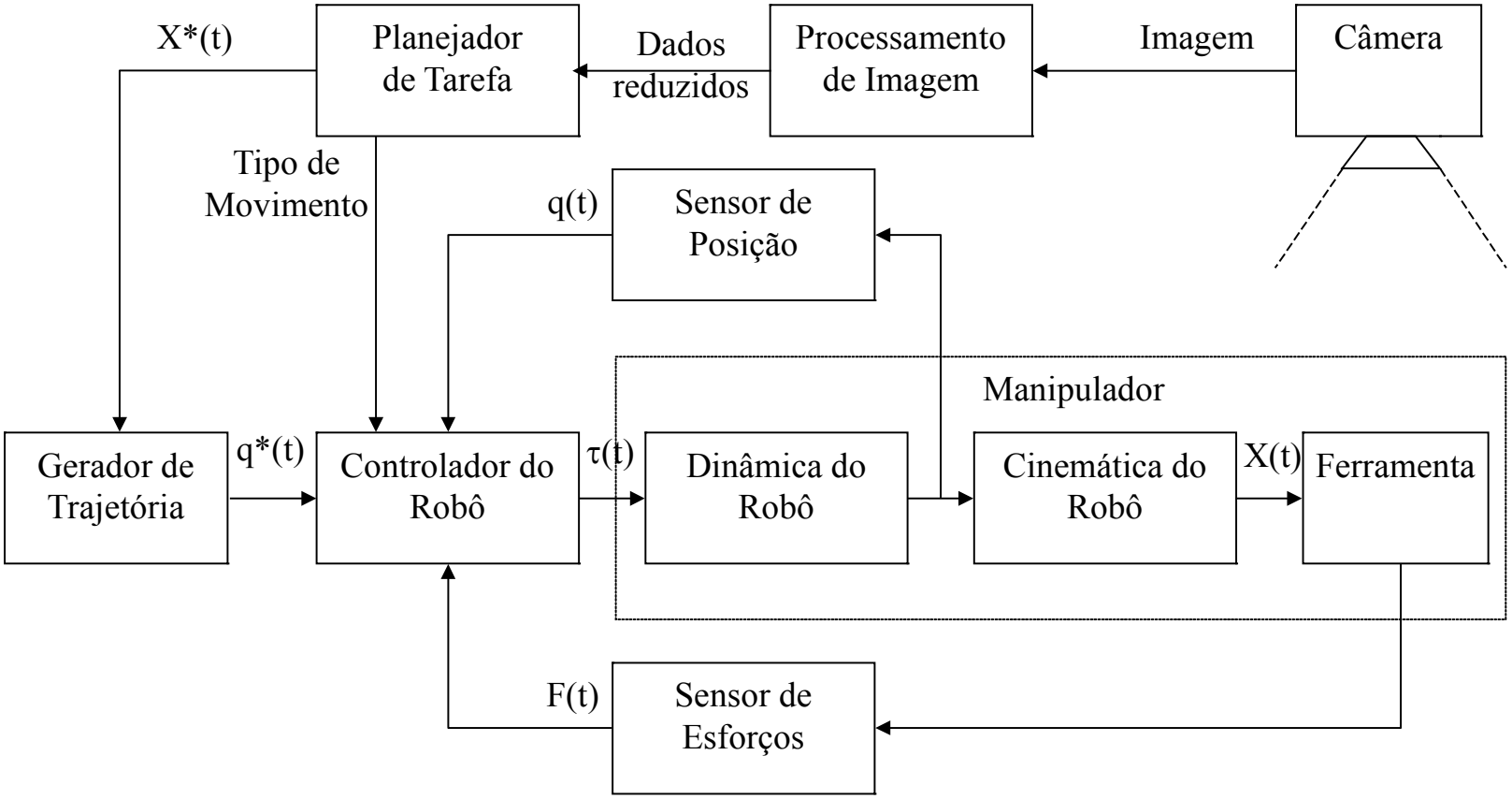
PLANEJAMENTO DE TAREFAS:

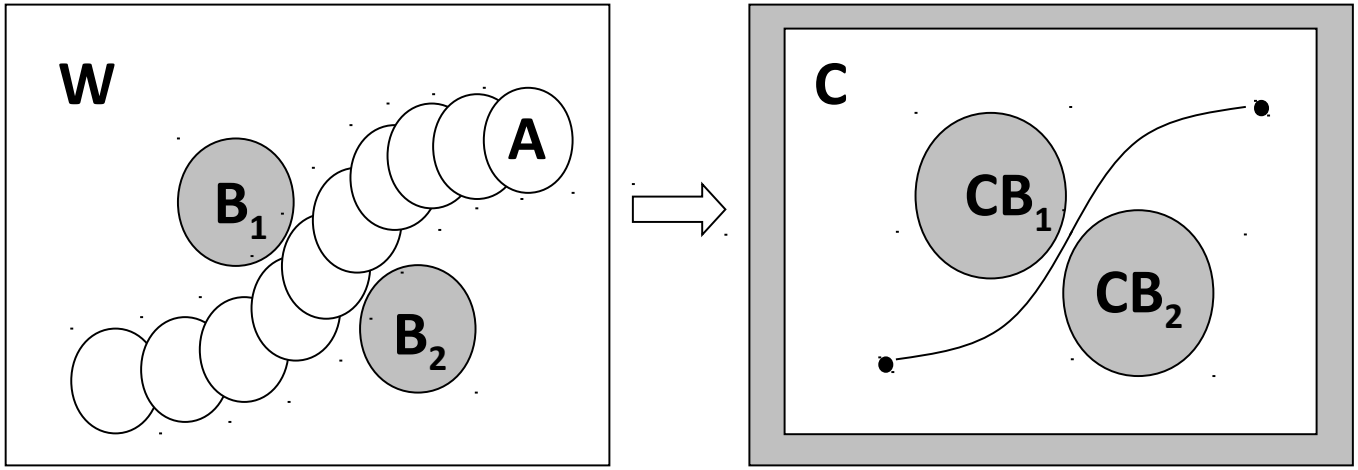
- **Planejamento de Tarefas: definição de objetivos gerais, independente dos meios de alcançá-los.**



Espaço de Configuração

- Configuração q de um robô A = especificação da sua pose em relação ao espaço de trabalho {W}.
- Espaço de Configuração = espaço N-dimensional C de todas as possíveis configurações de A
- A(q) é o subconjunto de W ocupado por A na configuração q.
- Mapear os obstáculos do Espaço de Trabalho para o Espaço de Configuração simplifica o problema de planejamento

Espaço de Configuração



Movimento de um robô A no Espaço de Trabalho **W** povoado de obstáculos Bi's.

Movimento de um ponto no Espaço de Configuração **C** povoado de C-obstáculos CBi's.

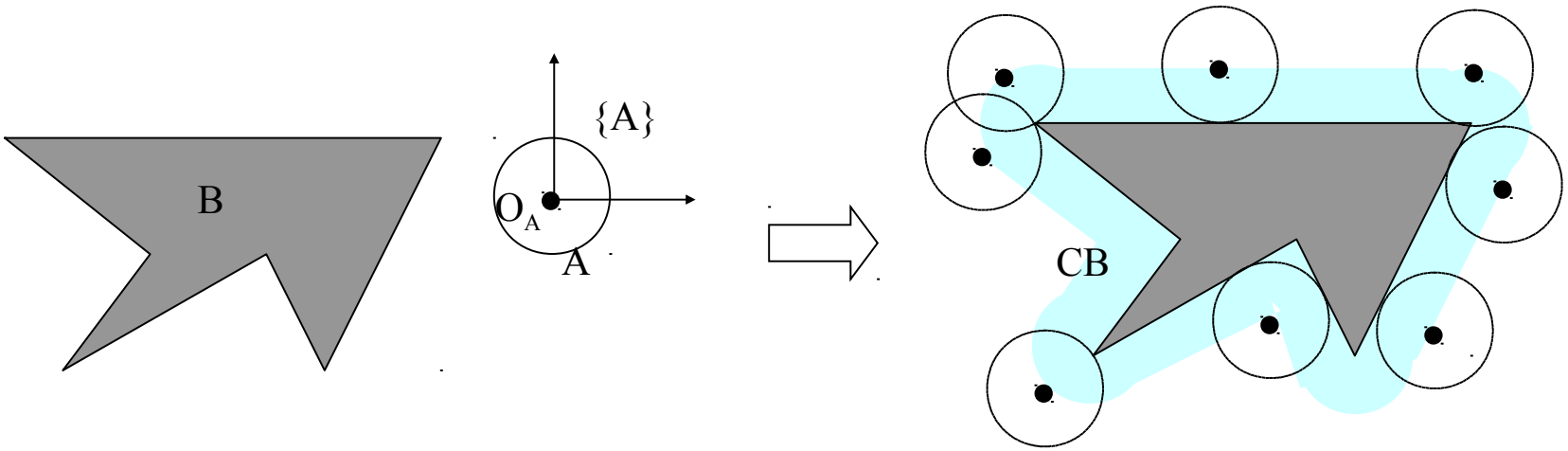
Obstáculos em Espaço de Configuração

- um obstáculo B_i em W é mapeado em um C-obstáculo CB_i em C:

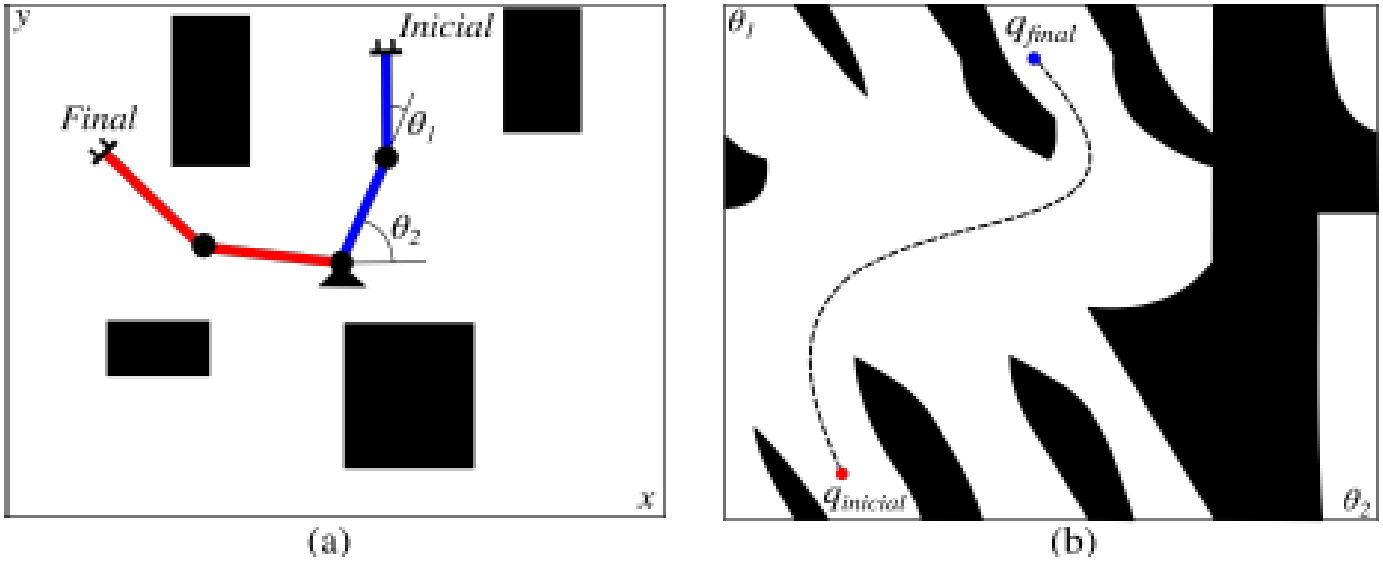
  $CB_i = \{q \in C \,/\, A(q) \cap B_i \neq \emptyset\}$

- CB_i é o conjunto de configurações em que o robô A se superpõe parcial ou totalmente com o obstáculo B_i

Obstáculos em Espaço de Configuração



Obstáculos em Espaço de Configuração



(a) (b)

C-Obstáculo - W Poligonal, orientação fixa

- $W = \mathbf{R}^2$.
- A e B polígonos convexos,
- A translada sem mudar orientação.
- A e B representados por lista de vértices enumerados em sentido anti-horário.

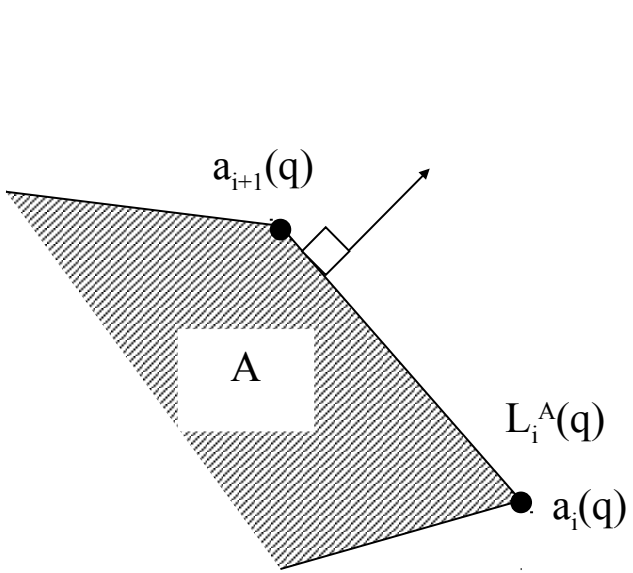
  $A = \{a_i = (x_{ai}, y_{ai}): 1 \leq i \leq n_A+1\}$

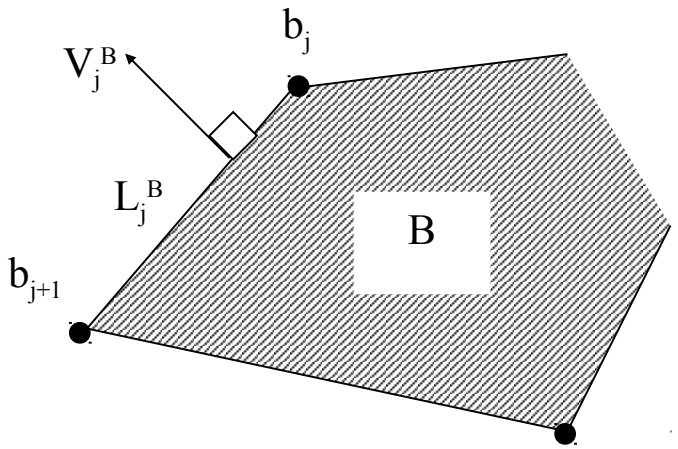
  $B = \{b_i = (x_{ai}, y_{ai}): 1 \leq i \leq n_B+1\}$

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

C-Obstáculo - W Poligonal, orientação fixa

- Vértices a_i e a_{i+1} definem **lado L_i^A** do polígono A.
- Vértices b_i e b_{i+1} definem **lado L_i^B** do polígono B.
- $\mathbf{V_i^A}$ = **normal externa** do lado $\mathbf{L_i^A}$.
- $\mathbf{V_i^B}$ = **normal externa** do lado $\mathbf{L_i^B}$.





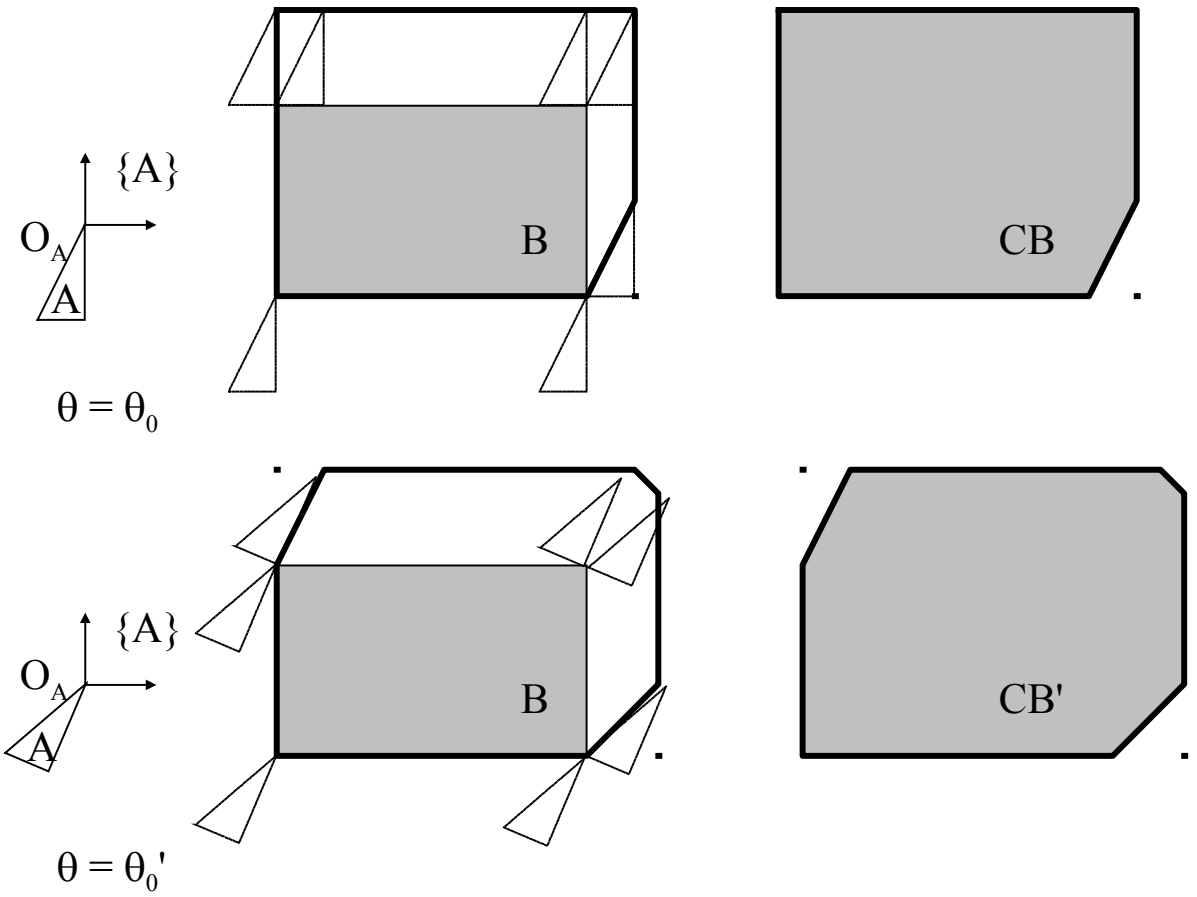
C-Obstáculo - W Poligonal, orientação fixa

- $A_\theta(x,y)$ = A na posição (x,y) em $\mathbf{R^2}$ com orientação fixa θ
- C-obstáculo CB_θ:

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

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

C-Obstáculo - W Poligonal, orientação fixa



C-Obstáculo - W Poligonal, orientação fixa

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

1) Fixar $-V_i^A$ (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 $CB(\theta_0)$:

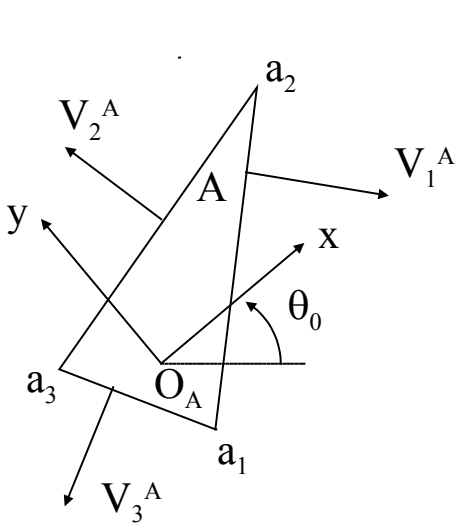
- Se $-V_i^A$ está entre V_{j-1}^B e V_j^B, criar vértice $(b_j-a_i(q_0))$.
- Se V_j^B está entre $-V_{i-1}^A$ e $-V_i^A$, criar vértice $(b_j-a_i(q_0))$.

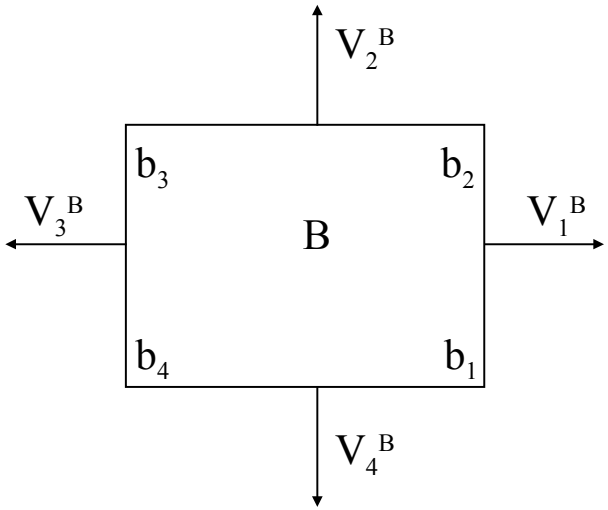
C-Obstáculo - W Poligonal, orientação fixa

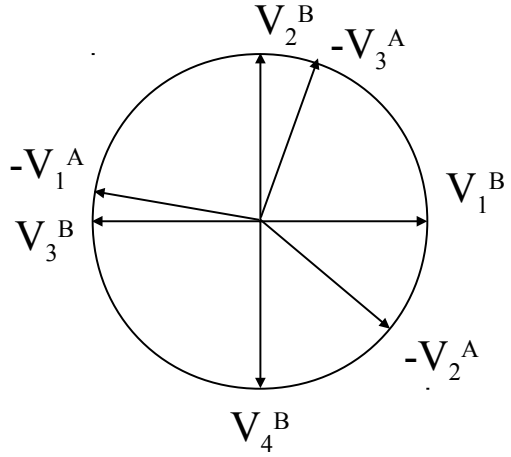
Observações:

- Para 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). $CB=\cup CB_{ij}$.

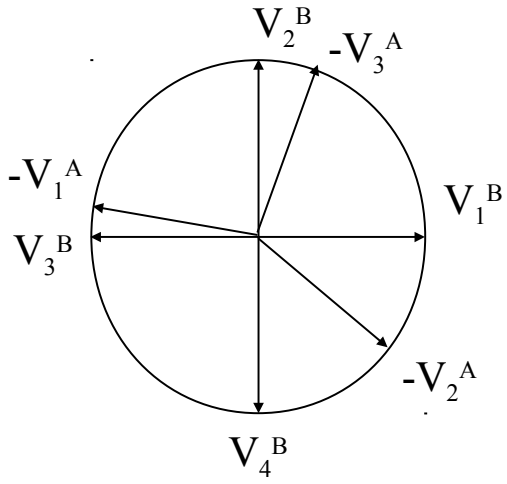
C-Obstáculo - W Poligonal, orientação fixa

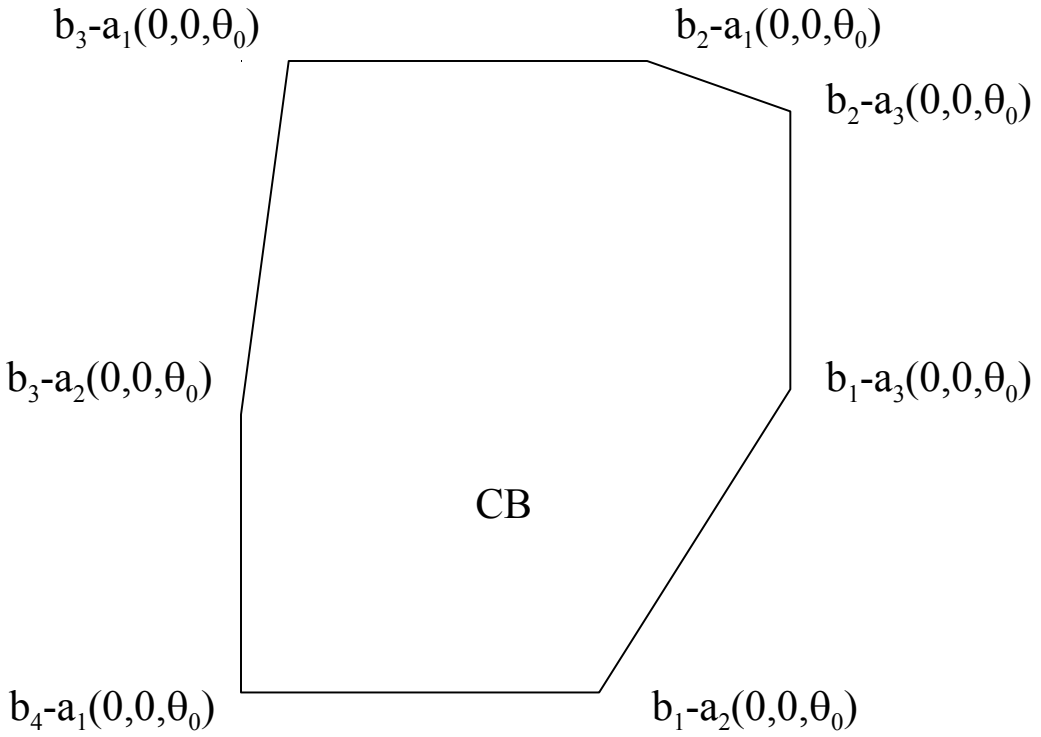






C-Obstáculo - W Poligonal, orientação fixa





C-Obstáculo - W Poligonal, orientação variável

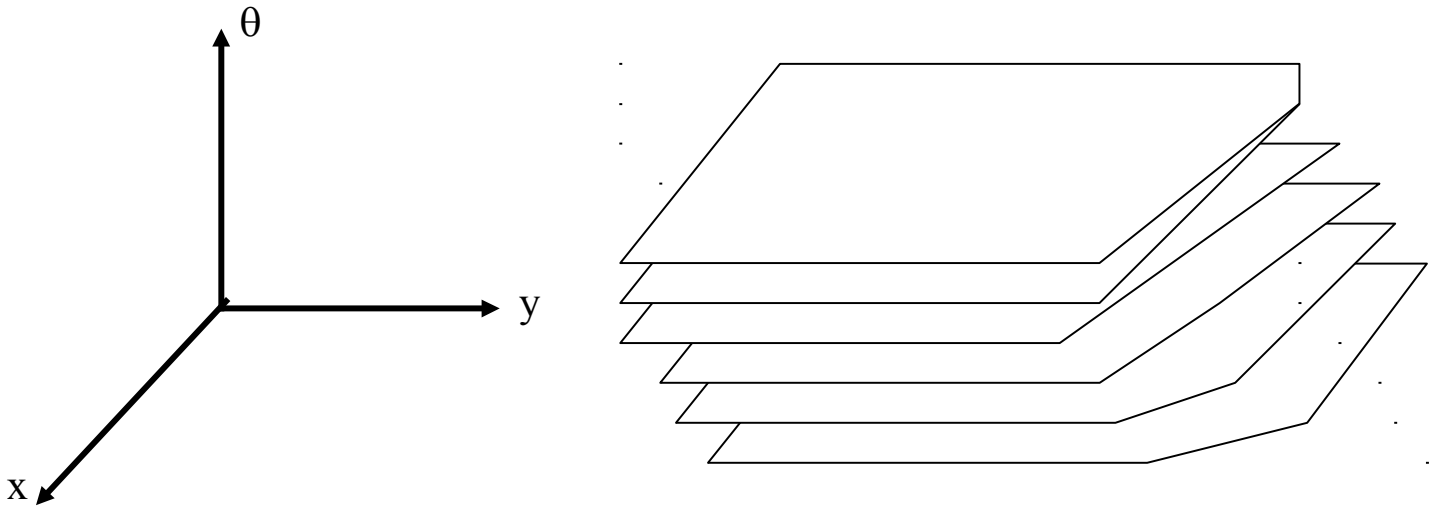
- Orientação variável , configuração $q = (x, y, \theta)$

$\Rightarrow CB \subset R^2 \times [0, 2.\pi]$

$\Rightarrow$ CB constituído por infinitas seções transversais empilhadas ao longo do eixo θ de C

$\Rightarrow$ Cada seção de CB constituída de um CB_θ bidimensional diferente.

C-Obstáculo - W Poligonal, orientação variável



Teste de Colisão

- Colisão entre dois polígonos A e B $\Rightarrow$

  $A \cap B \neq \varnothing$

- Métrica para testar colisão: distância de um ponto (x_0, y_0) a uma reta definida por dois pontos (x_1, y_1) e (x_2, y_2).

Teste de Colisão

- Reta por (x_1, y_1) e (x_2, y_2): $a.x + b.y + c = 0$

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

- Distancia de (x_0, y_0) à reta:

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

- 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 à esquerda da reta.
- Se $d(x_0, y_0) < 0$, o ponto está à direita da reta.

Teste de Colisão

- Se $A \cap B \neq \varnothing \Rightarrow$ qualquer uma das seguintes condições é satisfeita:

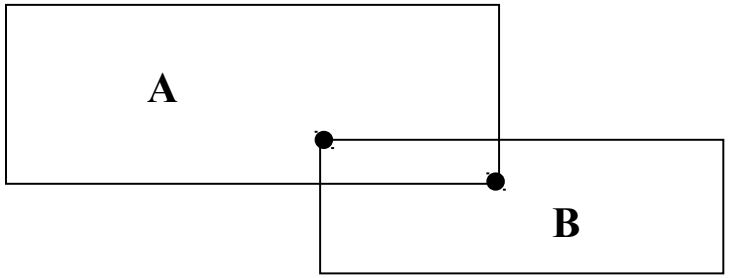
$\forall$ • Pelo menos um vértice a_i de A está dentro de B.

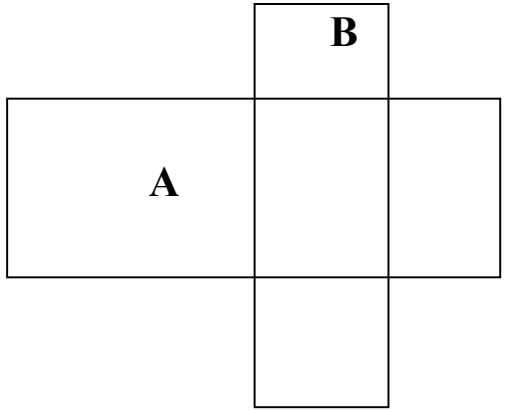
$\forall$ • Pelo menos um vértice b_k de B está dentro de A.

$\forall$ • Pelo menos um lado de A, (L_i^A, definido pelos vértices a_i e a_{i+i}), cruza com um lado de B, (L_k^B, definido pelos vértices b_k e b_{k+i}).

Teste de Colisão

- Situações de colisão:





Teste de Colisão

- Para testar as duas primeiras condições, devemos testar todos os vértices de A em relação a B e e todos os vértices de B em relação a A.
- Para testar a terceira condição, checar a intersecção de todas combinações possíveis de pares de lados (L_i^A, L_k^B).
- Na prática, para pequenos deslocamentos sobre uma curva contínua, dificilmente ocorrerá o terceiro caso.

Teste de Colisão

- Algoritmo de penetração de um Ponto (x_0,y_0) em um polígono P com n_P vértices $P = \{p_k = (x_k,y_k) : 1 \leq k \leq n_P+1\}$, tal que $p_{nP+1} = p_1$:

- 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.

- Saída do algoritmo = <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.

Teste de Colisão

- Teste de intersecção entre um par de lados (L_i^A, L_k^B) de dois polígonos convexos A e B, com com i = 1,..., n_A e k = 1,...,n_B,:

Verificar 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.

Métodos de Planejamento de Caminhos

- Mapa de Rotas
- Decomposição em Células Convexas
- Campos de Potencial

Mapa de Rotas

Princípio:

- Capturar a conectividade de C_L em uma rede de curvas unidimensionais R, (Mapa de Rotas), usada como um conjunto de caminhos padrão.
- Planejamento se reduz a buscar um caminho em R entre qini e qfin.
- Exemplos: Grafo de Visibilidade, Diagrama de Voronoi, Rede de Caminhos Livres, Método da Silhueta, etc.

Grafo de Visibilidade

- Aplicação: W = $\mathbf{R}^2$, robô A e obstáculos B_i poligonais, Robô com orientação fixa.

Princípio:

- Construir um caminho entre q_{ini} e q_{fin} formado por uma linha poligonal através dos vértices de CB.
- Vértices são ligados se são visíveis entre si.

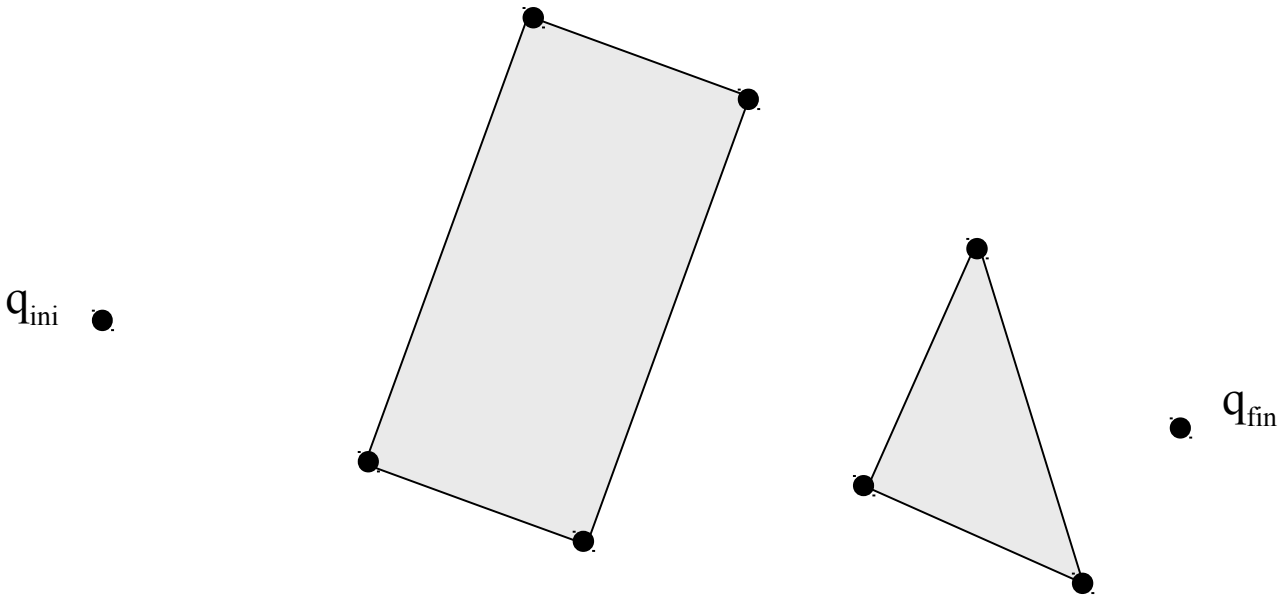
Grafo de Visibilidade

Grafo de Visibilidade G, não direcional:

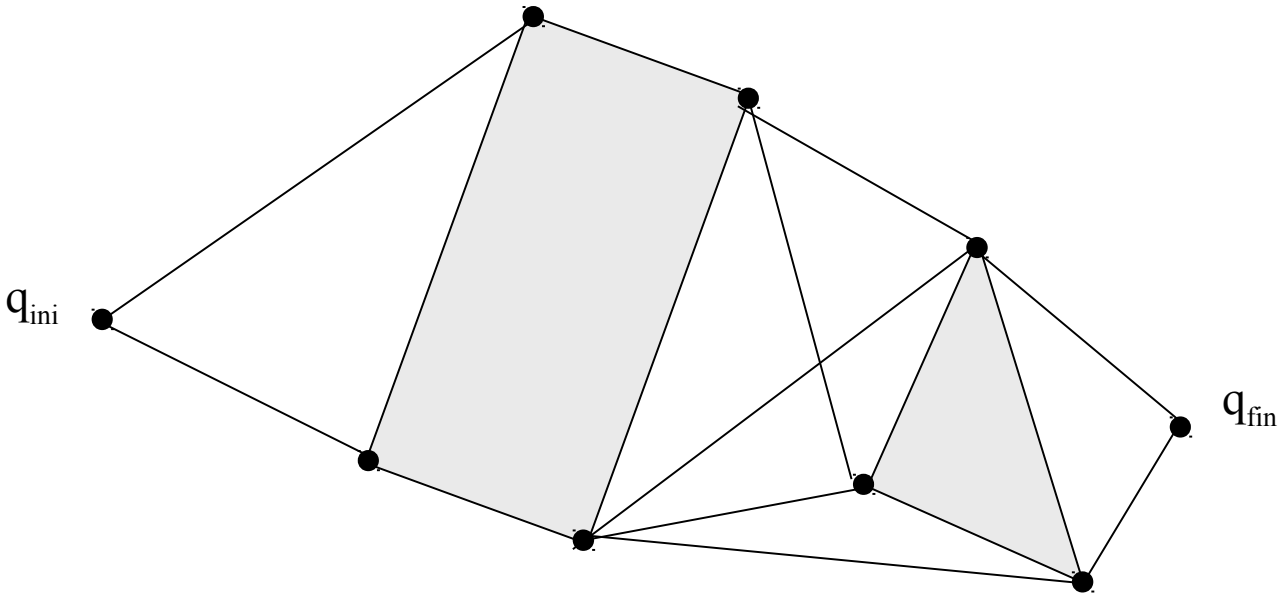
- Os Nós de G são q_{ini} e q_{fin} e os vértices de CB.
- Dois nós de G são conexos por um arco se e somente se o segmento que os une é um lado de CB ou está contido inteiramente em C_L.

$\Rightarrow$ O Grafo de Visibilidade contém o caminho mais curto entre q_{ini} e q_{fin}.

Grafo de Visibilidade



Grafo de Visibilidade



Grafo de Visibilidade

Algoritmo de Planejamento:

1. Construir o Grafo G.
2. Buscar um caminho em G de q_{ini} até q_{fin}.
3. Se o caminho é encontrado, retorná-lo, se não, reportar falha.

Grafo de Visibilidade

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. Caso contrário, computar as interseções da do segmento (X, X') com os lados de CB. Caso for vazia, 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

- Técnicas de busca em grafos.
- G = (X, A), onde X = conjunto de n nós, A = conjunto de r arcos.
- G representado por listas de adjacências, (uma para cada nó).

Algoritmo A*

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

Algoritmo A* - Procedimento

- G explorado iterativamente a partir de N_{ini}.
- Para cada nó visitado, 1 ou mais caminhos são gerados a partir de N_{ini}, Só o de menor custo é salvo.
- Conjunto de caminhos gerados forma uma árvore T do subconjunto de G já explorado.
- T representada através de ponteiros apontando de cada nó visitado para os correspondente nó pai.
- Cada nó N, recebe um custo, estimativa do custo do caminho de custo mínimo passando por N:

Algoritmo A* - Procedimento

Função de Custo: $f(N) = g(N) + h(N)$

- $g(N)$ = custo 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}.

Exemplo,

- $h(N)$ =distância euclidiana ao alvo.
- $h(N) = 0$ (Método de Dijkstra).

Algoritmo A* - Procedimento

Estruturas de dados e funções utilizadas:

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

Algoritmo A* - Procedimento

Operações em lista utilizadas:

- 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;

Decomposição em Células Convexas

Princípio:

- Decomposição de C_L em regiões não superpostas cuja união é exatamente C_L (Decomposição Exata), ou uma aproximação conservadora de C_L (Decomposição Aproximada).
- Construção de um grafo de conectividade G que representa as relações de adjacência entre células.
- Busca de um Canal (seqüência de células adjacentes da célula K_{ini} à célula K_{fin}.
- Extração de um caminho entre q_{ini} e q_{fin} no canal.

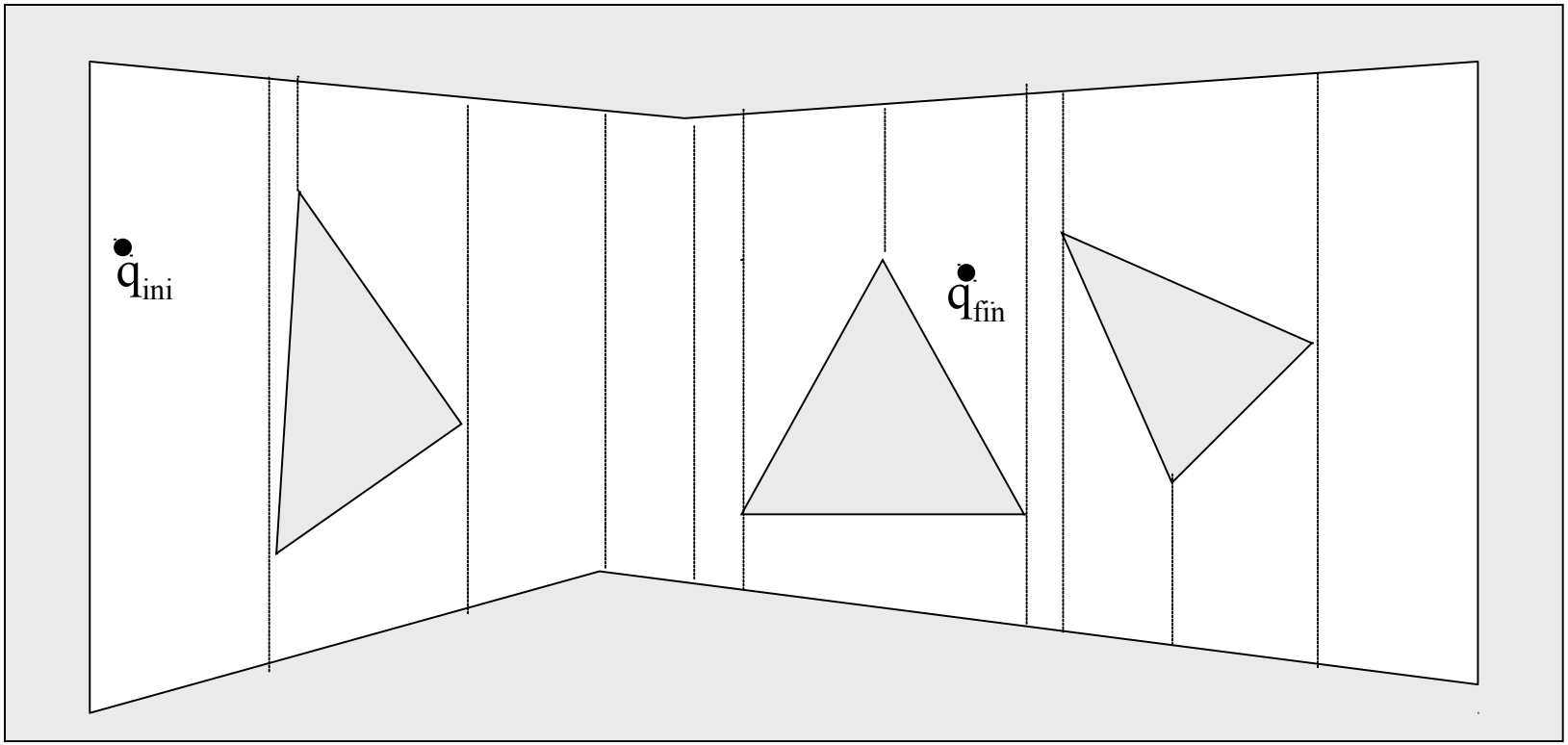
Decomposição Trapezoidal

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

Decomposição Trapezoidal

- Varrer C_L com uma linha reta vertical.
- Quando um vértice X de CB é encontrado, até dois segmentos verticais, são criados em C_L de modo a conectar X aos lados de CB 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.

Decomposição Trapezoidal



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 (compartilham um trecho de um segmento vertical). Como os lados adjacentes são verticais, basta testar as ordenadas de vértices de lados com a mesma abscissa.

Decomposição Trapezoidal

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

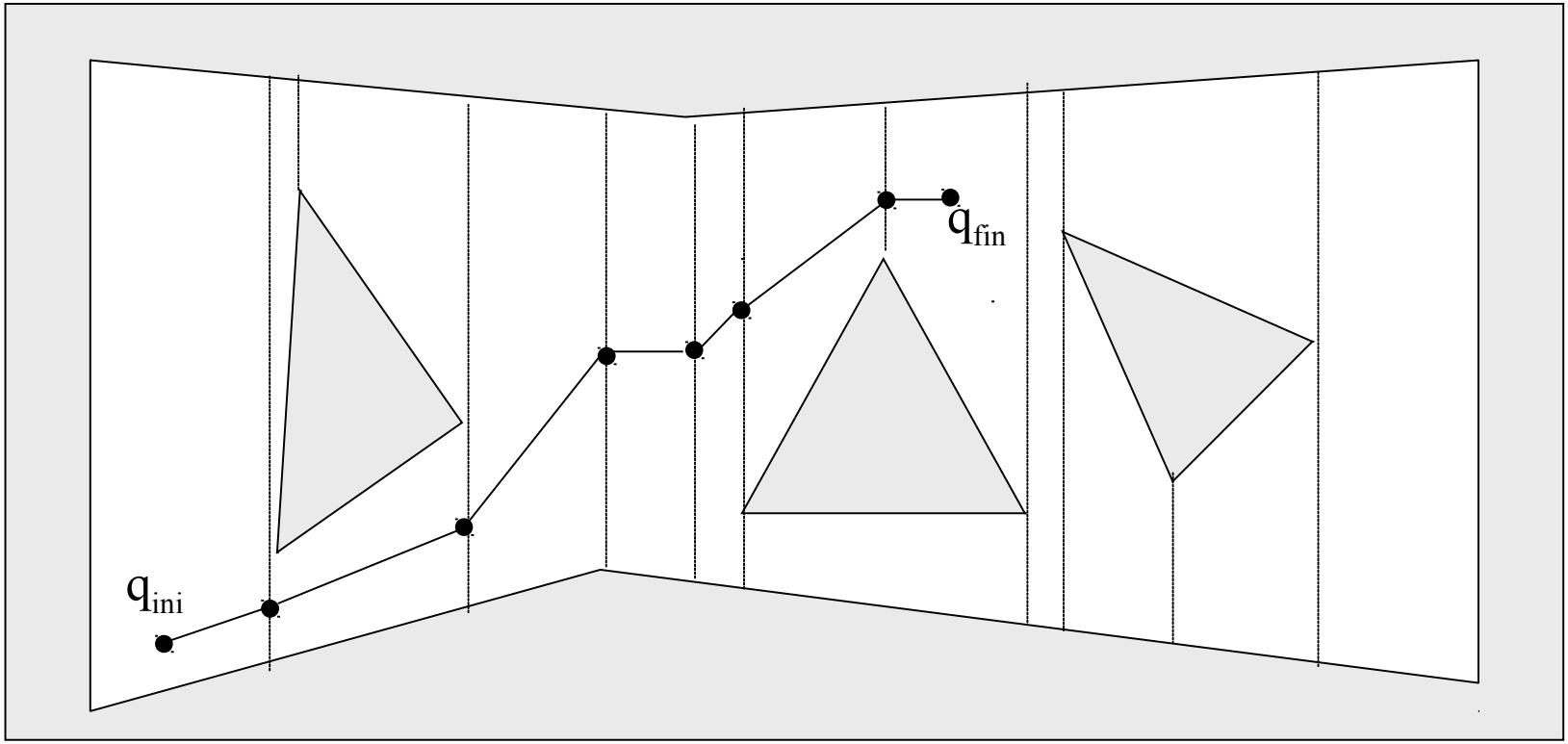
- Através do algoritmo A*, ponderando devidamente os arcos do grafo de conectividade.
- O arco entre os nós K_i e K_{i+1} pode ser ponderado, pela soma das distâncias que unem seus centróides C_i e C_{i+1} ao ponto central do segmento comum β_i.
- No caso de K_{ini} e K_{fin}, em lugar dos seus centróides, pode-se utilizar q_{ini} e q_{fin}.

Decomposição Trapezoidal

Extração de um caminho entre q_{ini} e q_{fin} no canal:

- Dados os limites $\beta_i = \partial k_i \cap \partial k_{i+1}$, entre 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}.
- Alternativa mais simples: 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.

Decomposição Trapezoidal

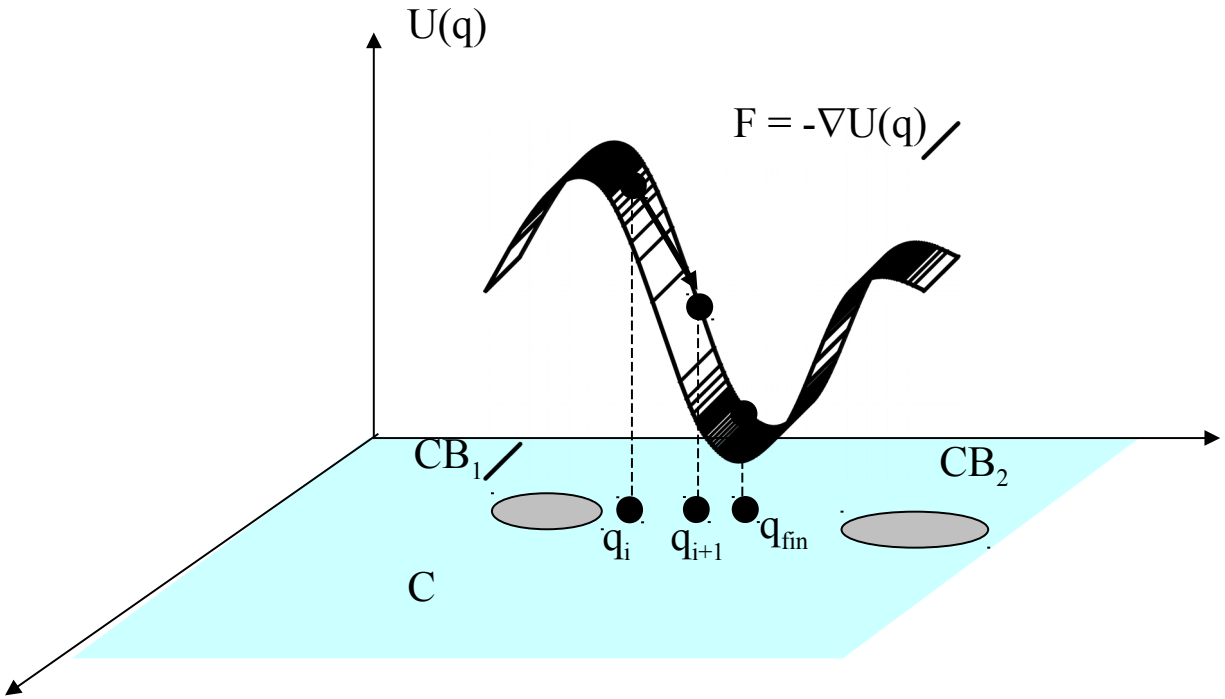


Campos de Potencial

Princípio:

- Considerar o robô, (ponto em C), como uma partícula sob a influência de um campo de potencial artificial **U**, cujas variações locais refletem a estrutura do espaço livre.
- Função de potencial = soma de potenciais repulsivos, (com influência local), que afastam o robô dos obstáculos e um potencial atrativo que atrai o robô em direção ao alvo.
- Iterativamente calcula-se a força $F(q) = -\nabla \mathbf{U}(q)$, que define direção e tamanho do deslocamento.

Campos de Potencial



Campos de Potencial

Características:

- Método local, desenvolvido originalmente para contorno de obstáculos percebidos *on-line* (modelo do ambiente desconhecido).
- Eficiente e rápido (tempo real).
- Método incompleto, (pode falhar na busca de um caminho, mesmo existindo um).
- Problemas com mínimos locais.

Campos de Potencial

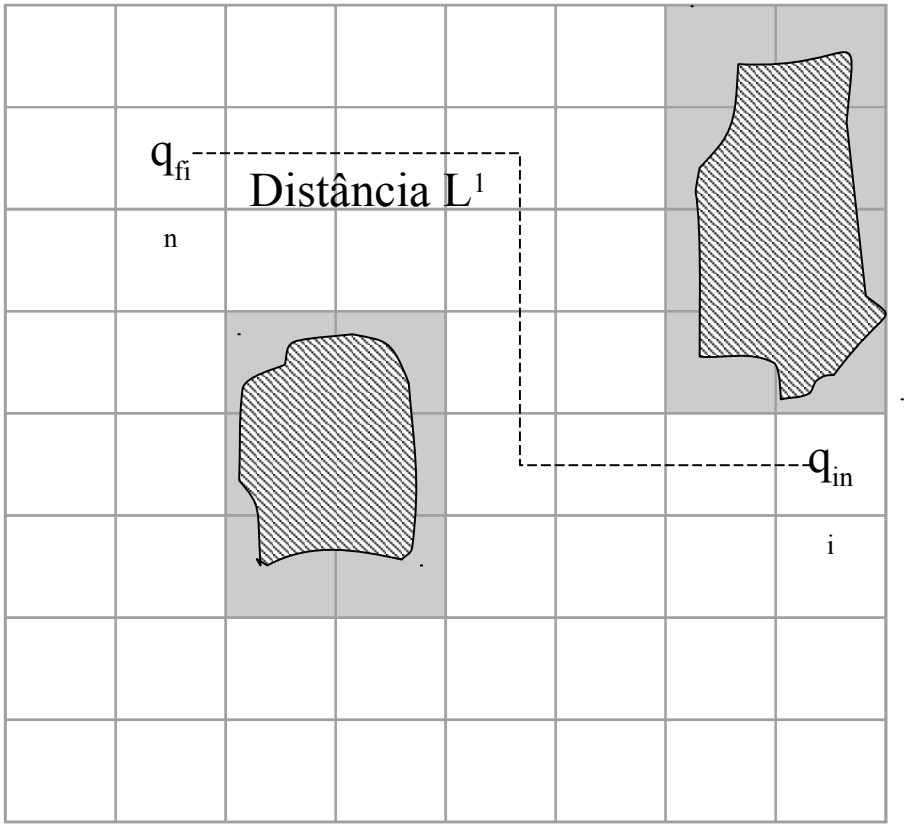
Abordagem para problema de mínimos locais:

- Incluir técnicas para escapar dos mínimos locais.
- Definir a função de potencial de modo a ter um ou uns poucos mínimos locais. $\Rightarrow$ tornar o método “Global”.

Função de Navegação Manhattan

- Método aproximado, aplicável a C = $\mathbf{R}^2$.
- Baseado em aproximação conservadora GR_L de C_L
- C discretizado em uma grade retangulóide GR.
- Assume-se que q_{ini} e q_{fin} pertencem a GR_L.
- Métrica = distância L^1 a q_{fin} (Função de Navegação Manhattan).

Função de Navegação Manhattan



GR

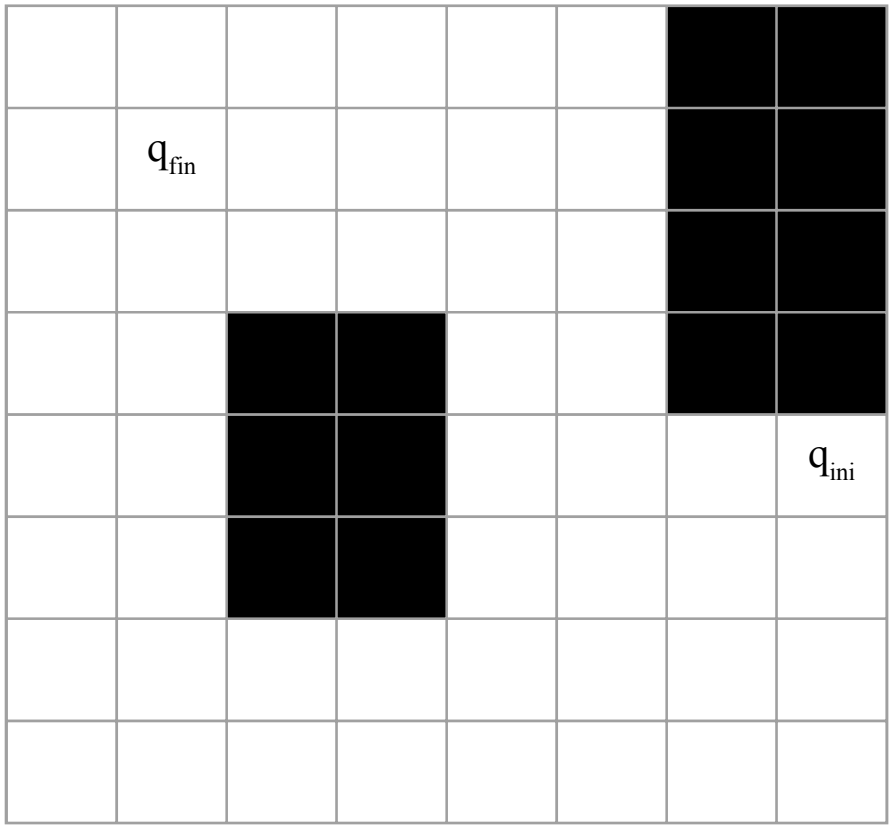
Função de Navegação Manhattan

- $U(q_{fin})$ é fixado em 0.
- Partindo de q_{fin} o potencial de cada célula 1-vizinha em GR_L não visitada é igual ao potencial da célula corrente mais um.
- Prossegue-se iterativamente até que o subconjunto de GR_L acessível a partir de q_{fin} tenha sido explorado completamente.

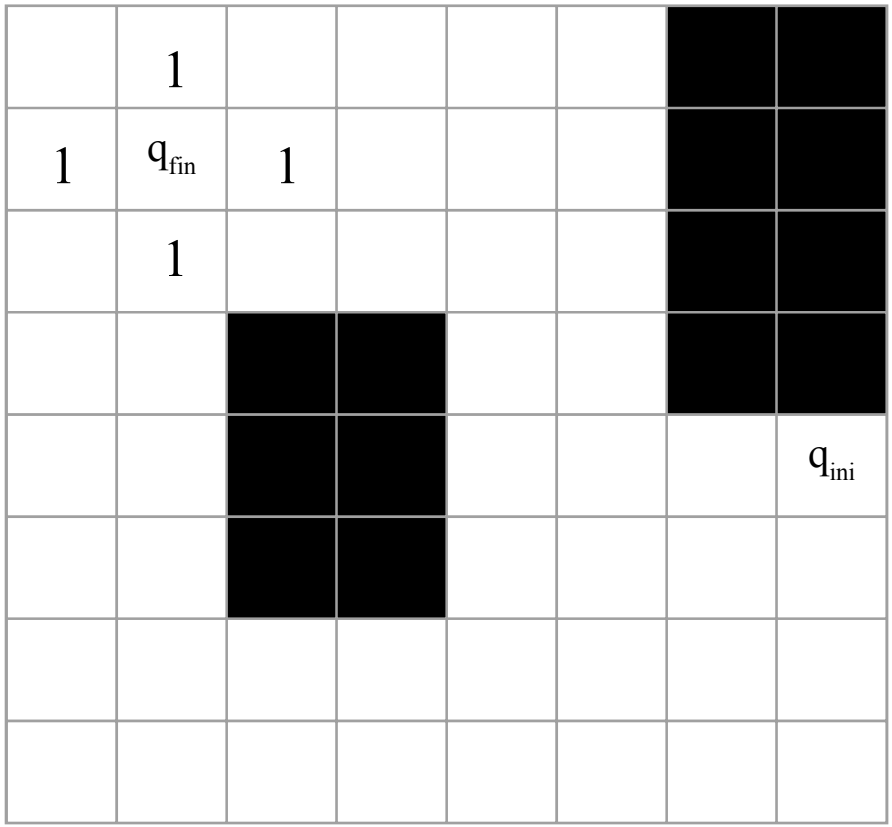
Função de Navegação Manhattan

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

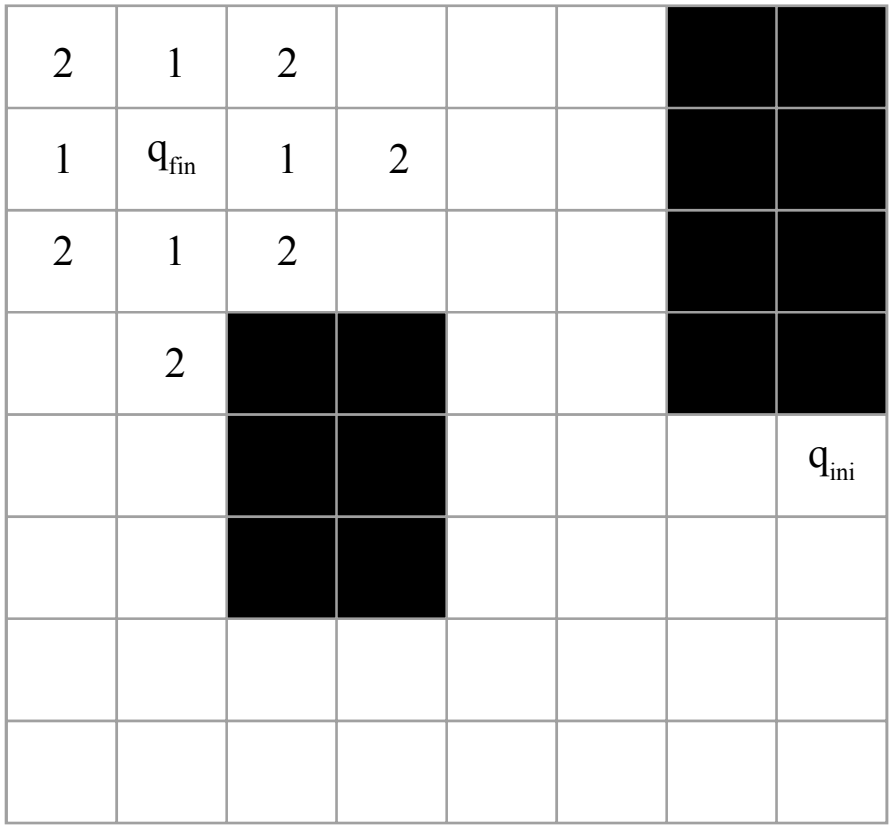
Função de Navegação Manhattan



Função de Navegação Manhattan



Função de Navegação Manhattan



Função de Navegação Manhattan

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

Função de Navegação Manhattan

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

Função de Navegação Manhattan

Observações:

- Gera potenciais em GR_L com um único mínimo em q_{fin}.
- 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.

Função de Navegação Manhattan

Observações:

- O caminho gerado é de comprimento mínimo (de acordo com a métrica L^1).
- Calcula U(q) somente no subconjunto de GR_L conexo a q_{fin}.
- Se $U(q_{ini})$ não foi computado $\Rightarrow$ não existe um caminho entre q_{ini} e q_{fin} em GR_L.

Função de Navegação Manhattan

Observações:

- Complexidade computacional é linear com o Nº de configurações em GR.
- Independe do Nº e forma dos C-Obstáculos.
- Eficiente para dimensão de C = 2 ou 3.
- Impraticável para dimensões maiores.