COL778: Principles of Autonomous Systems
Semester II, 2023-24

Planning Motions

Rohan Paul

Today’s lecture

- Last Class
  - Task Planning
- This Class
  - Motion Planning

How to describe a planning problem?

- What is planning?
  - Determining the sequence of actions that will attain a goal state.
- Planning Model
  - Planning problems require a model of the world.
  - Sometimes the term “model” means the parameters of T where the states and actions are fixed.

An example of a planning model.

- Set of states $s \in S$ that the world might be in.
- Set of actions $a \in A$ that we might take.
- Transition function $T : S \times A \mapsto S$

How the current state of the world will change as a result of taking an action.

Search applied to a planning model

- Note
  - There are only search algorithms, not planning algorithms.
- Various search algorithms
  - no cost function,
  - have a cost function but no heuristic,
  - cost function and heuristics etc.

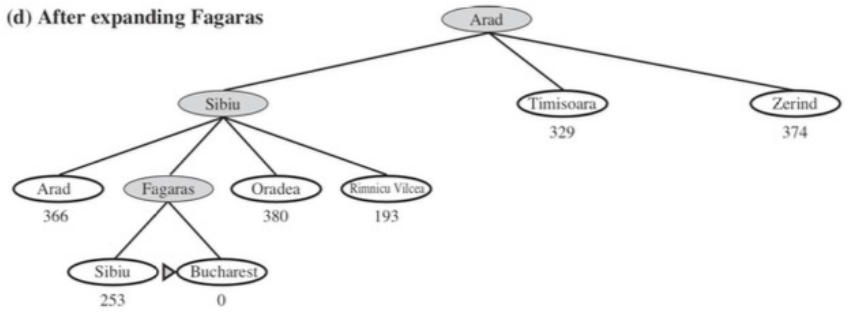


Figure: The A* Tree from Russell & Norvig.

Notions associated with planners

- Planning problem
  - Tuple: start state s_0, goal state s^g and a planning model M.

$$\{s_0, s^g, M\}$$

- Satisficing (feasibility)
  - Any plan that gets from the start state to the goal state.
- Optimality
  - Comparing multiple plans from the start to the goal.
  - Notion of costs C or rewards R (associated with taking an action a in a state s and reaching state s')
  - Optimal plans minimize the total cost or maximize the total reward.

If a plan τ is specified as a sequence of states and actions $\{s_0^\tau, a_0^\tau, \ldots, a_T^\tau, s_{T+1}^\tau\}$, then the best plan might be

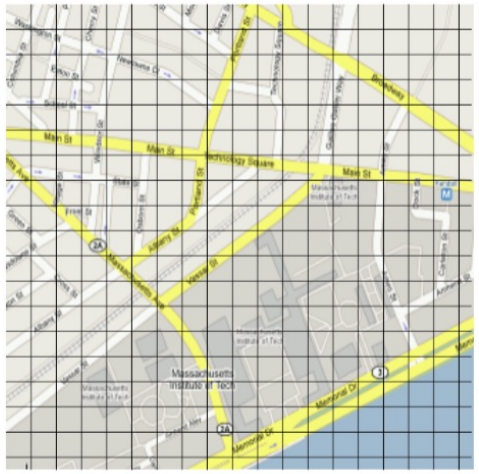
$$\tau^* = \underset{\tau}{\operatorname{argmax}} \sum_t R(s_t^\tau, a_t^\tau, s_{t+1}^\tau)$$

$$\text{such that } s_0^\tau = s_0 \text{ and } s_{T+1}^\tau = s^g$$

- Completeness
  - If an optimal plan exists (connecting the start to the goal state) then the search algorithm will find it.

Planning Model

- Constructing an appropriate model is often the most important part of planning.
- Example
  - Different discretization can lead to different models with different state and action spaces, even though the underlying problem hasn't changed.



Grid representation

Topological representation

Plan Execution

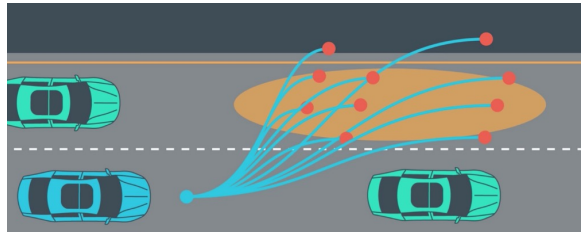
- Once we have a plan, it is to be executed.
- There could still be failures in plan execution due to uncertainty or unexpected events.
  - Aleatoric uncertainty
    - The uncertainty in action outcome because the real world is stochastic.
  - Epistemic uncertainty
    - The uncertainty because there is something that we do not know (the topology changes, new things added in the map).

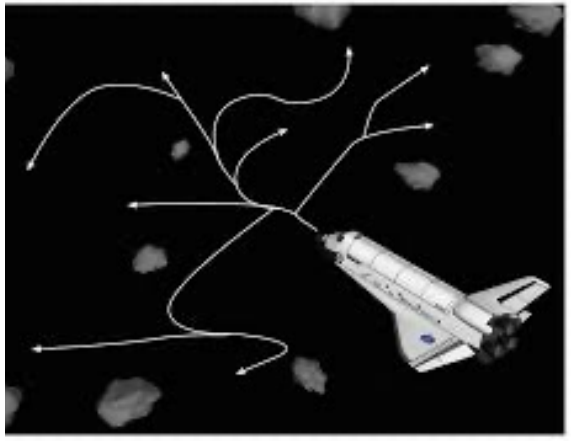
Maps can change over time.

Motion Planning

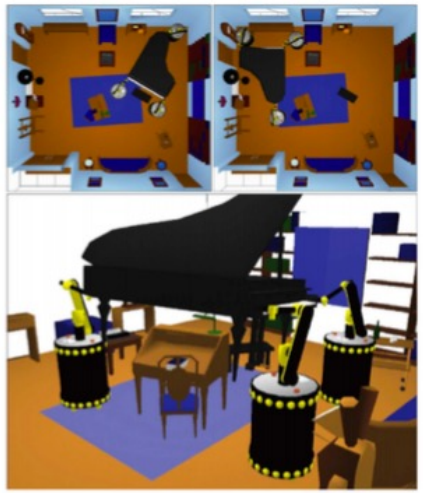
- Motion planning is a class of planning problems.
- Setup
  - Feasible and infeasible regions of space.
    - Obstacles and free spaces.
  - Given a start and and a goal state.
  - Here, we consider continuous spaces.
- Task
  - Determine a sequence of actions (control inputs) that leads from the start to the goal state.
- Constraints
  - Agent cannot enter infeasible regions of space.
  - The agent may not be able to move to any coordinate at will.
    - Example: car, helicopter, airplane, but also robot manipulator hitting joint limits

Motion planning for vehicles.
Constraints on the type of motions for the vehicle.

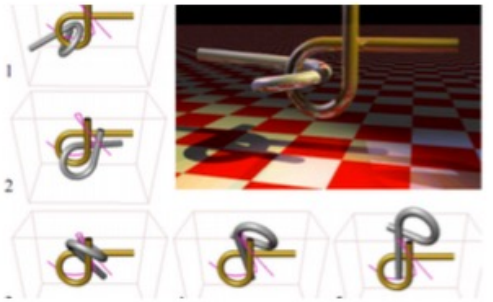




More general problems: Re-arrangement planning and the pulling bars apart benchmark (J. Kuffner)



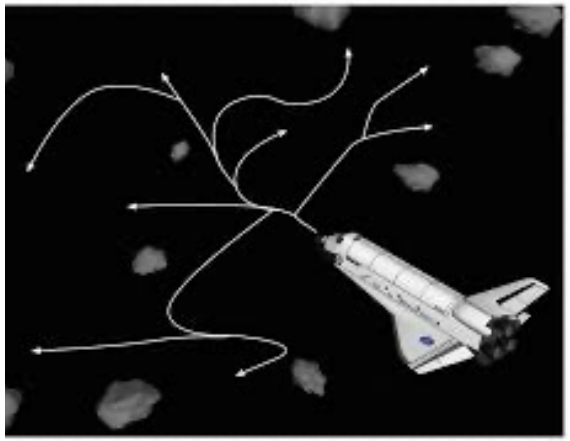
How to change the configuration of the piano in the room without colliding.

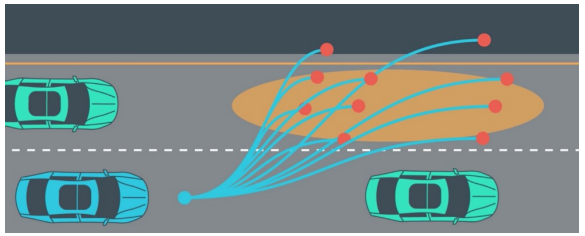


How to separate out the bars.

Formulating the Problem: State Space

- Which space to plan in?
  - The state space is usually a grid.
  - Discretization of the continuous space.
    - The world is continuous, a grid is an *approximation*.
- There can be a variety of grid types.
- The state space representation is crucial to performance and accuracy.

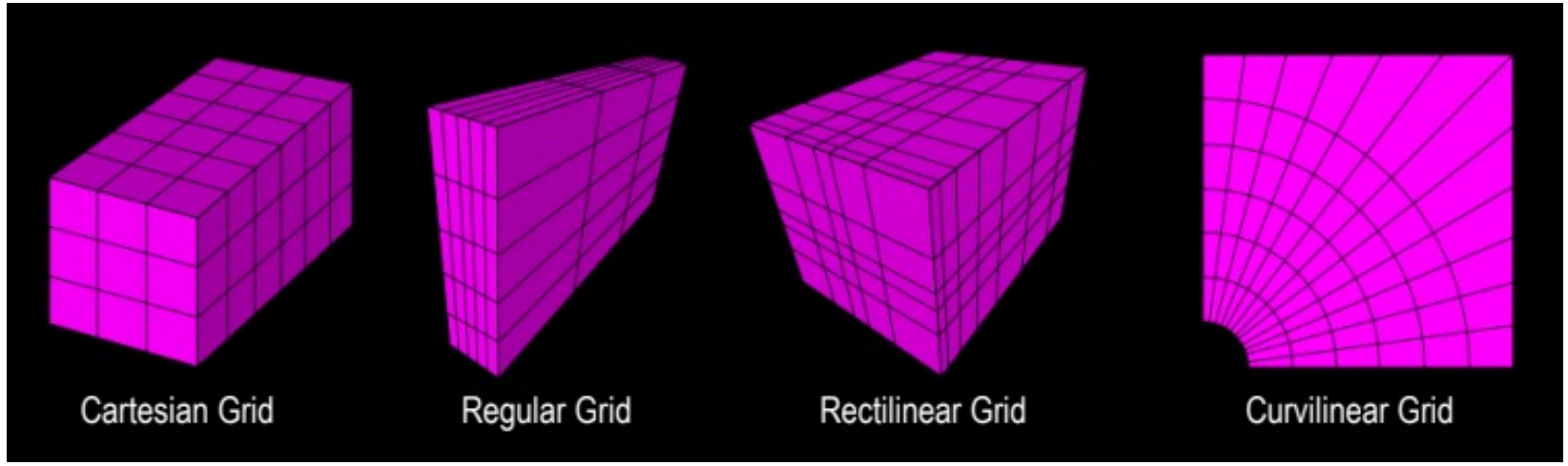




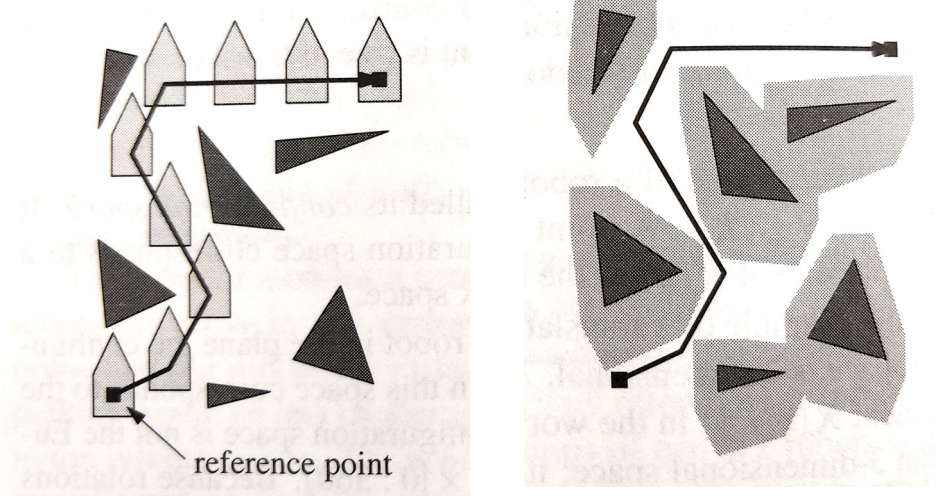
Variety of State Spaces

Types of grids

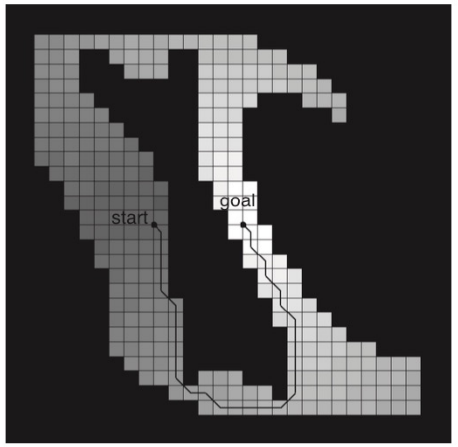
https://en.wikipedia.org/wiki/Regular_grid



Planar robot task space

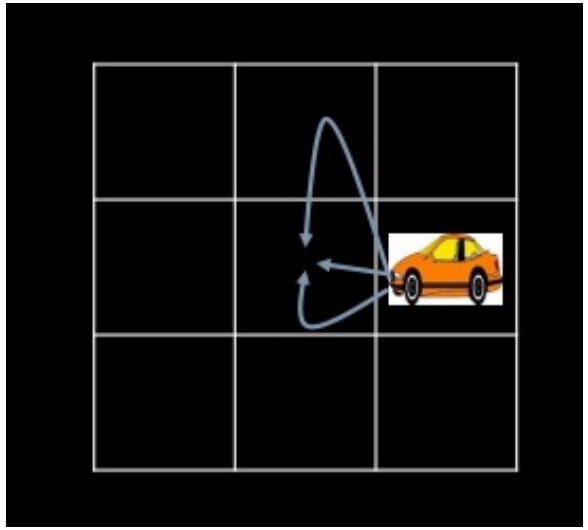


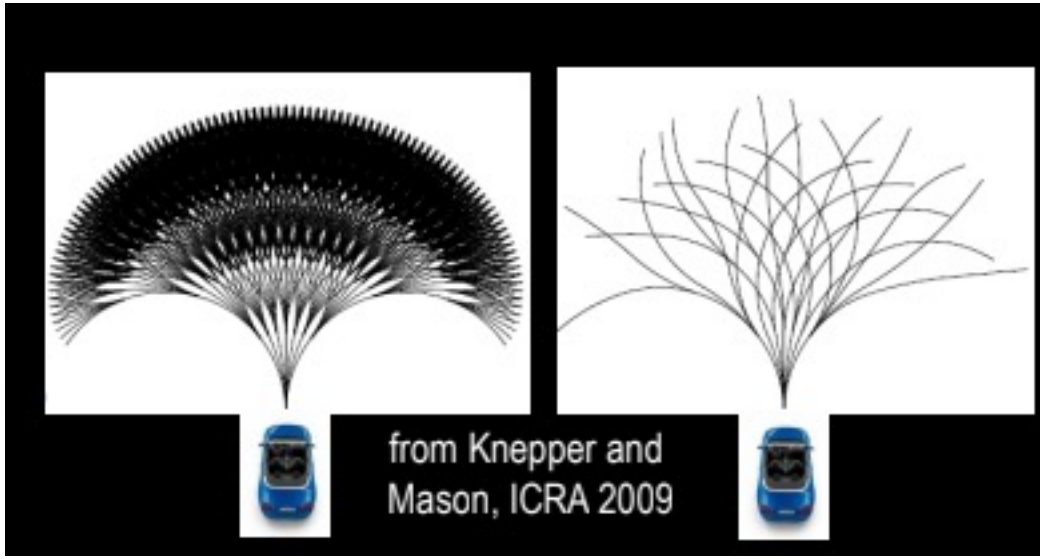
Manipulator configuration space.



Formulating the Problem: Action Space

- Actions in motion planning are also often continuous
- For example, many ways to move between neighboring cells
- Usually pick a discrete action set a priori.

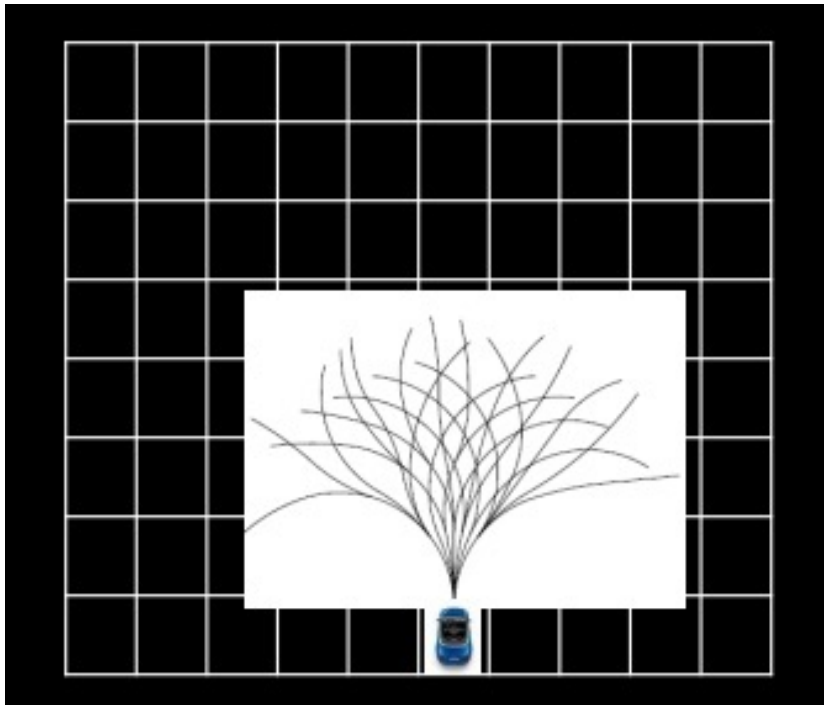




Formulating the Problem: Successors and Action Cost

- Successors
  - Determined by the action set.
  - Successors may not be known a-priori.
    - May need to try out actions to see which cell you land up in.

- Action cost
  - What do we value in a path. What are you trying to optimize
    - Cost: distance traversed when executing an action.
    - Could be others for smoothness or any other properties we care about.



Linear combination of cost functions (most common):

$$\text{Cost} = a_1C_1 + a_2C_2 + a_3C_3 \ldots.$$

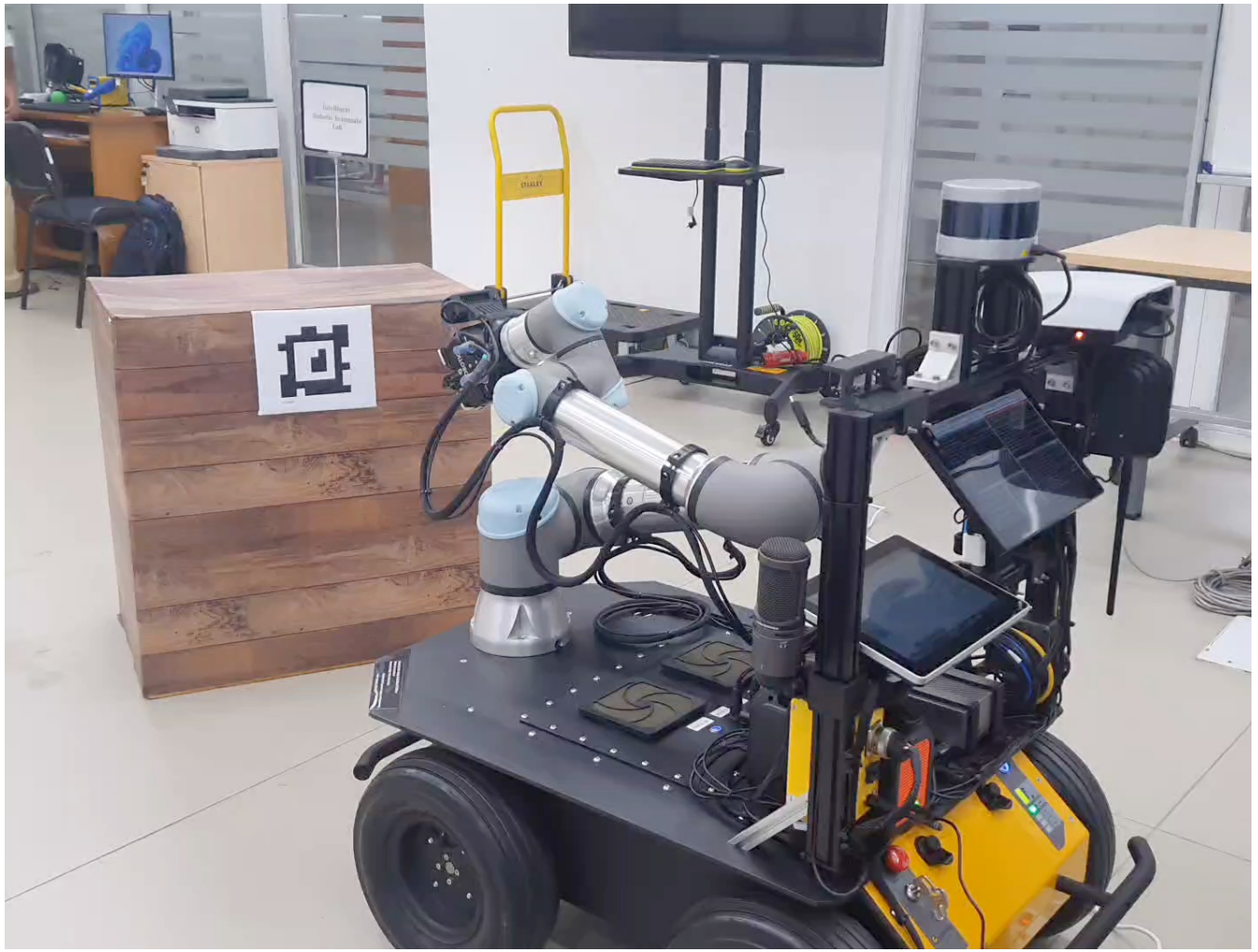
Formulating the Problem: Goal Test

- Goals are most commonly specific cells you want to get to.
- Can also be “semantic”
  - A state where some other object is visible.
  - A state where the robot is in contact with another object.
  - State(s) where the robot is near a landmark.

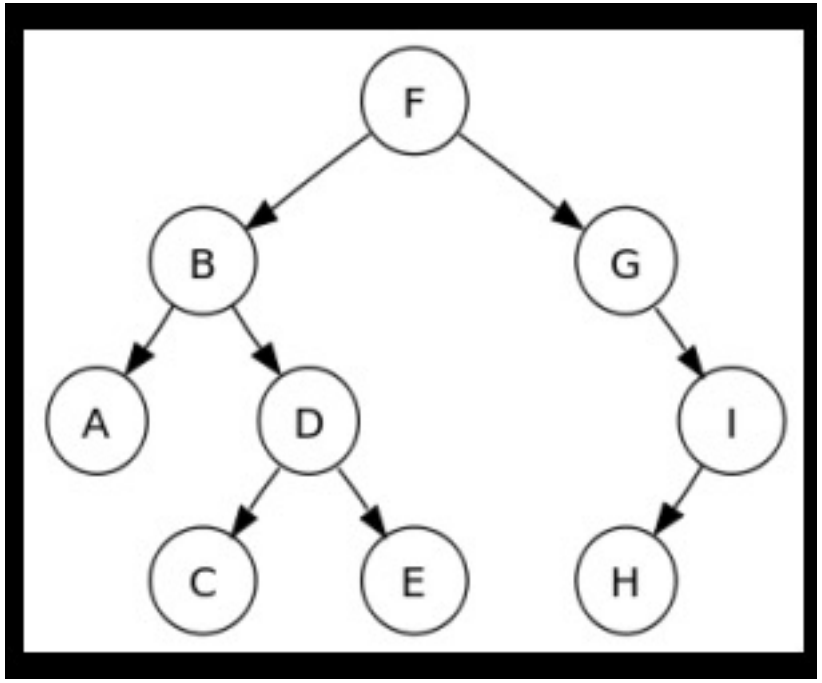
Example: target pose for robot pushing

Goal is to make contact with the object (for pushing the object out of the way).

There are multiple poses that will make contact with the object.



How to plan? Grow a tree of possibilities



Essentially, we grow a tree using successors and see if we can reach the goal state.

Monitoring Plan Execution

- Re-planning
  - A form of execution monitoring.
- Action monitoring
  - Is the *current action optimal/feasible* from the current state?
- Plan monitoring
  - Is the *current plan* still *optimal or feasible*?
- Goal monitoring
  - Is the *current goal* still *feasible/desirable*?

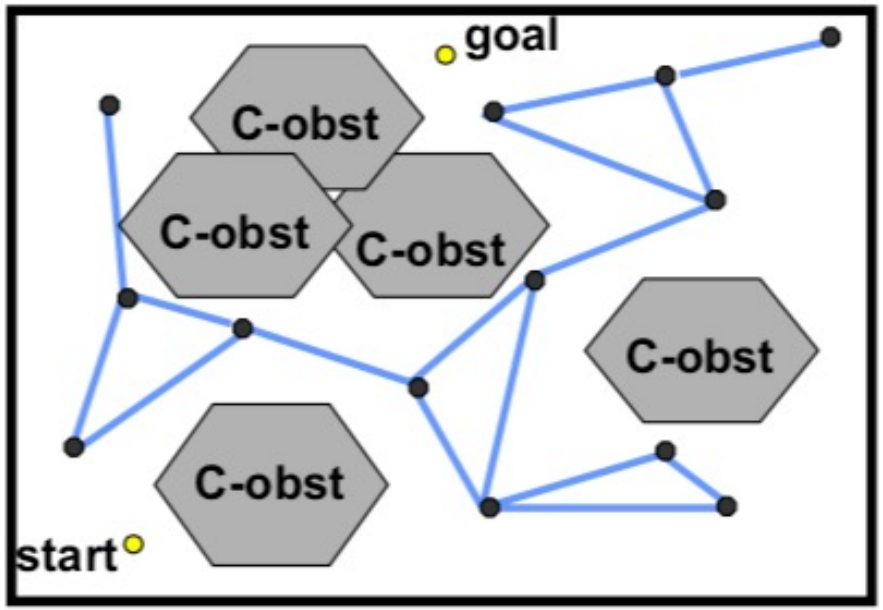
Sample-based Motion Planning

- **Monte Carlo methods**
  - Tackling continuous spaces (or when the space is too large), cannot exhaustively explore all possibilities.
  - Randomly explore a smaller subset of possibilities while keeping track of progress.
- **Tradeoff**
  - Between solution quality and runtime performance.
  - More sampling leads to a better solution.
- **Sample-based Planning**
  - Search for a path (collision-free) only by sampling paths.
  - Probabilistic Roadmaps (PRMs)
  - Rapidly exploring random trees (RRTs)

RRT sampling-based motion planner.

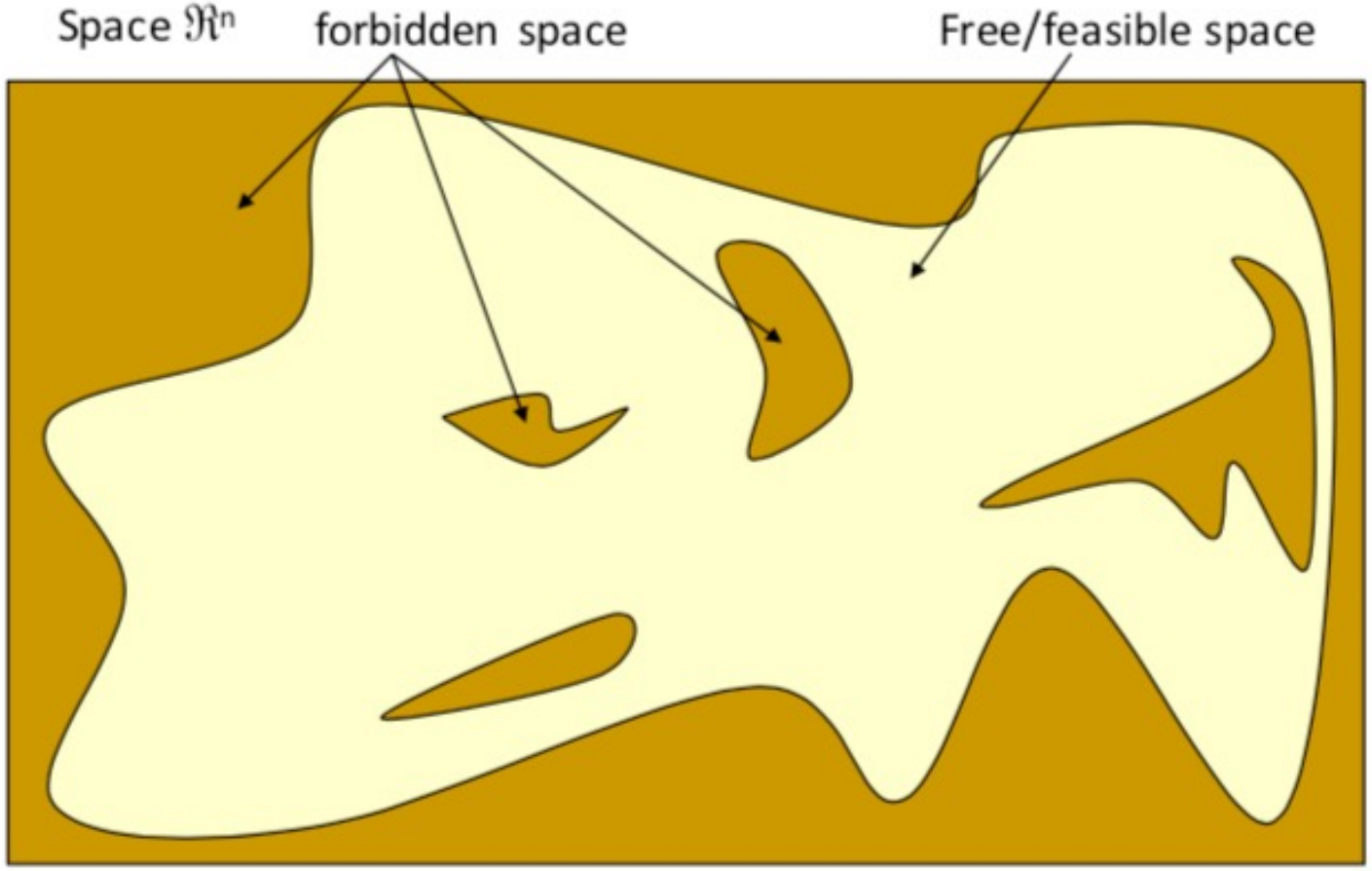
Probabilistic Roadmaps

- **Central Idea**
  - Sample and find collision-free configurations
  - Connect the configurations (create a graph)
  - Search the graph for solution
- **Phases**
  - Learning phase
  - Query phase. Inherently, a multi-query planner.



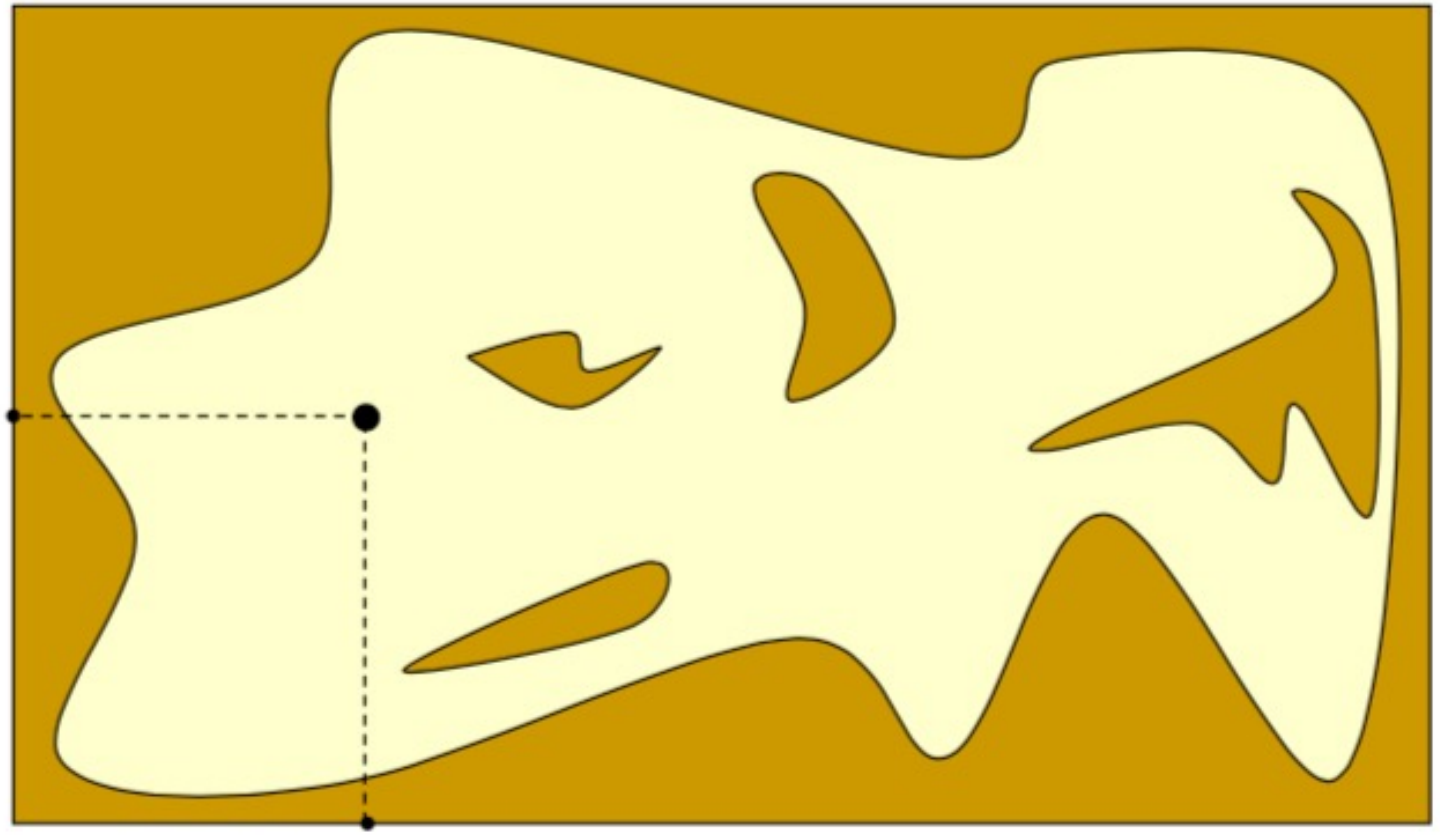
Probabilistic Roadmap (PRM)

- Configuration space
  - Forbidden and free space



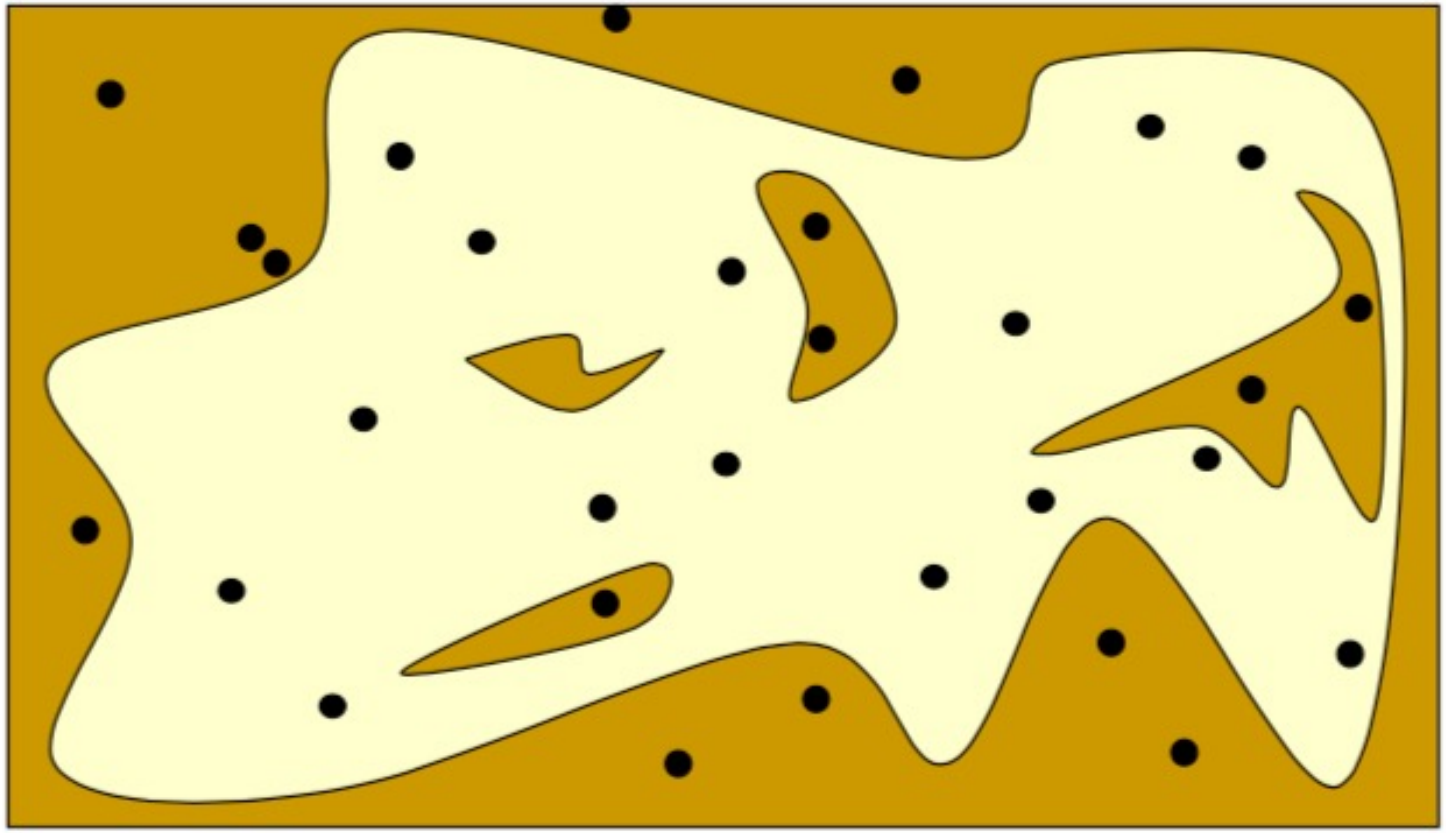
Probabilistic Roadmap (PRM)

- Configurations are sampled by picking coordinates at random.



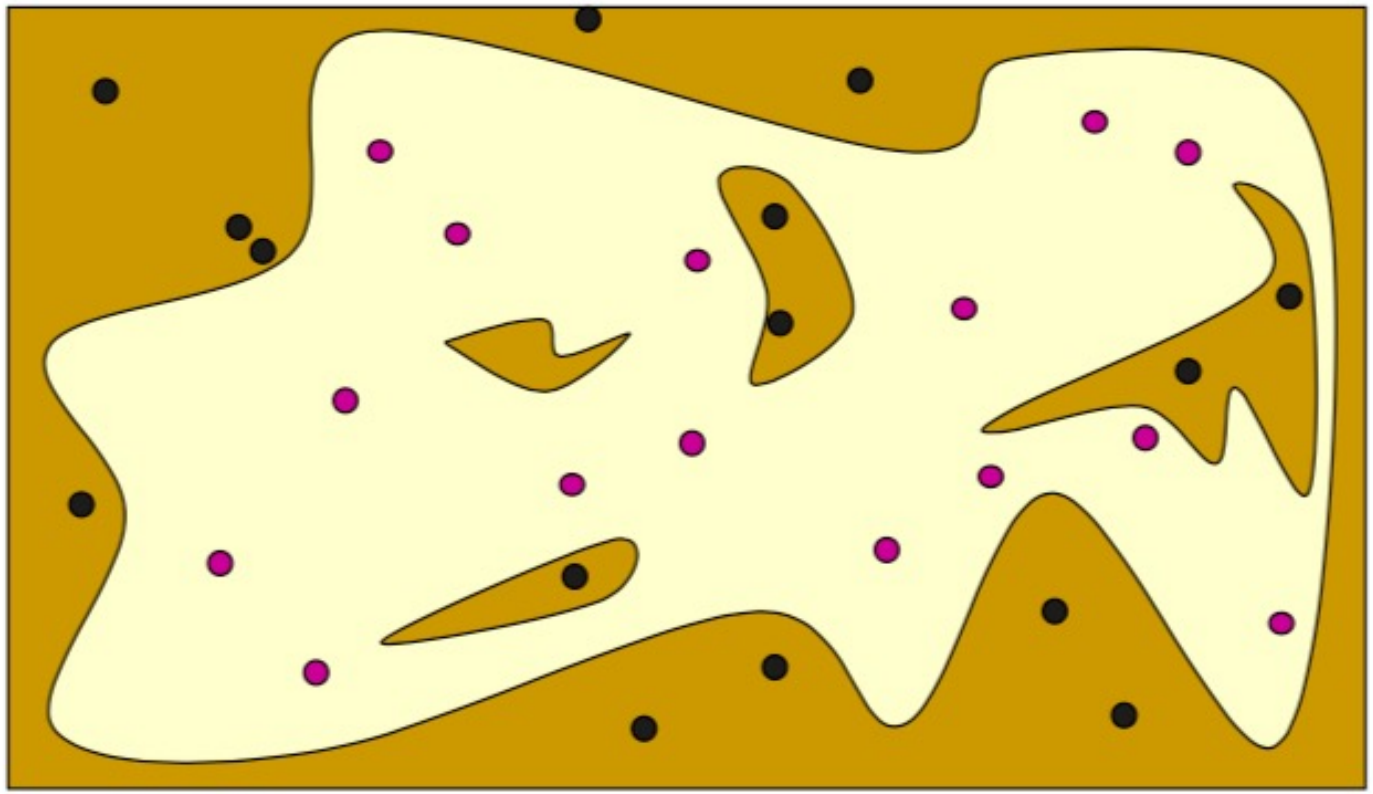
Probabilistic Roadmap (PRM)

- Configurations are sampled by picking coordinates at random.



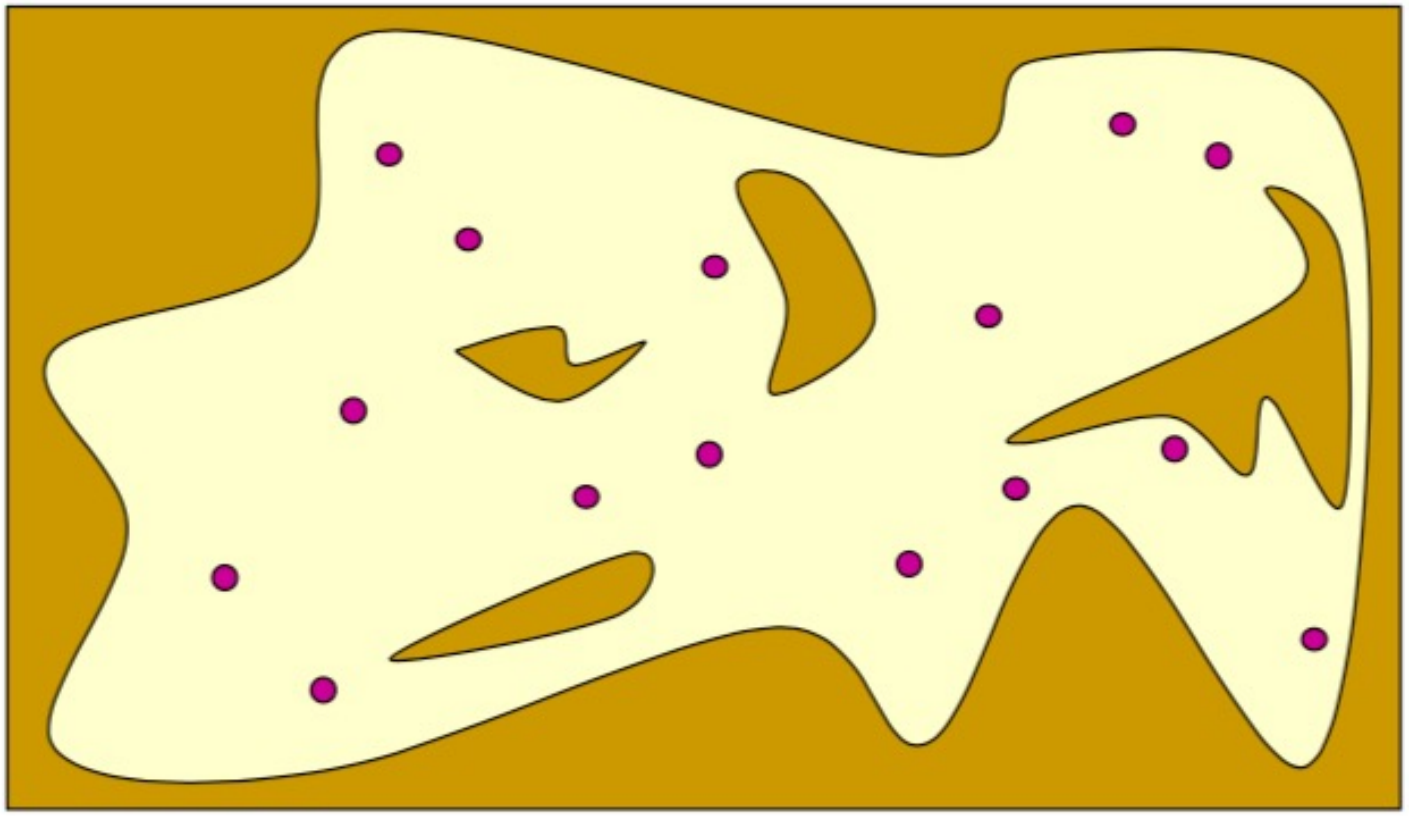
Probabilistic Roadmap (PRM)

- Sampled configurations are tested for collisions.



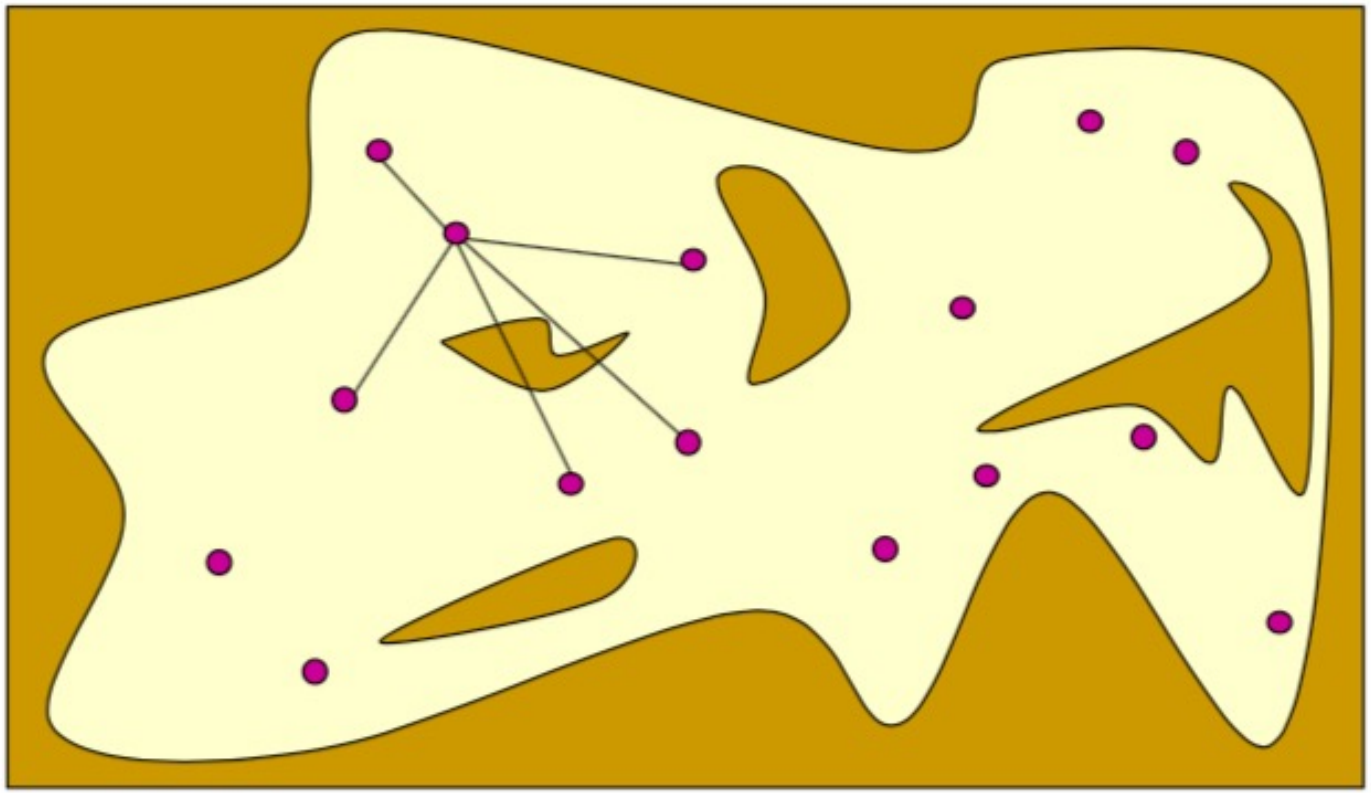
Probabilistic Roadmap (PRM)

- Feasible configurations (collision free) are retained as “milestones”.



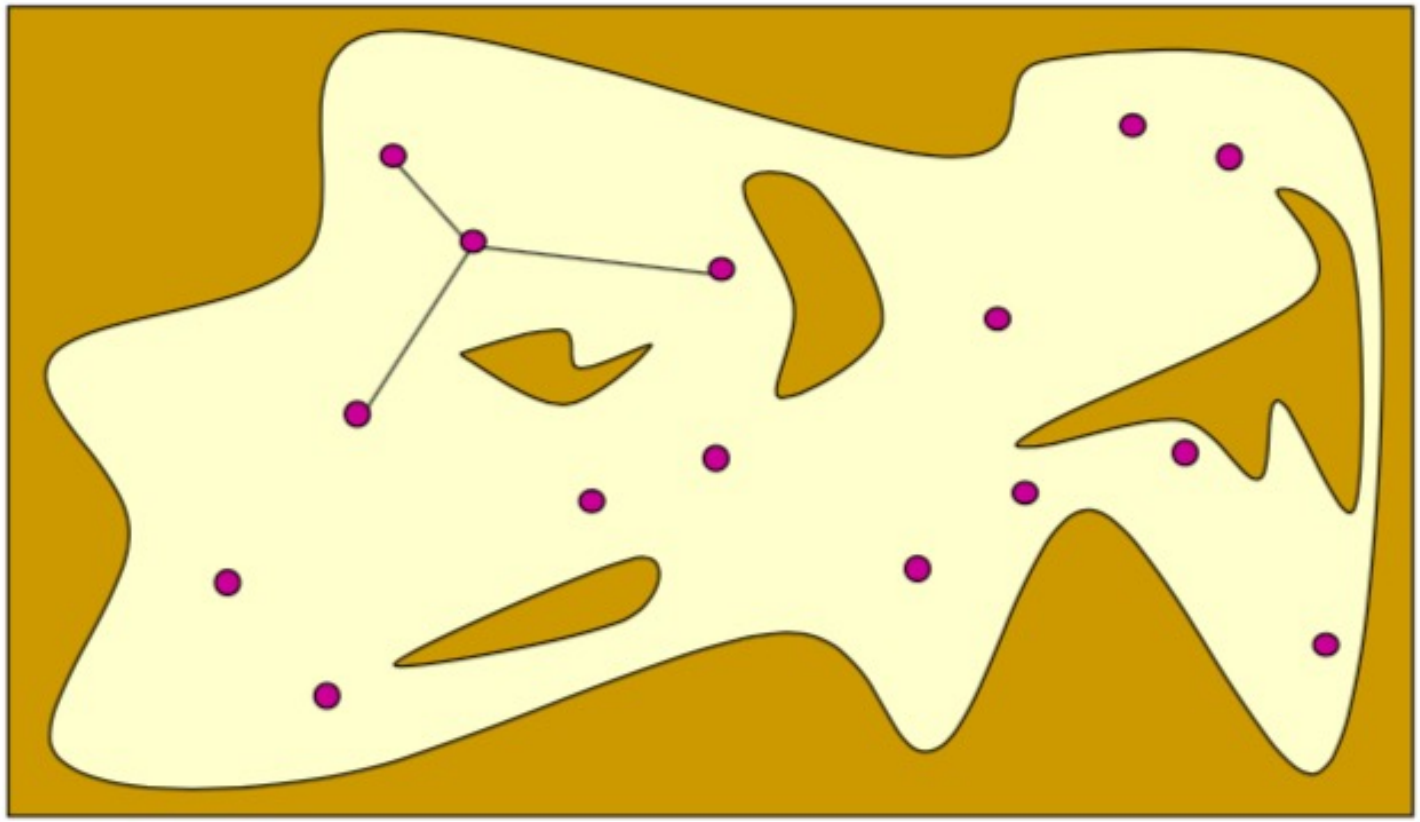
Probabilistic Roadmap (PRM)

- Connect each milestone with its nearest neighbours.



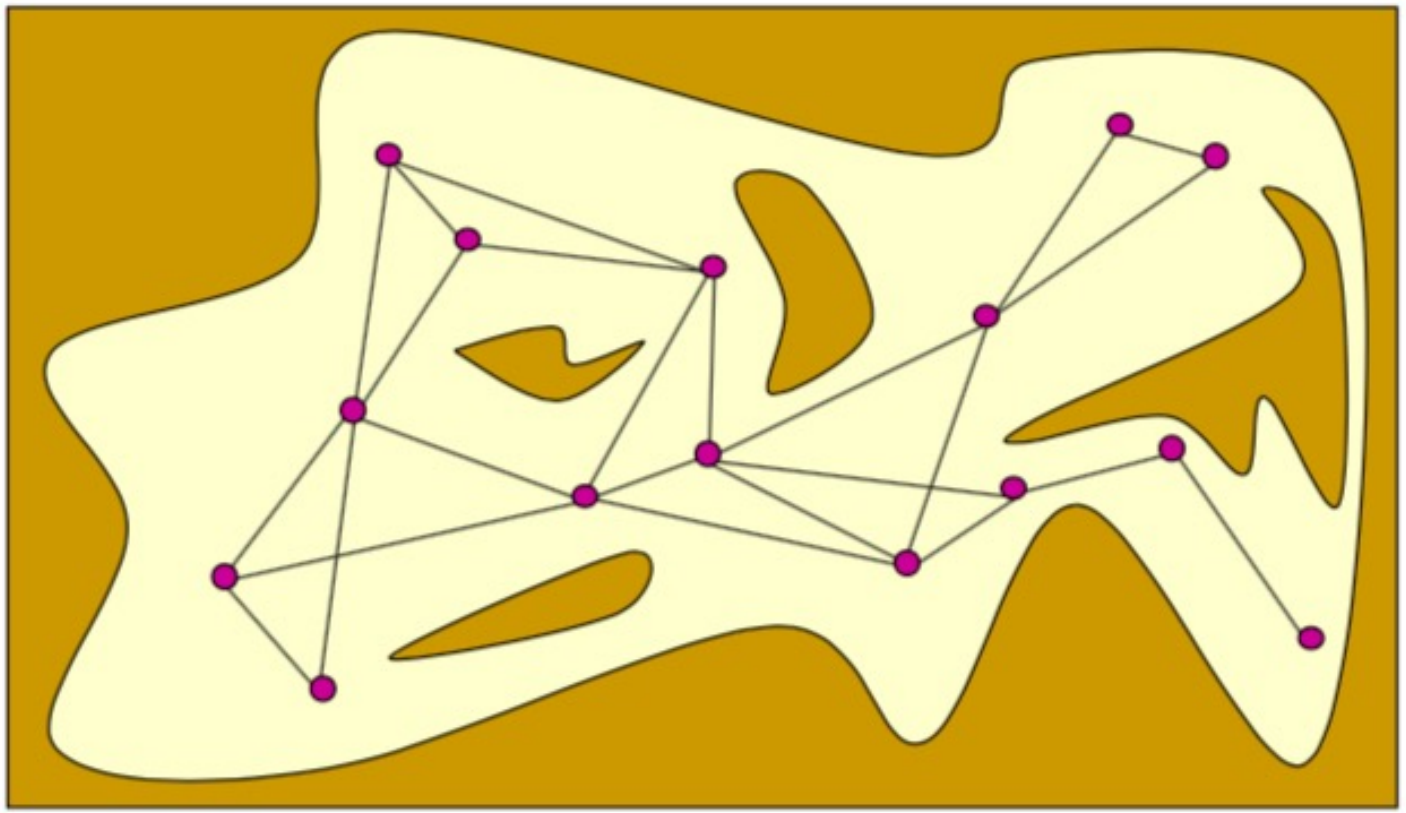
Probabilistic Roadmap (PRM)

- Each milestone is linked by straight paths to its nearest neighbours.
- Retain only feasible connections, reject others.



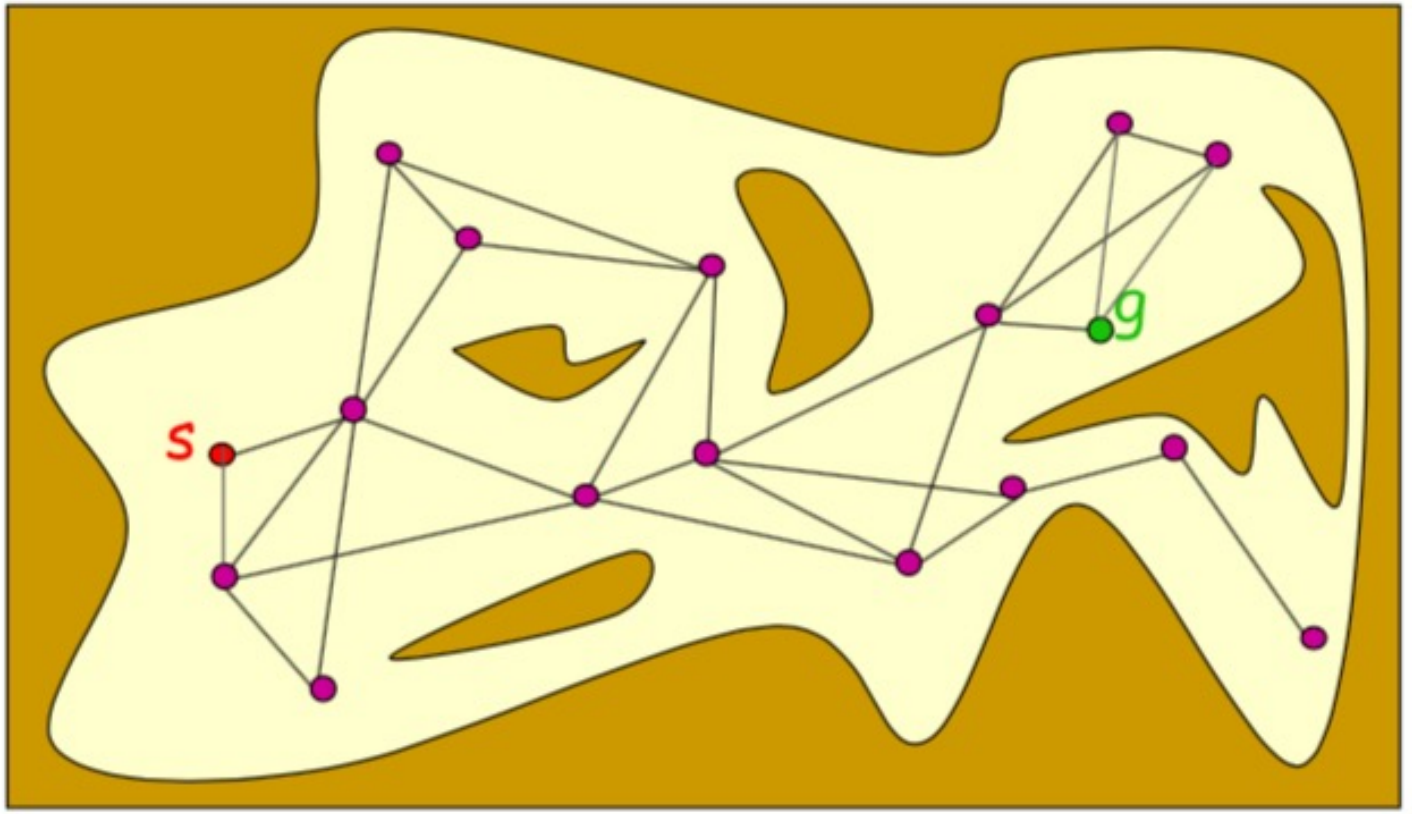
Probabilistic Roadmap (PRM)

- The collision-free links are retained as local paths. This is the output of the learning phase.
- The graph structure is the roadmap. The sampling is probabilistic, hence called PRM.



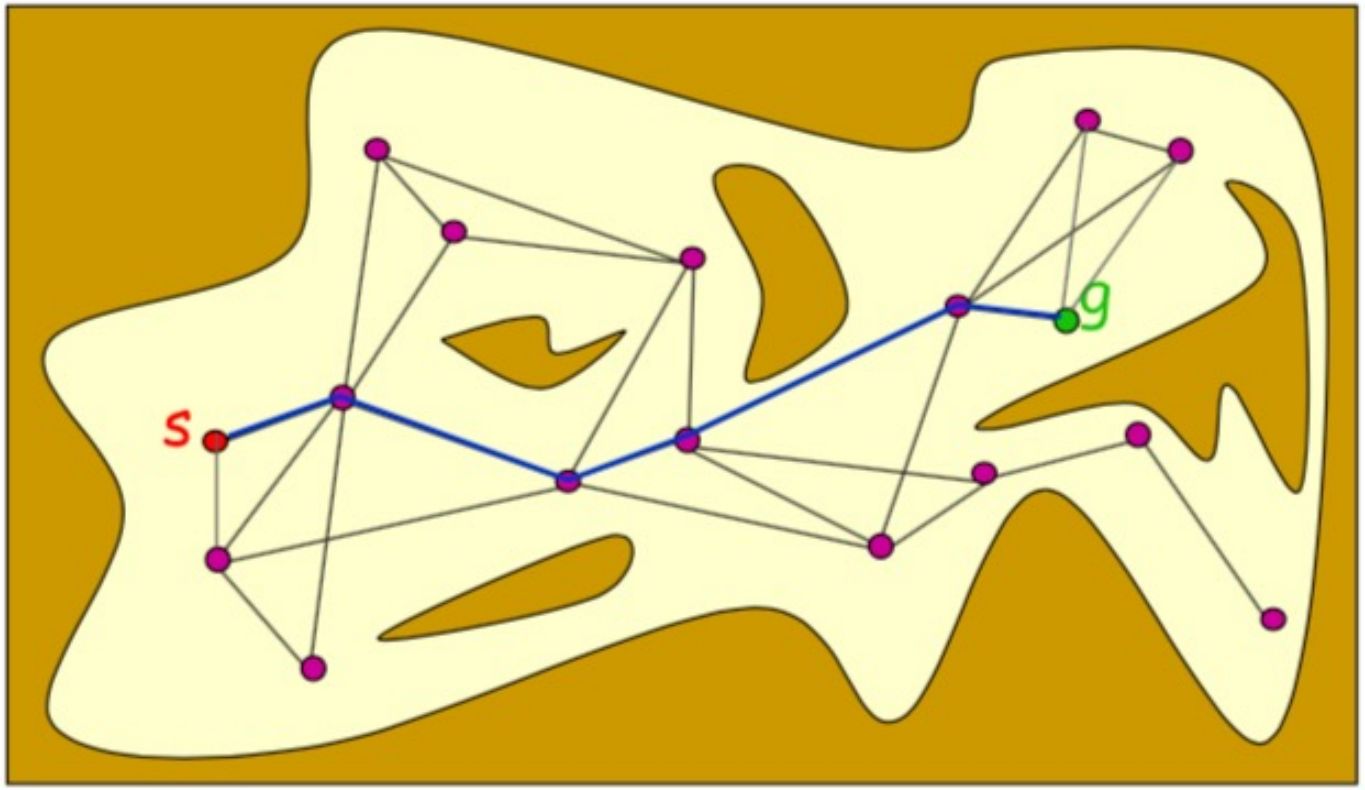
Probabilistic Roadmap: Query Phase

- Query phase.
- The start and the goal configurations are included as milestones.

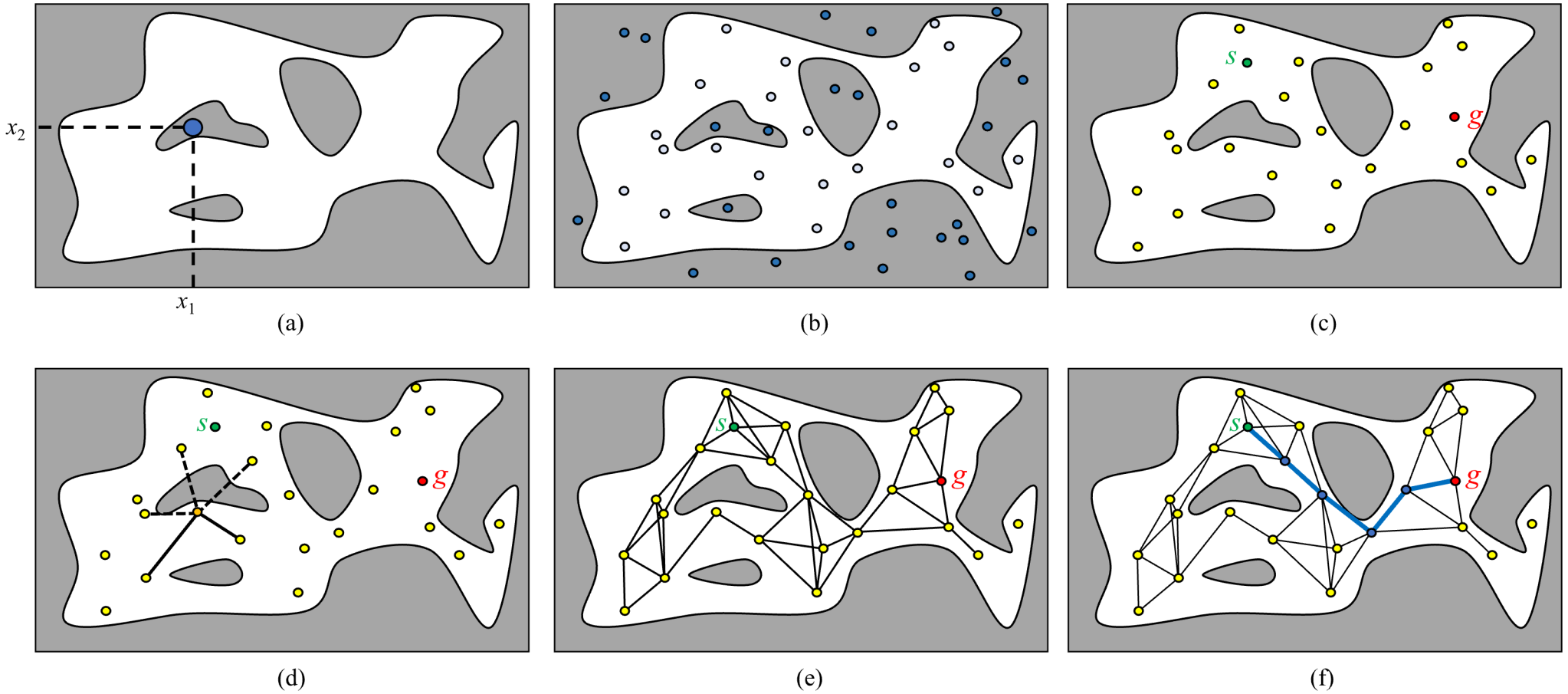


Probabilistic Roadmap: Query Phase

- The graph is searched for a path from s to g.



Probabilistic Roadmaps



Probabilistic Roadmaps

Algorithm Basic-PRM(s,g,N)

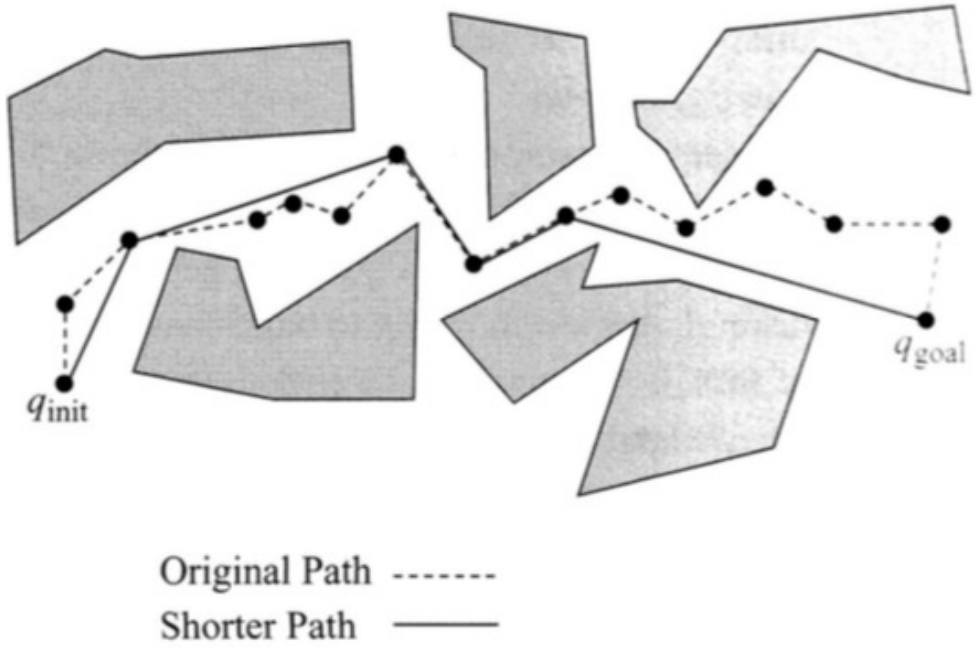
1. $V \leftarrow \{s, g\}$.
2. $E \leftarrow \{\}$.
3. **for** $i = 1, \ldots, N$ **do**
4. $\quad q \leftarrow Sample()$
5. $\quad$ **if** not Feasible(q) **then** return to Line 3.
6. $\quad$ Add q to V (add q as a new milestone)
7. $\quad$ **for** all $p \in near(q, V)$
8. $\quad\quad$ **if** Visible(p, q) **then**
9. $\quad\quad\quad$ Add (p, q) to E.
10. Search $G = (V, E)$, with Cartesian distance as the edge cost, to connect s and g.
11. **return** the path if one is found.

Random Sampling

- Need to sample nodes uniformly from Q_free (the free space)
- One approach
  - Randomly sample its coordinate from the interval of values for the corresponding degrees of freedom.
  - Use uniform sampling over the interval.
  - Check for collision both with the robot itself and with the obstacles.
  - If collision free then add, else discard the sample.

Path Smoothing

- Once a path is found, it can be optimized
- Try connecting non-adjacent configurations. Choose q_1 and q_2 randomly, try to connect.
- Greedy approach: try connecting points q_0, q_1, …q_n to q_goal.
- If q_k connects to q_goal, do the above with q_k as q_goal



Example: 6 DOF Path Planning

- Robot: rigid non-convex object in 3D space.
- Obstacle – solid wall with a small opening.
- State space is position and orientation.
- PRMs can be used for this problem
  - Edge collision checking can be by interpolating the two poses between start and end configuration.

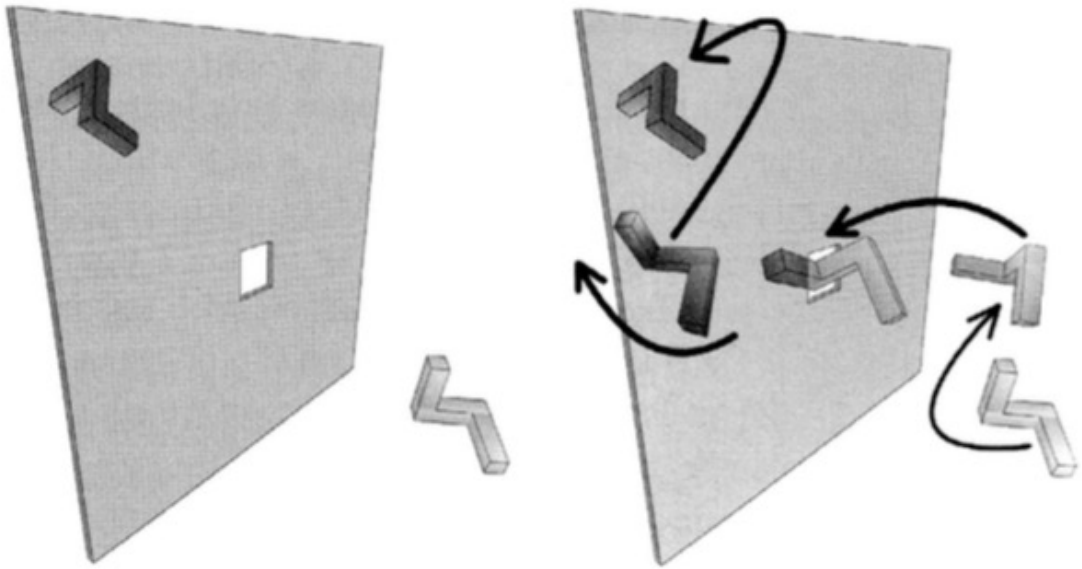
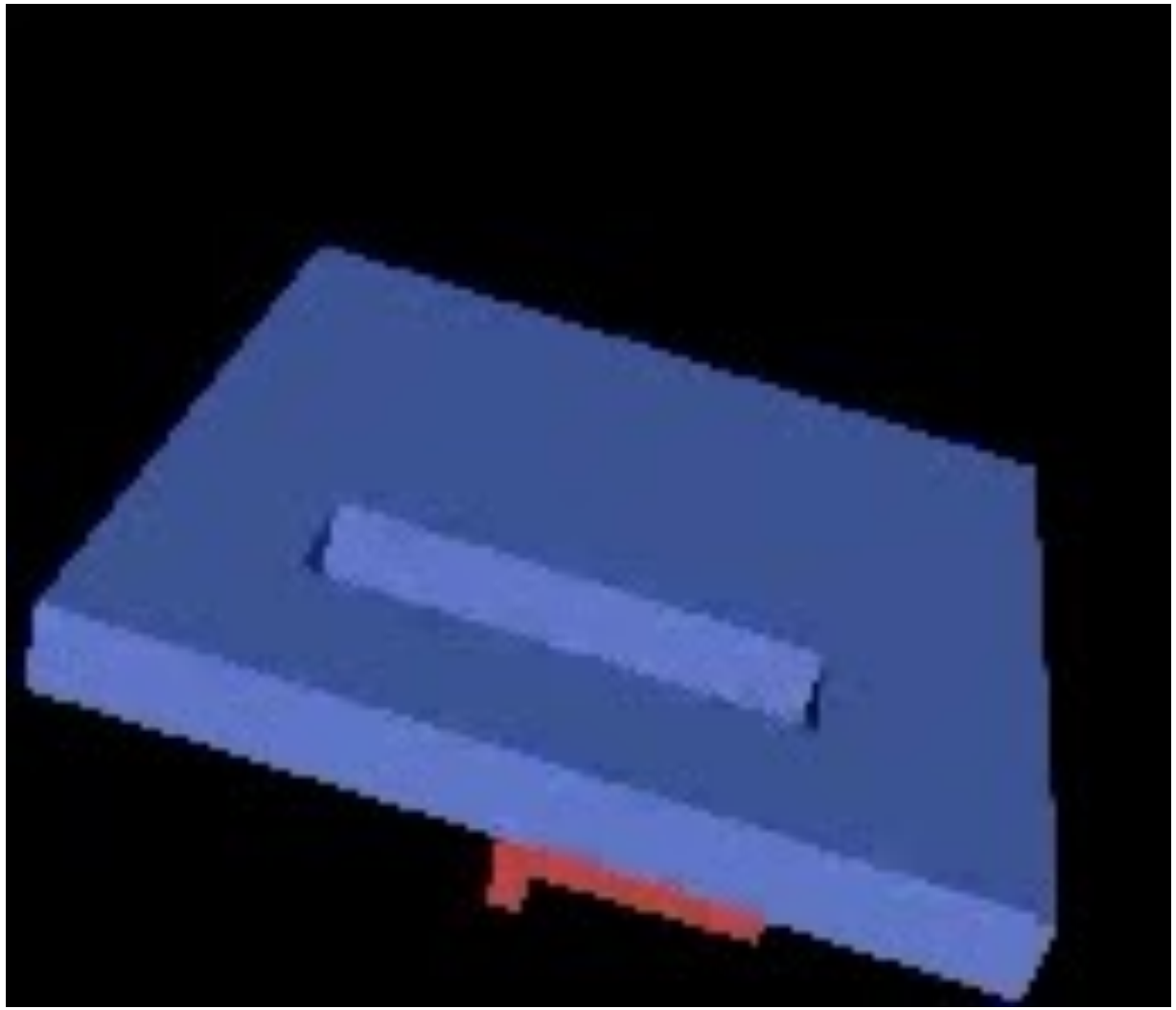


Figure 7.8 An example of a motion-planning problem where both the robot and the obstacles are a collection of polyhedral objects in three dimensions. Parts of the robot on the other side of the wall are indicated by the darker color. (a) The initial and goal configuration of the query. (b) A path produced from a PRM with $n = 1000$ and $k = 10$.

Example: 6 DOF Path Planning



Distance calculations in 3D

- Distance function needed between 2 configurations q, q'
- Ideally, distance is the swept volume of the robot as it moves between configurations q and q'. Difficult to compute exactly
- Method I: Can approximate this distance with an embedding in a Euclidean metric space: **d(q,q') = || embed(q) - embed(q')||**
  - Choose set of p points on robot, concatenate them, and create a vector of size p x dimension of workspace.
  - Example of rigid object in 3D: Create vector of size 3p, choosing p points on the object. Intuitively, a "sampling" of the object's Euclidean domain.
  - For configuration q, **embed(q)** is the vector of p points transformed by the translation and rotation that is configuration q. Transform each of the p points into the vector **embed(q).**
  - Do the same for configuration q', create **embed(q').**
  - Distance is now Euclidean distance between the 3p vectors:

    d(q,q') = || embed(q) - embed(q')||
- How do you choose the p points?
- Cheaper solution: choose 2 points p1 and p2 of maximum extent on the object.
- Method II: Separate a configuration q into a translation **X** and a rotation **R**: **q=(X,R)**
- Calculate a weighted distance function **d(q,q') = w1||X-X'|| + w2 f(R,R').**
- Need to use a metric on rotations – quaternions are good for this
- Weights **w1** and **w2** need to be chosen, no real insight into this

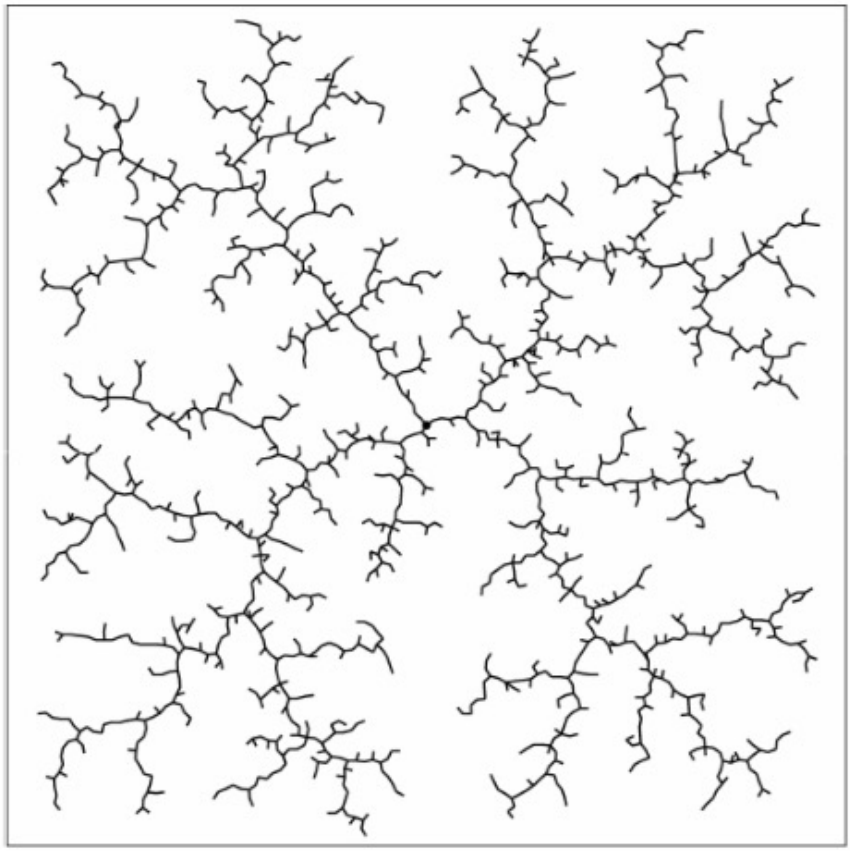
Probabilistic Roadmap

- **Key Steps**
  - Initialize set of points with X_S and X_G
  - Randomly sample points in the configuration space
  - Connect nearby points if they can be reached from each other
  - Find a path from X_S to X_G in the graph.

- **Probabilistically completeness**
  - If the algorithm is run infinitely often then by probability one it will contain a solution path if one exists.

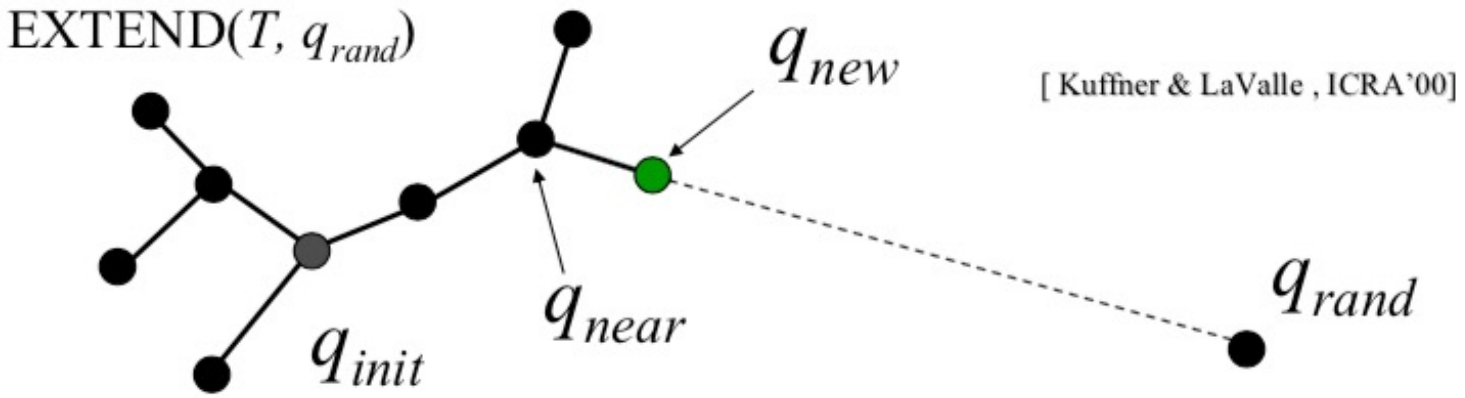
Rapidly Exploring Random Tree (RRT)

- **Central Idea**
  - Build up a tree in the search space through generating “next states”.
  - Select a random point and expand the nearest vertex in the tree towards the sampled point.



RRT Extension

- Select a random point
- Expand the nearest vertex in the tree towards the sampled point by adding an edge.



Rapidly Exploring Random Tree (RRT)

GENERATE_RRT($x_{init}, K, \Delta t$)

1 $\mathcal{T}$.init(x_{init});
2 **for** $k = 1$ **to** K **do**
3 $x_{rand} \leftarrow$ RANDOM_STATE();
4 $x_{near} \leftarrow$ NEAREST_NEIGHBOR($x_{rand}, \mathcal{T}$);
5 $u \leftarrow$ SELECT_INPUT(x_{rand}, x_{near});
6 $x_{new} \leftarrow$ NEW_STATE($x_{near}, u, \Delta t$);
7 $\mathcal{T}$.add_vertex(x_{new});
8 $\mathcal{T}$.add_edge(x_{near}, x_{new}, u);
9 Return $\mathcal{T}$

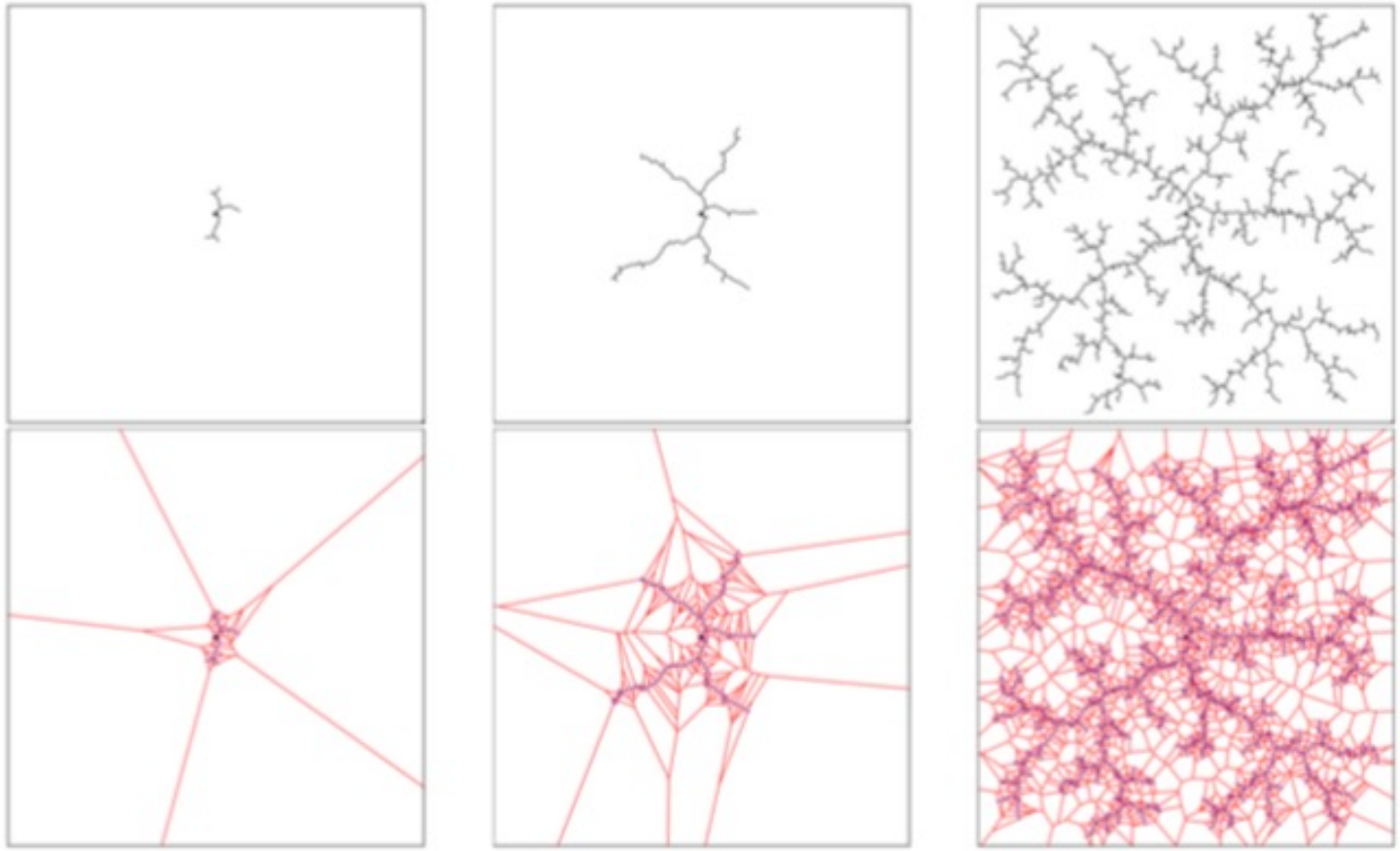
Growing an RRT

A visualization of an RRT graph after 45 and 390 iterations

An animation of an RRT starting from iteration 0 to 10000

https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree

Rapidly Exploring Random Tree (RRT)



http://msl.cs.uiuc.edu/rrt/gallery.html

Understanding the RRT Tree

There are a continuum of action possibilities

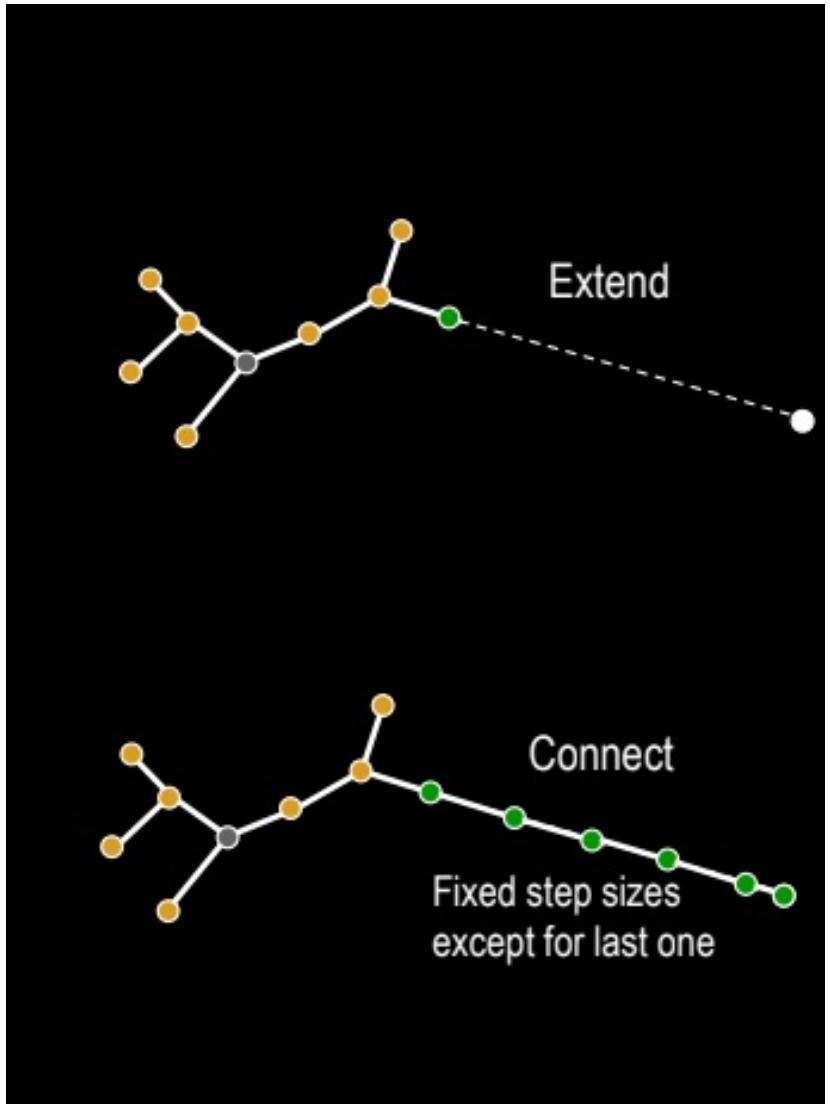
RRT is looking at some of them via sampling

Continuum of choices

Probabilistically subsample all edges

RRT Extension Types

- RRT-Extend
  - Take one step towards a random sample. Shorten if collides with an obstacle.
- RRT-Connect
  - Take a step towards the random sample until
    - The random sample is reached.
    - You hit an obstacle.



How to create a bias towards the goal?

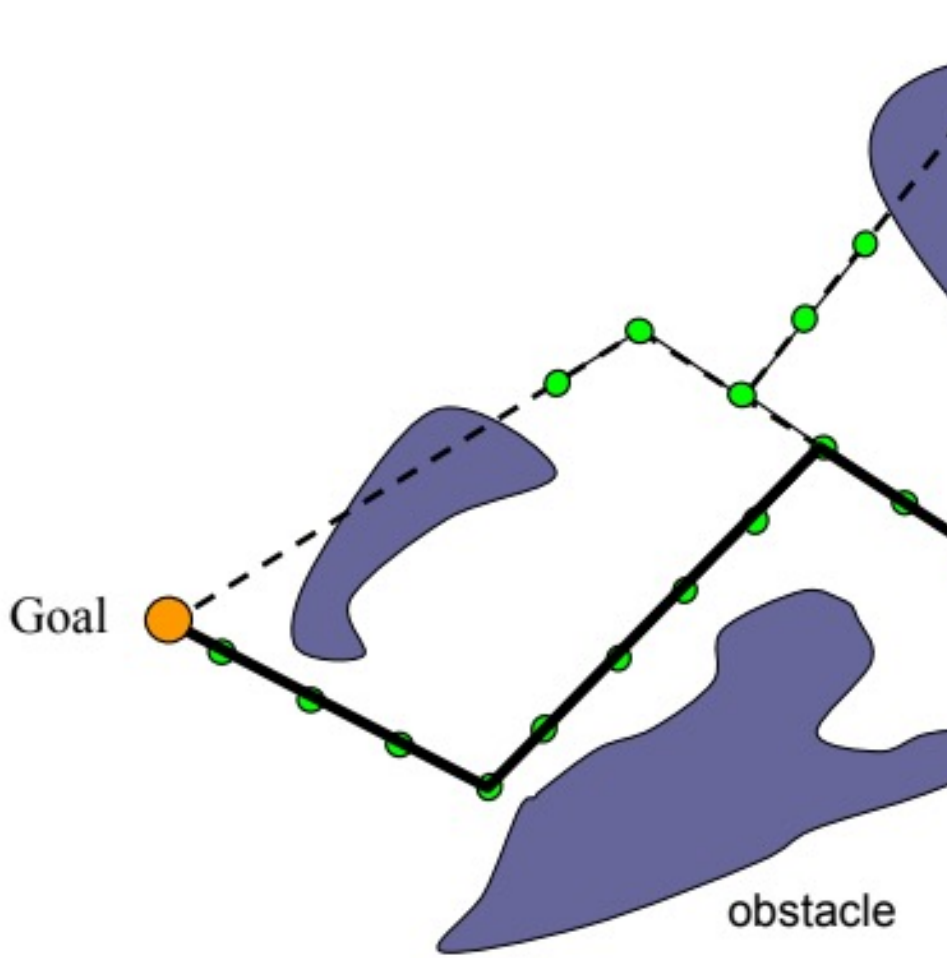
- RRT has a natural biases towards larger spaces (why? Large probability of landing a sample in that space)
- Creating a bias towards the goal.
  - When generating a random sample, with some probability pick the goal

RANDOM_STATE(): often uniformly at random over space with probability 99%, and the goal state with probability 1%, this ensures it attempts to connect to goal semi-regularly

GENERATE_RRT($x_{init}, K, \Delta t$)

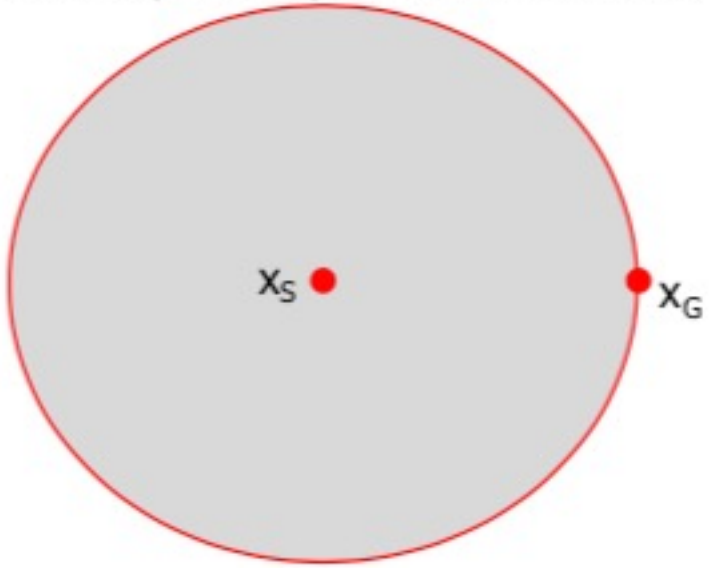
1 $\mathcal{T}$.init(x_{init});
2 **for** $k = 1$ **to** K **do**
3 $x_{rand} \leftarrow$ RANDOM_STATE();
4 $x_{near} \leftarrow$ NEAREST_NEIGHBOR($x_{rand}, \mathcal{T}$);
5 $u \leftarrow$ SELECT_INPUT(x_{rand}, x_{near});
6 $x_{new} \leftarrow$ NEW_STATE($x_{near}, u, \Delta t$);
7 $\mathcal{T}$.add_vertex(x_{new});
8 $\mathcal{T}$.add_edge(x_{near}, x_{new}, u);
9 Return $\mathcal{T}$

RRT with obstacles and goal bias

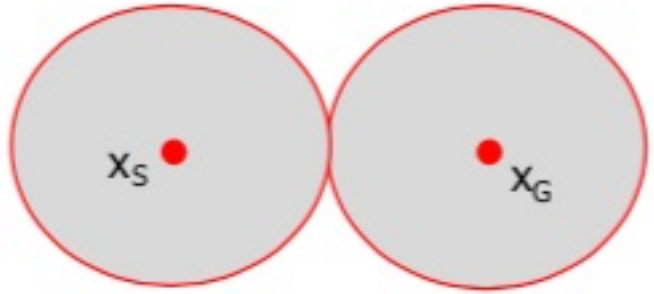


Bi-directional RRT

- Volume swept out by unidirectional RRT:



- Volume swept out by bi-directional RRT:

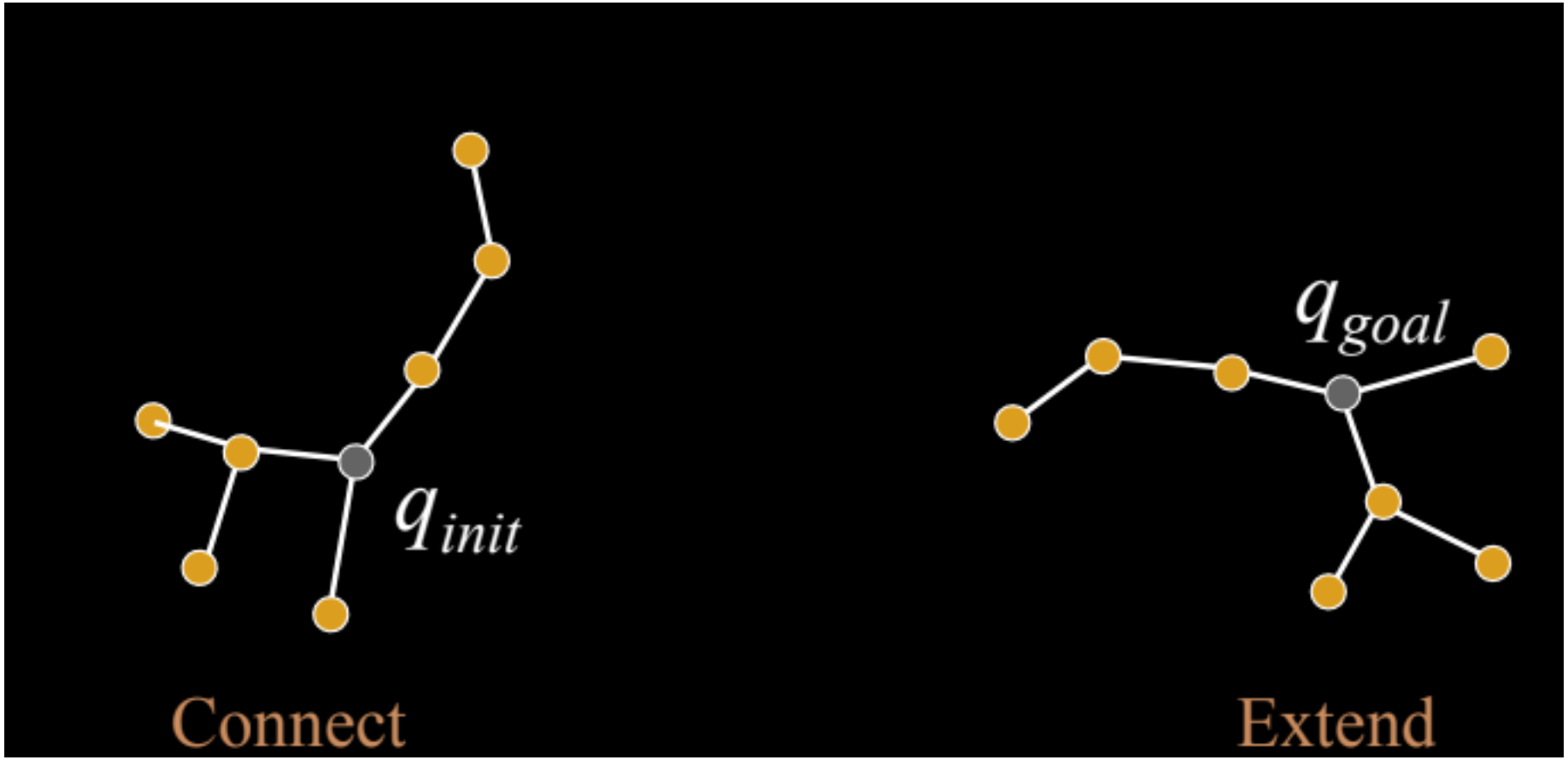


Idea: Growing one large tree (O(b^(d)) is more difficult that growing two smaller trees (O(b^(d/2)) where b and d are the breadth and the depth of the tree.

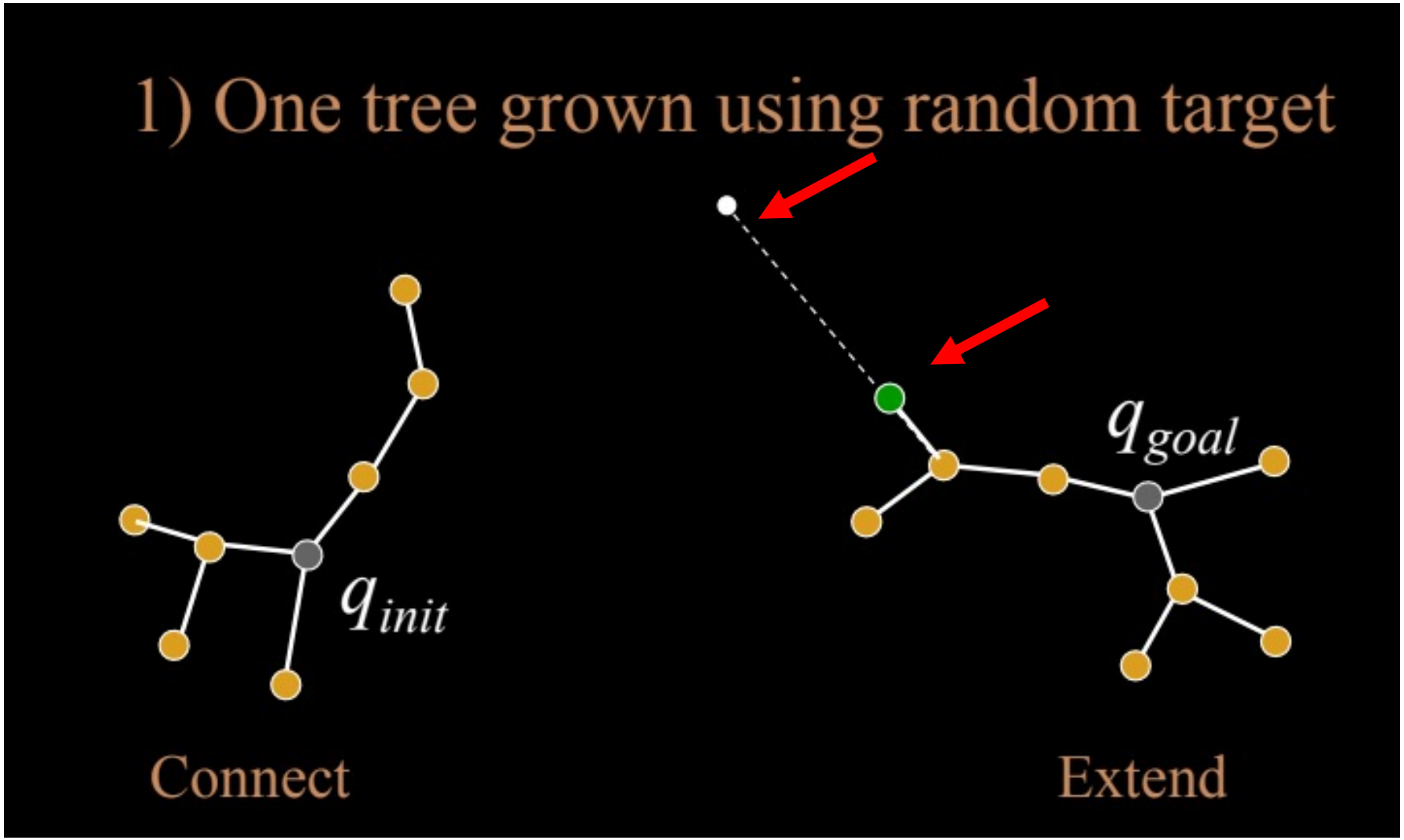
Bi-directional RRT

- Grow trees from both start and goal
- Try to get trees to connect to each other
- Trees can both use *Extend* or both use *Connect* or one use *Extend* and one *Connect*
- Usually, swap trees after each iteration
- Procedure
  - Grow the first tree by random sampling in free space.
  - The extended node node in the first tree serves as a target for growing the second tree.
  - Grow the second tree now with random sampling.
  - The extended node in the second tree serves as the target for the first tree.
  - Keep repeating till the trees meet.

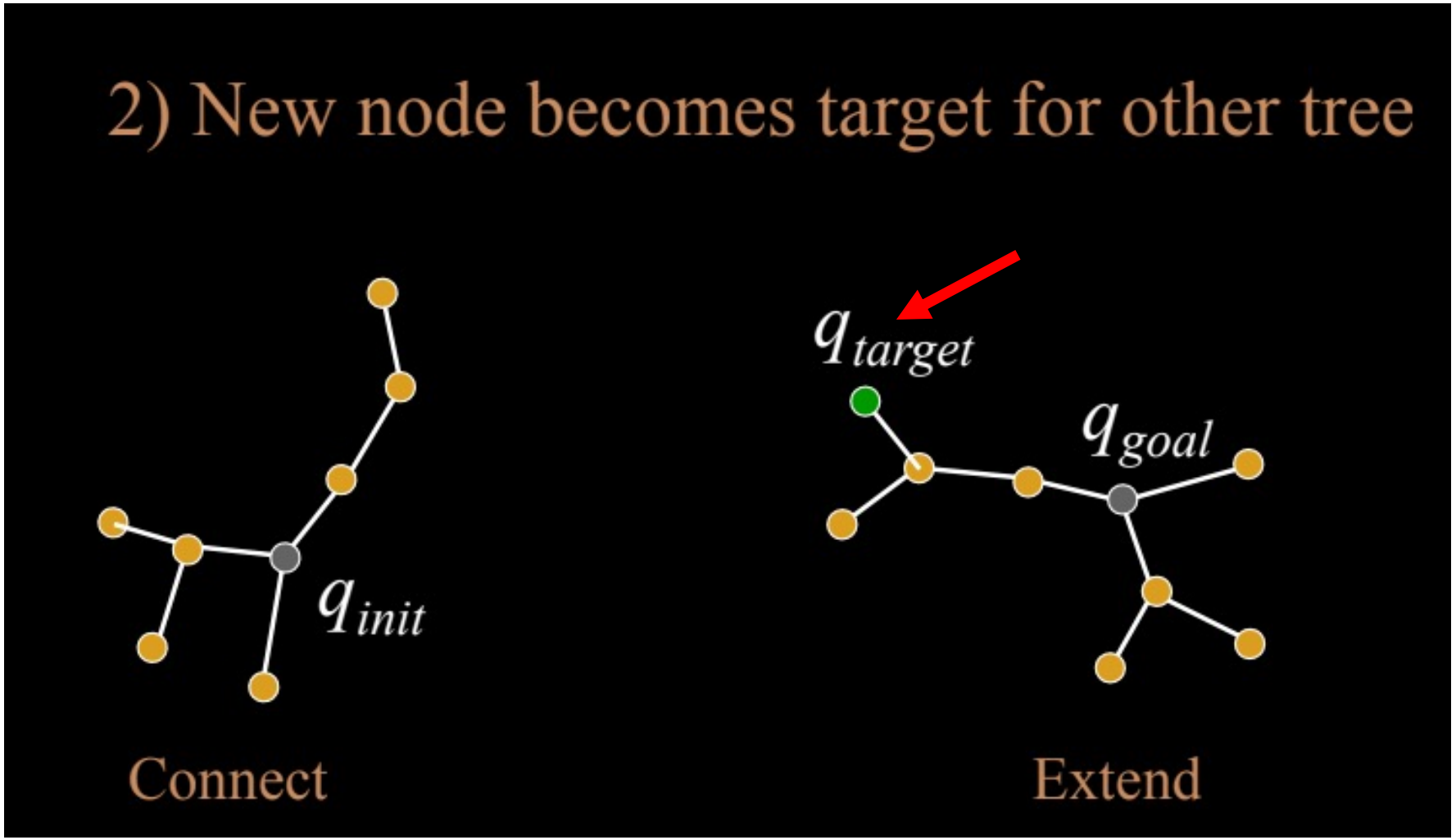
Bi-directional RRT



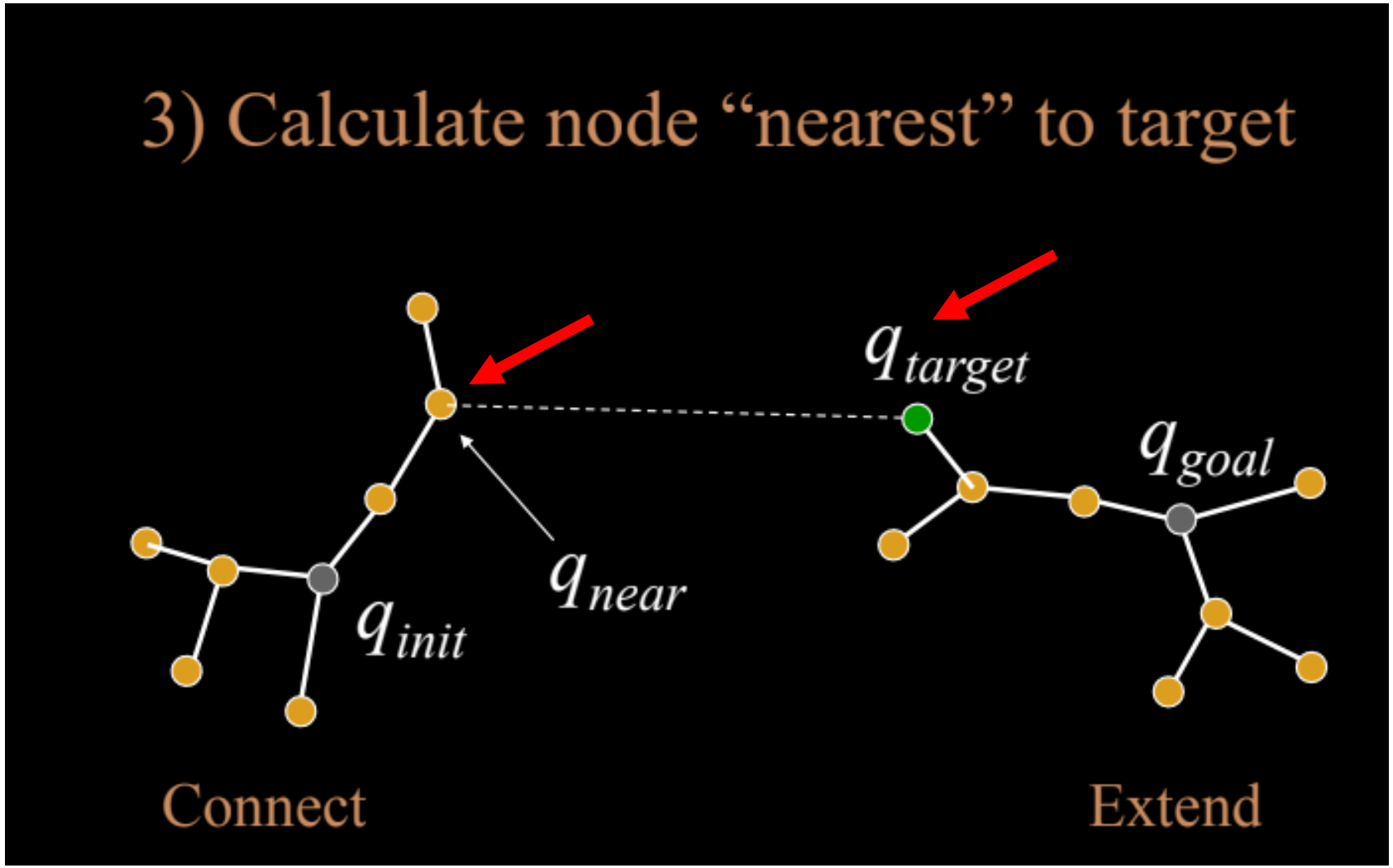
Bi-directional RRT



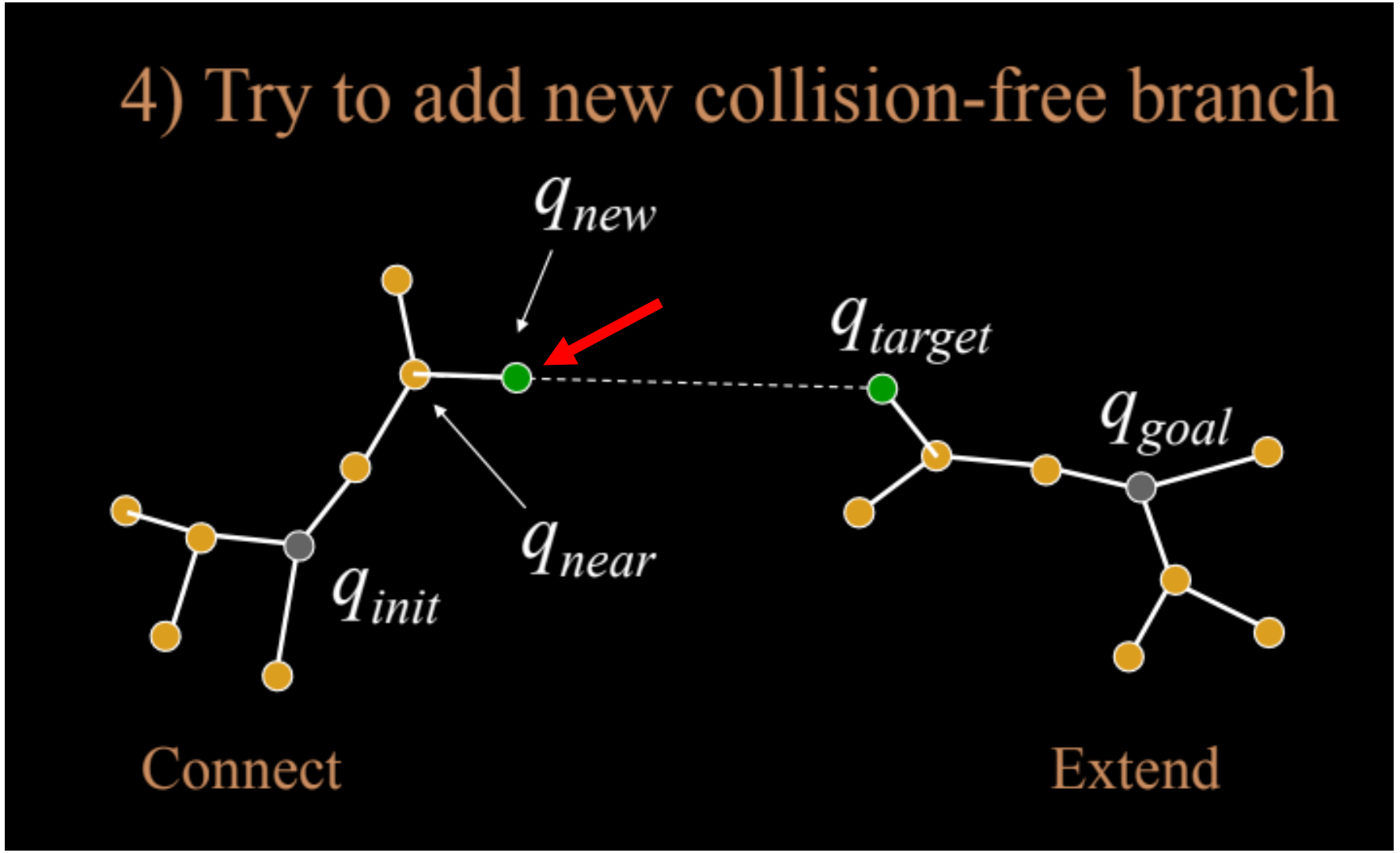
Bi-directional RRT



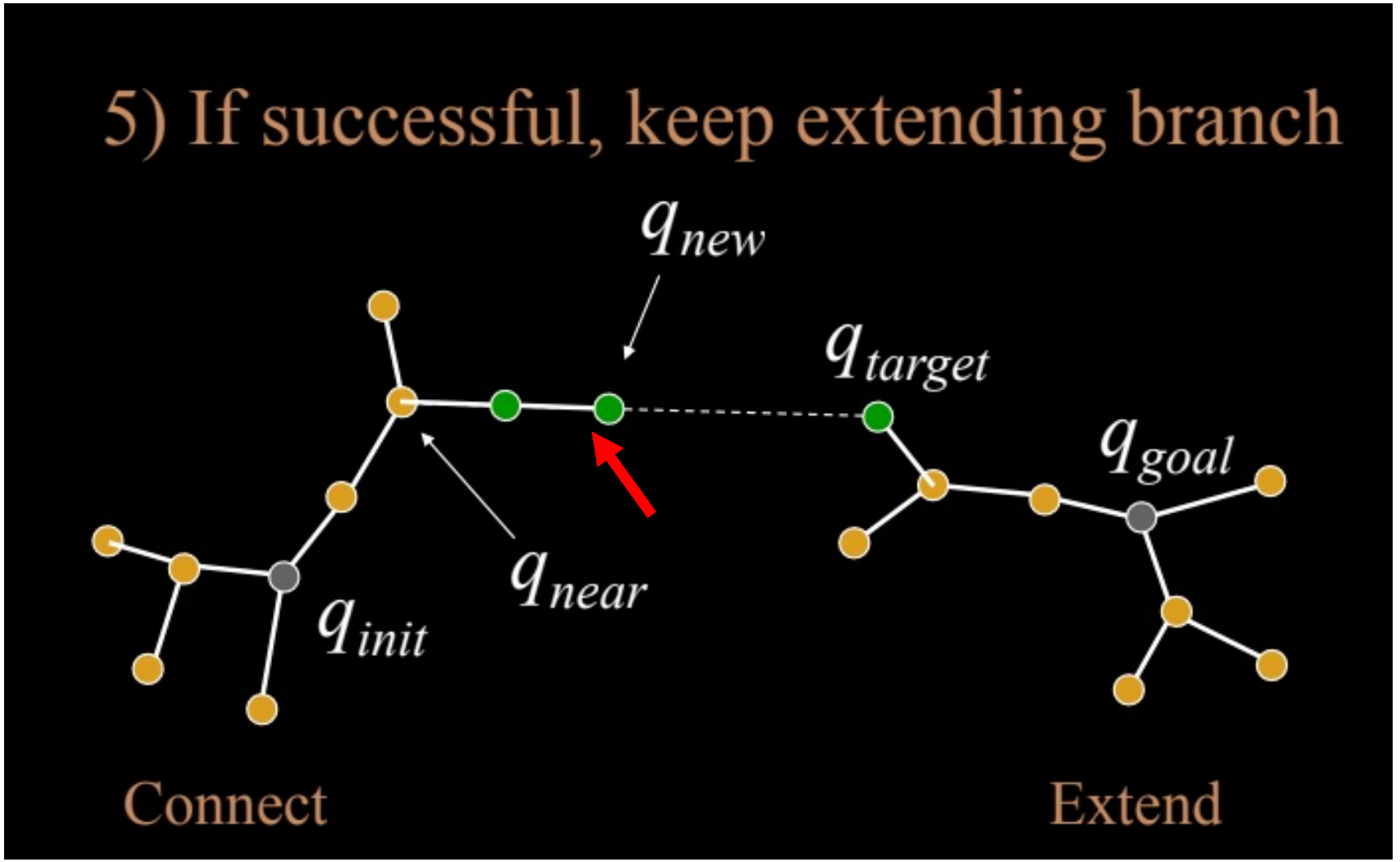
Bi-directional RRT



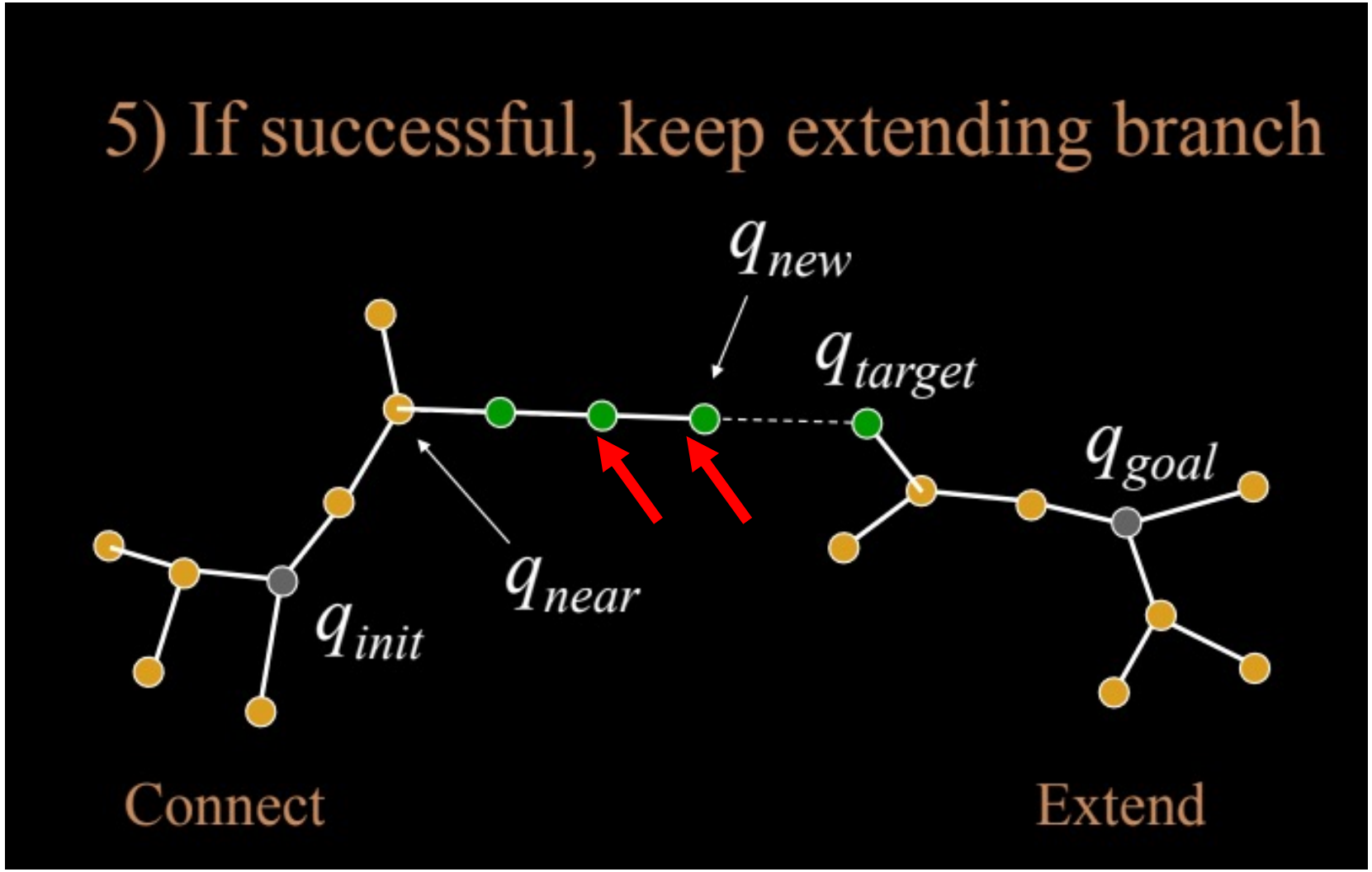
Bi-directional RRT



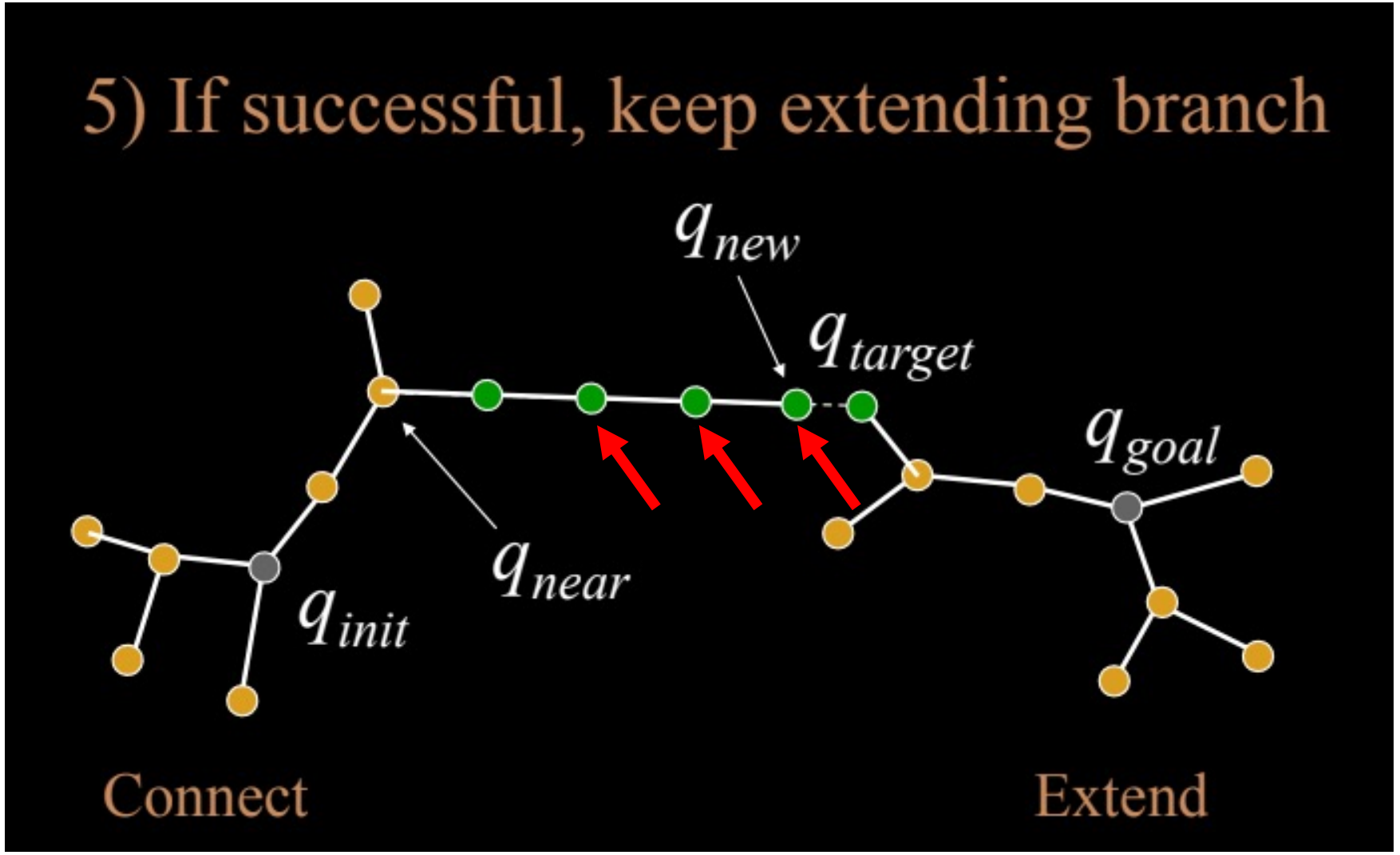
Bi-directional RRT



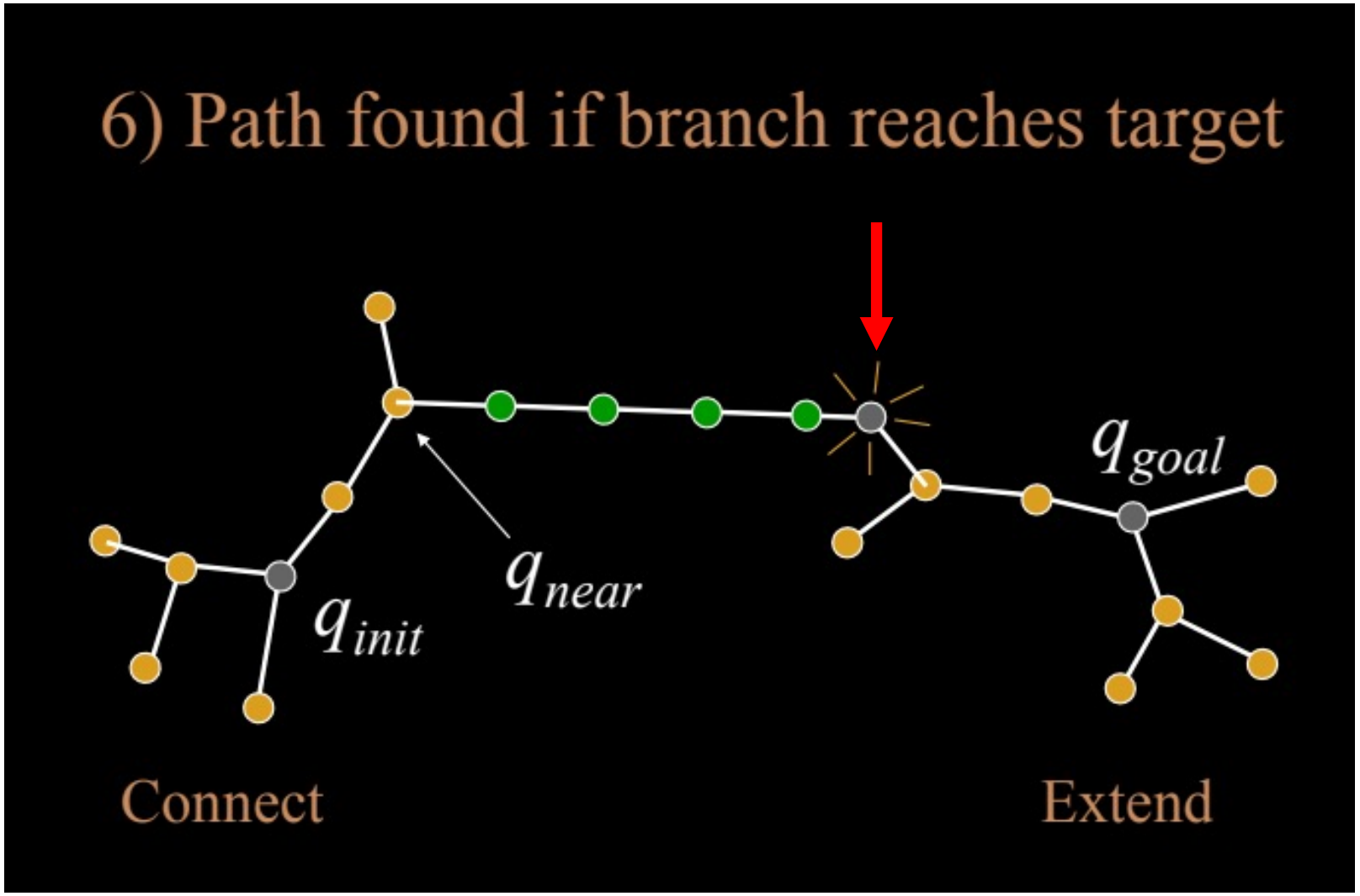
Bi-directional RRT



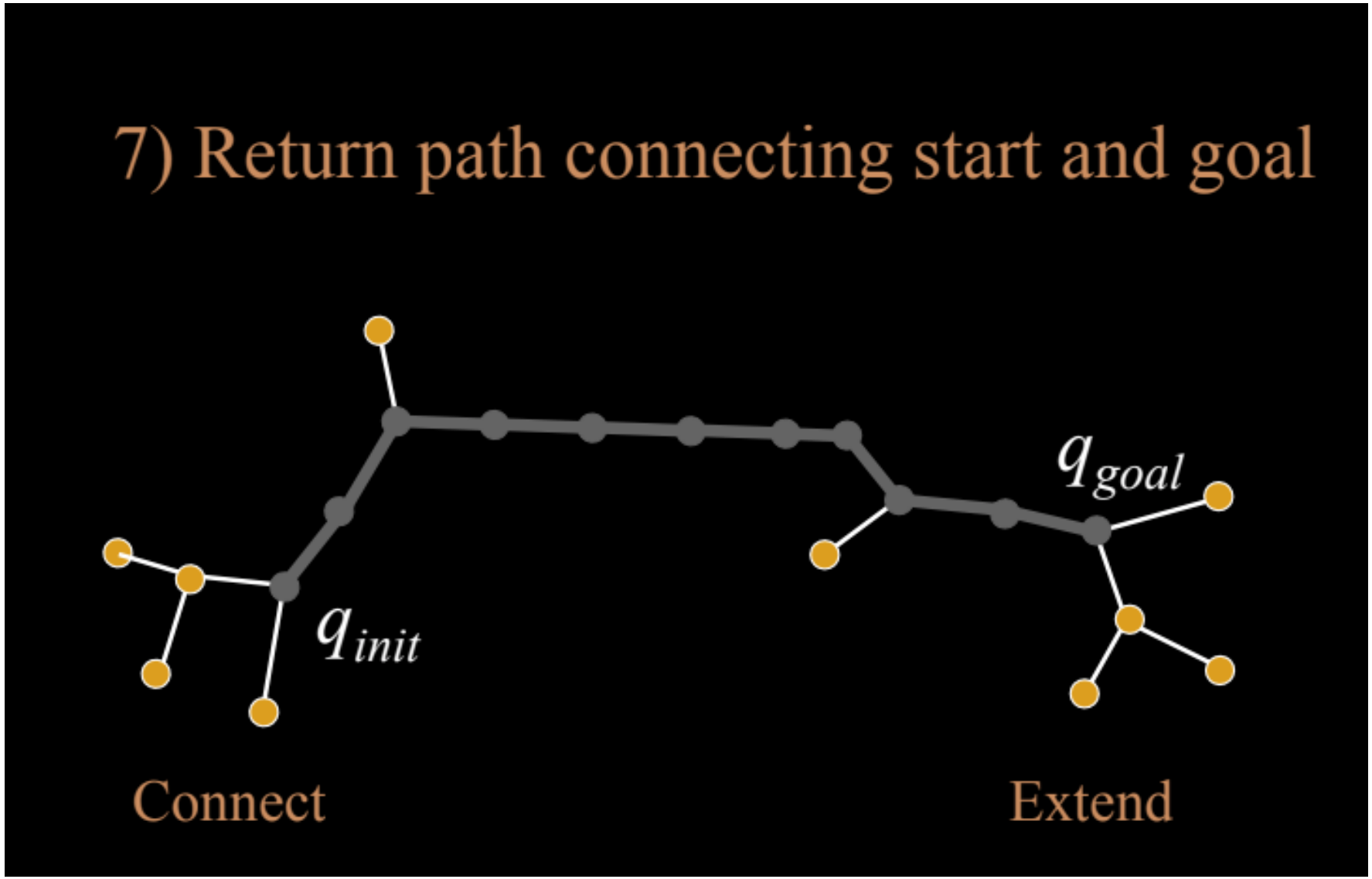
Bi-directional RRT



Bi-directional RRT

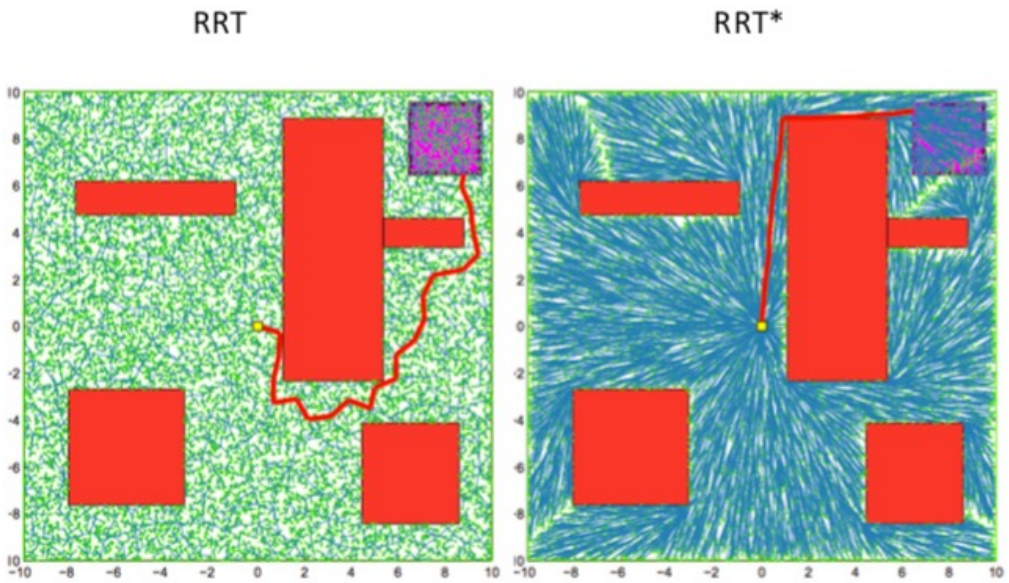


Bi-directional RRT



RRT*

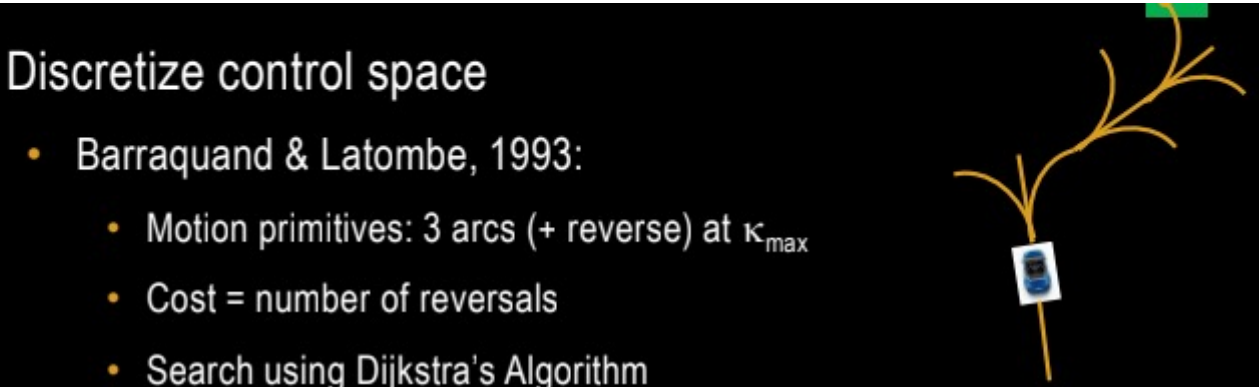
- Asymptotically optimal
  - In the limit, will find the optimal path.
  - Performs re-wiring of the tree.
- Karaman and Frazolli
  - https://dspace.mit.edu/handle/1721.1/63170
  - https://www.youtube.com/watch?v=6Pngam882hM
  - https://www.youtube.com/watch?v=2WOBMswcCA8



Incorporating Motion Constraints

- Non-holonomic systems have constrained ways to move in space.
- Connectivity
  - Connectivity in the state space depends on the allowed motions.
  - Cannot simply connect two configurations, that motion may not be allowed.
- Example: car

- Discretize control space
  - Barraquand & Latombe, 1993:
    - Motion primitives: 3 arcs (+ reverse) at κ_{max}
    - Cost = number of reversals
    - Search using Dijkstra's Algorithm
    - Disadvantage: Discontinuous curvature



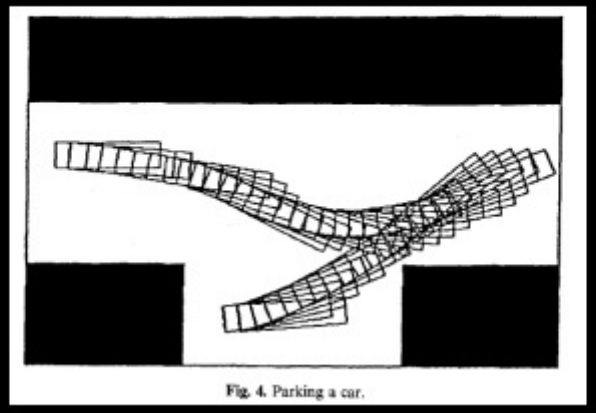
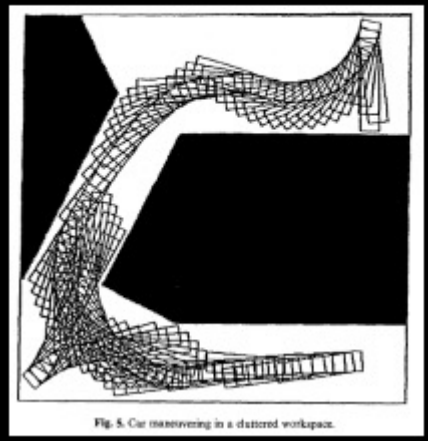
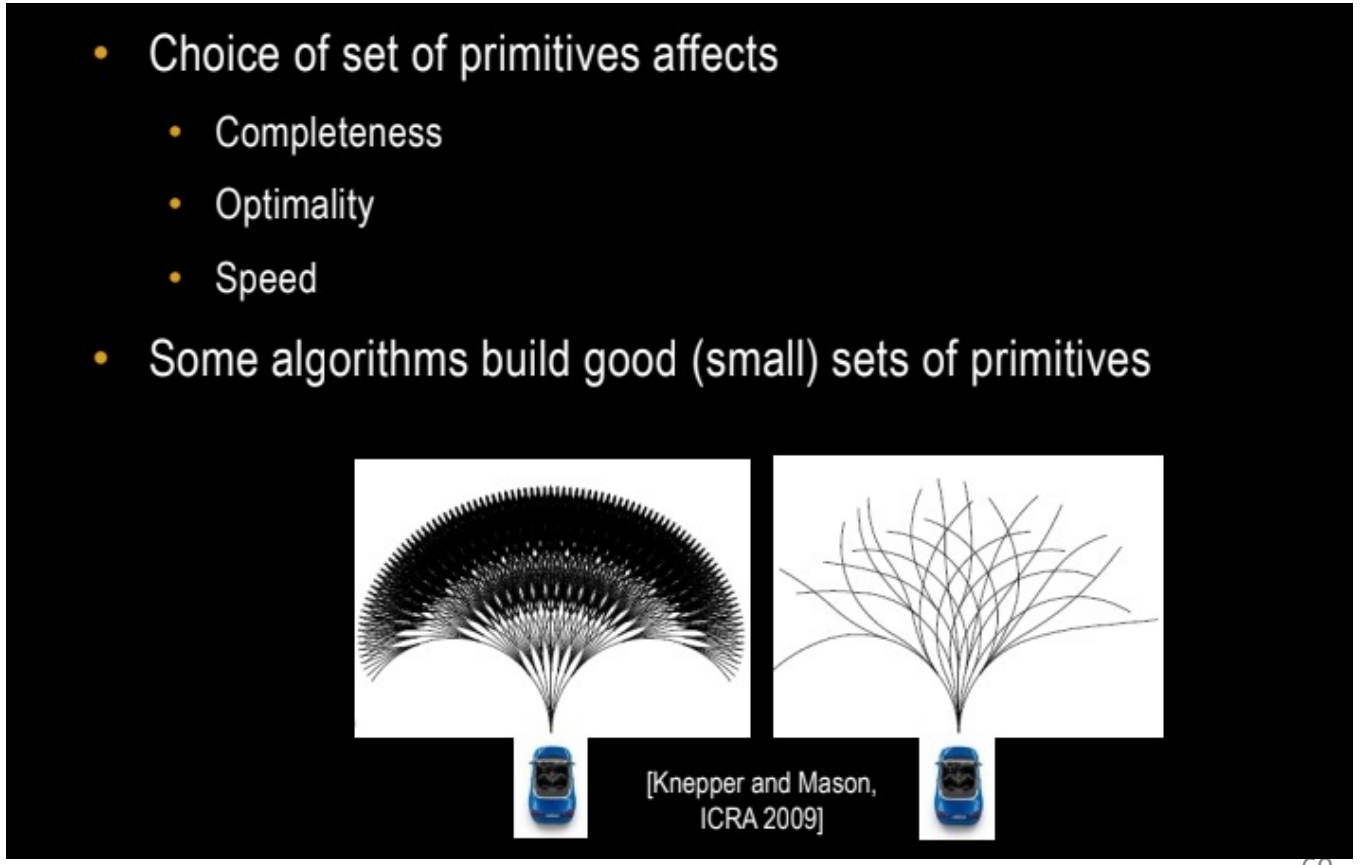


Fig. 4. Parking a car.



Incorporating Additional Motion Constraints

- Motion primitives: some basic motions that respect the constraints.
- Planning needs to sequence such motion primitives.
- Choice of primitives affects plans.

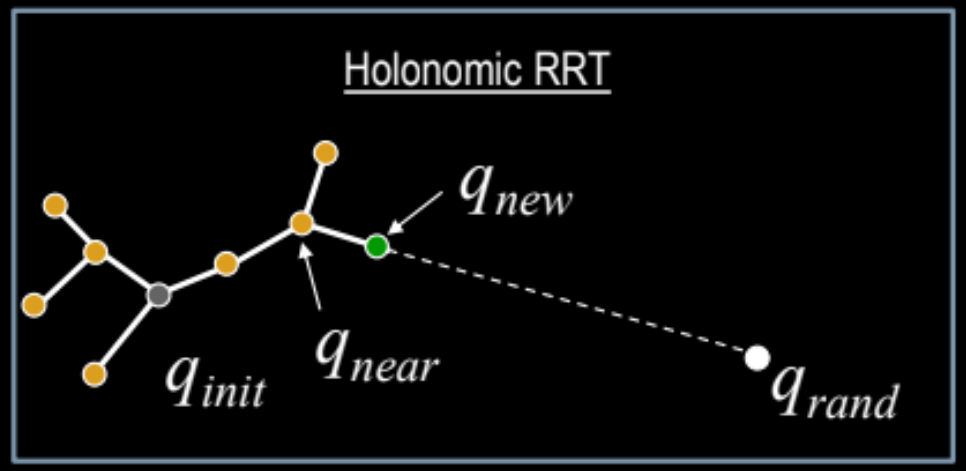


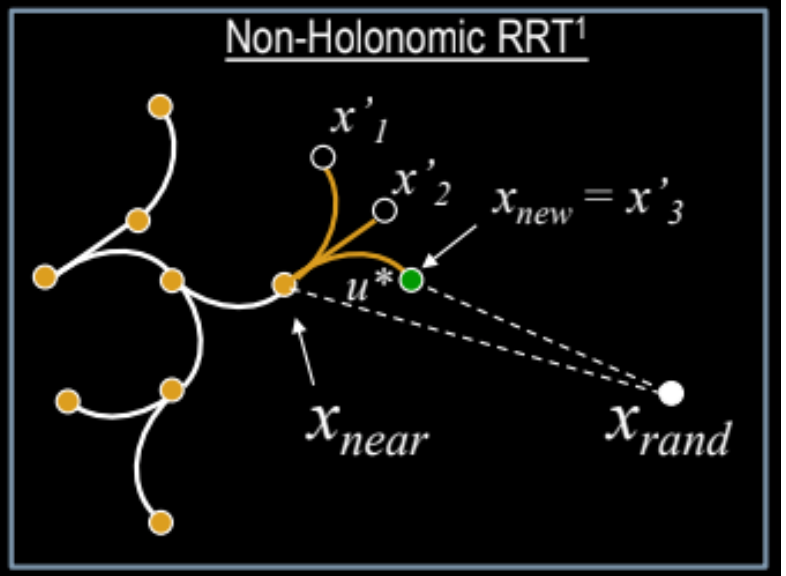
Kinodynamic (non-holonomic) RRT

- Apply motion primitives (i.e. simple actions) at q_{near}

$$x' = f(x,u) \text{ --- use action } u \text{ from } x \text{ to arrive at } x'$$

$$\text{chose } u_* = \arg\min_{u_i} d(f(x,u_i), x_{rand})$$





- You probably won't reach x_{rand} by doing this
  - Key point: No problem, you're still exploring!

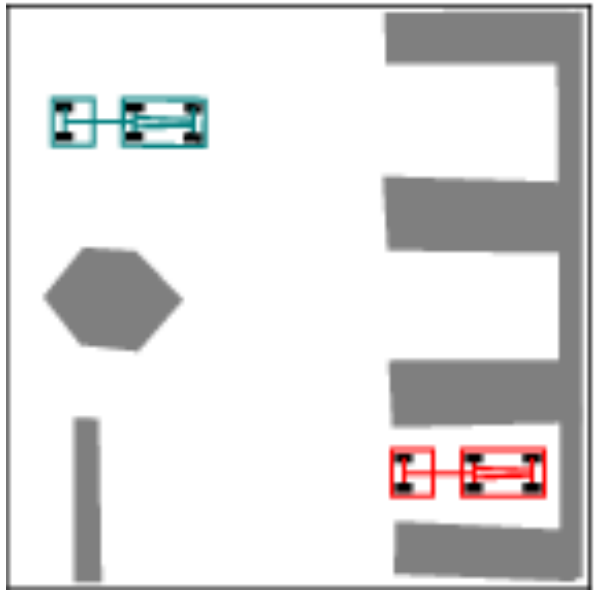
[1]often called *Kinodynamic RRT*

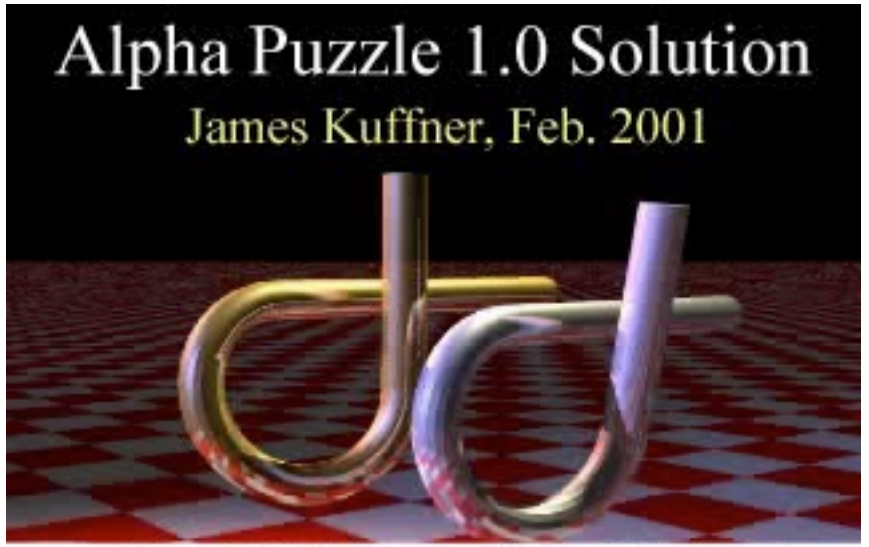
RRT Applications

Robotics Applications
- mobile robotics
- manipulation
- humanoids

Other Applications
- biology (drug design)
- manufacturing and virtual prototyping (assembly analysis)
- verification and validation
- computer animation and real-time graphics
- aerospace

RRT extensions
- discrete planning (STRIPS and Rubik's cube)
- real-time RRTs
- anytime RRTs
- dynamic domain RRTs
- deterministic RRTs
- parallel RRTs
- hybrid RRTs

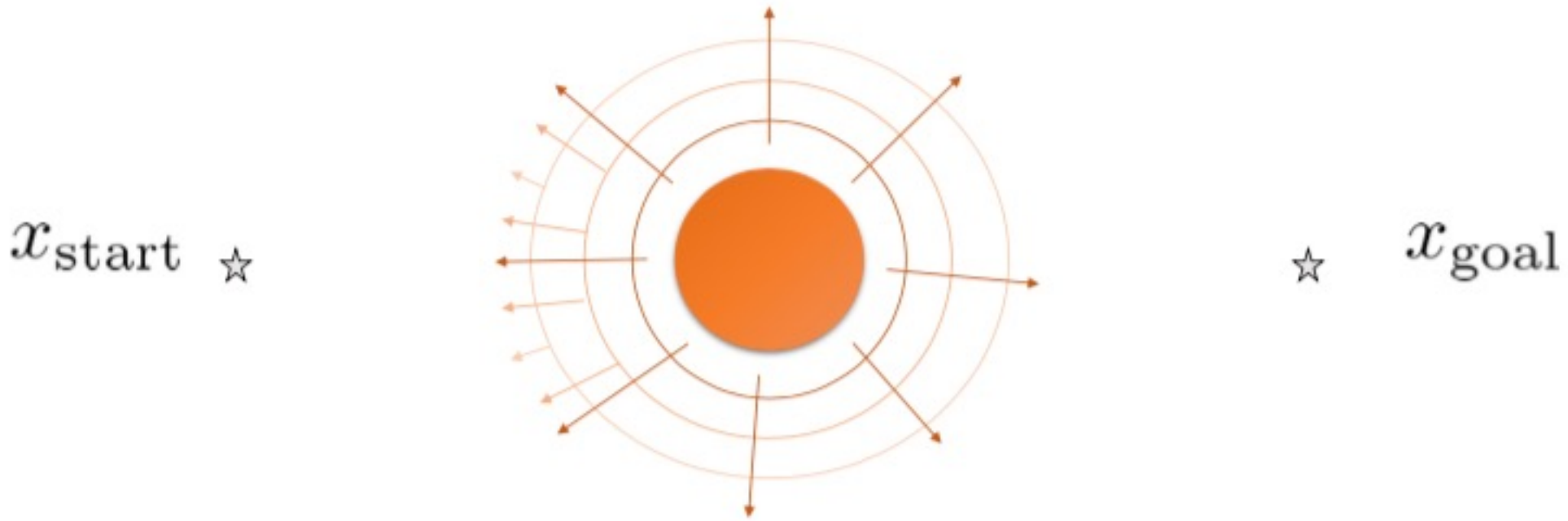




model by DSMFT group, Texas A&M Univ.
original model by Boris Yamrom, GE

http://lavalle.pl/rrt/gallery.html

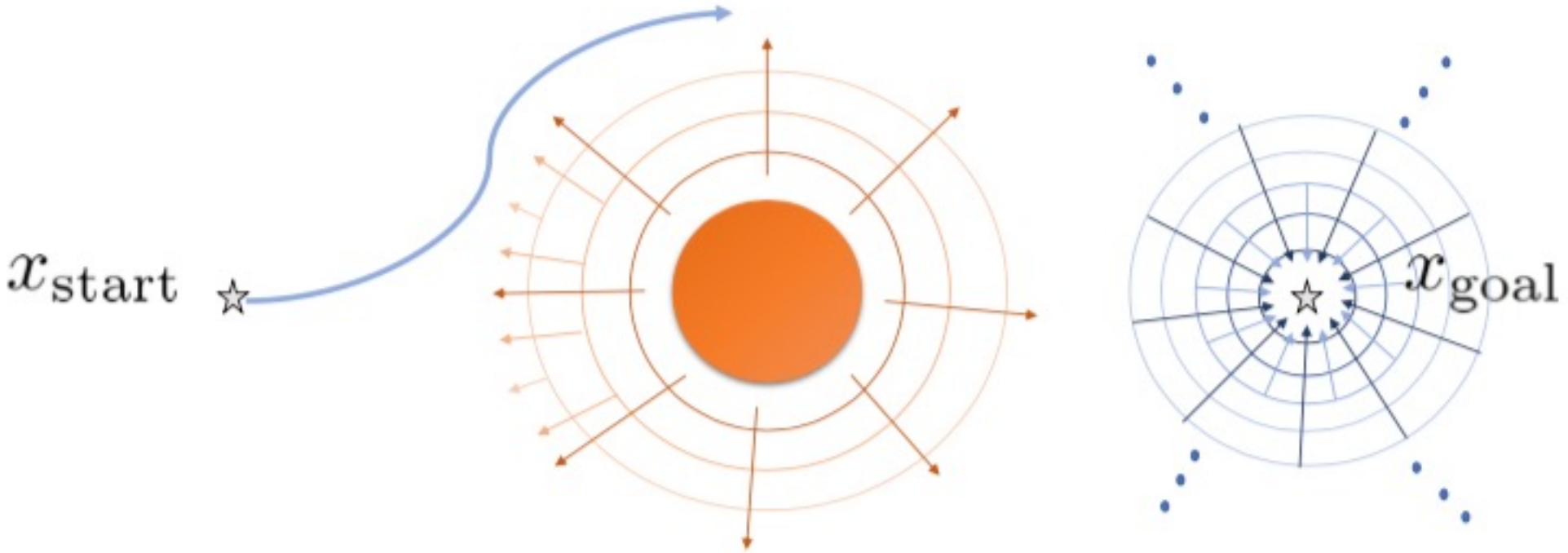
Alternate Approach: Potential Fields



Q: How do you control the robot to reach the goal state while avoiding the obstacle?

A: Place a repulsive potential field around obstacles

Potential Fields



Q: How do you control the robot to reach the goal state while avoiding the obstacle?

A: Place a repulsive potential field around obstacles
and an attractive potential field around the goal

Potential Fields

At any point $\mathbf{x}$ we can write the total potential $\mathbf{U}_\Sigma$ as a sum of the potential induced $\mathbf{U}_o$ by k obstacles and the potential induced by the goal $\mathbf{U}_g$:

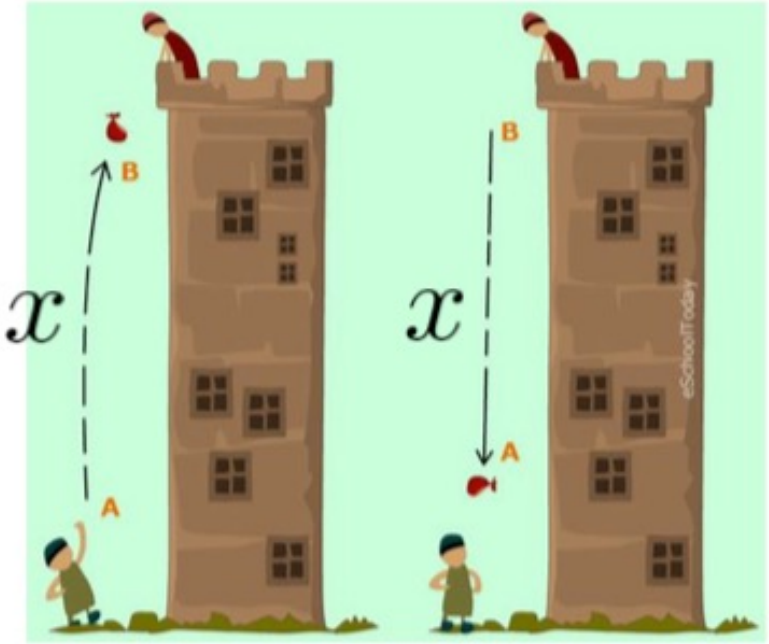
$$\mathbf{U}_\Sigma(\mathbf{x}) = \sum_{i=1:k} \mathbf{U}_{o,i}(\mathbf{x}) + \mathbf{U}_g(\mathbf{x}) \tag{2.7}$$

Now we know that the force $\mathbf{F}(\mathbf{x})$ exerted on a particle in a potential field $\mathbf{U}_\Sigma(\mathbf{x})$ can be written as :

$$\mathbf{F}(\mathbf{x}) = -\nabla \mathbf{U}_\Sigma(\mathbf{x}) \tag{2.8}$$

$$= -\sum_{i=1:k} \nabla \mathbf{U}_{o,i}(\mathbf{x}) - \nabla \mathbf{U}_g(\mathbf{x}) \tag{2.9}$$

Potential Fields



$$U(x) = mgx$$

$$F(x) = -mg$$

In both cases we have conversion from kinetic energy to potential energy U(x).

In both cases there is a force resulting from the potential field, and F(x)=-dU(x)/dx.

This is a general rule for conservative systems with no external forces.

Potential Fields

$\rho(\mathbf{x})$ Shortest distance between the obstacle and vehicle at x.

ρ_0 limit on the region of space affected by the potential field

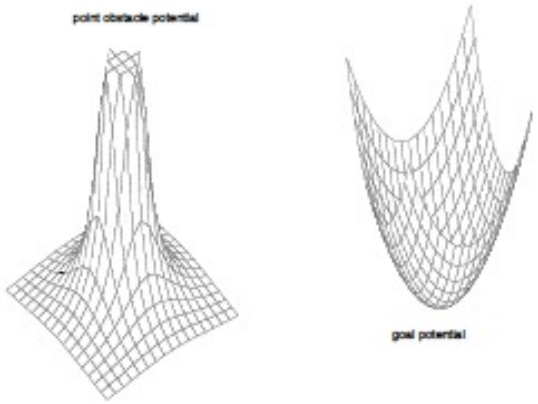
$$\mathbf{U}_{o,i}(\mathbf{x}) = \eta \begin{cases} \frac{1}{2}\left(\frac{1}{\rho(\mathbf{x})} - \frac{1}{\rho_0}\right)^2 & \forall \quad \rho(\mathbf{x}) \leq \rho_0 \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{U}_g(\mathbf{x}) = \frac{1}{2}(\mathbf{x} - \mathbf{x}_g)^2$$

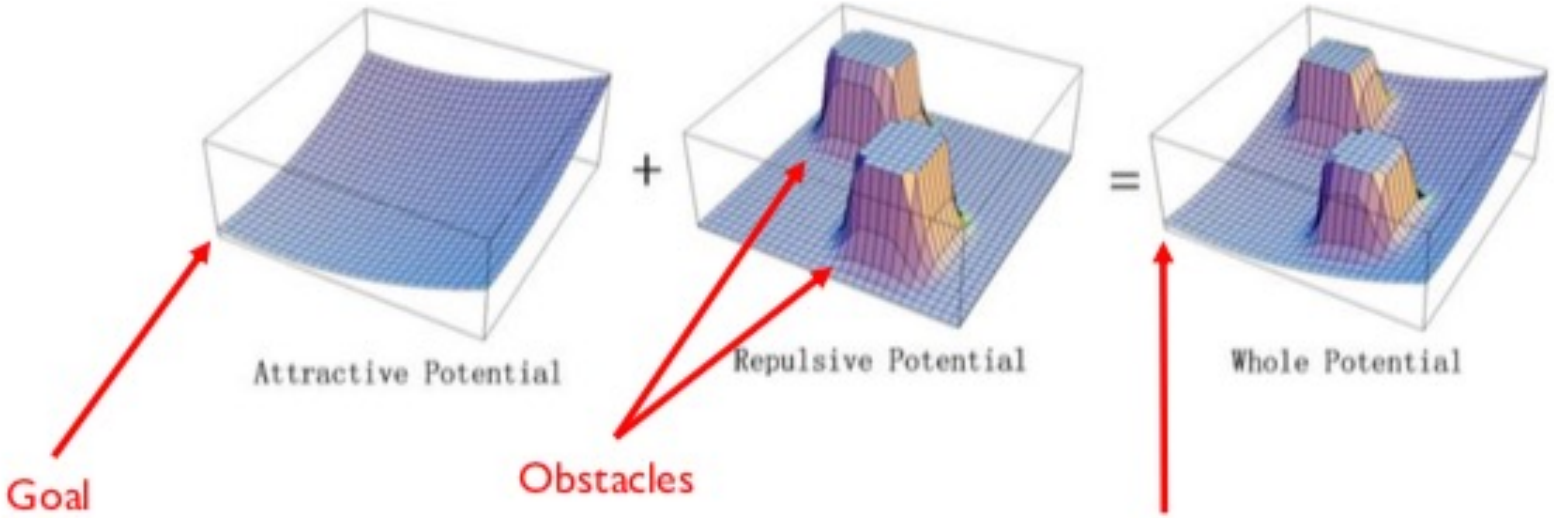
$$\mathbf{F}_{o,i} = \begin{cases} \eta\left(\frac{1}{\rho(\mathbf{x})} - \frac{1}{\rho_0}\right)\frac{1}{\rho(\mathbf{x})^2}\frac{\partial\rho(\mathbf{x})}{\partial\mathbf{x}} & \forall\rho(\mathbf{x}) \leq \rho_0 \\ 0 & \text{otherwise} \end{cases}$$

Core idea:

- Represent the influence of obstacles and the goal as potential functions.
- The derivative will give the force exerted.
- Two typical potential functions - inverse quadratic for the obstacles and quadratic for the goal.
- In essence, retain the continuous nature of the problem.



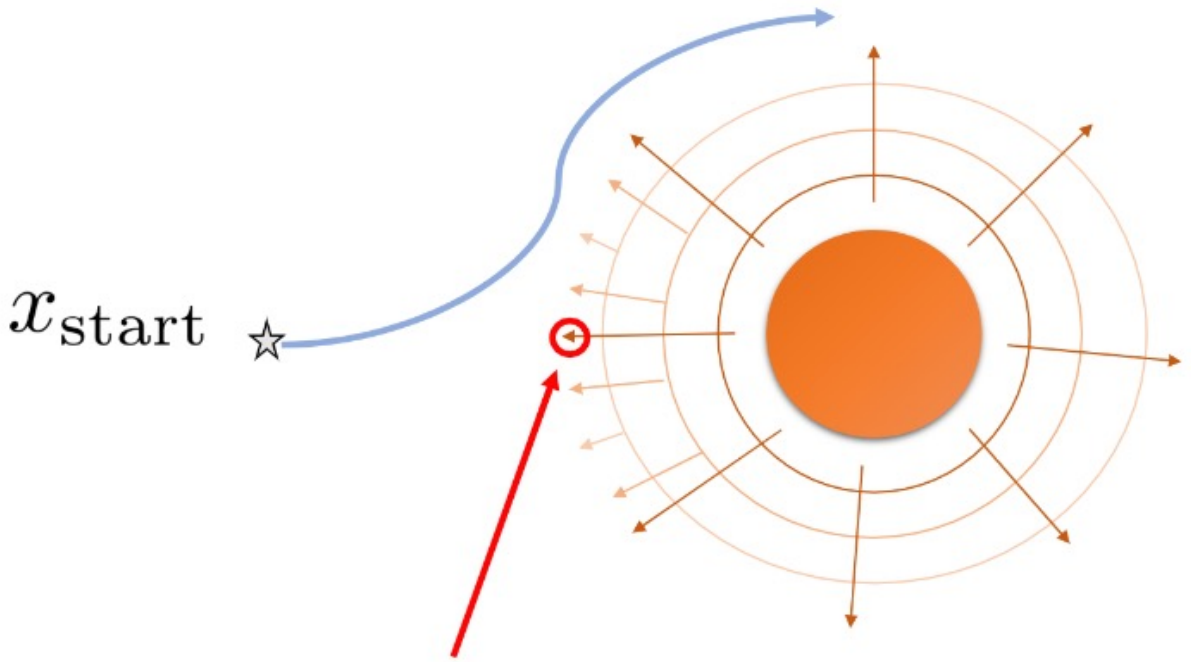
Potential Fields

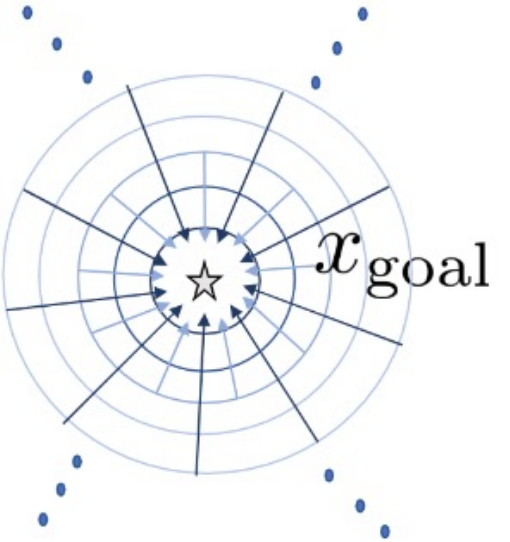


$$U(x) = \alpha U_{\text{attractive}}(x) + \beta U_{\text{repulsive}}(x)$$

Q1: How do we reach the goal state from an arbitrary state?

Potential Fields





What's the total potential here?

It's zero. The repulsive force is exactly the opposite of the attractive force (assuming alpha = beta)

$$F_{\text{total}}(x) = \alpha F_{\text{attractive}}(x) + \beta F_{\text{repulsive}}(x) = 0$$

Problem: gradient descent gets stuck

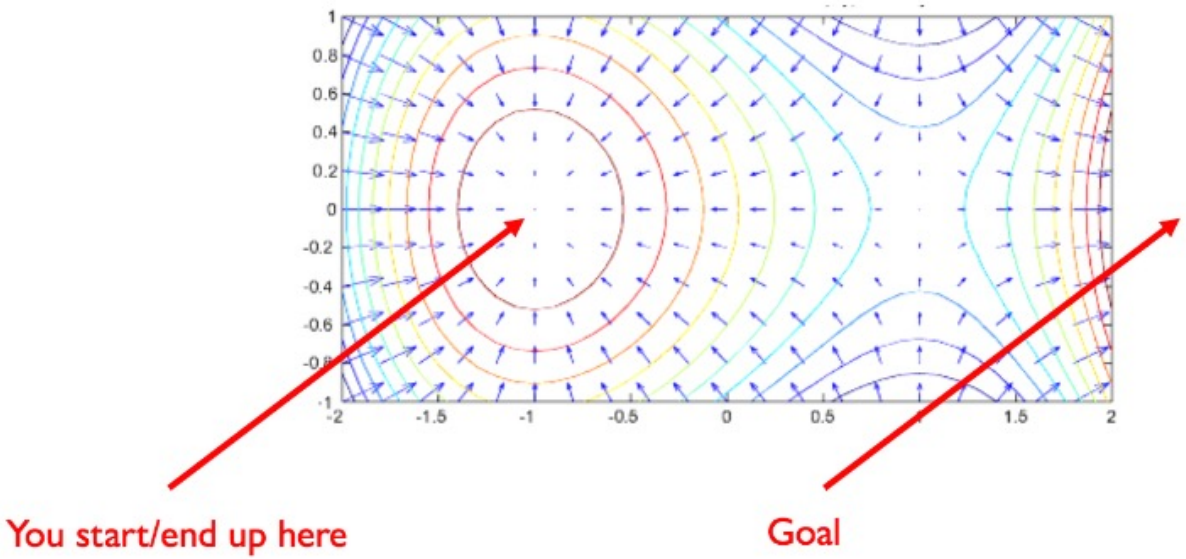
Potential Fields

Only acts locally. There is no global path planning.

The vehicle simply reacts to local obstacles, always moving in a direction of decreasing potential.

Example: the vehicle will descend into a local minima and stop. Any other motion will increase its potential.

States of zero total force correspond to local minima in the potential function:



Limitation: Susceptibility to Local Minima

Only acts locally. There is no global path planning.

The vehicle simply reacts to local obstacles, always moving in a direction of decreasing potential.

Example: the vehicle will descend into a local minima in front of the two features and will stop. Any other motion will increase its potential.