de sucessivos pontos.

Para maior clareza, está exemplificada a movimentação do robô utilizando o comando move diretamente e utilizando definição de pontos:

Através do comando move:

```
move 1 to 0 2 by 50 3 to 235
```

Definindo pontos:

```
p1 = "1 to 0 2 by 50 3 to 235 "  
move p1
```

As juntas não mencionadas continuam na mesma posição. Pode-se concatenar dois movimentos em um único, referindo-se a dois pontos em um único comando move. Suponha que o ponto p2 se refira ao fechamento da garra do robô.

```
p2 = "7 to 20"  
move p1 & p2
```

Este comando faz o robô executar os movimentos de p1 e p2 concatenados (o que é diferente de executar um movimento para p1 e após, um movimento para p2).

Existe uma opção de gravar todas as posições das juntas do robô ou apenas as juntas que foram movidas. Esta é mais uma opção de criação de programas estruturados, pois permite realizar uma mesma sequência de movimentos com dois valores distintos para uma junta que não é citada pelo comando move. Note-se que a opção de gravar apenas os servos modificados, ou todos, não interfere na performance da movimentação do robô.

4.2.4 O módulo Simulador

Um dos módulos do aplicativo desenvolvido é um simulador, capaz de criar uma representação gráfica do manipulador robótico e calcular a posição e orientação do elemento terminal.

Este simulador, além de abordar grande parte da teoria sobre modelagem de manipuladores, permite a criação de programas baseados em modelos, mostrando o funcionamento de diferentes tipos de manipuladores que não necessariamente existam fisicamente. Além disso, o modelo cinemático poderá servir como base para a criação de um modelo cinemático inverso (que não foi implementado devido à sua complexidade).

A programação de um manipulador robótico pode ser feita através do procedimento denominado "*teach by doing*", que consiste em mover o próprio braço robótico para determinar as coordenadas das juntas correspondentes a cada posição desejada para o robô.

A partir de um modelo cinemático do manipulador robótico, pode-se fazer um procedimento análogo, porém baseado na movimentação deste modelo matemático, ao invés de se utilizar o próprio manipulador robótico. Isto dispensa o uso do robô no procedimento de criação de programas, reduzindo o tempo improdutivo e também permite ao programador testar e *debugar* o programa sem correr o risco de danificar o equipamento.

Usualmente, quando um programa é criado usando um modelo será preciso testá-lo na prática, e realizar ajustes em alguns pontos do movimento. Como a estrutura do programa já está criada e os pontos já definidos de uma forma aproximada, este procedimento será rápido e seguro.

O simulador utiliza um modelo baseado nos parâmetros de Denavit-Hartenberg para desenhar uma representação gráfica do robô na tela do computador, além de calcular posição e orientação da garra ou uma das juntas.

A inserção do modelo no simulador merece uma atenção especial, sendo que o Anexo IV foi inteiramente dedicado à teoria relacionada ao modelo cinemático de um manipulador e ao procedimento de inserção de um modelo no aplicativo e também traz exemplos mostrando a modelagem das principais configurações cinemáticas de manipuladores.

A função deste módulo é fornecer, da melhor forma possível, um *feedback* a respeito do que se passa com o robô. Seguindo a filosofia de genericidade do aplicativo, o simulador é capaz de modelar qualquer configuração de manipulador, bastando para isto inserir os dados que compõem o modelo.

Este simulador não se propõe a criar um modelo complexo do ponto de vista visual e sim, utilizar um procedimento que na medida do possível seja o mais simples possível, facilitando a criação de modelos pelo próprio usuário.

Note-se que como o aplicativo é modular, esta janela foi inicialmente criada como um programa em separado e posteriormente inserida no aplicativo, após algumas alterações que permitem troca de dados com o módulo principal, o Painei.

A exibição de modelos tridimensionais no monitor pode se tornar a imagem um tanto confusa ao usuário e para minimizar este problema, foram implementados diversos recursos que poderão auxiliar a compreensão do modelo em questão. A cada um destes recursos existe um comando de configuração para desabilitá-lo, se necessário:

4.2.4 a) Simulação > Pontos de Vista

Determina se a imagem é desenhada no simulador apenas na perspectiva isométrica, ou nas perspectivas isométrica, topo e lateral (padrão).

4.2.4 b) Simulação > Linhas Finas

Habilitando esta função, todo o modelo é desenhado com linhas finas, o que será útil principalmente quando se está traçando os movimentos, evitando a sobreposição de linhas.

O robô é desenhado na tela como se fosse transparente, permitindo visualizar a posição de partes do robô que estariam ocultas por uma parte do robô mais à frente na perspectiva..

O desenho da espessura das linhas utiliza uma função interna do Windows, conseqüentemente não consome muito tempo de processamento e não tem grande influência sobre a velocidade de regeneração das imagens.

4.2.4 c) Simulação > Plotagem 3-D

Este é um recurso bastante interessante que desenha imagens estéreo para serem visualizadas usando óculos coloridos. Veja no Anexo I, item.5.1 o desenvolvimento da teoria relacionada, que foi usado para criar este recurso. A figura 4.4 mostra como a perspectiva isométrica passa a desenhando imagens estéreo em azul e vermelho, as outras perspectivas desenhando a imagem em preto.

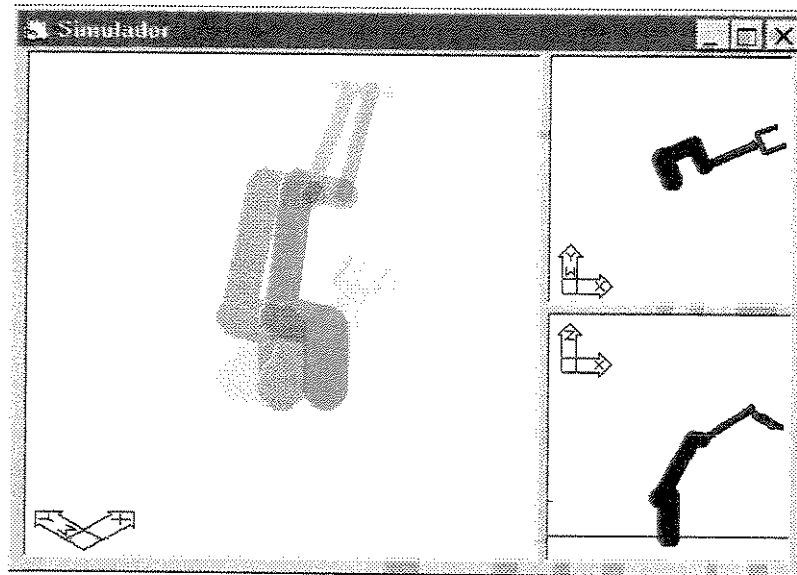


Figura 4.4: Desenho do modelo do robô em 3-D.

Uma vez habilitado este recurso, a cor das linhas passa a ser desconsiderada (transformada em tons de cinza), e a perspectiva isométrica passa a ser desenhada sob duas perspectivas levemente diferentes, em duas cores.

Principalmente, quando se está traçando o movimento do robô com linhas finas, esta função produz um efeito bastante interessante. A imagem criada em 3-D, desenhada com linhas finas

permite visualizar o volume de trabalho do robô.

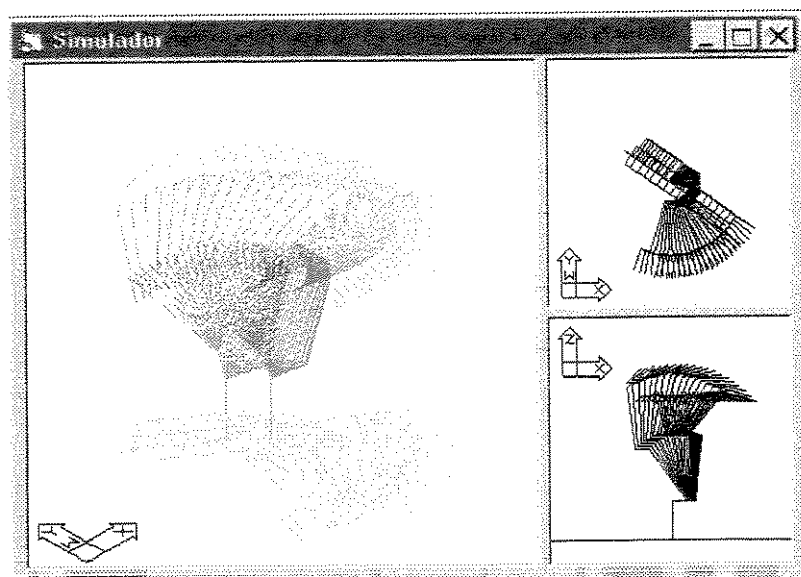


Figura 4.5: Recurso de traçar trajetória do robô.

Esta função não consome muito tempo de processamento, uma vez que somente a rotina de desenhar linhas na tela é afetada e não existe necessidade de recalcular os parâmetros de DH para criar as duas perspectivas.

O modelo do robô é desenhado como se fosse uma radiografia, ou seja, todo ele é transparente. Caso partes do robô fiquem sobrepostas, a imagem ainda assim pode ser distinguida, pois o programa atribui tons diferentes a cada parte do robô, começando do mais escuro na base do robô para o mais claro no *grip*. É preciso configurar o monitor em *High Color* ou superior, para evitar que a extremidade do robô, desenhada em tons mais claros, seja apagada.

4.2.4 d) Simulação > 3-D Paralaxe...

Esta função determina o paralaxe (distância entre as duas perspectivas 3-D), que projeta a imagem à frente do monitor, tornando a imagem mais fácil de ser visualizada. Só tem efeito quando a plotagem em 3-D está habilitada.

4.2.4 e) Simulação > Auto Regenerar

Esta opção habilita um modo de exibição de imagens no Windows, que transforma as linhas traçadas em um *bitmap* armazenado na memória do computador.

Isto produz dois efeitos positivos: se uma janela for arrastada à frente da imagem no simulador, a imagem será restaurada automaticamente. Caso esta opção não esteja habilitada, a imagem no simulador será apagada e regenerada automaticamente após algum tempo.

Além disto, a imagem é exibida após todas as linhas que a compõem terem sido traçadas, o que elimina o efeito da imagem ficar piscando ou aparecer apagada nas linhas desenhadas por último.

Porém, esta função afeta a performance do simulador, reduzindo a taxa de regeneração da imagem. Esta opção é desabilitada quando a opção Traçar Grip Apenas está ligada.

4.2.4 f) Simulação > Traçar Todo Robô

Desenha sucessivas imagens sem apagar a anterior. Isto é um recurso interessante para desenhar em uma única imagem toda a trajetória de uma movimentação do robô.

É recomendável usar linhas finas para criar a imagem quando se está traçando os movimentos, para evitar que as linhas se sobreponham, causando uma imagem confusa.

Fazendo-se um "duplo clique" sobre a imagem, limpa-se a tela.

4.2.4 g) Simulação > Traçar Grip Apenas

Esta é uma opção interessante, que traça a trajetória de um ponto localizado na extremidade da garra e regenera o desenho do robô, permitindo registrar somente a trajetória do elemento terminal. Para apagar a trajetória desenhada, deve-se fazer um "duplo clique" sobre uma das

molduras do simulador.

A figura 4.6 mostra como este recurso permite, inclusive, verificar a influência da resolução dos servos no posicionamento do robô. Neste caso, foi usada uma resolução de 8 *bits* para a rotação de 360 graus das juntas.

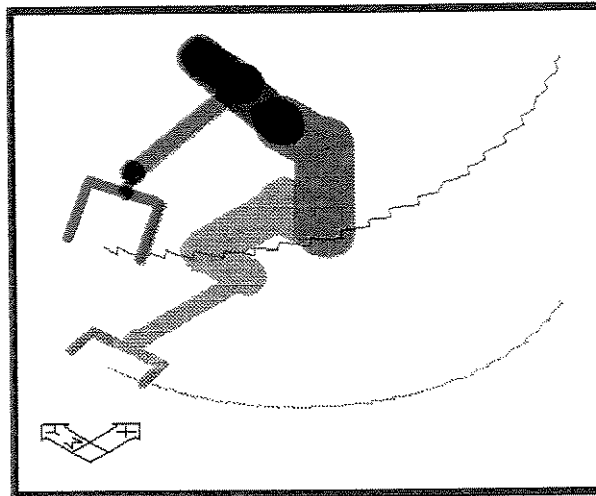


Figura 4.6: *O registro da trajetória do grip.*

Como o desenho do modelo do robô é feito em multitarefa, a frequência da regeneração do modelo normalmente é inferior que a frequência de envio de dados pela porta serial, sendo que a primeira gira em torno de 15 quadros por segundo e a segunda, vale 320 modificações de posição de servo por segundo (a 9600 BPS com o SSC).

Consequentemente, a representação da trajetória do *grip* desenhada graficamente não é uma representação exata da trajetória comandada aos servos.

Porém, se for utilizada uma velocidade muito baixa para o robô, a velocidade de regeneração do modelo se tornará superior à frequência de mudança da posição dos servos, e será possível observar exatamente a trajetória comandada para a extremidade do robô.

Isto dá uma noção da excitação vibratória que ocorre quando o robô está em movimento e

também da influência do número de incrementos do sistema de controle na precisão do robô.

4.2.4 h) Simulação > Efeito de Sombra

O efeito de sombra projeta todas as linhas desenhadas sobre o plano $Z=0$ em tom cinza. O efeito criado é similar ao de uma sombra causada por uma luz pontual, a uma distância infinita acima do modelo.

Este efeito ajuda bastante na interpretação da imagem reduzindo a ambigüidade que ocorre quando o modelo tridimensional é "achutado" na tela do computador, como pode ser constatado na figura 4.7.

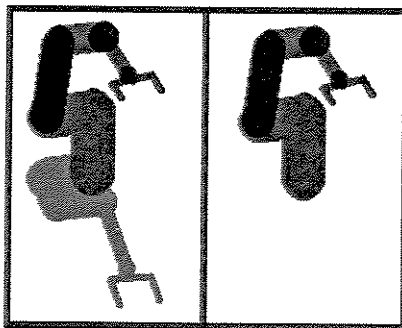


Figura 4.7: O efeito de sombra na perspectiva isométrica.

O efeito de sombra pode ser desligado, o que em algumas ocasiões evita um número excessivo de linhas sendo desenhado, tornando a imagem mais compreensível. Porém, não afeta de forma significativa a performance do simulador, uma vez que age somente sobre a rotina que desenha as linhas que representam o robô.

4.2.4 i) Simulação > Mostrar Grip

O *grip* desenhado é do tipo convencional, formado a partir de 3 linhas. Seja l o comprimento do *grip* (variável **griplen**) e w sua abertura (variável **gripwidth**), as linhas que formam o *grip* são desenhadas no sistema de coordenadas do *grip* através dos seguintes

segmentos de reta:

$(0, w, l) - (0, w, 0)$ (desenhado em vermelho escuro)

$(0, w, 0) - (0, -w, 0)$ (desenhado em vermelho claro)

$(0, -w, 0) - (0, -w, l)$ (desenhado em vermelho claro)

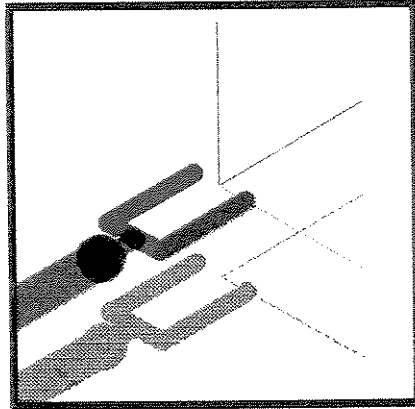


Figura 4.8: *A garra do robô desenhada no simulador.*
A garra é representada em vermelho, tendo a linha mais escura em Y positivo.

Como a garra do robô é simétrica, uma das linhas é desenhada em cor mais escura, permitindo facilmente identificar sua correta orientação.

4.2.4 j) Simulação > Tamanho dos Eixos...

Este comando no Simulador permite escolher o tamanho dos eixos cartesianos que representam o sistema de coordenadas.

4.2.4 k) Simulação > Verificar Performance do Simulador

Este comando realiza um teste para avaliar a taxa de regeneração da imagem pelo simulador, fornece o número de quadros por segundo e uma avaliação do resultado. A partir de testes com diversos modelos de robôs, foi constatado que é desejável utilizar um processador Pentium 166 ou superior, para obter uma boa qualidade da animação do modelo em movimento.

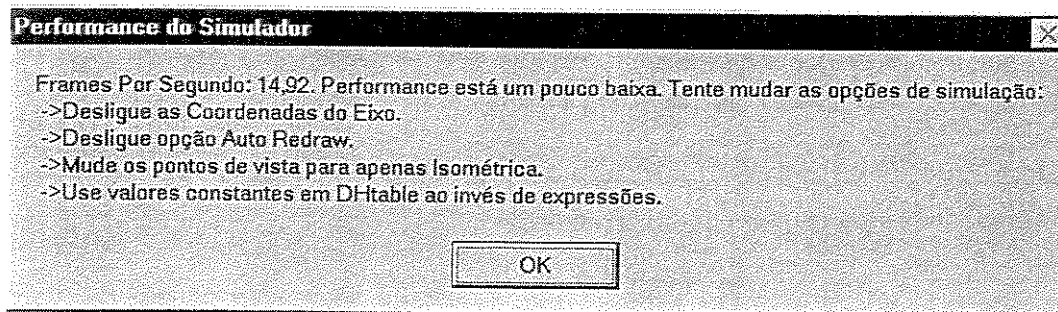


Figura 4.9: Avaliação da performance do simulador e sugestões para melhorá-la.

A performance da placa de vídeo exerce grande influência sobre a taxa de regeneração, visto que apesar de o número de operações matemáticas para cálculo do modelo ser bastante elevado, na verdade a regeneração das imagens é que acaba por consumir a maior parte do tempo de processamento.

4.2.4 I) Sistema de coordenadas:

Esta função é acionada a partir de uma pequena janela, que flutua na área de trabalho como uma barra de ferramentas (para exibir, clicar em Ver > Zoom e Coordenadas). Este controle determina qual dos sistemas de coordenadas será exibido, ou se selecionado em zero, não exibirá nenhum deles.

Modificando-se um botão de seleção, será exibido um dos sistemas de coordenadas (eixos X,Y e Z) que fazem parte do modelo de Denavit&Hartenberg do robô. Este recurso foi criado para facilitar a compreensão do modelo, permitindo visualizar as sucessivas transformações homogêneas que compõem o modelo. Há um sistema de coordenadas inercial, um sistema correspondente a cada junta e um sobre a extremidade da garra do robô.

Os eixos x, y e z são desenhados respectivamente em vermelho, verde e azul, numa alusão ao sistema de cores RGB).

Esta função faz surgir na tela do simulador, sobre a perspectiva isométrica, um quadro que

exibe a posição e orientação RPY do elo em questão. Este quadro pode ser arrastado dentro da moldura para o lugar que for mais conveniente.

grip	
Roll =	151
Pitch =	-63
Yaw =	-1
X =	6,0
Y =	-3,4
Z =	18,5

Figura 4.10: Orientação e posição dos sistemas de coordenadas que compõem o modelo do robô.

O cálculo da orientação inversa consome um número considerável de operações matemáticas, além da exibição dos valores, que também consome tempo de processamento. Por isso, esta função influi de forma significativa na performance do simulador, e deverá ser desabilitada caso haja necessidade de acelerar o processamento.

4.2.4 m) Zoom:

Este comando é acionado a partir de uma janela que flutua sobre a área de trabalho. O *zoom* amplia ou reduz o tamanho no qual o modelo é desenhado nas três perspectivas. A perspectiva isométrica sempre apresenta uma escala duas vezes maior que as outras duas perspectivas.

4.2.4 n) Outros recursos do simulador:

Outros recursos foram criados para facilitar a visualização do modelo no simulador:

A imagem do modelo pode ser arrastada para qualquer lugar, em cada uma das três perspectivas, possibilitando centralizá-la da melhor forma possível.

Fazendo um duplo clique sobre qualquer uma das molduras causa a regeneração da imagem.

Ao se modificar as proporções da janela do simulador, as molduras internas são automaticamente redimensionadas e o modelo, centralizado em cada uma das perspectivas.

4.3 A linguagem de programação do aplicativo

O aplicativo computacional utiliza uma linguagem de programação estruturada cuja sintaxe foi baseada no *Visual Basic*. A linguagem é composta por comandos, operadores e funções e variáveis especiais.

Um programa é composto por macros e estas por sua vez possuem diversas sentenças. Sentença é uma unidade sintaticamente completa que representa um tipo de ação, declaração ou definição.

Os *loops* `For...Next`, `If...Then...Else`, `Do...Loop Until/While` e `Do Until/While...Loop` são desvios condicionais que formam a estrutura interna de cada macro.

Uma macro pode executar outra, chamando-a pelo seu nome. A passagem de valores entre elas é feita pelo intermédio de variáveis.

Por se tratar de uma linguagem integrada em um ambiente de desenvolvimento, as variáveis são automaticamente declaradas quando a elas é atribuído um valor. Não existem regras de escopo das variáveis e além disto, não é feita diferenciação entre tipos de variáveis.

A execução de um programa é iniciada digitando-se o nome de uma das macros na área de execução imediata do console.

4.3.1 Regras gerais para sintaxe da linguagem de programação

A linguagem de programação não é sensível ao caso (maiúsculas ou minúsculas), sendo que todo texto do programa, exceto o que estiver entre aspas, será convertido para minúsculas durante

a execução.

Toda linha de programa possui alguns argumentos que são processados pelo operador matemático e devem obedecer a determinadas regras de sintaxe. Além disso, também será necessário que estejam adequados ao que a sentença aceita como argumentos.

São argumentos válidos para o operador matemático:

- **Valores numéricos, com vírgula separando decimais.**
- **Nomes de variáveis às quais haja um valor atribuído (e assim recursivamente, até chegar a uma expressão matemática numérica).**
- **Argumentos como To, By, Eq... (Palavras reconhecidas pela rotina isArgument)**
- **Texto entre aspas**
- **Expressões matemáticas a serem avaliadas a posteriori, separadas por apóstrofes.**

4.3.2 Comandos

A primeira palavra de uma linha de programa pode ser um comando, o nome de uma macro ou ainda nome de uma variável à qual se está atribuindo um valor.

No caso de ser um comando, as palavras seguintes representam os argumentos passados a este. Os argumentos podem ser passados como um valor numérico ou como uma expressão matemática, que será avaliada no momento da execução.

A linguagem de programação inclui, além dos comandos tradicionais das linguagens de programação, comandos especiais para controle do manipulador robótico, configuração etc.

No Anexo III há uma breve descrição de todos os comandos que compõem a linguagem de programação do aplicativo.

4.3.3 Operadores

Os operadores matemáticos transformam os operandos, que são as palavras logo anterior e a logo posterior ao operador, em um novo valor. O formato geral é:

operando1 operador operando2 \Rightarrow resultado

Exemplo: $5 + 2 \Rightarrow 7$

A hierarquia dos operadores determina a ordem na qual eles normalmente são avaliados (a não ser que sejam colocados parênteses para forçar outra ordem para avaliação dos operadores). Numa hierarquia superior estão as funções, depois a exponenciação (operador $^$), logo abaixo vêm a multiplicação e divisão ($*$ $/$) numa mesma hierarquia, avaliados na ordem em que aparecem e por último, a adição e subtração ($+$ $-$).

Para obter uma listagem dos operadores pode-se recorrer ao *help* interno do aplicativo. Digitando:

```
]help operators
```

Tem-se uma lista dos operadores implementados na linguagem de programação do aplicativo.

4.3.4 Funções

As funções transformam o elemento seguinte no resultado da função. O formato geral é:

função argumento \Rightarrow resultado

Exemplo: $\cos 60 \Rightarrow 0,5$

As funções estão em nível mais alto de hierarquia que os operadores e são avaliadas

primeiro. Logo, caso o argumento de uma função seja uma expressão matemática, será necessário o uso de parênteses para forçar que primeiramente seja avaliada esta expressão.

Também é uma função o sinal de menos, quando indicar que um número é negativo. Esta é uma ambigüidade inerente à grafia normalmente utilizada e poderá causar confusão em alguns casos. A sintaxe para definição de valores para variáveis foi estruturada de modo a eliminar este problema de ambigüidade.

4.3.5 Usando variáveis

O aplicativo possui uma memória que armazena valores de variáveis independente da execução de um programa, de forma semelhante ao Matlab. A definição de variáveis é feita automaticamente ao se atribuir um valor.

Caso uma expressão utilize o valor de uma variável que não foi definida, será gerado um erro, indicando que a palavra não foi reconhecida. São nomes válidos para variáveis palavras iniciadas com uma letra de A a Z, que não sejam nome de um comando ou função, nem uma das seguintes palavras reservadas: "to", "by", "eq", "step", "then". A subrotina *IsArgument* define quais palavras são argumentos e não podem ser usadas como nome de variável. Os nomes de variáveis não diferenciam letras maiúsculas de minúsculas.

A uma variável pode ser atribuído um valor numérico, resultado da avaliação de uma expressão matemática; uma expressão de matemática simbólica ou uma *string* de texto.

Quando uma variável recebe uma expressão de matemática simbólica, o valor da expressão somente será avaliado quando for solicitado o valor da variável. Em muitas situações, isto substitui o uso de funções (subrotinas tipo *function*), de uma forma compacta e elegante.

A avaliação das expressões é feita de forma recursiva, e aceita até 200 níveis de variáveis contendo expressões que reportem outras variáveis (a limitação de 200 níveis foi imposta para o

caso de haver uma referência circular). O aplicativo irá decodificando até chegar a um valor numérico.

A atribuição de uma expressão a uma variável é feita usando-se apóstrofes como delimitadores. Por exemplo, se fizermos:

```
]a=1+3 %Atribuição de um valor numérico a uma variável, resultado da
      avaliação de uma expressão matemática.
]b='2*a' %Atribuição de uma expressão simbólica a uma variável.
]Print b
8
]a=3
Print b %O valor de b é o resultado da avaliação da expressão '2*a', que
      depende do valor de a.
6
]Show b
2+a
```

O uso de expressões simbólicas estabelece uma relação entre o valor de uma variável e outra. Este recurso será utilizado mais adiante para construção do modelo do manipulador a partir de expressões matemáticas simbólicas.

Uma variável pode, também, conter uma *string* de texto, para emitir mensagens durante a execução de um programa. Elas são delimitadas por aspas, e podem ser concatenadas através do operador "&".

O uso de aspas é obrigatório em textos, e entre outras coisas, evita que o texto seja convertido para caixa baixa (minúsculas), como ocorre com o restante do código do programa. Por exemplo,

```
]a = 234
]print "A variável vale :" a
A variável vale : 234
```

A barra de *status* exibe a mensagem: “Comando Print: 2 argumento(s)”, indicando que haviam dois argumentos sendo exibidos.

Variáveis também podem conter múltiplos argumentos, característica usada para definição de pontos, por exemplo.

A avaliação destas variáveis resulta nos diversos argumentos que ela contém. Assim, a definição de pontos para movimentos é feita da seguinte maneira:

```
]p1 = 1 to 100 2 to 25  
]move p1
```

O resultado equivale à seguinte sentença:

```
]move 1 to 100 2 to 25
```

Além disso, é possível fazer alguns outros usos, por exemplo usar dois ou mais pontos como argumento de um comando move.

Exemplo:

```
]p1 = 1 to 100 2 to 25  
]a = 60  
]p2 equ 4 to 25 5 to a + 3  
]move p1 p2
```

O resultado equivale à seguinte sentença:

```
]move 1 to 100 2 to 25 4 to 25 5 to 63
```

Alternativamente, uma variável poderá receber um vetor como argumento. Os diversos elementos do vetor são separados pelo caractere ponto-e-vírgula. Isto será usado para definição do modelo do robô.

O aplicativo possui variáveis especiais, algumas tipo “somente leitura”, outras “escrita e

leitura”. Elas são usadas para armazenar configurações ou para ler valores especiais.

Alguns exemplos de variáveis somente leitura são a constante **pi**, a variável **time**, que retorna o número de segundos decorridos desde a meia-noite, as variáveis **servo1**, **servo2** ... **servon**, que retornam a posição do servo e as variáveis relacionadas ao valor das portas de entrada dos coprocessadores.

Outras variáveis especiais de escrita e leitura armazenam configurações, ou são relacionadas ao valor de uma porta de saída, como as variáveis **servoConstant1**, **servoConstant2**, ... **servoConstantn**.

4.4 Inserindo o modelo do robô no simulador

O aplicativo computacional é totalmente configurável sem que haja a necessidade de alteração no seu código-fonte. Os parâmetros do modelo são fornecidos pelo usuário através de variáveis especiais, usando-se o procedimento convencional para definição de variáveis. Ou em alguns casos, usando comandos de configuração.

É recomendável a criação de uma macro programa contendo todos os comandos de configuração para o robô. Uma vez lido um programa do disco, esta macro deve ser executada, para que os dados do modelo sejam transmitidos ao aplicativo.

Como já foi visto no capítulo 3, em "Adaptação do modelo de Denavit&Hartenberg para uso computacional", inicialmente é necessário converter os valores enviados pela porta serial ao controlador em valores relacionados a uma unidade de medida, angular ou linear, dependendo do tipo de junta implementada.

A transformação é linear, dada por: $s_i = a_i \cdot c_i + b_i$, sendo que as constantes a_i e b_i são determinadas empiricamente, a partir de medidas no manipulador. A variável s_i representa a posição do servo, e a variável c_i representa o valor enviado ao controlado pela porta serial.

Estas constantes são passadas ao aplicativo através das variáveis especiais **ServoConstant1**, **ServoConstant2**, ... **ServoConstant n** . A cada uma destas variáveis é atribuído um vetor contendo valores constantes separados por um caractere ponto-e-vírgula. O primeiro é a constante a_i e o segundo, a constante b_i . Eventualmente, estes valores podem ser vindos da avaliação de uma expressão, porém a expressão em si não será aceita.

Por exemplo, suponhamos que tenha sido convencionado que os servos têm seus ângulos medidos em graus, e que a partir de medições empíricas se tenha determinado que o servo 1 tem o ângulo de 10 graus na posição 17 e de 178 graus na posição 1023.

```
]s1=10    %angulo na posicao 1
]c1=17    %respectivo valor enviado ao controlador
]s2=178   %angulo na posicao 1
]c2=1023  %respectivo valor enviado ao controlador
]ServoConstant1 = (s2-s1)/(c2-c1) ; s2-((s2-s1)/(c2-c1))*c2 %a;b
]print ServoConstant1
8,79765395894428e-02 ; 0
```

Assim, é feita a conversão do valor que enviado pela porta serial ao controlador dos servos para o ângulo ou posição do servo. Por *default*, as variáveis **ServoConstant i** possuem o valor "1 ; 0", até que um valor diferente seja definido.

As variáveis especiais **servo1~servon** contêm os valores atualizados de posição para cada servo, e podem ser utilizadas dentro de qualquer expressão matemática.

É importante notar que a definição de **ServoConstant i** somente afeta o simulador, pois os valores de posição dos servos que são usados na *teach-in-box* e para definição dos pontos correspondem aos valores que são enviados ao controlador.

Os parâmetros de Denavit & Hartenberg e outros parâmetros de interesse para o simulador

são definidos usando-se o que foi denominado neste trabalho de Tabela de Denavit & Hartenberg Extendida. Esta tabela possui além das quatro colunas convencionais, outras seis colunas de preenchimento opcional, e por isso colocadas à direita, totalizando dez colunas.

O formato geral é:

$$\text{ExtDHtable}i = \text{ExprTheta}i; \text{EXPRd}i; ai; \text{alfa}i; a_{\text{width}}i; d_{\text{width}}i; k_{\text{Theta}}i; \text{Theta}0i; k_d i; d_0i$$

Cada coluna tem o seguinte significado:

ExprTheta*i*: Expressão que define o valor de **Teta*i***, em função de valores de posição de servo (**servo*i***), variáveis internas e funções matemáticas.

EXPRd*i*: Similar a **ExprTheta*i***, para **d**.

ai: = Valor numérico da constante **ai**. O valor pode ser obtido a partir do valor de uma expressão matemática que será avaliada no momento da definição do valor de **ai**.

Alfa*i*: Similar a **ai**, para **Alfa**.

a_{width}*i*: Diâmetro do elo no comprimento **a**.

d_{width}*i*: Diâmetro do elo no comprimento **d**.

K_{Theta}*i*: Constante contendo valor fixo que é multiplicador de **Teta*i***.

Theta₀*i*: Constante contendo valor somado a de **Teta*i***.

Kd*i*: constante contendo valor fixo que é multiplicador de **di**.

d₀*i*: Constante contendo valor somado a **di**.

Os valores de **TETA** e **D** serão obtidos a cada momento a partir das expressões:

$$\text{Teta}i = \text{ExprTheta}i(\text{servo}1 \sim \text{servon}, \text{variáveis}) + K_{\text{Theta}}i * \text{servo}i + \text{Theta}0i$$

$$di = \text{EXPRd}(\text{servo}1 \sim \text{servon}, \text{variáveis}) + K_d * \text{servo}i + d_0i$$

Como pode ser notado, as constantes **K_{Theta}*i*** e **Theta₀*i*** podem ser omitidas, uma vez que

ExprTheta i é uma expressão matemática genérica, e pode englobar também a função destas duas constantes. Porém, computacionalmente a avaliação de expressões matemáticas consome muito tempo e o uso destas duas constantes pode reduzir a complexidade da expressão **ExprTheta i** ou freqüentemente dispensar seu uso reduzindo-a a zero. O raciocínio análogo vale se aplicado a **d**.

Para verificar se as expressões foram inseridas corretamente, basta usar o comando `Print`, que exibe a expressão completa que será usada na criação do modelo. Por exemplo:

```
]ExtDHtable1 = 0 ; 0 ; 10 ; 90 ; 2 ; 2 ; 1 ; 90 ; 0 ; 0 %ExprTh; EXPRd; ai;
               alfai; aW; dW; kTh; Th0; kdi; d0i
]Show ExtDHtable1
'servol+90' ; 0 ; 10 ; 90 ; 2 ; 2
```

O comando `Show` exibiu a expressão simbólica que representa a linha da tabela de parâmetros que será usada para construção do modelo cinemático do robô. Para exibir os valores numéricos resultados da avaliação das expressões inseridas no modelo, deve ser utilizado o comando `Print`.

A quinta coluna determina o diâmetro do elo no comprimento **d** e a sexta coluna, o diâmetro do elo no comprimento **a**. Este recurso foi criado para dar uma dimensão de volume ao manipulador, tornando-o mais realístico e fácil de compreender. Estas colunas podem ser omitidas e neste caso todo o modelo será desenhado com linhas de 1 *pixel* de largura.

Outros valores que definem o modelo cinemático são:

Joints: Número de juntas do modelo (sem contar o *grip*). Não confundir com número de graus de liberdade, pois podem haver várias juntas comandadas por um mesmo servo, formando um único grau de liberdade (veja caso do robô *pick-and-place*, no Anexo IV, item 3.3). Este valor se sobrepõe à existência ou não de um correspondente número de linhas na tabela DH estendida. Foi feito desta maneira para evitar uso incorreto por parte do usuário.

GripOr = GripOrYaw; GripOrPitch; GripOrRoll: Orientação do *grip* em relação ao sistema de coordenadas da extremidade do robô. Aceita somente valores numéricos. Foi escolhido o sistema de orientação *Roll, Pitch, Yaw* porque é o mesmo usado para mostrar a orientação da garra em relação ao sistema de coordenadas inercial.

GripThick: Espessura de linha que será usada para representar a garra do robô.

GripLen = expr(servo1~servon, variáveis): Comprimento total da garra. Pode receber uma expressão matemática ou um valor numérico.

Width0GripWidth = expr(servo1~servon, variáveis) ; kWdth ; Width0: Expressão que determina a abertura do *grip*, a partir de expressão literal e/ou constantes. A largura do *grip* é dada pela expressão:

$$GripWidth = expr + kWdth * servo(joints+1) + Width0$$

Capítulo 5

Conclusões e perspectivas futuras

Este trabalho tratou de um aplicativo computacional desenvolvido para controlar um manipulador robótico de pequeno porte e baixo custo para ser utilizado em atividades de ensino.

Comparado às soluções disponíveis atualmente, os resultados alcançados são superiores, tanto pelas características do aplicativo computacional como também no projeto do manipulador robótico fisicamente.

Mais do que desenvolver um robô em particular, e mostrar a viabilidade deste tipo de aplicação, o objetivo deste trabalho foi criar soluções que permitam construir robôs didáticos a um baixo custo, possivelmente alterando partes do projeto para adequar suas características à necessidade em particular.

O aplicativo computacional possui recursos que o tornam bastante funcional para aplicação em atividades de ensino: Interface com o usuário elaborada, a existência de uma linguagem de programação estruturada e um simulador gráfico para um manipulador são características que tornam o sistema mais funcional, mais abrangente e mais próximo dos robôs industriais.

O aplicativo pode ser utilizado com qualquer configuração cinemática de robô, sem necessidade de alteração de seu código-fonte e as soluções apresentadas permitem a criação de

robôs de acordo com as características que forem desejadas.

Aplicado ao ensino em robótica é possível abordar tópicos como estrutura de programação, metodologia de modelagem cinemática de manipuladores, estudo de diferentes configurações de manipuladores e estruturação de sistemas automatizados de produção.

A área de robótica educacional, especialmente na parte de manipuladores robóticos, está necessitando de desenvolvimentos que tornem possível sua utilização mais ampla. A tecnologia para construção de robôs está disponível, e a custos cada vez menores. É preciso, no entanto, realizar pesquisas que criem soluções funcionais.

O robô conectado ao computador é capaz de manipular pequenos objetos, tais como blocos de madeira ou pequenas peças, que podem ser transportadas de um ponto a outro ou montadas em alguma estrutura, como mostra a figura 5.1.

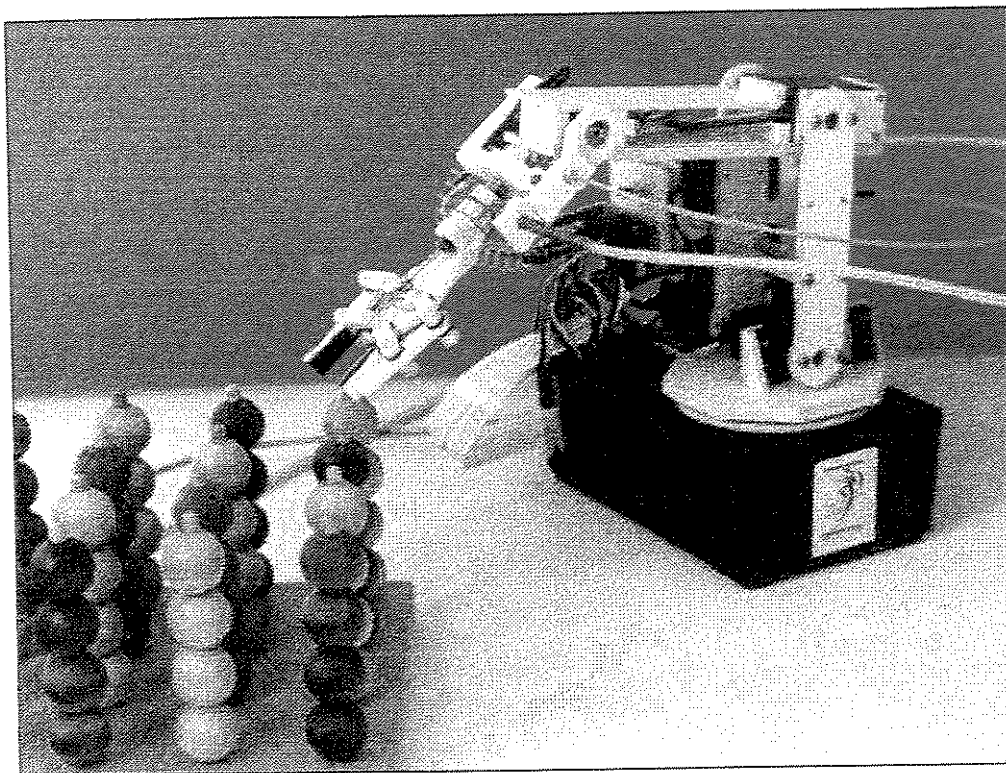


Figura 5.1: Uma das possíveis utilizações do robô é manipular pequenos objetos dispostos em um ambiente estruturado.

Utilizando um apontador LASER preso à garra é possível simular operações industriais como pintura, corte de chapas e furação, entre outras.

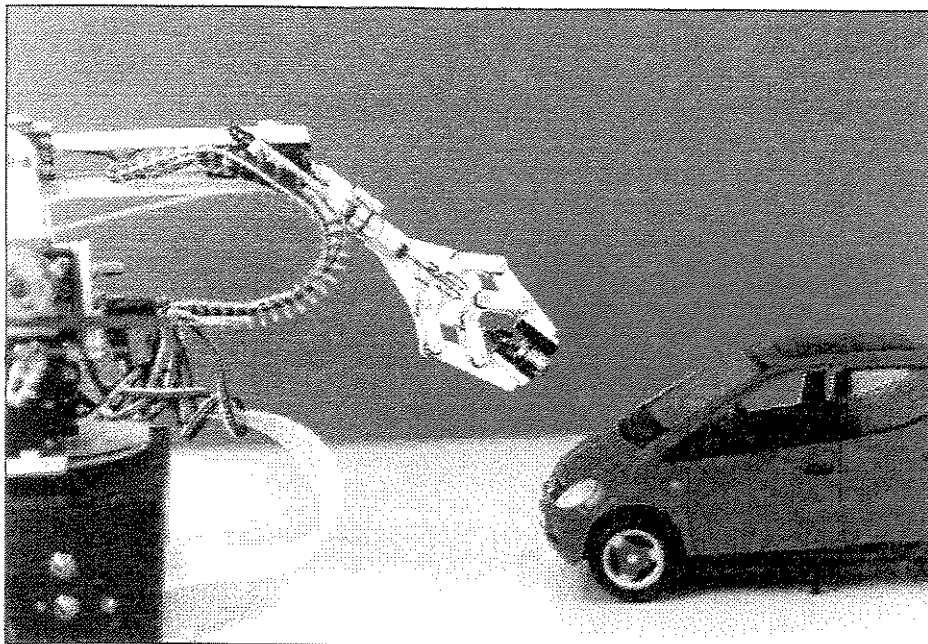


Figura 5.2: Robô utilizando um apontador laser para simular uma operação de pintura.

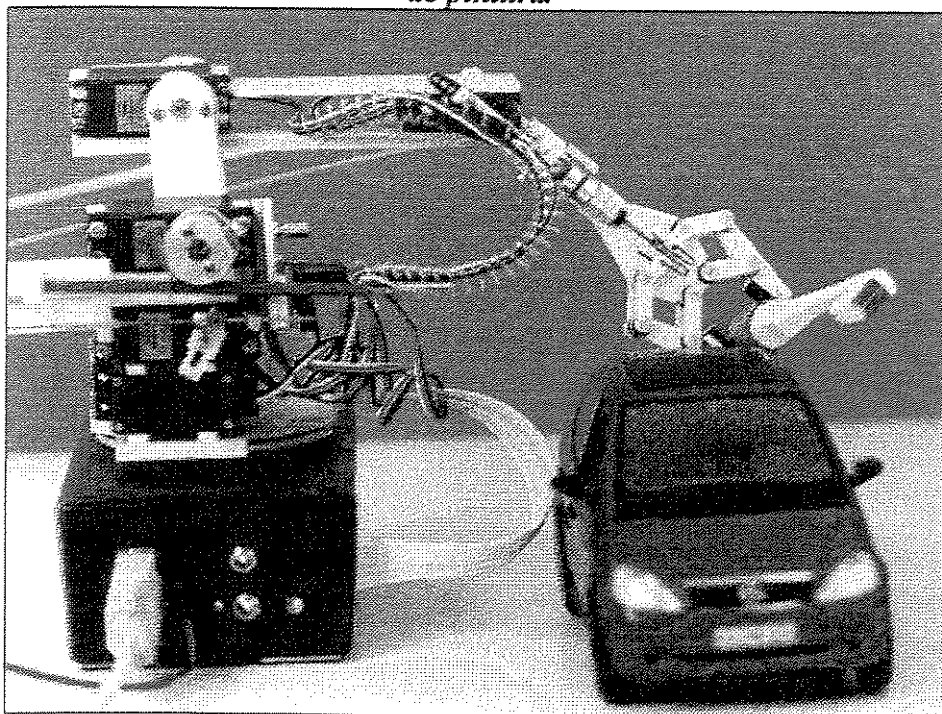


Figura 5.3: Robô simulando uma tarefa.

Este robô também é capaz de realizar tarefas leves tais como furar ou rebarbar peças, apertar parafusos, realizar pinturas etc..

5.1 Desenvolvimentos futuros

Algumas modificações podem ser implementadas no aplicativo, melhorando sua funcionalidade e tornando-o mais amplo e mais próximo dos aplicativos utilizados para controle de robôs industriais. Entre elas, podemos citar:

Inclusão de um modelo cinemático inverso utilizando o sistema de inserção de parâmetros do modelo cinemático direto já existente. Isto possibilitaria criar comandos para trajetórias retilíneas ou circulares e criar programas estruturados para operações de paletização.

Criação de um sistema de *debugging* permitindo execução passo a passo de programas etc.

Implementação de protocolos de comunicação mais sofisticados, permitindo a utilização de coprocessadores com melhores características.

Integração do aplicativo desenvolvido a um ambiente de desenvolvimento já existente, como o Logo, Matlab ou Octave (*freeware* espelho do Matlab). Estes possuem uma linguagem de programação mais elaborada que a linguagem interna do aplicativo e uma vasta gama de recursos que podem ser utilizados para a programação de robôs.

Utilizando conceitos de programação voltada a objeto, é possível transformar o aplicativo em um conjunto de módulos autônomos que seriam inseridos em um outro programa.

Na área de concepção e projeto de manipuladores, podem ser usadas diferentes tecnologias para acionamento das juntas, inclusive podendo-se combinar diferentes tecnologias em um mesmo robô. Podem ser criados projetos de robôs com diversos tipos de configuração, e também utilizando diferentes materiais na sua construção.

Atualmente o Nied está desenvolvendo um projeto semelhante, de controle de robôs utilizando o ambiente Logo. Será necessário o empenho por parte dos educadores em aplicar as soluções que estão sendo desenvolvidas nas universidades e centros de pesquisa.

Como pode ser visto no caso da Costa Rica, apresentado no item 2.3, a iniciativa em aplicar soluções didáticas é crucial para incentivar seu desenvolvimento.

Este trabalho possibilita três linhas para desenvolvimentos futuros: utilização do aplicativo diretamente para pesquisas e atividades de ensino em robótica; modificação do aplicativo, inserindo novos recursos; e também desenvolvimento de projetos e novas concepções de robôs a serem controlados pelo aplicativo.

Referências Bibliográficas

- Spong, Mark W. *Robots dynamics and control*. Nova Yorque: John Willey & Sons, 1989. 336p.
- Paul, Richard P. *Robot manipulators* 7.ed. Nova Yorque: MIT Press, 1981. 279p.
- Stone, Henry W. *Kinematic modeling, identification, and control of robotic manipulators*. Nova Yorque: Kluwer Academic Publishers, 1986. 224p.
- Nof, Shimon Y. *Handbook of industrial robots*. San Francisco: John Willey & Sons inc., 1985.
- Doughty, Samuel. *Mechanics of machines*. Nova Yorque: John Willey & Sons inc., 1988 467p.
- Rohreimer, Martin. *Telemanipulation eines roboters via internet mittels vrml 2.0 und java*, Institute of Robotics and Mechatronics DLR : Wessling Dissertação (mestrado), 1997
- Ruggiero, Márcia A. Gomes *Cálculo numérico aspectos teóricos e computacionais*. São Paulo: McGraw-Hill, 1988. 295p.
- Dibiella, Anthony J. *How organizations learn: an integrated strategy for building learning capability*. San Francisco: Jossey-Bass, 1998. 216p.
- Axelsson, Jan. *Serial port complete*. Madison: Lakeview Research, 1998. 306p.
- Aitken, Peter. G. *Visual basic 4: kit do explorador*. São Paulo: Berkeley Brasil Editora, 1996.

539p.

Pc Learneng Labs. *Desvendando o microsoft visual basic 4.0*. Rio de Janeiro: Livraria e Editora Infobook S.A., 1996. 335p.

Orvis, William J. *Visual basic for applications*. Rio de Janeiro: Axcel Books do Brasil Editora, 1994. 488p.

Valente, José Armando. *O professor no ambiente logo: formação e atuação*. Campinas, SP: UNICAMP/NIED, 1996. 435p.

Gardner, Howard. *Inteligências múltiplas: a teoria na prática*. Porto Alegre: Editora Artes Médicas Sul Ltda, 1995. 250p.

Gane, Chris. *Análise estruturada de sistemas*. Rio de Janeiro: Livros técnicos e científicos editora, 1983. 257p.

Maxim Integrated Products. *Rs 232 transceivers datasheet 19-0175 Rev 3 5/96* EUA. 24p.

Silva Júnior, Vidal Pereira da. *Microcontroladores pic:teoria e prática*. São Paulo:(s.c.p.), 1997. 140p.

Desenho Técnico I. Campinas: Unicamp, 1993 104p. (apostila)

Rembold, Ulrich (Ed.) *Robot Technology and Applications*. Nova Yorque: Marcel Dekker, inc. 1990. 682 p.

Groover, Mikell P. *Robótica tecnologia e programação*. São Paulo: McGraw-Hill, 1988. 399p.

David, Sergio Adriano et Rosário, J.M. *Prodedimento automático para aquisição e tratamento do movimento de um robô*, Campinas:Unicamp 1996 dissertação (mestrado)

Referências a informações disponíveis pela Internet

AppleWin URL: <http://geta.life.uiuc.edu/~badger/apple2.html> (disponível para download)
E.U.A.: 03/10/2000

Chipwits URL: ftp://ftp.apple.asimov.net/pub/apple_II/images/games/misc/ (disponível para download) EUA: 03/10/2000

Department of Electrical and Computer Engineering , University of West Florida
URL:<http://www.ee.uwf.edu/robotdraw.htm> E.U.A.: 03/10/2000

Eshed Robotec – Israel URL: <http://www.eshed.com/> Israel: 03/10/2000

Institute of Robotics and Systems Dynamics , do Centro Aeroespacial da Alemanha (DLR) URL:
<http://www.robotic.dlr.de/>

Logo Foundation URL: <http://el.www.media.mit.edu/groups/logo-foundation/> EUA: 03/10/2000

Lynxmotion URL: <http://www.lynxmotion.com/> E.U.A: 03/10/2000

Microchip URL: <http://www.microchip.com/> E.U.A: 03/10/2000

NIED URL: <http://www.nied.unicamp.br/> Campinas: 03/10/2000

Portland Area Robotics Society URL: <http://www.rdrop.com/users/marvin/> E.U.A.: 03/10/2000

Robix URL: <http://www.robix.com/> E.U.A.: 03/10/2000

Robot Store URL: <http://www.robotstore.com/> E.U.A.: 03/10/2000

School of Electrical, Computer and Telecommunications Engineering at the University of Wollongong, Australia URL: <http://robotoy.elec.uow.edu.au/robo3Dframe.html> Australia: 03/10/2000

Softronics Inc. URL: <http://www.softronix.com/> E.U.A.: 03/10/2000

Team Delta URL: <http://www.teamdelta.com/arm.htm> E.U.A.: 03/10/2000

Tower Hobbies URL: <http://www.towerhobbies.com> E.U.A.: 05/10/2000

Web 3D Consortium URL: <http://www.web3d.org/> E.U.A.: 03/10/2000

Anexo I

Estrutura interna do programa

Nesta seção estão abordados aspectos do funcionamento interno do aplicativo computacional. Eles ajudam a compreender como o aplicativo está estruturado e como as subrotinas interagem.

Existem inúmeras rotinas, porém somente algumas delas exercem as funções principais, enquanto as demais realizam tarefas específicas. A seguir, estão descritos os formulários que compõem o aplicativo e suas subrotinas mais importantes.

A estrutura do programa será apresentada usando um Diagrama de Fluxo de Dados (DFD), que é uma ferramenta de representação de sistemas para a criação de um modelo lógico (não físico), permitindo visualizar como as diversas partes se encaixam num funcionamento em conjunto.

Neste trabalho, foi a forma adotada para representação da estrutura dos programas criados, por ser a mais adequada, permitindo uma visão geral de seu funcionamento. Poderiam ter sido utilizados fluxogramas, por exemplo. Porém, estes se atêm a detalhes do funcionamento interno das subrotinas, que somente são de interesse para quem for trabalhar diretamente alterando o código-fonte.

Para este caso, a documentação da estrutura das rotinas do programa está feita como comentários no próprio código-fonte, trazendo detalhes a respeito do funcionamento de partes do programa e significado das variáveis.

Para maiores detalhes a respeito do assunto é recomendável se reportar a Gane, (1983).

Elementos básicos de um Diagrama de Fluxo de Dados lógico (DFD)

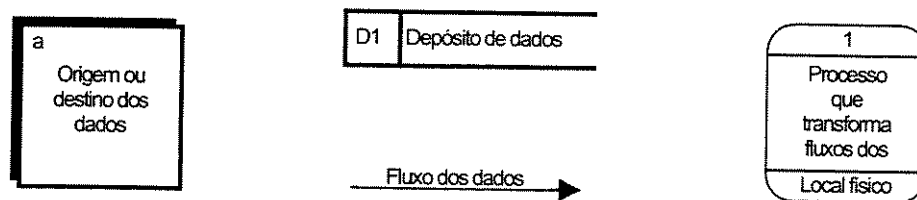


Figura A I.1: Os elementos básicos de um DFD.
O Diagrama de Fluxo de Dados Lógico é uma ferramenta de representação utilizada para Análise Estruturada de Sistemas.

Um DFD é composto por 4 elementos básicos:

Origem ou destino dos dados: Representa uma entidade externa, ou seja, que está fora dos limites do sistema considerado. No canto superior esquerdo, pode haver uma letra o identificando.

Depósito de dados: São entidades que armazenam dados para uso no processo, criando uma base de dados a serem acessados quando for conveniente. Conforme for conveniente, pode haver uma identificação do elemento, com uma letra "D" e um número.

Fluxo de dados: O fluxo de dados é simbolizado por meio de uma seta, com uma descrição significativa dos dados que estão sendo transmitidos.

Processo que transforma fluxo dos dados: Simboliza um processo que, a partir do fluxo de dados de entrada, geram fluxo de dados de saída. Sua descrição é feita por uma frase imperativa, com sujeito indefinido. Conforme for conveniente, a identificação do elemento aparece na parte superior, e o nome da entidade que fisicamente irá desempenhar a função (subrotina) aparece na parte inferior do elemento.

Diagrama de Fluxo de Dados Principal do Aplicativo

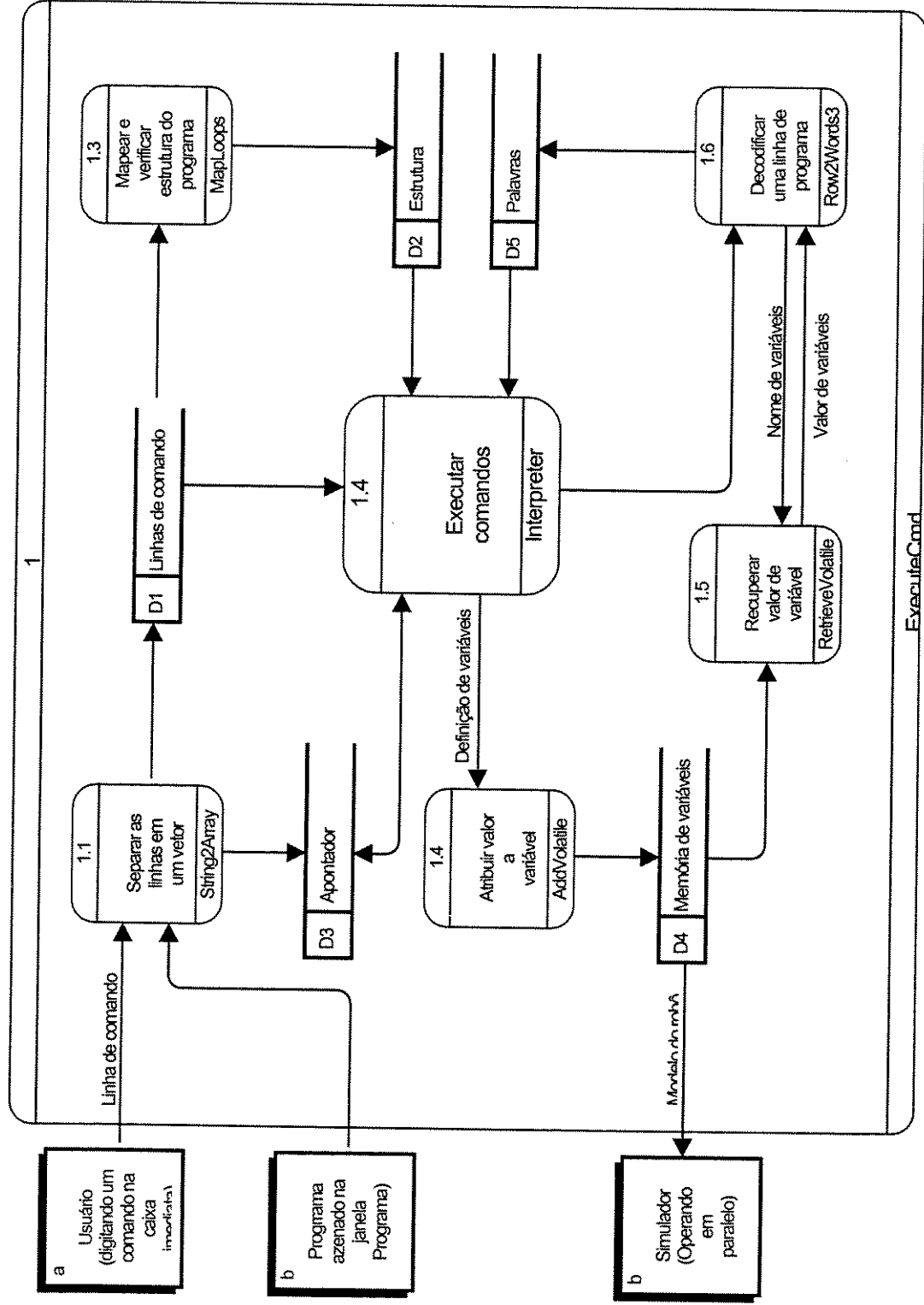


Figura A1.2: O fluxo de dados entre as principais rotinas do aplicativo

Depósitos de Dados:

D1: Vetor global dataArray

D2: Vetor global loopsMap, variável loopsMapPointer

D3: Vetor global Pointer

D4: Vetor global volatileMemory

D5: Vetor global WordsArray, variável WordsNumber

Todo o fluxo de dados é desencadeado quando o usuário digita uma entrada na caixa imediata ou faz um duplo-clique sobre uma linha do programa. Caso a entrada seja um nome de macro, é feito o mapeamento e verificação de todo o programa armazenado na memória e o apontador é direcionado para a primeira linha desta macro.

Caso a entrada seja somente um comando, o programa na memória não é decodificado, simplificando o fluxo de dados e apressando sua execução.

O depósito de dados que contém o valor de cada variável serve como base de dados para traduzir expressões matemáticas em valores numéricos que são transmitidos ao interpretador de comandos, e também para o simulador construir o modelo do robô.

A I.1 Formulário MDIForm1

O MDIForm1 é um formulário MDI (Multiple Document Interface) que atua como um contenedor dos demais formulários, que são tipo MDI Child. Ele atua como uma moldura, na qual os demais formulários são inseridos, e evita que eles fiquem espalhados pela área de trabalho.

Este formulário não realiza nenhuma outra função no aplicativo. Caso fosse criada uma barra de ferramentas com atalhos, ela seria implementada neste formulário

A I.2 Formulário frmMain

Este formulário cria a janela *Console*, e contém o núcleo do aplicativo. Os demais formulários exercem funções periféricas. Eles são de certa forma independentes, e seu funcionamento é ditado pelo frmMain. O fechamento do frmMain causa o fim da sessão do aplicativo como um todo.

Este formulário possui uma área de *prompt* de comandos, que será denominada área imediata (para execução imediata de comandos). Esta além de executar comandos digitados pelo usuário, exibe mensagens que ficam registradas na tela. Seu funcionamento simula um *prompt* de DOS em vários sentidos, inclusive na forma pela qual o texto se desloca para cima à medida que novas linhas são digitadas. Como o Visual Basic não possui um objeto com as características desejadas, esta área foi criada usando-se diversas caixas de texto, uma para cada linha na tela.

O número de linhas pode ser alterado conforme a necessidade, bastando ajustar a constante `text1Rows` adequadamente. Outras funções como repetição automática de funções, semelhante ao programa `DosKey`, e funções de recortar e colar também estão disponíveis.

Uma moldura no lado direito à cima exibe o status da comunicação com o robô, com mostradores tipo LED que se acendem quando está ocorrendo fluxo de informação.

Um botão de parada crítica interrompe imediatamente o funcionamento do robô, como ocorre nos modelos industriais.

Na parte inferior, uma barra de *status* constantemente exibe informações ao usuário. Na parte direita desta barra, existe uma figura mostrando se a conexão de dados com o robô está estabelecida ou cortada, e um cronômetro registra o tempo da sessão.

A I.2.1 Estrutura do interpretador de comandos

Apesar da linguagem de programação não ser compilada, é necessário realizar um pré-processamento para converter o texto do programa em dados que sejam rapidamente processados, tornando mais rápida sua execução.

Além desta etapa de pré-processamento, são feitas verificações e mapeamento da estrutura do programa, e caso seja detectada alguma falha, o aplicativo retorna uma mensagem de erro e cancela a execução do programa.

A I.2.2 A rotina String2Array

O pré-processamento do programa criado pelo usuário é feito pela rotina *String2Array*, que transforma a *string* contendo o texto do programa escrito em uma *array* de *strings*, sendo que cada elemento contém uma linha de programa.

As linhas em branco ou que contenham apenas comentários são eliminadas, e também os espaços antes do começo do texto (indentamento) assim como o texto que contém comentários, no final da linha.

A separação das linhas de programa é baseada apenas na ocorrência de caracteres CR e LF, não sendo aceito o caractere dois-pontos ":" como separador, pois complicaria muito esta rotina. Seria preciso mapear as aspas, para verificar se este caractere não é apenas parte de um texto a ser exibido pelo comando `Print`. Também complicaria as rotinas de inserção automática de linhas do formulário `Teach`, entre outras.

Em seguida é realizado o mapeamento da estrutura do programa executando a rotina `MapLoops`, que associa sentenças que podem causar um desvio na execução do programa às que contém o fim das sentenças.

Isto se faz necessário para que seja possível implementar com facilidade comandos como o **Exit For**, **Exit Loop** etc, além de *loops* dentro de outros *loops*.

Por exemplo:

```
if a = b then
  if a = c then
    a = 1
  else
    a = 2
  end if
else
  a = 3
end if
```

Neste caso, para $a \neq b$, o interpretador deve transferir o apontador do programa para a linha logo posterior à sentença **Else** correspondente. Porém existe outra sentença **IF... Else... End If** inserida dentro do **If** principal, sendo que não basta apenas procurar a próxima sentença **Else**; é preciso mapear qual sentença corresponde ao **If** em questão.

Este tipo de complicação ocorre sempre que um *loop* realiza um desvio "para baixo", ou seja, para uma linha de programa que ainda não foi executada. Existe um sistema de apontadores armazenados em uma pilha LIFO que cuida dos desvios "para cima".

O mapeamento da estrutura do programa é feito traçando as correspondências como pode ser visto na tabela a seguir:

Tabela A I.1: Correspondências entre desvios "para baixo".

Sentença		Sentença correspondente
for (<i>variável</i>)	=>	next (<i>variável</i>)
exit for	=>	next
do	=>	loop
exit do	=>	loop
if	=>	else (se não existir, para: end if)
else	=>	end if
exit macro	=>	end macro
select case	=>	case else / end select

A rotina MapLoops também verifica outras diversas regras de integridade na estrutura do programa, à procura de erros:

- Todas as *macros* devem ser definidas no nível zero, ou seja, não é permitido definir macros dentro de outras *macros*.
- Comandos For devem ter comando Next referenciando a mesma variável
- Comandos Do devem ter o correspondente Loop, além do que a sentença While ou Until deve estar presente uma vez.
- If deve ter um correspondente End If, sendo a sentença Else opcional
- Todos *loops* devem estar completos dentro de uma definição de *macro*.
- Select deve ter um correspondente End Select.

É importante notar que, quando uma falha de estrutura no programa é detectada, é emitida uma mensagem de erro às vezes ambígua, tipo: "Erro! (Definição de macro dentro e um loop ou macro) OU (próximo End Sub não possui correspondente sentença de definição de Sub)."

Isto se deve ao fato realmente ser possível desfazer esta ambigüidade ao verificar a estrutura do programa. Então a mensagem de erro indica os possíveis erros que ocorreram. O próprio Visual Basic não tem esta preocupação, de indicar várias possibilidades nas mensagens de erro, o que freqüentemente faz com que sejam exibidas mensagens de erro incompreensíveis

ao usuário e que não refletem o erro realmente ocorrido.

A I.2.3 A rotina Row2Words3

Para executar cada linha do programa, o interpretador usa a rotina Row2Words3 para transformá-la em uma *array* de *strings*, denominada **WordsArray**. É uma variável global, assim evita-se a necessidade de copiar seu conteúdo para transmitir entre subrotinas.

Este processamento de uma linha do programa inclui, além da separação em palavras, a decodificação recursiva das variáveis e avaliação das expressões matemáticas.

Esta rotina recebe, além da *string* contendo a linha de programa, um inteiro que indica quantos argumentos não devem ser processados pelo operador matemático. Dependendo da situação, não deve-se tentar aplicar o operador matemático em todos argumentos da linha de programa.

Por exemplo, quando um comando é executado, a primeira palavra é o nome do comando, e não deve passar pelo operador matemático. Caso seja uma linha atribuindo um valor a uma variável, as duas primeiras palavras (a variável e o sinal de " = ") não são decodificados. Por exemplo, se a linha de comando for: **For i = 1 to 10** , as 3 primeiras palavras não são decodificadas. Caso seja um comando **Show** (veja no Anexo IV significado deste comando), nenhuma das palavras é decodificada.

Tomemos como exemplo a seguinte linha de programa:

"move 1 to a"

Considerando-se que na memória já estejam armazenados os seguintes valores para variáveis:

Tabela A I.2: Exemplo de valores atribuídos a variáveis.

Nome da Variável		Valor
a	=	"b * 3 "
b	=	"c "
c	=	"2 "

A rotina Row2Words3 realiza a seguinte sequência para decodificar a linha de programa:

Tabela A I.3: Funcionamento da rotina Row2Words3.

Exemplo mostrando a sequência de valores de WordsArray durante a decodificação de uma linha de programa.

WordsArray(índice)							Texto remanescente
""	""	""	""	""	""	""	"move 1 to a
"move"	""	""	""	""	""	""	"1 to a"
"move"	"1"	""	""	""	""	""	"to a"
"move"	"1"	"to"	""	""	""	""	"a"
"move"	"1"	"to"	""	""	""	""	"b * 3"
"move"	"1"	"to"	""	""	""	""	"c * 3"
"move"	"1"	"to"	""	""	""	""	"2 * 3"
"move"	"1"	"to"	"1"	""	""	""	"* 3"
"move"	"1"	"to"	"1"	"*"	""	""	"3"
"move"	"1"	"to"	"1"	"*"	"3"	""	""
"move"	"1"	"to"	"6"	""	""	""	""

Durante a decodificação de uma linha de comandos, as palavras, separadas por espaços, são armazenadas no vetor WordsArray. Todas palavras que não sejam um número, um operador matemático (cadastrado no vetor HelpArray), uma função (cadastrada no vetor HelpArray) ou uma das palavras: " to ", " by " ou " eq " (veja a subrotina IsArgument), são substituídas pelo valor nelas contido.

Este valor será colocado como parte da linha de comando e novamente decodificado, sendo que o processo se repete várias vezes, decodificando vários níveis de variáveis contidas dentro de outras variáveis.

A recuperação do valor contido em uma variável é feita pela subrotina `RetrieveVolatile`, que trata das variáveis especiais e das variáveis declaradas pelo usuário. Caso a palavra não seja reconhecida, é gerado um erro de sintaxe.

Os trechos de uma linha de programa delimitados por apóstrofes ou por aspas não são decodificados, o que permite respectivamente o uso de expressões literais e o uso de *strings* de texto.

Na última etapa da decodificação, é executado o operador matemático, que transforma expressões matemáticas no correspondente valor numérico.

Como a *array* `WordsArray` é uma variável global, é preciso tomar alguns cuidados, pois alguma subrotina pode inadvertidamente alterar seu valor, causando um resultado inesperado. Por isso, entre a execução da rotina `Row2Words3` e a utilização dos valores contidos em `WordsArray` não podem ser executadas subrotinas que eventualmente possam modificar o vetor `WordsArray`.

Em consequência disto, rotinas acionadas por interrupção gerada pelo *Timer*, não podem levar a uma modificação em `WordsArray`. Além disto, após um comando `DoEvents` (liberação de processamento para multitarefa), o valor de `WordsArray` não é mais confiável e portanto não pode mais ser considerado válido.

AI.2.4 Operador de argumentos, operador de parênteses e operador matemático

Dentro da subrotina do operador matemático, a decodificação dos argumentos de uma linha de comando passa por 3 hierarquias de subrotinas de decodificação: Primeiramente, os argumentos separados por um caractere " ; " são decodificados em conjuntos separados.

Isto permite distinguir o sinal de menos como sendo um indicador de valor negativo ou um operador de subtração. Por exemplo, se a sintaxe fosse:

```
ExtDHtable1 = 3 -1 4 2
```

Durante a decodificação dos argumentos, o operador matemático interpretaria como se houvesse uma subtração, alterando inadvertidamente os argumentos da sentença. Por outro lado, o uso do operador matemático é indispensável para permitir que um valor seja obtido a partir de uma expressão. Utilizando sintaxe o comando fica:

```
ExtDHtable1 = 3;-1;4;2
```

Neste caso, o caractere " ; " separa os argumentos da sentença. Podem haver inúmeros argumentos sem separação pelo " ; ", como no caso do comando **Move**. Note que neste caso não há confusão com o sinal de menos, devido à presença dos separadores " **to** " e " **by** " que antecedem a posição do servo.

Numa hierarquia abaixo, vem o operador de parênteses, que procura dentro de uma expressão o maior nível de parênteses e envia seu conteúdo para o operador matemático decodificar.

Finalmente, a subrotina que interpreta a linha de comando irá verificar se os argumentos são apropriados em número e em tipo para a sentença.

Note-se que o processo de decodificação de argumentos é lento computacionalmente, uma vez que durante as etapas intermediárias de decodificação de uma expressão, os resultados parciais, mesmo que sejam numéricos, são convertidos em *strings* para homogeneizar o tipo de variáveis.

Posteriormente, estes valores são convertidos novamente em numéricos para realizar operações matemáticas e novamente seu resultado é convertido em uma *string*.

Para acelerar o processo de desenhar o modelo do robô, foram criadas subrotinas especiais, que tratam separadamente valores numéricos e *strings*. Por isto, a tabela de parâmetros **ExtDHtable** possui colunas que somente aceitam valores numéricos, enquanto outras também aceitam expressões matemáticas.

Na prática, testes mostraram que o operador matemático, apesar de não ser muito eficiente, é rápido o suficiente para não reduzir significativamente a performance do aplicativo.

AI.2.5 A geração de movimentos e simulação multitarefa

Durante a execução de um movimento do robô, seja ele proveniente de um programa ou de uma entrada na caixa imediata, a geração dos passos intermediários que compõem a trajetória são gerados pelo PC, e são enviadas posições dos servos aos controladores. Paralelamente a isto, deve ser feita a regeneração do modelo e também deve ser liberado processamento para o Windows, o que é necessário para o funcionamento do botão de parada crítica.

O sistema foi estruturado de modo a priorizar a geração de trajetória, o que evita degradação da precisão do robô ou variação no tempo de execução de movimentos devido a falta de capacidade de processamento por parte do computador.

Por outro lado, caso o computador seja rápido o suficiente a qualidade da simulação irá aproveitar ao máximo a capacidade de processamento. É preciso lidar com as duas possibilidades, já que os computadores têm uma evolução rápida na performance, mas ao mesmo tempo, computadores antigos continuam sendo utilizados.

A geração de trajetória usa a taxa de transmissão da porta serial para contagem de tempo, o que evita problemas de arredondamento do *timer* interno do PC, e mantém o *buffer* de saída da porta serial sempre acima de um nível de segurança. O aplicativo gera os passos intermediários do movimento enviando-os para a porta serial até que o *buffer* de saída atinja um determinado nível e somente após isso é liberado o processamento para outras tarefas.

Isto é feito pelo comando `DoEvents`, que libera o Windows para realizar algumas tarefas (isto é importante para possibilitar o funcionamento do botão de parada crítica, além de evitar que o sistema operacional fique "travado") e em seguida é executada a rotina que regenera o modelo do robô. Paralelamente a isto, os dados contidos no *buffer* de saída estão sendo enviados ininterruptamente pela porta serial.

O programa segue gerando a sequência de posições até completar novamente o *buffer* de saída. O tamanho deste *buffer* deve ser dimensionado para que ele não seja todo consumido enquanto o computador realiza outras tarefas, o que causaria uma pausa temporária no movimento do robô, causando inconsistência na execução de movimentos.

Ao terminar de gerar a sequência de passos, a rotina aguarda que o *buffer* de saída se esvazie, e então o aplicativo parte para a próxima linha de programa.

Assim, está estabelecida a prioridade: O sistema somente possibilita multitarefa quando o *buffer* de saída atinge um determinado nível, e também o modelo somente será regenerado quando houver capacidade de processamento suficiente.

Existe uma variável booleana, denominada `RedrawModel`, que atua como um semáforo. Quando setada indica que houve alguma modificação na posição do robô, e que o modelo deve ser regenerado. Se não estiver setada, indica que não há necessidade de redesenhar o modelo, evitando desperdício da capacidade de processamento.

A criação das posições intermediárias é feita através de uma varredura nas juntas, verificando se cada uma das juntas que teve de ser movida teve sua posição alterada. Caso nenhuma das juntas tenha sua posição alterada, é gerado um passo de espera, repetindo a posição atual de uma das juntas.

Da forma como foi criada, esta rotina otimiza o uso da porta serial, e dá oportunidade a todas os servos se moverem, sem priorizar algum em particular. A varredura somente ocorre nos

servos cuja posição de destino é diferente da posição de origem.

Todo o processo de geração de movimentos é sincronizado pela taxa de transferência da porta serial, devendo o PC enviar posições para o robô sempre à máxima taxa, possível. Não é necessário o uso do *timer* do PC, e a qualidade do movimento se torna extremamente consistente.

Por exemplo, se for utilizado o controlador SSC com a porta serial configurada a 9600 BPS, como cada *byte* é composto por 8 bits, mais 2 bits de sincronia (*start bit* e *stop bit*), tem-se o envio de 960 *bytes* por segundo. Cada pacote de dados é composto por 3 *bytes*, logo o sistema irá opera a uma taxa de 320 pacotes por segundo (uma a cada 3,125 ms.). Cada pacote de dados se refere a posicionamento de um servo individualmente.

O controlador SSC também pode operar a 2400 BPS, porém a esta taxa de transferência ocorre uma vibração indesejada no braço robótico. A frequência de envio das posições excita vibração no robô. Para que o movimento seja suave, é preciso que esta frequência seja superior às frequências de vibração naturais do manipulador.

Quando o robô está operando *offline*, a geração de trajetória é feita por uma subrotina específica para esta situação, denominada *SeqGenOffline*. Pelo fato da porta serial estar desligada, é necessário o uso de um *timer* para sincronizar o movimento.

São geradas interrupções na mesma frequência em que seriam enviadas as posições ao controlador. Na rotina de serviço, é incrementado um contador, que representa a base de tempo para geração da trajetória.

A qualidade da animação do modelo vai depender da velocidade do computador, mantendo-se fixo o tempo de execução dos movimentos. Caso o computador seja lento, pode-se mudar as opções do simulador para acelerar a simulação e melhorar a qualidade da animação.

Paralelamente a tudo isto (tanto na operação *on line* quanto na *off line*), existe um timer

para forçar a regeneração do modelo e executar o comando `DoEvents` a intervalos maiores (1000 ms), somente para o caso do computador não ter velocidade em excesso, e para regenerar a imagem do modelo periodicamente caso a opção `Auto Redraw` estiver desligada (isto faz como que a imagem seja regenerada caso uma janela seja arrastada à sua frente).

Através de testes com o programa, foi constatado que um Pentium 166 é tranqüilamente capaz de gerar a seqüência de posições a uma velocidade superior que é feito o envio dos dados. Assim, foi possível implementar uma rotina que calcula as posições intermediárias "*on the flight*", ou seja, fazer os cálculos enquanto o robô está em movimento.

Caso não fosse este o caso, seria preciso primeiro criar a seqüência e armazenar na memória para depois enviar os dados. Porém, isto causaria um retardo entre um movimento e outro. Isto foi inicialmente implementado para verificar a relação entre o tempo necessário para gerar a seqüência e o tempo do movimento propriamente.

O caso crítico de demanda de processamento para geração de trajetória ocorre quando o robô está se movendo a baixa velocidade, e o programa freqüentemente faz uma varredura em todos os servos que foram comandados, verificando que não houve mudança de posição, e envia um pacote repetindo a posição do servo, somente para criar uma espera.

Quando ficou claro que o PC tem capacidade suficiente de processamento, foi modificada a rotina para operar "*on the flight*". A primeira versão da rotina `SeqGen` acumulava toda a seqüência de dados que comandam o movimento do robô e posteriormente a enviava através da porta serial, usando a rotina `OutputData`.

AI.3 Formulário frmProgram

O formulário `frmProgram` foi criado a partir de uma caixa de texto, que exibe o código do programa na memória. Esta implementação foi, na verdade, uma grande fonte de problemas. O ideal seria poder acessar uma linha de programa diretamente pelo seu endereço, porém o Visual

Basic não possui nenhum objeto que cumpra bem este papel. Uma opção seria usar o objeto `ListBox`, porém ele não permite edição de seu conteúdo.

Isto levou à necessidade de armazenar todo o programa em um vetor durante a execução, para que fosse possível endereçar as linhas sem ter que percorrer todo o texto do programa. Também foi um grande complicador para as rotinas de inserção automática de linhas, indentamento etc.

Devido a estas dificuldades, foi eliminado o elemento ":", que divide uma linha de texto em duas linhas de comando. Embora fosse possível implementar esta função, o código se tornaria muito mais complicado toda vez que fosse necessário inserir uma linha de programa, por exemplo.

Quando a execução de um programa é interrompida, seja por erro de execução ou pelo botão de parada crítica, o aplicativo destaca a linha que estava sendo executada. Isto é possível graças a um mapeamento de quais linhas do `program.text` correspondem a quais elementos do vetor `dataArray` (lembrando que neste processo de conversão, linhas do programa inválidas são descartadas, então esta correspondência não é trivial, e deve ser mapeada).

AI.4 Formulário frmTeach

O `frmTeach` cria a janela Teach Box, que permite operar diretamente o robô. O principal elemento são as *Drag Boxes*, implementadas usando *Picture Boxes*. Toda vez que o se arrasta o cursor sobre uma destas, uma junta do robô é movida. Isto é feito armazenando a posição na qual o botão do mouse foi pressionado, comparando com a posição atual. Também é preciso compensar eventual movimentação da junta usando o teclado.

Foi necessário modificar diversas vezes esta função de arrastar até chegar ao resultado desejado, pois o Visual Basic se mostrou inconsistente na geração de eventos. Um código semelhante foi usado para as rotinas de arrastar a imagem no simulador e arrastar a moldura com

as coordenadas no simulador.

A movimentação pela Teach Box pode ser feita pelo teclado, e existem diversas teclas associadas a funções de mover o servo ou selecionar outro servo (veja no capítulo 4).

Os limites ao movimento de cada junta sempre são respeitados, e caso o usuário tente movimentar a junta fora destes limites, a *Drag Box* se torna vermelha, indicando o uso indevido.

Caso se esteja operando com mais de 8 servos (um manipulador robótico dificilmente usaria mais que isto, porém é possível controlar também outros dispositivos que contenham um grande número de graus de liberdade), é feita uma paginação das *Drag Boxes*, que passam a atuar sobre o servo $(i*8 \text{ a } 7+i*8 \mid i \in \mathbb{N} \text{ de } 0 \text{ a } 15)$.

Quando esta paginação é feita, a legenda que indica o nome do servo e o indicador de posição do servo também são paginados, mantendo-os sempre coerentes com a função do *Drag Box*.

AI.5 Formulário frmDHSim

O formulário frmDHSim dá origem à janela do simulador no aplicativo. Este formulário possui três elementos Picture Box, nos quais são desenhadas as linhas que formam o modelo do robô.

Para dar um efeito mais realístico ao desenho, o *draw mode* foi ajustado para *mask pen*, o que permite exibir o robô como se fosse transparente. Isto evita que as linhas se sobreponham, tornando a imagem confusa. As *Picture Box* são dimensionadas de modo que a número 1 tenha o dobro do lado que as duas demais, caso os três *Viewpoints* estejam sendo exibidos. Caso contrário, a `PictureBox(1)` é expandida, ocupando toda área útil da janela e as outras se tornam invisíveis.

Para auxiliar o desenvolvimento do programa, foram criadas quatro Text Box que exibem o conteúdo de uma matriz (o que o *debug* do Visual Basic não é capaz de fazer). Elas somente se tornam visíveis quando a subrotina Mostra é executada. No programa compilado, estes elementos nunca são visíveis.

A I.5.1 Criando imagens 3-D para ser vistas com óculos coloridos

Um dos recursos implementados no simulador foi a possibilidade de criar imagens tridimensionais que podem ser visualizadas com óculos coloridos.

O princípio de funcionamento é bastante simples: sobre cada olho é colocado um filtro colorido, que esconde parte da imagem. Assim, cada olho recebe uma imagem distinta, possibilitando a criação de um efeito tridimensional.

Sobre o olho esquerdo é colocado um filtro vermelho puro, representado no padrão RGB como (255,0,0) e sobre o direito um filtro ciano puro, representado no padrão RGB como (0,255,255).

Foi criada um desvio na rotina de plotagem que faz com que, caso a opção de plotar imagens em 3-D esteja habilitada, na moldura principal do simulador cada linha (assim como sua respectiva sombra) passa a ser desenhada duas vezes: Uma à direita, em ciano, formando a imagem que será vista pelo olho esquerdo e outra à esquerda, em vermelho, formando a imagem para o olho direito.

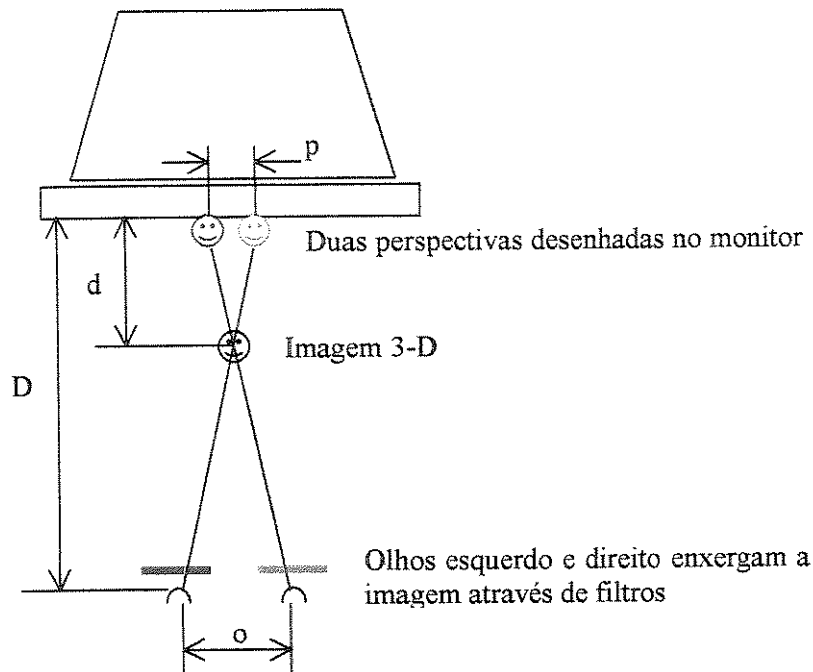


Figura A I.3: A visão em profundidade.

O paralaxe (separação entre as duas perspectivas da imagem) determina sua projeção à frente do monitor. Os filtros à frente dos olhos escondem parte da imagem exibida na tela, permitindo criar o efeito estéreo.

Como pode ser observado no esquema, o paralaxe resultante da projeção da imagem de um ponto a uma distância d a frente do monitor, vale a distância interocular vezes d dividido pela distância do olho ao monitor, D .

$$p = o \cdot \frac{d}{D}$$

Considerando para um caso típico, $o = 10$; $D = 60$ centímetros, a fórmula resulta:

$$p = d/6$$

Dado que o coeficiente de redução de escala dos eixos da perspectiva isométrica vale 0,816497 Desenho Técnico,(1993), p.12; vem que a dimensão de profundidade vale:

$$-d = x.0,577350 + y.0,577350$$

$$\text{pois } \sqrt{0,816497^2 + 0,577350^2} = 1$$

logo, o paralaxe vale:

$$p = -x.0,096225 - y.0,096225$$

Inserindo na fórmula que faz a projeção de x e y para coordenadas gráficas x' ;

$$x' = x \cdot \frac{\sqrt{3}}{2} \cdot 0,816497 - y \cdot \frac{\sqrt{3}}{2} \cdot 0,816497$$

$$\Rightarrow x' + p = x \cdot \frac{\sqrt{3}}{2} \cdot 0,816497 - y \cdot \frac{\sqrt{3}}{2} \cdot 0,816497 - x.0,096225 - y.0,096225$$

logo,

$$x' + p = x.0,610882 - y.0,803332 \text{ (desenhado em ciano)}$$

e

$$x' = x.0,707107 - y.0,707107 \text{ (desenhado em vermelho)}$$

Para projetar a imagem à frente do monitor, é escolhido pelo usuário um valor de paralaxe que será somado ao paralaxe devido à profundidade da imagem.

Pode-se ver como estas funções foram aplicadas para desenhar o modelo na tela do computador observando-se as subrotinas `DrawVector1` e `DrawVector2` na listagem do simulador.

AI.6 Formulário *Values*

Este formulário atua como uma barra de ferramentas, que permite seleccionar o *zoom* do simulador, e permite ao usuário escolher a exibição da posição e orientação de um sistema de coordenadas.

AI.7 Módulo *Module1*

O `Module1` contém as definições das variáveis globais para o projeto, e definições de constantes. A estrutura do programa na grande maioria das vezes transmite dados entre as funções através destas variáveis, que possuem um significado estritamente definido.

Isto facilita a compreensão do código do programa por terceiros, e facilita localizar as subrotinas que realizam determinada função, bastando realizar uma busca localizando quais procedimentos alteram determinada variável global.

Num programa no qual a troca de variáveis é feita sob forma de argumentos passados às rotinas, é preciso verificar em cada rotina como este processo ocorre, o que dificulta muito a compreensão do programa.

As variáveis estão classificadas por grupos, e junto à declaração de cada uma existe uma documentação de sua função e do significado de seus argumentos.

Anexo II

Referências na Área de Robótica Educacional

A II.1 Instituições e ferramentas relacionados à programação de robôs

Estão descritos os principais resultados de pesquisa relacionados à programação de robôs, incluindo desde robôs autônomos com fins educacionais aos mais modernos métodos de programação de robôs industriais.

A II.1 a) Logo Foundation

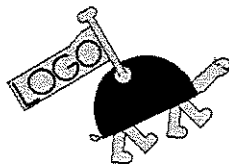


Figura A II.1: Logotipo da LOGO Foundation

A Logo Foundation é uma organização sem fins lucrativos com base em Nova York, E.U.A. e é responsável pelo desenvolvimento e treinamento de pessoal de apoio e suporte de serviços para a divulgação do Logo como uma linguagem educacional.

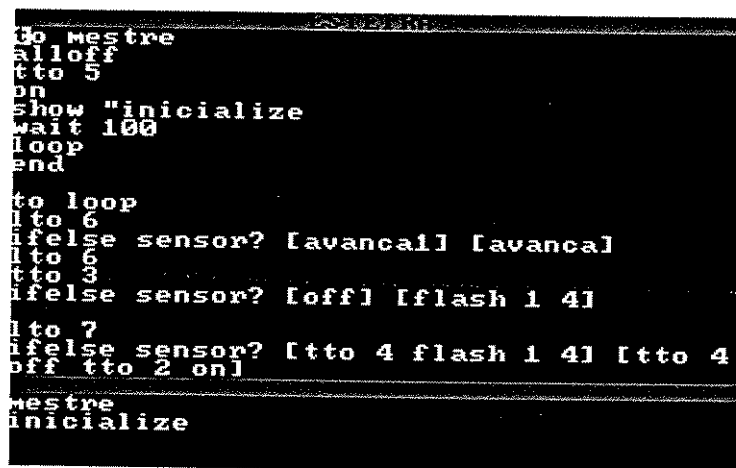
O Logo é uma linguagem computacional criada em 1967 no Laboratório de Inteligência Artificial do MIT por Seymour Papert, um matemático que trabalhou com Jean Piaget e que posteriormente se mudou para os Estados Unidos.

O Logo se disseminou nos anos 70, com a criação de uma versão para o Apple II, o precursor dos computadores caseiros.

Nos anos 80, a equipe de pesquisadores liderados por Seymour Papert desenvolveu o sistema LEGO-Logo, que alia à facilidade de construção do LEGO a possibilidade de controlar motores e luzes, e sensores a partir do Logo. Este sistema, denominado Tc Logo é usado atualmente no LAR para aulas de laboratório de automação industrial.

É basicamente um dialeto do Lisp, desenvolvida como uma ferramenta de aprendizado, baseado no construtivismo, filosofia educacional desenvolvida por Piaget. O construtivismo considera que o conhecimento é construído através da interação entre as pessoas e o mundo existente ao redor.

A aplicação mais popular do Logo envolve uma tartaruga, que é controlada para se mover na tela formando desenhos cada vez mais complexos, usando programação estruturada, subrotinas, etc..



```
to mestre
  alloff
  tto 5
  on
  show "inicialize
  wait 100
  loop
end

to loop
  lto 6
  ifelse sensor? [avanca1] [avanca]
  lto 6
  tto 3
  ifelse sensor? [loff] [flash 1 4]
  lto 7
  ifelse sensor? [tto 4 flash 1 4] [tto 4
  off tto 2 on]
mestre
inicialize
```

Figura A II.2: Tela de programação do TC Logo

Recentemente, a LEGO lançou um tijolo programável denominado *Mindstorms*, num projeto de pesquisa do MIT supervisionado por Fred Martin, 1997. Este sistema permite criar robôs autônomos (microprocessados por um Motorola MC68HC11) programados em um PC

usando o LEGO Logo.

Uma característica bastante interessante do LEGO Mindstorms é que, devido ao seu custo reduzido, a partir de US\$198,00 nos EUA (preço de catálogo da loja "Robotstore"), o aluno poderá adquirir o material para brincar em casa. Isto o transforma em um produto massificado. Os sistemas educacionais anteriores da LEGO eram bem mais caros e somente acessíveis às instituições de ensino.

A II.1 b) Aplicativo com linguagem de programação gráfica ChipWits

Este aplicativo, desenvolvido em 1985 pela empresa Epyx para as plataformas Apple II, Commodore 64 e Mac 128k é um exemplo dos primeiros desenvolvimentos em linguagens de programação voltadas a aplicações de ensino, numa época em que o conceito de computador caseiro era uma novidade, e havia grandes desenvolvimentos para uso dos computadores como ferramenta de ensino.

Atualmente, estas plataformas são obsoletas e fora de operação, porém foi possível recuperar o aplicativo em um FTP que distribui gratuitamente imagens de discos de Apple II e através de um emulador, foi possível executar este aplicativo dentro do ambiente Windows.



Figura A II.3: Aplicativo Chipwits sendo executado em um emulador para PC

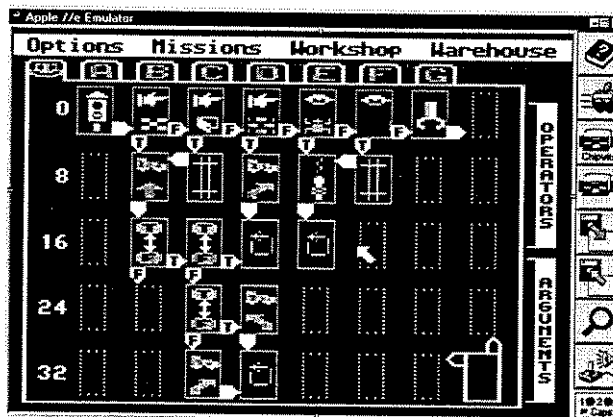


Figura A II.4: Programação por ícones no Chipwits

O aplicativo fornece uma linguagem de programação gráfica, na qual o usuário cria programas a partir de ícones, que representam símbolos de fluxograma. Uma vez criado o programa, o usuário vê uma simulação de seu robô se movendo em um cenário, desviando de predadores e buscando os bônus (as tortas e as xícaras de café). Ao final da simulação, é apresentado um escore para avaliar o desempenho do robô.

Embora o programa seja da época dos primórdios da computação caseira, traz conceitos interessantes. Sua utilização é toda feita através de um apontador (na época, costumava-se usar um joystick, já que o mouse necessitava uma placa especial, e tinha um preço muito elevado) e os ícones são de fácil compreensão, dispensando o aprendizado de comandos em texto. O programa é bastante visual e divertido.

As ações do robô são baseadas nas ações usuais de uma pessoa: andar, olhar, apalpar, cheirar, pegar. A linguagem de programação também envolve os conceitos de subrotina (as fichas A,B,C,D,E,F,G), pilha e programação estruturada.

Este programa ilustra a criação de um ambiente de desenvolvimento amigável e divertido, que utiliza a simulação do funcionamento de um robô para desenvolver conceitos de programação estruturada, uso de pilhas etc.

A II.1 c) PROBO-3 Programmable Robots For Battle Operation

Para a linha PC, surgiram diversas linguagens de programação que permitem simular robôs em um ambiente criado dentro do computador. Um exemplo é a linguagem PROBO-3, que foi criada a partir do Pascal, adicionando bibliotecas especiais que criam novos comandos de controle de robôs virtuais.

Neste caso, os robôs criados irão se enfrentar em uma arena e o objetivo é destruir os robôs adversários até restar apenas um sobrevivente. Com o desenvolvimento de robôs de baixo custo (por exemplo, utilizando o LEGO Mindstorms), está sendo possível criar competições semelhantes, agora usando robôs de verdade. Exemplos são o futebol de robô, promovido no Brasil pelo CTI.

A II.1 d) Softronics Inc.

A Softronics é uma empresa que desenvolve *softwares* educacionais. Produziu uma versão de Logo para PC, denominado Superlogo, escrito em Borland C++ 5.

Esta versão do Logo é *freeware* e o código fonte, assim como o executável são distribuídos gratuitamente no *site* da empresa.

A empresa também desenvolveu um outro software educacional, o MMLogic – MultiMedia Logic Design System, um simulador de circuitos lógicos no estilo do Simulink do Matlab, vendido a um preço bastante acessível (US\$ 34,95 a licença pessoal, cotado em 10/99).

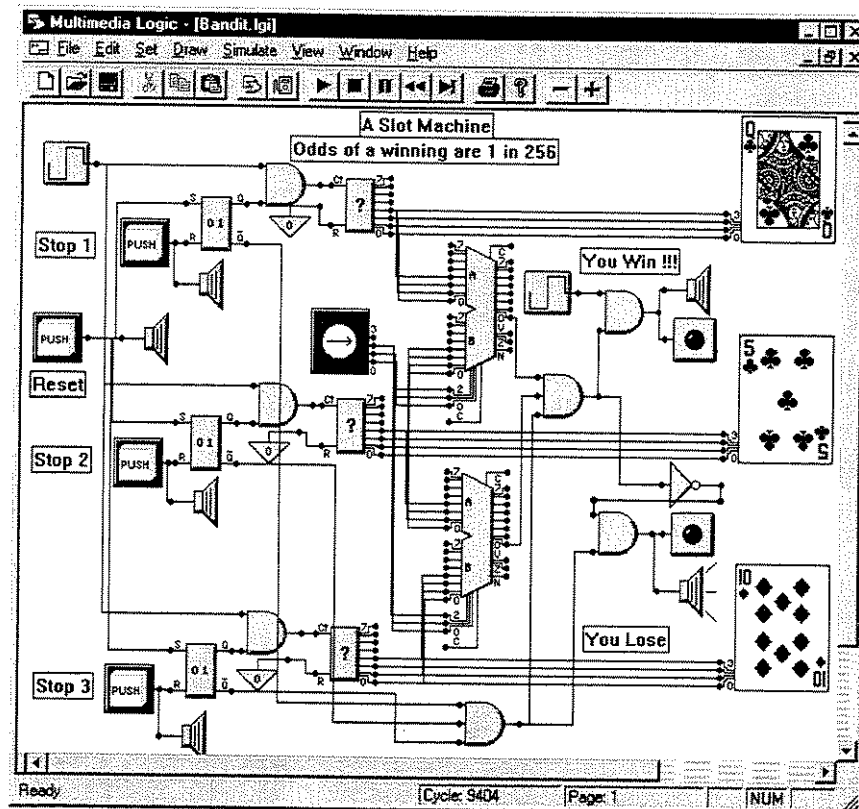


Figura A II.5: Tela do aplicativo MM Logic

Este programa possui um módulo que simula o funcionamento de um robô autônomo, que se move na tela do computador. O ambiente é semelhante ao Simulink do Matlab, e permite a criação de circuitos lógicos complexos.

A II.1 e) Núcleo de Informática Aplicada à Educação - Unicamp

O NIEd desenvolve pesquisa na área de informática aplicada à educação, e tem sido um dos parceiros do LAR no desenvolvimento de atividades de laboratório didático para ensino de fundamentos de automação Industrial, sob supervisão do pesquisador João Vilhete.

O NIEd criou e dá suporte a uma versão em português do WinLogo da Softronicx e desenvolve projetos baseados no Logo.

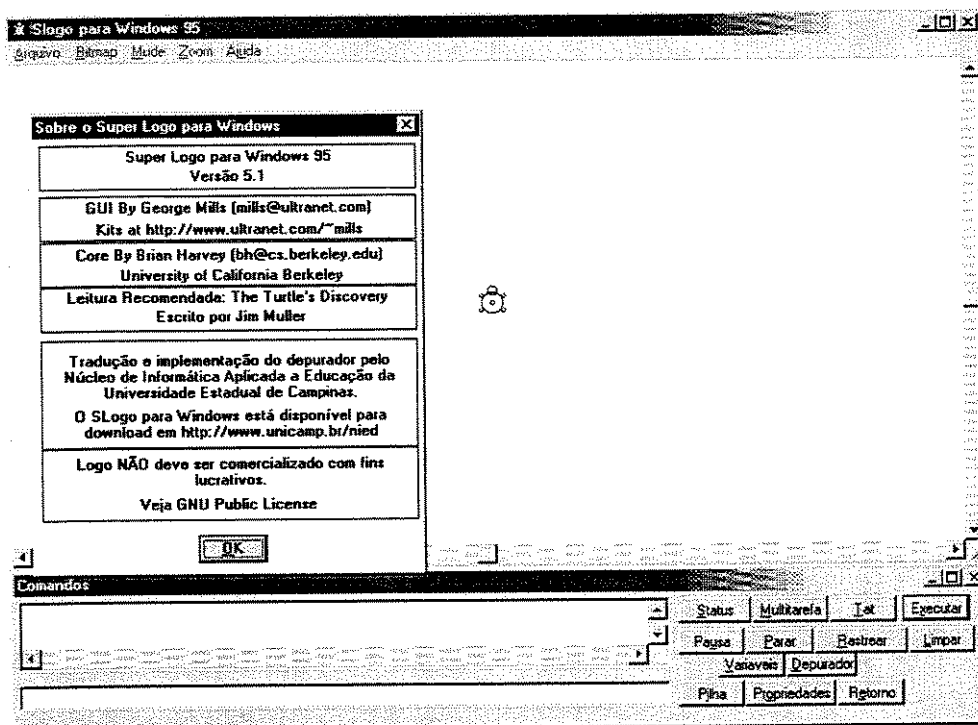


Figura A II.6: Tela do Winlogo, na versão em português.

A II.1 f) O aplicativo Workspace 4.0 (como gerador de programas)

Este aplicativo é inteiramente voltado à geração automática de programas baseado em modelos e representa o que há de mais moderno em termos de programação de robôs.

O aplicativo fornece um ambiente gráfico tipo CAD, no qual são inseridos os modelos do manipulador, garra, objetos sendo trabalhados, dispositivos auxiliares, enfim, toda a célula de produção pode ser representada no modelo.

A partir daí, a geração do programa é feita a partir da definição de pontos, trajetórias e restrições (*constrains*). Assim, pode-se facilmente programar uma trajetória seguindo a superfície irregular de uma peça que deve ser soldada a arco, com velocidade e ângulo controlados.

Outros recursos, como a detecção de colisões, permitem localizar erros no programa sem riscos de danificar o equipamento.

A II.2 Pesquisas e instituições ligados à simulação de robôs

Várias soluções possíveis para criar a visualização de um modelo de robô podem ser identificadas. Existem diversas linguagens e pacotes disponíveis que podem ser usados, além de diversas formas de representação de um manipulador robótico.

A II.2 a) School of Electrical, Computer and Telecommunications Engineering da University of Wollongong, Austrália

Esta faculdade criou um robô que pode ser controlado pelo *browser* da Internet. Para isto foi utilizado um braço robótico de pequeno porte, criado para atividades de ensino.

No *site* está disponível um simulador do robô, escrito em Java. A partir do ângulo das juntas, o simulador desenha o modelo em 3D do manipulador robótico.

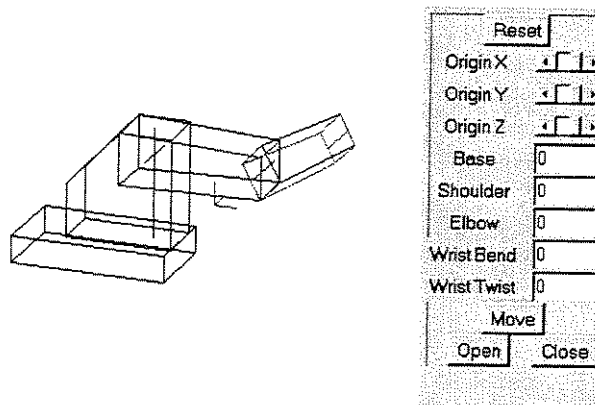


Figura A II.7: Simulador do Robotoy, em Java

Existem diversos outros *sites* na Internet que permitem controlar robôs à distância.

A II.2 g) Team Delta

Desenvolve robôs, componentes eletrônicos especiais para robótica, e criou um simulador de robô em Java, que pode ser executado em um *browser* de Internet. O simulador permite mover

as juntas do manipulador robótico e alterar ângulo da perspectiva.

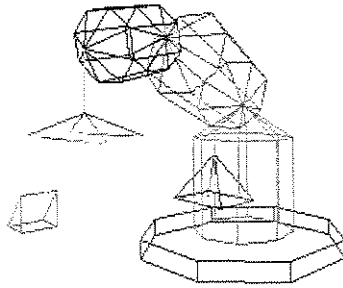


Figura A II.8: Simulador de robô, em Java

A II.4 h) Institute of Robotics and Systems Dynamics , do Centro Aeroespacial da Alemanha (DLR)

No URL: <http://www.robotic.dlr.de/Joerg.Vogel/Vrml/lib.html>, há uma página que monitora progressos em VRML e Java 3D relacionadas a robótica ao redor do mundo, e contém dezenas de *links*.

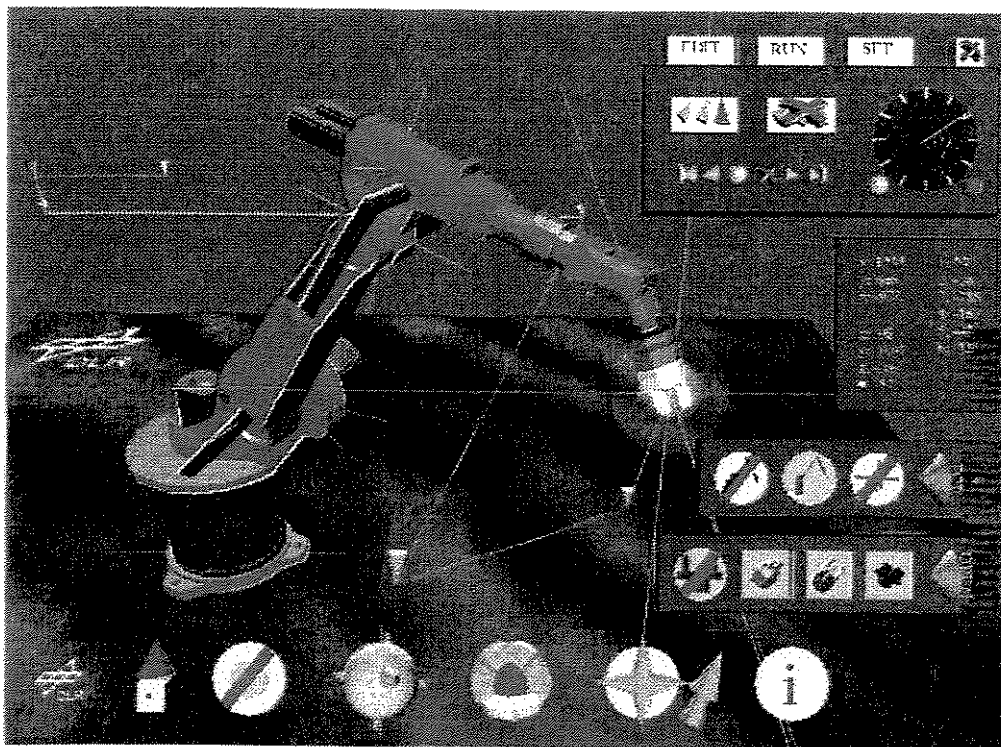


Figura A II.9: Simulação de um robô KuKa em VRML

A figura mostra a simulação em VRML de um robô Kuka, criada pelo Laboratório de Robótica do DLR. Esta simulação, além de muito completa do ponto de vista visual, possui um modelo cinemático inverso em Java Script que permite modificar a orientação e posição xyz da extremidade do braço robótico. O modelo foi resultado da tese de mestrado "*Telemanipulation eines Roboters via Internet Mittels VRML 2.0 und Java*", de Martin Rohrmeier, (Oct 1997)

O inconveniente deste simulador é que o tempo de regeneração das imagens é demasiadamente grande (foi experimentado em um microcomputador com processador Pentium 166, obtendo-se resultados próximos a um quadro por segundo), o que limita a criação de modelos complexos como este.

A II.2 i) Web 3D Consortium

VRML provém da abreviação em língua inglesa de "Linguagem de Modelamento de

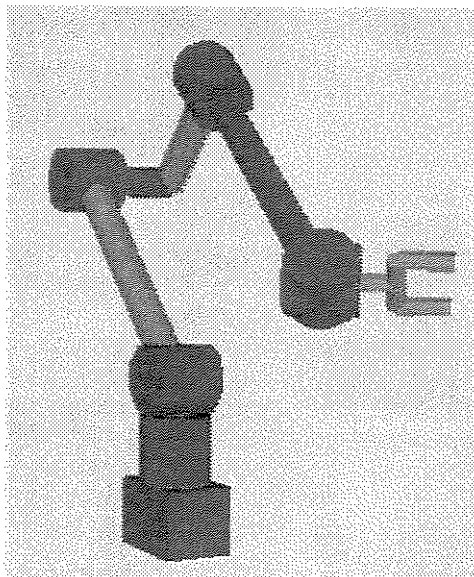
Realidade Virtual". É uma linguagem especial, dedicada à criação de modelos em realidade virtual, reconhecida internacionalmente pela ISO, na norma (ISO/IEC-14772-1:1997). Antes mesmo de ser normalizada, a VRML já era um padrão para repartir e publicar dados entre CAD, animações e modelagem 3D. Praticamente todos estes programas possuem um utilitário para exportação de dados neste padrão.

A VRML é propriamente uma linguagem de programação estruturada, formada por linhas de comandos que trazem dados descrevendo a geometria e comportamento de um cenário em 3D. Um conjunto de cenas forma um "mundo", que pode descrever uma cena bastante complexa.

Atualmente, há diversos programas *freeware* que exibem imagens em realidade virtual a partir de arquivos VRML (extensão WRL).

A II.2 j) Department of Electrical and Computer Engineering , University of West Florida

O departamento de Engenharia Elétrica e da Computação da Universidade da Flórida Oeste criou um aplicativo que gera modelos em VRML de manipuladores robóticos a partir dos parâmetros de Denavit-Hartenberg.



*Figura A II.10: Um modelo de robô em VRML.
Este modelo foi criado automaticamente, a partir dos parâmetros de Denavit-Hartenberg*

O sistema criado é bastante interessante, permitindo a modelagem de qualquer robô, de configuração genérica. O VRML permite criar facilmente modelos em 3-D com boa qualidade.

A II.2 k) Simuladores desenvolvidos pelo LAR

O LAR esteve empenhado em desenvolver simuladores que permitissem a programação de manipuladores robóticos em modo *off-line* para manutenção de "árvores de natal" de exploração de poços de petróleo a grandes profundidades. As simulações foram criadas utilizando-se a linguagem ADA.

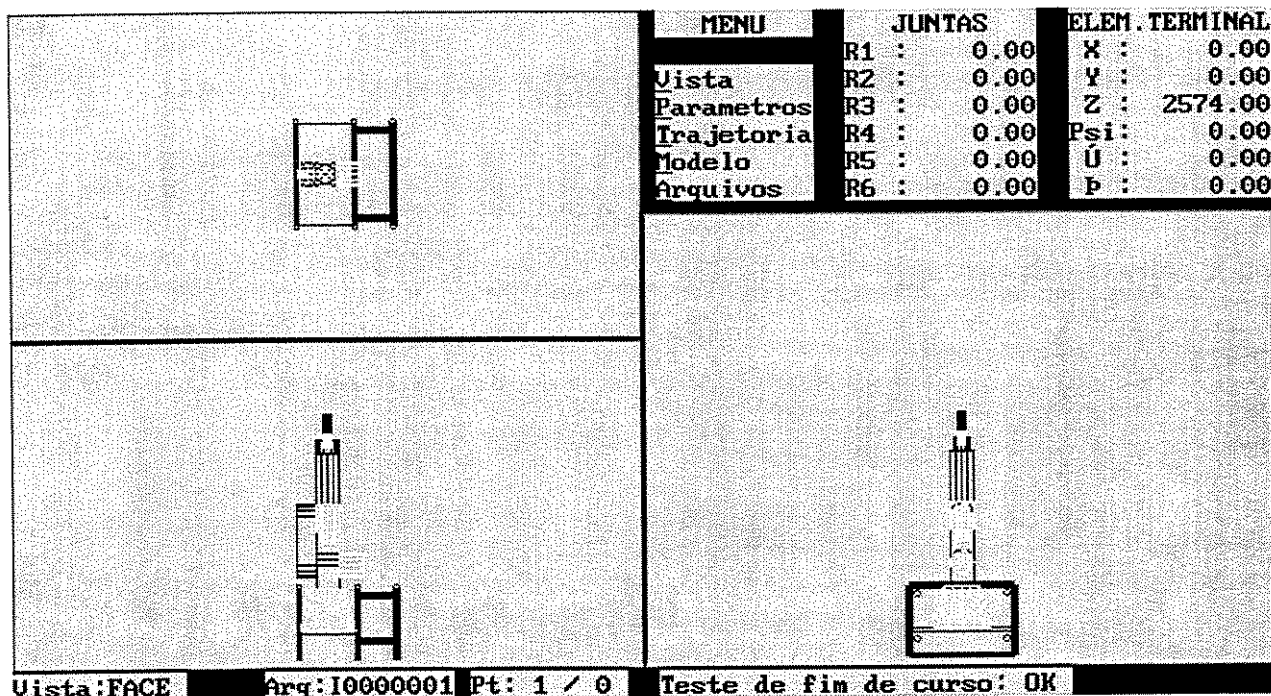


Figura A II.11: Simulador de robô para manutenção em águas profundas

Este simulador possui um modelo cinemático inverso, e foi criado para fazer a programação off-line baseada em um modelo.

Outro exemplo de simulador exibe a configuração das juntas de modo mais explícito e permite movimentar o robô.

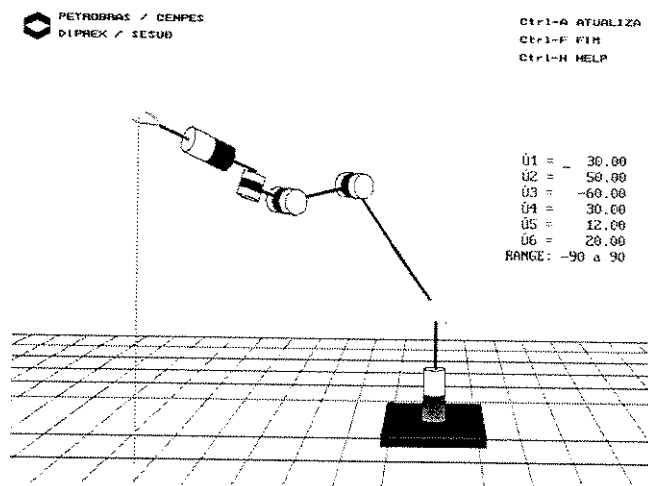


Figura A II.12: Simulador Kraft_C, desenvolvido para a Petrobrás.

A II.2 I) Aplicativo Workspace 4.0 (como simulador de robôs)

O aplicativo Workspace 4.0 possui uma vasta livreria com modelos dos principais robôs à venda no mercado, além de acessórios, como elementos terminais, mesas etc. Um ambiente tipo CAD 3-D permite igualmente a criação de outros modelos.

Uma vez criado um modelo idealizado de um manipulador (a partir das especificações técnicas do fabricante), o modelo pode ser calibrado para reconhecer as imperfeições de um determinado robô que está sendo programado.

A identificação de parâmetros é feita automaticamente, usando-se um sistema formado por de três fios presos à extremidade do robô e fazendo-o cumprir uma sequência de movimentos.

A II.3 Fornecedores de sistemas de robótica

A partir de diversas buscas na Internet e contatos com pessoas que atuam na área, foram identificados diversos produtos semelhantes disponíveis no mercado. Embora este trabalho seja de cunho científico e não comercial, a possível existência de simuladores robóticos semelhantes ao desenvolvido, de certa forma tiraria a originalidade deste trabalho. Além disto, é preciso

identificar os possíveis fornecedores de equipamentos para a montagem de robôs, uma vez que este trabalho trata essencialmente do programa de controle do robô, e não propriamente da parte construtiva.

A II.3 a) Portland Area Robotics Society – E.U.A.

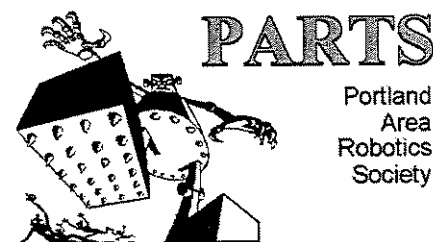


Figura A II.13: Logotipo da P.A.R.T.S.

Esta é uma organização de profissionais da área de robótica que divulgam uma revista de robótica pela Internet contendo artigos bastante interessantes ensinando a construir dispositivos robóticos simplificados.

Um dos colaboradores, o Sr. Marvin Green criou a BotBoard, utilizada neste trabalho para avaliar o microprocessador MC68HC11.

No exemplar 12, a revista traz o projeto de um braço robótico com 3 GDL, usando servos de aeromodelismo e a BotBoard como controlador.

Esta é uma das poucas referências encontradas a respeito de robôs didáticos de baixo custo, como o que foi usado para validação deste trabalho. Segundo a revista, a PARTS vende *kits* por apenas US\$12,95 (sem os servos e sem a BotBoard)

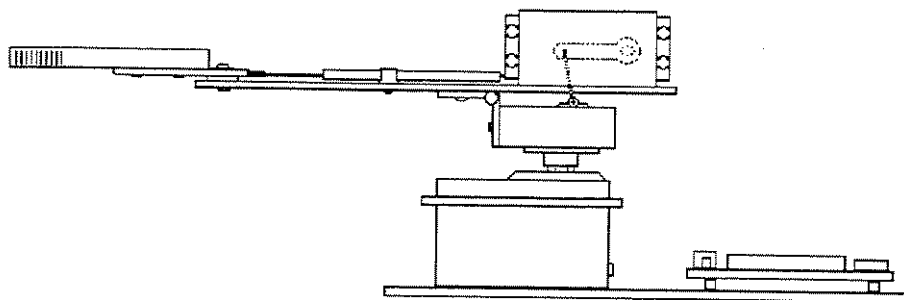


Figura A II.14: Kit de robô produzido pela PARTS

Este robô, apesar do custo extremamente reduzido, é capaz de realizar tarefas simples e mover pequenas peças. A maior dificuldade fica por conta de criar um programa no microcontrolador para mover o robô de forma adequada.

A II.3 b) – Robix – E.U.A.



Figura A II.15: Várias configurações para o Robix

São os robôs didáticos atualmente utilizados pelo LAR. Possuem uma construção reconfigurável, e são fornecidos com um aplicativo que permite a criação de seqüências de movimentos.

Porém, o aplicativo funciona em DOS, o que não permite o uso das facilidades de um sistema operacional multitarefas como o Windows. Além disso, o uso de condicionais nos programas e o acesso às portas de entrada e saída tem que ser feito com a criação de um programa em Linguagem C, específico para cada situação.

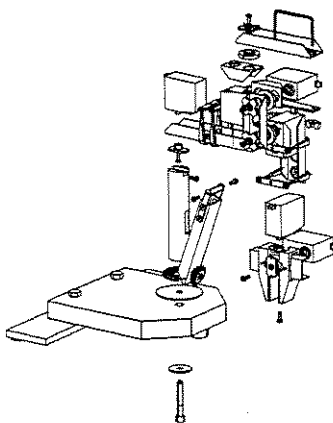


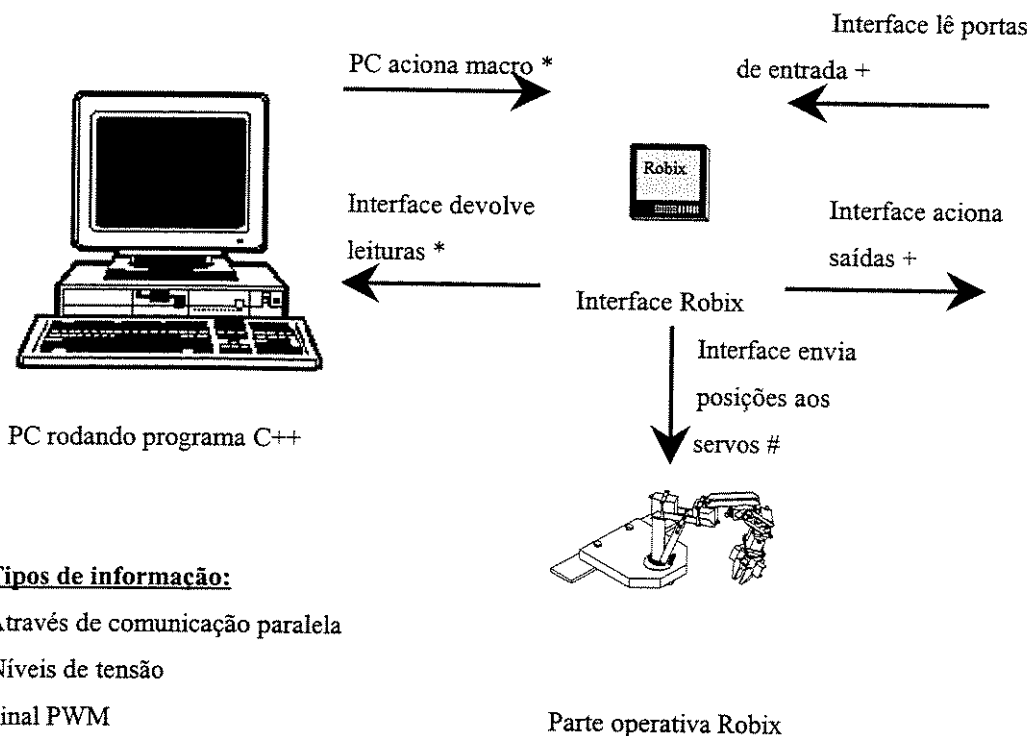
Figura A II.16: Perspectiva explodida de um manipulador Robix

Poucos usuários dominam linguagem C, e ainda assim têm dificuldade em utilizá-la, pois sua sintaxe é distinta da maioria das linguagens de programação mais modernas.

A crítica ao sistema Robix recai ainda sobre sua parte de comando conceitualmente mal concebida, utilizando uma complexa interface com processador próprio e memória para armazenar as seqüências de movimentos criadas com auxílio do PC. Além da interface não oferecer os recursos de programação estruturada, não há justificativa à utilização desta interface para armazenar programas e geração de movimentos, uma vez que isto poderia ser fisicamente realizado no PC.

O Kit custa, nos Estados Unidos US\$ 550,00 (cotação em 10/99)

Robô didático Robix



Tipos de informação:

- * Através de comunicação paralela
- + Níveis de tensão
- # Sinal PWM

Figura A II.17: Desenho esquemático da comunicação envolvendo o Robix e o PC.

Características:

- >Seis servos de radiocontrole
- >Sete entradas binárias
- >Oito 0-5 V ADCs 8 bits
- >Duas saídas 5V 125mA
- >Fonte de alimentação para o adaptador.

A II.3 c) Lynxmotion – E.U.A.

Os robôs produzidos por esta empresa eram vendidos como brinquedos sofisticados até 1999, quando a empresa passou a procurar atingir o mercado de produtos didáticos.

Estes robôs são estruturados de forma diferente dos Robix, possuindo somente uma interface que permite posicionar os servos do robô através de comunicação serial.

A interface é extremamente simplificada, e não é capaz de criar movimentos suaves, com aceleração e desaceleração. Também não possui saídas E/S, que são necessárias no desenvolvimento de aplicações que necessitam de comunicação como outros dispositivos.

A validação deste trabalho foi feita a partir de um destes robôs adquirido em 1996, um modelo anterior ao comercializado atualmente. Apesar da construção da parte operativa deste robô ser melhor que a do Robix, foi necessário o remodelamento completo para que tivesse precisão e graus de liberdade suficientes para a implementação de um modelo cinemático que funcionasse adequadamente.

A empresa não produz o *software* de controle do robô, mas existem diversos disponíveis à venda. Porém, à semelhança do que ocorre no Robix, estes *softwares* somente são capazes de criar seqüências de movimentos, sem recursos disponíveis em linguagens de programação estruturada.

A II.3 d) Eshed Robotec – Israel URL: <http://www.eshed.com/>

Produz sistemas didáticos variados, incluindo, robôs, *softwares* diversos, CIM etc.

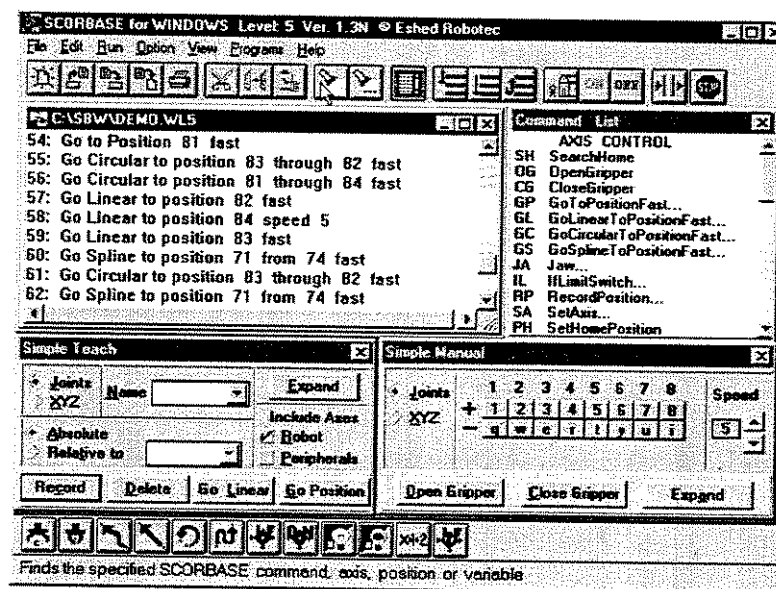


Figura A II.18: Tela do aplicativo Score Base

Os robôs didáticos são semelhantes aos industriais, inclusive em sua construção. Porém, o sistema é caro e completamente fechado, não permitindo alterações pelos usuários.

A II.3 e) Robotstore – EUA

Vende inúmeros kits de robótica, principalmente robôs autônomos. Comercializa um curioso braço robótico que pode ser controlado pelo PC, mas se trata de um mero brinquedo sofisticado pois não possui *feedback* nas juntas. Também distribui um programa para controlar robôs da Lynxmotion. O programa é capaz de memorizar e repetir seqüências de movimentos.

A II.3 f) Empresas que produzem robôs industriais

Foi tentado contato com as empresas ABB e Kuka do Brasil para levantamento de dados a respeito das características dos robôs industriais vendidos atualmente, mas estas não se prontificaram a prestar maiores esclarecimentos, ao menos a enviar um manual de utilização dos robôs comercializados.

Anexo III

A linguagem de programação

Este anexo traz uma breve explicação sobre cada um dos comandos da linguagem interna do aplicativo computacional.

Sentença Baudrate

Def. Determina a taxa de transmissão da porta serial.

Sintaxe:

Baudrate *valor*

valor: Valor da taxa de transmissão em bauds per second: 2400,9600 ou 19200.

Notas:

A taxa de transmissão é medida em *bauds per second*, e não deve ser confundido com *bits per second*. Para a transmissão de cada byte numa são usados, por exemplo, 10 bauds, se a comunicação estiver configurada para 8 bits, sem bit de paridade e com stop bit e start bit.

Sentença Clear

Def. Apaga o valor de uma variável definida pelo usuário.

Sintaxe:

Clear *varname*

varname: Nome da variável; ***all*** apaga todas variáveis.

Sentença Cls

Def. Apaga o conteúdo da janela imediata.

Sintaxe:

Cls

(não suporta argumentos)

Sentença Commport

Def. Seta a porta serial em uso.

Sintaxe:

Commport *porta*

porta: Número da porta serial a ser utilizada, ***none*** desabilita comunicação serial.

Sentença Do

Def. Repete um bloco de sentenças enquanto uma condição é verdadeira ou até uma a condição se tornar falsa.

Sintaxe:

Do [{While|Until}*condição*

[sentenças]

 [Exit Do]

[sentenças]

Loop

Ou, pode ser usada esta sintaxe:

Do

[sentenças]

[Exit Do]

[*sentenças*]

Loop [{While|Until}*condição*

Notas:

Qualquer número de *sentenças* Exit do pode ser colocado em qualquer parte no Do...Loop como uma forma alternativa de sair do loop, normalmente é executada após avaliar alguma condição, neste caso o controle é transferido para a sentença imediatamente após o Loop.

Quando houver um Do...Loop dentro de outro, Exit do sai do loop mais interno.

Caso {While|Until} esteja colocado junto ao Loop, o bloco é executado pelo menos uma vez.

Sentença For

Def. Repete um grupo de sentenças incrementando um contador..

Sintaxe:

For *contador* = *início* To *fim* [step *incremento*]

[*sentenças*]

[Exit For]

[*sentenças*]

Next *contador*

Notas:

Qualquer número de *sentenças* Exit do pode ser colocado em qualquer parte no Do...Loop como uma forma alternativa de sair do loop, normalmente é executada após avaliar alguma condição, neste caso o controle é transferido para a sentença imediatamente após o Loop.

Quando houver um Do...Loop dentro de outro, Exit do sai do loop mais interno.

Caso {While|Until} esteja colocado junto ao Loop, o bloco é executado pelo menos uma vez.

Sentença Help

Def. Exibe uma referência sobre a linguagem.

Sintaxe:

Help *palavra*

palavra: Palavra sendo consultada. Pode ser *commands*, *operators*, *functions* ou *specialvars*, para uma listagem ou nome de um comando, operador, função ou variável especial.

Sentença Here

Def. Define o Valor de uma Variável Como a Posição Atual do Robô.

Sintaxe:

Here *variável*

Variável: Nome da variável a receber o a posição atual do robô.

Sentença If...Then...Else

Def. Executa condicionalmente um grupo de sentenças, dependendo do valor de uma expressão.

Sintaxe:

If *condição* Then

[sentenças]

[Else

[sentenças else]]

End if

Condição: Uma expressão que retorne como valor true ou false

Sentenças: Sentenças executadas se a condição for verdadeira

Sentenças else: Sentenças executadas se a condição for falsa

Sentença LowerLimit

Def. Define o limite inferior para a posição dos servos. Usado para evitar danos ao robô.

Sintaxe:

LowerLimit *servo* eq *posição...*

servo: Número do servo a ter definido o limite.

posição: Mínimo valor de posição que poderá ser atribuído ao servo.

Sentença Macro...End Macro

Def. Declara o nome e código que forma o corpo de um bloco de instruções **Macro**.

Sintaxe:

Macro *nome*

 [*sentenças*]

 [Exit Sub]

 [*sentenças*]

End macro

Notas:

Qualquer número de sentenças Exit macro pode ser colocado em qualquer parte do bloco de instruções como uma forma alternativa de sair dele. Normalmente é executada após avaliar alguma condição, neste caso o controle retorna para a sentença imediatamente após a que chamou a macro.

Sentença Move

Def. Move os Servos da Posição Atual para a Posição de Destino.

Sintaxe:

Move *servo* {to|by} *posição*...

servo: Número do servo a ser movido.

posição: Posição de destino do movimento, caso seja usado to. Incremento da posição do servo caso seja usado by.

Sentença Print

Def. Imprime um texto, variável, valor de fórmula ou status de configuração na área imediata.

Sintaxe:

Print *argumento*...

argumento: Texto entre aspas, nome de variável, expressão matemática ou comando de configuração.

Notas:

Este comando tanto é útil para deixar registradas mensagens na área imediata durante a execução de um programa, quanto para verificar o valor de variáveis ou expressões matemáticas.

Sentença Prompt

Def. Interrompe a execução e exibe uma caixa de entrada, pedindo um valor ao usuário.

Sintaxe:

Prompt "*mensagem*" *variável*

mensagem: Texto entre aspas, que exibe uma mensagem indicando o significado do valor a ser digitado.

Variável: Nome da variável que irá receber o valor digitado.

Sentença Refresh

Def. Refresca a posição do robô.

Sintaxe:

Refresh

Notas:

Este comando envia a posição de todos servos conectados e estado de todas portas de saída ao(s) controlador(es). Deve ser usado na ocasião de falha na comunicação ou ao religar o robô.

Sentença UpperLimit

Def. Define o limite superior para a posição dos servos. Usado para evitar danos ao robô.

Sintaxe:

UpperLimit *servo* eq *posição*...

servo: Número do servo a ter definido o limite.

posição: Máximo valor de posição que poderá ser atribuído ao servo.

Sentença SetCaption

Def. Determina nome do servo.

Sintaxe:

SetCaption *servo* eq “*nome*”...

servo: Número do servo ao qual o nome será atribuído.

nome: Nome do servo

Notas:

O nome do servo é exibido na teach box e no simulador.

Sentença Show

Def. Exibe a expressão associada a uma variável sem avaliá-la.

Sintaxe:

Show *variável*

Notas:

Este comando difere do print porque com ele é exibido o valor da expressão atribuída a uma variável, e não o valor resultante de sua avaliação.

Sentença Speed

Def. Determina Velocidade Atual, Relativo ao SpeedLimit.

Sintaxe:

Speed *valor*

valor: Valor da velocidade máxima a ser usada para movimentar o robô. Deve ser entrado um valor de 0 a 100.

Sentença SpeedLimit

Def. Determina o limite de velocidade para cada servo.

Sintaxe:

SpeedLimit *servo* eq *valor*...

servo: Número do servo ao qual será atribuída a velocidade.

valor: Valor do tempo de resposta do servo, em centésimos de segundo para mover da posição 0 à máxima posição endereçável.

Notas:

A rigor, o comando SpeedLimit não define a velocidade máxima do servo, e sim o tempo de mover o servo da posição 0 à máxima posição de servo que o protocolo de comunicação aceita. O tempo é medido em centésimos de segundo.

Sentença Wait

Def. Cria uma espera, em 1/1000 s.

Sintaxe:

SpeedLimit *tempo*

tempo: Tempo da espera, em milissegundos.

Notas:

A execução do programa é interrompida até que seja decorrido o tempo de espera.

Sentença Where

Def. Exibe a posição atual do robô.

Sintaxe:

Where

Notas:

Este comando é útil para pegar a posição dos servos e usá-las para definir o UpperLimit ou LowerLimit.

Anexo IV

Representação cinemática de manipuladores robóticos

A IV.1 - Definição do modelo

Para modelar um manipulador robótico usando a modelagem de DH, foram necessárias diversas considerações adicionais ao método proposto, como por exemplo as equações de vínculo entre servos e juntas, orientação do elemento terminal etc..

Para que o aplicativo fosse genérico, capaz de modelar qualquer manipulador independente de sua configuração cinemática, foi necessário criar um conjunto de variáveis especiais cujo valor determina o modelo do manipulador.

Para tornar eficiente o cálculo do modelo do manipulador, foram inseridas algumas variáveis redundantes, que somente aceitam valores constantes, enquanto outras aceitam também expressões com matemática simbólica. Isto permite reduzir o uso do operador matemático, acelerando a regeneração do modelo do manipulador.

Para ilustrar melhor a metodologia adotada para modelagem de um robô, foram definidos quatro sistemas de coordenadas: coordenadas do controlador, dos servos, das juntas e cartesiano, e três transformações entre eles.

A IV.1.1 Coordenadas do controlador para coordenadas dos servos

Esta transformada realiza a conversão dos valores comandados aos servos, via porta serial, para o sistema de coordenadas dos servos (graus, radianos ou qualquer outra unidade angular, caso o servo seja angular ou milímetros, frações de polegada ou qualquer outra unidade linear caso o servo seja um atuador linear).

O controlador trabalha sempre com números inteiros que não representam diretamente uma unidade de medida específica. Estão direta ou indiretamente associados, à *leitura* do *encoder* dos servos, determinando um sinal de referência que será perseguido pelo controlador da junta.

Considerando que o *encoder* seja linear, usando-se uma transformação linear do tipo $y = a.x + b$, sempre será possível representar o valor comandado do servo, nas unidades de medida adotadas.

A principal vantagem de se definir esta transformada intermediária ao invés de tentar criar uma transformada diretamente do espaço do controlador para o espaço cartesiano é que se torna possível recalibrar os parâmetros das juntas do robô, sem necessidade de se alterar o modelo cinemático do robô.

Os valores das constantes desta transformada são de natureza essencialmente empírica, ou seja, a princípio devem ser obtidos a partir de medições no próprio manipulador robótico e portanto, podem necessitar de reavaliações periódicas. Possivelmente, dependendo do tipo de atuador utilizado, até será possível deduzir matematicamente a constante de proporcionalidade, mas este não representa o caso genérico.

No aplicativo desenvolvido, os coeficientes desta transformada são definidas usando-se a variável especial **ServoConstant*i***, cuja primeira coluna representa o coeficiente de multiplicação e a segunda, uma constante somada ao valor.

A transformada de coordenadas dos controladores para coordenadas dos servos fica definida como sendo:

$$C = A.S + B$$

Onde C é o vetor posição no espaço dos controladores e S , o mesmo vetor no espaço dos servos, e A e B são vetores constantes. O Anexo IV traz exemplos que ajudam a ilustrar a metodologia para definição do modelo de um robô.

A IV.1.2 Coordenadas dos servos para coordenadas das juntas

As coordenadas dos atuadores representam a posição comandada de cada um dos atuadores, em alguma unidade de medida adotada. Na maioria das configurações de robôs, não existe diferença entre coordenadas dos atuadores e das juntas. Porém em alguns casos será preciso considerar a existência de um espaço dos atuadores diferente do espaço das juntas.

O valor do ângulo ou comprimento de servo não necessariamente representa um ângulo ou comprimento de junta devido à ocorrência de qualquer uma das seguintes situações:

- i) A posição de uma junta é determinada em função da posição de outras juntas. Um exemplo é o caso do robô com estrutura em paralelogramo, no qual os dois servos que comandam a estrutura do paralelogramo são solidários à base, logo o ângulo da segunda junta será subtraído do ângulo da primeira junta.
- ii) Existe algum mecanismo que transmite movimento entre o servo e a junta que precisa ser levados em consideração. Por exemplo um sistema de transmissão de movimento não seja linear, como uma biela-manivela.
- iii) O modelo leva em consideração erro causado por distorção de elos ou erro estático de posicionamento de juntas, por exemplo que tenham controlador tipo proporcional. Neste caso, a transformada será obtida por uma equação matemática levando em consideração o peso próprio do robô, a rigidez dos servos e quaisquer outros fatores considerados relevantes.
- iv) Os servos estão endereçados pelo controlador em uma ordem diferente que estão montados no manipulador robótico.

Para considerarmos estes casos, a transformada entre o coordenadas dos atuadores e o

coordenadas das juntas fará a conversão dos valores comandados aos servos para os valores comandados às juntas.

A princípio esta transformada surge de uma dedução matemática baseada na geometria da construtiva do robô e expressões obtidas não são de natureza empírica e em geral sua expressão matemática é simples.

Em uma grande parte dos casos esta transformada é trivial, ou seja, a função que determina posição da junta em relação à posição do servo é do tipo: $junta1 = servo1$; $junta2 = servo2$ e assim por diante. Isto ocorre nos robôs tipo Puma, por exemplo.

Tomemos um exemplo para ilustrar o caso em que esta transformada será usada para reenumerar os servos, ou seja, modificar a ordem de associação entre servos e juntas. A princípio, o servo de índice 1 comanda a junta número 1, o servo 2 a junta 2 e assim por diante.

Suponha-se que, por alguma necessidade prática, os servos estejam montados em uma forma diferente da que são indexados pelo controlador. Isto pode ocorrer por exemplo, no caso do manipulador robótico utilizar motores de passo para controlar as juntas 2, 3 e 5 e servos para as juntas 1, 4 e 6.

A princípio, neste caso serão utilizados dois controladores diferentes; suponhamos que para controlar os servos, seja usado um controlador que assuma os endereços dos servos 1 a 8 e para os motores de passo, um controlador que assuma os endereços de 9 a 15. Neste caso, o endereço dos servos não coincide com o número da junta. Seria preciso associar às juntas 1 a 6 respectivamente os servos 1, 9, 10, 2, 11 e 3.

Esta transformada poderá ser usada na conversão de unidades diferentes usadas para representar posição dos servos e para das juntas. É possível trabalhar, por exemplo, com centésimos de grau como coordenadas de servos e graus para medir ângulo das juntas.

Como será visto mais adiante, esta transformada é incorporada na tabela de parâmetros de Denavit e Hartenberg.

A IV.1.3 Coordenadas de juntas para espaço cartesiano

Esta é a transformada proposta por Denavit e Hartenberg. Como descrito anteriormente, realiza a transição das coordenadas das juntas para o espaço cartesiano, criando o modelo cinemático do manipulador.

Além do manipulador robótico, o programa também modela o elemento terminal representado por uma garra montada na extremidade do último elo. A orientação da garra determina a maneira pela qual esta é montada sobre a extremidade do manipulador robótico.

É definida uma transformação que permite determinar a orientação da garra em relação ao último sistema de coordenadas. Para manter homogeneidade com a forma de exibição da orientação no simulador, optou-se por determinar pela utilização dos ângulos de *Yaw*, *Pitch* e *Roll*.

A princípio a garra é montada com seu eixo principal sobre o eixo Z do último sistema de coordenadas e conseqüentemente, transversal ao comprimento do último elo, **a**.

Caso a última junta faça a rotação da garra, o parâmetro **a** assume um valor igual a zero e **d** representa o comprimento da junta. Neste caso, os ângulos de orientação da garra são zero. Caso o parâmetro **a** seja diferente de zero e a garra esteja montada longitudinalmente ao último elo, é preciso usar uma orientação para a garra diferente de zero.

Além da orientação, o simulador necessita que seja definido o comprimento da garra e foi levado em consideração que este pode variar (por exemplo, estar associado à sua abertura e fechamento, como ocorre em muitos robôs).

Para permitir o caso genérico, o simulador trabalha com uma expressão matemática que determina o comprimento da garra, podendo este ser constante, ou uma expressão algébrica em função da posição dos servos ou de variáveis definidas pelo usuário no programa.

Da mesma forma, a abertura da garra utiliza uma expressão matemática. Esta modelagem genérica possibilita inclusive, simulação do caso em que o movimento de abertura e fechamento é feito a partir de um dispositivo pneumático ou semelhante, que somente comanda posição aberta ou fechada, sem posições intermediárias. Neste caso, a abertura da garra estaria associada a uma variável definida pelo usuário.

Outro recurso implementado no simulador é o cálculo da orientação inversa, ou seja, a partir de uma matriz de transformação homogênea, calcular os ângulos de *Roll*, *Pitch* e *Yaw*. O desenvolvimento da teoria associado a este recurso está documentado no Anexo IV.

A IV.2 Parâmetros de Denavit-Hartenberg

Usando o método proposto por Denavid e Hartenberg, é possível determinar os parâmetros cinemáticos que descrevem a geometria das sucessivas juntas de qualquer manipulador. Existem diferenças nas metodologias adotadas para implementar o modelo cinemático, e neste trabalho foram adotadas as definições propostas por Spong, (1989). Este livro traz explicações bastante detalhadas sobre o assunto e é largamente utilizado como referência bibliográfica.

O método consiste em se atribuir um sistema de referência para cada junta, sucessivamente, até a construção o modelo de todo o manipulador. Isto poderia ser feito com matemática genérica usando, por exemplo, as transformações homogêneas. Porém, o método proposto reduz de seis para quatro o número de parâmetros usados para relacionar uma junta com a seguinte.

Isto é possível porque supõe que cada junta possui apenas um grau de liberdade, restringindo as possibilidades de movimentos que podem ocorrer em um caso genérico de translação e rotação entre 2 vetores no espaço.

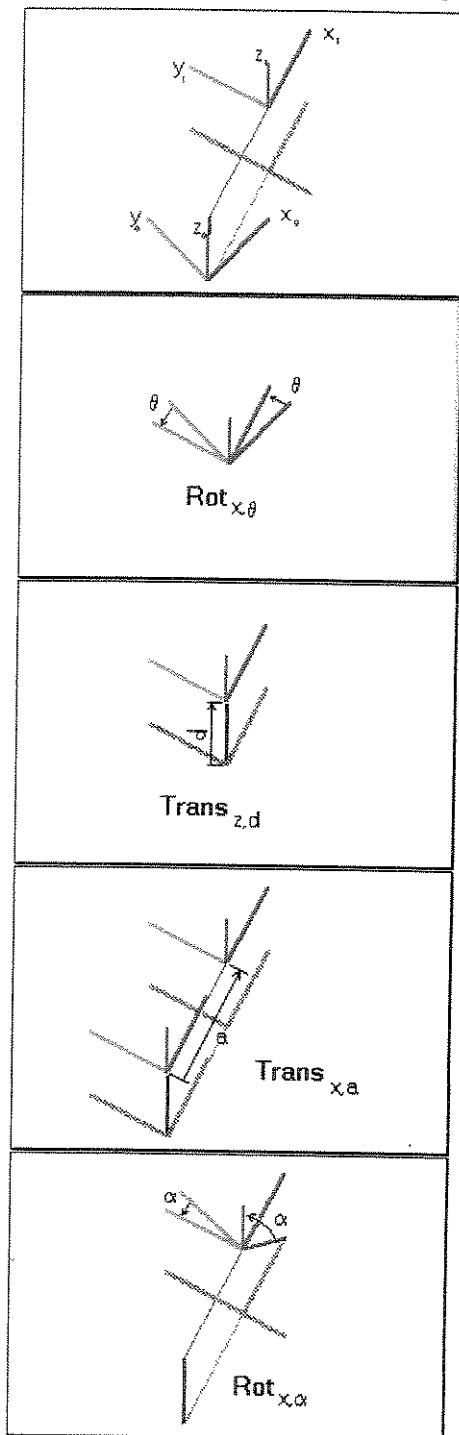
Este método é aplicável a qualquer tipo de manipulador, pois a única restrição imposta ao caso genérico é a de que cada junta possua apenas um grau de liberdade. Juntas que eventualmente possuam dois graus de liberdade podem ser modeladas como se fossem duas colocadas em série.

O método proposto por Denavit e Hartenberg consiste em, a partir de um sistema de coordenadas, descrever o sistema seguinte através de 4 transformações elementares, realizadas cada uma em referência ao novo sistema de coordenadas obtido:

$$A_i = \text{Rot}_{z,\theta} \times \text{Trans}_{z,d} \times \text{Trans}_{x,a} \times \text{Rot}_{x,\alpha}$$

Ou seja, uma rotação no eixo z , de θ , uma translação no novo sistema obtido, de d na direção do eixo z , seguido de uma translação no eixo x de a e após, uma rotação no eixo x de α .

Transformada de Denavit-Hartenberg



A transformada de Denavit-Hartenberg cria uma matriz de transformação homogênea a partir de 4 parâmetros, representando 4 transformações elementares

Partindo de um sistema $x_0y_0z_0$ chegamos a um novo sistema de coordenadas $x_1y_1z_1$.

1- Rotação de θ em torno do eixo z do sistema inicial. Caso se trate de uma junta rotacional, θ será a variável independente e os demais parâmetros são constantes.

2- Uma translação de d em relação ao eixo z do novo sistema obtido. Caso a junta seja prismática, d será uma variável independente e os demais parâmetros são constantes.

3- Uma translação de a em relação ao eixo x do novo sistema obtido. O valor de a representa o comprimento do elo, e é constante.

4 - Rotação de α em torno do eixo x do novo sistema de coordenadas obtido. O valor de α é sempre fixo para uma determinada configuração e não pode representar um grau de liberdade da junta.

Figura A IV.1: A seqüência de transformadas elementares que formam a transformada de Denavit-Hartenberg

Na forma matricial, a equação fica:

$$A_i = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & 0 \\ s\theta_i & c\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\alpha_i & -s\alpha_i & 0 \\ 0 & s\alpha_i & c\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_i = \begin{bmatrix} c\theta_i & -s\theta_i c\alpha_i & s\theta_i s\alpha_i & a_i c\theta_i \\ s\theta_i & c\theta_i c\alpha_i & -c\theta_i s\alpha_i & a_i s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Determinados os parâmetros cinemáticos θ_i , d_i , a_i e α_i , aplicando-se sucessivas vezes a transformada obtida a partir destas quatro transformações elementares, se obtém o modelo cinemático de qualquer manipulador robótico.

A colocação dos sistemas de coordenadas merece algumas observações importantes:

- i) A disposição dos sistemas de coordenadas para um dado manipulador não é necessariamente única.
- ii) O eixo z aponta na direção da rotação ou translação da junta. Se a junta for rotacional, θ representa o grau de liberdade da junta, d representa o deslocamento entre as juntas e a , o comprimento do elo. Se a junta for prismática, a representa o grau de liberdade da junta deslizante e θ é constante. Por convenção, α e d sempre são constantes.
- iii) O eixo x irá sempre apontar a direção da rotação entre o sistema de coordenadas atual e o sistema seguinte, (do qual a princípio temos a direção do eixo z , apontando o grau de liberdade da junta) sendo α o valor desta rotação.

iv) Devido à sucessão de transformações que compõe o método de Denavit & Hartenberg, surge a regra para prática que dita: O eixo x sempre é ortogonal ao eixo z anterior, e o eixo x sempre intersecciona o eixo z do sistema anterior. Isto decorre da translação em x de a , que é seguida por uma rotação também em x .

Para melhorar a visualização do modelo, a transformação entre um sistema de coordenadas e o seguinte foi quebrado em duas partes: um vetor simbolizando o deslocamento entre as juntas d e um segundo vetor, simbolizando o comprimento do elo, a .

Para dividir o deslocamento entre um sistema de coordenadas e o seguinte em duas partes, foi usado um pequeno artifício matemático: observando a seqüência de transformadas elementares, a primeira rotação que compõe a primeira transformação e a translação que se segue são realizadas sobre o mesmo eixo x , foi possível trocar a ordem destas duas transformações, o que não altera o resultado. Veja a comprovação matemática:

$$A_i = \text{Trans}_{z,d} \times \text{Rot}_{z,\theta} \times \text{Trans}_{x,a} \times \text{Rot}_{x,a}$$

Para isto, é preciso que:

$$\text{Rot}_{z,\theta} \times \text{Trans}_{z,d} = \text{Trans}_{z,d} \times \text{Rot}_{z,\theta}$$

Em forma matricial, temos:

$$\begin{bmatrix} c\theta_i & -s\theta_i & 0 & 0 \\ s\theta_i & c\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & 0 \\ s\theta_i & c\theta_i & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

E também, que:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} c\theta_i & -s\theta_i & 0 & 0 \\ s\theta_i & c\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & 0 \\ s\theta_i & c\theta_i & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

C.Q.D., a igualdade é válida, e as equações são equivalentes.

Desta forma, pode-se extrair de uma transformação o deslocamento d_i , que é desenhado em relação ao sistema anterior. Uma linha traçada entre este ponto e o ponto obtido após a transformação A_i representa o segundo deslocamento, de a em x . Isto permite economia de operações computacionais gastas para a criação do modelo, aproveitando de uma matriz homogênea para desenhar dois segmentos de linha.

O valor de d pode ser obtido diretamente a partir da transformação homogênea A_i , pegando-se o elemento (3,4). O algoritmo que desenha os modelos pode ser visto na subrotina Plota, do formulário frmDHsim.

A IV.3 Orientação inversa RPY

A orientação do manipulador no espaço pode ser representada de diversas maneiras, como ângulos de Euler *Yaw*, *Pitch* e *Roll*, na forma matricial etc. No caso, optou-se pelos ângulos *Yaw*, *Pitch* e *Roll*, para representar a orientação da garra em relação ao sistema de coordenadas inercial.

Estes ângulos representam uma seqüência de três transformações homogêneas de rotação em relação a um mesmo sistema de coordenadas :

Rotação em x , do ângulo *Yaw*, representado pela letra grega Ψ ; rotação em y , do ângulo de

Pitch , representado por θ e rotação em z , do ângulo de *Roll.*, representado por ϕ .

A transformação resultante é, segundo Paul, (1981), p.70:

$$R_0^1 = R_{z,\phi} \cdot R_{y,\theta} \cdot R_{x,\psi}$$

Note-se que as rotações aparecem na ordem invertida, pois por definição são realizadas relativamente a um mesmo sistema de coordenadas. Na forma matricial, tem-se:

$$R_0^1 = \begin{bmatrix} \cos \phi & -\sin \phi & 0 \\ \sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \psi & -\sin \psi \\ 0 & \sin \psi & \cos \psi \end{bmatrix}$$

$$R_0^1 = \begin{bmatrix} \cos \phi \cdot \cos \theta & -\sin \phi \cdot \cos \psi + \cos \phi \cdot \sin \theta \cdot \sin \psi & \sin \phi \cdot \sin \psi + \cos \phi \cdot \sin \theta \cdot \cos \psi \\ \sin \phi \cdot \cos \theta & \cos \phi \cdot \cos \psi + \sin \phi \cdot \sin \theta \cdot \sin \psi & -\cos \phi \cdot \sin \psi + \sin \phi \cdot \sin \theta \cdot \cos \psi \\ -\sin \theta & \cos \theta \cdot \sin \psi & \cos \theta \cdot \cos \psi \end{bmatrix}$$

Para o caso inverso, ou seja, dada a transformação homogênea, encontrarmos os ângulos, é preciso certos cuidados especiais no uso das funções trigonométricas inversas.

Será resolvida a equação:

$$R_{z,\phi}^{-1} \cdot R_0^1 = R_{y,\theta} \cdot R_{x,\psi}$$

onde:

$$\begin{bmatrix} f_{11}(n) & f_{12}(o) & f_{13}(a) & 0 \\ f_{21}(n) & f_{22}(o) & f_{23}(a) & 0 \\ f_{31}(n) & f_{23}(o) & f_{33}(a) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta \cdot \sin \psi & \sin \theta \cdot \cos \psi & 0 \\ 0 & \cos \psi & -\sin \psi & 0 \\ -\sin \theta & \cos \phi \cdot \sin \psi & \cos \phi \cdot \cos \psi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$f_{11}(n) = \cos \phi . x + \sin \phi . y$$

$$f_{12}(n) = -\sin \phi . x + \cos \phi . y$$

$$f_{13}(n) = z$$

Equacionando, vem:

$$\phi = \arctan 2(n_y, n_x)$$

Equacionando os elementos 3,1 e 1,1 no lado direito da equação, vem:

$$\theta = \arctan 2(-n_z, \cos \phi . n_x + \sin \phi . n_y)$$

E finalmente equacionando os elementos 2,3 de 2,2 temos:

$$\psi = \arctan 2(\sin \phi . a_x - \cos \phi . a_y, -\sin \phi . o_x + \cos \phi . o_y)$$

Estas funções estão inseridas na subrotina `Plot` do simulador para calcular os ângulos de orientação da garra, exibidos na perspectiva isométrica.

Foi criado um modelo para testar o funcionamento do algoritmo de orientação inversa. É um manipulador cujas juntas 1, 2 e 3 movimentam, respectivamente, o *Roll*, *Pitch* e *Yaw* da garra e assim é possível verificar se os valores encontrados pelo modelo inverso correspondem à correta orientação da garra.

Verificando-se os resultados, ficou evidente a necessidade de se estabelecer $Roll = 0$, quando $Pitch = 90$, pois este é um caso degenerado, no qual o cálculo do ângulo de rolagem resulta do arco tangente de um valor muito pequeno e neste caso, a orientação da garra depende

de *Roll* + *Yaw*. O valor de *Roll* pode ser arbitrado como zero sem prejuízo à correta orientação da garra. Porém, os valores obtidos para os ângulos oscilam bruscamente.

Este caso pode ser evitado se for somado um pequeno valor ao ângulo de *Pitch* quando este atingir 90 ou -90 graus. Porém, a princípio a matriz de transformação homogênea é calculada em consequência dos ângulos das juntas (para um dado robô) e seria necessário um procedimento tipo tentativa e erro, variando-se de um pequeno valor o ângulo das juntas e recalculando a matriz de transformação homogênea.

Como o programa foi feito utilizando variáveis de 32 bits, uma pequena diferença do *Pitch* fora dos 90 graus é suficiente para que o cálculo seja feito com precisão. A partir de testes com valores de *Pitch* perto do caso degenerado, foi estabelecido um critério de que se o módulo da soma dos elementos (1,1) e (2,1) da matriz homogênea for menor que 0.001, considera-se que se trata de um caso degenerado.

Valores menores que o adotado para este critério levam a valores aleatórios no cálculo do ângulo de *Roll*, porém mesmo neste caso os ângulos de orientação continuam a representar a orientação correta para a garra.

Como os usuários tendem a entrar números inteiros no modelo, a probabilidade de se atingir o caso degenerado pode ser reduzida somando-se 0.001 a todos os valores de teta da tabela dos parâmetros de Denavit-Hartenberg (para teta medido em radianos).

Como $\cos(\pi + 0.001) = 0.001000$, e no mínimo a soma dos elementos (1,1) e (2,1) da matriz homogênea resulta este valor vezes raiz de dois (para *Roll* = 45 graus), tem-se que o caso degenerado nunca é atingido se o ângulo de *Pitch* for inteiro.

Outra consideração importante que, dada uma matriz de transformação homogênea, sempre existem duas soluções para os ângulos RPY. Uma vez que o valor de *Roll* é avaliado a partir dos elementos (1,1) e (2,1) da matriz homogênea, e ambos os elementos são multiplicados por $\cos(\theta)$,

este termo pode inverter o sinal de ambos os elementos avaliados, causando uma diferença de 180 graus no cálculo do ângulo.

Esta ambigüidade não pode ser eliminada, uma vez que ambos os resultados levam à mesma matriz homogênea, ou seja, as duas soluções para o cálculo da orientação inversa da garra são igualmente válidas.

Por isto, adotou-se que o ângulo de *Roll* sempre está entre -90 e 90 graus, e se o cálculo resultar fora deste intervalo, soma-se ou subtrai-se 180 do valor obtido para *Roll* conforme o caso (esta operação deve ser feita antes da avaliação dos ângulos de *Pitch* e *Yaw*, para manter a coerência dos cálculos)..1.2.6 A função Arco Tangente 2

No cálculo de funções trigonométricas inversas, é preciso se tomar cuidado com as singularidades que podem ocorrer. A melhor forma de realizar os cálculos é utilizando a função Arco Tangente 2, que nada mais é que a função Arco Tangente, porém avaliando o sinal do numerador e denominador. Deve-se evitar o uso das funções ArcoSeno e ArcoTangente, pois estas podem assumir um valor próximo a zero, em casos degenerados.

Como foram usados nos programas variáveis de ponto flutuante tipo Double, o número de casas decimais representadas é muito elevado, tornando os programas bastante tolerantes a números demasiado pequenos ou demasiado grandes.

Segundo Doughty (1988), p.386~387, a função Arco Tangente 2 pode ser descrita como:

Tabela A IV.1: Valores de ângulos e função atan2 associados a intervalos de seno e cosseno de um ângulo.

sen(θ)	cos(θ)	atan2(sen(θ),cos(θ))	θ [graus]
sen(θ) ≥ 0	cos(θ) > 0	atan(sen(θ)/cos(θ))	$0^\circ \leq \theta < 90^\circ$
sen(θ) > 0	cos(θ) = 0	90	$\theta = 90^\circ$
sen(θ) > 0	cos(θ) < 0	atan(sen(θ)/cos(θ))+180	$90^\circ < \theta < 180^\circ$
sen(θ) ≤ 0	cos(θ) < 0	atan(sen(θ)/cos(θ))-180	$0^\circ \leq \theta < -90^\circ$
sen(θ) < 0	cos(θ) = 0	-90	$\theta = -90^\circ$
sen(θ) < 0	cos(θ) > 0	atan(sen(θ)/cos(θ))	$-90^\circ \leq \theta < -180^\circ$

Na prática, quando se trabalha com variáveis de ponto flutuante tipo *double*, não é preciso considerar os casos especiais em que $\cos(\theta)=0$, pois $\cos(90)=6,12574\text{E-}17$, ao invés de zero e tangente de $90=16324552277619100,00$, ao invés de infinito. Análogo para 270 graus. Apesar disto, estes casos foram considerados à parte como medida de precaução.

Outra forma de definir a função atan2 seria considerar valores sempre positivos para o ângulo, num intervalo de $[0,360[$. Esta definição seria igualmente válida, visto que $\theta = (\theta + 360)$.

No aplicativo desenvolvido, a função Arco Tangente 2 foi implementada na *Funcion* ArcTan2.

A IV.4 Estudos de caso: Modelagem de diversos manipuladores

Esta parte do trabalho apresenta diversas configurações de robôs e mostra como foi feita a modelagem, utilizando o aplicativo desenvolvido neste trabalho.

Isto demonstra a versatilidade e facilidade na criação de modelos representando diferentes configurações de robôs.

A partir do modelo, é possível fazer a programação *off-line* e posteriormente verificar o funcionamento do programa com um robô real, ou mesmo simular o funcionamento de um robô

que não existe fisicamente.

A IV.4.1 Robô Puma

A configuração Puma é composta por seis graus de liberdade, todos com juntas rotativas. É uma das mais divulgadas, devido à sua versatilidade.

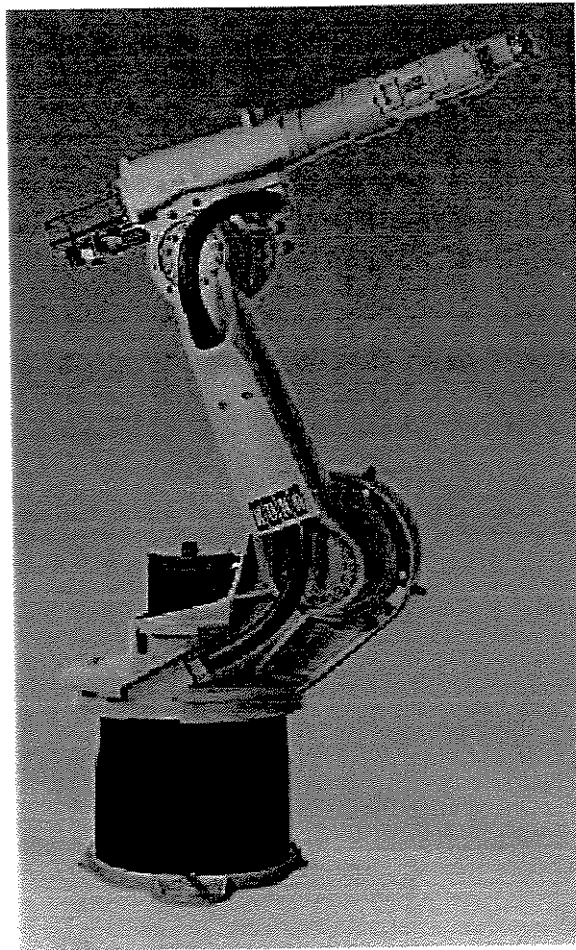


Figura A IV.2: Robô com configuração Puma.
Kuka KR-6 Technical Data, URL: www.kuka-roboter.de, 20/12/99

Um exemplo de instruções que criam o modelo de um robô Puma no aplicativo é composto pelas sentenças:

Macro Config

```
s1=-180    %ângulo da junta na posição 1
c1=0       %valor enviado ao controlador
s2=180     %ângulo da junta na posição 2
c2=1023    %valor enviado ao controlador

ServoConstant1 = (s2-s1)/(c2-c1) ; s2-((s2-s1)/(c2-c1))*c2 %a;b
ServoConstant2 = ServoConstant1
ServoConstant3 = ServoConstant1
ServoConstant4 = ServoConstant1
ServoConstant5 = ServoConstant1
ServoConstant6 = ServoConstant1
ServoConstant7 = ServoConstant1

SetCaption 1 eq "waist" 2 eq "should" 3 eq "elbow" 4 eq "wrist" 5 eq
            "hand" 6 eq "flange" 7 eq "grip"
```

End macro

Macro SetUp

```
% modelo do robô Puma - tabela usando expressões
Config
joints = 6

% ExtDHTable = ExpTheta    Expd    a    alfa    aWid    dWid
extdhtable1 = 'servo1';    5;    0;    90;    3;    0
extdhtable2 = 'servo2+90'; 3;    9;    0;    2,8; 2,2
extdhtable3 = 'servo3';    -3;    0;    90;    2;    2
extdhtable4 = 'servo4';    8;    0;    -90;    1;    1,2
extdhtable5 = 'servo5';    0;    0;    90;    0;    1
extdhtable6 = 'servo6';    1,2; 0;    0;    0,5; 0

%GripOr = GripOrYaw GripOrPitch GripOrRoll
gripor = 0;    0;    0;
GripLen = '3-servo7/360'
%GripWidth= ExpWidth    KWidth Width0
```

```
GripWidth = 'servo7/90+3'
Gripthick = 0,5
end macro
```

Estas duas macros determinam todo o modelo do robô, sendo que a macro Config trata das configurações dos servos, e a macro SetUp insere a tabela de parâmetros de Denavit & Hartenberg extendida e outros parâmetros, como a orientação do grip em relação ao último elo, sua abertura, comprimento etc.

Note-se a dependência estabelecida entre o comprimento do *grip* e sua abertura. A expressão estabelece que o comprimento varia de 3 unidades, fechado a 2 unidades, quando completamente aberto.

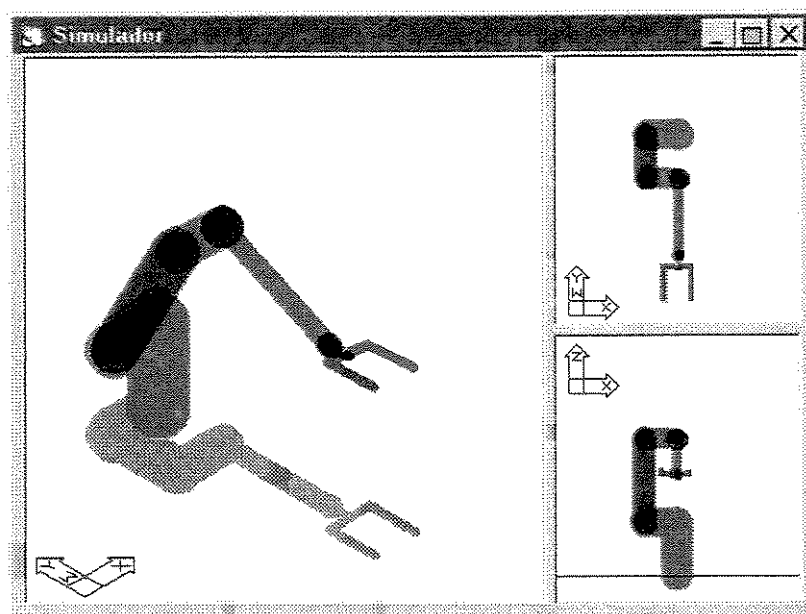


Figura A IV.3: Janela do simulador exibindo o modelo do robô Puma.

Como a relação entre os servos e juntas é trivial, ou seja, o servo 1 comanda a junta 1, o servo 2 a junta 2 e assim por diante, é possível evitar o uso de expressões no modelo do robô, o que torna o processamento ligeiramente mais rápido. O modelo usando valores constantes fica:

```
Macro SetUp2
```

```

% modelo do robô Puma - tabela com valores constantes
config
joints = 6
%ExtDHtable = ExpTh Expd a alfa aWid dWid kTheta Theta0 kd d0
extdhtable1 = 0; 0; 0; 90; 3; 0; 1; 0; 0; 5
extdhtable2 = 0; 0; 9; 0; 2,8; 2,2; 1; 90; 0; 3
extdhtable3 = 0; 0; 0; 90; 2; 2; 1; 0; 0; -3
extdhtable4 = 0; 0; 0; -90; 1; 1,2; 1; 0; 0; 8
extdhtable5 = 0; 0; 0; 90; 0; 1; 1; 0; 0; 0
extdhtable6 = 0; 0; 0; 0; 0,5; 0; 1; 0; 0; 1,2
%GripOr = GripOrYaw GripOrPitch GripOrRoll
gripor = 0; 0; 0;
GripLen = '3-servo7/360'
%GripWidth= ExpWidth KWidth Width0
GripWidth = 0; 1/90; 3
Gripthick = 0,5
end macro

```

Para averiguar se o modelo está inserido corretamente, pode-se verificar a fórmula inserida para cada linha da tabela DH estendida. Por exemplo, digitando:

```

]show ExtDhtable2
servo2 + 90 ; 3 ; 9 ; 0 ; 2,8 ; 2,2

```

O comando **show** exibe as expressões matemáticas que estão sendo usadas na tabela de parâmetros DH estendida. Assim, é possível verificar as expressões para teta e obtidas a partir dos valores constantes entrados nas colunas 1, 7, 8 e 2, 9, 10 respectivamente.

Como pode ser verificado, a linha da tabela DH estendida corresponde exatamente à que foi inserida no modelo anterior, o que significa que o modelo é o mesmo. No entanto, a performance do simulador subiu de 10,33 quadros por segundo para 12,13. Isto devido ao fato do modelo não

fazer uso do operador matemático do aplicativo na geração do modelo (avaliado em um Pentium 166MHz, em resolução 800x600, High Color).

O modelo anterior usava 7 vezes a mais o operador matemático para desenhar cada quadro que este segundo modelo. O tempo de regeneração de cada quadro caiu de 97 para 82 milissegundos, indicando que a diferença, de 14 milissegundos corresponde ao uso de 7 vezes o operador matemático, numa média de 2 milissegundos por expressão matemática avaliada.

De forma análoga, o comando show exhibe a fórmula usada para abertura do grip, agora inserida no modelo usando valores constantes (Obs.: $1/90 = 0,011111\dots$).

```
]show GripWidth  
1,11111111111111E-02 * servo7 + 3
```

A IV.4.2 Robô Quatro-barras

Esta configuração de robô é bastante semelhante à do Puma, no entanto um mecanismo quatro-barras transmite o movimento da terceira junta através de uma barra de comando. Os servos 2 e 3 são montados sobre o primeiro elo, o que leva ao desacoplamento do movimento da junta 2 em relação à junta 3.

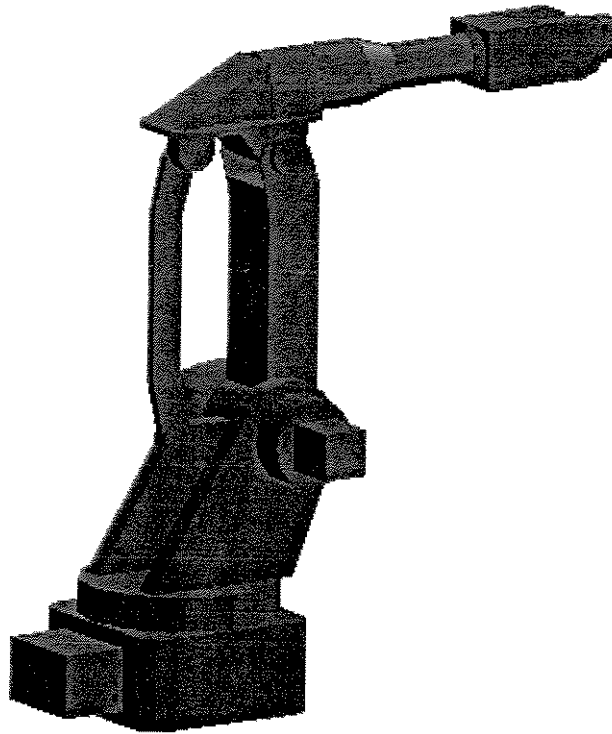


Figura A IV.4: Estrutura cinemática de um robô quatro barras.
Semelhante ao Puma, este robô possui uma barra de comando que transmite o movimento para a junta. Isto permite montar o servo junto à base e desacoplar o movimento das juntas 2 e 3. (Imagem gerada pelo software Virtual Workspace 4.0)

Neste caso, a tabela de parâmetros de DH estendida é dada por:

Macro SetUp3

% modelo do robô Puma, configuração 4 barras, usando expressões

SetUp

joints = 6

%ExtDHtable = ExpTheta	Expd a	alfa	aWid	dWid
------------------------	--------	------	------	------

extdhtable1 = 'servo1';	5;	0;	90;	3; 0
-------------------------	----	----	-----	------

extdhtable2 = 'servo2 + 90';	3;	9;	0;	2,8; 2,2
------------------------------	----	----	----	----------

extdhtable3 = 'servo3-servo2-90';	-3;	0;	90;	2; 2
-----------------------------------	-----	----	-----	------

extdhtable4 = 'servo4';	8;	0;	-90;	1; 1,2
-------------------------	----	----	------	--------

extdhtable5 = 'servo5';	0;	0;	90;	0; 1
-------------------------	----	----	-----	------


```

        extdhtable6 = 'servo6';          1,2; 0;    0;    0,5; 0
end macro

```

Note-se, que a única diferença no modelo em relação ao Puma está na influência dos servos na junta 3. Como a posição do servo 2 não influencia a junta 3, no modelo é preciso subtrair seu valor (servo2+90) da junta 3, uma vez que o modelo considera que cada junta é solidária ao elo anterior.

O desenho do modelo é exatamente o mesmo do robô Puma, já que o restante do modelo permaneceu inalterado (as linhas para configuração do grip foram suprimidas, logo o aplicativo mantém inalterado seus valores).

A IV.4.3 Robô Pick-and-place

Este tipo de robô é semelhante a um robô Quatro-barras, no entanto um mecanismo se encarrega de manter o *grip* do robô sempre vertical, e não existe o grau de liberdade de rotação do pulso.

Para robôs que executam tarefas de movimentação de peças, esta configuração permite economia na construção do robô, com uma agilidade semelhante à de um robô Quatro-barras.

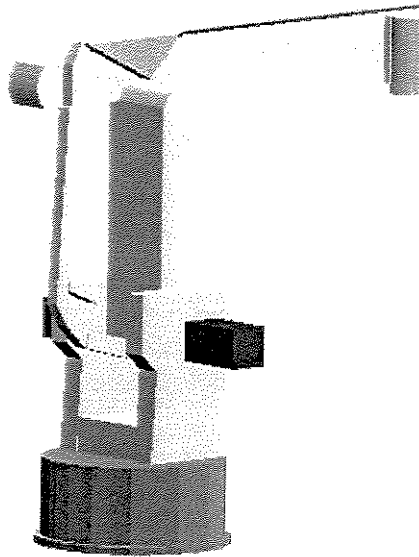


Figura A IV.5: Estrutura cinemática de um robô para paletização.
Semelhante ao robô quatro barras, este possui um mecanismo de transmissão de movimentos que comanda o pulso para ficar sempre vertical, dispensando o motor para acionar esta junta. Logo, é um robô com 4 juntas e 3 GDL.
(Imagem gerada pelo software Virtual Workspace 4.0)

Macro SetUp4

%modelo de robô pick-and-place, com garra sempre vertical

SetUp

joints = 6

% ExtDHtable = ExpTheta	Expd	a	alfa	aWid	dWid
extdhtable1 = 'servo1';	5;	0;	90;	3;	0
extdhtable2 = 'servo2+90';	3;	9;	0;	2,8;	2,2
extdhtable3 = 'servo3-servo2-90';	-3;	0;	90;	2;	2
extdhtable4 = 0;	8;	0;	-90;	1;	1,2
extdhtable5 = '-servo3';	0;	0;	90;	0;	1
extdhtable6 = 'servo6';	1,2;	0;	0;	0,5;	0

%GripOr = GripOrYaw GripOrPitch GripOrRoll

gripor = 0; 0; 0;

GripLen = 3

%GripWidth= ExpWidth KWidth Width0

```
GripWidth = 0;      0,05;  0
Gripthick = 0,5
end macro
```

Neste modelo, nota-se que os servos 4 e 5 estão desacoplados, ou seja, não comandam nenhuma das juntas. O modelo continua a ter 6 juntas, embora possua apenas 4 graus de liberdade. O modelo é quase igual ao do paralelogramo, porém foi subtraída a influência da junta 3 sobre a junta 5.

A IV.4.4 Robô Scara

Este tipo de robô é bastante usado em operações de montagem e paletização, mantendo sua garra sempre vertical. A configuração com eixos de rotação das juntas sempre verticais faz com que o peso próprio do robô não gere força resultante sobre os servos.

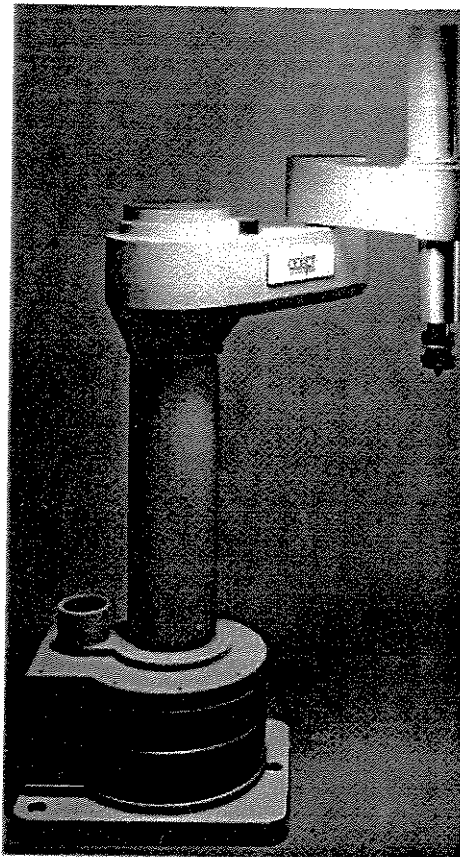


Figura A IV.6: Robô com configuração Scara

Este tipo de robô é composto por três juntas rotativas com eixo vertical e uma junta de translação vertical. Spong (1989), p.121

O modelo correspondente é:

Macro SetUpScara

% modelo do Scara - tabela usando expressões

config

SetCaption 1 eq "Rot1" 2 eq "Rot2" 3 eq "Elev" 4 eq "Flange" 5 eq
"Grip" 6 eq "" 7 eq ""

joints = 4

%ExtDHtable = ExpTh	Expd	a	alfa	aWid	dWid
extdhtable1 = 'servo1';	10;	6;	0;	2,5;	2,5;
extdhtable2 = 'servo2';	2;	4,5;	180;	1,5;	1,5;
extdhtable3 = 0	;	'servo3/48-5';	0;	0;	0,5; 1,5;

```

extdhtable4 = 'servo4';          10;   0;   0; 0,5; 0,5;
%GripOr = GripOrYaw GripOrPitch GripOrRoll
gripor = 0;          0;          0;
GripLen = 3
%GripWidth= ExpWidth KWidth Width0
GripWidth = 'servo5/120+2'
Gripthick = 0,5
end macro

```

O modelo resulta:

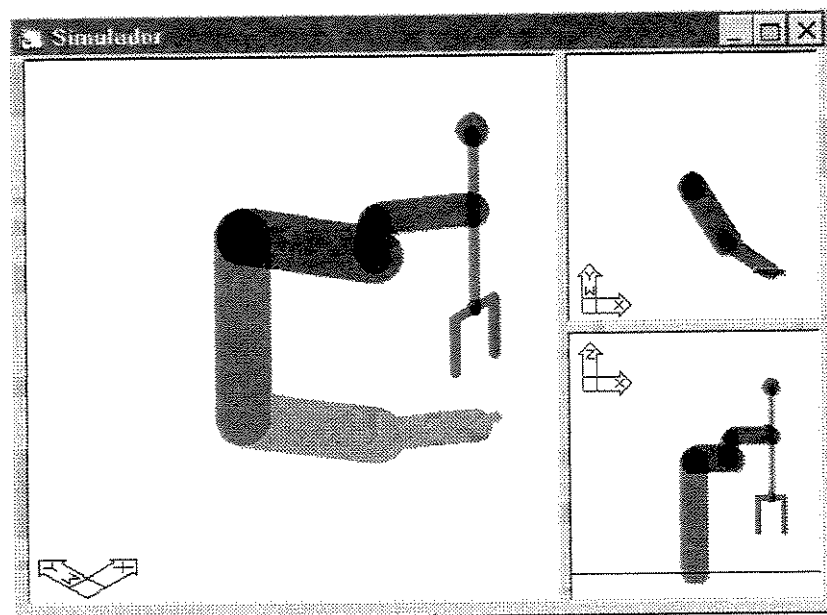


Figura A IV.7: Modelo do robô Scara no simulador.

A IV.4.5 Teste do algoritmo de orientação inversa

Para verificar se a rotina de cálculo da orientação inversa da garra estava operando adequadamente, foi criado um modelo de robô no qual a primeira, segunda e terceira juntas realizam respectivamente um movimento de *Roll*, *Pitch* e *Yaw* na garra do robô. Assim, atribuindo-se valores às juntas, obtém-se uma transformada RPY, verificando se os valores dos ângulos de orientação encontrados correspondem aos valores dos respectivos ângulos das juntas.

O modelo criado foi:

Ex. Teste do RPY

Macro RPY

Config

SetCaption 1 eq "roll" 2 eq "pitch" 3 eq "yaw"

joints = 3 %número de GDL

% ExtDHTable = ExpTheta Expd a alfa aWid dWid

extdhtable1 = 'servo1'; 5; 0; -90; 1; 1

extdhtable2 = 'servo2+90'; 5; 0; 90; 1; 1

extdhtable3 = 'servo3'; 5; 0; 0; 1; 1

%GripOr = GripOrYaw GripOrPitch GripOrRoll

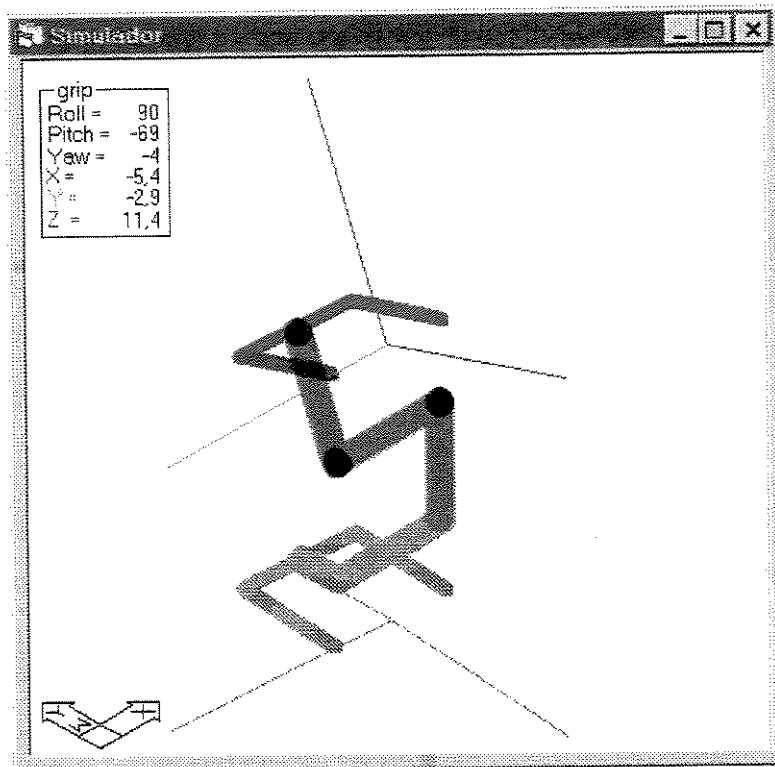
gripor = 0; -90; 0;

GripLen = 5

GripWidth = 5

Gripthick = 0,5

end macro



*Figura A IV.8: Modelo criado para verificar o algoritmo de orientação inversa.
Cada uma das juntas modifica respectivamente o Roll, Pitch e Yaw da garra do modelo.*

Anexo V

Estudo de caso: Construção, modelagem e testes com um manipulador robótico experimental

O robô apresentado se baseia em tecnologia usada para construção de aviões de radiocontrole, usando materiais relativamente sofisticados, de baixo custo (pois são produzidos em larga escala para atender ao mercado de *hobbistas*) e fáceis de serem trabalhados, sem requerer ferramentas especiais.

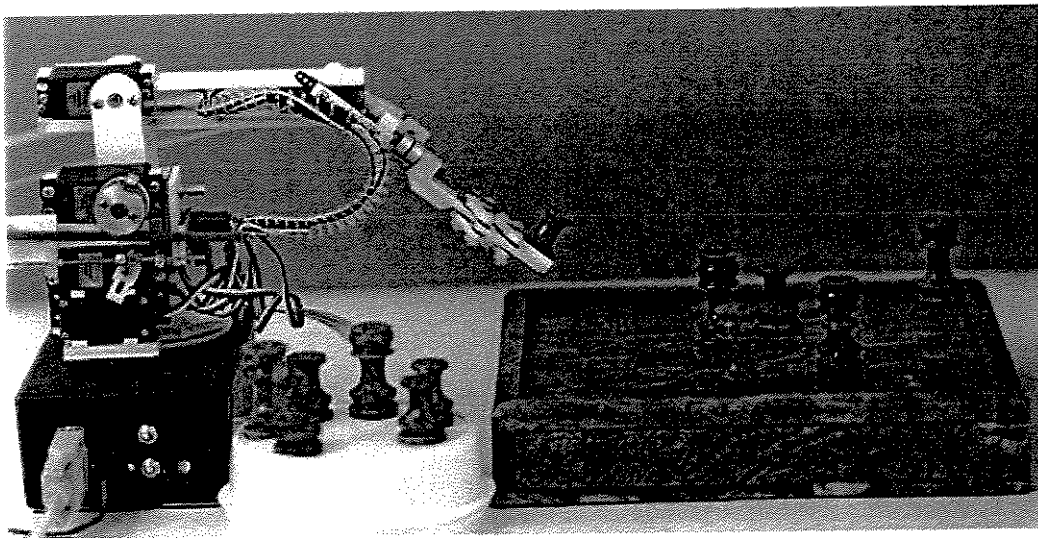


Figura A V.1: O robô desenvolvido

A seguir estão documentados alguns aspectos do manipulador robótico implementado, que

podem servir como referência a projetos semelhantes. Um robô que tenha um projeto bem elaborado irá funcionar de modo bastante satisfatório, podendo ser agregado como material didático de um laboratório.

A V.1 A construção das juntas

As juntas são a parte mais delicada da construção. Para preservar o peso do robô, e como não se trata de um produto feito em larga escala, é preciso evitar especificar peças de construção complexa.

As soluções adotadas neste robô foram classificadas de acordo com sua aplicação, podendo ser aplicadas como for melhor conveniente dependendo das características do robô.

Para junta rotativa com extremidade livre:

Neste caso, a junta é simplesmente apoiada, devendo suportar forças normais e torque. Este tipo de junta é usado, por exemplo, quando o eixo de rotação é longitudinal ao comprimento do elo, e portanto, apoiada em um ponto e com a outra extremidade necessariamente livre.

Foram criadas duas soluções para construção deste tipo de junta: A construção de um rolamento axial, como usado na base e rotação do pulso e o uso de eixos concêntricos, como usado para rotação da garra.

Em cada uma das soluções existe uma dimensão que é expandida, para suportar o momento. O rolamento axial tem um diâmetro grande foi usada na base enquanto a junta com eixos concêntricos, possui um comprimento grande e diâmetro reduzido foi usada na garra.

Para junta rotativa bi-apoiada:

Em situações nas quais é possível apoiar o elo seguinte em dois pontos que passam pelo

eixo de rotação, é preferível implementar uma junta bi-apoiada, eliminando a necessidade da junta suportar momentos, simplificando sua construção.

Nestes casos, uma extremidade foi apoiada diretamente sobre o próprio eixo de saída do servo e outra, apoiada sobre um mancal rotativo. O eixo de saída dos servos é desenhado para suportar esforços radiais.

Como mancal pode ser usado um micro-rolamento, como foi usado para a mão do robô ou então um mancal de deslizamento.

Para juntas prismáticas:

Juntas prismáticas podem ser criadas utilizando guias de deslizamento, rolamentos lineares, mancais de esferas recirculantes ou micro-rolamentos rolando sobre um trilho. A última opção parece ser a mais apropriada para a construção de dispositivos de pequeno porte.

A V.2 Sistemas de transferência de movimento:

Basicamente por três motivos podem tornar necessária a transmissão de movimento de um atuador: Por falta de espaço, para deslocar o servo para outro ponto onde sua massa cause menor influência, ou também para ampliar sua amplitude de movimentos.

Um sistema possível é o uso de cabos de comando de pequeno tamanho. Este é o sistema mais comumente utilizado em aeromodelismo, de modo que existem sistemas já prontos disponíveis no mercado. Porém existem problemas relacionados a atrito e flexibilidade do cabo.

O sistema de montagem tradicional é tipo biela-manivela, o que torna o movimento não-linear. Para solucionar este problema, foi desenvolvido um sistema de transmissão de movimentos linear, utilizando uma haste semi-rígida.

Outra possibilidade é o uso de correias dentadas, como as utilizadas em impressoras (e que poderiam ser adquiridas no mercado). Isto permite transmitir rotação de um eixo a outro eixo paralelo, sem limite de amplitude de rotação. Se forem utilizadas polias com diâmetro diferente, é possível aumentar a amplitude de movimento do servo, o que em geral é desejável.

Também se pode usar barras de comando (como em um robô com 4 barras), porém neste caso, a rotação está restrita menos de 180 graus.

Outros sistemas comumente utilizados em engenharia mecânica também podem ser usados, porém a necessidade de baixo peso e tamanho reduzido acaba reduzindo muito as opções realmente factíveis.

A V.3 A base do robô

A solução adotada foi a construção de rolamento axial de grande diâmetro, usando plástico e uma placa de fórmica para criar a pista para as esferas. A estrutura é pré-tensionada, eliminando as folgas. Neste caso, a dimensão longitudinal da junta é relativamente pequena, mas seu diâmetro é grande e o peso também, devido ao seu tamanho.

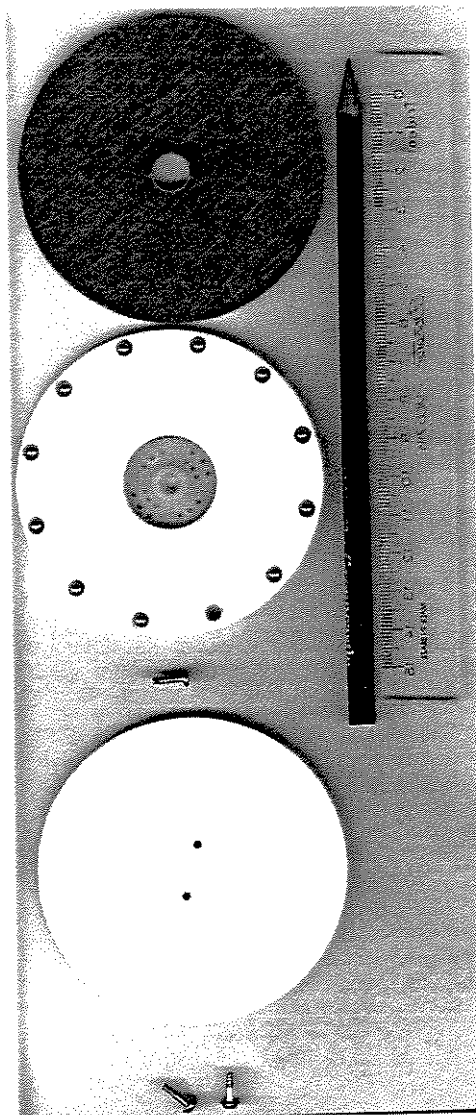


Figura A V.2: As camadas que compõem o rolamento axial.
Dois discos de fórmica e uma gaiola que guia esferas de aço. Os discos devem ser apoiados sobre chapas rígidas, e sua função não é estrutural, e sim formar uma pista plana e resistente para as esferas.

A V.4 Garra ou elemento terminal

O desenvolvimento dos elementos terminais está associado à proposição de quais tipos de tarefas estes robôs podem realizar. Devido à sua construção delicada e de pequeno porte, estes robôs não devem ser submetidos a esforços, e esta nem é sua finalidade.

No LAR temos usado estes robôs para transportar peças de LEGO, podendo também ser usados blocos de madeira, peças de alumínio, ou objetos feitos em espuma rígida. É recomendável implementar um sistema de fechamento da garra comandado por um cabo de aço, por duas razões:

Isto reduz consideravelmente a massa da garra, reduzindo o esforço nos servos, tornando o funcionamento do robô mais rápido e preciso e reduz o comprimento da garra, o que é recomendável para melhorar a destreza do robô.

A solução mais indicada para fazer a rotação da garra é o uso de um eixo longitudinal ôco, com um eixo externo concêntrico. Neste caso, o peso e o diâmetro são extremamente reduzidos, porém o comprimento da junta é grande.

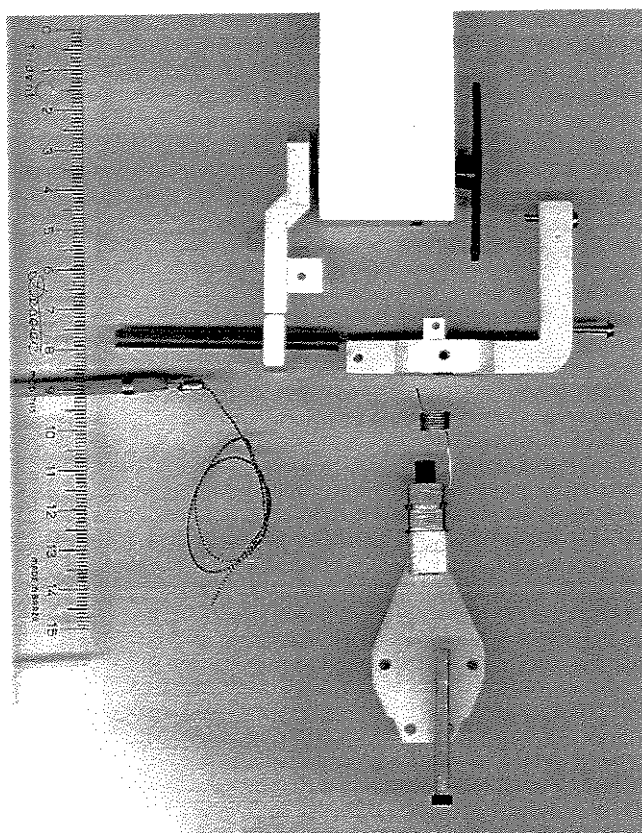


Figura A V.3: Partes do pulso e rotação da garra.

O pulso é preso ao chifre do servo de um lado e a um rolamento do outro lado. À esquerda, o cabo de comando para a rotação da garra, é acoplado a um fio flexível. O eixo ôco está na parte inferior da imagem, e será montado solidariamente ao pulso. Uma mola helicoidal de torção faz o cabo sempre trabalhar tracionado.

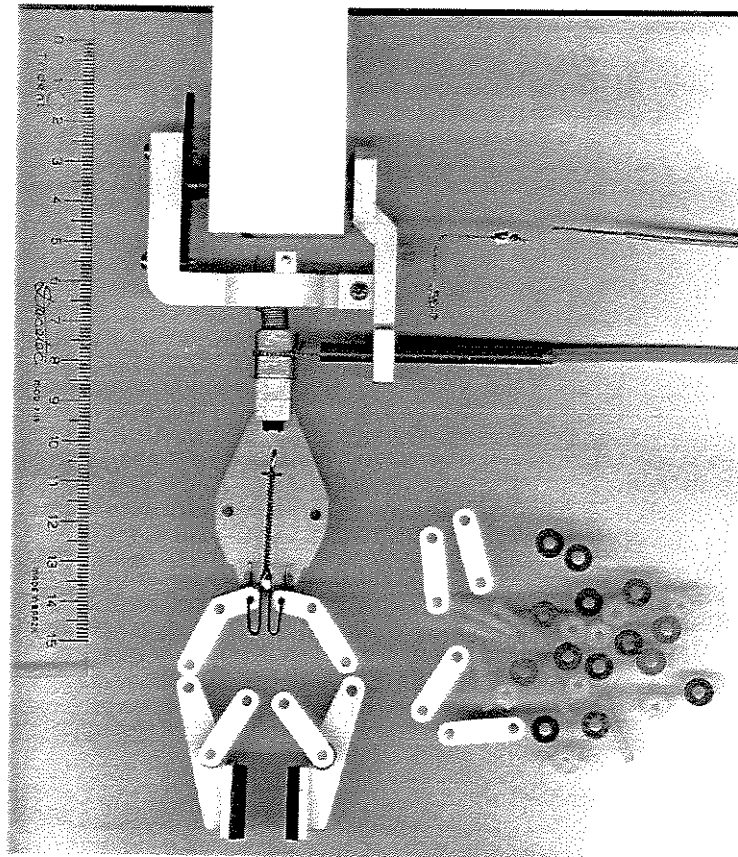


Figura A V.4: Partes que compõem a garra.

Uma vez montado o sistema de rotação da garra, é montado o sistema de abertura e fechamento. É usado outro cabo de aço ligado à garra por um fio que passa por dentro do eixo de rotação.

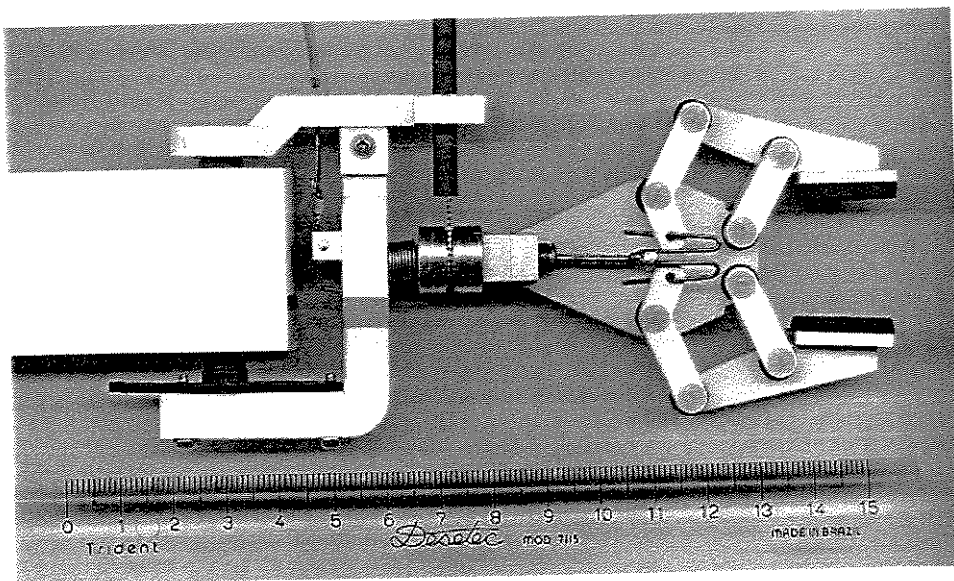


Figura A V.5: o sistema de rotação e fechamento da garra montado.
Dois cabos de comando ligados a servos localizados na base do robô transmitem o movimento dos servos. Uma pequena mola de compressão mantém a garra naturalmente aberta, fazendo o cabo trabalhar somente a tração.

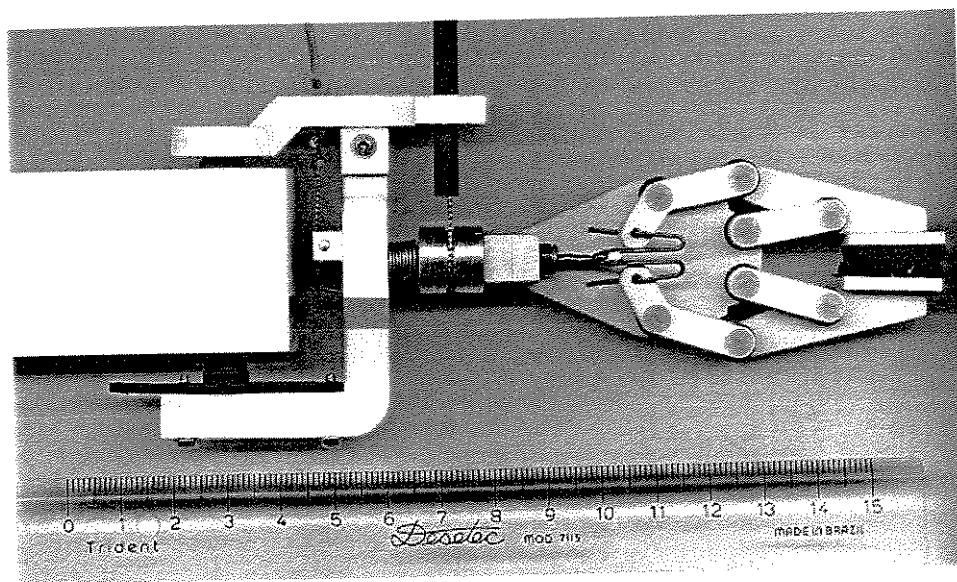


Figura A V.6: a garra fechada.
Note-se que seu comprimento varia dependendo da abertura, o que deve ser levado em consideração na criação de um modelo do robô.

A construção final ainda resultou em uma garra exageradamente longa, em grande parte

porque o mecanismo de fechamento é longo. Seria recomendável re-projetar a garra com um sistema de fechamento transversal, semelhante ao utilizado no Robix.

A V.5 Especificações técnicas dos servos utilizados

Servos utilizados nas juntas 1, 2 (acionamento duplo), 3, 6 (acionamento à distância) e garra (acionamento à distância):

Tabela A V.1: Especificações técnicas dos servos utilizados nas juntas 1, 2, 3, 6 (acionamento a distância) e garra (acionamento a distância).

Ficha técnica que acompanha o produto, (1994)

Fabricante, modelo	Hobbico CS-51 Stanard Servo
Plug	Futaba J
Torque	42 [oz.in]
Tempo de resposta	0.19 sec.@60°
Tensão nominal	4,8 [V]
Pulso do sinal	Pulso positivo, posição neutra a 1520 μ s
Dimensões	41x20x36 [mm]
Massa	1.75 [oz]

Tabela A V.2: Especificações técnicas dos servos utilizados nas juntas 4, e 5.

Ficha técnica que acompanha o produto, (1998)

Fabricante, modelo	Futaba FP-S3101 Micro Servo
Plug	Futaba J
Torque	34,7 [oz.in]
Tempo de resposta	0.18 sec.@60°
Tensão nominal	4,8 [V]
Pulso do sinal	Pulso positivo, posição neutra a 1520 μ s
Dimensões	41x20x36 [mm]
Massa	0,6 [oz]

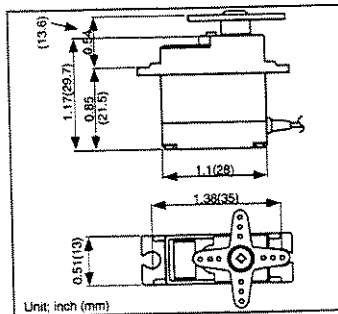


Figura A V.7: dimensões do Futaba S3101.
Especificações técnicas que acompanham o produto (1998).

A V.6 Modelagem do robô

A configuração do robô em questão segue a de um robô Puma, com as seis juntas rotacionais. Este tipo de configuração é bastante funcional e largamente utilizado na indústria.

A seguir está descrito todo o procedimento usado para criar um modelo cinemático do manipulador robótico que foi utilizado no desenvolvimento do aplicativo.

Inicialmente, são determinados os parâmetros de Denavit&Hartenberg que descrevem a cinemática do manipulador (tipo Puma).

Em seguida, a partir da matriz de parâmetros de Denavit&Hartenberg, é criada a tabela de Denavit&Hartenberg estendida, contendo parâmetros adicionais utilizados pelo aplicativo. Também são determinados outros parâmetros adicionais, como orientação do *grip*, comprimento, abertura etc.

Por último, é feita a calibragem dos servos, determinando as constantes que convertem os valores comandados ao coprocessador em ângulos dos servos. Este procedimento é realizado a partir de dois valores de posição de servo associados a ângulos de juntas medidos experimentalmente.

Determinação dos parâmetros cinemáticos do robô:

Tabela A V.3: Parâmetros cinemáticos de um robô tipo Puma

Esta é uma das possíveis modelagens.

I	alfa	a	d	teta	nome
1	0	0	0	θ_1	cintura
2	-90	0	0	θ_2	ombro
3	0	a2	d3	θ_3	cotovelo
4	-90	a3	d4	θ_4	mão
5	90	0	0	θ_5	pulso
6	-90	0	0	θ_6	flange

A partir dos parâmetros cinemáticos, obtém-se as sucessivas transformações de coordenadas até chegar à extremidade da garra do robô.

UNICAMP
BIBLIOTECA CENTRAL
SECÃO CIRCULANTE