DAS Departamento de Automação e Sistemas
CTC Centro Tecnológico
UFSC Universidade Federal de Santa Catarina

Desenvolvimento e Testes de um Gerador de Trajetórias para o Robô Inter - SCARA

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

EEL 5901: Projeto de Fim de Curso

Julio Feller Golin

Florianópolis, Maio de 1999

Desenvolvimento e Testes de um Gerador de Trajetórias para o Robô Inter - SCARA

Julio Feller Golin

Esta monografia foi julgada no contexto da disciplina
EEL 5901: 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 Roberto de Pieri, Avaliador

Giulliano Carlo Jesus Pereira, Debatedor

Hugo Leonardo Gosmann, Debatedor

Agradecimentos

Ao Prof. Raul Guenther, que me ofereceu a oportunidade de realizar este trabalho, pela orientação;

Ao Eng. Carlos Bier, colega do Laboratório de Robótica, pelos esclarecimentos gerais e sobre trajetórias de robôs manipuladores;

Ao Eng. Clóvis Fernandes Júnior, pela paciência de suas explicações sobre o sistema que desenvolveu anteriormente a este;

Aos colegas do Laboratório de Robótica, que têm se unido num esforç conjunto para a superação de dificuldades, que este esforço continue e gere novos avanços;

Aos colegas do LAI, em especial ao técnico Waldoir Gomes Júnior, pela prestatividade;

Aos meus pais, incentivadores constantes e compreensíveis nos momentos difíceis.

Resumo

Este trabalho visa apresentar uma ferramenta desenvolvida para a geração de trajetórias de referência para um manipulador SCARA. Este gerador permite a programação totalmente *off-line* do robô a que foi aplicado, caso onde todos os valores de referência para o controlador são previamente calculados, bem como a geração *on-line* de trajetórias a partir de parâmetros de funções interpolantes, anteriormente calculados e enviados ao seu armário de controle. As trajetórias de referência são geradas no espaço das juntas do manipulador, a partir de um caminho especificado pelo usuário no espaço cartesiano, e permitem a execução de movimentos suaves e contínuos. Metodologias, resultados e as dificuldades encontradas neste trabalho são apresentadas.

Abstract

This work focus on the trajectory planning problem for robotic mechanisms. It is applied to an industrial SCARA manipulator and the goal is to calculate smooth trajectories in the joints space for this robot, from paths denoteds in the operational space. Two solutions are provided: off-line and on-line. Methodoligies, results and difficulties are discussed.

Sumário

Capítulo 1: Introdução

Um robô pode ser entendido conforme uma definição genérica proposta pelo *Robot Institute of America* [Spong&Vidyasagar89] que diz que "um robô é um manipulador multifuncional e reprogramável projetado para a movimentação de materiais, peças, ferramentas ou outros dispositivos, através de diferentes movimentos programados para a execução de uma variedade de tarefas".

Para tanto, um robô ou manipulador é constituído, genericamente, dos componentes apresentados na figura 1.1. O manipulador tem liberdade para movimentar seus elos através do acionamento dos atuadores das respectivas juntas. As juntas são elementos mecânicos que fazem a conexão entre dois elos. A combinação dos acionamentos das juntas do manipulador determina o movimento resultante do efetuador final do robô, responsável pela realização de uma tarefa, como por exemplo soldagem, corte ou montagem de peças. Para que a tarefa especificada pelo usuário seja corretamente realizada pelo manipulador, é necessário um sistema de controle que planeje adequadamente os movimentos, enviando sinais de controle para as juntas, na forma de forças ou torques, e que interrompa imediatamente o manipulador na presença de alguma falha.

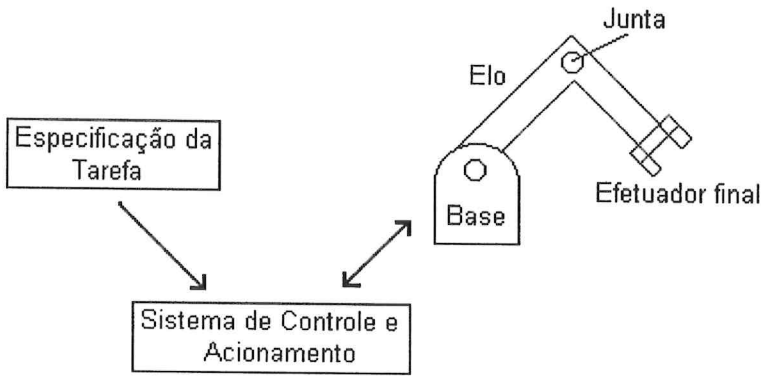


Figura 1.1 – Esquema geral de um manipulador

Pelas considerações acima, podemos dividir a execução de uma tarefa em duas fases genéricas:

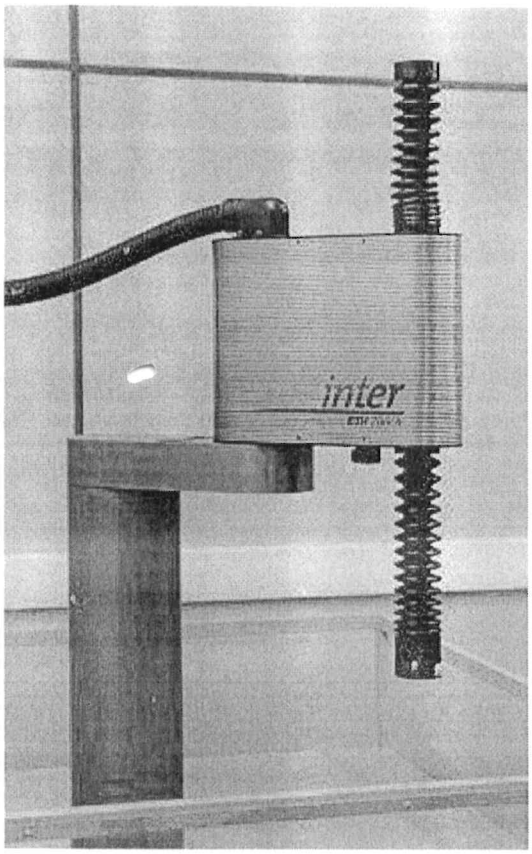
- a especificação da tarefa e o planejamento das trajetórias a serem executadas pelo manipulador, e
- a execução das trajetórias de referência, sob comando e supervisão do sistema de controle e acionamento.

Este trabalho abrange atividades das duas fases. Na primeira, trata da especificação, por parte do usuário, de uma trajetória no espaço da tarefa do manipulador (espaço cartesiano), e da geração das respectivas trajetórias de referência espaço das juntas para o controlador do robô. Na segunda etapa, trata da comunicação entre estas trajetórias de referência, representadas na forma de um arquivo de dados, e o controlador do manipulador, isto é, em como interpretar estes dados e enviar valores de referência ao controlador.

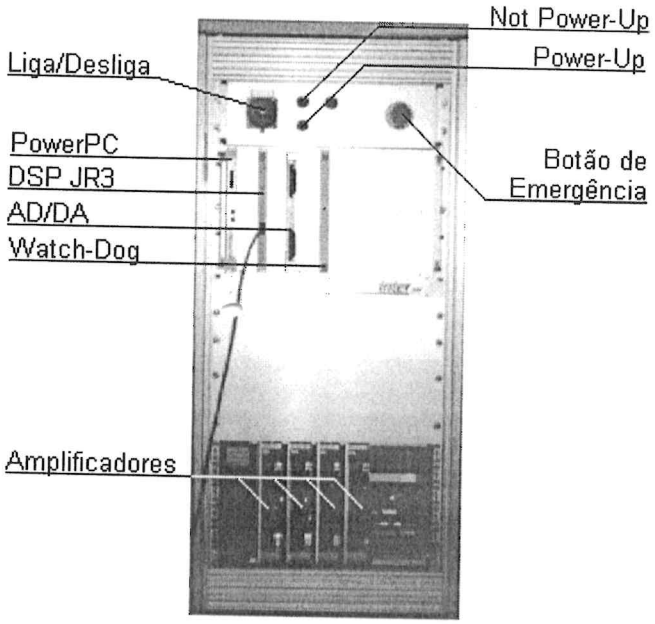
Neste contexto, este trabalho envolve o estudo teórico e a comparação de algumas técnicas de geração de trajetórias de referência e suas implementações, bem como os aspectos relativos ao robô em que serão empregadas – configuração, limitações físicas, ambiente de desenvolvimento, entre outros.

O objeto deste trabalho foi, então, o robô Inter, do Laboratório de Robótica da UFSC. Este robô foi construído, sob encomenda da Universidade, no Instituto de Robótica da Universidade Técnica Federal de Zurique (ETH), na Suíça. Trata-se de um robô de porte industrial de configuração SCARA para fins de pesquisa. Sua principal característica é ter uma arquitetura aberta, ou seja, a possibilidade de programação e implementação de diferentes algoritmos de controle, ao contrário da maioria dos robôs industriais, onde o usuário não tem acesso ao ambiente de programação e controle do equipamento. Neste equipamento o usuário opera em um microcomputador, chamado *host*, conectado via LAN à CPU do robô, referida como "armário de controle" ou *target* e onde também estão montados os dispositivos de alimentação do robô, botoeiras de liga-desliga e emergência, placas de conversão AD/DA, e placa controladora do sensor de força. A comunicação entre

host e *target* é feita através de um ambiente computacional chamado XOberon (que é portanto a interface entre usuário e robô). A figura 1.1 ilustra o robô e o armário de controle.



(a)



(b)

Figura 1.1 – Robô Inter (SCARA) e armário de controle

A justificativa para se trabalhar com geração de trajetórias aplicadas ao robô Inter é que o manipulador, originalmente, só trabalha com trajetórias ponto-a-ponto. Assim, para se executar uma seqüência de movimentos o usuário tem duas alternativas:

- executar uma seqüência de comandos textuais equivalente aos movimentos desejados, ou
- escrever um módulo que contenha estes comandos.

Em qualquer dos casos estes comandos são escritos no ambiente de trabalho do manipulador. Este ambiente, o XOberon, de imediato apresenta algumas dificuldades principalmente por tratar-se de uma linguagem nova (pelo menos para o Laboratório) e ainda pouco documentada, e da complexidade das relações entre os diferentes módulos que controlam o manipulador. Além disso, como os movimentos são ponto a ponto, não é possível realizar desvios (como, por exemplo, de obstáculos) sem que se tenha velocidades nulas neste trajeto.

Posto isso, propomos neste trabalho o desenvolvimento de uma ferramenta gráfica que auxilie o usuário na tarefa da geração de trajetórias para este manipulador, conforme mencionado anteriormente. De maneira geral, do ponto de vista do usuário é desejável que este possa especificar a trajetória desejada no espaço cartesiano do manipulador, pela facilidade que isto representa. Assim, é necessário o conhecimento das cinemáticas direta e inversa dos manipuladores e sua implementação no desenvolvimento de uma ferramenta geradora de trajetórias.

As alternativas de solução para este problema conduziram à se optar por programar as trajetórias desejadas em *off-line*. Isto permite uma análise prévia dos resultados gerados e deixa o robô livre para outras atividades enquanto é feita esta programação. O fator mais determinante à esta escolha, entretanto, é que a programação de trajetórias *off-line* em XOberon requer capacidades do sistema, principalmente de manipulação gráfica, que o sistema não dispõe. Não se pode deixar de citar aqui que já havia sido desenvolvido um gerador prévio a este, como atividade de projeto de fim de curso. Este gerador, desenvolvido em Matlab por Clóvis Fernandes Jr. sob orientação do Prof. Raul Guenther, gera trajetórias do tipo *splines* no espaço cartesiano do manipulador para as suas três primeiras juntas. Como resultado, obtém-se um arquivo de valores de referência para posição, velocidade e aceleração para cada uma destas juntas, amostrados em intervalos de 1ms e que é posteriormente enviado ao *target* para movimentação do robô.

A partir daí, desenvolvemos outros dois aplicativos, também em Matlab, para a geração de trajetórias suaves (utilizando *splines* cúbicas) no espaço de juntas a partir de valores desejados no espaço cartesiano. Este gerador utiliza a mesma proposta de criação de um arquivo de pontos de referência, porém agora incluindo a quarta junta do manipulador e utilizando como variáveis de referência para o controlador posição e velocidade, o que contribui para a redução do tamanho do arquivo gerado.

Numa segunda etapa, foi desenvolvido um outro conjunto de procedimentos em XOberon que possibilita a leitura de um arquivo contendo apenas os coeficientes das *splines* de cada segmento da trajetória, para cada junta do manipulador. Com isso, o *target* tem condições de gerar as trajetórias de referência em tempo real de execução. A

metodologia utilizada para isso bem como as dificuldades que enfrentamos são tratadas ao longo deste documento.

Feitas estas considerações, este documento está organizado da seguinte forma:

- no capítulo dois são apresentadas técnicas de geração de trajetórias de manipuadores;
- o capítulo três trata do robô Inter, suas características aqui relevantes e o ambiente XOberon;
- no quarto capítulo apresentamos o aplicativo desenvolvido em Matlab e XOberon;
- no capítulo cinco são apresentados alguns resultados e discussões e,
- finalmente, no capítulo seis há as conclusões e perspectivas deste trabalho.

Capítulo 2: Trajetórias de Referência

Para que se possa controlar o problema de movimentação de manipuladores, é necessário o desenvolvimento de um planejador de trajetórias, responsável por calcular a história temporal das posições, velocidades e acelerações desejadas para cada junta. Este problema inclui as questões da interface com o usuário e as metodologias utilizadas para geração de trajetórias de referência, que podem ou não ser em tempo real de movimentação.

A primeira questão trata de como o usuário pode especificar, de maneira simples, um conjunto mínimo de dados que será utilizado no cálculo das trajetórias de referência. Estas informações incluem o *caminho* desejado – isto é, os pontos espaciais que o manipulador deve seguir – podendo também especificar parâmetros como tempo, orientação, velocidade e/ou aceleração em cada ponto do caminho. Tipicamente, esta descrição é feita no espaço cartesiano por nele se poder representar de forma "natural" a tarefa que se deseja executar no manipulador.

A segunda questão trata da "estratégia" utilizada para o cálculo destas trajetórias de referência. Evidentemente, existem diversas maneiras de se gerar uma função ou um conjunto de funções que interpole um conjunto de pontos espaciais nos devidos instantes de tempo. Para que uma trajetória possa ser fisicamente realizada pelo manipulador, ela deve obedecer certas restrições impostas pelas características cinemáticas e dinâmicas do mesmo. Uma condição necessária na metodologia escolhida é que as trajetórias de referência sejam suaves o suficiente para não excederem os limites suportados pela estrutura mecânica do manipulador. Características como continuidade de velocidade e aceleração são desejáveis pois garantem torques também contínuos, evitando a ativação de modos de ressonância. Assim, qualquer função "suave" poderia ser usada para se especificar as trajetórias de referência a partir do caminho desejado.

Com isso, vemos que a escolha do tipo de trajetória é uma decisão importante na implementação de técnicas de controle. Esta trajetória, além de aplicável, deve ser computacionalmente tratável.

A figura 2.1 ilustra um esquema geral para geração e controle de trajetórias.

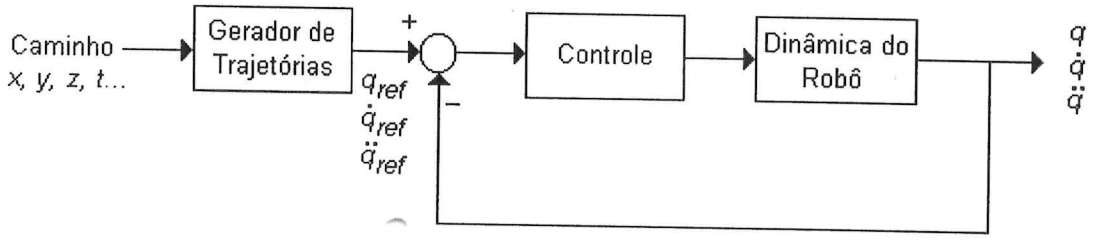


Figura 2.1 – Visão geral da geração e controle de trajetórias de manipuladores

Neste capítulo apresentamos uma revisão de algumas técnicas utilizadas na geração de trajetórias de referência de manipuladores. Estas técnicas foram implementadas em Matlab, sendo duas delas implementadas posteriormente para uso no robô Inter. Para isso desenvolvemos uma interface, também em Matlab, em que o usuário especifica o caminho desejado no espaço cartesiano do robô, bem como demais parâmetros que sejam necessários. Uma consideração que se faz é que o tempo requerido para movimentação do manipulador entre quaisquer dois pontos do caminho é o mesmo para todas as juntas. Com isso, a função de acionamento de cada junta não depende das demais, sendo o sistema desacoplado. As trajetórias de referência são aqui tratadas, também, no espaço das juntas do manipulador. Consideramos, portanto, que a etapa de cálculo da cinemática inversa já foi concluída.

Esta revisão foi baseada em [BCMP98], [Craig86] e [Sciavicco&Siciliano96].

2.1 – Trajetórias Ponto-a-Ponto

Na trajetória ponto-a-ponto deseja-se que o manipulador efetue o movimento de um ponto inicial q_i para um ponto final q_f em um certo período de tempo t_f. Evidentemente existem infinitas soluções para este problema e a questão centra-se em achar uma trajetória que atenda os requisitos de tempo e seja fisicamente realizável pelo manipulador. Este é um problema simples e nesta seção apresentamos uma revisão de três técnicas, que são: polinômio cúbico, 5ª ordem e trapezoidal. É interessante salientar que as trajetórias ponto-a-ponto não permitem o desvio de obstáculos.

2.1.1 – Polinômio cúbico

O polinômio mínimo necessário para que se possa especificar os valores $q_i, q_f, \dot{q}_i$ e $\dot{q}_f$ desejados para um determinado movimento ponto-a-ponto em um intervalo de tempo t_f é o de 3ª ordem:

$$q(t) = a_3 t^3 + a_2 t^2 + a_1 t + a_0 \qquad (2.1)$$

Derivando-se este polinômio duas vezes, obtemos um perfil parabólico de velocidade e um perfil linear de aceleração. A determinação de uma trajetória específica é dada pela solução do seguinte sistema de equações:

$$\begin{cases} a_o = q_i \\ a_1 = \dot{q}_i \\ a_3 t_f^3 + a_2 t_f^2 + a_1 t_f + a_0 = q_f \\ 3a_3 t_f^2 + a_2 t_f + a_1 = \dot{q}_f \end{cases}$$

que fornece os valores dos coeficientes para a equação (1). A figura 2.2 ilustra os perfis gerados para $q_i = 0, q_f = \pi, \dot{q}_i = \dot{q}_f = 0$ e t_f = 1. Uma forte limitação deste método é que não se consegue restringir a aceleração gerada, que pode não ser fisicamente suportada pela junta.

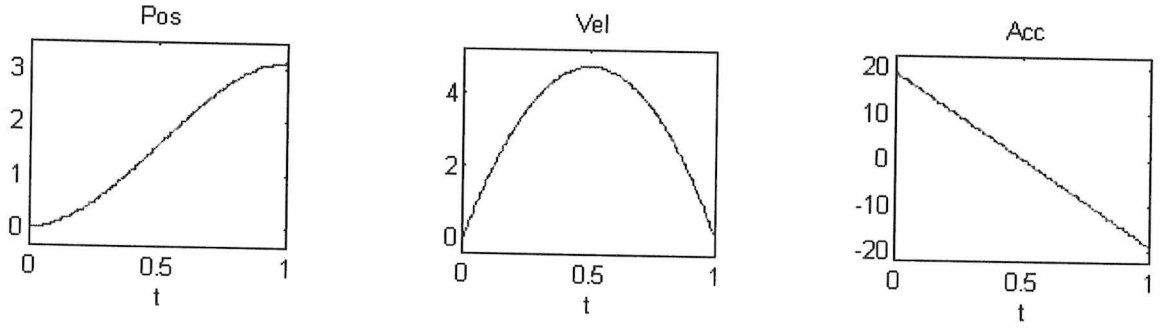


Figura 2.2 – Trajetória polinomial cúbica

2.1.2 – Polinômio de 5ª ordem

Quando se deseja movimento ponto-a-ponto com perfis de trajetórias contínuos em posição, velocidade e aceleração, deve-se utilizar polinômios de no mínimo 5ª ordem:

$$q(t) = a_5 t^5 + a_4 t^4 + a_3 t^3 + a_2 t^2 + a_1 t + a_0 \qquad (2.2).$$

Com isso é possível se especificar, além as posições e velocidades iniciais e finais, os respectivos valores desejados de aceleração. A resolução do polinômio (2) sugere um sistema de equações conforme apresentado abaixo:

$$\begin{cases} q_i = a_0 \\ \dot{q}_i = a_1 \\ \ddot{q}_i = 2a_2 \\ q_f = a_5 t^5 + a_4 t^4 + a_3 t^3 + a_2 t^2 + a_1 t + a_0 \\ \dot{q}_f = 5a_5 t^4 + 4a_4 t^3 + 3a_3 t^2 + 2a_2 t + a_1 \\ \ddot{q}_f = 20a_5 t^3 + 12a_4 t^2 + 6a_3 t + 2a_2 \end{cases} \qquad (2.3).$$

A figura 2.3 ilustra os perfis gerados para $q_i = 0, q_f = \pi, \dot{q}_i = \dot{q}_f = \ddot{q}_i = \ddot{q}f = 0$ e $t_f =$ 1.

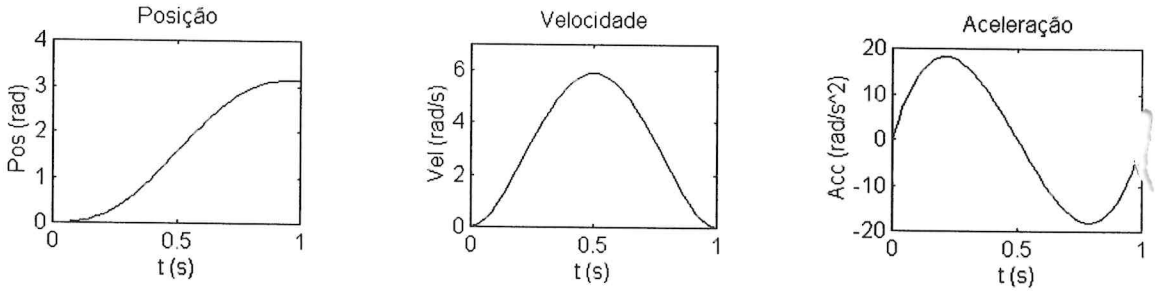


Figura 2.3 – Trajetória polinomial de 5ª ordem

2.1.3 – Perfil trapezoidal de velocidade

Uma trajetória bastante difundida para o movimento ponto a ponto é o perfil de velocidade trapezoidal. Trata-se de uma combinação de movimento linear e parabólico, em que um segmento linear "conecta" dois segmentos parabólicos de posição. A trajetória da junta pode ser dividida em três fases: inicialmente impõe-se uma aceleração constante, em

seguida esta aceleração é anulada ao atingir-se uma determinada *velocidade de cruzeiro* $\dot{q}_c$ num tempo t_c e finalmente realiza-se uma desaceleração na fase final, igual em módulo à primeira - e consequentemente de mesma duração de tempo. Esta imposição leva à simetria dos perfis em relação ao tempo médio $t_m = t_f / 2$.

A partir das posições inicial e final (q_i e q_f) e do tempo de execução do movimento (t_f), é gerado o referido perfil de acordo com uma aceleração $\ddot{q}_c$ previamente definida. Esta técnica considera que a junta a ser acionada inicia e termina seu movimento com velocidade nula, isto é, $\dot{q}_i = \dot{q}_f = 0$. A figura 2.4 ilustra os perfis gerados para $q_i=0$, $q_f=\pi$, $t_f=1$ e $|\ddot{q}_c = 6\pi|$.

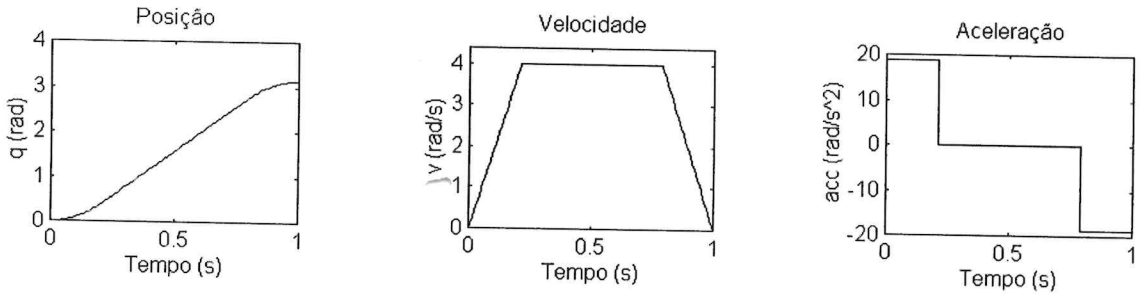


Figura 2.4: Perfil trapezoidal de velocidade

O tempo de cada um destes segmentos depende do tempo total do movimento, das posições inicial e final e da aceleração (ou velocidade) especificada. Vê-se que quanto maior for $|\ddot{q}_c|$ mais "linear" é a trajetória total, isto é, mais tempo se permanece na velocidade de cruzeiro.

Uma restrição que deve ser respeitada para se obter o perfil trapezoidal é que

$$|\ddot{q}_c| > \frac{4(q_f - q_i)}{t_f}.$$

Caso contrário obtém-se um perfil triangular de velocidade (quando há igualdade na equação acima) ou não é possível gerar um perfil real (quando o módulo de $\ddot{q}_c$ é menor que o restante da equação).

Devido à descontinuidade na aceleração, o perfil de torque também será descontínuo, o que, dependendo do controlador utilizado, pode danificar os atuadores e inclusive ativar modos de alta freqüência.

A tabela 2.1 resume o equacionamento para os perfis de posição, velocidade e aceleração para cada trecho da trajetória.

Intervalo	Posição (q)	Velocidade ($\dot{q}$)	Aceleração ($\ddot{q}$)
$0 \leq t < t_c$	$q_i + \frac{1}{2}\ddot{q}_c t$	$\ddot{q}_c t$	$\ddot{q}_c$
$t_c < t \leq t_f - t_c$	$q_i + \ddot{q}_c t_c (t - \frac{1}{2} t_f)$	$\ddot{q}_c t_c$	0
$t_f - t_c < t \leq t_f$	$q_f - \frac{1}{2}\ddot{q}_c (t_f - t)^2$	$\ddot{q}_c (t_f - t)$	$\ddot{q}_c$

Tabela 2.1 – Equacionamento do perfil trapezoidal de velocidade

O perfil trapezoidal de velocidade veio originalmente implementado no robô Inter através do módulo *Bahnplaner.Mod*.

2.2 – Trajetórias por caminho contínuo (*path motion*)

Em diversas aplicações na robótica é necessário descrever o caminho com mais que dois pontos apenas. Isto verifica-se mesmo para tarefas simples, como por exemplo em aplicações tipo *pick and place*, em que pode ser conveniente setar dois pontos intermediários aos pontos inicial e final de modo a se garantir deslocamentos mais lentos e suaves. No caso de aplicações mais complexas é comum a necessidade de se atribuir uma seqüência de pontos para se evitar determinadas regiões do espaço de trabalho do manipulador (como desvio de obstáculos) e/ou para melhor monitoramento das trajetórias executadas.

Em qualquer destas situações, porém, a quantidade de pontos pelos quais o robô deve passar é muito maior que apenas os dois pontos, inicial e final, das trajetórias ponto-a-ponto. Com isso, o problema a se resolver é: dados n pontos pelos quais o manipulador deve passar, chamados *path points*, gerar uma trajetória que os interpole nos instantes de tempo especificados.

Duas diretrizes podem ser seguidas para a resolução deste problema. Uma seria a geração de um único polinômio interpolador de ordem *(n-1)*. A outra opção é buscar

construir uma série de curvas de baixa ordem que, concatenadas nos pontos do caminho, produzam uma trajetória suave e contínua.

A primeira alternativa apresenta uma série de desvantagens, como por exemplo:

- não é possível a especificação das velocidades inicial e final desejadas;
- à medida que o número de *path points* aumenta, têm-se comportamentos mais oscilatórios;
- o polinômio pode ter uma demanda computacional grande e com o aumento da ordem deste polinômio a precisão computacional diminui;
- os coeficientes do polinômio dependem de todos os pontos especificados e, com isso, a mudança de um único ponto implica em um novo polinômio.

Postas estas desvantagens, a segunda alternativa torna-se evidentemente mais apropriada.

Conforme mencionado na seção anterior, o polinômio cúbico é o de ordem mínima para que se possa garantir continuidade de posição e velocidade nos *path points*, isto é, para que seja possível especificar os valores desejados de velocidade. Assim, com relação a uma única junta, temos que:

- a função $q(t)$ é agora uma seqüência de $(n-1)$ polinômios cúbicos $P_k(t)$, contínuos e com derivadas-primeira contínuas para $k=1, 2, ..., n-1$;
- a função $q(t)$ assume os valores q_k em $t=t_k$ para $k=1,2, ...,n$, ou seja, $q_1=q_i$ em $t_1=0$ e $q_n=q_{k+1}=q_f$ em $t_n=t_{k+1}=t_f$;
- os q_k são os *path points*, isto é, o caminho quando $t=t_k$.

Podem ser considerados, então, três casos:

- valores de $\dot{q}(t)$ são especificados nos *path points*;
- valores de $\dot{q}(t)$ são calculados nos *path points* segundo algum critério;

- a aceleração $\ddot{q}(t)$ deve ser contínua nos *path points* e deve-se garantir velocidades inicial e final nulas - chamada "solução por pontos virtuais".

Nesta seção revisaremos algumas técnicas utilizadas para resolução das situações acima, fazendo os comentários que se julgam pertinentes. Será dada maior atenção ao último caso por ser de tratamento matemático ligeiramente mais apurado e por ser a melhor solução entre as citadas.

2.2.1 – Interpolação polinomial cúbica com velocidades especificadas nos pontos intermediários

Esta solução requer que o usuário forneça, além das posições e respectivos tempos, as velocidades desejadas nos pontos intermediários.

O sistema de equações resultante para cada um dos *(n-1)* polinômios cúbicos é obtido impondo-se as seguintes condições para o *k*-ésimo polinômio, para *k=1, 2, ..., n-1*:

$$\begin{cases} P_k(t_k) = a_{3k}t_k^3 + a_{2k}t_k^2 + a_{1k}t_k + a_{0k} = q_k \\ P_k(t_{k+1}) = a_{3k}t_{k+1}^3 + a_{2k}t_{k+1}^2 + a_{1k}t_{k+1} + a_{0k+1} = q_{k+1} \\ \dot{P}_k(t_k) = 3a_{3k}t_k^2 + 2a_{2k}t_k + a_{1k} = \dot{q}_k \\ \dot{P}_k(t_{k+1}) = 3a_{3k}t_{k+1}^2 + 2a_{2k}t_{k+1} + a_{1k} = \dot{q}_{k+1} \end{cases} \tag{2.4}$$

onde deseja-se determinar os coeficientes a_{ik} do polinômio cúbico de posição $P_k(t)$.

É portanto um sistema de quatro equações com quatro incógnitas. Como são *(n-1)* polinômios, temos *(n-1)* sistemas independentes de equações a resolver. Com isso, cada sistema pode ser colocado em formato matricial do tipo **Ax=b**, ou seja:

$$\begin{bmatrix} t_k^3 & t_k^2 & t_k & 1 \\ t_{k+1}^3 & t_{k+1}^2 & t_{k+1} & 1 \\ 3t_k^2 & 2t_k & 1 & 0 \\ 3t_{k+1}^2 & 2t_{k+1} & 1 & 0 \end{bmatrix} \begin{bmatrix} a_{3k} \\ a_{2k} \\ a_{1k} \\ a_{0k} \end{bmatrix} = \begin{bmatrix} q_k \\ q_{k+1} \\ \dot{q}_k \\ \dot{q}_{k+1} \end{bmatrix} \tag{2.5}$$

As velocidades inicial e final são tipicamente nulas, ou seja, $\dot{q}_i = \dot{q}_1 = \dot{q}_f = \dot{q}_n = 0$, e a continuidade das velocidades nos pontos intermediários é garantida por $\dot{P}_k(t_{k+1}) = \dot{P}_{k+1}(t_{k+1})$, com *k=1, 2, ..., n-2* .

A figura 2.5 ilustra o caso para $q_1 = 0, q_2 = 2\pi, q_3 = \pi/2, q_4 = 2\pi, \dot{q}_1 = \dot{q}_4 = 0, \dot{q}_2 = \pi$, $\dot{q}_3 = -\pi, t_1 = 0, t_2 = 2, t_3 = 3$ e $t_4 = 5$.

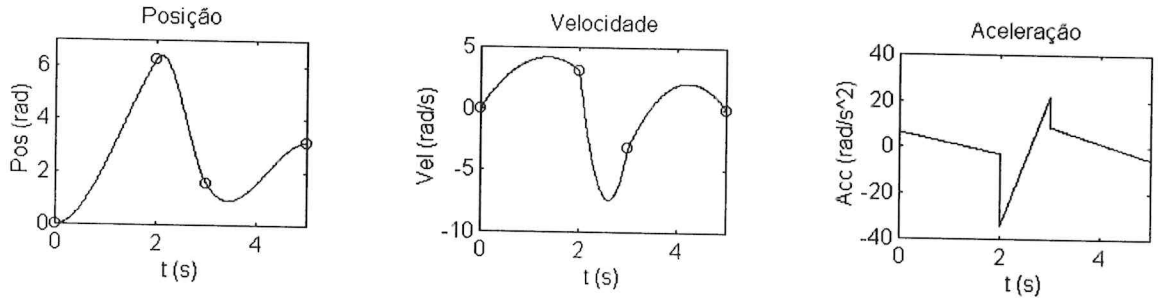


Figura 2.5 – Interpolação polinomial cúbica com velocidades intermediárias especificadas

Como neste caso não são especificadas as acelerações intermediárias mas somente as velocidades, obtem-se um perfil de aceleração descontínuo que pode causar descontinuidades nos torques gerados pela técnica de controle. Isto pode levar a problemas nos atuadores do manipulador e até excitar freqüências indesejadas.

2.2.2 – Interpolação polinomial cúbica com velocidades calculadas nos pontos intermediários

Trata-se da mesma problemática do caso anterior com a diferença que agora as velocidades das juntas são calculadas conforme algum critério. Supõem-se então uma interpolação linear entre os pontos intermediários e determina-se com isso a velocidade média entre os pontos, dada por

$$v_k = \frac{q_k - q_{k-1}}{t_k - t_{k-1}}$$

que determina a inclinação do segmento em $[t_{k-1}, t_k]$. O critério para a determinação das velocidades nos pontos intermediários é:

$$\begin{aligned} \dot{q}_1 &= 0 \\ \dot{q}_k &= \begin{cases} 0 & sinal(v_k) \neq sinal(v_{k+1}) \\ \frac{1}{2}(v_k + v_{k+1}) & sinal(v_k) = sinal(v_{k+1}) \end{cases} \\ \dot{q}_n &= 0 \end{aligned} \tag{2.6}$$

Vê-se que $\dot{q}_k = 0$ no caso de uma mudança no sentido de deslocamento da junta. É no ponto q_k, portanto, que a junta muda o sentido de seu deslocamento, devendo ali a velocidade ser nula. Isto não ocorre necessariamente nos *path points* da solução anterior, o que fica evidenciado através de uma comparação com a figura 2.6, cujos perfis são também para $q_1 = 0, q_2 = 2\pi, q_3 = \pi/2, q_4 = 2\pi, \dot{q}_1 = \dot{q}_4 = 0, \dot{q}_2 = \pi$, $\dot{q}_3 = -\pi, t_1 = 0, t_2 = 2, t_3 = 3$ e $t_4 = 5$.

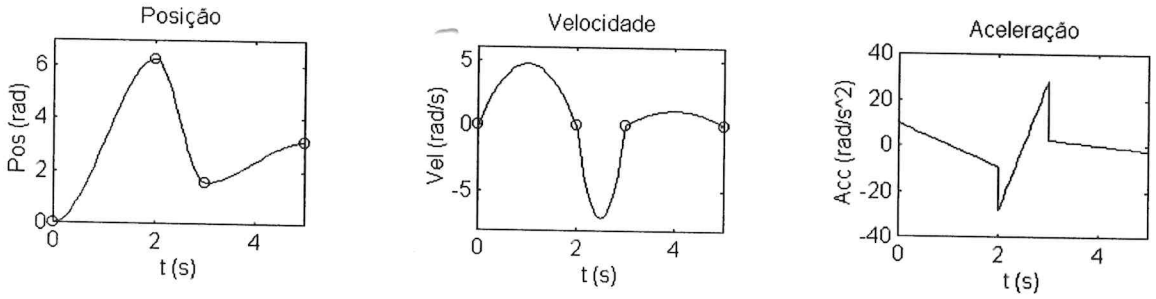


Figura 2.6 – Interpolação polinomial cúbica com velocidades intermediárias calculadas

Da mesma forma que para a figura 2.5, há descontinuidades nas acelerações, podendo assim originar os mesmos problemas apresentados anteriormente.

2.2.3 – Interpolação polinomial cúbica com aceleração contínua – solução de splines por "pontos virtuais"

A interpolação polinomial pela técnica dos "pontos virtuais" é utilizada para se garantir a continuidade dos valores de aceleração em segmentos vizinhos. Para isso é necessário estabelecer uma condição a mais que nos casos anteriores, isto é, que os valores

das acelerações sejam iguais nos pontos de intersecção de segmentos vizinhos: $\ddot{P}_{k-1}(t_k) = \ddot{P}_k(t_k) = \ddot{q}_k$ para $k=2, \ldots, n-1$, sendo n o número de pontos do caminho.

Com isso, são estabelecidas quatro condições (ou equações) a serem satisfeitas:

$$\begin{cases} P_{k-1}(t_k) = q_k & \text{definição do polinômio } P_k(t) \text{ em } t = t_k \\ P_{k-1}(t_k) = P_k(t_k) & \text{especifica continuidade das posições} \\ \dot{P}_{k-1}(t_k) = \dot{P}_k(t_k) & \text{especifica continuidade das velocidades} \\ \ddot{P}_{k-1}(t_k) = \ddot{P}_k(t_k) & \text{especifica continuidade das acelerações .} \end{cases} \tag{2.7}$$

que, para o conjunto de pontos de posição fornecido, vão resultar num sistema acoplado de quatro equações para cada ponto intermediário, três para o ponto inicial e três para o ponto final, conforme apresentado a seguir:

Sistema de quatro equações para cada ponto intermediáro, $k=2, \ldots, n-1$:

$$\begin{cases} P_{k-1}(t_k) = q_k \\ P_{k-1}(t_k) = P_k(t_k) \\ \dot{P}_{k-1}(t_k) = \dot{P}_k(t_k) \\ \ddot{P}_{k-1}(t_k) = \ddot{P}_k(t_k) \end{cases}$$

que resulta em $4(n-2)$ equações.

Sistema de três equações para o ponto inicial e três para o ponto final:

$$\begin{cases} P_1(t_1) = q_1 \\ \dot{P}_1(t_1) = \dot{q}_1 \\ \ddot{P}_1(t_k) = \ddot{q}_1 \end{cases} \qquad \begin{cases} P_{n-1}(t_n) = q_n \\ \dot{P}_{n-1}(t_n) = \dot{q}_n \\ \ddot{P}_{n-1}(t_n) = \ddot{q}_n \end{cases} \tag{2.8}$$

resultando seis equações para estes dois pontos.

Com isso tem-se $4(n-2) + 6 = 4n - 2$ equações para $4n - 4$ incógnitas, resultando em um sistema com infinitas soluções. Para se resolver esta indeterminação, uma solução possível é o uso de chamados "pontos virtuais" para se garantir continuidade dos polinômios interpoladores de posição de suas primeira e segunda derivadas.

Os pontos virtuais são duas condições iniciais a mais para este problema, dadas no tempo. Inicialmente, para este problema, não importa a localização espacial destes dois pontos mas somente seus valores temporais. Estes pontos são inseridos entre os dois

primeiros e entre os dois últimos pontos da seqüência original. Para maior clareza nas referências a estes pontos no restante desta seção, eles são reindexados conforme ilustra a figura 2.6.

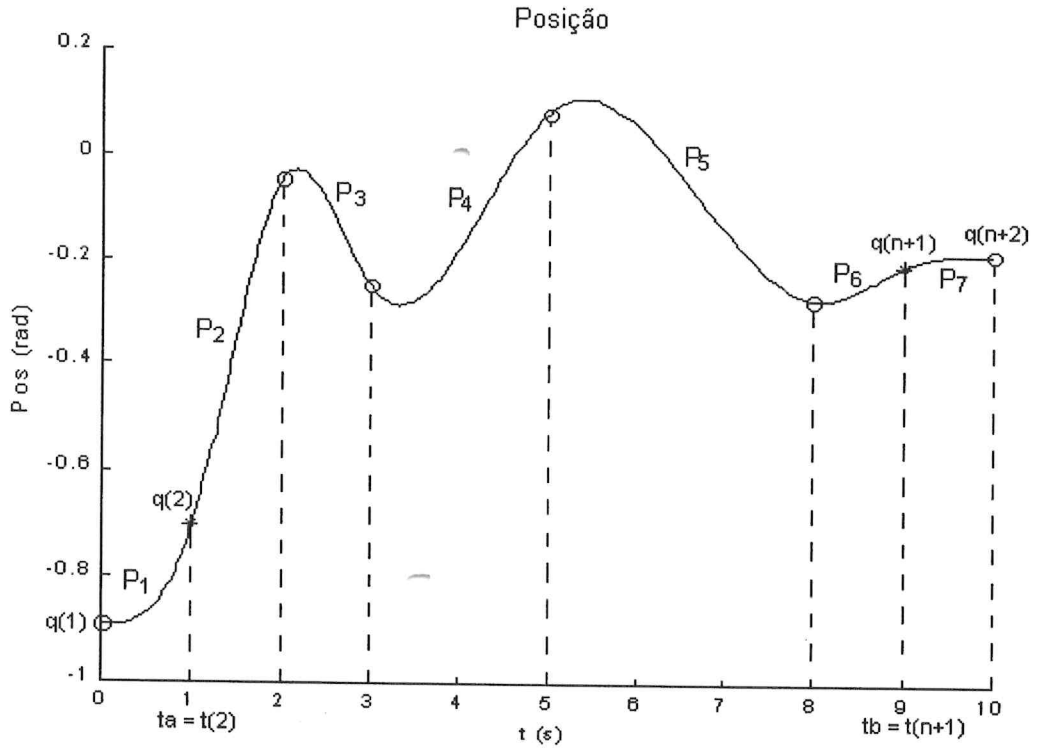


Figura 2.6 – Exemplo de curvas splines: ilustração da reindexação da seqüência de pontos e polinômios após a inserção de dois "pontos virtuais", identificados com asteriscos. Os pontos originais estão identificados pelos círculos.

A tabela 2.2 relaciona os pontos da seqüência original com a seqüência após a inserção dos dois pontos virtuais. Como resultado, tem-se então $n+2$ instantes de tempo t_k, onde t_2 e t_{n+1} correspondem aos novos instantes de tempo. Serão então determinados $n+1$ polinômios cúbicos P_1, ..., P_{n+1} para $n+2$ pontos, através das equações (2.9), (2.10) e (2.11).

Seqüência original	Pontos virtuais	Seqüência reindexada, correspondente a $(t_1, t_a, t_2, ..., t_b, t_n)$
$t_1, t_2, ..., t_n$	t_a t_b	$t_1, t_2, t_3, ..., t_n, t_{n+1}, t_{n+2}$

Tabela 2.2 – Inserção dos dois "pontos virtuais": nova indexação.

Para os $n - 2$ pontos intermediários, nomeados de $k = 3, ..., n$, resultam as seguintes $4(n - 2)$ equações, que são equivalentes às equações (2.7):

$$\begin{cases} P_{k-1}(t_k) = q_k \\ P_{k-1}(t_k) = P_k(t_k) \\ \dot{P}_{k-1}(t_k) = \dot{P}_k(t_k) \\ \ddot{P}_{k-1}(t_k) = \ddot{P}_k(t_k) \end{cases} \tag{2.9}$$

Para os pontos inicial t_1 e final t_{n+2}, resultam outras seis equações, equivalentes às equações (2.8):

$$\begin{cases} P_1(t_1) = q_i = q_1 \\ \dot{P}_1(t_1) = \dot{q}_i = \dot{q}_1 \\ \ddot{P}_1(t_k) = \ddot{q}_i = \ddot{q}_1 \end{cases} \qquad \begin{cases} P_{n+1}(t_{n+2}) = q_f = q_{n+2} \\ \dot{P}_{n+1}(t_{n+2}) = \dot{q}_f = \dot{q}_{n+2} \\ \ddot{P}_{n+1}(t_{n+2}) = \ddot{q}_f = \ddot{q}_{n+2} \end{cases} \tag{2.10}$$

Há finalmente outras seis equações para os "pontos virtuais" q_2 e q_{n+1}:

$$\begin{cases} P_1(t_2) = P_2(t_2) \\ \dot{P}_1(t_2) = \dot{P}_2(t_2) \\ \ddot{P}_1(t_2) = \ddot{P}_2(t_2) \end{cases} \qquad \begin{cases} P_n(t_{n+1}) = P_{n+1}(t_{n+1}) \\ \dot{P}_n(t_{n+1}) = \dot{P}_{n+1}(t_{n+1}) \\ \ddot{P}_n(t_{n+1}) = \ddot{P}_{n+1}(t_{n+1}) \end{cases} \tag{2.11}$$

O sistema resulta então em $4(n-2) + 6 + 6 = 4n + 4$ equações para $4(n+1) = 4n + 4$ incógnitas – pois cada polinômio contribui com 4 incógnitas.

O problema imediato que surge na resolução do sistema representado pelas equações (2.9), (2.10) e (2.11) é o da demanda computacional à medida que n aumenta. Deve-se considerar também que para cada junta do manipulador é necessária a resolução de um sistema desta ordem. Com isso, deve-se buscar uma metodologia que seja numericamente eficiente na resolução deste problema, que a princípio deve funcionar em tempo real para a geração da trajetória, competindo e tendo que se sincronizar com outros processos do controlador do robô.

Conforme apresentado em [BCMP98], [Chapra&Canale], [Sciavicco&Siciliano96] e [Sedgewick83], a forma numericamente eficiente de se determinar as $n+1$ splines interpolantes é a partir do cálculo das acelerações $\ddot{P}_k(t)$.

Como o polinômio genérico interpolante $P_k(t)$ é uma cúbica em t, a sua segunda derivada é linear em relaçãoa t e pode ser escrita como:

$$\ddot{P}_k(t) = \frac{\ddot{P}_k(t_k)}{\Delta t_k}(t_{k+1} - t) + \frac{\ddot{P}_k(t_{k+1})}{\Delta t_k}(t - t_k) \tag{2.12}$$

com $k = 1, 2, ..., n+1$ e $\Delta t_k = (t_{k+1} - t_k)$. Integrando-se a equação (2.12) duas vezes obtém-se o polinômio de posição, escrito da seguinte forma:

$$P_k(t) = \frac{\ddot{P}_k(t_k)}{6\Delta t_k}(t_{k+1} - t)^3 + c_k(t - t_k) + \frac{\ddot{P}_k(t_{k+1})}{6\Delta t_k}(t - t_k)^3 + d_k(t_{k+1} - t)$$

As constantes de integração c_k e d_k são determinadas nos tempos $t = t_k$ e $t = t_{k+1}$, respectivamente:

$$P_k(t) = P_k(t_k) \Rightarrow d_k = \frac{P_k(t_k)}{\Delta t_k} - \frac{\ddot{P}_k(t_k)}{6}\Delta t_k \qquad P_k(t) = P_k(t_{k+1}) \Rightarrow c_k = \frac{P_k(t_{k+1})}{\Delta t_k} - \frac{\ddot{P}_k(t_{k+1})}{6}\Delta t_k$$

resultando em

$$\begin{aligned} P_k(t) = {} & \frac{\ddot{P}_k(t_k)}{6\Delta t_k}(t_{k+1} - t)^3 + \frac{\ddot{P}_k(t_{k+1})}{6\Delta t_k}(t - t_k)^3 + \\ & + \left(\frac{P_k(t_{k+1})}{\Delta t_k} - \frac{\ddot{P}_k(t_{k+1})}{6}\Delta t_k\right)(t - t_k) + \left(\frac{P_k(t_k)}{\Delta t_k} - \frac{\ddot{P}_k(t_k)}{6}\Delta t_k\right)(t_{k+1} - t) \end{aligned} \tag{2.13}$$

A equação (2.13), ainda que numa representação não muito usual, é menos complicada do que parece pois contém apenas dois coeficientes ainda não são conhecidos, que são os valores das acelerações no início e no fim do segmento k. Assim, determinando-se estes valores, a equação (2.13) representa unicamente o polinômio $P_k(t)$ a partir dos valores inicial e final de posição e aceleração neste segmento. A questão torna-se, com isso, determinar os valores apropriados destas acelerações. Isto pode ser feito utilizando a condição de continuidade das velocidades, isto é, $\dot{P}_k(t_k) = \dot{P}_{k+1}(t_k)$. Derivando-se (2.13) para o segmento k obtemos a seguinte representação para o perfil de velocidade:

$$\dot{P}_k(t) = -\frac{\ddot{P}_k(t_k)}{2\Delta t_k}(t_{k+1} - t)^2 + \frac{\ddot{P}_k(t_{k+1})}{2\Delta t_k}(t - t_k)^2 + \frac{P_k(t_{k+1}) - P_k(t_k)}{\Delta t_k} + \frac{\ddot{P}_k(t_k) - \ddot{P}_k(t_{k+1})}{6}\Delta t_k \tag{2.14}$$

O mesmo é feito para o segmento *k+1*. Como são conhecidos *n* valores de posição, é dada a condição de continuidade nos dois pontos virtuais q_2 e q_{n+1}, e os valores iniciciais e finais de velocidade e aceleração são conhecidos, utilizamos a condição $\dot{P}_k(t_k) = \dot{P}_{k+1}(t_k)$ para escrever um sistema de *n* equações com *n* incógnitas:

$$\begin{cases} \dot{P}_1(t_2) = \dot{P}_2(t_2) \\ \vdots \\ \dot{P}_n(t_n) = \dot{P}_{n+1}(t_{n+1}) \end{cases}$$

A partir daí, então, é possível escrever um sistema de equações lineares do tipo **Ax=b**:

$$A \begin{bmatrix} \ddot{P}_2(t_2) \\ \vdots \\ \ddot{P}_{n+1}(t_{n+1}) \end{bmatrix} = \boldsymbol{b}$$

sendo **x** o vetor com as acelerações intermediárias a se determinar e **b** um vetor com termos conhecidos, associados aos tempos de cada posição – já que através de (2.14) pode-se conhecer $\dot{P}_k(t_{k+1})$ e $\dot{P}_{k+1}(t_{k+1})$ para *k = 1, 2, ..., n*. A matriz **A** é tridiagonal e idêntica para todas as juntas do manipulador, pois seus coeficientes dependem apenas dos intervalos de tempo especificados pelo usuário.

Resolvendo-se este sistema, obtêm-se os valores das acelerações intermediárias e determinam-se completamente as splines de posição, velocidade e aceleração para cada segmento da trajetória que deve ser executada por uma junta.

Definindo-se $\phi_k = \ddot{P}_k(t_k)$ e sabendo-se que $q_k = P_k(t_k)$, podemos reescrever as equações de posição, velocidade e aceleração da seguinte forma:

$$\begin{aligned} P_k(t) &= \alpha_k(t_{k+1} - t)^3 + \beta_k(t - t_k)^3 + \gamma_k(t_{k+1} - t) + \delta_k(t - t_k) \\ \dot{P}_k(t) &= -3\alpha_k(t_{k+1} - t)^2 + 3\beta_k(t - t_k)^2 - \gamma_k + \delta_k \\ \ddot{P}_k(t) &= 6\alpha_k(t_{k+1} - t) + 6\beta_k(t - t_k) \end{aligned} \qquad (2.15)$$

onde $\alpha_k = \frac{\phi_k}{6\Delta t_k}, \beta_k = \frac{\phi_{k+1}}{6\Delta t_k}, \gamma_k = \frac{q_k}{\Delta t_k} - \frac{\phi_k \Delta t_k}{6}, \delta = \frac{q_{k+1}}{\Delta t_k} - \frac{\phi_{k+1}\Delta t_k}{6}$.

A figura 2.7 ilustra os perfis de posição, velocidade e aceleração para uma trajetória com $q_1 = 0, q_2 = 2\pi, q_3 = \pi/2, q_4 = 2\pi$, $t_1 = 0, t_2 = 2, t_3 = 3$ e $t_4 = 5$. Os pontos virtuais foram alocados em $t_a = 0,5$ e $t_b = 4,5$.

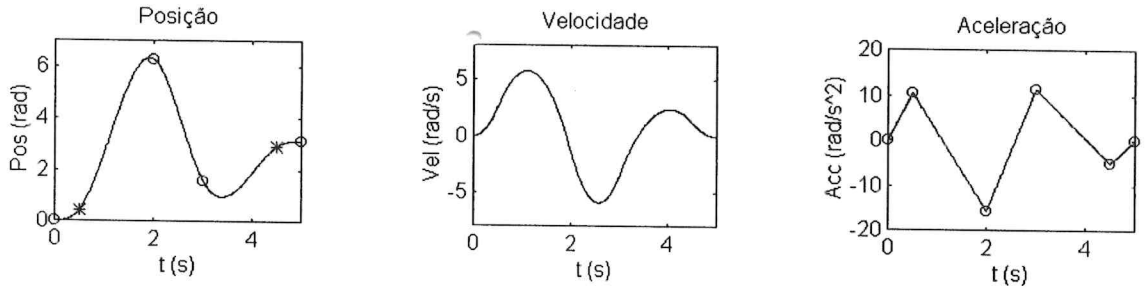
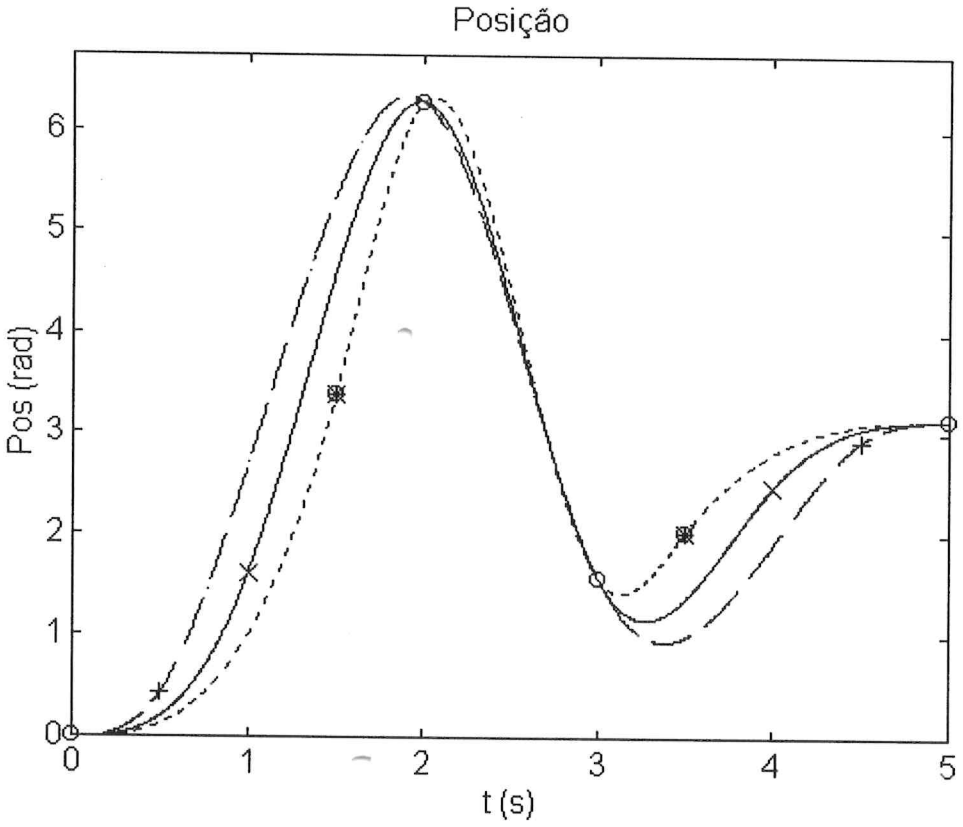


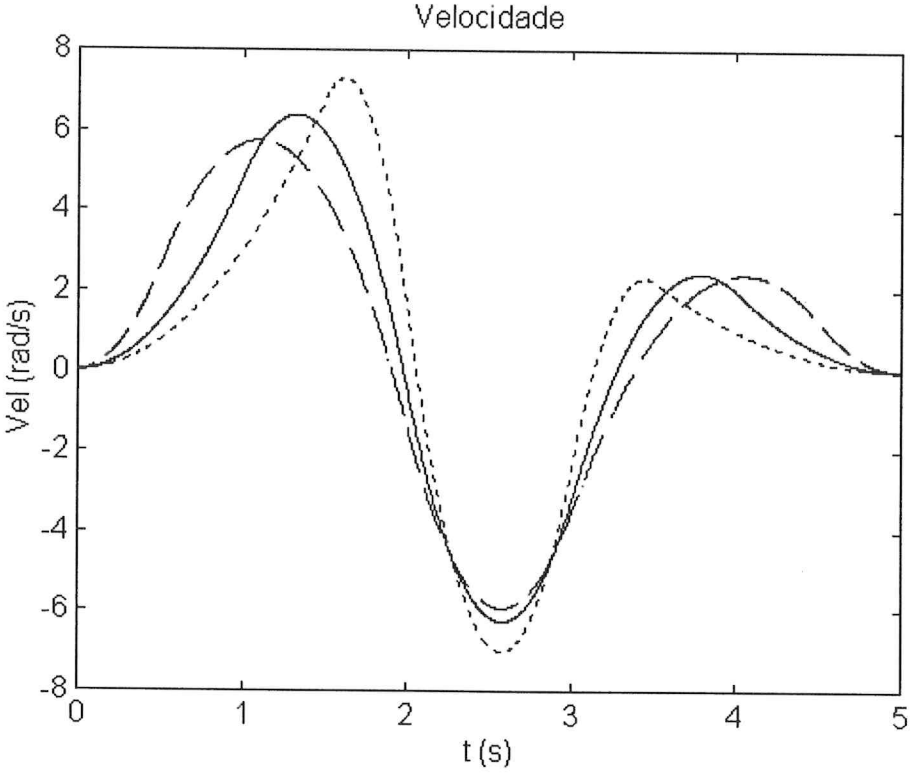
Figura 2.7 – Solução com pontos virtuais (asteriscos)

A vantagem desta solução, ainda que matematicamente mais trabalhosa, é que garante a continuidade na aceleração. Consequentemente, os torques gerados pelo controlador são contínuos e trajetórias mais suaves podem ser executadas pelo manipulador.

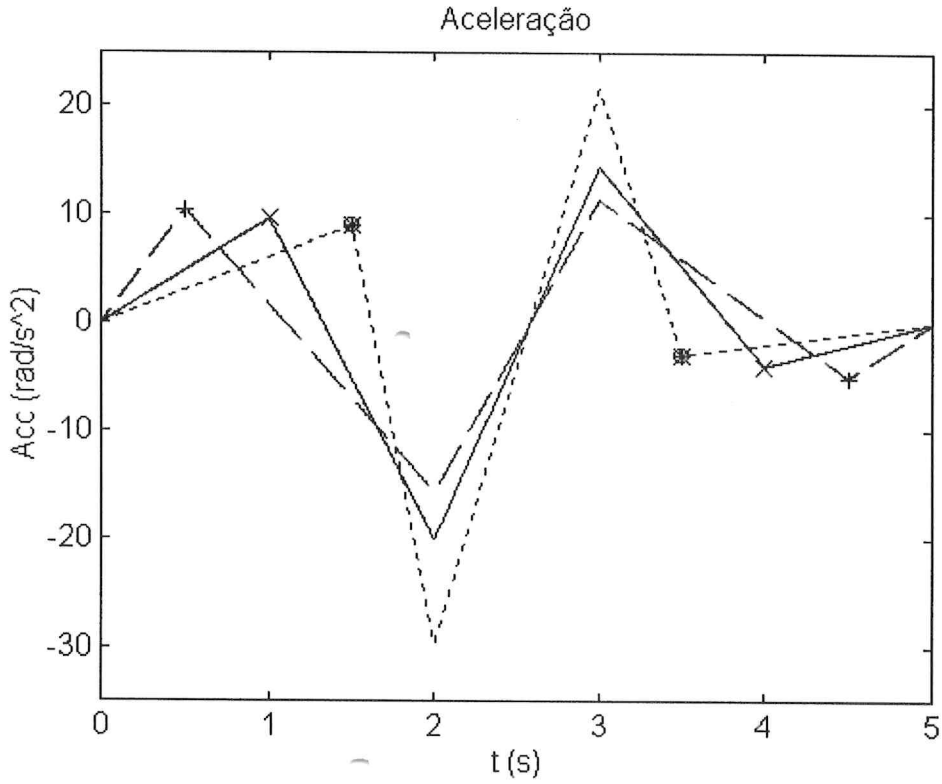
O efeito do posicionamento temporal dos pontos virtuais sobre o resultado das acelerações intermediárias – e consequentemente velocidades e posições – pode ser visto na figura 2.8. Os resultados apresentados são para $q_1 = 0, q_2 = 2\pi, q_3 = \pi/2, q_4 = 2\pi$, $t_1 = 0, t_2 = 2, t_3 = 3$ e $t_4 = 5$ com as seguintes variações dos tempos virtuais: t_a=*0,5* e t_b=*4,5 (linha tracejada),* t_a = *1,5* e t_b= *3,5 (linha pontilhada)* e t_a =*1* e t_b=*4 (linha contínua).* Evidentemente, em qualquer das configurações as splines atingem e interpolam os pontos desejados de posição. Vemos também que, à medida que t_a e t_b se aproximam respectivamente dos pontos intermediários t_3 e t_{n+1}, as acelerações nos segmentos intermediários tendem a aumentar, já que os instantes de tempo ali envolvidos são menores. Por outro lado, se t_a e t_b se aproximam dos tempos inicial e final as acelerações geradas nos segmentos intermediários são menores e consequentemente o movimento da junta é mais suave.


Posição
Pos (rad)
t (s)
0
1
2
3
4
5
6

a)


Velocidade
Vel (rad/s)
t (s)
8
6
4
2
0
-2
-4
-6
-8
0
1
2
3
4
5

b)



c)

Figura 2.8 – Efeito da localização dos pontos virtuais nos perfis de a) posição, b) velocidade e c) aceleração

Capítulo 3: OBJETO DO TRABALHO: O ROBÔ INTER

3.1 – Características Gerais

Conforme mencionado no capítulo 1, este trabalho foi desenvolvido e aplicado sobre o robô Inter, do Laboratório de Robótica da UFSC. A configuração deste manipulador é do tipo SCARA, ilustrada na figura 3.1, e possui quatro graus de liberdade. SCARA é sigla de *Selective Compliant Articulated Robot for Assembly*. Nesta configuração as duas primeiras juntas são de rotação, girando em torno de eixos verticais e trabalhando portanto num plano horizontal. A terceira junta é de translação e realiza deslocamentos no sentido vertical. O efetuador final realiza movimentos rotacionais sobre um eixo vertical, podendo ser adequado de acordo com o tipo de tarefa a ser executada.

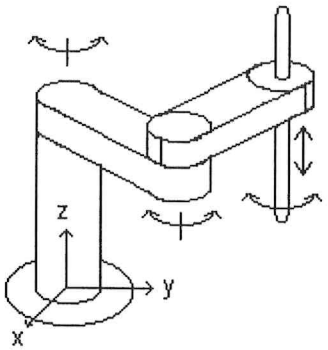


Figura 3.1 – Configuração SCARA

Na figura 3.2 é apresentado o robô Inter juntamente com sua unidade de alimentação e controle. Este manipulador é de porte industrial e sua maior característica é ter uma arquitetura aberta, o que possibilita a implementação e testes de diferentes técnicas de controle. Uma observação que se faz necessária desde o início é que convencionamos uma notação para elos e juntas diferente daquela normalmente encontrada na literatura. A convenção aqui adotada visa concordar com a do ambiente de programação do robô, XOberon. Isto se justifica pois na representação de um vetor nesta linguagem o primeiro

termo é referenciado com o índice 0, o segundo com o índice 1 e assim por diante. Desta forma referenciamos a primeira junta e o primeiro elo móvel do manipulador com o índice zero para que haja concordância com a representação vetorial no XOberon.

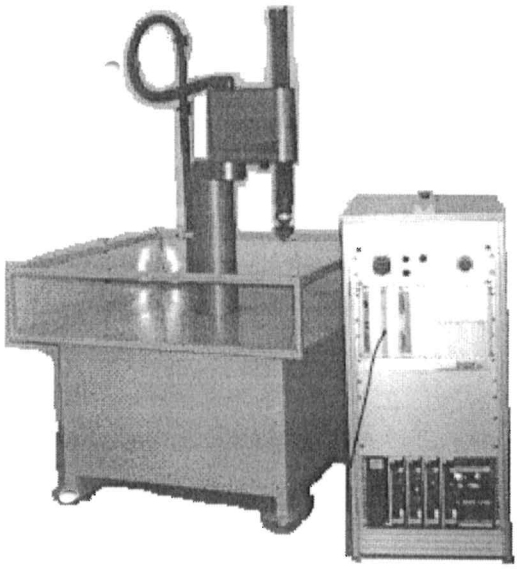


Figura 3.2 – Robô Inter e armário de controle

3.2 – Espaço de Trabalho

O centro do sistema de coordenadas cartesianas utilizado como referência para definir a posição e a orientação do robô foi alocado no centro do elo da base, conforme sugere a figura 3.1, e no plano da mesa em que está instalado. O espaço de trabalho do efetuador é definido a partir deste sistema de coordenadas. Para maiores detalhes sobre a alocação dos sistemas de coordenadas deste manipulador pode-se consultar [GWG98].

3.2.1 – Espaço de Trabalho - Plano X-Y

Na figura 3.3 está ilustrado o espaço de trabalho do robô Inter no plano x-y, gerado exclusivamente pelos movimentos de rotação das juntas 0 e 1. O elo 0 realiza rotações de ± 2,25 rad (aproximadamente ± 129°). Já o elo 1 é limitado em ± 1,9 rad (≈ ±109°).

Aqui cabem algumas observações sobre singularidades, conforme [Spong&Vidyasagar89]: em posições singulares há movimentos infinitesimais que não são atingíveis, ou seja, o elo em questão não pode se mover em determinadas direções e consequentemente perde-se um ou mais graus de liberdade. Também, nas proximidades de configurações singulares, não existirá solução única para o problema da cinemática inversa - neste caso haverá infinitas soluções ou não haverá solução e velocidades limitadas no espaço cartesiano podem gerar velocidades ilimitadas no espaço de juntas - isso também vale para forças e torques. Cabe notar ainda que uma movimentação de θ_1^+ para θ_1^-, no espaço cartesiano, a junta 1 teria que passar por uma região singular, o que o próprio software de controle do manipulador não permite ao interromper a movimentação quando se atinge um limite de 0,15rad para a posição singular.

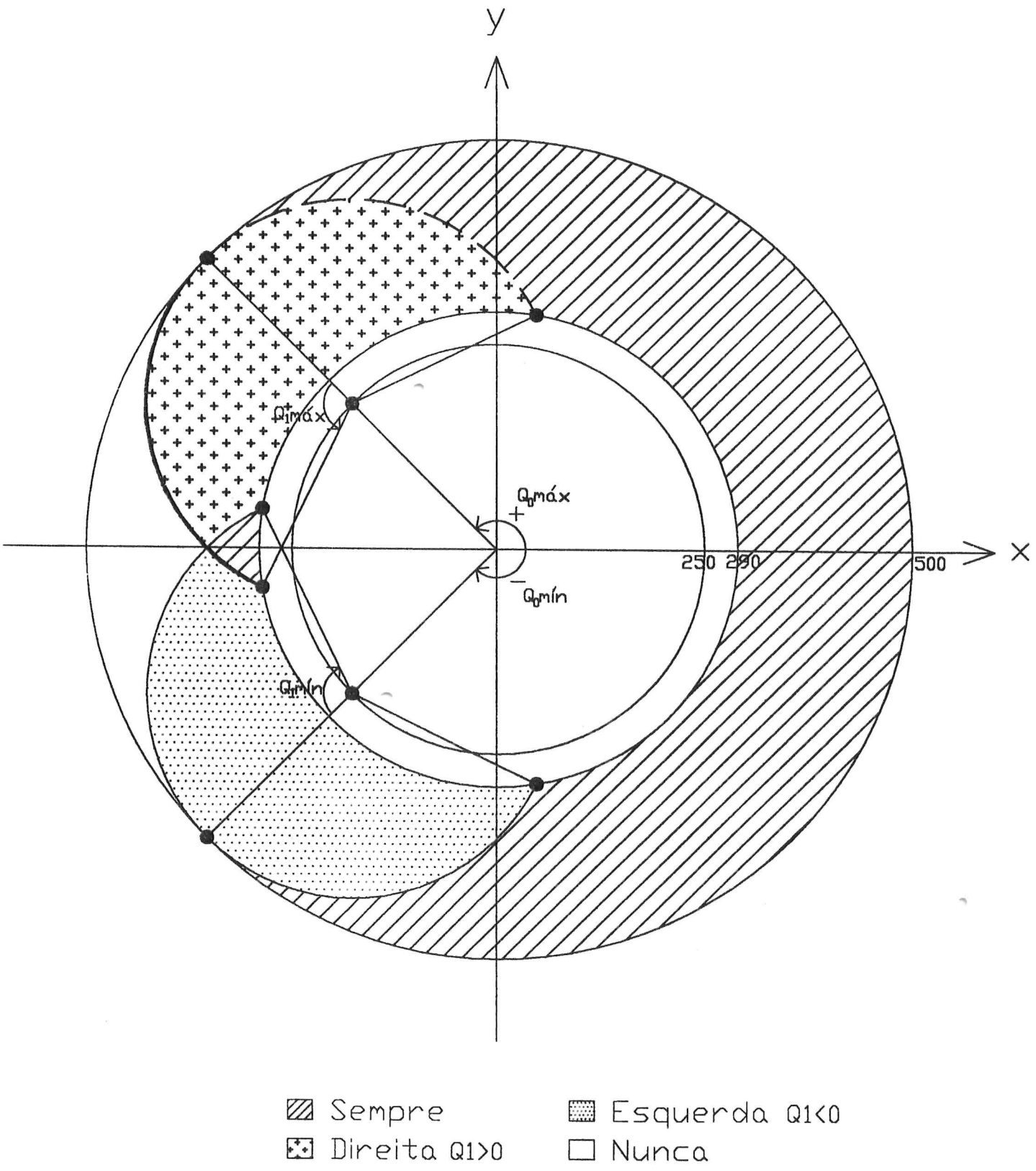


Figura 3.3 – Espaço de trabalho no plano *xy*

$Q_{0máx} = |Q_{0mín}| = 2{,}25\text{rad}$; $Q_{1máx} = |Q_{1mín}| = 1{,}9\text{rad}$

A figura 3.4 ilustra a trajetória[1] que seria seguida pela extremidade do elo 1 caso se desejasse um deslocamento do ponto "**o**" para o ponto "**p**" no espaço cartesiano, estando os elos numa posição singular. Nesta situação, o software de controle identifica o problema e não permite nenhum tipo de movimento do robô.

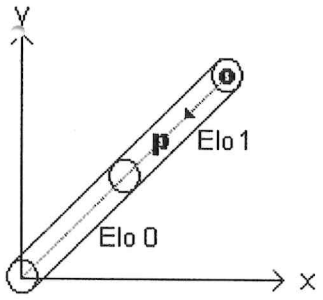


Figura 3.4 – Uma configuração singular

Configurações à Direita e à Esquerda

Na figura 3.3 estão identificadas duas áreas no espaço de trabalho do robô Inter identificadas como "Direita" (AD) e "Esquerda" (AE). Estas áreas são geradas a partir de $\theta_1 > 0$ para AD e $\theta_1 < 0$ para AE. Conforme será visto, estas áreas somente são atingidas pelo efetuador final em configurações particulares das juntas 0 e 1.

Ao se trabalhar com comandos de movimentação no espaço cartesiano, uma destas duas áreas não poderá ser atingida devido à necessidade de se passar obrigatoriamente por uma configuração singular. Isto diminui o espaço de trabalho do manipulador para apenas uma das áreas, AD ou AE, além da região atingível em qualquer configuração. No espaço de juntas, entretanto, o conceito de "configuração singular" não existe e consequentemente não há as limitações impostas por ela e citadas anteriormente. Com isso é possível atingir ambas as áreas AD e AE do plano *xy*. Em qualquer dos casos, entretanto, estas regiões só são atingidas com $\theta_1 > 0$ para a região AD e $\theta 1 < 0$ para a região AE.

[1] O módulo gerador de trajetórias original do robô Inter só trabalha com deslocamentos retilíneos no espaço cartesiano.

A fim de esclarecer como estas regiões podem ser atingidas, descreveremos a seguir o caso da área "Direita", identificada com símbolos de (+) na figura 3.3.

Suponha que a junta 0 tenha sofrido um deslocamento angular de 2,25 rad a partir da origem, ou seja, está no seu deslocamento máximo positivo (θ_0 = 2,25 rad). Suponha ainda que o elo 1 esteja "alinhado" com o elo 0, ou seja $\theta_1 = 0$. Se a partir desta posição a junta 1 realizar um deslocamento angular de 0 a -1,9 rad, a extremidade do elo 1 percorrerá a linha tracejada da figura 1.10, não sendo possível atingir o interior da região (+). Isto é válido mesmo que a junta 0 sofra algum deslocamento. Por outro lado, se a partir desta posição de alinhamento a extremidade do elo 1 realizar um deslocamento de 0 a +1,9 rad, irá percorrer o caminho da linha em negrito. Aqui porém, se o elo 0 sofrer deslocamentos angulares, será possível atingir qualquer ponto da região (+) e da região comum, com algum deslocamento positivo de θ_1 .

O interior da região (+), portanto, só é atingido pela extremidade do elo 1 se a junta 0 não estiver no seu deslocamento angular máximo e, além disso, se a junta 1 se deslocar "da direita para a esquerda", ou seja, somente se $\theta_1 > 0$. Claro que se o limite de $\theta_0 = 2{,}25$ rad fosse maior, digamos 3 rad, seria possível atingir o interior desta área com um θ_1 negativo. Como θ_0 é limitado via software em 2,25 rad, isso não é possível. Assim, vemos que o interior da região denominada "Direita" só é atingido com $\theta_1 > 0$.

Para a região "Esquerda" vale o análogo, ou seja, seu interior só é atingido pela extremidade do elo 1 com $\theta_1 < 0$.

As áreas em branco claramente nunca são atingidas, qualquer que seja a combinação dos ângulos θ_0 e θ_1.

3.2.2 – Espaço de Trabalho - Eixo Z

Na figura 3.5 estão definidas as posições máxima e mínima atingidas pelo efetuador final na direção z, representadas em milímetros. Os deslocamentos verticais do elo 2 podem ser referenciados segundo o sistema da junta ou segundo o sistema cartesiano, mostrados na figura. Há duas representações para o efetuador final: uma pontilhada e outra em linha cheia. A primeira se refere aos limites mecânicos de deslocamento vertical do elo 2,

determinados por sensores de fim de curso. A segunda, em linha cheia, se refere aos limites configurados em software, no ambiente XOberon, para estes deslocamentos. Em termos práticos, para o acionamento da junta 2 apenas os limites configurados em software são considerados. Maiores informações sobre esta diferenciação entre limites físicos e de software, que existe também pra as demais juntas, pode-se consultar [GWG98].

Portanto, no espaço de juntas, vemos que o espaço de trabalho na direção z fica compreendido entre 140 e 400 mm. Já no espaço cartesiano trabalha-se com valores entre 265 e 525 mm. Estes limites no eixo z podem ser atingidos em qualquer ponto do espaço de trabalho definido para o plano xy.

Evidentemente, estas considerações que fizemos sobre o espaço de trabalho no plano xy e no eixo z do espaço cartesiano, conforme [GWG98], foram observadas no desenvolvimento do gerador de trajetórias para o robô Inter a fim de torná-lo mais fiel às características construtivas do manipulador.

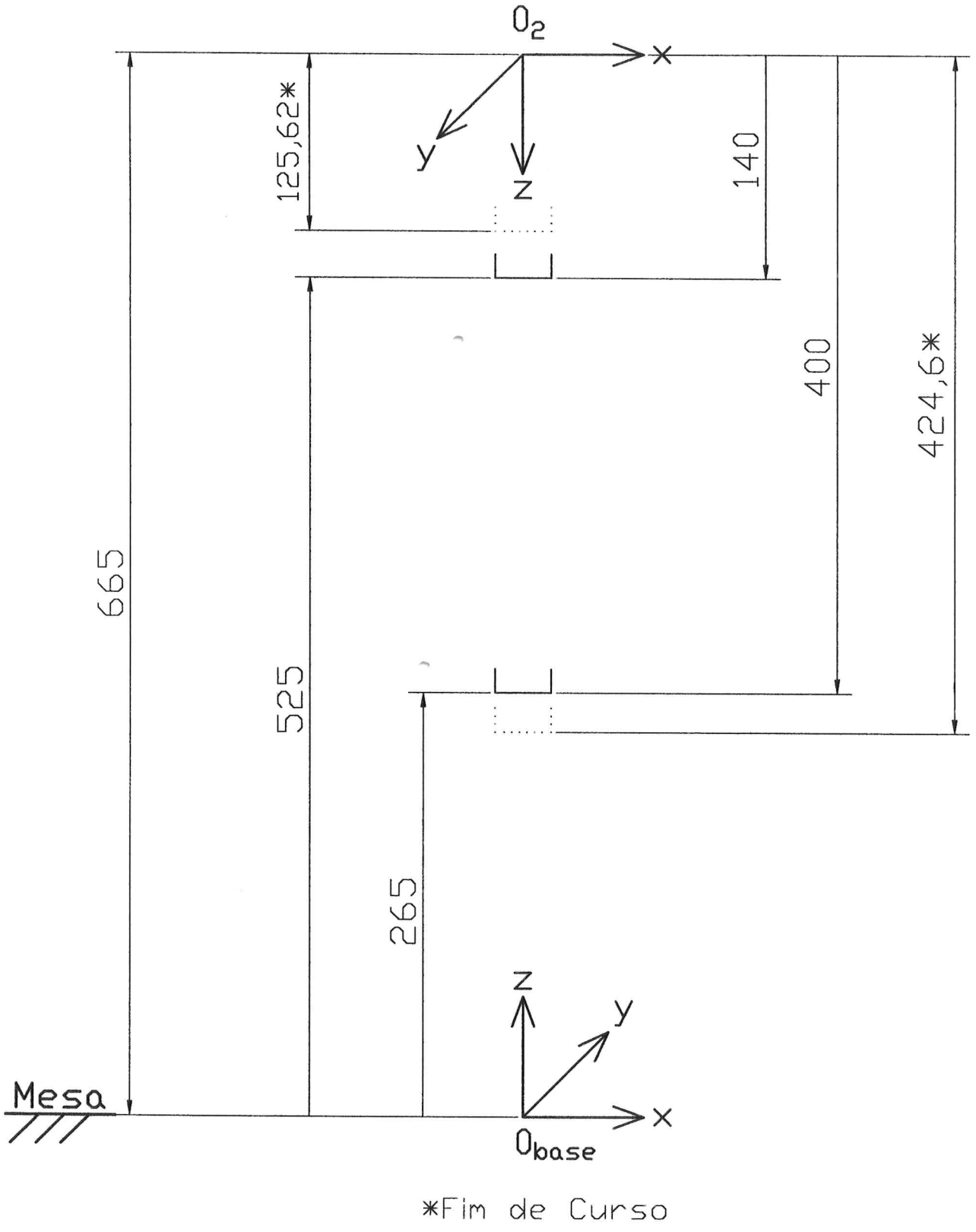


Figura 3.5 – Espaço de trabalho do robô Inter no eixo *z*. Espaço da junta: 140 a 400 mm; espaço cartesiano: 265 a 525 mm

3.3 – Modelagem Cinemática do Robô Inter

Cinemática Direta

A cinemática direta de robôs manipuladores trata do "mapeamento" dos acionamentos no espaço de juntas e a correspondência no espaço cartesiano. Nesta seção apresentaremos apenas os resultados da cinemática direta do robô Inter, e o desenvolvimento destas relações pode ser visto em [GWG98].

Os parâmetros de Denavit-Hartenberg para o robô Inter são apresentados na tabela 3.1:

No. Elo	α_i	a_i (mm)	d_i (mm)	θ_i
0	0	250	665	θ_0
1	π	250	0	θ_1
2	0	0	d_2	0
3	0	0	0	θ_3

Tabela 3.1 – Parâmetros de Denavit-Hartenberg para o robô Inter

A matriz de transformação T_b^3 do sistema de coordenadas O_3 para O_{base} é:

$$T_b^3 = \begin{bmatrix} C3C01+S3S01 & -S3C01+C3S01 & 0 & 0{,}25(C01+C0) \\ C3S01-S3C01 & -S3S01-C3C01 & 0 & 0{,}25(S01+S0) \\ 0 & 0 & -1 & -d2+0{,}665 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

onde: $C3 = \cos(\theta_3)$, $S01 = \text{sen}(\theta_0 + \theta_1)$ etc.

Com isso, dadas as posições das juntas 0, 1 e 2, respectivamente θ_0 , θ_1 e d_2, a posição do efetuador final no espaço cartesiano será:

$$\begin{cases} x = 0{,}25(\cos(\theta_0+\theta_1)+\cos\theta_0) \\ y = 0{,}25(\text{sen}(\theta_0+\theta_1)+\text{sen}\,\theta_0) \\ z = -d_2+0{,}665 \end{cases}$$

Cinemática inversa

O problema da cinemática inversa trata de encontrar posições de juntas necessárias para se atingir uma posição {x, y, z} no espaço cartesiano com uma orientação θ.

Para um mecanismo articulado, caso das juntas 0 e 1 do robô Inter e ilustrado na figura 3.7, através das relações trigonométricas e da lei dos cossenos obtêm-se as seguintes relações para θ_1 e θ_2 [Spong&Vidyasagar89]:

$$\theta_1 = arctg\left(\pm\frac{\sqrt{1-T^2}}{T}\right)$$

$$\theta_0 = arctg\left(\frac{y}{x}\right) - arctg\left(\frac{l_1 \operatorname{sen}\theta_1}{l_0 + l_1\cos\theta_1}\right)$$

onde $T = \dfrac{x^2 + y^2 - l_0^2 - l_1^2}{2l_0 l_1}$. Para o robô Inter, $l_0 = l_1 = 250$ mm.

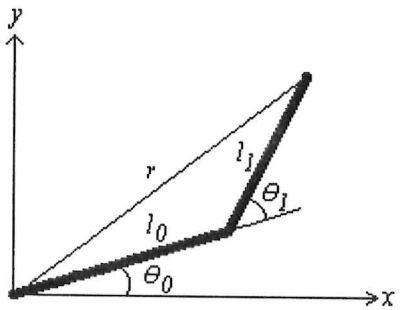


Figura 3.7 – Esquemático de um mecanismo articulado

Para a junta 2, que é prismática, a relação que transforma posições z do espaço da tarefa para posições d_2 da junta é simplesmente $d_2 = D - z$, onde D é a distância vertical entre o sistema de coordenadas da base e o sistema de coordenadas da junta 2. Para o robô Inter, $D = 0{,}665\ mm$ e portanto

$$d_2 = 0{,}665 - z$$

3.4 – Especificações do Robô Inter e Dados de Configuração

Nesta seção apresentamos um resumo das especificações e dados de configuração do robô Inter.

A tabela 3.2 apresenta os dados de configuração do robô Inter:

	Junta 0	**Junta 1**	**Junta 2**	**Junta 3**
Deslocamento máx. negativo (rad)	-2,5102	-2,0980	0,1256 (m)	-2,2837
Deslocamento máx. positivo (rad)	2,5335	2,0992	0,4246 (m)	3,3644
Média dos deslocamentos (rad)	± 2,5219	± 2,0986	0,2751 (m)	± 2,82405
Posição de sincronização (rad)	-2,5102	2,0992	0,1256 (m)	-2,2837
Posição. mín. de configuração (rad)	-2.25	-1,9	0,14	-2,05
Posição. máx. de configuração (rad)	2,25	1,9	0,4	3,0
Velocidade máx. suportada (rad/s)	3,77	3,77	0,888 (m/s)	23,27
Velocidade. máx. configurada (rad/s)	3,00	3,00	0,888 (m/s)	20
Velocidade. mín. configurada (rad/s)	-3,00	-3,00	-0,888 (m/s)	-20
Torque máximo suportado (Nm)	± 333	± 157	± 3,493	± 33,53
Aceleração máx. desenvolvida(rad/s^2)	± 80	± 100	± 3000 (mm/s^2)	± 500
Aceleração medida (rad/s^2)	95 (θ_1 estendido) 139 ($\theta_1$retraído)	175		

Tabela 3.2 – Resumo dos dados de configuração

A aceleração medida vem de testes práticos realizados pelo fabricante. Nestes testes, θ_1 estendido refere-se à aceleração estimada com o elo 1 "estendido", ou seja, com $\theta_1=0$. Da mesma forma, θ_1 retraído refere-se à aceleração medida com o elo 1 "retraído", ou seja, com θ_1=-1,9 rad.

No robô Inter, as informações da tabela 3.2, entre outras, são configuradas no arquivo *ROBAllBoot.Config* e enviadas para o controlador durante a sua inicialização. Estes dados correspondem aos limites que devem ser respeitados durante o acionamento do

manipulador. Nota-se que os limites mecânicos do robô são maiores que aqueles configurados em software. Isto é feito por dois motivos. Primeiro para a autoproteção do equipamento contra valores excessivos gerados pelo usuário ou algum erro de cálculo, como por exemplo torques muito altos e consequente dano ao equipamento. Em segundo lugar, para se evitar a interrupção amiúde do manipulador pelo acionamento dos sensores de fim de curso e consequente execução do estado *emergency*. O acionamento destes sensores pode ser causado por oscilações da junta antes de estabilizar em uma posição desejada próxima à de um deles. Caso algum destes limites configurados em software seja atingido, o manipulador pára imediatamente seu acionamento e entra no chamado "estado de emergência".

As informações da tabela 3.2 foram levadas em consideração no desenvolvimento do gerador de trajetórias em questão. Durante os cálculos para a geração das trajetórias de referência estes valores são comparados para que se tenha como resultado trajetórias factíveis pelo manipulador.

3.5 – O Ambiente XOberon

O ambiente computacional utilizado como interface com o robô e para a sua programação é o XOberon. Este sistema é baseado na linguagem Oberon, desenvolvida na Universidade Técnica de Zurique - Suíça. Oberon segue características das linguagens Pascal e Modula-2. O XOberon foi desenvolvido no Instituto de Robótica desta Universidade e é uma extensão de Oberon. Esta extensão confere à linguagem as características de sistema operacional em tempo real.

Para exemplificar a necessidade de suporte a processamento em tempo real no caso da robótica, suponha que dispõe-se de um robô cujo valor de controle deve ser atualizado a cada 5 milisegundos. Caso o processamento envolvido seja muito complexo e o computador só consiga completar os cálculos a cada 10ms, o programa será parado. Se o programa não fosse interrompido, o sinal de controle teria um atraso que aumentaria 5ms em cada ciclo. Isto pode acarretar conseqüências graves, principalmente quando for necessário algum tipo de sincronização com outros robôs ou sistemas mecatrônicos.

Para aplicações em robótica é fundamental uma linguagem confiável e que realmente possua as características de sistema operacional em tempo real e o XOberon atende a estes requisitos. Nele é possível definir tarefas e o tempo de execução destas tarefas. Se por algum motivo o tempo estipulado não puder ser cumprido, o sistema detectará o erro e o programa será finalizado, enviando uma mensagem de erro.

Como a linguagem XOberon é bastante nova e está em fase de aperfeiçoamento, ainda não existe um manual completo que sirva como guia para seu aprendizado. O que se dispõe atualmente no Laboratório de Robótica são dois livros sobre Oberon e duas apostilas.

A figura 3.8 ilustra uma seção do ambiente XOberon. A área de trabalho do XOberon é dividida em quatro partes, cada uma sendo um quadro ou área diferente. No monitor do computador é mostrada uma área de cada vez e para passar de uma área para outra utiliza-se um recurso chamado *navigator*.

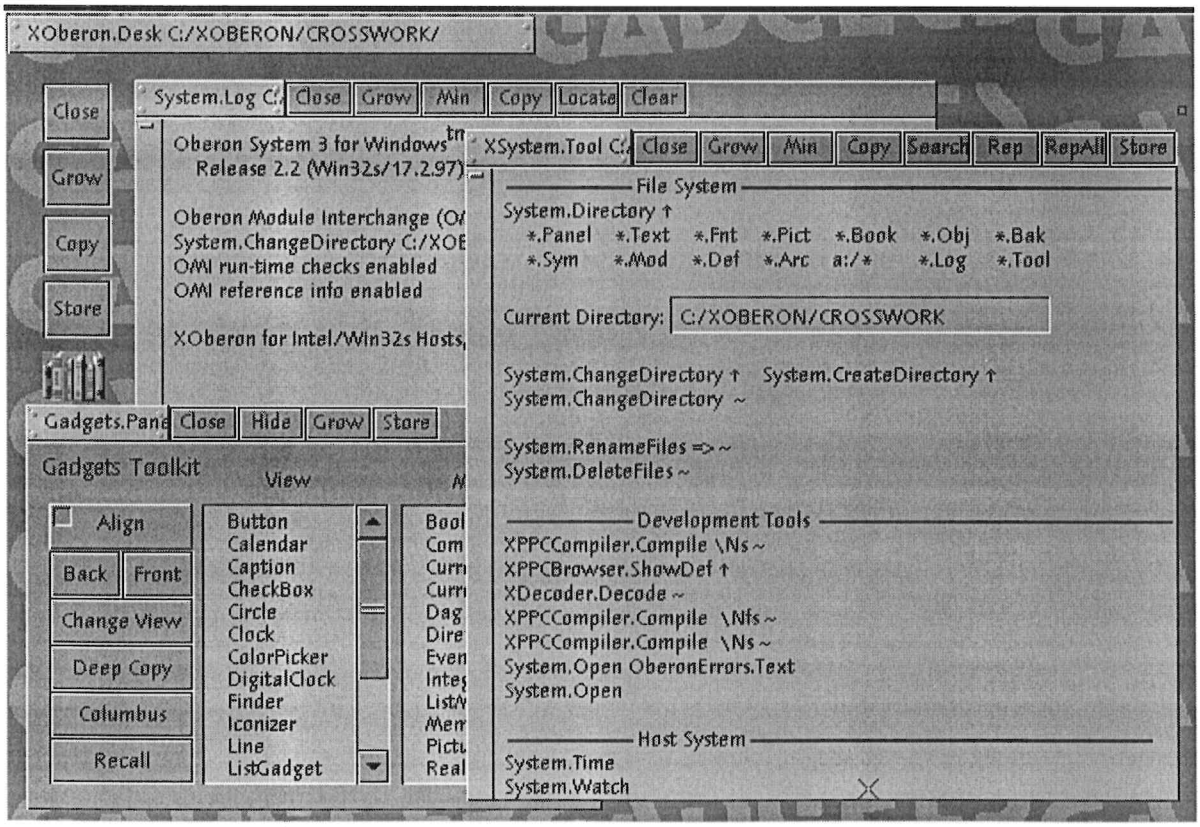
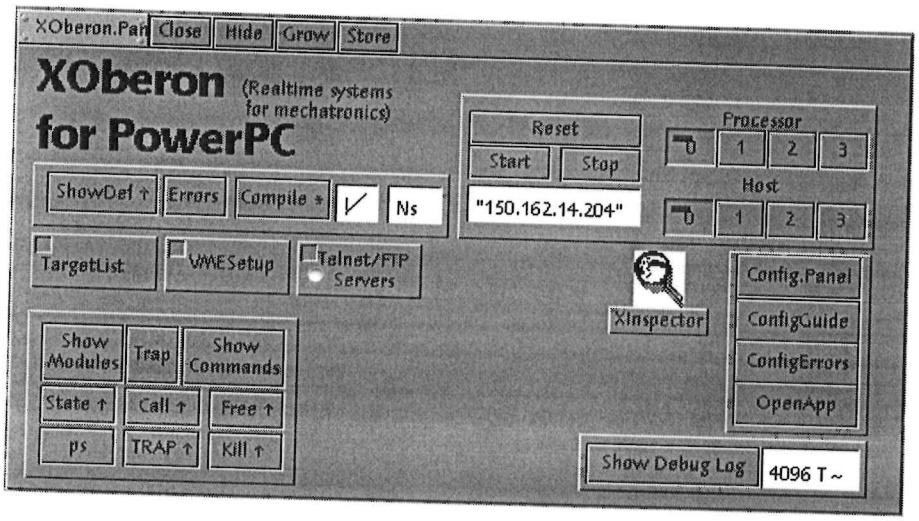
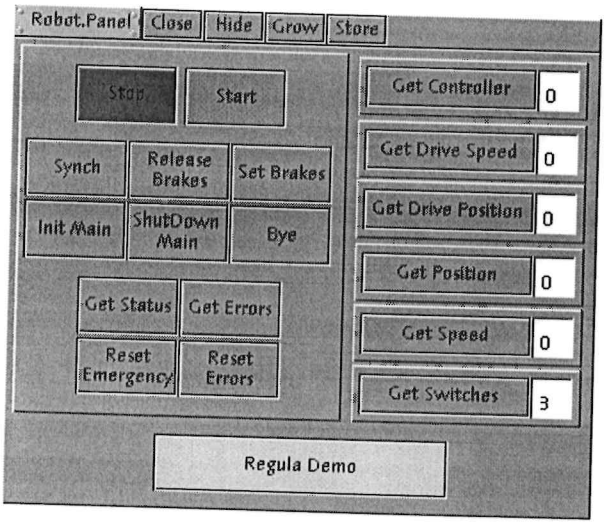


Figura 3.8 – Uma tela típica do ambiente XOberon

As interfaces mais importantes para a interação com o armário de controle e o robô são *XOberon.Panel* e *Robot.Panel*, mostradas na figuras 3.9a e 3.9b, respectivamente. Através de *XOberon.Panel* inicializa-se a comunicação entre o computador operado pelo usuário e o processador do armário de controle *(target)* e pode-se visualizar estados de variáveis globais, módulos carregados no *target*, entre outros. Com a interface *Robot.Panel* faz-se a interação com o robô propriamente dito, alterando-se seu *estado*, habilitando-se ou não os amplificadores que controlam os servomotores, faz-se a sincronização das juntas e é possível ler valores de posição e velocidades instantâneas, por exemplo.



(a)



(b)

Figura 3.9 – Interfaces *XOberon.Panel* e *Robot.Panel*

Capítulo 4: Desenvolvimento de um Gerador de Trajetórias para o Robô Inter

Neste capítulo trataremos das implementações em software necessárias no desenvolvimento do gerador de trajetórias. Conforme mencionado no capítulo 1, podemos ver o desenvolvimento de uma tarefa de um manipulador em duas etapas:

- especificação da tarefa;
- execução dos movimentos.

No nosso caso, a especificação da tarefa consiste em especificar um *caminho* para o manipulador no espaço cartesiano. Como caminho entende-se um conjunto de pontos de referência que o robô deve interpolar. Para isso é interessante que o usuário especifique o menor número de parâmetros possível, como tempo, velocidade ou aceleração nos pontos, cabendo ao software em questão todo o processamento matemático envolvido.

Para isso decidiu-se utilizar como ferramenta de desenvolvimento o software *Matlab*. Esta escolha deu-se, entre outros, pelos seguintes motivos:

- trata-se de um ambiente bastante difundido e com uma grande base de usuários;
- como é voltado para aplicações científicas, tem todo um suporte a operações matemáticas, manipulações matricias, geração de gráficos, etc.;
- apesar de um pouco limitado no que diz respeito ao desenvolvimento de interfaces para o usuário, permite a entrada gráfica de dados e não somente textual;
- possibilidade de criação de novos *toolboxes* – conjuntos de funções dirigidas a aplicações específicas do usuário;
- foi utilizado na primeira versão do gerador, podendo-se reutilizar elementos conforme a necessidade.

Uma outra necessidade na etapa da especificação da tarefa é a comparação de resultados teóricos, sua visualização e testes, e o Matlab atende bem estes requisitos.

Como, evidentemente, o Matlab não faz parte do sistema XOberon, tem-se o problema de se estabelecer uma comunicação entre estes dois softwares de modo que se possa movimentar o robô de acordo os resultados obtidos na fase da especificação. Uma solução proposta para este caso é a criação de um arquivo padronizado e compatível para ambos os sistemas (Windows/Matlab e XOberon), que quando carregado para o armário de controle possa ser corretamente lido e interpretado, utilizando estas informações para a posterior execução dos movimentos desejados. Esta viabilidade foi estudada e uma implementação foi feita por [FernandesJr98]. A partir daí, bastaria criar novos módulos e procedimentos em XOberon que, de maneira semelhante àquela, também processe um conjunto de dados de acordo com o tipo de trajetória desejada.

A seguir trataremos das implementações em Matlab e em XOberon.

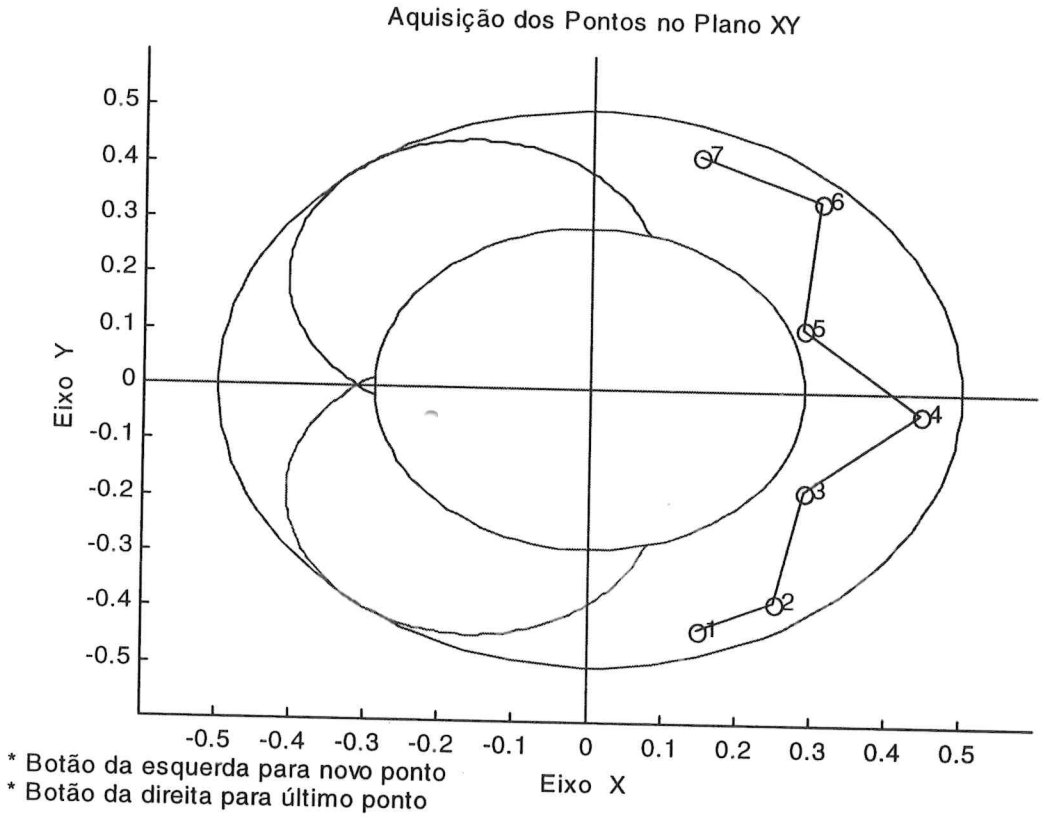
4.1 – O Gerador de Trajetórias em Matlab

O objetivo desta etapa foi desenvolver um ambiente amigável e de fácil utilização, em que se possa criar o caminho de maneira simples e sem a necessidade que o usuário tenha conhecimento profundo das características do robô Inter, como velocidades e acelerações máximas suportadas.

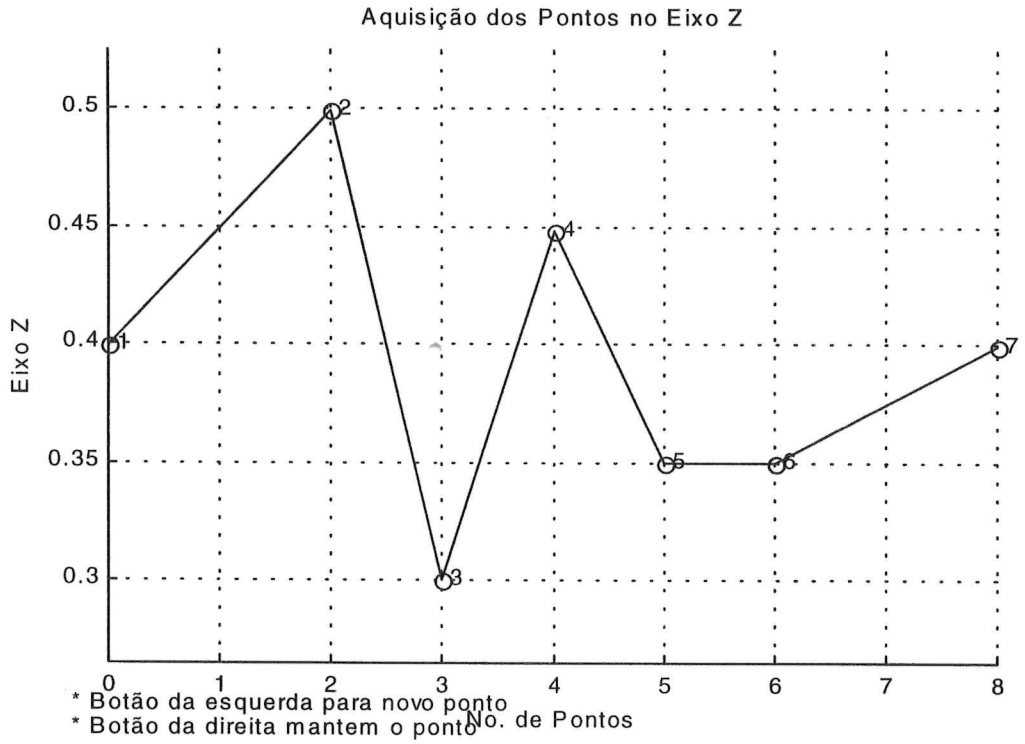
A etapa inicial deste trabalho foi estudar o gerador anterior e compreender seu funcionamento. Vimos que apenas uma pequena parte desta implementação poderia ser utilizada, dado que a solução fora ali desenvolvida para se obter trajetórias do tipo spline no espaço cartesiano, utilizando para isso um *toolbox* de splines. Este *toolbox* também não é adequado à nossa proposta por não interpolar exatamente os pontos desejados do caminho e sim sua vizinhança. Com isso decidimos partir para uma implementação completamente autônoma daquela, aproveitando somente a parte de inserção gráfica do caminho no plano xy e no eixo z do manipulador – até então não havia especificação da orientação.

Para a especificação do caminho, o usuário insere os pontos desejados em figuras que representam graficamente o espaço de trabalho do robô. Esta interação é feita

totalmente através do mouse e pode ser vista nas figuras 4.1a e 4.1b. Ao usuário cabe, primeiramente, inserir os pontos desejados no plano *xy*. Em seguida, é requisitada a informação temporal destes pontos, ou seja, em quais instantes de tempo deseja-se que o manipulador passe por eles. A partir daí o usuário especifica as posições no eixo *z* e a orientação do efetuador final, já associadas à informação anterior dos tempos.



(a)



(b)

Figura 4.1 – Figuras das interfaces do espaço operacional do robô para aquisição dos pontos do caminho: a) plano *xy* e b) eixo *z*. A interface da *orientação* é semelhante à b).

Como se pode ver pela figura 4.1a, estão definidas as regiões AD e AE para que o usuário tenha uma melhor noção do espaço de trabalho. Caso um ponto seja inserido em uma destas regiões, é exibida uma mensagem na mesma janela confirmando a entrada nesta região. Neste trabalho, entretanto, não consideramos a região AE. Evidentemente para qualquer das quatro juntas pontos clicados fora da área de trabalho não são considerados.

Estes pontos são então convertidos em posições de juntas e armazenados em matrizes. A partir daí são feitos os cálculos necessários à implementação das trajetórias, conforme a metodologia escolhida. Durante a execução destes cálculos é gerado um relatório no *prompt* do Matlab, contendo informações como:

- cinemática inversa de cada junta;

- velocidades e acelerações calculadas nos pontos do caminho;
- verificação dos valores máximos gerados para posições, velocidades e acelerações de cada junta, comparando-os com os valores configurados no robô para se determinar se a trajetória é factível ou não;
- apresentação das principais matrizes utilizadas no cálculo das trajetórias;
- apresentação dos coeficientes dos polinônios de cada junta em cada segmento e
- gravação de um arquivo de pontos outro de coeficientes para posterior utilização no XOberon.

Um exemplo de relatório (aqui resumido) é apresentado na figura 4.2 para uma trajetória de 6 segmentos.

```
Insira os respectivos tempos de cada posição:
(o 1o. instante já está configurado como zero)
Tempo: 2
Tempo: 3 ...

Introdução dos pontos virtuais
Tempo do 1o. ponto virtual: 1 ...

Vetor de tempos:
     0     1     2     3     ...

Pontos adquiridos no espaco cartesiano [x y z phi] :

    0.1493   -0.4308    0.3997   -0.0030
    0.2516   -0.3797    0.4994    1.4803 (...)

Posicoes calculadas no espaco das juntas [j0 j1 j2 j3] :

   -1.4107    0.8504    0.1656   -1.4803
   -1.3726    1.6402    0.3650    2.0248 (...)

Matriz A:

     6     1     0     0     0     0     0
     0     4     1     (...)

Aceleracoes calculadas no espaco das juntas,
[acc_j0 acc_j1 acc_j2 acc_j3] :

        0         0         0         0
   0.3831   -0.3549   -0.2101   -4.4610 (...)
```

```
Velocidades máximas geradas são inferiores às configuradas. OK!
Velocidades máximas geradas e configuradas para as juntas 0, 1, 2 e
3:
    1.1536    3.7700 (...)

Acelerações máximas geradas são inferiores às configuradas. OK!
Acelerações máximas geradas e configuradas para as juntas
0, 1, 2 e 3:
    2.3631   80.0000 (...)

Apresentação dos coeficientes de cada junta na forma Alfa, Beta,
Gama, Delta

Coeficientes dos polinômios da junta 0:
         0    0.0639   -1.6600   -1.6600
    0.0639   -0.1338   -1.6600   -1.2769 (...)
```

Figura 4.2 – Exemplo dos resultados de uma trajetória

Em seguida são apresentados os gráficos de posições, velocidades e acelerações geradas no espaço de juntas.

Por fim, gravam-se dois tipos de arquivos para posterior utilização no XOberon. Um destes arquivos contém pontos referentes às posições e velocidades de cada junta amostrados a cada 1ms. O outro é um arquivo com os coeficientes das *splines* em cada segmento e os respectivos tempos. A utilização destes arquivos pelo XOberon é discutida na próxima seção deste documento.

Para este gerador foram implementadas duas soluções: *splines* cúbicos pelos pontos virtuais e também a chamada "solução natural"[2] de *splines*, onde a condição de velocidades nulas no início e no fim da trajetória é relaxada - com isso, obtém-se diretamente um sistema de *4n* equações para *4n* incógnitas.

Como o gerador foi desenvolvido totalmente em dois módulos separados, um para cada solução, alguns trechos de código encontram-se em ambos. Mesmo estando internamente bem documentados, pode ser um tanto trabalhoso para outro programador estudar estas implementações e aproveitar trechos delas em outras aplicações, devendo fazer uma cuidadosa compatibilização de variáveis, etc. Pensando nisso e em uma maior versatilidade para o gerador, foi iniciado o desmembramento das funções principais destes

[2] A solução natural de splines, assim chamada por representar a forma de splines mecânicas no primeiro e no último nó (ou pontos), é bastante conhecida e a resolução pode ser encontrada em [Chapra&Canale92?] e [Sedgewick83], ainda que este último contenha um erro no equacionamento da pág. 71.

módulos em *funções matlab*, isto é, para a criação de um pequeno *toolbox* aplicado ao robô Inter. A proposta é que este *toolbox* contenha algumas funções genéricas que possam ser reutilizadas em outras aplicações envolvendo o mesmo robô, como por exemplo aquisição de pontos no espaço cartesiano, as cinemáticas direta e inversa, a aquisição direta de vetores de dados de acordo com o número de pontos do caminho (por exemplo tempos, velocidades ou acelerações), funções de cálculos das splines utilizadas, entre outras. Isso viria a facilitar certas melhorias que propomos a esta versão do gerador, como por exemplo o cancelamento e/ou substituição de pontos e tempos por outros sem a necessidade de se repetir todo o processo e o cálculo de trajetórias de referência por diferentes métodos utilizando o mesmo caminho, facilitando estudos de comparações entre elas. Foram desenvolvidas funções para aquisição de pontos referentes ao plano xy, eixo z e orientação do efetuador final bem como da cinemática inversa.

Dado que o objetivo principal desta etapa não era o desenvolvimento de tal *toolbox* e considerando o tempo necessário para implementações e testes de "migração" destas funções – já que todas elas estão presentes em um ou outro módulo, porém referenciadas a variáveis diferentes e implementadas de maneiras diferentes – , optamos por partir para as implementações no XOberon, dado que não tínhamos como avaliar com certeza o tempo necessário para estas atividades.

4.2 – Implementações em XOberon

Conforme mencionado na seção 3.8, a linguagem XOberon é uma extensão de Oberon que dá suporte ao sistema para a execução de tarefas em tempo real. A figura 3.8 ilustra uma janela deste ambiente. Deve-se fazer uma distinção entre o que é executado pelo computador operado pelo usuário (*host*) e o armário de controle (*target*). O *host* é apenas a interface com o robô, onde está instalado o ambiente e seus módulos, ferramentas, etc. O *target* é que é responsável pelo processamento das tarefas e comandos dados no *host*.

No XOberon é possível definir tarefas e o tempo de execução destas tarefas. Se por algum motivo o tempo estipulado não puder ser cumprido, o sistema detectará o erro e o programa será finalizado, enviando uma mensagem de erro. Os dois tipos de eventos mais utilizados na programação de sistemas em tempo real são o *main event* e *every event*.

O *main event* é utilizado para implementação de tarefas que são executadas somente quando houver tempo computacional disponível, isto é, quando não houver uma tarefa de tempo crítico sendo executada. Por isto são consideradas tarefas de tempo não crítico (NTC).

O *every event* implementa tarefas de tempo crítico (TC). Neste caso a tarefa tem de ser obrigatoriamente executada e os resultados disponibilizados no tempo especificado.

No nosso caso, o objetivo é desenvolver módulos que leiam um arquivo pré-definido e enviado para a memória do manipulador, processe adequadamente estes dados e envie sinais de referência de posição e velocidade de cada junta ao controlador do robô. Isso envolve basicamente a instalação de duas tarefas:

- uma tarefa *every event*, para o cálculo dos valores de referência e envio ao controlador, e
- uma tarefa *main event*, para supervisão do *every event* e sua desinstalação quando finalizada a trajetória desejada.

Na figura 4.3 é apresentado um diagrama contendo os blocos funcionais dos módulos desenvolvidos – para leitura de arquivos de dados já fornecidos em termos de velocidades e acelerações das juntas, e de coeficientes.

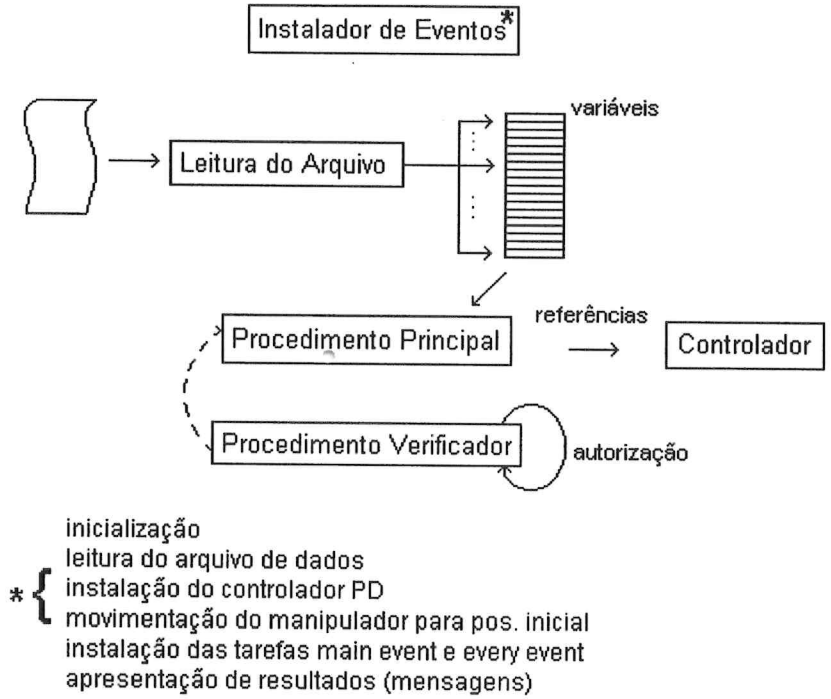


Figura 4.3 - Diagrama genérico da geração de trajetórias no XOberon

Este módulo funciona da seguinte maneira:

- o procedimento instalador de eventos (*Instalar*) inicializa as variáveis globais e chama o leitor de arquivo (*LerTodosVal*). Posteriormente solicita autorização para uso do controlador e posiciona as juntas nas posições iniciais para o início do movimento. Finalmente, instala as tarefas *EnvDados* (procedimento principal, do tipo *every event*) e *Verifica* (verificador, do tipo *main event*).
- o leitor de arquivos confere a consistência do arquivo na memória do armário de controle, isto é, se o número de colunas está de acordo com o especificado, que depende do tipo de solução adotado para a geração da trajetória, e associa os valores lidos às variáveis que serão utilizadas pelo procedimento principal, fazendo a conversão dos strings para representações de números reais. As implicações desta conversão e os problemas que se verificaram nos testes da geração *on-line* das trajetórias são discutidas no capítulo 5 Ao final, este procedimento retorna o controle ao *Instalar*.

A tarefa *EnvDados* se repete obrigatoriamente a cada 1ms e o que ela faz é calcular os valores de referência para o controlador (no caso de um arquivo de coeficientes) ou simplesmente enviar os valores lidos no arquivo e convertidos, quando este já contém os pontos desejados.

"Paralelamente" à *EnvDados* é executado *Verifica*, responsável por verificar se há autorização de uso do controlador e se a trajetória foi finalizada, caso em que explicitamente desinstala EnvDados.

Deve-se entender que *EnvDados* nunca é interrompido antes de chegar ao final de um ciclo e por essa razão deve ser executado de tempos em tempos, caso contrário – na utilização de um *loop* interno errado, por exemplo – tomaria o tempo total do processador.

Já *Verifica*, por ser um *main event*, pode ser interrompido em qualquer ponto de sua execução se houver necessidade de se executar um *every event*. Ao contrário do *every event*, o *main event* por sua definição é executado somente uma vez. Por isso a utilização de um *loop while* enquanto as condições de *EnvDados* forem verdadeiras, ou seja, *Verifica* não é desinstalado até que alguma condição seja satisfeita.

A diferença de *Verifica* ou de qualquer outro *main event* sobre processos normais é que eles têm prioridade de execução e não têm restrições quanto ao tempo em que resposta para essa execução deve estar disponível. A utilização de um processo *main event* para verificação/desintalação de *EnvDados* é também adequada já que quando a execução de um programa fica presa no interior de um *loop* de um *main event*, é possível fazer chamadas externas para a execução de outros comandos.

A seguir descreveremos as variações do leitor de dados e do procedimento principal conforme a solução adotada.

Como o início de todo o processo dá-se pela leitura de um arquivo, este arquivo deve ser padronizado. Esta padronização deve ser tanto no conteúdo, isto é, em como os dados são armazenados (no nosso caso foi utilizado o padrão ASCII) e na sua organização dentro do arquivo, para que se possa saber o que cada conjunto de números representa. No caso da leitura direta de valores de referência, foi adotado o seguinte padrão:

$$\begin{bmatrix} \theta_0 & \cdots & \theta_3 & \dot{\theta}_0 & \cdots & \dot{\theta}_3 \\ \vdots & & \vdots & \vdots & & \vdots \\ \theta_0 & \cdots & \theta_3 & \dot{\theta}_0 & \cdots & \dot{\theta}_3 \end{bmatrix}$$

onde cada linha representa os valores a serem enviados ao controlador a cada 1ms. Assim, se as trajetórias de referência têm uma duração de 10s, este arquivo será uma matriz de strings de tamanho 10^3x8. Esta representação difere ligeiramente da utilizada por [FernandesJr98] por não utilizar acelerações de referência e por incluir as informações para a orientação do efetuador final, o que, para uma mesma trajetória, produz arquivos menores. Com isso, foram necessárias poucas modificações na implementação original, fazendo basicamente as alterações referentes ao padrão do arquivo a ser lido.

É interessante ressaltar que este novo padrão pode ser adotado para a geração de trajetórias de qualquer perfil, ou seja, independente do método utilizado para gerá-las, desde que se utilize o controlador PD originalmente implementado ou um equivalente à este.

Para o caso da leitura de arquivo de coeficientes, foram propostas duas soluções. A idéia é que a geração das trajetórias seja feita *on-line*, ou seja, a partir da leitura dos coeficientes e do tempo de cada segmento, uma tarefa *every event* faça os cálculos dos valores de referência a cada 1ms e os envie ao controlador para execução dos movimentos. Esta tarefa *every event* é uma rotina *EnvDados*.

No caso da "solução natural" de splines, onde se obtém como resultado do Matlab polinômios de posição e velocidade da forma

$$P_k(t) = a_{3k}t^3 + a_{2k}t^2 + a_{1k}t + a_0$$
$$\dot{P}_k(t) = 3a_{3k}t^2 + 2a_{2k}t + a_{1k}$$

para $t \in [t_k, t_{k+1}]$ com k o *k-ésimo* segmento, $k=1... n-1$, o padrão adotado para o arquivo foi

$$\begin{bmatrix} a_{30k} & a_{20k} & a_{10k} & a_{00k} & a_{31k} & \cdots & a_{01k} & \cdots & a_{34k} & \cdots & a_{04k} & t_{k+1} \\ & & & & & \vdots & & & & & & \\ a_{30n-1} & a_{20n-1} & a_{10n-1} & a_{00n-1} & a_{31n-1} & \cdots & a_{01n-1} & \cdots & a_{34n-1} & \cdots & a_{04n-1} & t_n \end{bmatrix}$$

onde a_{30k} representa o coeficiente a_3 da junta 0 no segmento k. Assim, para uma trajetória de 10 segmentos, tem-se uma matriz de 17x10.

Padrão semelhante foi adotado para o caso da "solução por pontos virtuais", sendo que agora os polinômios são dados pelas duas primeiras equações (2.15), abaixo, e os coeficientes do arquivo são os α ... δ de cada junta em cada segmento.

$$P_k(t) = \alpha_k(t_{k+1} - t)^3 + \beta_k(t - t_k)^3 + \gamma_k(t_{k+1} - t) + \delta_k(t - t_k)$$
$$\dot{P}_k(t) = -3\alpha_k(t_{k+1} - t)^2 + 3\beta_k(t - t_k)^2 - \gamma_k + \delta_k$$

Evidentemente os procedimentos de verificação da integridade do arquivo são adaptados conforme o caso em que se esteja e o mesmo ocorre quando o instalador de eventos move o manipulador para a posição inicial.

A metodologia utilizada no desenvolvimento da tarefa principal, para o caso da geração *on-line* das trajetórias, é baseada em uma comparação constante do tempo atual com o tempo de duração do segmento que se está gerando. Ou seja, a cada ciclo de 1ms que a tarefa *EnvDados* é chamada e executada, compara-se o tempo corrido desde o início da movimentação do manipulador com o tempo que finaliza o segmento corrente. Assim que o tempo do segmento corrente é atingido, passa-se a utilizar os próximos coeficientes dos vetores de coeficientes de cada junta. Neste ciclo, em particular, é enviado ao controlador o mesmo conjunto de referências do ciclo anterior, já que as splines garantem continuidade das variáveis de interesse. Os cálculos das posições e velocidades desejadas são então feitos utilizando-se estes novos valores até que se atinja o final do segmento.

Os problemas encontrados nessas implementações e os resultados obtidos são discutidos no capítulo 5.

Capítulo 5: Resultados e Discussões

Neste capítulo apresentaremos inicialmente alguns resultados obtidos com o gerador desenvoldido no Matlab para então fazermos algumas discussões sobre problemas nas implementações no XOberon.

Para a programação *off-line,* o usuário inicia especificando o caminho desejado na área de trabalho do plano *xy* do manipulador. Um exemplo pode ser visto na figura 5.1. Em seguida o sistema requisita as informações dos tempos de cada segmento, ilustrado em 5.2. Conforme mencionado no capítulo 2, consideramos que os tempos dos segmentos é o mesmo para todas as juntas e elas iniciam e terminam seus movimentos ao mesmo tempo. Em seguida, é aberta uma janela para a especificação dos pontos no eixo *z*, conforme ilustrado em 5.3. De forma semelhante o usuário insere as informações da orientação do efetuador final, na figura 5.4, encerrando esta etapa. Com estas informações, o Matlab apresenta uma série de resultados e os arquivos de pontos e de coeficientes são gerados.

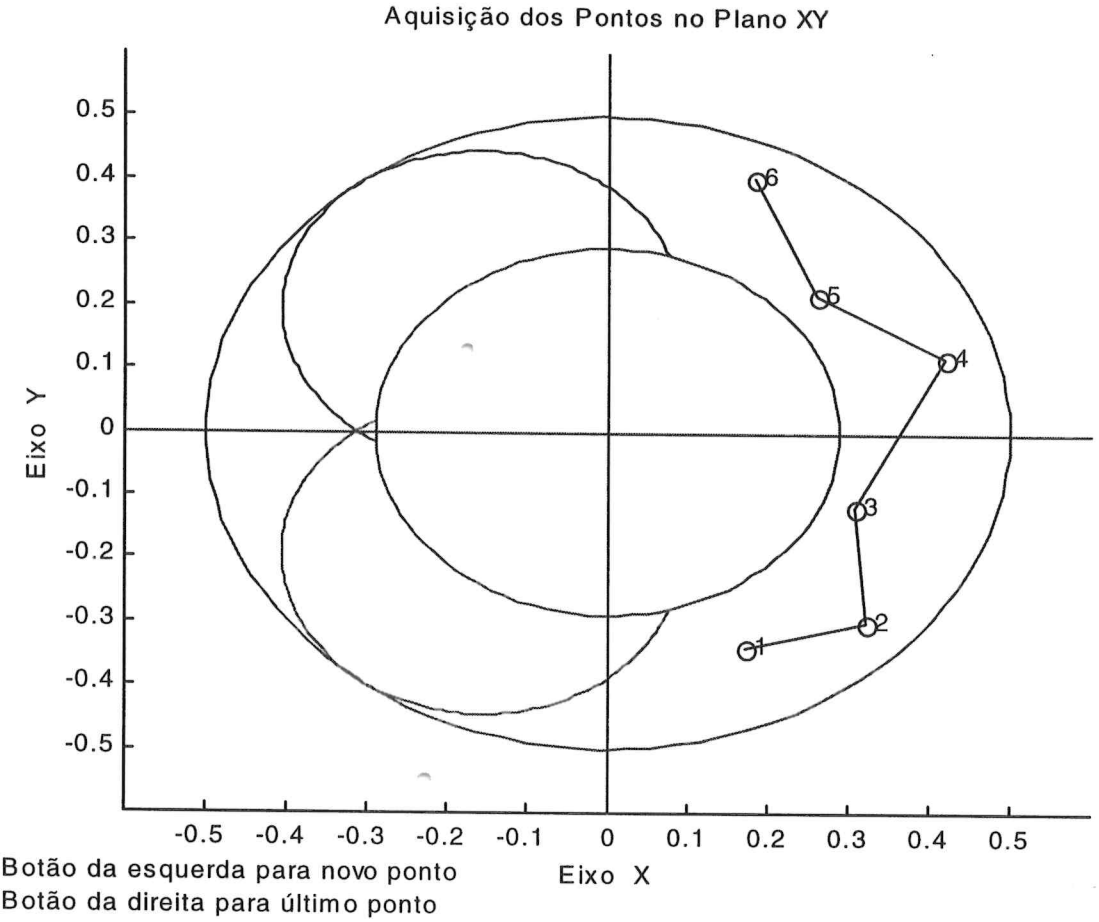


Figura 5.1 – Representação do espaço de trabalho do robô Inter no plano *xy* e aquisição dos pontos do caminho

```
Insira os respectivos tempos de cada posição:
(o 1o. instante já está configurado como zero)
Tempo: 2
Tempo: 4
Tempo: 3
Tempo inválido (menor que o anterior)
Digite novamente o tempo: 5
Tempo: 7
Tempo: 10
Introdução dos pontos virtuais
Tempo do 1o. ponto virtual: 1
Tempo do 2o. ponto virtual: 9

Vetor de tempos:
     0     1     2     4     5     7     9    10
```

Figura 5.2 – Aquisição dos tempos de cada ponto

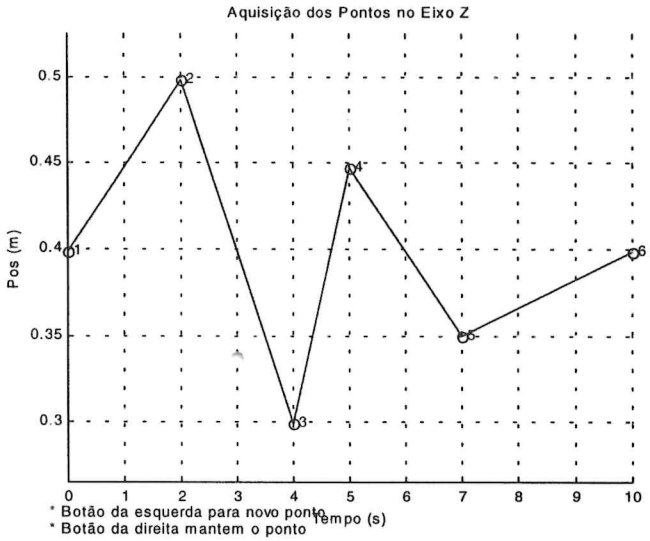


Figura 5.3 – Aquisição dos pontos no eixo *z*

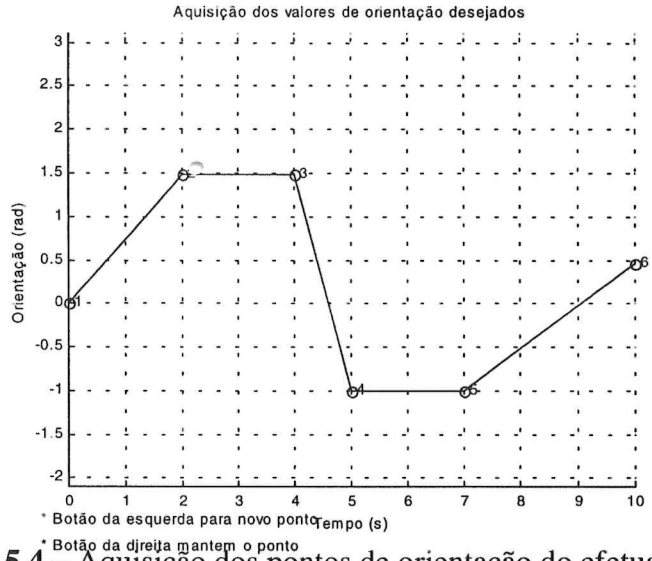


Figura 5.4 – Aquisição dos pontos de orientação do efetuador final

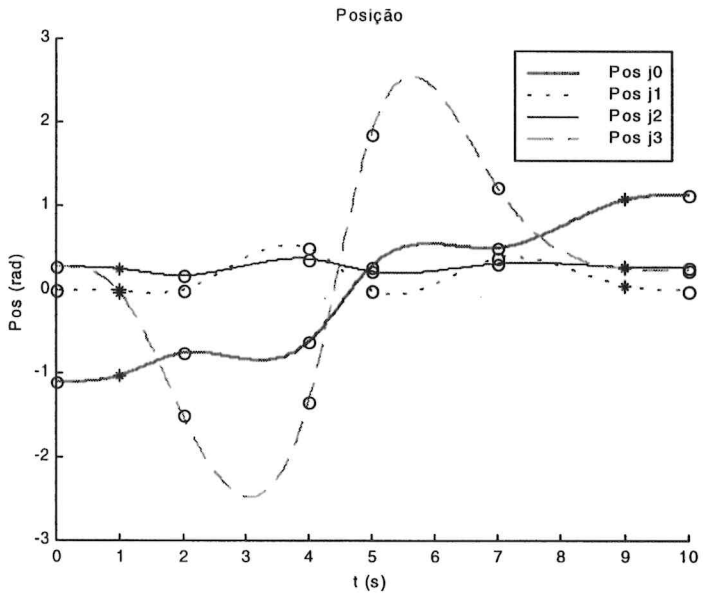
Os dados obtidos para esta trajetória foram:

```
Pontos adquiridos no espaco cartesiano:
   x      y      z      phi
    0.1742   -0.3404    0.3988   -0.0018
    0.3235   -0.3018    0.4984    1.4883
    0.3097   -0.1193    0.2992    1.4883
    0.4203    0.1193    0.4475   -1.0053
    0.2627    0.2175    0.3494   -1.0053
    0.1853    0.4035    0.3988    0.4544

As velocidades inicial e final são consideradas nulas
As acelerações inicial e final são consideradas nulas

Posicoes no espaco das juntas [j0 j1 j2 j3] :

   -1.0977         0    0.2662    0.2680
```

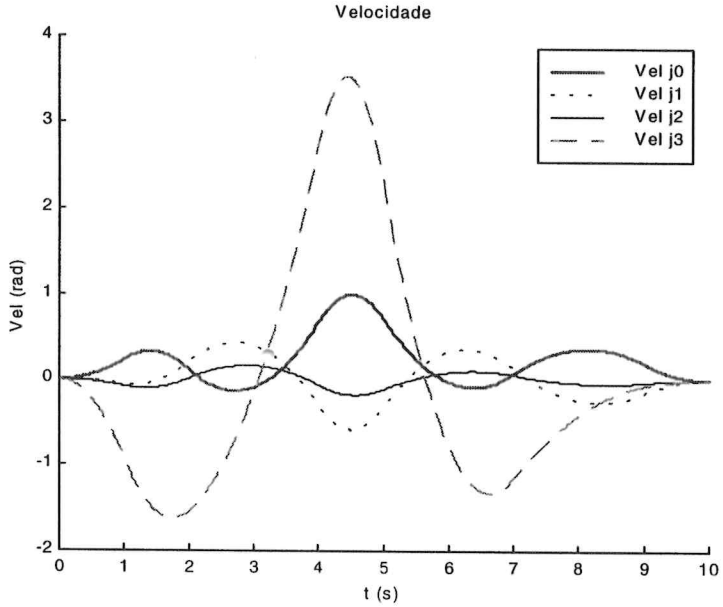
```
         0         0         0         0
   -0.7506         0    0.1666   -1.4883
   -0.6141    0.4927    0.3658   -1.3217
    0.2766         0    0.2175    1.8638
    0.5043    0.3748    0.3156    1.2228
         0         0         0         0
    1.1404         0    0.2662    0.2360

Matriz A:

    6.0000    1.0000         0         0         0         0
         0    6.0000    2.0000         0         0         0
         0    2.0000    6.0000    1.0000         0         0
         0         0    1.0000    6.0000    2.0000         0
         0         0         0    2.0000    8.0000    1.5000
         0         0         0         0    2.0000    7.5000

Aceleracoes calculadas no espaco das juntas,
[acc_j0 acc_j1 acc_j2 acc_j3] :

         0         0         0         0
    0.4628   -0.1035   -0.1557   -1.8612
   -0.6944    0.6212    0.3367    0.6295
    1.2466   -1.1244   -0.4124    3.6303
   -1.1565    1.0699    0.3143   -4.4279
    0.5158   -0.6070   -0.1448    0.9508
   -0.3920    0.3118    0.0584    0.1412
         0         0         0         0

Velocidades máximas geradas são inferiores às configuradas. OK!
Velocidades máximas geradas e configuradas para as juntas 0, 1, 2 e 3:
    0.9912    3.7700

    0.5842    3.7700

    0.1802    0.8800

    3.5307   20.0000

Acelerações máximas geradas são inferiores às configuradas. OK!
Acelerações máximas geradas e configuradas para as juntas 0, 1, 2 e 3:
    1.2466   80.0000

    1.1244  100.0000

    0.4124    3.0000

    4.4279  300.0000


Apresentação dos coeficientes de cada junta na forma
Alfa, Beta, Gama, Delta

Coeficientes dos polinômios da junta 0:
         0    0.0771   -1.0977   -1.0977
    0.0771   -0.1157   -1.0977   -0.6349
   -0.0579    0.1039   -0.1439   -0.7226
    0.2078   -0.1927   -0.8218    0.4693
   -0.0964    0.0430    0.5238    0.0802
    0.0430   -0.0327    0.0802    0.6682
   -0.0653         0    1.1404    1.1404

Coeficientes dos polinômios da junta 1:
```

```
         0   -0.0173         0         0
   -0.0173    0.1035         0   -0.1035
    0.0518   -0.0937   -0.2071    0.6212
   -0.1874    0.1783    0.6801   -0.1783
    0.0892   -0.0506   -0.3566    0.3897
   -0.0506    0.0260    0.3897   -0.0779
    0.0520         0         0         0

Coeficientes dos polinômios da junta 2:
         0   -0.0260    0.2662    0.2662
   -0.0260    0.0561    0.2662    0.1105
    0.0281   -0.0344   -0.0289    0.3204
   -0.0687    0.0524    0.4345    0.1652
    0.0262   -0.0121    0.0240    0.2061
   -0.0121    0.0049    0.2061    0.1185
    0.0097         0    0.2662    0.2662

Coeficientes dos polinômios da junta 3:
         0   -0.3102    0.2680    0.2680
   -0.3102    0.1049    0.2680   -1.5932
    0.0525    0.3025   -0.9540   -1.8710
    0.6051   -0.7380   -1.9267    2.6017
   -0.3690    0.0792    2.4079    0.2945
    0.0792    0.0118    0.2945    0.0827
    0.0235         0    0.2360    0.2360

Arquivo de pontos: pontos.m
Arquivo de coeficientes: coefs.m
```

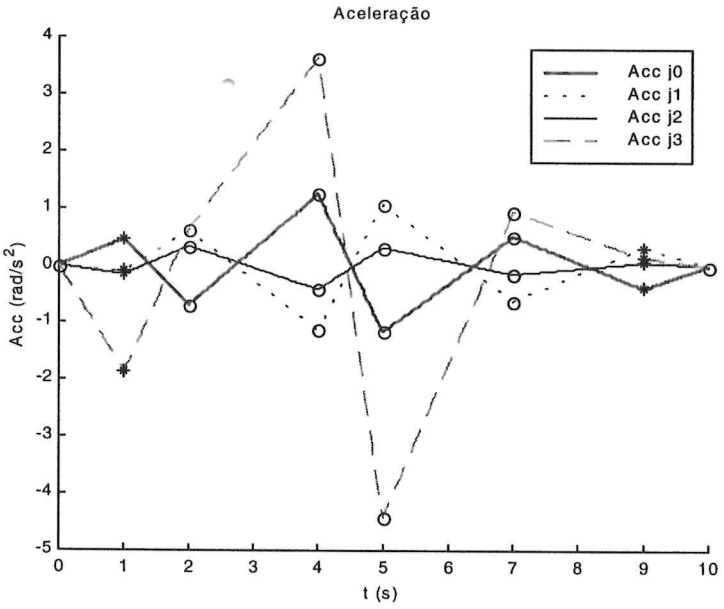
As curvas de posição, velocidade e aceleração das juntas para esta trajetória são mostradas respectivamente nas figuras 5.5a, 5.5b e 5,5c. Linha contínua partindo de ~ –1: junta 0, pontilhada: junta 1, contínua: junta2 e tracejada: orientação.



(a)



(b)



(c)

Figura 5.5 – Perfis de posição, velocidade e aceleração no espaço das juntas para os caminhos especificados de 5.1 a 5.4

Sobre as regiões AD e AE já foi mencionado que o manipulador só atinge estas áreas em configurações particulares das juntas 0 e 1. A região AD só é atingida com θ_0 e θ_1 maiores que zero. O inverso vale para AE.

A figura 5.6 ilustra um caminho que atinge a região AD a partir do quarto segmento. A partir daí, portanto, o software gerador das trajetórias de referência deve garantir que os ângulos das juntas 0 e 1 sejam somente positivos. Pela figura 5.7 vê-se que a especificação para θ_0 e θ_1 é atendida.

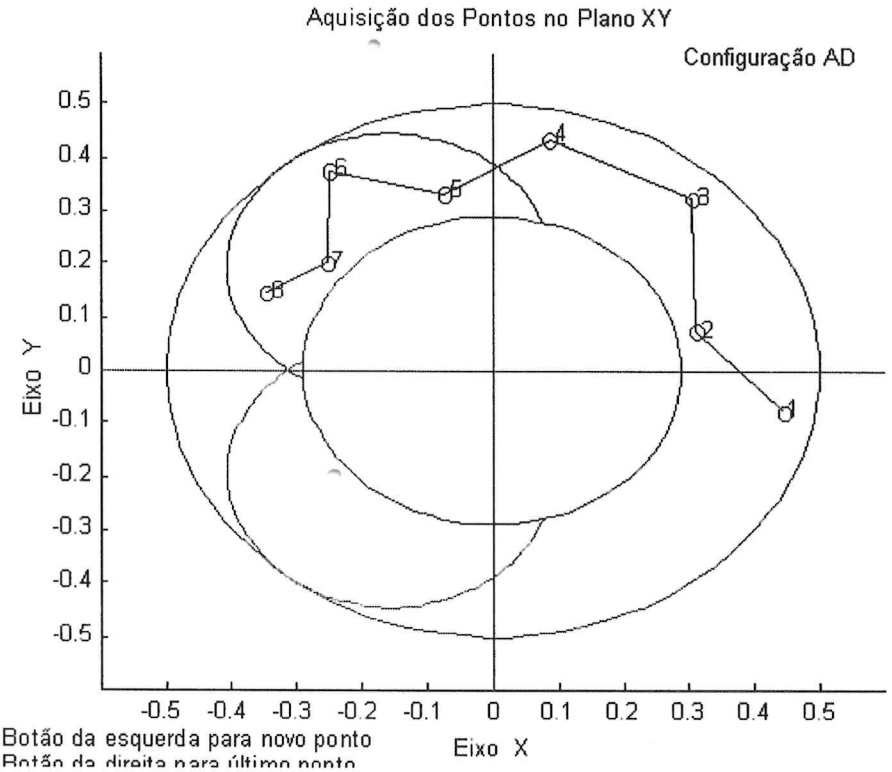


Figura 5.6 – Um caminho que atinge a região AD

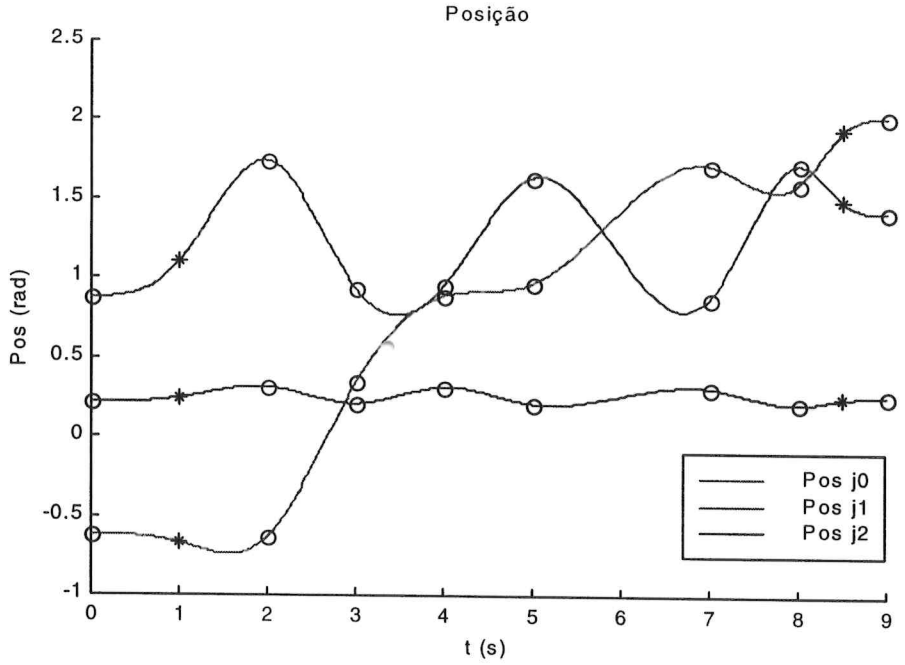


Figura 5.7 – Perfis de posição gerados para o caminho da figura 5.7. O perfil central refere-se à junta 2 do manipulador e pode ser desconsiderado neste exemplo.

A figura 5.7 ilustra um outro caminho especificado no plano *xy* do manipulador. Utilizaremos este caminho para fazermos uma comparação entre as trajetórias geradas pelos métodos da "solução natural" e "pontos virtuais".

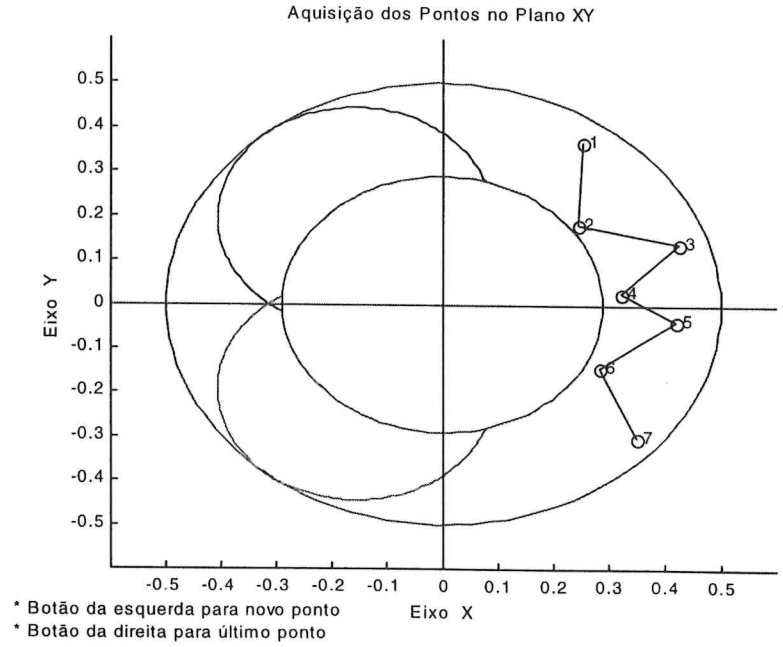
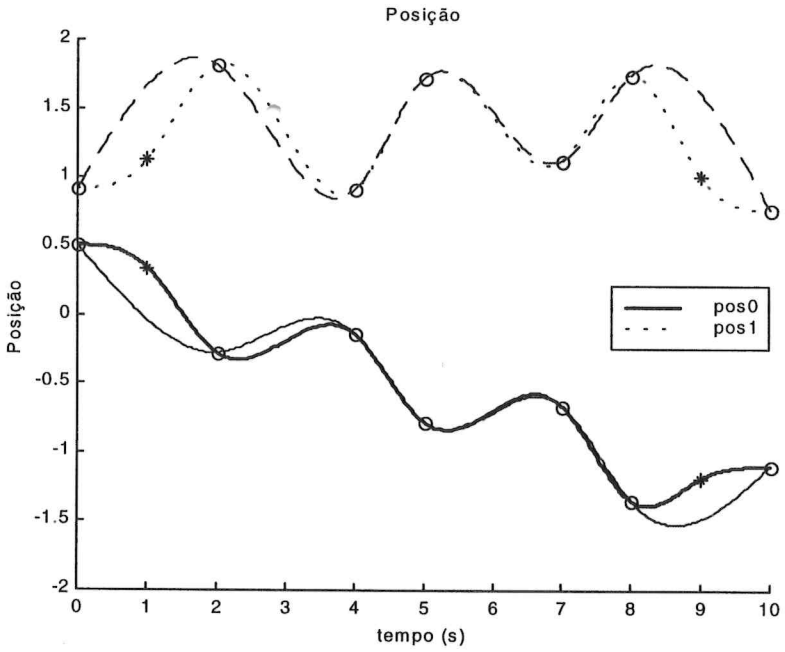


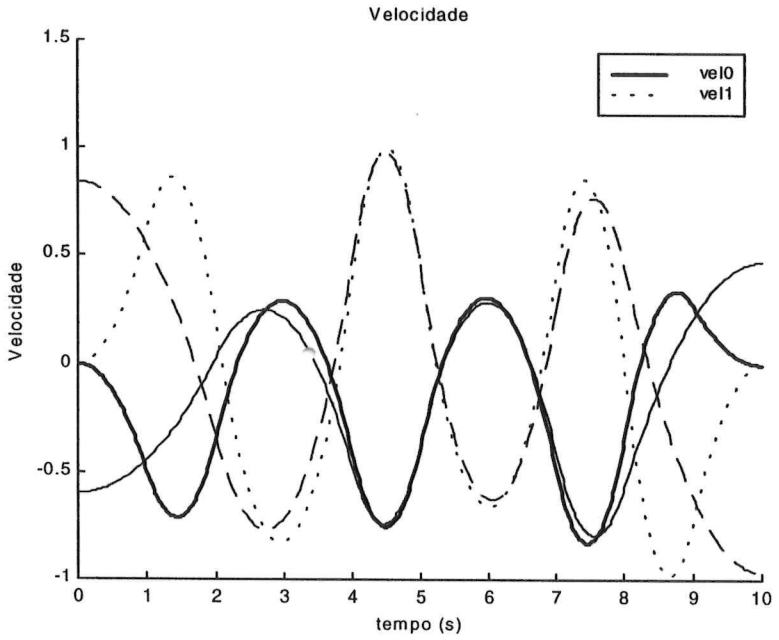
Figura 5.7 – Caminho utilizado para comparação das trajetórias de referência obtidas pelos métodos da "solução natural" e "pontos virtuais"

Os resultados obtidos para as juntas 0 e 1 do manipulador podem ser vistos nas figuras 5.8a, 5.8b e 5.8c. Nestas figuras, os resultados pela solução com pontos virtuais estão representados nas linhas em negrito (junta 0) e pontilhada (junta 1). Para a solução natural, utilizamos as linhas fina e tracejada para as juntas 0 e 1, respectivamente.

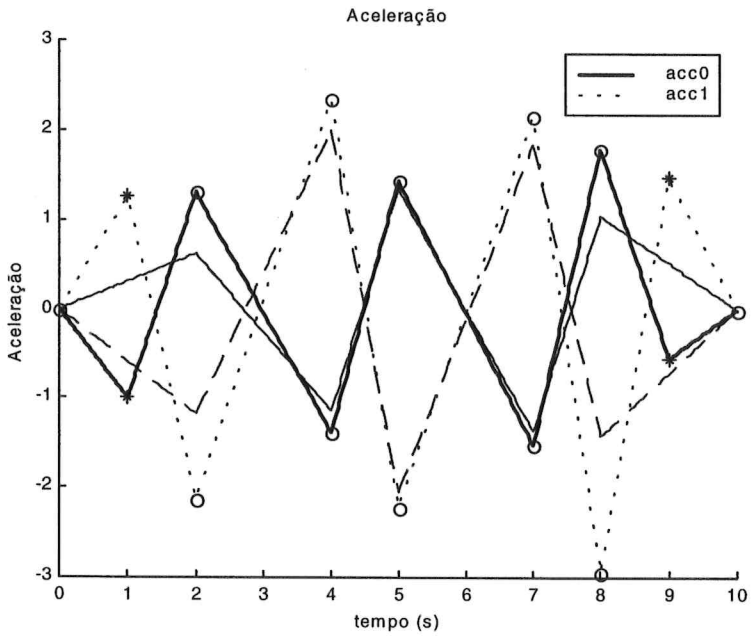
Nota-se de imediato a diferença nos perfis de velocidade (b) entre os dois métodos, em que a solução natural não garante velocidades inicial e final nulas. No caso do método dos pontos virtuais, como há duas condições iniciais a mais, os perfis de posição (a) nos segmentos inicial e final são mais suaves. Em compensação, as acelerações (c) para este caso são maiores nestes trechos.



(a)



(b)



(c)

Figura 5.8 – Resultados para posição, velocidade e aceleração das juntas 0 e 1 pelos métodos da solução natural e pontos virtuais

A seguir faremos alguns comentários sobre as implementações em XOberon.

Na fase da geração das trajetórias de referência *on line*, percebeu-se que em vários casos o robô interrompia sua execução antes do final esperado, travando. Não havia motivo aparente para esta falha, já que a metodologia adotada para instalação das tarefas bem como os resultados dos cálculos enviados ao controlador nos pareciam adequados.

Nas tentativas de se descobrir alguma razão para estas falhas, várias trajetórias com características diferentes – como número de segmentos e duração – foram testadas, mas não levaram a nenhuma conclusão sobre o que estava ocorrendo. Com isso a primeira suspeita caiu sobre a representação sequencial dos coeficientes na matriz gerada pelo Matlab e a leitura destes dados no XOberon. Por exemplo, a 1ª coluna do arquivo gerado pelo Matlab representa os coeficientes a_{30}, a 2ª coluna os coeficientes a_{20} e assim sucessivamente até a última coluna, representante dos tempos de cada segmento e efetivamente era essa mesma associação que o leitor de dados estava fazendo com as variáveis de interesse no XOberon. Ao mesmo tempo, o leitor de dados não indicava que o arquivo pudesse estar corrompido com uma linha incompleta, por exemplo.

A partir daí desenvolvemos um módulo auxiliar de teste, o *Testador.Mod*, que, através dos valores lidos dos coeficientes no arquivo de dados, calcula as trajetórias de referência – sem enviá-las ao controlador, para evitar novos travamentos do manipulador – e as grava em um arquivo temporário na memória do robô. Assim, copiando este arquivo para um computador pessoal, pudemos visualizar através do Matlab o que efetivamente estava sendo calculado pelo procedimento principal do gerador *on line*. Ou seja, foi feito o processo contrário, que teoricamente deveria levar aos mesmos resultados. Este procedimento foi feito para diferentes trajetórias e os seus gráficos – a partir dos dados do XOberon – mostraram descontinuidades. Houve casos de trajetórias que mostraram descontinuidades apenas na posição de uma junta. Em outras, as descontinuidades ocorreram para diferentes juntas nos perfis de posição e velocidade, em tempos diferentes.

O que fizemos, então, foi criar outro procedimento de teste no testador para gravar um segundo arquivo na memória do armário de controle, contendo agora os mesmos coeficientes recebidos do Matlab e também mostrá-los na janela de *log* do ambiente. A princípio estes valores evidentemente deveriam ser os mesmos. Foi com este teste que

pudemos descrobrir o motivo das descontinuidades geradas nos cálculos *on line*: os números com bases a partir de 10^1, ou seja, maiores que 10, não estavam sendo devidamente convertidos. Justamente a informação da base era perdida. Um exemplo pode ser visto na figura 5.9, onde em (a) apresentam-se os coeficientes gerados pelo Matlab para uma junta em uma trajetória de teste e em (b) os mesmos coeficientes lidos no XOberon.O que ocorre é o seguinte: quando o Matlab grava um conjunto de vetores na forma de um arquivo, utiliza a representação exponencial, utilizando o padrão ASCII. Assim, um número como –123,4 é gravado como –1.234e2 (que é o mesmo que $-1{,}234 \times 10^2$) e -0,1234 é gravado como –1.234e-1 (equivalente a $-1{,}234 \times 10^{-1}$).

```
2.4564132e-002  0.0000000e+000 -4.3554632e-001  4.1536457e-001
-4.6899897e-002  4.2878418e-001 -1.2931147e+000  9.8707681e-001
2.9491507e-002 -4.8791268e-001  2.3736727e+000 -3.9019731e+001
-4.3802991e-002  1.0512718e+000 -8.4006184e+000  2.1238040e+001
```

a)

```
 0.024564132    0  -0.43554632  0.41536457  -0.070671991
-0.046899897    0.42878418 -1.2931147   0.98707681
 0.029491507   -0.48791268  2.3736727  -3.9019731
-0.043802991    1.0512718  -8.4006184   2.123804
```

b)

Figura 5.9 – Em a) números (coeficientes) gravados em arquivo pelo Matlab; em b) a leitura e conversão deste arquivo para números reais no XOberon. Nota-se que a base dos módulos dos números maiores ou iguais a 10 é perdida nesta conversão

Este erro de conversão foi bastante surpreendente – e um tanto frustrante – já que existe um filtro no leitor de dados justamente para considerar os caracteres (–), (+), (.) e (e) bem como os números de 0 a 9 para a conversão destes *strings* em números reais. Este filtro utiliza os procedimentos do módulo *StrConv.Mod*. Uma possibilidade do porquê desta falha no filtro é um aparente erro no procedimento *StrToReal* do módulo *StrConv.Mod*, que faz a conversão genérica de *strings*. Olhando este procedimento não há explicitamente uma forma de conversão do caracter (+), o que leva crer que ele não o considera. De qualquer forma esta afirmação só pode ser feita com certeza após um estudo mais detalhado deste módulo.

Assim, buscamos uma forma alternativa para o Matlab gravar a matriz de coeficientes, que não use esta representação. Infelizmente o comando *save* do Matlab só

trabalha com este padrão e seria necessário utilizar o comando *fprintf*, mas este trabalha com padrões da linguagem C e é bem mais complicado. Apesar de permitir a representação na base desejada, os espaços entre números, as quebras de linha e identificadores de fim de arquivo, que devem ser explicitados no comando para que o arquivo gerado não seja uma única linha, utilizam outro padrão que não parece ser o ASCII. Fizemos testes com a leitura de arquivos gerados desta forma com o Notepad – que lê perfeitamente os arquivos do Matlab e XOberon – e caracteres como ♦, ♠, etc. não puderam ser filtrados.

Considerando-se tudo isso, cremos que a melhor alternativa é um estudo do módulo *StrConv.Mod* para uma alteração criteriosa de forma que esta conversão em particular seja feita com sucesso. Por enquanto, é necessário então que o usuário modifique manualmente os números com base maior ou igual a 10^1 no arquivo de coeficientes, antes dele ser enviado à memória do robô.

Uma observação importante é que a falha na conversão deste *string* (+) em particular não foi percebida no caso da geração *off-line* das trajetórias, já que os valores de referência gravados no arquivo de dados são todos menores que 10: as posições e velocidades no espaço de trabalho são dadas em radianos e rad/s para as juntas 0, 1 e 3 e em metros e metros/s para a junta 2, respectivamente. Isto pode ser observado também na tabela 3.2.

Por fim, é necessário citar que até o "fechamento desta edição" o procedimento *EnvDados* para o caso dos pontos virtuais não está sendo corretamente desinstalado do sistema, o que pode provocar erros. Isto está sendo investigado pois a metodologia utilizada neste caso é semelhante às dos outros dois, em que se obteve sucesso.

Capítulo 6: Conclusões e Perspectivas

Este trabalho teve como objetivo o estudo de técnicas para a geração de trajetórias de referência para robôs manipuladores. Foi desenvolvida uma ferramenta gráfica em Matlab para a programação *off-line* do robô Inter do Laboratório de Robótica. Esta ferramenta possibilita a geração de trajetórias suaves no espaço das juntas do manipulador, o que não existia anteriormente, interpolando e atingindo os pontos do caminho desejado, especificados pelo usuário no espaço cartesiano.

Uma outra contribuição é a possibilidade de geração das trajetórias *on-line*, em que o robô, neste caso, recebe um conjunto reduzido de parâmetros referentes ao espaço das juntas para a sua movimentação. Esta forma evita o cálculo da cinemática inversa em tempo real. Neta etapa, foram enfrentadas algumas dificuldades com o sistema XOberon. Elas decorreram principalmente de uma falha de um módulo interno utilizado para a conversão de conjuntos de caracteres em representações de números reais.

Neste trabalho foi considerado também a orientação do efetuador final, podendo-se agora gerar trajetórias para as quatro juntas do manipulador.

Foram implementadas duas técnicas para as trajetórias de referência, utilizando curvas *splines* cúbicas. A primeira solução não garante que o manipulador tenha velocidades nulas no início e no final das trajetórias. A segunda solução garante esta especificação. Em ambos os casos os perfis de aceleração são contínuos.

Foram feitos testes práticos com ambas as técnicas, tanto para a programação *off-line* como *on-line*.

A programação *off-line* não apresentou problemas, porém é restrita a tempos de movimentação curtos – de até 14 segundos aproximadamente – já que depende da transmissão de um grande arquivo de dados. O padrão criado para este arquivo e a

implementação no XOberon para o seu processamento independem do método adotado para a geração das trajetórias de referência.

No caso da programação *on-line*, as trajetórias geradas pela primeira solução das *splines* vêm sendo executadas com sucesso. Já para o segundo caso, o processo responsável pelo cálculo das referências para o controlador apresentou um problema e com isso essas trajetórias acabam, em alguns casos, não sendo executadas até o fim.

Como perspectivas de continuidade deste trabalho, sugerimos inicialmente uma modificação no módulo responsável pela conversão de strings, permitindo a conversão adequada para números reais dos caracteres que representam números maiores ou iguais a dez. Uma outra sugestão é a utilização das rotinas genéricas de um *toolbox* para o robô Inter, bem como o desenvolvimento de novas funções. Isto contribuiria para uma maior flexibilidade do gerador, permitindo por exemplo a substituição de um ponto do caminho por outro, sem a necessidade de se refazer todo o processo de especificação.

Outra sugestão é um estudo comparativo entre as trajetórias de referência e as efetivamente executadas pelo manipulador. Para isso é necessário uma adaptação do módulo *MatlabFile2.Mod*, responsável pela leitura dos valores instantâneos de posição, velocidade, aceleração e outras variáveis de interesse e plotagem dos respectivos gráficos. Com isso, este gerador pode ser utilizado para a comparação das trajetórias reais executadas por diferentes controladores, a partir das mesmas referências.

As listagens da programação feita no Matlab e os módulos e procedimentos desenvolvidos no XOberon estão documentados e disponíveis no Laboratório de Robótica – LAI.

Bibliografia

[BCMP98]
BIER, C.C., CUNHA, A.E.C., MARTINS, D., PASSOLD, F. - Planejamento de Trajetória – Documento interno do Laboratório de Robótica da UFSC, 1998.

[Chapra&Canale92]
CHAPRA, S. E CANALE, R. – Numerical Methods for Engineers – McGraw-Hill, New York.

[Craig98]
CRAIG, JOHN J. – Introduction to robotics mechanics & control – Addison-Wesley, Reading, MA, 1986.

[FernandesJr98]
FERNANDES JÚNIOR, C. E GUENTHER,R. (ORIENTADOR) – Desenvolvimento de um Gerador de Trajetórias para um Robô do Tipo SCARA – Monografia – ECAI – UFSC, 1998.

[GWG98]
GOLIN, J., WEIHMANN, L. E GUENTHER, R. – Manual do Usuário do Robô Inter – Documento interno do Laboratório de Robótica da UFSC, 1998.

[Mathworks97]
THE MATHWORKS, INC. – Matlab – Versão do Estudante – Guia do Usuário – Makron Books, São Paulo, 1997.

[Sciavicco&Siciliano96]
SCIAVICCO, L. E SICILIANO, B. – Modeling and Control of robot Manipulators – McGraw-Hill, New York, 1996.

[Sedgewick83]
SEDGEWICK, R. – Algorithms – Addison-Wesley, Reading, MA, 1983.

[Spong&Vidyasagar89]
SPONG, M. E VIDYASAGAR, M. – Robot Dynamics and Control – John Wiley & Sons, New York, 1989.

[Vestli97]
VESTLI, S. – Xoberon (apostila) – IfR, ETH – Zurique, 1997.