navegação

MAC318 - Introdução à Programação de Robôs Móveis

parte 1

navegação

- Dados:
  - conhecimento (parcial) sobre ambiente
    - localização inicial
    - meta
- Agir, baseado nos sensores e conhecimento interno, de maneira a atingir meta de forma eficiente

sub-tarefas

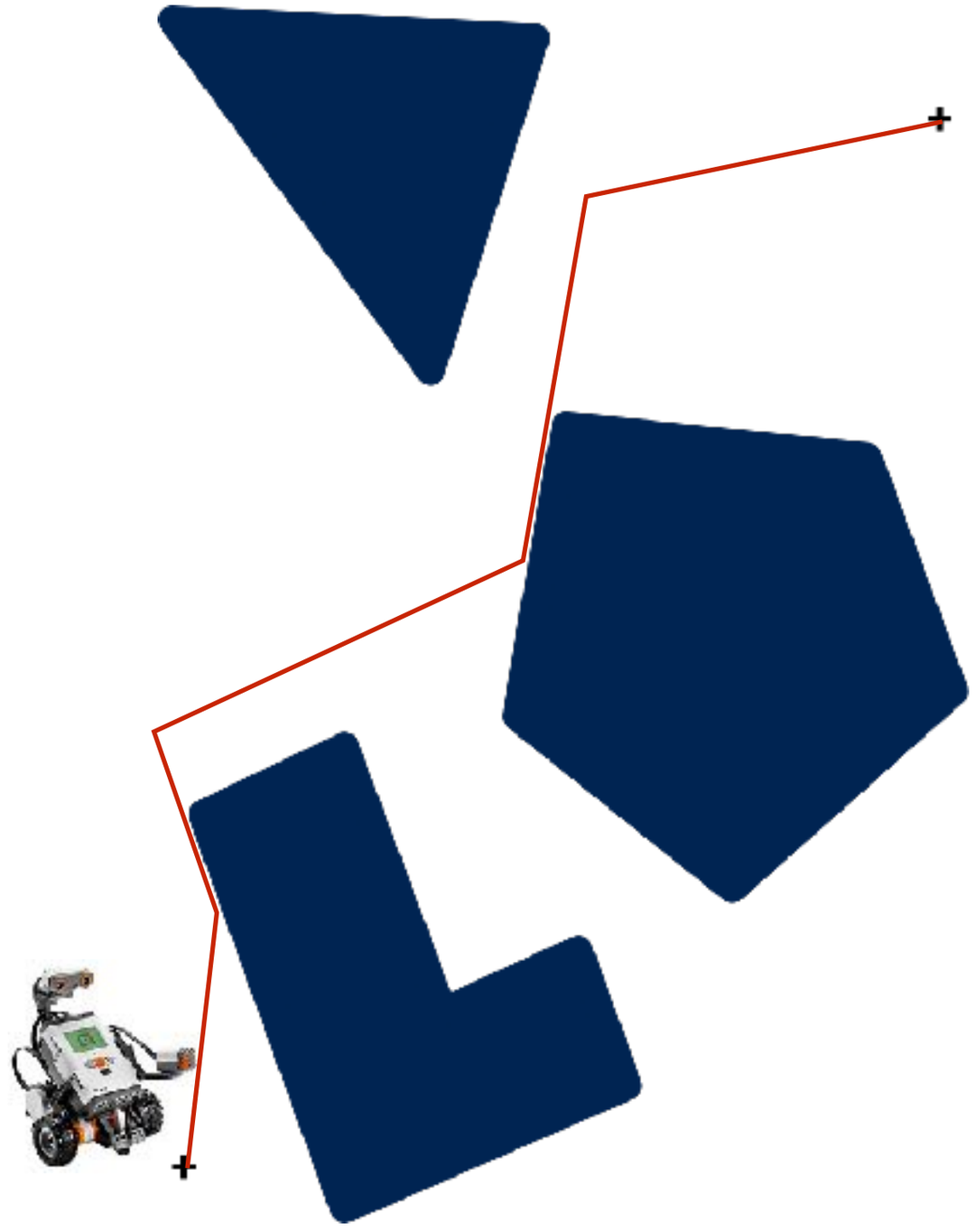
- **Localização**: determinar estado do robô
- **Locomoção**: determinar sinais de controle para levar robô a estado desejado
- **Planejamento de caminho**: determinar trajetória em mapa (modelo) que leve à meta
- **Desvio de obstáculos**: alterar localmente trajetória de forma a evitar colisões

localização

- Vamos assumir que odometria é perfeita

locomoção

- Problema de controle
  - Controlador PID
- Simplificação:
  - trajetórias são compostas de segmentos de retas entre pontos razoavelmente distantes
  - Pontos intermediários são conhecidos como “waypoints” ou pontos de rota



lejOS Navigator

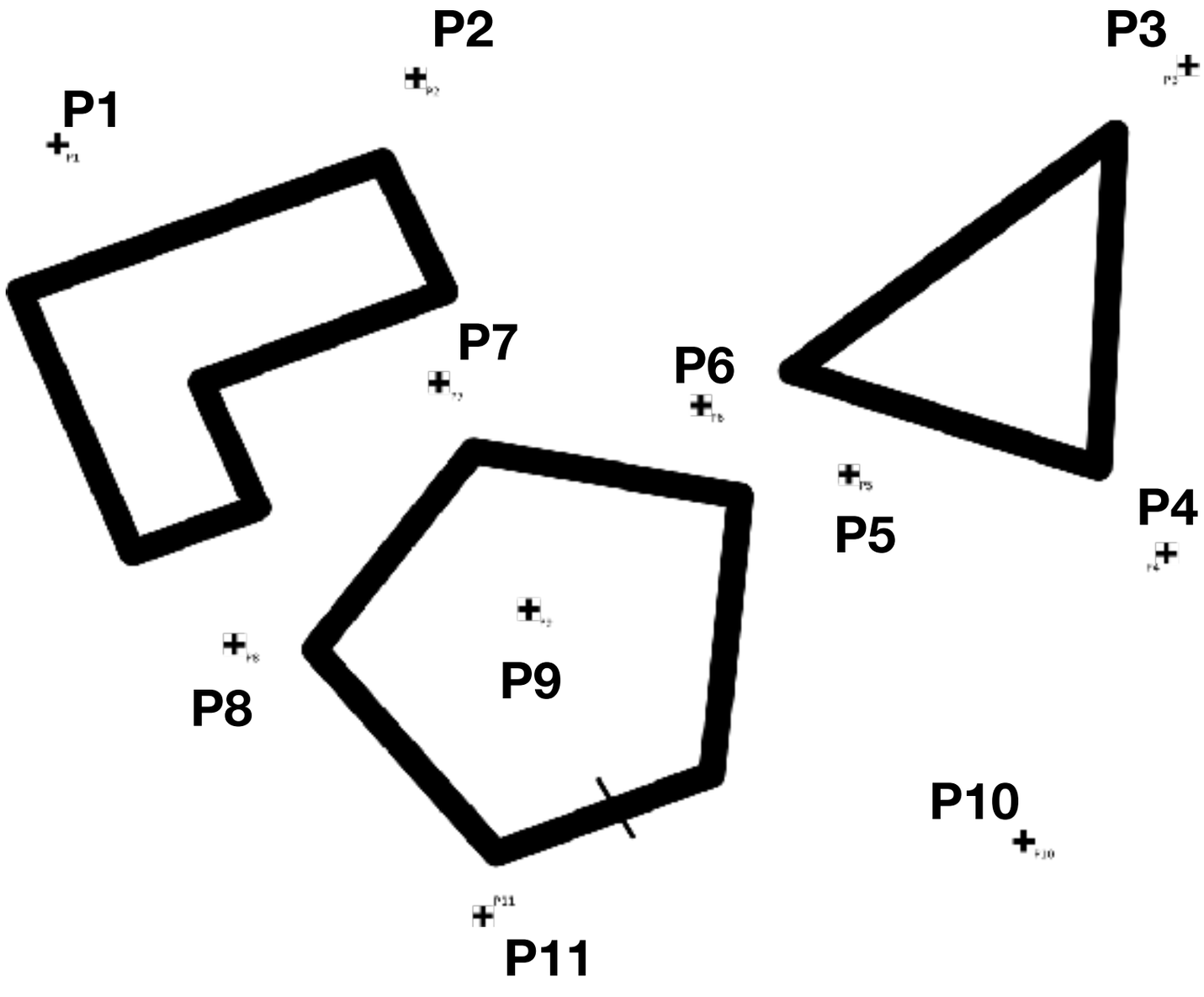
```
DifferentialPilot robot = new DifferentialPilot(5.6f, 11.2f, Motor.C, Motor.B, true);
navigator = new Navigator(robot);
navigator.addWaypoint(50.0f, 50.0f); // adiciona waypoint (usa mesma unidade
navigator.addWaypoint(0.0f, 80.0f);   //  usada para definir dimensões no
navigator.addWaypoint(-80.0f, 0.0f);  //  piloto — cm, nesse caso)
navigator.followPath();                // percorre pontos
navigator.waitForStop();               // bloqueia execução
```

http://www.lejos.org/nxt/nxj/api/lejos/robotics/navigation/Navigator.html

Projeto 5: Parte A

- Baixe arquivos de comunicação PC-Brick do PACA
  - MasterNav.java e SlaveNav.java
- Descreva 3 trajetórias de P1 a P10 usando segmentos de retas
  - Opcional: desenhe trajetórias no mapa
- Faça o robô executar as trajetórias e meça distância final à meta em cada uma

waypoints



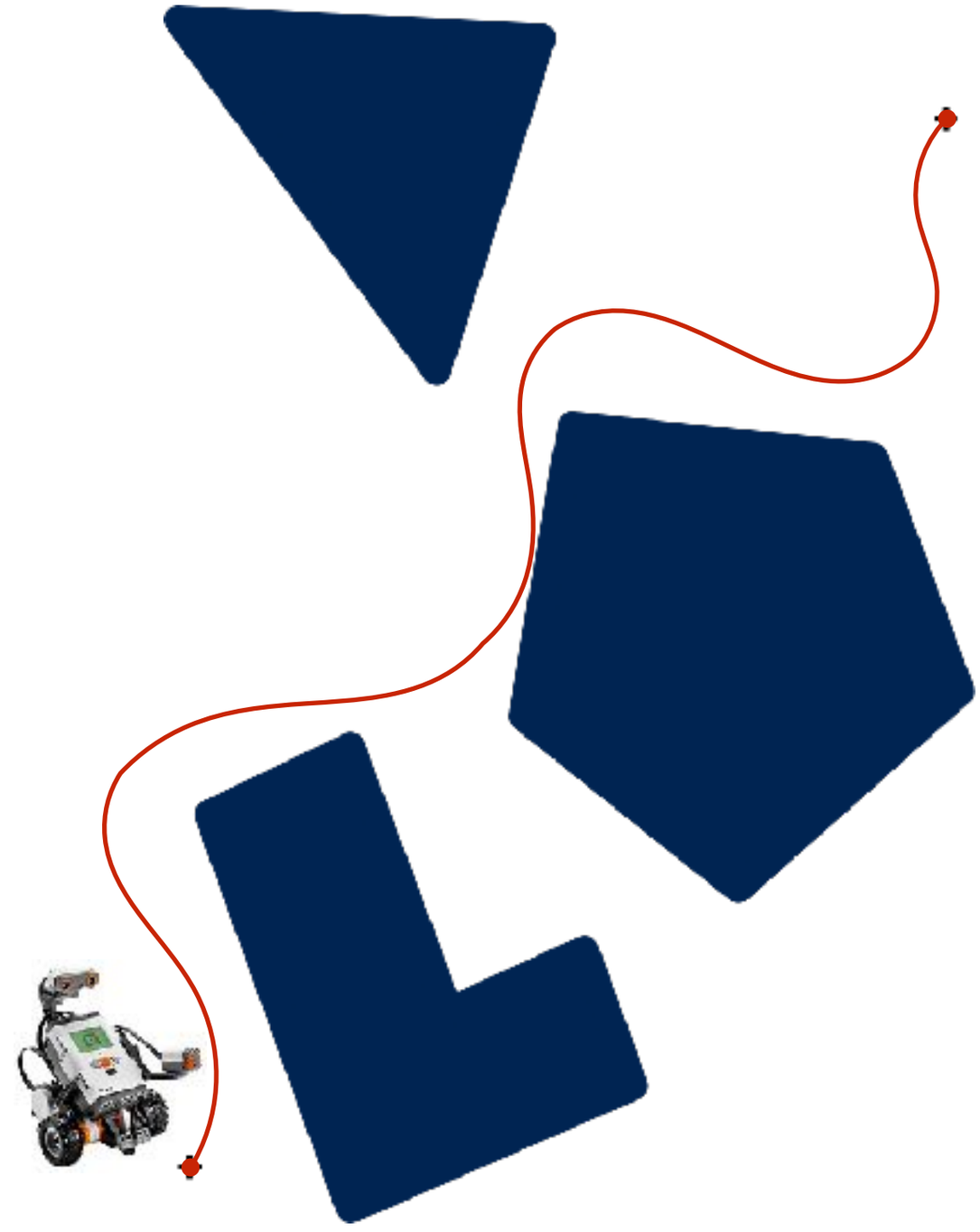

```
import lejos.geom.*;

Point[] points = {
  new Point(9.4,73.7),      /* P1 */
  new Point(42.8,80.0),     /* P2 */
  new Point(114.7,81.1),    /* P3 */
  new Point(112.7,35.5),    /* P4 */
  new Point(83.1,43.0),     /* P5 */
  new Point(69.3,49.4),     /* P6 */
  new Point(44.9,51.5),     /* P7 */
  new Point(26.0,27.1),     /* P8 */
  new Point(53.4,30.3),     /* P9 */
  new Point(99.4,8.8),      /* P10 */
  new Point(48.9,1.8)       /* P11 */
}; // em cm
```

planejamento de caminho

- Espaço de configurações C
  - Conjunto de estados do robô
  - Robô diferencial móvel no plano:
    - configurações de cada roda (θ_e,θ_d)
    - pose do robô (x,y,θ)
    - robô ponto (x,y)

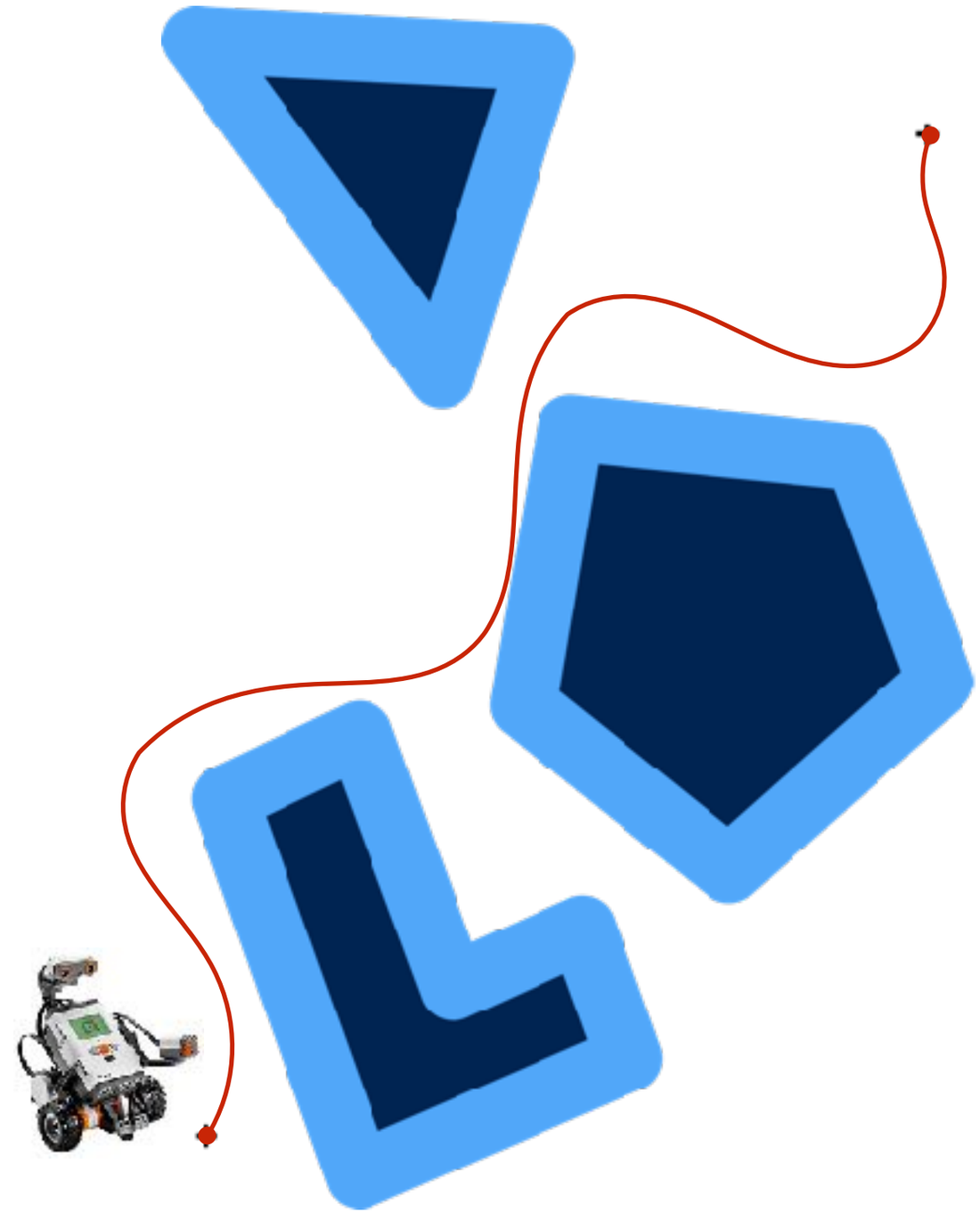
planejamento de caminho

- Espaço de configurações obstruídas O
  - Conjunto de estados inválidos para robô
- Espaço livre: F = C - O
- Rota ou caminho
  - π(t) ↦ (x,y) “bem-comportada”
  - π(0) = início, π(1) = meta



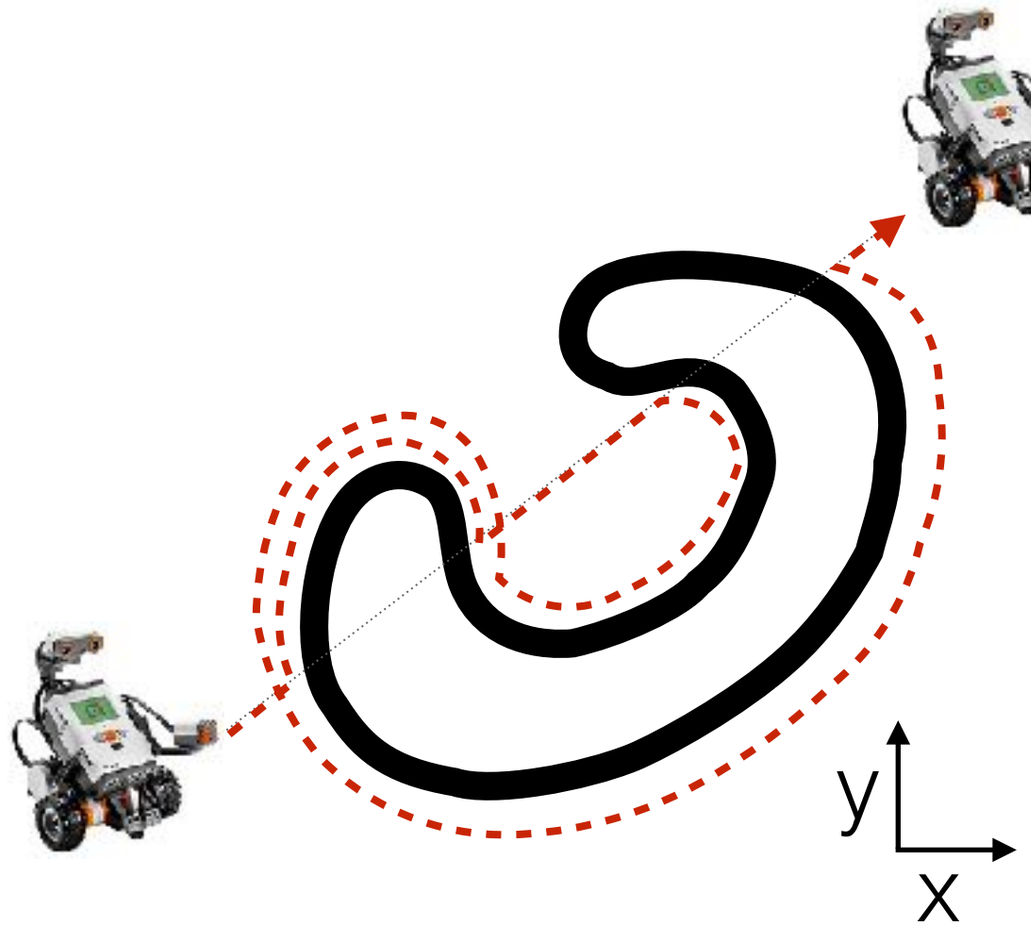
planejamento de caminho

- Robô pontual
- Dilatação de obstáculos
- Evita colisão devido à aproximação por ponto
- Altera conectividade do espaço
- considera robô com área (círculo ou quadrado)



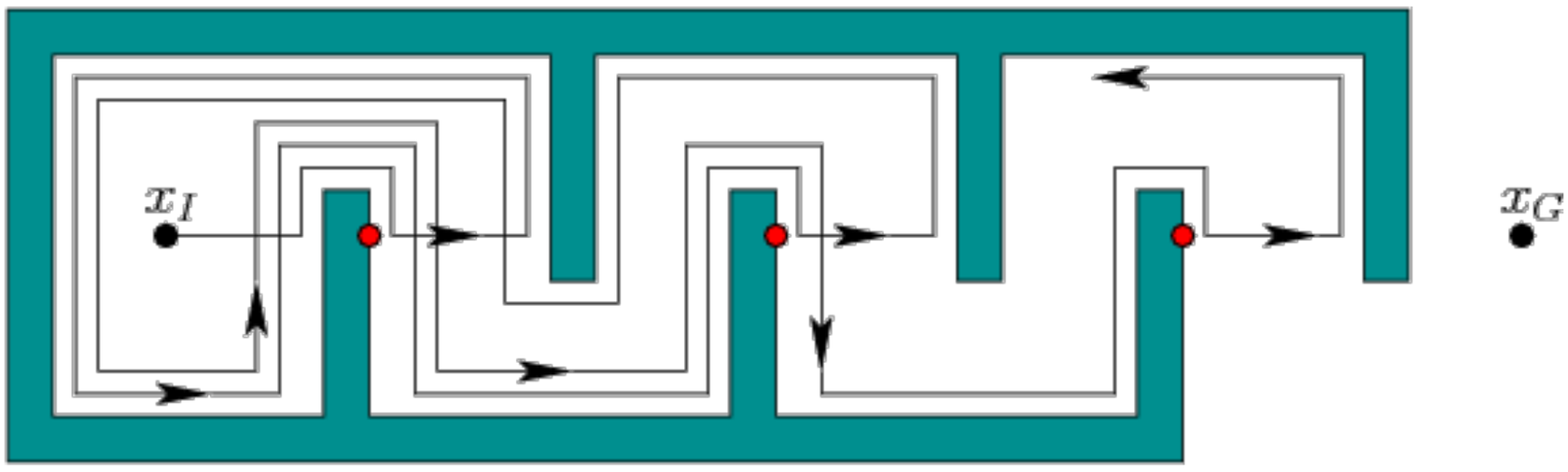
planejamento de caminho

- baseado em comportamento
  - “ambiente é o melhor modelo”
  - programação específica
  - subótima
- baseado em modelo
  - modelo do ambiente
  - programação genérica
  - portável



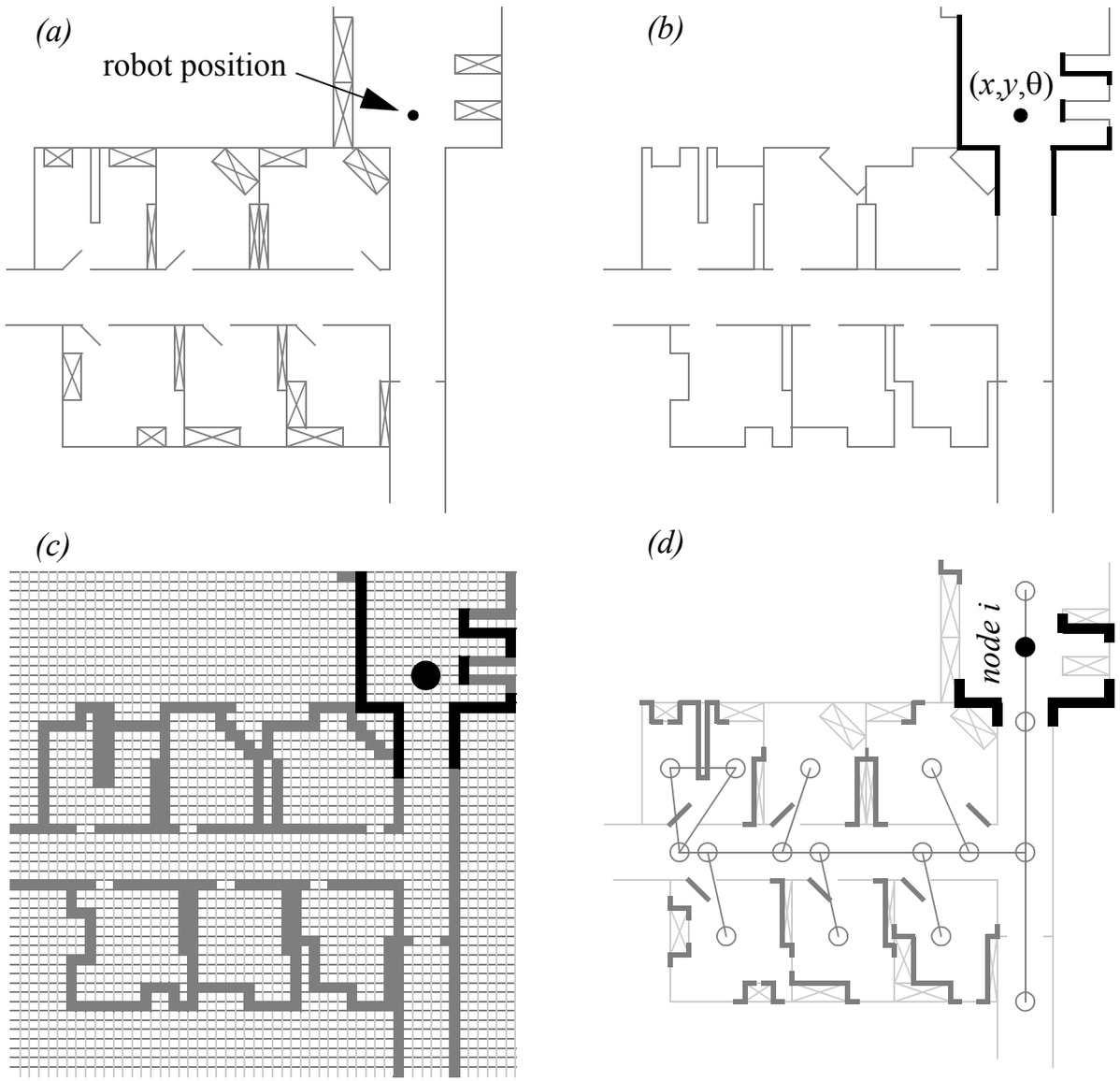
planejamento de caminho baseado em comportamento

- Problemas com algoritmos puramente reativos (ex. Bug2)



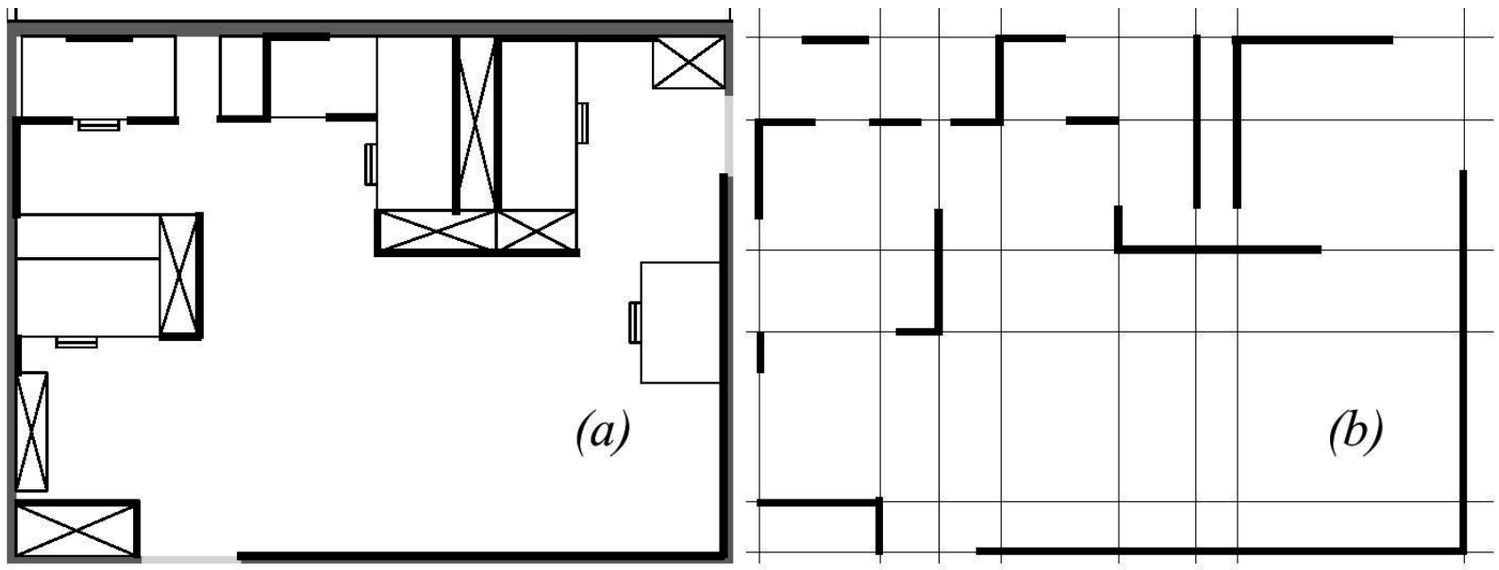
representação de mapas

- adequado à tarefa, à capacidade de percepção e de atuação do robô
- mapas métricos
  - mapas geométricos
  - decomposição em células
- mapas topológicos/rotas
- mapas sensoriais/ características



mapas de linhas

- representam segmentos de linhas
- não diferenciam objetos, texturas, cor etc
- eficiente e adequado para sensores de distância/contato
- podem ser anotados com informações adicionais

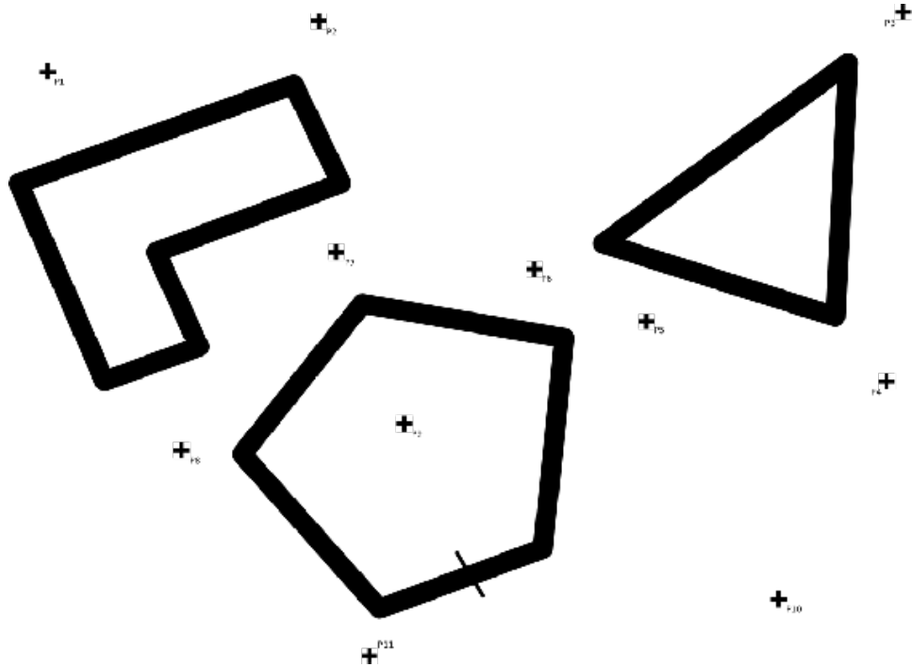


mapas de linhas

- No leJOS mapas de linhas são representados pela classe LineMap
- Retângulo demarca limites máximos (classe Rectangle)
- Linhas representadas pela classe Line

```
Line[] lines = {  new Line(x11,y11,x12,y12),  new Line(x21,y21,x22,y22)  };
Rectangle bounds = new Rectangle(0, 0, largura, altura);
LineMap mymap = new LineMap(lines, bounds);
```

mapa de linhas

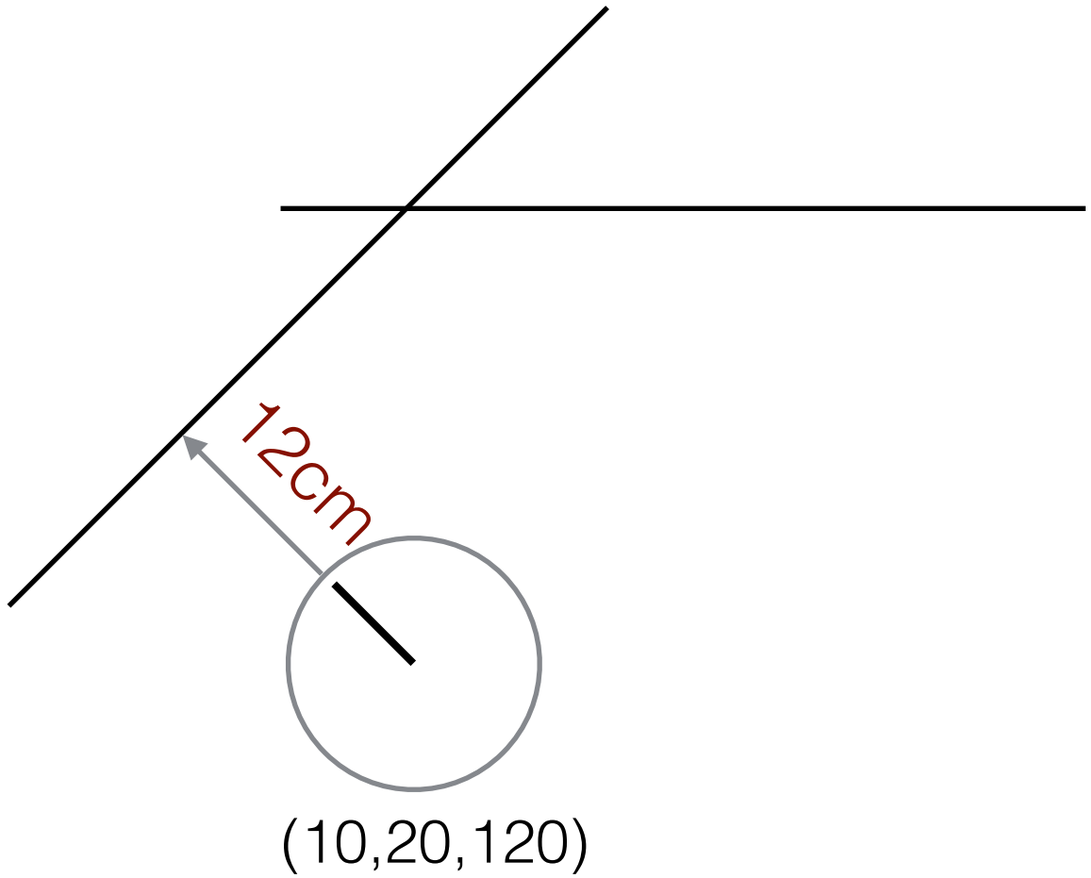


```
import lejos.geom.*;
import lejos.robotics.mapping.LineMap;

Line[] lines = {
  /* L-shape polygon */
  new Line(164,356,58,600),
  new Line(58,600,396,721),
  new Line(396,721,455,600),
  new Line(455,600,227,515),
  new Line(227,515,280,399),
  new Line(280,399,164,356),
  /* Triangle */
  new Line(778,526,1079,748),
  new Line(1079,748,1063,436),
  new Line(1063,436,778,526),
  /* Pentagon */
  new Line(503,76,333,267),
  new Line(333,267,481,452),
  new Line(481,452,730,409),
  new Line(730,409,704,150),
  new Line(704,150,503,76)
};
Rectangle bounds = new Rectangle(0, 0, 1189, 841);
LineMap mymap = new LineMap(lines, bounds);

Point[] points = {
  new Point(94,737),    /* P1 */
  new Point(428,800),   /* P2 */
  new Point(1147,811),  /* P3 */
  new Point(1127,355),  /* P4 */
  new Point(831,430),   /* P5 */
  new Point(693,494),   /* P6 */
  new Point(449,515),   /* P7 */
  new Point(260,271),   /* P8 */
  new Point(534,303),   /* P9 */
  new Point(994,88),    /* P10 */
  new Point(489,18)     /* P11 */
};
```

mapas de linhas

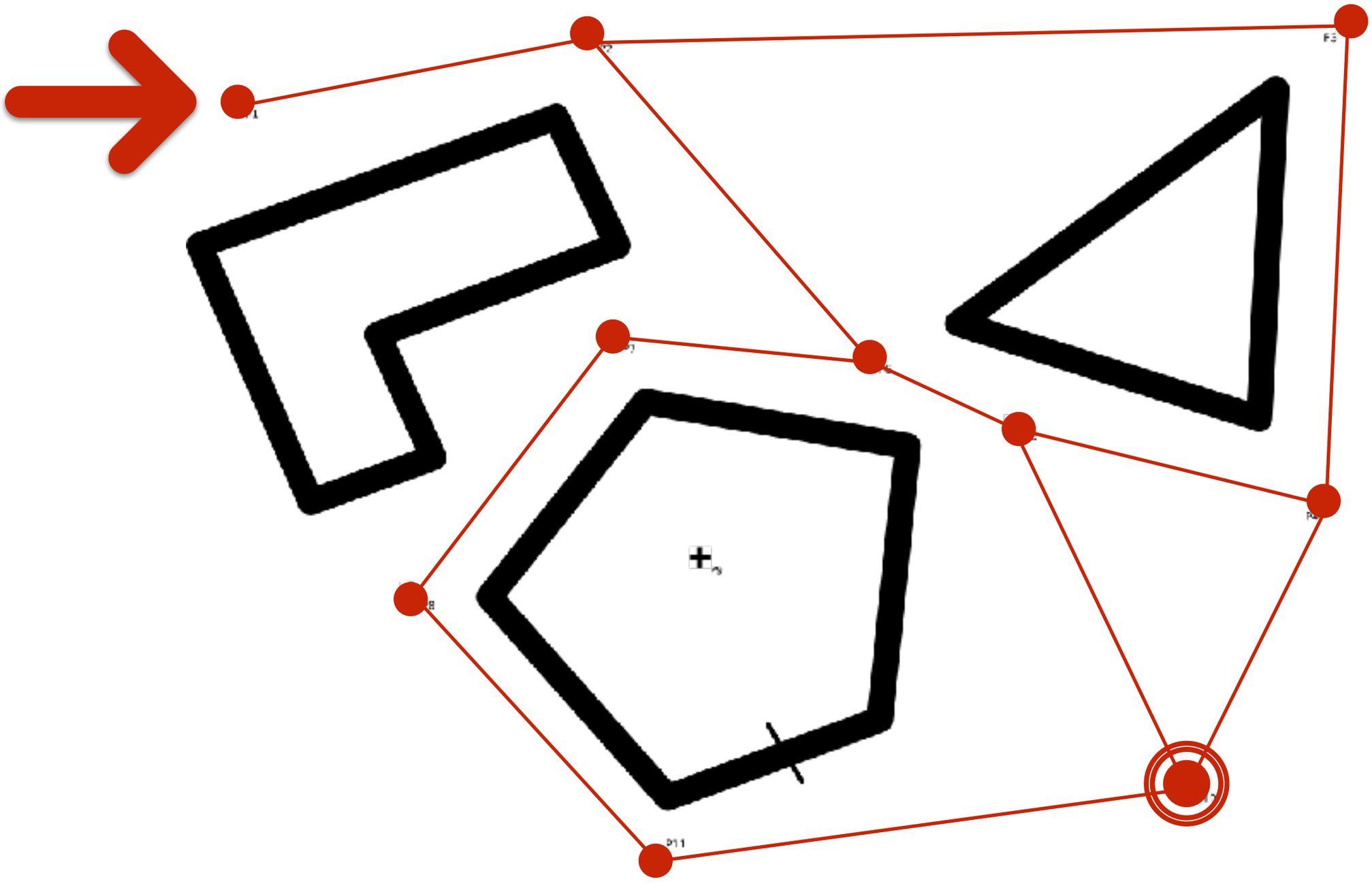
- Pose pode ser representada pela classe Pose

- Distância até objeto mais próximo em uma dada pose pode ser obtida pelo método range da classe LineMap

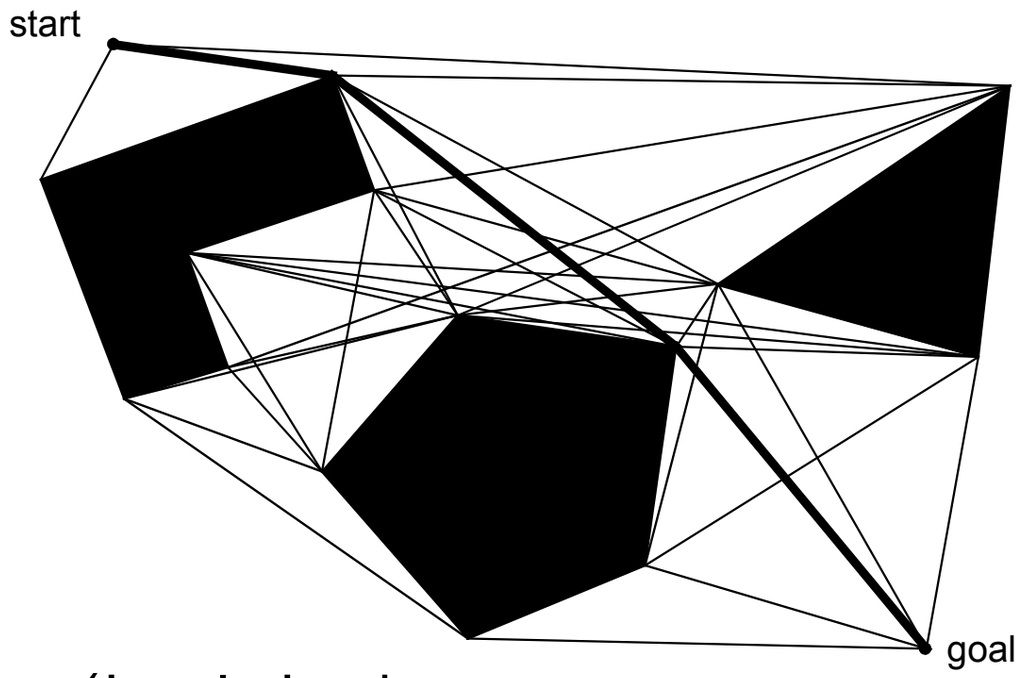



```
Pose mypose = (10, 20,120);
mymap.range(pose); // retorna 12
```

mapa topológico

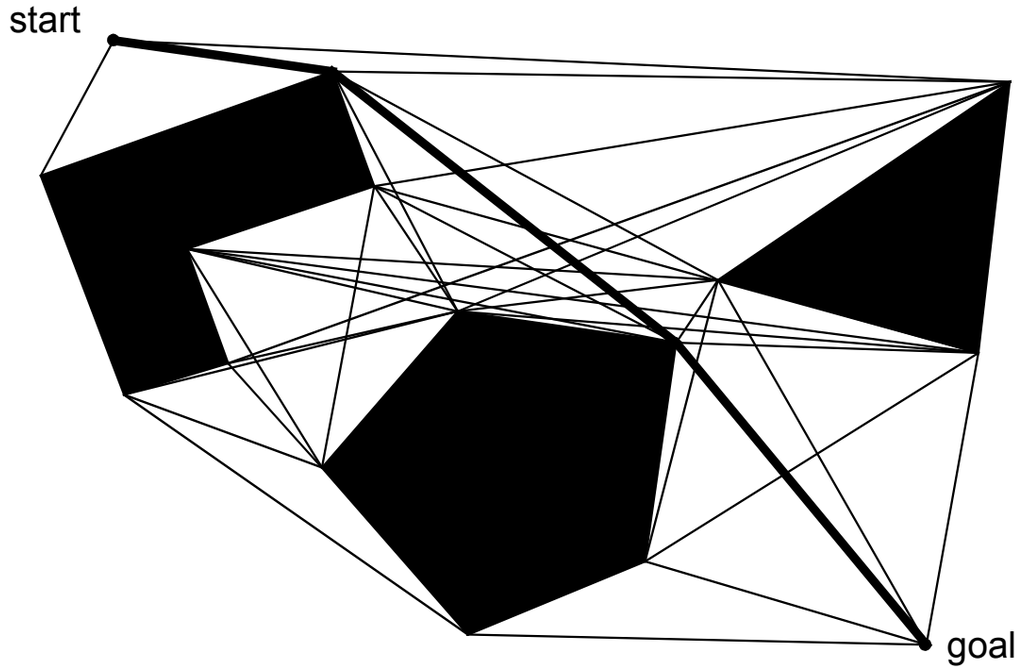


mapa de visibilidade



- Mapa de rotas
- Assume obstáculos são polígonos (incluindo pontos inicial e final)
- Pontos são conectados se podem se ‘ver’
  - caminho mais curto liga apenas vértices

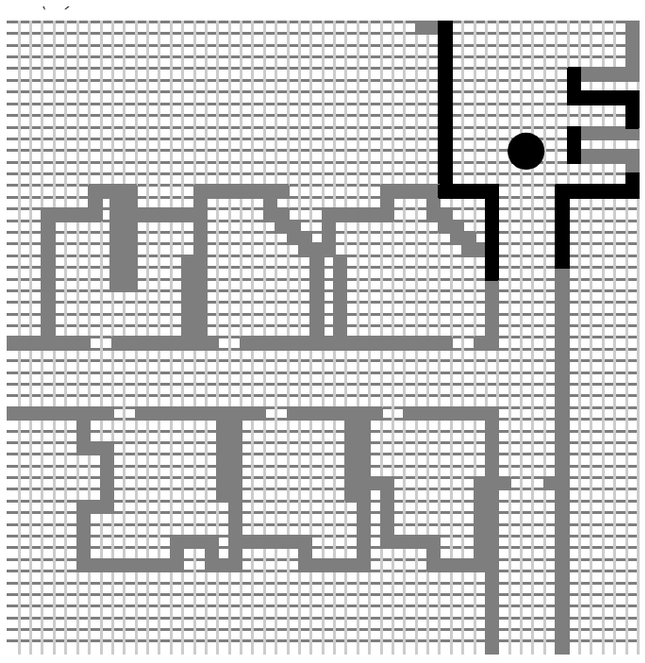
mapa de visibilidade



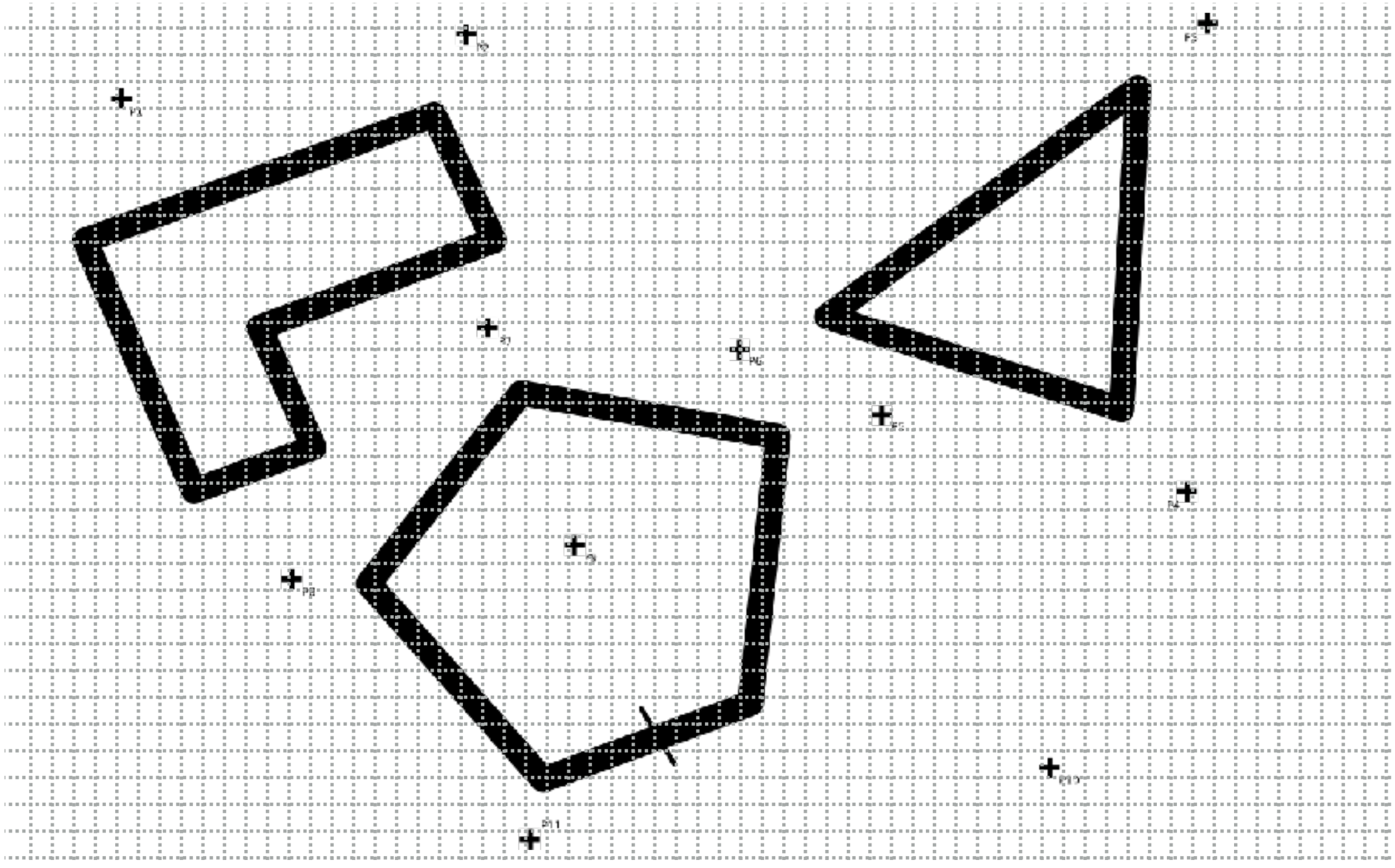
- Simples obtenção
- Número de nós e arestas depende de número e complexidade de polígonos
- Caminhos passam rente a obstáculos (não robustos)

mapa de ocupação

- Discretização do espaço em células
- Célula é **ocupada** ou **desocupada**
- Ações são mover-se à célula vizinha
- tamanho não depende de número de objetos ou sua complexidade
- não assume polígonos

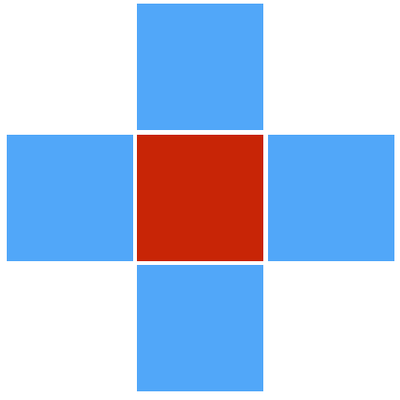


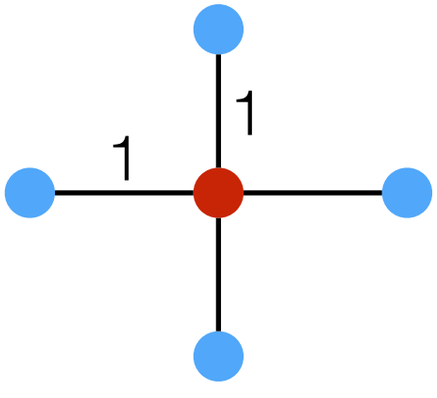
decomposição em células de tamanho fixo



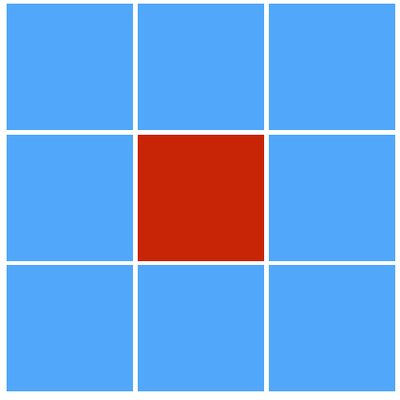
mapa métrico: decomposição em células uniforme

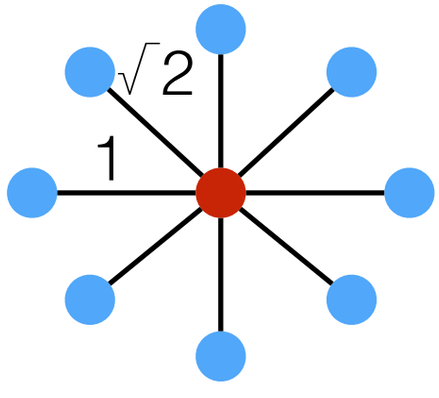
4-vizinhança





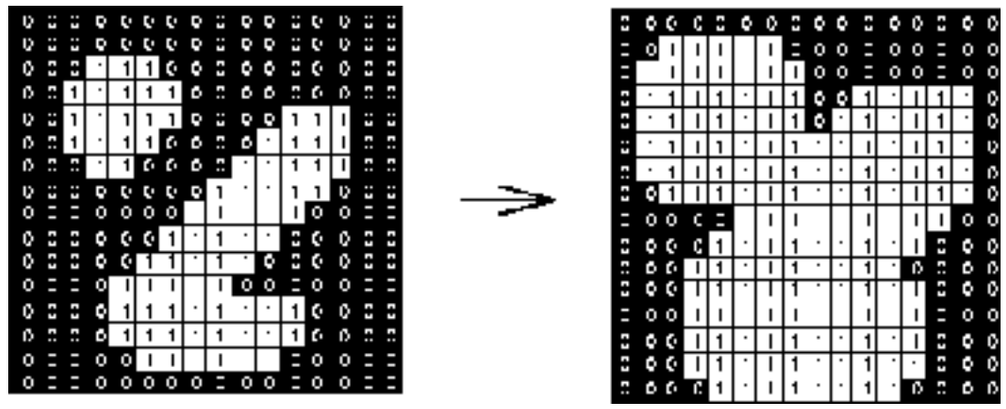
8-vizinhança





dilatação em mapas de ocupação

- Pode ser visto como processamento de imagem binária
  - dilatação/thickening

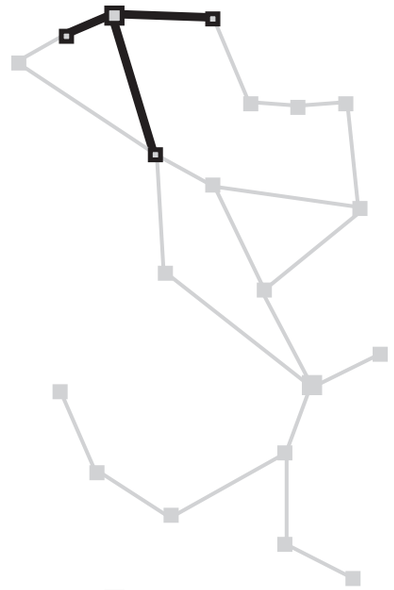


- Abordagem simples:
  - União das imagens obtidas deslocando-se matriz 1,2, …,(d-1)d pixels à direita/esquerda/acima/abaixo
- Abordagens mais complexas tentam recuperar caminhos fechados por dilatação

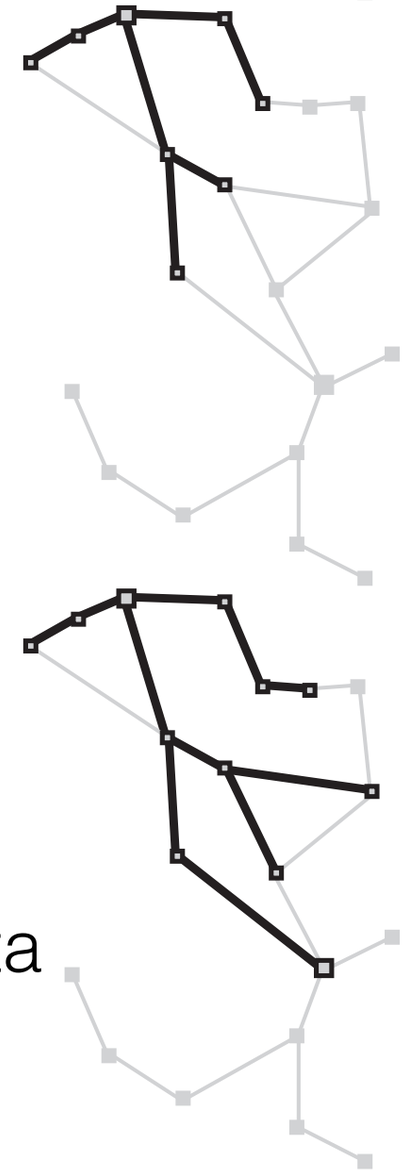
planejamento de caminho

- Busca heurística em grafo finito
  - discretização por rotas (topológico)
  - discretização por decomposição em células
  - discretização por amostras
- Campos potenciais

busca heurística

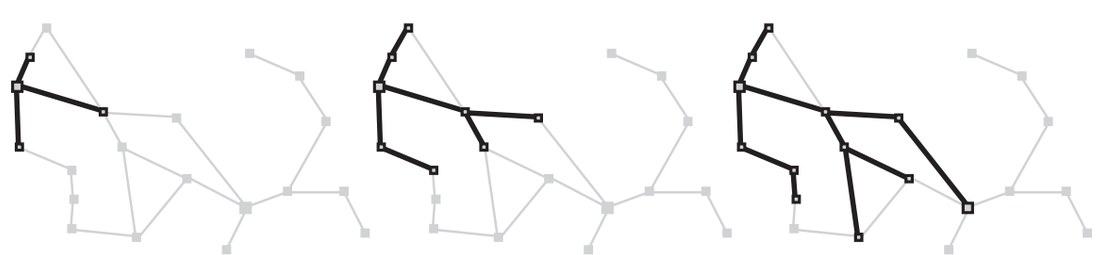


- Dados:
  - Conjunto de estados S e de ações A
  - Ações aplicáveis num estado: $A(s) \rightarrow A' \subseteq A$
  - Modelo de transição (arcos): $T(s,a) \rightarrow s'$
  - Função de custo: $g(s,a,T(s,a)) \rightarrow [-\infty,\infty]$
  - Estado inicial s_0 e estado meta s_m
- Objetivo: Encontrar sequência de ações que levam à meta através de trajeto menos custoso



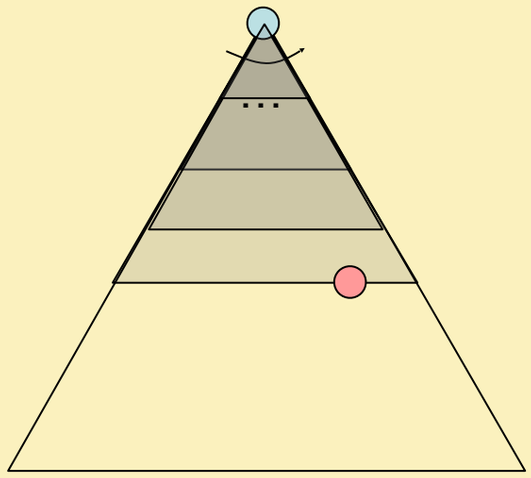
busca heurística

- Busca cega
  - usa apenas conhecimento sobre a especificação do problema
- Busca informada
  - usa conhecimento heurístico sobre domínio

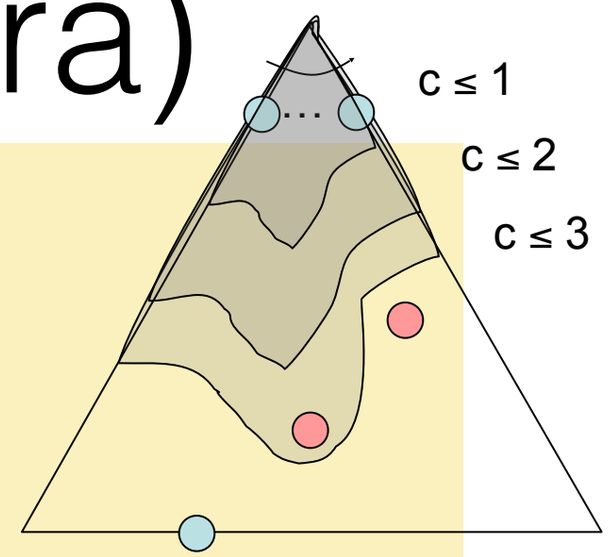


busca em largura

- Inicie uma *fila* com nó inicial
- Repita:
  - remova nó menos recente *n*
  - se algum sucessor de *n* for meta, retorne solução correspondente
  - adicione sucesores de n à *fila*



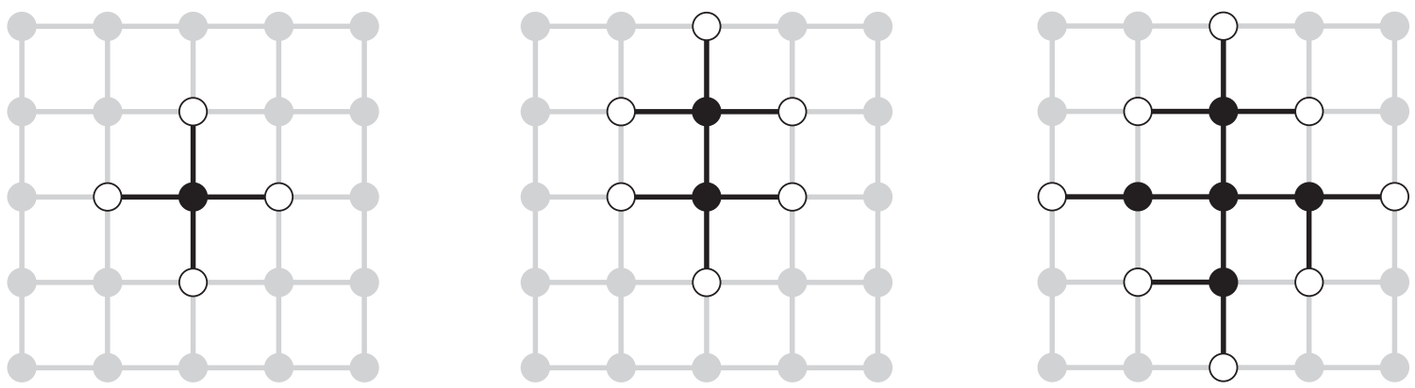
busca de custo uniforme (algoritmo de Dijkstra)



- Inicie *fila de prioridade* com nó inicial
- Repita:
  - selecione nó *n* não explorado com menor prioridade
  - acrescente *n* a conjunto *explorados* e adicione seus sucessores à *fila* com prioridade dada pelo custo do caminho do início até o estado correspondente
  - se algum sucessor de *n* for estado-meta, retorne solução correspondente

busca em largura com memória

- Manter conjunto explorados
- Evita visitar nós múltiplas vezes



frente de onda

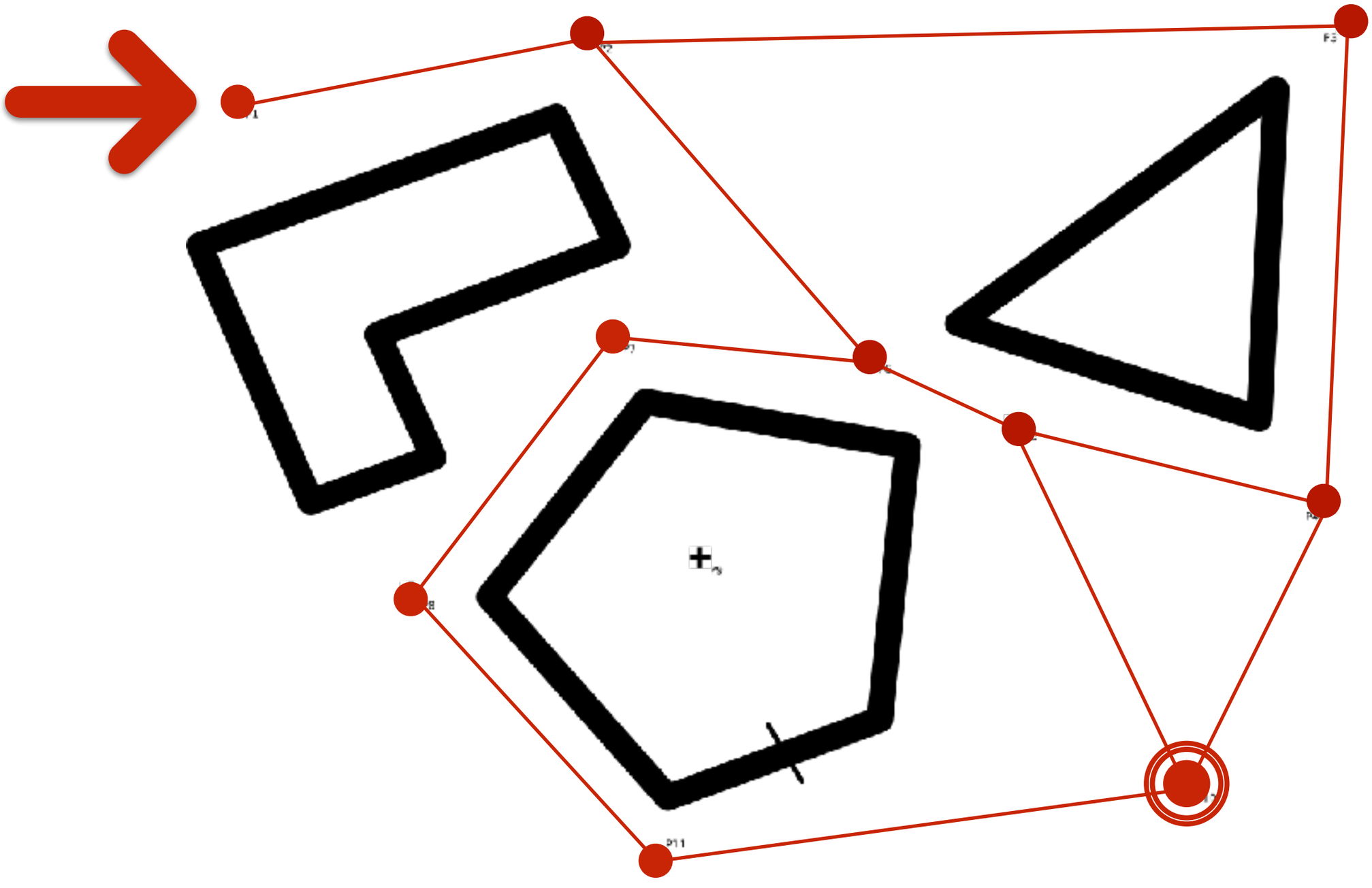
- Busca em largura a partir da meta
  - 4-vizinhança, custo fixo na busca
  - 8-vizinhança na volta

Pode ser implementado usando a matriz de ocupação: permite replanejamento

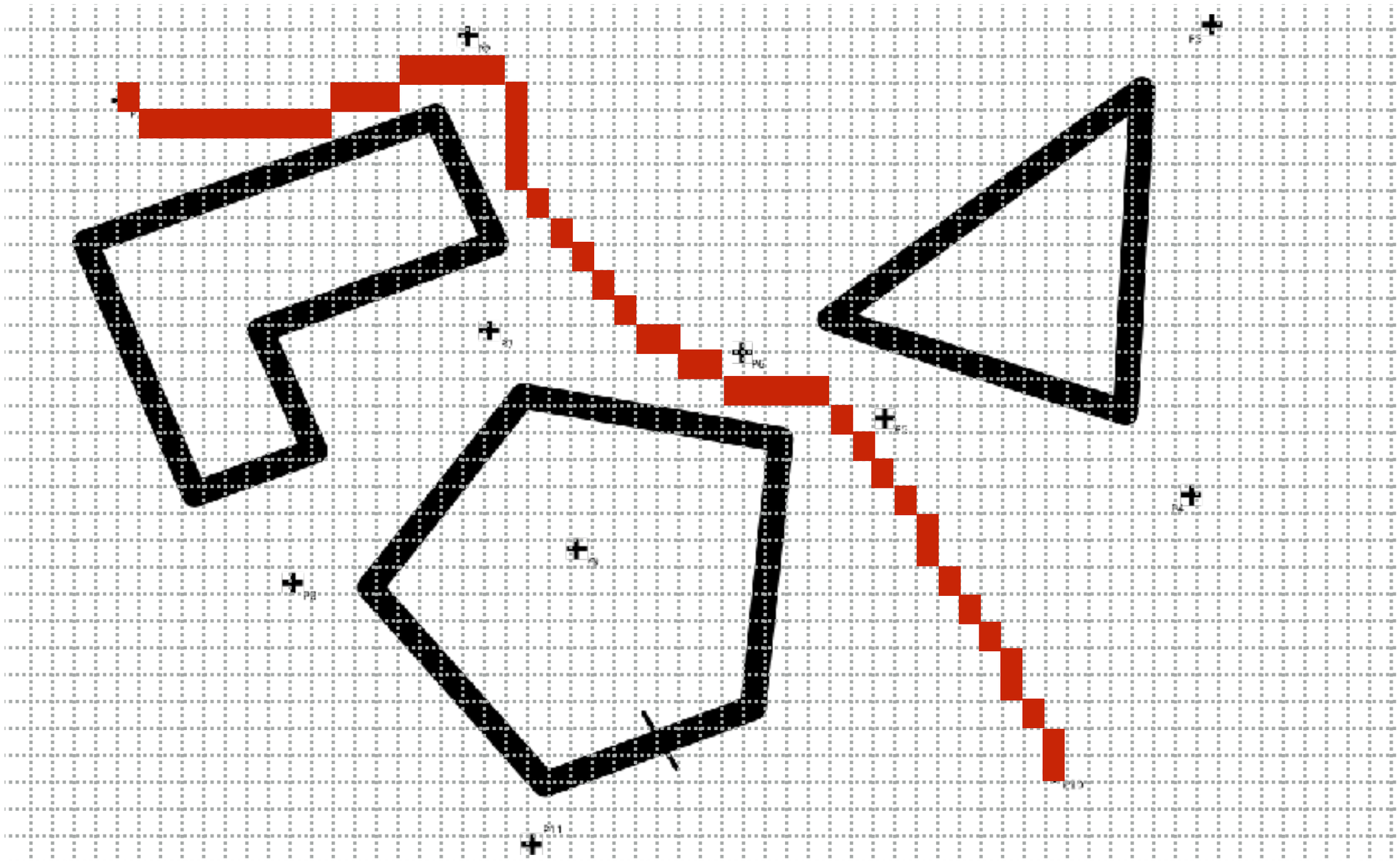
Projeto 5: Parte B

- Implemente o algoritmo de busca de custo uniforme (Dijkstra)
  - custo de cada ação é distância entre posições
- Utilize o algoritmo para encontrar trajetória levando robô de
  - P1 a P10
  - P11 a P1
- Faça o robô executar trajetórias e meça distância final à meta
  - planejamento é feito no computador (MasterNav.java) e executado pelo robô a partir da trajetória (SlaveNav.java)

mapa topológico



trajetórias em células



linearização

- Transformar trajetória em segmentos de reta conectados
  - cada segmento deve ter tamanho maior que comprimento mínimo pré-especificado
- Algoritmo guloso (exemplo):
  - inicie em ponto p
  - encontre ponto q mais distante de p tal que segmento pq não intersecta obstáculo
  - adicione segmento pq e descante pontos entre p e q
  - atualize p para q e repita

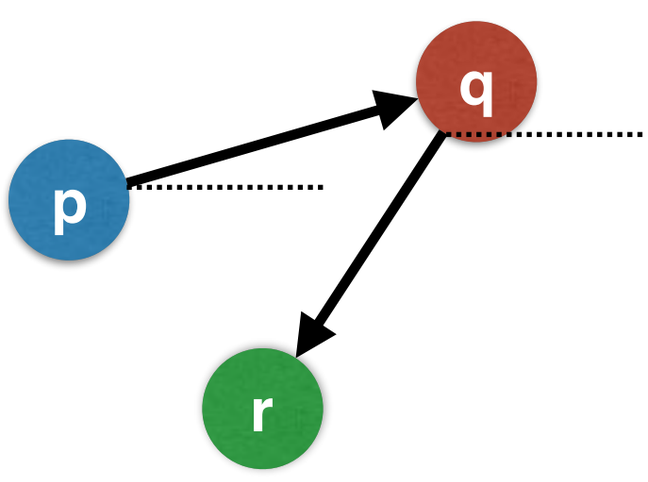
Projeto 5: Parte C

- Implemente o algoritmo de planejamento por frente de onda
  - discretize espaço em células de 5cm x 5cm
  - use “linearização” de trajetória por segmentos de retas
  - use dilatação por engrossamento (thickening)
- Faça o robô executar trajetória
  - meça distância à meta utilizando diferentes discretizações (número de células)

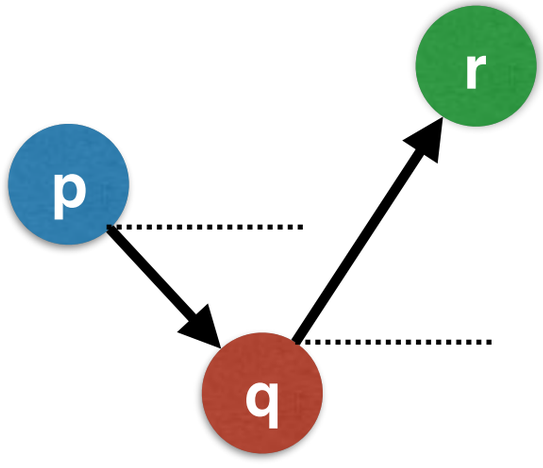
visibilidade de vértices

- Dois vértices ***a*** e ***b*** são visíveis se segmento ***ab*** não intersecta nenhum outro segmento de reta

- Orientação de 3 pontos:



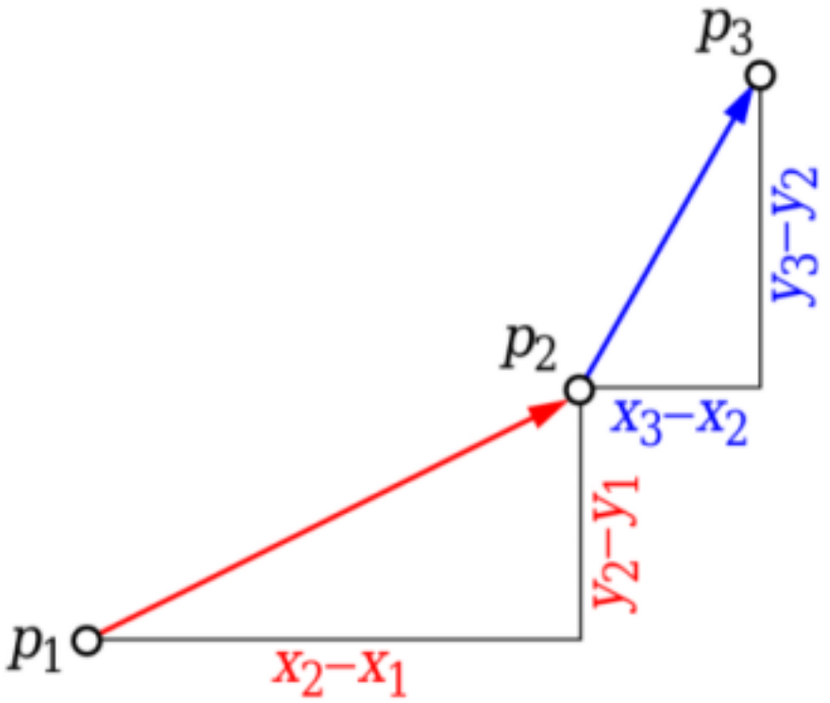
horário



anti-horário

colinear

orientação de pontos



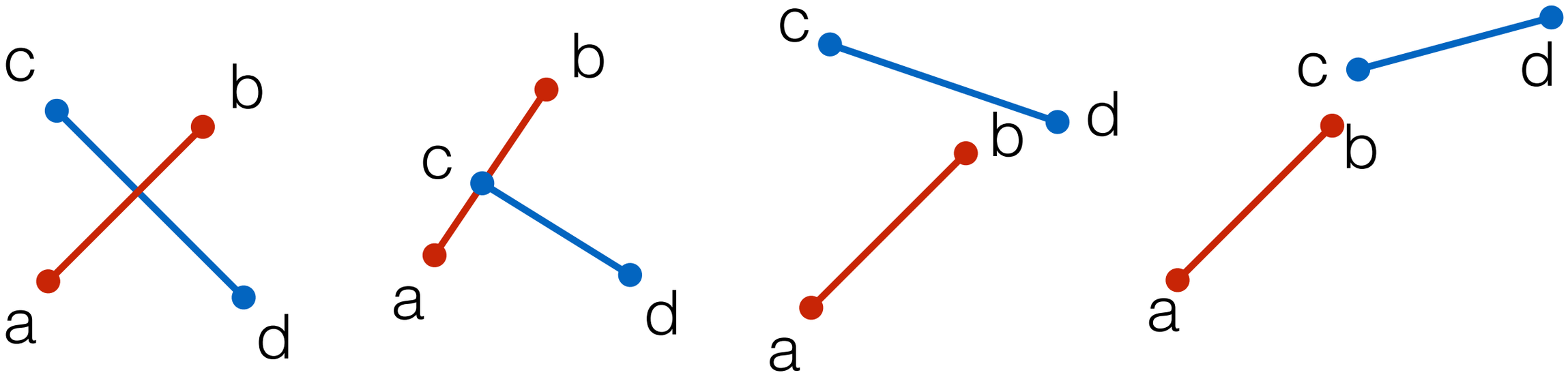
horário: $\tan(p_2\text{-}p_1) > \tan(p_3\text{-}p_2)$

antihorário: $\tan(p_2\text{-}p_1) < \tan(p_3\text{-}p_2)$

colinear: $\tan(p_2\text{-}p_1) = \tan(p_3\text{-}p_2)$

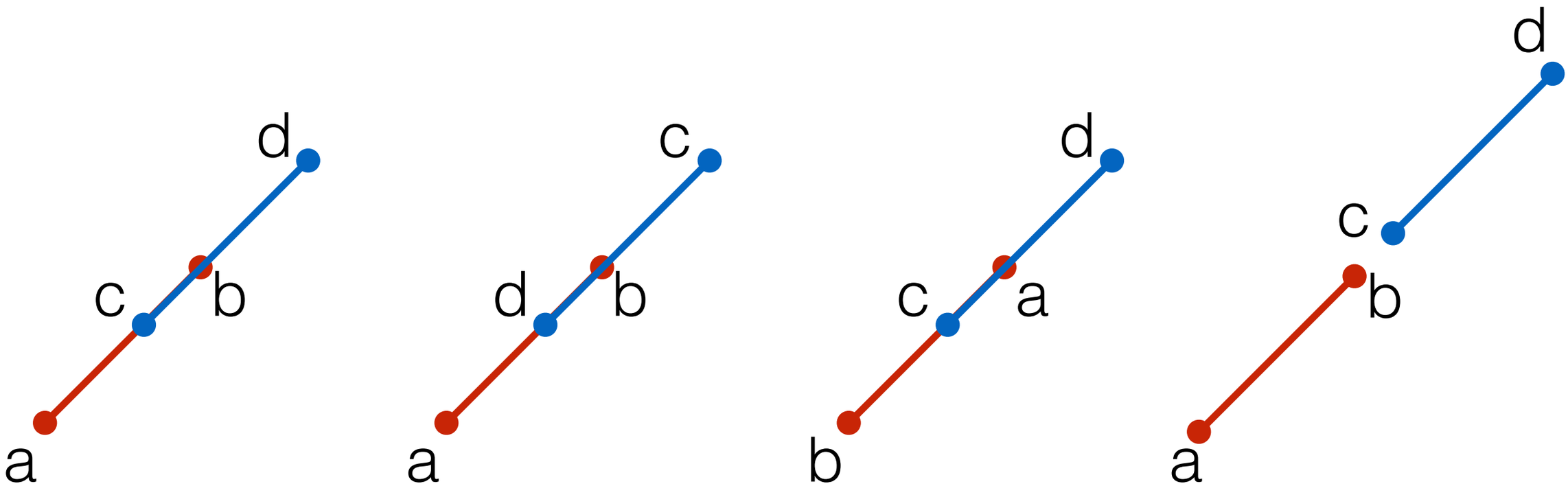
interseção de segmentos

- Segmentos (a,b) e (c,d) não colinares intersectam sse
  - (a,b,c) e (a,b,d) possuem orientações distintas ***e***
  - (c,d,a) e (c,d,b) possuem orientações distintas



interseção de segmentos

- Segmentos (a,b) e (c,d) colinares intersectam sse
    - projeções em x de (a,b) e (c,d) intersectam **e**
    - projeções em y de (a,b) e (c,d) intersectam



Projeto 5: Parte D

- Implemente o algoritmo de “dilatação” de uma mapa de linhas
    - considere que robô é um quadrado
- Implemente o algoritmo de construção do mapa de visibilidade com o mapa dilatado (inclua os pontos P1 a P11)
- Entregue código e relatório contendo:
    - representação por segmento de linhas (coordenadas)
    - figura
    - trajetórias obtidas pela busca de frente de onda e custo uniforme
- Compare diferentes técnicas (tamanho do espaço de busca, custo da trajetória encontrada, facilidade de modificação de mapa/plano etc)

bibliografia

- S. Russel, P. Norvig. Artificial Intelligence: A Modern Approach, Vol 3, Pearson, 2009.

- R. Siegwart, I. R. Nourbakhsh, D. Scaramuzza. Introduction to Autonomous Mobile Robots. MIT Press, 2011, Cap. 6

- R. Haralick, L. Shapiro. Computer and Robot Vision, Vol 1, Addison-Wesley Publishing Company, 1992, Cap. 5, pag. 168—173.