CS686:
Path Planning for Point Robots

Sung-Eui Yoon
(윤성의)

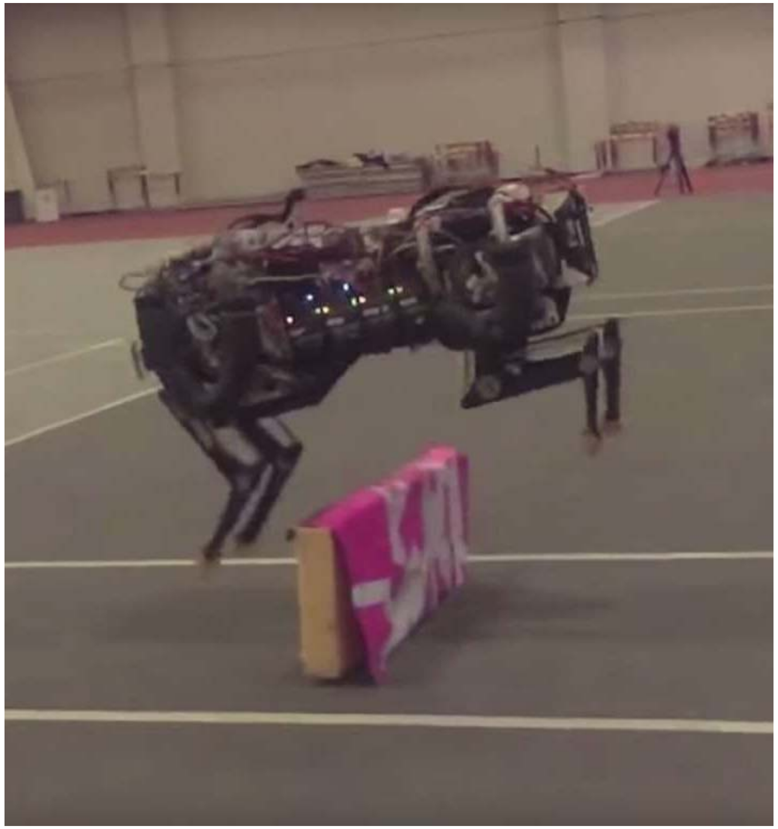
Course URL:
http://sglab.kaist.ac.kr/~sungeui/MPA

Class Objectives

- Motion planning framework
  - Representations of robots and space
  - Discretization into a graph
  - Search methods

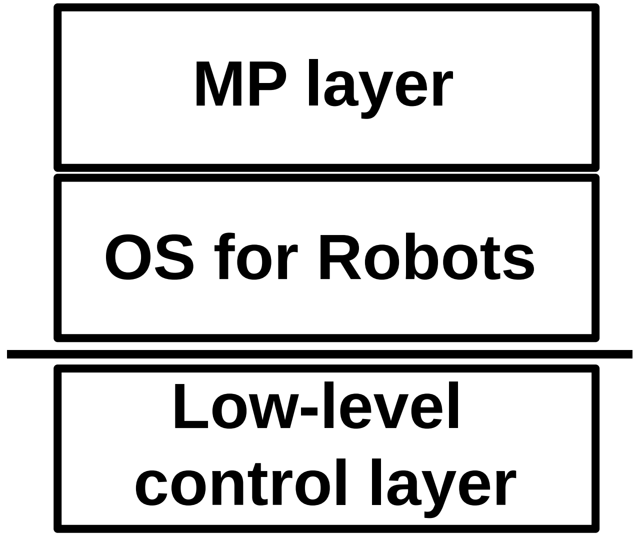
My View on Research Directions

- Many robots are available
  - Have different sensors and controls
- Basic controls are developed with such robots
  - Primitive motions are developed together
- Therefore, motion/path planning are widely researched



My View on Research Directions

- General motion planning tools
  - Primitive controls are available at HW vendors
  - How can we design a standard MP library working with those different robots?
  - For example, OpenGL for the robotics field; vendors support OpenGL, and programmer uses OpenGL for their applications



My View on Research Directions

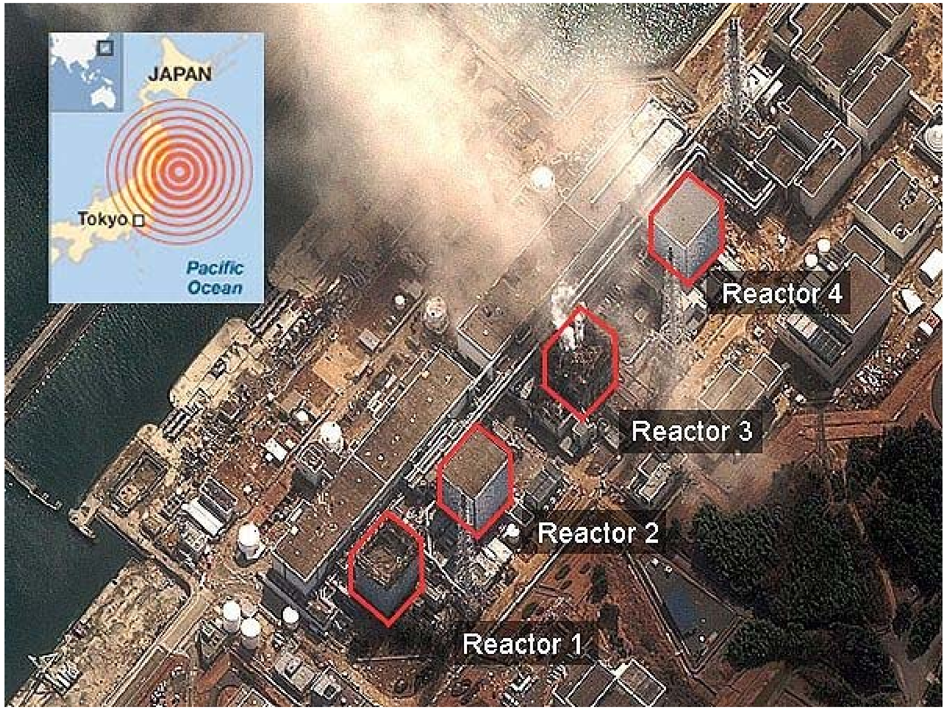
- High-level motion strategy are necessary
  - Optimal paths given constraints
  - Handling multiple robots for certain tasks
  - E.g., how can we efficiently assemble and disassemble the Boeing plane?

My View on Research Directions

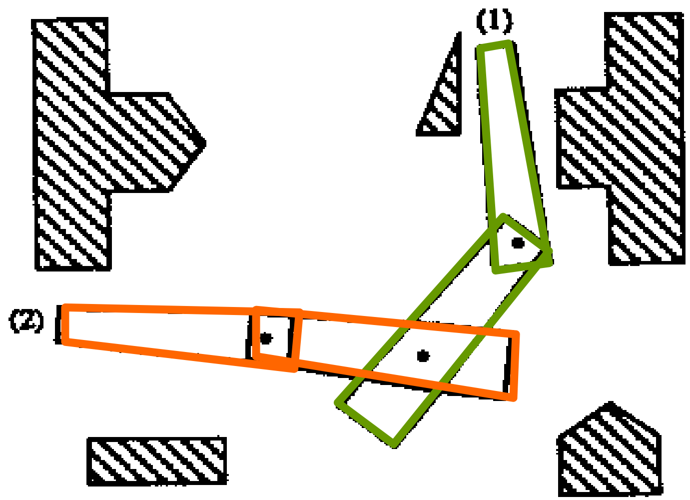
- High-level motion strategy are necessary
  - Optimal paths given constraints
  - Handling multiple robots for certain tasks
  - E.g., “Clean them!”

My View on Research Directions

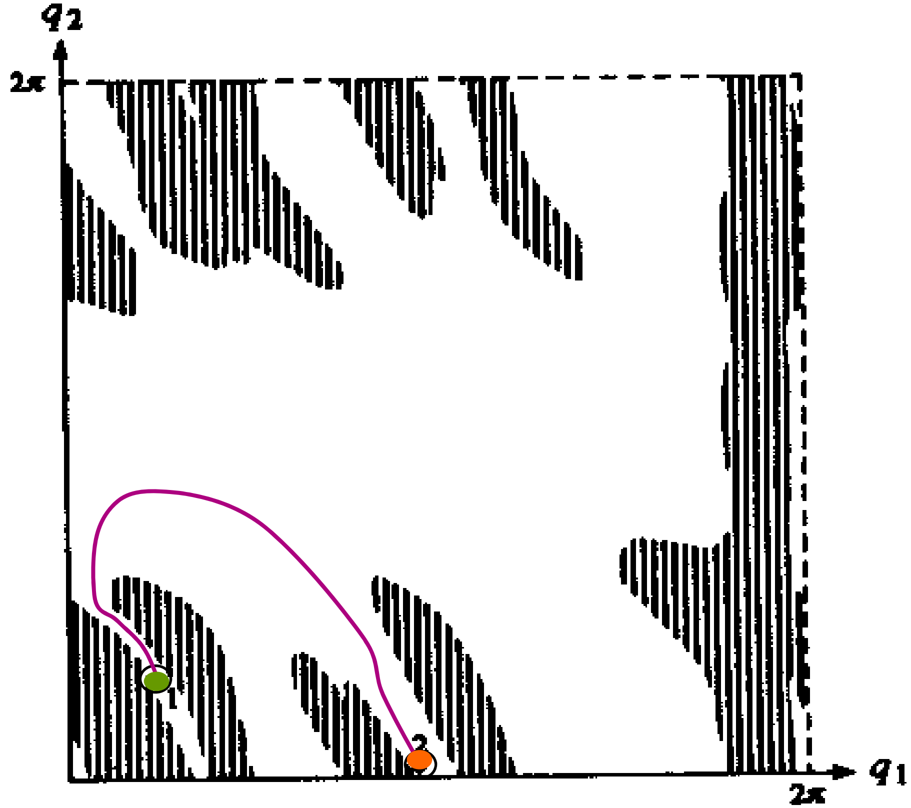
- High-level motion strategy are necessary
  - Optimal paths given constraints
  - Handling multiple robots for certain tasks
  - E.g., dangerous places for human



Configuration Space: Tool to Map a Robot to a Point



Workspace

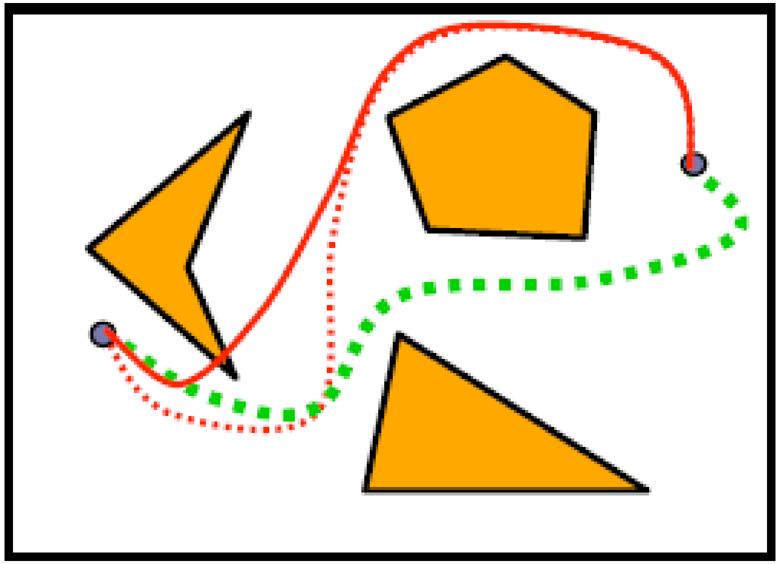


Configuration space (C-Space)

Problem

- Input
  - Robot represented as a **point** in the **plane**
  - Obstacles represented as polygons
  - Initial and goal positions
- Output

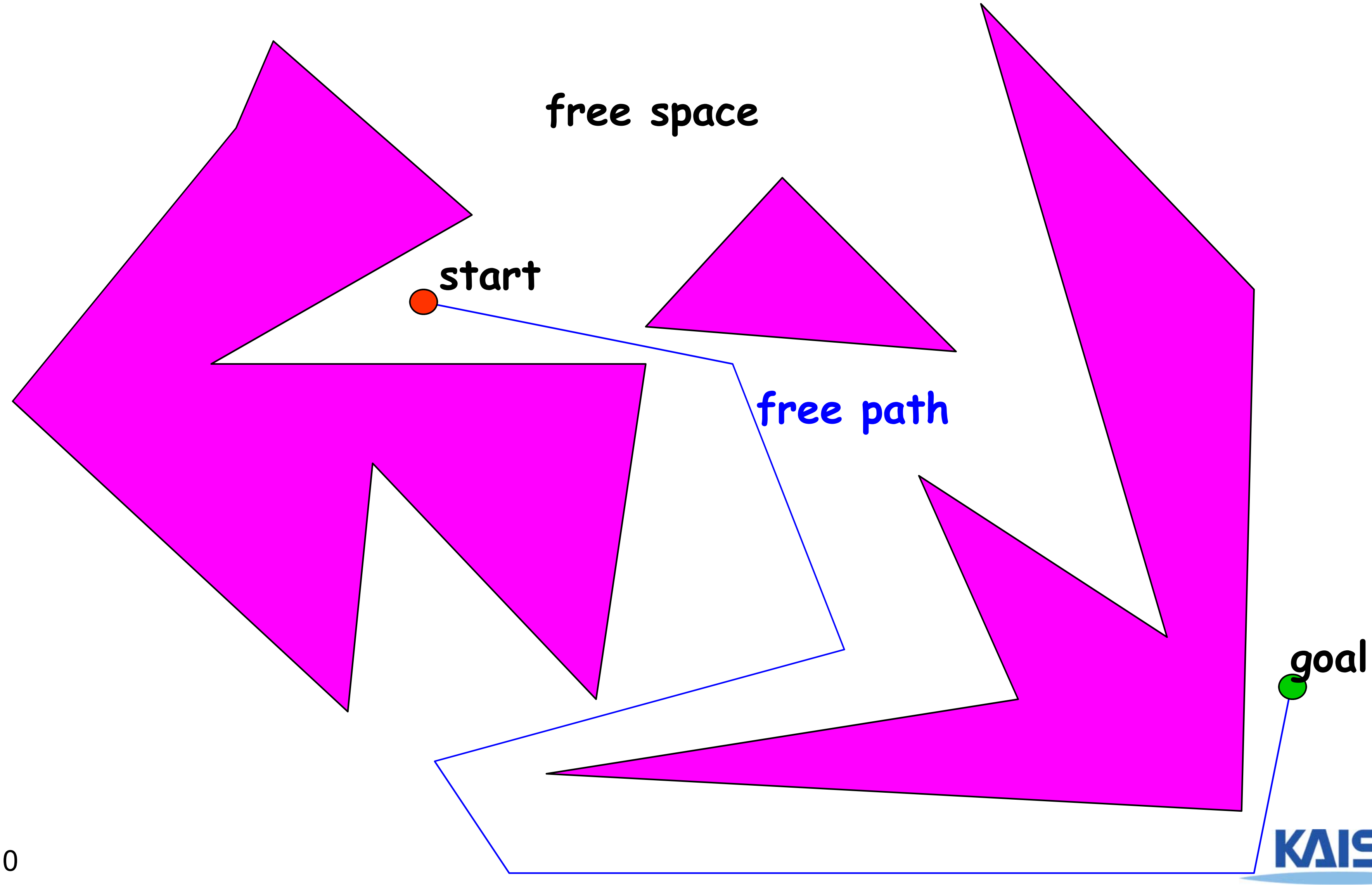
  A collision-free path between the initial and goal positions



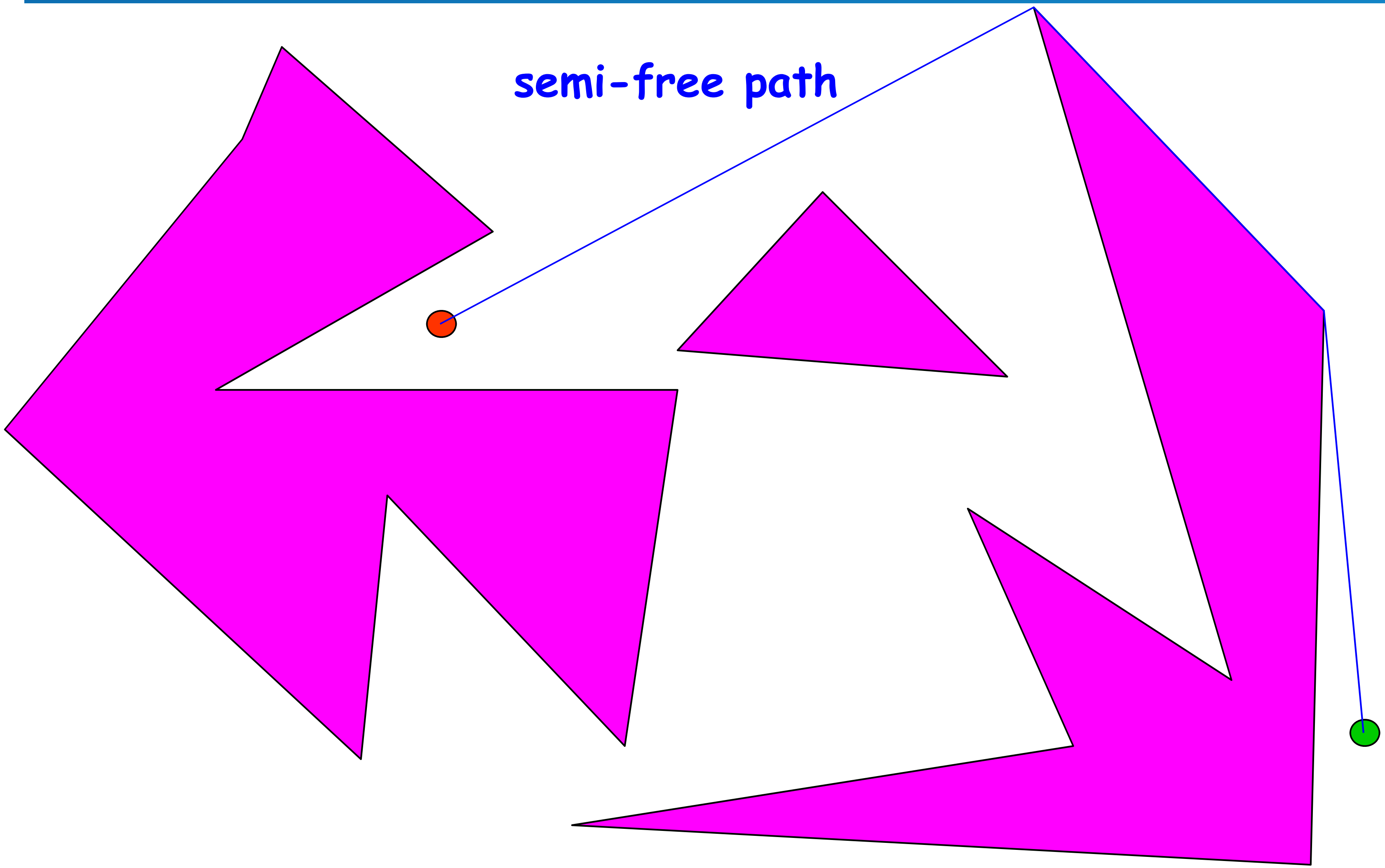
Workspace == C-Space in this simple case!

Courtesy of Prof. David Hsu

Problem



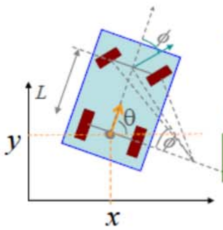
Problem



Types of Path Constraints

- **Local** constraints:
  lie in free space
- **Differential** constraints:
  have bounded curvature
- **Global** constraints:
  have minimal length

Example: Car-Like Robot



$$dx/dt = v\cos\theta, \quad dy/dt = v\sin\theta \;\rightarrow\; dx\sin\theta - dy\cos\theta = 0$$

$$d\theta/dt = (v/L)\tan\phi$$

$$|\phi| \leq \Phi$$

Lower-bounded turning radius

An example of differential constraints

Motion-Planning Framework

Continuous representation

(configuration space formulation)

↓

Discretization

(random sampling, processing critical geometric events)

↓

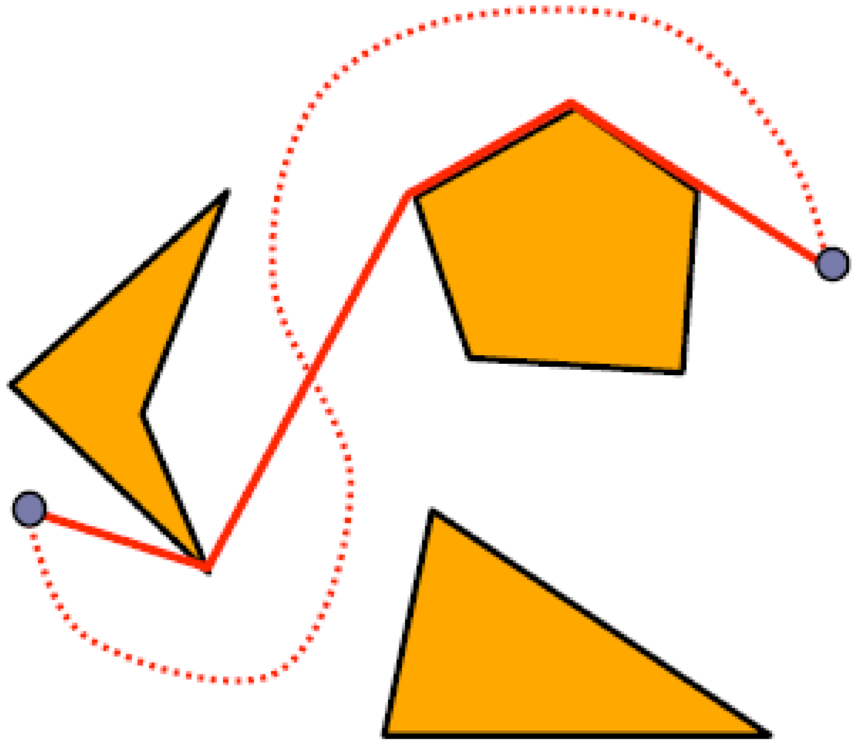
Graph searching

(blind, best-first, A*)

KAIST

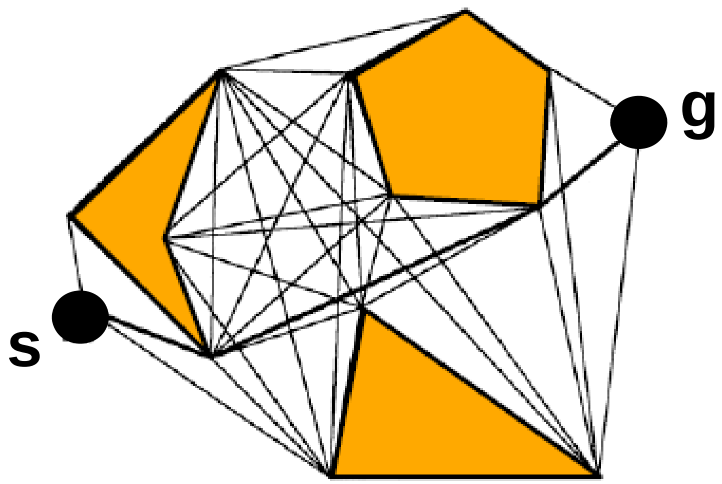
Visibility graph method

- **Observation**: If there is a 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.

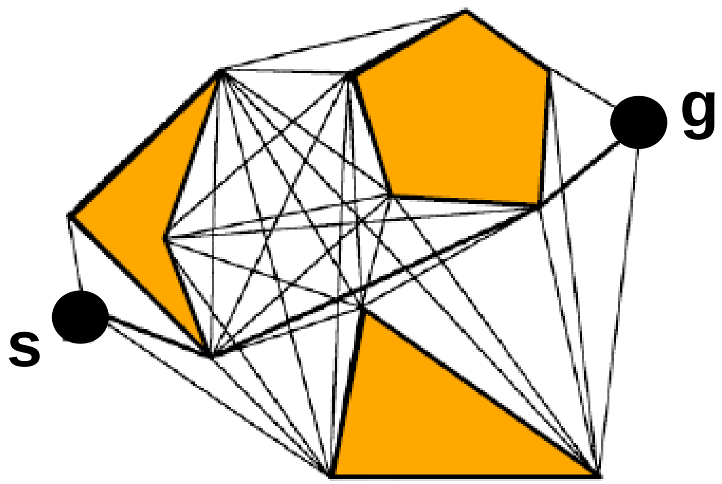
Visibility Graph



- A **visibility graph** is a graph such that
  - Nodes: s, g, or obstacle vertices
  - Edges: An edge exists between nodes u and v if the line segment between u and v is an obstacle edges or it does not intersect the obstacles

KAIST

Visibility Graph



- A **visibility graph**
  - Introduced in the late 60s
  - Can produce shortest paths in 2-D configuration spaces

KAIST

Simple Algorithm

- **Input: s, q, 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;
9: Search a path with G using A*
```

Computation Efficiency

```
1: for every pair of nodes u, v                          O(n^2)
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, etc.**
- **$O(n^2)$ space**

Motion-Planning Framework

Continuous representation

(configuration space formulation)

↓

Discretization

(random sampling, processing critical geometric events)

↓

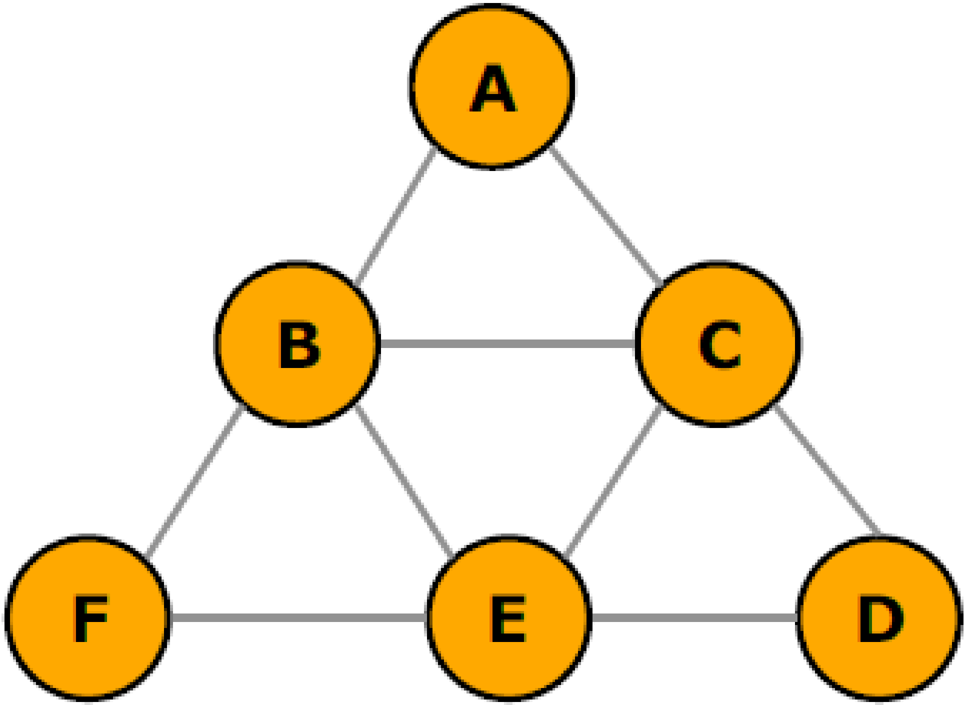
Graph searching

(blind, best-first, A*)

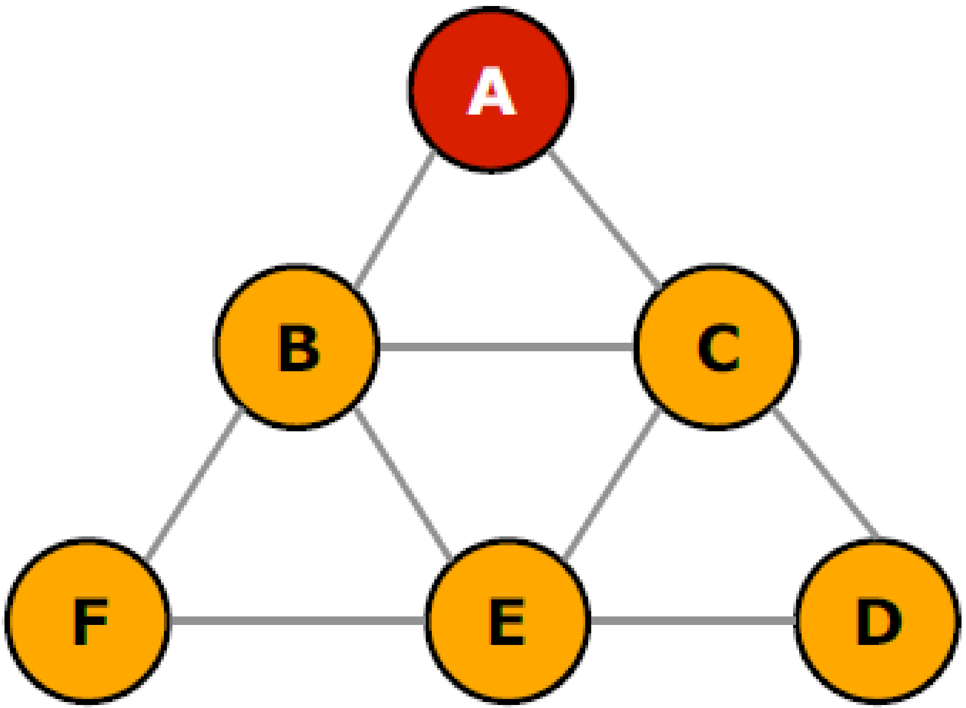
Graph Search Algorithms

- Breadth, depth-first, best-first
- Dijkstra’s algorithm
- A*

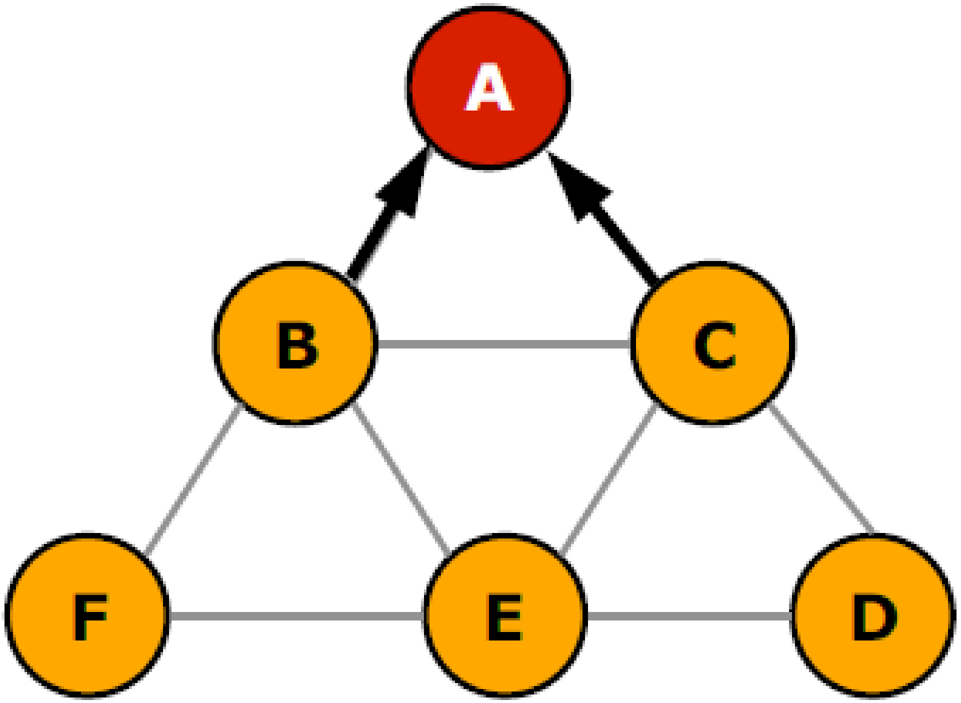
Breadth-first search



Breadth-first search

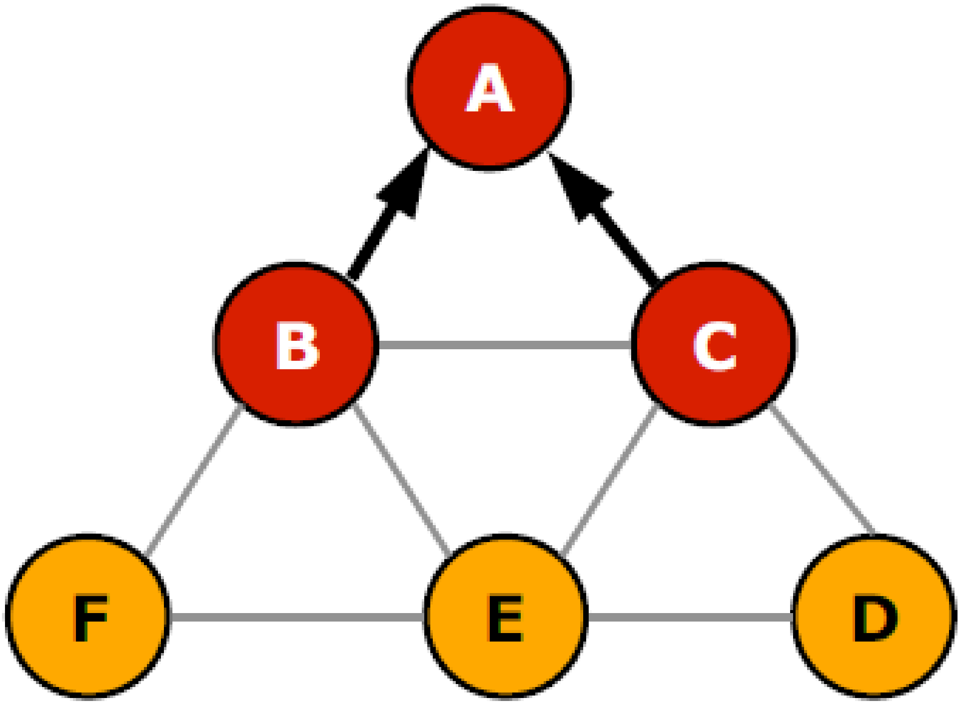


Breadth-first search



Breadth-first search

Traverse the graph by using the queue, resulting in the level-by-level traversal



Dijkstra's Shortest Path Algorithm

- Given a (non-negative) weighted graph, two vertices, s and g:
  - Find a path of minimum total weight between them
  - Also, find minimum paths to other vertices
  - Has O (|V| lg|V| + |E|), where V & E refer vertices & edges

Dijkstra’s Shortest Path Algorithm

- **Set S**
  - **Contains vertices whose final shortest-path cost has been determined**

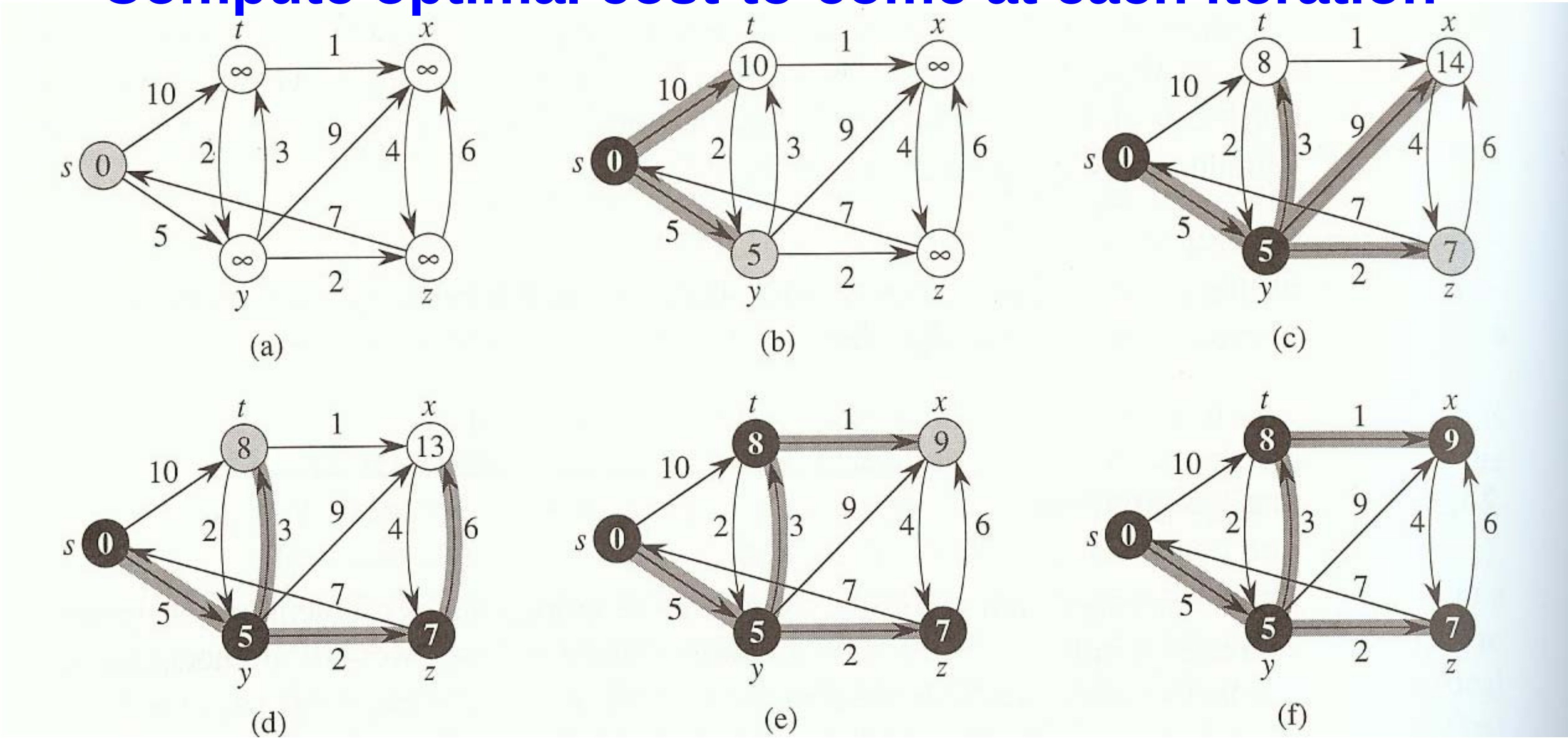
- **DIJKSTRA (G, s):**

Input: G is an input graph, s is the source

```
Initialize-Single-Source (G, s)
S ← empty
Queue ← Vertices of G
While Queue is not empty
    Do u ← min-cost from Queue
        S ← union of S  and {u}
        for each vertex v in Adj [u]
            do RELAX (u, v)
```

Dijkstra's Shortest Path Algorithm

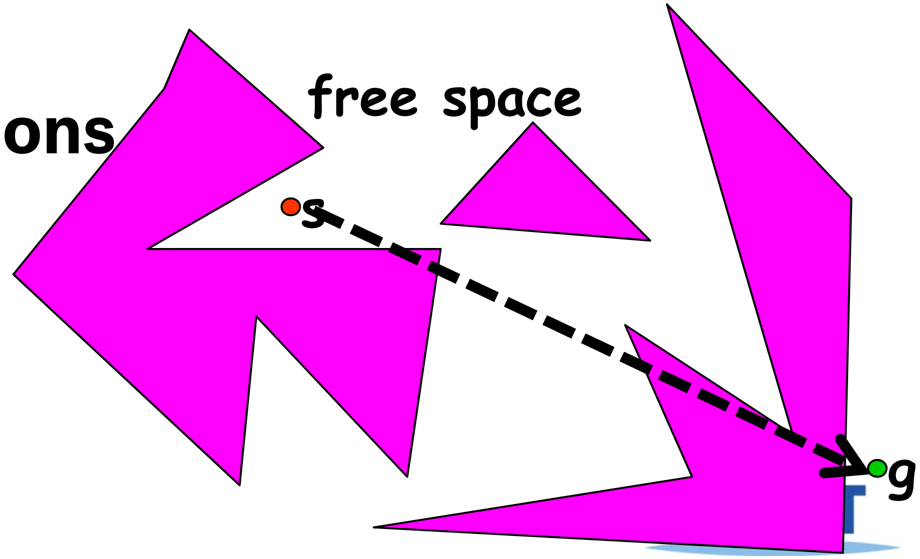
Compute optimal cost-to-come at each iteration



Black vertices are in the set.
White vertices are in the queue.
Shaded one is chosen for relaxation.

A* Search Algorithm

- An extension of Dijkstra's algorithm based on a heuristic estimate
  - Conservatively estimate the cost-to-go from a vertex to the goal
  - The estimate should not be greater than the optimal cost-to-go
  - Sort vertices based on "cost-to-come + the estimated cost-to-go"
  - Can find optimal solutions with fewer steps



Best-First Search

- **Pick a next node based on an estimate of the optimal cost-to-go cost**
  - **Greedily finds solutions that look good**
  - **Solutions may not be optimal**
  - **Can find solutions quite fast, but can be also very slow**

Framework

continuous representation

↓

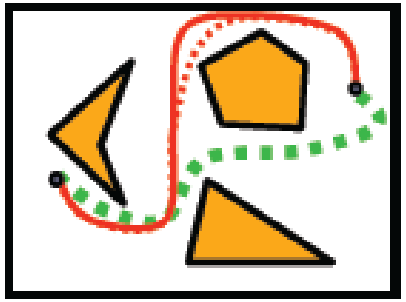
discretization

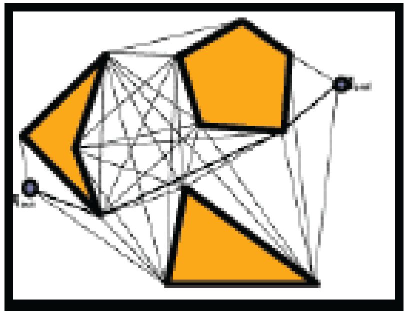
construct visibility graph

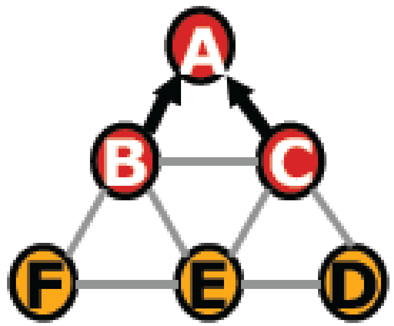
↓

graph searching

breadth-first search







Computational Efficiency

- Running time $O(n^3)$
  - Compute the visibility graph
  - Search the graph
- Space $O(n^2)$

- Can we do better?
  - Lead to classical approaches such as roadmap

Class Objectives were:

- Motion planning framework
  - Representations of robots and space
  - Discretization into a graph
  - Search methods

Homework

- **Browse 2 ICRA / IROS / RSS / WAFR / TRO / IJRR papers**
  - **Prepare two summaries and submit at the beginning of every Tue. class, or**
  - **Submit it online before the Tue. Class**

- **Example of a summary (just a paragraph)**

  Title: XXX XXXX XXXX

  Conf./Journal Name: ICRA, 2015

  Summary: this paper is about accelerating the performance of collision detection. To achieve its goal, they design a new technique for reordering nodes, since by doing so, they can improve the coherence and thus improve the overall performance.

KAIST

Homework for Every Class

- **Go over the next lecture slides**
- **Come up with one question on what we have discussed today and submit at the end of the class**
  - **1 for typical questions**
  - **2 for questions with thoughts or that surprised me**

- **Write a question at least 4 times before the mid-term exam**

Homework

- **Read Chapter 1 of our textbook**
- **Optional:**
  - **Motion planning: A journey of robots, molecules, digital Actors, and other artifacts. J.C. Latombe. Int. J. Robotics Research, 18(11):1119-1128, 1999**

Next Time….

- **Classic path planning algorithms**