Este exemplar corresponde a redação final da tese defendida por Carlos Florian Heuberger e aprovada pela comissão julgadora em 25/10/90.

Douglas F. Zampieri

Determinação Automática de Trajetórias Ótimas para um Manipulador na Presença de Obstáculos

25/90

por
Carlos Florian Heuberger

Outubro 1990

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

Tese de Mestrado

Determinação Automática de Trajetórias Ótimas para um Manipulador na Presença de Obstáculos

Autor: **Carlos Florian Heuberger**

Orientador: Dr. **Douglas Eduardo Zampieri**

Aprovado por

Prof. Dr. Douglas Eduardo Zampieri, Presidente

Prof. Dr. Clésio Luiz Tozzi

Prof. Dr. Paulo Eigi Miyagi

Campinas, 25 de outubro 1990

Agradecimentos

Ao amigo e orientador Douglas Eduardo Zampieri pela dedicação e orientação deste trabalho, feita desde o início com muita disposição e constante incentivo.

Ao Dr. Clésio Tozzi por ter colaborado na definição do conteúdo e pelas sugestões feitas durante o andamento deste trabalho.

Aos meus pais pela educação e pelo amor que me dedicaram, possibilitando este trabalho.

A Maria Bernadete Lima por não me deixar desanimar nas fases difíceis, e pelo estímulo e interesse dispensados com muito carinho durante todo o trabalho.

A Maria Elena Pousa Seara por sempre estar disposta a ajudar.

Ao DMC por prover a infra-estrutura e ceder os equipamentos necessários à realização deste trabalho, e aos colegas do departamento pela amizade, sugestões e auxílio para resolver os diversos problemas.

A UNICAMP pelo apoio financeiro para a realização deste trabalho.

Resumo

Neste trabalho propõe-se um algoritmo para a determinação automática de caminhos para um manipulador com duas juntas rotacionais, na presença de obstáculos conhecidos e fixos. A detecção de colisão é simplificada tranformando-se os obstáculos para o espaço de configuração do manipulador. A determinação do caminho consiste então, em achar a linha de menor comprimento no espaço de configuração, ligando os pontos referentes às posições inicial e final, evitando as regiões de colisão. Esta linha será determinada pelo algoritmo de procura em grafos A*, que sempre encontra o caminho ótima segundo um critério adotado, neste caso, o de menor movimento das juntas. O simulador foi implementado em um computador compatível ao IBM AT, permitindo a apresentação de resultados práticos.

Abstract

A method for automatically planning a path for a two link manipulator in the presence of fixed and known obstacles is presented. The transformation of the Cartesian workspace into the manipulator joint coordinates (configuration maps) simplify the colision detection. So the find-path problem is solved finding the shortest line which connects starting and final manipulator configurations in this map without traversing a colision region. The graph-search A* algorithm is used to calculate this line because it always finds the optimal path, which we define as being the one with the least joint movement. To show some paths obtained by this method, a simulator was implemented on an IBM AT.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

1.1 Histórico

A robótica surgiu há aproximadamente 30 anos e, desde então, vem sendo alvo de interesse cada vez maior, oferecendo um amplo campo de pesquisa e prometendo muitos desenvolvimentos para o futuro.

Os antecessores dos primeiros robôs eram manipuladores do tipo *mestre-escravo* desenvolvidos após a II Guerra Mundial principalmente para manipular materiais radioativos [N-Nagy]. Estes manipuladores eram compostos por dois braços mecânicos, sendo que o primeiro (mestre) era movido pelo operador e o segundo (escravo) era controlado de maneira a reproduzir fielmente os movimentos do primeiro. Já nos anos 50 foi desenvolvido o primeiro manipulador capaz de repetir uma seqüência de movimentos e, em 1959, a "Unimation Inc." lançou o primeiro manipulador industrial que utilizava um computador para o seu controle e programação [Fu].

O fato da seqüência de movimentos poder ser alterada tornava o manipulador programável e, portanto, não restrito a uma única tarefa, podendo ser usado em operações distintas. Esta versatilidade é a principal característica dos robôs, pois desta maneira o mesmo modelo pode ser usado em processos industriais totalmente diferentes sem grandes alterações. E,

no caso de uma mudança no processo de produção, não é necessário trocar o manipulador, apenas sua programação. Esta versatilidade é devido a possibilidade dos robôs serem programáveis e poderem interagir com o meio ambiente através de sensores. Entretanto, a programação pode ser complicada a ponto de inviabilizar o uso dos manipuladores em algumas aplicações, principalmente para complexas operações tridimensionais. Devido a isto são desenvolvidas novas técnicas para facilitar a programação dos robôs, como, por exemplo, as linguagens de definição de tarefas (ou programação a nível de tarefa). Um dos requisitos básicos para os novos sistemas é a determinação automática das trajetórias necessárias para a realização de determinado movimento ou tarefa.

Muitas vezes ainda se considera a robótica como sendo apenas uma aplicação da engenharia mecânica, mas na realidade ela deve ser vista como um campo interdisciplinar pois engloba várias áreas de conhecimento, como: mecânica (cinemática, dinâmica), eletrônica (sensoriamento, controle), planejamento de sistemas, informática, inteligência artificial, ... A contribuição de novas áreas do conhecimento humano é uma tendência irreversível pois, com o aumento da complexidade das aplicações e das tarefas que são atribuidas aos robôs, são necessários cada vez mais profissionais com formação interdisciplinar.

Atualmente estão sendo projetados robôs cada vez mais sofisticados, salientando-se as seguintes aplicações [Dillm]:

- inspeção em usinas nucleares,
- manipuladores com vários braços,
- robôs móveis,
- máquinas andantes,
- trabalho subaquático ou espacial.

Todas estas aplicações pressupõem um certo grau de autonomia dos robôs de maneira que estes possam lidar com situações imprevistas, mudar ou,

até mesmo determinar sua trajetória independente de qualquer intervenção humana. Deste modo propõe-se neste trabalho investigar alguns aspectos relevantes na determinação automática de trajetórias de robôs, aplicados ao exemplo de um manipulador atuando no espaço bidimensional. Assim serão abordados os aspectos da programação, da geração de trajetórias e da detecção de colisão. Isto implica no uso de *Inteligência Artificial*, tema abordado mais abaixo na seção 1.3

1.2 Programação

Atualmente a programação de uma variada gama de manipuladores industriais é feita através do método de aprendizagem, que consiste basicamente em *ensinar* o manipulador *mostrando* os movimentos desejados [Craig]. Este método pode ser resumido nos seguintes passos:

Ensino A tarefa é executada fazendo-se o manipulador executar o trabalho desejado, movendo-o através de controle manual com velocidade reduzida, e armazenando-se as *configurações*[1] apropriadas de maneira a ser possível reproduzir os movimentos especificados posteriormente.

Correção Os movimento são repetidos um a um, ainda com velocidade reduzida, para permitir uma verificação e uma possível correção ou edição dos movimentos erroneamente programados.

Execução Corrigidos os erros, o manipulador executará os movimentos programados em sua velocidade normal de operação em modo repetitivo. O aumento da velocidade poderá introduzir algum erro devido aos amortecimentos e às inércias envolvidas, causando uma diferença entre a trajetória programada e a realmente obtida com velocidade normal, sendo necessário, neste caso, retornar ao passo anterior.

O método de aprendizagem tem como grande desvantagem a necessidade da linha de produção ser paralisada durante o período de tempo em

[1] Posição relativa dos braços.

que o manipulador é programado e testado, acarretando uma sensível perda econômica para a indústria. Além disto existe a possibilidade de um erro de programação causar um acidente avariando o próprio manipulador, algum equipamento da célula de trabalho, e, pior ainda, ferindo alguma pessoa.

Por outro lado, a evolução da tecnologia e a necessidade de procura de novas fontes de energia, têm feito surgir algumas aplicações com acesso difícil ou onde a presença humana é impossível, como em trabalhos subaquáticos ou no caso de geradores como citado em [Caze], inviabilizando o método de aprendizagem.

Torna-se, então, essencial para tarefas mais exigentes, a interação do manipulador com a célula de trabalho[2] ou com o ambiente de trabalho. Para isto o manipulador deverá executar movimentos diferentes de acordo com os sinais recebidos por sensores externos ou internos, o que requer métodos condicionais de programação, exigindo, portanto, técnicas de programação avançadas. O surgimento de computadores cada vez mais poderosos e acessíveis possibilitou o desenvolvimento de linguagens de programação, tais como "AML" ou "VAL". Normalmente estas linguagens de programação apenas substituem o passo de *ensino* do método de aprendizagem, pois o passo de *correção* continuará sendo necessário devido aos inevitáveis desvios dimensionais que ocorrem no modelamento do manipulador, e por causa dos possíveis enganos ocorridos durante a programação.

Um passo adicional é o uso de simuladores gráficos, que normalmente são implementados em estações de trabalho[3], e cuja programação é semelhante à do próprio manipulador, mas que é, obviamente, feita através do teclado, enquanto que a posição do manipulador é mostrada no monitor deste. Desta maneira pode-se analisar a disposição dos equipamentos na célula de trabalho, bem como visualizar os movimentos resultantes de uma programação qualquer sem o perigo de algum erro de programação provocar

[2]Conjunto local de equipamentos que pode incluir um ou mais manipuladores, sistemas de transporte e alimentação, máquinas-ferramenta, ...

[3]Computadores projetados especialmente para trabalhar com imagens, possuindo um elevado desempenho computacional, ou seja, alta velocidade e grande capacidade de armazenamento.

um desastre. Isto também elimina o problema de se ter que interromper a linha de produção para programar o manipulador, que é feito, depois de depurado o programa, transferindo-se este do simulador diretamente ao manipulador.

O uso de simuladores gráficos é bastante restrito no caso de ambientes de trabalho com vários obstáculos, pois será difícil determinar e acompanhar a trajetória no monitor que estará congestionado graficamente. Este problema é resolvido, ou pelo menos amenizado, com o uso de algoritmos de *Superfícies Escondidas*[4] [Suther] e através de sistemas de *Janelas*[5]. Desta maneira pode-se analisar os detalhes e as posições mais críticas, mas ainda tem-se dificuldade para obter uma visualização global da trajetória e dos obstáculos. Para este caso é interessante o simulador ter a capacidade de determinar automaticamente o caminho entre as posições escolhidas, ou, pelo menos, fornecer uma primeira sugestão ao programador, que poderá fazer ajustes e correções onde estes forem necessários ou desejados.

1.3 Inteligência Artificial

Outro método é através da programação com linguagens a nível de tarefas, que estão sendo amplamente desenvolvidas atualmente. Estas linguagens de programação permitem ao programador especificar a tarefa que deve ser executado pelo robô ou pela célula de trabalho e os resultados desejados, ao invés de se ter que definir cada detalhe dos movimentos necessários para tal. Os sistemas programáveis através destas linguagens devem possuir inteligência para executar as várias etapas de planejamento envolvidas na decomposição da tarefa em movimentos básicos do manipulador, bem como permitir uma correção se ocorrer algum imprevisto.

Por exemplo, para a tarefa 'pegar um parafuso', o sistema deverá determinar a trajetória a ser percorrida, escolher a orientação ideal para pegar

[4] Algoritmo que calcula a cor e preenche as superfícies removendo as linhas que não poderiam ser vistas numa imagem real.

[5] Divisão da tela em *janelas* possibilitando a visualização de vários detalhes.

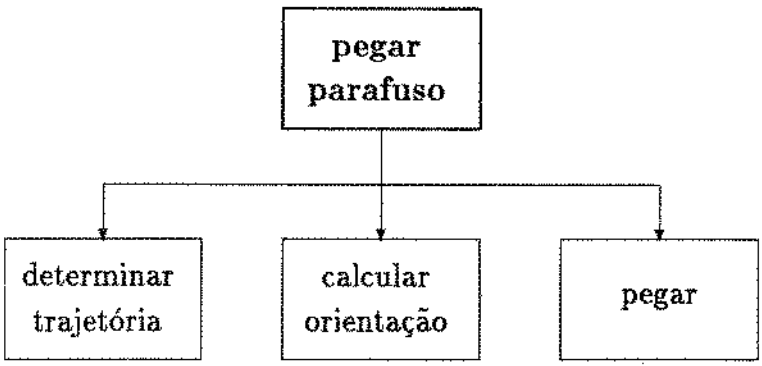


Figura 1.1: Decomposição de tarefa.

o parafuso e, finalmente, pegá-lo, como indicado no diagrama 1.1. Isto simplifica a programação dos robôs, pois a maneira natural de descrever uma tarefa é através dos resultados desejados e dos objetos envolvidos, ao invés de especificar todos os movimentos do manipulador, como é o caso dos métodos atualmente usados. É evidente que estes sistemas devem ser capazes, entre outras coisas, de determinar o caminho[6] a ser percorrido pelo manipulador evitando qualquer colisão com os obstáculos existentes na célula de trabalho.

Como foi mencionado no fim da seção 1.1, os robôs atualmente desenvolvidos (móveis, subaquáticos, ...) tem como importante característica o fato de possuirem algum tipo de autonomia, portanto devem ter um certo grau de *inteligência* para que possam lidar com situações imprevistas. Isto se aplica principalmente para os robôs móveis e "AGV"s[7], que devem ser capazes de planejar o seu caminho levando em conta os obstáculos conhecidos, e, no caso do surgimento de um obstáculo inesperado, refazer o plano estipulando outra trajetória, [Cozman].

Nos últimos parágrafos foi visto que os robôs e manipuladores, atual-

[6] Usa-se o termo caminho para denominar a parte física da trajetória, diferenciando-a das partes dependentes do tempo como a velocidade e a aceleração, ver seção 1.4.

[7] **A**utomated **G**uided **V**ehicles, veículos dirigidos automaticamente, usados principalmente para transporte de material.

mente em desenvolvimento, requerem a capacidade da determinação automática de trajetórias. Isto não deve ser visto como utopia e sim como uma realidade que logo será apenas mais um fato cotidiano, da mesma maneira como ocorreu com a própria robótica, que há alguns anos era somente assunto de livros de ficção científica. Com o aumento do poder dos computadores, num futuro não muito distante, iremos ter sistemas mais avançados e menos ortodoxos , que já vêm sendo objeto de muitos projetos [Asada,Ander]. Um exemplo nesta direção é o desenvolvimento da lógica de incertezas, ou "fuzzy logic" [Madrid], que trata com variáveis que tem significado vago como, por exemplo: mais quente, meio rápido, devagar, ...

1.4 Geração de Trajetórias

Nós, os *seres humanos*, não temos nenhuma dificuldade para definir o caminho que devemos ou queremos percorrer. Isto é feito inconscientemente e normalmente nem percebemos tal fato. Por exemplo, quando pretendemos pegar qualquer objeto, apenas pensamos em pegá-lo e o nosso cérebro já terá uma trajetória determinada para o braço executar esta tarefa. É verdade que são necessários vários anos de treinamento, também inconsciente, para atingirmos um certo grau de destreza, nem sempre suficiente. Já para uma máquina, isto é uma tarefa absurdamente complexa, quando não impossível, pois praticamente não entendemos o funcionamento de nosso cérebro, o que dificulta a transferência do nosso conhecimento para a máquina. No futuro provavelmente iremos ter computadores e robôs aptos a aprender com os próprios erros, idéia que já vem sendo amplamente desenvolvida, mas no momento estamos limitados a programá-los de maneira que estes simulem algumas poucas das nossas aptidões. Mesmo assim não podemos desprezar os benefícios oferecidos pela robótica, sem a qual não podemos imaginar o atual sistema de produção de bens.

A geração de trajetórias obviamente é uma destas aptidões, sendo que o termo trajetória é usado para descrever qual o caminho percorrido pelo

manipulador, bem como de que forma isto é feito em função do tempo, ou seja, qual a velocidade e a aceleração aplicadas ao manipulador. O caminho, que representa a parte independente do tempo, é o termo usado para designar o espaço físico percorrido por algum ponto de referência da garra do manipulador e, no caso de robôs móveis, o próprio caminho descrito por estes. A determinação de trajetórias pode, portanto, ser dividida em dois passos, a saber:

Caminho Determinação do caminho de maneira a evitar uma colisão com os obstáculos, incluindo algum critério de otimização, mas sem considerar as velocidades ou acelerações envolvidas.

Dinâmica Cálculo das velocidades e acelerações compatíveis com o caminho determinado no passo anterior.

Na realidade os critérios de otimização do primeiro passo podem incluir alguma parte dinâmica, o que não permite esta divisão, tornando a determinação do caminho dependente das características dinâmicas [Shill].

Neste trabalho é abordado o problema da determinação do caminho, isto é, da parte independente do tempo, para um manipulador composto por dois braços com juntas[8] rotacionais, como esquematizado na figura 1.2. O espaço de configuração deste manipulador será, portanto, bidimensional pois teremos dois graus de liberdade, um para cada braço. Isto não desabona o algoritmo, pois como será visto na seção 4.3 não é necessária nenhuma alteração significante para adaptá-lo a problemas com mais graus de liberdade. Obviamente estas mudanças irão dificultar alguns cálculos envolvidos, como na deteção de colisão, dificultando a compreensão do algoritmo e aumentando o tempo necessário para a procura do caminho.

Além de evitar os obstáculos, o caminho procurado deve satisfazer determinados critérios de otimização, principalmente para escolher-se um dos vários caminhos possíveis. Neste trabalho obtou-se pelo caminho com o

[8] Articulação entre dois braços consecutivos, que pode ser do tipo prismático, quando um dos braços desliza em relação ao anterior, ou rotacional, braço gira em torno da junta.

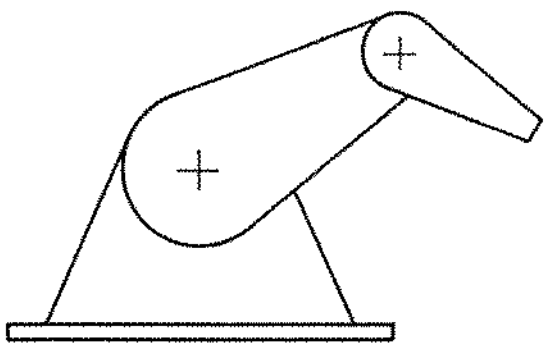


Figura 1.2: Esquema do manipulador.

menor movimento das juntas, mas outros critérios, bem como o procedimento para sua implementação, são sugeridos na seção 4.3. O algoritmo foi implementado em um microcomputador tipo IBM AT, e os resultados, para alguns exemplos, serão mostrados no capítulo 4.

Este trabalho não pretende esgotar o assunto referente à determinação automática de trajetórias, mas estabelecer bases para uma linha de pesquisa sobre o assunto, servindo como incentivo para trabalhos futuros. Neste sentido algumas sugestões são apresentadas na seção 4.3

Capítulo 2

Modelagem

No capítulo anterior foi mostrado que os sistemas de robótica requerem a capacidade da determinação automática de trajetórias, e neste capítulo são apresentados os modelos do manipulador e dos obstáculos, os principais dados e os tratamentos destes que serão usados pelo algoritmo a ser exposto finalmente no capítulo 3. Estes podem ser divididos em três classes:

Entrada Descrição do manipulador e dos obstáculos, bem como as posições inicial e final da trajetória desejada.

Intermediários Usados nos cálculos internos, ressaltando-se o mapa de configuração, que indica as configurações para as quais não há ocorrência de colisão.

Saída A finalidade do algoritmo é a determinação de um caminho, que será descrito através de segmentos de reta.

A descrição do manipulador é dividida em *modelo 'cinemático'*, descrito na seção 2.2, e *modelo geométrico*, discutido na seção 2.1, sendo que este último também engloba a descrição dos obstáculos. O mapa de configuração é apresentado na seção 2.3.

2.1 Modelo Geométrico

O objetivo do algoritmo apresentado neste trabalho é achar um caminho que evite uma colisão entre o manipulador e os obstáculos existentes no volume de trabalho. A colisão é caracterizada, em termos matemáticos, pelo fato do manipulador ocupar o mesmo espaço físico que um dos obstáculos. Portanto, para prever a ocorrência de uma colisão, para uma determinada configuração do manipulador, é necessário representar o volume ocupado tanto pelo manipulador como pelos obstáculos. O termo *modelo geométrico* é usado neste trabalho para denominar esta representação.

Além de ser usado na detecção de colisão, o modelo geométrico também será usado para representar o manipulador e os obstáculos no monitor gráfico durante a simulação, que irá mostrar visualmente qual o caminho encontrado pelo algoritmo.

A detecção de colisão deve ser otimizada pois será usada intensamente devido ao fato de uma célula de trabalho englobar vários obstáculos, e por ela ser usada para todos os pontos ao longo dos caminhos em teste, eliminando aqueles nos quais há ocorrência de colisão. Isto implica também na necessidade da otimização do modelo geométrico, principal dado usado na detecção.

Neste trabalho, como já mencionado, a representação do manipulador e dos obstáculos será no espaço bidimensional. Isto é feito projetando-se o volume ocupado tanto pelo manipulador quanto pelos obstáculos no plano de trabalho. Este procedimento garante que se houver ocorrência de colisão no espaço tridimensional, também haverá colisão das projeções no plano de trabalho. A escolha deste plano deve ser feita de modo a permitir uma representação completa, ou pelo menos suficiente, dos volumes ocupados para qualquer posição do manipulador.

Para otimizar os cálculos de detecção de colisão, as projeções são representadas por polígonos, como exemplificado na figura 2.1, devido a sua simplicidade. Estes polígonos são descritos através da lista ordenada das coordenadas de seus vértices, que, por sua vez, definem implicitamente

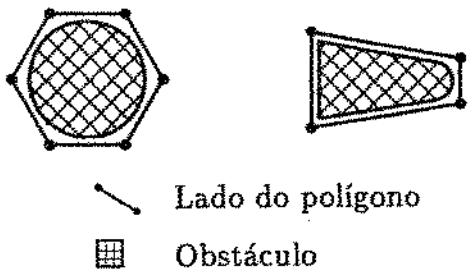


Figura 2.1: Exemplo de polígonos.

suas arestas. Para uma maior segurança pode-se definir estes polígonos adicionando uma margem vazia às projeções dos obstáculos e/ou do manipulador. As coordenadas dos polígonos que representam a projeção de um braço, são relativas ao sistema referencial[1] fixo a este, de tal modo que os polígonos reflitam automaticamente o movimento do braço. Já as coordenadas dos polígonos referentes aos obstáculos, tem suas coordenadas definidas em relação ao sistema referencial global (S_0)[1], por serem imóveis.

A detecção de colisão resume-se em determinar a existência de uma interseção não nula entre os polígonos referentes aos obstáculos e aqueles referentes ao manipulador, para uma dada posição deste. A interseção entre dois polígonos quaisquer pode ocorrer de duas maneiras distintas, mostradas na figura 2.2:

- Um polígono totalmente contido no outro.
- Um polígono parcialmente contido no outro.

No primeiro caso, para se determinar a ocorrência de colisão, basta verificar se um dos vértices do primeiro polígono está contido no segundo. Isto já não é suficiente para afirmar que não ocorre colisão no segundo caso,

[1]Descrito na seção 2.2.

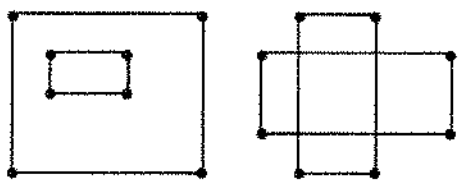


Figura 2.2: Interseção de polígonos.

pois neste todos os vértices estão fora do outro polígono, ainda ocorrendo a colisão, o que é exemplificado na figura 2.2 a direita. Neste caso a verificação é feita através de um teste de interseção entre as arestas do primeiro polígono com aquelas do segundo. Resumindo, a detecção de colisão envolve cálculo da interseção entre dois segmentos de reta e a determinação se um ponto está contido em um polígono.

A seguir descreve-se o cálculo da interseção entre dois segmentos de reta, supondo que estes são definidos pelos pontos extremos. Por exemplo, se o primeiro segmento for definido por (x_1, y_1) e (x_2, y_2), sua representação paramétrica será:

$$\left.\begin{array}{rcl} x & = & x_1 + \alpha(x_2 - x_1) \\ y & = & y_1 + \alpha(y_2 - y_1) \end{array}\right\} 0 \leq \alpha \leq 1$$

Se os extremos do segundo segmento são (x_3, y_3) e (x_4, y_4), então a interseção entre estes dois segmentos é deduzida abaixo:

$$\left.\begin{array}{rcl} x & = & x_1 + \alpha\Delta X_{12} \\ y & = & y_1 + \alpha\Delta Y_{12} \end{array}\right\} 0 \leq \alpha \leq 1 \qquad (2.1)$$

$$\left.\begin{array}{rcl} x & = & x_3 + \beta\Delta X_{34} \\ y & = & y_3 + \beta\Delta Y_{34} \end{array}\right\} 0 \leq \beta \leq 1 \qquad (2.2)$$

onde

$$\begin{aligned} \Delta X_{ij} &= x_i - x_j \\ \Delta Y_{ij} &= y_i - y_j \ . \end{aligned}$$

Igualando as coordenadas obteremos:

$$\begin{aligned} x_1 + \alpha\Delta X_{21} &= x_3 + \beta\Delta X_{43} \\ y_1 + \alpha\Delta Y_{21} &= y_3 + \beta\Delta Y_{43} \end{aligned}$$

e rearranjando teremos o seguinte sistema:

$$\begin{aligned} \alpha\Delta X_{21} - \beta\Delta X_{43} &= \Delta X_{31} \\ \alpha\Delta Y_{21} - \beta\Delta Y_{43} &= \Delta Y_{31} \end{aligned}$$

resolvendo para α e β :

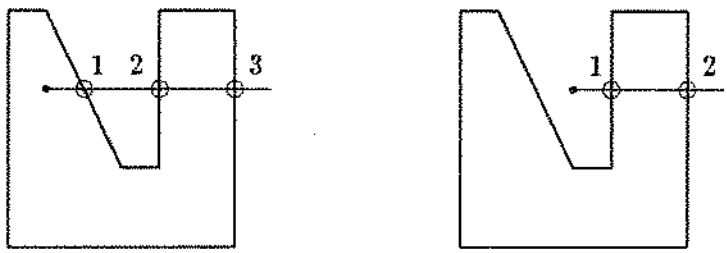
$$\begin{aligned} \alpha &= (\Delta XY_{34} + \Delta XY_{13} + \Delta XY_{41})/D \\ \beta &= (\Delta XY_{32} + \Delta XY_{13} + \Delta XY_{21})/D \end{aligned}$$

com

$$\begin{aligned} \Delta XY_{ij} &= x_i y_j - x_j y_i \\ D &= \Delta X_{21}\Delta Y_{43} - \Delta X_{43}\Delta Y_{21} \end{aligned}$$

Se $D = 0$ então $\Delta X_{21}/\Delta Y_{21} = \Delta X_{43}/\Delta Y_{43}$, ou seja, as retas são paralelas e podemos considerar este caso como tendo interseção nula, pois esta será verificada através dos outros lados dos polígonos. Por outro lado se α ou β estiverem fora do intervalo [0,1] a interseção ocorre fora dos limites do(s) segmento(s), o que deve ser considerado como uma interseção nula. Portanto, se $D \neq 0$, $0 \leq \alpha \leq 1$ e $0 \leq \beta \leq 1$ existe uma interseção cujas coordenadas são determinadas substituindo-se o valor de α nas equações 2.1, ou o de β nas equações 2.2.

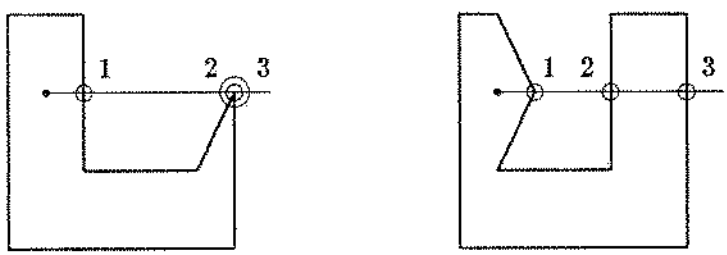
Existem três métodos na computação gráfica para determinar se um ponto está contido em um polígono [Suther]. O primeiro é através da contagem do número de interseções que ocorrem entre os lados do polígono e uma reta "semi-infinita", isto é, iniciando no ponto em teste e indo ao infinito. Se o número de interseções for par, o ponto estará fora do polígono, se for impar, estará dentro, como pode ser visto na figura 2.3. Uma complicação deste método ocorre no caso da reta passar exatamente por um



• ponto em teste
○ interseção da reta com o polígono

Figura 2.3: Método da contagem de interseções.

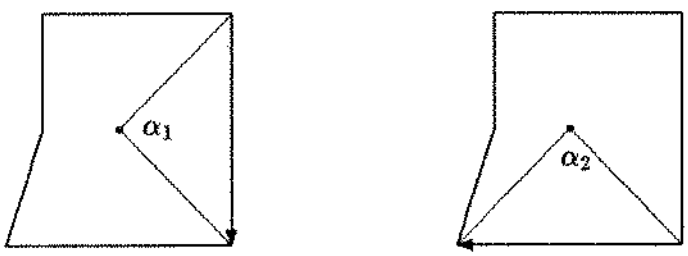
dos vértices do polígono, pois neste ponto temos dois lados do polígono e deve-se contar uma ou duas interseções, dependendo da posição relativa das arestas (fig. 2.4). Este procedimento é esquematizado abaixo, para uma reta "semi-infinita" horizontal:



• ponto em teste
○ interseção da reta com o polígono

Figura 2.4: Reta que passa por um vértice.

X, Y = coordenadas do ponto a ser analisado
$X(i), Y(i)$ = coordenadas do vértice i
N = número de vértices
$C = 0$
para $I = 1 \cdots N$
    se $(Y = Y(I))$ e $(X = X(I))$
        $C = C + 1$ ou $C + 2$, (ver texto)
    caso contrário
        $XT = \frac{X(I+1)-X(I)}{Y(I+1)-Y(I)} \times (Y - Y(I)) + X(I)$
        se $(XT > X)$ e $(XT$ entre $[X(I), X(I+1)])$
            $C = C + 1$
se C for par
    retornar($fora$)
caso contrário
    retornar($dentro$)



$\alpha_1 > 0$ $\alpha_2 > 0$

$\sum \alpha_i = 2\pi$

Figura 2.5: Método da soma de ângulos (ponto dentro).

O segundo método consiste em se somar os ângulos que cada aresta do polígono forma em relação ao ponto em teste, seguindo sempre a mesma orientação das arestas, como pode ser visto na figura 2.5 (ponto dentro), ou na fig. 2.6 (fora). Esta soma sempre será nula ou um múltiplo, positivo ou negativo, de 2π. Se for nula, o ponto estará fora do polígono, caso contrário

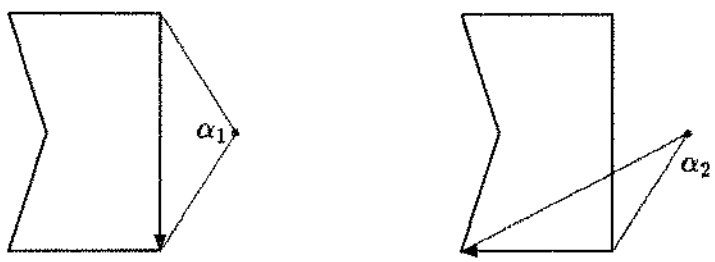


Figura 2.6: Soma de ângulos (ponto fora).

estará dentro. A soma só será maior, em módulo, que 2π quando o polígono se sobrepor a sí mesmo. Esquematizando:

X, Y = coordenadas do ponto a ser analisado
$X(i), X(i)$ = coordenadas do vértice i
N = número de vértices
$A = 0$
para $I = 1 \cdots N$
    $XA = X(I) - X$
    $YA = Y(I) - Y$
    $XB = X(I+1) - X$
    $YB = Y(I+1) - Y$
    $AA = \arctan(YA/XA)$
    $AB = \arctan(YB/XB)$
    $A = A + AB - AA$
se $A = 0$
  retornar(*fora*)
caso contrário
  retornar(*dentro*)

Figura 2.7: Método vetorial (ponto dentro).

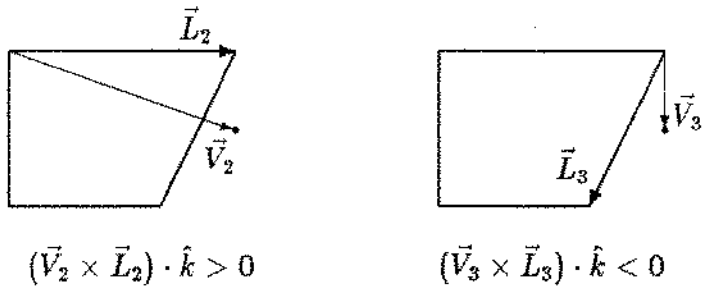


Figura 2.8: Método vetorial (ponto fora).

O terceiro método, que foi adotado neste trabalho, consiste em se verificar qual a posição do ponto em relação às arestas do polígono, que deverá ser convexo. O ponto estará dentro do polígono, se estiver do mesmo lado, em relação às arestas, que um dos vértices que não faz parte desta aresta. Uma otimização a este método é o uso de produto vetorial, ou seja, considerando as arestas como vetores, segundo uma certa orientação, faz-se o produto vetorial entre a aresta e o vetor do ponto à aresta como mostrado na figura 2.7. Se os produtos para todas as arestas tiver o mesmo sentido, o ponto estará no interior do polígono (fig. 2.7), mas se um dos produtos

tiver sentido contrário, o ponto estará fora (fig. 2.8).

Figura 2.9: Polígonos convexos modelando um polígono não-convexo.

Os primeiros dois métodos envolvem a divisão de números, que é computacionalmente muito mais custoso que a multiplicação, enquanto que o terceiro envolve apenas a multiplicação, além de soma e subtração. O inconveniente do terceiro método é que este é restrito a polígonos convexos, enquanto os anteriores também aceitam polígonos não-convexos. Contudo este polígonos poderão ser representados, no algoritmo proposto, através da união de polígonos convexos, o que deve ser considerado no modelamento e, obviamente, na detecção de colisão, figura 2.9.

2.2 Modelo Cinemático

O modelo 'cinemático', de um modo geral, descreve o movimento dos objetos sem considerar as forças causadoras, ou seja, é uma descrição da posição, velocidade e aceleração do objeto sendo estudado. Como já foi mencionado, não se está interessado neste trabalho, a princípio, nos fatores dependentes do tempo, apenas no caminho percorrido pelo manipulador. Esta é a razão das aspas na palavra *cinemático*, pois na verdade o modelo apenas descreve as características geométricas do manipulador. Usa-se o termo 'cinemático' por se basear nos métodos da cinemática para representar a posição dos braços do manipulador e para diferenciar este modelo do descrito na seção 2.1.

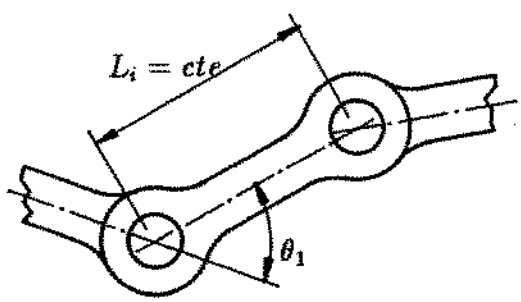


Figura 2.10: Junta rotacional.

O manipulador pode ser considerado como um conjunto de corpos rígidos, denominados *braços*, ligados em série através de articulações ou *juntas*. As juntas podem ser do tipo *rotacional*, quando o braço roda em torno da junta como ilustrado na figura 2.10, ou *prismático*, se o movimento relativo entre os braços for de translação, fig. 2.11. No caso de se ter uma junta com n graus de liberdade, esta pode ser modelada como sendo composta por n juntas com um grau de liberdade cada, unidas por $n-1$ braços sem comprimento.

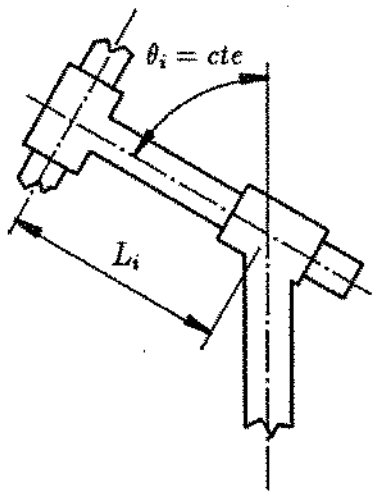


Figura 2.11: Junta prismática.

Motores atuantes nas juntas movem os braços e, conseqüentemente, controlam o posicionamento da garra[2] ou da ferramenta. Para representar esta posição, define-se um ponto fixo no braço ou na junta que é denominado Ponto Central de Ferramenta ou "PCF"[3], e normalmente é o ponto em torno do qual a garra gira [N-Nagy, cap.4]. Os dois principais problemas envolvidos com o cálculo da posição do manipulador, abordados pela

[2]Ferramenta usada para pegar ou manusear objetos.

[3]Tradução de "TCP", Tool Center Point.

cinemática, são:

Direto Conhecida a configuração do manipulador, isto é, a posição de cada braço em relação ao anterior, procura-se calcular a posição e orientação do "PCF".

Inverso Dada a posição (e/ou orientação) do "PCF', deseja-se saber qual a posição relativa dos braços para obter esta posição, e de quantas maneiras isto é possível.

As dimensões e posições dos braços são descritas através dos coeficientes de "Denavit e Hartenberg", amplamente usados na robótica [Denav]. Baseado no fato que são suficientes quatro variáveis para definir uma reta no espaço tridimensional (figura 2.12), Denavit e Hartenberg propõem uma notação matricial para descrever mecanismos, usando quatro parâmetros para cada junta. Três serão constantes, dependendo apenas da geometria do braço e da junta, enquanto que o quarto corresponde à variável da junta.

O método de Denavit e Hartenberg descreve a relação entre dois corpos rígidos adjacentes no mecanismo, através da rotação e translação entre sistemas referenciais fixos nestes. A transformação entre estes sistemas referenciais é representada por uma "matriz homogênea" de ordem 4×4, que incorpora a matriz de rotação (3×3) e o vetor de translação (3×1), o que implica na adição de uma quarta coordenada aos vetores de posição [Craig, cap.2]. A matriz homogênea correspondente a uma rotação, definida pela matriz R, e uma translação T, é representada por:

$$M = \left[\begin{array}{ccc|c} R_{xx} & R_{xy} & R_{xz} & T_x \\ R_{yx} & R_{yy} & R_{yz} & T_y \\ R_{zx} & R_{zy} & R_{zz} & T_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

com:

$$R = \left[\begin{array}{ccc} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{array}\right] \quad \text{e} \quad T = \left\{\begin{array}{c} T_x \\ T_y \\ T_z \end{array}\right\} \ .$$

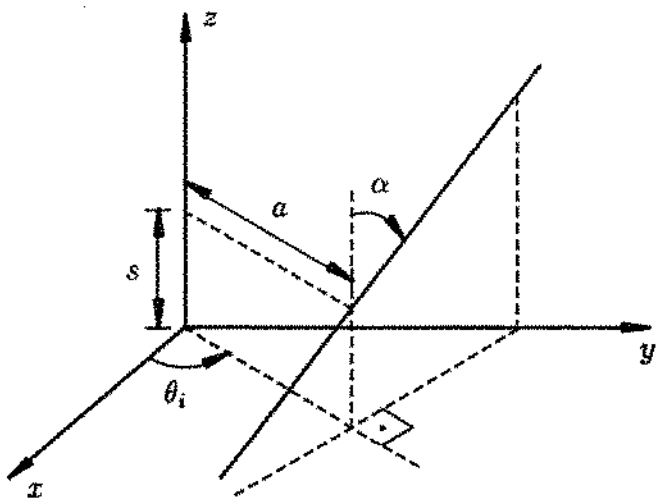


Figura 2.12: Representação de uma reta no espaço tridimensional.

Um vetor posição genérico terá a seguinte configuração:

$$\begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix}$$

Estes vetores são normalizados após uma operação, dividindo-os pelo valor da quarta coordenada, de maneira que esta sempre tenha valor unitário:

$$\begin{Bmatrix} a \\ b \\ c \\ d \end{Bmatrix} \equiv \begin{Bmatrix} a/d \\ b/d \\ c/d \\ d/d \end{Bmatrix} = \begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix}$$

Os braços são numerados a partir da base, que é considerada como se fosse o 'braço' zero, e as juntas de maneira que, a primeira de cada braço receba o mesmo número que este, isto é, a junta entre os braços $i-1$ e i

é numerada por i. O sistema referencial fixo ao braço i será identificado por S_i e um vetor representado neste sistema será $\vec{P}_i$. Desta maneira, para o braço i, a transformação de um ponto $\vec{P}_{i-1}$, representado em S_{i-1}, para sua representação em S_i é dada por $\vec{P}_i = M_i^{i-1} \times \vec{P}_{i-1}$, onde M_i^{i-1} é a matriz de transformação homogênea do sistema S_{i-1} para o sistema S_i. A transformação de S_i para S_{i-1} é dada por $\vec{P}_{i-1} = M_{i-1}^{i} \times \vec{P}_i$, onde M_{i-1}^{i} é a inversa de M_i^{i-1}. A matriz transformação M_{i-2}^{i} entre dois sistemas $i-2$ e i, é obtida através da multiplicação das duas matrizes envolvidas, como mostrado na seguinte dedução:

$$\begin{aligned}
\vec{P}_{i-2} &= M_{i-2}^{i} \times \vec{P}_i \\
\vec{P}_{i-2} &= M_{i-2}^{i-1} \times \vec{P}_{i-1} \\
\vec{P}_{i-1} &= M_{i-1}^{i} \times \vec{P}_i \\
\vec{P}_{i-2} &= M_{i-2}^{i-1} \times \left(M_{i-1}^{i} \times \vec{P}_i \right) \\
\text{logo} \quad M_{i-2}^{i} &= M_{i-2}^{i-1} \times M_{i-1}^{i}
\end{aligned}$$

Este procedimento pode ser extendido para quaisquer sistemas j e k, de maneira que $\vec{P}_j = M_j^k \times \vec{P}_k$, então:

$$M_j^k = \prod_{i=j+1}^{k} M_{i-1}^{i} \qquad \text{para } j < k$$

ou

$$M_j^k = M_j^{j+1} \times M_{j+1}^{j+2} \times \cdots \times M_{k-2}^{k-1} \times M_{k-1}^{k}$$

O manipulador estudado neste trabalho tem dois braços e se movimenta no espaço bidimensional. Portanto podemos simplificar a notação de Denavit e Hartenberg, aumentando a eficiência do algoritmo. No plano a descrição de cada braço é feita através de apenas dois parâmetros, ao invés dos quatro necessários no espaço tridimensional. Fixou-se um sistema a cada braço e um sistema adicional à base do manipulador, como indicado

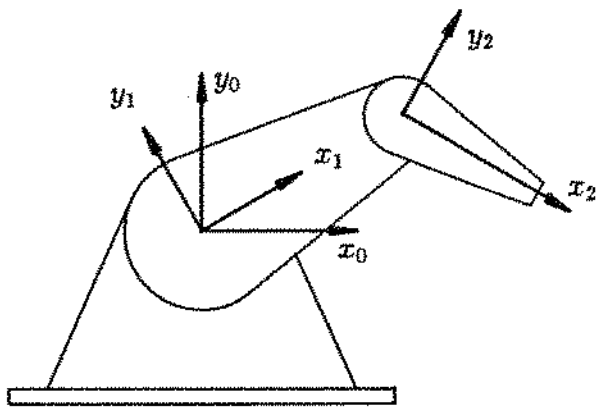


Figura 2.13: Sistemas referenciais.

na figura 2.13. A matriz de transformação relacionada ao braço i $(i = 1, 2)$ que transforma do sistema S_i para S_{i-1}, será:

$$M_{i-1}^{i} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & L_{i-1} \\ \sin\theta_i & \cos\theta_i & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

e, conseqüentemente, sua inversa será:

$$M_{i}^{i-1} = \begin{bmatrix} \cos\theta_i & \sin\theta_i & -L_{i-1} \times \cos\theta_i \\ -\sin\theta_i & \cos\theta_i & L_{i-1} \times \sin\theta_i \\ 0 & 0 & 1 \end{bmatrix}$$

onde θ_i é o ângulo formado pelo braço i com o braço $i-1$, e L_i é a distância entre as juntas do braço i ($L_0 = 0$, por definição). No nosso caso os valores de L_i serão constantes e as variáveis das juntas serão θ_1 e θ_2.

Através da mudança da origem dos sistemas referenciais de cada braço para a outra junta deste, estes acompanham a variação do comprimento do braço. Desta maneira o algoritmo também funciona, sem qualquer alteração, para juntas prismáticas, pois os polígonos definidos no referencial

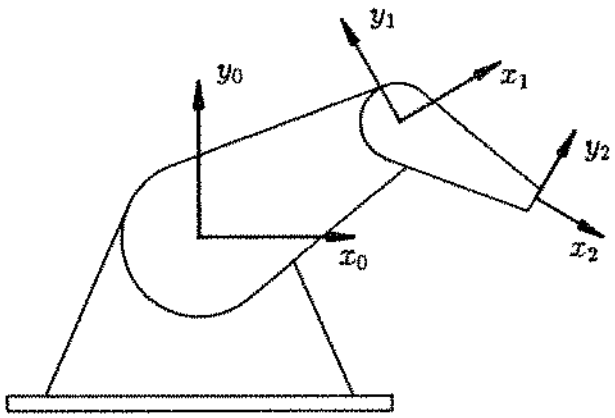


Figura 2.14: Sistemas referenciais alterados.

fixo no braço, irão acompanhar os movimentos deste. Os sistemas alterados podem ser vistos na figura 2.14. As novas matrizes, para este caso, serão:

$$M_{i-1}^{i} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & L_i \times \cos\theta_i \\ \sin\theta_i & \cos\theta_i & -L_i \times \sin\theta_i \\ 0 & 0 & 1 \end{bmatrix}$$

$$M_{i}^{i-1} = \begin{bmatrix} \cos\theta_i & \sin\theta_i & -L_i \\ -\sin\theta_i & \cos\theta_i & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2.3 Mapa de Configuração

O *espaço* de configuração de um manipulador é o espaço cujas coordenadas são as variáveis das juntas deste, isto é, uma coordenada para cada grau de liberdade do manipulador. Qualquer posição do manipulador é representada neste espaço através de um ponto, cujas coordenadas representam as posições relativas entre os braços. Um caminho é representado neste espaço através de uma linha, possivelmente curva, ligando o ponto que representa a configuração inicial, ao ponto referente à configuração final. Quanto mais longa esta linha mais as juntas se moverão para seguir o caminho correspondente. Portanto o caminho de menor movimento, aquele que está sendo procurada neste trabalho, será obtido ligando os pontos por uma reta no espaço de configuração. Neste caminho as juntas se movem durante o mesmo intervalo de tempo, de maneira que as juntas que necessitariam de menos tempo se movam com uma velocidade inferior à máxima permitida dinamicamente, para acompanhar aquelas que necessitam de maior tempo, estratégia usada em muitos manipuladores.

Para o caso de duas juntas rotacionais este espaço será bidimensional (apenas duas variáveis) e as coordenadas serão os ângulos das duas juntas (θ_1 e θ_2). Já para um manipulador com seis juntas, teriamos um espaço de configuração com seis dimensões, de impossível representação num monitor. Mas como o seu uso somente é interno ao programa, esta representação não é necessária.

O "mapa de configuração" é o termo usado para designar a representação dos obstáculos no espaço de configuração [Red85]. Na realidade os obstáculos não são apenas representados neste espaço, e sim transformados para este considerando-se também o volume ocupado pelo manipulador. Esta transformação consiste em estabelecer quais as configurações do manipulador nas quais haveria colisão, e marcá-las no mapa. Isto significa que teremos um *mapa* das configurações nas quais há ocorrência de colisão e, portanto devem ser evitadas. Para duas juntas, o mapa é feito da seguinte

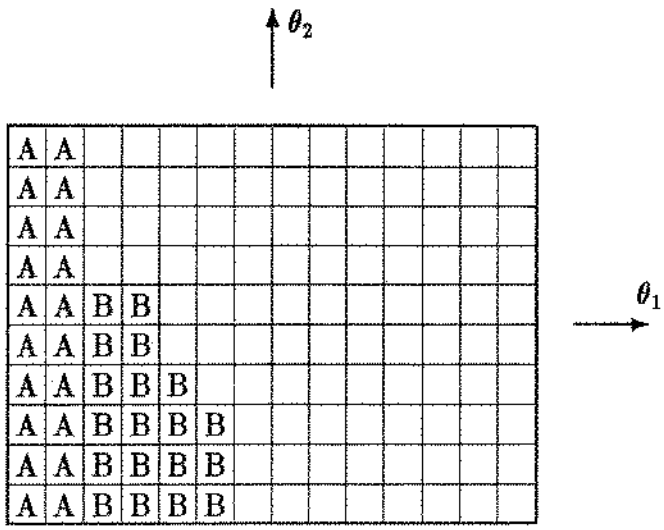


Figura 2.15: Exemplo de elaboração do mapa de configuração.

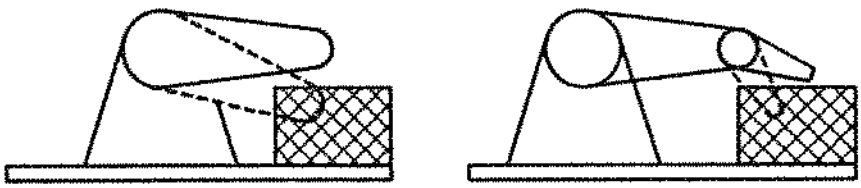
forma:

1. Discretiza-se o espaço de configuração, estabelecendo-se um incremento para cada grau de liberdade do manipulador (θ_1 e θ_2).

2. Para cada posição discretizada do primeiro braço, faz-se o teste de colisão entre este e os obstáculos sem considerar o segundo braço. Se houver colisão, são marcados no mapa os pontos correspondentes a esta posição do primeiro braço para todas as posições do segundo.

3. Em todas as posições sem colisão do primeiro braço, determinadas no passo anterior, testa-se a ocorrência de colisão do segundo braço para todas as posições discretas deste. No caso de colisão marca-se o ponto correspondente no mapa.

Este processo é exemplificado na figura 2.15, onde 'A' marca as posições de colisão do primeiro braço (item dois acima) e 'B' as do segundo (terceiro item), figura 2.16 esquerda e direita respectivamente.



▦ Obstáculo

Figura 2.16: Colisão de cada braço.

Desta forma o volume ocupado pelo manipulador é adicionado ao volume ocupado pelos obstáculos, reduzindo a representação do volume do manipulador a um ponto no mapa de configuração, o que facilita em muito a determinação do caminho, tendo em vista que para qualquer configuração sabe-se de imediato se haverá colisão. A determinação do caminho é reduzida, após esta transformação, à procura de um caminho para um ponto evitando as regiões de colisão no mapa de configuração, que corresponde ao problema de navegação com os obstáculos equivalendo às regiões de colisão. Esta transformação do espaço real para o espaço de configuração, é necessária uma vez que a orientação do volume do manipulador varia, e muito, com a configuração deste.

Capítulo 3

Algoritmo

A implementação do algoritmo, bem como o método usado para tal, serão discutidos neste capítulo. O método de programação escolhido foi o da programação orientada por objetos, que será visto na seção 3.1. O algoritmo é composto por três módulos, com funções distintas, que serão tratados em seções separadas:

3.2 Elaboração do mapa de configuração.

3.3 Procura do caminho ótimo.

3.4 Simulação do caminho encontrado.

Cada módulo é independente dos outros, com exceção dos dados transferidos entre eles. Devido a esta independência é possível implementar cada módulo como um programa distinto, e fazer a troca de informação entre os módulos através de arquivos de dados, facilitando sua implementação e teste. A figura 3.1 mostra a relação entre os módulos, representados por retângulos, e os dados trocados, entre parênteses, que são identificados pela

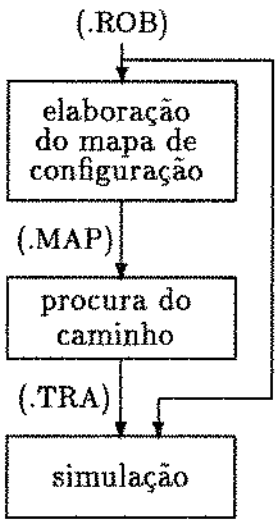


Figura 3.1: Módulos e dados do algoritmo.

extensão do respectivo arquivo:

(.ROB) contém a descrição do manipulador e dos obstáculos.

(.MAP) é composto por

- nome do arquivo de entrada do primeiro módulo,
- valores iniciais e incrementos das variáveis das juntas,
- dimensão do mapa de configuração, e
- o próprio mapa de configuração.

(.TRA) formado por

- nome do arquivo de entrada do primeiro módulo,
- valores iniciais e incrementos das variáveis das juntas,
- os pontos que descrevem o caminho.

3.1 Programação

O algoritmo apresentado neste trabalho foi implementado usando a linguagem C, seguindo os princípios da programação orientada por objetos, que será detalhada nesta seção. Este método de programação está sendo amplamente desenvolvido e aplicado aos mais diversos problemas. Exemplos de linguagens neste contexto são:

Smalltalk Na realidade é um ambiente de programação totalmente orientado por objetos, isto é, o próprio ambiente é feito em *Smalltalk*, usando as técnicas da programação por objetos [Small]. O grande inconveniente desta linguagem é a quantidade de memória necessária, bem como o tempo gasto para programas de médio porte. Entretanto ela é interessante para o aprendizado dos conceitos e da metodologia da programação orientada por objetos. Existem duas versões principais: o "Smalltalk/V" e o "Smalltalk-80".

C++ Consiste numa extensão da linguagem C que permite ao programador implementar objetos, mas não é uma linguagem puramente por objetos como é o *Smalltalk*. Possui as vantagens da linguagem C, discutidas adiante, incentivando a programação por objetos [Strous].

C_Talk Um ambiente de programação que mistura os aspectos de C com as do *Smalltalk*.

A linguagem C está sendo cada vez mais usada no meio científico, o que também vem acontecendo com o sistema operacional *UNIX*, desenvolvido nesta linguagem, devido a sua eficiência, economia no uso de memória e

portabilidade. A linguagem C é composta por:

- Conjunto completo de comandos condicionais, de "loop's" e de transferência, permitindo um controle lógico do programa e incentivando sua estruturação.

- Amplo conjunto de operadores possibilitando uma especificação clara das operações desejadas. Para permitir a codificação direta para linguagem de máquina, vários operadores correspondem às próprias operações da máquina.

- Vários tipos de variáveis, incluindo diversos tamanhos para inteiros, além da possibilidade de definir tipos mais complexos como vetores e estruturas (conjunto de variáveis diferentes).

- Ponteiros para dados e para funções, correspondendo ao método usado pela máquina para endereçar os dados internamente. O uso adequado de ponteiros aumenta consideravelmente a eficiência do programa, além de possibilitar o acesso direto às memórias.

- O pré-processador de C é um processador de textos que prepara o texto dos programas para a compilação permitindo a definição de constantes e a compilação condicional, ou seja, uma parte do programa só será compilada se uma determinada condição for verdadeira como, por exemplo, uma variável estar definida.

- 'C' praticamente não contem funções internas, mas o programador dispõe de uma ampla biblioteca com as mais variadas funções e subrotinas para entrada e saída, manipulação da tela gráfica, controle de processos, e alocação dinâmica de memória.

3.1.1 Programação Orientada por Objetos

A idéia básica da *programação orientada por objetos*, abreviada 'POO', é tornar a programação mais próxima da linguagem natural, facilitando a

programação e a compreensão dos programas [Taze]. A maneira natural de lidarmos com o ambiente que nos cerca é através de *objetos*. Uma planta não é vista como um conjunto de átomos e sim como uma unidade: "a planta". Esta pode ser dividida em partes como folhas, flores, caule e raiz, mas que continuam sendo vistos por nós como unidades. O método da POO é semelhante a este exemplo, o programa é tratado como um objeto formado por vários subobjetos, que, por serem objetos, também podem ser formados por outras subpartes.

Um objeto representando um manipulador, por exemplo, é formado pelo objeto BASE[1], e por dois BRAÇO's. O BRAÇO é formado pelo objeto representando o volume ocupado (REPRESENTAÇÃO) e pela JUNTA, que, por sua vez, pode ser formada também por uma REPRESENTAÇÃO e por um MOTOR. Ou, como exemplo mais simples, a representação de um ponto no espaço bidimensional, é definida por um objeto composto por dois subobjetos (número INTEIRO ou REAL) representando suas coordenadas.

Na programação convencional, os dados são entidades separadas dos procedimentos que irão lidar com estes. Já na POO, os dados e os respectivos procedimentos são integrados nos objetos, definidos através das chamadas *classes* que são compostas por duas partes ou características:

Estado Descreve o estado de cada objeto. Atribui-se a cada *instância* de um objeto uma região da memória onde é armazenado o estado deste, ou mais especificamente, os dados que formam esta instância. Estes dados somente são acessíveis à própria instância.

Comportamento É definido através de funções que tem acesso aos dados privativos da instância. Cada função é chamada na POO de método e pode ter argumentos de chamada, bem como retornar alguma informação. Em *Smalltalk*, por exemplo, os argumentos e o dado retornado também são objetos.

Desta maneira cada instância de um objeto terá seus próprios dados,

[1]Base é escrita com letras maiúsculas para definir a instância do objeto que representa uma base, o mesmo acontecendo para as demais instâncias.

independentes das outras instâncias e objetos, mas os métodos serão os mesmos para todas as instâncias de um mesmo objeto. Cada PONTO, por exemplo, tem associado a sí dois números que representam a sua posição, mas todos compartilham o mesmo método para mudar sua posição, ou fornecer sua coordenada 'y', por exemplo.

3.1.2 Métodos e Mensagens

O único meio de acesso aos dados do objeto é através dos seus métodos. Por causa disto um objeto representando um ponto bidimensional deve possuir um método para retornar as suas coordenadas, e outro para alterá-las. Estes métodos são chamados (acionados) através de mensagens que são enviadas aos objetos. Por exemplo, para se desenhar uma linha envia-se a mensagem 'desenhar' ao objeto que representa esta linha, e que irá executar a função 'desenhar' associada a este objeto. Este método, por sua vez, irá enviar as mensagens necessárias aos PONTOs, que definem esta linha, para obter suas coordenadas e gerar a linha desejada. Já um objeto representando uma circunferência pode aceitar a mesma mensagem 'desenhar', mas irá executar a sua função de desenho, diferente da do PONTO.

3.1.3 Polimorfismo

A habilidade, intensamente usada nos sistemas de POO, de objetos diferentes responderem à mesma mensagem, como no exemplo do parágrafo anterior, é chamada de *polimorfismo*. Isto simplifica muito a programação como se pode ver no exemplo de uma figura definida por vários objetos gráficos, como LINHAS e CIRCUNFERÊNCIAS. O método 'desenhar' da FIGURA, chamada pela mensagem de mesmo nome, simplesmente irá enviar a mesma mensagem 'desenhar' para todos os objetos que a compõem, fazendo-os se desenharem. Desta maneira é simples adicionar um novo objeto, por exemplo o RETÂNGULO, para ser usado com este objeto (FIGURA), pois já teremos a definição de LINHA, usada para desenhar o retângulo, e só é

preciso incluir o método 'desenhar' à definição do RETÂNGULO, não sendo necessário nenhuma alteração do objeto FIGURA.

3.1.4 Herança

Outra característica importante das linguagens de POO é a capacidade de uma classe de objetos *herdar* as definições de outra classe, denominada neste caso de superclasse da anterior. Isto significa que a nova classe incluirá em sua definição o estado e o comportamento de sua superclasse, mas podendo ter dados ou métodos adicionais. Em termos práticos, as instâncias da nova classe terão entre seus próprios dados, os dados da superclasse, e se um método não for definido na nova classe será procurado na sua superclasse. Por exemplo, uma circunferência pode ser considerada como um ponto com um determinado raio, ou seja, irá herdar as definições do PONTO acrescentando-se o raio e alterando-se o método 'desenhar'. Desta forma a CIRCUNFERÊNCIA irá responder da mesma maneira às mensagens do PONTO, com exceção dos métodos alterados, como por exemplo 'desenhar'. Se a mensagem 'mover' muda a posição do PONTO, por exemplo, o mesmo irá ocorrer com a CIRCUNFERÊNCIA.

3.1.5 Encapsulamento

Encapsulamento é o termo técnico usado para denominar o fato dos dados de um objeto só poderem ser acessados através dos métodos deste objeto e, portanto, serem *protegidos* dos demais objetos, isto é, os dados são 'encapsulados' com os procedimentos que os acessam nos objetos, ao invés de serem globais.

O encapsulamento separa a implementação dos dados e procedimentos, de seu uso, sendo que o único elo entre estes é o envio de mensagens. Desta maneira pode-se mudar a implementação de um objeto sem ser necessário alterar os procedimentos que o usam. O processo interno pode ter mudado, mas o novo objeto irá responder da mesma maneira às mensagens recebidas,

apesar destas mudanças. Isto também implica que, uma vez implementado o objeto, não é necessário saber de que maneira isto foi feito, sendo suficiente conhecer os métodos e os parâmetros destes.

O encapsulamento evita muitos problemas que ocorrem devido a erros na programação convencional, principalmente advindos do uso de ponteiros que, em programas de grande porte, provocam os mais complicados problemas, inclusive causando uma paralisação do sistema operacional. Já em *Smalltalk* isto nunca (ou quase nunca) ocorre, pois no caso de um erro é reportado que uma mensagem não foi entendida pelo objeto em questão.

3.1.6 Objetos em 'C'

Como já foi dito, o algoritmo foi implementado em 'C' seguindo algumas regras da POO. Não se pretendia fazer um sistema de programação por objetos, apenas usar os conceitos desta para facilitar a programação e o entendimento dos programas. Os principais conceitos usados foram o encapsulamento dos dados (subseção 3.1.5) e a integração dos dados com as respectivas funções nos 'objetos' (3.1.1).

Cada objeto, composto por seus dados e métodos, é definido em um arquivo diferente implicando na integração dos métodos com os respectivos dados, e ao mesmo tempo, no encapsulamento destes como explicado a seguir. Os dados do objeto são definidos como estruturas, ou seja, uma coleção de diferentes dados, e referenciados através de ponteiros. A instância é definida, no respectivo arquivo, como um ponteiro para esta estrutura para possibilitar que as sub-rotinas tenham acesso aos dados. Por outro lado, fora do arquivo, as instâncias são definidas apenas como ponteiros, sem definição de tipo[2], de maneira a impossibilitar o acesso aos dados da estrutura garantindo o encapsulamento dos dados.

O envio das mensagens é simulado através da chamada da sub-rotina que representa o método desejado, tendo como argumento a instância, ou melhor, o ponteiro para os dados desta, seguido dos demais argumentos.

[2]em 'C' esta definição é programada através de '`void *`'.

Nota-se que este procedimento não permite o polimorfismo (seção 3.1.3), isto é, métodos de diferentes objetos chamados com a mesma mensagem, pois não se pode ter duas sub-rotinas com mesmo nome. Para contornar esta restrição, o nome do objeto foi adicionado ao de cada método para formar o nome da sub-rotina como, por exemplo, para o objeto PONTO a sub-rotina representando o método 'mover' é denominada "moverPonto", mas é responsabilidade do programador chamar a sub-rotina correta.

A capacidade de um objeto herdar a definição de outro objeto (3.1.4) não é obtida automaticamente através do procedimento descrito anteriormente, mas pode ser simulada sem grandes complicações. Para isto inclui-se na definição do novo objeto, como uma variável adicional, uma instância do outro objeto a ser herdado. Os métodos que devem ser herdados devem ser redefinidos no novo objeto de maneira a chamar os métodos originais do antigo objeto. Por exemplo o objeto CIRCUNFERÊNCIA, que deve herdar a definição do PONTO, terá o método 'mover' implementado através da sub-rotina chamada "moverCircunferência" que somente irá enviar a menssagem 'mover' ao PONTO.

3.2 Mapa de Configuração

O primeiro módulo é responsável em produzir o *mapa de configuração*, introduzido na seção 2.3, que será detalhado nesta seção. O mapa de configuração consiste na representação dos obstáculos, como já foi apresentado, e é usado como parâmetro para a segunda parte do algoritmo responsável pela procura do caminho.

A descrição do manipulador e dos obstáculos, ambas juntas denominadas de ambiente, é um dos dados de entrada. A representação interna destes dados repete o formato do arquivo de entrada, esquematizado abaixo:

Ambiente:
- **Representação** do obstáculo
- Braço 1
- Braço 2

Braço:
- tipo da junta (Rotacional ou Prismática)
- comprimento
- ângulo
- **Representação** do Braço

Representação:
- número de polígonos
- **Polígono 1**
- **Polígono 2**
- ⋮
- **Polígono** n

Polígono:
- número de vértices (= lados)
- coordenadas do vértice 1
- ⋮
- coordenadas do vértice m

Além destes dados são necessários os incrementos de cada variável de

junta e o nome do arquivo de saída. Estes incrementos não podem ser muito grandes pois, neste caso, há o perigo de se perder um obstáculo pequeno entre os incrementos. Por outro lado um incremento muito pequeno implica em um mapa de configuração muito grande, ocupando muita memória, e causando também um aumento inútil do tempo de cálculo. O módulo também permite a mudança dos limites das variáveis das juntas, para permitir que se examine apenas uma região do espaço de trabalho do manipulador.

Internamente o mapa de configuração é uma matriz compactada de 'bits'[3] que indicam se há colisão na configuração correspondente (bit=1). O número da coluna corresponde à variável da primeira junta e o da linha à variável da segunda. Portanto o número de colunas é obtido por $n_c = \nabla\theta_1/\Delta\theta_1$, onde $\nabla\theta_1$ é a diferença entre os limites da variável da primeira junta, e $\Delta\theta_1$ o incremento escolhido para esta. O mesmo é válido para o número de linhas n_l em relação à variável da segunda junta. O tamanho da matriz é dado em bytes por $n_c \times \text{int}((n_l + 7)/8)$, devido à compactação e ao alinhamento das linhas na memória (cada linha começa um novo byte).

Inicialmente todos os bits têm valor nulo, equivalendo a nenhuma colisão, e em seguida altera-se o valor dos bits correspondentes às configurações de colisão para um, através seguinte processo:

[3]Unidade básica de armazenamento em computadores, pode estar desligado, valor zero, ou ligado, valor um.

```
B1 = primeiro braço
B2 = segundo braço
R0 = REPRESENTAÇÃO(obstáculo)
V1I = limite inferior da variável de B1
V1S = limite superior da variável de B1
Δ1 = incremento da variável de B1
NC = (V1S − V1I)/Δ1
V2I = limite inferior de B2
V2S = limite superior de B2
Δ2 = incremento de B2
NL = (V2S − V2I)/Δ2
V1 = V1I
para I = 1 ··· NC
        MOVER(B1,V1)
        R1 = REPRESENTAÇÃO(B1)
        se COLISÃO(R0,R1)
           para J = 1 ··· NL
                 marcar linha I coluna J
        caso contrário
           V2 = V2I
           para J = 1 ··· NL
                 mover(B2,V2)
                 R2 = REPRESENTAÇÃO(B2)
                 se COLISÃO(R0,R2)
                    marcar linha I coluna J
                 V2 = V2 + Δ2
        V1 = V1 + Δ1
salvar mapa
```

onde a função REPRESENTAÇÃO retorna a representação absoluta[4] do seu argumento, isto é, se o argumento for um braço, a representação será transformada para a posição ocupada por este em relação ao sistema referencial fixo à base. A função MOVER altera o braço indicado pelo primeiro argumento para a posição definida quando a variável da respectiva junta assumir o

[4] As coordenadas da representação absoluta são referentes ao sistema referencial global.

valor do segundo argumento.

A função COLISÃO, que testa a ocorrência de colisão, é esquematizada como:

$R1$ = primeira representação
$R2$ = segunda representação
$N1$ = número de polígonos de R1
$N2$ = número de polígonos de R2
para $I = 1 \cdots N1$
    $P1$ = polígono I de $R1$
    para $J = 1 \cdots N2$
        $P2$ = polígono J de $R2$
        se INTERSEÇÃO($P1$,$P2$)
            retornar(*verdadeiro*)

retornar(*falso*)

com a função INTERSEÇÃO testando a existência de uma interseção entre dois polígonos, dada por:

$P1$ = primeiro polígono
$P2$ = segundo polígono
$N1$ = número de pontos de $P1$
$N2$ = número de pontos de $P2$
para $I = 1 \cdots N1$
    $A1$ = aresta I de $P1$
    se CONTIDO($A1$,$P2$)
        retornar(*verdadeiro*)
para $I = 1 \cdots N2$
    $A2$ = aresta I de $P2$
    se CONTIDO($A2$,$P1$)
        retornar(*verdadeiro*)

```
se (N1 > 1 e N2 > 1)
   A2 = última aresta de P2
   para I = 1 ··· N2
         A1 = última aresta de P1
         para J = 1 ··· N1
               A3 = aresta J de P1
               A4 = aresta I de P2
               se LINHA(A1,A3,A2,A4)
                  retornar(verdadeiro)
               A1 = A3
         A2 = A4
retornar(falso)
```

onde a função CONTIDO testa se um ponto (primeiro argumento) está contido num polígono (segundo argumento), e LINHA testa se há interseção entre os dois segmentos de retas definidos pelos quatro argumentos, como foi explicado na seção 2.1.

3.3 Procura

O segundo módulo do algoritmo proposto, faz a procura do caminho no *mapa de configuração* gerado pelo primeiro módulo. Além do mapa de configuração, os dados necessários para este módulo são a configuração inicial e o destino. O caminho gerado será armazenado num arquivo para servir de entrada para o último módulo. Foi adotado o algoritmo A*, descrito em [Fu, cap. 10], para achar o caminho ótimo.

O mapa de configuração, que é o principal dado de entrada deste módulo, é fornecido em um arquivo codificado como texto, podendo portanto ser editado com um editor comum com o intuito de facilitar os testes deste módulo. Basicamente o arquivo contém as dimensões do mapa e uma cópia deste com cada letra do arquivo correspondendo a uma posição no mapa, sendo que um espaço vazio significa a inexistência de colisão, e qualquer letra a ocorrência desta.

O algoritmo A* é amplamente usado nos sistemas de *inteligência artificial* para a procura em grafos por ser admissível, como provado em [Nilss]. O termo admissível significa que o algoritmo de procura irá achar o melhor caminho entre dois nós, se existir algum caminho entre estes. Um grafo é um conjunto de nós unidos através de arcos direcionados representando um determinado sistema, como exemplificado na figura 3.2. Os nós equivalem aos diferentes estados do sistema, enquanto os arcos indicam as possíveis transições entre estes estados e têm associados a sí o custo desta transição. O problema a ser resolvido neste sistema consiste em achar uma seqüência de arcos que levem do nó inicial a um nó que satisfaça a condição procurada, com o menor custo total que é a soma dos custos de cada arco percorrido.

Os estados do manipulador são representados por sua configuração, ou mais especificamente, por um ponto no mapa de configuração. Os arcos correspondem ao movimento do manipulador de uma configuração para outra, e só podem existir se não ocorrer colisão durante esta transição, ou seja, se a linha representando este movimento no mapa de configuração não passar por uma região de colisão.

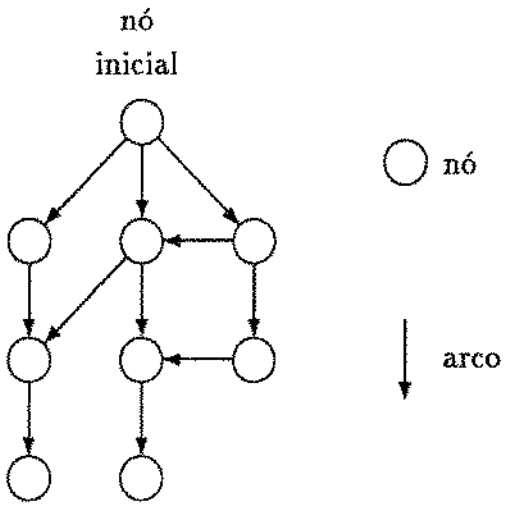


Figura 3.2: Exemplo de grafo de procura.

O algoritmo A* usa uma função heurística para determinar qual o nó a ser analisado primeiro. Esta função, para um determinado nó, é composta pela soma do custo para se atingir este nó, com a previsão do custo para ir deste ao alvo, sendo portanto uma estimativa do custo total para ir do início ao final passando pelo nó em questão. Esta previsão não pode ser maior do que o custo real para garantir a admissibilidade do algoritmo, mas deve ser bem escolhida pois a procura será mais extensa quanto menor esta for, e, por outro lado, quanto mais a previsão se aproximar do custo real, mais rápido se achará o caminho ótimo.

Um caminho encontrado é representado no mapa por uma linha ligando os pontos referentes às configurações inicial e final, sem passar por uma região marcada como colisão. Esta linha pode ser uma única reta ligando os dois pontos, no caso de não haver obstáculo entre estes, ou então será composta por várias retas contornando os obstáculos. Como vemos na figura 3.3 os vértices destas retas sempre estarão nos 'cantos' das regiões de colisão, pois não há razão para se 'quebrar' uma reta no meio do espaço,

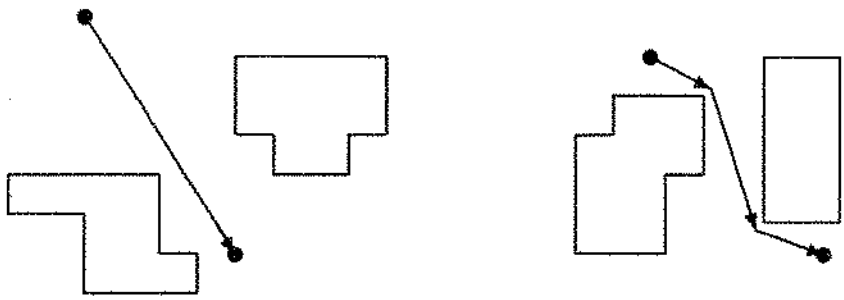


Figura 3.3: Exemplo de caminhos no mapa de configuração.

a não ser para evitar um obstáculo. Este fato é usado para diminuir o grafo de procura, restringindo os nós de interesse àqueles que têm fronteira com uma região de obstáculo como indicado na figura 3.4. Para testar o algoritmo, incluíu-se a possibilidade de analisar todas as configurações discretizadas, ou todas aquelas que tem fronteira com as regiões de colisão, não só as do 'canto'.

Para a procura do caminho mais curto no mapa de configuração, o custo entre dois nós é a distância Euclidiana percorrida entre as posições correspondentes a estes nós no mapa, ou seja, $\sqrt{(\Delta\theta_1)^2 + (\Delta\theta_2)^2}$. A previsão para ir de um nó até o destino também é a distância Euclidiana até o destino, sem levar em conta as possíveis regiões de colisão. Desta maneira a previsão sempre será menor que o custo real, se houver um obstáculo entre o nó e o destino, que deverá ser contornado aumentando o custo, ou no máximo igual no caso de não haver obstáculo, garantindo, em ambos os casos, o sucesso do algoritmo.

3.3.1 Algoritmo A*

Basicamente o procedimento de procura determina, partindo da posição inicial, quais as posições que podem ser alcançadas diretamente, sem passar por uma região de colisão (denominados nós visíveis). Em seguida escolhe a

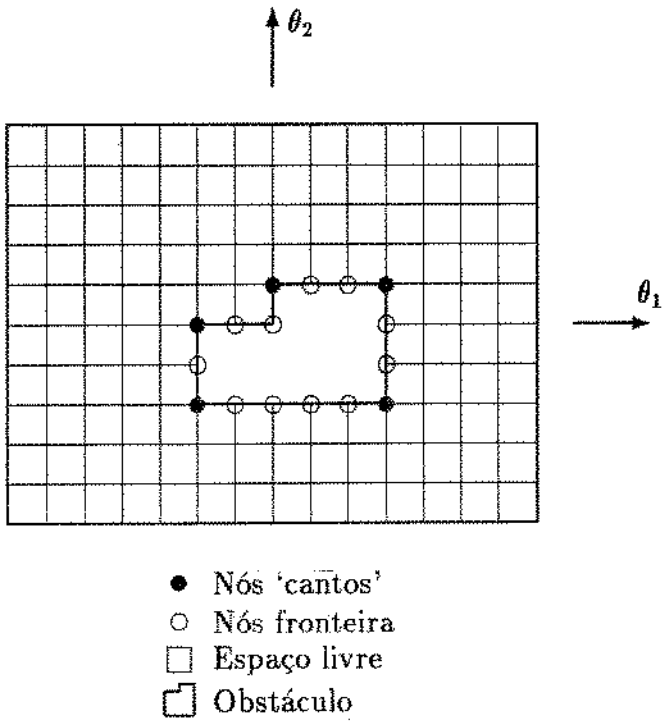


Figura 3.4: Nós de interesse (cantos e fronteiras).

posição mais promissora destas, em relação à previsão do custo para chegar ao destino, e adiciona as novas posições que podem ser alcançadas, partindo desta última, àquelas determinadas no passo inicial. Enquanto não atingir o destino irá continuar escolhendo a melhor posição para determinar quais podem ser alcançadas a partir desta. Supondo os nós compostos pelos seguintes dados:

$$\begin{aligned} x, y &= \text{posição no mapa} \\ A &= \text{antecessor do nó} \\ C &= \text{custo total desde o nó inicial} \\ P &= \text{previsão do custo até o destino} \end{aligned}$$

e, que o antecessor de N é indicado por $N \cdot A$, o algoritmo de procura A* é reproduzido resumidamente, já com as devidas alterações para a procura

de caminho, a seguir:

L = lista dos nós de procura
I = nó inicial
F = nó final
zerar A, C e P dos nós em L
$N = I$
$N \cdot P = \sqrt{(N \cdot x - F \cdot x)^2 + (N \cdot y - F \cdot y)^2}$
O = lista vazia
C = lista vazia
enquanto $(N \neq F)$ e $(N \neq 0)$
    adicionar N a C
    para M = cada um dos nós de L
        se $(M \neq N)$ e $(M \neq N \cdot A)$ e LINHA(N,M)
            $C = N \cdot C + \sqrt{(N \cdot x - M \cdot x)^2 + (N \cdot y - M \cdot y)^2}$
            $P = C + \sqrt{(M \cdot x - F \cdot x)^2 + (M \cdot y - F \cdot y)^2}$
            se $M \cdot A$ nulo
                $M \cdot C = C$
                $M \cdot P = P$
                $M \cdot A = N$
                adicionar M a O em ordem crescente de P
            caso contrário
                se $M \cdot P > P$
                    $M \cdot C = C$
                    $M \cdot P = P$
                    $M \cdot A = N$
    N = nó retirado de O (com menor $N \cdot P$)

Ao final deste procedimento, se N for nulo não existe caminho, caso contrário, este pode ser reconstituido seguindo os antecessores de N, isto é, o caminho será $I \rightarrow \cdots \rightarrow (N \cdot A) \cdot A \rightarrow N \cdot A \rightarrow N$. Os valores de θ_1 e θ_2, correspondentes a estes nós, definem os extremos das retas que compõem o caminho e são armazenados em um arquivo para serem usados na simulação.

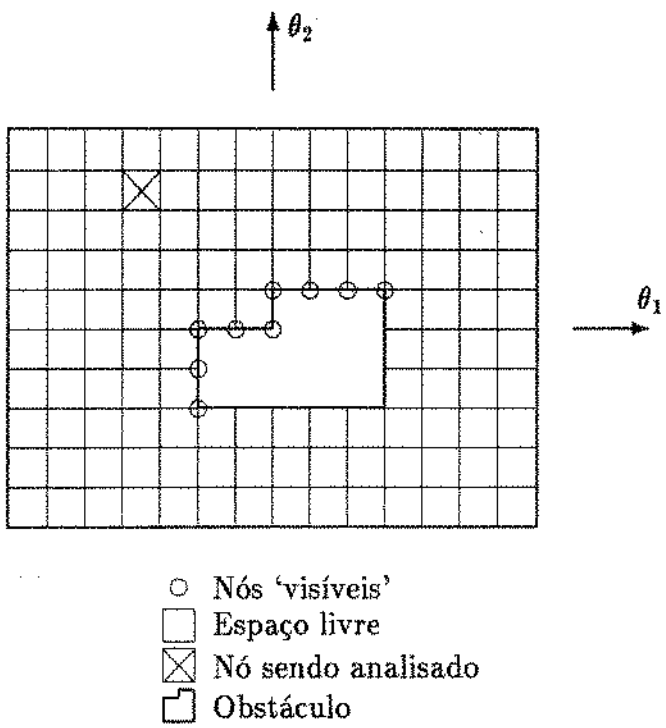


Figura 3.5: Posições 'visíveis'.

A sub-rotina LINHA testa se é possível ligar as posições correspondentes aos dois argumentos através de uma reta sem passar por uma região de colisão, fig. 3.5. Isto é feito através de um procedimento muito usado em computação gráfica para gerar retas, que consiste no cálculo dos pontos (pixels) que devem ser acionados em relação à distância destes à reta, como mostrado a seguir. Iniciando em um dos extremos, cada ponto já acionado terá dois pontos vizinhos ainda não acionados, um na horizontal e o outro na vertical. O procedimento irá escolher aquele ponto, cuja distância à linha for menor. Uma reta pode ser representada por:

$$x \cos \alpha + y \sin \alpha - p = 0$$

onde p é a distância da reta à origem e α é o ângulo entre a *normal* da reta com o eixo x. A distância de um ponto (x_0, y_0) a esta reta será [Brons, pag.219]:

$$d = x_0 \cos\alpha + y_0 \sin\alpha - p \quad ,$$

Se a reta for definida por (x_1, y_1) e (x_2, y_2), podemos fazer a origem do referencial coincidir com o primeiro ponto e também trabalhar somente no primeiro quadrante ($x \geq 0$ e $y \geq 0$), então:

$$\begin{aligned} p &= 0 \\ \cos\alpha &= -\Delta y / L \\ \sin\alpha &= \Delta x / L \\ \Delta x &= |x_2 - x_1| \\ \Delta y &= |y_2 - y_1| \\ L &= \sqrt{\Delta x^2 + \Delta y^2} \\ d &= -x_0 \frac{\Delta y}{L} + y_0 \frac{\Delta x}{L} \quad . \end{aligned}$$

Como queremos apenas comparar a distância entre os pontos podemos multiplicá-la por L para obtermos:

$$d = -x_0 \Delta y + y_0 \Delta x \quad ,$$

então nota-se que para um incremento em x a distância *diminui* de Δy e em y esta *aumenta* de Δx. Para decidir qual o próximo ponto, calcula-se as distâncias para cada um dos eixos, escolhendo aquela direção com a menor distância em módulo. Se a distância for igual para ambos os eixos, o próximo ponto será o resultante de um movimento na diagonal, isto é, um incremento em x e outro em y.

Para o mapa de configuração, cada ponto corresponde a uma configuração do manipulador e ao invés de acionarmos estes pontos só iremos testar se pertencem a uma região de colisão. A sub-rotina LINHA pode ser esquematizada da seguinte maneira:

$(X_1, Y_1), (X_2, Y_2)$ = extremos da reta
$\Delta X = X_2 - X_1$
$\Delta Y = Y_2 - Y_1$
$SX = SY = 1$
$X = Y = DIST = 0$
se $\Delta X < 0$
  $\Delta X = -\Delta X$
  $SX = -1$
se $\Delta Y < 0$
  $\Delta Y = -\Delta Y$
  $SY = -1$
enquanto $(X + X_1 \neq X_2)$ e $(Y + Y_1 \neq Y_2)$
    $DIX = |DIST - DY|$
    $DIY = |DIST + DX|$
    se $DIX = DIY$
      se colisão em $(X + X_1, Y + Y_1 + SY)$ ou
          em $(X + X_1 + SX, Y + Y_1)$
        retornar(*colisão*)
      $DIST = DIST - DY + DX$
      $X = X + SX$
      $Y = Y + SY$
    caso contrário
      se $DIY < DIX$
        $DIST = DIST + DX$
        $Y = Y + SY$
      caso contrário
        $DIST = DIST - DY$
        $X = X + SX$
    se colisão em (X, Y)
      retornar(*colisão*)
retornar(*sem colisão*)

Como pode ser notado, neste método apenas são usados números inteiros e as operações de soma, subtração, valor absoluto e comparação, tornando-o rápido e eficiente por não usar números reais e multiplicação e, principalmente, divisão que requerem muito mais tempo computacional

que as anteriores. Esta também é a principal razão deste procedimento ser usado na computação gráfica.

3.4 Simulação

A idéia básica da simulação é mostrar os movimentos que compõem o caminho determinado pelo algoritmo mencionado nas seções anteriores, e será feita pelo módulo apresentado a seguir. Também existem simuladores desenvolvidos para serem usados como uma ferramenta no estudo das características de alguns manipuladores [Silva], mas este não é o nosso caso.

O principal dado de entrada deste módulo é o caminho determinado pelo módulo apresentado na seção anterior. Necessitamos também do modelo geométrico do manipulador e dos obstáculos, que são usados para a representação gráfica destes no monitor de vídeo. Estes modelos são fornecidos através do mesmo arquivo que serviu de entrada para o módulo responsável em fazer o mapa de configuração, seção 3.2.

A simulação consiste em se mostrar rapidamente uma seqüência de imagens num monitor, sendo que cada imagem corresponde a uma posição do manipulador para um determinado instante do intervalo de tempo sendo analisado. Com isto se reproduz o mesmo efeito usado em desenhos animados.

O caminho é representado por vários pares de ângulos, θ_1 e θ_2, que definem as posições dos braços do manipulador. Para a simulação estabelece-se o número de imagens desejadas, e divide-se o caminho em intervalos com mesmo comprimento, associando cada intervalo a uma imagem. O valor de θ_1 e θ_2 de cada intervalo, é calculado através de interpolação, e a imagem correspondente é gerada, desenhando-se os polígonos referentes ao manipulador na configuração correspondente.

Como se está trabalhando somente no espaço bidimensional, e não há colisão, não é necessário incluir um método de linhas escondidas na simulação. Já no espaço tridimensional uma melhor representação é obtida através do uso de técnicas de linhas ou superfícies escondidas [Suther].

A simulação inclui opções para permitir a análise de cada imagem, correspondente a uma configuração do manipulador, bem como visualizar o

movimento como um todo, e também de se desenhar o caminho do Ponto Central da Ferramenta. No modo normal mostra-se uma imagem e, depois de transcorrido um determinado intervalo de tempo, esta é apagada para que se mostre a próxima. Para permitir uma cópia em impressora e facilitar a visualização de determinado trecho, pode-se mostrar uma imagem sem apagar a anterior, de modo a acumular na tela imagens de diferentes instantes.

Em computadores lentos pode-se gerar todas as imagens e armazená-las em memória, para então mostrá-las rapidamente conseguindo o efeito de movimento na tela. Isto obviamente requer uma capacidade de memória suficiente para o armazenamento das imagens, que podem ser compactadas para este fim.

Capítulo 4

Resultados e Perspectivas

Neste capítulo são apresentados os resultados que foram obtidos aplicando-se o algoritmo proposto a um exemplo que será detalhado mais adiante.

Na seção 4.3 são sugeridas algumas mudanças ao algoritmo para aprimorar as características dos caminhos obtidos, bem como várias extensões ao algoritmo, que deverão ser objetivo de trabalhos futuros, visando principalmente adaptá-lo para o caso tridimensional e para um manipulador com um número maior de graus de liberdade.

Vários foram os testes executados com cada um dos módulos implementados para testar todos (ou quase todos) os detalhes do algoritmo e de sua implementação, como por exemplo: leitura dos dados de entrada, formato dos dados de saída, representação gráfica, compactação do mapa de configuração, ...

4.1 Exemplo: Manipulador

Para apresentar alguns resultados foi escolhido, como exemplo, o manipulador e os obstáculos da figura 4.1, com o propósito de determinar um caminho entre as duas configurações mostradas nesta figura. Nota-se que as

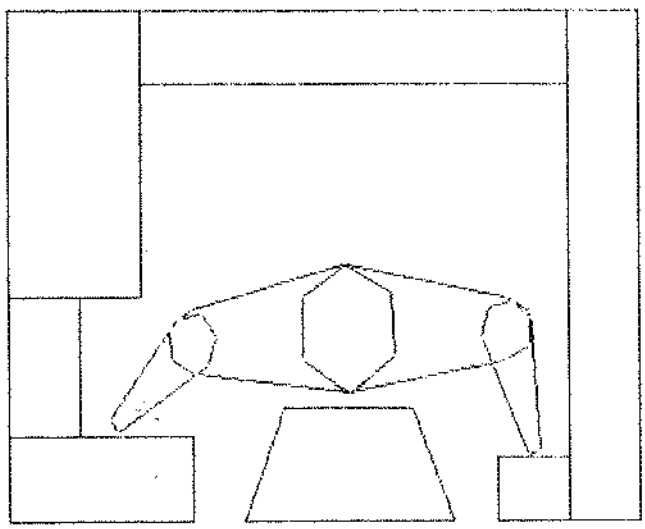


Figura 4.1: Manipulador e obstáculos usados como exemplo.

configurações inicial e final são diferentes, isto é, numa o segundo braço tem rotação no sentido horário e, na outra, no sentido anti-horário em relação ao primeiro braço. O arquivo de entrada, que descreve este problema, e os arquivos intermediários são mostrados no apêndice.

O caminho e o mapa de configuração obtidos pelo algoritmo para este exemplo são mostrados na figura 4.2. As posições correspondentes do manipulador, geradas pelo módulo de simulação, são apresentadas na figura 4.3.

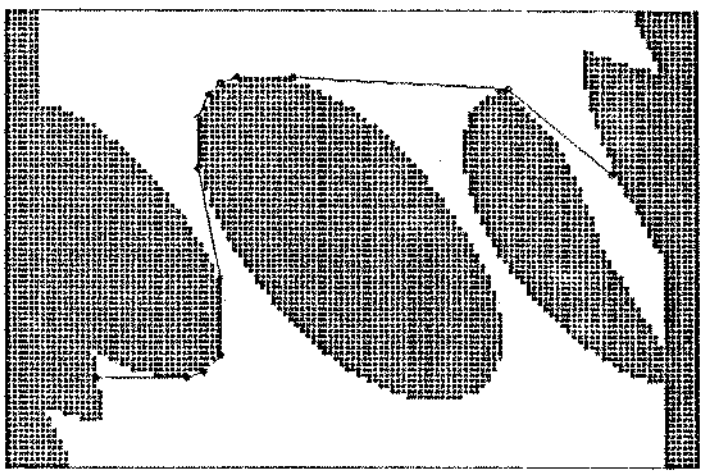


Figura 4.2: Caminho no mapa de configuração.

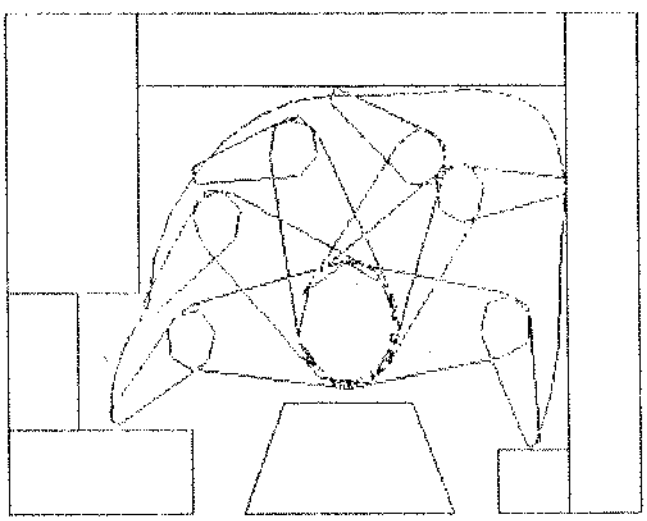


Figura 4.3: Posições do manipulador obtidas pelo simulador.

4.2 Exemplo: Navegação

O algoritmo também pode ser usado para o problema de navegação de um robô ou 'AGV', isto é, para a determinação automática do caminho de um veículo na presença de diversos obstáculos, por exemplo de uma fábrica. Para este caso o mapa de configuração é uma cópia do espaço real, já que as coordenadas destes coincidem, mas teremos um aumento dos obstáculos no mapa, dependendo das dimensões do objeto que se move. Sem a necessidade de alterar o algoritmo, o problema da navegação pode ser implementado como um manipulador com dois braços com juntas prismáticas, sendo que o primeiro não tem volume, e o segundo é o próprio objeto cujo caminho se deseja determinar. Desta maneira a variável de cada junta representa uma das coordenadas do espaço real, ou seja, o primeiro braço, sem volume, irá mover o objeto na direção x, e o segundo na direção y. Como o algoritmo foi implementado apenas para dois graus de liberdade não é possível considerar a orientação do objeto. Pode-se, entretanto, definir a representação do objeto (segundo braço) de maneira a englobá-lo para qualquer orientação deste, ou, como feito neste exemplo, fixar a orientação do objeto.

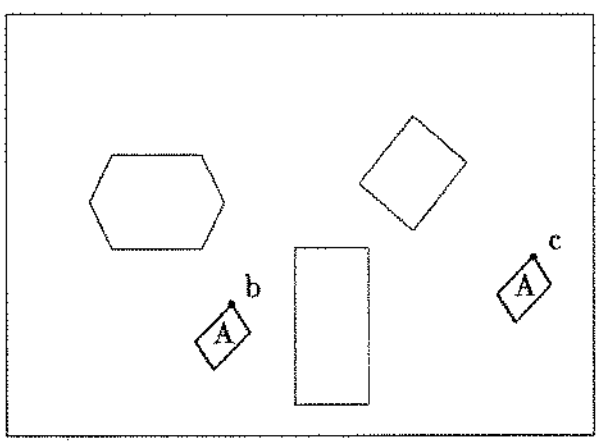


Figura 4.4: Exemplo para demonstrar o problema de navegação.

Para o caso da navegação adotou-se o problema mostrado na figura 4.4, baseado no exemplo de [Brady, cap.6]. O mapa obtido pode ser visto na figura 4.5, juntamente com o caminho determinado pelo algoritmo. A simulação deste pode ser vista na figura 4.6. O respectivos arquivos também são mostrados no apêndice.

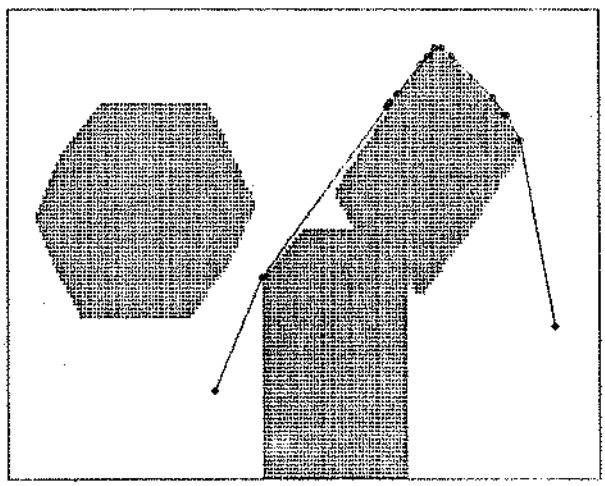


Figura 4.5: Caminho no mapa de configuração.

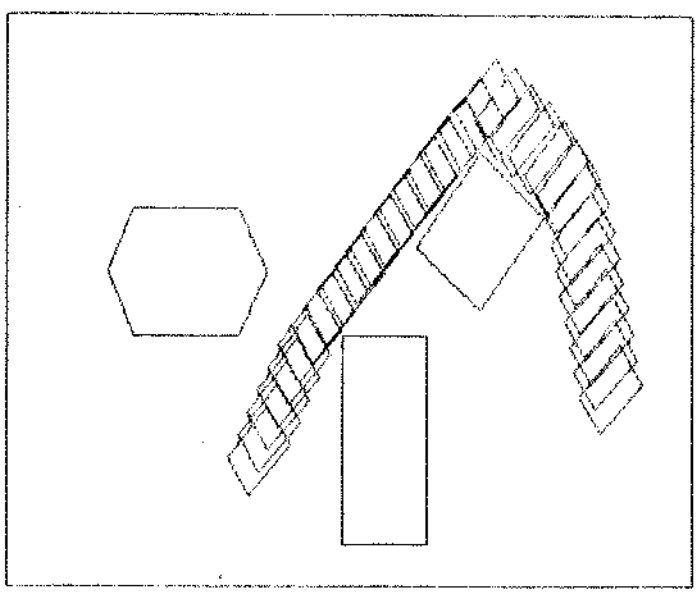


Figura 4.6: Simulação do caso de navegação.

4.3 Perspectivas

Primeiramente são sugeridas algumas alterações ao algoritmo para melhorar o desempenho deste, sem descaracterizá-lo. Mais adiante são apresentadas propostas para aumentar o número de juntas e trabalhar no espaço tridimensional, bem como as implicações disto.

4.3.1 Caixa Envolvente

Uma possibilidade de diminuir o tempo necessário para a determinação do caminho é através da redução do tempo gasto na detecção de colisão. Para cada ponto do mapa de configuração testa-se a ocorrência de colisão entre os polígonos do primeiro braço com todos os do obstáculo e o mesmo para os polígonos do segundo braço.

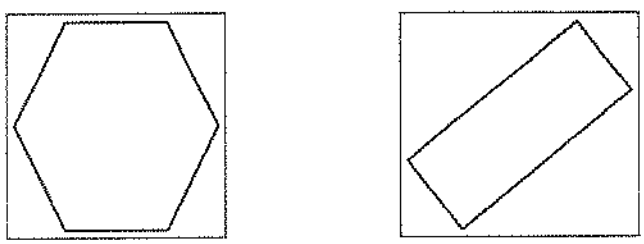


Figura 4.7: Exemplos de "caixa envolvente".

A primeira sugestão para diminuir este procedimento é através do uso de 'caixas envolventes' (*bounding box*), cuja idéia será explicada a seguir. Para cada polígono define-se o menor retângulo possível com os lados paralelos aos eixos referenciais, que contenha integralmente este polígono, como na figura 4.7. O teste de colisão é evitado nos casos em que não há interseção dos retângulos correspondentes aos polígonos sendo testados. Para verificar esta interseção é suficiente comparar as coordenadas destes retângulos. As coordenadas dos retângulos referentes ao obstáculo, são os valores mínimos

e máximos das coordenadas dos vértices dos polígonos que o representa. Para os polígonos referentes aos braços devemos levar em conta a variação da orientação destes, e podemos, neste caso, usar circunferências ao invés de retângulos.

4.3.2 Aceleração da Detecção de Colisão

Outra sugestão para aumentar a eficiência da detecção de colisão é baseada no fato de que qualquer ponto do braço descreve uma circunferência com raio constante e centro na junta. Sob este ponto de vista podemos caracterizar dois tipos de colisão, mostrados na figura 4.8:

1. Colisão de um vértice do manipulador com um lado do obstáculo, e
2. colisão entre um lado do manipulador com um vértice do obstáculo.

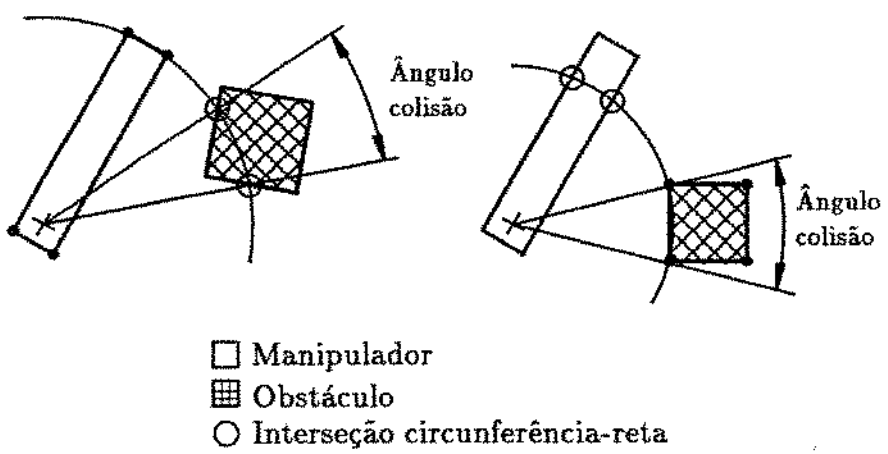


Figura 4.8: Tipos de colisão.

As outras duas possibilidades, ou seja, a colisão de dois vértices e a de dois lados, podem ser classificadas como uma das duas acima. Pode-

mos então determinar quais as posições nas quais haverá colisão entre dois polígonos, calculando a interseção da circunferência gerada pelos vértices de um deles com os lados do outro e vice-versa. Desta maneira a detecção de colisão só será necessária nestas posições ao invés de todas as posições discretizadas para o primeiro braço. E, para o segundo braço, este cálculo só será feito quando não houver colisão do primeiro.

4.3.3 Arredondamento do Caminho

O caminho determinado, como já dito, é representado por vários segmentos de reta no espaço de configuração. Na mudança de um segmento para o seguinte teremos uma variação na inclinação destes o que implica numa descontinuidade do caminho, ou seja, uma alteração instantânea das velocidades das juntas. Isto não é possível fisicamente, portanto é interessante 'arredondar' a passagem de um segmento para o próximo, fazendo a transição ocorrer de modo contínuo, considerando para tanto as características dinâmicas do manipulador [Jacobs]. Para isto ser possível é necessário uma margem de segurança entre os extremos dos segmentos e os obstáculos, para ser usada no arredondamento. Esta distância é obtida através da inclusão de mais nós na fase de procura e da definição de um custo adicional para os extremos localizados a uma distância dos obstáculos, menor que um certo limite. Esta definição será discutida mais adiante na subseção 4.3.5.

4.3.4 Vários Caminhos

Normalmente não se está interessado somente em uma trajetória, e sim num conjunto de trajetórias. Por exemplo: pegar um objeto em 'A', levá-lo para 'B' pegar outro em 'C', deixando este em 'D'. Para isto pode-se incluir a possibilidade de definir vários pontos pelo qual o maniuplador deverá passar, e chamar, automaticamente, o algoritmo de procura para cada segmento. Mas, quando se deseja carregar um objeto, também deve-se incluí-lo nos cálculos de colisão. Devido a isto o programa deve permitir,

além da definição de vários objetos, ao manipulador pegar um destes, ou seja, incorporá-lo como se fosse parte da representação do último braço, aumentando os obstáculos no mapa de configuração que deve ser refeito ou corrigido. Isto também requer a determinação de quais as posições e orientações nas quais é possível pegar o objeto. A fase da simulação também deve ser alterada para incluir esta possibilidade, mostrando os objetos em sua nova posição ou sendo movidos pelo manipulador. Portanto é necessário alterar as coordenadas dos objetos bem como os sistemas referenciais aos quais estas se referem.

4.3.5 Custos

Outros critérios de otimização podem ser usados para escolher entre as várias trajetórias possíveis: menor torque, menos energia, maior velocidade, maior distância aos obstáculos, ... O primeiro passo para implementar um novo critério é alterar a definição do custo usado pelo algoritmo A*, lembrando que a previsão também deverá ser alterada pois esta deve sempre ser menor ou igual ao custo adotado para garantir o sucesso do algoritmo. Outro passo é a mudança dos nós a serem analisados, pois, dependendo do caso, a trajetória desejada não será representada por retas no espaço de configuração. Provavelmente será preciso incluir todos os nós, não apenas os das fronteiras.

Um dos critérios que requerem esta mudança de custos é usado quando se quer a trajetória de menor movimento no espaço real, isto é, deseja-se que o manipulador percorra uma reta no espaço cartesiano. Para isto é necessário, além de fazer o custo ser a distância neste espaço, alterar a determinação dos nós visíveis que serão aqueles que podem ser atingidos por uma reta no espaço cartesiano sem causar uma colisão, pois esta trajetória não é representada por uma reta no espaço de configuração.

Uma alteração interessante, já mencionada em subseção anterior, é a definição de um custo adicional para cada ponto, dependendo inversamente da distância deste aos obstáculos. Com isto o caminho não será o mais

curto, pois o algoritmo irá escolher um caminho um pouco mais longo para aumentar a distância deste aos obstáculos. Obviamente isto requer que todos os nós sejam analisados, não apenas os de fronteira.

Definindo o custo, para o caso de navegação, como uma função apenas da distância do robô aos obstáculos, a determinação de trajetórias corresponde ao método sugerido por Takahashi e Schilling em [Takah]. Este método usa "Diagramas Generalizados de Voronoi", que consistem no conjunto dos pontos equidistantes dos contornos dos obstáculos. No caso de obstáculos poligonais, estes pontos formarão linhas retas ou parabólicas. Após ter determinado este diagrama, o método determina a trajetória permitindo somente movimentos sobre estas linhas, com exceção dos trechos inicial e final. Desta forma a distância do robô aos obstáculos sempre será a maior possível.

Um problema deste método ocorre no caso de ambientes pouco congestionados, quando temos muito espaço vazio, nos quais o método irá determinar desnecessariamente a trajetória com a maior distância dos obstáculos. Este método fornece uma trajetória ótima em relação à distância dos obstáculos, o que é bastante interessante em ambientes congestionados.

4.3.6 Aumento do Número de Graus de Liberdade

O aumento do número de graus de liberdade implica, obviamente, no aumento do número de variáveis do espaço de configuração e, conseqüentemente, da dimensão do mapa de configuração. Pode-se, inicialmente, continuar a trabalhar no espaço bidimensional, isto é, o manipulador e os obstáculos continuariam a ser representados através de polígonos. O procedimento praticamente continuaria o mesmo, sendo a maior diferença o aumento no procedimento de determinação do mapa de configuração e uma maior dificuldade na determinação dos nós 'visíveis'. Isto inclui um aumento do número de chamadas da detecção de colisão bem como do número de nós a serem analisados pelo algoritmo de procura, causando um aumento do tempo necessário para gerar o mapa e achar ocaminho. Além disto te-

mos a dificuldade de visualizar o mapa de configuração tridimensional na tela do computador.

Para o caso da navegação, o aumento do número de graus de liberdade é essencial, pois a localização de um objeto no espaço bidimensional exige três variáveis, duas para definir sua posição e a terceira para definir sua orientação [Brost].

4.3.7 Espaço Tridimensional

A representação tridimensional de manipulador e dos obstáculos implica numa alteração no modelo geométrico e na detecção de colisão. O procedimento para montar o mapa de configuração e o de procura não necessitam de mudança. O modelo geométrico sofre as maiores alterações, pois deve permitir a representação dos volumes ocupados sem tornar inviável o cálculo de colisão. Existem várias maneiras para descrever objetos tridimensionais na memória de um computador.

Uma destas maneiras, conhecida por "B-Rep" (**B**oundary **Rep**resentation), é através da representação das superfícies e fronteiras do objeto sendo modelado, que podem ser planos ou, para casos mais complexos, superfícies matemáticas (Bezier).

Outra maneira é conhecida por Geometria Construtiva de Sólidos, ou CSG (**C**onstructive **S**olid **G**eometry) [Hoffm]. Neste método os objetos são representados através da aplicação de operações de conjuntos a uns poucos volumes básicos, como cubos, esferas e cones, que podem ser rodados e ter seu tamanho alterado. Estas operações são do tipo união, interseção e subtração, e permitem a definição de novos volumes. Este método é muito usado pois permite uma representação fácil de componentes mecânicos, por causa da semelhança com a confecção destes. A operação de fazer um furo, por exemplo, equivale à subtração de um cilindro.

Uma terceira maneira é através do uso de "Octree's", que consiste basicamente na subdivisão recursiva do espaço [Glass]. Primeiramente o espaço é dividido em oito partes iguais (divisão por dois em cada eixo), as par-

tes totalmente livres ou totalmente ocupadas são marcadas como vazias ou cheias, respectivamente, e as demais partes serão subdivididas em oito partes iguais. Novamente se marca as partes totalmente ocupadas ou livres, subdividindo as demais. Este procedimento é repetido até o tamanho das novas partes ser menor que um determinado limite. Desta maneira forma-se uma árvore onde cada nó tem oito ramos.

Em [Red87] é sugerido outro método para a determinação de caminhos no espaço tridimensional, que também usa mapas de configuração como descrito a seguir. Define-se um plano no espaço de configuração e determina-se quais as posições, neste plano, nas quais não ocorre colisão, para em seguida, se achar o caminho neste plano. O problema deste procedimento é que o caminho ótimo pode não estar em um único plano, mas sua vantagem é a grande velocidade do algoritmo para o caso tridimensinal, sendo o tempo gasto semelhante ao de procedimentos para o caso bidimensional.

4.4 Conclusão

Neste trabalho apresenta-se a determinação automática de caminhos para um manipulador com duas juntas rotacionais, bem como a utilização deste para o caso da navegação. Os obstáculos são conhecidos e fixos, isto é, o seu formato e posição são conhecidos a priori., e estes não variam com o tempo. A representação do manipulador e dos obstáculos é feita através de polígonos convexos, e será usada na detecção de colisão e na simulação.

A detecção de colisão é simplificada através do uso do "mapa de configuração" que consiste na definição das configurações seguras para o manipulador, que são aquelas nas quais não há ocorrência de colisão. A determinação do caminho consiste então em achar a linha de menor comprimento no espaço de configuração, ligando os pontos referentes às posições inicial e final sem abandonar as regiões seguras. Esta linha é determinada pelo algoritmo de procura em grafos A^*, que é muito usado em sistemas de *Inteligência Artificial*, por sempre achar a solução ótima (se existir alguma) segundo o custo ou critério definido para o problema. Neste trabalho adotou-se como critério , o caminho com o menor movimento das juntas.

São sugeridas, no final, várias alterações ao algoritmo, para que este possa ser usado para outros casos como para a determinação de vários caminhos, espaço tridimensional, outros critérios de escolha de caminhos, ...

O algoritmo foi implementado em um computador compatível ao IBM AT, na linguagem C seguindo alguns princípios da programação orientada por objetos. Foi implementado também um simulador para possibilitar a visualização de alguns caminhos e permitir a verificação destes.

Bibliografia

[Ander] R. L. ANDERSSON, "Understanding and Applying a Robot Ping-Pong Player's Expert Controller", *IEEE International Conference on Robotics and Automation*, 1989, 1284–1289.

[Asada] H. ASADA e B. H. YANG "Skill Acquisition from Human Experts through Processing of Teaching Data", *IEEE International Conference on Robotics and Automation*, 1989, 1302–1307.

[Brady] M. BRADY,
J. M. HOLLERBACH, T. L. JOHNSON, T. LOZANO-PÉREZ, e M. T. MASON, *Robot Motion, Planning and Control*, The MIT Press, Massachusetts 1984.

[Brons] I. N. BRONSTEIN e K. A. SEMENDJAJEW, *Taschenbuch der Mathematik*, Verlag Harri Deutsch 22. Auflage, 1985.

[Brost] R. C. BROST, "Computing Metric and Topological Properties of Configuration-Space Obstacles", *IEEE International Conference on Robotics and Automation*, 1989, 170–176.

[Caze] A. CAZENAVE, J. M. ROSÁRIO, e J. P. BLANCHARD, "Etude et Developpement d'un Logiciel de Programmation Hors-Ligne – Creation Hors-Ligne de Trajectoire pour le Robot ISIS-2", Hispano-Suiza, projet nº 4/SPE/1989, France.

[Cozman] F. G. COZMAN, J. C. ADAMOWSKI, M. G. SIMÕES, M. R. P. BARRETTO, P. E. MIYAGI, e L. A. MOSCATO,

"O Projeto de Veículos Autônomos de Transporte", *Anais 1º Simpósio de Automação Integrada*, CEFET-PR, Paraná 1990, 119–123.

[Craig] J. J. CRAIG, *Introduction to Robotics, Mechanics & Control*, Addison-Wesley Publishing Company, Massachusetts 1986

[Denav] J. DENAVIT e R. S. HARTENBERG, "A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices", *Journal of Applied Mechanics*, junho 1955, 215–221.

[Dillm] R. DILLMANN, "Application of AI Methods in Robotics", *Conferência no 1º Simpósio de Automação Integrada*, CEFET-PR, Paraná 1990.

[Fu] K. S. FU, R. C. GONZALEZ, e C. S. G. LEE, *ROBOTICS: Control, Sensing, Vision, and Intelligence*, McGraw-Hill Book Company 1987

[Glass] A. S. GLASSNER, "Space Subdivision for Fast Ray Tracing", *IEEE Computer Graphics and Applications*, outubro 1984, 15–22.

[Hoffm] R. HOFFMAN, "Automated Assembly in a CSG Domain", *IEEE International Conference on Robotics and Automation*, 1989, 210–215.

[Jacobs] P. JACOBS e J. CANNY, "Planning Smooth Paths for Mobile Robots", *IEEE International Conference on Robotics and Automation*, 1989, 2–7.

[Madrid] M. K. MADRID, "Controle de Tráfego de Elevadores com Inteligência Artificial Aplicando Lógica Fuzzy", *Anais 1º Simpósio de Automação Integrada*, CEFET-PR, Paraná 1990, 296–301.

[N-Nagy] F. N-NAGY e A. SIEGLER, *Engineering Foundations of Robotics*, Prentice-Hall International, Inglaterra 1986.

[Nilss] N. J. NILSSON, *Principles of Artificial Intelligence*, Springer-Verlag, California 1982.

[Red85] W. E. RED e H. V. TRUONG-CAO, "Configuration Maps for Robot Path Planning in Two Dimensions", *Transactions of the ASME*, Vol. 107, dezembro 1985, 292–298. (O nome do segundo autor provavelmente foi impresso errado, o correto deve ser 'Troung' como no seguinte artigo).

[Red87] W. E. RED, H. V. TROUNG-CAO, e K. H. KIM, "Robot Path Planning in Three-Dimensions Using the Direct Subspace", *Transactions of the ASME*, Vol 109, N. 9, setembro 1987, 238–244.

[Shill] Z. SHILLER, "Interactive Time Optimal Robot Motion Planning and Work-Cell Layout Design", *IEEE International Conference on Robotics and Automation*, 1989, 964–969.

[Silva] J. C. SILVA e N. BACK, "Desenvolvimento de um Simulador para Análise e Modelamento Dinâmico de Robôs Industriais", *Anais 1º Simpósio de Automação Integrada*, CEFET-PR, Paraná 1990, 272–276.

[Small] *Smalltalk/V, Tutorial and Programming Handbook*, Digitalk Inc., California 1986.

[Strous] B. STROUSTRUP, *The C++ Programming Language*, 1986.

[Suther] I. E. SUTHERLAND, R. F. SPROULL, e R. A. SCHUMACKER, "A Characterization of Ten Hidden-Surface Algorithms", *Computing Surveys*, Vol. 6, No. 1, março 1974, 387–441.

[Takah] O. TAKAHASHI e R. J. SCHILLING, "Motion Planning in a Plane Using Generalized Voronoi Diagrams", *IEEE Transactions on Robotics and Automation*, Vol.5 No.2, abril 1989, 143–150.

[Taze] J. M. TAZELAAR, "Object-Oriented Programming"; D. THOMAS, "What's in an Object?"; P. WEGNER, "Learning the Language"; BYTE, março 1989, 228–253.

Apêndice A

Arquivos: Manipulador

Neste apêndice são mostrados os arquivo de entrada e intermediários usados para o manipulador e obstáculos descritos na seção 4.1. Nestes arquivos o símbolo de porcentagem (%) inicia um comentário que termina no fim da linha que o contém , e as linhas em branco também são ignoradas.

O arquivo de entrada reflete a representação interna dos dados que é mostrada na seção 3.2. O arquivo gerado pelo primeiro módulo contém em cada linha:

1. nome do arquivo de entrada,
2. valores mínimos das variávies das juntas,
3. incremento das variáveis das juntas,
4. número de linhas e de colunas do mapa de configuração, e
5. a representação do mapa, onde um espaço vazio representa uma posição sem colisão, e 'x' uma de colisão.

O arquivo de saída do segundo módulo descreve finalmente o caminho

determinado por este, onde cada linha contém:

1. nome do arquivo de entrada do primeiro módulo,
2. valores mínimos das variávies das juntas,
3. incremento das variáveis das juntas, e
4. coordenadas (em incrementos) dos extremos das retas, que descrevem o caminho encontrado.

A.1 Arquivo de Entrada

```
% Manipulador com duas juntas rotacionais.
%
% REPRESENTACAO dos obstaculos
% numero de poligonos na representacao (obstaculo) =

7

    % POLIGONO 1 - bloco direita
    % numero de vertices do poligono (primeiro) =

    4
       125   -90    % coordenadas do primeiro vertice
       165   -90    % segundo vertice
       165    150
       125    150

    % POLIGONO 2 - bloco superior
    % numero de vertices =

    4
      -120    150   % coordenadas do primeiro vertice (segundo poligono)
       125    150
       125    115
      -120    115
```

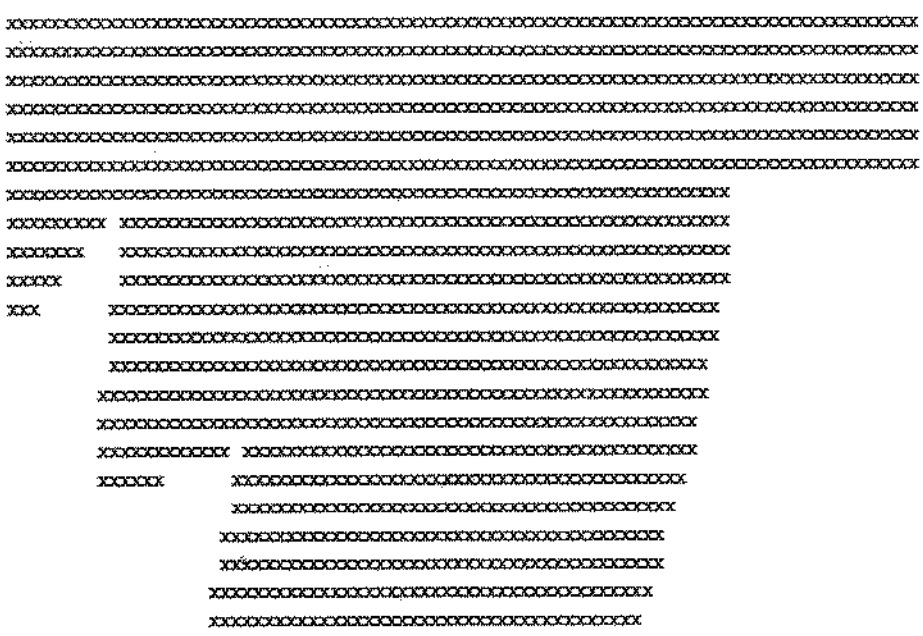
```
% POLIGONO 3 - bloco superior esquerda
% numero de vertices =

4
  -195    150
  -120    150
  -120    15
  -195    15

% POLIGONO 4 - bloco inferior esquerda
4
  -195   -90
  -90    -90
  -90    -50
  -195   -50

% POLIGONO 5 - bloco centro esquerda
4
  -195   -50
  -155   -50
  -155    15
  -195    15

% POLIGONO 6 - bloco pequeno direita
4
   85    -90
   125   -90
   125   -60
   85    -60
% POLIGONO 7 - base
4
  -60    -90
   60    -90
   37    -37
  -37    -37

% numero de juntas =

2
```

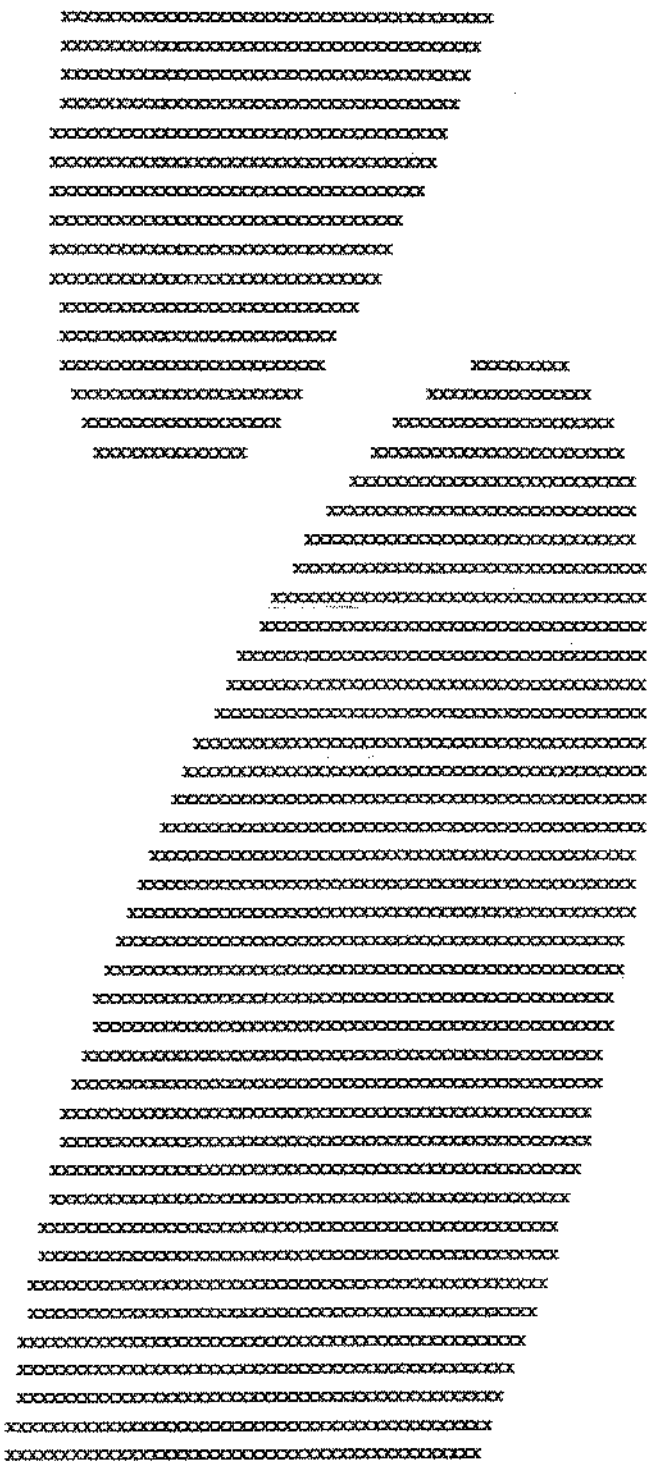
```
% BRACO 1
% tipo =
Rotacional          % so necessario a primeira letra

% comprimento =
90

% angulo =
-30 210             % limites do angulo

% REPRESENTACAO do primeiro braco
% numero de poligonos =

1

  % POLIGONO 1
  % numero de vertices =

  8
     13     -7.5   % coordenadas do primeiro vertice
     0      -15
    -90     -30
    -116    -15
    -116     15
    -90      30
     0       15
     13      7.5

% BRACO 2
% tipo =
Rotacional

% comprimento =
60

% angulo =
-120 120

% REPRESENTACAO do segundo braco
% numero de poligonos =
```

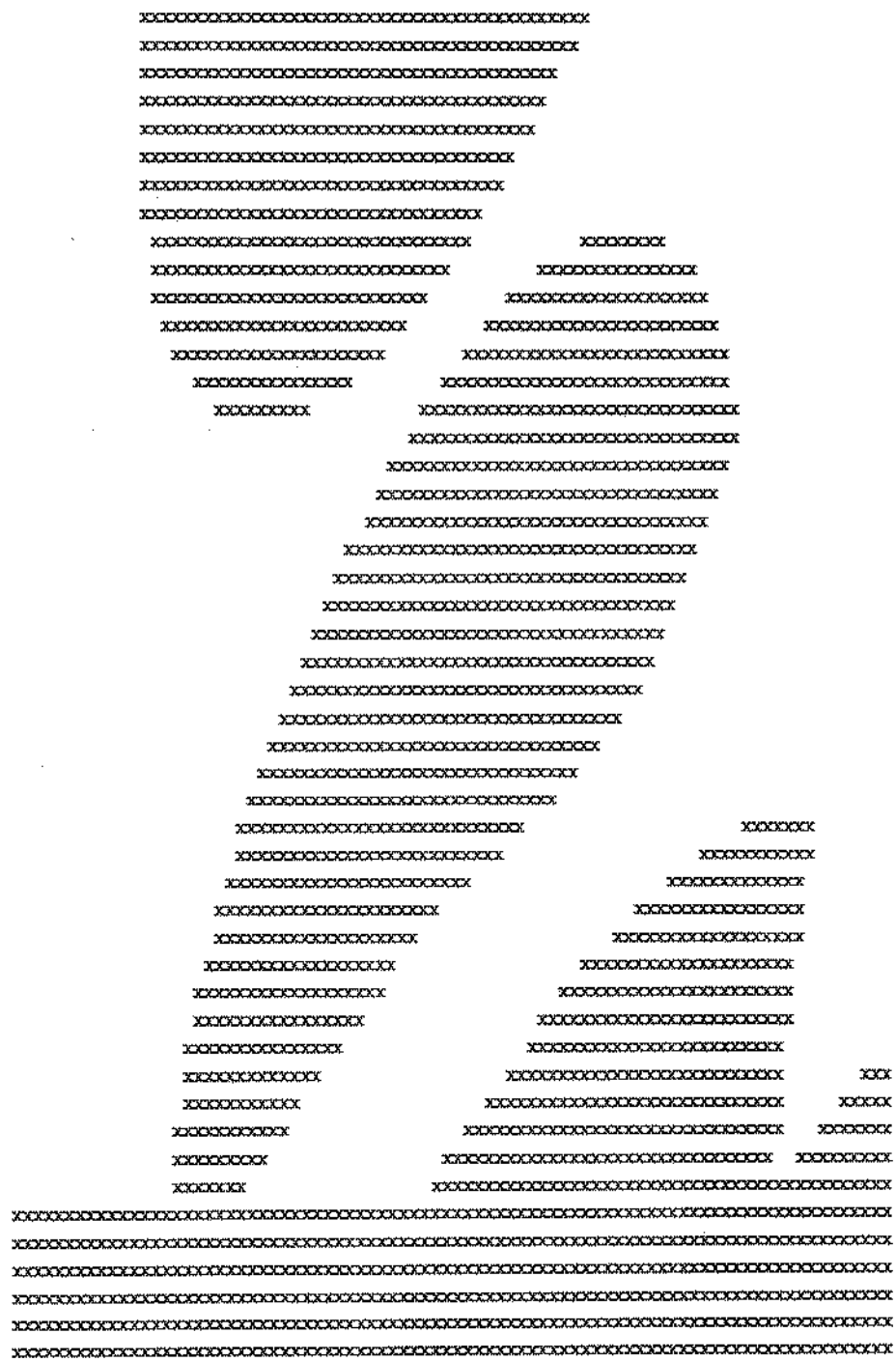
```
1

   % POLIGONO 1
   % numero de vertices =
   6
      0      -4
     -60     -14
     -72     -7
     -72      7
     -60      14
      0       4

% Fim do arquivo.
```

A.2 Mapa de Configuração

```
ex\sai.ROB  % nome do arquivo de entrada
-30 -120    % valores iniciais das variaveis das juntas
2 3         % incremento de cada variavel
122    82   % numero de linhas e de colunas do mapa reproduzido a seguir
```







% Fim do arquivo

A.3 Caminho

```
ex\sai.ROB  % nome do arquivo de entrada do primeiro modulo
-30 -120    % valores iniciais das variaveis
2 3         % incremento das variaveis das juntas
16 16       % primeiro ponto do caminho = configuracao inicial
32 16       % segundo ponto
35 17
38 20
38 34
34 54
34 63
36 67
38 69
41 70
51 70
89 68
107 53
108 53      % ultimo ponto do caminho = configuracao final
% Fim do arquivo
```

Apêndice B

Arquivos: Navegação

Este apêndice contém o arquivos para o exemplo de navegação mostrado na seção 4.2. A estrutura destes arquivos é a mesma dos apresentados no apêndice anterior. O mapa de configuração não será mostrado neste apêndice, mas pode ser visto na figura 4.5. São mostradas apenas as primeiras linhas do arquivo contendo o mapa para permitir uma comparação com o caso anterior.

B.1 Arquivo de Entrada

```
% Navegacao
%
% REPRESENTACAO dos obstaculos
% numero de poligonos na representacao (obstaculo) =

7    % quatro laterais + tres obstaculos

  % POLIGONO 1 - lateral esquerda
  % numero de vertices =

  2    % reta
    -210  -120      % coordenadas do primeiro vertice
    180   -120      % coordenadas do segundo vertice
```

```
% POLIGONO 2 - lateral direita
% numero de vertices =

2  % reta
  -210  150
  180   150

% POLIGONO 3 - lateral inferior
2  % reta
  -210  150
  -210  -120

% 4 - lateral superior
2  % reta
  180   150
  180  -120

% 5 - primeiro obstaculo
4  % quadrilatero
  -18   -100
   30   -100
   30   0
  -18   0

% 6 - segundo obstaculo
6  % hexagono
  -138  0
  -78   0
  -61   30
  -78   60
  -138  60
  -153  30

% POLIGONO 7 - obstaculo
4  % quadrilatero
  24    42
  60    12
  96    56
  60    86
```

```
% numero de juntas =

2

  % BRACO 1
  % tipo da junta =
  Prismatica        % so primeira letra

  % comprimento =
  -180  165         % limites do comprimento

  % angulo
  0

  % REPRESENTACAO
  0         % sem representacao, somente como coordenada

  % BRACO 2
  Prismatico
  -75  145          % comprimento
  90                % angulo

  % REPRESENTACAO do segundo braco = objeto se movendo
  1    % poligono
    4  % vertices
      1    1
      -24  24
      -42  12
      -18  -12
% Fim do arquivo
```

B.2 Mapa

```
ex\nav.ROB  % nome do arquivo de entrada
-180 -75    % valores iniciais das variaveis
2 2         % incrementos
174    112  % numero de linhas e de colunas do mapa
```

```
%
% Mapa de configuracao nao mostrado
%
% Fim do arquivo
```

B.3 Caminho

```
ex\nav.ROB  % nome do arquivo de entrada
-180 -75    % valores iniciais das variaveis
2 2         % incrementos das variaveis
61 21       % primeiro ponto do caminho
75 48       % segundo ponto
112 89
113 90
115 92
122 99
124 101
126 103
128 103
131 101
143 91
147 87
151 81
161 36      % ultimo ponto do caminho = destino
% Fim do arquivo
```