AI Robotics

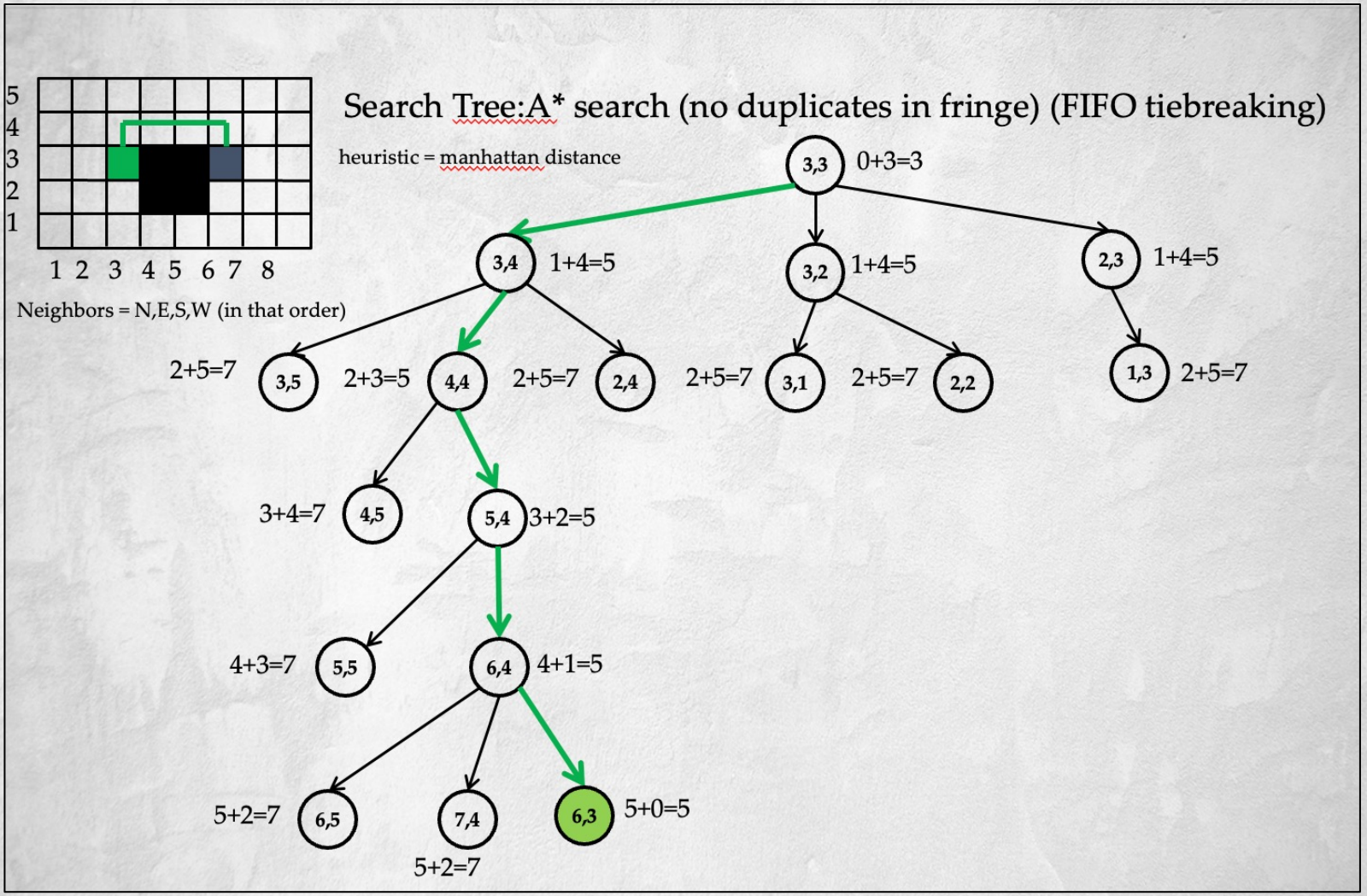
Randomized Planning

Overview

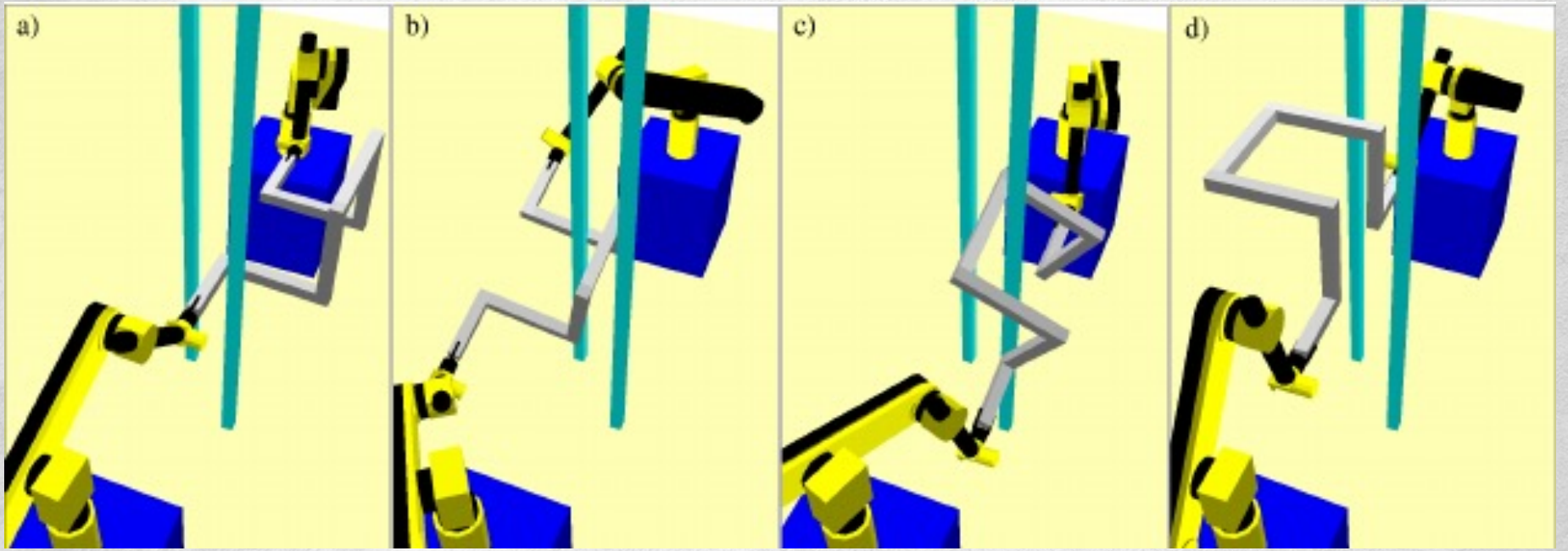
Planning

- Mission planning
  - Which tasks to perform and what order to perform them to achieve a specified goal
  - Tasks may have preconditions and effects
- Path-planning
  - Focus is on robot's (*x, y, theta*) pose
    - What is it now?
    - What do we want it to be?
    - What sequence of poses will get me from where it is now to where it should be?
- Motion-planning
  - Focus also on robot's pose (called a configuration), but in general this can be a high-dimensional pose, consisting of many joint angles (think a robotic arm)

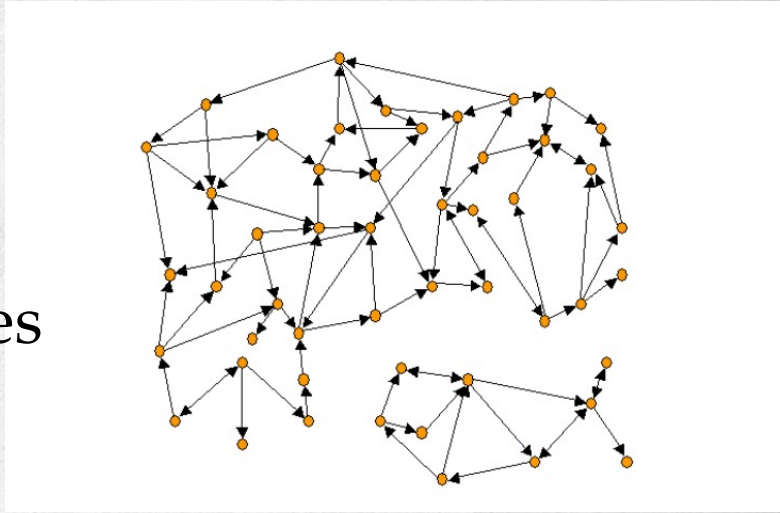
Path Planning



Motion Planning



Search space



- Vertices – states
- Edges – connect neighboring states

- Planning algorithm needs to find a *solution* or a path from the initial state to the goal state
- The *cost* of a solution is the sum of the costs of each edge in the path
  - For our purposes, this will be the length of the path
- An *optimal solution* is a solution with the lowest cost of any solution

Search Space

- The **Search Space** for a planning problem is the set of all poses/configurations/states that we will planning in
- In robotics problems the search spaces are typically *continuous*. This means that there is an uncountably infinite number of possible states
- The planning algorithms we will discuss today require a *discrete* state space to work with
- How can we come up with a discrete version of a continuous state space?

Configuration Space

Position and orientation of end effector can be determined from joint angles

Configuration, $q = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$

Position

$$x = x_2 = \alpha_1 \cos\theta_1 + \alpha_2 \cos(\theta_1 + \theta_2)$$
$$y = y_2 = \alpha_1 \sin\theta_1 + \alpha_2 \sin(\theta_1 + \theta_2)$$

Orientation

$$\begin{bmatrix} \cos(\theta_1 + \theta_2) & -\sin(\theta_1 + \theta_2) \\ \sin(\theta_1 + \theta_2) & \cos(\theta_1 + \theta_2) \end{bmatrix}$$

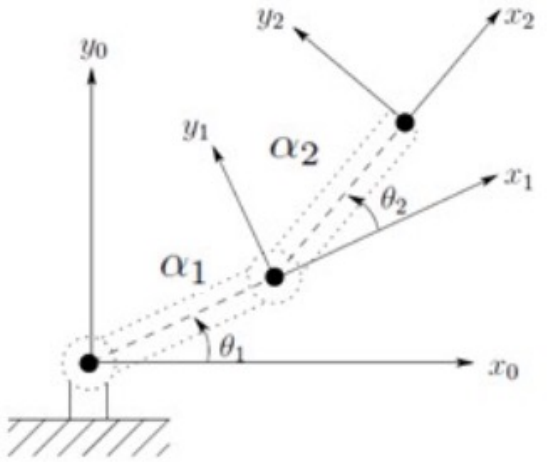


Fig. 1.24 Coordinate frames for two-link planar robot.

Configuration Obstacle

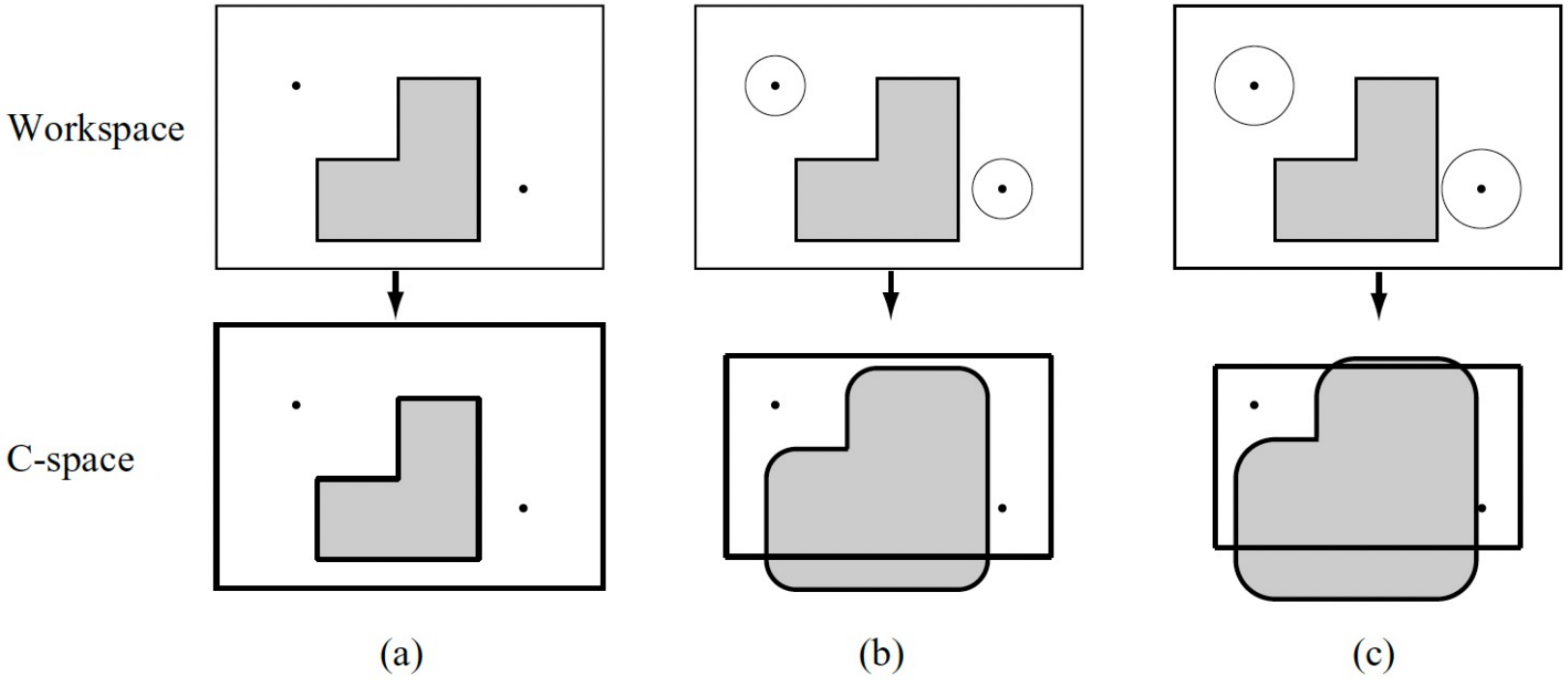
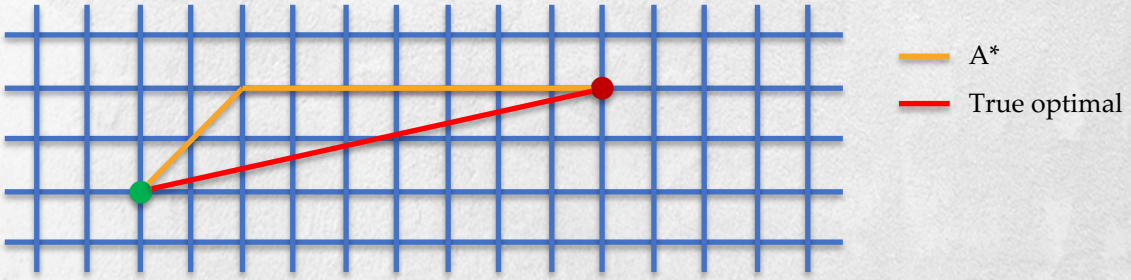


Figure 3.5 The top row shows the workspace and the bottom row shows the configuration space for (a) a point mobile robot, (b) a circular mobile robot, and (c) a larger circular mobile robot.

Problems

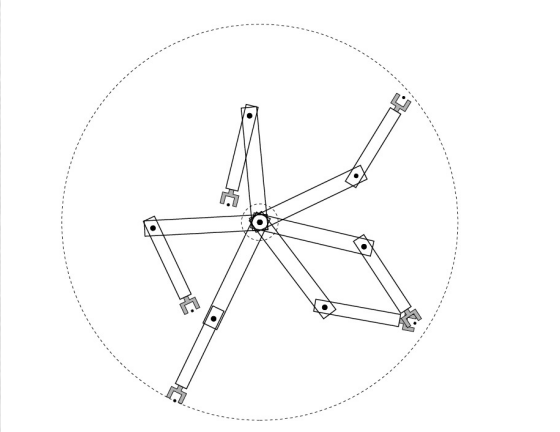
Problem with Graph Search

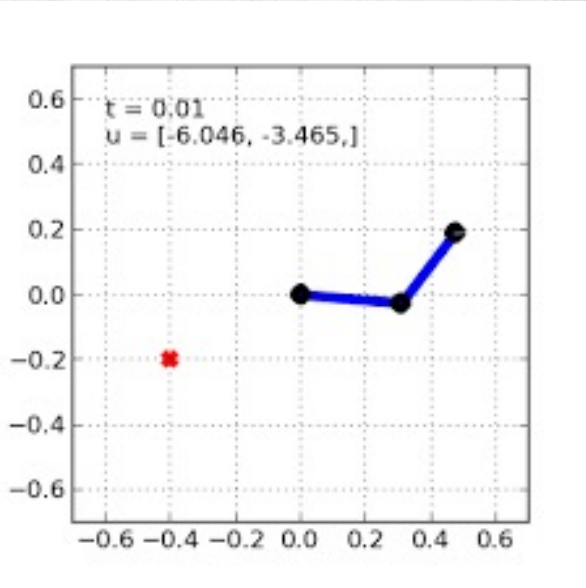
- A* gives us an optimal solution, but
  - no solution is returned until the algorithm terminates
  - the space consumption does not scale well
- Does it give us a true-optimal solution?



Configuration Space

- Need to take shape of robot into account
- Need to take shape of obstacles into account
- Need to create a plan in terms of robot joint angles





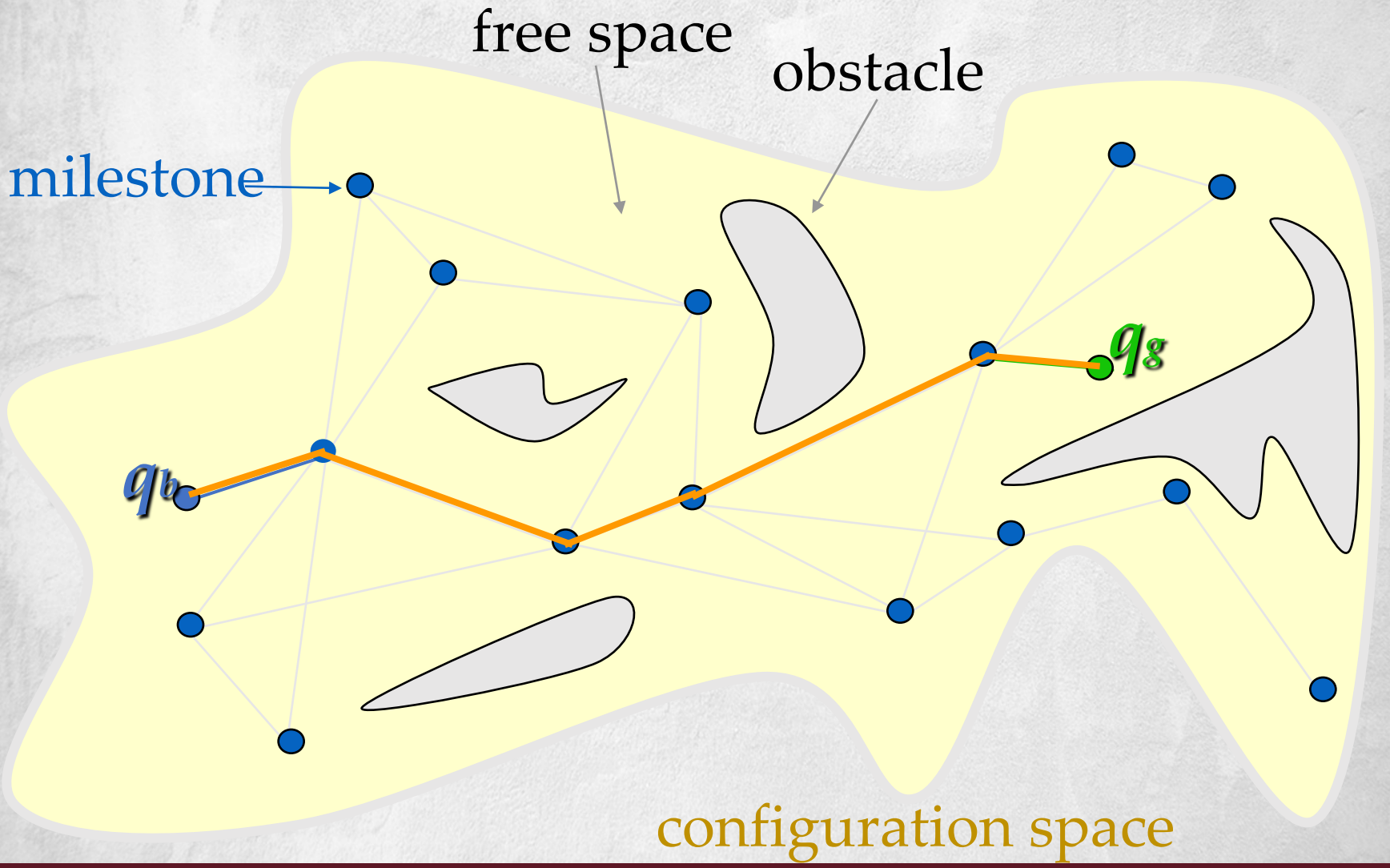
Probabilistic Roadmaps (PRM)

Probabilistic Roadmap

Steps:

1. Randomly generate set of configurations (milestones) that are in free space
2. Connect each milestone to nearby milestones that can be reached without going through an obstacle
3. This is the roadmap (graph)
4. To plan path, connect start and goal to roadmap, plan path on roadmap
   - Use graph-planning techniques that we covered before

Principle of Randomized Planning



Desirable Properties of a PRM

- Coverage:
  The milestones should see most of the admissible space to guarantee that the initial and goal configurations can be connected to the roadmap
- Connectivity:
  There should be a 1-to-1 map between the components of the admissible space and those of the roadmap

In Theory, a PRM Planner …

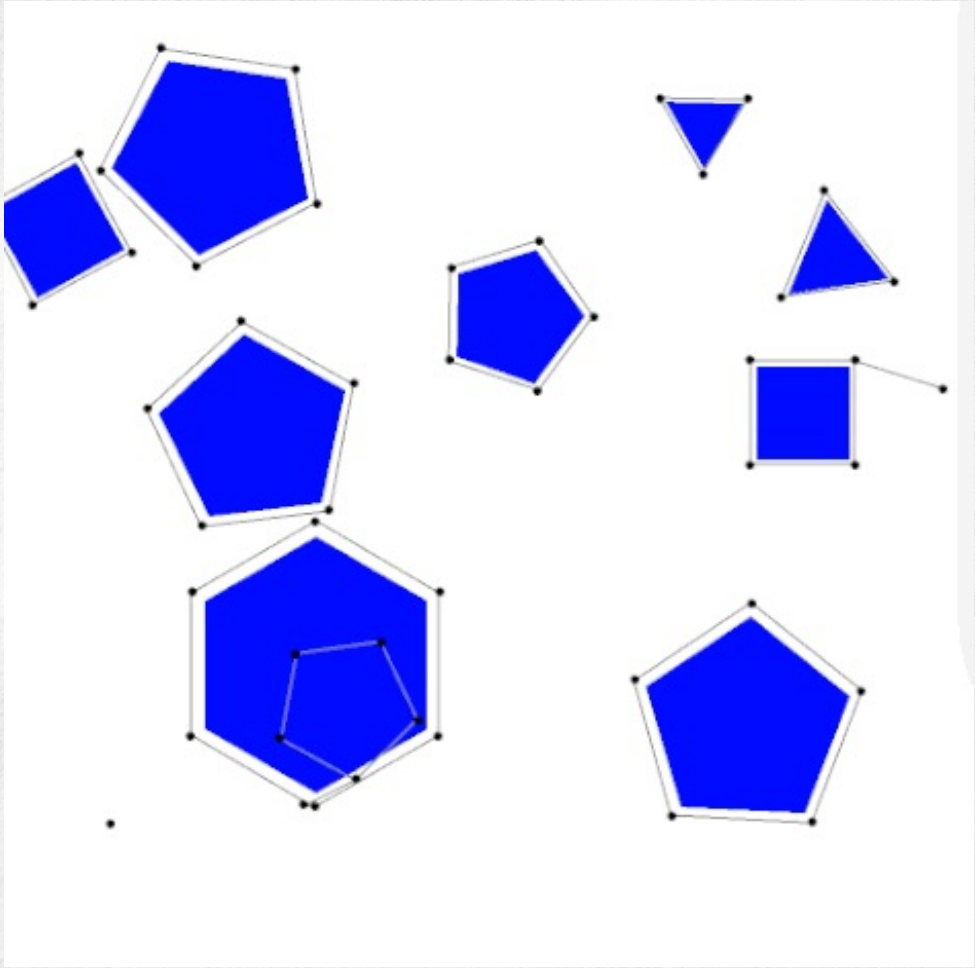
- Is probabilistically complete, i.e., whenever a solution exists, the probability that it finds one tends toward 1 as the number N of milestones increases
- Under rather general hypotheses, the rate of convergence is exponential in the number N of milestones, i.e.:

$$\text{Prob[failure]} \sim \exp(-N)$$

In practice, PRM Planners …

- Are fast
- Deal effectively with many-dof robots
- Are easy to implement
- Can solve both problems in Euclidean space and configuration space

PRM Example



Rapidly-Exploring Random Trees (RRT)

Rapidly-Exploring Random Trees

- Input: Initial configuration *S*, number of vertices in RRT *K*, and incremental distance *d*
- Output: RRT Graph *G*

1. Add *S* to *G*
2. for *k* = 1 to *K*
    1. *r* = Random_Configuration( ) #Not in obstacle
    2. *n* = Nearest Vertex(*r*, *G*) #Get's the closest vertex
    3. *w* = New_Configuration(*n*, *r*, *d*) #Move from *n* towards *r* by distance *d*
    4. Add *w* to *G*
    5. Add edge from *n* to *w* to *G*
3. Return *G*

Rapidly-Exploring Random Trees

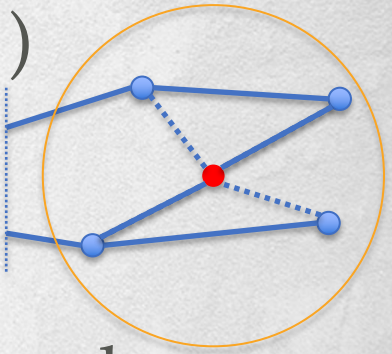
Extensions/Additions

- Simply use random point if it is close enough to nearest point
- With a small probability, sample the goal configuration
  - This makes the tree grow greedily towards the goal
- New configuration can involve constraints
  - Ex: Car: which steering angle from neighbor configuration can get me closest to the random configuration (this involves a small search)

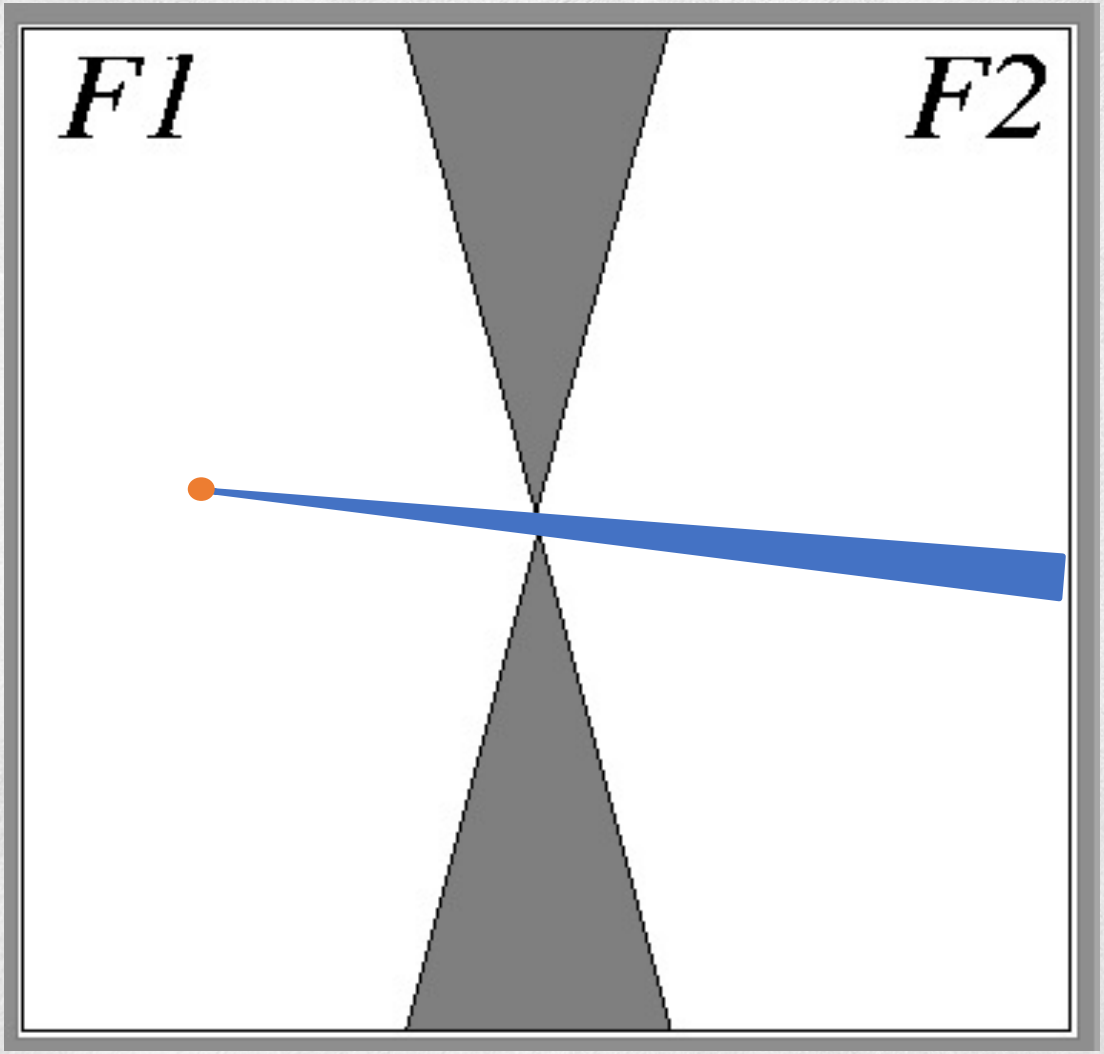
RRT*

Similar to RRT, but to find an optimal path.

- ChooseParent
  - Choose the vertex that makes the shortest path among near vertices
  - x_p = argmin(cost(x) + cost($x \rightarrow x_{new}$))
  - Connect $x_p \rightarrow x_{new}$
- Rewire
  - If near vertices have shorter path through x_{new},
  - Connect $x_{new} \rightarrow x_{near}$

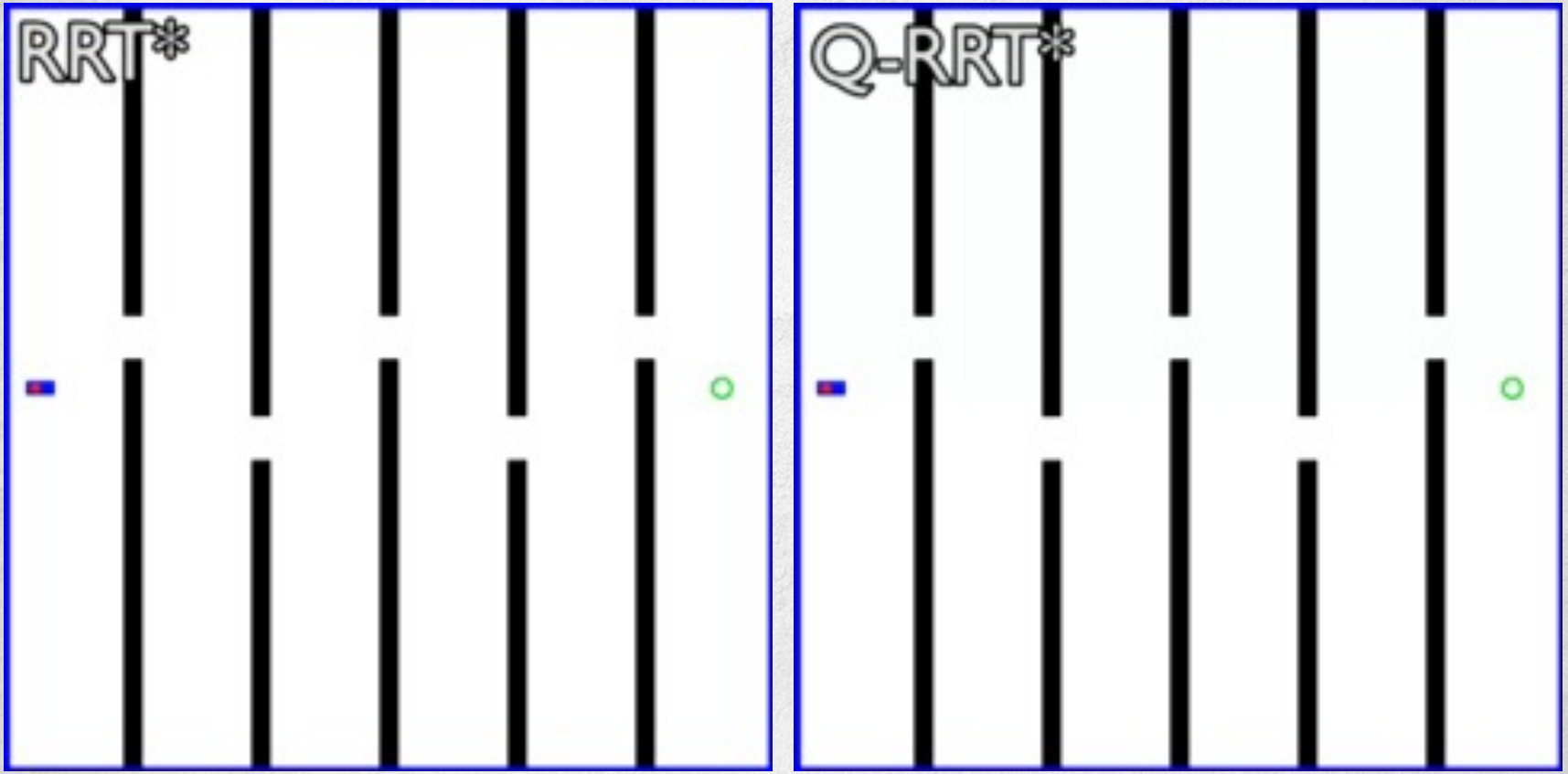


Narrow Passage Issue



Q-RRT*

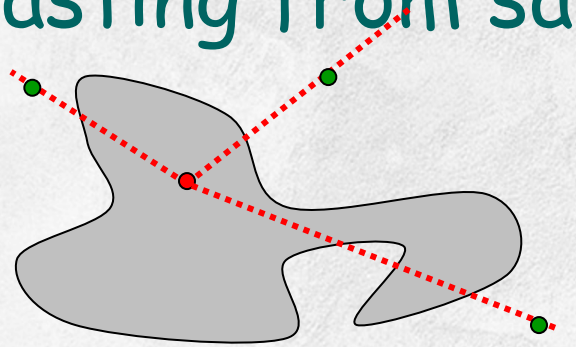
Increased convergence rate + solves narrow passage problem by reusing failed samples



Jeong et al. (2019). Quick-RRT: Triangular inequality-based implementation of RRT**

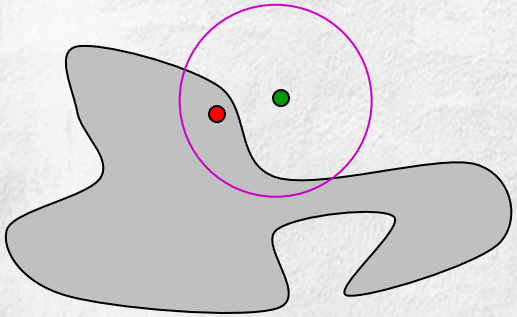
Obstacle-Sensitive Strategies

- Ray casting from samples in obstacles

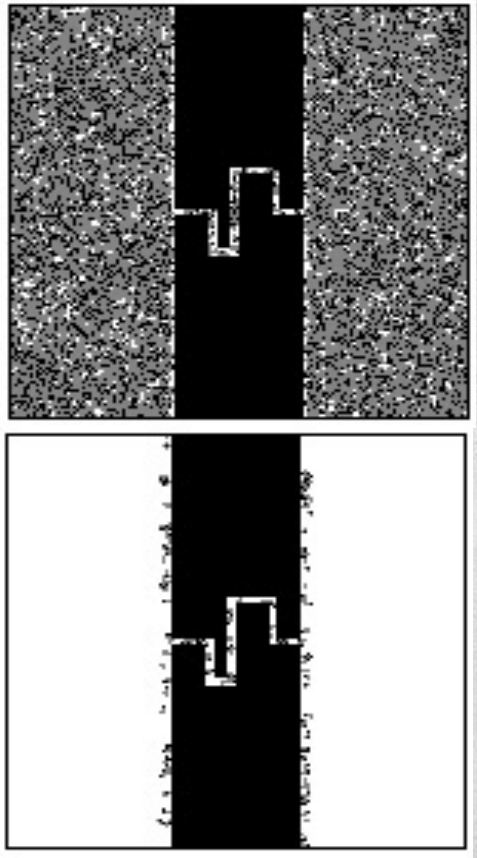


[Amato, Overmars]

- Gaussian sampling



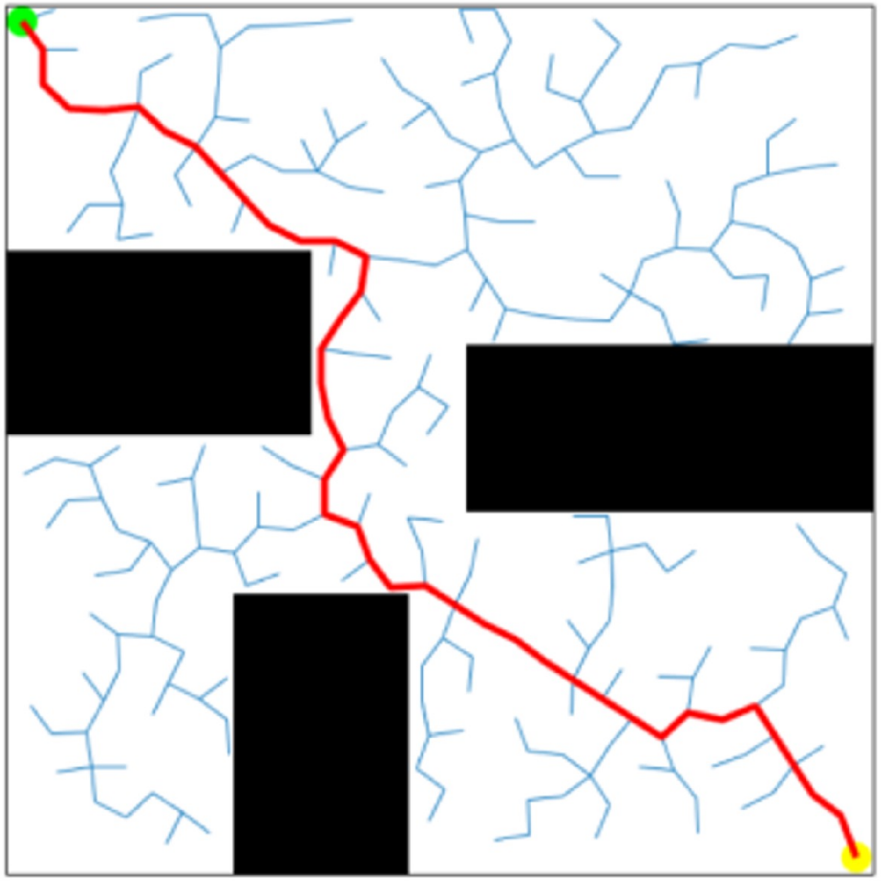
[Boor, Overmars, van der Stappen, 99]



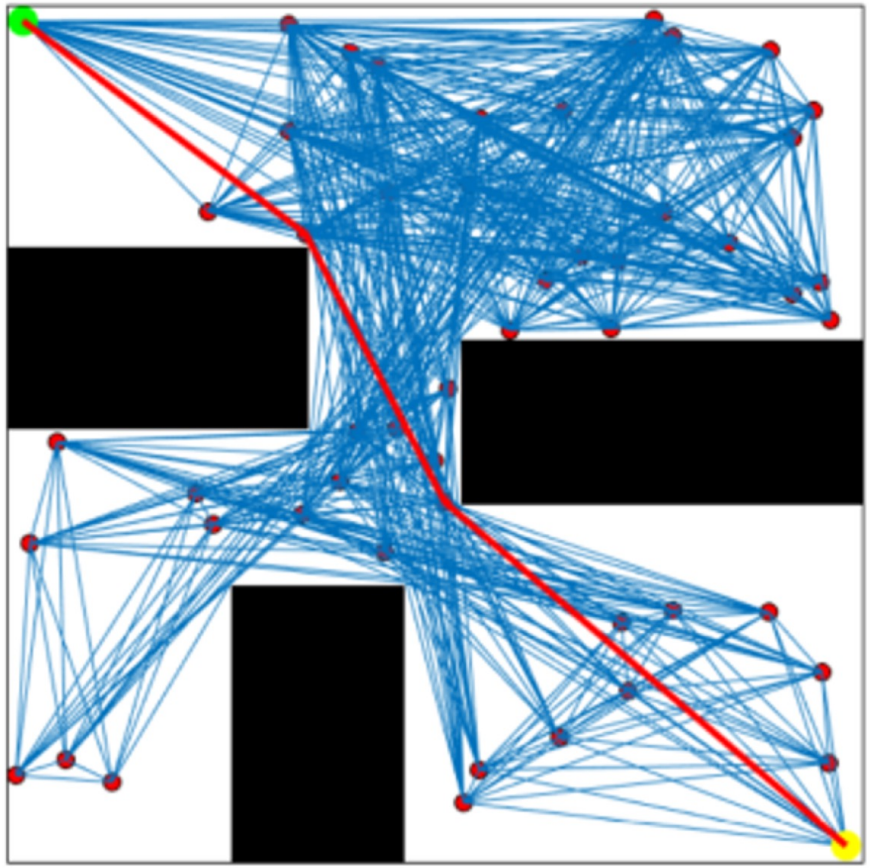
Randomized Planning

- Advantages
  - Easy to get paths with kinematic constraints
  - High responsiveness: get a rough initial path, keep improving it.
  - It guarantees
    - to find a path if exists → (probabilistically) complete
    - to find an optimal path → (asymptotically) optimal
- Disadvantages
  - Nondeterministic time to solve a problem
  - Difficult to solve narrow passages problems

Analysis



(**a**) Path exploring based on RRT



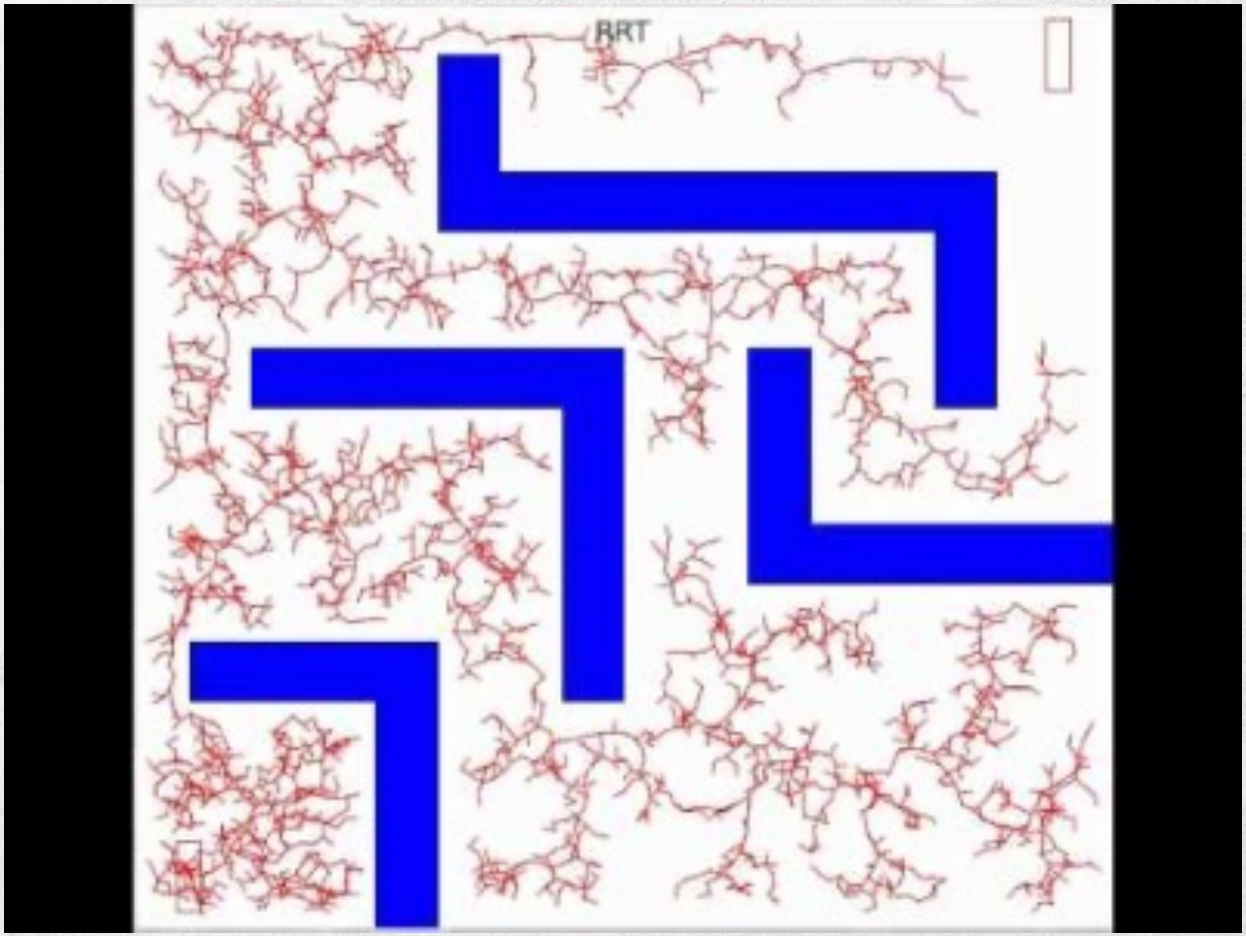
(**b**) Path exploring based on PRM

PRM vs RRT

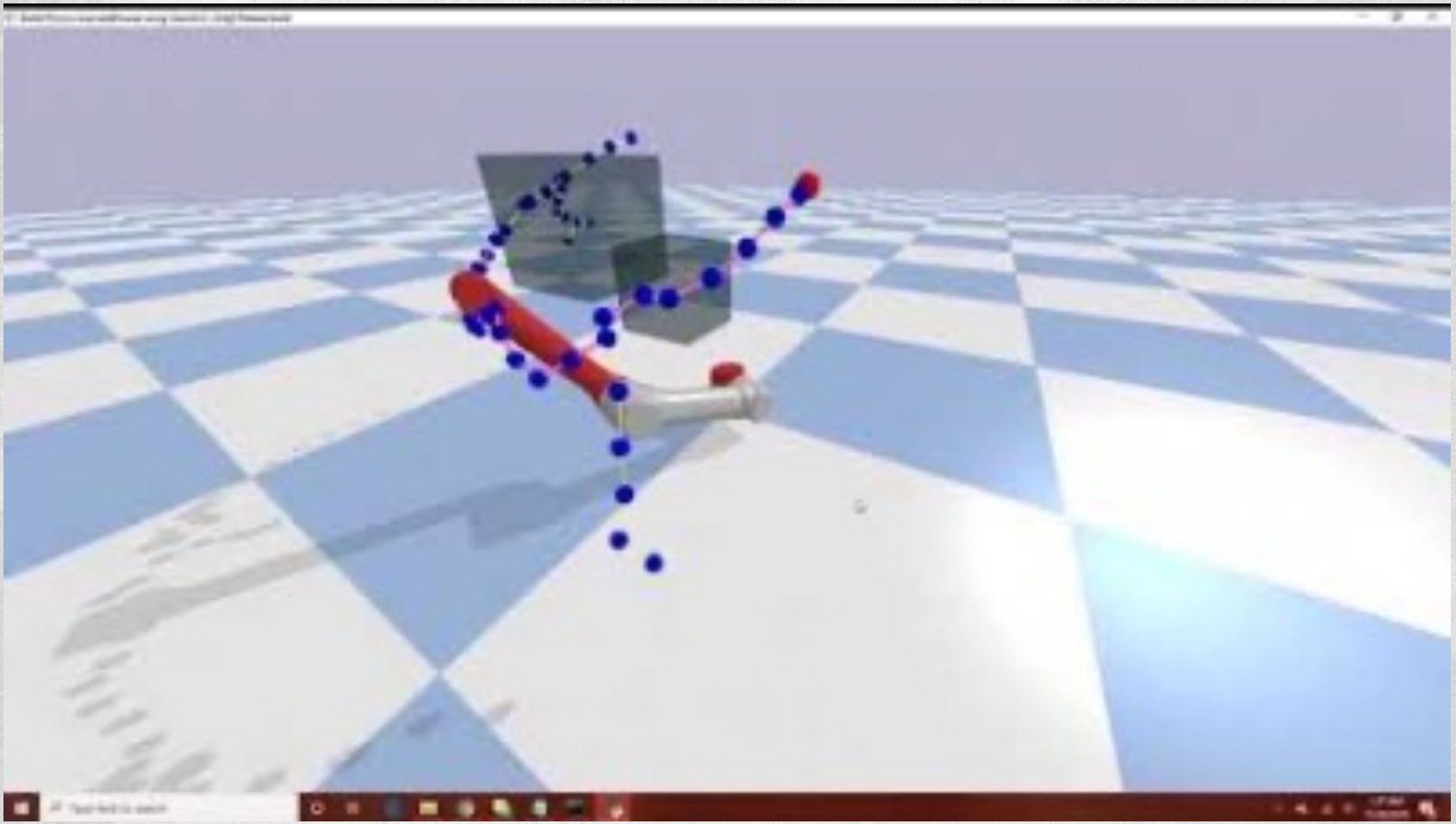
	Multi-query	**Single-query**
Phases	1. Road construction 2. Searching	Roadmap construction (possible to rewire) and searching online
Typical algorithm	Probabilistic Roadmaps (PRM and PRM*)	Rapidly Exploring Random Trees (RRT and RRT*)
Pros	Fast Searching	No preprocessing
Cons	Can not deal with environment changes	No memory

Examples

RRT for Piano Moving

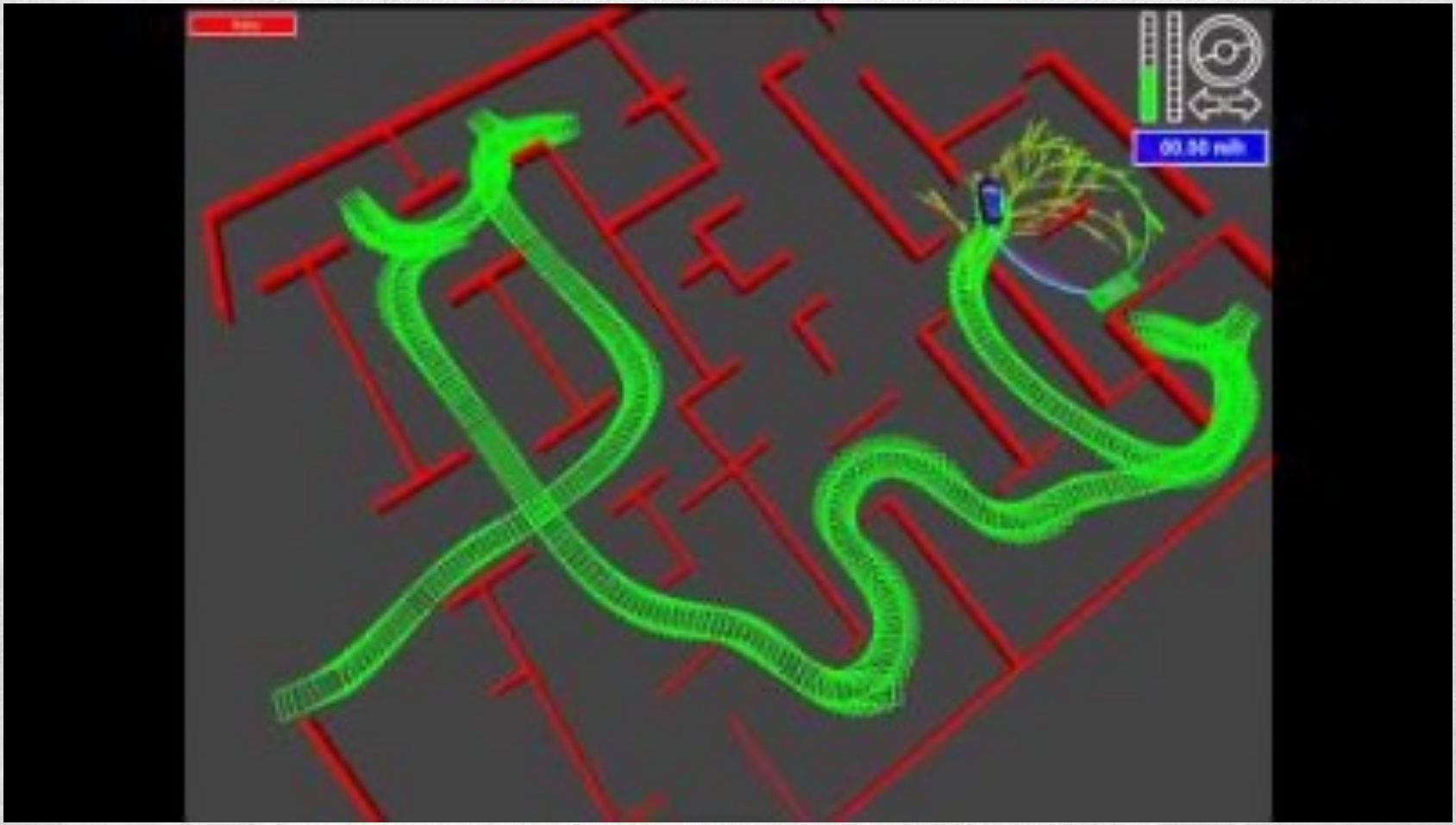


RRT for Robot Arm



RRT for Autonomous Vehicle

RRT for Unknown Environments



References

1. https://en.wikipedia.org/wiki/Probabilistic_roadmap
2. https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree
3. https://www.youtube.com/watch?v=Ob3BIJkQJEw
4. https://www.sciencedirect.com/science/article/abs/pii/S0957417419300326