CS 4758/6758: Robot Learning

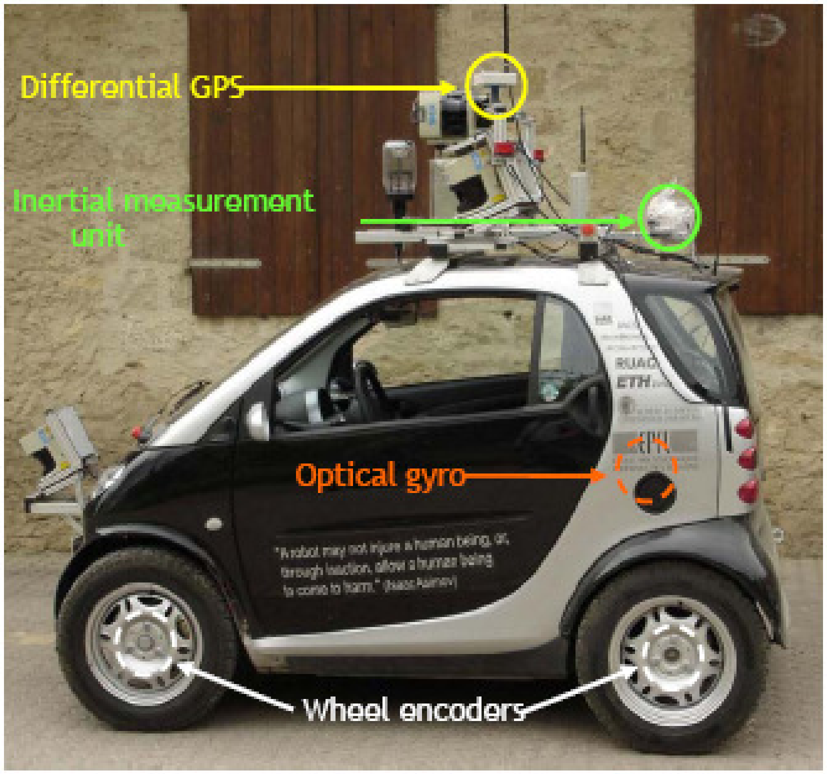
Spring 2010: Lecture 9

Ashutosh Saxena

Why planning and control?

Video

Typical Architecture



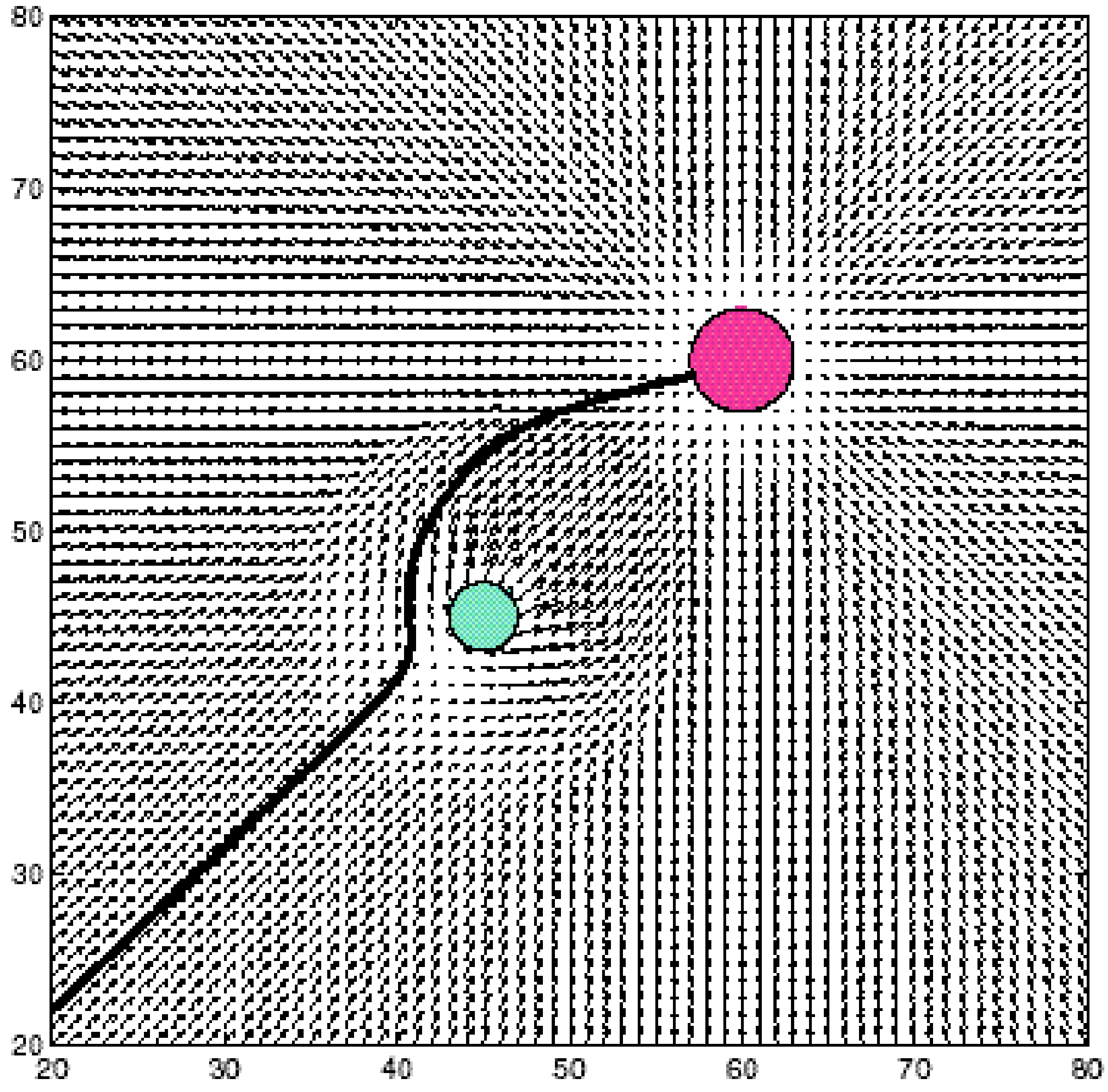
Planning
0.1 Hz

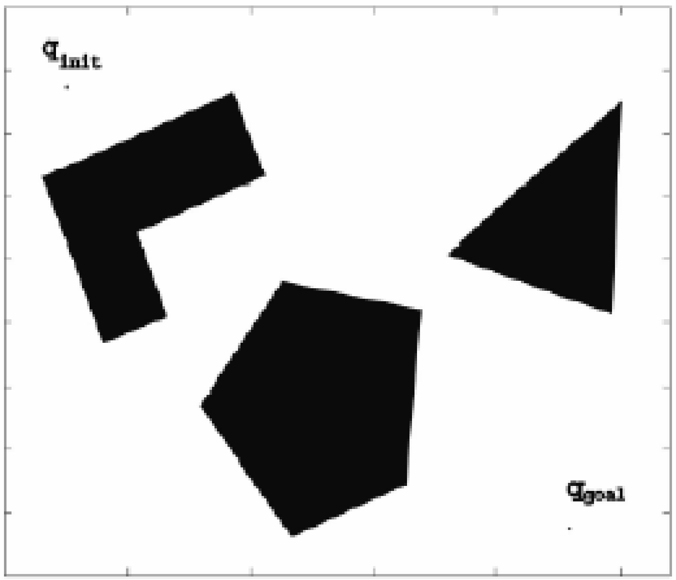
↓

Control
50 Hz

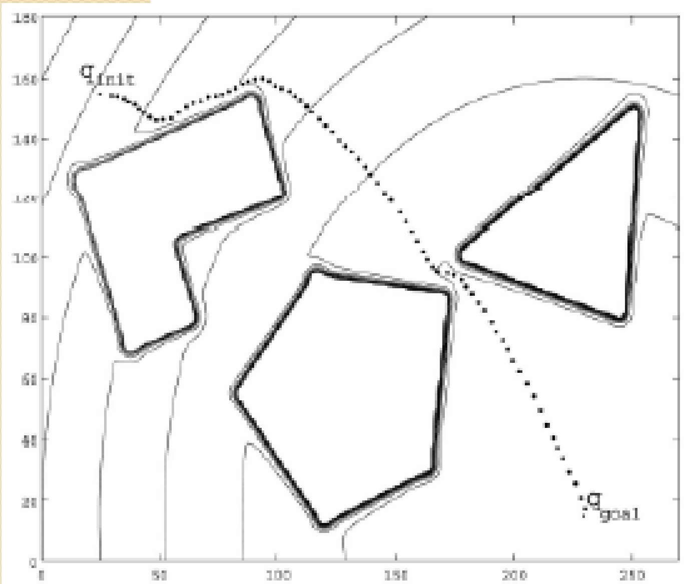
Does it apply to all robots and all scenarios?

Previous Lecture: Potential Field

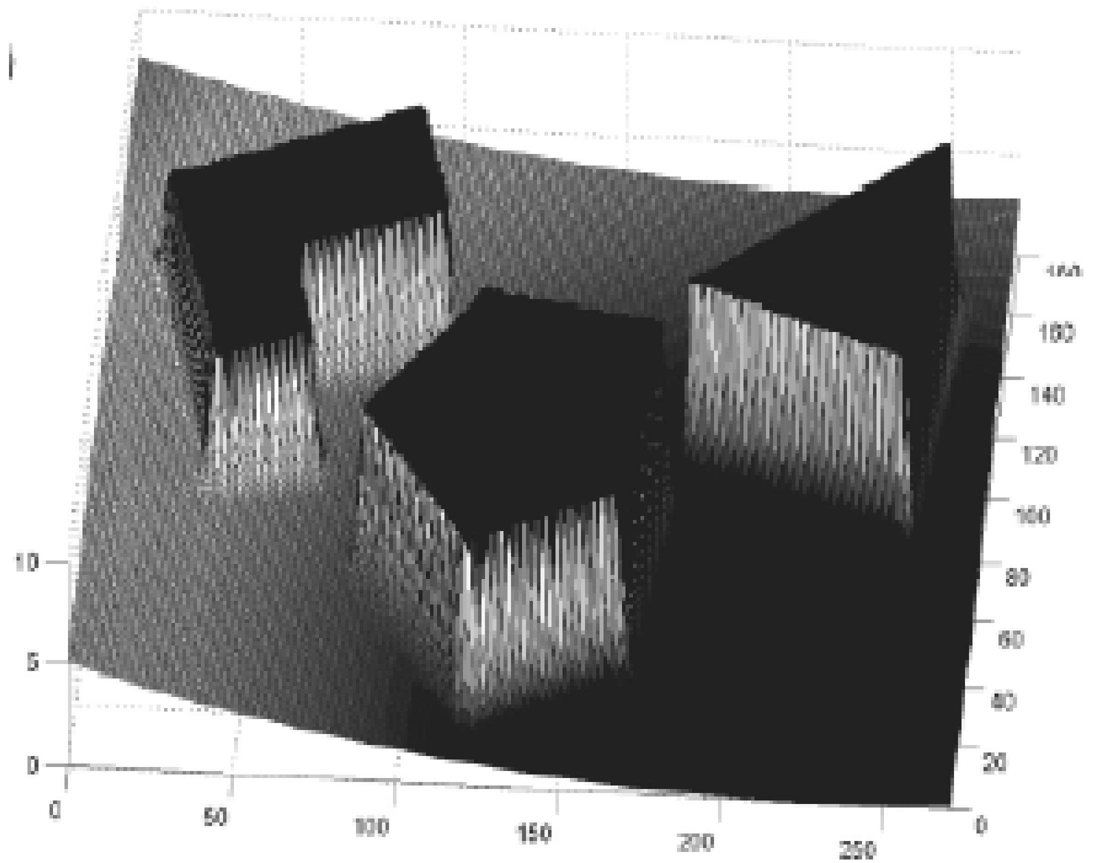




q_{init}
q_{goal}



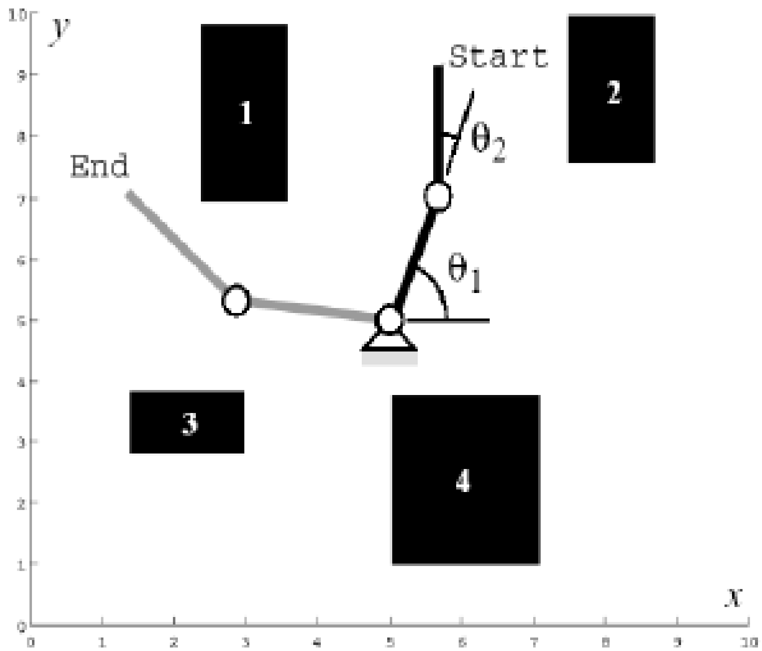
q_{init}
q_{goal}



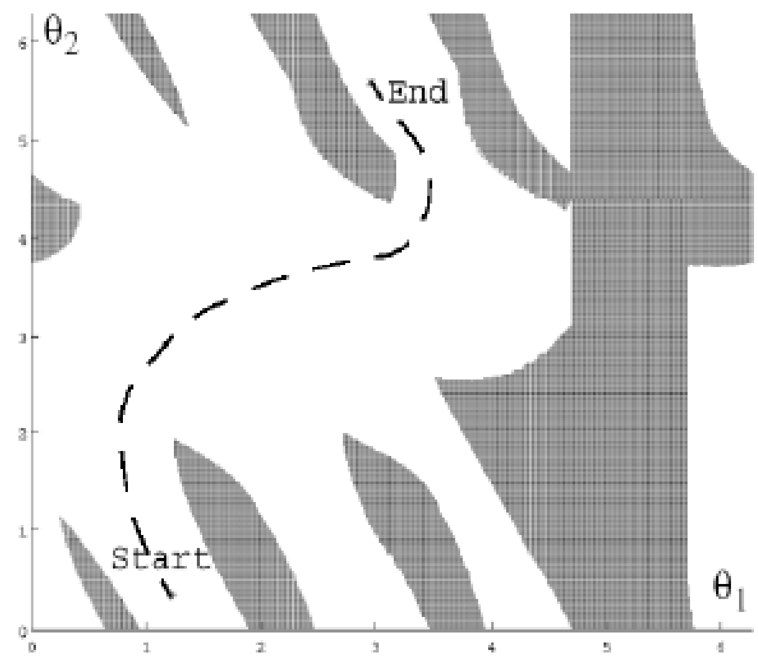
10
5
0
0
50
100
150
200
250
0
20
40
60
80
100
120
140
160

Planning: Articulated Robots

Path Planning



Work Space



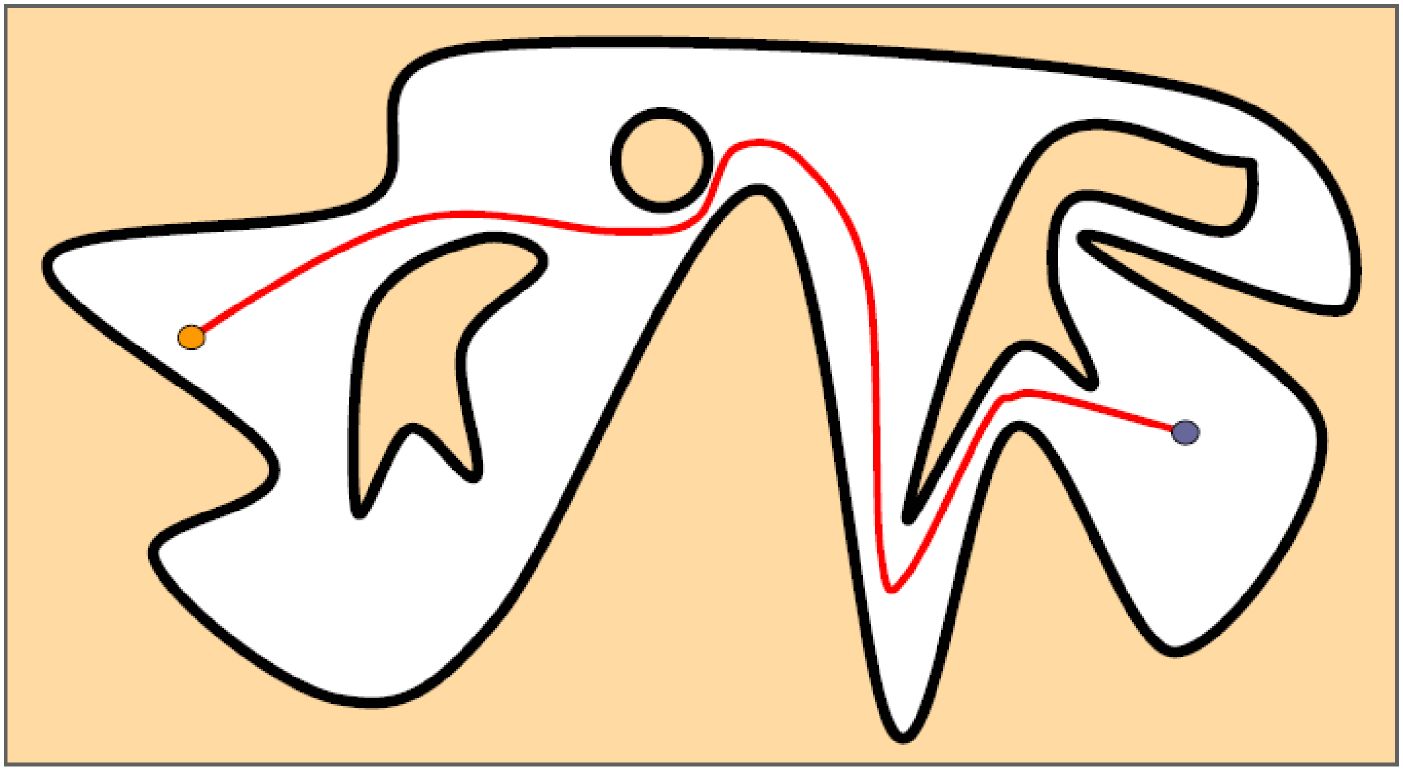
Configuration Space:
the dimension of this space is equal to the Degrees of Freedom (DoF) of the robot

Planning: Goals

- **Compute** motion strategies, **e.g.,**
  - **Geometric paths**
  - **Time-parameterized trajectories**
  - **Sequence of sensor-based motion commands**

- **Achieve** high-level goals, **e.g.,**
  - **Go to the door and do not collide with obstacles**
  - **Assemble/disassemble the IKEA bookshelf.**

Fundamental Question



Are two given points connected by a path?

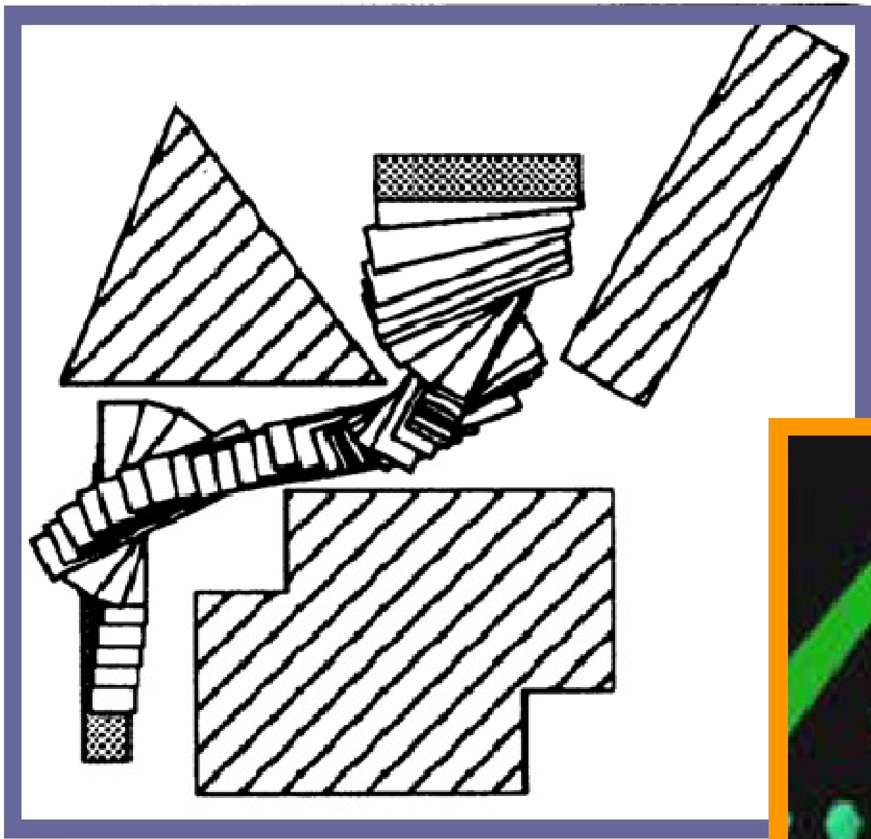
Motion Planning: Basic Problem

- Problem statement:

  Compute a collision-free path for a rigid or articulated moving object among static obstacles.
- Input
  - Geometry of a moving object, robot, and obstacles
  - How does the robot move?
  - Kinematics of the robot (degrees of freedom)
  - Initial and goal robot configurations (positions & orientations)
- Output

  Continuous sequence of collision-free robot configurations connecting the initial and goal configurations

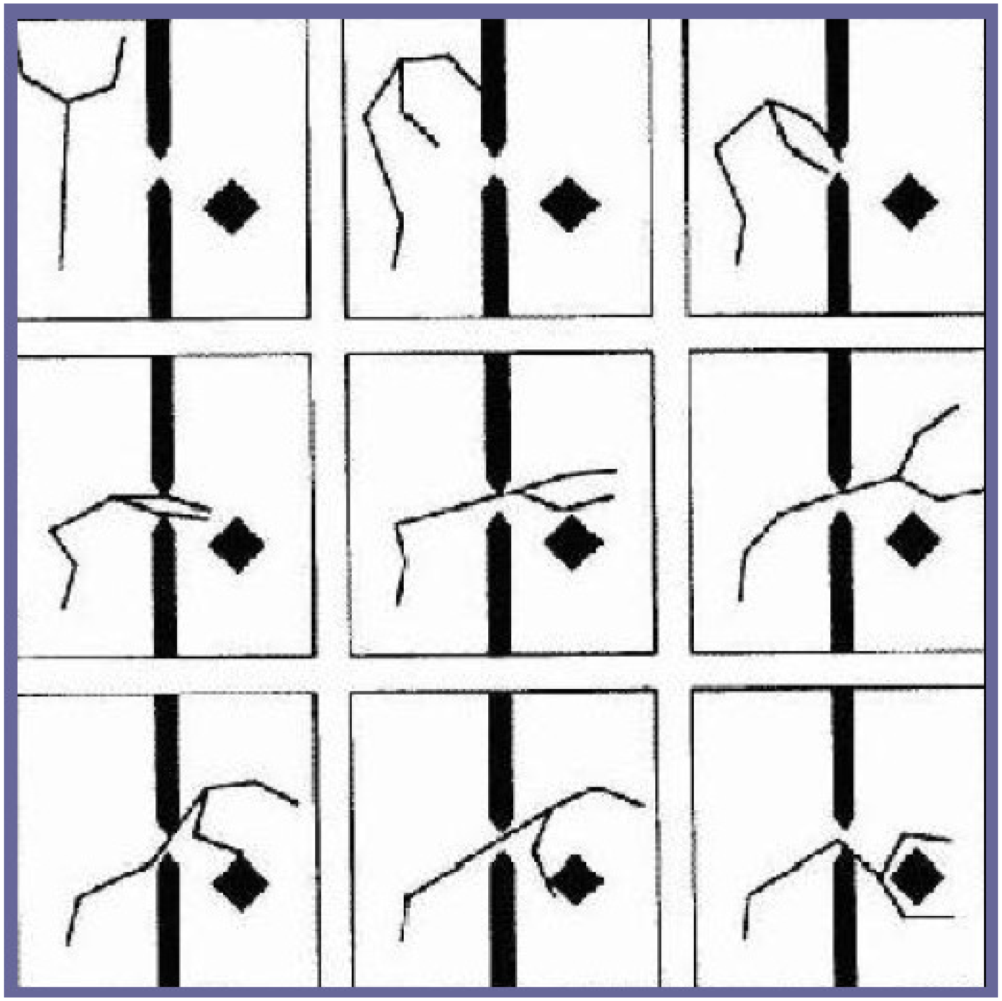
Example: Rigid Objects



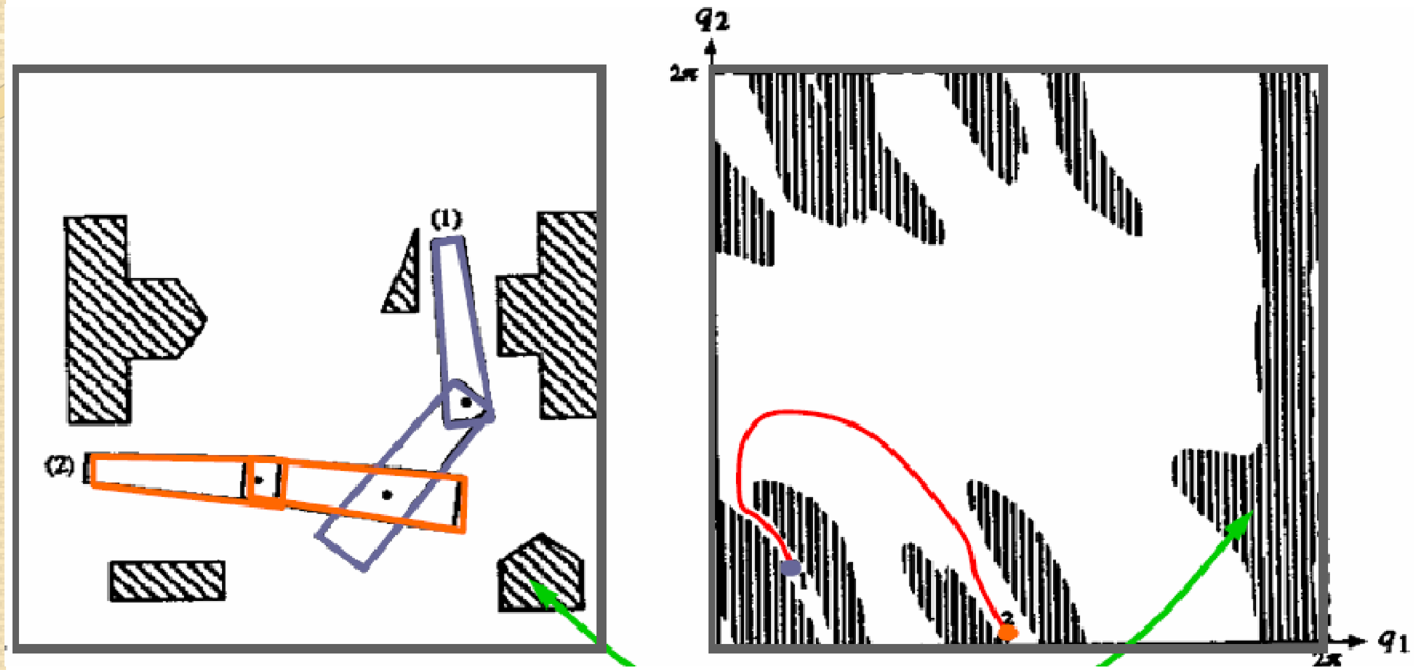
→ Ladder problem

Piano mover's problem ←

Example: Articulated Robot



Tool: Configuration Space



Difficulty

- Number of degrees of freedom (dimension of configuration space)
- Geometric complexity

Extensions of the Basic Problem

- More complex robots
  - **Multiple robots**
  - **Movable objects**
  - **Nonholonomic & dynamic constraints**
  - **Physical models and deformable objects**
  - **Sensorless motions (exploiting task mechanics)**
  - **Uncertainty in control**

Extensions of the Basic Problem

- More complex environments
  - **Moving obstacles**
  - **Uncertainty in sensing**
- More complex objectives
  - **Integration of planning and control**
  - **Sensing the environment**
    - **Model/map building**
    - **Target finding, tracking**

Practical Algorithms

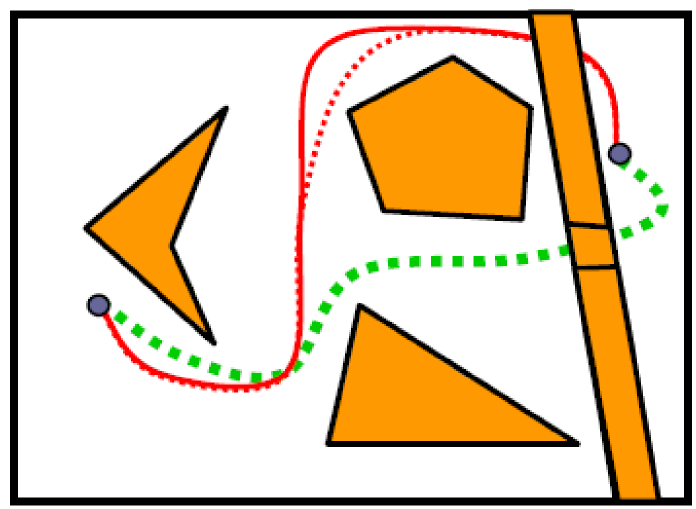
- **A** complete motion planner **always returns a solution when one exists and indicates that no such solution exists otherwise.**
- **Most motion planning problems are hard, meaning that complete planners take** exponential time **in the number of degrees of freedom, moving objects, etc.**

Practical Algorithms

- Theoretical algorithms **strive for completeness and low** worst-case **complexity**
  - **Difficult to implement**
  - **Not robust**
- Heuristic **algorithms strive for efficiency in commonly encountered situations.**
  - **No performance guarantee**
- **Practical algorithms with performance guarantees**
  - **Weaker forms of completeness**
  - **Simplifying assumptions on the space: "exponential time" algorithms that work in practice**

Problem Formulation for Point Robot

- **Input**
  - **Robot represented as a** point **in the** plane
  - **Obstacles represented as polygons**
  - **Initial and goal positions**
- **Output**
  - **A collision-free path between the initial and goal positions**



Framework

continuous representation

(configuration space formulation)

↓

discretization

(random sampling, processing critical geometric events)

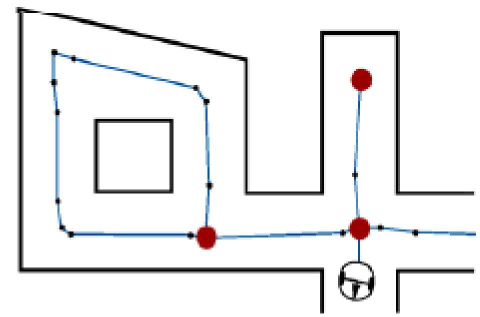
↓

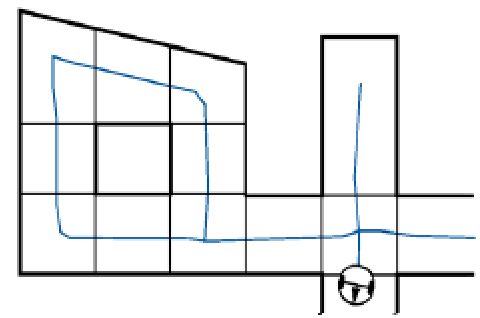
graph searching

(breadth-first, best-first, A*)

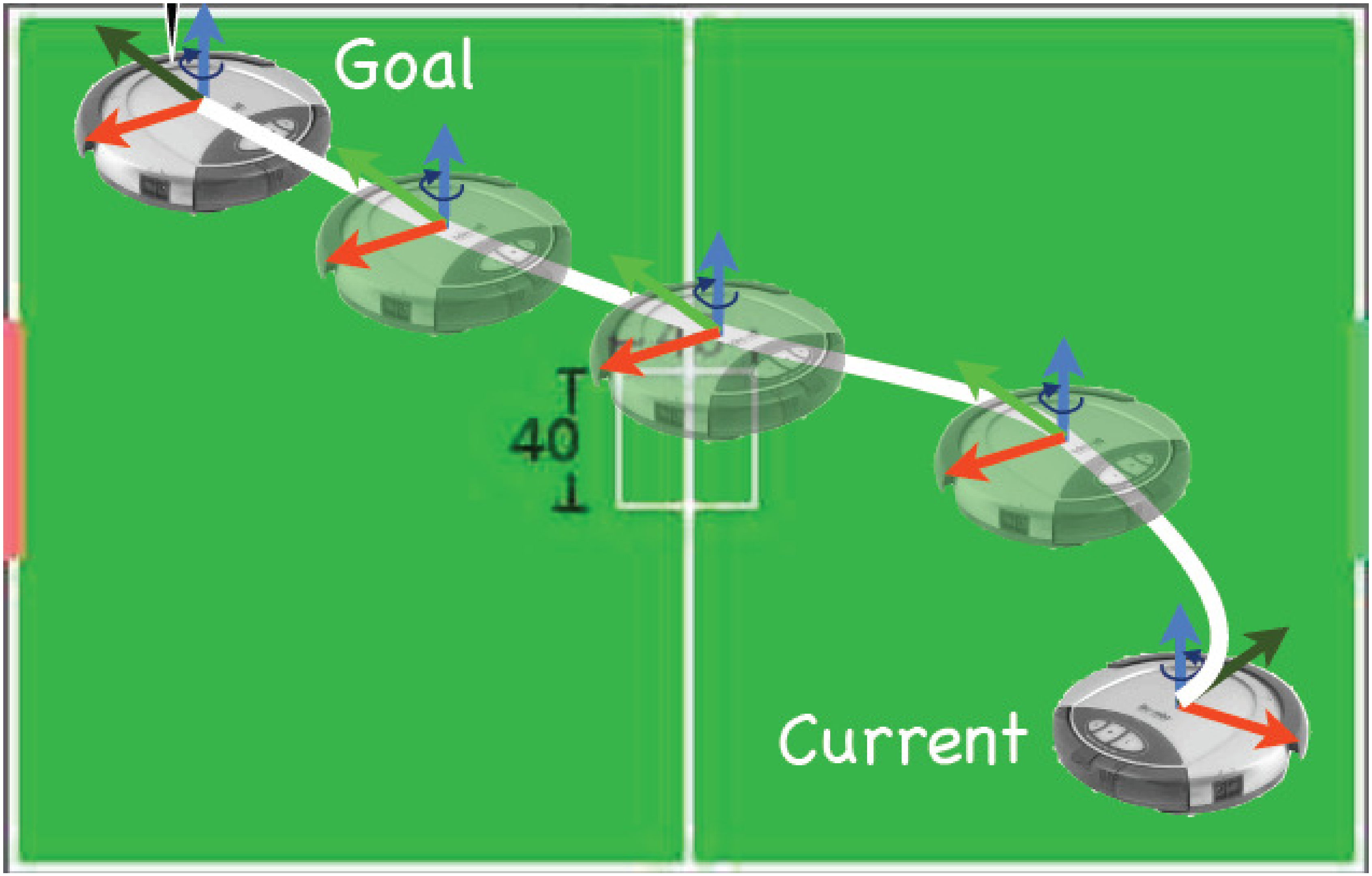
Discretization Methods

- Full Grid
- Visibility Graph
- Voronoi Diagram
- Cell Decomposition
- Probabilistic Roadmaps

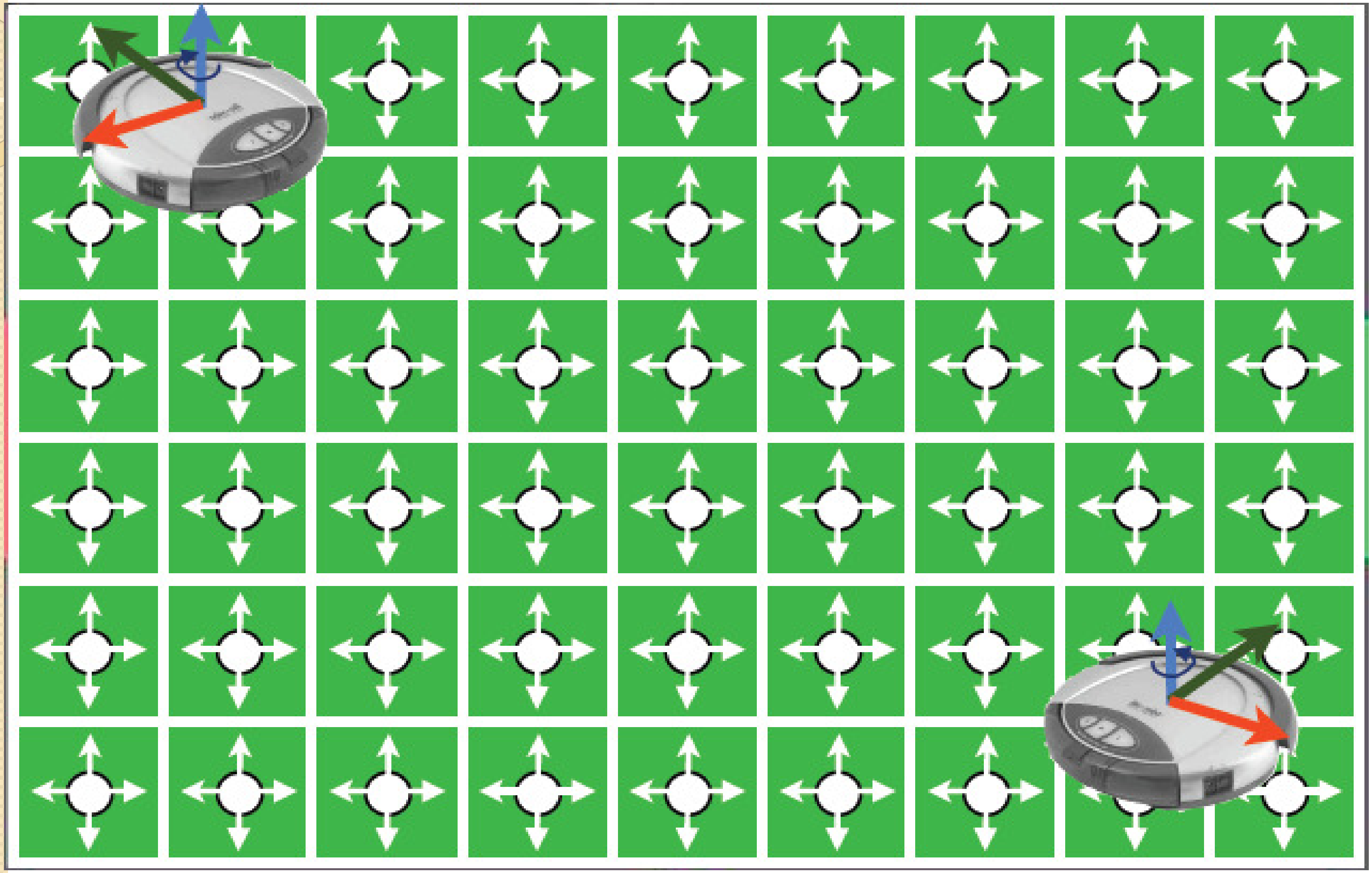




Full Grid

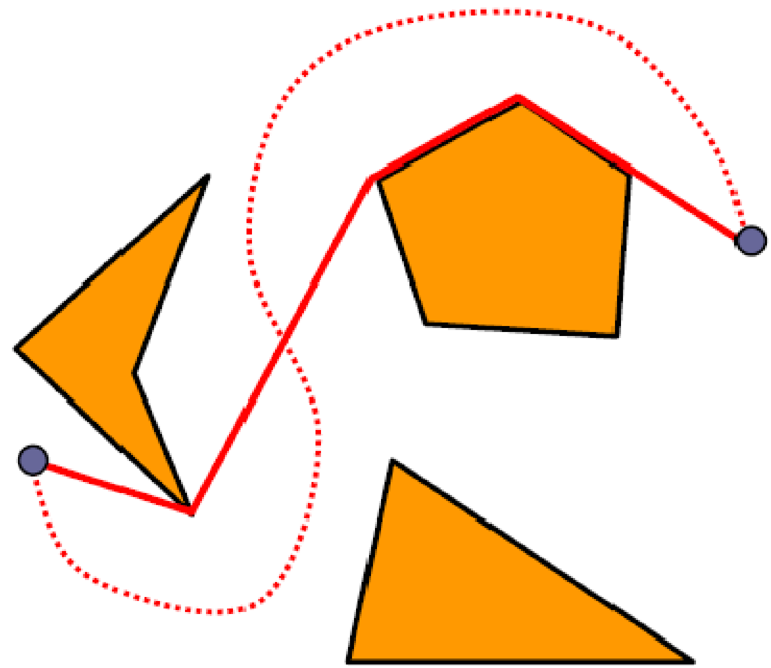


Full Grid

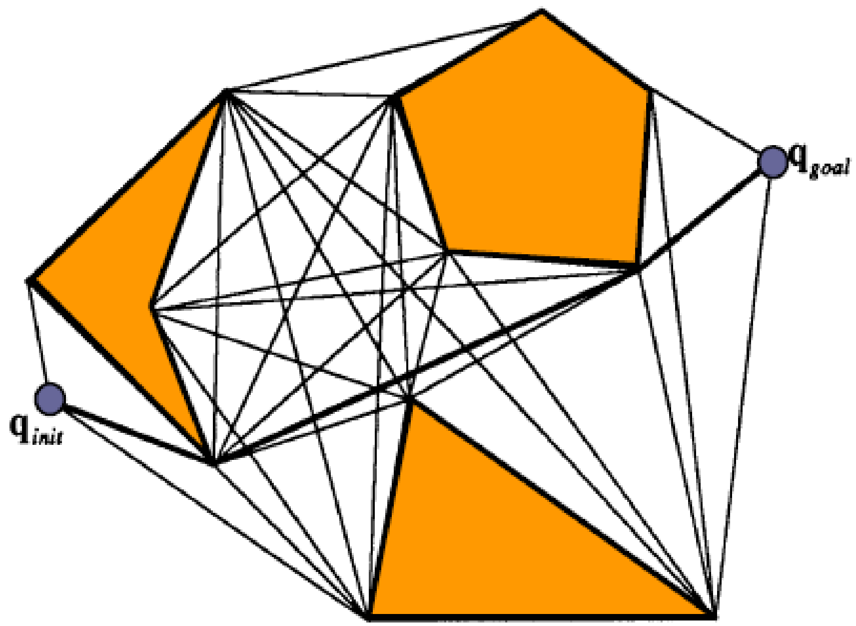


Visibility Graph Method

- Observation: **If there is a collision-free path between two points, then there is a polygonal path that bends only at the obstacles vertices.**
- **Why?**
  - **Any collision-free path can be transformed into a polygonal path that bends only at the obstacle vertices.**
- **A** polygonal path **is a piecewise linear curve.**



Visibility Graph



- **A** visibility graph **is a graph such that**
  - **Nodes:** q_{init}, q_{goal}, **or an obstacle vertex.**
  - **Edges: An edge exists between nodes** u **and** v **if the line segment between** u **and** v **is an obstacle edge or it does not intersect the obstacles.**

A Simple Algorithm for Building Visibility Graphs

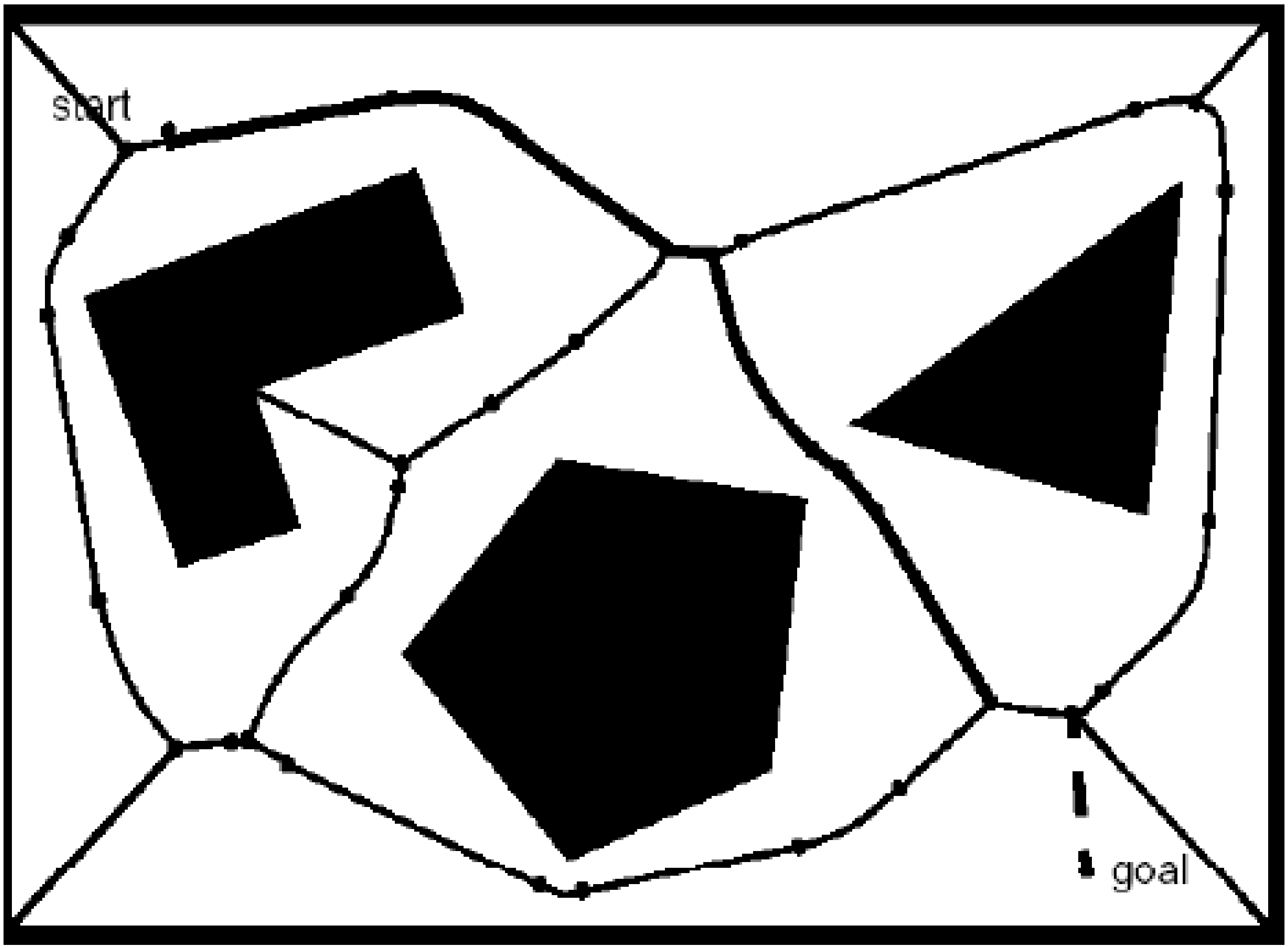
```
Input: q_init, q_goal, polygonal obstacles
Output: visibility graph G

1: for every pair of nodes u,v
2:  if segment(u,v) is an obstacle edge then
3:    insert edge(u,v) into G;
4:  else
5:    for every obstacle edge e
6:      if segment(u,v) intersects e
7:        go to (1);
8:    insert edge(u,v) into G.
```

Computational Efficiency

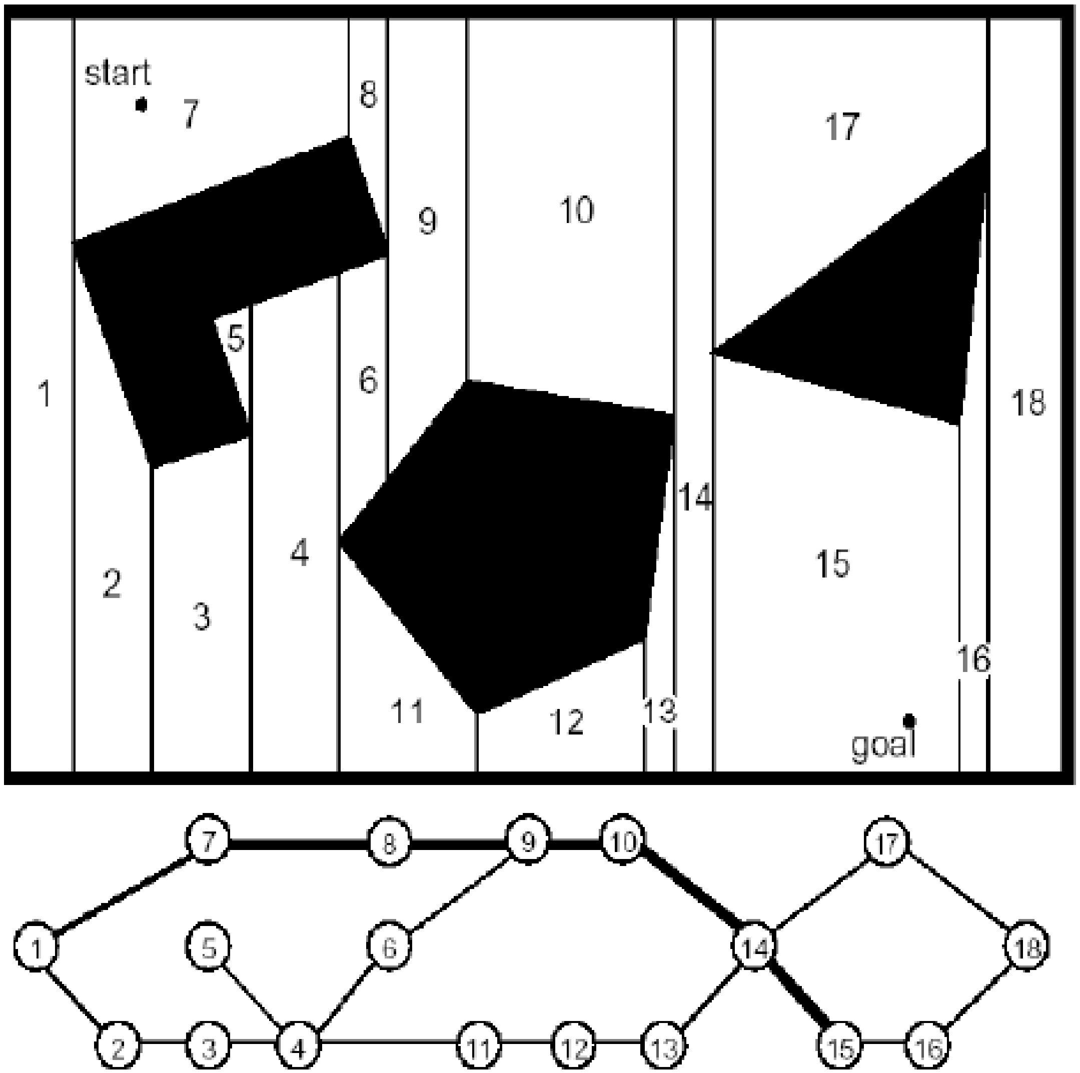
```
1: for every pair of nodes u,v                           O(n²)
2:  if segment(u,v) is an obstacle edge then             O(n)
3:    insert edge(u,v) into G;
4:  else
5:    for every obstacle edge e                          O(n)
6:      if segment(u,v) intersects e
7:        go to (1);
8:    insert edge(u,v) into G.
```

- **Simple algorithm $O(n^3)$ time**
- **More efficient algorithms**
  - **Rotational sweep $O(n^2 \log n)$ time**
  - **Optimal algorithm $O(n^2)$ time**
  - **Output sensitive algorithms**
- **$O(n^2)$ space**

Voronoi Diagram

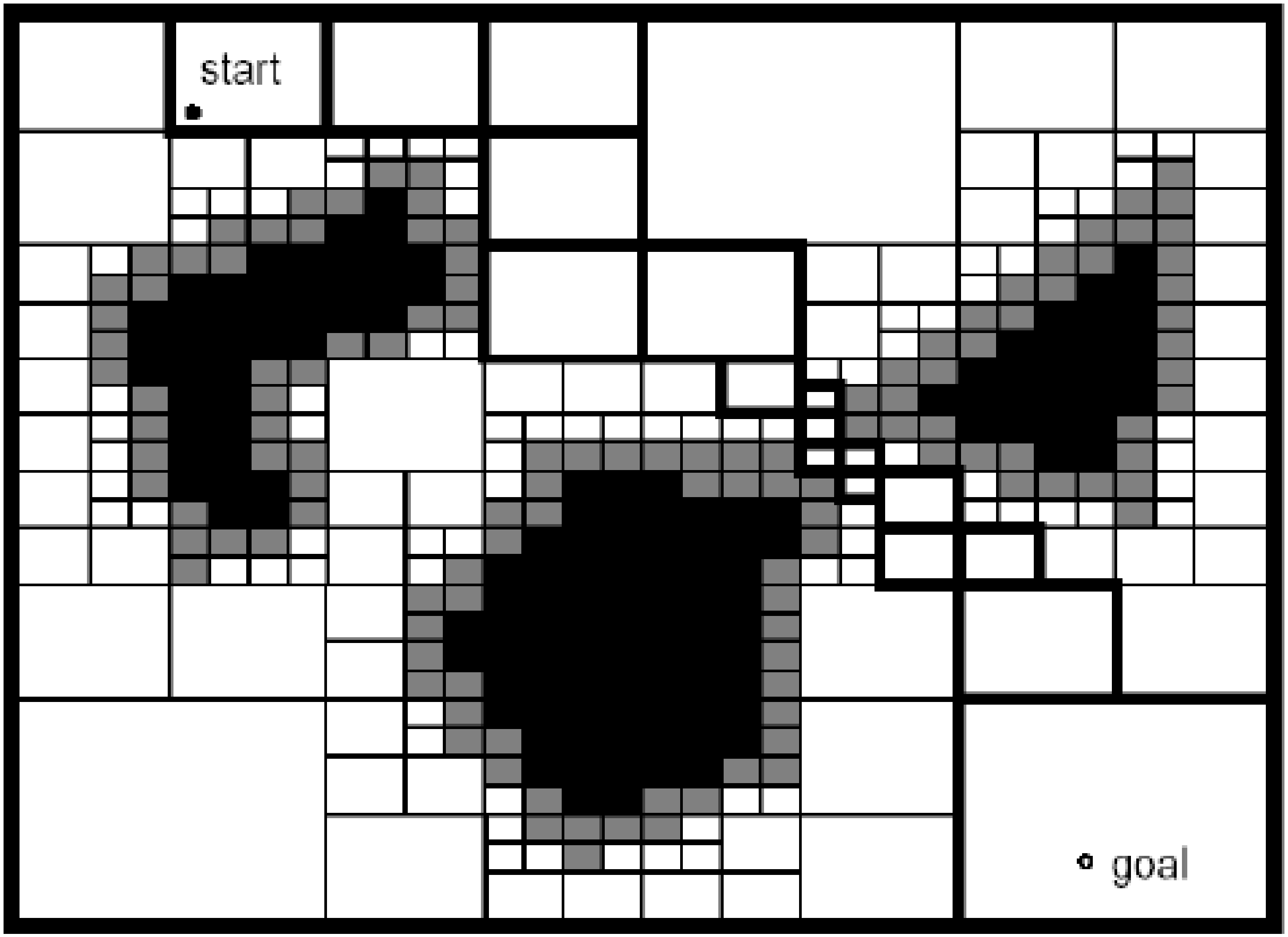


Cell Decomposition



Ashutosh Saxena

Approximate Cell Decomposition



Problem Formulation for General Robot

- What is the state space?
- Does it include other objects
  - E.g., when interacting with other objects

- Full Grid discretization is huge (consider 7-dof arm, 100^7 possibilities)
- How should I discretize?

PRM: construction phase

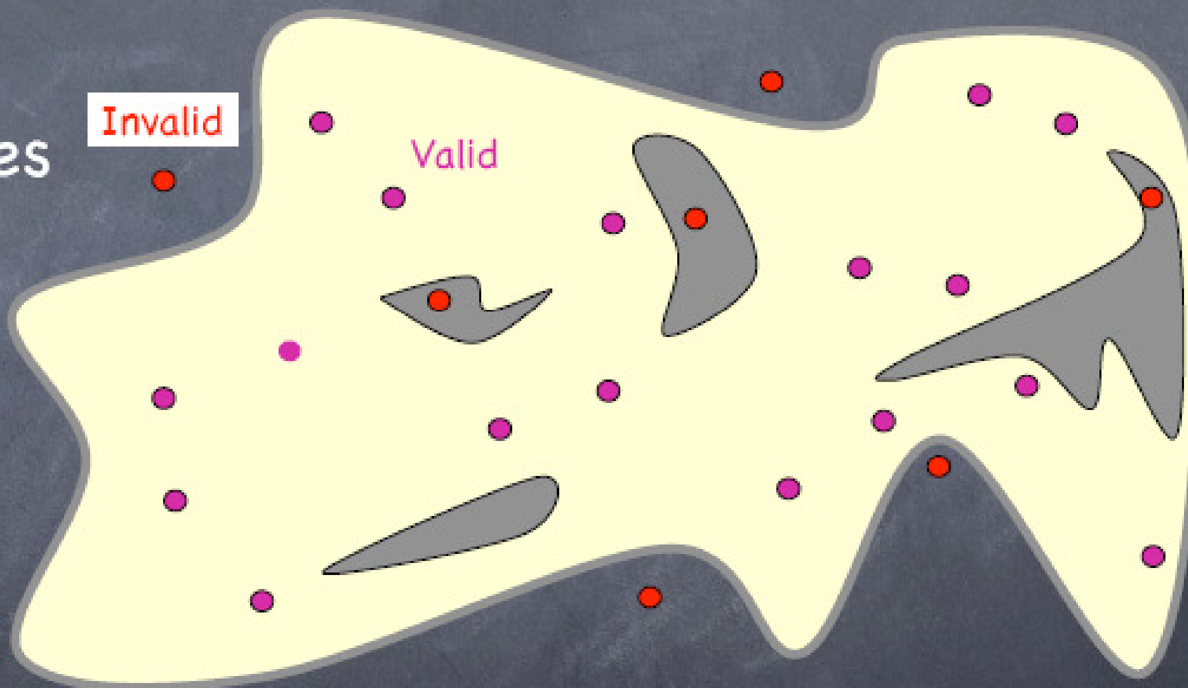
- Select sample poses at random
- Eliminate invalid poses
- Connect neighboring poses

PRM: construction phase

- Select sample poses at random
- Eliminate invalid poses

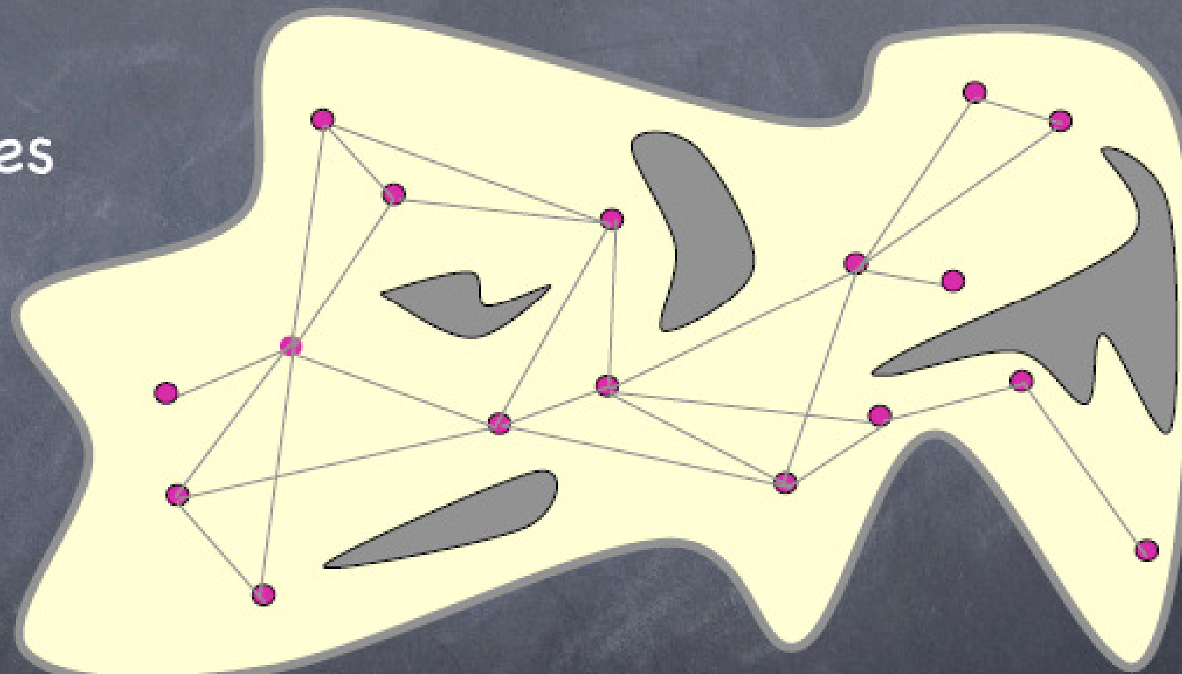
Collision detection

- Connect neighboring poses



PRM: construction phase

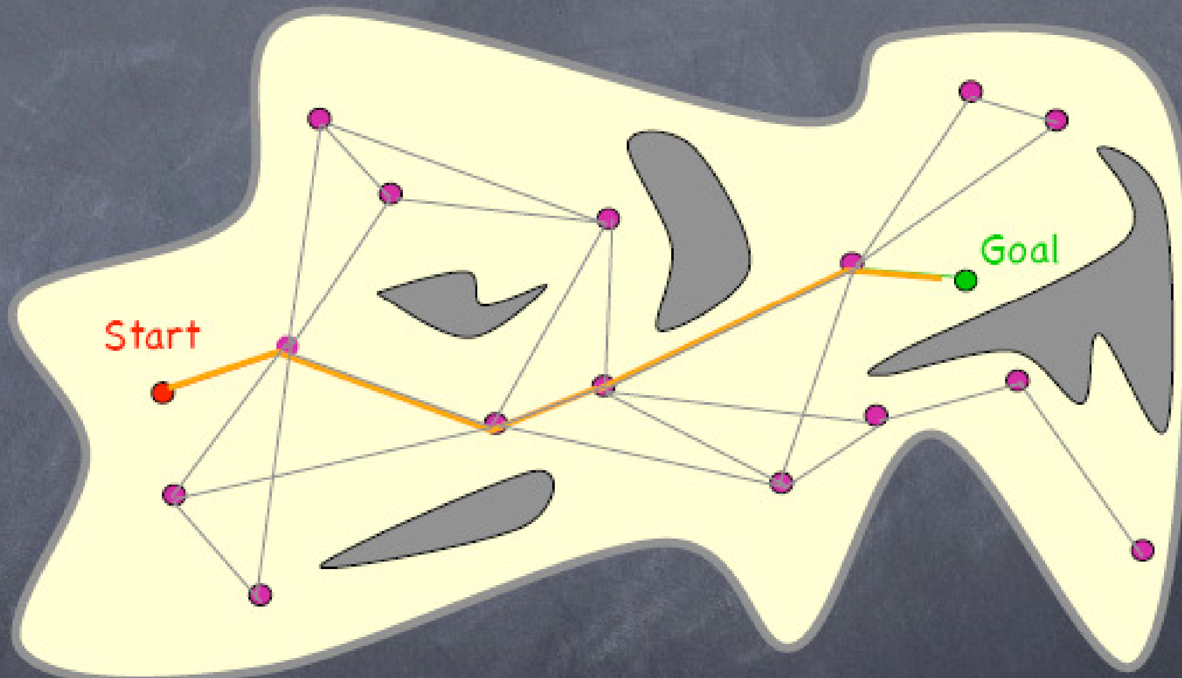
- Select sample poses at random
- Eliminate invalid poses
- Connect neighboring poses



Threshold neighborhood radius or population

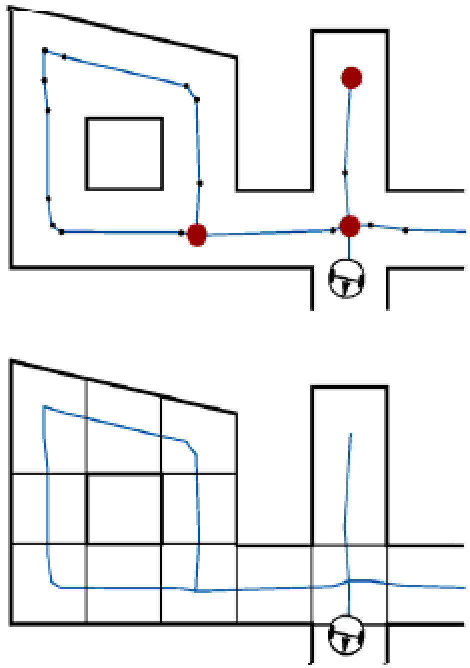
PRM: query phase

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



Discretization Methods

- Full Grid
- Visibility Graph
- Voronoi Diagram
- Cell Decomposition
- Probabilistic Roadmaps



Framework

continuous representation
(configuration space formulation)

↓

discretization
(random sampling, processing critical geometric events)

↓

graph searching
(breadth-first, best-first, A*)

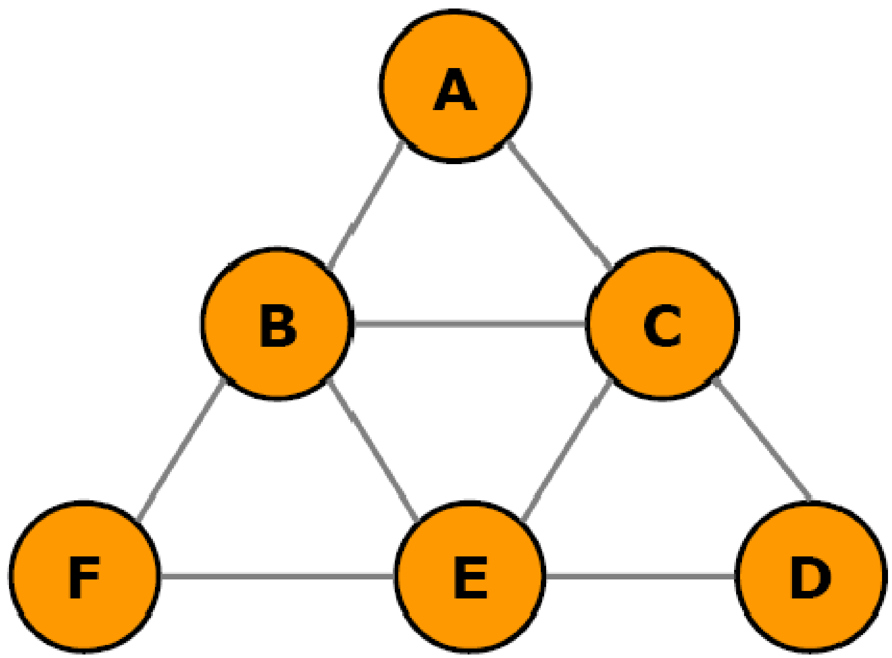
Summary

- **Discretize the space by constructing visibility graph**
- **Search the visibility graph with breadth-first search**

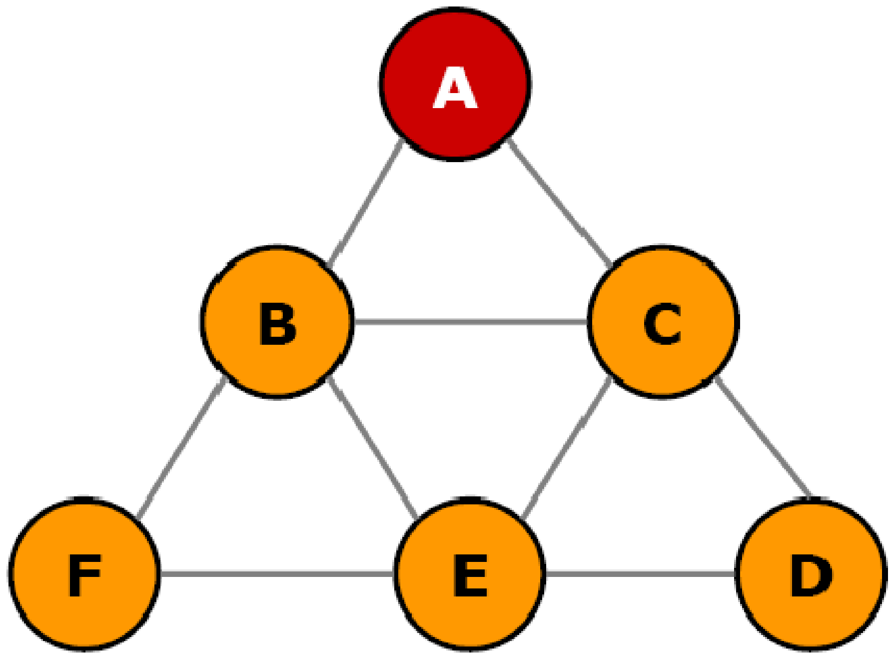
Search: Overview

- Breadth-first search
- Depth-first search
- A* star search
- Dijkstra

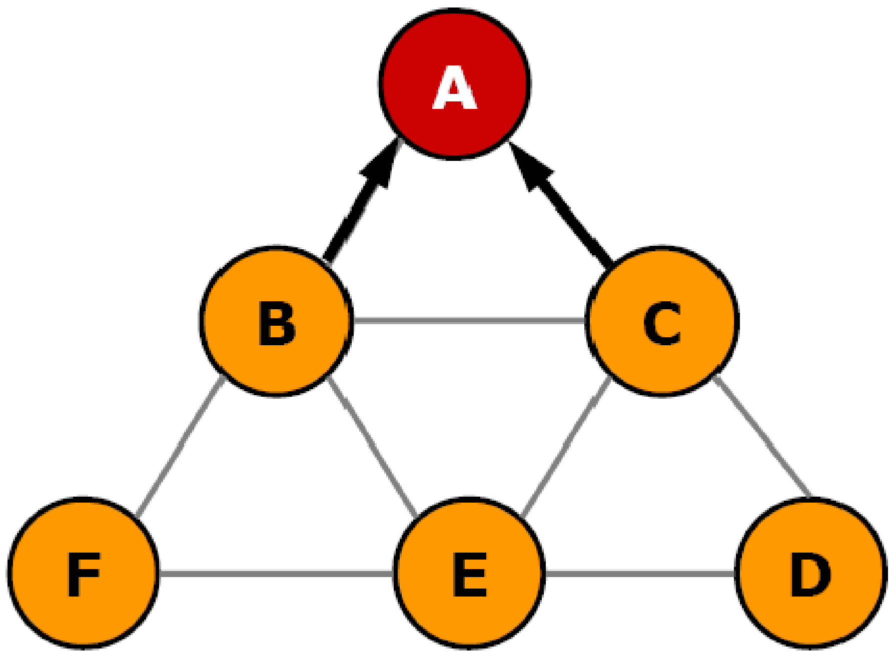
Breadth-First Search



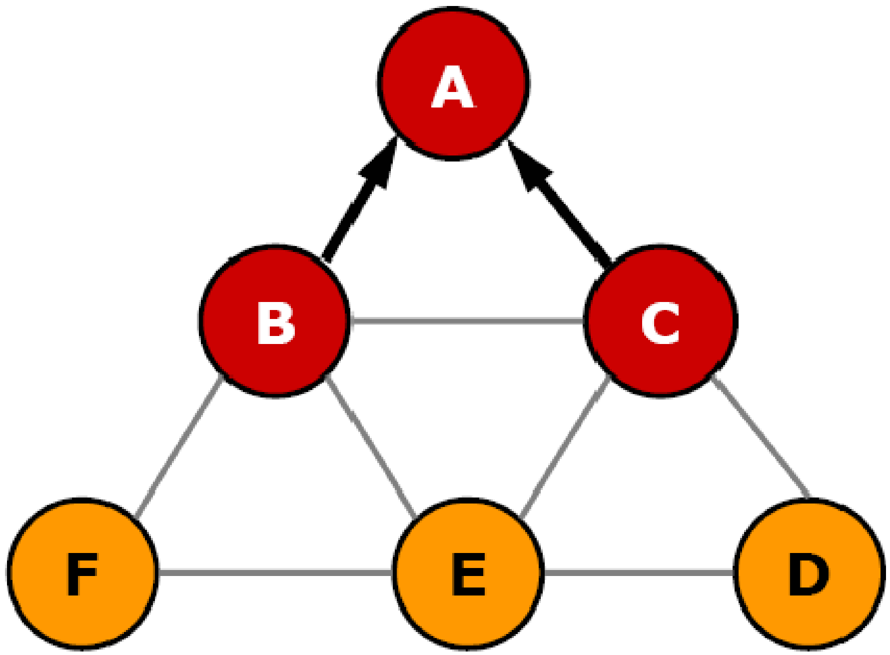
Breadth-First Search



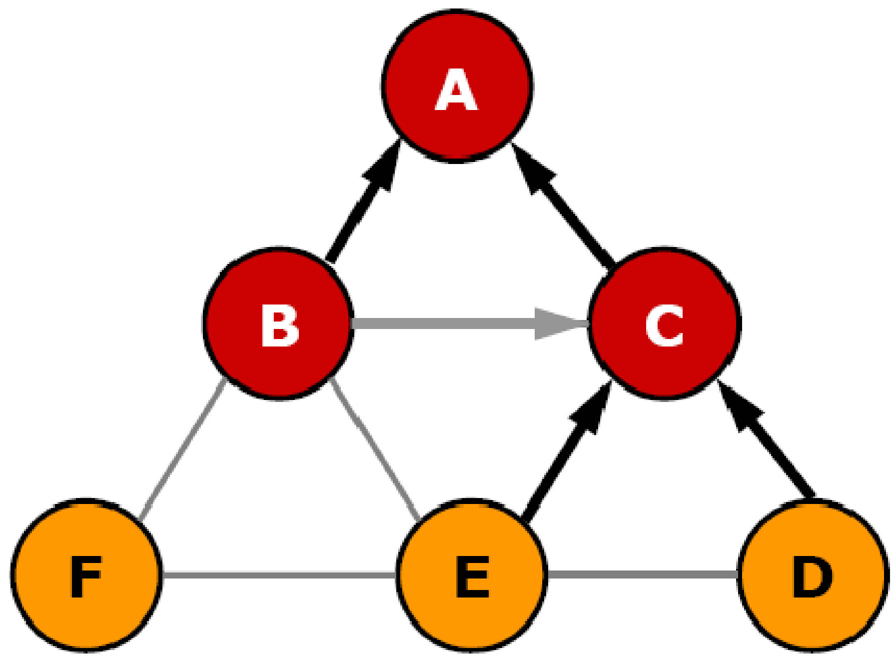
Breadth-First Search



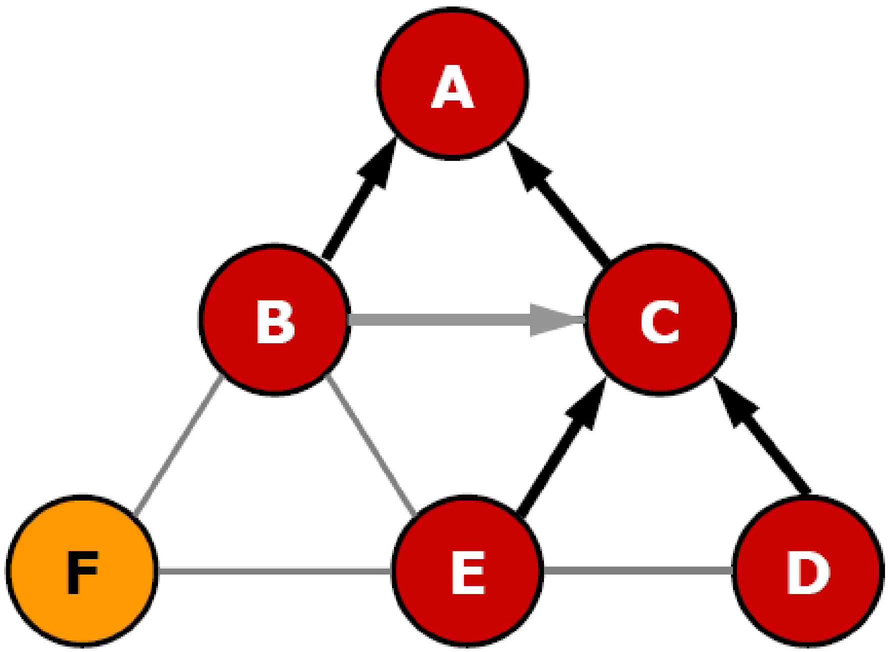
Breadth-First Search



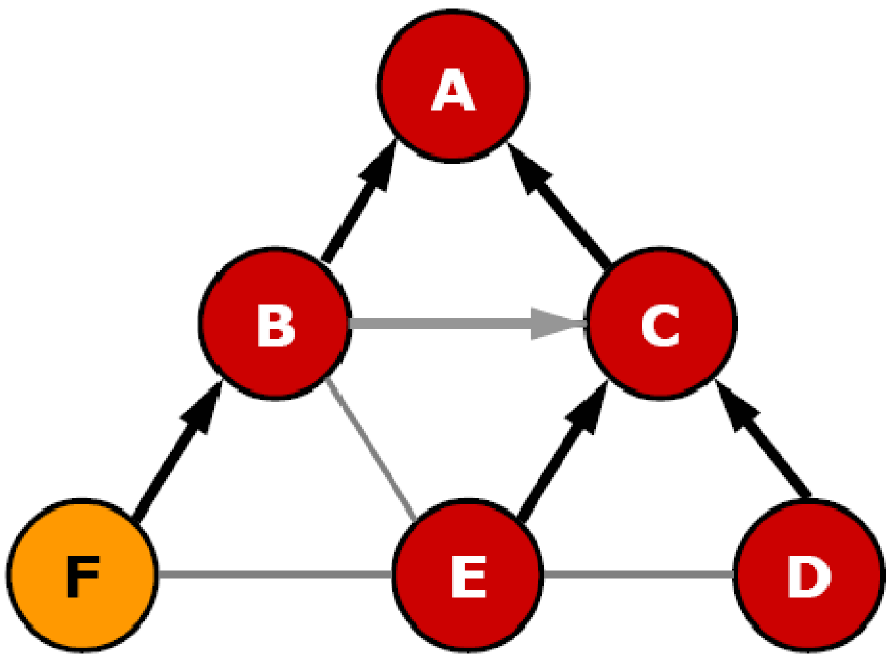
Breadth-First Search



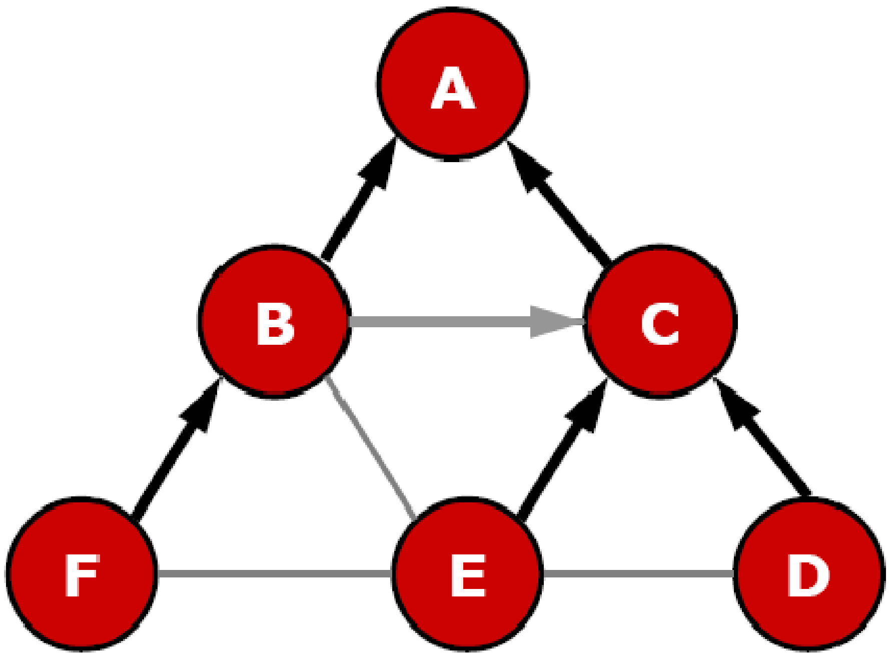
Breadth-First Search



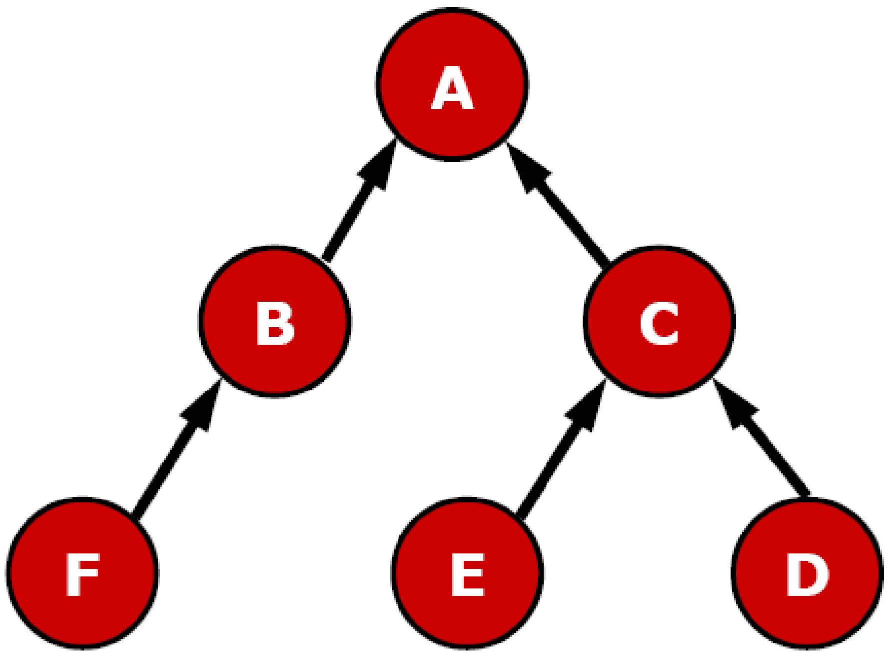
Breadth-First Search



Breadth-First Search



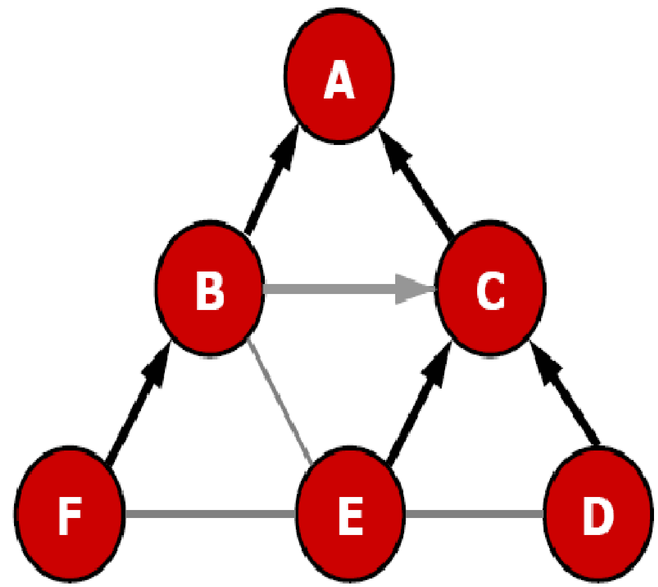
Breadth-First Search



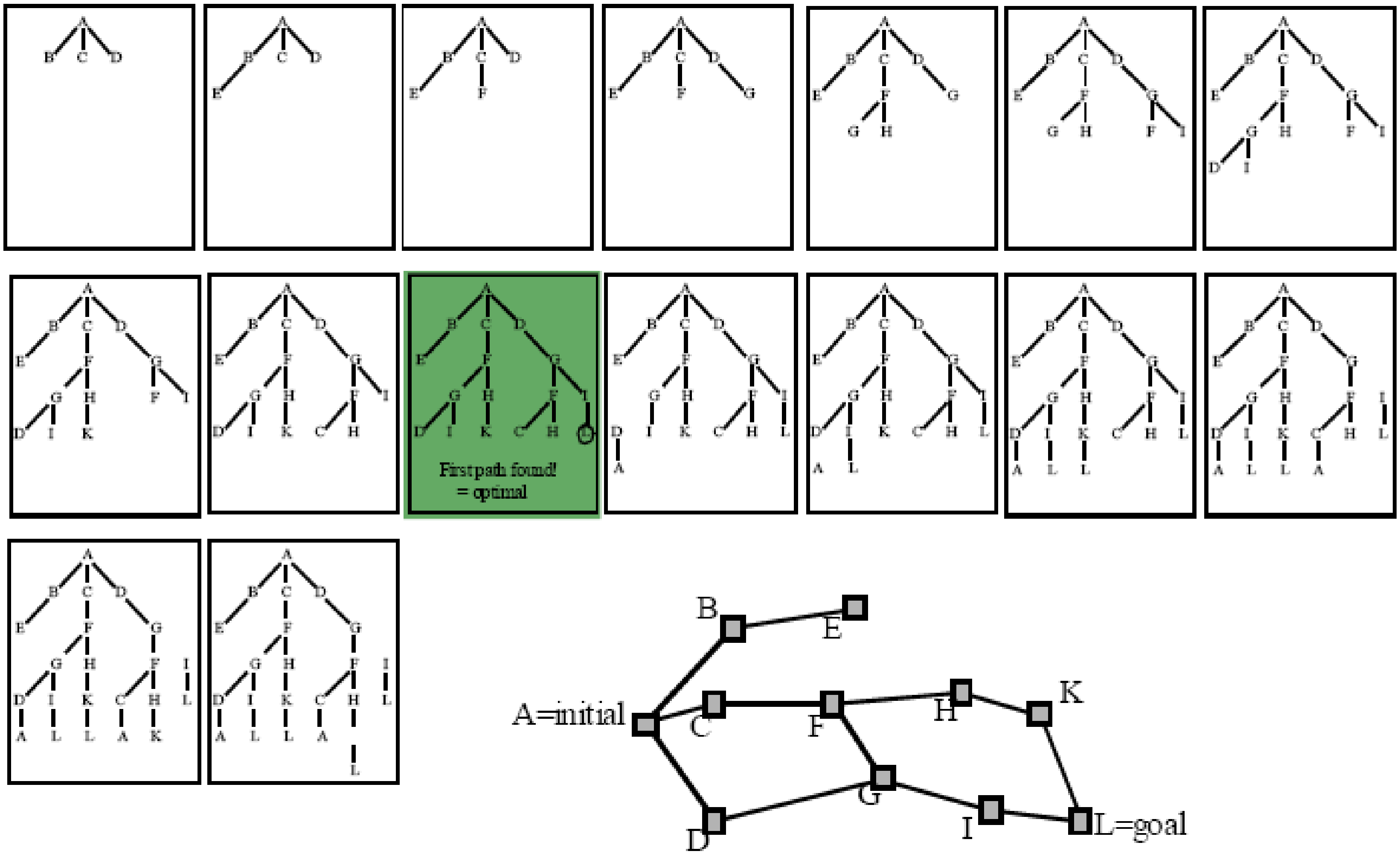
Breadth-First Search

Input: q_{init}, q_{goal}, visibility graph G
Output: a path between q_{init} and q_{goal}

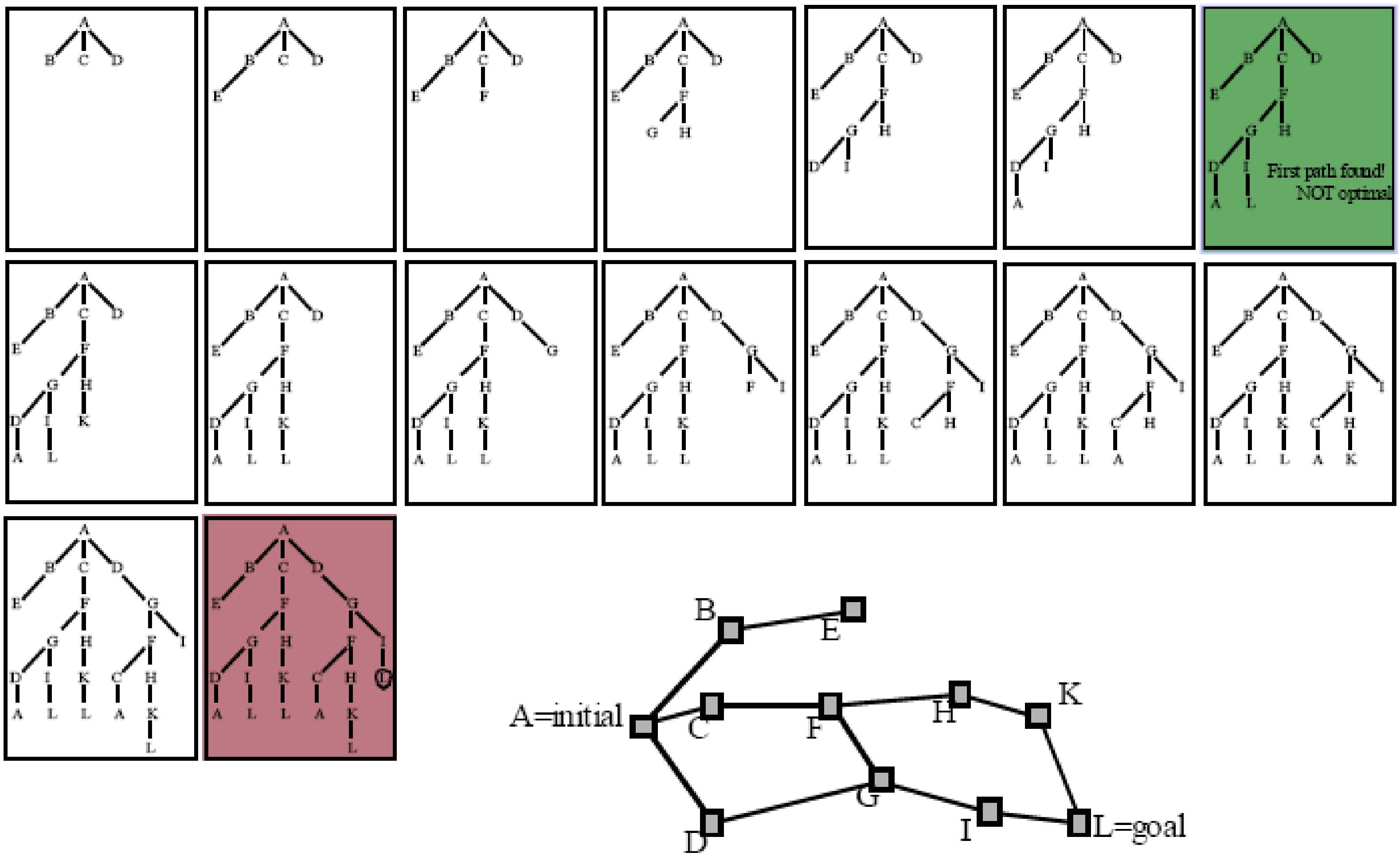
```
Q = new queue;
Q.enqueue(q_init);
mark q_init as visited;
while Q is not empty
  curr = Q.dequeue();
  if curr == q_goal then
    return curr;
  for each w adjacent to curr
    if w is not visited
      w.parent = curr;
      Q.enqueue(w)
      mark w as visited
```



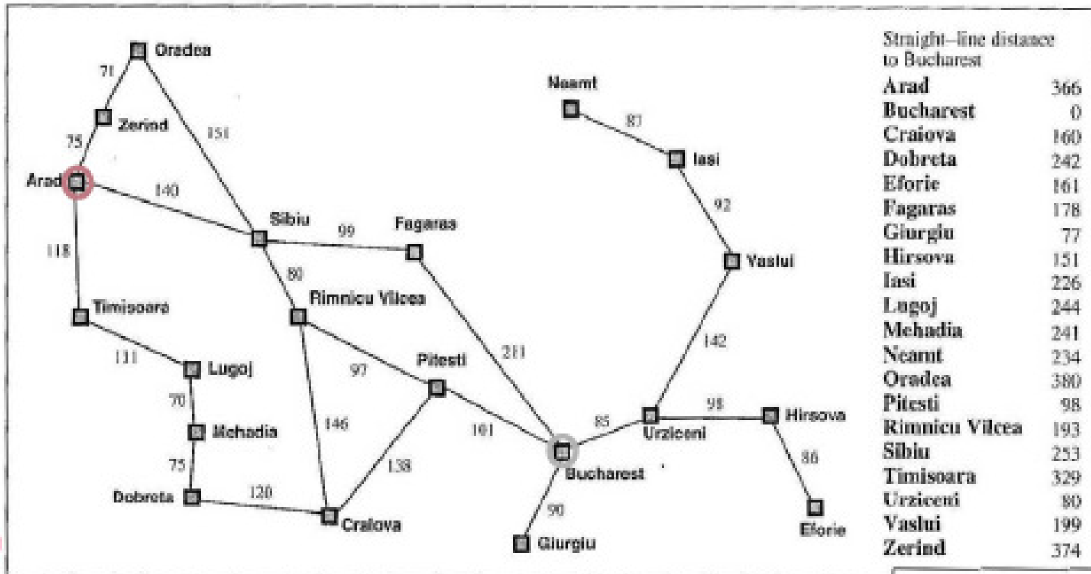
Breadth-first Search



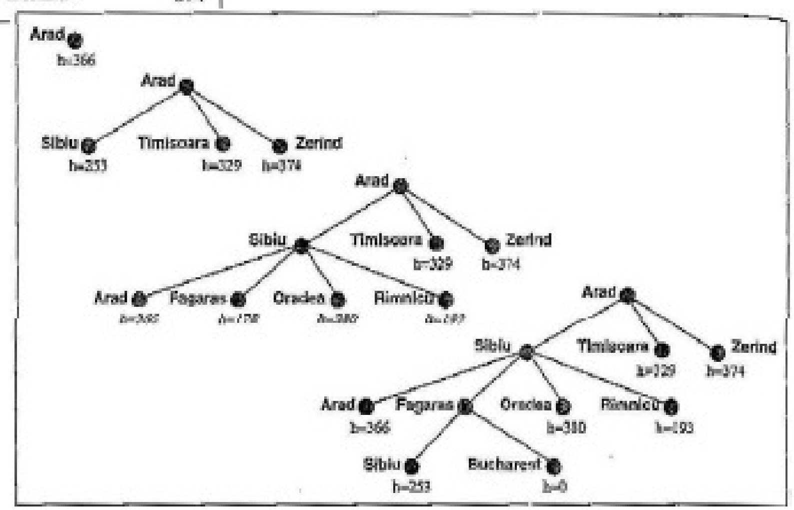
Depth-First Search



A* Search

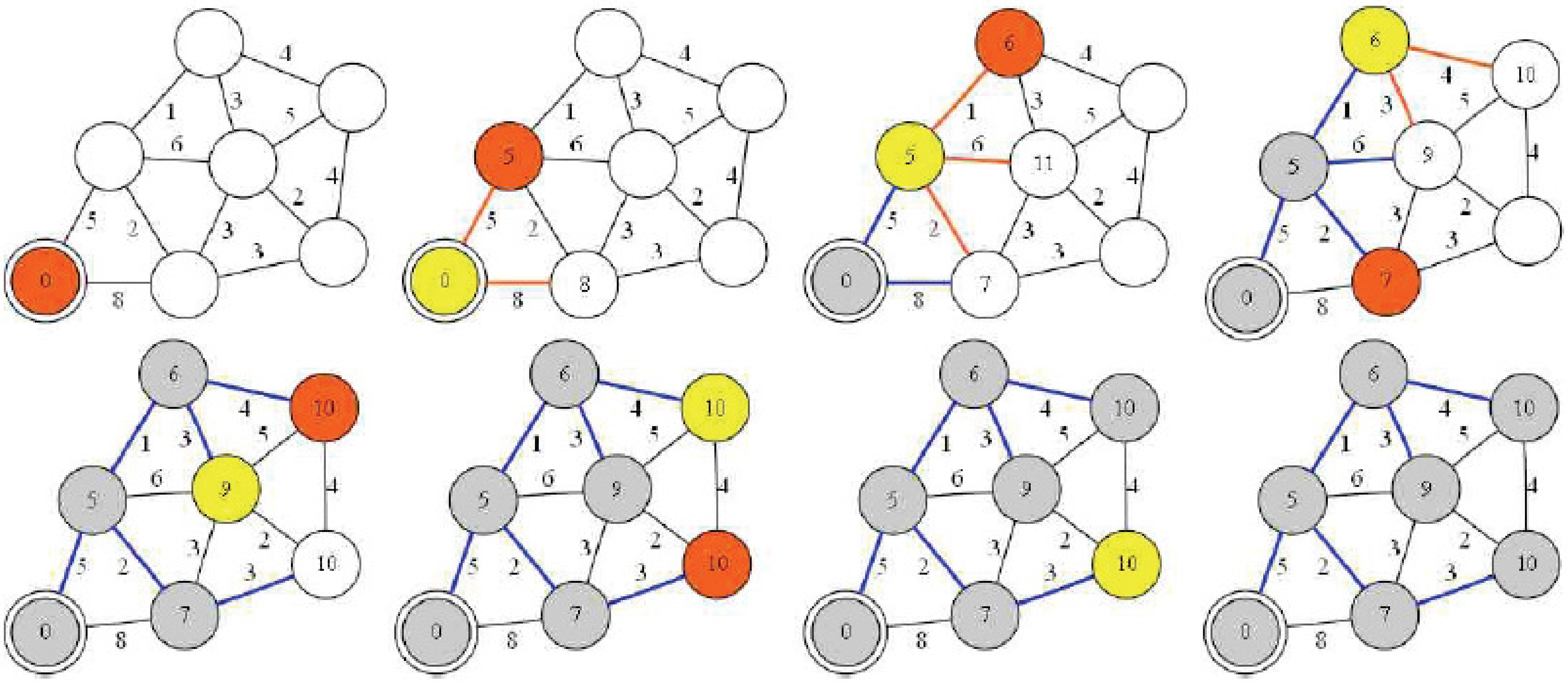


- cost estimate of the cheapest path from state at node n to the goal



Dijkstra's Algorithm:

Given weighted graph and start node, find shortest path back to start for all nodes

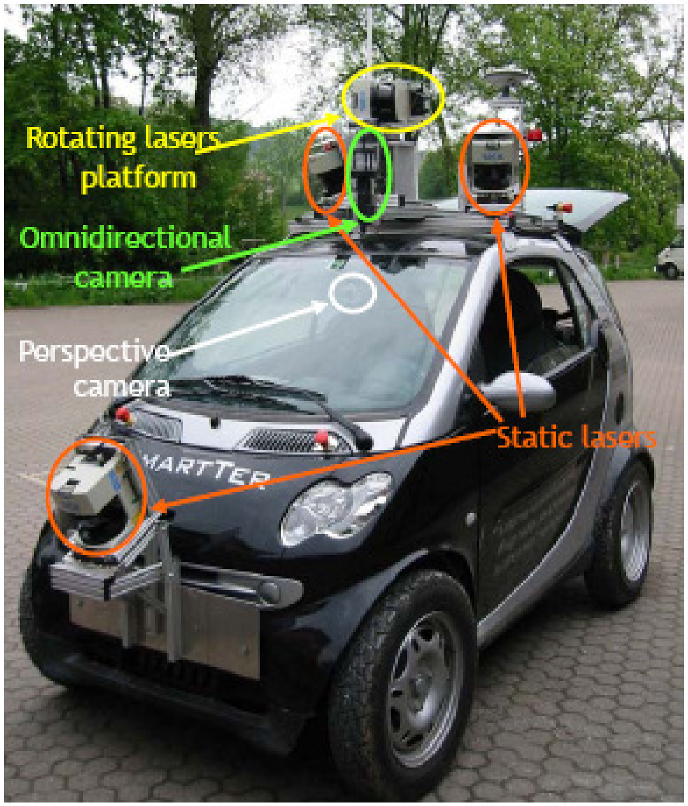


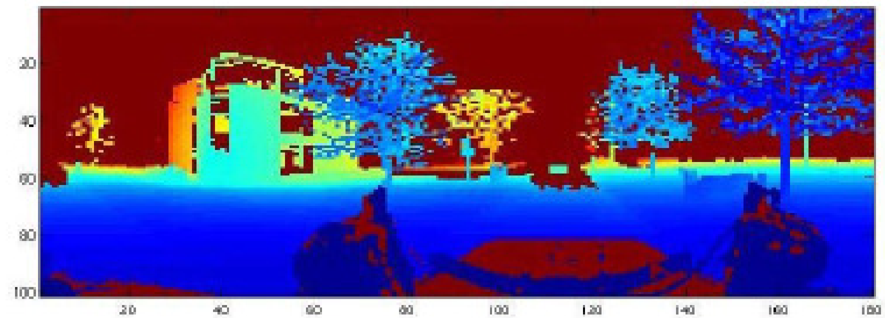
Explore each node one-by-one, calculate path distance from explored tree

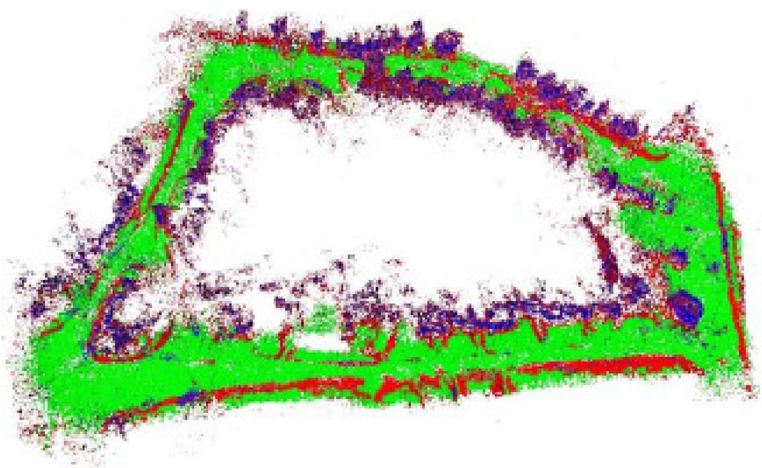
Given *perfect* sensing and map, one can do:

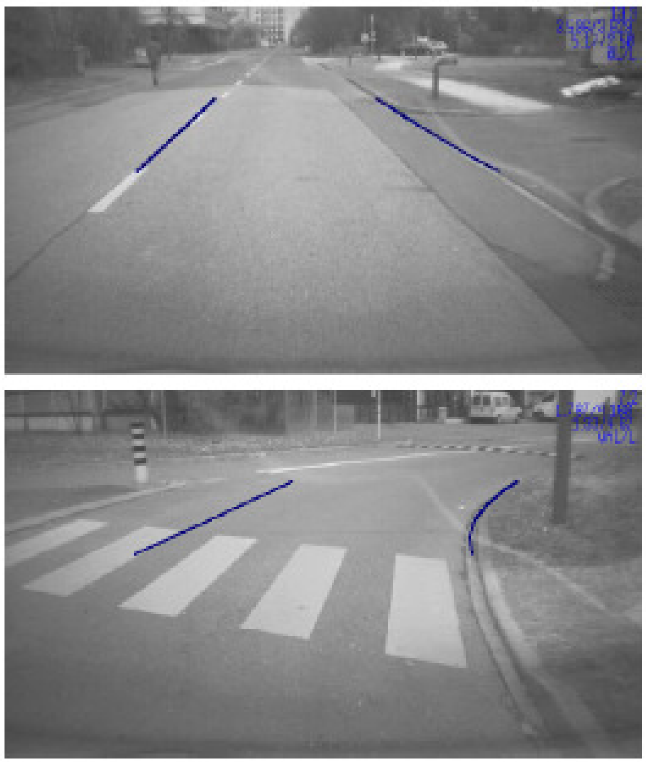
1. Planning
    1. Discretize
    2. Search
2. Control
    1. Kinematics
    2. PID control

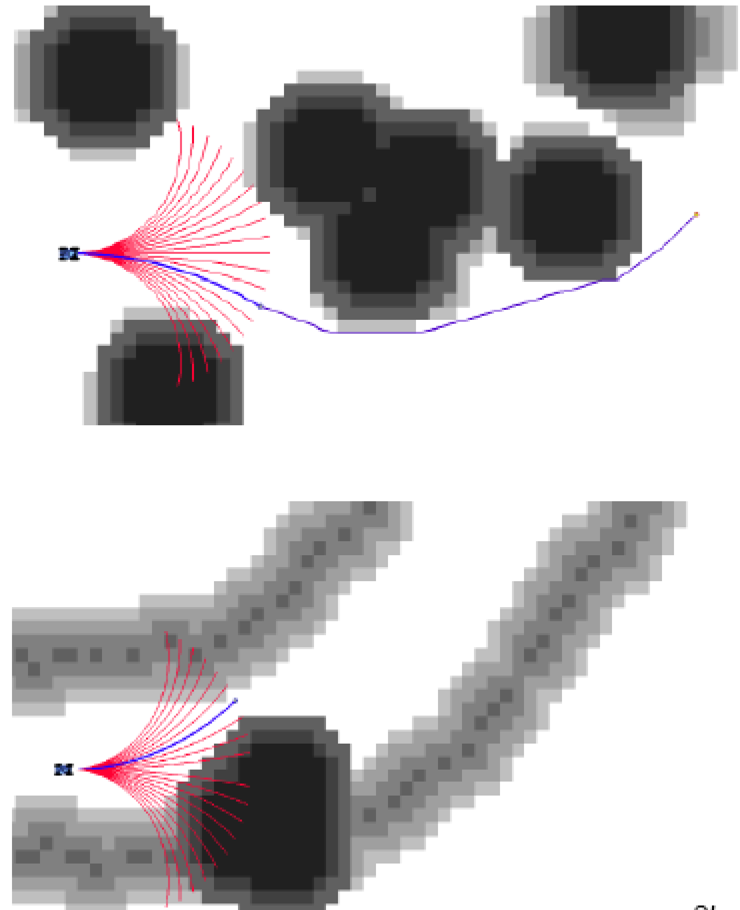
Example video


Rotating lasers
platform
Omnidirectional
camera
Perspective
camera
Static lasers









SL

Rapidly-exploring Random Trees

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

```
EXTEND(T, q)
q_near ← NEAREST_NEIGHBOR(q, T);
if NEW_CONFIG(q, q_near, q_new) then
    T.add_vertex(q_new);
    T.add_edge(q_near, q_new);
    if q_new = q then
        Return Reached;
    else
        Return Advanced;
Return Trapped;
```

Figure 2: The basic RRT construction algorithm.

[Kuffner, LaValle 2000]

Explore incrementally along straight line path to goal

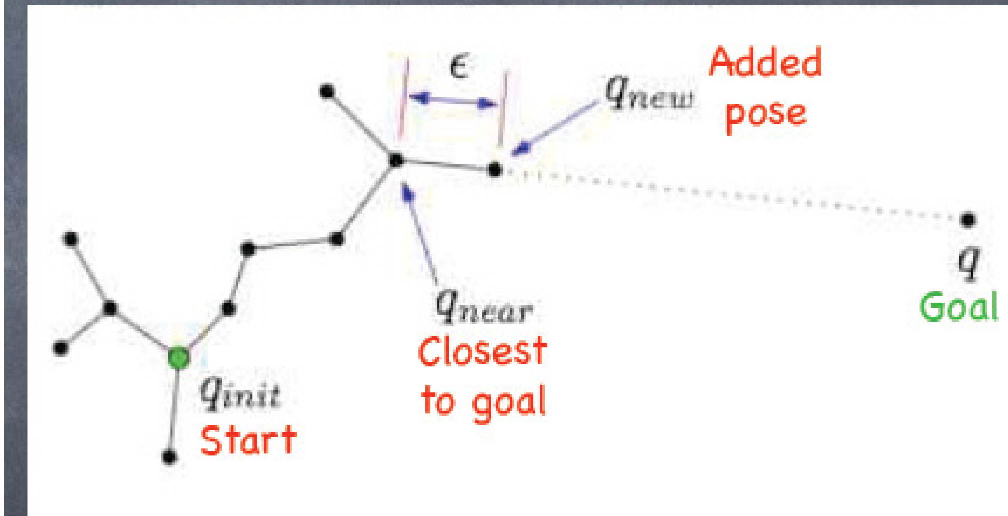


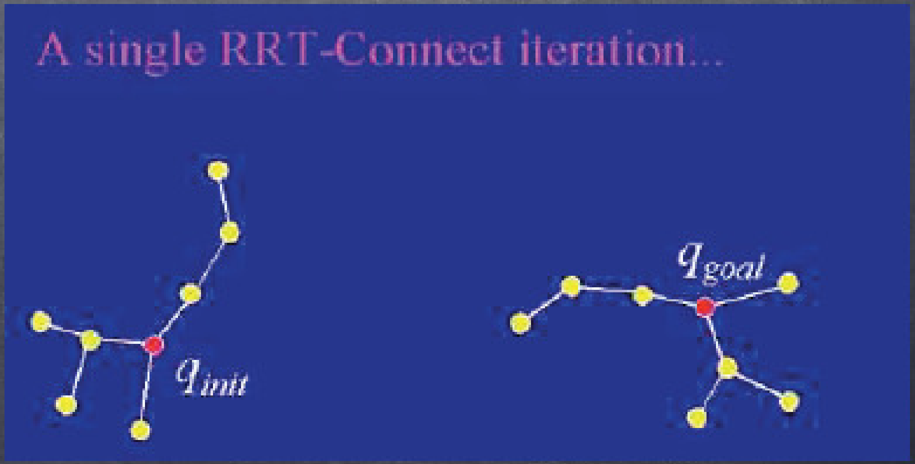
Figure 3: The EXTEND operation.

Single vs. multiple query

RRT

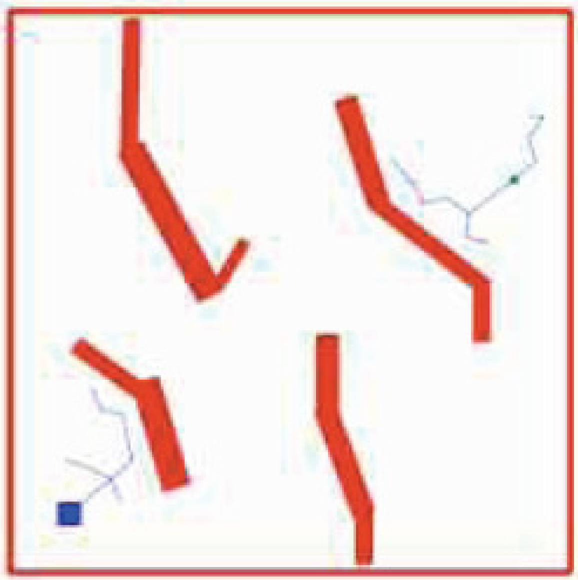
Kuffner's RRT Animations

Extending roadmap



Bidirectional exploration

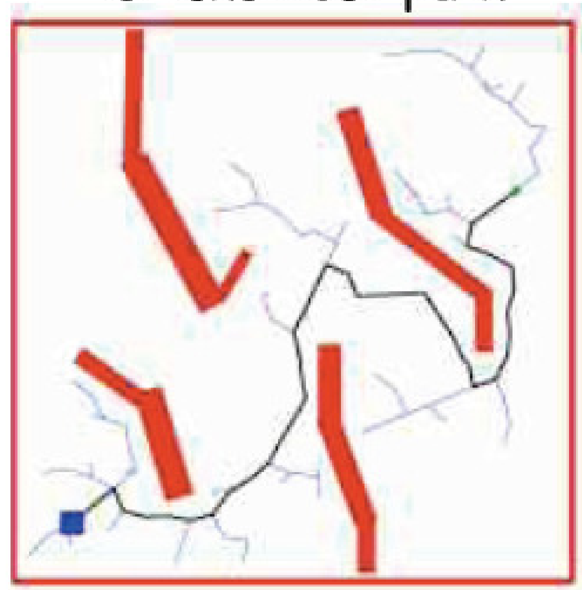
RRT bidirectional search example



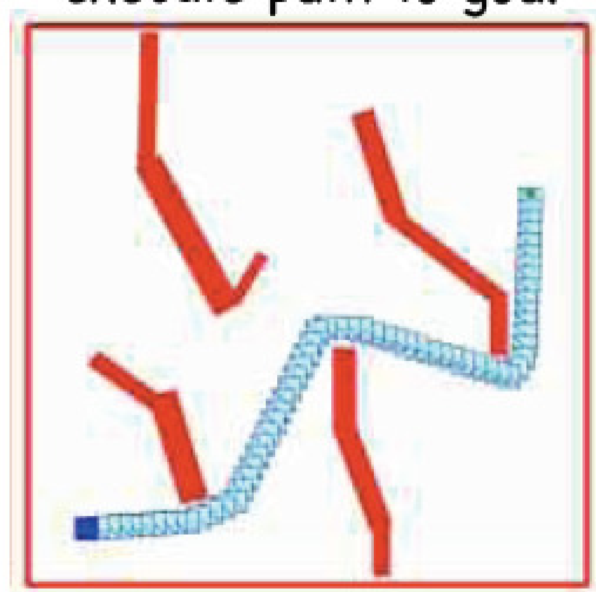
extend graph from
both start and goal

continue exploration until
start and goal connected

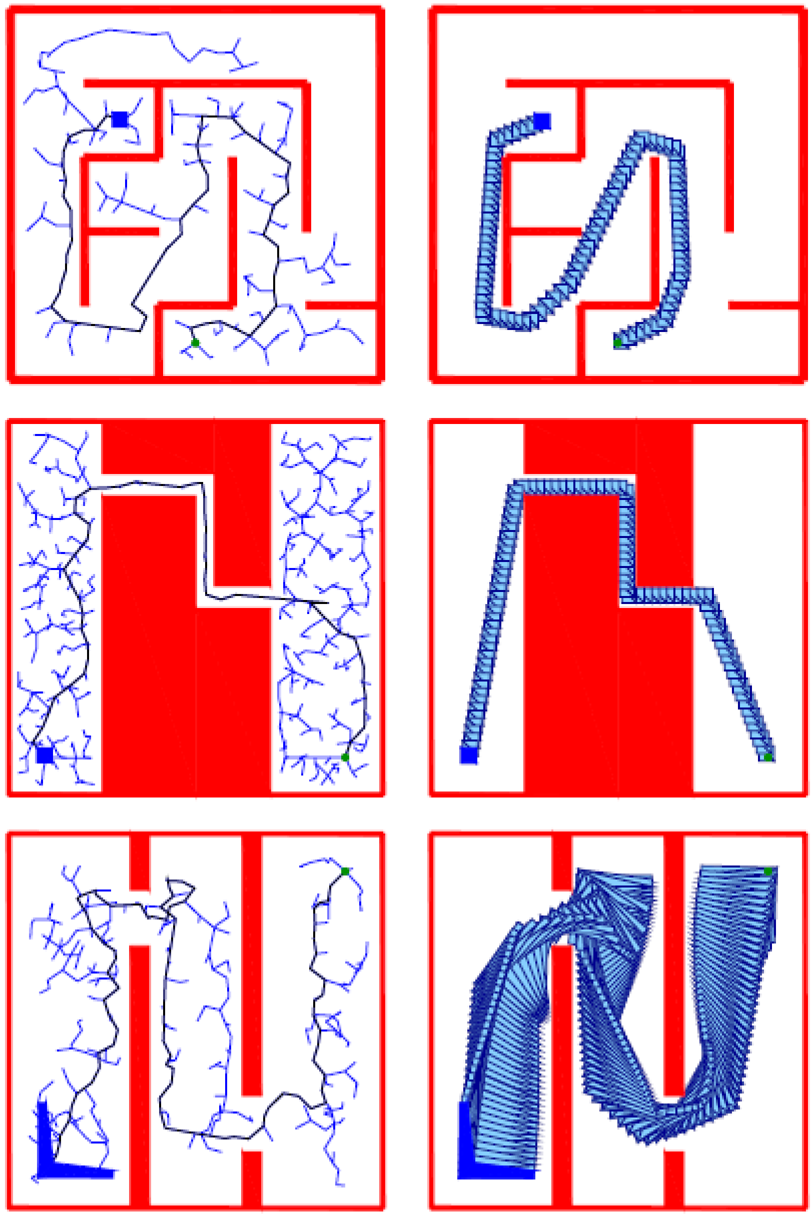
once connected, search
for shortest path



execute path to goal



RRT Examples

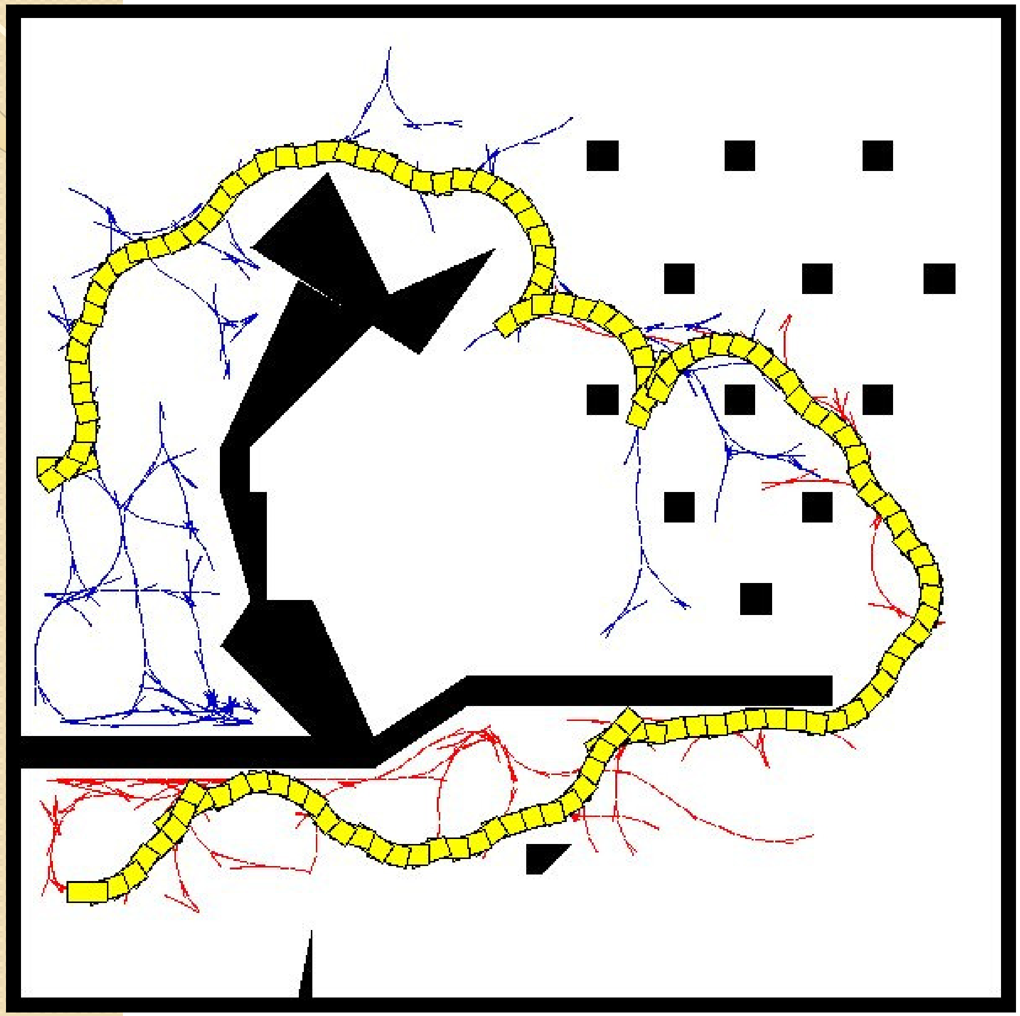


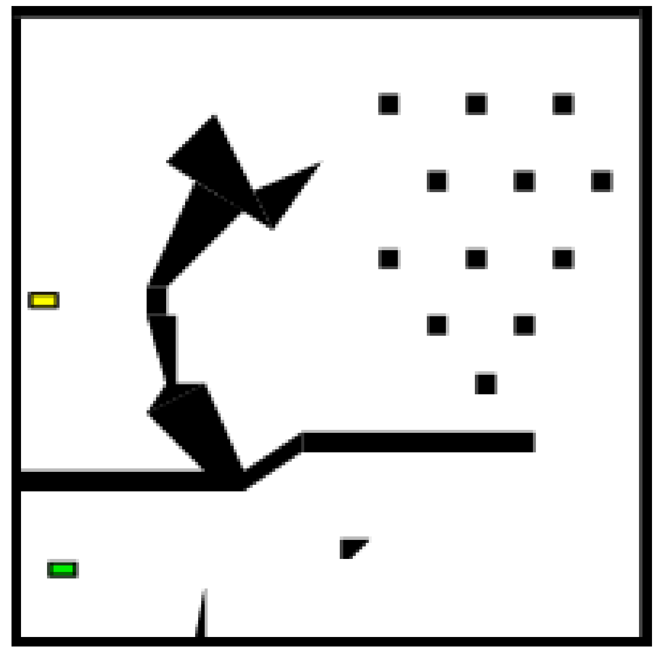
2 DOF maze

2 DOF single passway

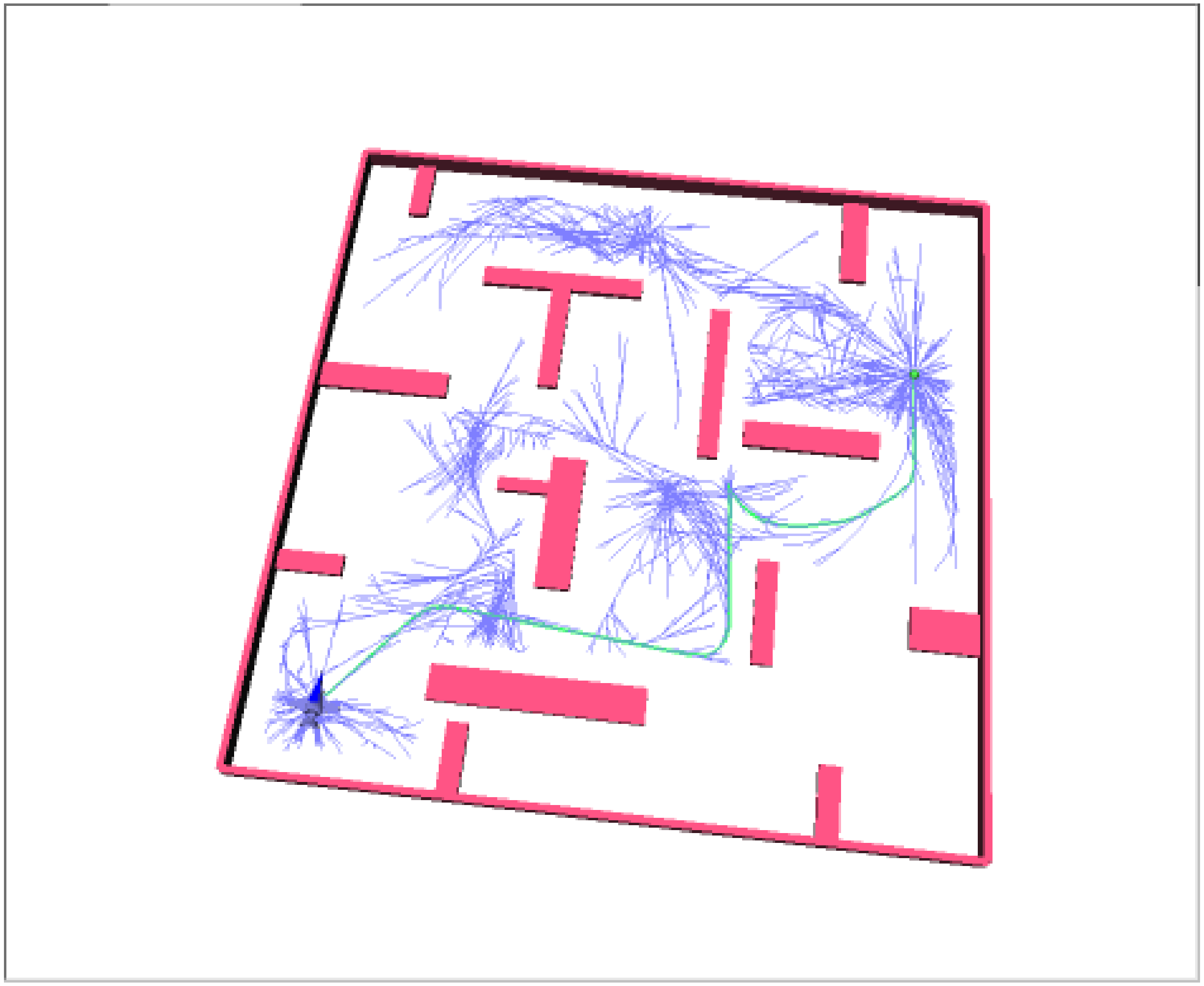
3 DOF single passway

RRT: Car like robot

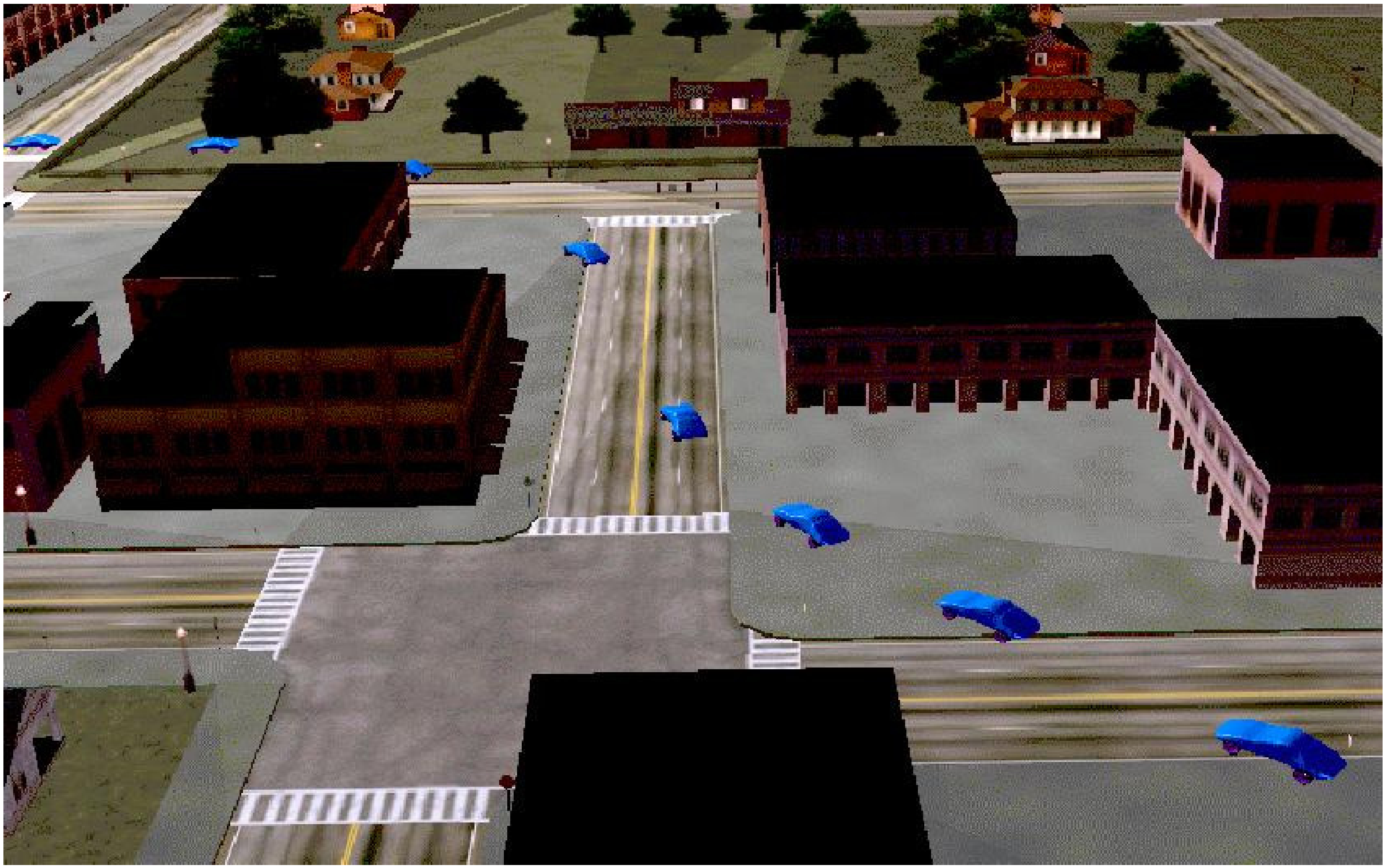




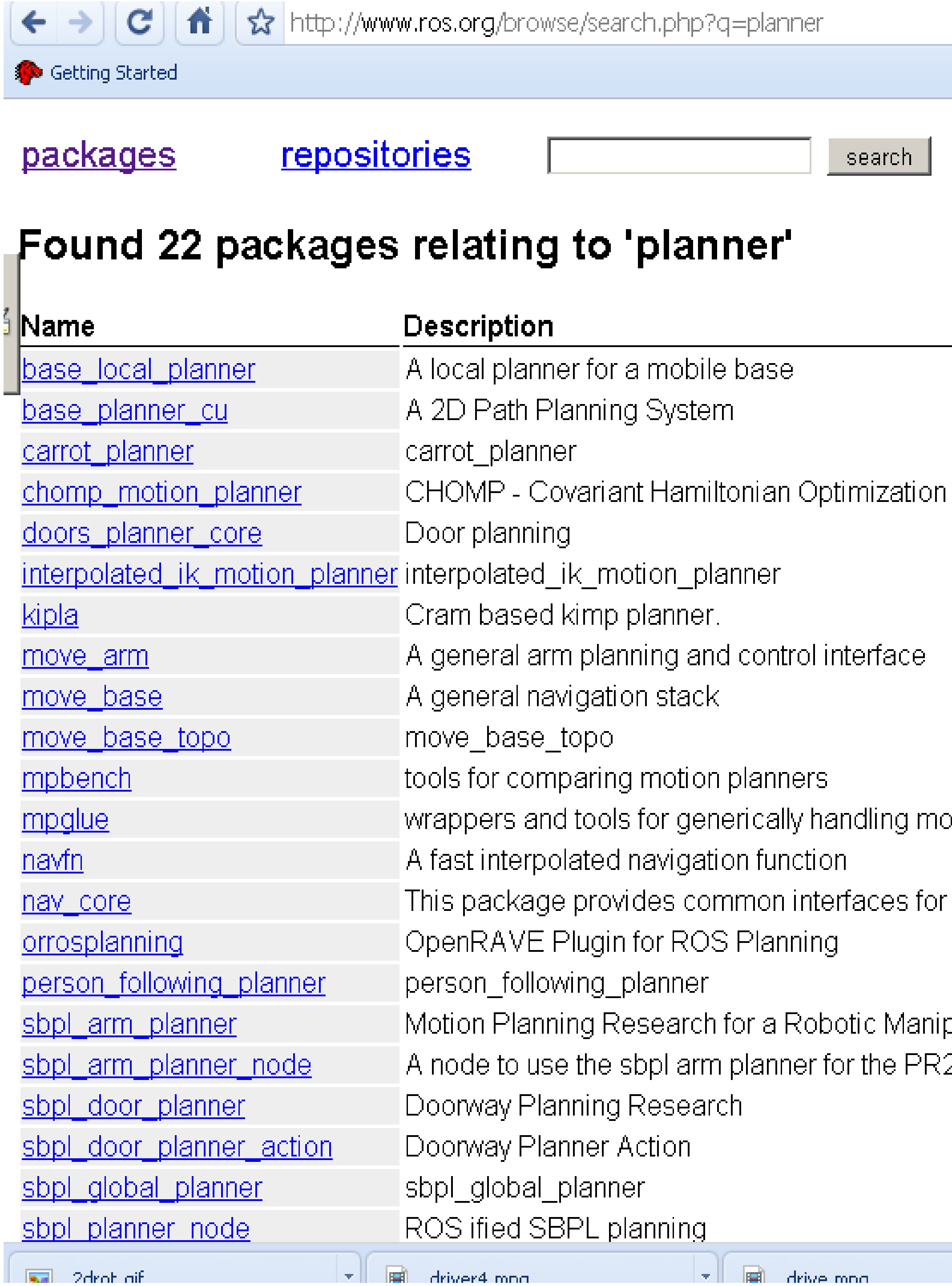
Hovercraft



RRT Reckless Driver



Driver's perspective



Found 22 packages relating to 'planner'

Name	Description
base_local_planner	A local planner for a mobile base
base_planner_cu	A 2D Path Planning System
carrot_planner	carrot_planner
chomp_motion_planner	CHOMP - Covariant Hamiltonian Optimization f
doors_planner_core	Door planning
interpolated_ik_motion_planner	interpolated_ik_motion_planner
kipla	Cram based kimp planner.
move_arm	A general arm planning and control interface
move_base	A general navigation stack
move_base_topo	move_base_topo
mpbench	tools for comparing motion planners
mpglue	wrappers and tools for generically handling mot
navfn	A fast interpolated navigation function
nav_core	This package provides common interfaces for r
orrosplanning	OpenRAVE Plugin for ROS Planning
person_following_planner	person_following_planner
sbpl_arm_planner	Motion Planning Research for a Robotic Manip
sbpl_arm_planner_node	A node to use the sbpl arm planner for the PR2
sbpl_door_planner	Doorway Planning Research
sbpl_door_planner_action	Doorway Planner Action
sbpl_global_planner	sbpl_global_planner
sbpl_planner_node	ROS ified SBPL planning

es in ROS

- OpenRave
- planar_patch_map (to convert point-cloud to polygonal structures)