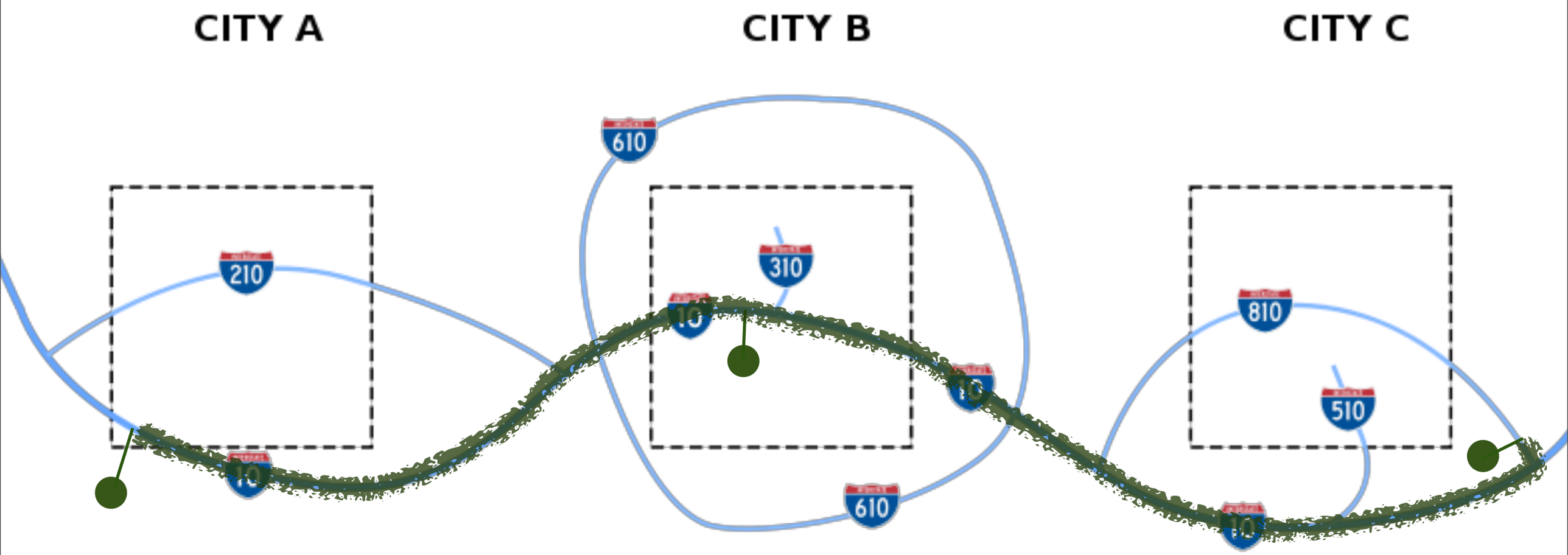
Topic 8

Planning with Roadmaps



Building a path to goal

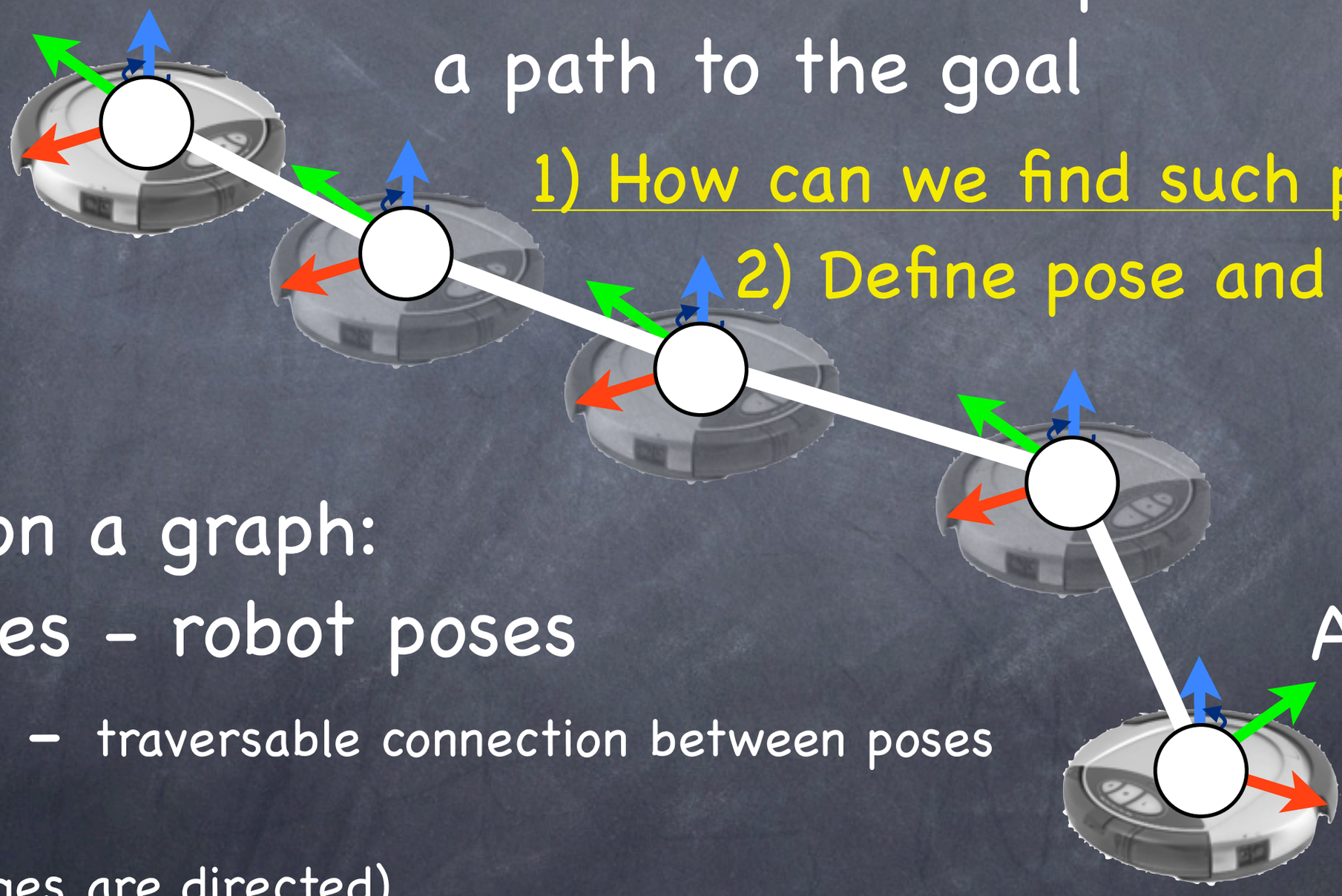
Path Planning

B: Goal

Find intermediate poses forming a path to the goal

1) How can we find such paths?

2) Define pose and controls?



A: Start

Path on a graph:

vertices - robot poses

edges - traversable connection between poses

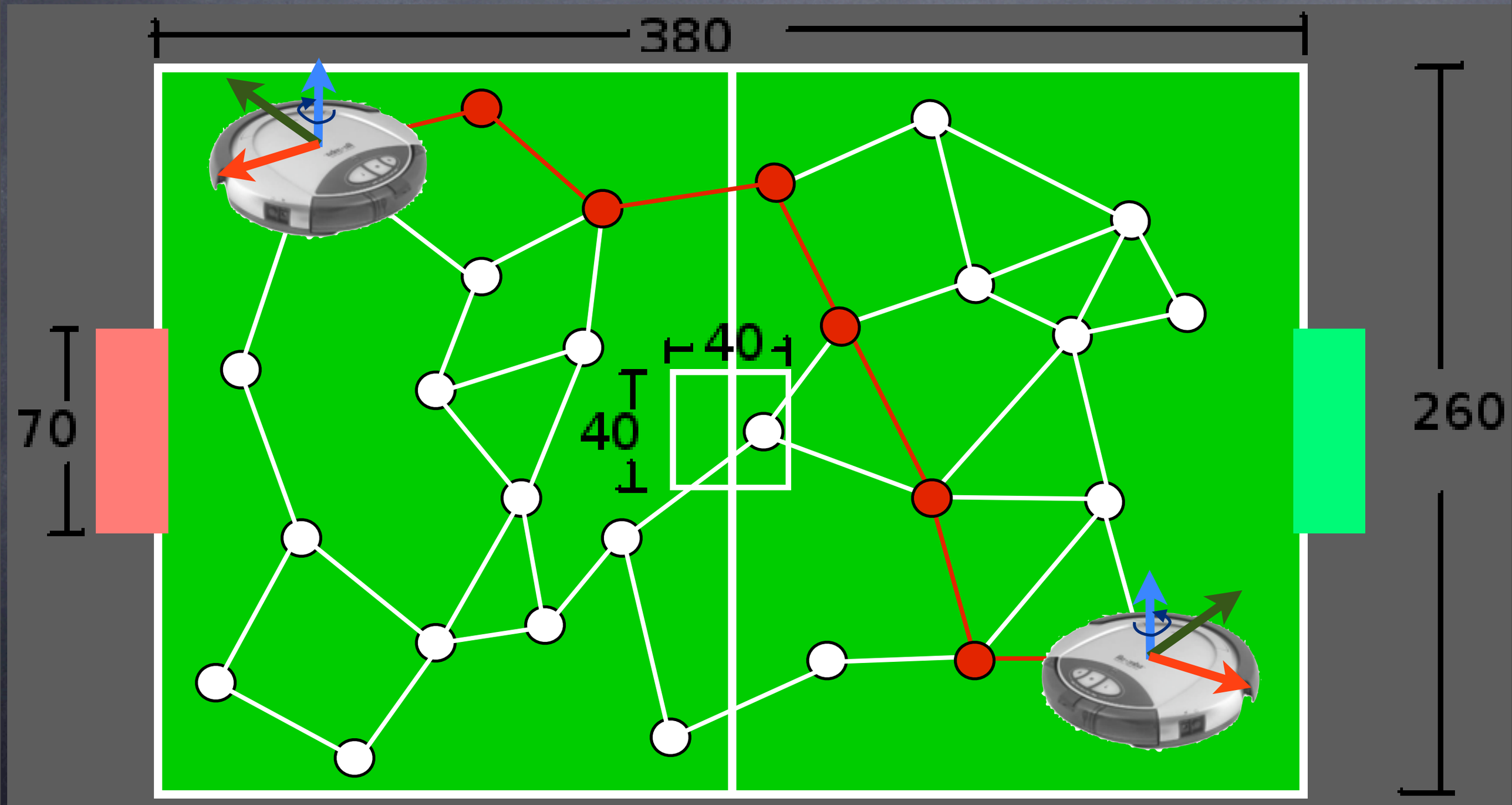
(note edges are directed)

Approaches to path planning

- Search (fixed graph)
  - DFS, BFS, Dijkstra, A*
- Search (build graph):
  - Probabilistic Road Maps
  - Rapidly-exploring Random Trees
- Optimization (local search):
  - Potential fields, gradient descent

PRMs/ RRT

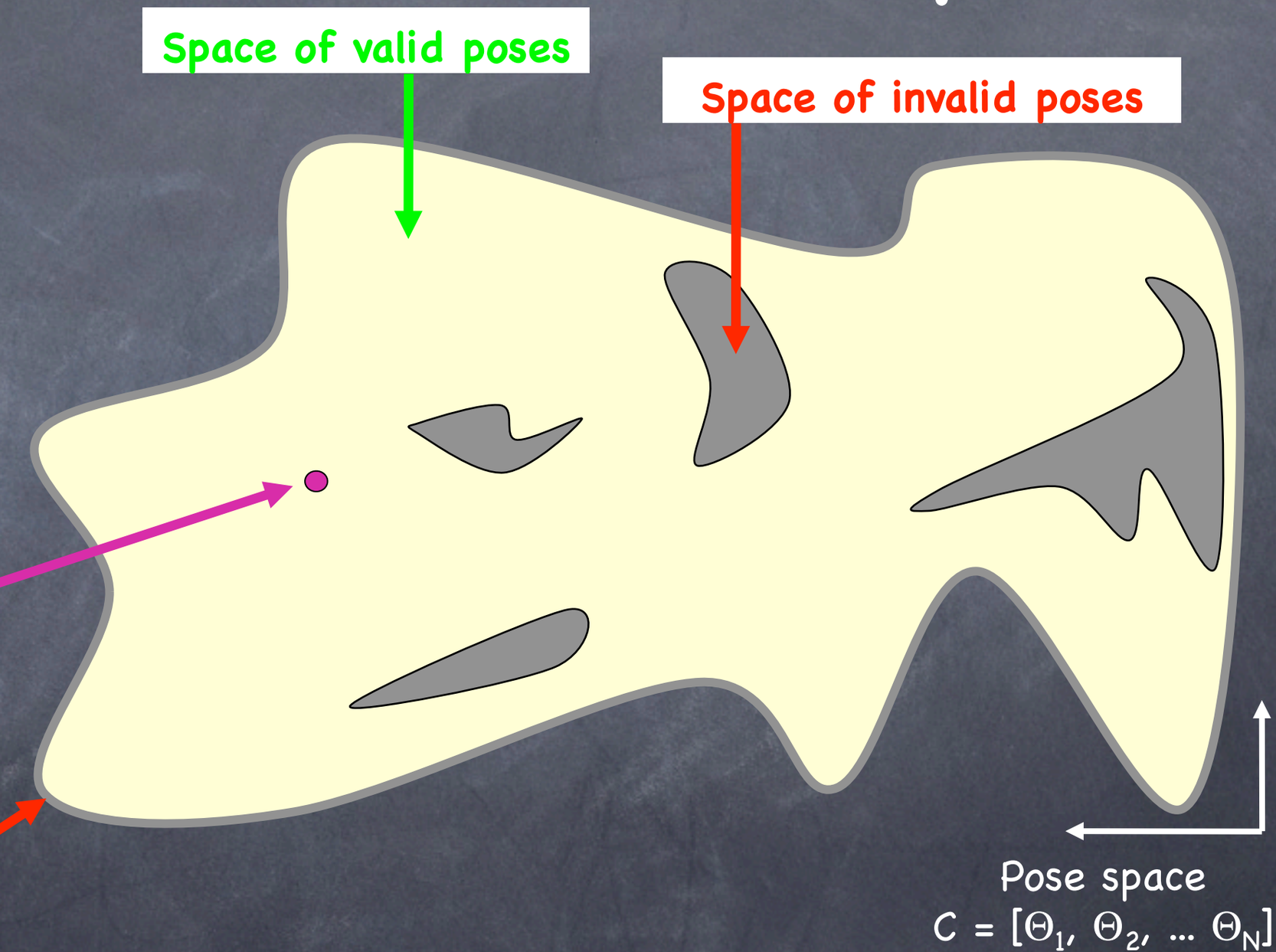
Explore poses and connectivity;
Find shortest path in built graph



[Kavraki, Svetska, Latombe,Overmars, 95]

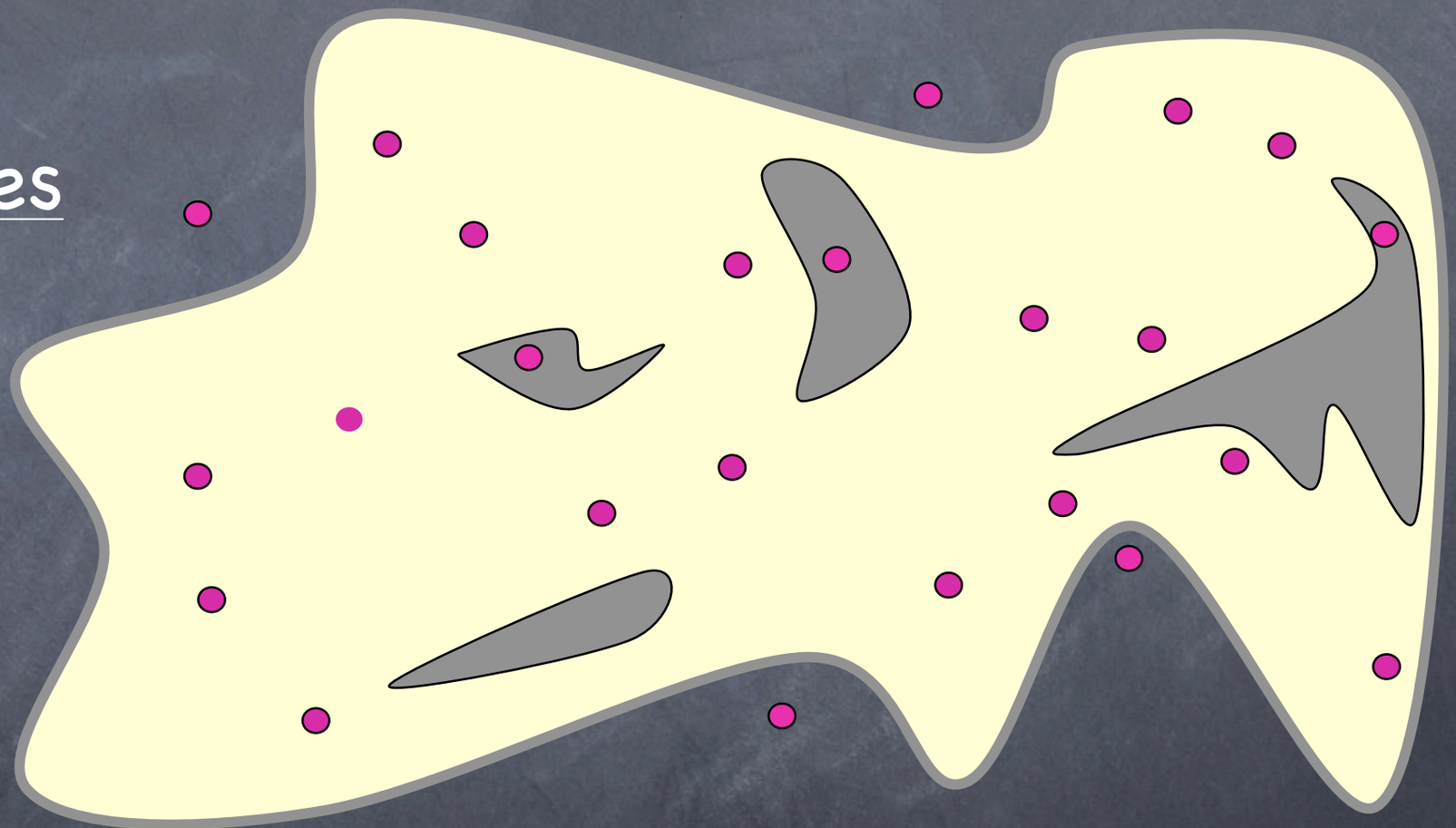
Probabilistic road maps

- Two phases
  - Roadmap construction
  - Path Query



PRM: construction phase

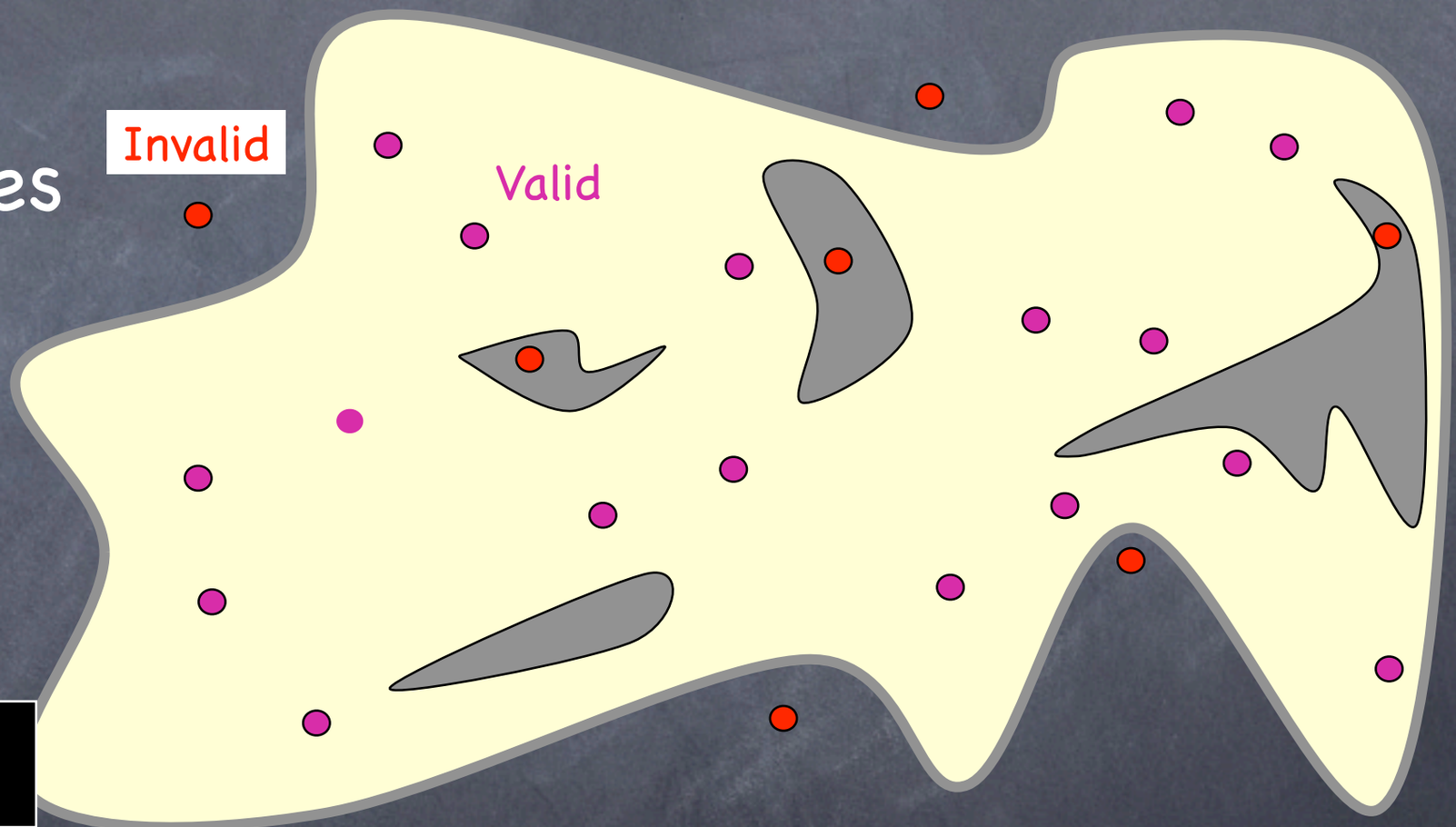
- Select sample poses at random
- Eliminate invalid poses
- Connect neighboring poses



PRM: construction phase

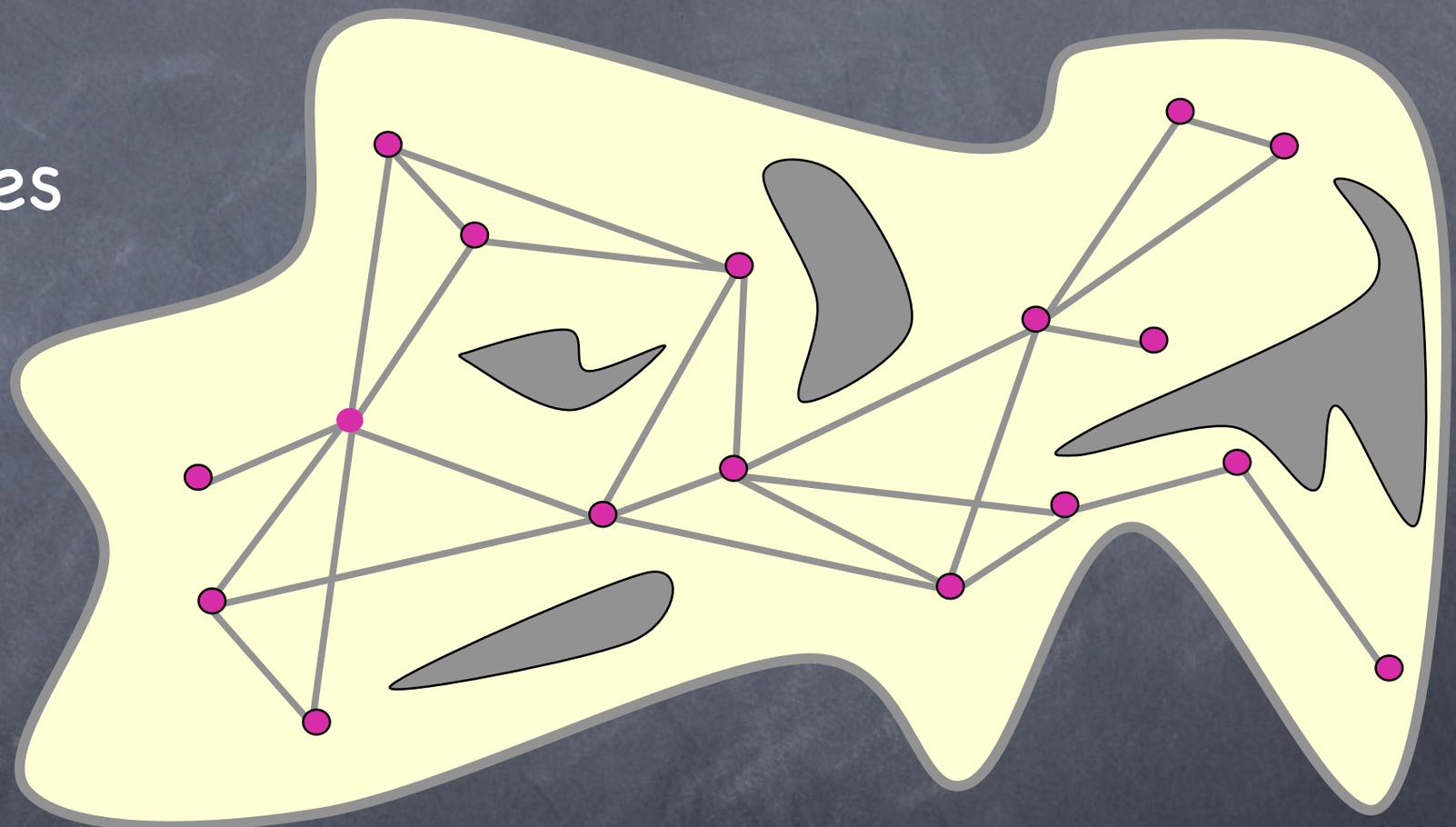
- Select sample poses at random
- Eliminate invalid poses

  Collision detection
- Connect neighboring poses



PRM: construction phase

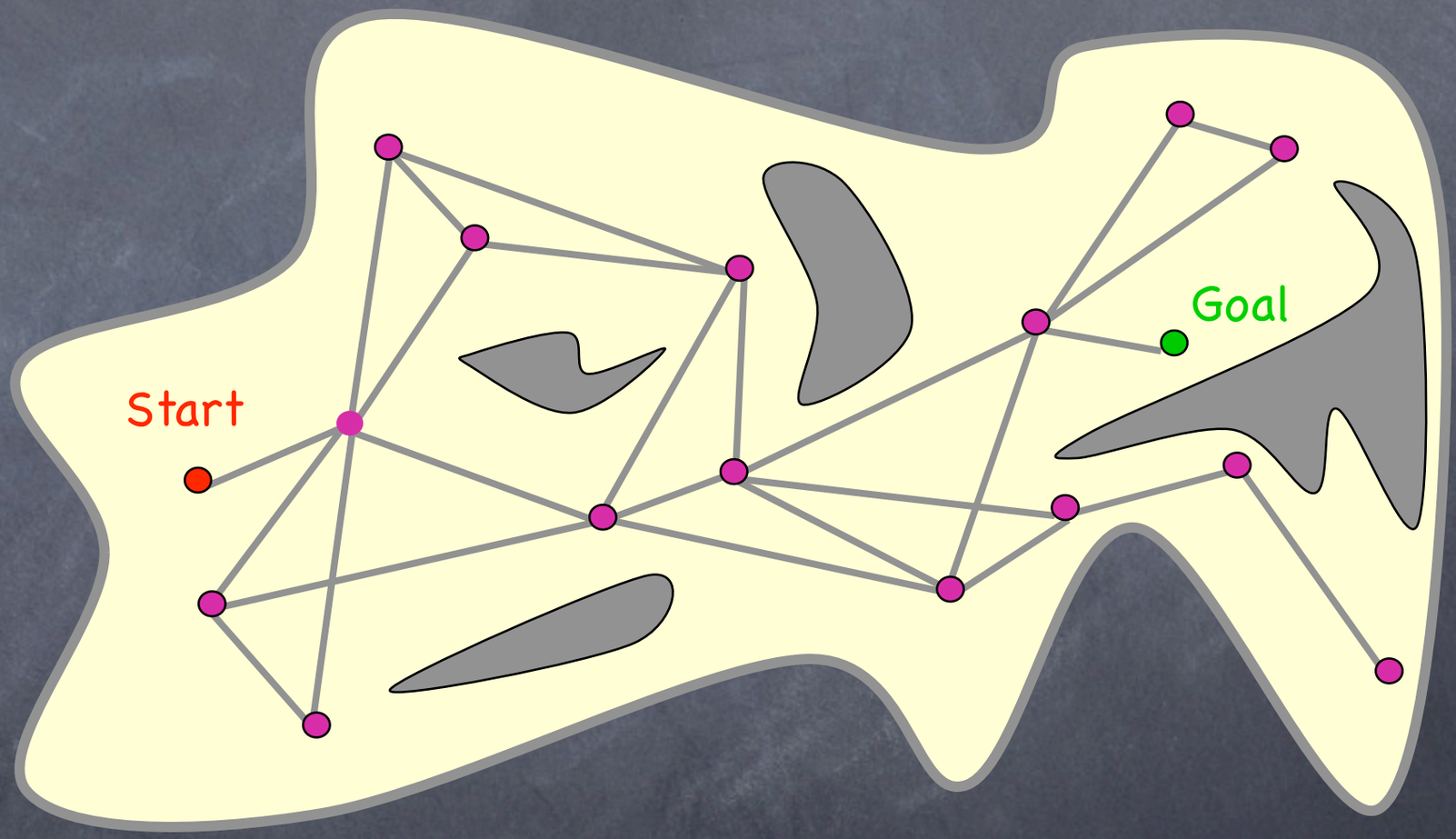
- Select sample poses at random
- Eliminate invalid poses
- Connect neighboring poses



Threshold nearest neighbors (radius, population).
Euclidean edge distances

PRM: query phase

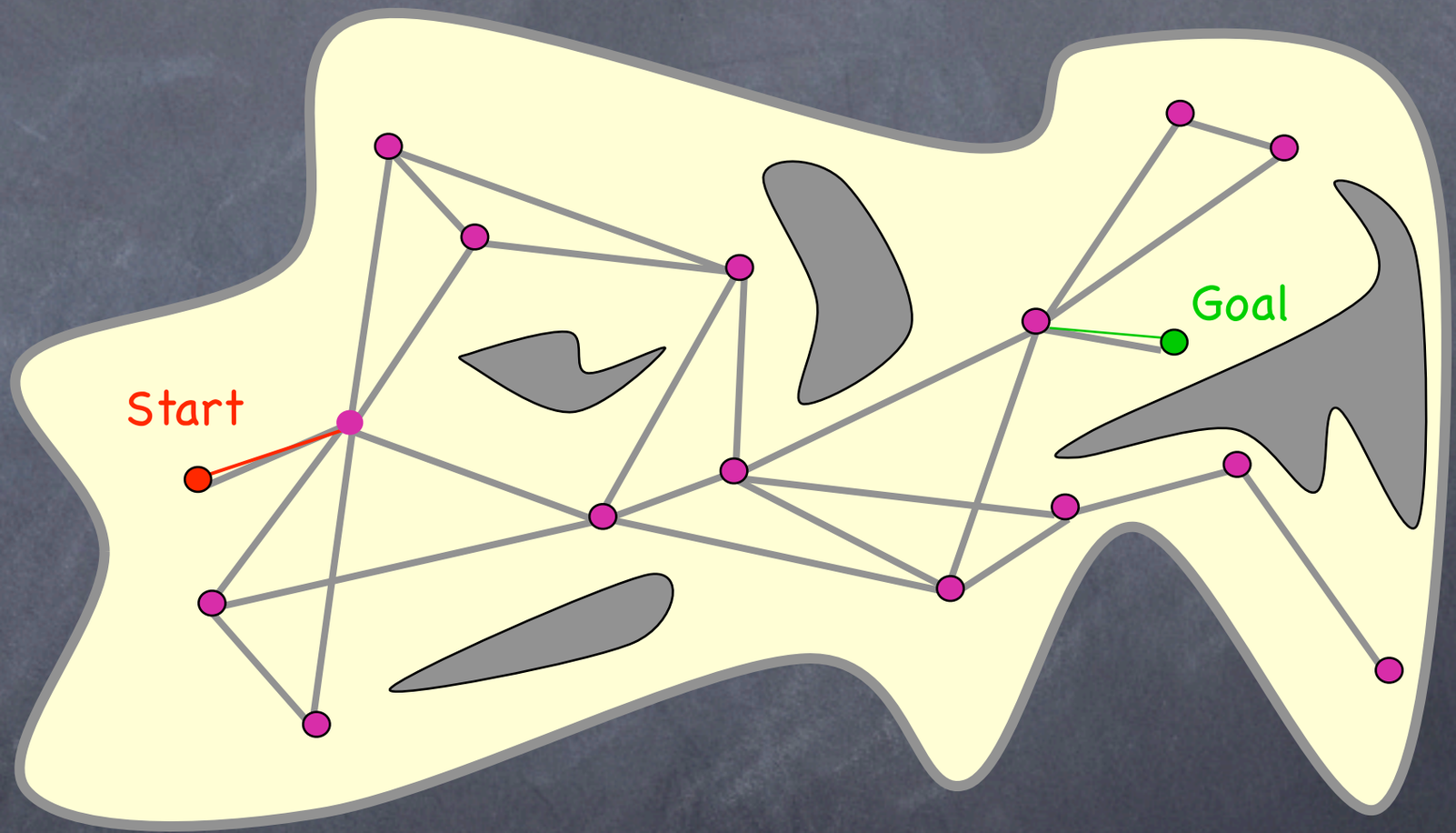
- Given constructed roadmap
- Find path in roadmap between two poses
- Search on an undirected graph



Remember: A*, Dijkstra, BFS, DFS.
BFS and Dijkstra behave differently?

PRM: query phase

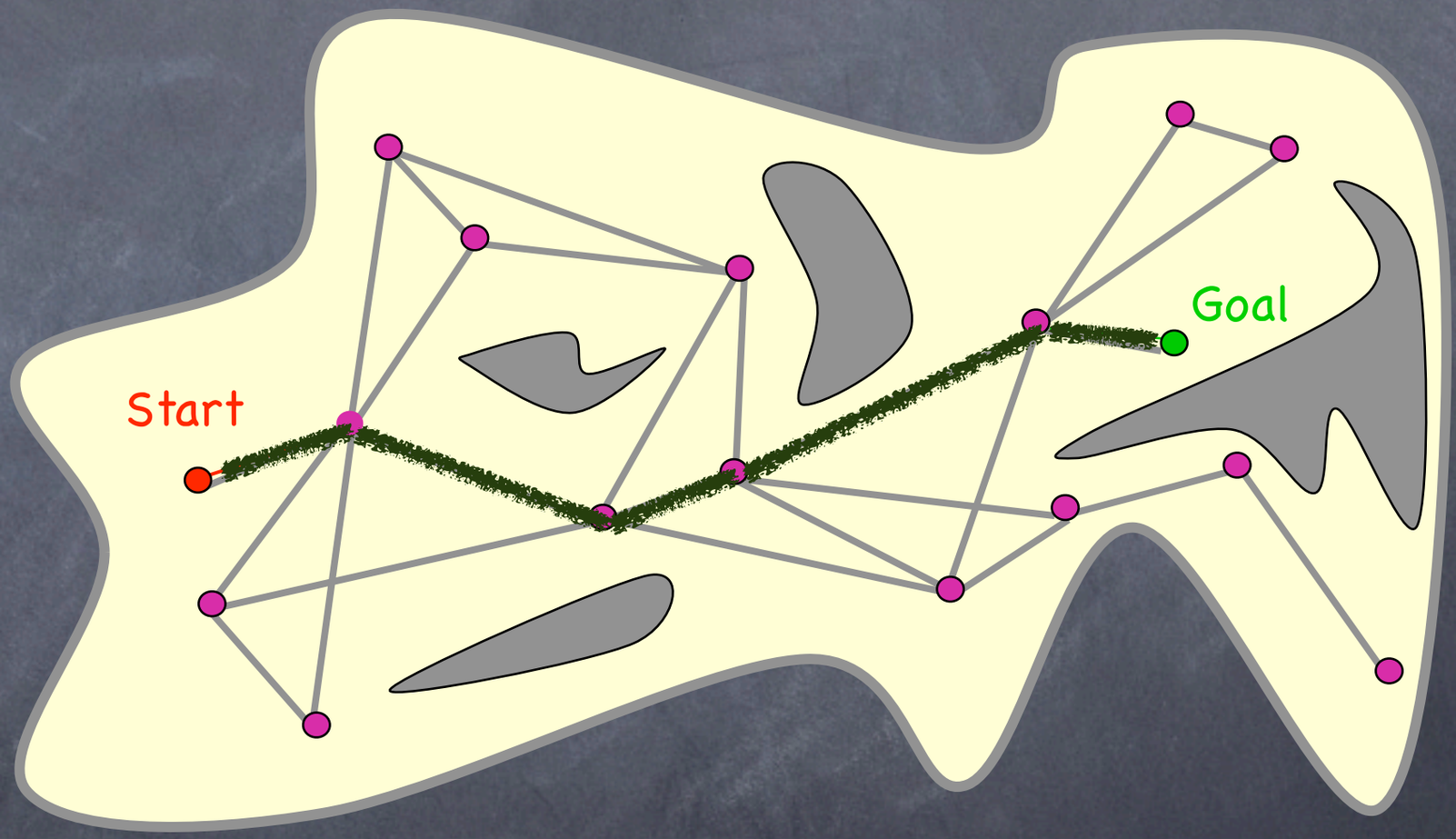
- Given constructed roadmap
- Find path in roadmap between two poses
- Search on an undirected graph



Remember: A*, Dijkstra, BFS, DFS.
BFS and Dijkstra behave differently?

PRM: query phase

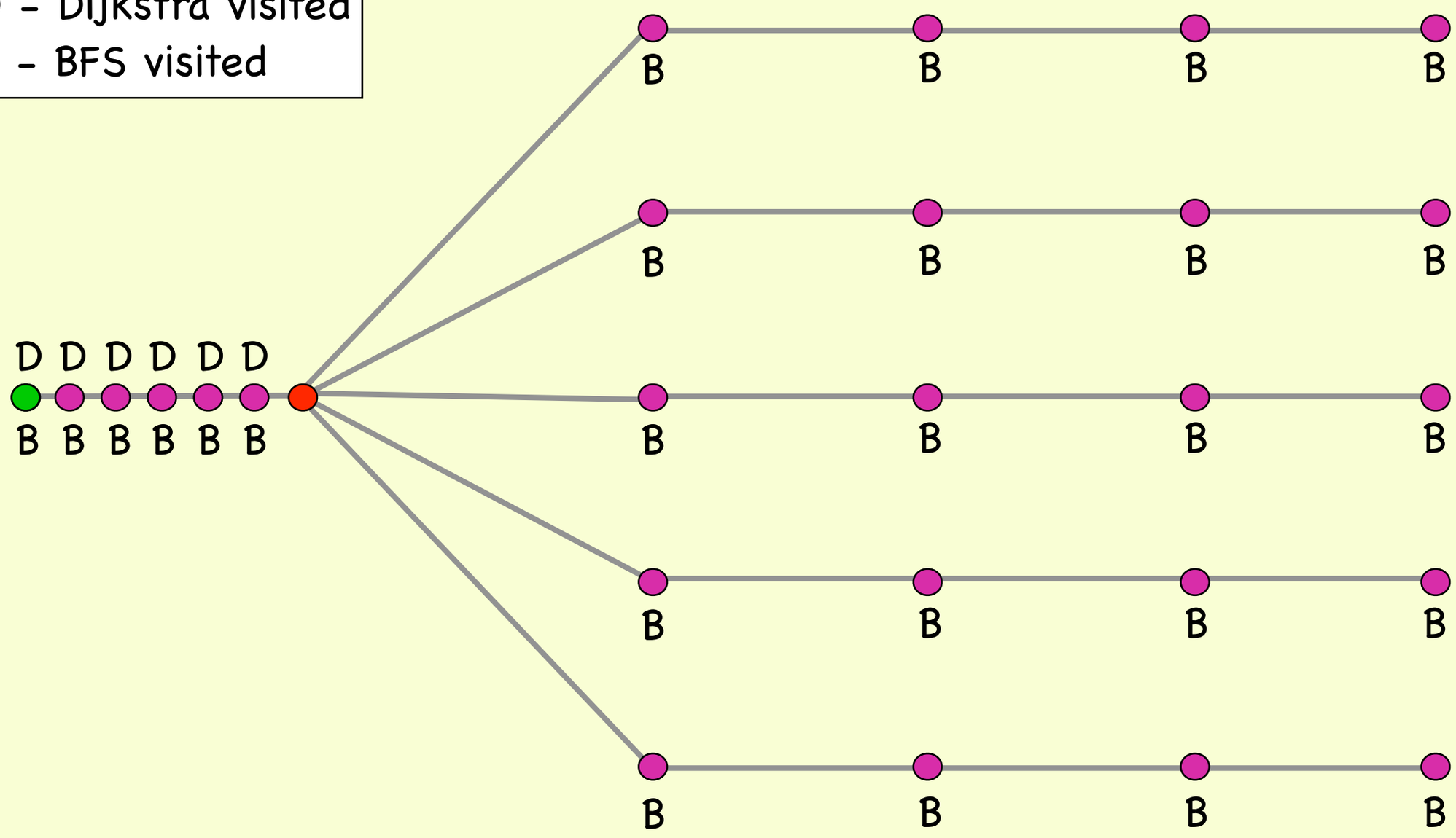
- Given constructed roadmap
- Find path in roadmap between two poses
- Search on an undirected graph



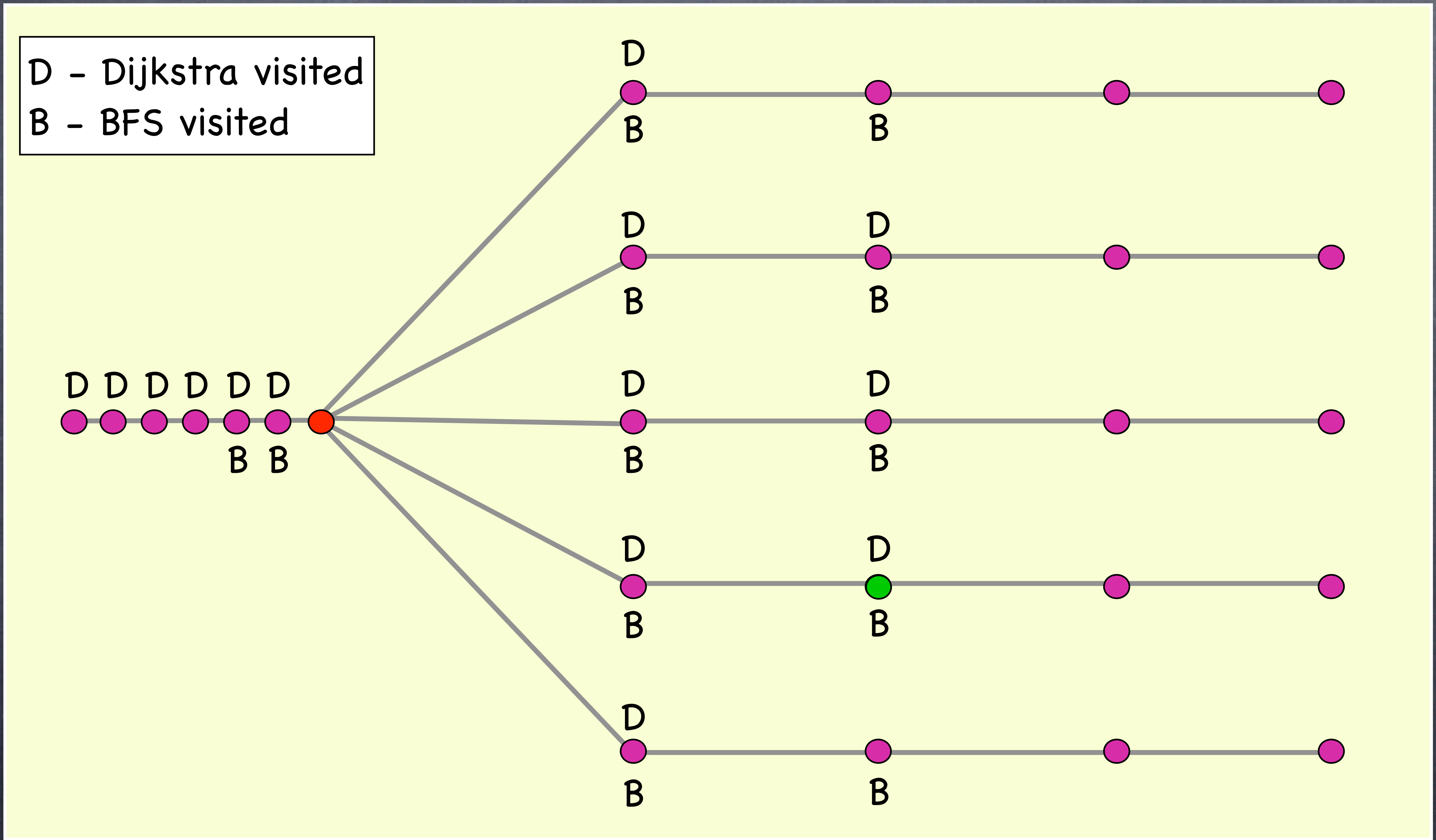
Remember: A*, Dijkstra, BFS, DFS.
BFS and Dijkstra behave differently?

Dijkstra v. BFS

D - Dijkstra visited
B - BFS visited



Dijkstra v. BFS



[Kuffner, LaValle 2000]

Rapidly-exploring Random Trees

Explore incrementally along straight line path to goal

```
BUILD_RRT(q_init)
 1  T.init(q_init);
 2  for k = 1 to K do
 3       q_rand ← RANDOM_CONFIG();
 4       EXTEND(T, q_rand);
 5  Return T
```

```
EXTEND(T, q)
 1  q_near ← NEAREST_NEIGHBOR(q, T);
 2  if NEW_CONFIG(q, q_near, q_new) then
 3       T.add_vertex(q_new);
 4       T.add_edge(q_near, q_new);
 5       if q_new = q then
 6            Return Reached;
 7       else
 8            Return Advanced;
 9  Return Trapped;
```

Figure 2: The basic RRT construction algorithm.

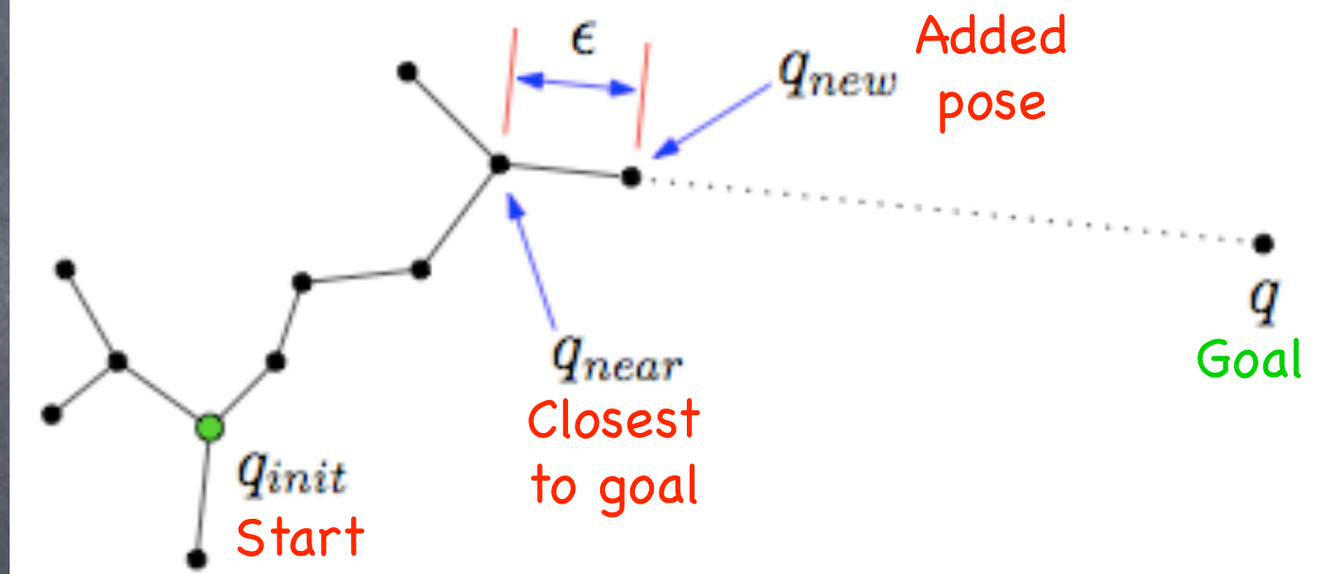
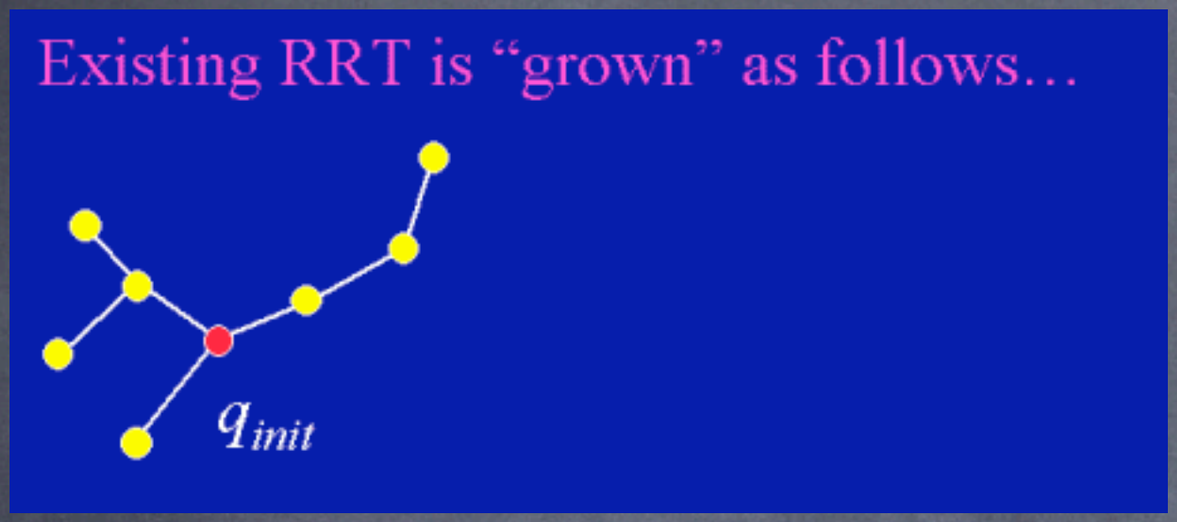


Figure 3: The EXTEND operation.

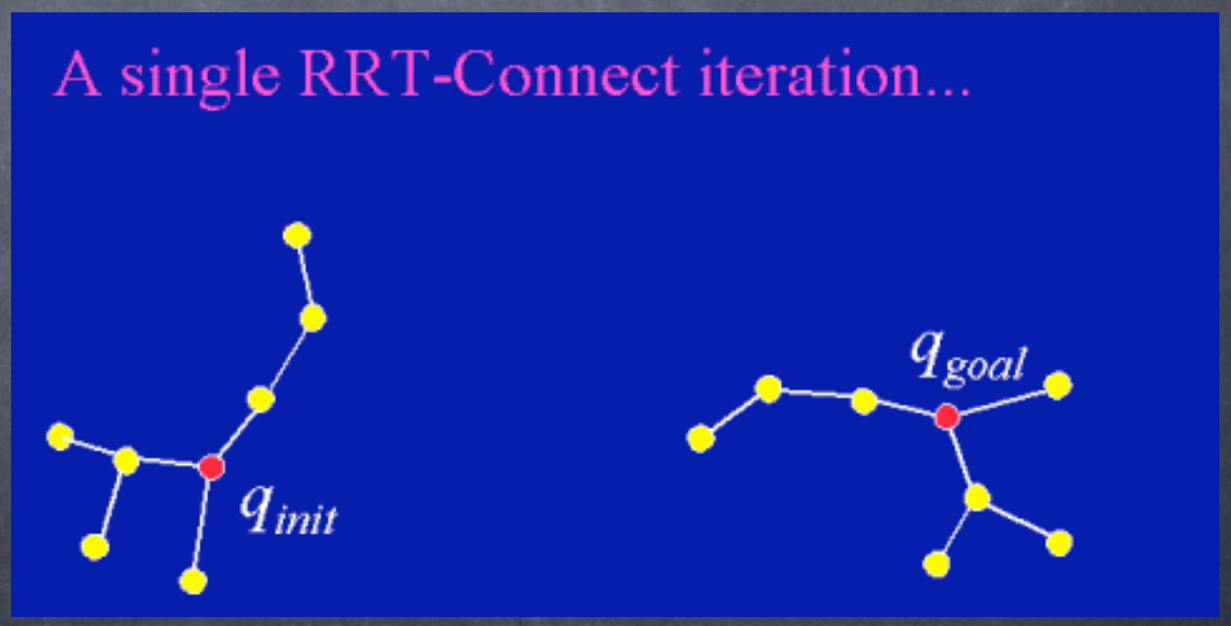
Single vs. multiple query

Kuffner's RRT Animations



Extending roadmap

Bidirectional exploration



http://www.kuffner.org/james/plan/algorithm.php

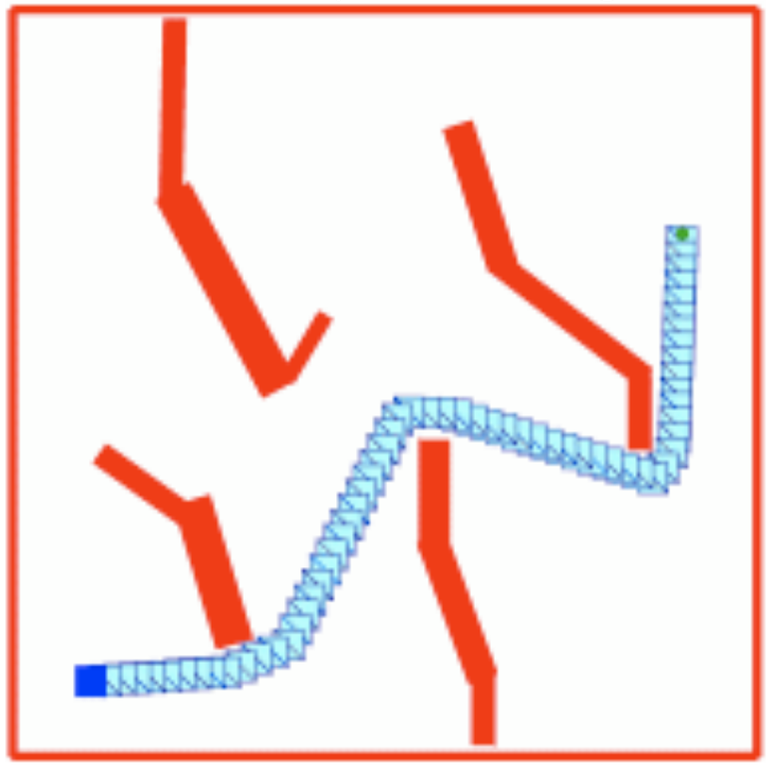
RRT bidirectional search example

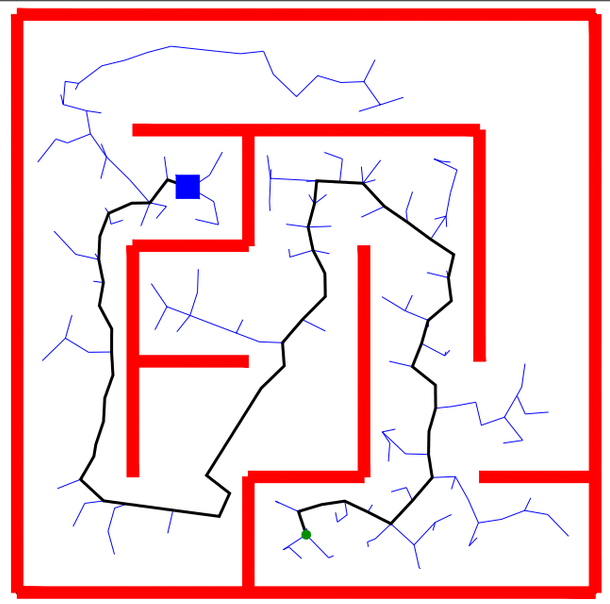
extend graph from both start and goal

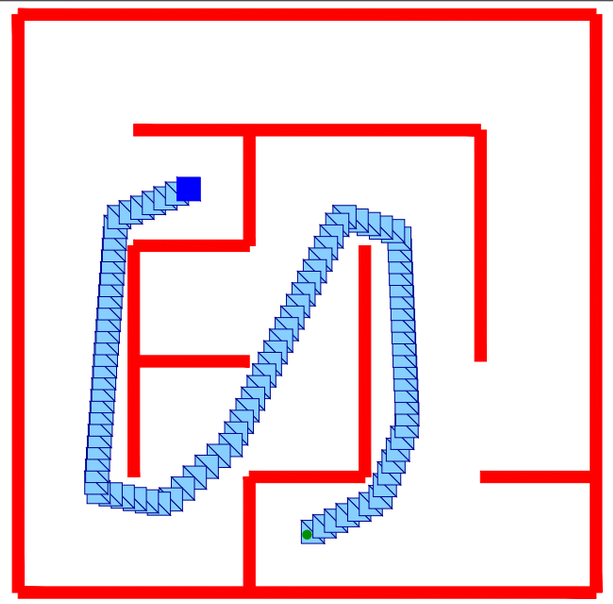
continue exploration until start and goal connected

once connected, search for shortest path

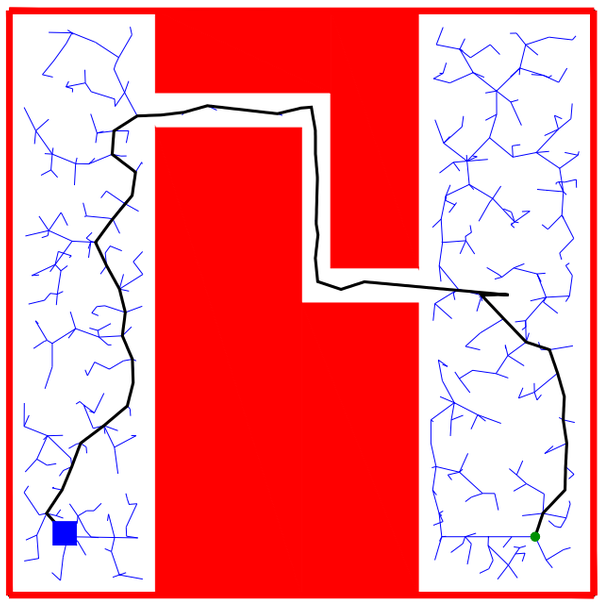
execute path to goal

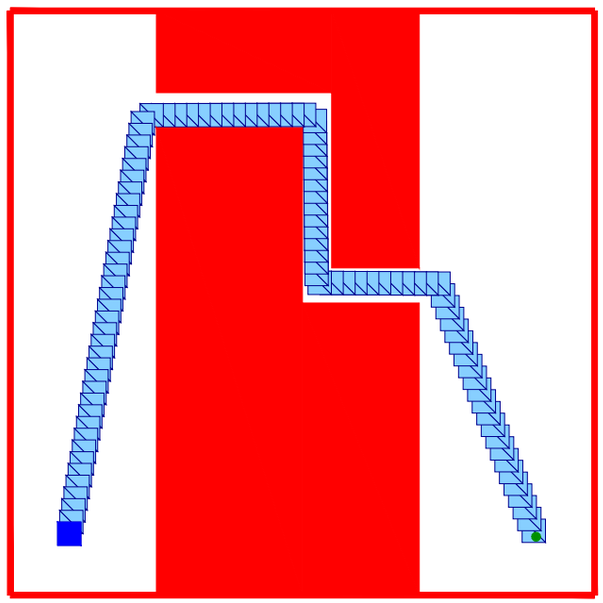






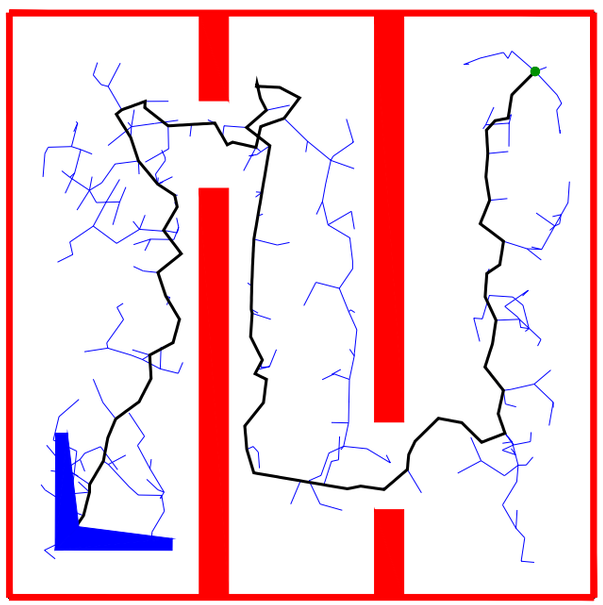
2 DOF maze

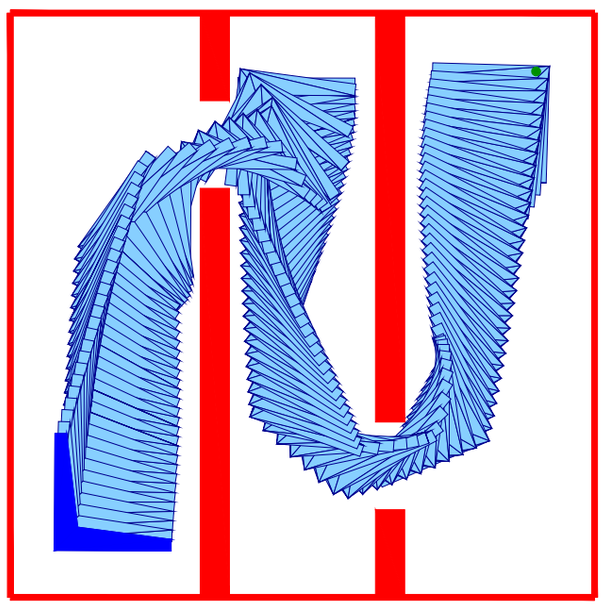




2 DOF single passway

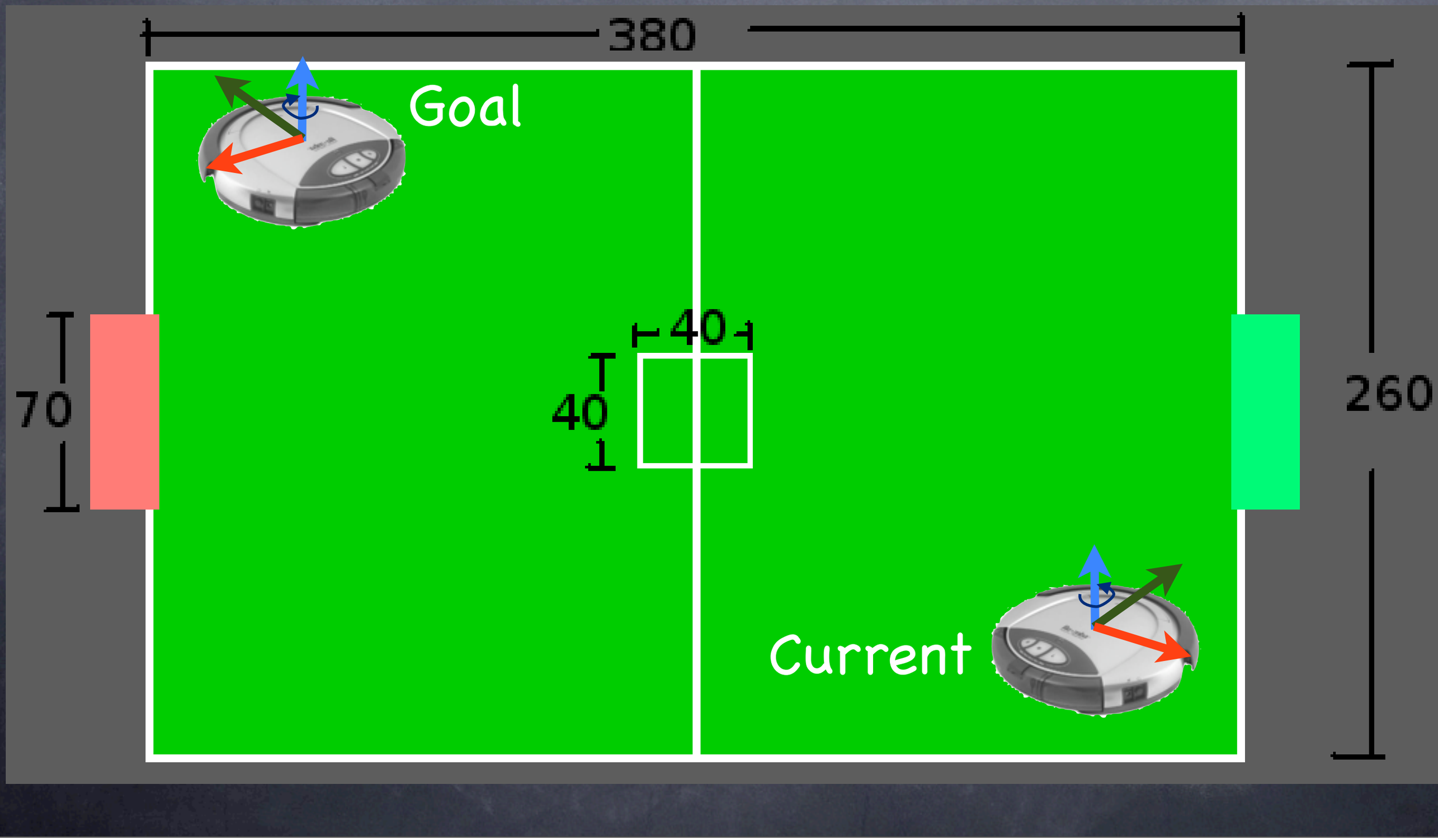
RRT Examples



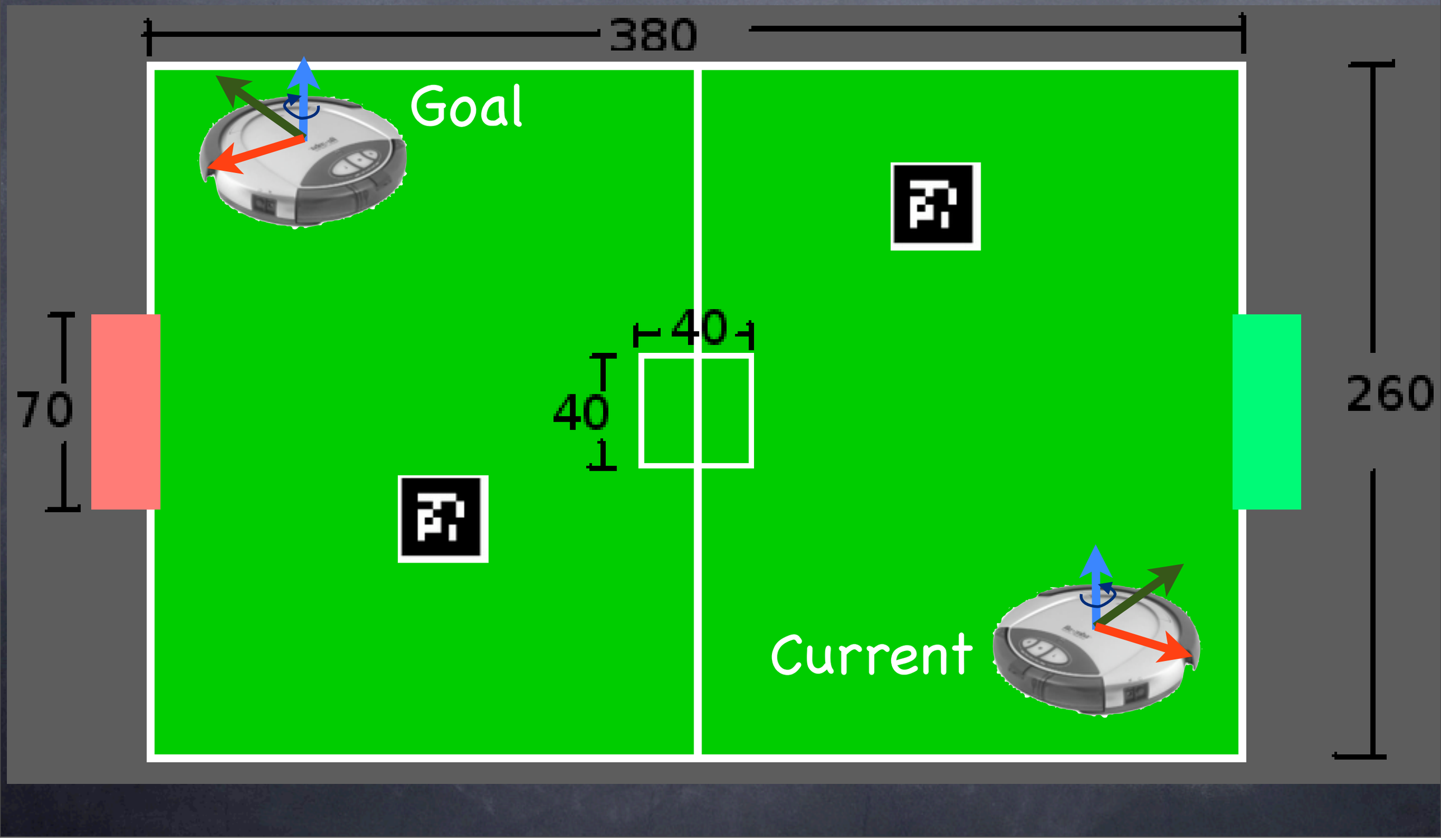


3 DOF single passway

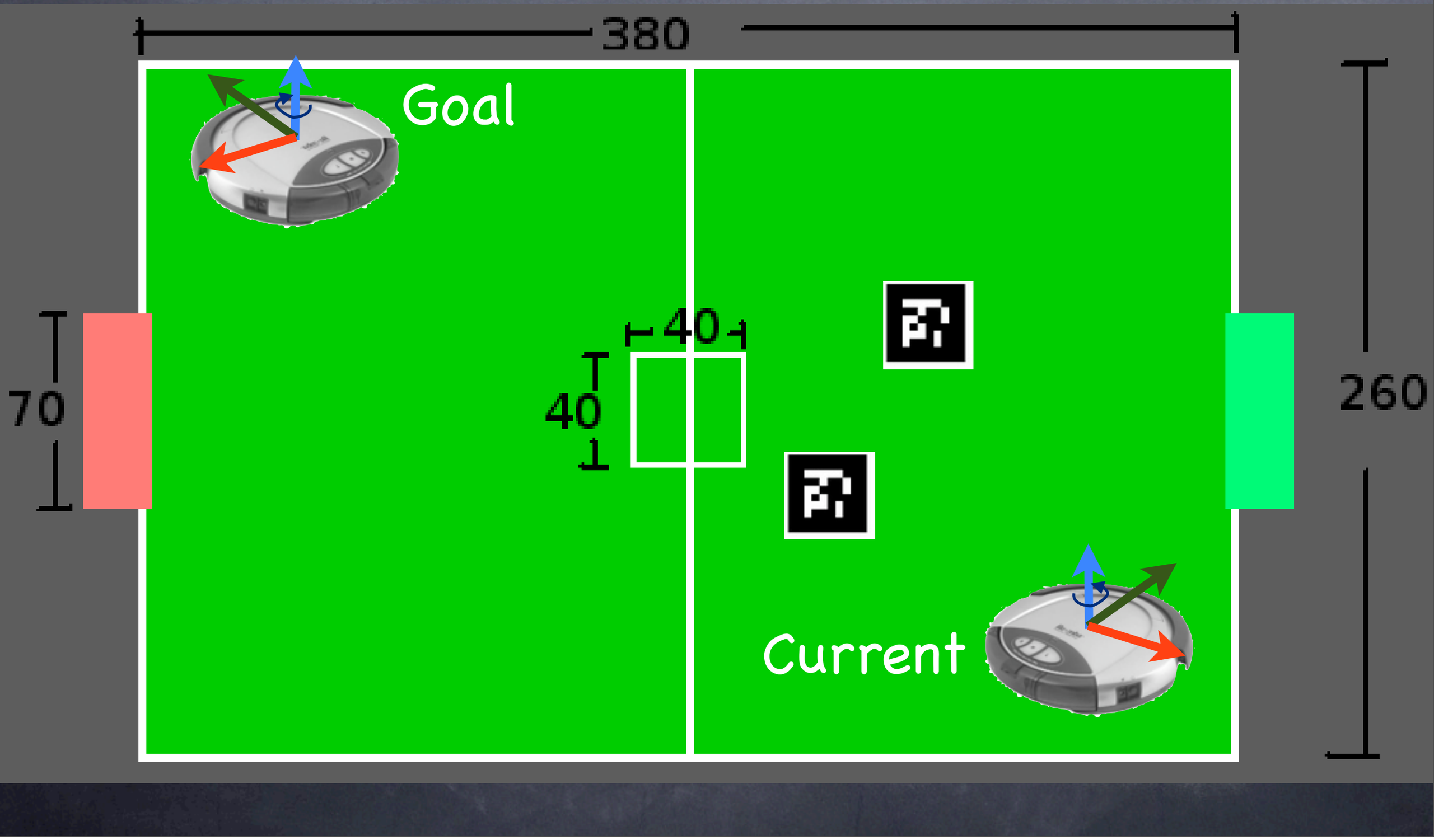
describe performance for this case



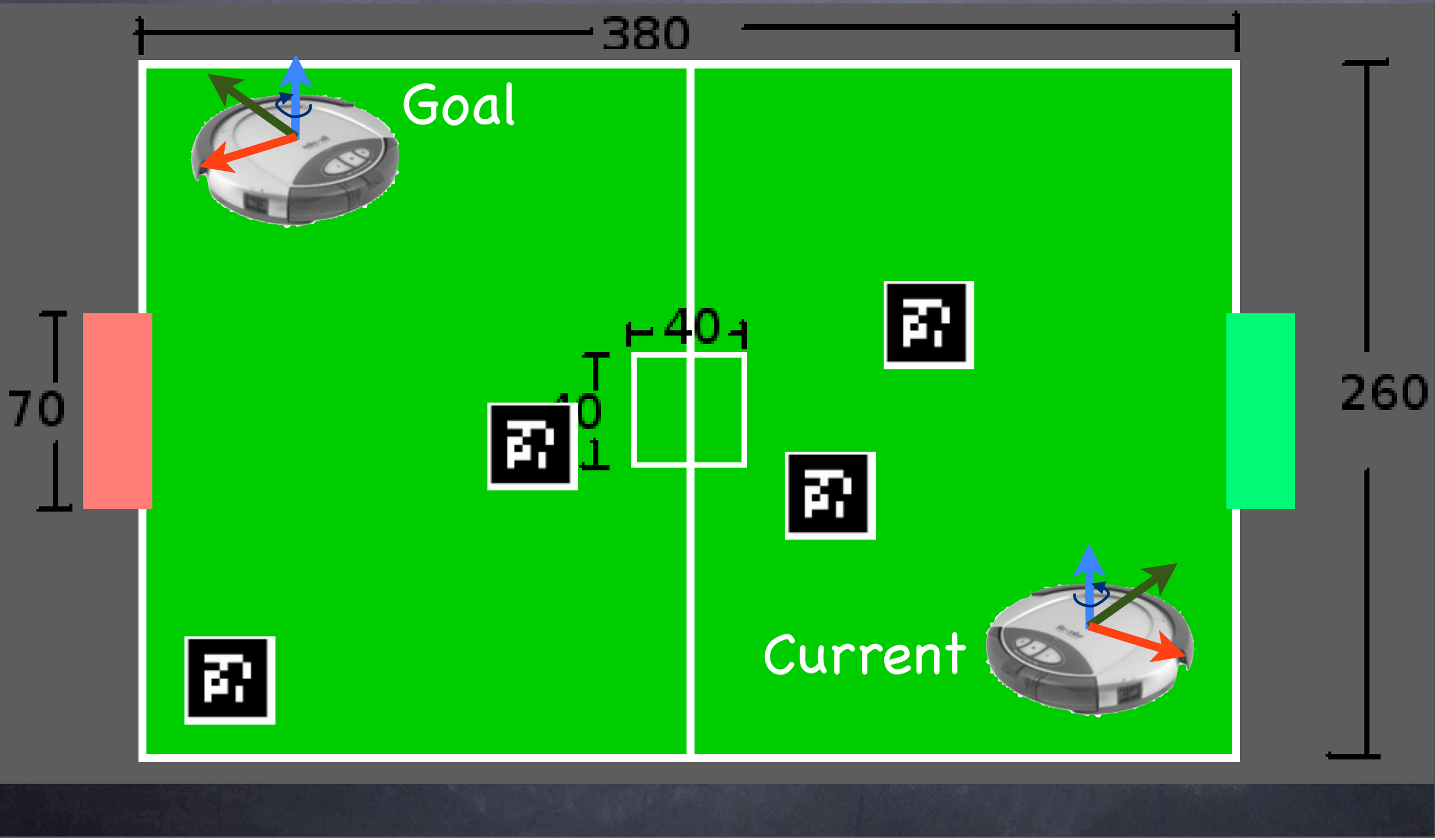
describe performance for this case



describe performance for this case



describe performance for this case



describe performance for this case