Combinatorial Planning, Sampling-based Motion Planning and Potential Field Method

Lecture 8 – Thursday December 1, 2016

Objectives

When you have finished this lecture you should be able to:

- Understand Combinatorial Planning, Sampling-based Motion Planning and Potential Field Method.

Outline

- Combinatorial Planning
- Sampling-based Motion Planning
- Potential Field Method

Outline

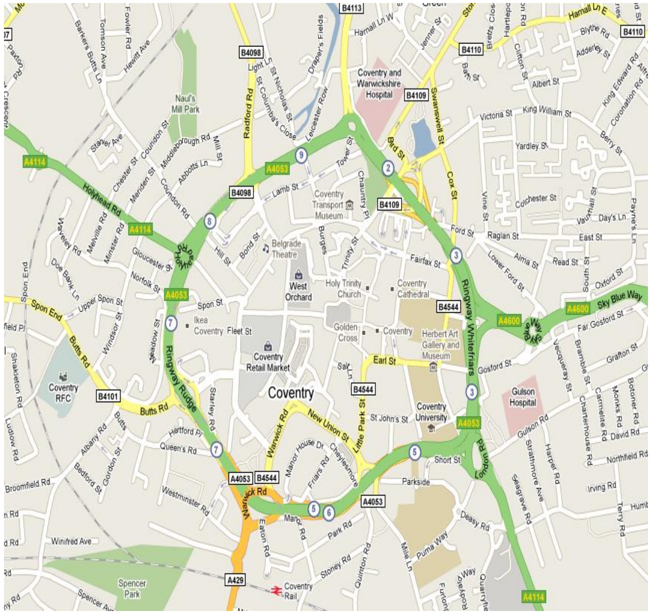
- **Combinatorial Planning**
- Sampling-based Motion Planning
- Potential Field Method

Combinatorial Planning

- **Road map**

  The road map approach consists of generating **connecting cells** within mobile robot's free space into a network of one-dimensional curves called a **road map**.

  - **Properties of a Roadmap:**
    - ◇ **Accessibility:** there exists a collision-free path from the start to the road map
    - ◇ **Departability:** there exists a collision-free path from the roadmap to the goal.
    - ◇ **Connectivity:** there exists a collision-free path from the start to the goal (on the roadmap).



A roadmap exists ⇔ A path exists

Combinatorial Planning

- **Road map**

Road Map Approaches

- Visibility Graph
- Maklink
- Voronoi Diagrams
- Free Way Net
- Silhouette
- …

Combinatorial Planning

- **Road map: Visibility Graph**
  - ◇ Suppose someone gives you a **CSPACE** with polygonal obstacles.
  - ◇ Visibility graph is formed by **connecting all "visible" vertices**, the start point and the end point, to each other.
  - ◇ For two points to be "**visible" no obstacle** can exist between them, i.e., paths exist on the perimeter of obstacles.

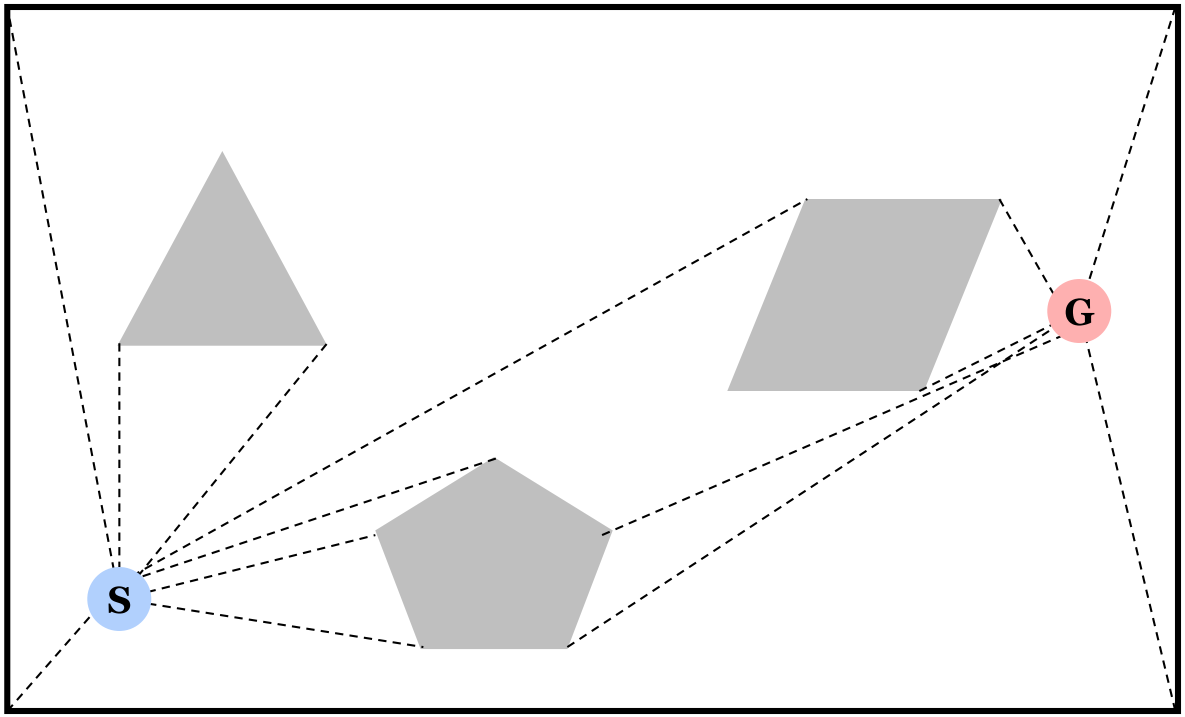
Combinatorial Planning

- **Road map: Visibility Graph**

1. Start with a map of the world, draw lines of sight from the **start and goal** to **every "corner" of the world** and **vertex of the obstacles**, not cutting through any obstacles.
2. Draw lines of sight **from every vertex of every obstacle** like above. Lines along edges of obstacles are lines of sight too, since they don't pass through the obstacles.
3. If the map was in Configuration space, each line potentially represents part of a **path from the start to the goal**.

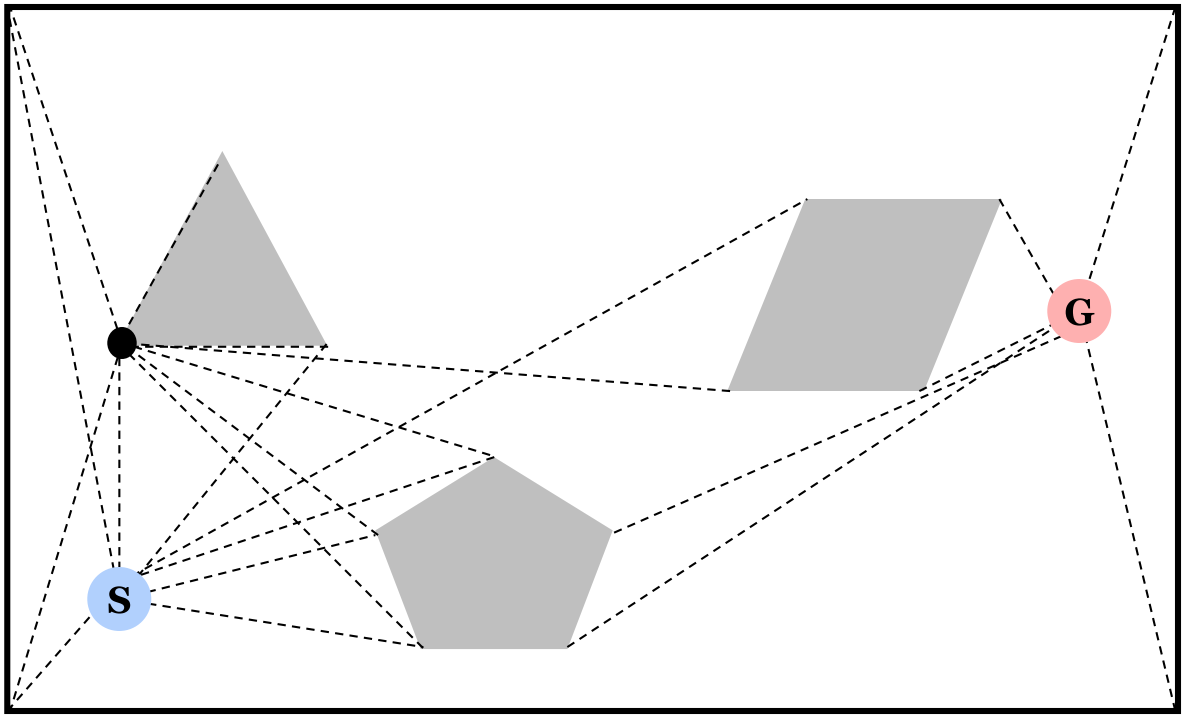
Combinatorial Planning

- **Road map: Visibility Graph**
  - ◇ First, draw lines of sight from the start and goal to all "visible" vertices and corners of the world.



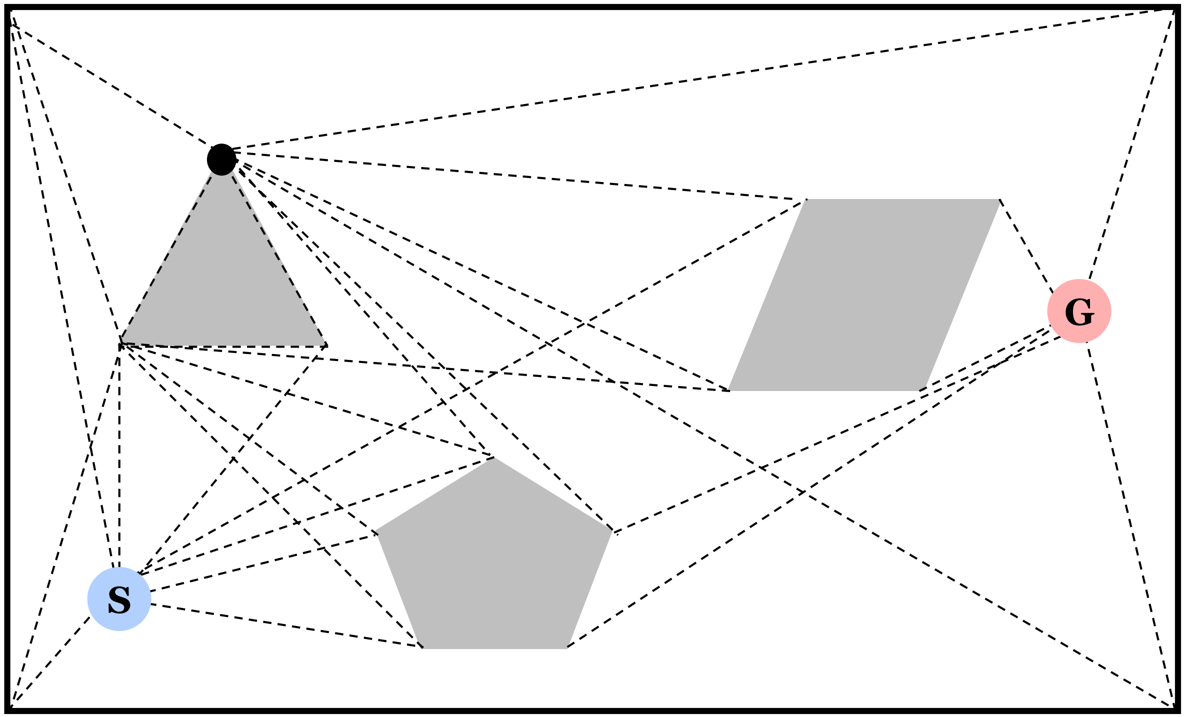
Combinatorial Planning

- **Road map: Visibility Graph**
  - ◇ Second, draw lines of sight from every vertex of every obstacle like before. Remember lines along edges are also lines of sight.



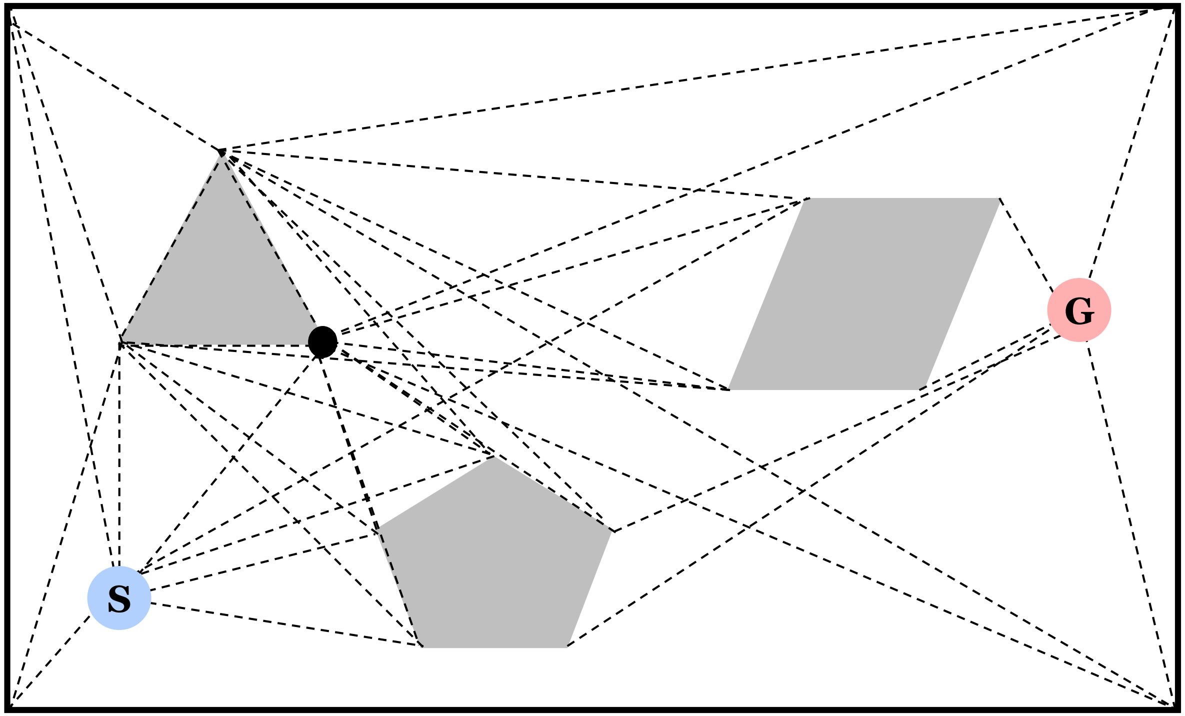
Combinatorial Planning

- **Road map: Visibility Graph**
  - ◇ Second, draw lines of sight from every vertex of every obstacle like before. Remember lines along edges are also lines of sight.



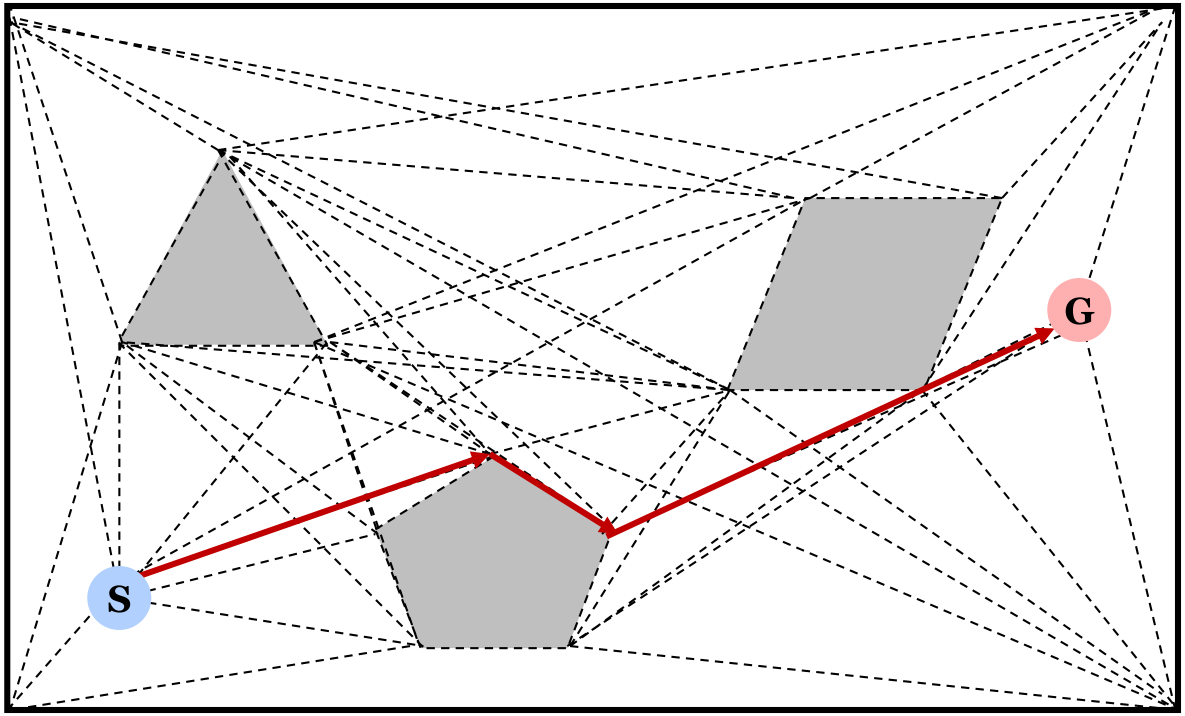
Combinatorial Planning

- **Road map: Visibility Graph**
  - ◇ Second, draw lines of sight from every vertex of every obstacle like before. Remember lines along edges are also lines of sight.



Combinatorial Planning

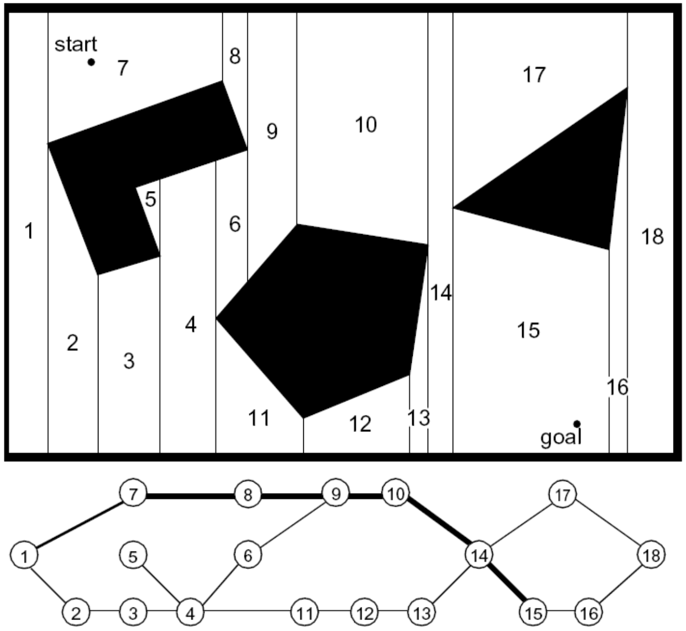
- **Road map: Visibility Graph**
  - ◊ Repeat until you're done.
  - ◊ Search the graph of these lines for the shortest path (using Dijkstra algorithm for example).



Combinatorial Planning

- **Cell Decomposition**

  The idea behind cell decomposition is to discriminate between geometric areas, or cells, that are free and areas that are occupied by objects.



Cell Decomposition

Exact Cell Decomposition

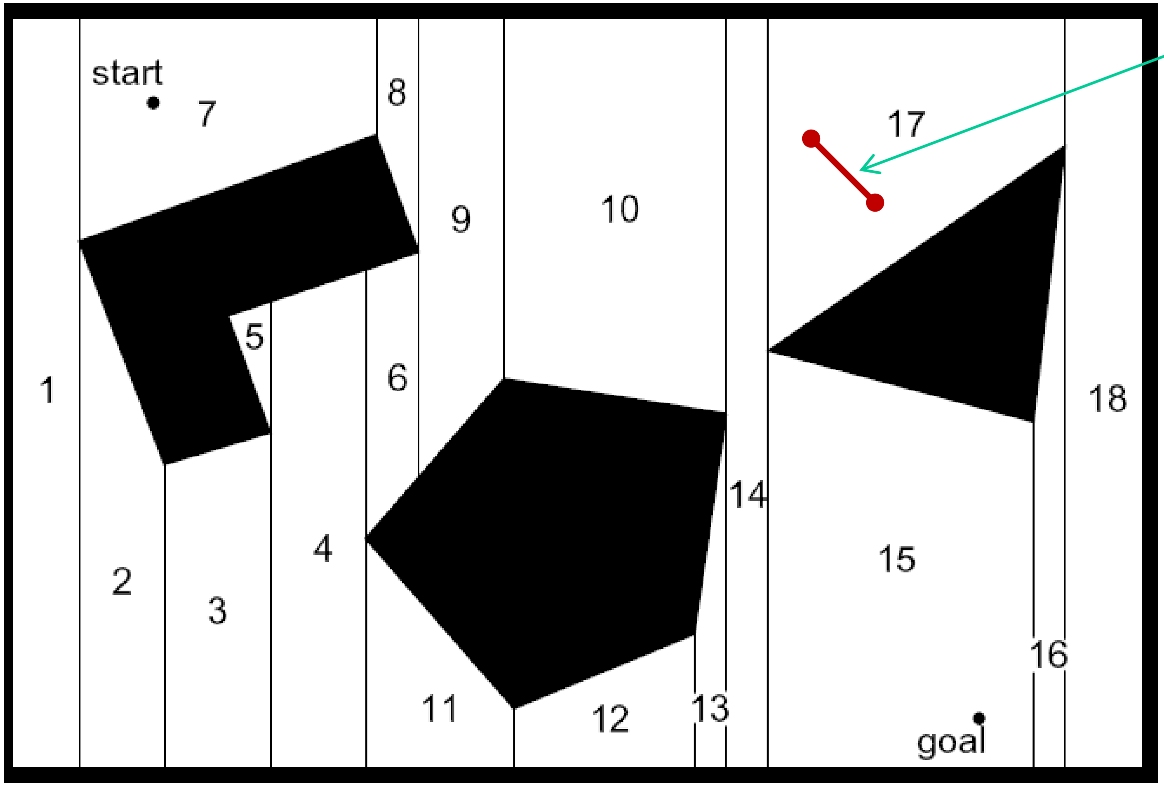
Approximate Cell Decomposition

Fixed Decomposition

Adaptive Decomposition

Combinatorial Planning

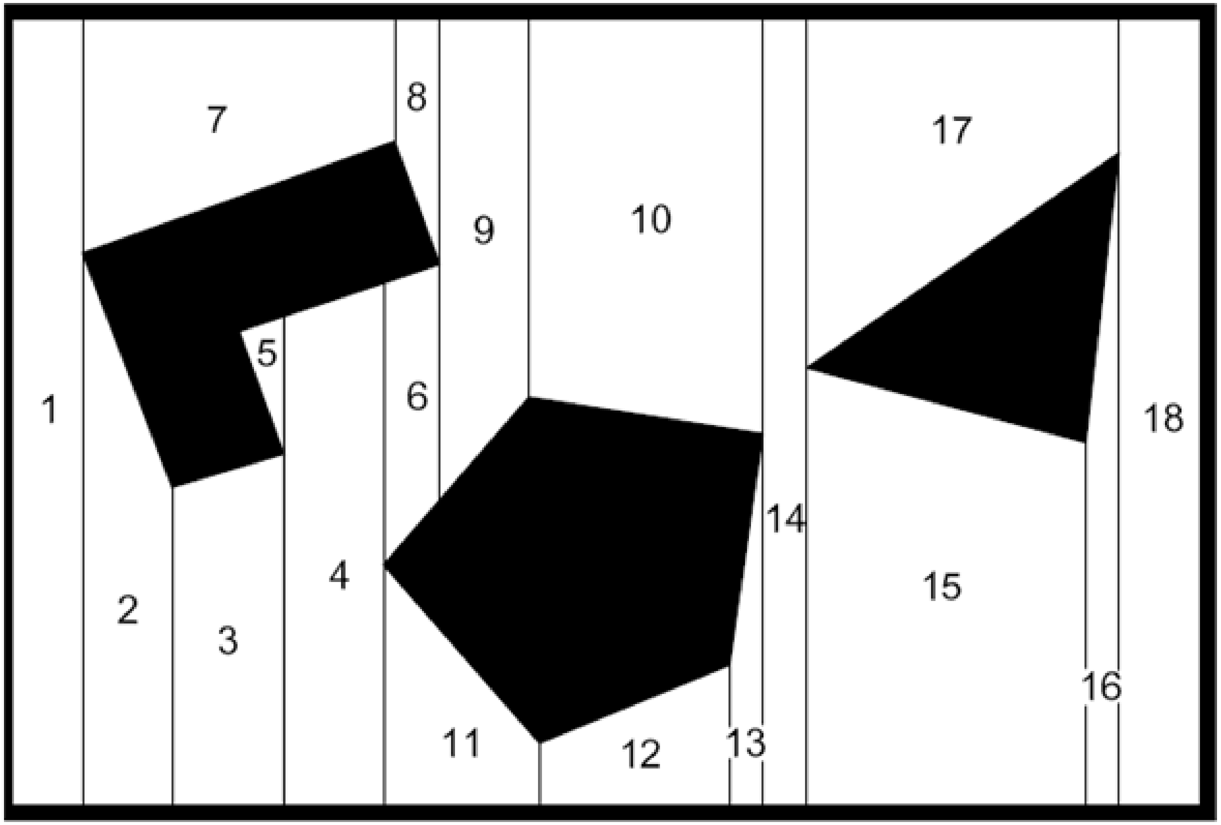
- **Exact Cell Decomposition**
  - ◇ Divide environment into simple, connected regions called "cells".



Any path within one cell is guaranteed to not intersect any obstacle

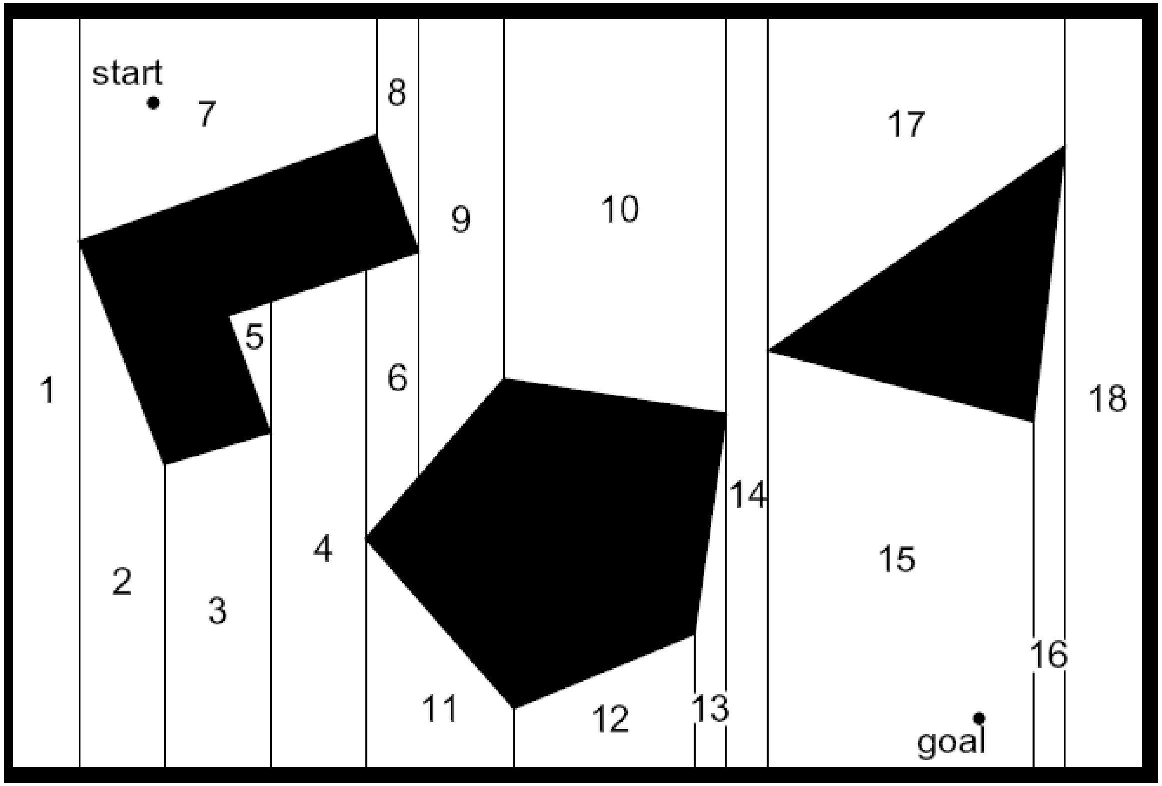
Combinatorial Planning

- **Exact Cell Decomposition**
  - ◊ Determine which opens cells are adjacent.



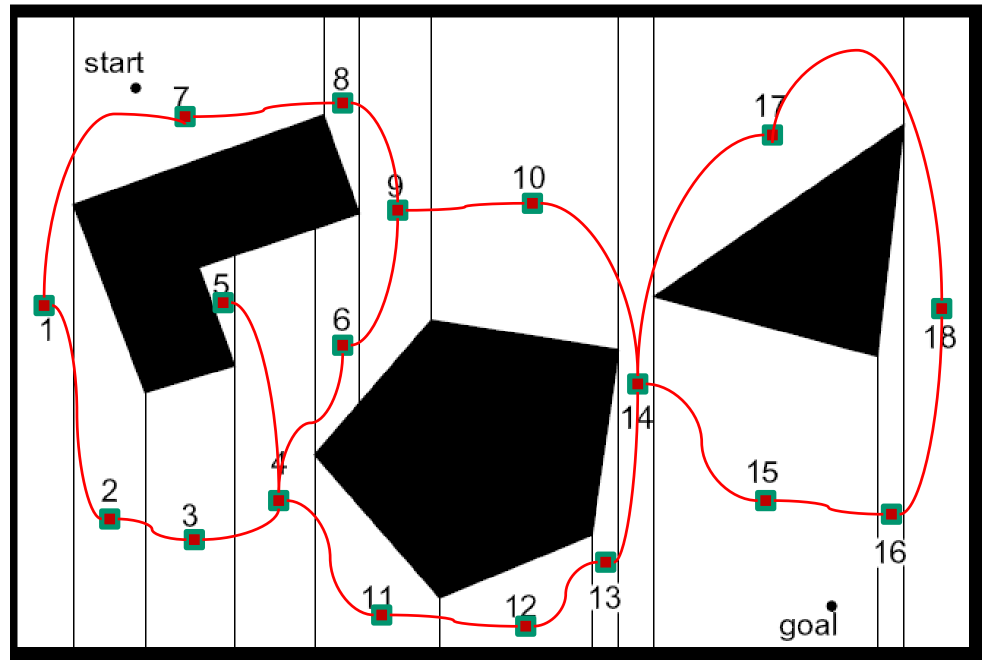
Combinatorial Planning

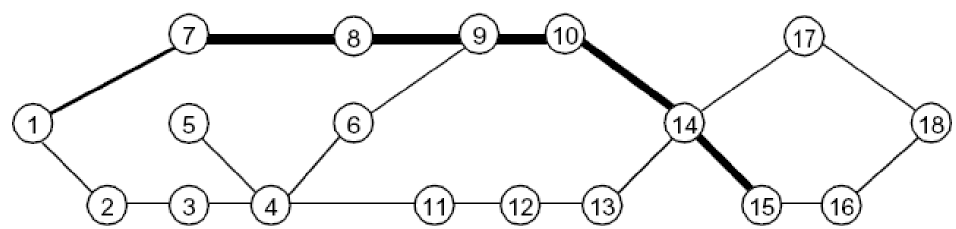
- **Exact Cell Decomposition**
  - ◇ Find the cells in which the **initial and goal configurations** lie and search for a path in the connectivity graph to join the **initial and goal cell**.



Combinatorial Planning

- **Exact Cell Decomposition**
  - ◇ Construct the connectivity graph.

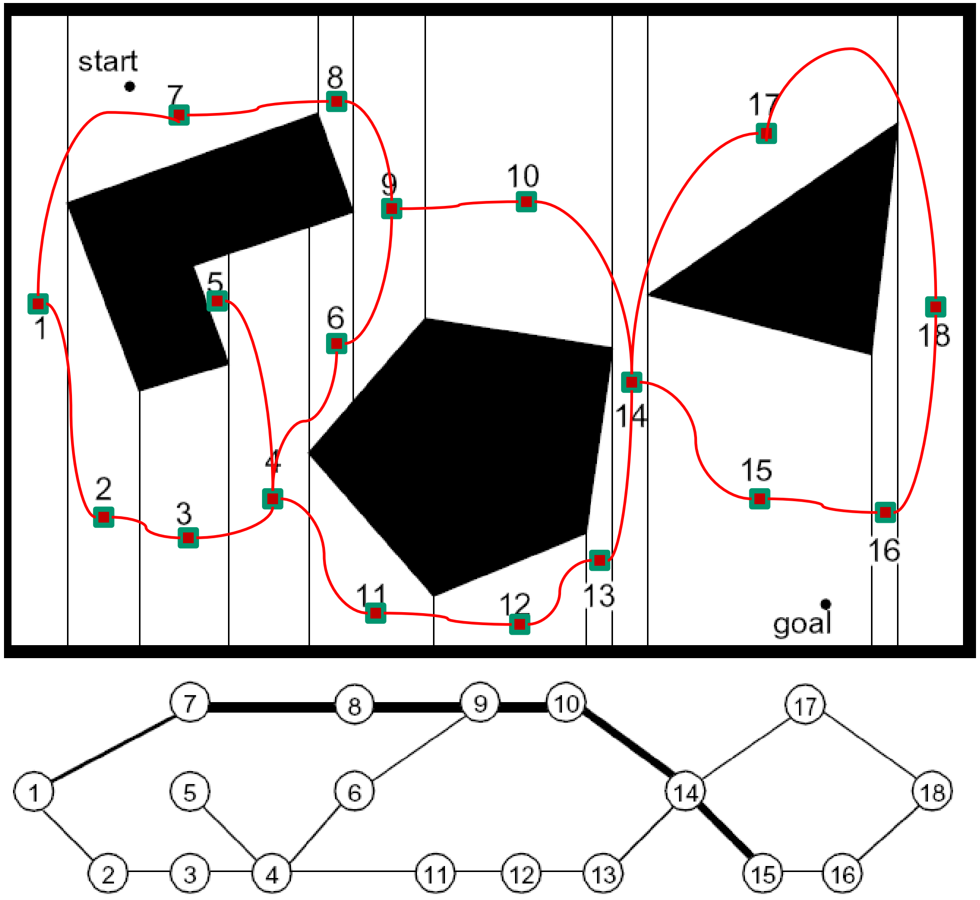




The connectivity graph of cells defines a roadmap

Combinatorial Planning

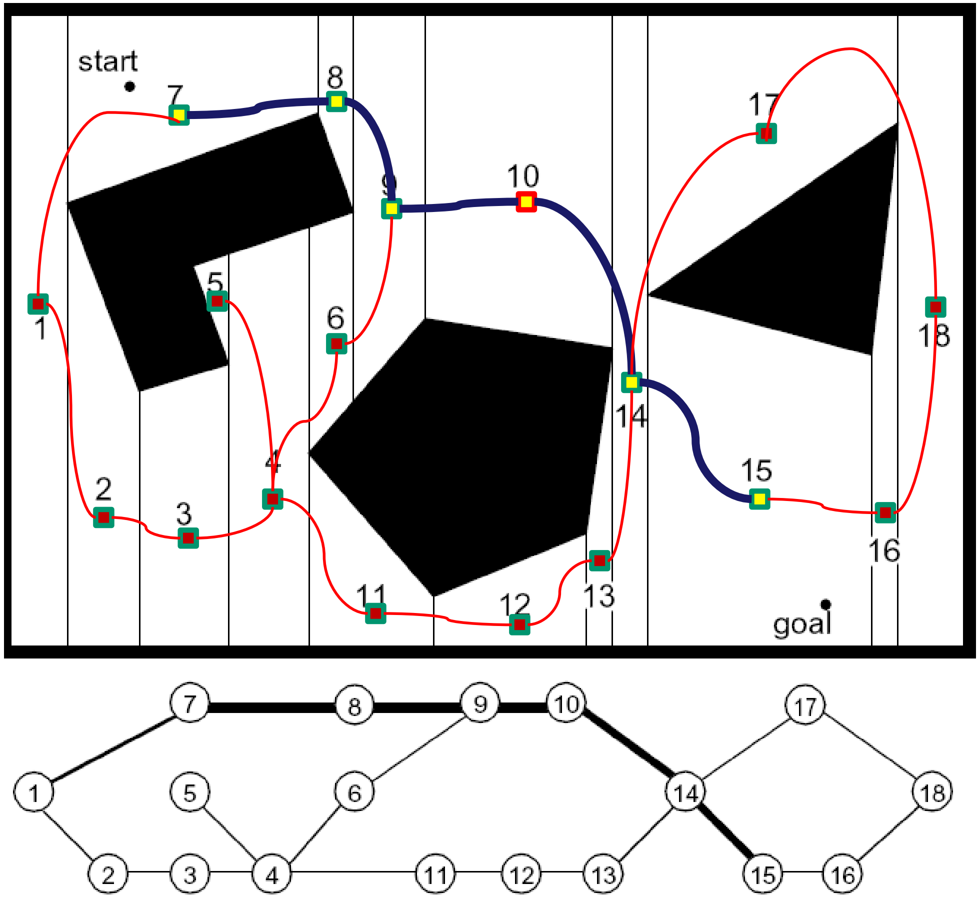
- **Exact Cell Decomposition**
  - ◇ From the sequence of cells found with an appropriate **searching algorithm**, compute a path within each cell, for example, passing through the **midpoints of the cell boundaries** or by a **sequence of wall-following motions** and movements along straight lines.



The connectivity graph of cells defines a roadmap

Combinatorial Planning

- **Exact Cell Decomposition**
  - ◇ From the sequence of cells found with an appropriate **searching algorithm**, compute a path within each cell, for example, passing through the **midpoints of the cell boundaries** or by a **sequence of wall-following motions** and movements along straight lines.



The graph can be used to find a path between any two configurations

Combinatorial Planning

- **Exact Cell Decomposition**
  - ◇ The **key disadvantage** of exact cell decomposition is that the number of cells and, therefore, overall path planning **computational efficiency** depends upon the **density and complexity of objects** in the environment, just as with **road map-based** systems.
  - ◇ Practically speaking, due to complexities in implementation, the exact cell decomposition technique **is used relatively rarely** in mobile robot applications, although it remains a solid choice when a lossless representation is highly desirable, for instance to preserve completeness fully.

Combinatorial Planning

- **Approximate Cell Decomposition**
  - ◇ By contrast, approximate cell decomposition is **one of the most popular techniques** for mobile robot path planning.
  - ◇ This is partly due to the popularity of grid-based environmental representations.

Approximate Cell Decomposition

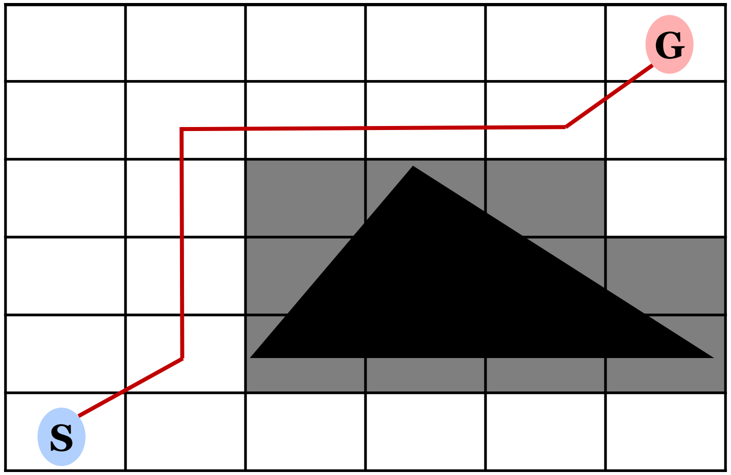
Fixed Decomposition

Adaptive Decomposition

Combinatorial Planning

- **Fixed Decomposition**

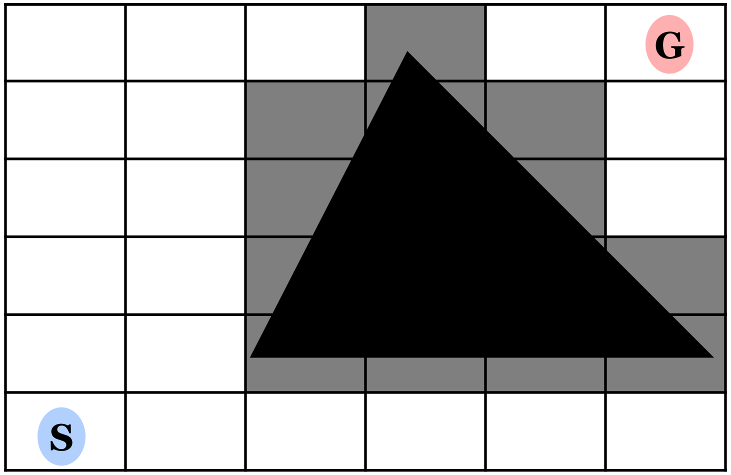
1. Define a discrete grid in C-Space
2. Mark any cell of the grid that intersects obstacles as blocked
3. Find path through remaining cells by using (for example) A* (e.g., use Euclidean distance as heuristic)



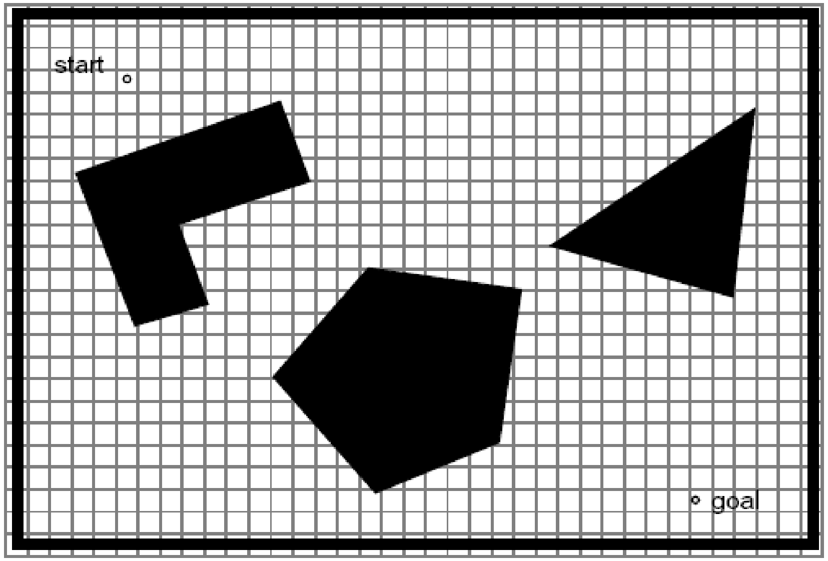
◇ Cannot be complete as described so far. **Why?**

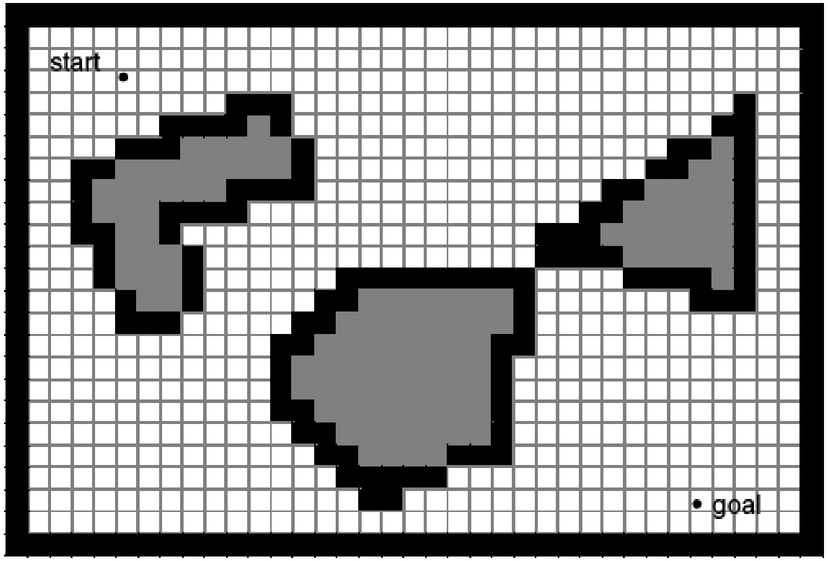
Combinatorial Planning

- **Fixed Decomposition**



Cannot find a path in this case even though one exists.





The key **disadvantage** of this approach stems from its inexact nature. It is possible for **narrow passageways** to be lost during such a transformation.

Combinatorial Planning

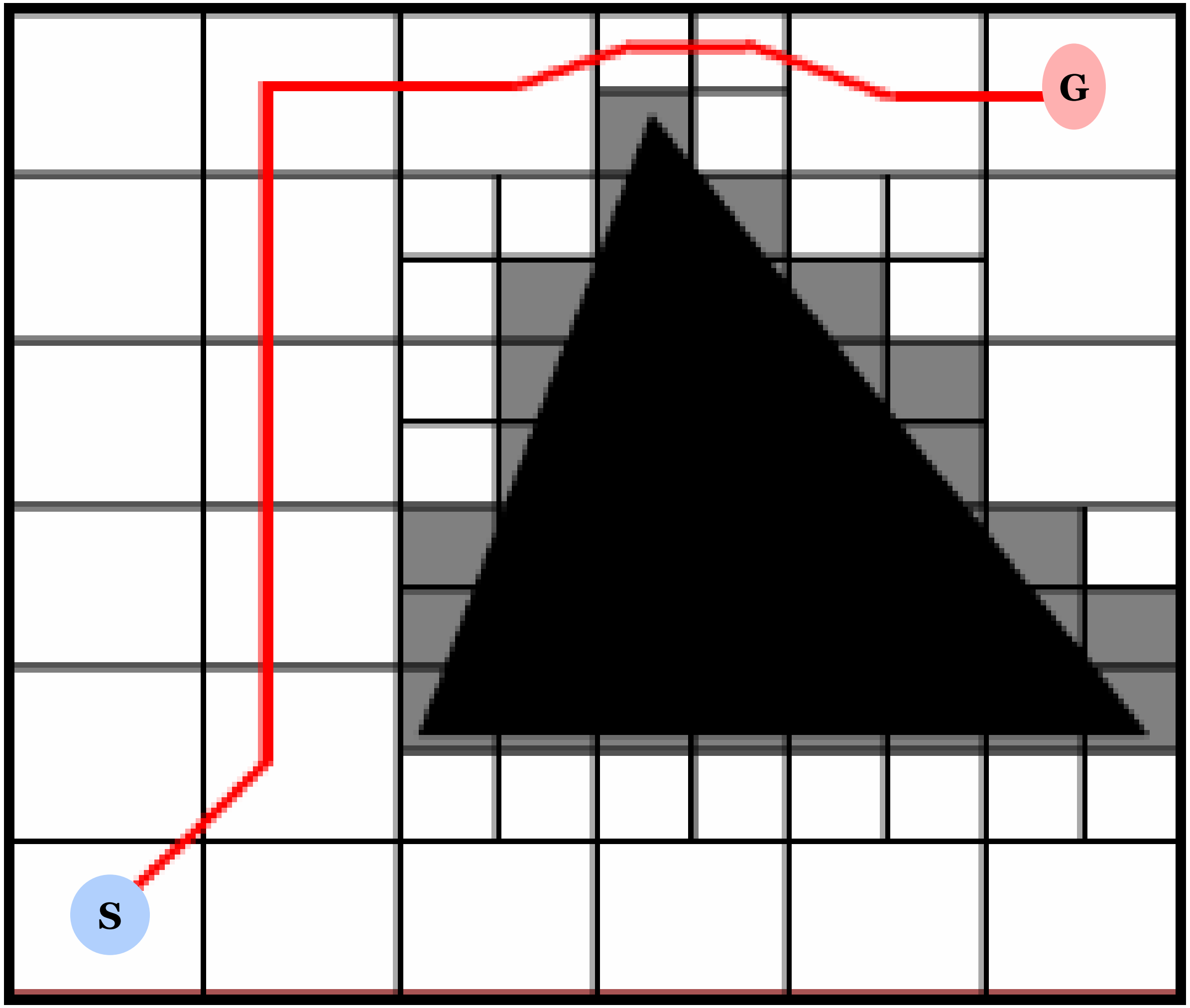
- **Adaptive Decomposition**

1. Distinguish between Cells that are entirely contained in obstacles (FULL) and Cells that partially intersect obstacles (MIXED)
2. Try to find a path using the current set of cells.
3. **If** no path found:

   Subdivide the MIXED cells and try again with the new set of cells.

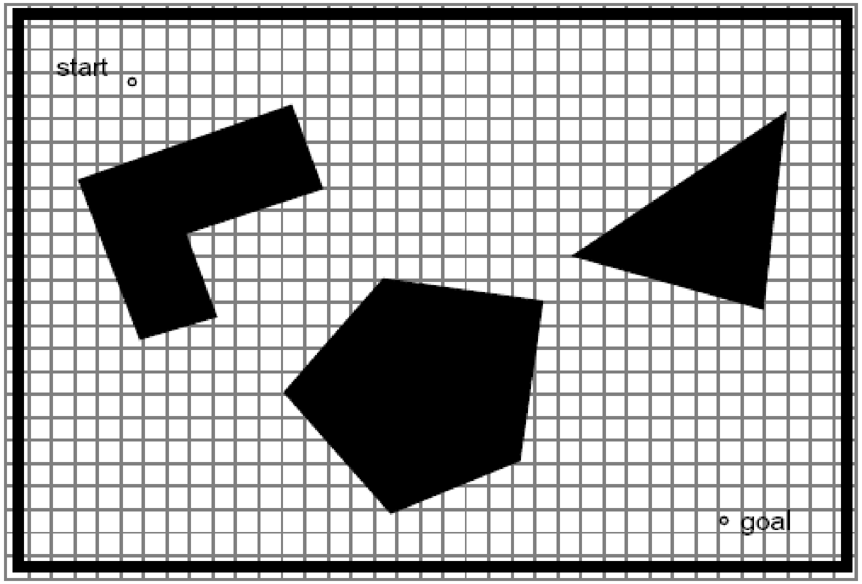
Combinatorial Planning

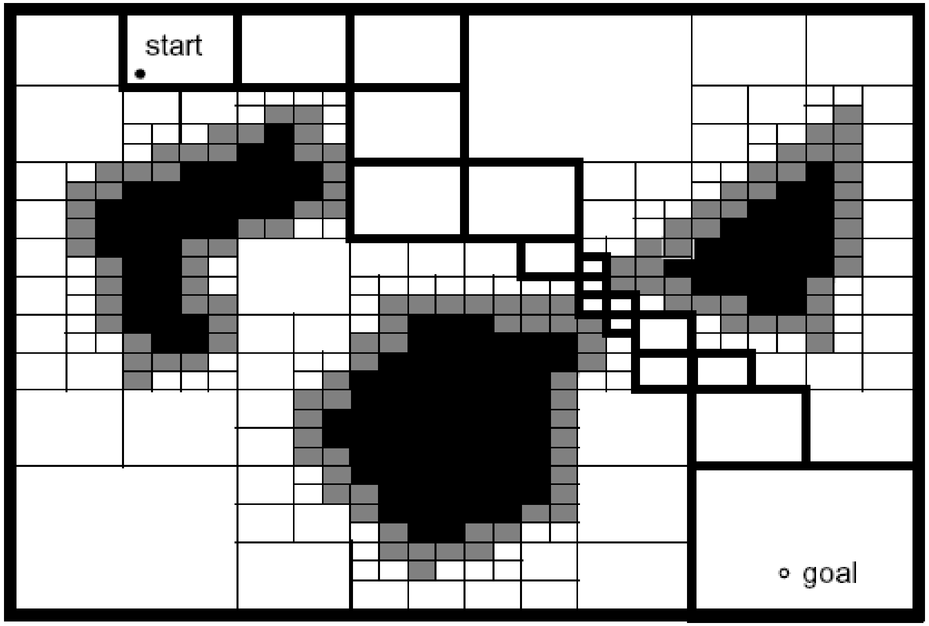
- **Adaptive Decomposition**



Combinatorial Planning

- **Adaptive Decomposition**





Combinatorial Planning

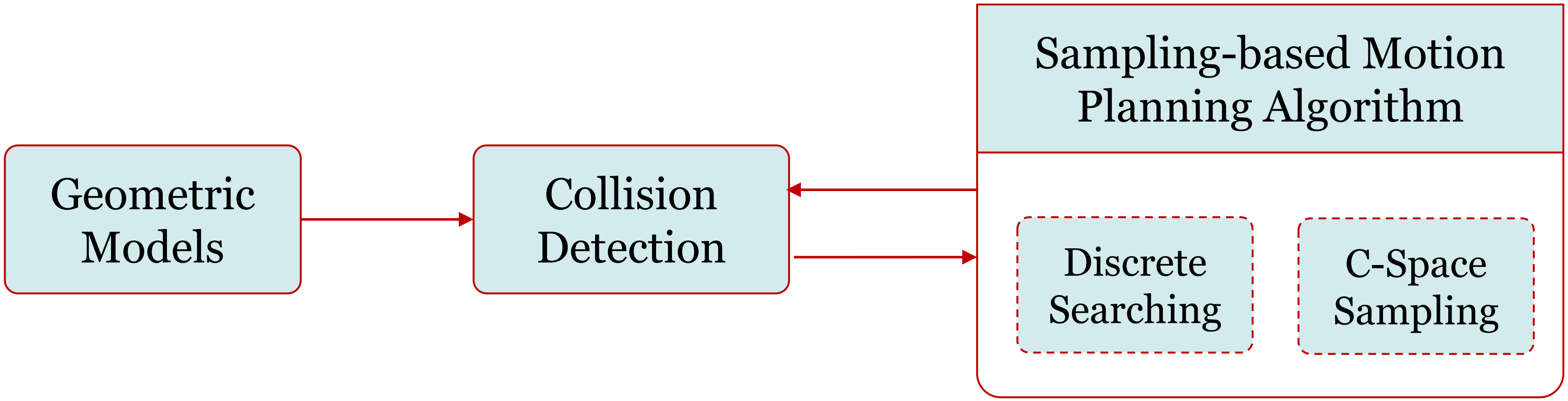
- **Approximate Cell Decomposition**
  - ◇ Pros:
    - Limited assumptions on obstacle configuration
    - Approach used in practice
    - Find obvious solutions quickly
  - ◇ Cons:
    - No clear notion of optimality (“best” path)
    - Trade-off completeness/computation
    - Still difficult to use in high dimensions

Outline

- Combinatorial Planning
- **Sampling-based Motion Planning**
- Potential Field Method

Sampling-based Motion Planning

- The main idea of sampling-based motion planning is to avoid the explicit construction of $\mathcal{C}_{\text{obs}}$, and instead conduct a search that probes the C-space with a sampling scheme.



- The sampling-based planning philosophy uses collision detection as a “black box” that separates the motion planning from the particular geometric and kinematic models.

- C-space sampling and discrete planning (i.e., searching) are performed.

[1]

Sampling-based Motion Planning

Sampling-based Motion Planning

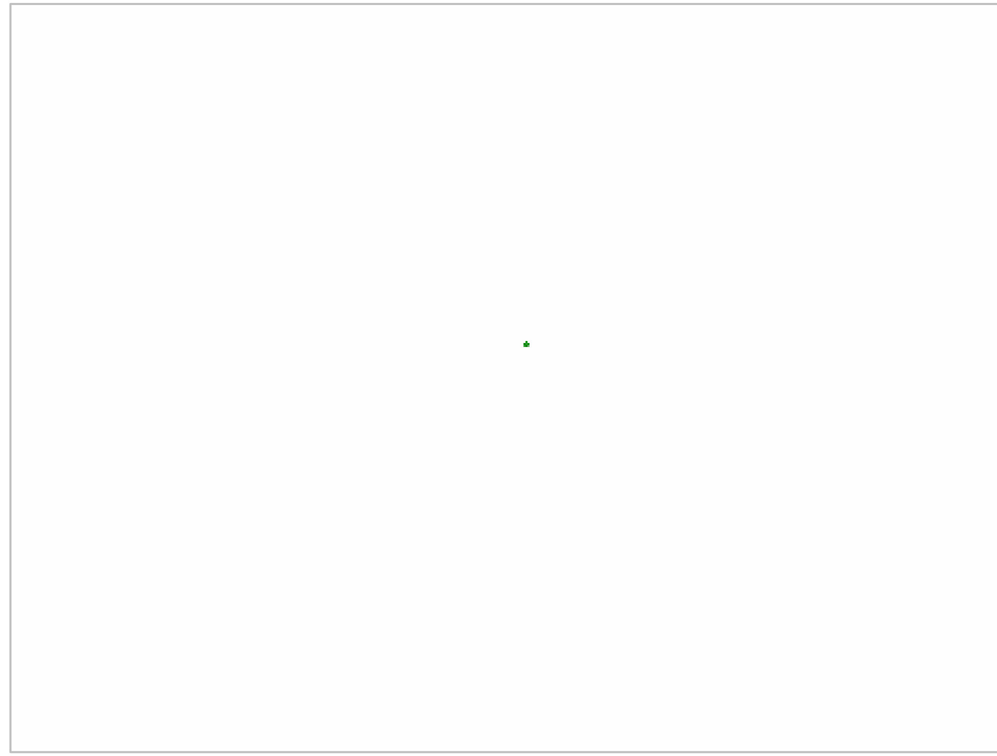
Single-query motion planning problems (single initial-goal pair)
Rapidly-exploring Random Trees (RRTs)

Multiple-query motion planning problems (numerous initial-goal queries)
Probabilistic Roadmaps (PRMs) or sampling-based roadmaps

Sampling-based Motion Planning

- **Rapidly-Exploring Random Tree (RRT)**

1. Initially, start with the initial configuration as root of tree
2. Pick a random state in the configuration space
3. Find the closest node in the tree
4. Extend that node toward the state if possible
5. Go to (2)



Sampling-based Motion Planning

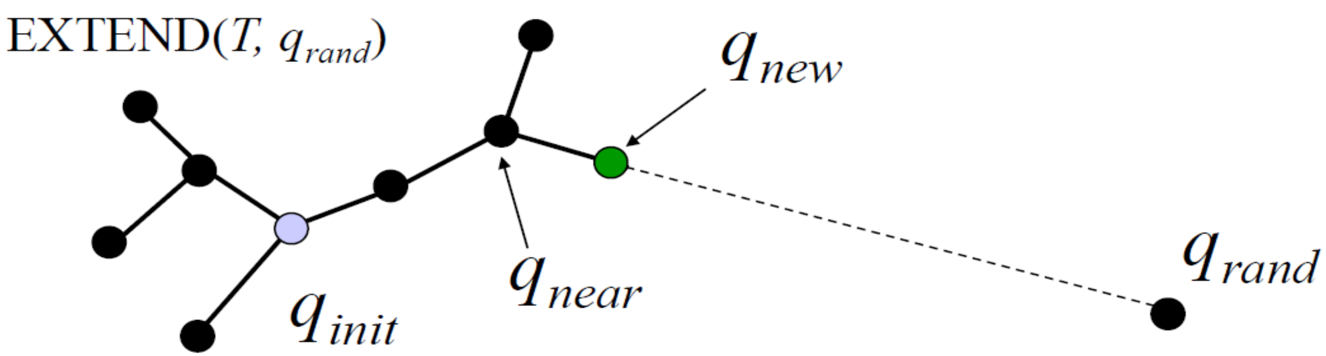
- **Rapidly-Exploring Random Tree (RRT)**

Algorithm BuildRRT

Input: Initial configuration qinit, number of vertices in RRT K, incremental distance Δq)

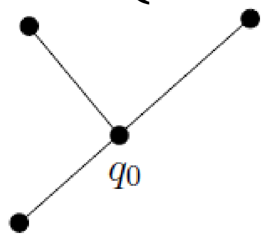
Output: RRT graph T

T.init(q_{init})
for k = 1 to K
    $q_{rand} \leftarrow$ RAND_CONF()
    $q_{near} \leftarrow$ NEAREST_VERTEX(q_{rand}, T)
    $q_{new} \leftarrow$ NEW_CONF(q_{near}, q_{rand}, Δq)
    T.add_vertex(q_{new})
    T.add_edge(q_{near}, q_{new})
return T

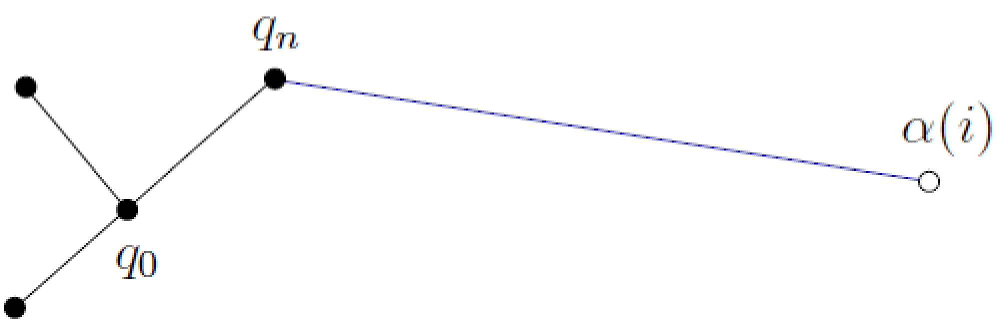


Sampling-based Motion Planning

- **Rapidly-Exploring Random Tree (RRT)**
  - ◇ Initially, a vertex is made at q_o



  - ◇ A new edge is added that connects $\alpha(i)$ from the sample to the nearest point in the swath S, which is the vertex q_n.



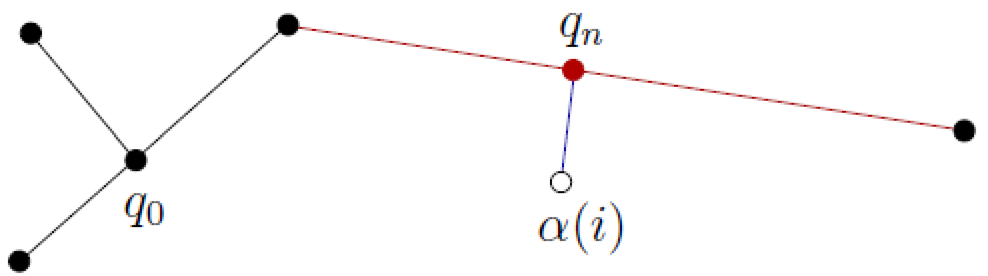
where swath, S, of the graph

$$S = \bigcup_{e \in E} e([0,1])$$

$e([0,1]) \subseteq C_{free}$ is the image of the path e [1]

Sampling-based Motion Planning

- **Rapidly-Exploring Random Tree (RRT)**
  - ◇ For ***k* iterations**, a tree is **iteratively grown** by connecting $\alpha(i)$ to its nearest point in the swath, S.



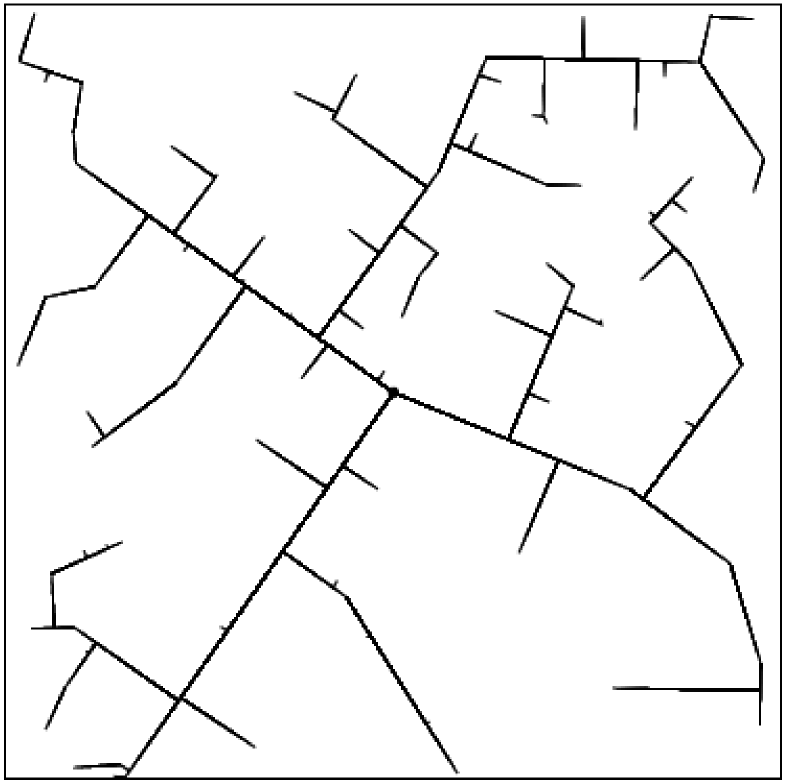
If the nearest point in S lies in an edge, then the edge is split into two, and a new vertex is inserted into the graph

  - ◇ The connection is usually made along the **shortest possible path**. In every iteration, $\alpha(i)$ becomes a vertex. Therefore, the **resulting tree is dense**.

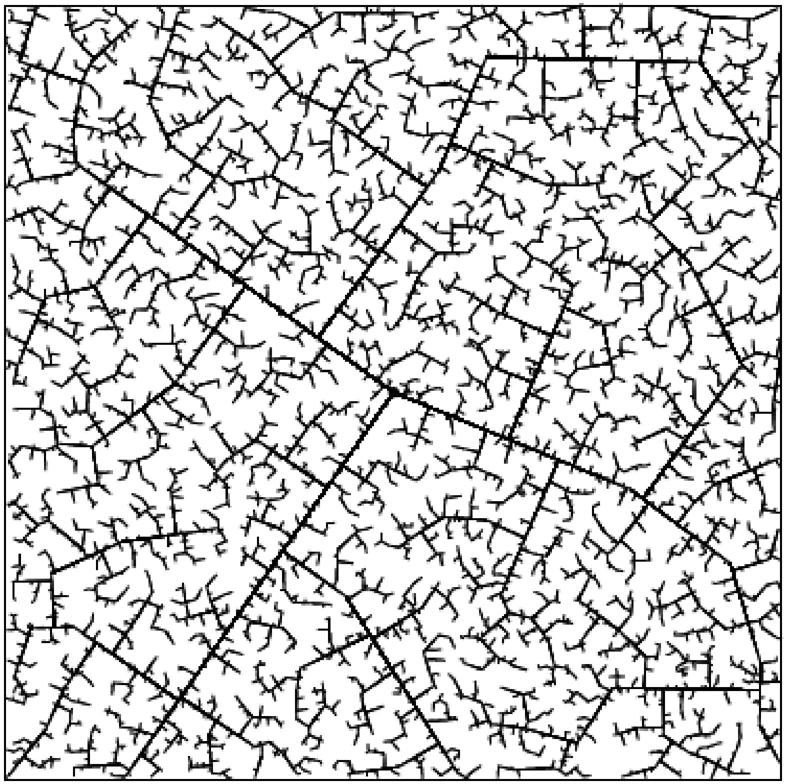
[1]

Sampling-based Motion Planning

- **Rapidly-Exploring Random Tree (RRT)**



45 iterations



2345 iterations

In the early iterations, the RRT quickly reaches the unexplored parts. However, the RRT is dense in the limit (with probability one), which means that it gets arbitrarily close to any point in the space.

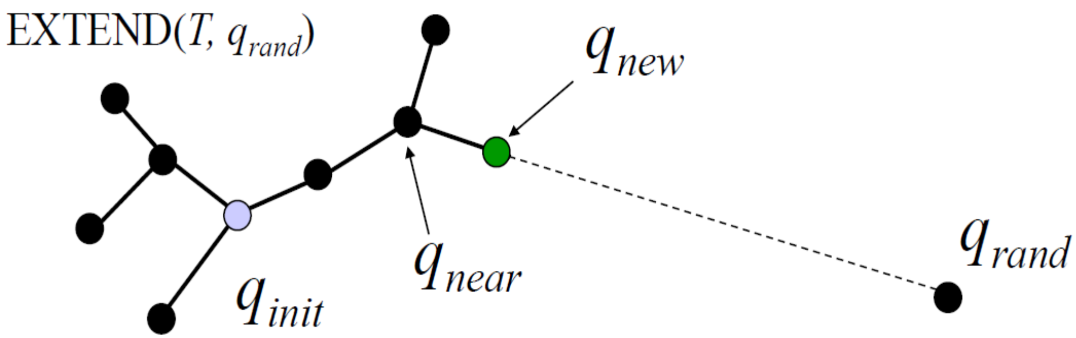
[1]

Sampling-based Motion Planning

- **Rapidly-Exploring Random Tree (RRT)**

STEP_LENGTH: How far to sample

1. Sample just at end point
2. Sample all along
3. Small Step



Extend returns

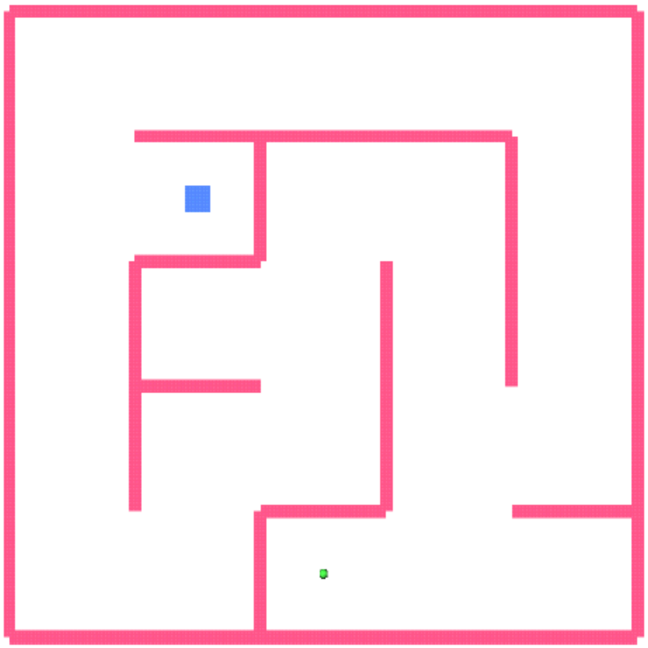
1. Trapped, cant make it
2. Extended, steps toward node
3. Reached, connects to node

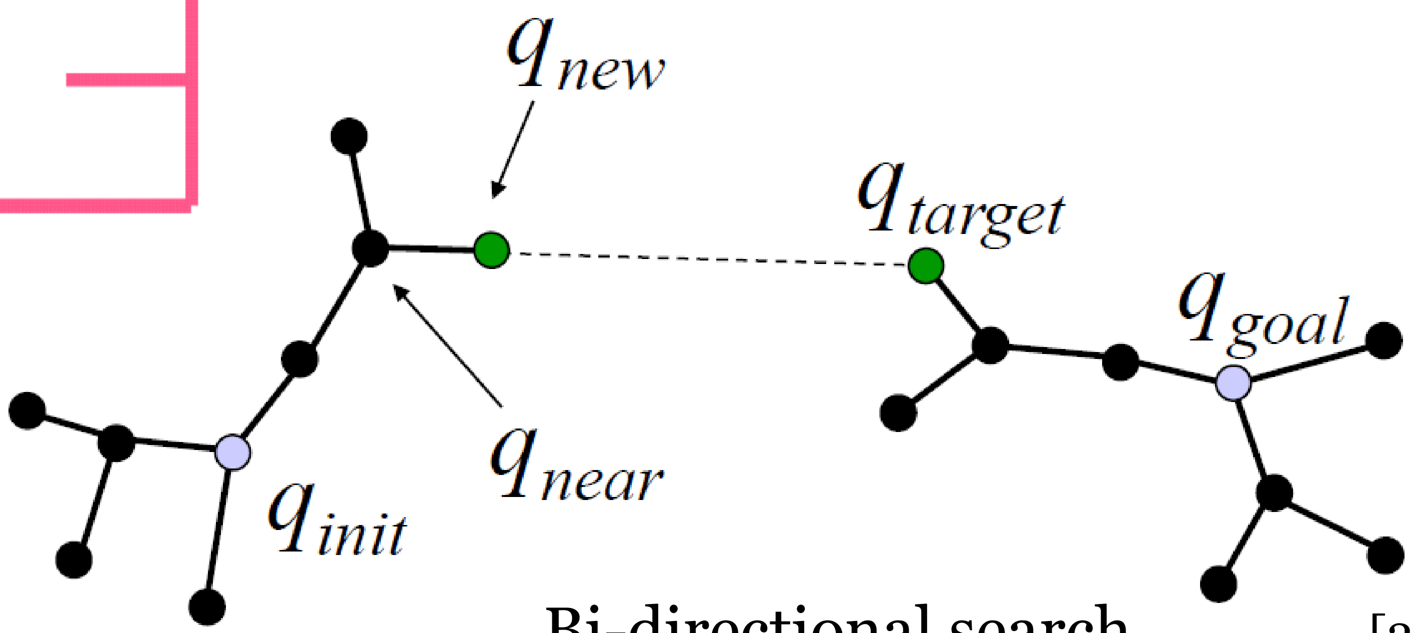
Collection Check

Lazy collision checking 6x faster than checking every single point for collision at the time when it's added

Sampling-based Motion Planning

- **Rapidly-Exploring Random Tree (RRT)**

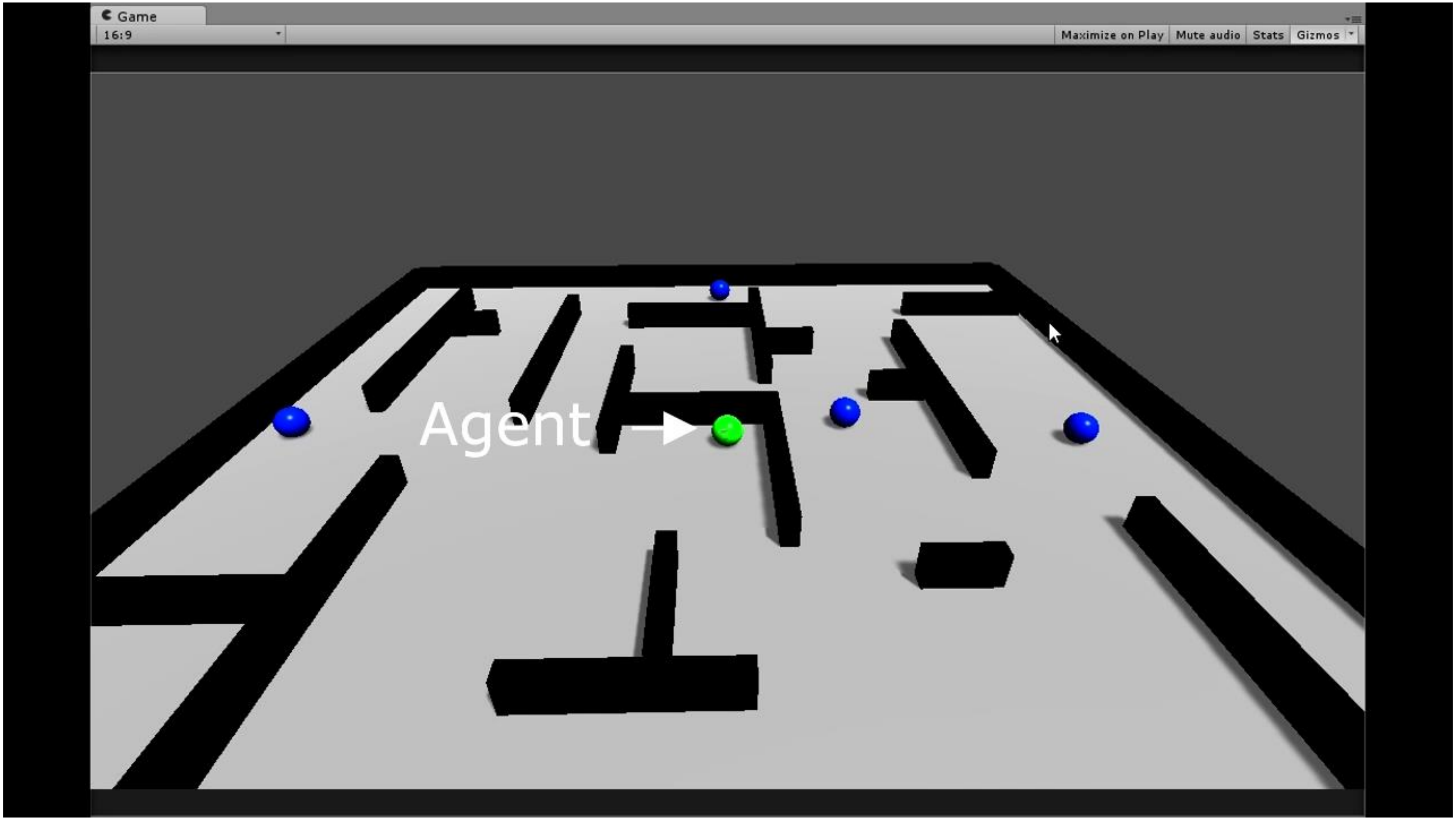




Bi-directional search [2]

Sampling-based Motion Planning

- **Rapidly-Exploring Random Tree (RRT)**

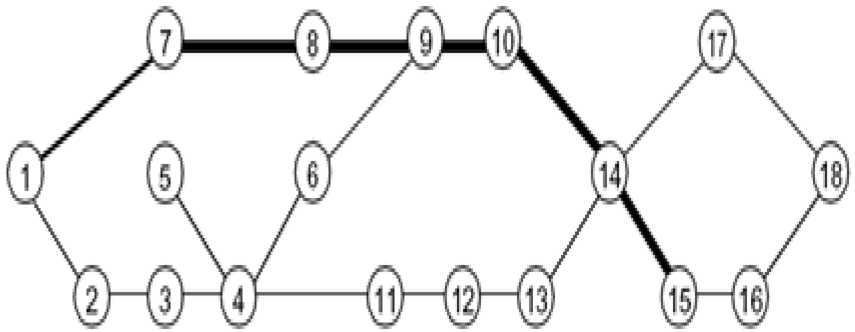


A real-time path planning algorithm based on RRT*

Sampling-based Motion Planning

- **Probabilistic Roadmaps (PRMs)**

Given: G(V,E) represents a topological graph in which V is a set of vertices and E is the set of paths that map into $\mathcal{C}_{\text{free}}$.



Under the multiple-query philosophy, motion planning is divided into two phases of computation:

Preprocessing/Learning Phase → roadmap → Query Phase → Path

Build G in a way that is useful for quickly answering future queries. For this reason, it is called a roadmap, which in some sense should be accessible from every part of $\mathcal{C}_{\text{free}}$.

query
(q_{init},q_{goal}) pair

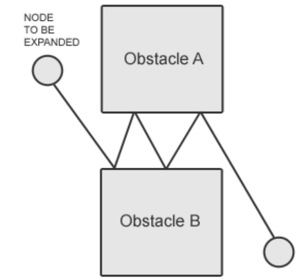
[1]

Sampling-based Motion Planning

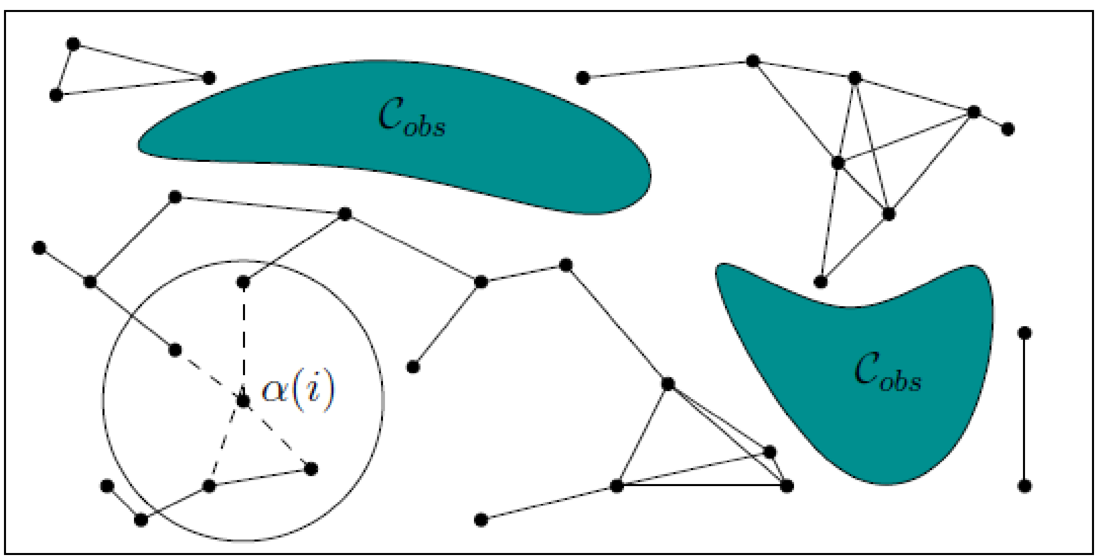
- **PRM: Preprocessing/Learning Phase**

```
BUILD_ROADMAP
G.init(); i ← 0;
while i < N
    if α(i) ∈ C_free then
        G.add_vertex(α(i)); i ← i + 1;
        for each q ∈ NEIGHBORHOOD(α(i),G)
            if ((not G.same_component(α(i), q)) and CONNECT(α(i), q)) then
                G.add_edge(α(i), q);
```

Possible selection methods:
- Nearest K;
- Radius;
- Visibility
- …



Note that *i* is not incremented if *α*(*i*) is in collision. This forces *i* to correctly count the number of vertices in the roadmap.

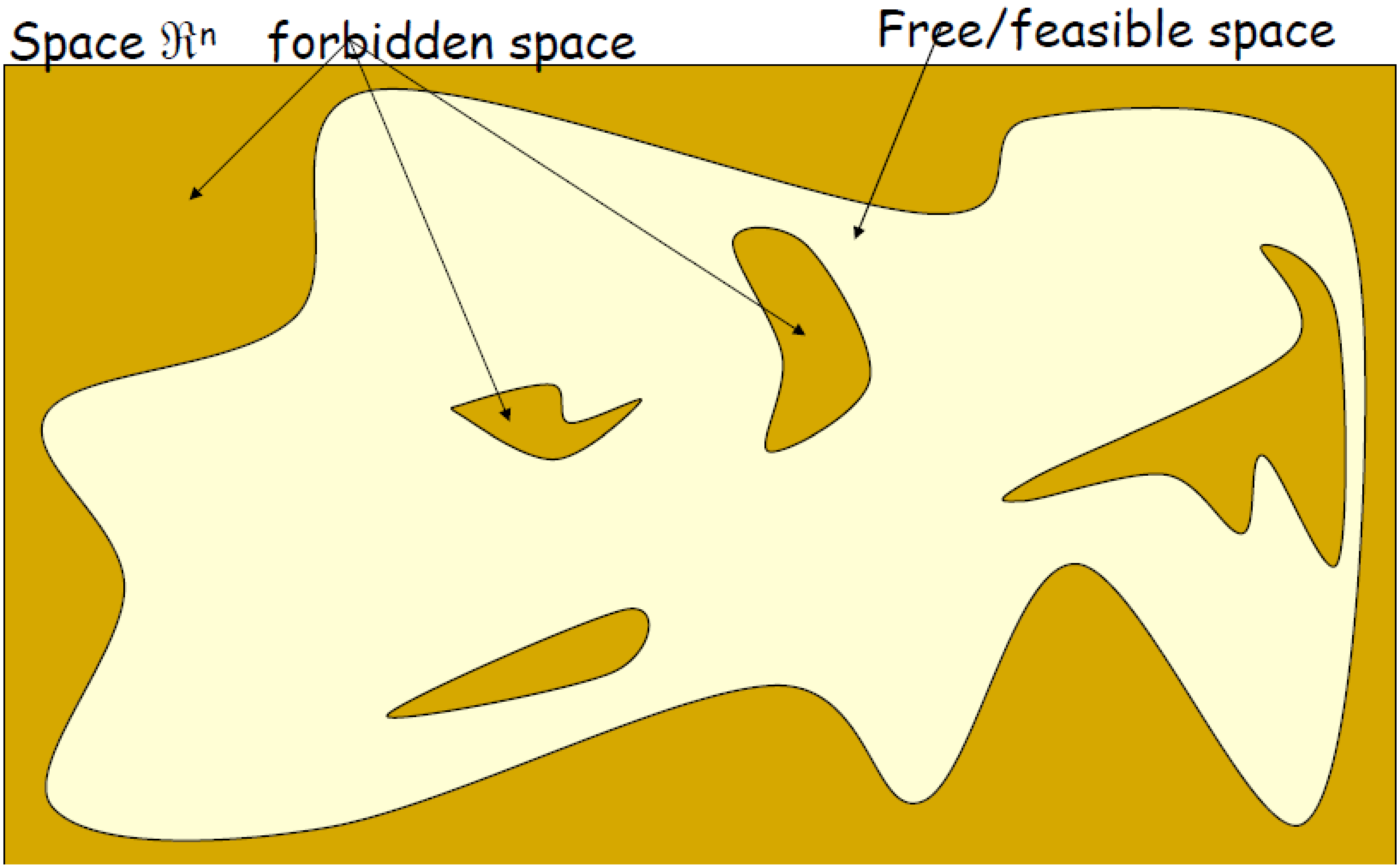


The sampling-based roadmap is constructed incrementally by attempting to connect each new sample, *α*(*i*), to nearby vertices in the roadmap

[1]

Sampling-based Motion Planning

- **PRM: Preprocessing/Learning Phase**

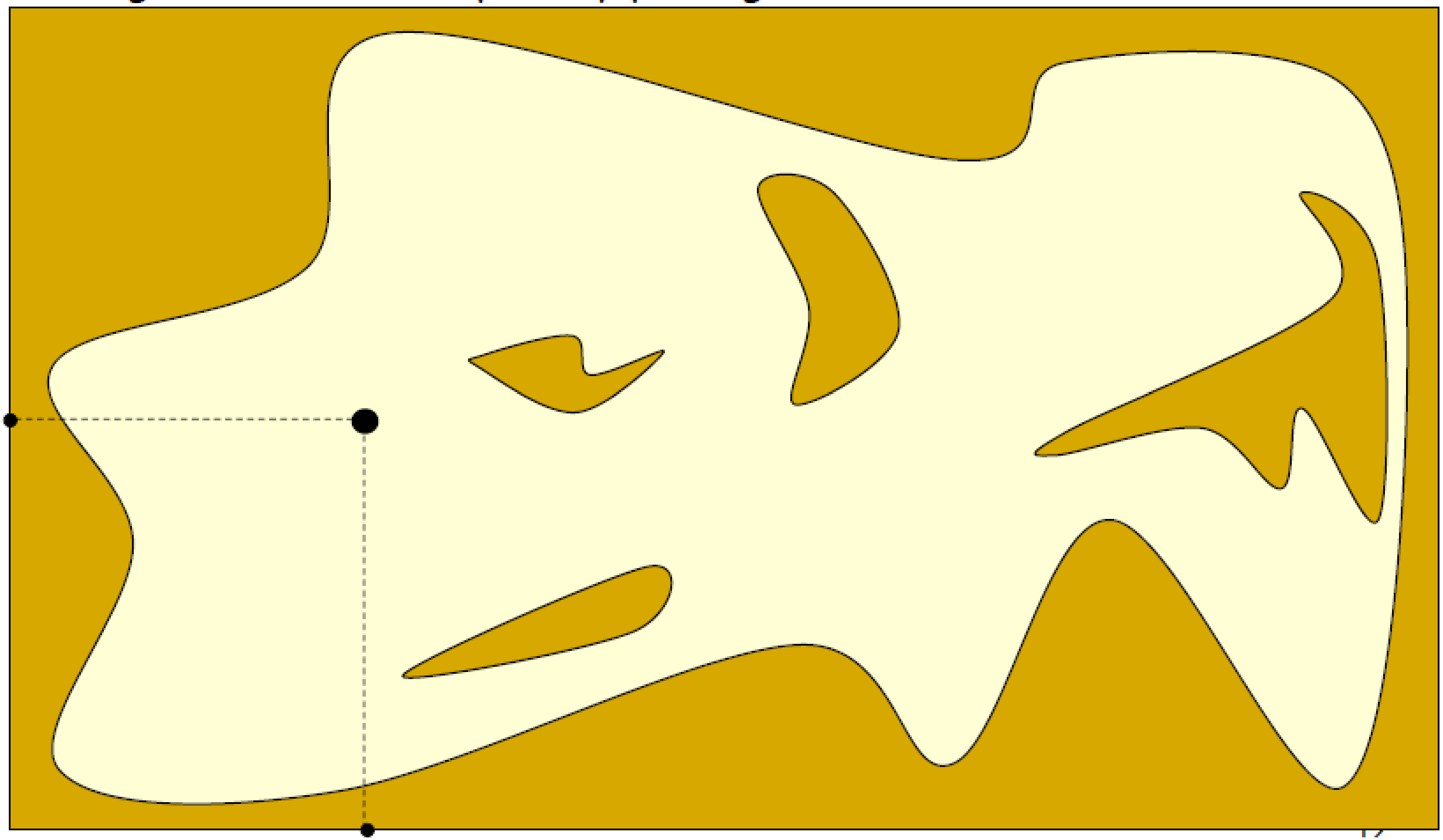


[3]

Sampling-based Motion Planning

- **PRM: Preprocessing/Learning Phase**

Configurations are sampled by picking coordinates at random

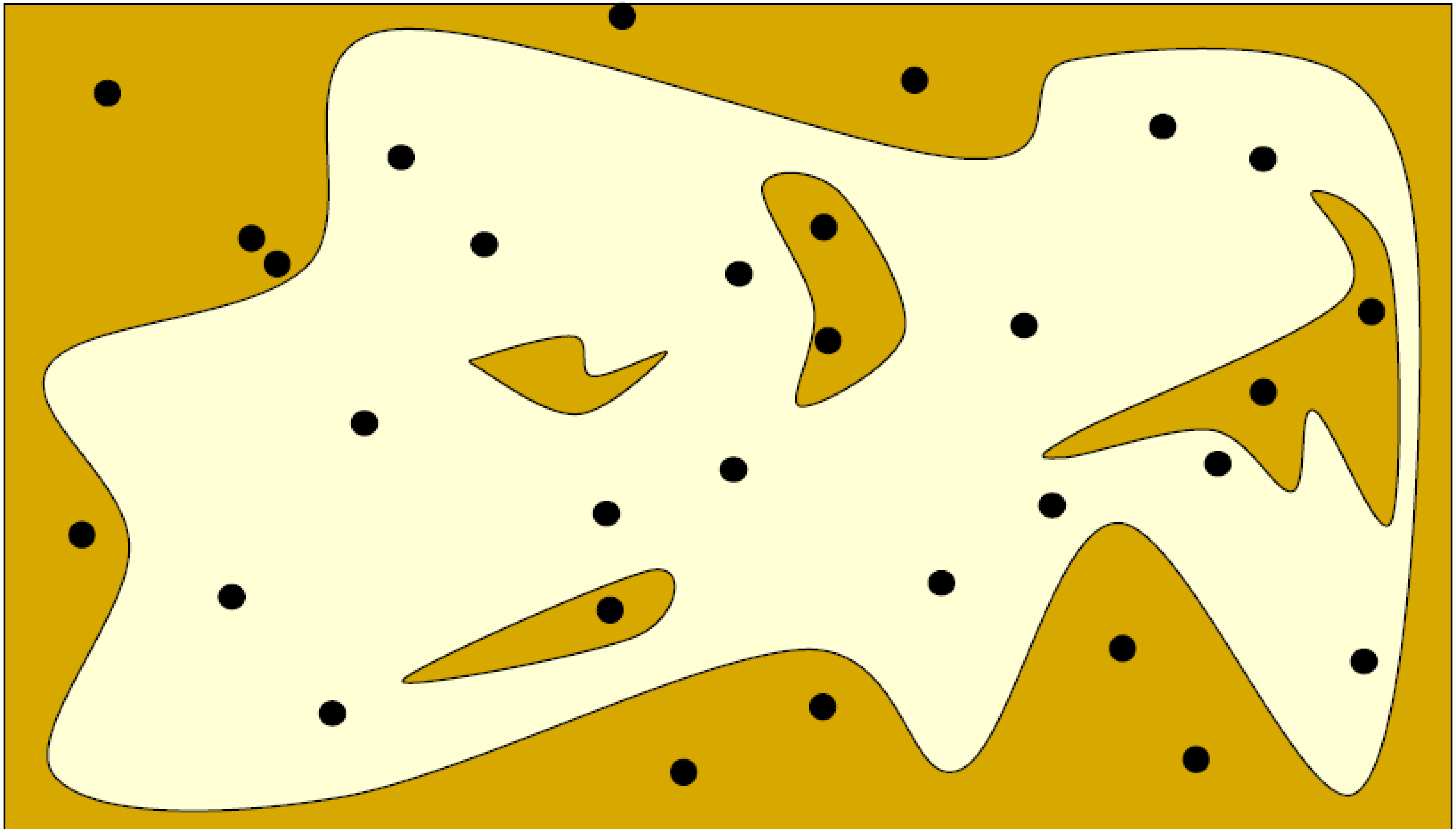


[3]

Sampling-based Motion Planning

- **PRM: Preprocessing/Learning Phase**

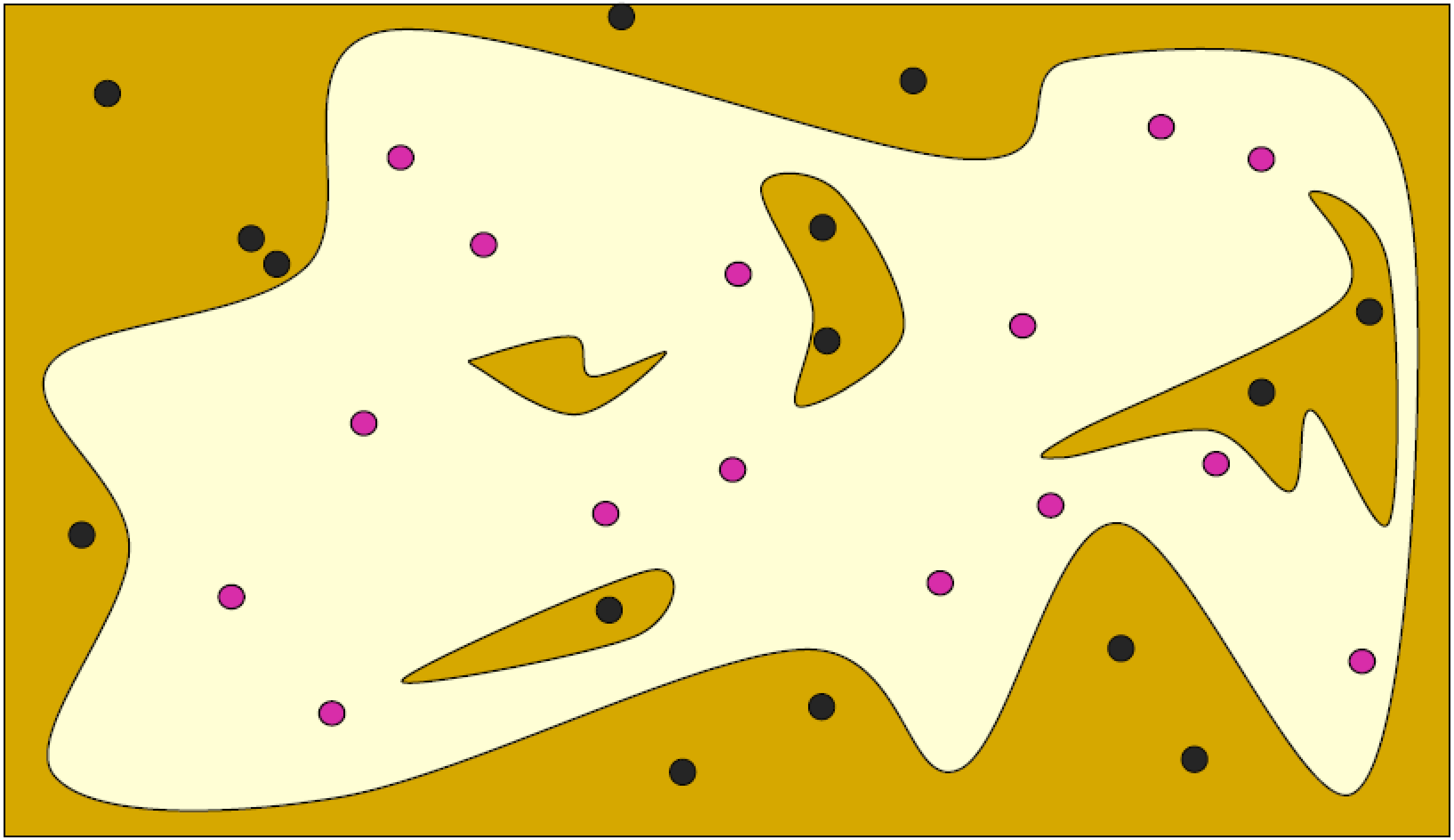
Configurations are sampled by picking coordinates at random



[3]

Sampling-based Motion Planning

- **PRM: Preprocessing/Learning Phase**

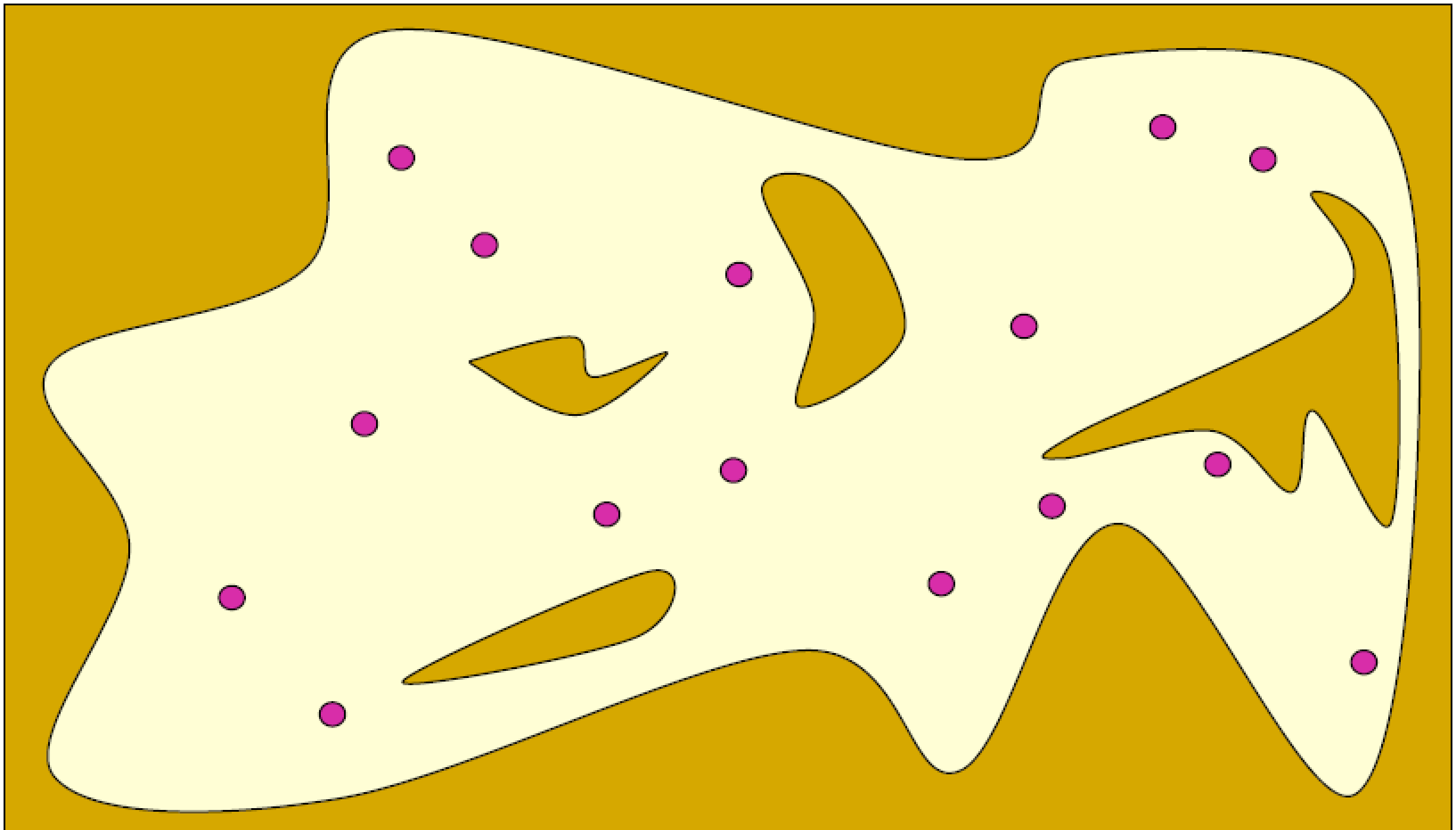


[3]

Sampling-based Motion Planning

- **PRM: Preprocessing/Learning Phase**

The collision-free configurations are retained as milestones

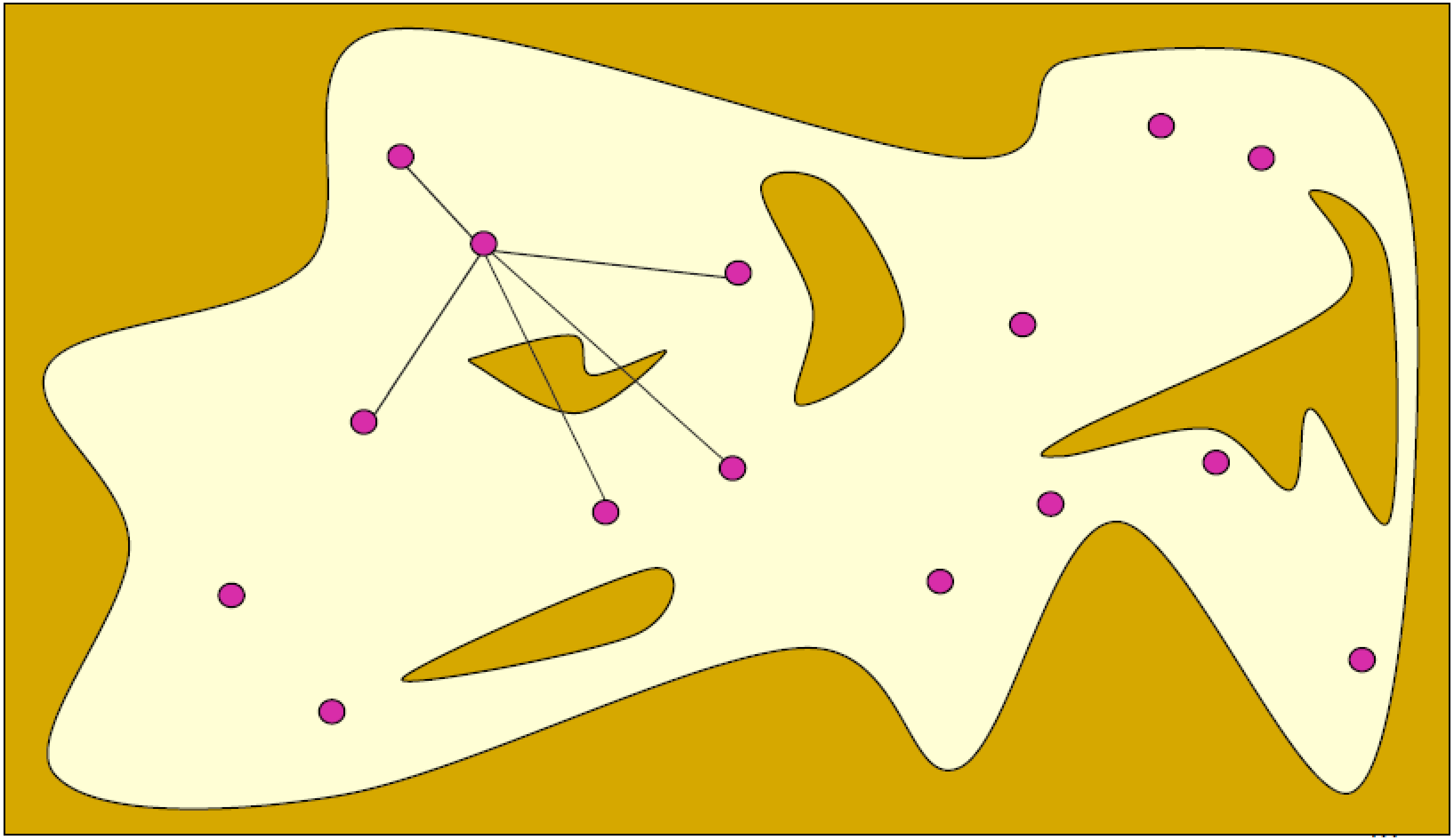


[3]

Sampling-based Motion Planning

- **PRM: Preprocessing/Learning Phase**

Each milestone is linked by straight paths to its nearest neighbors

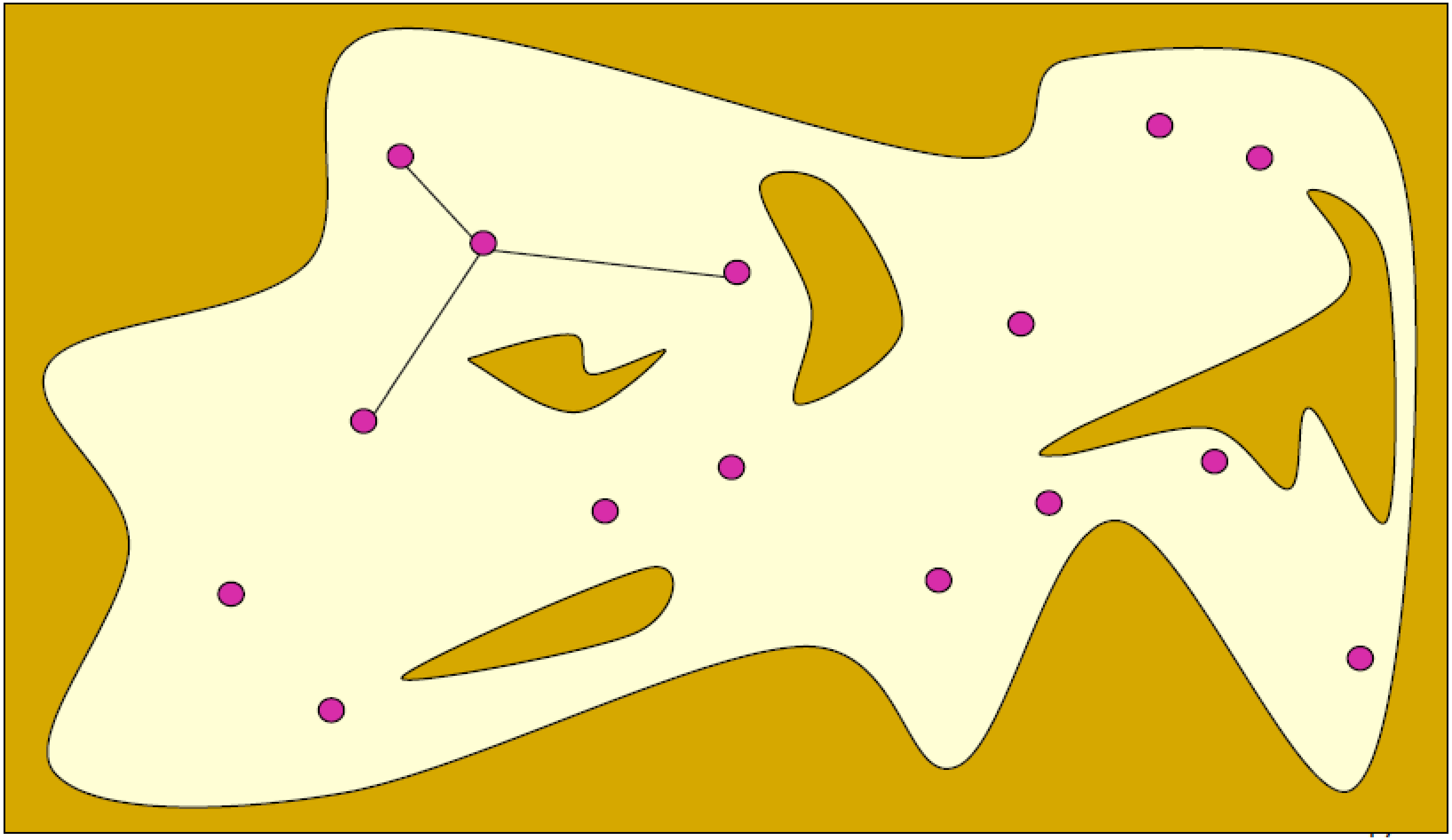


[3]

Sampling-based Motion Planning

- **PRM: Preprocessing/Learning Phase**

Each milestone is linked by straight paths to its nearest neighbors

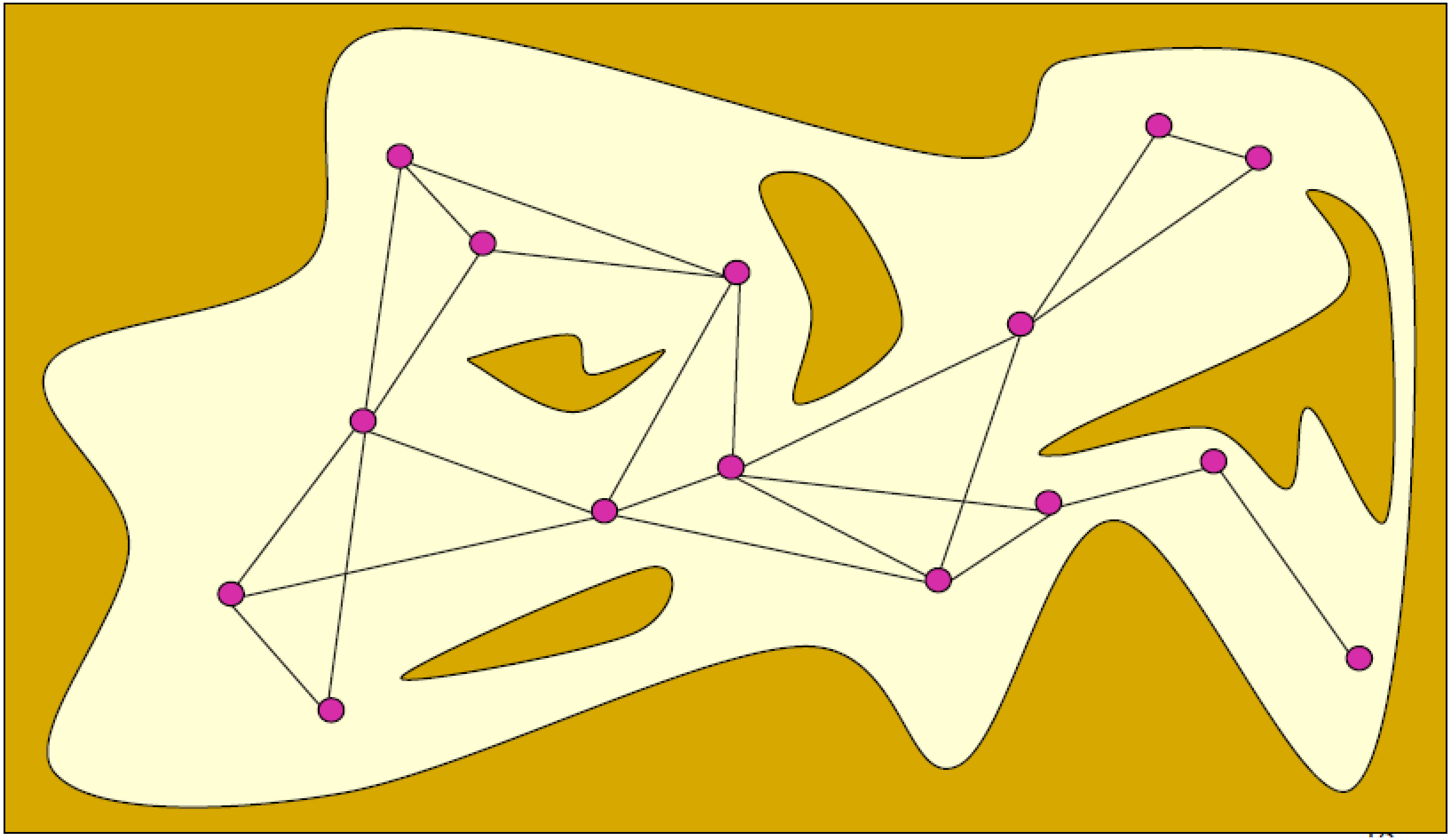


[3]

Sampling-based Motion Planning

- **PRM: Preprocessing/Learning Phase**

The collision-free links are retained as local paths to form the PRM

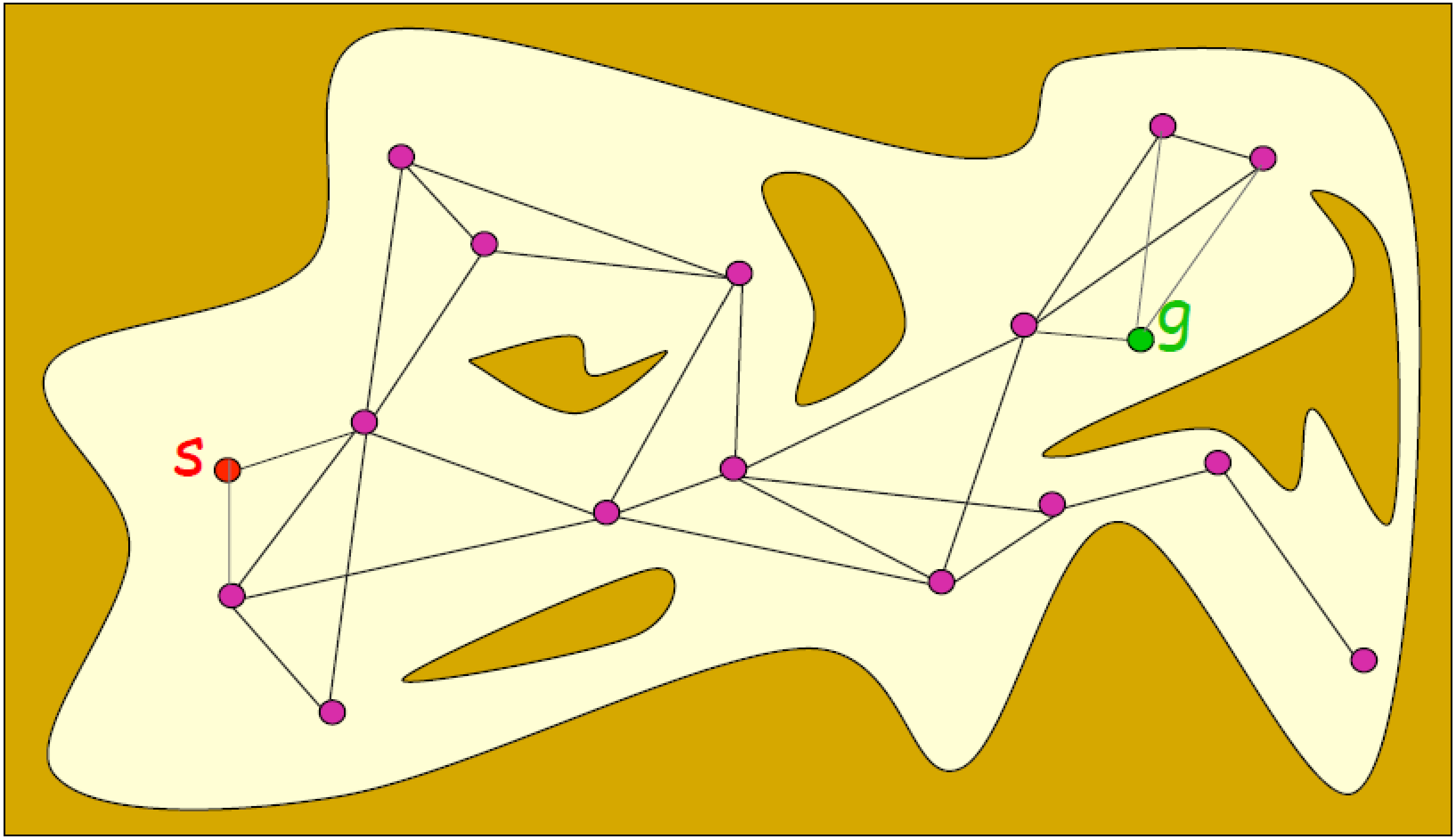


[3]

Sampling-based Motion Planning

- **PRM: Preprocessing/Learning Phase**

The start and goal configurations are included as milestones

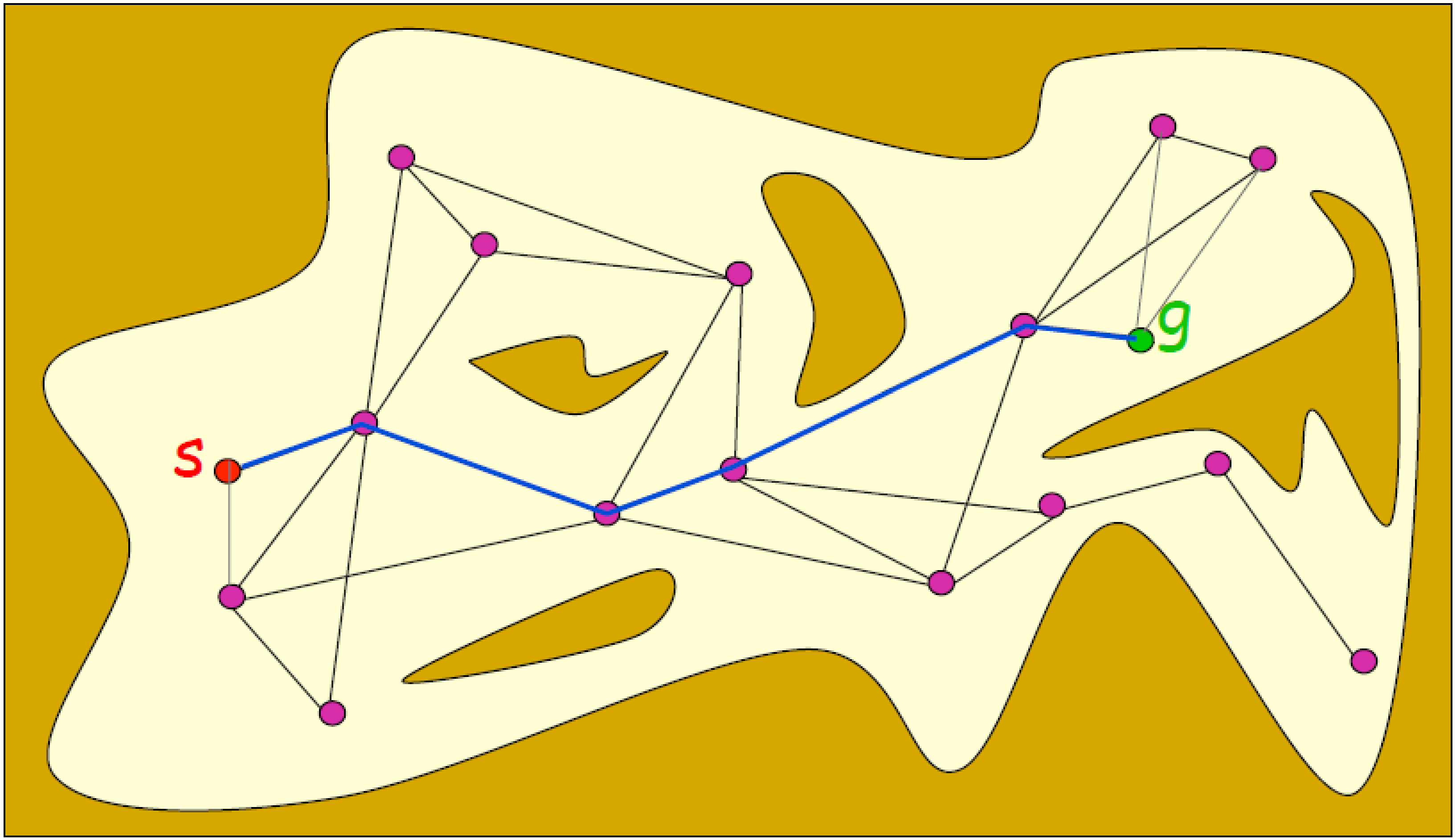


[3]

Sampling-based Motion Planning

- **PRM: Preprocessing/Learning Phase**

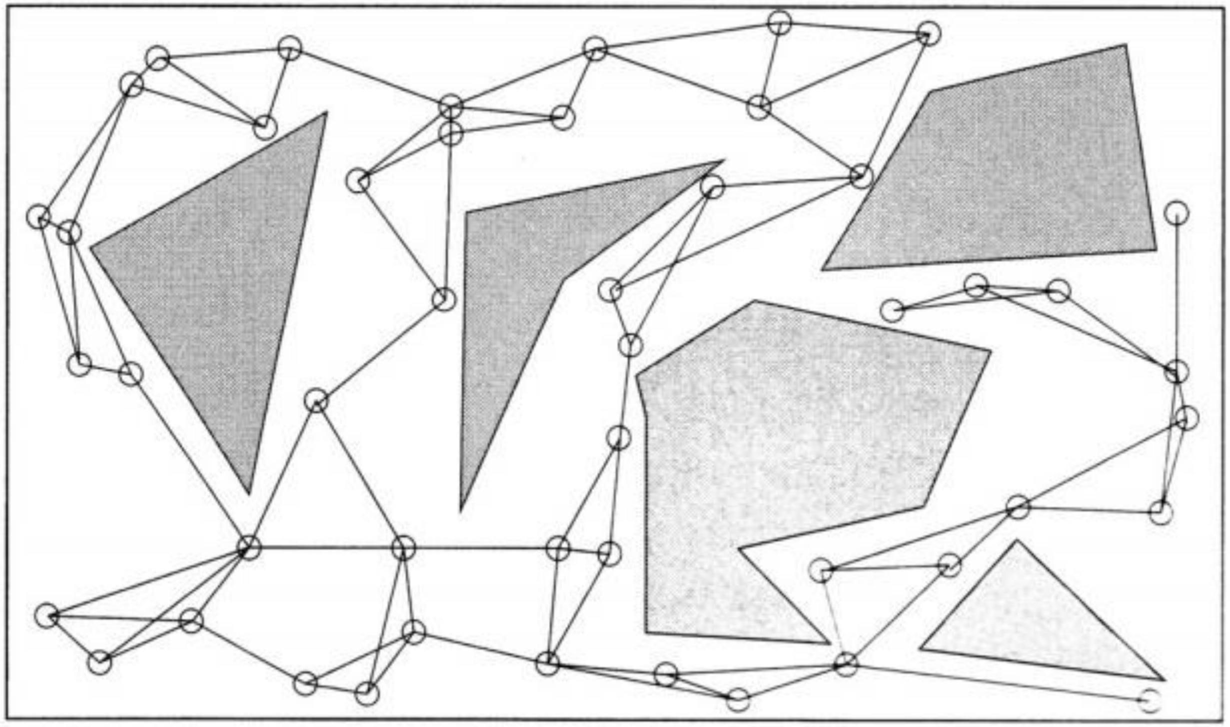
The PRM is searched for a path from s to g



[3]

Sampling-based Motion Planning

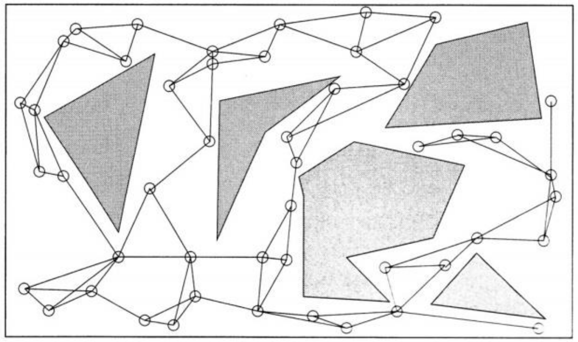
- **PRM: Preprocessing/Learning Phase**



Example of a roadmap for a point root in a two-dimensional Euclidean space. The gray areas are obstacles. The empty circles correspond to the nodes of the roadmap. The straight lines between circles correspond to edges. The number k closet neighbors for the construction of roadmap is three. The degree of a node can be greater than three since it may be included in the closest neighbor list of many nodes.

Sampling-based Motion Planning

- **PRM: Query Phase**
  - ◇ Given an initial position and a final position or (q_{init}, q_{goal}) pair, the roadmap should compute a collision-free path between these two configurations.

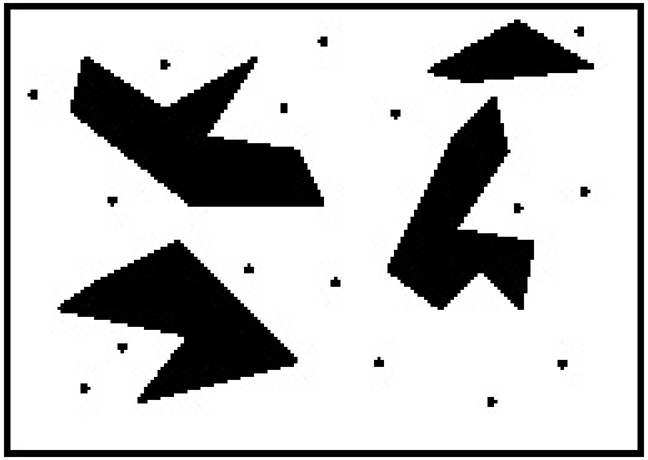


  - ◇ First, we create new nodes for each of the initial and final position we add them to the graph after a collision test if needed.
  - ◇ Then, we try to connect the two nodes to the graph to any of their neighbors, just like as we did in the learning phase.
  - ◇ If the path planner fails to compute a feasible path between the new nodes and the existing nodes, the query phase fails.

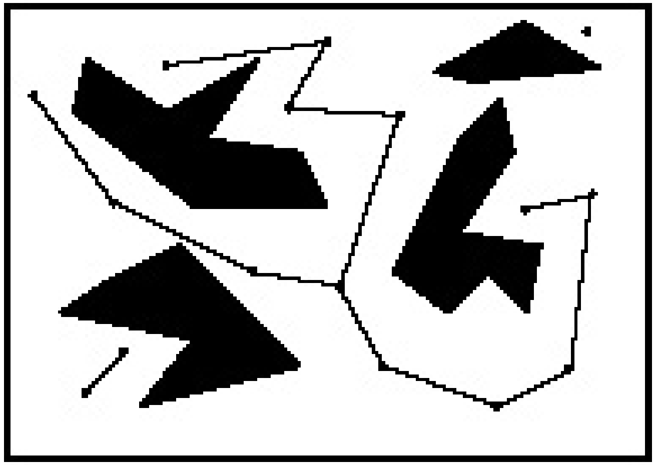
Sampling-based Motion Planning

- **PRM: Summary**

a) A set of random sample is generated in the configuration space. Only collision-free samples are retrained.



b) Each sample is connected to its nearest neighbors using a simple, straight-line path. If such a path causes a collision, the corresponding samples are not connected in the roadmap

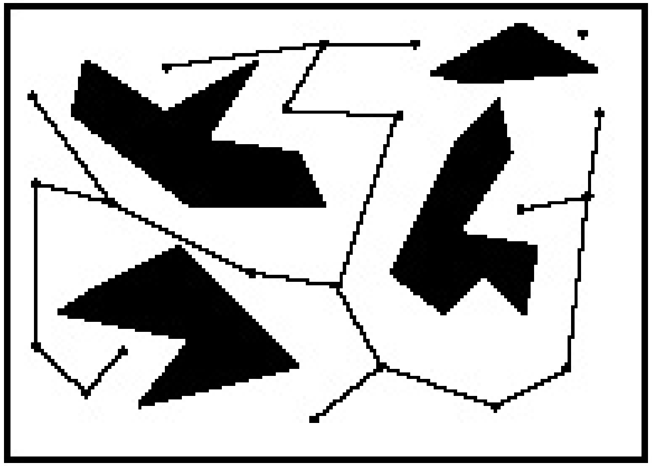


[4]

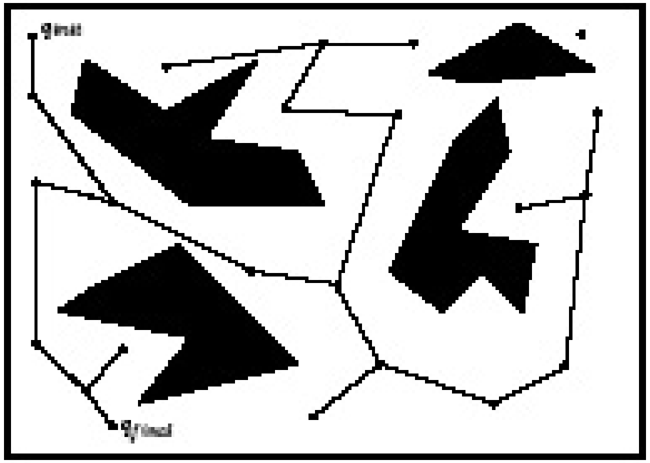
Sampling-based Motion Planning

- **PRM: Summary**

  c) Since the initial roadmap contains multiple connected components, additional samples are generated and connected to the roadmap.



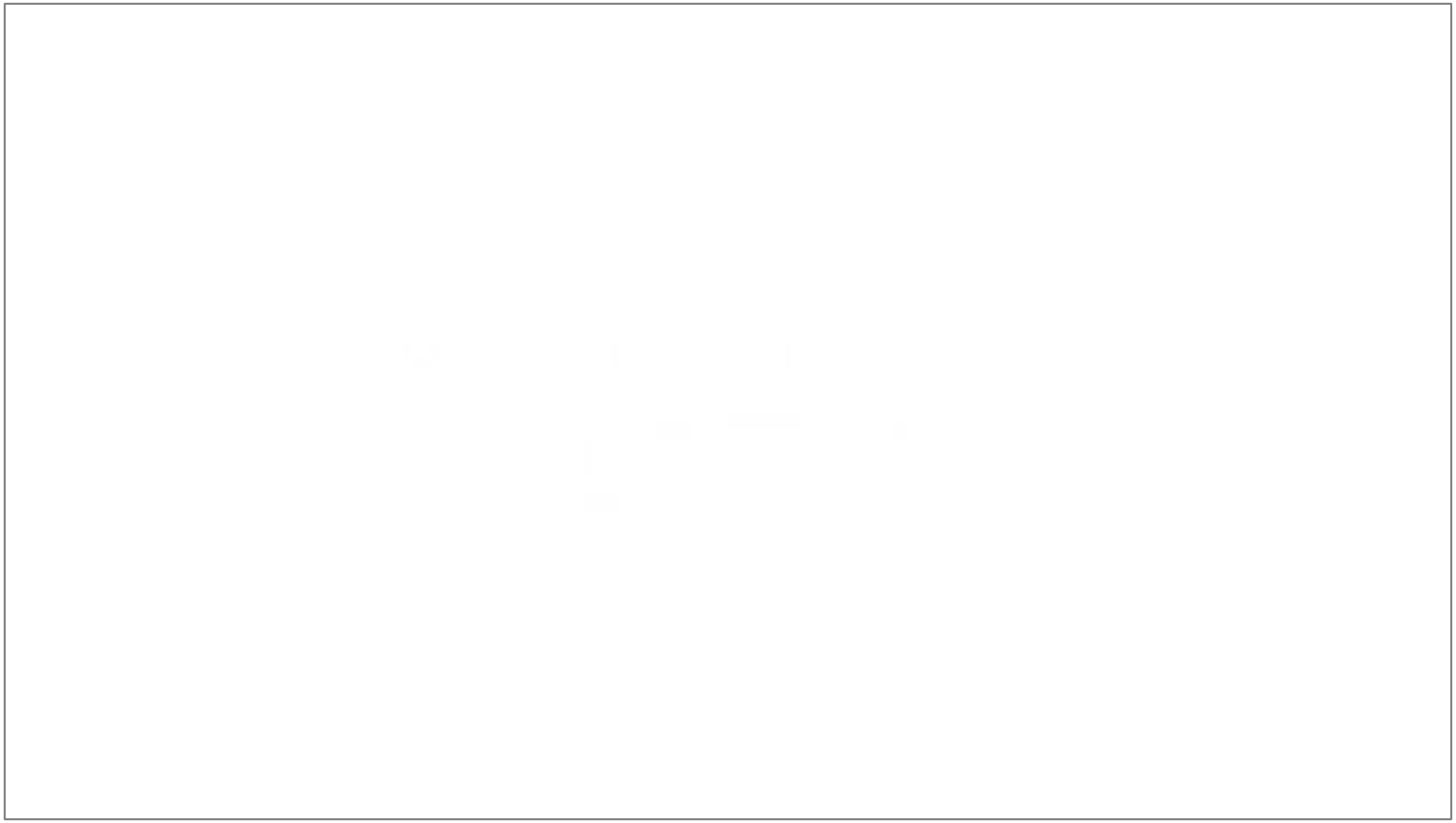
  d) A path from q_{init} to q_{goal} is found by connected q_{init} and q_{goal} to the roadmap and then searching this augmented roadmap for a path from q_{init} to q_{goal}



[4]

Sampling-based Motion Planning

- **PRM: Summary**



[Custom Processor Speeds Up Robot Motion Planning by Factor of 1,000]

Outline

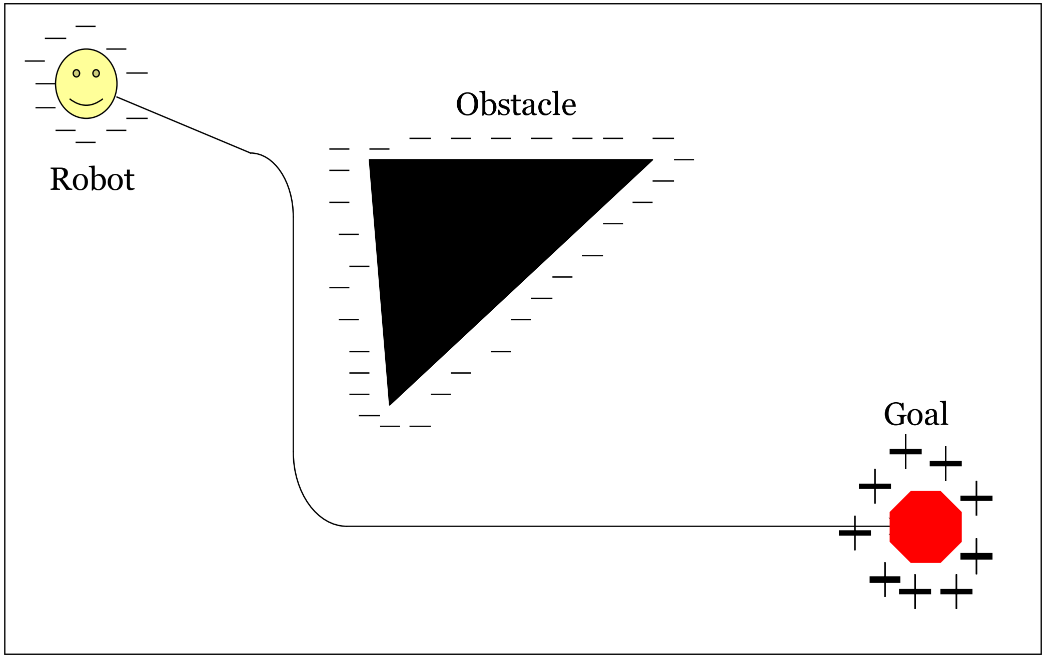
- Combinatorial Planning
- Sampling-based Motion Planning
- **Potential Field Method**

Potential Field Method

The potential field method **incrementally explores $\mathcal{C}_{\text{free}}$**, searching for a path from q_{init} to q_{final}. At termination, this planner returns a single path.

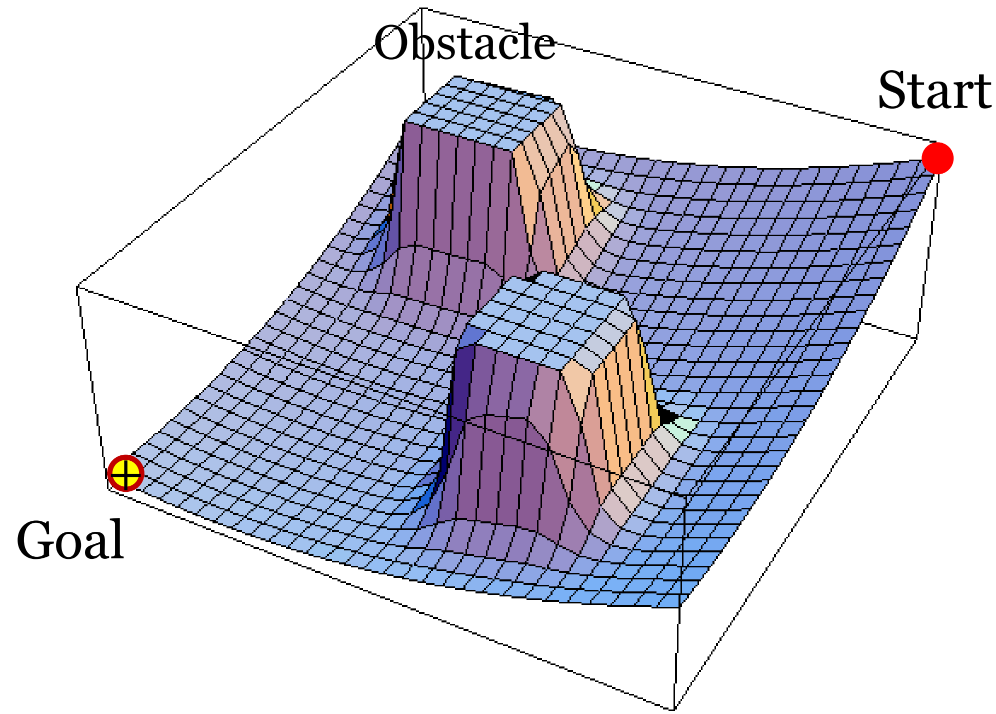
Potential field path planning creates a **field**, or gradient, across the robot's map that **directs the robot** to the goal position from multiple prior positions.

The potential field method treats the robot as a **point** under the influence of an **artificial potential field, U(q)**.



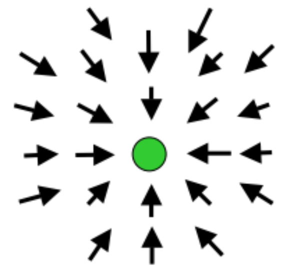
Potential Field Method

The robot moves by following the field, just as a **ball would roll downhill**. The **goal (a minimum in this space)** acts as an **attractive force** on the robot and the obstacles act as peaks, or repulsive forces. The superposition of all forces is applied to the robot.

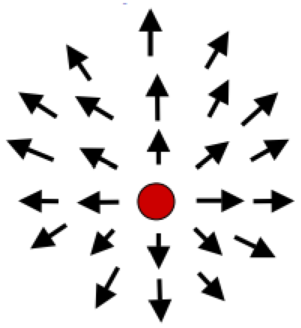


Potential Field Method

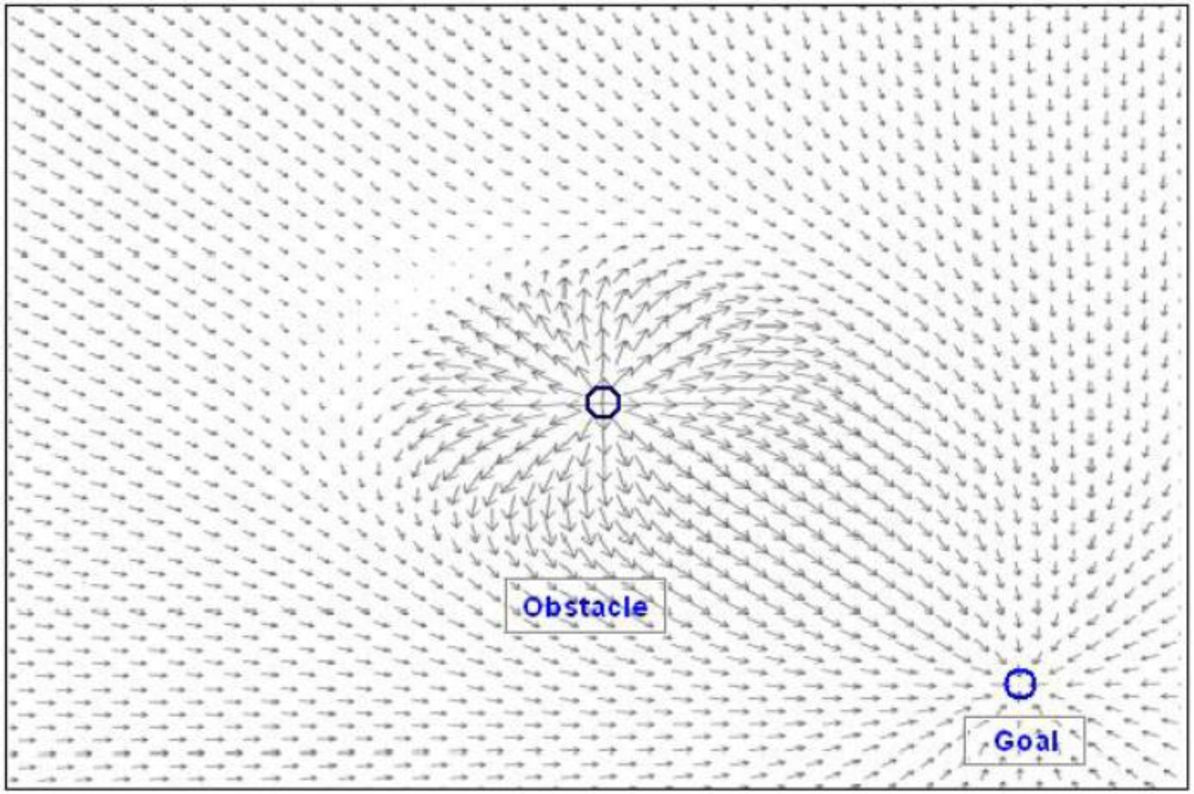
The **artificial potential field** smoothly **guides the robot** toward the goal while simultaneously **avoiding known obstacles**.



Attraction



Repulsion



Potential Field Method

Assume that the robot is a **point**, thus the robot's orientation θ is neglected and the resulting potential field is only 2D (*x*,*y*).

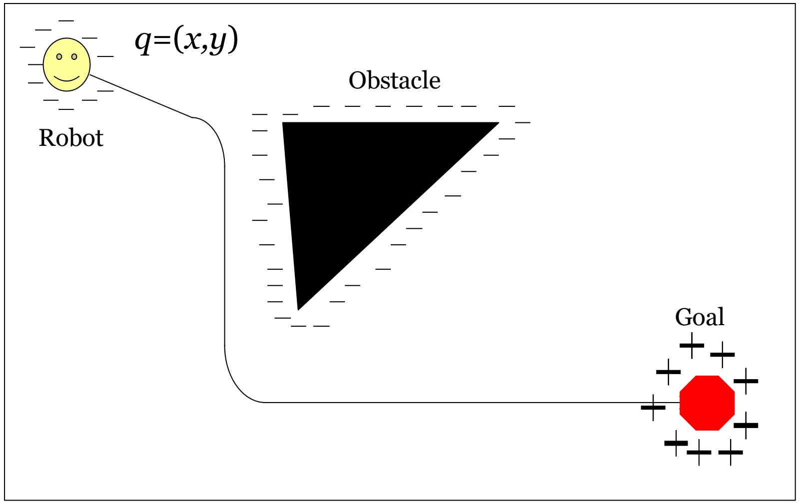
Assume a **differentiable potential field function** U(q).

The related **artificial force** acting at the position *q*=(*x*,*y*) is

$$F(q) = -\nabla U(q)$$

where $\nabla U(q)$ denotes the **gradient** vector of U at position q.

$$\nabla U(q) = \begin{bmatrix} \dfrac{\partial U}{\partial x} \\ \dfrac{\partial U}{\partial y} \end{bmatrix}$$



Potential Field Method

The **potential field** acting on the robot is then computed as the sum of the **attractive field of the goal** and **the repulsive fields of the obstacles**:

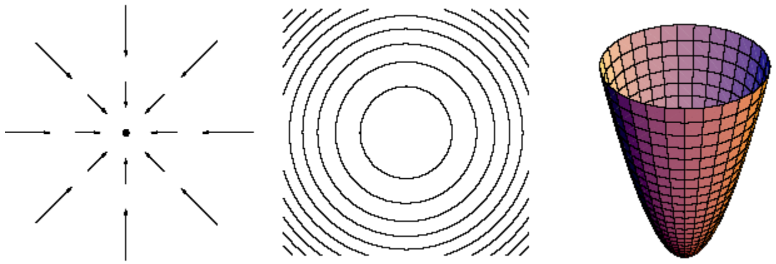
$$U(q) = U_{att}(q) + U_{rep}(q)$$

- ◊ U_{att} is the "attractive" potential --- move to the goal
- ◊ U_{rep} is the "repulsive" potential --- avoid obstacles

Potential Field Method

- **Attractive Potential**

Quadratic Potential



(a) (b) (c)

$$U_{att}(q) = \frac{1}{2} k_{att} . d^2_{goal}(q)$$

where

◇ k_{att} is a positive scaling factor

◇ d_{goal} denotes the Euclidean distance $\left\| q - q_{goal} \right\|$

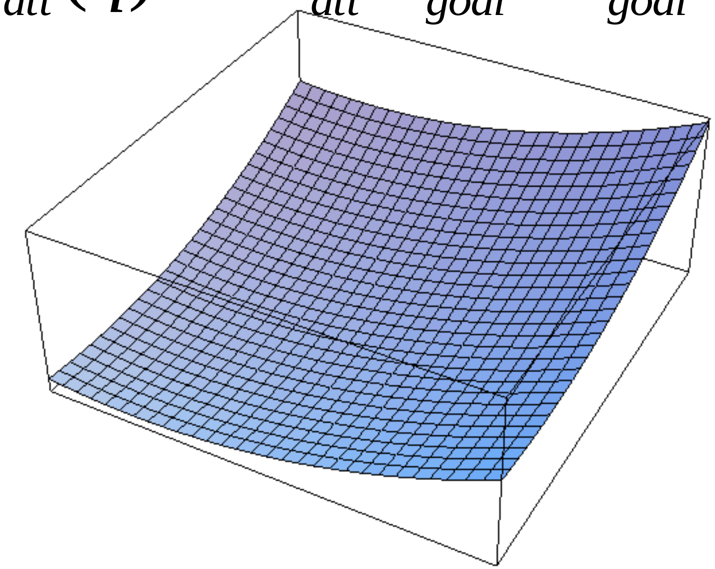
Potential Field Method

- **Attractive Potential**

$$U_{att}(q) = \frac{1}{2} k_{att} . d_{goal}^{2}(q)$$

This attractive potential is **differentiable**, leading to the **attractive force:**

$$F_{att}(q) = -\nabla U_{att}(q) = -k_{att} . d_{goal} \nabla d_{goal} = -k_{att} . (q - q_{goal})$$



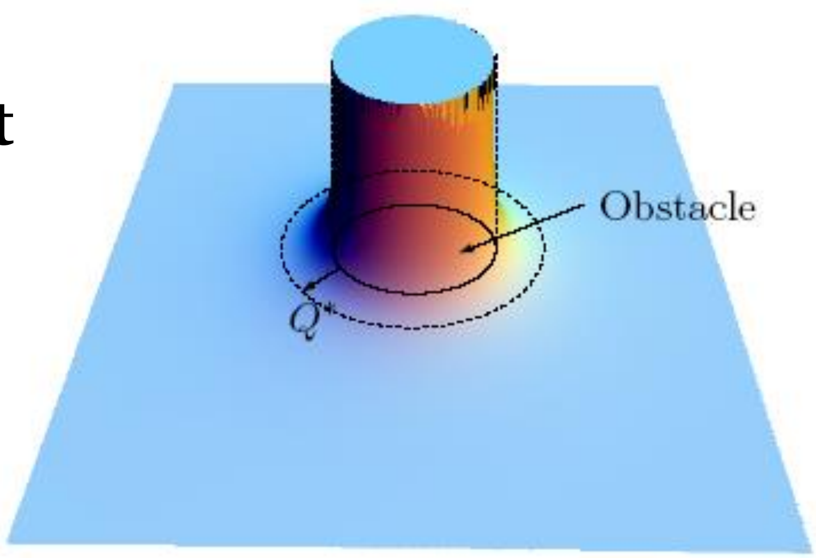
This converges linearly toward **0** as the robot reaches the **goal**

NOTE: q is a vector corresponding to a position in the workspace

$$q = \begin{bmatrix} q_x \\ q_y \end{bmatrix}$$

Potential Field Method

- **Repulsive Potential**
  - ◇ The idea behind the repulsive potential is to generate a **force away** from all known obstacles.
  - ◇ This repulsive potential **should be very strong** when the **robot is close to the object**, but **should not** influence its movement when the robot is **far** from the object.



Potential Field Method

- **Repulsive Potential**

$$U_{rep}(q) = \begin{cases} \frac{1}{2} k_{rep} \cdot \left(\frac{1}{d_{obj}(q)} - \frac{1}{Q^*} \right)^2 & d_{obj}(q) \le Q^* \\ 0 & d_{obj}(q) > Q^* \end{cases}$$

where

◇ k_{rep} is again a scaling factor,

◇ d_{obj} is the minimal distance from q to the object and

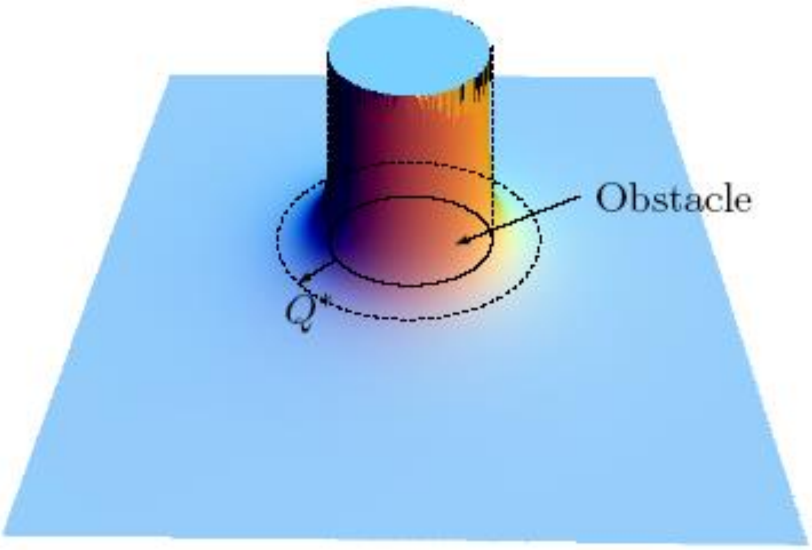
◇ Q^* is the distance of influence of the object.

Potential Field Method

- **Repulsive Potential**

$$U_{rep}(q)=\begin{cases}\frac{1}{2}k_{rep}\cdot\left(\frac{1}{d_{obj}(q)}-\frac{1}{Q^*}\right)^2 & d_{obj}(q)\le Q^*\\ 0 & d_{obj}(q)>Q^*\end{cases}$$

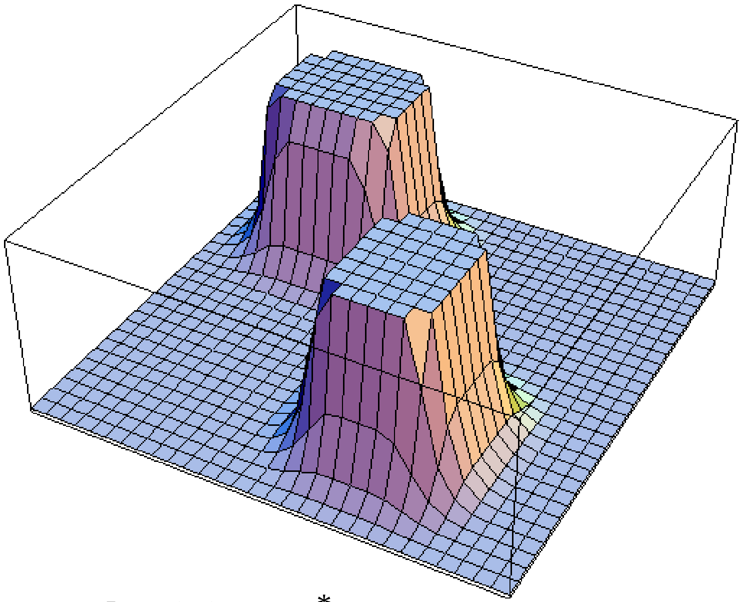
◇ The repulsive potential function is **positive** or **zero** and tends to **infinity** as gets closer to the object.



Potential Field Method

- **Repulsive Potential**

$$U_{rep}(q) = \begin{cases} \frac{1}{2} k_{rep} . \left(\frac{1}{d_{obj}(q)} - \frac{1}{Q^*} \right)^2 & d_{obj}(q) \le Q^* \\ 0 & d_{obj}(q) > Q^* \end{cases}$$



The **repulsive force** is

$$F_{rep}(q) = -\nabla U_{rep}(q) = \begin{cases} k_{rep} . \left(\frac{1}{d_{obj}(q)} - \frac{1}{Q^*} \right) . \frac{1}{d_{obj}^2} . \nabla d_{obj} & d_{obj}(q) \le Q^* \\ 0 & d_{obj}(q) > Q^* \end{cases}$$

where ∇d_{obj} denotes the partial derivate vector of the distance from the point subject to potential (PSP or q) to he obstacle or object.

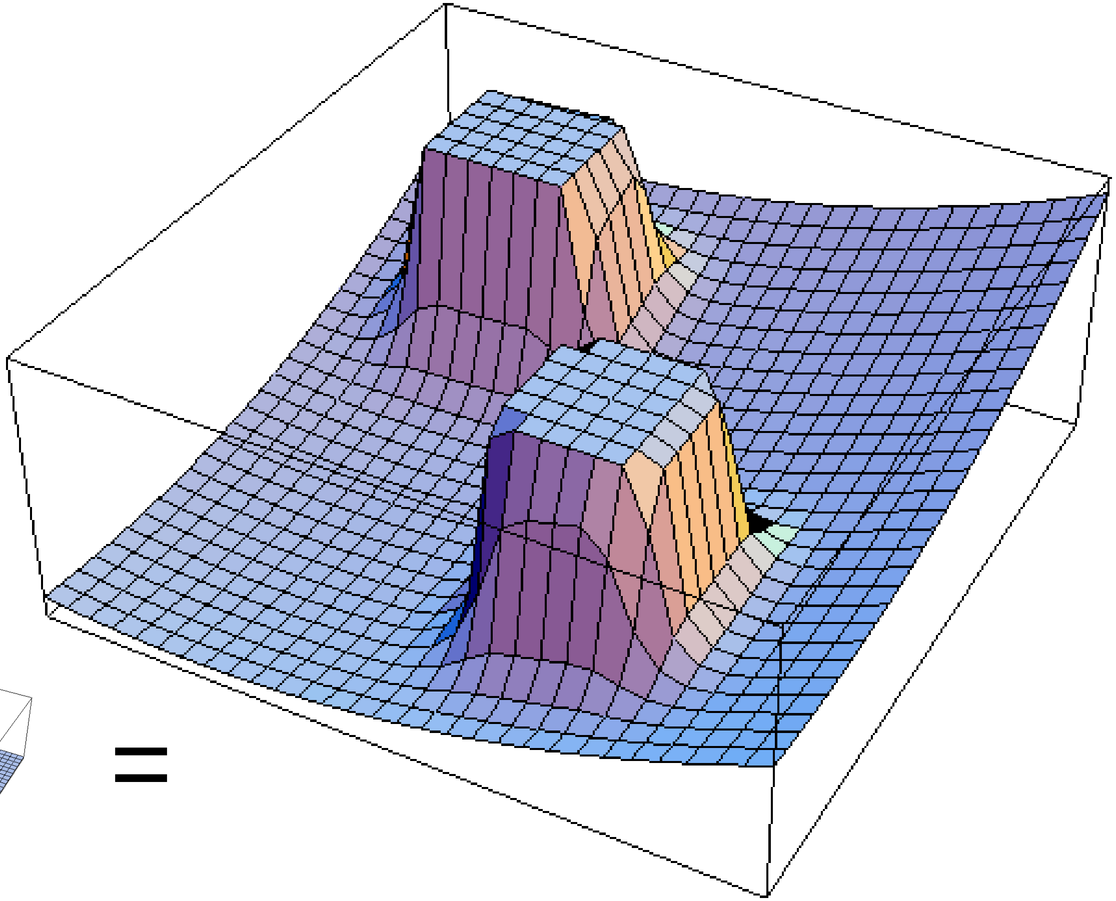
$$\nabla d_{obj} = \left(\frac{\partial d_{obj}}{\partial x}, \frac{\partial d_{obj}}{\partial y} \right)^T$$

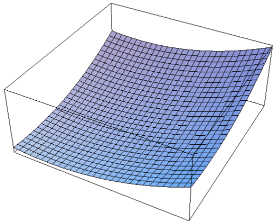
Potential Field Method

- **The resulting force**

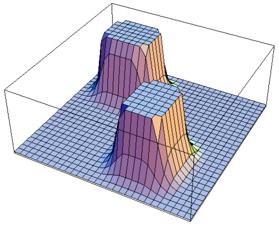
$$F(q) = F_{att}(q) + F_{rep}(q) = -\nabla U(q)$$

A first-order optimization algorithm such as **gradient descent** (also known as **steepest descent**) can be used to minimize this function by taking steps proportional to the negative of the gradient.





+



=

Potential Field Method

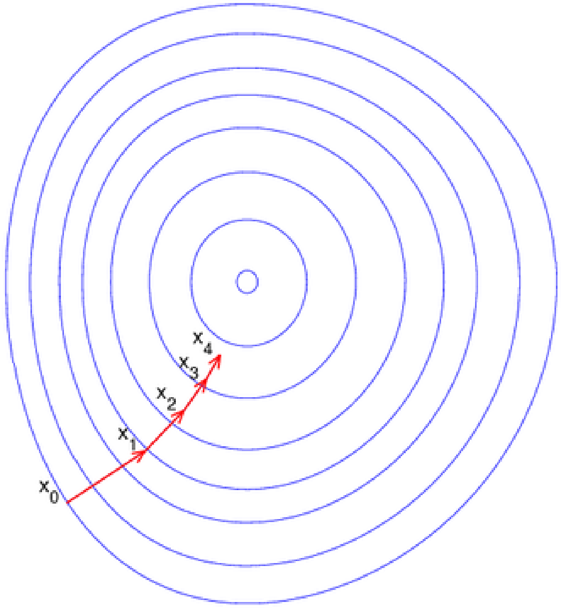
- **Gradient Descent or Steepest Descent**
  - ◇ Gradient descent is a first-order optimization algorithm.
  - ◇ To find a local minimum of a function using gradient descent, one takes **steps proportional to the negative of the gradient** (or of the approximate gradient) of the function at the current point.

GradientDescent($\boldsymbol{x}_{\mathbf{init}}$, $\boldsymbol{x}_{\mathbf{final}}$, - $\nabla \boldsymbol{f}$)

while $x_{\text{init}} \neq x_{\text{final}}$

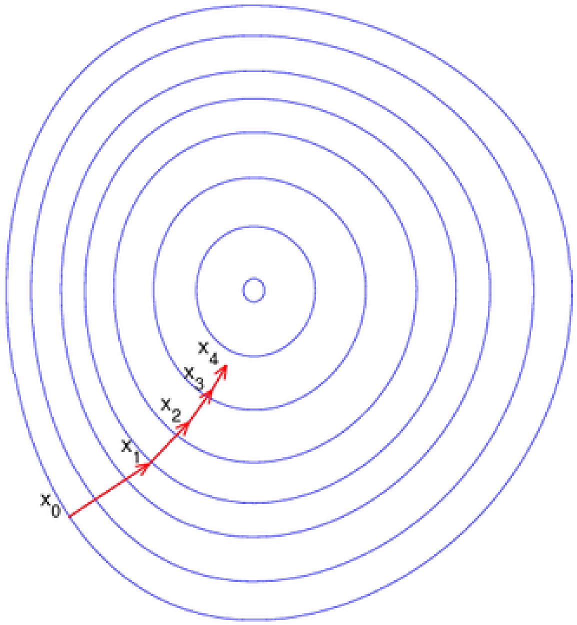
$$x_{n+1} = x_n - \gamma_n \nabla f(x_n), \quad n \geq 0$$

end



Potential Field Method

- **Gradient Descent or Steepest Descent**



GradientDescent($\boldsymbol{x_o}$, $\boldsymbol{x_{final}}$, - $\nabla \boldsymbol{f}$)

while $x_o \neq x_{final}$

$$x_{n+1} = x_n - \gamma_n \nabla f(x_n), \quad n \geq 0$$

end

where

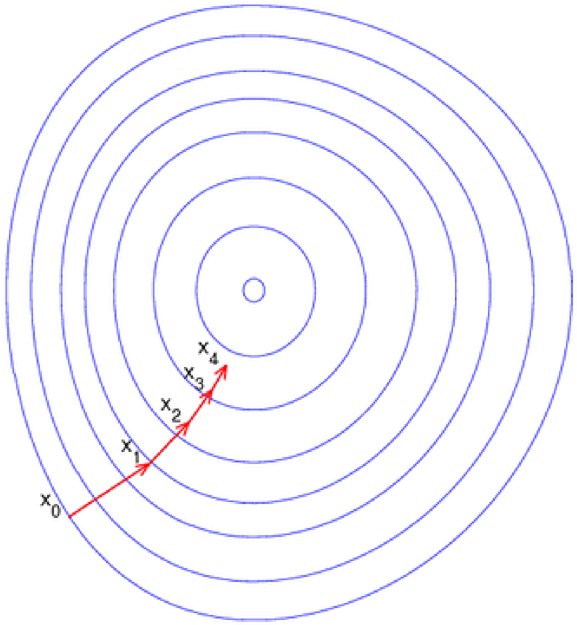
$\gamma > 0$ is a small enough number.

Note that the **step size** γ must be small enough to ensure that we do not collide with an obstacle or overshoot our goal position.

The value of the step size γ is allowed to change at every iteration.

Potential Field Method

- **Gradient Descent or Steepest Descent**



GradientDescent($\boldsymbol{x_o}$, $\boldsymbol{x_{final}}$, - ∇f)

while $x_o \neq x_{final}$

$$x_{n+1} = x_n - \gamma_n \nabla f(x_n), \quad n \geq 0$$

end

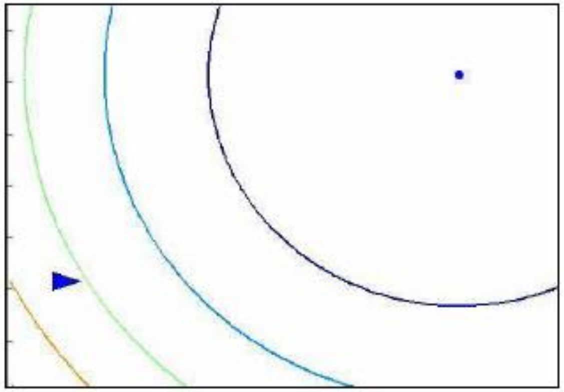
We have:

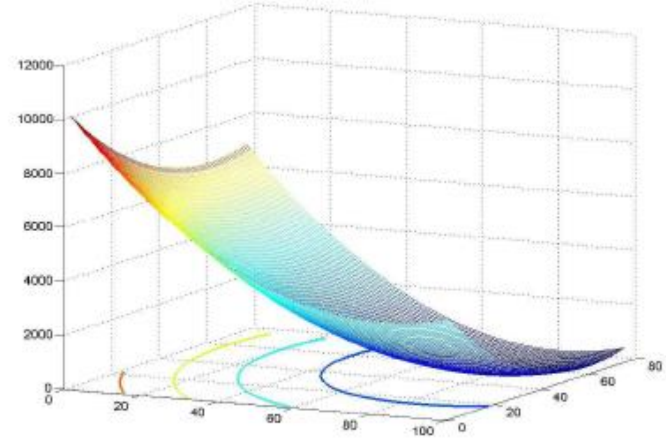
$$F(x_o) \geq F(x_1) \geq F(x_2) \geq \ldots F(x_{final})$$

so hopefully the sequence converges to the **desired local minimum $\boldsymbol{x_{final}}$**. Note that in practice, we will stop within some tolerance (like γ) of the final position to account for positional uncertainties, etc.

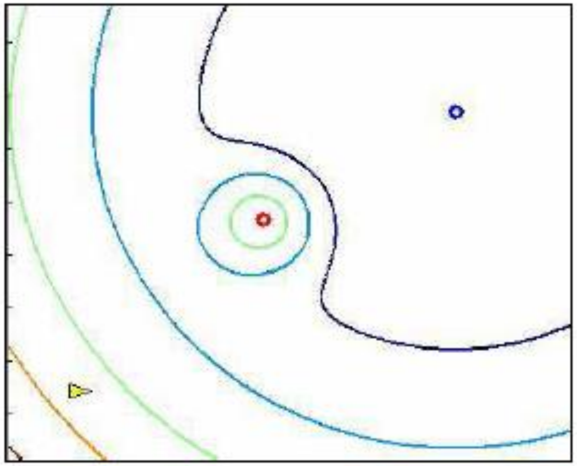
Potential Field Method

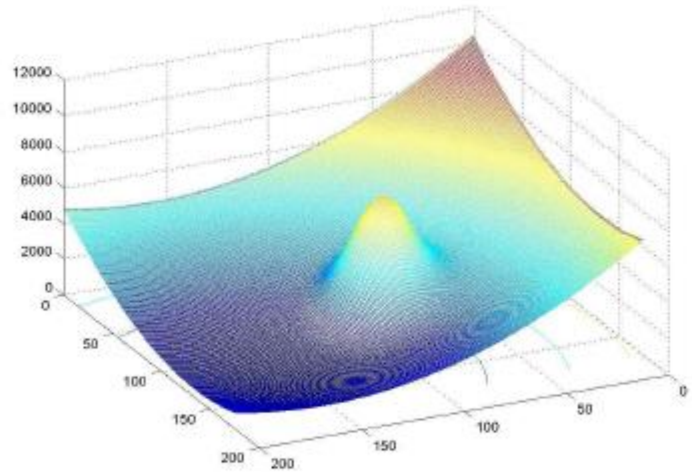
- **Exemples**





A Parabolic Well for Attracting to Goal

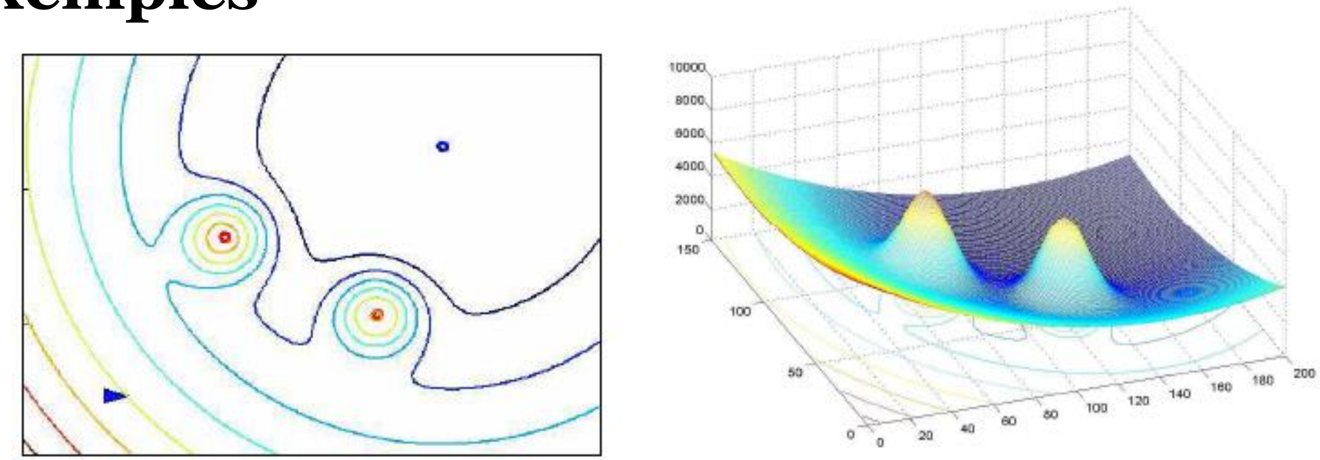




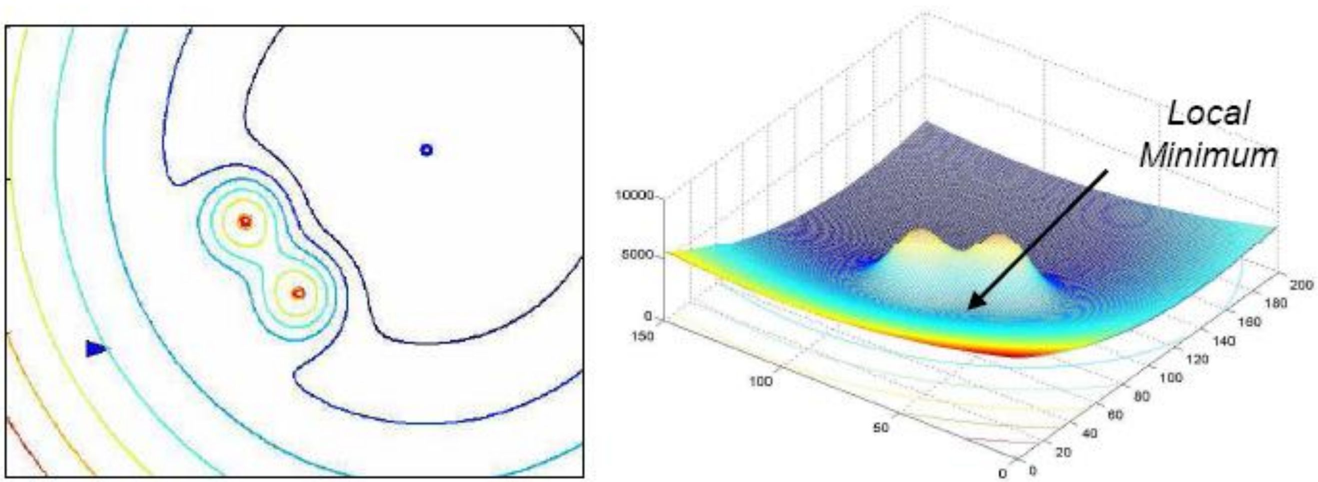
Parabolic Well Goal & Exponential Source for Obstacle

Motion Planning Solvers

- **Exemples**



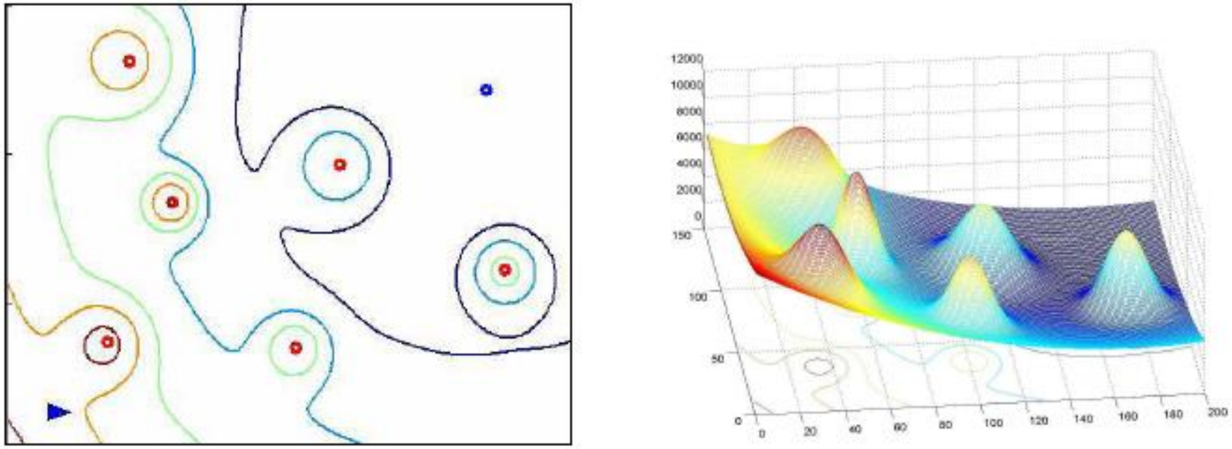
Parabolic Well Goal & Two Exponential Source Obstacles



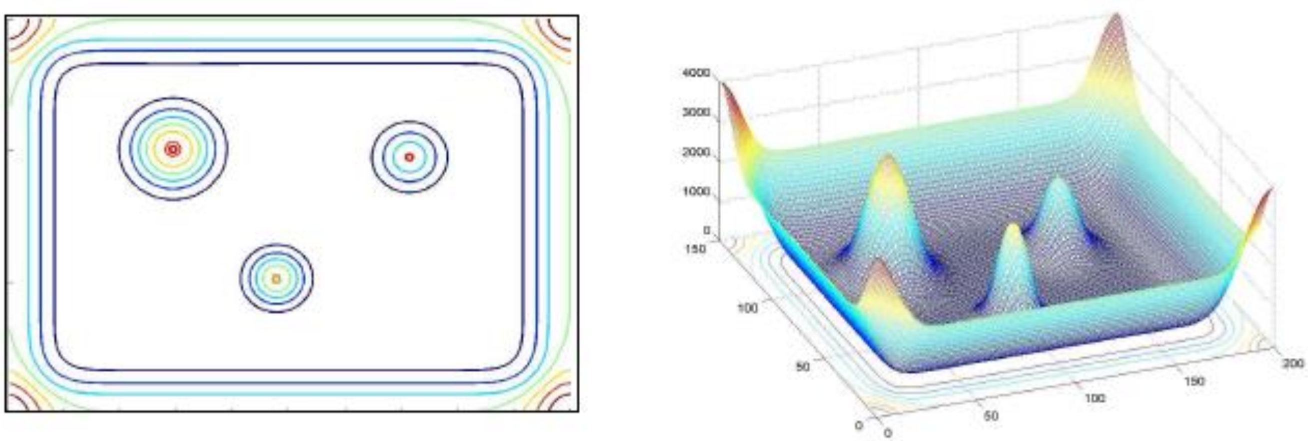
Parabolic Well Goal & Two Exponential Source Obstacles

Potential Field Method

- **Exemples**



Parabolic Well Goal & Multiple Exponential Source Obstacles



Modeling Walls in a Closed Workspace

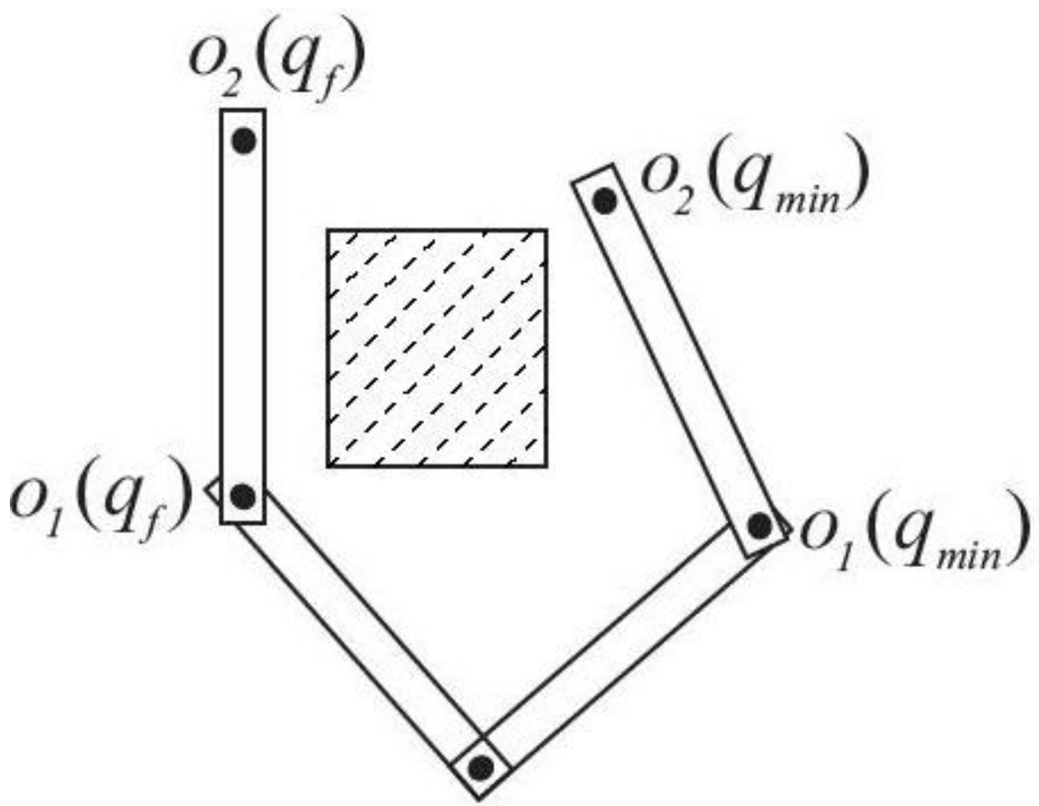
Potential Field Method

- **Problems of Potential Field Method**
  - ◇ **Trap situations due to local minima:** One major problem with this algorithm is preventing local minima. These are the points where the attractive and repulsive forces cancel each others. Local minima can exist by a variety of different obstacle configurations. There are several techniques to decrease or even avoid the local minima.

Potential Field Method

- **Problems of Potential Field Method**

  The configuration q_{min} is a local minimum in the potential field. At q_{min} the attractive force exactly cancels the repulsive fore and the planner fails to make further progress.

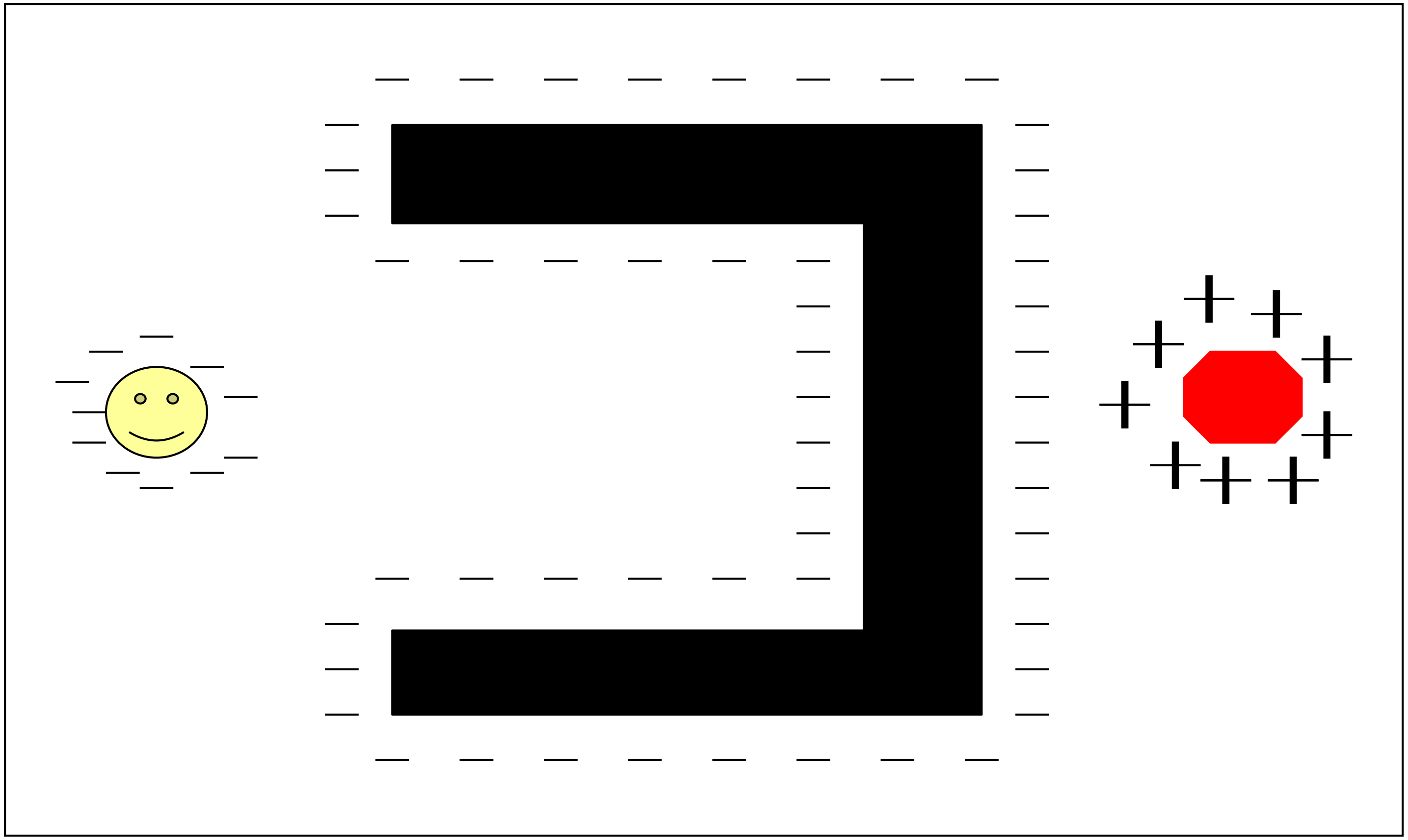


[4]

Potential Field Method

- **Problems of Potential Field Method**

Local Minimum Problem with the Charge Analogy



◇ The robot can get stuck in local minima.

◇ In this case the robot has reached a spot with zero force (or a level potential), where repelling and attracting forces **cancel each other** out.

◇ So the robot will stop and never reach the goal.

Potential Field Method

- **Problems of Potential Field Method**
  - **No passage between closely spaced obstacles:** There is no passage between closely spaced obstacles; as the **repulsive forces due to the first and the second obstacle add up to a force pointing away from the passage**. Thus the robot will either approach the passage further or it will turn away.

Potential Field Method

- **Problems of Potential Field Method**
  - ◇ **Oscillations in Narrow Passages:** The robot experience **repulsive forces simultaneously from opposite sides** when traveling in narrow corridors resulting in an unstable motion.
  - ◇ The scenario where the **goal is located near an obstacle** such that the repulsive force can be larger than the attractive force resulting in a **motion away from the goal** instead of reaching it.

References

1. LaValle, S. M. (2006). *Planning Algorithms*. Cambridge university press.
2. JJ Kuffner, SM LaValle, "RRT-connect: An efficient approach to single-query path planning", ICRA 2000.
3. Pieter Abbeel. Sampling-Based Motion Planning. EECS, UC Berkeley.
4. Spong, M. W., Hutchinson, S., & Vidyasagar, M. *Robot modeling and control*. John Wiley & Sons, 2006.