ECAI

Universidade Federal de Santa Catarina

Centro Tecnológico

Curso de Engenharia de Controle e Automação Industrial

UFSC

Desenvolvimento de um Módulo Gerador de Trajetórias para um Robô do Tipo SCARA

Monografia submetida à Universidade Federal de Santa Catarina como requisito para a aprovação da disciplina:

DAS 5511: Projeto de Fim de Curso

Clóvis Fernandes Júnior

Florianópolis,30 de outubro de 1998

Desenvolvimento de um Módulo Gerador de Trajetórias Aplicado à Robótica

Clóvis Fernandes Júnior

Esta monografia foi julgada no contexto da disciplina

DAS 5511: Projeto de Fim de Curso

e aprovada na sua forma final pelo

Curso de Engenharia de Controle e Automação Industrial

Banca Examinadora:

Prof. Raul Guenther
Orientador

Prof. Augusto Humberto Bruciapaglia
Responsável pela disciplina e Coordenador do Curso

Prof. Edson de Pieri, Avaliador

Adrián Fritz, Debatedor

Alexandre Emanuel Camargo, Debatedor

Agradecimentos

Ao final deste trabalho, gostaria de agradecer a todos que de alguma forma contribuíram para a conclusão deste Projeto de Fim de Curso, e em especial;

Ao Prof. Raul Guenther, meu orientador, pelo apoio prestado para o desenvolvimento deste trabalho;

Ao Eng. Mecânico Lucas Weihmann, meu co-orientadore e companheiro de trabalho, pela grande ajuda prestada, sem a qual não teria concluído este trabalho;

Ao Eng. Elétrico Emerson Raposo, na ocasião responsável pelo laboratório, pela paciência e prestatividade;

A todos os amigos, que me acompanharam ao longo destes anos de curso, sempre presentes nos momentos de descontração e alegria;

A minha família, que até hoje têm sido o alicerce que sustenta minha vida, onde sempre encontro o apoio emocional necessário para enfrentar os problemas do dia a dia.

Resumo

Este trabalho trata do desenvolvimento de um gerador de trajetórias e visa substituir o atualmente utilizado na movimentação do robô Inter do Laboratório de Robótica da Universidade Federal de Santa Catarina. O novo gerador é capaz de representar trajetórias curvas no espaço através de funções suaves, interpoladas a partir de uma sequência de pontos previamente definidos pelo usuário. Os valores de posição, velocidade e aceleração angular que representam a trajetória são gerados através de uma programação *off-line* e enviados ao sistema de controle do manipulador para efetivar o movimento.

Abstract

The Trajectory Planning Problem for Robots is the main focus of this work. It intends to replace the currenty trajectory planner of a Scara Robot called "Inter" for na alternative solution.

The proposed trajectory planner does the planning task by connecting points that are defined by the user. The desired trajectory is represented in the joints space by smooth functions of position, velocity and aceleration. These functions generated a sequence of time-based "control set points" that are used by the manipulator's control system to move the robot's arm.

Sumário

1. Introdução

A principal função de um robô industrial é a de posicionar seu efetuador-final, seu elemento ativo responsável pela realização de uma tarefa, tal como pintura, execução de um cordão de solda, manipulação de peças, entre outras. Este posicionamento é realizado mediante a movimentação coordenada de todas as suas juntas, que são acionadas por seus respectivos atuadores. Por sua vez, os atuadores seguem sinais de referência que lhes são enviados pelo sistema de controle do robô. Estes sinais são gerados pelo gerador de trajetórias que utilizam os dados do caminho que foram definidos pelo usuário.

Para evitar confusão entre termos usados como sinônimos, deve ser ressaltada a diferença entre caminho e trajetória. Um caminho é formado por pontos que o manipulador deve seguir para executar o movimento. O caminho é, desta forma, a descrição geométrica do movimento. A trajetória é um caminho no qual a variável tempo é adicionada, por exemplo, em termos da velocidade e da aceleração em cada ponto.

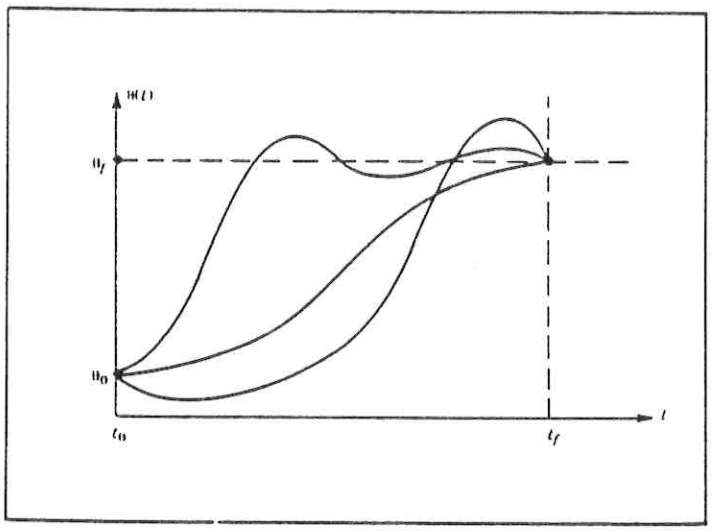


Figura 1.1 - Formas de caminhos possíveis para uma junta simples

A princípio, o gerador de trajetórias pode ser visto como um bloco funcional, como ilustra a figura 1.2. As suas entradas são a descrição geométrica do caminho, especificações temporais e os parâmetros impostos pela dinâmica do manipulador.

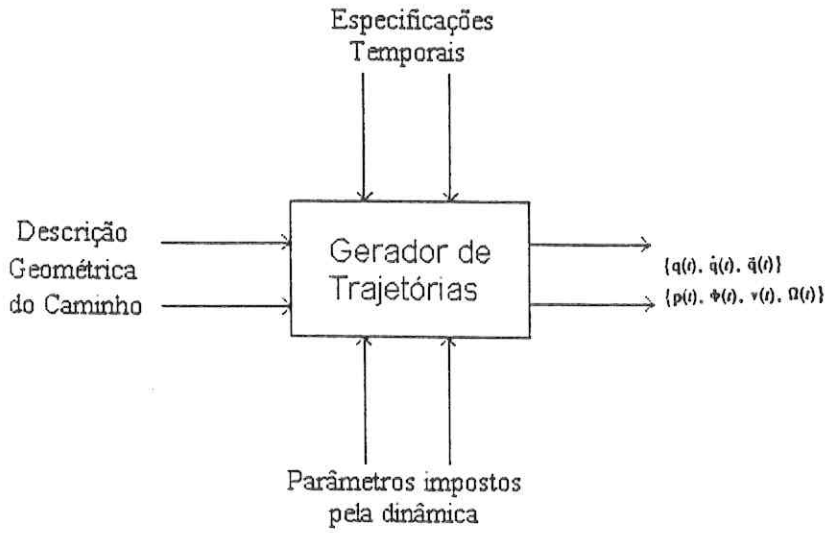


Figura 1.2 - Diagrama de blocos do Planejador de Trajetórias

1.1 Enfoque Proposto

O objeto de estudo deste projeto é o robô Inter, do tipo SCARA, que pertence ao Laboratório de Robótica. O objetivo é desenvolver um novo módulo gerador de trajetórias em substituição ao atual. Este gerador possibilita somente o movimento ponto-a-ponto, ou seja, movimento em que apenas o ponto inicial e o ponto final podem ser especificados. Além disso, o uso deste gerador provoca uma variação brusca da aceleração angular nas juntas do manipulador que faz com que os atuadores utilizem torques excessivos nas juntas do robô, o que pode danificar a sua estrutura. Um outro problema refere-se a determinação dos valores de velocidades. A transformação destes valores do espaço cartesiano para valores equivalentes no espaço de juntas é efetuado sem qualquer verificação e pode gerar valores que excedem os limites de fabricação estabelecidos pelo fabricante.

O objetivo principal deste trabalho é desenvolver um módulo gerador de trajetórias que permita o robô percorrer caminhos curvilíneos, indo de um ponto a outro em seu espaço de trabalho, passando por pontos intermediários pré-determinados sem paradas. Deseja-se representar a trajetória através de funções suaves de posição, velocidade e aceleração. Estas funções garantem a continuidade do movimento e evitam o problema de variação brusca nos valores de aceleração. Além disto, é necessário que os valores de velocidade e aceleração angular sejam verificados para que se possa assegurar que os mesmos não ultrapassam os seus limites máximos.

O usuário deve especificar a seqüência de pontos inicial, final e intermediários que vão representar o caminho a ser seguido pelo manipulador. Além destes, deve ser também especificado o tempo desejado para que o movimento seja executado e possivelmente, a velocidade e aceleração desejada em cada ponto do caminho. O planejador deve gerar uma seqüência temporal de valores de posição, velocidade e aceleração angular e enviá-los ao sistema de controle do robô para efetivar o

movimento. Antes porém, deve verificar se estes valores estão dentro dos limites estabelecidos. Caso não estejam, ou o sistema interrompe o movimento informando o problema ao usuário, ou resolve o mesmo utilizando alguma lógica interna.

Este módulo será implementado com a intenção de substituir o atualmente utilizado, proporcionando maior flexibilidade na execução de tarefas que envolvam a movimentação do manipulador e contribuindo para evitar o desgaste prematuro de suas juntas.

No capítulo dois são apresentados em maiores detalhes o desenvolvimento do gerador de trajetórias e os tópicos de estudo necessários à fundamentação deste trabalho. A seção 2.1 apresenta conceitos vinculados à robótica e na sequência é feita uma descrição geral do robô Inter (seção 2.2). Em seguida são apresentados técnicas mais freqüentemente utilizadas no planejamento de trajetórias (seção 2.3). A seção 2.4 apresentada um estudo do linguagem de programação que será empregada na implementação do módulo gerador, e após é feita uma discussão sobre a solução proposta (seção 2.5). No capítulo três o projeto é restruturado de modo a garantir que as exigências iniciais sejam cumpridas. É proposta uma solução em duas partes, onde a primeira é a geração *off-line* dos parâmetros da trajetória utilizando funções do Matlab (seção 3.1), e a outra parte é o desenvolvimento do um software capaz de ler, converter e enviar estes valores ao sistema de controle do robô (seção 3.2). Finalmente, nos capítulos quatro e cinco são apresentados os resultados obtidos e a conclusão sobre o projeto desenvolvido, respectivamente.

2. Desenvolvimento do Gerador de Trajetórias e Tópicos de Estudo

Neste capítulo são apresentados alguns conceitos básicos de robótica, vinculados à cinemática direta e inversa. Estes conceitos são de fundamental importância para o posterior estudo e definição dos métodos e técnicas utilizadas para a geração de trajetórias propriamente dita. Na seqüência é feita a descrição das características gerais do robô SCARA utilizado neste projeto. São vistos alguns aspectos de sua estrutura mecânica, espaço de trabalho e armário de controle.

Finalmente, após definir a técnica de planejamento a ser empregada, é feito um estudo da linguagem XOberon, que é utilizada na programação do robô. Esta linguagem foi desenvolvida na universidade de Zurique, Suíça, para aplicações na área de robótica.

2.1 Conceitos de Robótica

Antes de iniciar o estudo da geração de trajetórias, é necessária a fixação de conceitos básicos que compõem um sistema robótico.

Pode-se dizer que um sistema robótico é formado por um conjunto de componentes que objetivam a execução de uma tarefa, definida por um usuário, e que é composto, basicamente, pelos seguintes elementos:

1) Manipulador:

Mecanismo concebido para a execução de tarefas, através do posicionamento de seu efetuador-final. É composto por elos e juntas. Estas são elementos construtivos que fazem a conexão entre dois elos consecutivos, permitindo o movimento relativo entre os mesmos. Elos são elementos rígidos na estrutura do manipulador, como pode ser observado na figura 2.1.

2) Acionamentos:

Compostos por atuadores, que são elementos que imprimem movimento ao manipulador através da variação em cada junta.

3) Sensores:

São dispositivos que avaliam grandezas físicas, associando-as a valores de fácil tratamento, geralmente valores elétricos. Podem ser classificados em sensores de posição, de velocidade, entre outros.

4) Controle:

Sistema que controla todo o funcionamento do robô. Responsável pelo tratamento das informações a respeito da tarefa a ser realizada, através dos sinais de referência para o atuador que executa a movimentação das juntas do manipulador.

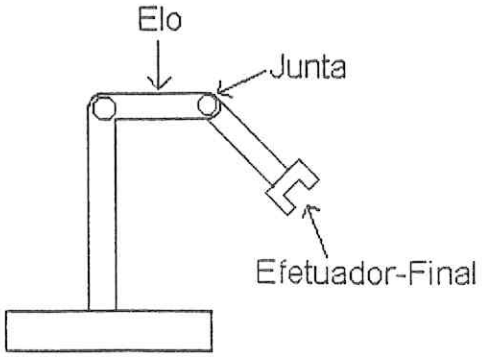


Figura 2.1 - Esquema estrutural de um manipulador

O sistema de controle do manipulador recebe as informações da tarefa a ser executada de forma codificada e verifica a posição atual de cada uma das juntas do robô, comparando-as com a posição desejada, conforme solicitado pelo programa. Se existir algum erro de posição, o atuador da junta correspondente é excitado pelo sistema de controle, de modo a reduzir o erro até, teoricamente, o anular. Após o posicionamento do efetuador-final, este executa a tarefa que foi definida pelo usuário, que pode ser, por exemplo, uma tarefa de movimentação de peças, pintura, solda, etc (Figura 2.2)

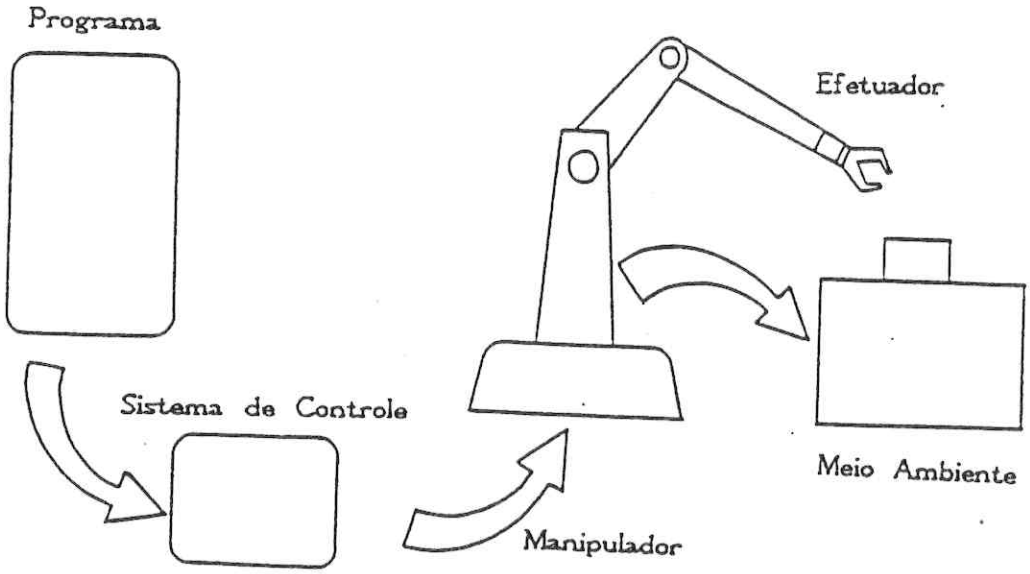


Figura 2.2 - Esquema geral de trabalho de um robô

Um tópico de fundamental importância na geração de trajetórias é o que trata do estudo geométrico do braço do robô, onde está-se interessado na posição e

orientação relativa do efetuador-final em relação à base do manipulador. Este tópico e outros como a cinemática inversa e direta são tratados na próxima seção.

2.1.1 A Geometria de Manipuladores

Um manipulador pode ser considerado como uma cadeia de corpos rígidos (fig. 2.3) na estrutura cinemática. Desta forma, uma ligação com estrutura serial, ou malha aberta, é chamada cinemática aberta, que constituem a maioria dos robôs.

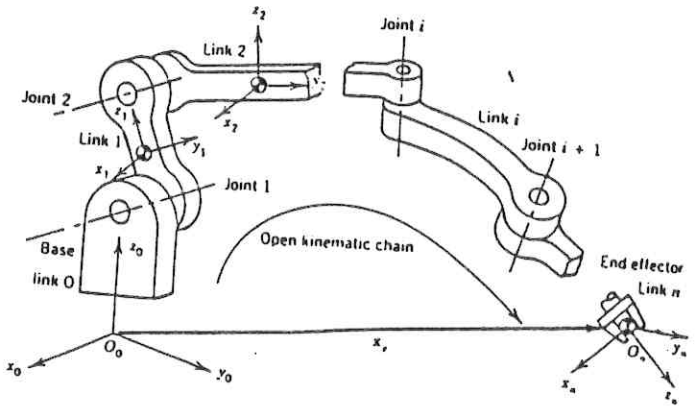


Figura 2.3 - Manipulador visto como uma cadeia de corpos rígidos

O problema da cinemática de manipuladores consiste basicamente em um problema geométrico, onde a intenção é definir a posição relativa entre as várias ligações que formam a estrutura mecânica do robô e obter a posição e orientação do extremo do último elo, que é o efetuador-final.

Existem diversos enfoques que podem ser utilizados para o tratamento deste problema [3], mas, de um modo geral, predomina o uso das matrizes de transformação homogênea e a notação de Denavit-Hartenberg. Esta notação introduz um método sistemático para descrever a relação entre duas ligações adjacentes de uma cadeia cinemática aberta.

Uma matriz de transformação homogênea pode ser representada por uma matriz 4x4, com a estrutura:

$$A = \begin{bmatrix} R & X_0 \\ \Theta^T & 1 \end{bmatrix}$$

onde R é uma matriz 3x3, que corresponde à transformação de rotação entre as duas juntas analisadas, e X_0 é o vetor posição da junta i+1 em relação à junta i. Θ^T é o vetor zero e 1 é o escalar 1. Para estudar a posição relativa entre duas juntas adjacentes de acordo com a notação de Denavit-Hartenberg (D-H), é necessário definir

o conceito de quatro parâmetros escalares, que representam dois ângulos e duas distancias (fig. 2.4):

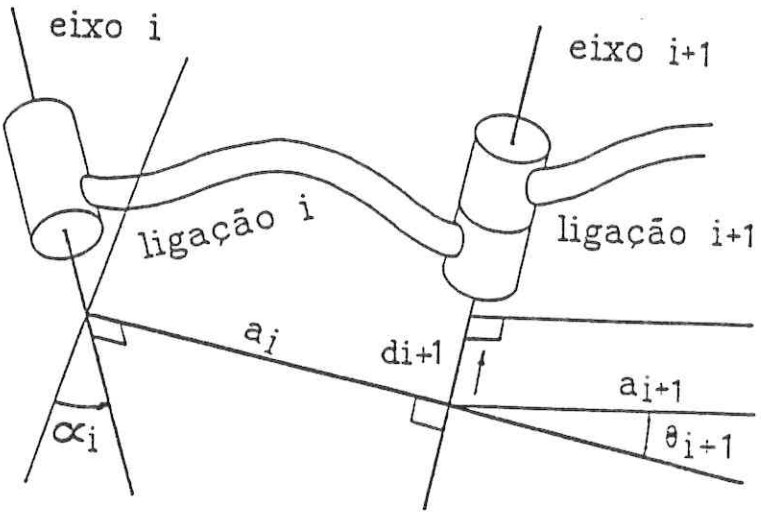


Figura 2.4 - Parâmetros de Denavit-Hartenberg

a_i: Comprimento da ligação i, definido como a distância entre os eixos das juntas do elo i;

d_{i+1} Excentricidade ou *off-set* da junta, medida entre as normais comuns para o elo i e i+1;

α_i: Ângulo de torção do elo i, medido como o ângulo entre o eixo i e o eixo i+1, sobre a reta definida por a_i;

θ_{i+1}: Ângulo da junta i+1, medido pelo prolongamento de a_i e a_{i+1}, em torno do eixo i+1.

Os parâmetros a_i e α_i são parâmetros constantes determinados pela geometria do elo. Os parâmetros d_i e θ_i são parâmetros da junta e ao menos um deles é variável com o movimento da junta, dependendo se for uma junta prismática ou de revolução. Utilizando a notação de D-H, encontra-se a seguinte matriz:

$$A_i^{i-1} = \begin{bmatrix} C\theta_i & -S\theta_i C\alpha_i & S\theta_i S\alpha_i & a_i C\theta_i \\ S\theta_i & C\theta_i C\alpha_i & -C\theta_i S\alpha_i & a_i S\theta_i \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

onde $C\theta_i$, $C\alpha_i$, $S\theta_i$ e $S\alpha_i$ são os cosenos e senos dos ângulos correspondentes,

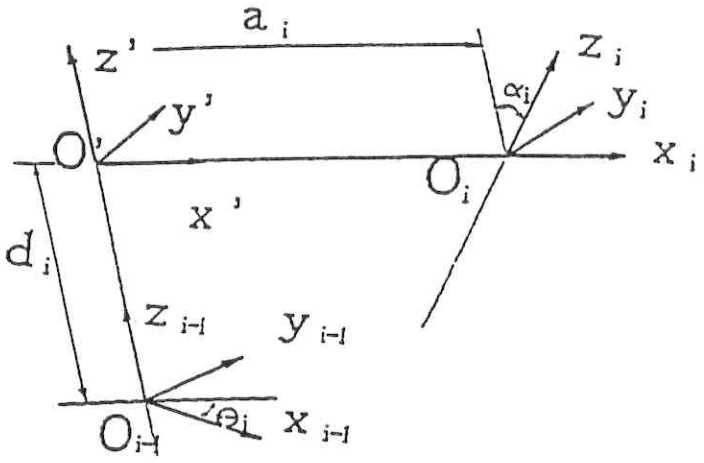


Figura 2.5 - Sistemas de coordenadas em duas juntas

Esta matriz representa a transformação de um vetor, descrito no sistema de coordenadas i, para o sistema de coordenadas i-1. Ou seja, sendo X^i e X^{i-1} a representação do vetor, respectivamente em i e i-1, temos:

$$X^{i-1} = A_i^{i-1} A^i$$

No caso geral de um manipulador de cadeia cinemática aberta, com *n* ligações, o objetivo é de determinar a matriz de transformação desde o extremo do efetuador-final até a base do manipulador. Assim, no caso de juntas prismáticas, d_i é variável, ficando a_i, α_i e θ_i constantes e para as juntas de rotação, θ_i é a variável, com a_i, d_i e α_i constantes. Deste modo, a matriz A_i^{i-1} é função ou de d_i ou de θ_i, dependendo do tipo de junta que é utilizada. Então a transformação do vetor X^i será dada por:

$$X^{i-1} = A_i^{i-1}(q_i) X^i$$

onde q_i é uma coordenada de junta generalizada.

Como o manipulador é formado por um encadeamento seqüencial de seus elos, para definir a posição de um ponto do efetuador-final (elemento ativo do manipulador), dada pelo sistema de coordenadas deste em relação ao sistema de coordenadas da base do manipulador, basta aplicar seqüencialmente as sucessivas matrizes de transformação de cada uma das ligações, desde o efetuador-final até a base do manipulador. Assim, sendo X^n o vetor posição de um ponto em relação ao sistema de coordenadas da ligação n, o vetor posição do mesmo ponto, X^0, agora em relação ao sistema de coordenadas da base, é obtido como:

$$X^0 = A_1^0(q_1)X^1$$

$$X^1 = A_2^1(q_2)X^2$$

$$\vdots$$

$$X^{n-1} = A_n^{n-1}(q_n)X^n$$

ou seja:

$$X^0 = A_1^0(q_1)A_2^1(q_2)A_3^2(q_3)A_4^3(q_4)\ldots A_n^{n-1}(q_n)X^n$$

onde o produto de matrizes 4x4 pode ser substituído por uma única matriz T:

$$X^0 = T(q_1, q_2, q_3, \ldots q_n)X^n$$

Desta forma, a matriz T (q_1, q_2, ..., q_n) contém todas as informações a respeito da posição e da orientação do efetuador, em relação as sistema de coordenadas da base, na mesma estrutura da matriz A, onde agora X_0 fornece a posição da origem do sistema de coordenadas do efetuador e R é a matriz de rotação entre o sistema da base e o do efetuador [3,8].

A matriz com os parâmetros de D-H para o caso do robô Inter é apresentada na tabela 2.1:

No. Elo	α_i	a_i	d_i	θ_i
0	*0*	*250*	*665*	θ_0
1	*180*	*250*	*0*	θ_1
2	*0*	*0*	d_2	*0*
3	*0*	*0*	*0*	θ_3

Tabela 2.1 - Os parâmetros de D-H do robô Inter

E sua correspondente matriz de Transformação T_3^0 do sistema de coordenadas O_3 para O_0 é [11]:

$$T_3^0 = \begin{bmatrix} C\theta_3C\theta_{01} + S\theta_3S\theta_{01} & -S\theta_3C_{01} + C\theta_3S\theta_{01} & 0 & 0{,}25(C\theta_{01} + C\theta_0) \\ C\theta_3S\theta_{01} - S\theta_3C\theta_{01} & -S\theta_3S_{01} - C\theta_3C\theta_{01} & 0 & 0{,}25(S\theta_{01} + S\theta_0) \\ 0 & 0 & -1 & -d_2 + 0{,}665 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

onde: $C\theta_3 = \cos(\theta_3)$ e $C_{01} = \cos(\theta_0 + \theta_1)$

Esta matriz trata da relação entre os deslocamentos nas juntas do robô Inter e a posição do efetuador-final. O problema de encontrar posição e orientação do efetuador-final a partir de deslocamentos nas juntas refere-se à cinemática direta.

A cinemática inversa refere-se ao problema encontrar deslocamentos angulares dados a posição e orientação do efetuador-final.

O problema da cinemática inversa para o caso do robô Inter é tratado em maiores detalhes em [11]. As equações são:

$$\theta_1 = \arccos\left(\frac{x^2 + y^2 - 0.125}{0.125}\right)$$

$$\theta_0 = \operatorname{arcsen}\left(\frac{y(0.25 + 0.25\cos\theta_1 - \operatorname{sen}\theta_1)}{0.125 + 0.125\cos\theta_1}\right)$$

$$d_2 = 0.665 - z$$

$$\theta_3 = \theta_0 + \theta_1 - phi$$

onde *phi* representa o ângulo de rotação em torno do eixo Z.

Outro ponto de interesse é o que trata da velocidade e da aceleração com que o manipulador se move. Estas podem ser simplesmente definidas como [10]:

$$v = J\dot{\theta}$$

$$a = J\ddot{\theta} + \dot{J}\dot{\theta}$$

onde v a são a velocidade e aceleração no espaço cartesiano, θ e θ são a velocidade e aceleração no espaço de junta, respectivamente.

A matriz J é o Jacobiano do manipulador e representa os incrementos infinitesimais entre deslocamentos nas juntas e a localização do efetuador-final. $\dot{J}$ é a derivada do Jacobiano.

Considerando o robô Inter, as equações de velocidade aceleração angulares são:

$$\begin{bmatrix} \dot{\theta}_0 \\ \dot{\theta}_1 \\ \dot{d}_2 \\ \dot{\theta}_3 \end{bmatrix} = J^{-1} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{phi} \end{bmatrix} \quad \text{e} \quad \begin{bmatrix} \ddot{\theta}_0 \\ \ddot{\theta}_1 \\ \ddot{d}_2 \\ \ddot{\theta}_3 \end{bmatrix} = J^{-1} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{phi} \end{bmatrix} - J^{-1}\dot{J} \begin{bmatrix} \dot{\theta}_0 \\ \dot{\theta}_1 \\ \dot{d}_2 \\ \dot{\theta}_3 \end{bmatrix}$$

e a sua matriz Jacobiana é:

$$J = \begin{bmatrix} -(0.25\operatorname{sen}\theta_0 + 0.25\operatorname{sen}\theta_{01}) & -0.25\operatorname{sen}\theta_{01} & 0 & 0 \\ 0.25\cos\theta_0 + 0.25\cos\theta_{01} & 0.25\cos\theta_{01} & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 1 & 1 & & -1 \end{bmatrix}$$

onde $\operatorname{sen}\theta_{01} = \operatorname{sen}(\theta_0 - \operatorname{sen}\theta_1)$.

Com isto, definiu-se importantes conceitos que são utilizados posteriormente para a definição da trajetória. A seguir são apresentados alguns aspectos da estrutura do robô utilizado neste trabalho.

2.2 Descrição do Robô SCARA (*Selective Compliant Articulated Robot for Assembly*).

O robô *Inter*, como também é chamado, foi construído na Universidade Federal de Zurique, ETH, sob encomenda da Universidade Federal de Santa Catarina, UFSC, e, atualmente, pertence ao Laboratório de Robótica (Fig. 2.6). A idéia principal, relacionada a sua encomenda, está na possibilidade de sua utilização em atividades de pesquisas a serem realizadas neste laboratório, tais como implementação de algoritmos de controle de força, geradores de trajetórias, entre outros.

Este robô é uma versão especial de robô articulado e possui quatro graus de liberdade. A primeira, segunda e quarta junta do robô são de rotação e giram em torno de eixos verticais; a terceira junta é de translação e se desloca verticalmente. A figura 2.7a apresenta um esquema do manipulador e suas respectivas juntas e elos e a figura 2.7b, os motores que movimentam estas juntas. Cada um destes motores possui um sensor, também chamado de *encoder*, que é responsável pela leitura de posição, de acordo com o sentido de rotação do motor.

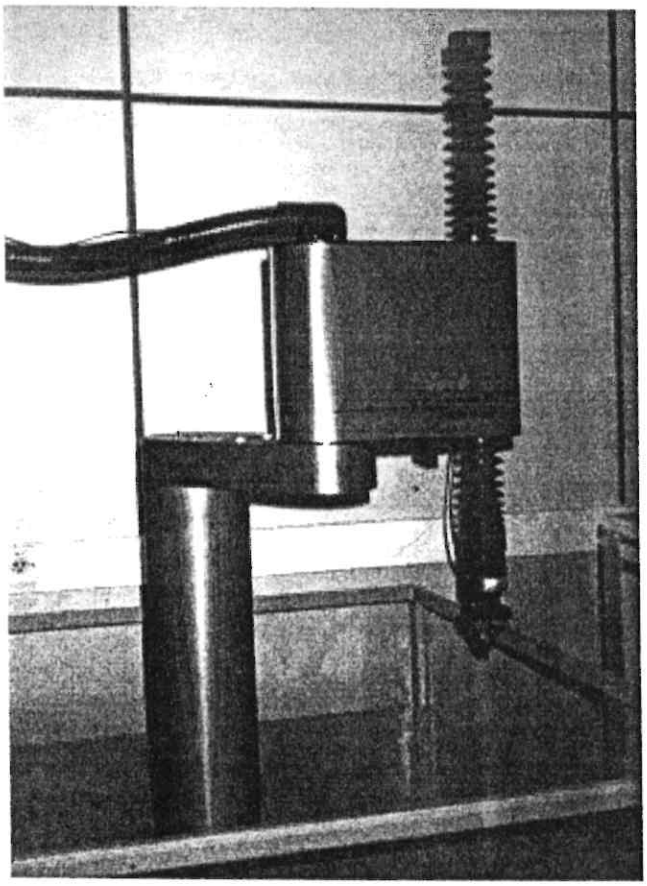


Figura 2.6 - Foto do Robô SCARA

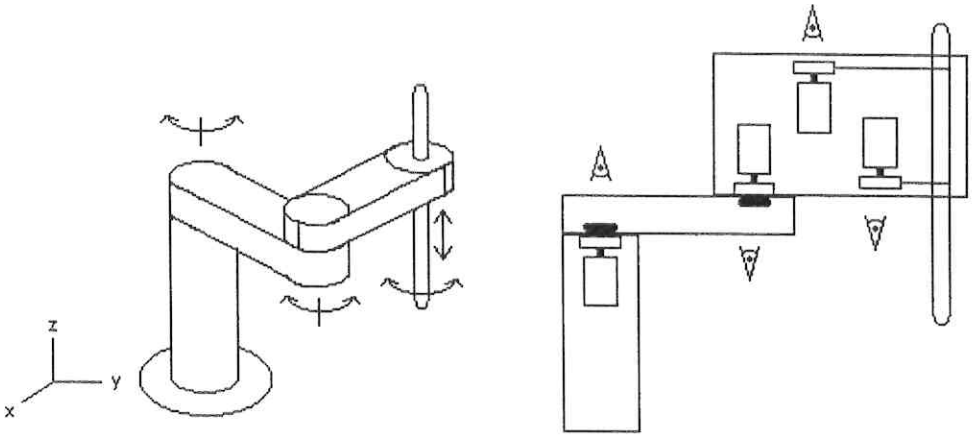


Figura 2.7 - Esquema da Estrutura. a) Aspecto de Juntas do robô e b) A montagem de seus motores

Espaço de Trabalho

A figura 2.8 ilustra o espaço de trabalho do robô no plano XY, que é gerado pelo movimento de rotação das juntas 1 e 2 (q_1 e q_2, respectivamente).

Além da área tracejada, sempre atingidas, e da área em branco, nunca atingida, existem duas outras áreas especiais, chamadas de região à Direita, preenchida pelo símbolo +, e região à Esquerda, preenchida pelo símbolo *. Estas regiões só são atingidas em configurações particulares das juntas 1 e 2. Quando o manipulador está em uma destas áreas, diz-se que está trabalhando na configuração à Direita ou à Esquerda, respectivamente.

A região à Direita é definida quando a junta 1 está em seu deslocamento angular máximo no sentido horário (q_1 = 2,25 radianos), que na figura 2.3 é definido como $Q_{0máx}$, e estando nesta posição, faz-se a junta 2 variar entre seus valores máximo e mínimo permitidos (q_2 = 1,9 radianos), definido como Q_1, na mesma figura.

A particularidade desta região está no fato de não podermos atingir o seu interior quando a junta 1 está em seu valor máximo de deslocamento angular e a junta 2 sofre uma rotação no sentido horário, ou seja, $Q_1 < 0$.

Para determinar-se a região à Esquerda o procedimento é análogo.

A figura 2.9 define o espaço de trabalho do robô no eixo vertical, onde estão ilustradas as posições máximas e mínimas possíveis num movimento de translação para a junta 3.

Conforme pode ser observado, esta junta é responsável pela movimentação vertical do robô e permite ao efetuador-final variar sua posição entre 265mm e 525mm.

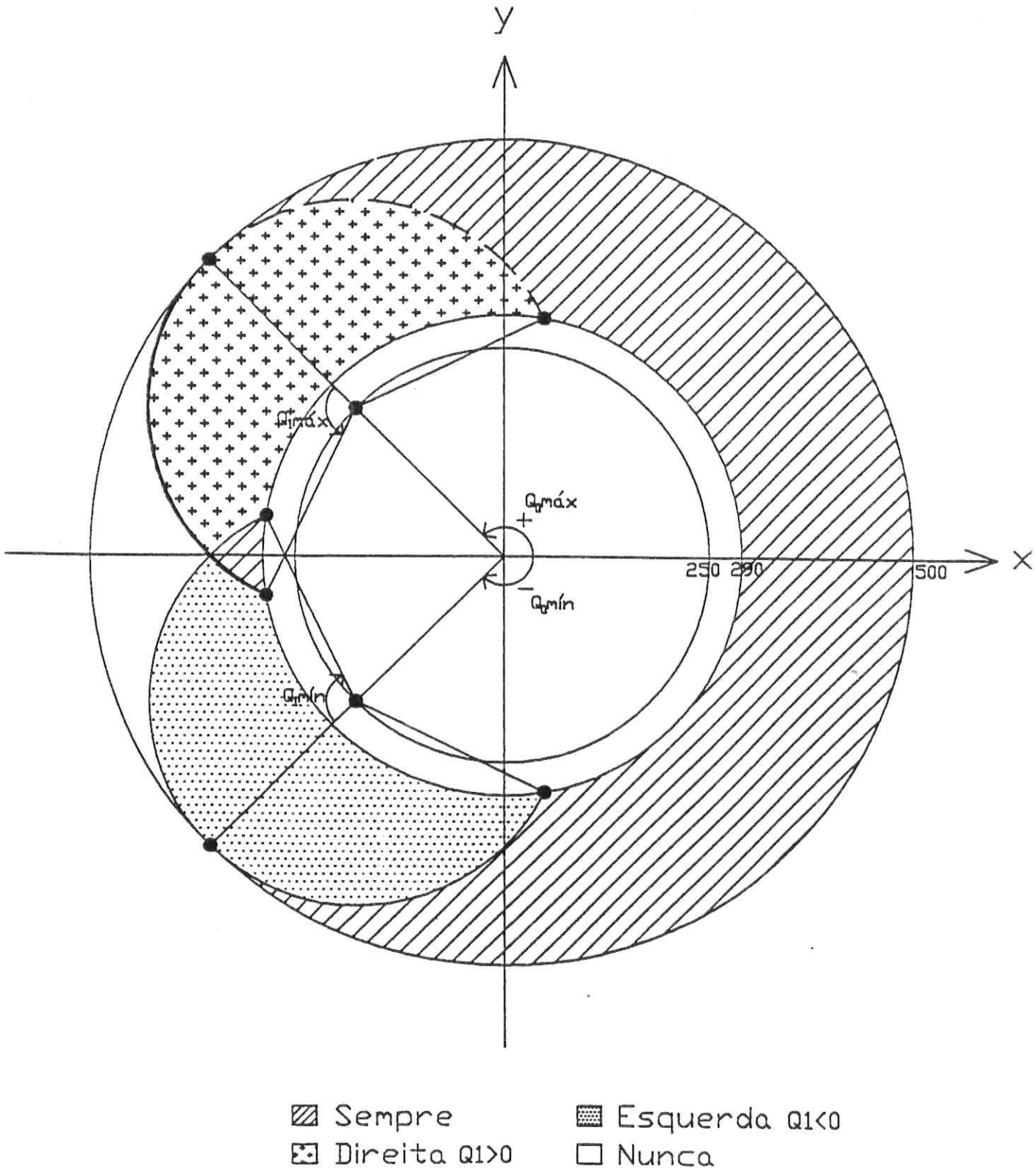


Figura 2.8 - Espaço de Trabalho do robô Inter no plano XY

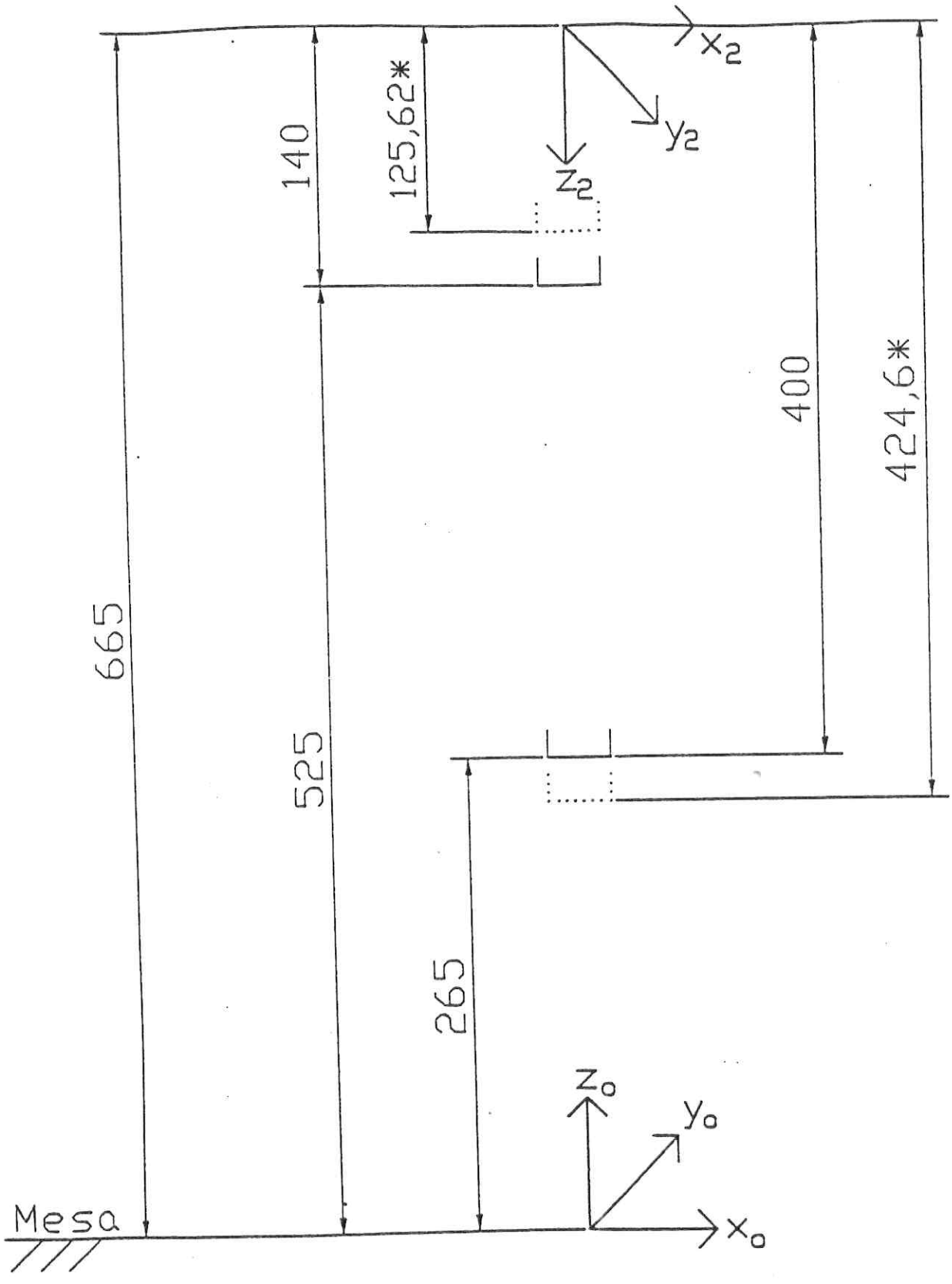


Figura 2.9 - Espaço de Trabalho do Robô Inter no Eixo Z

Armário de Controle

Na figura 2.10 pode ser observado o armário de controle, onde estão montados todos os dispositivos de comunicação, controle e alimentação elétrica do robô Inter.

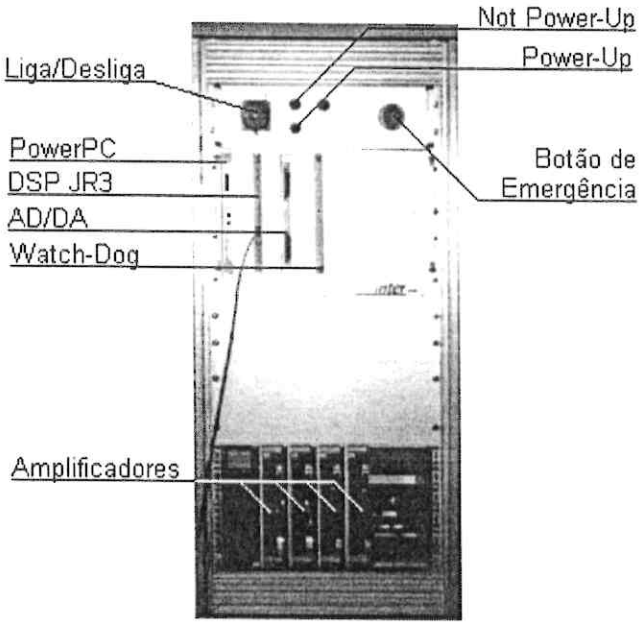


Figura 2.10 - O Armário de Controle do Robô Inter

Segue abaixo uma relação dos principais componentes do armário de controle:

- *Placa POWERPC:*
  1. Barramento VME (*clock* bus de 67 MHz);
  2. *Clock* de 200 MHz;
  3. Memória RAM de16 Mb;
  4. Memória Cache de 256kb.
- *Amplificadores:*

  Num total de quatro. Podem ser configurados individualmente, através do *software* que os acompanha, de acordo com os motores que alimentam, podendo-se configurar correntes de trabalho e corrente máxima.

- *Botão de Emergência:*

  Botão que pode ser acionado tanto no armário como no "controle remoto de emergência". Quando acionado, o robô pára imediatamente a execução da tarefa e o botão *power-up* é automaticamente desativado.

- *Power-Up e Not-Power-Up:*

  Botões que servem, respectivamente, para ligar e desligar o robô.

- *JR3:*

  Placa do sensor de força, responsável pelo processamento digital dos sinais do sensor de força.

- *Watch-Dog:*

  Dispositivo de Segurança que é responsável pelo isolamento de corrente entre os dispositivos, evitando interferência entre placas e amplificadores (que trabalham com níveis de tensão e correntes muito maiores).

A comunicação entre o computador que contém a linguagem utilizada na programação do robô Inter e o armário de controle ainda não foi totalmente definida. O que se sabe atualmente é que esta comunicação é viabilizada a partir de algum protocolo de rede que é responsável por efetuar a troca de mensagens entre computador onde está o ambiente XOberon e armário de controle.

A parte mais importante desta seção refere-se principalmente a área de trabalho do robô e a definição de sua estrutura, pois tem relação direta com o presente trabalho. Informações técnicas mais completas podem ser adquiridas nos manuais que acompanham o robô e que estão disponíveis no Laboratório de Robótica. A seguir são apresentados alguns conceitos de robótica utilizados posteriormente para a definição da técnica de planejamento da trajetória a ser empregada.

2.3 Geração de Trajetórias para Robôs

A geração de trajetória de um manipulador consiste no estudo de métodos computacionais para encontrar uma trajetória que represente o caminho desejado a ser descrito pelo efetuador-final para execução de uma tarefa

A trajetória pode ser definida como a história temporal da posição, velocidade e aceleração em cada um dos graus de liberdade do robô. Assim, deseja-se encontrar uma seqüência temporal de pontos que descrevam o movimento do manipulador quanto a sua posição, velocidade e aceleração, definindo totalmente a sua configuração a cada instante de tempo. Esta seqüência de pontos é posteriormente enviada ao sistema de controle do robô. Este sistema é responsável pelo acionamento dos atuadores que movimentam o manipulador.

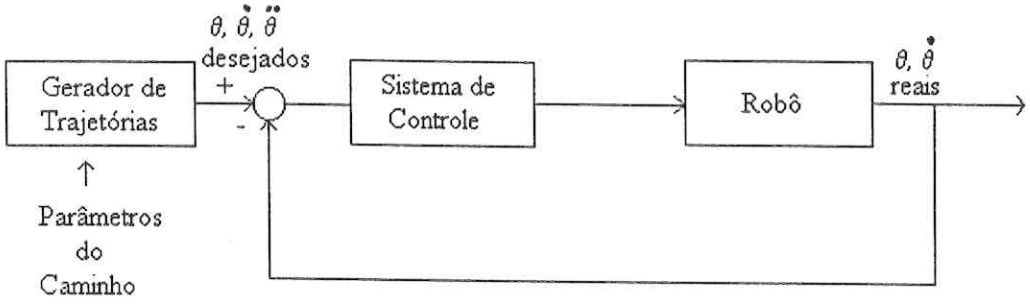


Figura 11 - Esquema do Sistema de Geração de Trajetórias

2.3.1 Considerações Gerais da Geração de Trajetórias

O problema básico é mover o efetuador-final de uma localização inicial até uma localização final. Às vezes, porém, deseja-se especificar mais detalhes para o movimento do que a simples posição final. Uma maneira de incluir maiores detalhes é a definição de *via points*, que são pontos intermediários entre o ponto inicial e final. Assim, o efetuador-final deve passar através de posições e orientações intermediárias como definidas nestes *via points*. Cada um deles especifica a posição e orientação do efetuador-final em relação a estação de trabalho. Uma outra maneira de adicionar mais detalhes à trajetória é a especificação de atributos temporais ao movimento, como, por exemplo, a definição do tempo exigido para cumprir parte do caminho.

Com relação ao usuário, deve-se levar em consideração o problema do interfaceamento humano no processo de planejamento da trajetória. O objetivo é fazer uma descrição do movimento do manipulador que seja o mais simples possível para a interpretação de um usuário humano, de modo que não lhe seja exigido escrever complicadas funções para especificar a tarefa desejada. Ao invés disto, deve-se permitir a especificação de trajetórias com simples descrições e deixar o sistema calcular os detalhes. Por exemplo, o usuário pode especificar a posição e orientação desejada e deixar o sistema decidir a forma exata do caminho, a duração, o perfil de velocidade e aceleração, entre outros detalhes.

Também é preciso levar em consideração que a trajetória é calculada em computadores digitais que possuem uma certa taxa típica de amostragem. Esta taxa deve ser respeitada para que o movimento seja suave. Desta forma, a cada instante de tempo os novos valores de posição, velocidade e aceleração devem estar disponíveis para serem enviados ao sistema de controle do robô. Este é mais um fator contribuinte para que se trabalhe com trajetórias definidas em espaço de juntas, já que geralmente tem-se uma elevada freqüência de amostragem e a cinemática inversa, para transformar a seqüência de pontos cartesianos para o espaço de juntas, exige muito esforço - e tempo - computacional.

Usualmente é desejado que o movimento do manipulador ocorra de maneira suave. Para isto, pode ser definida uma função suave de modo que ela e sua primeira derivada sejam contínuas. Também pode ser exigido, dependendo da aplicação, que sua segunda derivada seja contínua para evitar que ocorra um aumento no desgaste dos órgãos mecânicos do robô, e também para evitar excitações de altas frequências em seu mecanismo articulado. Finalmente deve-se ter a preocupação de avaliar a trajetória proposta, verificando se esta leva o braço do manipulador para fora da área de trabalho, ou, então, se os limites de velocidade ou de aceleração são respeitados.

É apresentada a seguir uma discussão sobre os métodos usuais estudados para o planejamento de trajetórias - equações polinomiais e funções lineares com curvas parabólicas acopladas - avaliando estes métodos dentro do espaço de juntas. A idéia é proporcionar uma visão geral ao leitor, de modo que ele possa se situar melhor no contexto deste trabalho. Para uma pesquisa mais aprofundada é necessário uma consulta às referências [3,8,9].

2.3.2 Métodos de Geração de Trajetórias em Espaço de Juntas

A geração da forma da trajetória é descrita aqui em função dos ângulos das juntas do manipulador. Geralmente os pontos da trajetória são especificados no espaço cartesiano que, através da cinemática inversa, são transformados para o espaço de juntas.

2.3.2.1 Equações Polinomiais no Espaço de Juntas

As equações polinomiais definidas no espaço de juntas têm a seguinte forma:

$$q(t) = a_0 + a_1 t + a_2 t^2 + \ldots + a_n t^n$$

onde, a_i, para i=0,1,...,n são os coeficientes a serem determinados para que seja definida a equação polinomial do deslocamento na junta em função do tempo.

Inicialmente é apresentado o estudo de um movimento simples, ponto a ponto, onde o interesse é encontrar uma função "suave" que represente a posição angular de cada uma das juntas do robô. Para este caso em particular, podem ser utilizados polinômios de pequena ordem, como por exemplo, polinômios cúbicos.

Polinômios Cúbicos

Um polinômio cúbico tem a seguinte forma:

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \qquad \text{Eq. 2.1}$$

e suas primeira e segunda derivadas são:

$$\dot{q}(t) = a_1 + 2a_2 t + 3a_3 t^2$$

$$\ddot{q}(t) = 2a_2 + 6a_3 t$$

estas duas últimas equações representam velocidade e aceleração angular, respectivamente.

Como são quatro parâmetros a serem determinados, a_0, a_1, a_2 e a_3, são necessários quatro condições iniciais, que são posição e velocidade angular inicial e final q_i, q_f, $\dot{q}_i$, $\dot{q}_f$. Para este caso particular, a velocidade inicial e final são nulas. A determinação dos parâmetros de 2.1 é feita através da resolução do seguinte sistema de equações:

$$a_0 = q_i$$

$$a_1 = \dot{q}_i$$

$$a_3 t_f^3 + a_2 t_f^2 + a_1 t_f + a_0 = q_f$$

$$3a_3 t_f^2 + 2a_2 t_f + a_1 = \dot{q}_f$$

A fig. 2.12 ilustra uma função obtida com os seguintes valores de posição, velocidade e tempo: $q_i = 0$, $q_f = \pi$ e $q_f = q_i = 0$. Pode ser observado um perfil parabólico para velocidade angular e um perfil linear para aceleração, esta com descontinuidade em seus extremos.

Polinômios de Ordem n

Os polinômios de maior ordem são utilizados quando se deseja especificar, além de posição e velocidade, a aceleração de início e fim de cada segmento

da trajetória. Neste caso é possível utilizar polinômios de 5ª ordem, que têm a seguinte forma:

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5$$

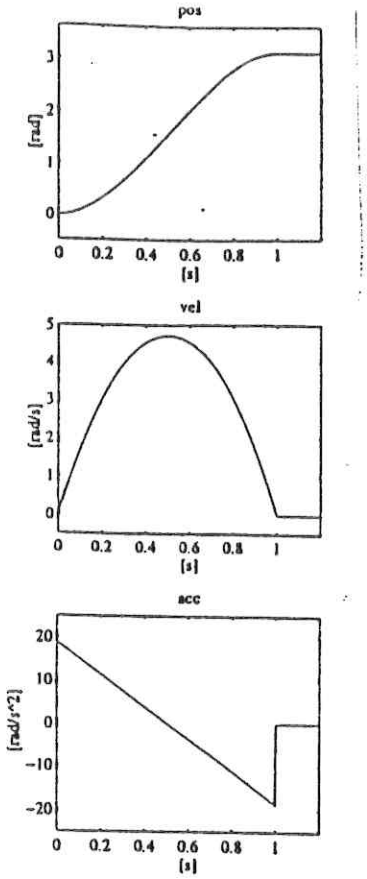


Figura 2.12 - História temporal da posição, velocidade e aceleração utilizando polinômios cúbicos

Considerando que os valores de posição, velocidade e aceleração no início e fim do segmento estejam disponíveis, os coeficientes do polinômio de 5ª. ordem podem ser encontrados com as seguintes equações [3]:

$$a_0 = q_i$$

$$a_1 = \dot{q}_i$$

$$a_2 = \frac{\ddot{q}_i}{2}$$

$$a_3 = \frac{20q_f - 20q_i - (8\dot{q}_f + 12\dot{q}_i)t_f - (3\ddot{q}_i - \ddot{q}_f)t_f^2}{2t_f^3}$$

$$a_4 = \frac{30q_f - 30q_i + (14\dot{q}_f + 16\dot{q}_i)t_f + (3\ddot{q}_i - 2\ddot{q}_f)t_f^2}{2t_f^4}$$

$$a_5 = \frac{12q_f - 12q_i - (6\dot{q}_f + 6\dot{q}_i)t_f - (\ddot{q}_i - \ddot{q}_f)t_f^2}{2t_f^5}$$

Equação dos coeficientes de um polinômio de 5ª ordem

2.3.2.2 Funções Lineares com Curvas Parabólicas Acopladas

Uma outra escolha para representar o caminho, é a interpolação linear dos pontos que o definem. O problema é que este tipo de interpolação faz a velocidade ser descontínua no início e fim do movimento. Para criar um movimento com posição e velocidade contínua, é possível começar com uma função linear e no final da reta, adicionar uma região curva parabólica. Durante a parte curva da trajetória, uma aceleração constante é usada para mudar a velocidade. Na figura 2.13, a função linear e as duas parabólicas são unidas para formar um caminho contínuo em posição e velocidade. Pode ser observado na mesma figura o perfil de velocidade e aceleração angular.

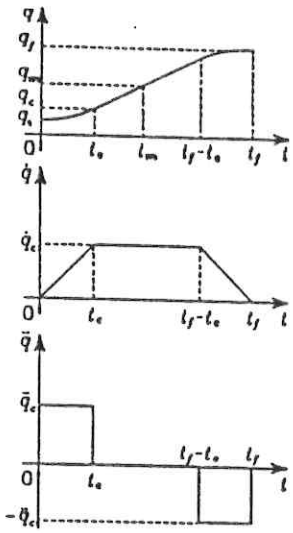


Figura 2.13 - Funções Lineares com Curvas Parabólicas Acopladas

Pode ser observado que a aceleração tem a mesma magnetude no inicio e fim do movimento. Também será considerado que as parábolas têm o mesmo tempo de duração. Há várias formas de solução, mas a resposta é sempre simétrica ao ponto médio no tempo $t_m = \frac{t_f}{2}$ e ao ponto médio de posição $q_m = \frac{(q_f + q_i)}{2}$. A trajetória tem que satisfazer algumas condições para assegurar a transição da posição inicial para a final no tempo. A velocidade no final da curva deve ser igual a da seção linear, então:

$$\ddot{q}_c t_c = \frac{(q_m - q_c)}{t_m - t_c}$$

onde, $\ddot{q}_c$ é a aceleração na curva, q_c é o valor de q no fim da curva no tempo t_c.

O valor de q_c é:

$$q_c = q_i + \frac{\ddot{q}_c t_c^2}{2}$$

Agrupando, tem-se:

$$\ddot{q}_c t_c^2 - \ddot{q}_c t_c t_f + q_f - q_i$$

onde t_f é o tempo de duração total da trajetória.

Usualmente, $\ddot{q}_c$ é escolhido de acordo com t_c e deve ser suficientemente alto para que a solução exista, ou seja:

$$t_c = \frac{t_f}{2} - \frac{1}{2}\sqrt{\frac{t_f^2 \ddot{q}_c - 4(q_f - q_i)}{q_c}}$$

onde a seguinte restrição deve ser respeitada:

$$\left|\ddot{q}_c\right| \geq \frac{4\left|q_f - q_i\right|}{t_f^2} \qquad \text{Eq. 2.3}$$

Se a aceleração é alta, a região de curvatura é pequena e no limite de aceleração infinita tem-se um caso de interpolação linear. Caso a aceleração escolhida satisfaça a Eq. 2.3, a trajetória resultante terá um perfil de velocidade triangular. A figura 2.14 apresenta o perfil de curvas de posição, velocidade e aceleração para uma interpolação linear utilizando esta metodologia usando valores diferentes de aceleração.

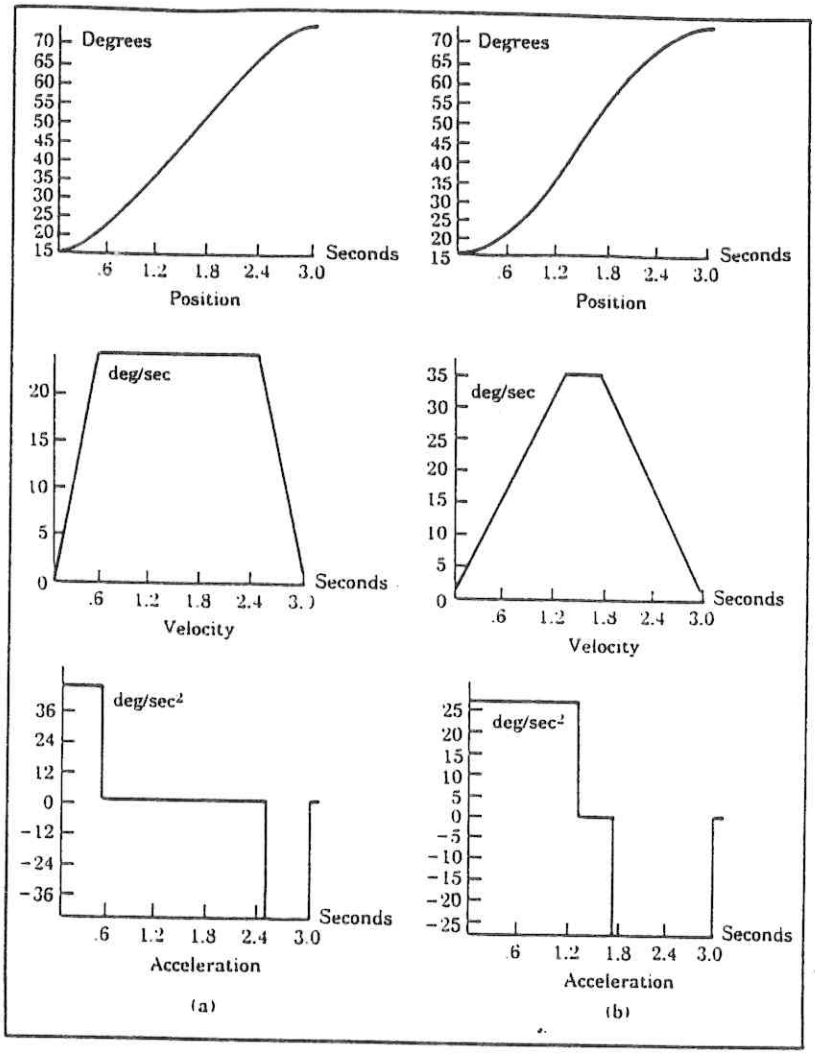


Figura 2.14 - Interpolação linear utilizando curvas parabólicas acopladas para diferentes valores de aceleração

Dados q_0, q_f e t, qualquer caminho pode ser seguido escolhendo apropriadamente q_c e t_c. A Eq. 2.3 pode ser usada para calcular q_c e as seguintes equações são geradas:

$$\ddot{q}_c = \begin{cases} q_i + \dfrac{\ddot{q}_c t^2}{2} & 0 \le t \le t_c \\ q_i + \ddot{q}_c t_c (t - \dfrac{t_c}{2}) & t_c < t \le t_f - t_c \\ q_f - \dfrac{\ddot{q}_c (t_f - t)^2}{2} & t_f - t_c < t \le t_f \end{cases}$$

onde t_f é o tempo final.

A figura 2.15 ilustra uma representação de movimento onde foram definidos os seguintes dados: $q_0 = 0$, $q_f = \pi$, $t_f = 1s$ e $|\ddot{q}_c| = 6\pi$.

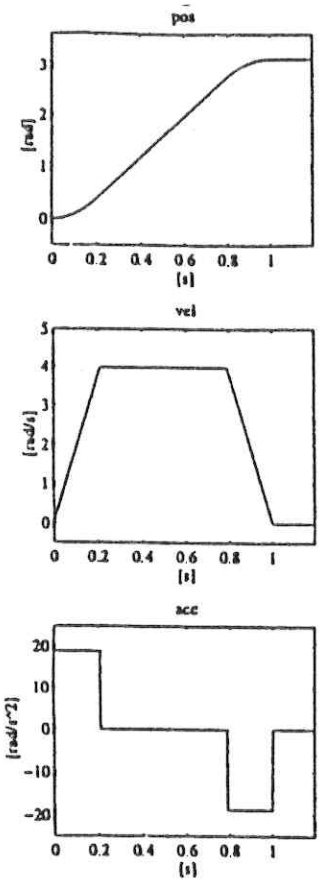


Figura 2.15 - História temporal da posição, velocidade e aceleração

2.3.2.3 Caminho com Pontos Intermediários

Na maioria das aplicações, o caminho é definido como uma seqüência de pontos para garantir um maior monitoramento da trajetória. Uma maior quantidade de pontos é escolhida, por exemplo, nos segmentos que contêm obstáculos a serem evitados. Este caminho multisegmentado é analisado considerando as duas técnicas propostas.

Equações Polinomiais

O problema é gerar uma trajetória quando N pontos são especificados para serem atingidos pelo manipulador em determinados instantes de tempo. Para cada junta, existem N parâmetros a calcular para que seja definido um polinômio de ordem N-1 que representa a trajetória. Embora estes polinômios sempre existam, esta

metodologia apresenta desvantagens. Por exemplo, se um polinômio de grau elevado é utilizado, a precisão de cálculo do polinômio diminui, já que o erro também é função do tempo e será amplificado pela potência t^n. Outro exemplo está na dificuldade de verificar se um polinômio de grau n gera uma trajetória que ultrapassa os limites físicos do braço do manipulador durante o movimento. Isto acontece porque, para verificar se uma dada trajetória polinomial ultrapassa o espaço de trabalho, é necessário achar os máximos de uma função de grau n, logo é necessário resolver uma equação algébrica de grau n-1, ou seja, sua primeira derivada. Com isto, deve-se considerar que quanto maior o grau do polinômio, maior o esforço computacional a ser realizado para o determinar.

Ao invés de representar o caminho através de uma equação geral de grau elevado, podem ser utilizados polinômios de baixa ordem para representar cada segmento individual que compõem o caminho, lançando mão de uma interpolação polinomial. No caso de utilizar polinômios cúbicos para cada segmento, a velocidade deve ser definida em cada um dos pontos do caminho. Para manter a sua continuidade, deve ser assegurado que a velocidade no final de um segmento qualquer (exceção do último) seja igual a velocidade inicial do segmento seguinte, ou seja, $\dot{q}_{k-1} = \dot{q}_k$.

A velocidade em cada ponto pode ser definida pelo usuário ou pelo próprio sistema. Neste último caso, um critério que pode ser utilizado para calcular estas velocidades pode ser o seguinte:

$$\dot{q}_i = 0$$

$$\dot{q}_k = \begin{cases} 0 & \text{sgn}(v_k) \neq \text{sgn}(v_{k+1}) \\ 0{,}5(v_k + v_{k+1}) & \text{sgn}(v_k) = \text{sgn}(v_{k+1}) \end{cases}$$

$$\dot{q}_n = 0$$

onde $v_k = \dfrac{q_k - q_{k-1}}{t_k - t_{k-1}}$ no intervalo $\left[t_{k-1}, t_k\right]$. A figura 2.16 apresenta a história temporal da posição, velocidade e aceleração obtida com os seguintes dados: $q_0 = 0$, $q_1 = 2\pi$, $q_2 = \pi/2$, $q_3 = \pi$, $t_0 = 0$, $t_1 = 2s$, $t_2 = 3s$, $t_3 = 4s$ e velocidades inicial e final nulas.

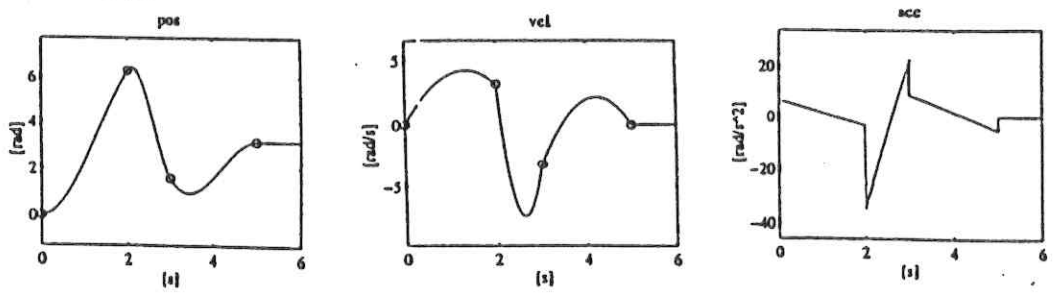


Figura 2.16 - História temporal da posição, velocidade e aceleração para caminho multisegmentado

Funções Lineares com Curvas Parabólicas Acopladas

Para o caso de caminhos com pontos intermediários é utilizada a seguinte notação. Considere três pontos vizinhos chamados j, k e l. A duração da curva é t_k no ponto k. A duração da porção linear entre j e k é t_{jk}. A duração total entre j e k é t_{djk}. A velocidade durante a porção linear é $\dot{q}_{jk}$ e a aceleração na curva no ponto j é $\ddot{q}_j$ (fig. 2.17).

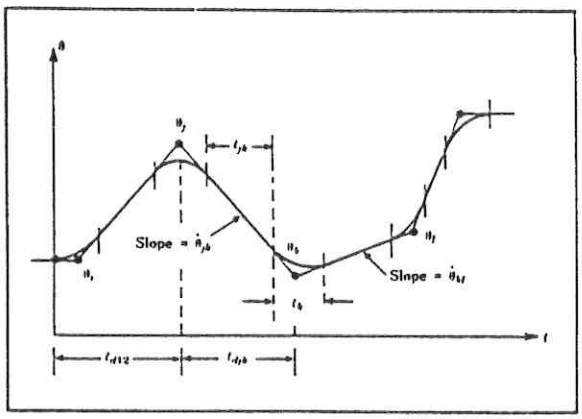


Figura 2.17 - Caminho multisegmentado linear com curvas parabólicas acopladas

A várias maneiras de solução, dependendo do valor de $\dot{q}_j$. Dados todos os pontos do caminho, q, duração desejada t_{djk} e a magnitude da aceleração, o tempo na curva t_k pode calculado. As equações para calcular valores de segmentos internos são [3]:

$$\dot{q}_{jk} = \frac{(q_k - q_j)}{t_{djk}}$$

$$\ddot{q}_j = \mathrm{sgn}(\dot{q}_{kl} - \dot{q}_{jk})\left|\ddot{q}_k\right|$$

$$t_k = \frac{(\dot{q}_{kl} - \dot{q}_{jk})}{\ddot{q}_k}$$

$$t_{jk} = t_{djk} - \frac{t_j}{2} - \frac{t_k}{2}$$

Equações para Segmentos Internos

onde, SGN significa que somente será utilizado o sinal do resultado entre os parênteses.

O primeiro e último segmentos devem ser tratados diferentemente (Tabela 2.2).

Segmento Inicial	Segmento Final
$\dot{q}_{12} = \frac{(q_2 - q_1)}{(t_{d12} - \frac{t_1}{2})}$	$\dot{q}_{(n-1)n} = \frac{(q_n - q_{n-1})}{(t_{d(n-1)n} - \frac{t_{n-1}}{2})}$
$\ddot{q}_1 = \text{sgn}(\dot{q}_2 - \dot{q}_1)\lvert\ddot{q}_1\rvert$	$\ddot{q}_{n-1} = \text{sgn}(\dot{q}_n - \dot{q}_{n-1})\lvert\ddot{q}_{n-1}\rvert$
$t_1 = t_{d12} - \sqrt{t_{d12}^2 - \frac{2(q_2 - q_1)}{\ddot{q}_1}}$	$t_{n-1} = t_{d(n-1)n} - \sqrt{t_{d(n-1)n}^2 - \frac{2(q_n - q_{n-1})}{\ddot{q}_n}}$
$t_{12} = t_{d12} - t_1 - \frac{t_2}{2}$	$t_{(n-1)n} = t_{d(n-1)n} - t_n - \frac{t_{n-1}}{2}$

Tabela 2.2 - Equações para cálculo de valores de segmentos inicial e final

Com estas equações é possível resolver casos de caminhos multisegmentados. Usualmente especifica-se os via *points* e t_{dkl} e o sistema usa valores *defaut* para $\ddot{q}$ em cada junta.

Os resultados constituem um plano para a trajetória. Em tempo de execução, estes números seriam usados pelo gerador de trajetórias para calcular q , $\dot{q}$ e $\ddot{q}$ enviando-os na seqüência ao sistema de controle do manipulador.

Com isto termina-se a parte de análise de técnicas para planejamento de trajetórias em espaço de juntas. É apresentado a seguir, de forma resumida, como trabalhar em espaço cartesiano e, na seqüência, é feita uma discussão sobre o melhor método a ser utilizado neste trabalho.

2.3.3 Esquemas para o Espaço Cartesiano

No esquema de espaço de juntas, a forma espacial do caminho percorrido pelo efetuador-final tem uma forma complicada que depende da cinemática do manipulador sendo usado. Usando o método de geração de trajetórias no espaço cartesiano é possível especificar a forma do caminho entre os pontos escolhidos, ou

seja, se o caminho é uma linha reta, um círculo, uma senóide, entre outros. O problema é que caminhos definidos em espaço cartesiano são mais custosos de serem executados em tempo-real devido a necessidade de se calcular a transformação de valores cartesianos em valores no espaço de junta durante a execução.

Nesta seção é feita uma breve discussão sobre o planejamento trajetórias no espaço cartesiano para o caso de trajetórias retilíneas, devido a ser o caso mais simples e também pelo fato de que outras trajetórias mais complexas poderem ser formadas pela justaposição de segmentos de reta. Outra razão é o fato de trajetórias retilíneas serem mais simples para visualização, além de apresentarem uma ampla aplicação em processos industriais, tais como operações de montagem, realização de cordão de solda entre duas chapas, etc.

Usando o esquema de função lineares com curvas parabólicas para o cálculo dos valores que representam posição e orientação, basta substituir o símbolo X no lugar de símbolos que representam ângulos nas equações já estudadas. Ele representa a coordenada de posição cartesiana do efetuador-final em relação ao eixo X (Vale o mesmo para os eixos Y e Z). Então, na porção linear do movimento, tem-se:

$$x = x_j + \dot{x}_{jk}\, t$$

$$\dot{x} = \dot{x}_{jk}$$

$$\ddot{x} = 0$$

e na região curva:

$$t_c = t_{djk} - (\frac{t_j}{2} + \frac{t_k}{2})$$

$$x = x_j + \dot{x}_{jk}(t_{djk} - t_c) + (\frac{\ddot{x}_k}{2})t_c^2$$

$$\dot{x} = \dot{x}_{jk} + \ddot{x}_k\, t_c$$

$$\ddot{x} = \ddot{x}_k$$

Com os valores da trajetória cartesiana (x, dx e ddx) calculados, deve-se utilizar a cinemática inversa e encontrar os valores no espaço de juntas.

A técnica que utiliza equações polinomiais descrita na seção anterior também pode ser estendida para o espaço cartesiano. Ao invés de estudar todo o procedimento de cálculo de valores dos parâmetros, que é análogo a seção anterior, é visto agora a definição da metodologia que é empregada para o planejamento de trajetórias, através do confronto direto entre estes dois esquemas de representação. Para um melhor entendimento do que foi exposto nesta seção, é necessário recorrer às referências [3,8], onde podem ser encontrados maiores detalhes sobre esquemas em espaços cartesianos.

2.3.4 Análise da Metodologia da Geração de Trajetórias

Como ponto de partida, é considerado que o conjunto de pontos escolhidos são definidos no espaço cartesiano e que podem ser mapeados em pontos no espaço de juntas através da cinemática inversa. Sobre estes pontos é colocada a condição de que os pontos inicial e final sejam atingidos e que a trajetória descrita se aproxime o máximo possível dos pontos intermediários, de modo que estes sirvam somente para orientar o caminho a ser descrito pelo efetuador-final. Há ainda a restrição de escolher funções contínuas para representar a posição, velocidade e aceleração do manipulador durante o seu movimento.

Antes de analisar a técnica a ser utilizada na geração de trajetórias, é necessário discutir alguns problemas que aparecem com a escolha de esquemas no espaço cartesiano.

Um problema típico ocorre quando existem pontos intermediários que não podem ser alcançados pelo manipulador na trajetória pretendida. Apesar da posição inicial e final estarem dentro do espaço de trabalho, é possível que o caminho contenha pontos que não pertençam ao espaço de trabalho do robô (fig. 2.18). Este problema não ocorre em um esquema de espaço de juntas e é facilmente executado, mas o sistema cartesiano falha. Alguns sistemas avisam o problema antes que o movimento inicie e em alguns as juntas se movem até seu limite e param.

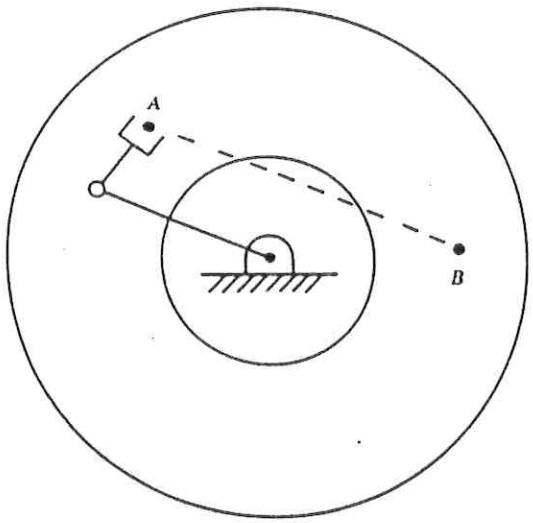


Figura 2.18 - Pontos Intermediários não atingíveis

Outro problema aparece quando se está trabalhando próximos a singularidades. Há localizações no espaço de trabalho que é impossível escolher uma variação finita nas juntas do robô, que produza a velocidade desejada, ou seja, existem caminhos que são impossíveis de percorrer no esquema cartesiano. Quando o

manipulador se aproxima de uma singularidade, uma ou mais velocidades nas juntas podem tender ao infinito e isto geralmente ocasiona um desvio do caminho desejado.

Uma solução é reduzir a velocidade das juntas que estão próximas de seus limites. Isto pode causar a perda dos atributos temporais desejados, mas garante a forma espacial do caminho (fig. 2.19).

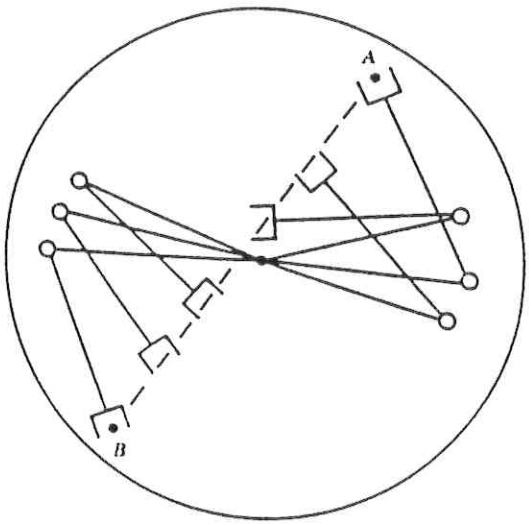


Figura 2.19 - Altas velocidades próximas a singularidades

Pode-se dizer que caminhos com pontos intermediários são descritos de maneira simples no espaço de juntas e mais complexa no espaço cartesiano. Assim, esquemas no espaço de juntas são mais fáceis de calcular e, devido a não haver correspondência entre espaço de juntas e cartesiano, não há problemas de singularidades com o mecanismo.

Por estas razões, a escolha preliminar que se propõe é uma representação em espaço de juntas como solução geral e, caso desejado, o desenvolvimento posterior em espaço cartesiano.

Com relação à técnica a ser utilizada, viu-se nas seções anteriores que, apesar dos dois métodos apresentarem uma simplicidade de execução, o método de equações polinomiais apresenta desvantagens devido a necessidade de utilizar polinômios de maior ordem para garantir a suavidade nas funções de posição, velocidade e aceleração. Com a utilização de polinômios de maior ordem está-se diminuindo a precisão do cálculo dos parâmetros da trajetória, além de aumentar o esforço computacional necessário, entre outros fatores (Ver seção 2.3.2).

Assim, como é necessário garantir algumas restrições já definidas, a utilização de uma única função para representar todo o caminho se torna insatisfatória. É mais conveniente utilizar um conjunto de funções de pequena ordem, cada uma definindo um trecho da trajetória, o que melhora o comportamento da função e simplifica o seu cálculo.

A técnica que utiliza funções lineares com curvas parabólicas acopladas apresenta a desvantagem de ser necessário empregar uma variação muito brusca nos valores de aceleração. Com isto, não é possível cumprir uma das exigências, ou seja, a de ter um movimento com perfil suave para a curva de aceleração.

Decidiu-se então que a melhor solução é a de utilizar uma interpolação polinomial, usando polinômios de 5ª. ordem para representar cada segmento do caminho. A decisão de utilizar polinômios de 5ª. ordem é para garantir a suavidade no perfil da curva de aceleração, além da possibilidade desta ser definida pelo usuário ou pelo sistema. Esta técnica é desenvolvida em termos do espaço de juntas, que apresentou diversas vantagens em relação ao espaço cartesiano, já discutidas anteriormente. Passa-se agora ao estudo do sistema *XOberon*, que é a linguagem utilizada na programação do robô Inter e na sequência é discutida a viabilicade de sua utilização no desenvolvimento do gerador de trajetórias.

2.4 O Sistema XOberon

Antes de iniciar esta seção, é necessário esclarecer que não existe uma publicação que defina de maneira clara e completa este sistema. Todo o material que ajudou a esclarecer um pouco as dúvidas encontradas foi conseguido através de algumas publicações que se referenciavam superficialmente a tópicos específicos da linguagem. Outras informações foram conseguidas através de constantes buscas à Internet, mas que também não surtiram efeitos satisfatórios. A maior contribuição de informações sobre o XOberon veio do estudo do sistema do qual ele evoluiu, o sistema *Oberon*. Estas informações, mesmo que limitadas, foram de fundamental importância para o entendimento da programação em XOberon.

O Oberon

O Sistema Oberon foi planejado para ser uma evolução a partir do Módula-2. Esta é uma linguagem de alto nível, mas que não permite a definição de novos tipos de dados a partir de outros já existentes [6]. Desta forma, a principal novidade que pode ser observada no Oberon é a facilidade da extensão de tipos de dados e abstração de informações. Estas filosofias estão atualmente bem entendidas devido, principalmente, ao estudo de linguagens já existentes, como Módula-2, Smaltalk, entre outros.

De acordo com seus desenvolvedores, o "Oberon evoluiu do Módula por muito pouca adição e muitas subtrações (...) e é parte de um desenvolvimento que começa em ALGOL, para Pascal, então para Módula-2 e agora para Oberon".

Devido a facilidade para extensão de tipos de dados, o Oberon permite uma programação em estilo orientação objeto e, segundo estes mesmos desenvolvedores, alcança esta filosofia com um número mínimo de instruções. Um Objeto Oberon é

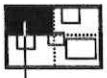
uma instância de um tipo abstrato de dados que é especificado por um ponteiro para uma estrutura de dados (*record*), chamada *Message*, que contém as informações de estado do objeto.

Basicamente, o Oberon pode ser definido como sendo um sistema formado por uma coleção de módulos para diferentes aplicações, como, por exemplo, módulos para criação de janelas de comunicação. Cada um destes módulos possuem objetos próprios além de procedimentos ou métodos que são usados para manipulação destes objetos.

O XOberon

Pode-se dizer que o sistema XOberon é uma extensão do Oberon e visa proporcionar um ambiente de desenvolvimento para Tempo-Real e que permite uma implementação de novas estratégias de controle, como por exemplo algoritmos para o controle de robôs e sistemas mecatrônicos.

A fig. 2.20 apresenta uma das áreas de trabalho do XOberon. A área total pode ser vista como um grande "quadro negro" dividido em quatro partes, cada uma sendo representado por uma janela ou área diferente. Na inicialização, o monitor mostra uma das janelas (fig. 2.20) enquanto as outras três permanecem ocultas. Para poder trabalhar nas outras áreas, um recurso chamado *Navegator* é utilizado (figura abaixo). Ele possibilita a "navegação" pelo ambiente, ativando as diferentes janelas. Para passar de uma área a outra, basta clicar com o botão esquerdo do mouse em uma das janelas nele representadas.



Janela atual

Com relação a programação em XOberon, pode-se dizer que seu estudo baseou-se principalmente na busca das semelhanças existentes com o sistema Oberon, o qual dispõe de publicações que o definem completamente. Além deste estudo feito através de comparações com o Oberon, também foram estudados *softwares* desenvolvidos nos laboratórios da Universidade de Zurique, ETH.

A maior dificuldade encontrada foi que os módulos que podiam ser encontrados em ambos os sistemas ou não possuíam os mesmos objetos e métodos, ou eram utilizados de maneira diferente - e desconhecida - em XOberon. Por exemplo, o Oberon possui o módulo File que dispõe de um importante objeto manipulador de arquivo, o objeto *rider*, e o XOberon possui o módulo *XFile* que, embora possua objetos diferentes, não possui este objeto em particular e não foi encontrado um outro objeto que fosse capaz de realizar a mesma tarefa que o *rider* realiza no Oberon.

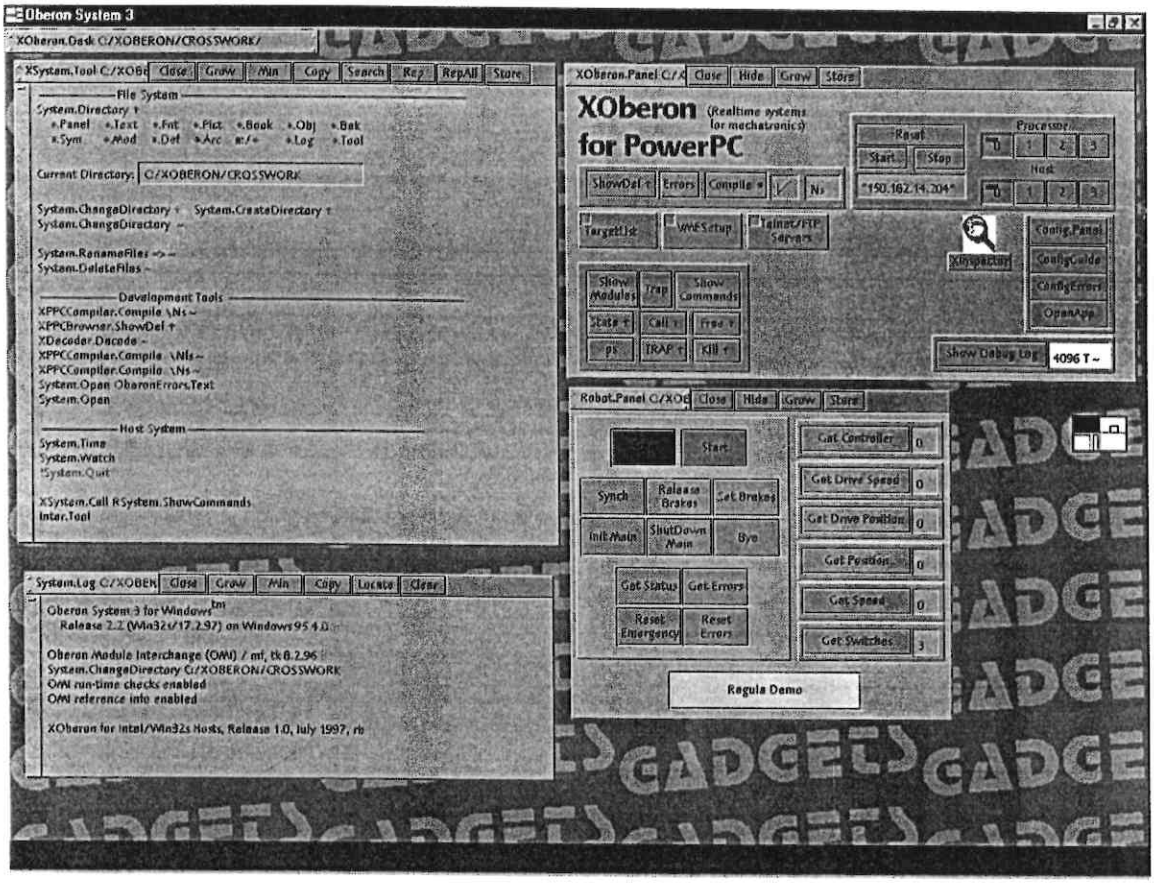


Figura 2.20- A Interface do Sistema XOberon

Por outro lado, os *softwares* já desenvolvidos e disponíveis para estudo, não eram suficientemente comentados, isto é, não possuíam explicações claras sobre a função das variáveis e procedimentos utilizados no programa. Além disso, grande parte dos programas foram escritos com mnemônicos e abreviações da língua alemã, o que dificultou ainda mais a sua compreensão, conforme pode ser observado num trecho extraído de um dos programas:

```
kin.TrigoCalc(q[0], q[1], trigo);
kin.JakobiCalc(trigo, jak);
kin.SpeedCarthToRobot(sollspeed, jak, trigo, vq);
FOR i := 0 TO 3 DO
                (* Test auf maxq und minq mit Zulassen des Rausfahrens *)
        outOfRange :=  outOfRange OR ((q[i]<minq[i])&(vq[i]<0.0)) OR
((q[i]>maxq[i])&(vq[i]>0.0));
END;
outOfRange := outOfRange OR
                (* Test auf NearSingular mit Zulassen des Rausfahrens *)
```

Durante esta parte do trabalho foi desenvolvida uma apostila [7] para auxílio aos usuários do XOberon. Esta apostila contém as definições dos módulos mais

utilizados nos programas que foram estudados. A definição do módulo fornece somente a listagem dos objetos e métodos deste módulo, mas não fornece informações sobre quais as funções disponíveis e como utilizá-las. No apêndice desta mesma pode ser encontrada uma listagem dos erros de compilação de programas.

Agora é necessário fazer um apanhado geral sobre todo o material pesquisado e discutir sobre a implementação da técnica de planejamento de trajetória encontrada em linguagem XOberon.

2.5 Discussão da Solução Encontrada

Nesta seção é analisada a viabilidade de implementação do módulo gerador de trajetórias na linguagem XOberon utilizando a metodologia proposta anteriormente. São vistos principalmente as dificuldades encontradas para atender todas as especificações propostas no início do trabalho.

Conforme foi visto na seção 2.3.4, decidiu-se que a melhor técnica a ser utilizada para o desenvolvimento deste gerador de trajetórias em particular, é a que utiliza o conceito de interpolação polinomial, onde polinômios de 5ª. ordem são acoplados para formar o caminho desejado. Esta técnica é desenvolvida em termos de espaço de juntas.

Como especificação inicial do caminho, é necessário que o usuário defina toda a sequência de pontos deste caminho, bem como o tempo para o efetuador-final atingir cada um destes pontos. Além disto, o usuário deve especificar a velocidade e aceleração angular em cada ponto do caminho. Caso desejado, a escolha destes valores de velocidade e aceleração poderia ser feita pelo próprio sistema, conforme visto na seção 2.3.2.3.

O sistema de geração dos dados também deve ser capaz de verificar se o limite de velocidade angular em cada uma das juntas não extrapola o máximo permitido. Caso isto ocorra, ou o sistema deve ser capaz de resolver o problema através de alguma lógica interna, o que resultaria no aumento do tempo necessário para cumprir aquela parte do caminho, ou ele deve informar ao usuário sobre o problema e solicitar uma nova definição dos parâmetros temporais. Este último caso representa uma definição *off-line* dos dados da trajetória.

Observa-se que o sistema deve possuir funções matemáticas para o cálculo dos parâmetros da função de cada um dos segmentos e para o cálculo da cinemática direta e inversa. O sistema deve ainda apresentar uma interface "amigável" que, além de permitir ao usuário a inserção dos dados iniciais que irão definir a trajetória, permita também a apresentação dos resultados obtidos e/ou solicite novos valores para nova especificação, no caso de extrapolação de limites de velocidade.

A partir da definição das características gerais que o sistema deve apresentar, foi constatado que a sua implementação na linguagem XOberon não é nada trivial. Como visto na seção anterior, a dificuldade de utilizar tal linguagem tornou-se ainda mais evidente pelo fato de se necessitar fazer algo diferente do que já existia em programas conhecidos, como por exemplo, plotar resultados e abrir janelas de

comunicação com o usuário. Estas funções, aparentemente simples e que são de fácil desenvolvimento em linguagens mais conhecidas, não são tão simples de serem implementadas em XOberon. A falta de material de consulta para o estudo apropriado da linguagem, aliado ao fato deste ser um trabalho pioneiro neste laboratório utilizando esta linguagem, praticamente inviabilizou o seu uso. Caso se prosseguisse com a intenção de utilizá-la, certamente ter-se-ia que abrir mão de algumas exigências propostas, em sua maioria com relação ao interfaceamento entre usuário e sistema. Além disso, não se sabia estimar ao certo quanto tempo seria necessário para implementar uma solução menos exigente que a original. Do mesmo modo, já tinha sido gasto um tempo considerável até esta parte do projeto e não podia ser esquecido a necessidade de produzir algo que pudesse ser avaliado e aceito como um projeto de conclusão de curso.

Como o grande problema desta forma de solução é a utilização da linguagem XOberon, uma idéia alternativa que surge é a de empregar uma outra linguagem para desenvolver a parte onde encontra-se maior dificuldade em utilizar o XOberon. Assim, esta linguagem pode ser utilizada para implementar um *software* responsável pela geração *off-line* dos dados da trajetória e um outro *software*, implementado em linguagem XOberon, pode ser responsável em enviar estes dados ao sistema de controle do robô. Com esta nova proposta, vê-se a possibilidade de alcançar todas as exigências iniciais que foram impostas e ainda adicionar outras opções que podem melhorar o interfaceamento entre usuário e sistema, além de facilitar a análise e correção dos resultados encontrados para a trajetória, conforme é melhor explicado na próxima seção.

Vale lembrar novamente que a solução inicial utilizando o XOberon para a implementação seria muito limitada, devido a necessidade de negligenciar algumas das propostas iniciais para a conclusão deste trabalho. Além disso, mesmo diminuindo algumas destas exigências, ainda assim não se tinha a certeza de que era possível implementá-las em tempo hábil. Desta forma, apesar de não abandonar totalmente àquela idéia, vê-se que a nova solução é muito "tentadora" e merece pelo menos que seja investido algum tempo para analisá-la em maiores detalhes.

3. A Restruturação do Trabalho: Nova Proposta de Solução

A principal novidade a ser observada é a idéia de desenvolver o módulo gerador de trajetórias em duas etapas distintas. A primeira parte utiliza uma programação *off-line* para geração dos dados da trajetória, utilizando para isto uma linguagem conhecida que possua funções matemáticas que permitam efetuar os cálculos de maneira simples e satisfatória, e ainda possua recursos apropriados para a análise dos resultados encontrados. De posse destes dados, um *software* desenvolvido em linguagem XOberon é utilizado para fazer a leitura dos mesmos e após, enviá-los ao sistema de controle do manipulador.

A linguagem escolhida para o desenvolvimento do *software* gerador de dados foi o Matlab, pois ele é um poderoso ambiente para álgebra linear e representação gráfica que está disponível para a maioria dos computadores atuais. O seu corpo funcional pode ser estendido através da utilização de várias ferramentas de aplicação, chamadas *"Toolboxes"*. Para o desenvolvimento deste trabalho, está-se particularmente interessado nos *Toolboxes* de Robótica e *Splines*. Estes possuem funções específicas que podem ser utilizadas para a geração de dados para trajetórias no espaço e são a grande contribuição que o Matlab pode promover para o desenvolvimento desta parte do projeto.

A possibilidade de utilizar uma linguagem conhecida que possa ser empregada tanto para geração de dados como para o desenvolvimento da maior parte do interfaceamento com o usuário, também é um fator de grande importância. A geração de dados envolve cálculos e análises matemáticas que são tidos como a parte mais trabalhosa do projeto, já que o envio destes dados pode ser feito utilizando funções já desenvolvidas anteriormente. Com relação ao interfaceamento entre sistema e usuário, vê-se que o problema de restringir algumas especificações para alcançar um resultado ficou assim resolvido.

A programação *off-line* oferece a vantagem de evitar os cálculos dos dados da trajetória em tempo real, que podem inviabilizar o movimento. Isto ocorre caso os dados referentes à próxima atualização do caminho não estejam disponíveis para o envio ao controlador do robô. Outra vantagem é a de poder analisar os dados antes que estes sejam utilizados pelo sistema de controle. Desta forma, pode-se impedir que dados que ultrapassem os limites permitidos danifiquem os componentes mecânicos do robô. Além disto a programação *off-line* permite que a técnica utilizada na geração de trajetórias possa ser definida no espaço cartesiano, pois caso ocorra algum problema com a trajetória (seção 2.3.4), esta pode ser modificada antes da execução do movimento.

A parte final do gerador de trajetórias é desenvolvida em linguagem XOberon. A nível de implementação, e não de importância, esta é a parte mais simples e a que menos exige do usuário. Nesta fase, a trajetória já deve estar definida e o usuário deve apenas executar o programa para que os dados sejam lidos e enviados ao controlador do robô.

Devido a estas simplificações, também estima-se uma diminuição no tempo necessário para o desenvolvimento do projeto. Conforme já havia sido comentado, os prazos para a conclusão de algumas partes do trabalho haviam sido extrapolados e

estávamos bastante atrasados. Com isto, não se esperava adiantar a conclusão do projeto, e sim terminá-lo no tempo previsto.

3.1 A Linguagem Matlab e seus *Toolboxes*

Conforme visto anteriormente, a grande vantagem em utilizar o Matlab é que ele dispõe de um poderoso ambiente para álgebra linear e representação gráfica, proporcionando meios para criar uma interface mais amigável ao usuário, além de métodos mais eficazes para a avaliação dos resultados obtidos, isto de uma forma mais eficiente e completa, e de maneira bem menos complicada, comparando-se ao caso de usar o sistema XOberon para esta tarefa. A seguir, são apresentadas as ferramentas de aplicação do Matlab que são estudadas antes de partir para a geração dos dados da trajetória.

3.1.1 Os *Toolboxes* do Matlab

O *Toolbox* de Robótica provê muitas funções que são utilizadas em robótica, tais como na área de cinemática, dinâmica e geração de trajetória. Combinado com o ambiente interativo do Matlab e suas poderosas funções gráficas, ele provê uma grande ferramenta de simulação e análise em robótica.

O *Toolbox* é baseado em um método geral de representação da cinemática e dinâmica de manipuladores de ligações seriais, através de matrizes de descrição. Elas consistem, no caso mais simples, nos parâmetros de Denavit-Hartenberg de um robô e podem ser criadas pelo usuário para um manipulador de ligações seriais. A descrição do manipulador pode ser mais elaborada, para incluir a inércia dos componentes do motor e dos elos do braço do robô, além dos parâmetros de fricção, entre outros..

O *Toolbox Spline* torna possível criar e trabalhar com curvas e funções polinomiais em um ambiente conveniente que o Matlab provê [2].

Polinômios são funções aproximadas escolhidas para representar uma função "suave". Um exemplo clássico é a série de Taylor truncada:

$$\sum_{i=0}^{n} \frac{(x-a)^i}{i! D^i f(a)}$$

ela provê uma aproximação satisfatória para f(x) se f é suficientemente suave e x está suficientemente próximo de a. Mas se uma função a ser aproximada tem um intervalo muito grande, o grau do polinômio aproximado será muito grande. A alternativa é subdividir este intervalo de aproximação em intervalos suficientemente pequenos:

$$[a..b] \rightarrow [a = \xi_1 < \xi_2 < ... < \xi_{l+1} = b]$$

em cujos intervalos pode-se calcular um polinômio suave de ordem suficientemente pequena para se encontrar uma boa aproximação de $\underline{f}$. Isto pode ser feito de tal modo que os polinômios de cada intervalo podem ser acoplados de maneira suave e com derivadas contínuas. Cada polinômio suave que compõe a função aproximada é chamado de *spline*.

O uso típico para este *toolbox* envolve a construção e subseqüente uso de aproximações polinomiais. Na situação mais simples, é dado um conjunto de pontos (t_i, y_i) e procura-se por uma função polinomial que satisfaça $f(t_i)=y_i$, ou o mais próximo possível. Uma função aproximada pode ser descrita de diferentes maneiras implícitas, como por exemplo a solução de uma equação diferencial ou integral. Em outro tipo de especificação, poderia desejar-se construir uma curva *spline* cuja localização exata é menos importante que a forma geral da curva.

Este *Toolbox* possui atualmente duas formas para representar uma *spline*. Uma delas, a *B-form*, é usada durante a construção da *spline* utilizando-se uma sequência de pontos como parâmetro. A outra forma de representação é a *pp-form*, que é utilizada para avaliar a *spline* já construída em intervalos pré-definidos. Por exemplo, para encontrar uma função que represente um caminho utilizando este *Toolbox*, inicialmente é necessário definir um conjunto de pontos para criar a spline na forma *B* e após convertê-la na forma *pp*. Nesta forma, a *spline* permite que pontos do caminho sejam coletados e armazenados em variáveis de posição, como é utilizado no desenvolvimento do programa de geração de dados, apresentado a seguir.

3.1.2 O *Software* Gerador de Dados

A idéia principal para esta parte é criar um ambiente com interface amigável ao usuário, de modo que este possa criar uma trajetória espacial sem que precise conhecer a fundo as características do robô, como por exemplo, saber se o ponto escolhido está dentro ou fora da área de trabalho do robô.

Para proporcionar esta interface amigável criou-se uma figura no plano XY com o desenho da área de trabalho do robô (fig. 3.1). A região entre as duas circunferências concêntricas representa a área de trabalho do robô no plano XY. Assim, o usuário utiliza o *mouse* para indicar os pontos da trajetória que o robô deve percorrer. Se o usuário acidentalmente escolhe um ponto fora da área de trabalho do robô, este ponto é negligenciado pelo programa, ou seja, o ponto é descartado.

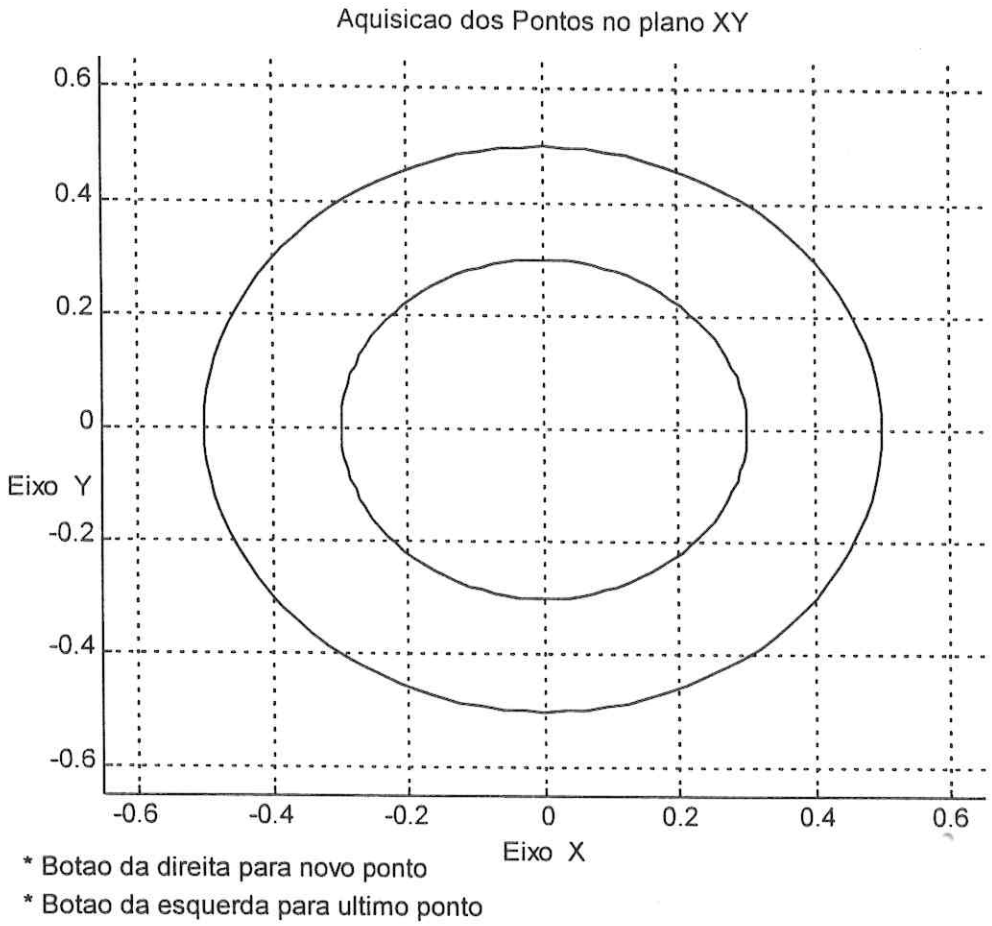


Figura 3.1 - Área de Trabalho do robô Inter no plano XY

Outro problema que pode ser evitado com a utilização desta interface é com relação às áreas que não podem ser atingidas a partir de determinadas posições do braço do robô. O robô possui duas áreas que uma vez que o manipulador está em uma delas, ele não pode alcançar a outra. Como visto na seção 2.1, o robô pode estar em sua configuração à esquerda ou à direita, dependendo da área em que estiver. Quando o usuário escolhe um ponto em uma destas áreas, a outra área é automaticamente hachurada, indicando uma área proibida para a escolha do novo ponto. Além disto, é exibida uma mensagem informando em qual configuração se encontra o robô (fig. 3.2).

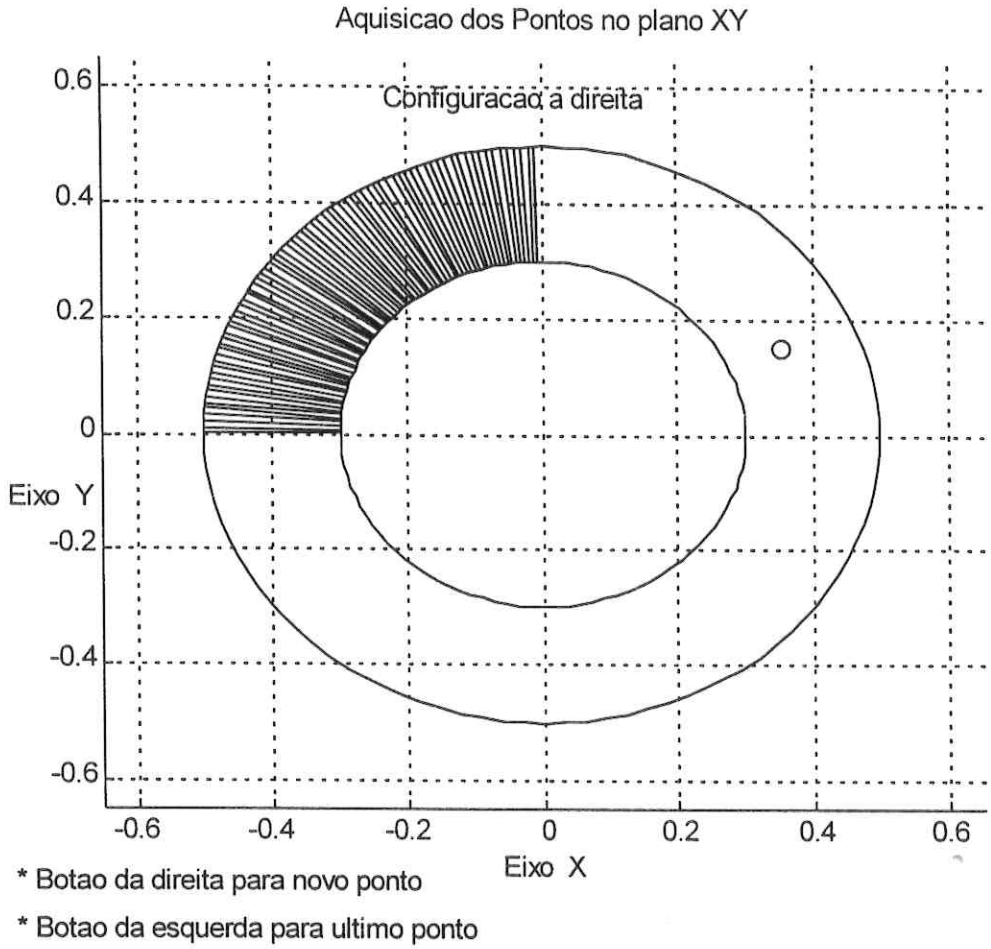


Figura 3.2 - Área de Trabalho do robô Inter na configuração à direita

Após a escolha dos pontos da trajetória no plano XY, é a vez do usuário escolher os pontos no eixo Z. Para isto uma nova janela é criada, com a área de trabalho do robô no eixo Z (fig. 3.3). O usuário só se preocupa com a posição vertical que é representada pelo eixo das ordenadas, já que o eixo das abcissas representa o número de pontos escolhidos na fase anterior.

Os pontos inicial e final e os pontos de passagem (*via-points*) são armazenados e utilizados para gerar a *spline* que irá representar o caminho entre os pontos. A *spline* gerada deve garantir que o ponto final é alcançado e que a trajetória passa próximo dos pontos intermediários escolhidos.

Esta *spline* é avaliada a cada pequeno intervalo de tempo para que sejam armazenadas as posições cartesianas desejadas do manipulador. O número de pontos coletados da spline depende do tempo que foi especificado para cumprir a trajetória. A cada segundo do movimento são coletados 1.000 (mil) pontos, ou seja, para uma

trajetória a ser realizada em 5 segundos são coletados 5.000 pontos. Esta "taxa de amostragem" (1.000 pontos/segundo) é definida desta maneira por duas razões:

- Um número suficiente de pontos são coletados para representar a trajetória;
- A taxa de atualização de caminho do sistema de controle do robô está definida como sendo de 1ms (seção 3.3). Então, se o controlador recebe novos pontos para atualizar a trajetória a cada 1ms significa que, para executar um movimento de 5s, ele necessita de 5.000 pontos, o que está de acordo com o definido acima.

Através da primeira e segunda derivadas da *spline*, encontra-se funções de velocidade e aceleração, respectivamente. Amostrando estas funções - utilizando o mesmo intervalo de tempo usado para amostrar a posição -, encontra-se a velocidade e aceleração desejadas do manipulador no Espaço Cartesiano.

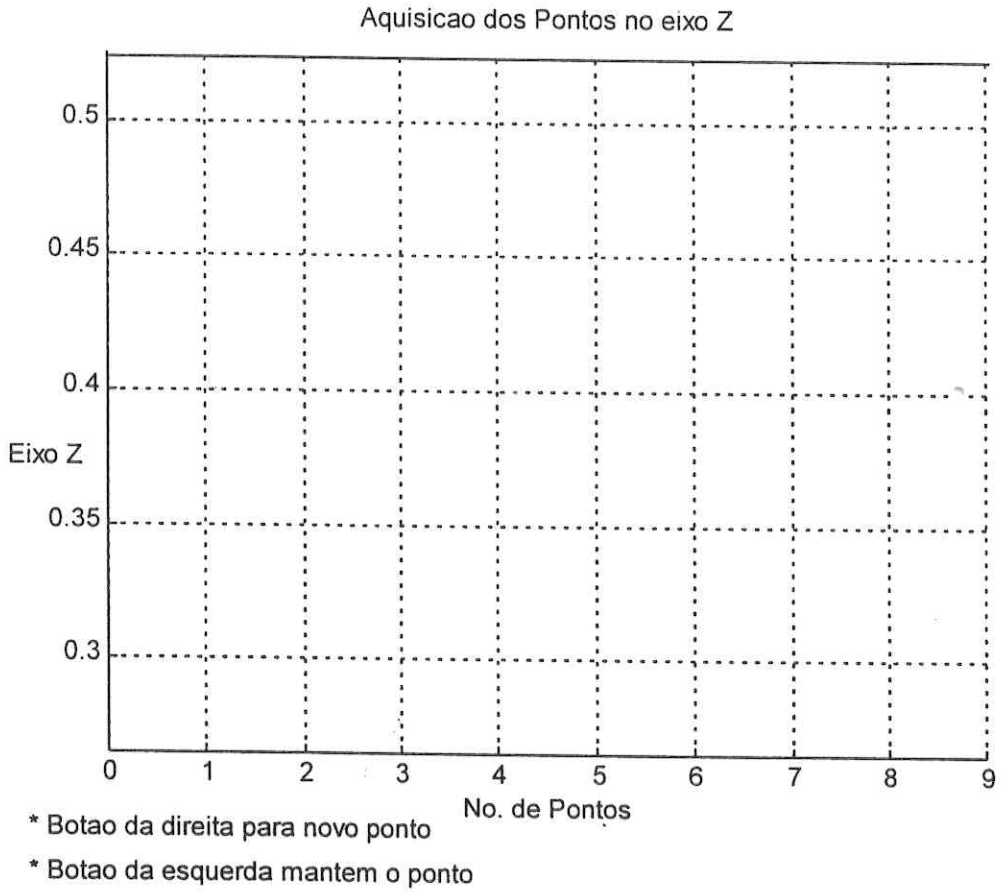


Figura 3.3 - Área de Trabalho do robô Inter no Eixo Z

De posse dos valores de posição, velocidade e aceleração cartesianos, é feito a transformação destes em valores de deslocamento, velocidade e aceleração angulares, respectivamente, ou seja, é necessário fazer a cinemática inversa para transformar os valores do espaço cartesiano para valores no espaço de juntas. Para isto, são usadas as equações mencionadas na seção 2.1.1.

Os valores calculados são plotados para verificar se estão dentro dos limites especificados. Neste momento, os valores máximos de velocidade e aceleração angular atingidos são comparados com os máximos permitidos. Caso estes sejam extrapolados, a trajetória pretendida deve ser recalculada para que a estrutura do robô não seja danificada (Tab. 3.1). Esta parte é de extrema importância, já que, como visto anteriormente, uma mesma posição cartesiana pode ser atingida através de diferentes deslocamentos e velocidades angulares, e isto não era verificado no módulo gerador de trajetória anterior.

	Velocidade Máxima	Aceleração Máxima
Junta 0	±3.0 rad/s	±80 rad/s^2
Junta 1	±3.0 rad/s	±100 rad/s^2
Junta 2	±0.888 m/s	±3.0 m/s^2

Tabela 3.1 - Os Limites de Velocidade e Aceleração Angular nas juntas do robô Inter

Os valores de deslocamento, velocidade e aceleração angulares calculados e verificados são armazenados em uma matriz n x m, onde n representa o número de pontos amostrados que compõem a trajetória e m são as nove colunas dos valores calculados. Esta matriz é salva em um arquivo de dados que mais tarde pode ser transmitido à memória do robô. Antes de efetuar esta transmissão, é necessário fazer um teste de compatibilidade entre os sistemas MS/DOS e XOberon, para verificar se os parâmetros da trajetória que estão no arquivo gerado no Matlab podem ser realmente utilizados pelo sistema de controle do manipulador, conforme é explicado a seguir.

Um importante ponto a ser ressaltado é que a geração da trajetória é feita no espaço cartesiano e não no espaço de juntas. Conforme discutido anteriormentea geração em espaço cartesiano pode gerar diversos problemas, mas estes podem ser minimizados com este tipo de solução. Isto porque os dados da trajetória são obtidos através de uma programação é *off-line* e todos os valores de posição, velocidade e aceleração são verificados antes de enviá-los ao sistema de controle. O planejamento de trajetórias é desenvolvido em espaço cartesiano neste trabalho, principalmente por ter sido considerado que a utilização deste esquema facilitaria o desenvolvimento do *software* gerador de dados. No anexo A est

O anexo A apresenta a descrição deste software e são feitos alguns comentários de sua estrutura.

3.2 Leitura e Conversão de Dados

Antes de implementar o *software* específico para a leitura e conversão do arquivo de dados gerado no Matlab, é necessário fazer um estudo sobre a compatibilidade entre arquivos gerados neste sistema e os gerados no XOberon.

3.2.1 Teste de Leitura e Conversão de Números

O programa teste proposto deve carregar e ler um arquivo que contenha blocos de números de dimensões variáveis. Este arquivo é gerado por outro *software*, que não o próprio XOberon, para que seja analisada a compatibilidade entre os sistemas. Durante a leitura, os números devem ser armazenados em variáveis para que, ao final da leitura do arquivo, possam ser exibidos em uma janela de comunicação com o usuário, no ambiente do XOberon.

A compatibilidade entre arquivos gerados no sistema XOberon e os gerados em sistemas destes dois sistemas MS/DOS é alcançada a nível de arquivos no formato ASCII, ou seja, a manipulação de arquivos deve ser feita neste formato. A principal diferença que deve ser observada é que para um arquivo texto do XOberon ser exibido pelo editor do DOS, é necessário que os atributos dos caracteres (fonte, cor e *offset* vertical) sejam eliminados. O comando *Miscellaneous.Cleanup* pode ser usado para este propósito. Para que um arquivo do MS/DOS seja exibido no ambiente XOberon é necessário eliminar caracteres especiais que não podem ser exibidos neste sistema. Neste caso, um filtro deve ser adicionado ao código do programa para eliminar os caracteres incompatíveis [5,6].

O arquivo teste que é gerado no editor de texto do DOS e então transferido à memória do robô, no armário de controle, para que seja possível exibir o mesmo em uma janela de comunicação no ambiente XOberon. A transferência é feita por um *software* chamado TFTP (*"Trivial File Transfer Protocol"*), que o envia o arquivo em formato ASCII através da rede de comunicação existente entre o computador com o ambiente XOberon e o armário de controle.

Após a transferência do arquivo teste, várias funções do XOberon foram utilizadas e bastante tempo gasto na tentativa de desenvolver um programa capaz de ler um bloco de caracteres deste arquivo e transformá-lo em um número real. Somente após a descoberta de um módulo específico para transformação de "*Strings*" (blocos de caracteres) em números reais é que foi conseguido algum progresso. O problema agora é que a função empregada não lia corretamente os números, dividindo-os ou agrupando-os erroneamente, formando números diferentes dos propostos. Para contornar este problema, foi necessário analisar o arquivo proposto em seu formato hexadecimal para verificar detalhadamente a sua representação. Nesta parte foi descoberta uma diferença entre os dois sistemas, MS/DOS e XOberon, em relação a representação de caracteres de quebra de linha e fim de arquivo. No MS/DOS, a quebra de linha é representada pelos hexadecimais 0DX-0AX e o fim de arquivo por 0DX-0AX-1AX. No XOberon a quebra de linha é representada somente por 0DX e o

fim de arquivo não é representado. Era esta quantidade a mais de caracteres que estava "confundindo" a função conversora.

Foi adicionado ao programa um filtro que somente considera caracteres específicos a leitura de números, ou seja, os caracteres de zero a nove, os sinais (+ ou -), ponto, vírgula e a letra "e" (maiúscula e minúscula) que pode formar números exponenciais. Todos os outros caracteres são considerados como o final do número e a conversão é efetuada. Um caractere especial é adicionado pelo filtro para representar o final do arquivo. Os números convertidos são verificados e conferidos para ver se realmente representavam números reais e após são enviados para serem exibidos na tela.

Por se tratar de um simples teste, provavelmente esta não foi a parte mais importante do projeto, mas talvez possa ser considerada a mais trabalhosa e que exigiu maior dedicação e tempo devido as dificuldades encontradas durante o seu desenvolvimento. É por esta razão que foi feita uma descrição tão detalhada para esta "pequena" parte do projeto. A viabilidade de ler e converter um arquivo de dados foi confirmada e agora é necessário efetuar algumas modificações no mesmo, como é apresentado a seguir.

3.2.2 O *Software* Específico para Leitura e Conversão

Nesta seção é feita uma adaptação do *software* anteriormente desenvolvido para que o mesmo possa ser especificamente utilizado na leitura e conversão de um arquivo de dados gerados no Matlab. Antes é necessário definir como deve ser a forma exata deste arquivo.

Devido ao fato de o arquivo ser transferido como sendo uma única linha de dados, é necessário ter a preocupação adicional de conhecer a posição exata dos números que representam as posições, velocidade, e aceleração. Então, definiu-se que o arquivo de dados deve ser de tal maneira que cada uma de suas colunas representem a posição, velocidade e aceleração das juntas do robô. Deste modo, a cada linha tem-se um novo passo de atualização de caminho, ou seja, cada linha contém as posições, velocidades e acelerações atuais de cada junta (fig. 3.4).

$$\begin{matrix} \theta_{01} & \theta_{02} & \theta_{03} & \dot{\theta}_{01} & \cdots & \ddot{\theta}_{03} \\ \theta_{11} & \theta_{12} & \theta_{13} & \dot{\theta}_{11} & \cdots & \ddot{\theta}_{13} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ \theta_{n1} & \theta_{n2} & \theta_{n3} & \dot{\theta}_{n1} & \cdots & \ddot{\theta}_{n3} \end{matrix}$$

Figura 3.4 - Arquivo Idealizado

Além disto, definiu-se que cada número do arquivo seria separado por um caractere espaço ou "*TAB*" e ao final de cada linha haveria um "*RETURN*" e "*LINE FEED*".

O *software* anterior é então adaptado para ler e converter blocos de nove números, ou seja, são pegos inicialmente todas as posições, velocidades e acelerações das juntas que originalmente representam uma linha no arquivo. Este bloco de números é, ou transferido diretamente para o controlador do robô, ou armazenado em outras variáveis - isto é discutido na próxima seção. De qualquer maneira, o processo de leitura e conversão prossegue até que o caracter que representa o final de arquivo seja identificado pelo programa.

A leitura pode ser interrompida caso seja detectado algum problema com o arquivo de dados. Um deles seria a não localização do arquivo. Se isto acontece, uma mensagem de "arquivo não encontrado" é mostrada. Outro tipo de erro é o de encontrar o arquivo, mas este estar, ou vazio, ou com uma de suas linhas com menos de nove números, o que evidencia uma falha no mesmo. Caso isto ocorra, uma mensagem apropriada é emitida e a execução do programa é igualmente interrompida.

Caso não ocorra nenhum erro na leitura/conversão, uma mensagem de fim de leitura de arquivo é mostrada e os dados são exibidos na tela. Mais tarde, em vez de enviar os dados para serem exibidos, eles são enviados ao controlador do robô, a fim de serem tratados pelo sistema de controle que por sua vez acionará os atuadores para movimentar o manipulador.

3.3 O *Software* Adaptado: O Módulo Gerador de Trajetórias

O *software* desenvolvido na seção anterior é responsável pela leitura, conversão e armazenagem dos dados de posição, velocidade e aceleração de toda a trajetória. Como o arquivo de dados já foi gerado (seção 3.3.2), resta agora adaptar este software para que, em vez de enviar os dados para serem exibidos na tela do computador, estes sejam enviados ao sistema de controle do robô para efetivar o movimento.

Quase toda a estrutura do programa de leitura/conversão pôde ser aproveitada. Porém novas funções foram adicionadas, principalmente para garantir a segurança na utilização do controlador do robô. Desta forma, após a leitura e conversão de todos os dados da trajetória, são verificados diversas variáveis de estado, onde as principais são:

- Verificação se o módulo de controle está livre, ou sendo utilizado por outra tarefa;
- Verificação se o efetuador-final se encontra no ponto inicial da trajetória (Esta questão será trata posteriormente);
- Seleção do tipo de controlador a ser empregado (No presente trabalho, o controlador utilizado é do tipo Proporcional-Derivativo - PD).

O posicionamento do efetuador-final na posição inicial do movimento é efetuado em baixa velocidade (definida previamente) e executado pelo antigo gerador

de trajetórias, que é empregado para movimentos ponto-a-ponto. Isto é realizado desta maneira porque de outra forma é necessário definir previamente uma posição inicial padrão, já que a trajetória é calculada em programação *off-line*. Desta forma, definir uma posição fixa para iniciar o movimento implica em diminuir a flexibilidade do usuário em escolher o ponto inicial da trajetória pretendida. Outro fator que contribui para esta escolha é o fato de iniciar o movimento fora da posição padrão, como é o caso de definir uma segunda trajetória e o ponto final da primeira não coincidir com a posição padrão.

A taxa de atualização de caminho é definida como sendo de 1ms. Isto que dizer que a cada milisegundo novos valores são enviados ao sistema de controle do robô, que irá tentar corrigir o erro de posição existente, neste intervalo de tempo.

Com relação a armazenagem e envio dos dados, haviam duas estratégias possíveis a serem seguidas para o desenvolvimento do gerador de trajetórias. Uma através da minimização do espaço em memória necessário para esta armazenagem e a outra pela minimização do tempo necessário para o envio deles.

A primeira opção foi utilizar a primeira estratégia para evitar a armazenagem intermediária de valores de posição, velocidade e aceleração, mantendo o arquivo original com os dados da trajetória e ir lendo e enviando linha por linha ao sistema de controle até o final do arquivo. O problema é que a leitura, conversão e envio dos dados deve ser efetuada dentro da taxa de atualização de caminho, ou seja, em 1ms e a função de conversão de números não é rápida o suficiente para efetuar esta tarefa, inviabilizando esta forma de solução.

Parte-se, então, para a conversão e armazenagem prévia dos dados, onde cada uma das nove colunas do arquivo são armazenadas em variáveis próprias, antes de serem utilizadas pelo controlador. Isto resolve o problema anterior, mas ainda não foi determinado qual o valor máximo de elementos que uma variável pode conter. No anexo A é apresentado este software e são feitos comentários sobre sua estrutura e utilização.

Resumindo:

- O usuário define o caminho proposto utilizando o mouse para indicar os pontos da trajetória;
- O programa utiliza estes pontos para gerar uma spline que representa o caminho que o manipulador deve seguir;
- Os dados de posição, velocidade e aceleração no espaço cartesiano são calculados a partir desta spline;
- Procede-se então a cinemática inversa para encontrar valores equivalentes de deslocamento, velocidade e aceleração no espaço de juntas;
- Estes valores são verificados e, caso sejam válidos, salvos em um arquivo de dados;
- Este arquivo é enviado à memória do robô através do software TFTP;
- O programa desenvolvido no XOberon lê, converte e envia os dados ao sistema de controle do robô para executar o movimento.

4. Ajuste Geral e Resultados Obtidos

O primeiro teste utilizando o gerador de trajetórias desenvolvido, é feito mediante a realização de um movimento simples. Para isto, é definido um caminho no plano XY, composto por três pequenos segmentos. A figura 4.1 ilustra a área de trabalho do robô, no plano horizontal, onde pode ser observada a trajetória pretendida, composto pelos três segmentos de reta, juntamente com a função interpolada que representa este caminho. Conforme descrito na seção 3.3.2, os pontos escolhidos pelo usuário são inseridos através da utilização do *mouse*, indicando-os diretamente na área de trabalho do robô.

A figura 4.2 ilustra o mesmo caminho descrito anteriormente. Pode ser observado que a função interpolada começa no ponto inicial e atinge o ponto final do caminho proposto. Estas duas afirmações foram verificadas, constatando-se que a posição relativa entre os pontos inicial e final do caminho proposto e da função interpolada que o representa era praticamente nula. Nesta mesma figura, é possível observar que os dois pontos intermediários do caminho são responsáveis por sua forma, devido a função interpolada se aproximar destes dois pontos.

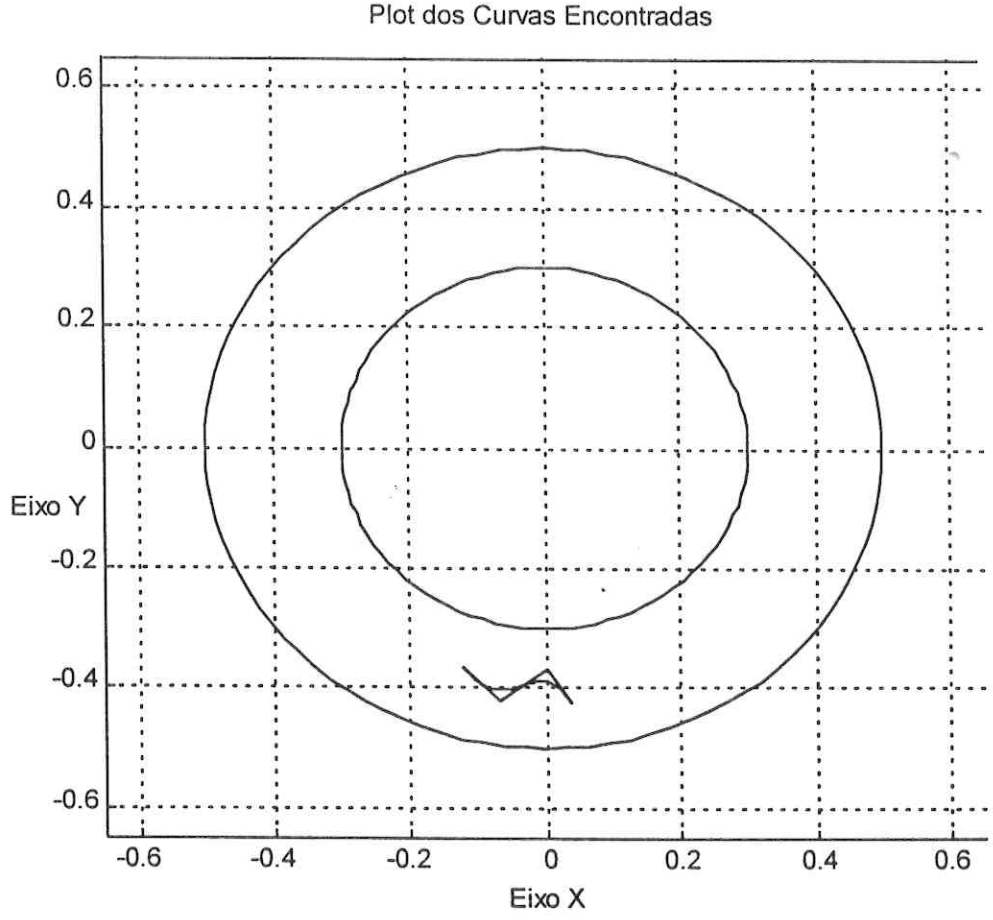


Figura 4.1 - Movimento no plano XY

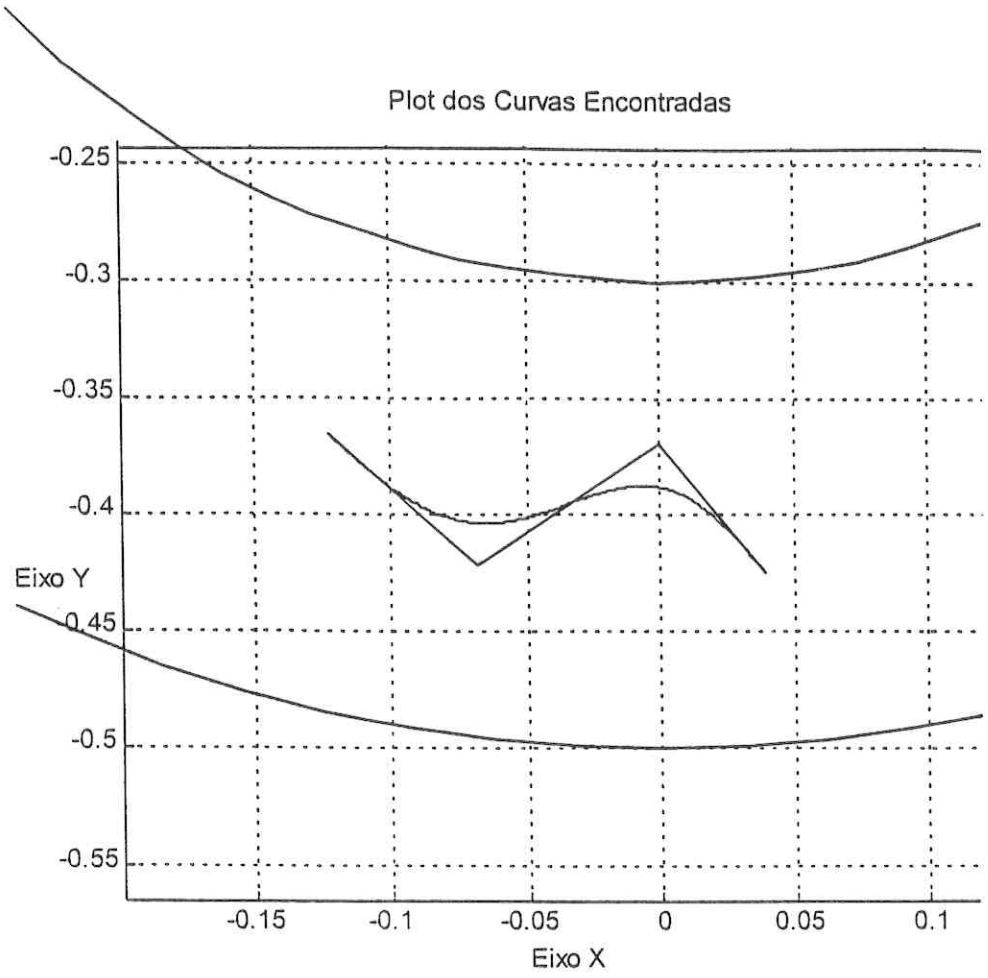


Figura 4.2 - Caminho Proposto e Função Interpolada

O movimento foi realizado sem a ocorrência de nenhum problema e cumpriu a forma do caminho que havia sido definida.

Após este pequeno teste, novos ajustes são efetuados no *software* de geração de dados da trajetória e no desenvolvido em linguagem XOberon, responsável pelo envio destes dados ao sistema de controle do robô. No primeiro, é inserida a área de trabalho do manipulador no eixo Z, que ainda não havia sido criada, para proporcionar o movimento em três dimensões. Também são adicionados novas funções para plotar os dados da trajetória, bem como a representação de sua curva no espaço. No *software* de leitura e envio dos dados, é ajustada a capacidade de armazenamento dos valores de posição, velocidade e aceleração angular da trajetória e feita uma melhor documentação do mesmo, de modo a facilitar o seu entendimento e futuras correções.

A figura 4.3 ilustra o segundo teste realizado, onde pode ser observada a função interpolada do caminho pretendido no espaço cartesiano, representada em três dimensões. Esta função representa um caminho através de uma elipsóide, com a intenção de testar o movimento do manipulador na execução de uma trajetória mais complexa. A construção desta curva é feita a partir de uma circunferência aproximada no plano horizontal e uma reta no plano vertical.

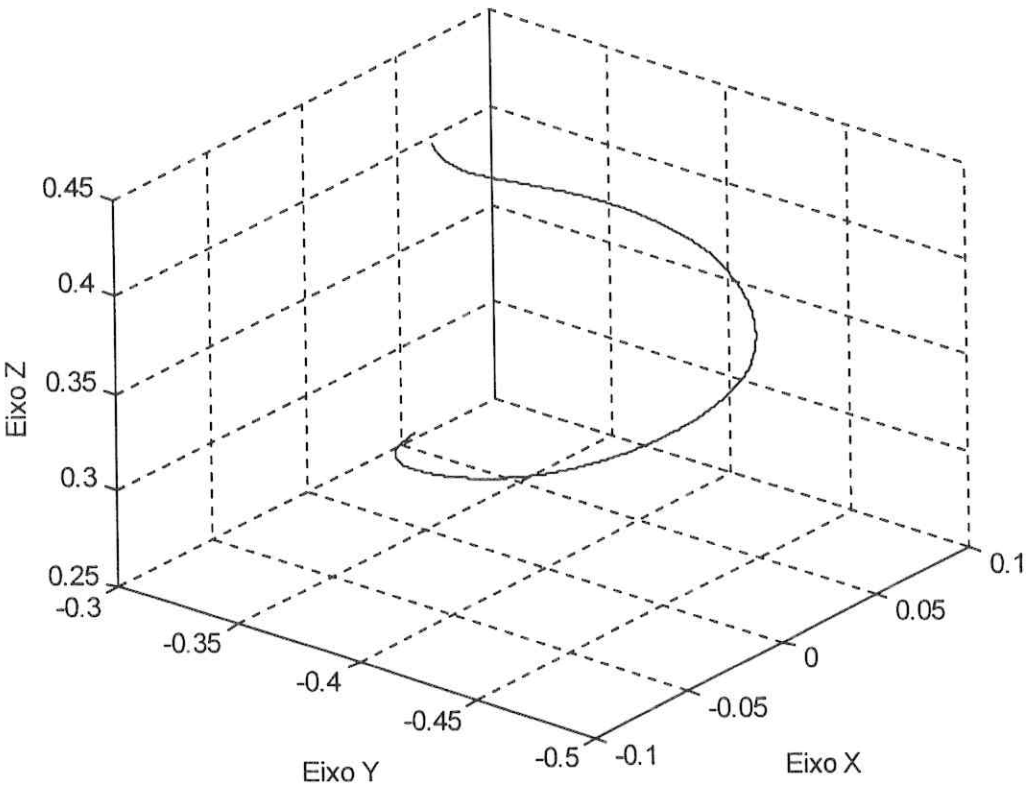


Figura 4.3 - Representação em três dimensões da trajetória

A figura 4.4 ilustra o caminho composto por seus 8 segmentos de reta. Este caminho é representado no eixo Z para ilustrar a área de trabalho no plano vertical que fora posteriormente adicionada, como havia sido comentado acima.

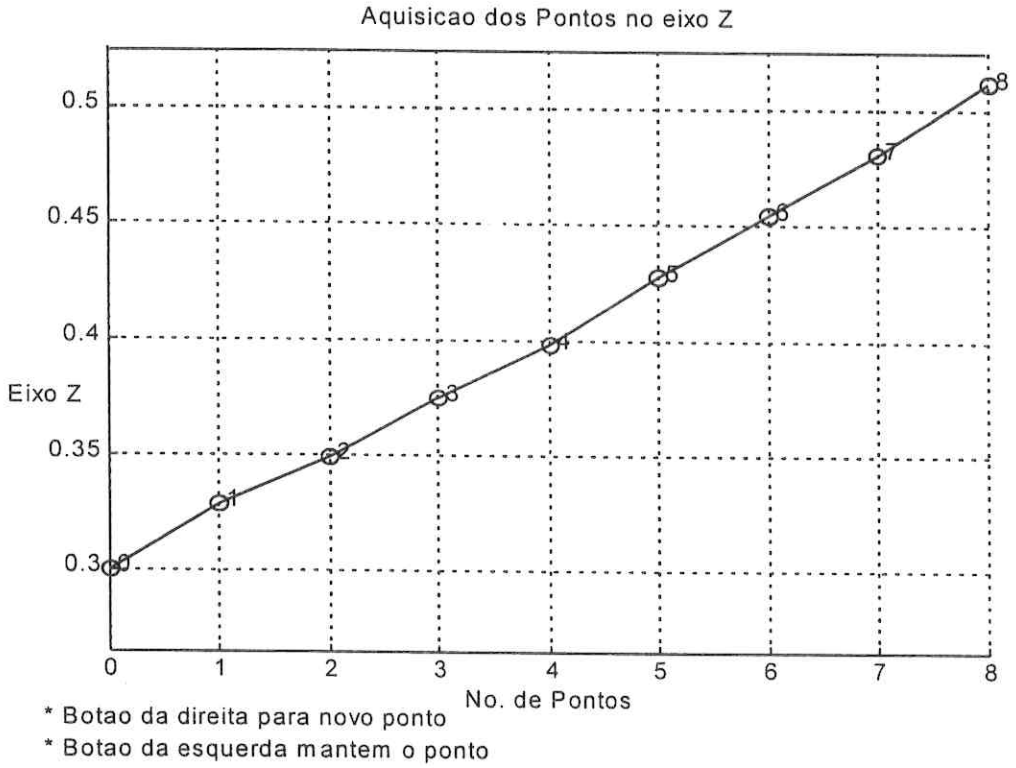


Figura 4.4 - Representação do caminho no plano vertical

As figuras 4.5, 4.6 e 4.7 ilustram a história temporal da posição, velocidade e aceleração angular da trajetória pretendida, respectivamente.

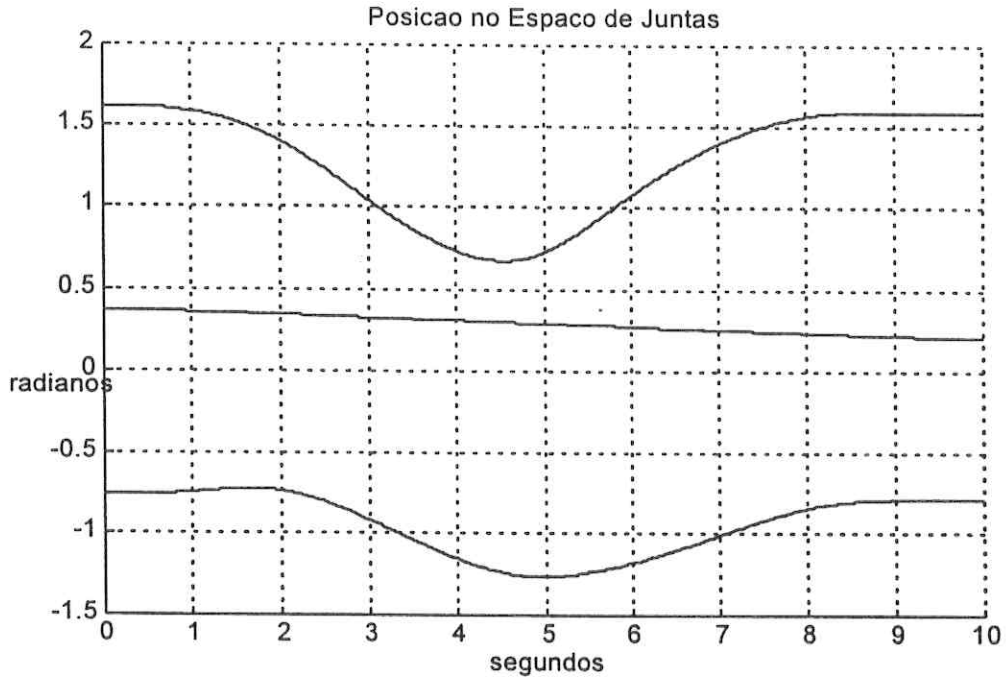


Figura 4.5 - História temporal da posição angular

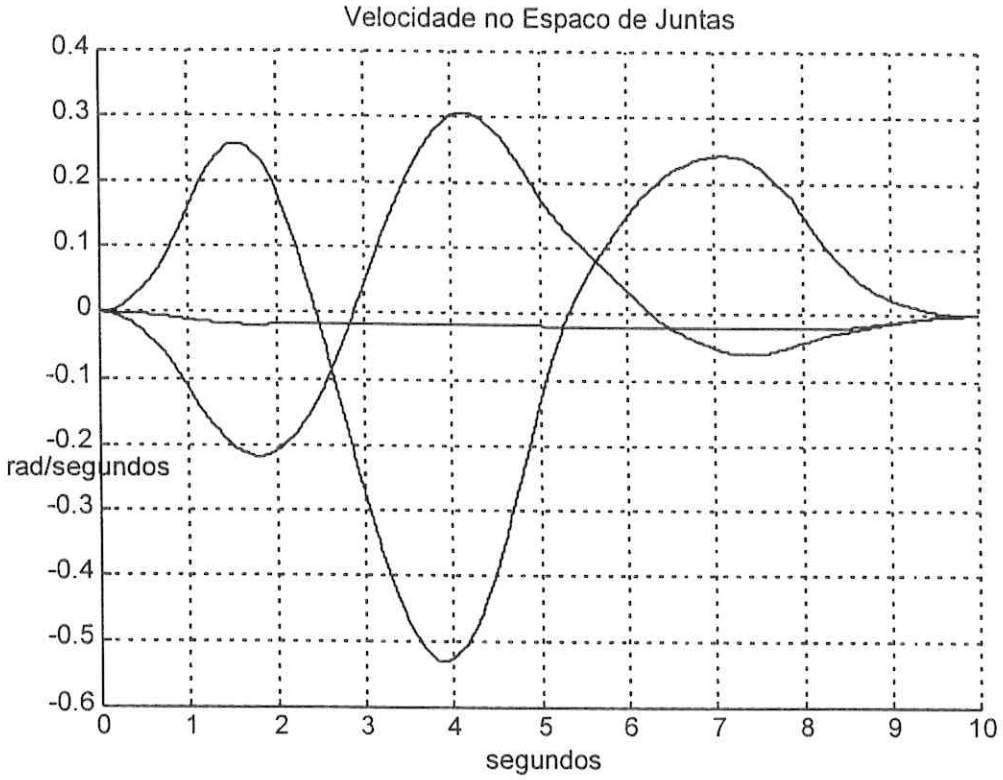


Figura 4.5 - História temporal da posição angular

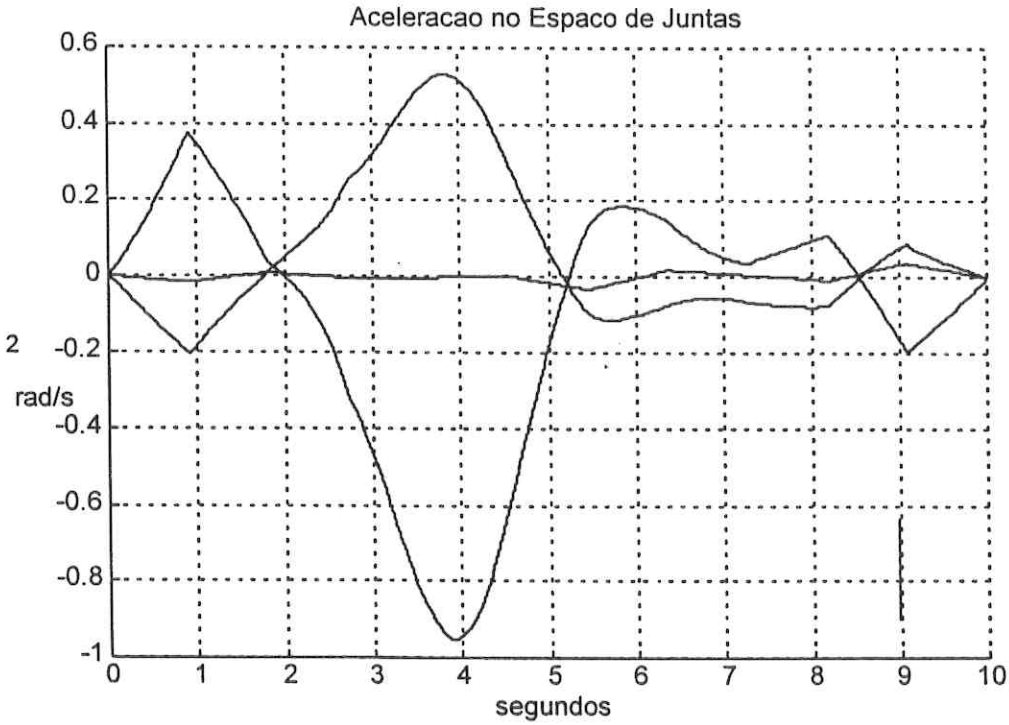


Figura 4.5 - História temporal da posição angular

Comparando os gráficos obtidos com os valores constantes na tabela 3.1, pode ser observado que os limites máximos de velocidade e aceleração não são extrapolados e que, desta forma, o movimento pode ser executado com segurança, pelo menos até este ponto da geração dos dados. As curvas de velocidade e aceleração apresentam a continuidade desejada para a realização do movimento. Também pode ser observado que o perfil de aceleração apresenta uma forma suave no tempo, cumprindo uma das exigências iniciais impostas.

Após verificar todos os dados referentes a trajetória, termina a parte da programação *off-line* e o arquivo com estes dados é enviado à memória do robô para ser usado para efetivar o movimento, que foi realizado com sucesso.

5. Conclusão

Conforme discutido ao longo do trabalho e em particular no capítulo quatro, este módulo de geração de trajetórias pode ser empregado no planejamento e execução de caminhos no espaço com pontos intermediários. Esta é a principal característica a ser ressaltada, pois a meta era empregar este novo gerador de trajetórias em substituição ao atualmente utilizado, que somente permite a definição de caminhos ponto-a-ponto.

Dois outros objetivos secundários também foram atingidos. Um deles se refere a transformação dos valores de velocidade do espaço cartesiano em valores equivalentes no espaço de juntas que ocorre sem nenhum controle ou verificação, caso o caminho seja planejado utilizando o outro gerador de trajetórias. Como visto, o perfil da curva de velocidade angular é verificada e seus valores comparados com os máximos permitidos, antes de efetuar o movimento. O outro objetivo se refere ao perfil da curva de aceleração angular, que era muito acentuada devido a grandes variações nos valores de aceleração. Isto causava desgaste prematuro dos componentes da estrutura mecânica do robô. Conforme visto no capitulo quatro, o gráfico com a história temporal da aceleração não possui variações acentuadas em seus valores, permitindo que este problema seja evitado.

Outro ponto a ser destacado se refere a facilidade alcançada para a definição caminho desejado. Com a utilização de figuras para representar a área de trabalho do robô, atingiu-se uma grande flexibilidade na escolha dos pontos, além de uma melhor visualização da forma do caminho proposto. Isto evita, entre outros fatores, que o usuário defina caminhos fora da área de trabalho do robô, principalmente porque não são aceitos pontos definidos fora desta área.

Em todos os movimentos executados, a trajetória foi planejada no espaço cartesiano e após definida no espaço de juntas através da cinemática inversa, permitindo que os dados de deslocamento, velocidade e aceleração angular pudessem ser diretamente acessados pelo sistema de controle do robô, o qual aciona os atuadores de cada uma de suas juntas para efetuar o movimento.

A forma de programação *off-line* permitiu que as exigências fossem cumpridas, proporcionando maior flexibilidade e segurança na definição dos parâmetros da trajetória. Além disto, esta forma de planejamento evitou que o cálculo da cinemática inversa fosse efetuada em tempo real.

Quanto à possibilidade de continuação deste trabalho, uma primeira sugestão seria desenvolver um *software* para geração dos parâmetros da trajetória em coordenadas de juntas, utilizando para isto os conceitos definidos na seção 3.3.2, e não as funções existentes no Matlab. A principal vantagem desta aplicação seria permitir ao usuário definir as velocidades e acelerações em cada ponto da trajetória, as quais atualmente são geradas a partir da primeira e segunda derivadas dos valores de posição, respectivamente. Outra vantagem seria a de planejar a trajetória em espaço de juntas e evitar os problemas discutidos na seção 2.3.4.

Outra sugestão seria o estudo do comportamento e desempenho do sistema quando na execução do movimento proposto. Isto quer dizer que é necessário desenvolver uma forma de comparar a trajetória idealizada com a trajetória real

executada pelo manipulador. Esta comparação entre valor real e desejado é de grande importância para o auxílio de trabalhos realizados no laboratório de Robótica, como o controle de velocidade, por exemplo.

Outro ponto de interesse é estender esta solução também para o caso da orientação do robô, que não foi implementada neste trabalho. Somente o posicionamento do manipulador foi efetuado, enquanto que a quarta junta do robô, responsável pela orientação, foi mantida constante durante a execução das trajetórias.

Finalmente, apesar da idéia de desenvolver o gerador de trajetórias somente utilizando a linguagem XOberon em sua implementação ter sido abandonada para que o trabalho pudesse ser concluído, talvez este possa ser um outro ponto de interesse. Isto possibilita trabalhar em um único sistema e evita a necessidade de transferência externa dos dados da trajetória à memória do robô.

Bibliografia

[1] **Asada**, Haruhito & Slotine, Jean-Jaques E. *"Robot Analysis and Control"*.1986. John Wiley & Sons, Inc. NY.

[2] **Carl de Boor**. *"Spline Toolbox User's Guide"*.1992. The Mathworks, Inc. Natick, Mass.

[3] **Craig**, J. J. *"Introduction to Robotics, Mecanisms and Control"*.1986. Addison-Wesley, Inc.

[4] **Fernandes**, Clóvis Júnior. *"Definições dos Principais Módulos da Linguagem XOberon"*.1998.Apostila.UFSC.

[5] **Ferreira**, Edson de Paula. *"Robótica Industrial: Aspectos Macroscópicos"*.1987.I EBAI.

[6] **Mössenböck**, H. *"Object-Oriented: Programing in Oberon-2"*.1993.Springer-Verlag. Berlin.

[7] **Reiser**, Martin. *"The Oberon System: User Guide and Programers Manual"*.1991. ACC Press, Inc. NY.

[8] **Rosa**, E. da. *"O Uso de Polinômios Cúbicos de Hermite no Planejamento de Trajetórias de Manipuladores"*.1991.Tese de Doutorado, UFSC.

[9] **Siciliano, Bruno & Sciavicco, Lorenzo.** *"Modeling and Control of Robot Manipulators"*.1996.McGraw Hill Companies, Inc.

[10] **Taylor**, R. H. *"Planning and Execution of Straight Line Manipulator Trajectories"*.1979. IBM J. Research and Development, vol. 23.

[11] **Weihmann**, L. 1997. *"Descrição, Instalação, Programação e Funcionamento de um Robô Manipulador do tipo Scara"*. Dissertação de Mestrado.UFSC.

Apêndice A

Neste apêndice estão descritos os *softwares* implementados durante o desenvolvimento deste projeto.

O software Scara.m

O *software* Scara.m foi desenvolvido no Matlab 5.1 e tem a função de gerar um arquivo de dados de posição, velocidade e aceleração que representam uma trajetória no espaço de trabalho do robô Inter.

Este *software* cria figuras que representam a área de trabalho do robô, onde o usuário pode indicar os pontos desejados do caminho que o manipulador deve seguir. O tempo total da trajetória está fixado em 10 segundos e pode ser modificado alterando a variável "tt".

Os pontos escolhidos servem para construir uma *spline* que representa a trajetória do efetuador-final no espaço cartesiano. A 1ª. e 2ª. derivada desta *spline* representam a velocidade e aceleração cartesiana da trajetória desejada, respectivamente. Estas *splines* são amostradas em pequenos intervalos de tempo para que sejam coletados os dados de posição, velocidade e aceleração da trajetória. A cinemática inversa transforma estes valores em outros equivalentes no espaço de juntas que são salvos em um arquivo de nome "*pontos.mat*". Segue abaixo a descrição deste *software*.

```
%-----------------------------------------
% Programa Gerador de Dados para Trajetorias
% Versao 1.0 17/05/98
% Laboratorio de Automacao Industrial
% Centro Tecnologico - UFSC
%-----------------------------------------

%Este programa coleta pontos do caminho desejado, atraves do uso do
mouse
%e faz uma interpolacao cubica dos pontos transformando-os em uma funcao
%suave. Os pontos, velocidades e aceleracoes gerados sao plotados para
%conferencia e um arquivo (pontos.mat) eh gerado no final com as
posicoes, velo-
%cidades e aceleracoes no Espaco de Juntas. Ainda eh necessario um
ajuste
% nas areas de configuracao a Direita e a Esquerda. Para uso posterior,
sao cal-
%culados vetores com a distancia  total percorrida.

echo off

% Definicao da area de trabalho do robo no plano XY.
cax = newplot;
cla;
prmtrs_rng=[-.65 .65 -.65 .65];
```

```
axis(prmtrs_rng);
grid;
hold on;
ylabel('Eixo  Y');

xlabel('Eixo  X');

title('Aquisicao dos Pontos no plano XY');
text(-0.7758,-0.7374,['* Botao da direita para novo
ponto'],'era','back');
text(-0.7758,-0.7974,['* Botao da esquerda para ultimo
ponto'],'era','back');
xcg=[];xcp=[];ycg=[];ycp=[];
for theta=0:2*pi/100:2*pi;
   xcg=[xcg .5*sin(theta)];
   ycg=[ycg .5*cos(theta)];
   xcp=[xcp .3*sin(theta)];
   ycp=[ycp .3*cos(theta)];
end;
plot(xcg,ycg,'r-');
hold on;
plot(xcp,ycp,'r-');
view(0,90);
x = [];
y = [];
n = 0;

% Loop para aquisicao dos valores de (X,Y).
but = 1;
while but == 1
   [xi,yi,but] = ginput(1);
   fora=sqrt(xi^2+yi^2);
   if (fora>0.5)|(fora<0.3)
      but=1;
   else
      plot(xi,yi,'go','era','back');
            if (yi>0)
            if (atan(xi/yi)<1.57)&(atan(xi/yi)>0)
                  for phi=-1.57:2*pi/300:0;
               line([.3*sin(phi) .5*sin(phi)],[.3*cos(phi)
.5*cos(phi)]);
                  end;
             text(-0.2316,.5740,'Configuracao a direita');
             prim=0;
            elseif (atan(xi/yi)>-1.57)&(atan(xi/yi)<0)
                  for phi=0:2*pi/500:1.57;
               line([.3*sin(phi) .5*sin(phi)],[.3*cos(phi)
.5*cos(phi)]);
                  end;
                  text(-0.2316,.5740,'Configuracao a esquerda');
                  prim=0;
            end;
         end;
      n = n + 1;
      text(xi,yi,[' ' int2str(n-1)],'era','back');
      x = [x; xi];
      y = [y; yi];
      if n > 1
         line([x(n-1) x(n)],[y(n-1) y(n)]);
      end;
   end;
```

```
end;
np=length(x);

% Definicao da area de trabalho do robo no eixo Z.
cax = newplot;
cla;
prmtrs_rng=[0 (np-1) 0.265 0.525];
axis(prmtrs_rng);
grid on;
hold on;
ylabel('Eixo Z');

xlabel('No. de Pontos');

title('Aquisicao dos Pontos no eixo Z');
text(-0.34,0.247,['* Botao da direita para novo ponto'],'era','back');
text(-0.34,0.237,['* Botao da esquerda mantem o ponto'],'era','back');
view(0,90);

% Loop para aquisicao dos valores de Z.
z = [];
pontoz = [];
n = 0; a=0; b=0;
but = 1;
while n < np
   [b,a,but] = ginput(1);
   if (but ~= 1)&(n > 1)
      zi = z(n);
      but = 1;
   else
      zi = a;
      but = 1;
   end;
   pontoz = [pontoz; n];
   n = n + 1;
   plot((n-1),zi,'go','era','back');
   text((n-1),zi,[' ' int2str(n-1)],'era','back');
   z = [z; zi];
   if n>1
      line([(n-2) (n-1)],[z(n-1) z(n)]);
   end;
end;

xp=[x(1,1);x(1,1);x(:,1);x(np,1);x(np,1)];
yp=[y(1,1);y(1,1);y(:,1);y(np,1);y(np,1)];

zp=[];
zp=[z(1,1);z(1,1);z(:,1);z(np,1);z(np,1)];

% Calculo do Vetor Distancia Total a Cada Ponto
np=length(xp);
vetdistT = [];
for i=2:np;
   distTi=sqrt((xp(i,1)-xp(i-1,1))^2+(yp(i,1)-yp(i-1,1))^2+(zp(i,1)-
zp(i-1,1))^2);
   vetdistT = [vetdistT; distTi];
end;

npt=length(vetdistT);
%Calculo do Tempo Total Exigido a cada ponto
%Considerando que se saiba o vetor velocidade cartesiana maxima
```

```
vettempoT = [];
velxyz=8.775;
for i=1:npt;
   tempoTi=abs(vetdistT(i,1))/velxyz;
   vettempoT = [vettempoT; tempoTi];
end;


%Interpolacao dos Pontos da Trajetoria.

%Toolbox B_Spline Aproximation
tt=10; k=4; np=tt*1000;
knots=[0 0 0 0:tt/(length(xp)+k-7):tt tt tt tt];
spx=spmak(knots,xp');
spy=spmak(knots,yp');
spz=spmak(knots,zp');
t=0:tt/(np-1):tt;

%Pontos de posicao da Trajetoria (Espaco Cartesiano - EC)
Xt=fnval(sp2pp(spx),t);
Yt=fnval(sp2pp(spy),t);
Zt=fnval(sp2pp(spz),t);

%Pontos de posicao da Trajetoria (Espaco de Juntas - EJ)
Jp1=[]; angulo1=0;distan=0;
for i=1:np
   distan=Xt(1,i)*Xt(1,i)+Yt(1,i)*Yt(1,i);
   angulo1 = acos((distan-0.125)/0.125);
   Jp1 = [Jp1; angulo1];
end;
Jp0=[]; angulo0=0;den=0;
for i=1:np
   den=.125+.125*cos(Jp1(i,1));
   angulo0 = asin((Yt(1,i)*(0.25+0.25*cos(Jp1(i,1)))-
0.25*Xt(1,i)*sin(Jp1(i,1)))/den);
   Jp0 = [Jp0; angulo0];
end;
Jp2=[];
for i=1:np
   Jp2 = [Jp2; 0.665-Zt(1,i)];
end;

%Pontos de velocidade da Trajetoria - EC
DXt=fnval(fnder(sp2pp(spx)),t);
DYt=fnval(fnder(sp2pp(spy)),t);
DZt=fnval(fnder(sp2pp(spz)),t);

%Pontos de velocidade da Trajetoria - EJ
DJp0=[]; DJp1=[]; DJp2=[];
Jac=[]; Jac11=0; Jac12=0; Jac21=0; Jac22=0;
Vel01=0; Vel02=0;
for i=1:np
   Jac(1,1)=-(0.25*sin(Jp0(i,1))+0.25*sin(Jp0(i,1)+Jp1(i,1)));
   Jac(1,2)=-(0.25*sin(Jp0(i,1)+Jp1(i,1)));
   Jac(2,1)=0.25*cos(Jp0(i,1))+0.25*cos(Jp0(i,1)+Jp1(i,1));
   Jac(2,2)=0.25*cos(Jp0(i,1)+Jp1(i,1));
   %Inversa do Jacobiano
   IJac=inv(Jac);
   %Velocidade na Junta 0
   Vel01=IJac(1,1)*DXt(1,i)+IJac(1,2)*DYt(1,i);
   DJp0=[DJp0; Vel01];
```

```
   %Velocidade na Junta 1
   Vel02=IJac(2,1)*DXt(1,i)+IJac(2,2)*DYt(1,i);
   DJp1=[DJp1; Vel02];
   %Velocidade na Junta 2
   DJp2=[DJp2; -(DZt(1,i))];
end;

%Pontos de aceleracao da Trajetoria - EC
DDXt=fnval(fnder(fnder(sp2pp(spx))),t);
DDYt=fnval(fnder(fnder(sp2pp(spy))),t);
DDZt=fnval(fnder(fnder(sp2pp(spz))),t);

%Pontos de aceleracao da Trajetoria - EC
DDJp0=[]; DDJp1=[]; DDJp2=[]; M=[];
Jac=[]; Jac11=0; Jac12=0; Jac21=0; Jac22=0;
DJac=[]; DJac11=0; DJac12=0; DJac21=0; DJac22=0;
Ac01=0; Ac02=0;
for i=1:np
   %Jacobiano
   Jac(1,1)=-(0.25*sin(Jp0(i,1))+0.25*sin(Jp0(i,1)+Jp1(i,1)));
   Jac(1,2)=-(0.25*sin(Jp0(i,1)+Jp1(i,1)));
   Jac(2,1)=0.25*cos(Jp0(i,1))+0.25*cos(Jp0(i,1)+Jp1(i,1));
   Jac(2,2)=0.25*cos(Jp0(i,1)+Jp1(i,1));
   %Derivada do Jacobiano
   DJac(1,1)=-
(0.25*cos(Jp0(i,1))+0.25*cos(Jp0(i,1)+Jp1(i,1))*DJp0(i,1));
   DJac(1,2)=-0.25*cos(Jp0(i,1)+Jp1(i,1))*(DJp0(i,1)+DJp1(i,1));
   DJac(2,1)=-
(0.25*sin(Jp0(i,1))+0.25*sin(Jp0(i,1)+Jp1(i,1))*DJp0(i,1));
   DJac(2,2)=-0.25*sin(Jp0(i,1)+Jp1(i,1))*(DJp0(i,1)+DJp1(i,1));

   %Inversa do Jacobiano
   IJac=inv(Jac);
   %Matriz Intermediaria
   M=IJac*DJac;
   %Aceleracao na Junta 0
   Ac01=IJac(1,1)*DDXt(1,i)+IJac(1,2)*DDYt(1,i)-
(M(1,:)*([DJp0(i,1);DJp1(i,1)]));
   DDJp0=[DDJp0; Ac01];
   %Aceleracao na Junta 1
   Ac02=IJac(2,1)*DDXt(1,i)+IJac(2,2)*DDYt(1,i)-
(M(2,:)*([DJp0(i,1);DJp1(i,1)]));
   DDJp1=[DDJp1; Ac02];
   %Aceleracao na Junta 2
   DDJp2=[DDJp2; -(DDZt(1,i))];
end;

%Plot dos valores de posicao no plano XY
cax = newplot;
cla;
prmtrs_rng=[-.65 .65 -.65 .65];
axis(prmtrs_rng);
xlabel('Eixo X');
ylabel('Eixo Y');
grid on;
hold on;
xcg=[];xcp=[];ycg=[];ycp=[];
for theta=0:2*pi/100:2*pi;
   xcg=[xcg .5*sin(theta)];
   ycg=[ycg .5*cos(theta)];
   xcp=[xcp .3*sin(theta)];
```

```
   ycp=[ycp .3*cos(theta)];
end;
plot(xcg,ycg,'r-');
hold on;
plot(xcp,ycp,'r-');
view(0,90);
plot(x,y,'b');
hold on;
plot(Xt,Yt,'m');
%legend('Posicoes Escolhidas','Curva Encontrada');
title('Plot dos Curvas Encontradas');
zoom;


%Posicao no tempo
%figure;
%grid on;
%zoom on;
%hold on;
%plot(t,Xt,'r');
%plot(t,Yt,'m');
%plot(t,Zt,'b');
%legend('X(t)','Y(t)','Z(t)');
%ylabel('metros');
%xlabel('segundos');
%title(' Posicao no Espaco Cartesiano');


%Juntas no tempo
figure;
grid off;
zoom on;
hold on;

plot(t,Jp0,'r');
plot(t,Jp1,'m');
plot(t,Jp2,'b');
legend('Jp0(t)','Jp1(t)','Jp2(t)');
ylabel('radianos');
xlabel('segundos');
title(' Posicao no Espaco de Juntas');


%Velocidade das juntas no Tempo
figure;
grid off;
zoom on;
hold on;
plot(t,DJp0,'r');
plot(t,DJp1,'m');
plot(t,DJp2,'b');
legend('DJp0(t)','DJp1(t)','DJp2(t)');
ylabel('rad/segundos');
xlabel('segundos');
title('Velocidade no Espaco de Juntas');


%Aceleracao nas Juntas
figure;
grid on;
```

```
zoom on;
hold on;
plot(t,DDJp0,'r');
plot(t,DDJp1,'m');
plot(t,DDJp2,'b');
legend('DDJp0(t)','DDJp1(t)','DDJp2(t)');
ylabel('rad/s^2');
xlabel('segundos');
title('Aceleracao no Espaco de Juntas');

Pontos=[Jp0 Jp1 Jp2 DJp0 DJp1 DJp2 DDJp0 DDJp1 DDJp2];

%SALVA EM ARQUIVO
save pontos.mat Pontos -ascii;

%Figura 3D do Caminho Cartesiano
%figure;
%plot3(Xt,Yt,Zt);
%grid on;
%rotate3d;
%xlabel('Eixo X');
%ylabel('Eixo Y');
%zlabel('Eixo Z');


end;
```

O Software GeTraj.Mod

O *software GeTraj.Mod* foi desenvolvido em linguagem XOberon e pode ser utilizado para movimentar o braço do robô Inter através da leitura e envio de dados de uma trajetória ao sistema de controle do manipulador. Estes dados estão em um arquivo de nome "pontos.mat", gerado no Matlab, e deve ser transmitido à memória do robô com o nome de "Traj.m", utilizando para esta transmissão um *software* chamado TFTP Server 95.

Este software possui quatro rotinas:

- *Instalar*:

  É a primeira rotina a ser executada. Efetua a inicialização das variáveis, chama a rotina *LerTodosVal*, posiciona o efetuador-final no inicio da trajetória, muda o tipo de controlador e solicita autorização para o utilizar e chama as rotinas *Verificar* e *EnvDados*;

- *LerTodosVal*:

  Carrega o arquivo Traj.m que contém os dados da trajetória, efetua a leitura e conversão dos mesmos, armazenando-os em variáveis apropriadas;

- *EnvDados*:

  Fornece os dados da trajetória ao controlador do robô;

- *Verificar*:

  Responsável por liberar o controlador quando a trajetória já tiver sido terminada.

O programa é executado a partir da linha de comando *Xsystem.Call GeTraj.Instalar*. Abaixo é apresentada a descrição do GeTraj.Mod.

```
(*-------------------------------------------------
        Modulo Gerador de Trajetorias
        Versao 1.0 25/05/98
        Laboratorio de Automacao Industrial
        Centro Tecnologico - UFSC
-------------------------------------------------------*)

(* Este Modulo carrega um arquivo de dados de posicao, velocidade e aceleracao
provenientes do MatLab ( Traj.mat ), converte-os em LONGREAL  e manda-os para
o controlador do robo. O posicionamento inicial eh feito pelo Gerador de Trajetorias
ponto a ponto antigo ( Bahnplaner.Mod ) em baixa velocidade.  Atualmente esta
preparado para ler ate 10.000 linhas de pontos e ainda nao foi determinado o numero
maximo de linhas possiveis. *)

MODULE GeTraj;

IMPORT
        F:= XFiles, XT:= XTexts,
        XO:=XOberon, StrConv,
        GD:=GlobalDefs, XOK:=PPCXOKernel,
        M := Main3, BP := Bahnplaner;

CONST
        Clock = 1;

VAR
        convOk,
          (* Verifica se a conversao foi feita de acordo        *)
        dummy,
                (* Boolean utilizado sem proposito especial *)
        primeiraVez,
        (* Utilizado para posicionar o Rider          *)
        leituraOk,
                (* Verificacao dos dados do Arquivo        *)
        fileOk,
                 (* Procura pelo Arquivo Traj.mat  *)
        autoritOk,
          (* Verifica se o controlador esta livre       *)
        controlOk,
          (* Troca o tipo de controlador para StateCtrl        *)
        ptIniOk: BOOLEAN;                                                    (*
Verifica se o robo foi para o ponto inicial da trajetoria          *)
```

```
        tamanhoFile: LONGINT;                                   (* Tamanho do Arquivo
em caracteres *)
        m: LONGINT;
        R: F.Rider;
                (* Posicionador para leitura do Arquivo      *)
        posDes, velDes, acelDes: GD.Vector4;
        pos0, vel0, acel0,
        pos1, vel1, acel1,
                        (* Valores Convertidos de Posicao, Velocidade e *)
        pos2, vel2, acel2,
                        (* Aceleracao.    *)
        pos3, vel3, acel3: ARRAY 10100 OF LONGREAL;
        q0, q1, vq0, vq1,
        qm0, qm1, vmq0, vmq1,                                   (* Usados para testes e
checagem        *)
        qf0, qf1, vfq0, vfq1 : LONGREAL;
        coluna,
                (* Numero da coluna do Arquivo   *)
        linha,
                  (* Numero da linha       *)
        envio: LONGINT;
        (* Numero de linhas enviadas       *)
        mainevent : XOK.MainEvent;
        ctrlevent : XOK.EveryEvent;
        matautority : LONGINT;


PROCEDURE LerTodosVal*(): BOOLEAN;
VAR
        f: F.File;
        (* Arquivo carregado       *)
        Valor: ARRAY 32 OF LONGREAL;        (* Valor transformado dos caracteres          *)
        a: ARRAY 1024 OF CHAR;                                  (* ARRAY com os caracteres que
sao numeros      *)
        i, j, x: LONGINT;
        ch: CHAR;
        (* Caracter carregado para verificacao        *)

BEGIN
        f:= F.Old("Traj.m");
        IF f#NIL THEN
                i:=0; x:=0;
                tamanhoFile:=F.Length(f);
                IF primeiraVez THEN F.Set(R,f,0);
                ELSE F.Set(R,f,F.Pos(R));
                END;
                F.Read(R,ch);
                WHILE ~R.eof DO
                        IF ((ch>2AX)&(ch<2FX))OR((ch>29X)&(ch<40X))OR(ch=65X) THEN
                                WHILE
((ch=2DX)OR(ch=2EX)OR((ch>29X)&(ch<40X))OR(ch=65X)) DO
                                        a[i]:= ch; INC(i);
                                        F.Read(R,ch);
                                END;
                                convOk:= StrConv.StrToReal(a, Valor[x]);
```

```
                                INC(x);
                                IF (x=9) THEN
                                        coluna:=0;
                                        pos0[linha]:=Valor[coluna]; INC(coluna);
                                        pos1[linha]:=Valor[coluna]; INC(coluna);
                                        pos2[linha]:=Valor[coluna]; INC(coluna);
                                        pos3[linha]:=0;
                                        vel0[linha]:=Valor[coluna]; INC(coluna);
                                        vel1[linha]:=Valor[coluna]; INC(coluna);
                                        vel2[linha]:=Valor[coluna]; INC(coluna);
                                        vel3[linha]:=0;
                                        acel0[linha]:=Valor[coluna]; INC(coluna);
                                        acel1[linha]:=Valor[coluna]; INC(coluna);
                                        acel2[linha]:=Valor[coluna]; INC(coluna);
                                        acel3[linha]:=0;
                                        primeiraVez:=FALSE;
                                        x:=0;
                                        INC(linha);
                                END;
                                FOR j:=0 TO i DO a[j]:=" " END;
                                i:=0;
                        ELSE F.Read(R,ch);
                        END;
                END;
                i:=0;
                F.Close(f);
        ELSE fileOk:=FALSE;
        END;
        fileOk:=FALSE;
        IF primeiraVez THEN RETURN FALSE;
        ELSE RETURN TRUE;
        END;
END LerTodosVal;

PROCEDURE Resultado*;
VAR w: XT.Writer; media : LONGINT;
BEGIN
        ASSERT(XO.Writer(w));
        IF fileOk THEN
                q0:=pos0[0];
                q1:= pos1[0];
                vq0 := vel0[0];
                vq1 := vel1[0];

                media:= ENTIER(linha/2);
                qm0 :=pos0[media];
                qm1:= pos1[media];
                vmq0 := vel0[media];
                vmq1 := vel1[media];

                qf0 :=pos0[linha-1];
                qf1:= pos1[linha-1];
                vfq0 := vel0[linha-1];
                vfq1 := vel1[linha-1];
```

```
                XT.WriteLn(w);XT.WriteLn(w);
                XT.WriteString(w, " Movimento Realizado com sucesso absoluto");
                XT.WriteLn(w); XT.WriteLn(w);

                XT.WriteString(w, " primeira linha");XT.WriteLn(w);
                XT.WriteLongReal(w, q0, 12); XT.WriteLn(w);
                XT.WriteLongReal(w, q1, 12); XT.WriteLn(w);
                XT.WriteLongReal(w, vq0, 12); XT.WriteLn(w);
                XT.WriteLongReal(w, vq1, 12); XT.WriteLn(w); XT.WriteLn(w);

                XT.WriteString(w, " metade do movimento");XT.WriteLn(w);
                XT.WriteLongReal(w, qm0, 12); XT.WriteLn(w);
                XT.WriteLongReal(w, qm1, 12); XT.WriteLn(w);
                XT.WriteLongReal(w, vmq0, 12); XT.WriteLn(w);
                XT.WriteLongReal(w, vmq1, 12); XT.WriteLn(w); XT.WriteLn(w);


                XT.WriteString(w, " ultima  linha");XT.WriteLn(w);
                XT.WriteLongReal(w, qf0, 12); XT.WriteLn(w);
                XT.WriteLongReal(w, qf1, 12); XT.WriteLn(w);
                XT.WriteLongReal(w, vfq0, 12); XT.WriteLn(w);
                XT.WriteLongReal(w, vfq1, 12); XT.WriteLn(w);
        ELSE
                XT.WriteLn(w);XT.WriteLn(w);
                XT.WriteString(w, " Ocorreram graves problemas");
        END;
        XT.WriteLn(w); XT.Append(XO.XLog(), w.buf);
END Resultado;

PROCEDURE EnvDados ( e: XOK.Event);
VAR
BEGIN
        IF m < linha THEN
        posDes[0]:=pos0[m];
        posDes[1]:=pos1[m];
        posDes[2]:=pos2[m];
        posDes[3]:=pos3[m];
        velDes[0]:=vel0[m];
        velDes[1]:=vel1[m];
        velDes[2]:=vel2[m];
        velDes[3]:=vel3[m];
        INC(m);
        (* dummy := M.SetRobPosSpeed(matautority, posDes, velDes);*)
        ELSE leituraOk:=FALSE;
        END;
END EnvDados;

PROCEDURE  Verificar( e: XOK.Event);
BEGIN
        WHILE leituraOk DO
        END;
        ctrlevent.UnInstall;
(*      fileOk:= TRUE; *)
        IF ~ M.AuthorityFree(matautority) THEN HALT(120) END;
END Verificar;
```

```
PROCEDURE Instalar*;
VAR wt: XT.Writer;
BEGIN
	ASSERT(XO.Writer(wt));
	leituraOk:=TRUE;
	fileOk:= FALSE;
	convOk:= TRUE;
	primeiraVez:=TRUE;
	ptIniOk := FALSE;
	linha:=0; envio:=0;
	leituraOk:= LerTodosVal();
	ptIniOk:=  BP.MoveAbsAllJoint(  SHORT(pos0[0]),  SHORT(pos1[0]),  SHORT(pos2[0]),
SHORT(pos3[0]), 0.05, 0.1);
	autoritOk := FALSE;
	controlOk := FALSE;
	controlOk:= M.ChangeCtrl(GD.SStateCtrl);
	IF( controlOk & leituraOk & ptIniOk & fileOk ) THEN
		autoritOk:=M.GetAuthority(matautority);
		IF autoritOk THEN
			XT.WriteLn(wt);XT.WriteLn(wt);
			XT.WriteString(wt, " Iniciado o movimento");
			m:=0;
			XOK.InitMain(mainevent); mainevent.Install(Verificar); mainevent.Notify;
			XOK.InitEvery(ctrlevent, Clock, XOK.ONEms); ctrlevent.Install(EnvDados);
		ELSE
			XT.WriteLn(wt);XT.WriteLn(wt);
			XT.WriteString(wt, " O controlador esta sendo utilizado por outro modulo");
		END;
	ELSE
		XT.WriteLn(wt);XT.WriteLn(wt);
		IF ~controlOk THEN
			XT.WriteString(wt, " Nao foi  possivel alterar o controlador!!");
			XT.WriteLn(wt);
		END;
		IF ~leituraOk THEN
			XT.WriteString(wt, " Encontrou o arquivo, mas nao converteu os dados
corretamente!!");
			XT.WriteLn(wt);
			XT.WriteString(wt, " Verifique se todas as linhas estao completas.");
			XT.WriteLn(wt);
		END;
		IF ~ptIniOk THEN
			XT.WriteString(wt, " Nao foi  possivel movimentar o robo para a posicao
inicial!!");
			XT.WriteLn(wt);
		END;
		IF ~fileOk THEN
			XT.WriteString(wt, " Arquivo nao encontrado!!");
			XT.WriteLn(wt);
		END;
	END;
	XT.WriteLn(wt); XT.Append(XO.XLog(), wt.buf);
END Instalar;
```

```
BEGIN
        NEW(mainevent);
        NEW(ctrlevent);
END GeTraj.

XSystem.Call GeTraj.Instalar ~
XSystem.Call GeTraj.Resultado ~
```