6. PLANEJAMENTO DE TAREFAS

Neste capítulo abordamos o problema de planejamento de tarefas de robôs manipuladores. Planejamento de Tarefas está mais relacionado aos objetivos gerais de uma tarefa de manipulação do que aos meios de alcançar estes objetivos. A Figura 6.1 mostra a relação entre um planejador de tarefas e outros módulos do sistema de controle do robô.

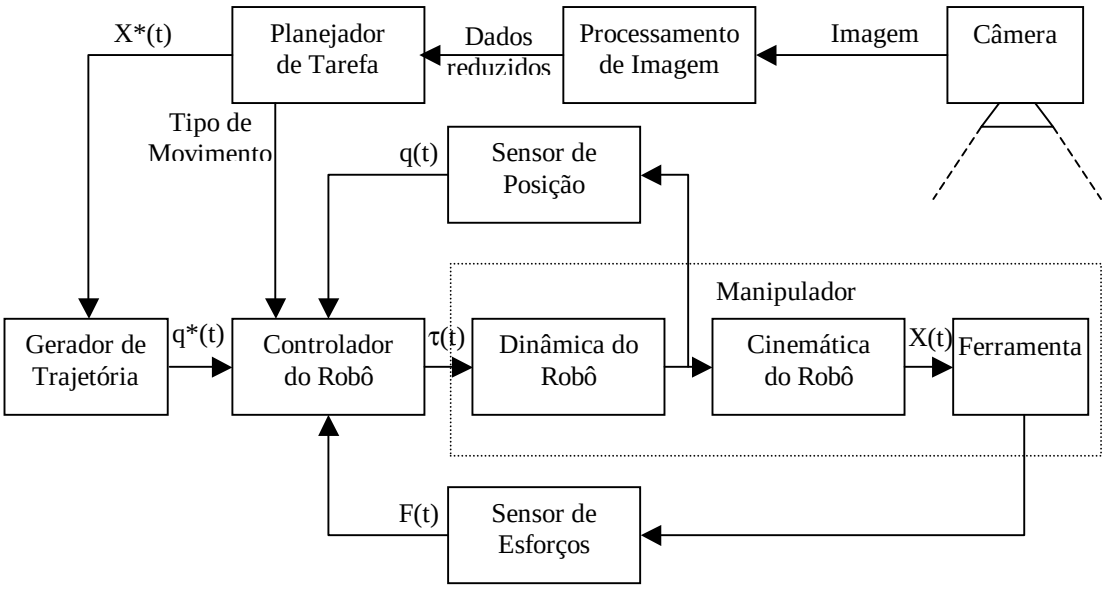

Figura 6.1. Planejador de Tarefas.

6.1. Espaço de Configuração:

Uma técnica muito utilizada no planejamento de tarefas é o mapeamento do problema originalmente especificado em espaço de trabalho (W - subconjunto de $\mathbf{R}^2$ ou $\mathbf{R}^3$) para espaço de configuração (C). Neste espaço, o robô (A), que no espaço de trabalho original é um objeto extenso, é transformado em um ponto. Por sua vez, os obstáculos presentes no espaço de trabalho (B) são mapeados em uma nova representação em espaço de configuração (C-obstáculos – CB). Assim, o problema de movimentar um robô de dimensões finitas em um espaço de trabalho povoado de obstáculos extensos é transformado no problema mais simples de movimentar um ponto em um espaço de configuração povoado de C-obstáculos.

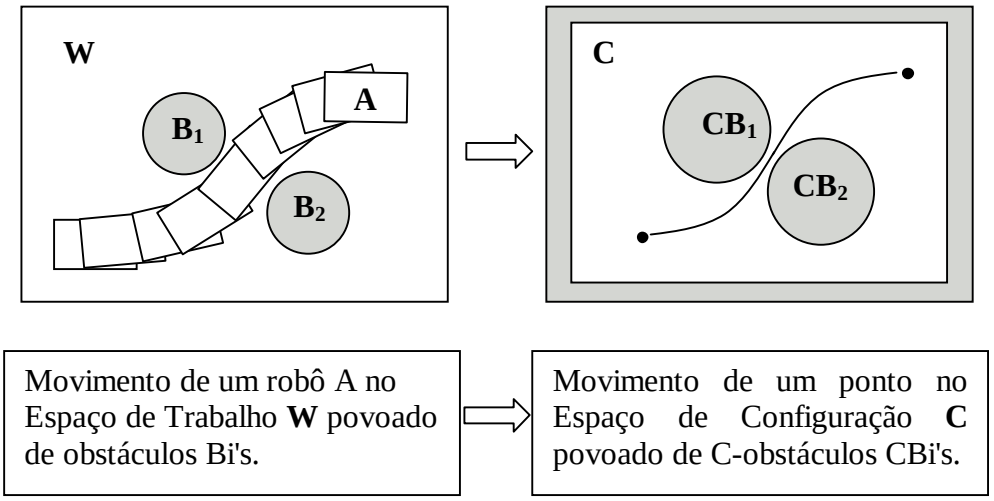

Figura 6.2. Mapeamento para espaço de configuração.

Dados {A}, Referencial fixo em um robô A, e {W}, Referencial fixo no espaço de trabalho W, define-se a configuração q de A como uma especificação da localização (posição e orientação = pose) de {A} em relação a {W}. Por exemplo, para um robô manipulador, o vetor de variáveis de junta q pode ser uma representação da sua configuração. O espaço N-dimensional C de todas as possíveis configurações de A é denominado "Espaço de Configuração" $\Rightarrow q \in C \; \forall \; q$. A(q) é o subconjunto de W ocupado por A na configuração q.

Resumindo:

- **Espaço de Trabalho W:** é o espaço físico no qual o robô se movimenta.
- **Robô A:** conjunto de corpos rígidos interligados que pode movimentar-se dentro do Espaço de Trabalho.
- **Obstáculo no Espaço de Trabalho B:** região de **W** na qual não se pode posicionar qualquer ponto do Robô.
- **Configuração q:** Especificação da Posição e Orientação do robô.
- **Espaço de Configuração C:** é o conjunto de todas as possíveis configurações do robô.
- **Espaço de Configuração Livre C_L:** é o conjunto de todas as possíveis configurações em que o robô não colide com os obstáculos Bi's.
- **Obstáculo em Espaço de Configuração CB (C-Obstáculo):** é o conjunto de todas as configurações em que o robô se superpõe parcial ou totalmente ao obstáculo. Um obstáculo **B** no espaço de trabalho **W** pode ser representado de forma equivalente por um C-obstáculo **CB** no espaço de configuração **C**.

6.2. Obstáculos em Espaço de Configuração:

Dado um objeto A(q) modificando a sua configuração q ao movimentar-se em um espaço de trabalho W povoado de obstáculos B_i's, um obstáculo B_i é mapeado em um C-obstáculo CB_i no espaço de configuração definido como:

$$CB_i = \{q \in C \,/\, A(q) \cap B_i \neq \varnothing\}$$

Por exemplo, se um objeto circular se movimenta em um espaço planar povoado de obstáculos, os obstáculos originais devem ser aumentados do tamanho do raio.

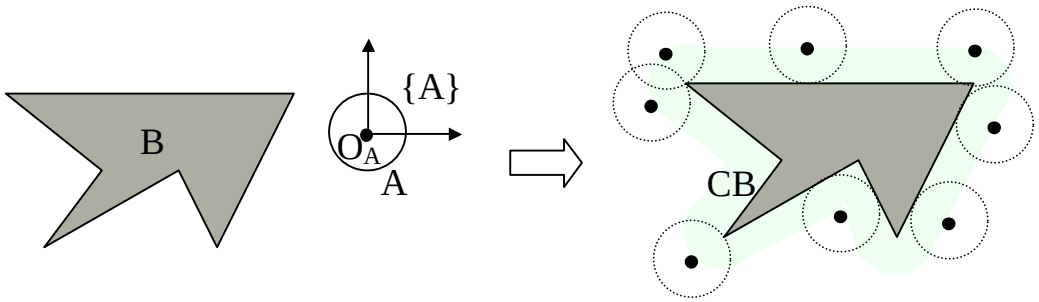

Figura 6.3. C-Obstáculo para objeto circular movendo-se em $\mathbf{R^2}$.

No caso de um manipulador planar de dois graus de liberdade, por exemplo, o problema de planejar o movimento do braço em um espaço de trabalho onde está presente um obstáculo pode ser mapeado no problema mais simples de planejar o movimento de um ponto $q = (\theta_1, \theta_2)$ no espaço de configuração.

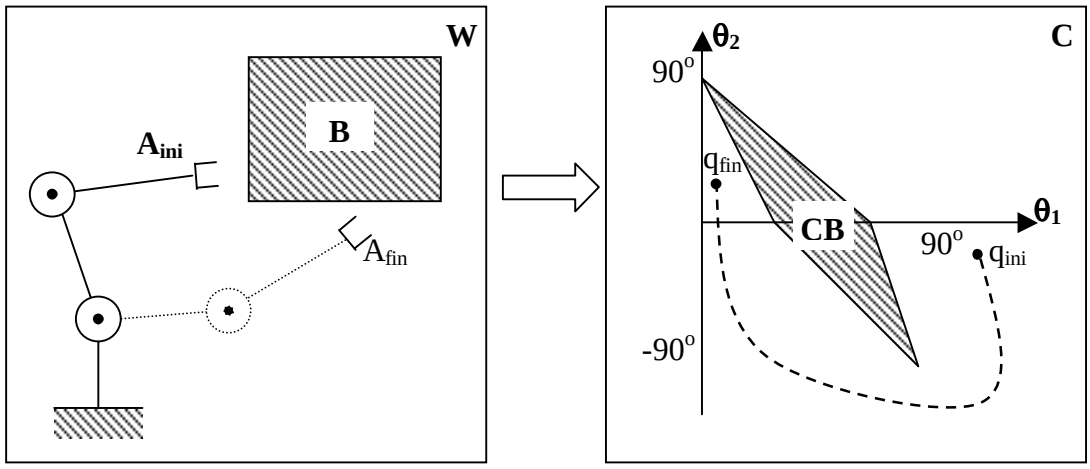

Figura 6.4. Planejamento de tarefa em W e em C.

Cálculo de C-Obstáculo para o Caso W Poligonal – Objeto A com orientação fixa:

Considere o seguinte caso particular:

- $W = \mathbf{R}^2$.
- Objeto A e obstáculo B são polígonos convexos, com o objeto A transladando sem mudar a sua orientação.

Considere que A e B são representados como uma lista de vértices enumerados em sentido anti-horário. Cada vértice representado pelas suas coordenadas.

$A = \{a_i = (x_{ai}, y_{ai}): 1 \leq i \leq n_A+1\}$ $\quad\quad$ $B = \{b_i = (x_{ai}, y_{ai}): 1 \leq i \leq n_B+1\}$

Onde,

n_A = número de vértices de A, tal que $a_{nA+1} = a_1$,
n_B = número de vértices de B, tal que $b_{nB+1} = b_1$,

Os vértices a_i e a_{i+1} definem o **lado $\mathbf{L_i^A}$** do polígono A. de forma análoga, Os vértices b_i e b_{i+1} definem o **lado $\mathbf{L_i^B}$** do polígono B.

- Define-se o vetor $\mathbf{V_i^A}$ como sendo a **normal externa** do lado $\mathbf{L_i^A}$, vetor unitário normal a $\mathbf{L_i^A}$ e apontando para fora do polígono A. De forma análoga, define-se o vetor $\mathbf{V_i^B}$ como sendo a **normal externa** do lado $\mathbf{L_i^B}$, vetor unitário normal a $\mathbf{L_i^B}$ e apontando para fora do polígono B.

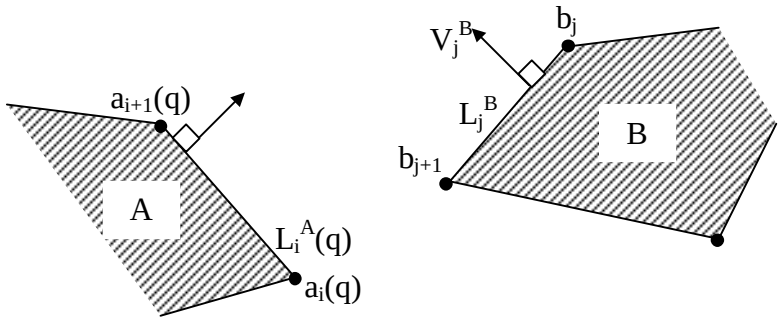

Figura 6.5. Objeto A e Obstáculo B em $\mathbf{W = R^2}$.

Para o objeto poligonal A movimentando-se em $\mathbf{R^2}$ com orientação fixa θ, seja $A_\theta(x,y)$ este objeto na posição (x,y) e dado um obstáculo B, O C-obstáculo CB_θ é calculado como:

$CB_\theta = B \ominus A_\theta(0,0) = \{P \in W / \exists\, b \in B, \exists\, a_0 \in A_\theta(0): P = b - a_0\}$

onde $\ominus$ denota o operador de Minkowski para diferença de conjuntos, b é um ponto do obstáculo e a_0 é um ponto do objeto A_θ na posição (0,0).

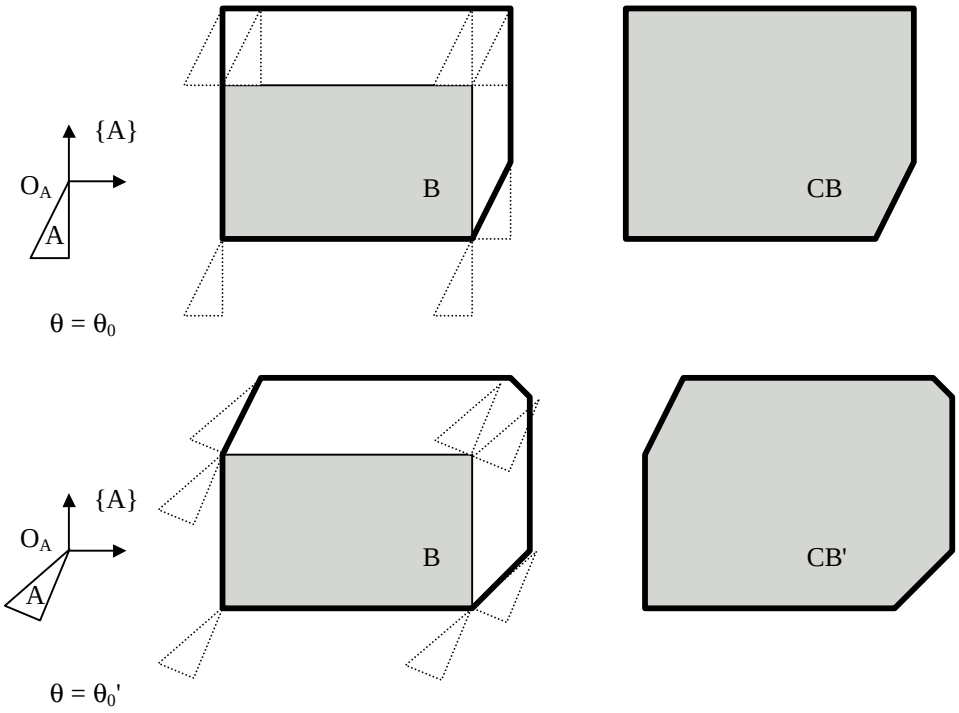

Figura 6.6. Exemplo de C-Obstáculo para objeto planar transladando em $\mathbf{R}^2$.

Procedimento prático para construir $CB(\theta_0)$:

1. Fixar os vetores normais $-V_i^A$ (com $i=1,...,n_A$) e V_j^B (com $j=1,...,n_B$) no círculo unitário S^1.

2. Varrer o círculo unitário em sentido anti-horário e criar os n_A+n_B vértices de $CB(\theta_0)$:

- Se $-V_i^A$ está entre V_{j-1}^B e V_j^B, criar o vértice $(b_j-a_i(q_0))$.
- Se V_j^B está entre $-V_{i-1}^A$ e $-V_i^A$, criar o vértice $(b_j-a_i(q_0))$.

Observações:

- Para uma orientação crítica θ_0, tal que $L_i^A(q_0)$ se desloca paralelamente em contato com L_j^B, temos $-V_i^A = V_j^B$. Os pontos (b_j-a_i), (b_{j+1},a_i), (b_j-a_{i+1}) e (b_{j+1},a_{i+1}) são co-lineares. Os pontos (b_{j+1},a_i) e (b_j-a_{i+1}) não são vértices de $CB(\theta_0)$.
- A complexidade do algoritmo é de ordem $O(n_A+n_B)$.
- Para A e B não convexos, decompor em componentes convexas A_i e B_j. Computar CB_{ij} para cada (A_i,B_j). $CB=\cup CB_{ij}$.

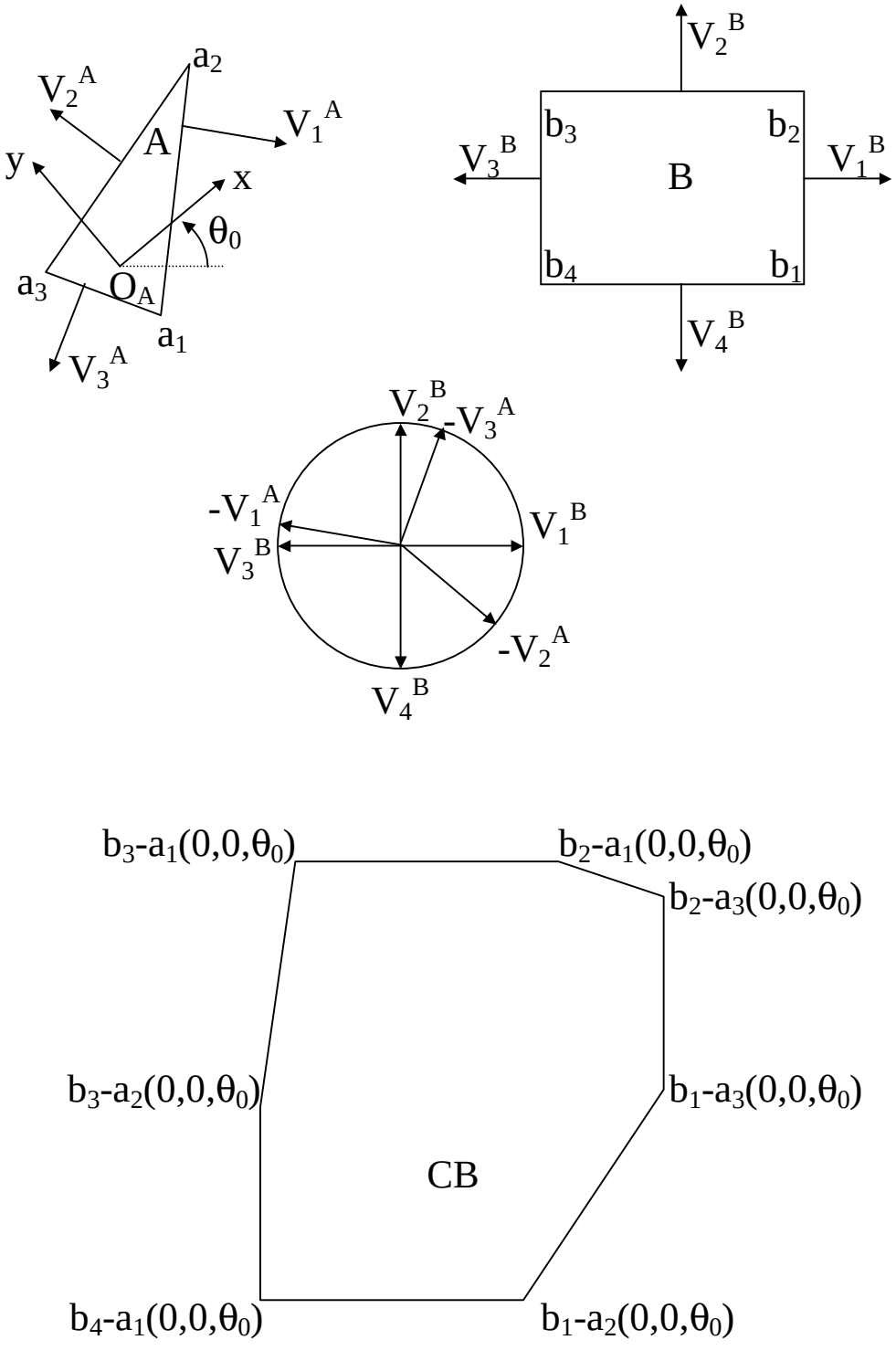

Figura 6.7. Exemplo de cálculo de C-Obstáculo para polígono transladando em $\mathbf{R^2}$.

C-Obstáculo para o Caso W Poligonal – Objeto A com orientação variável:

Quando a orientação do objeto plano também é variável, com a sua configuração dada por q= (x, y, θ), CB é o subconjunto de $R^2 \times [0, 2.\pi]$ constituído por infinitas seções transversais empilhadas, cada uma consistindo de um CB_θ bidimensional diferente.

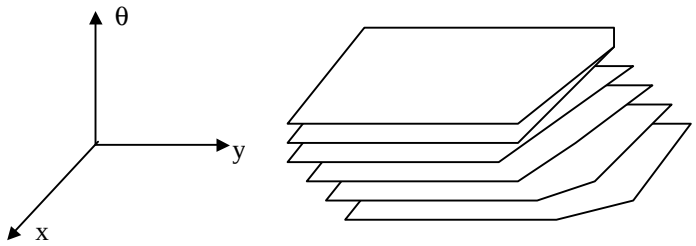

Figura 6.8. C-Obstáculo para objeto planar transladando e girando em $\mathbf{R^2}$.

Teste de Penetração de Polígonos:

O planejamento de movimento de um robô em um espaço povoado de obstáculos envolve a determinação de um caminho livre de colisões. No caso em que é possível modelar o problema em um espaço de trabalho poligonal, para detectar a colisão de objetos modelados como polígonos movimentando-se sobre um plano, é necessário testar se estes possuem interseção não nula, ou seja se há interpenetração entre eles. Para realizar este tipo de teste, uma métrica muito utilizada é a distância de um ponto (x_0, y_0) a uma reta definida por dois pontos (x_1, y_1) e (x_2, y_2). Dada a reta no plano xy passando por estes dois pontos:

$a.x + b.y + c = 0$

os coeficientes a, b e c são:

$a = y_1 - y_2$
$b = x_2 - x_1$
$c = x_1.y_2 - x_2.y_1$

a distancia $d(x_0, y_0)$ do ponto (x_0, y_0) à reta é dada por:

$$d(x_0, y_0) = (a.x_0 + b.y_0 + c)/(a^2 + b^2)^{1/2}$$

Note que $d(x_0, y_0)$ é uma função que pode admitir valores positivos, negativos ou nulos. Se $d(x_0, y_0) = 0$, o ponto (x_0, y_0) está sobre a reta. Se $d(x_0, y_0) > 0$, o ponto está no semiplano à esquerda da reta (de acordo com o sentido do vetor que vai de (x_1, y_1) ao ponto (x_2, y_2)). Se $d(x_0, y_0) < 0$, o ponto está no semiplano à direita da reta.

Assim, dado um polígono com n_P vértices $P = \{p_k = (x_k,y_k) : 1 \leq k \leq n_P+1\}$, tal que o vértice $p_{nP+1} = p_1$, o algoritmo a seguir permite determinar se um dado ponto (x_0,y_0) está ou não dentro do polígono P e, em caso afirmativo, o grau de penetração do ponto em P (distância ao lado mais próximo).

Algoritmo de Penetração de um Ponto (x_0,y_0) em um polígono P:

1. Inicializar: $k = 1$, d_0 = valor real máximo > perímetro de P.
2. Computar:
   $a = y_k - y_{k+1}$
   $b = x_{k+1} - x_k$
   $c = x_k.y_{k+1} - x_{k+1}.y_k$
   $d = (a.x_0 + b.y_0 + c)/(a^2 + b^2)^{1/2}$
3. Se $d < d_0$, então faça $d_0 = d$
4. Faça $k = k+1$. Se $k \leq n_P$, voltar ao passo 2.

A saída do algoritmo é a <u>distância de penetração</u> d_0. Se $d_0 > 0$, o ponto (x_0,y_0) está dentro do polígono P e d_0 é uma medida da penetração do ponto dentro do polígono.

Para testar se dois polígonos convexos A e B possuem intersecção não nula, devemos testar qualquer uma das seguintes condições é satisfeita:

- Verificar se pelo menos um vértice a_i de A está dentro de B.
- Verificar se pelo menos um vértice b_k de B está dentro de A.
- Verificar se um pelo menos um lado de A, (L_i^A, definido pelos vértices a_i e a_{i+i}), cruzar com um lado de B, (L_k^B, definido pelos vértices b_k e b_{k+i}).

As duas primeiras condições verificam a situação em que um vértice de um dos polígonos penetrou dentro do outro polígono. A terceira condição verifica a situação em que, mesmo sem existir algum vértice de um polígono dentro do outro, há cruzamento de lados, como por exemplo, no caso de dois retângulos superpostos formando uma cruz.

Para testar as duas primeiras condições, devemos executar o algoritmo de penetração para todos os vértices do polígono A em relação ao polígono B e para todos os vértices do polígono B em relação ao polígono A.

Para testar a terceira condição, devemos verificar se cada um dos lados de A intersecta qualquer um dos lados de B. Dados dois lados quaisquer L_i^A e L_k^B, dos polígonos A e B, respectivamente, os mesmos se intersectam se os vértices a_i e a_{i+1} estão em lados opostos de L_k^B e, simultaneamente, os vértices b_k e a_{k+1} estão em lados opostos de L_i^A. Ou seja:

Para todo par de lados (L_i^A, L_i^B), com $i = 1,..., n_A$ e $k = 1,...,n_B$, devemos testar se as distâncias $d(a_i)$ e $d(a_{i+1})$ a L_k^B possuem sinais opostos e se as distâncias $d(b_k)$ e $d(b_{k+1})$ a L_i^A também possuem sinais opostos.

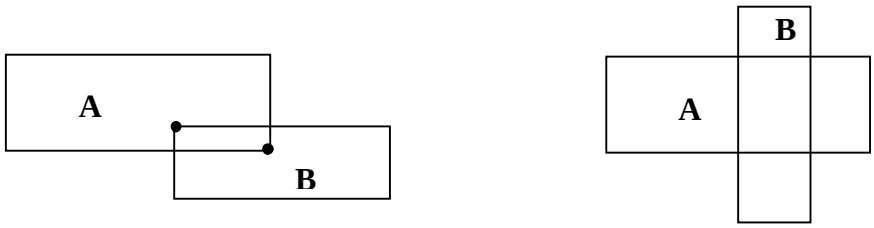

Fig. 6.9. Situações de teste de colisão.
a) Vértice de A dentro de B ou vértice de B dentro de A; b) Cruzamento de lados.

Na prática, em uma aplicação de planejamento de movimento em que o caminho é constituído de configurações sobre uma curva contínua, supondo que a mesma seja percorrida em pequenos deslocamentos discretos e o teste de colisão seja feito para essa seqüência de deslocamentos, dificilmente aparecerá o terceiro caso, pois para chegar a ele, provavelmente deve-se passar antes por algum dos dois primeiros casos. Assim, aplicando o teste de penetração para todos os vértices seria suficiente. Por outro lado, o teste de cruzamento de lados incorpora os dois primeiros casos, uma vez que quando um vértice está dentro de um polígono, os dois lados dos quais o mesmo é extremo cruzarão algum dos lados desse polígono (Devem ser tratadas ainda os casos excepcionais quando. Um lado de um polígono cruza o limite de outro polígono exatamente através de um vértice).

6.3. Métodos de Planejamento de Caminhos:

6.3.1. Métodos Baseados em Mapa de Rotas

Princípio: Capturar a conectividade de C_L na forma de uma rede de curvas unidimensionais R, (Mapa de Rotas), que é usada como um conjunto de caminhos padrão. O planejamento se reduz a buscar um caminho em R entre qini e qfin. Exemplo: Grafo de Visibilidade, Diagrama de Voronoi, Rede de Caminhos Livres, Método da Silhueta, etc.

Planejamento baseado em Grafo de Visibilidade:

- Aplicação: W = $\mathbf{R^2}$, robô A poligonal e com orientação fixa, com obstáculos B_i poligonais.
- Princípio: Construir um caminho semi-livre entre q_{ini} e q_{fin} formado por uma linha poligonal através dos vértices de CB.

Grafo de Visibilidade - Grafo não direcional, G:

- Os Nós de G são q_{ini} e q_{fin} e os vértices de CB.
- Dois nós de G são conexos por um arco se e somente se o segmento que os une é um eixo de CB ou está contido inteiramente em CL, com possível exceção dos seus extremos.

$\Rightarrow$ O Grafo de Visibilidade contém o caminho mais curto, (em métrica euclidiana em $\mathbf{R}^2$), entre q_{ini} e q_{fin}.

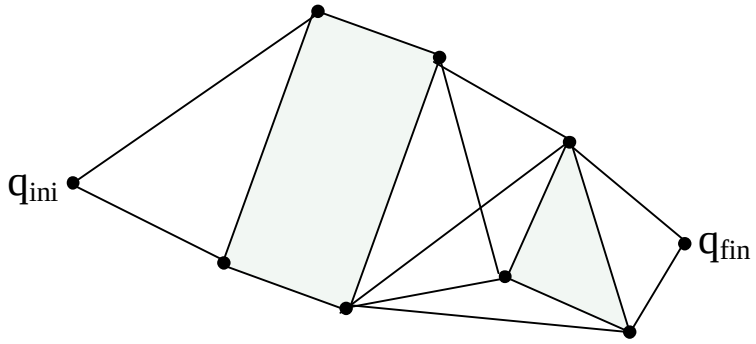

Fig. 6.10. Grafo de Visibilidade.

Algoritmo de Planejamento:

1. Construir o Grafo G.
2. Buscar um caminho em G de qini até qfin.
3. Se o caminho é encontrado, retorná-lo, se não, reportar falha.

Construção do Grafo de Visibilidade:

1. Tomar pares (X, X') de nós de G.
2. Se X e X' são extremos do mesmo lado em CB, conectá-los por um arco em G.
3. Se X e X' não são extremos do mesmo lado em CB, computar as interseções da do segmento (X, X') com os lados de CB. Caso não houver nenhuma interseção, ligar os nós por um arco em G.

Observação: complexidade computacional de ordem $O(n^3)$, onde n = número de vértices.

Busca do Menor Caminho em G de q_{ini} até q_{fin}:

Para buscar o menor caminho, usam-se técnicas de busca em grafos, como, por exemplo, o Algoritmo A^*, que usa a distância euclidiana ao alvo como função heurística para guiar a busca.
O Grafo é representado pela estrutura de dados G = (X, A), onde X é o conjunto de n nós e A é o conjunto de r arcos. O grafo G é representado por listas de adjacências, uma para cada nó N (lista de todos os nós N' ligados por um arco a N).

O Algoritmo A* possui as seguintes características:

- Possui complexidade computacional $O(r.\log(n))$.
- É aplicável a grafos em que os arcos têm custos associados, por exemplo, distância euclidiana entre os nós. O Custo de um caminho =é definido como a soma dos custos dos seus arcos.
- Permite obter o menor caminho em termos da métrica adotada.

Procedimento:

- G explorado iterativamente seguindo caminhos a partir de N_{ini}.
- Para cada nó visitado, um ou mais caminhos são gerados a partir de N_{ini}, mas só o de menor custo é memorizado.
- O conjunto de caminhos gerados forma uma árvore T do subconjunto de G já explorado.
- T é representada através de ponteiros, associados a cada nó visitado, os quais apontam para os correspondentes nós pais.
- Para cada nó N, atribui-se uma função de custo, estimativa do custo do caminho de custo mínimo passando por N:

  $f(N) = g(N) + h(N)$

  $g(N)$ = custo do caminho entre N_{ini} e N em T corrente.
  $h(N)$ = estimativa heurística do custo $h^*(N)$ do caminho de mínimo custo entre N e N_{fin}. Por exemplo, distância euclidiana ao alvo. Se h(N) for nula (nenhuma informação heurística disponível), o método de busca fica reduzido ao Método de Dijkstra.

Dadas as seguinte estruturas de dados e funções:

- G(X,A) = Grafo com n nós $\in$ X e r arcos $\in$ A, com conectividade representada por listas de adjacências entre nós.
- T = Árvore do subconjunto de G já visitado.
- L = Lista que armazena nós de G ordenados por f(N).
- $k : X \times X \rightarrow \mathbf{R}^+$ função que especifica o custo de cada arco.
- h(N): estimativa do custo do caminho mínimo entre N e N_{fin}.
- g(N) = custo do caminho entre N_{ini} e N em T corrente.

Dadas as seguinte operações sobre a lista L:

- PRIMEIRO(L): retorna o nó com menor valor de f(N) em L e o remove da lista.
- INSERIR(N,L): insere o nó N na lista L.
- APAGAR(N,L): apaga o nó N da lista L.
- MEMBRO(N,A): determina se o nó N é membro da lista L.
- VAZIA(L): determina se a lista L está vazia.

```
Procedimento A*(G, N_ini, N_fin, k, h);
começar
    N_ini em T;
    INSERIR(N_ini, L); marcar N_ini como visitado;
    enquanto ¬VAZIA(L), faça
    começar
        N ← PRIMEIRO(L);
        se N = N_fin, então sair do laço while;
        para cada nó N' adjacente a N em G, faça
            se N' é não visitado, então
            começar
                adicionar N' a T com um ponteiro para N;
                INSERIR(N',L); marcar N' como visitado;
            fim
            se não, se g(N') > g(N) + k(N, N'), então
            começar
                redirecionar o ponteiro de N' para N em T;
                se MEMBRO(N',L),então APAGAR(N',L);
                INSERIR(N',L);
            fim
    fim;
    se ¬VAZIA(L), então
        retornar o caminho traçando os ponteiros de N_fin a N_ini;
    se não reportar falha;
fim;
```

6.3.2. Métodos baseados em Decomposição em Células Convexas

Princípio:

- Decomposição de C_L em regiões não superpostas cuja união é exatamente C_L (Métodos de Decomposição Exata), ou uma aproximação conservadora de C_L (Métodos de Decomposição Aproximada).
- Construção de um grafo de conectividade G que representa as relações de adjacência entre as células.
- Busca de um Canal (seqüência de células adjacentes ligando a célula contendo q_{ini} (K_{ini}) à célula contendo q_{fin} (K_{fin})) em G.
- Extração de um caminho entre q_{ini} e q_{fin} a partir do canal.

Exemplo - Decomposição Trapezoidal (não ótima):

Aplicável a: $C = \mathbf{R}^2$, CB = Região Poligonal, C_L limitado.

Procedimento:

- Varrer C_L com uma linha reta vertical.

- Quando um vértice X de CB é encontrado, um máximo de dois segmentos de reta verticais, contidos em C_L, são criados de modo a conectar X aos eixos de CB imediatamente acima e abaixo do mesmo.
- Os limites de CB e os segmentos verticais determinam a Decomposição Trapezoidal de C_L. $\Rightarrow$ Cada célula é um trapezóide ou um triângulo.
- Duas células são adjacentes se e somente se seus limites partilham um dos segmentos verticais gerados na varredura.

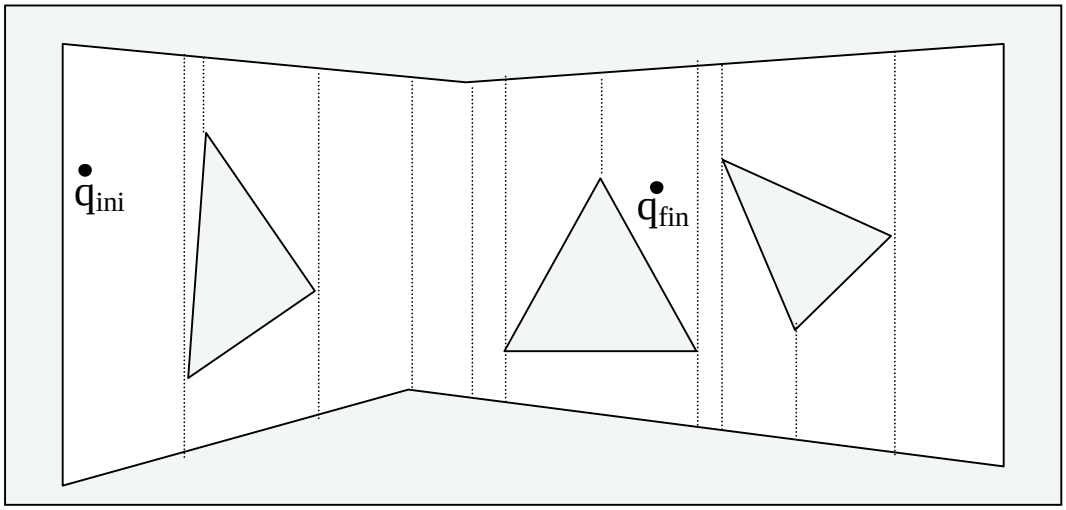

Fig. 6.11. Decomposição Trapezoidal.

Construção do Grafo de Conectividade:

- Os nós do grafo são as células da decomposição.
- Dois nós são conectados por um arco se as células correspondentes são adjacentes (pelo menos um vértice de uma célula está contido em um lado da outra e vice-versa, ou os dois vértices de um lado de uma célula estão contidos em um lado da outra). Como os lados adjacentes são verticais, basta testar as ordenadas de vértices de lados com a mesma abscissa.

Busca de um Canal entre K_{ini} e K_{fin}:

A busca de um canal entre as células K_{ini} e K_{fin} pode ser realizada através do algoritmo A*, ponderando devidamente os arcos do grafo de conectividade.

O arco que liga duas células adjacentes K_i e K_{i+1} pode ser ponderado, por exemplo pela soma das distâncias que unem os centróides das mesmas C_i e C_{i+1} ao ponto central do segmento de reta β_i em que são adjacentes. No caso das células K_{ini} e K_{fin}, em lugar dos seus centróides pode-se utilizar q_{ini} e q_{fin} , respectivamente.

Extração de um caminho entre q_{ini} e q_{fin} a partir do canal:

- Dados os limites $\beta_i = \partial k_i \cap \partial k_{i+1}$, entre duas células adjacentes k_i e k_{i+1}, determinar os pontos médios Q_i de β_i.

- Determinar o centróide C_i de cada célula K_i do canal.
- ligar q_{ini} a q_{fin} através da linha poligonal por Q_1, C_2, Q_2, C_3, ..., C_{p-1}, Q_{p-1}.

Uma alternativa mais simples é ligar q_{ini} a q_{fin} através da linha poligonal por Q_1, Q_2, ..., Q_{p-1} e, apenas no caso em que β_i e β_{i+1} estejam contidos na mesma reta suporte, passar pelo centróide C_i.

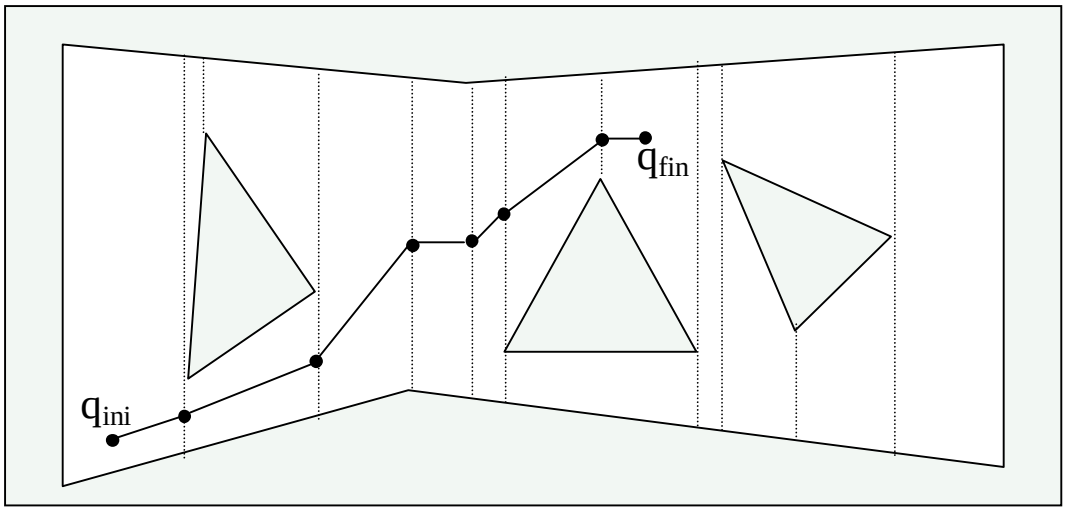

Fig. 6.12. Extração de um Caminho a partir de um canal.

6.3.3. Métodos baseados em Campos de Potencial

Princípio:

- Considerar o robô, representado como um ponto no espaço de configuração, como uma partícula sob a influência de um campo de potencial artificial **U**, cujas variações locais refletem a estrutura do espaço livre.
- Tipicamente, a função de potencial é uma soma de potenciais repulsivos, (geralmente com influência local), que afastam o robô dos obstáculos e um potencial atrativo que atrai o robô em direção ao alvo.
- O planejamento é realizado iterativamente. A cada iteração, a força artificial induzida por **U** na configuração q, $F(q) = -\nabla \mathbf{U}(q)$, define a direção de movimento mais promissora e o incremento de posição nesta direção.

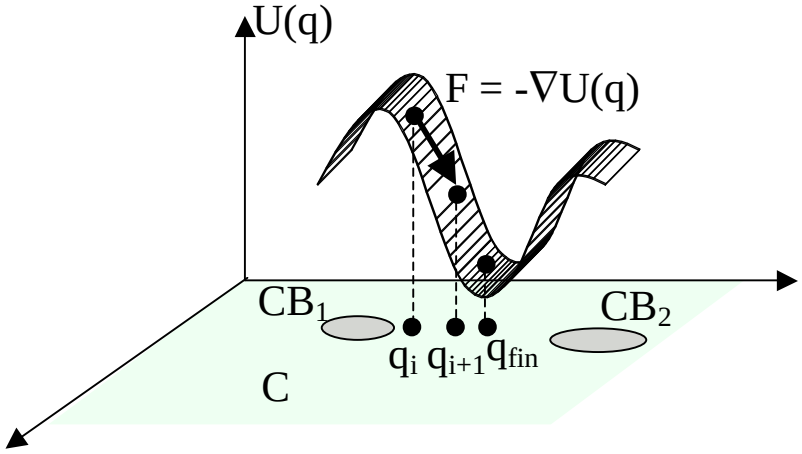

Fig. 6.13. Campo de Potencial.

Características:

- Desenvolvido originalmente para contorno de obstáculos por robôs móveis, aplicável quando não se dispõe de um modelo *a priori* dos mesmos, mas são percebidos *on-line*. ⇒ Método "Local".
- Ênfase em eficiência em tempo real em detrimento da garantia de alcançar o alvo. ⇒ Método incompleto, (pode falhar na busca de um caminho, mesmo existindo um), mas de implementação simples; geralmente rápido, eficiente e confiável para a maioria das aplicações.
- Método de otimização baseado em gradiente ("Descida da Ladeira"). ⇒ Sujeito a problemas de mínimos locais da função de potencial, (o seu maior problema). Abordagens de solução:

1. Definir a função de potencial de modo a ter um ou uns poucos mínimos locais. ⇒ tornar o método "Global".
2. Incluir técnicas para escapar dos mínimos locais.

Exemplo: Função de Navegação Manhattan:

O método é aplicável a C = $\mathbf{R}^2$. É um método aproximado (pode existir caminho livre e o algoritmo falhar em encontra-lo. Gera-se uma aproximação conservadora GR_L de C_L discretizando C em uma grade retangulóide GR. Assume-se que q_{ini} e q_{fin} pertencem a GR_L.

O método é baseado em uma métrica denominada Função de Navegação Manhattan = distância L^1 a q_{fin}. Cada configuração q em GR é rotulada com um potencial U(q) = -M se $q \in C_L$ e U(q) = M caso contrário, onde M é um número inteiro muito grande (maior do que a maior dimensão de GR). Assim, as configurações livres da grade pertencem ao conjunto $GR_L = \{q \in GR / U(q) = -M\}$ e as configurações ocupadas por C-obstáculos possuem um potencial M. A função de navegação Manhattan (potencial U(q) é computada facilmente usando uma "expansão de frente de onda" a partir de q_{fin}

- $U(q_{fin})$ é fixado em 0.

- O potencial de cada 1-vizinha de q_{fin} em GR_L é igual ao potencial da célula corrente mais um.
- O procedimento prossegue iterativamente incrementando o potencial das configurações 1-vizinhas à configuração atual em GR_L, cujo potencial não tenha ainda sido computado.
- Terminar quando o subconjunto de GR_L acessível a partir de q_{fin} tenha sido explorado completamente.

Observações:

- A função de navegação gera potenciais em GR_L com um único mínimo em q_{fin}.
- Pode-se usar o algoritmo de navegação "Fundo Primeiro" para buscar um caminho entre q_{ini} e q_{fin} em GR_L: movimentar-se para a configuração vizinha em GR_L com o menor potencial.
- É garantido encontrar um caminho entre q_{ini} e q_{fin} caso este exista em GR_L.
- O caminho gerado em GR_L entre q_{ini} e q_{fin} é de comprimento mínimo (de acordo com a métrica L^1).
- O algoritmo calcula U(q) somente no subconjunto de GR_L conexo a q_{fin}.
- Se $U(q_{ini})$ não foi computado, pode-se concluir imediatamente que não existe um caminho entre q_{ini} e q_{fin} em GR_L.

- A complexidade computacional do algoritmo é linear no número de configurações em GR e independente do número e forma dos C-Obstáculos.
- O algoritmo é eficiente para um espaço de configuração de dimensão baixa (m = 2 ou 3), tornando-se impraticável para dimensões maiores.

```
procedimento Manhattan
começar
        para cada q ∈ CB faça U(q) ← M;   /* M = Nº grande */
        para cada q ∈ GR_L faça U(q) ← -M;
        U(q_fin) ← 0; inserir q_fin em L_0;
        /* L_i = lista de configurações, inicialmente vazia (i=0,1,...)*/
        para i = 1, 2, ..., até L_i = vazia, faça
                para cada q em L_i, faça
                        para cada q' 1-vizinha de q em GR_L, faça
                                se U(q') = -M, então
                                começar
                                        U(q') ← i+1;
                                        inserir q' no fim de L_i+1;
                                fim
fim
```

2	1	2	3	4	5		
1	q_{fin}	1	2	3	4		
2	1	2	3	4	5		
3	2			5	6		
4	3			6	7	8	q_{ini}
5	4			7	8	9	10
6	5	6	7	8	9	10	11
7	6	7	8	9	10	11	12

Fig. 6.14. Planejamento baseado em Campo de Potencial.