USC Viterbi
School of Engineering

Sampling-Based Motion Planning: Single-Query Planners

CSCI 545 Introduction to Robotics
Instructor: Stefanos Nikolaidis

Samping-Based Methods

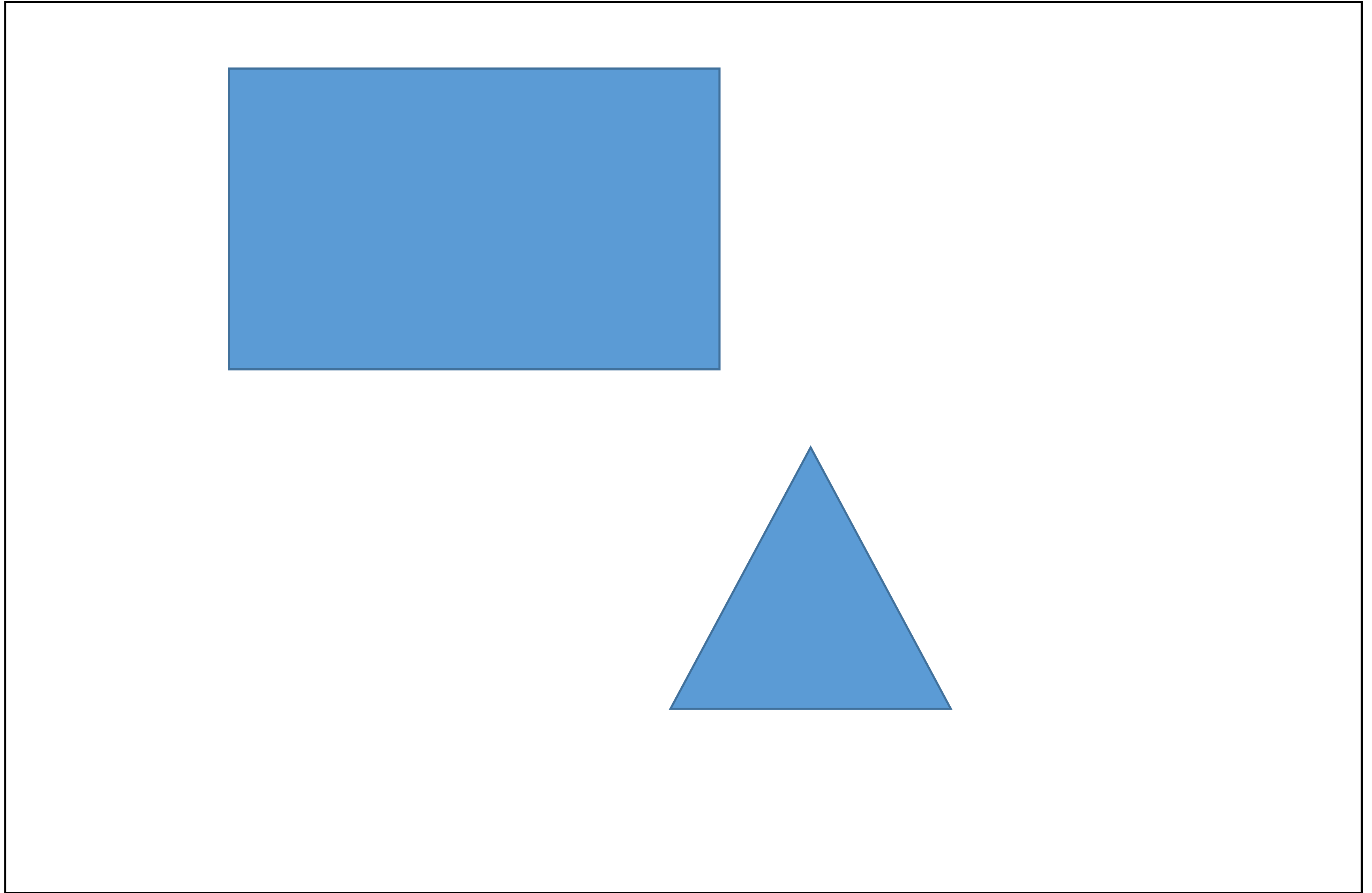
- Multi-Query Planning Algorithms

- Single-Query Planning Algorithms

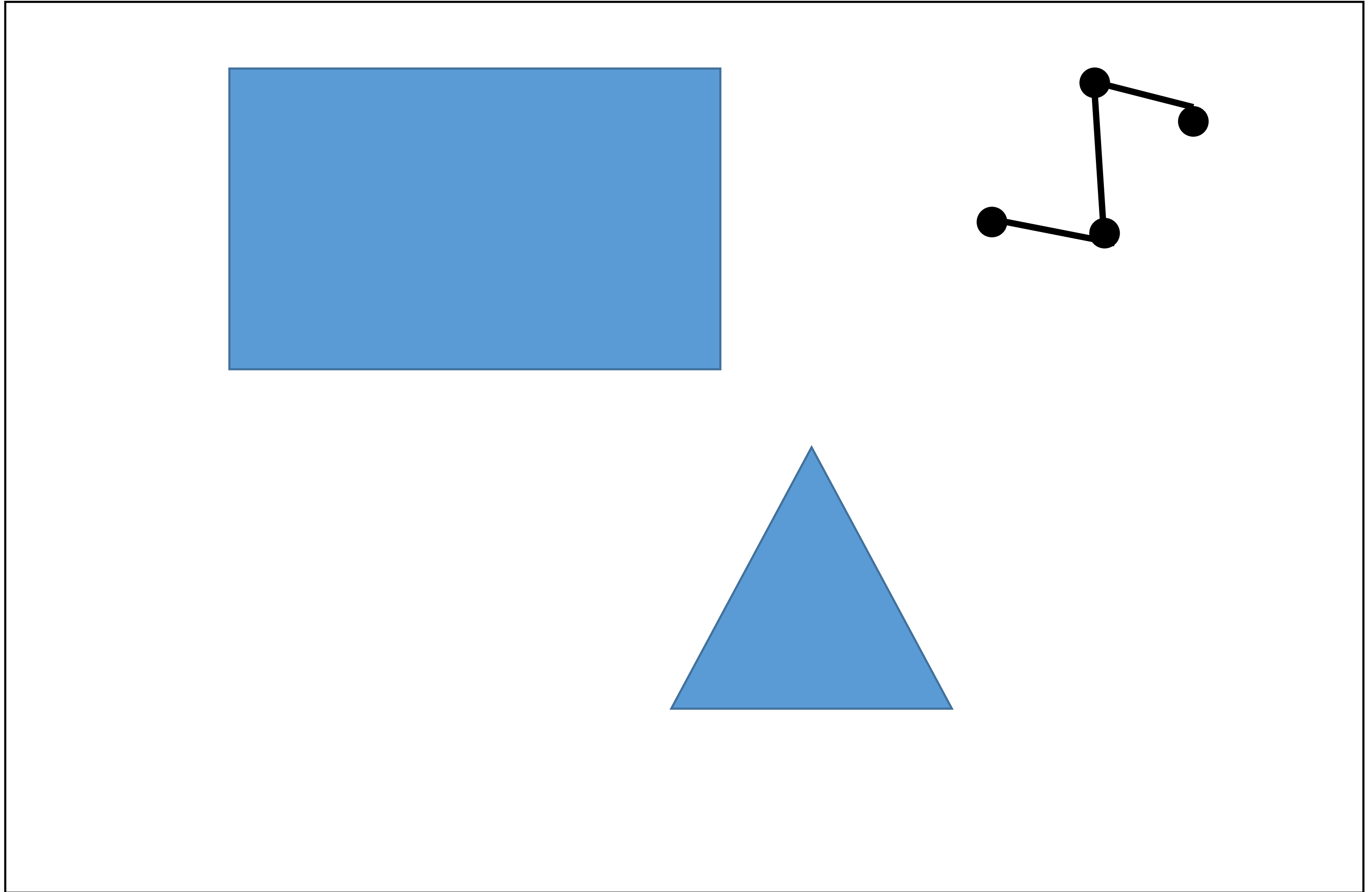
Samping-Based Methods

- Multi-Query Planning Algorithms
  - Cache roadmap for multiple queries
  - Environment is static
- Single-Query Planning Algorithms
  - Recompute roadmap for each query
  - Environment can change across queries (not within)

Rapidly-Exploring Random Trees (RRTs)



Rapidly-Exploring Random Trees (RRTs)



Algorithm 10 Build RRT Algorithm

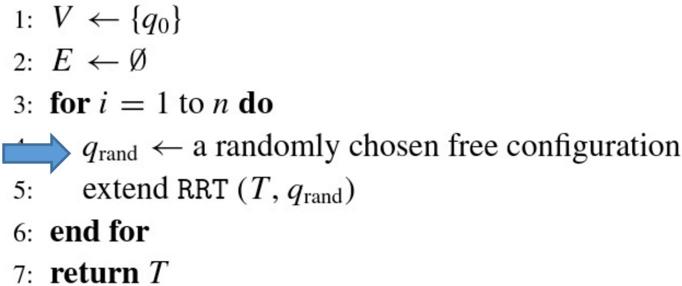
Input:

q_0: the configuration where the tree is rooted

n : the number of attempts to expand the tree

Output:

A tree $T = (V, E)$ that is rooted at q_0 and has $\leq n$ configurations



1: $V \leftarrow \{q_0\}$
2: $E \leftarrow \emptyset$
3: **for** $i = 1$ to n **do**
4: $\quad q_{\text{rand}} \leftarrow$ a randomly chosen free configuration
5: $\quad$ extend `RRT` (T, q_{rand})
6: **end for**
7: **return** T

Algorithm 11 Extend RRT Algorithm

Input:

$T = (V, E)$: an `RRT`

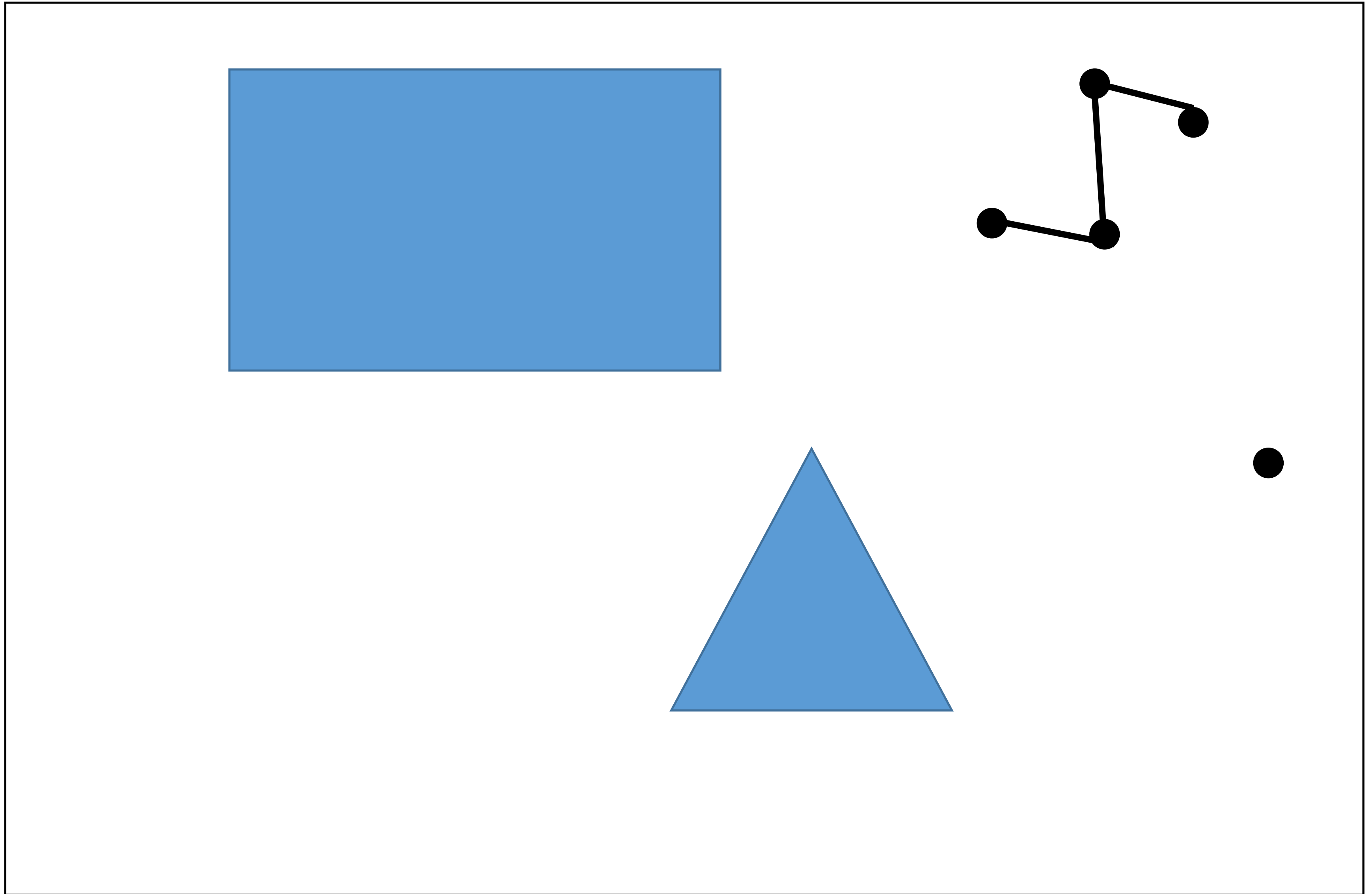
q: a configuration toward which the tree T is grown

Output:

A new configuration q_{new} toward q, or `NIL` in case of failure

1: $q_{\text{near}} \leftarrow$ closest neighbor of q in T
2: $q_{\text{new}} \leftarrow$ progress q_{near} by `step_size` along the straight line in $\mathcal{Q}$ between q_{near} and q_{rand}
3: **if** q_{new} is collision-free **then**
4: $\quad V \leftarrow V \cup \{q_{\text{new}}\}$
5: $\quad E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$
6: $\quad$ **return** q_{new}
7: **end if**
8: **return** `NIL`

Rapidly-Exploring Random Trees (RRTs)



Rapidly-Exploring Random Trees (RRTs)

Algorithm 10 Build RRT Algorithm

Input:

q_0: the configuration where the tree is rooted

n : the number of attempts to expand the tree

Output:

A tree $T = (V, E)$ that is rooted at q_0 and has $\leq n$ configurations

1: $V \leftarrow \{q_0\}$
2: $E \leftarrow \emptyset$
3: **for** $i = 1$ to n **do**
4: $\quad q_{\text{rand}} \leftarrow$ a randomly chosen free configuration
5: $\quad$ extend `RRT` (T, q_{rand})
6: **end for**
7: **return** T

Algorithm 11 Extend RRT Algorithm

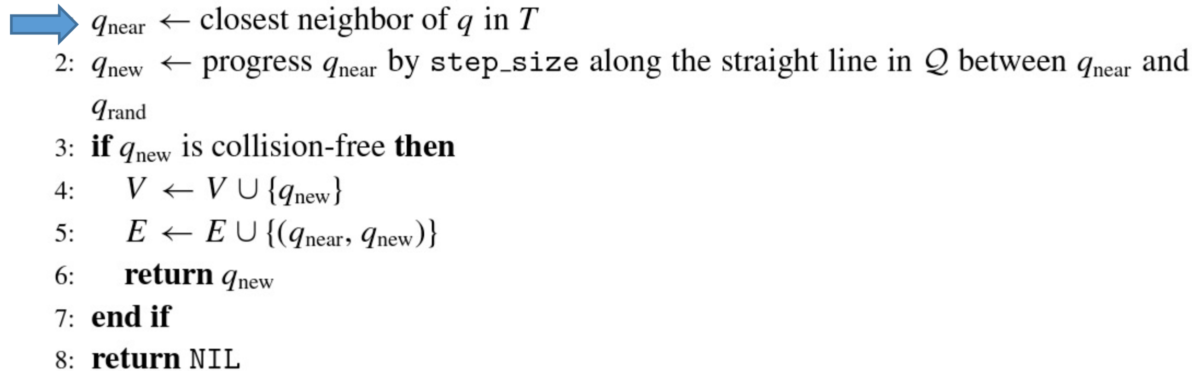
Input:

$T = (V, E)$: an `RRT`

q: a configuration toward which the tree T is grown

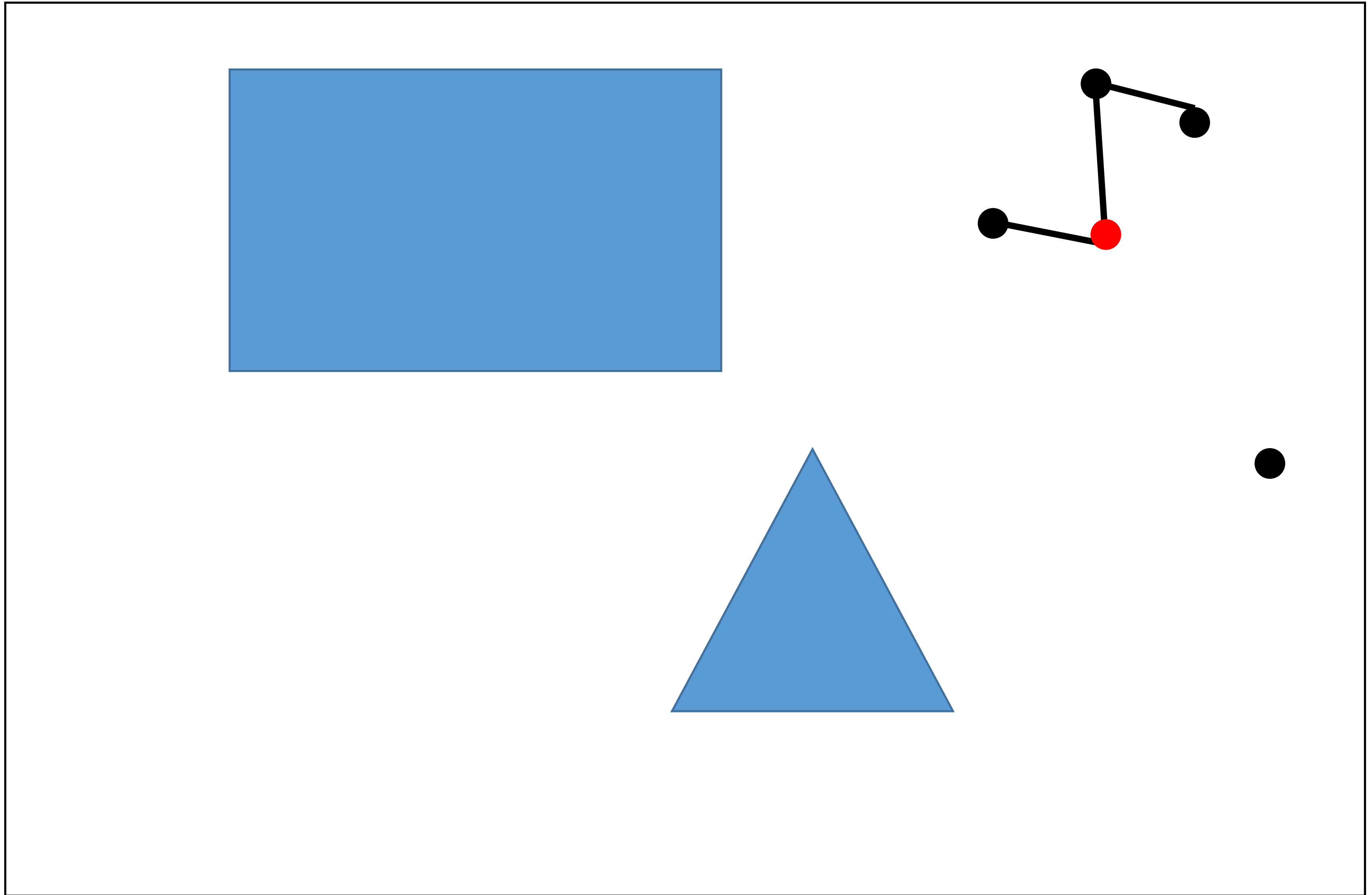
Output:

A new configuration q_{new} toward q, or `NIL` in case of failure



$q_{\text{near}} \leftarrow$ closest neighbor of q in T
2: $q_{\text{new}} \leftarrow$ progress q_{near} by `step_size` along the straight line in $\mathcal{Q}$ between q_{near} and q_{rand}
3: **if** q_{new} is collision-free **then**
4: $\quad V \leftarrow V \cup \{q_{\text{new}}\}$
5: $\quad E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$
6: $\quad$ **return** q_{new}
7: **end if**
8: **return** `NIL`

Rapidly-Exploring Random Trees (RRTs)



Algorithm 10 Build `RRT` Algorithm

Input:

q_0: the configuration where the tree is rooted

n : the number of attempts to expand the tree

Output:

A tree $T = (V, E)$ that is rooted at q_0 and has $\leq n$ configurations

1: $V \leftarrow \{q_0\}$
2: $E \leftarrow \emptyset$
3: **for** $i = 1$ to n **do**
4: $\quad q_{\text{rand}} \leftarrow$ a randomly chosen free configuration
5: $\quad$ extend `RRT` (T, q_{rand})
6: **end for**
7: **return** T

Algorithm 11 Extend `RRT` Algorithm

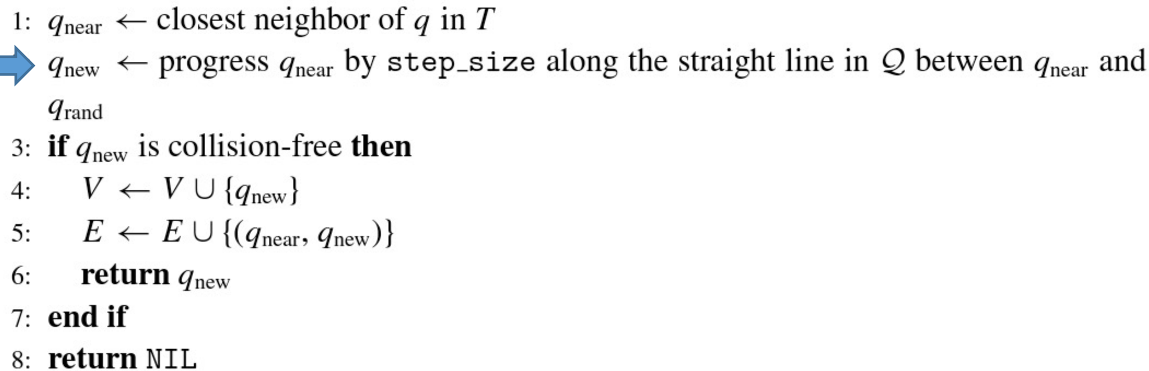
Input:

$T = (V, E)$: an `RRT`

q: a configuration toward which the tree T is grown

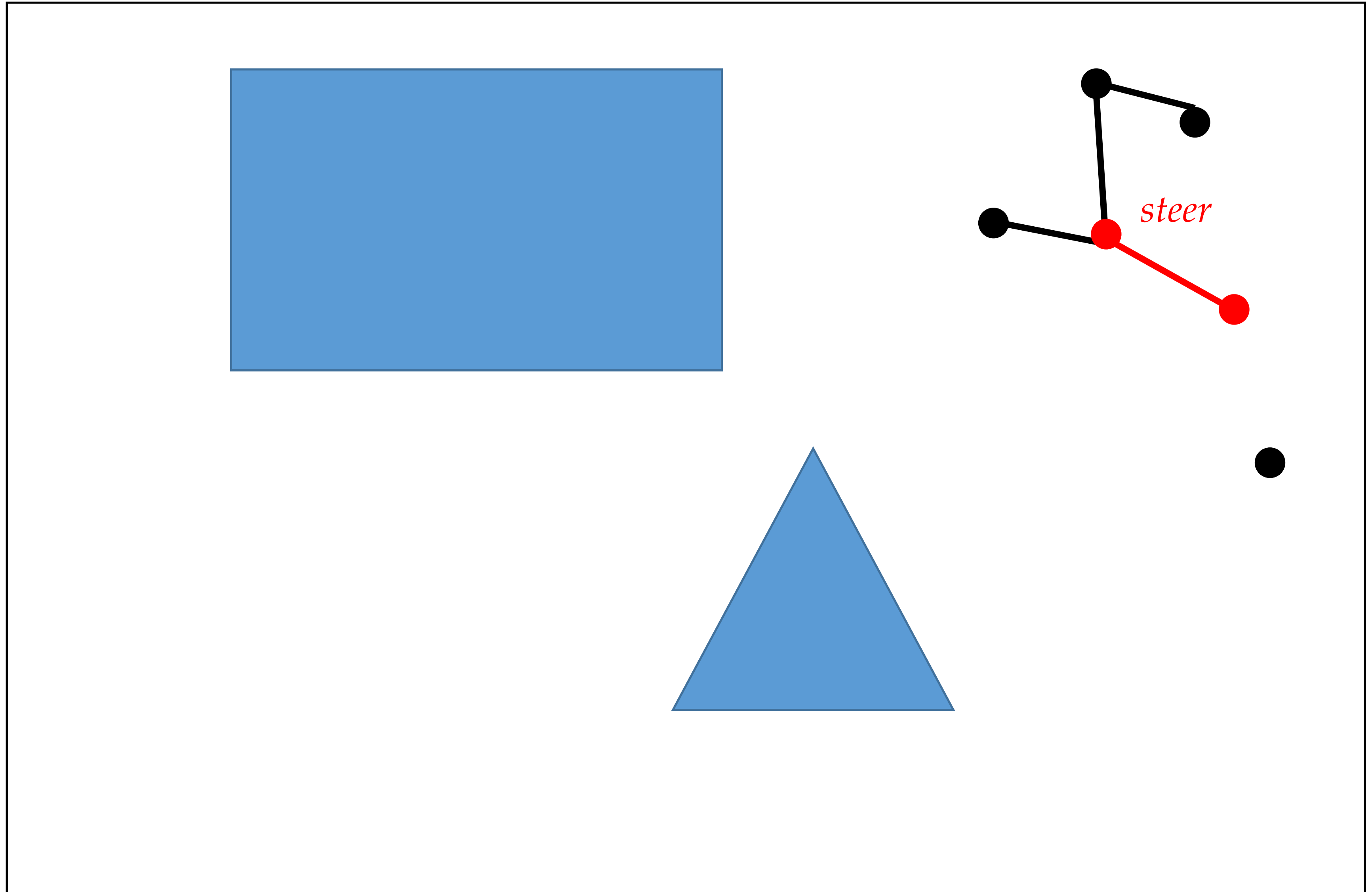
Output:

A new configuration q_{new} toward q, or `NIL` in case of failure



1: $q_{\text{near}} \leftarrow$ closest neighbor of q in T
$q_{\text{new}} \leftarrow$ progress q_{near} by `step_size` along the straight line in $\mathcal{Q}$ between q_{near} and q_{rand}
3: **if** q_{new} is collision-free **then**
4: $\quad V \leftarrow V \cup \{q_{\text{new}}\}$
5: $\quad E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$
6: $\quad$ **return** q_{new}
7: **end if**
8: **return** `NIL`

Rapidly-Exploring Random Trees (RRTs)



Algorithm 10 Build `RRT` Algorithm

Input:

q_0: the configuration where the tree is rooted

n : the number of attempts to expand the tree

Output:

A tree $T = (V, E)$ that is rooted at q_0 and has $\leq n$ configurations

1: $V \leftarrow \{q_0\}$
2: $E \leftarrow \emptyset$
3: **for** $i = 1$ to n **do**
4: $q_{\text{rand}} \leftarrow$ a randomly chosen free configuration
5: extend `RRT` (T, q_{rand})
6: **end for**
7: **return** T

Algorithm 11 Extend `RRT` Algorithm

Input:

$T = (V, E)$: an `RRT`

q: a configuration toward which the tree T is grown

Output:

A new configuration q_{new} toward q, or `NIL` in case of failure

1: $q_{\text{near}} \leftarrow$ closest neighbor of q in T
2: $q_{\text{new}} \leftarrow$ progress q_{near} by `step_size` along the straight line in $\mathcal{Q}$ between q_{near} and q_{rand}
3: **if** q_{new} is collision-free **then**
4: $V \leftarrow V \cup \{q_{\text{new}}\}$
$E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$
6: **return** q_{new}
7: **end if**
8: **return** `NIL`

Rapidly-Exploring Random Trees (RRTs)

- When to stop?
  - when inside a prespecified goal region

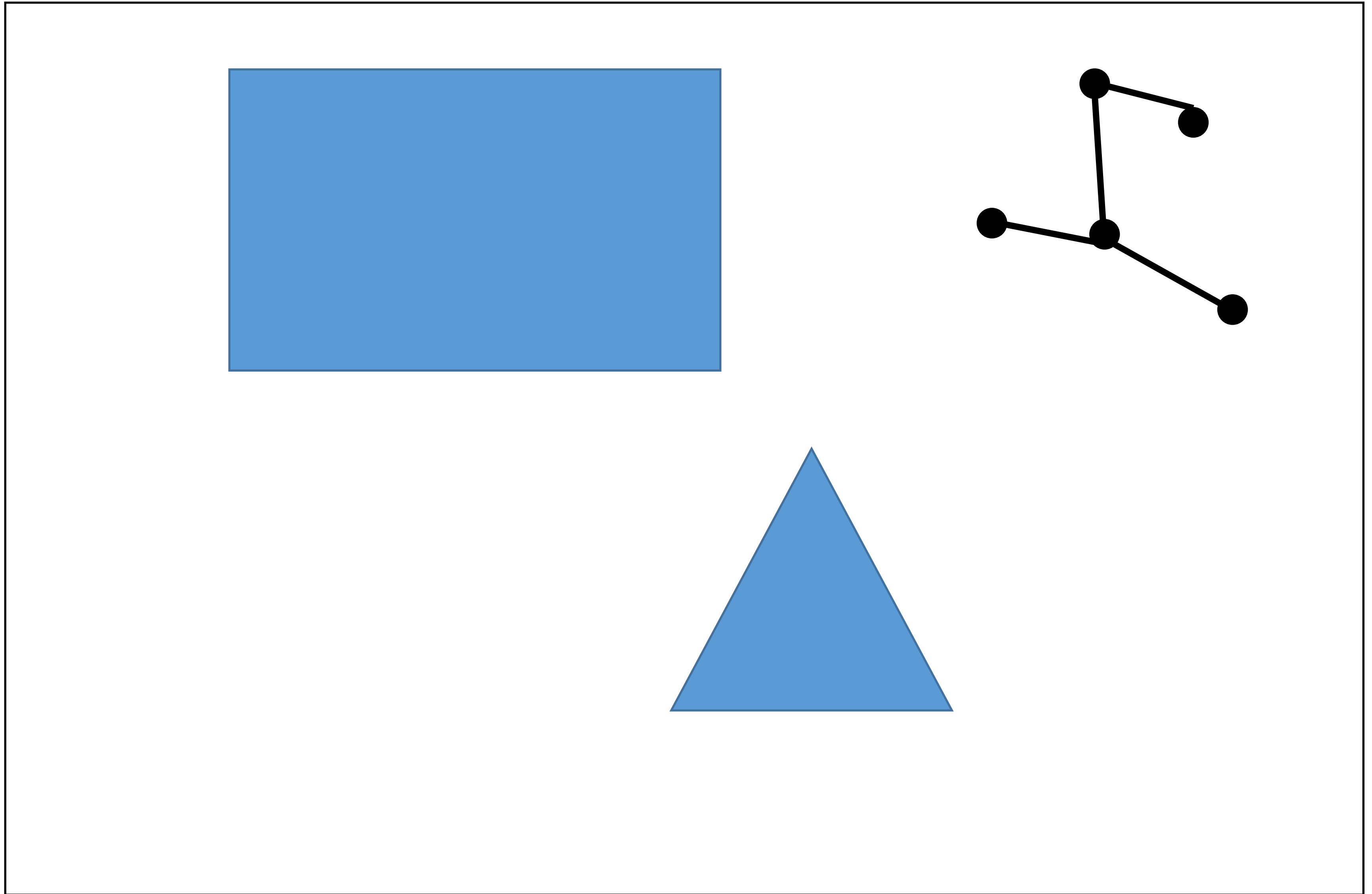
Rapidly-Exploring Random Trees (RRTs)

- When to stop?
  - when inside a prespecified goal region
- How to retrieve the path?

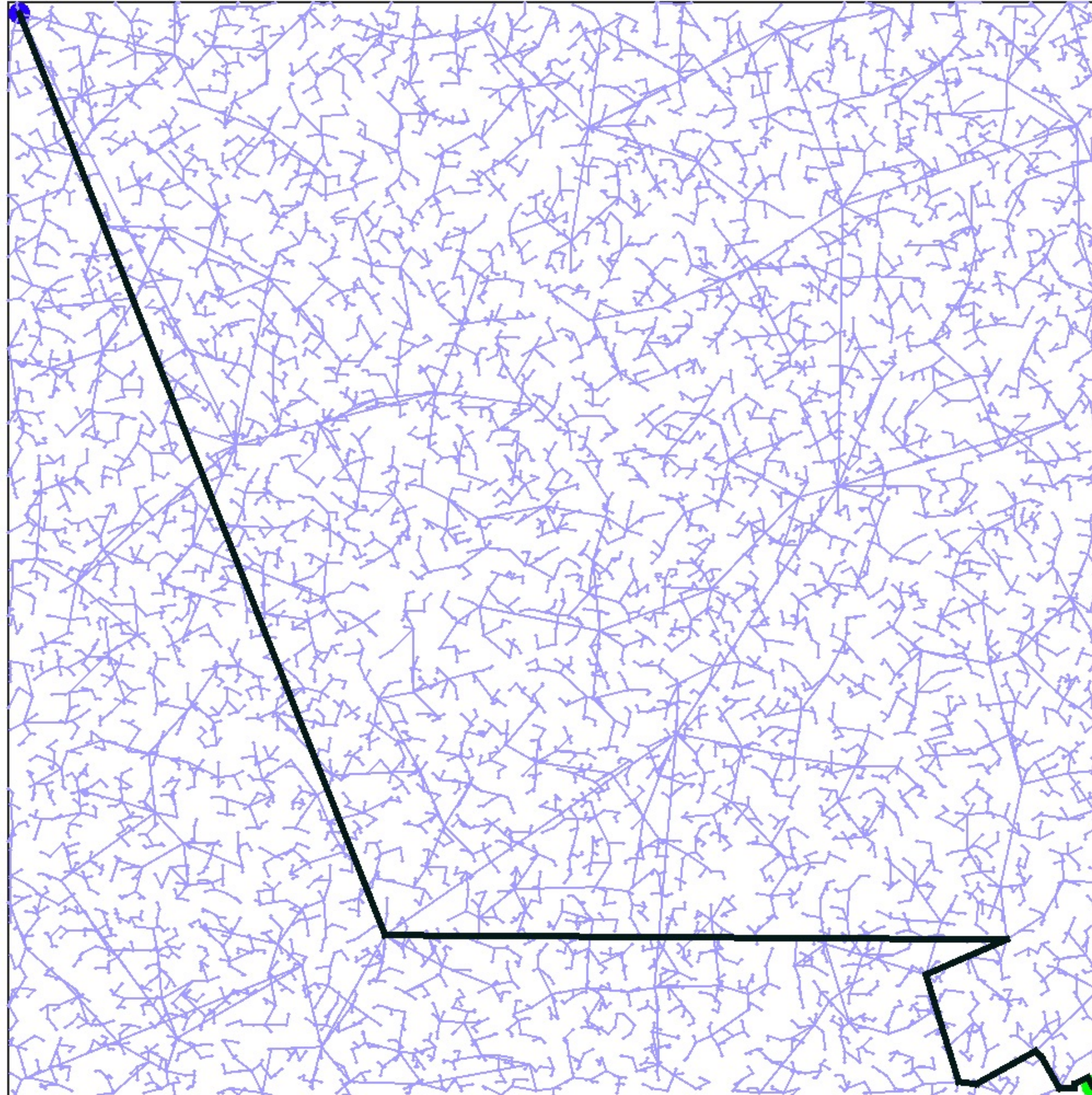
Rapidly-Exploring Random Trees (RRTs)

- When to stop?
  - when inside a prespecified goal region
- How to retrieve the path?
  - retain a pointer to parents

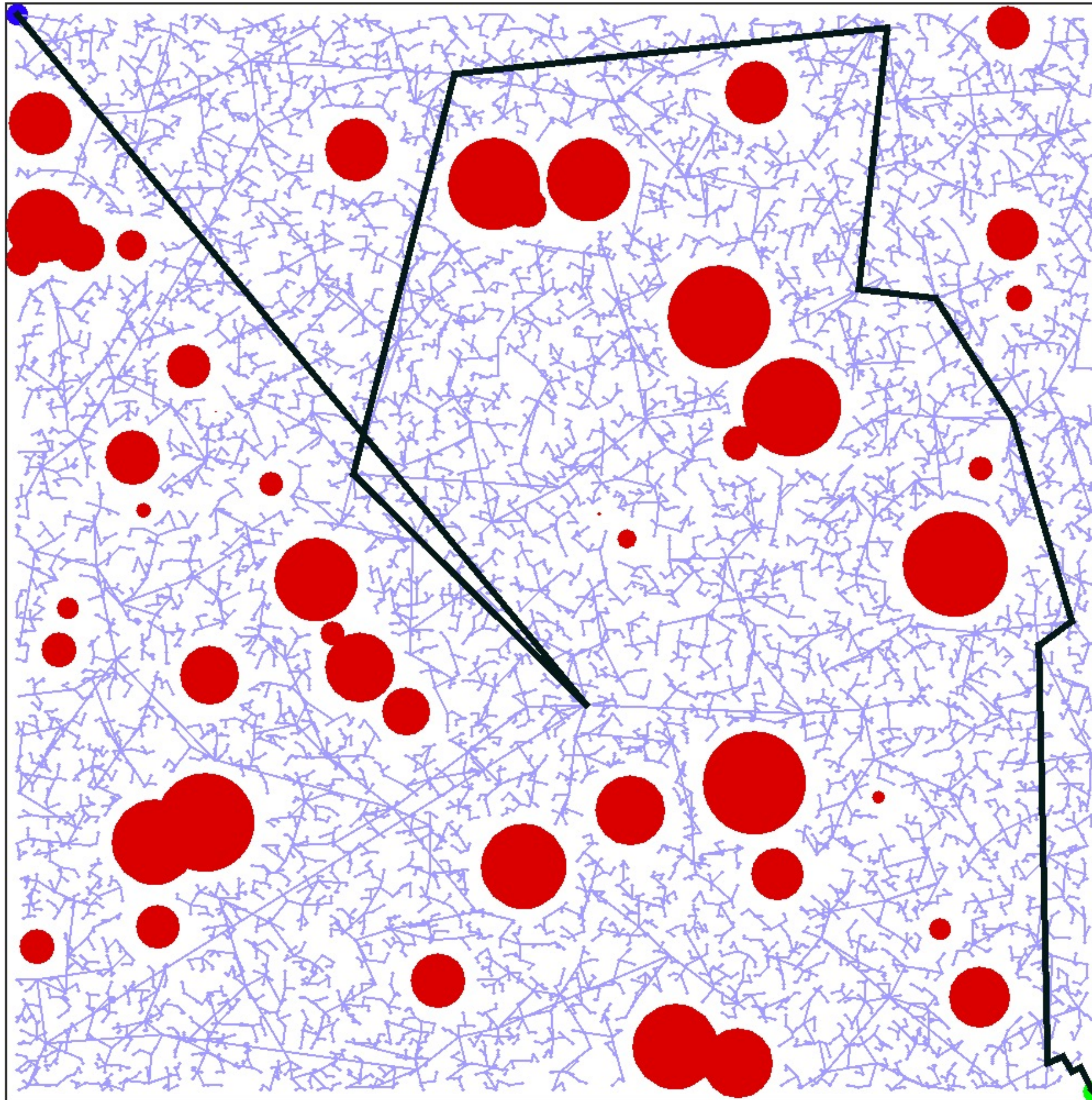
Rapidly-Exploring Random Trees (RRTs)



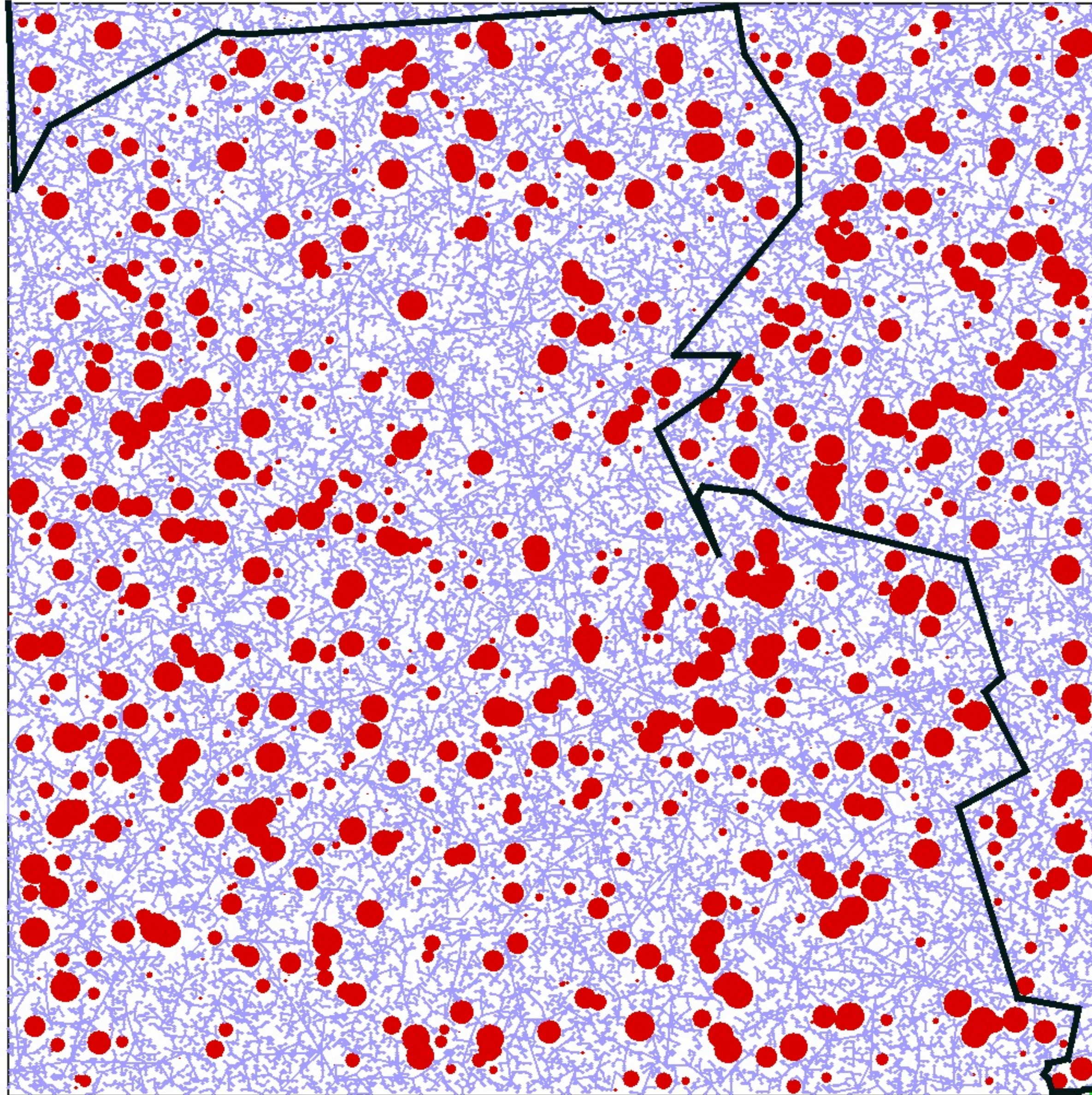
Rapidly-Exploring Random Trees (RRTs)



Rapidly-Exploring Random Trees (RRTs)



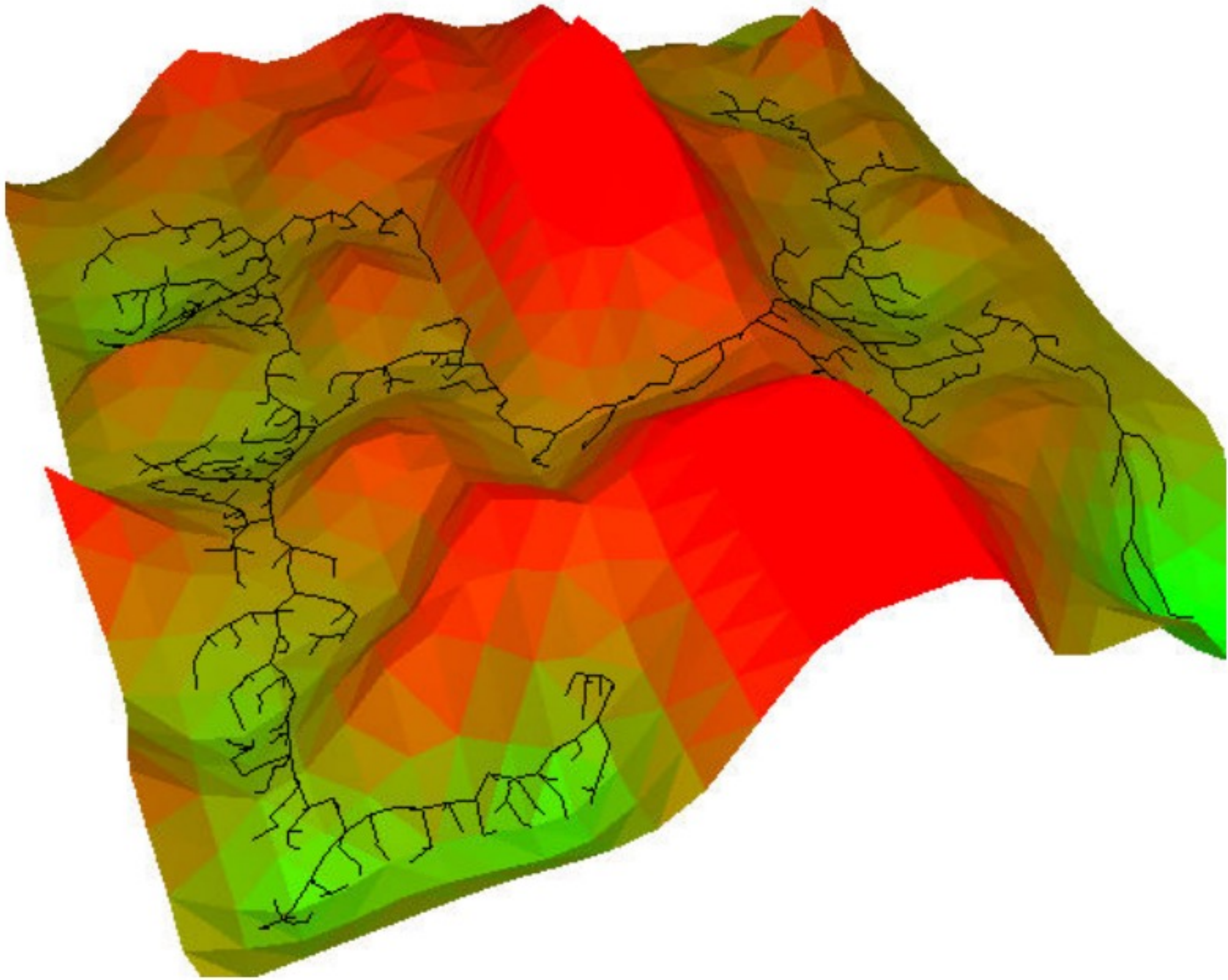
Rapidly-Exploring Random Trees (RRTs)



Intuition behind RRT

- Vertices that are isolated, on the boundary, have higher "visibility" and are more likely to be selected.

Transition-RRTs



Transition-RRTs

```
Algorithm 1: Transition-based RRT

input  : the configuration space CS;
         the cost function c : CS → ℝ*+;
         the root q_init and the goal q_goal;
output : the tree T;
begin
    T ← InitTree(q_init);
    while not StopCondition(T, q_goal) do
        q_rand ← SampleConf(CS) ;
        q_near ← BestNeighbor(q_rand, T);
        if not Extend(T, q_rand, q_near, q_new) Continue;
        if TransitionTest(c(q_near), c(q_new), d_near-new)
        and MinExpandControl(T,q_near,q_rand) then
            AddNewNode(T,q_new);
            AddNewEdge(T,q_near, q_new);
end
```

Transition-RRTs

Algorithm 1: Transition-based RRT

```
input  : the configuration space CS;
         the cost function c : CS → R*+;
         the root q_init and the goal q_goal;
output : the tree T;
begin
    T ← InitTree(q_init);
    while not StopCondition(T, q_goal) do
        q_rand ← SampleConf(CS) ;
        q_near ← BestNeighbor(q_rand, T);
        if not Extend(T, q_rand, q_near, q_new) Continue;
        if TransitionTest(c(q_near), c(q_new), d_near-new)
        and MinExpandControl(T, q_near, q_rand) then
            AddNewNode(T, q_new);
            AddNewEdge(T, q_near, q_new);
end
```

Transition-RRTs

Algorithm 1: Transition-based RRT

input : the configuration space CS;
the cost func[illegible] $CS \rightarrow \mathbb{R}^+$
the root q_{in}[illegible]

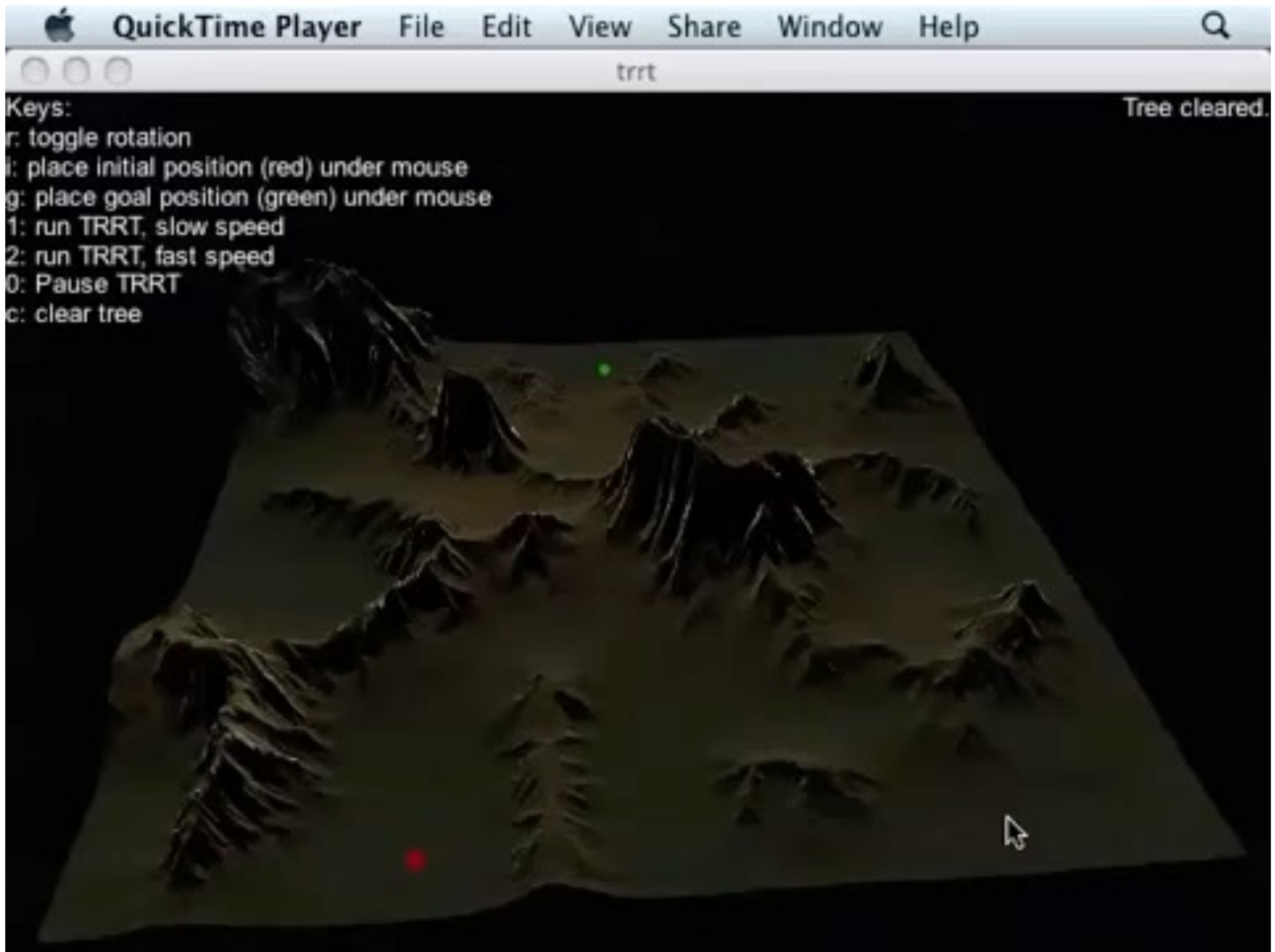
output : the tree $\mathcal{T}$;

begin
- $\mathcal{T} \leftarrow$ `InitTree`(q_{init});
- **while not** `StopCondition`($\mathcal{T}$, q_{goal}) **do**
  - $q_{rand} \leftarrow$ `SampleConf`(CS) ;
  - $q_{near} \leftarrow$ `BestNeighbor`($q_{rand}, \mathcal{T}$);
  - **if not** `Extend`($\mathcal{T}, q_{rand}, q_{near}, q_{new}$) **Continue**;
  - **if** `TransitionTest`($c(q_{near}), c(q_{new}), d_{near-new}$) **and** `MinExpandControl`($\mathcal{T}$,$q_{near}$,$q_{rand}$) **then**
    - `AddNewNode`($\mathcal{T}$,q_{new});
    - `AddNewEdge`($\mathcal{T}$,q_{near}, q_{new});

end

$$p_{ij} = \begin{cases} exp(-\frac{\Delta c^*_{ij}}{K*T}) & \text{if } \Delta c^*_{ij} > 0 \\ 1 & \text{otherwise.} \end{cases}$$

Transition-RRTs



Transition-RRTs

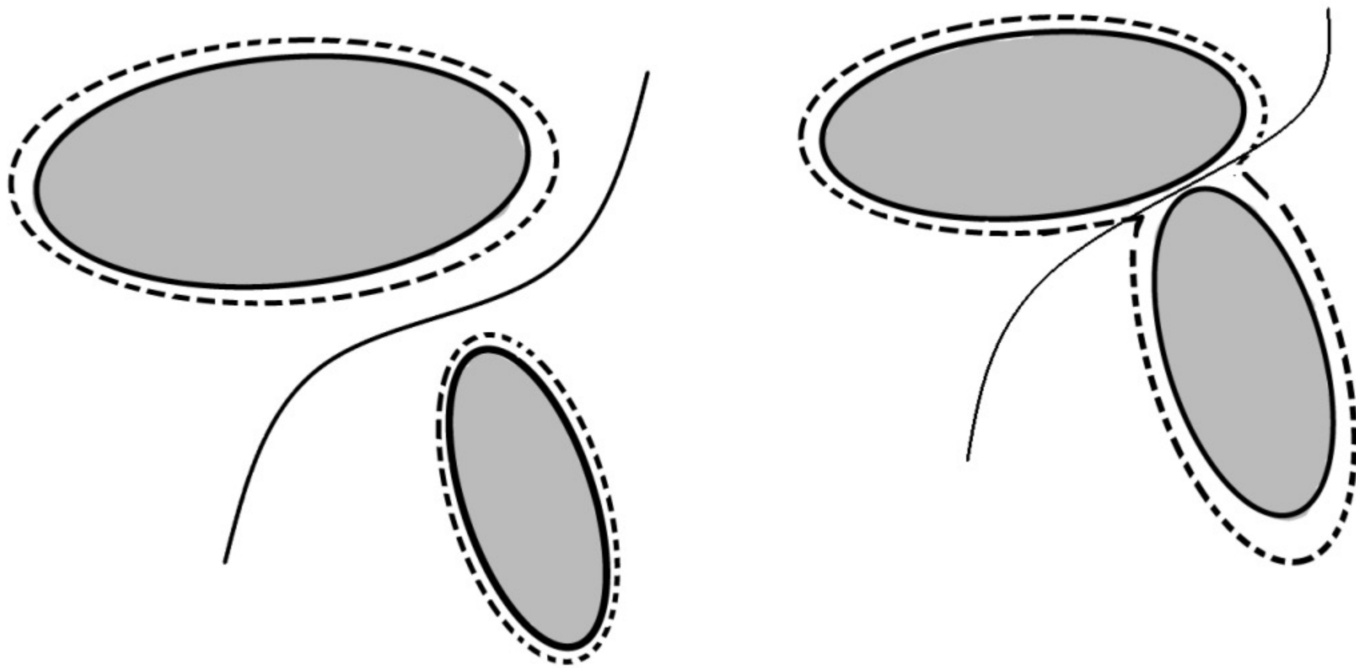
Robotic Lime Picking
by considering leaves as Permeable Obstacles

Heramb Nemlekar*, Ziang Liu*, Suraj Kothawade, Sherdil Niyaz,
Barath Raghavan and Stefanos Nikolaidis
*** equal contribution**

ICAROS Lab
University of Southern California

Definitions

- A robustly feasible planning problem



Completeness

- When is an algorithm *complete*?

Completeness and Optimality

- When is an algorithm *complete*?
  - If there is a solution, the algorithm will find it, otherwise it will report failure
- Probabilistic Completeness: a *weaker* notion of completeness
- Probabilistic Optimality: a *weaker* notion of optimality

Probabilistic Completeness

- An algorithm ALG is probabilistically complete if, for any robustly feasible motion planning problem defined by $P = (Q_{free}, q_s, q_g)$

$$\lim_{N \to \infty} P(ALG \text{ returns a solution to } P) = 1$$

Is RRT probabilistically complete?

Asymptotic Optimality

- We let. Y_N^{RRT} the cost of the best path from running RRT N times and c^* the cost of the optimal path.

Asymptotic Optimality

- We let. Y_N^{RRT} the cost of the best path from running RRT N times and c^* the cost of the optimal path.

$$P\left(\left\{\lim_{N\to\infty} Y_N^{RRT} > c^*\right\}\right) = 1$$

Anytime RRTs

- Find a suboptimal path quickly, then improve over time.
  - Produce an initial path of cost C_s
  - Next run, we stop only if $C'_s < C_s$

Anytime RRTs

- Find a suboptimal path quickly, then improve over time.
  - Produce an initial path of cost C_s
  - Next run, we stop only if $C'_s < C_s$
- Idea 1: keep old trees
  - Might converge to a local minimum
  - Might bias future solutions
- Idea 2: build a new tree from scratch

Anytime RRTs

- Find a suboptimal path quickly, then improve over time.
  - Produce an initial path of cost C_s
  - Next run, we stop only if $C'_s < C_s$

- Idea 1: keep old trees
  - Might converge to a local minimum
  - Might bias future solutions

- Idea 2: build a new tree from scratch

- Better Ideas: Use a heuristic to consider a subset of nodes of previous nodes that lead to an improved solution

Bidirectional RRTs

- Start from both start node and goal node and merge trees

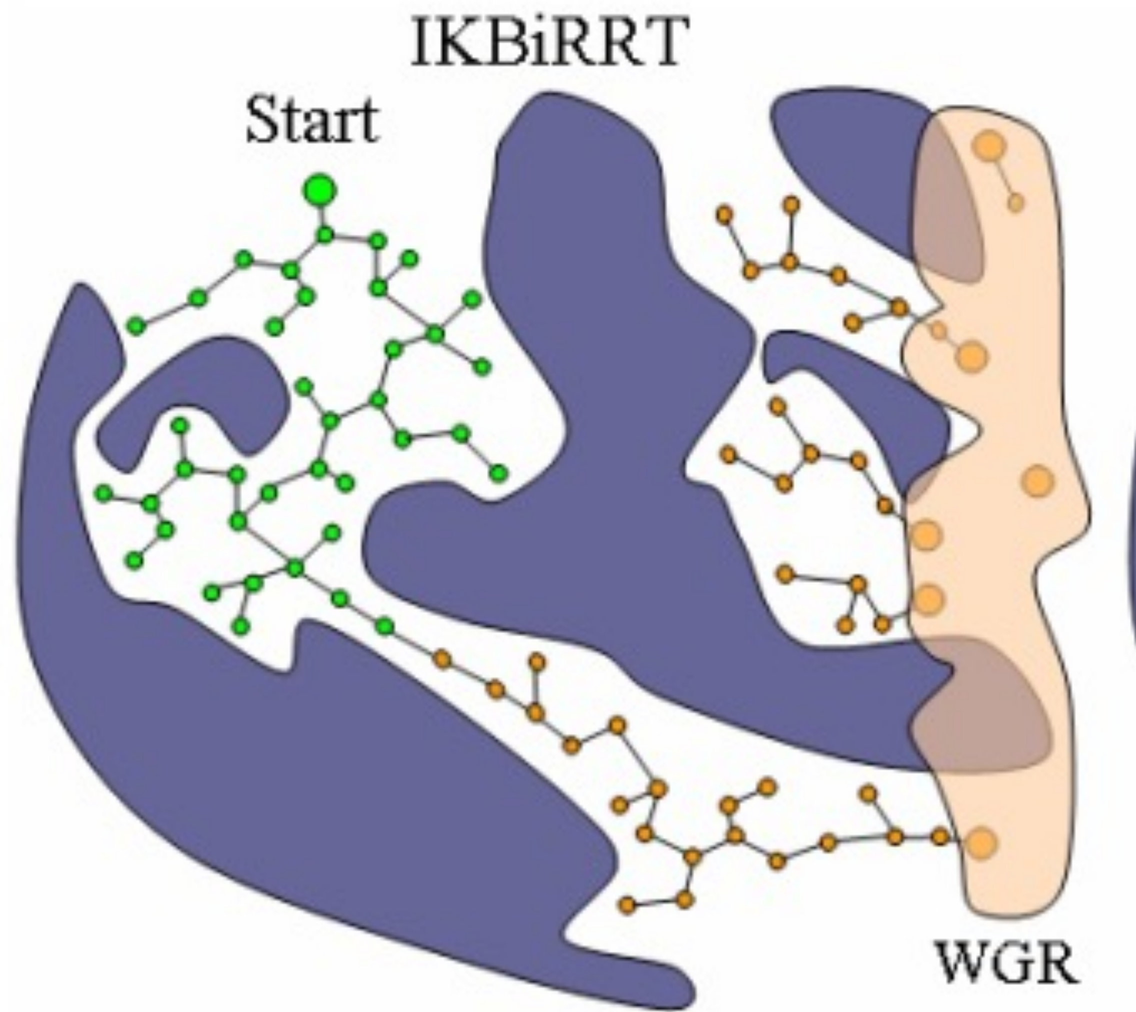
```
hw5 — -bash — 69×24
~ — jupyter-notebook ▸ python    ...bia Courses/Semester 04/Robotics
no last new
no last new
no last new
no last new
no last new
no last new
no last new
no last new
no last new
no last new
no last new
no last new
no last new
no last new
no last new
no last new
no last new
[$
[$ python hw5/rrt_2.py hw5/world_obstacles.txt hw5/sta
[$ python hw5/rrt_2.py hw5/world_obstacles.txt hw5/sta
```

Bidirectional RRTs

Algorithm 1: IKBiRRT(q_s, W)

```
1  T_a.Init(q_s); T_b.Init(NULL);
2  while TimeRemaining() do
3      T_goal = GetBackwardTree(T_a, T_b);
4      if T_goal.size = 0 or rand(0,1) < P_sample then
5          AddIKSolutions(T_goal, W);
6      else
7          q_rand ← RandConfig();
8          q^a_near ← NearestNeighbor(T_a, q_rand);
9          q^a_reached ← Extend(T_a, q^a_near, q_rand);
10         q^b_near ← NearestNeighbor(T_b, q^a_reached);
11         q^b_reached ← Extend(T_b, q^b_near, q^a_reached);
12         if q^a_reached = q^b_reached then
13             P ← ExtractPath(T_a, q^a_reached, T_b, q^b_reached);
14             return SmoothPath(P);
15         else
16             Swap(T_a, T_b);
17         end
18     end
19 end
20 return NULL;
```

Bidirectional RRTs



Shortcutting

- Output of RRTs is almost unusable.

- Simple Shortcutting Algorithm

Shortcutting

- Output of RRTs is almost unusable.

- Simple Shortcutting Algorithm:

  Repeat:

  Select two points randomly
  Attempt to connect them
  If path shortest than previous one, replace path

Special Case: Time-Varying Planning

Time-Varying Planning

- Time-Varying Obstacles O(t)
- State-space?

Time-Varying Planning

- Time-Varying Obstacles O(t)
- State-space: x = (q, t)
- X_obs?

Time-Varying Planning

- Time-Varying Obstacles O(t)
- State-space: x = (q, t)
- $X_{obs} = \{(q, t) \in X | A(q) \cap O(t) \neq \varnothing\}$
- Goal: compute a continuous time-monotonic path $\tau[0, 1] \rightarrow X_{free}$

Time-Varying Planning

- Time-Varying Obstacles O(t)
- State-space: x = (q, t)
- $X_{obs} = \{(q, t) \in X | A(q) \cap O(t) \neq \varnothing\}$
- Goal: compute a continuous time-monotonic path $\tau[0, 1] \rightarrow X_{free}$

$$s_1 \leq s_2 \rightarrow t_1 \leq t_2, (q_1, t_1) = \tau(s_1), (q_2, t_2) = \tau(s_2)$$

Time-Varying Planning Example

- Piecewise-linear motion

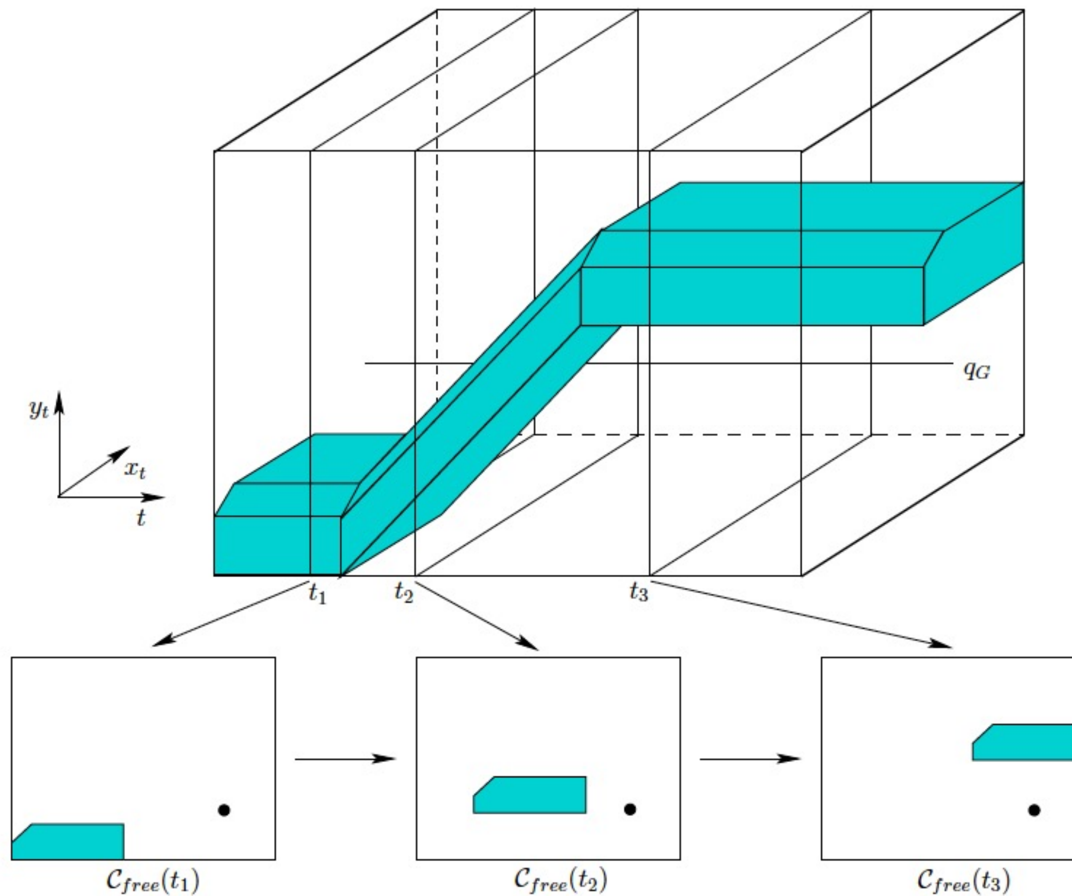


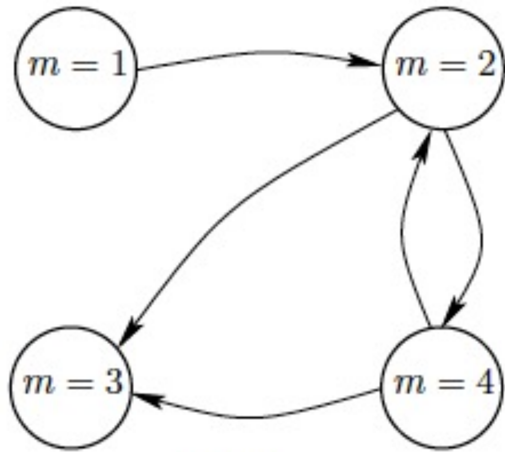
Figure 7.1: A time-varying example with piecewise-linear obstacle motion.

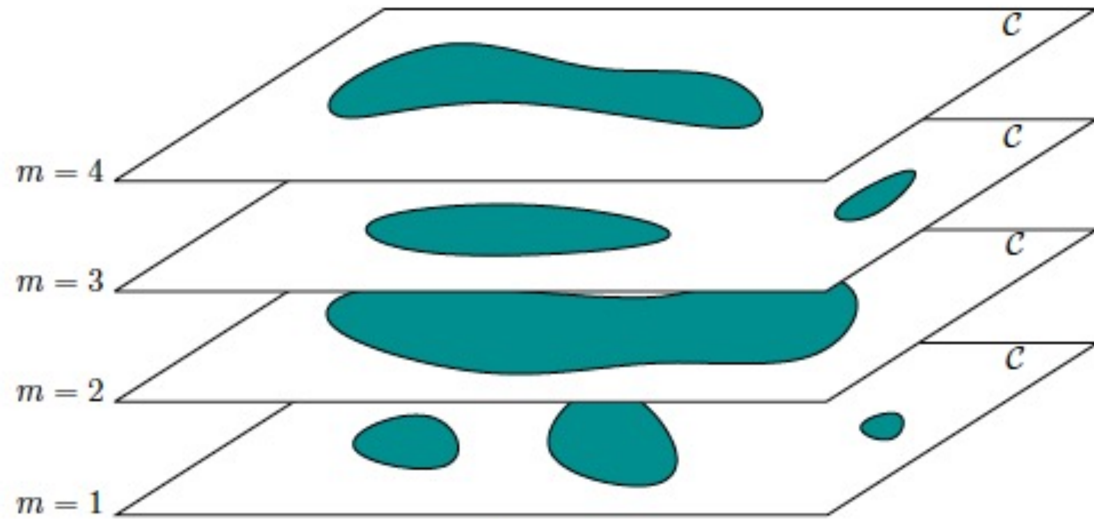
USC Viterbi School of Engineering

Planning in Discrete-Continuous Spaces

CSCI 545 Introduction to Robotics
Instructor: Stefanos Nikolaidis

Modes





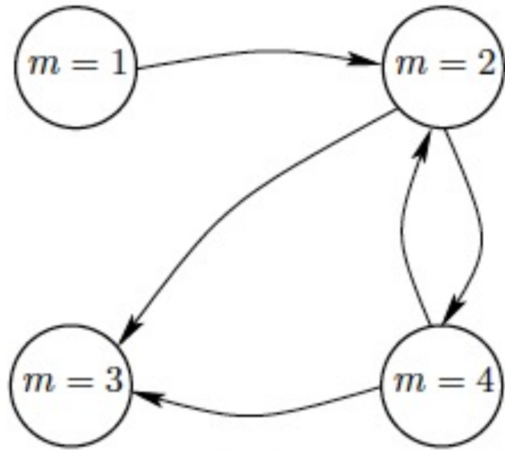
Problem Setting

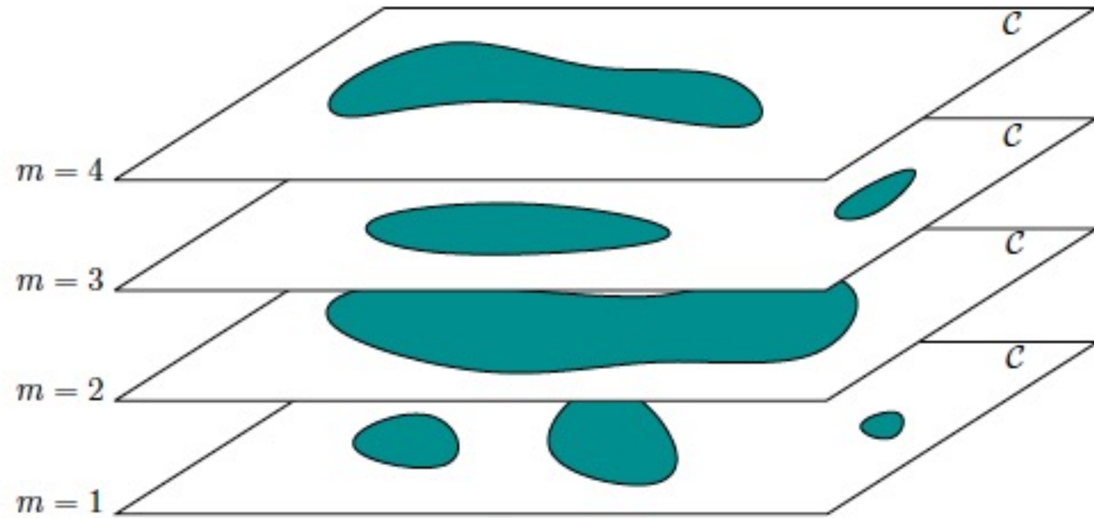
- State space $X = Q \times M$. A state is represented as $x = (q, m)$, where $q \in Q$ and $m \in M$
- An obstacle region $O(m)$ that depends on the mode m
- A robotic arm $A(m)$
- $Q_{obs} = \{(q, m) \in X | A(q, m) \cap O(m) \neq \varnothing\}$
- Each state x has a finite action space $U(x)$

Problem Setting

- Each state x has a finite action space $U(x)$
- A mode transition function $f_m : X \times U \to M$

Modes





Problem Setting

- Each state x has a finite action space $U(x)$
- A mode transition function $f_m : X \times U \to M$
- A state transition function $f(x, u) = (q, f_m(x, u))$
- An initial state $x_s \in X_{free}$
- A goal region $X_G \in X_{free}$

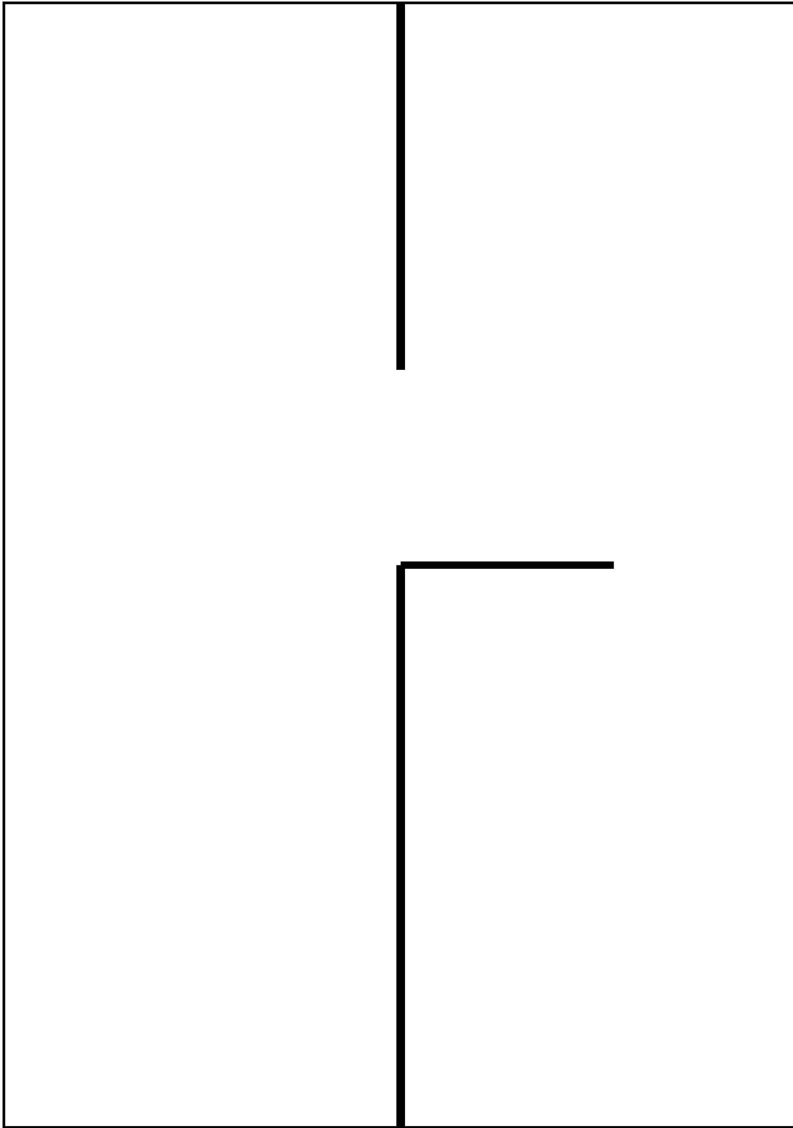
Goal

- Compute a continuous path: $\tau : [0, 1] \rightarrow Q_{free}$

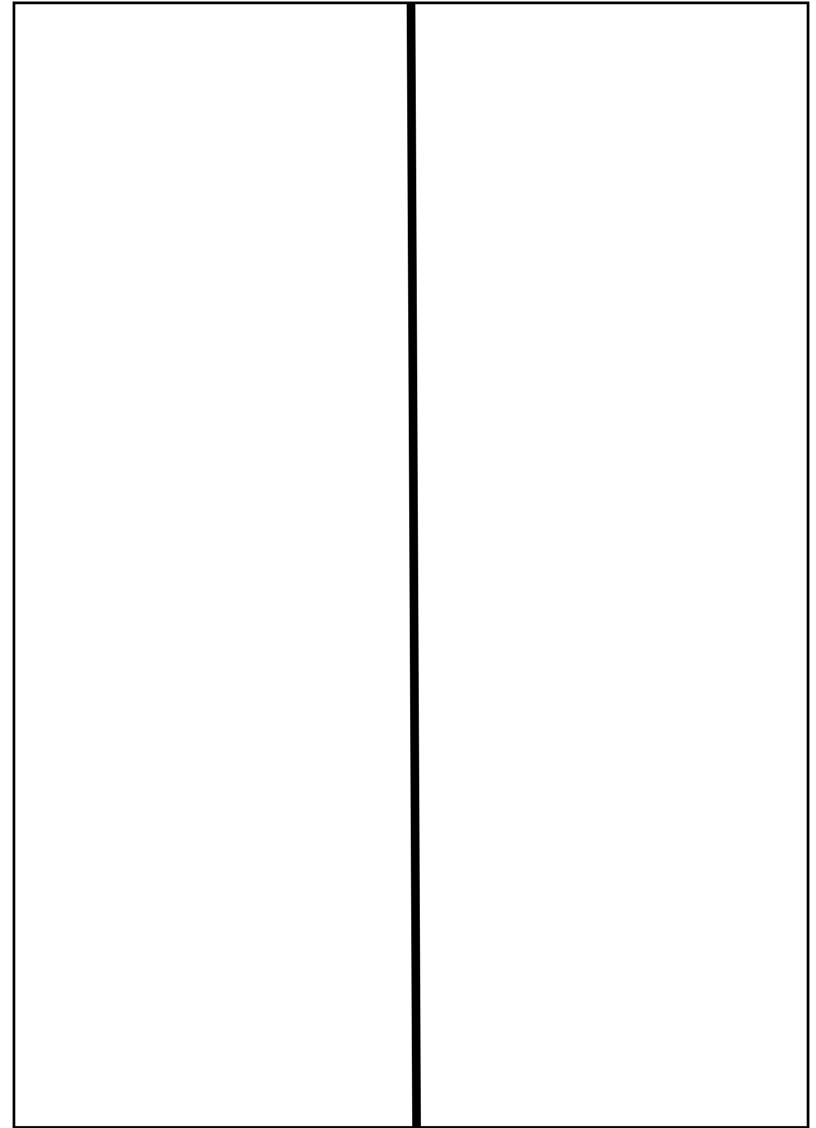
- Compute an action trajectory: $\sigma : [0, 1] \rightarrow U$

Example

M = {OPEN, CLOSED}



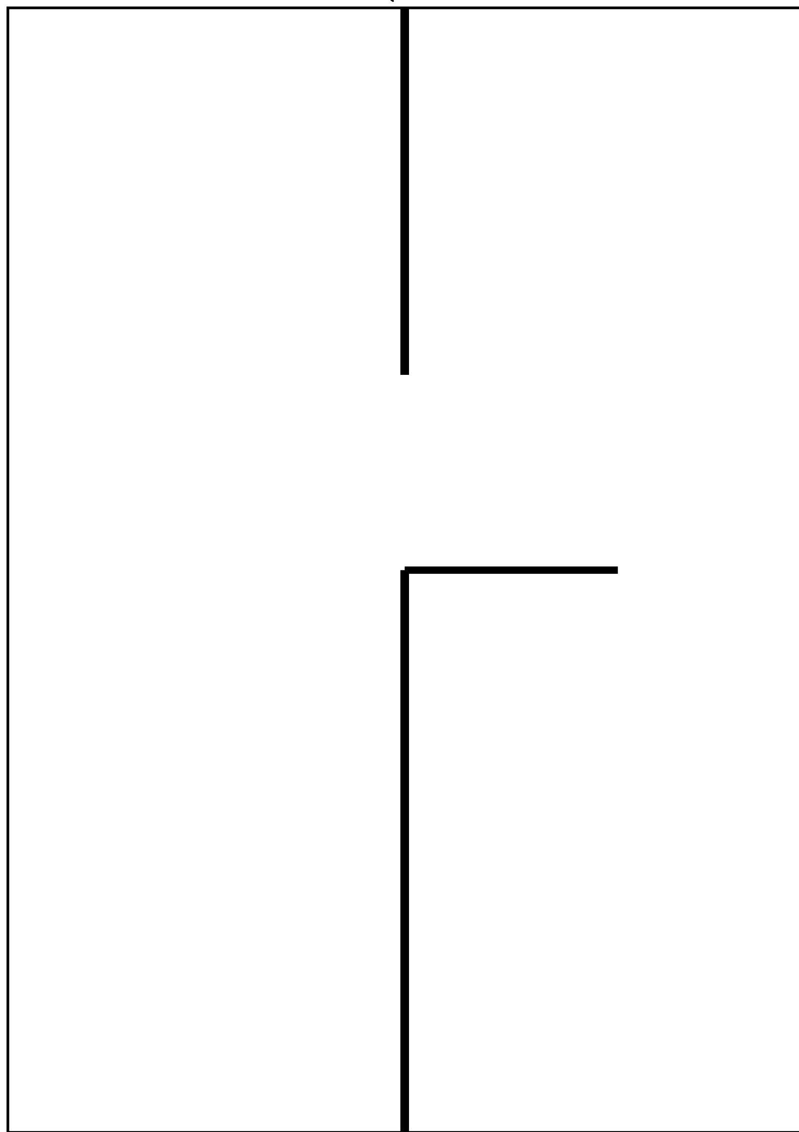
O(OPEN)



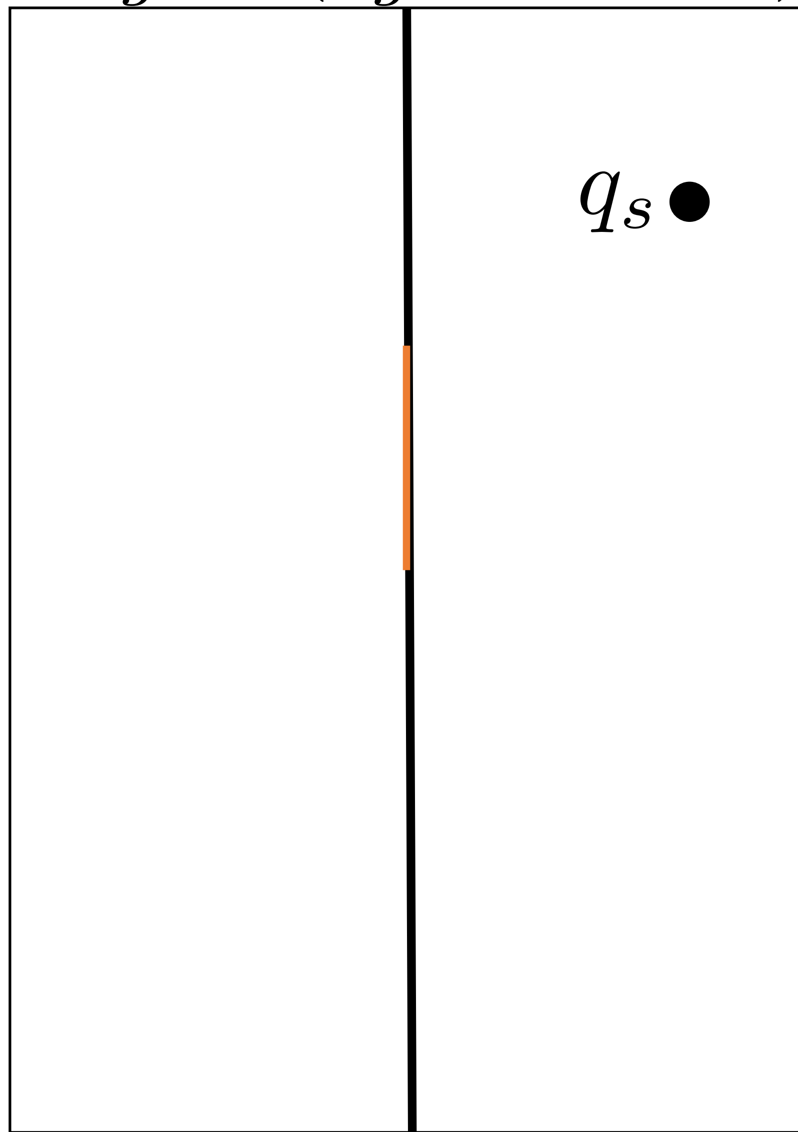
O(CLOSED)

Example

$$x_s = (q_s, CLOSED), x_g = (q_g, OPEN)$$



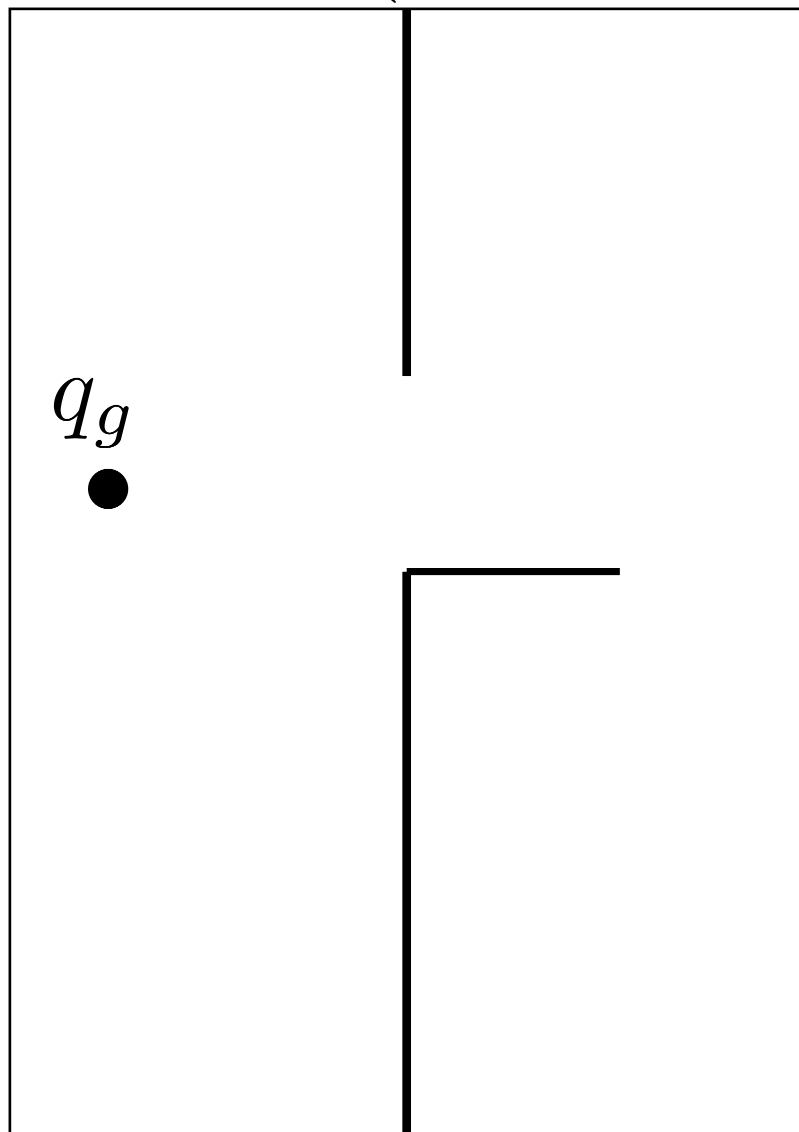
O(OPEN)



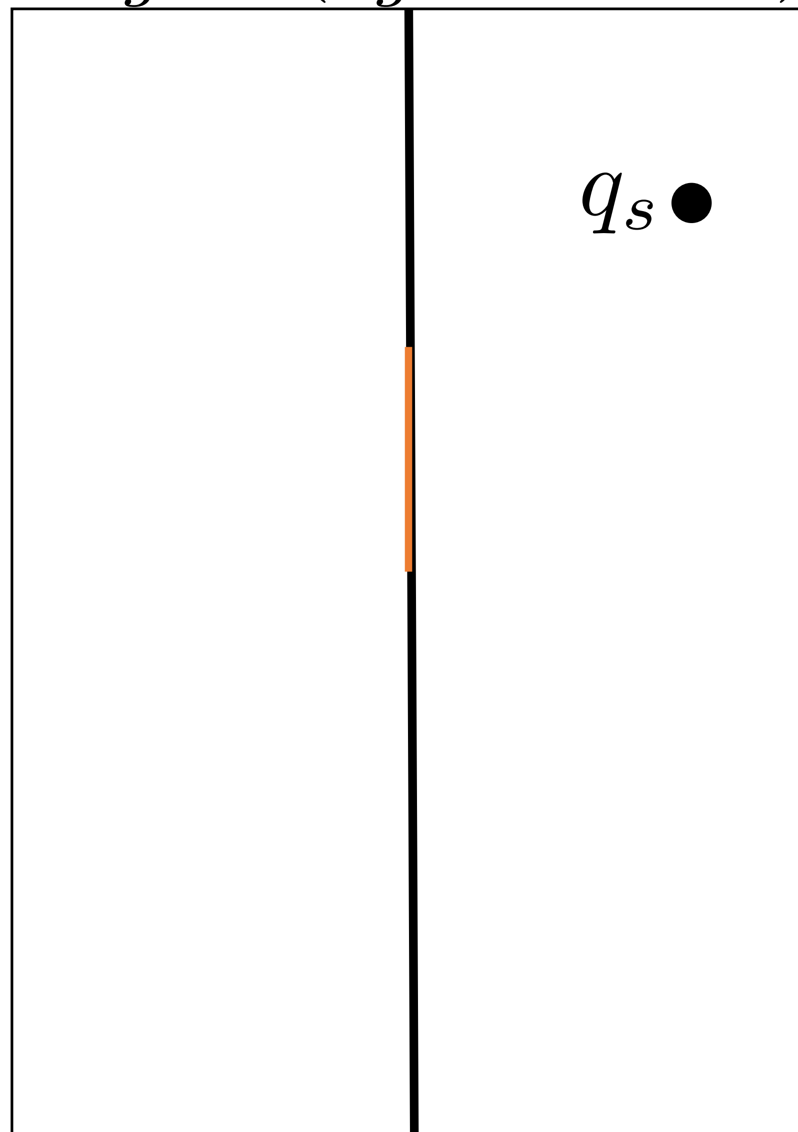
O(CLOSED)

Example

$$x_s = (q_s, CLOSED), x_g = (q_g, OPEN)$$



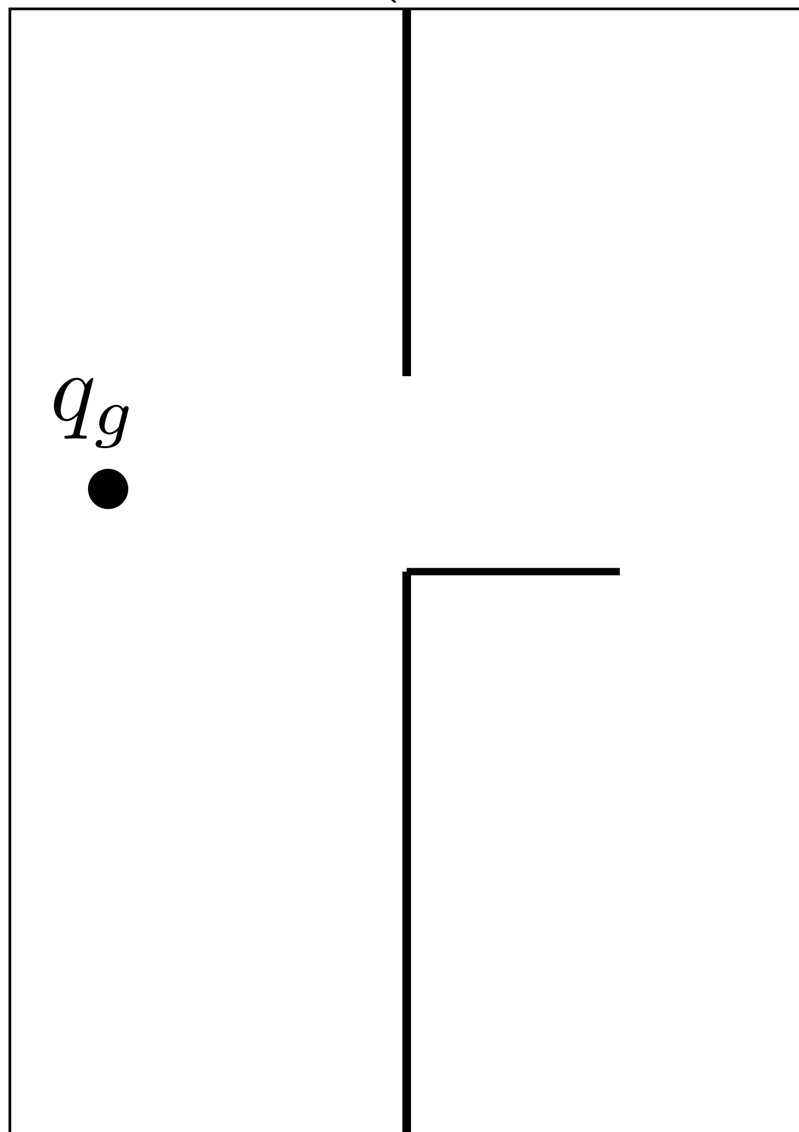
O(OPEN)



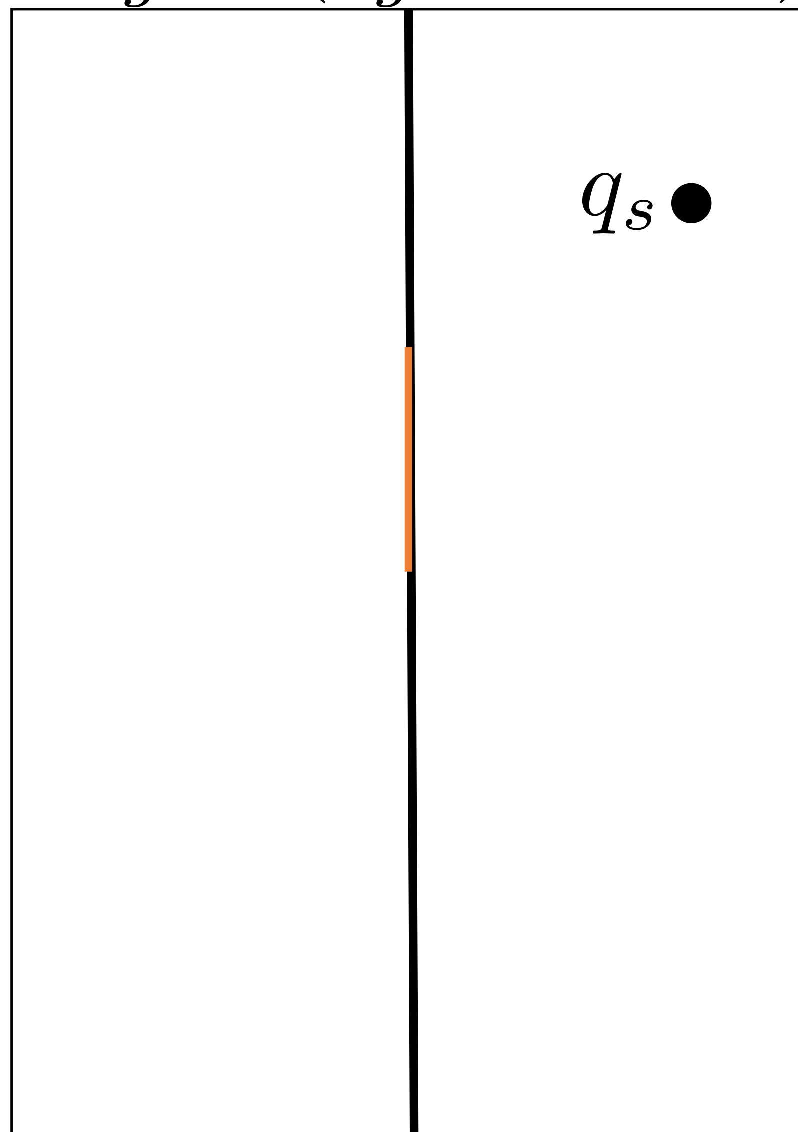
O(CLOSED)

Example

$$x_s = (q_s, CLOSED), x_g = (q_g, OPEN)$$

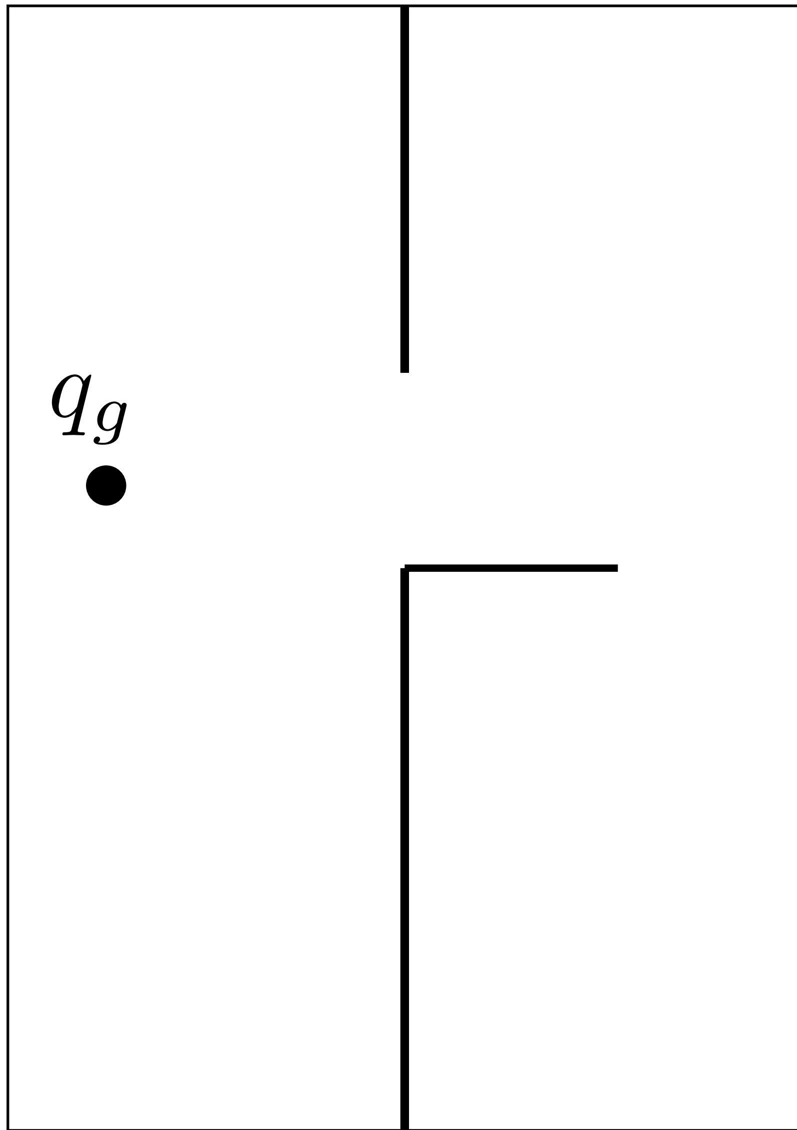


O(OPEN)

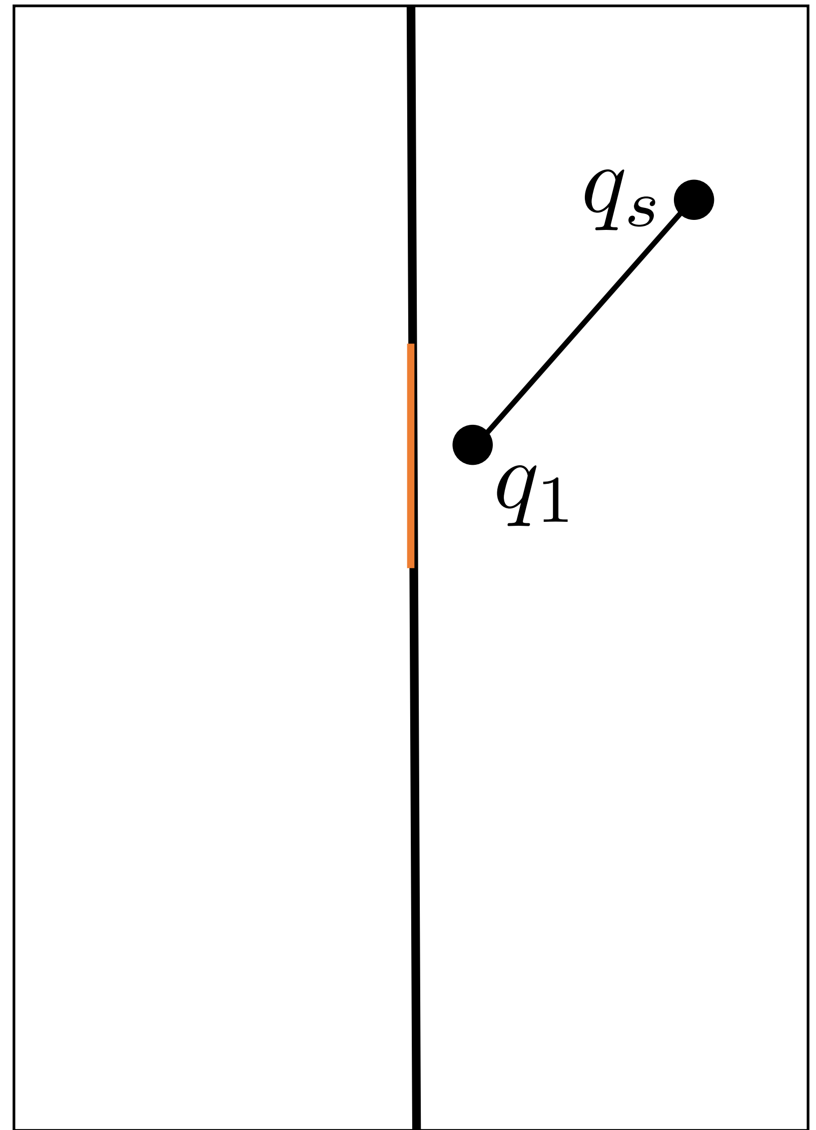


O(CLOSED)

Example

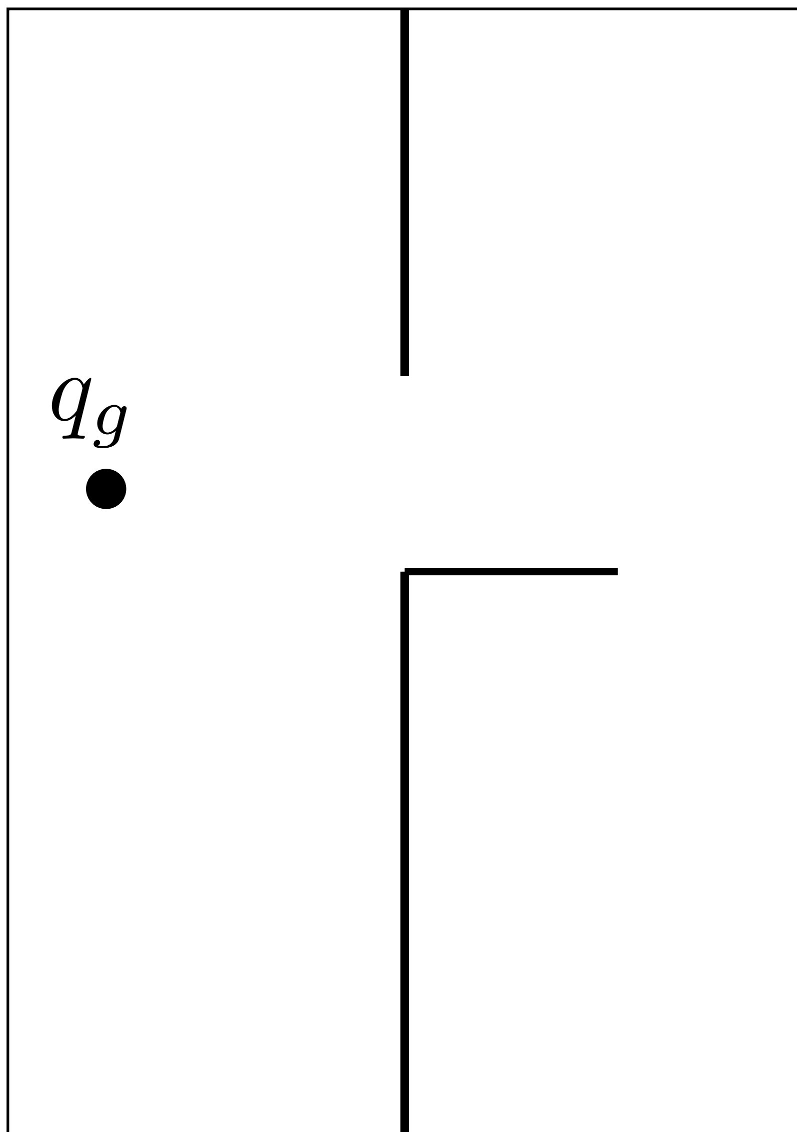


O(OPEN)

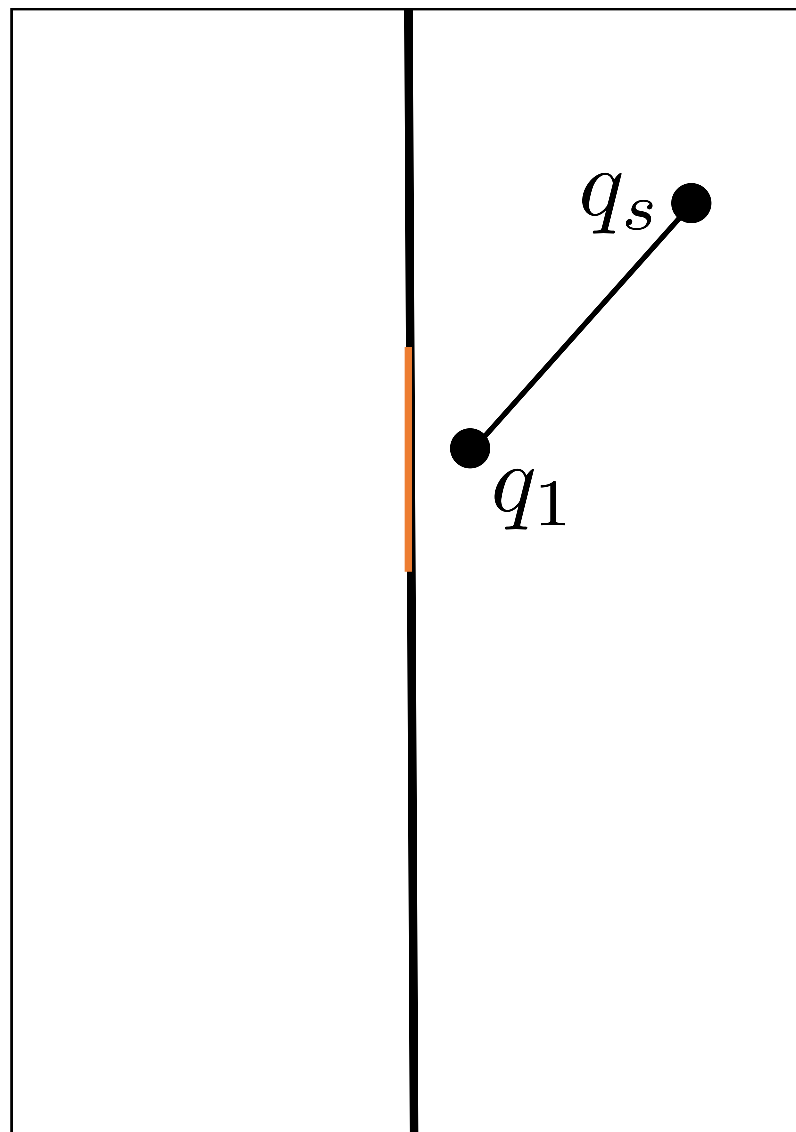


O(CLOSED)

Example

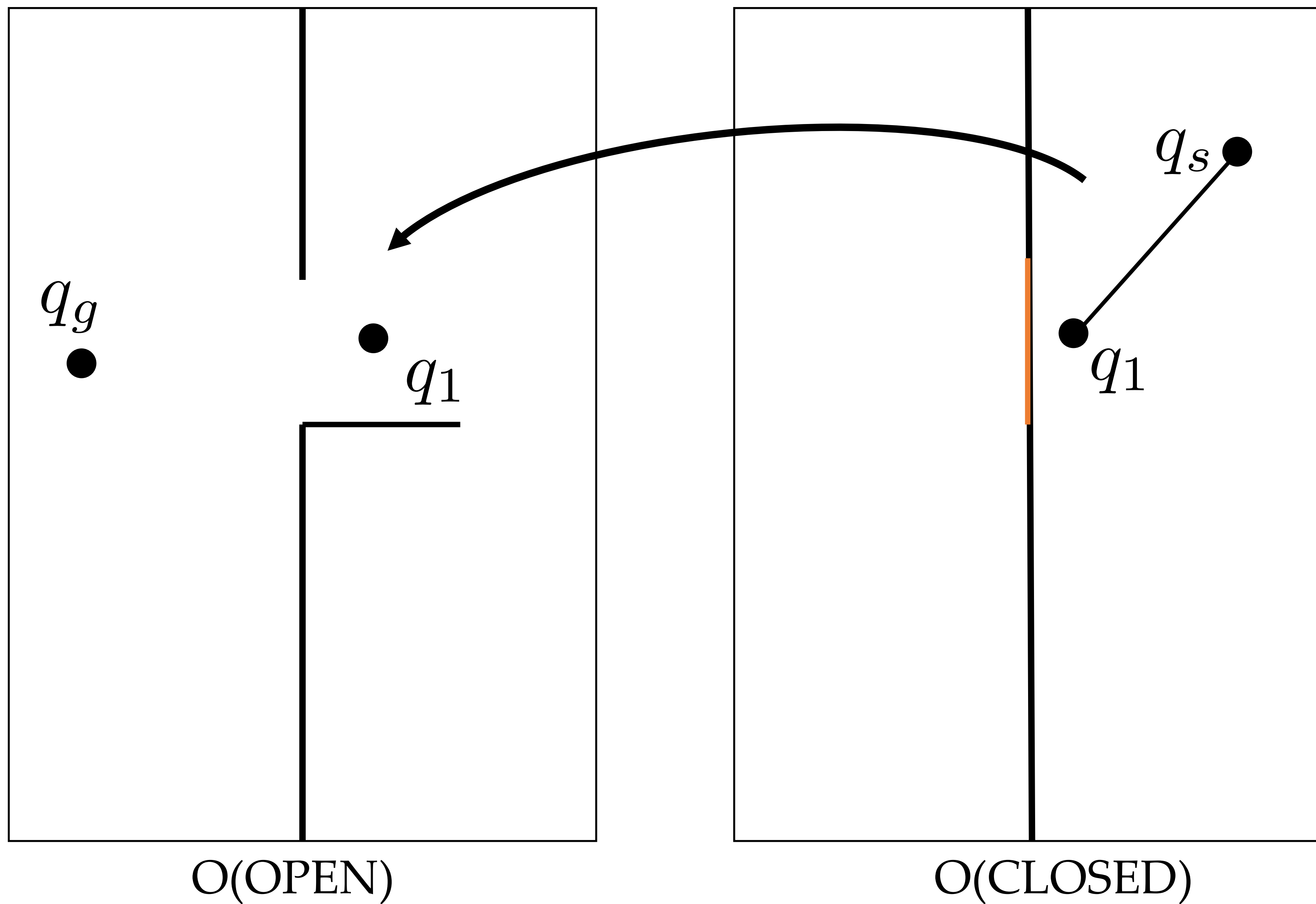


O(OPEN)

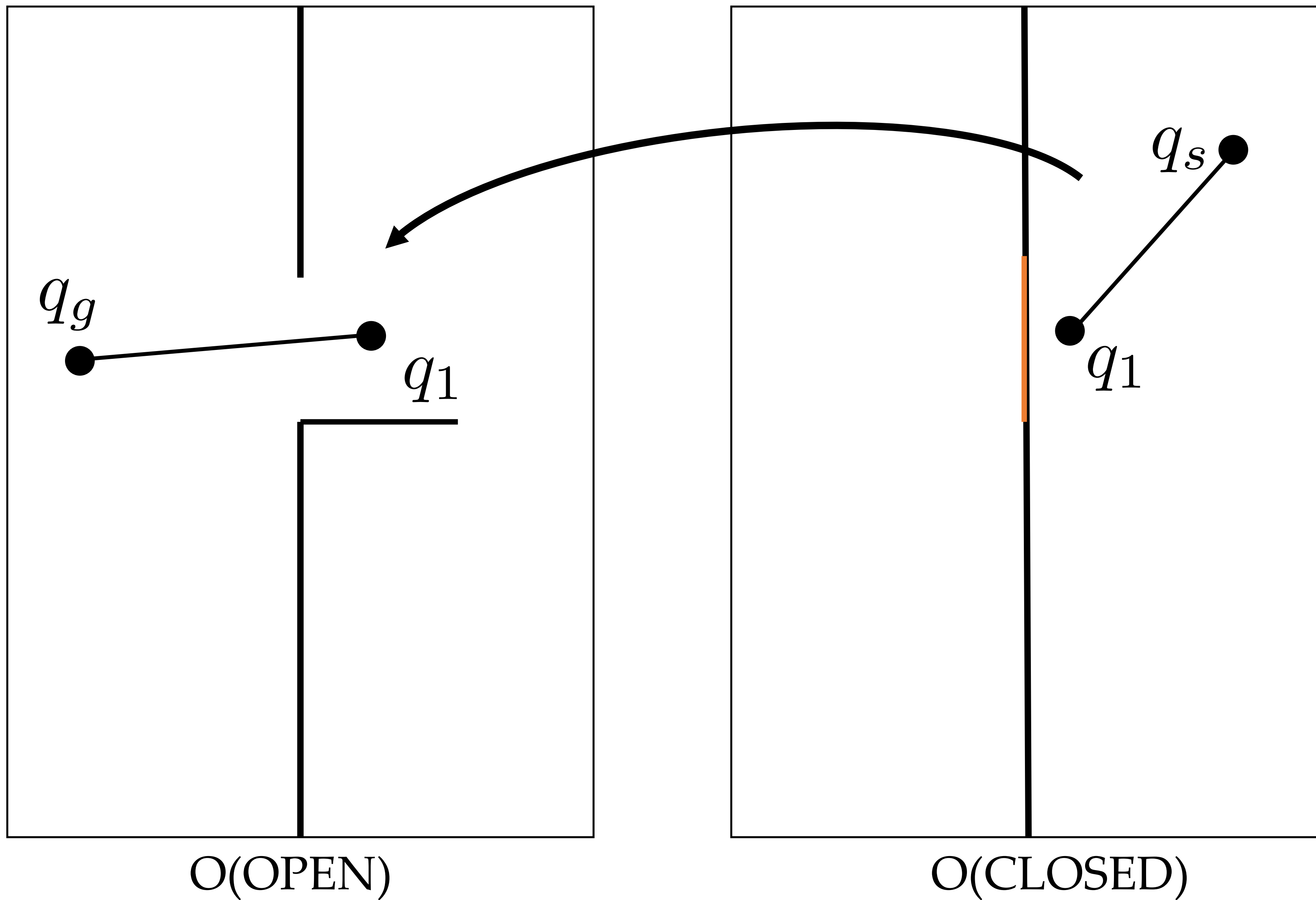


O(CLOSED)

Example



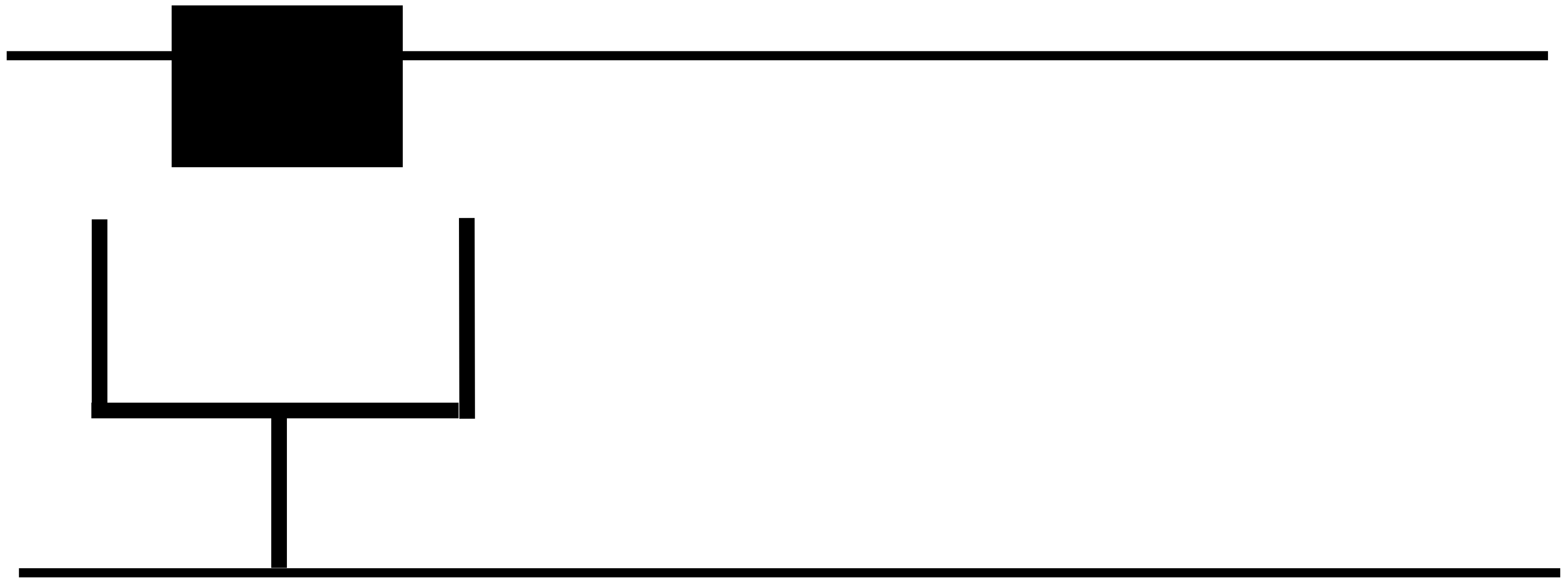
Example



Special Case: Manipulation Planning

Example

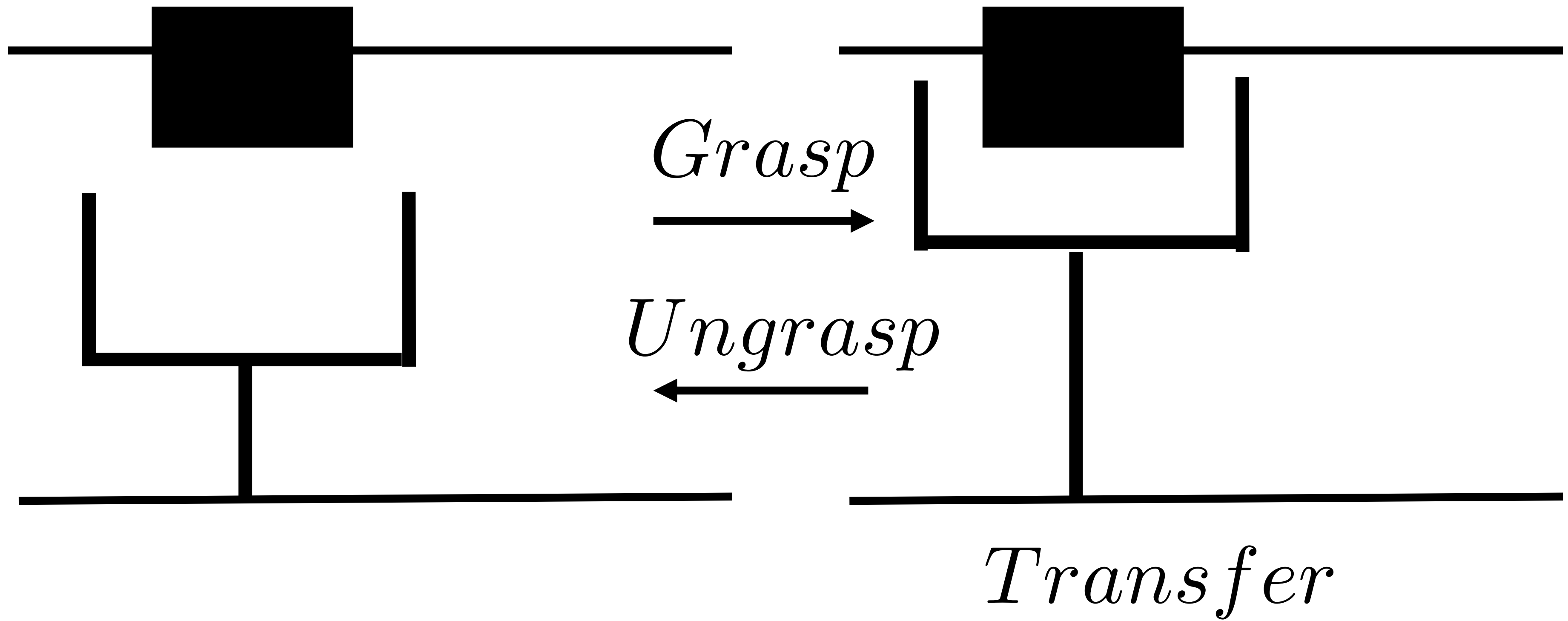
$$M = \{Transit, Transfer\}$$



Robot : $A, Q^A = \mathbb{R}^1$ $\quad U = \{Grasp, Ungrasp\}$

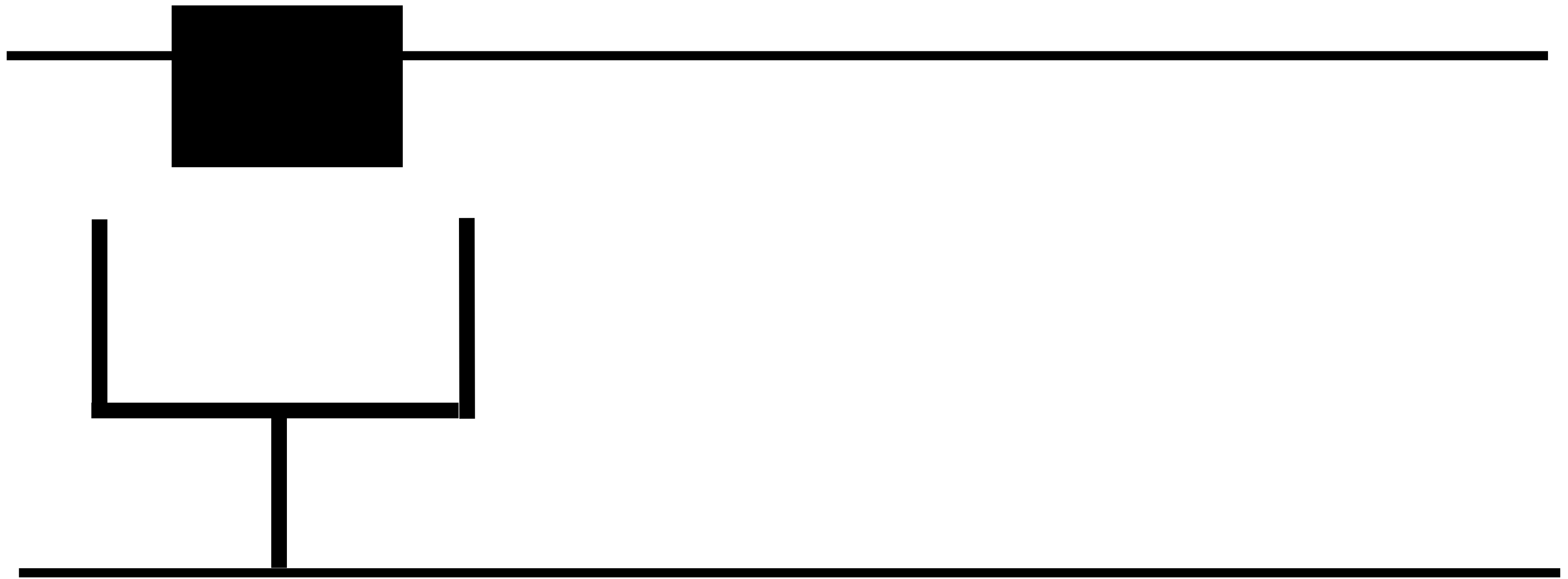
Part : $P, Q^P = \mathbb{R}^1$

Example



Example

$$M = \{Transit, Transfer\}$$

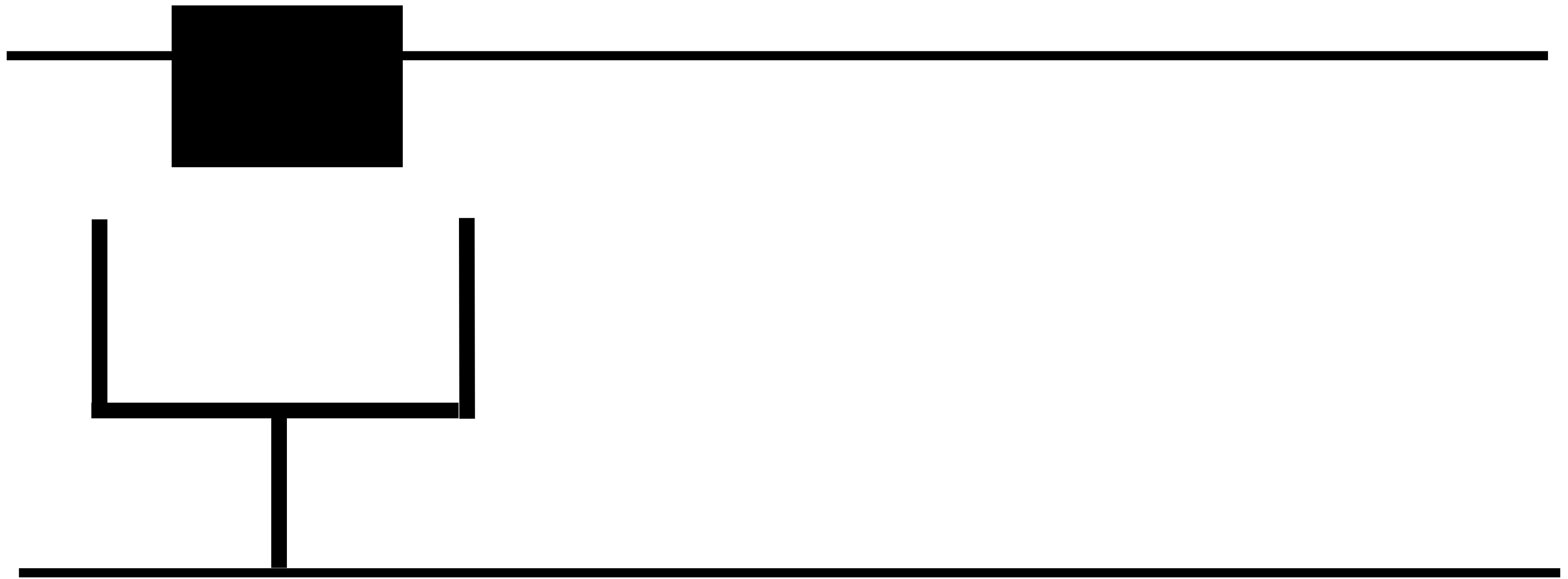


Robot : $A, Q^A = \mathbb{R}^1$ $U = \{Grasp, Ungrasp\}$

Part : $P, Q^P = \mathbb{R}^1$ *When can grasp occur?*

Example

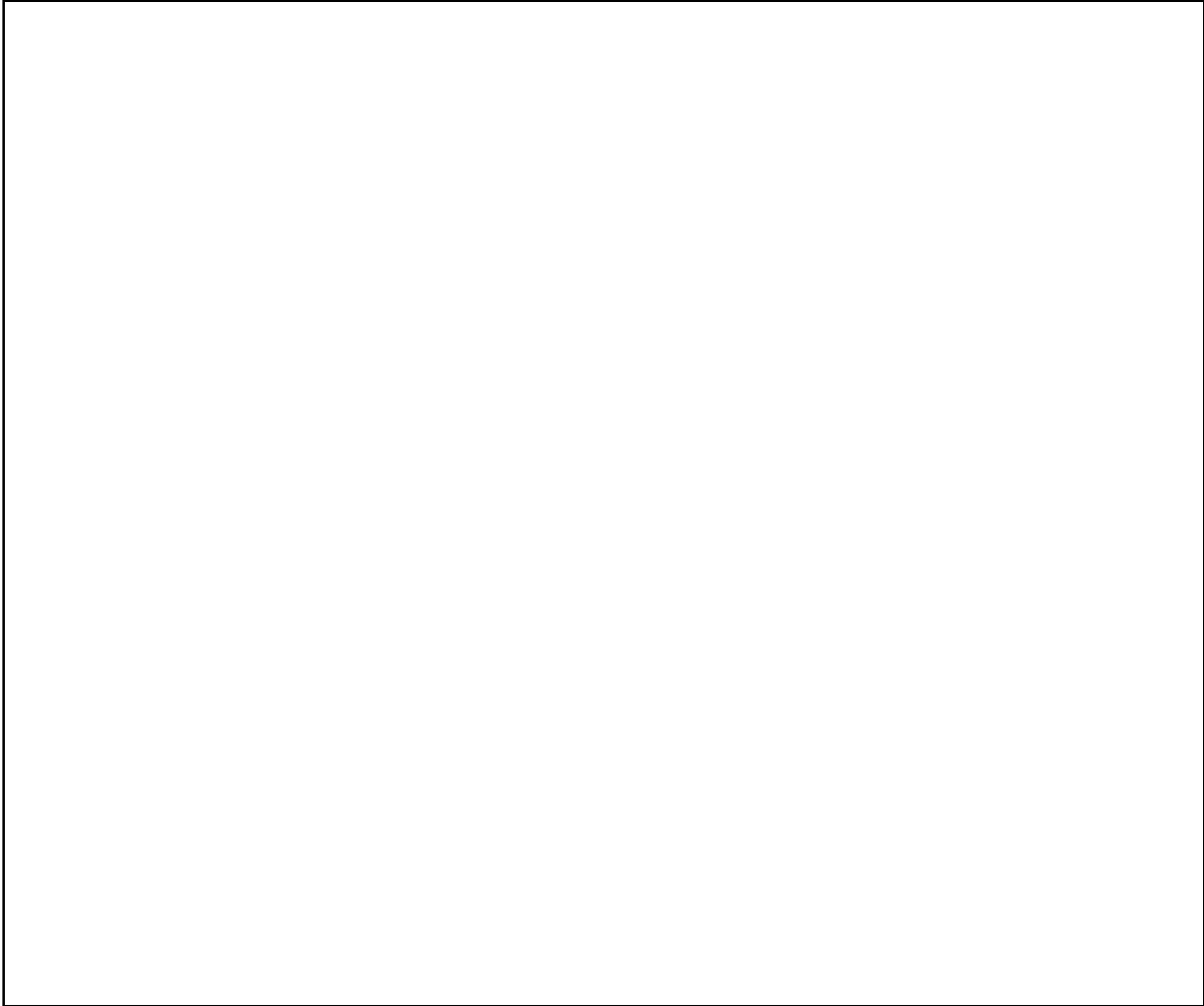
$$M = \{Transit, Transfer\}$$



$$\text{Robot} : A, Q^A = \mathbb{R}^1 \quad U = \{Grasp, Ungrasp\}$$

$$\text{Part} : P, Q^P = \mathbb{R}^1 \quad q^A = q^P$$

Example



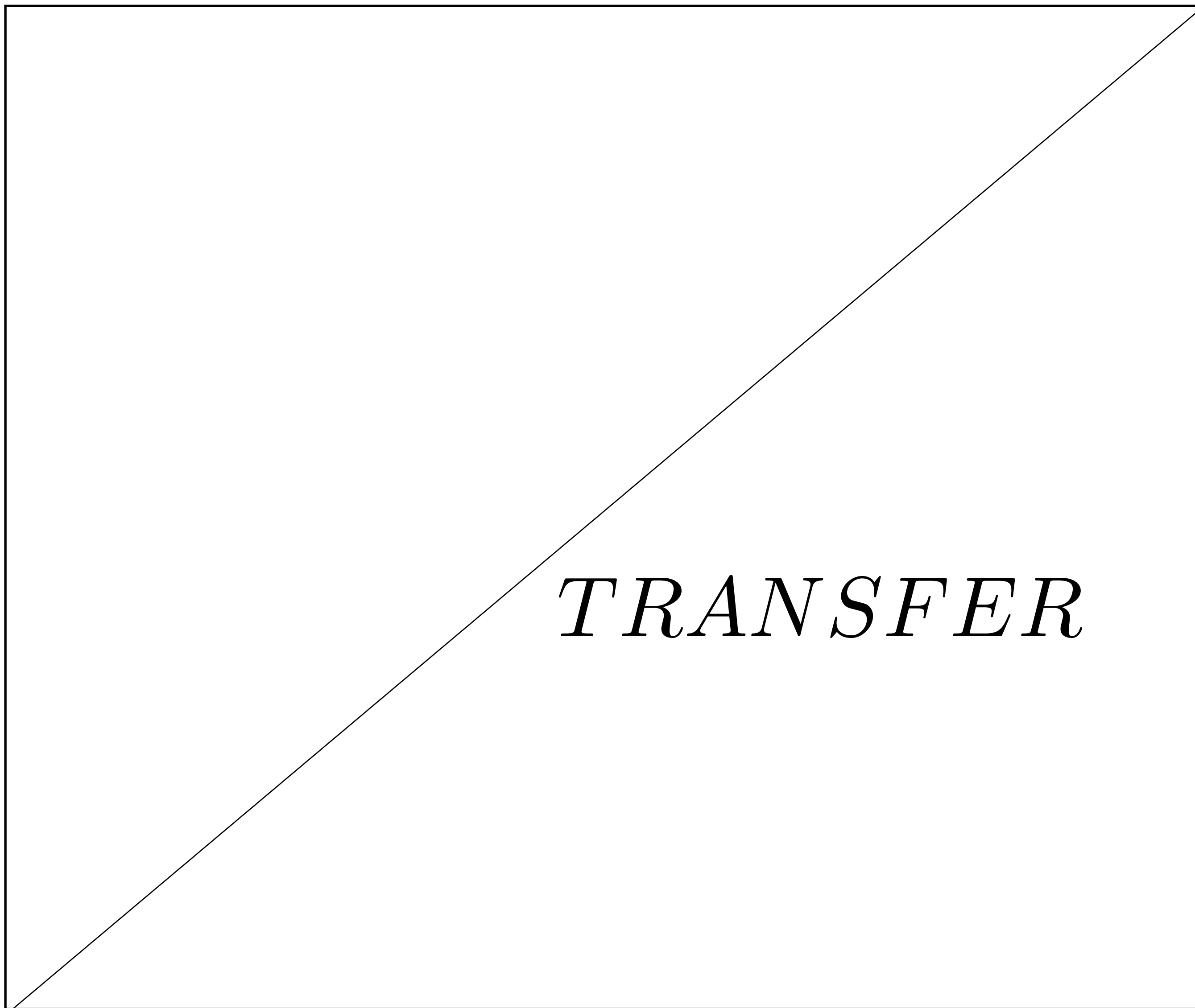
Example

q^P

How can we represent the transfer phase?

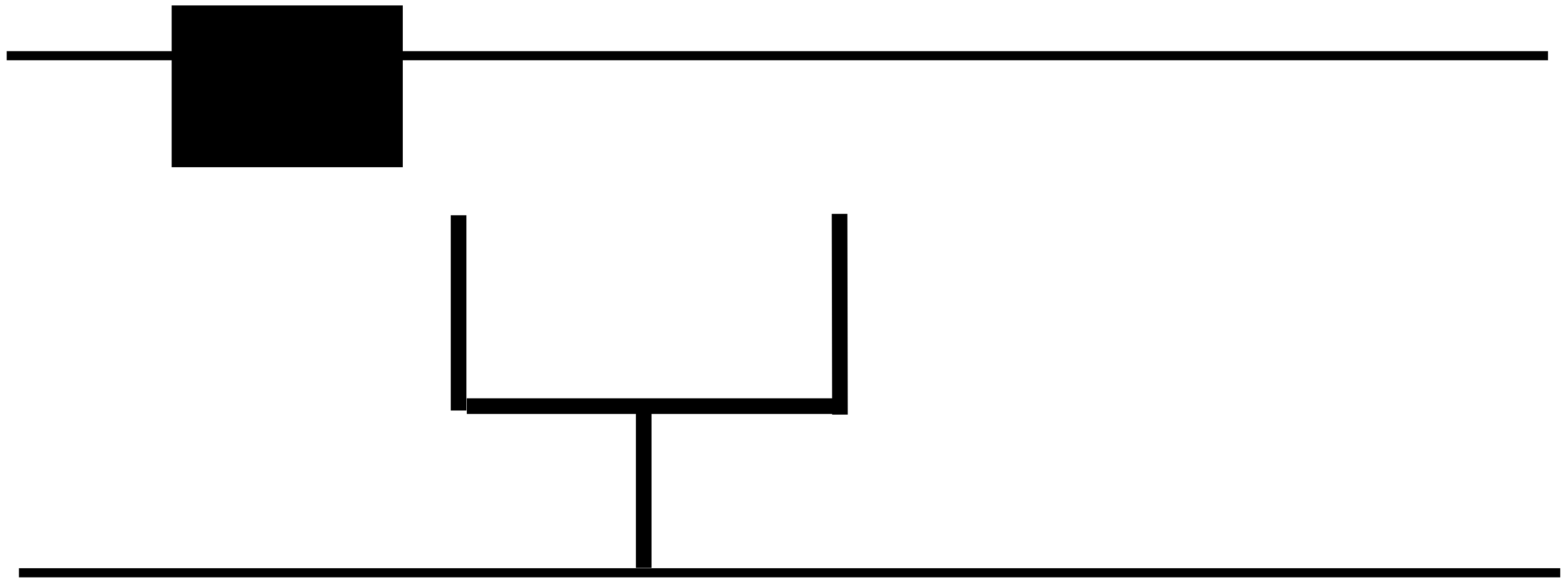
q^A

Example



Example

$$M = \{Transit, Transfer\}$$

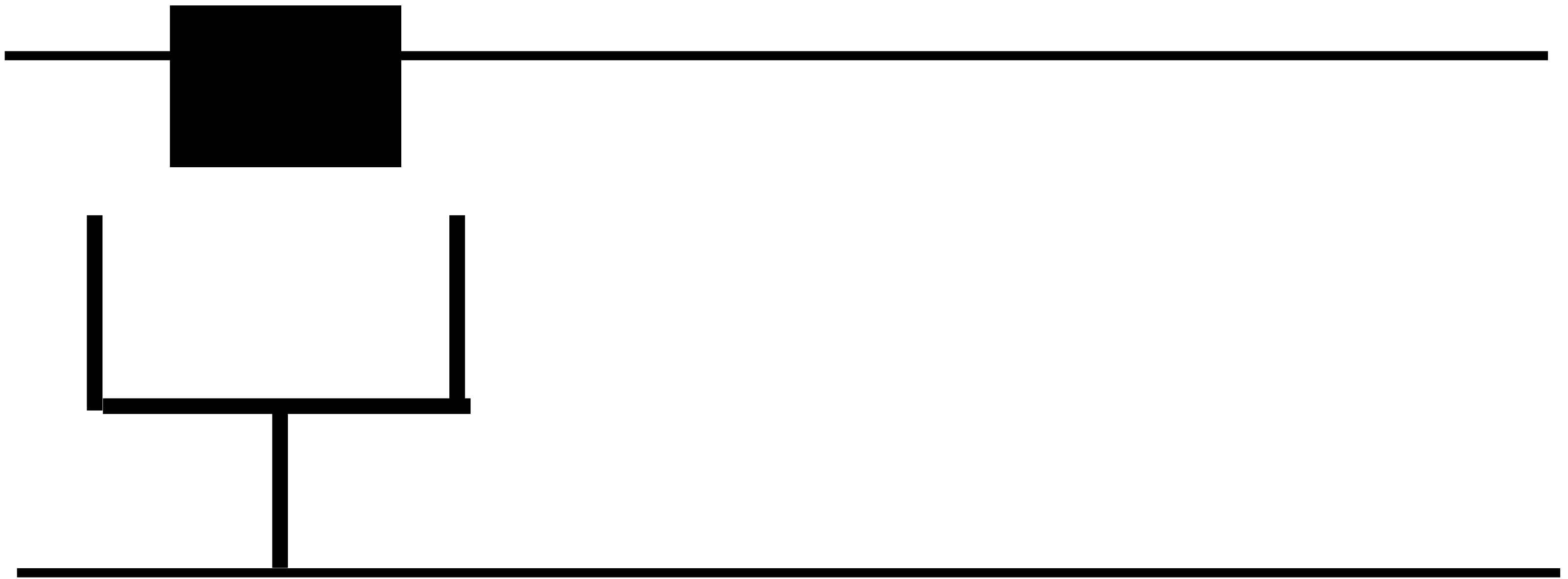


Robot : $A, Q^A = \mathbb{R}^1$ $\quad U = \{Grasp, Ungrasp\}$

Part : $P, Q^P = \mathbb{R}^1$

Example

$$M = \{Transit, Transfer\}$$

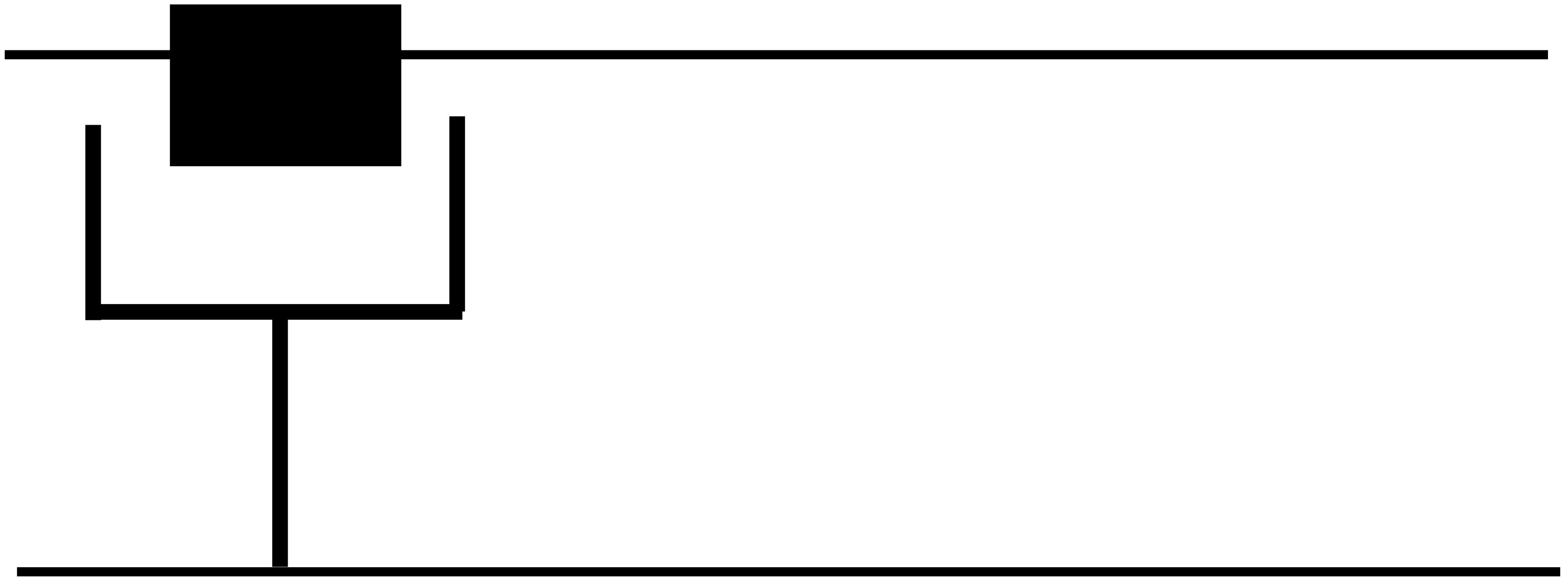


$$\text{Robot} : A, Q^A = \mathbb{R}^1 \qquad U = \{Grasp, Ungrasp\}$$

$$\text{Part} : P, Q^P = \mathbb{R}^1$$

Example

$$M = \{Transit, Transfer\}$$



Robot : $A, Q^A = \mathbb{R}^1$ $\quad U = \{Grasp, Ungrasp\}$

Part : $P, Q^P = \mathbb{R}^1$

Example

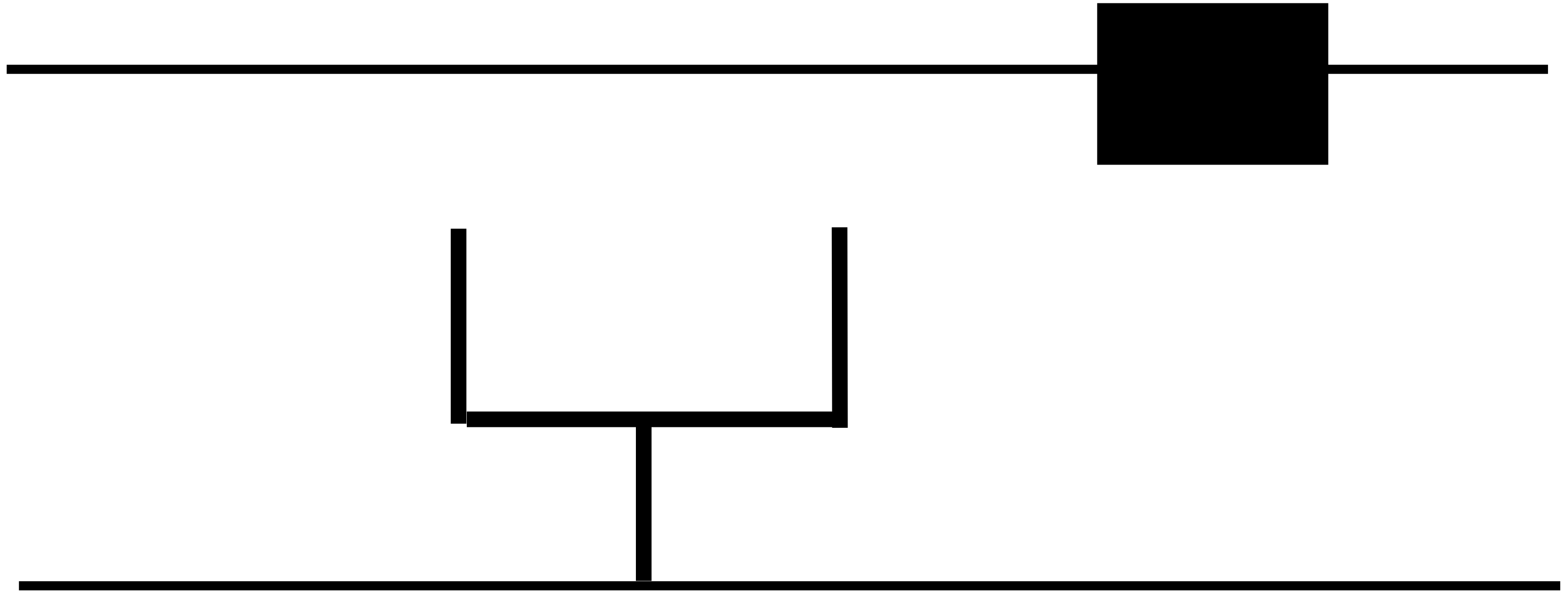
$$M = \{Transit, Transfer\}$$

$$\text{Robot} : A, Q^A = \mathbb{R}^1 \quad U = \{Grasp, Ungrasp\}$$

$$\text{Part} : P, Q^P = \mathbb{R}^1$$

Example

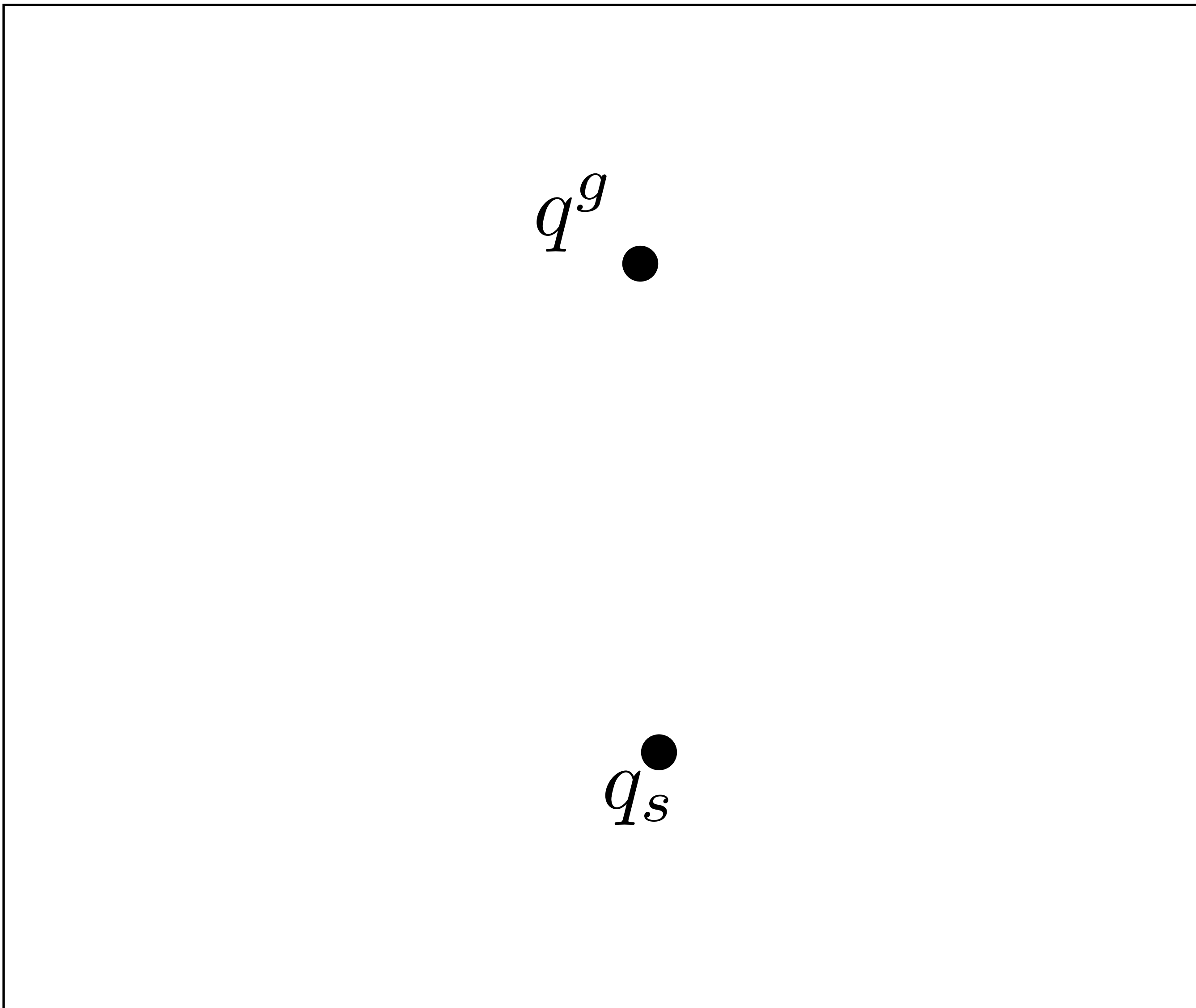
$$M = \{Transit, Transfer\}$$



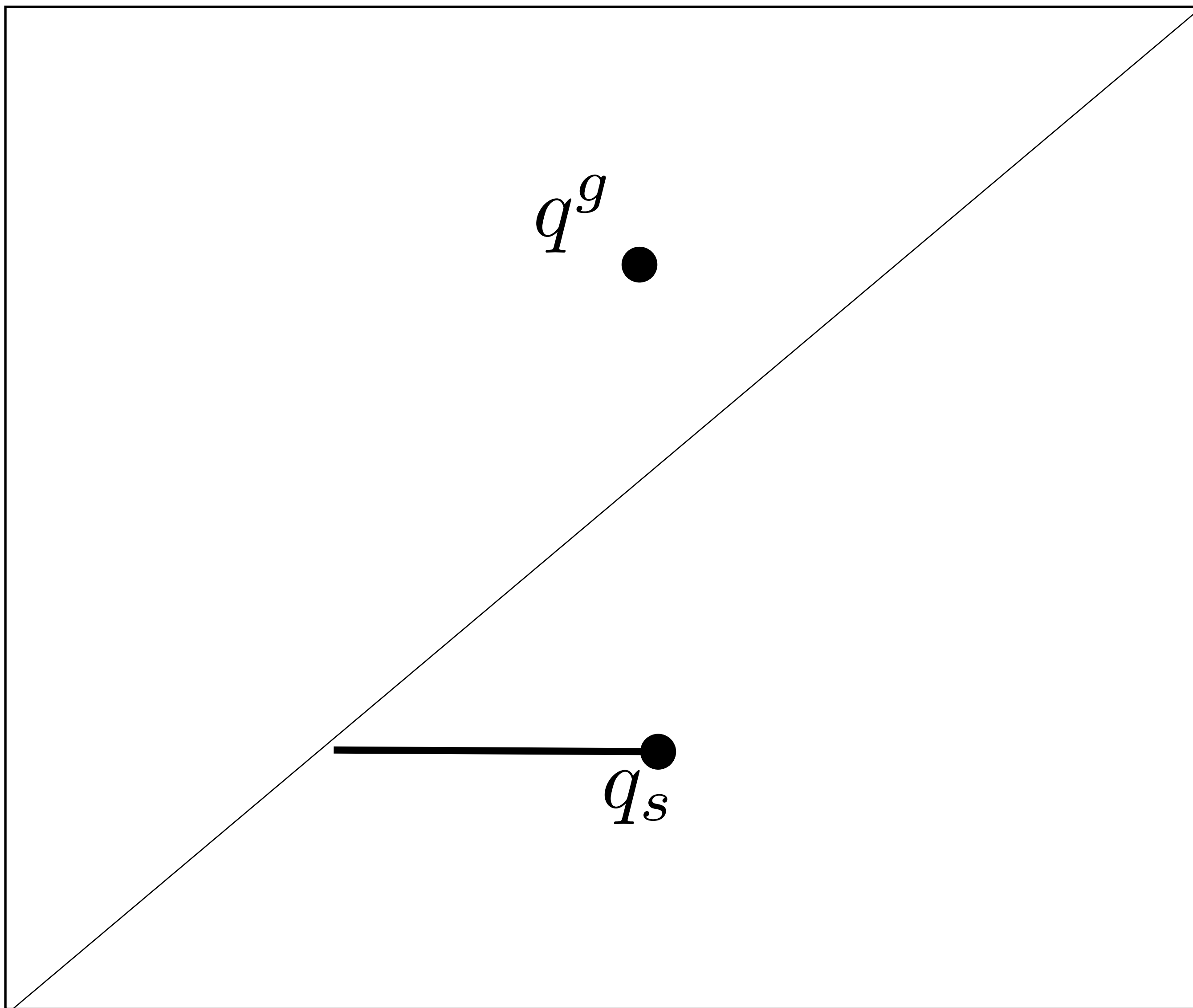
$$\text{Robot} : A, Q^A = \mathbb{R}^1 \quad U = \{Grasp, Ungrasp\}$$

$$\text{Part} : P, Q^P = \mathbb{R}^1$$

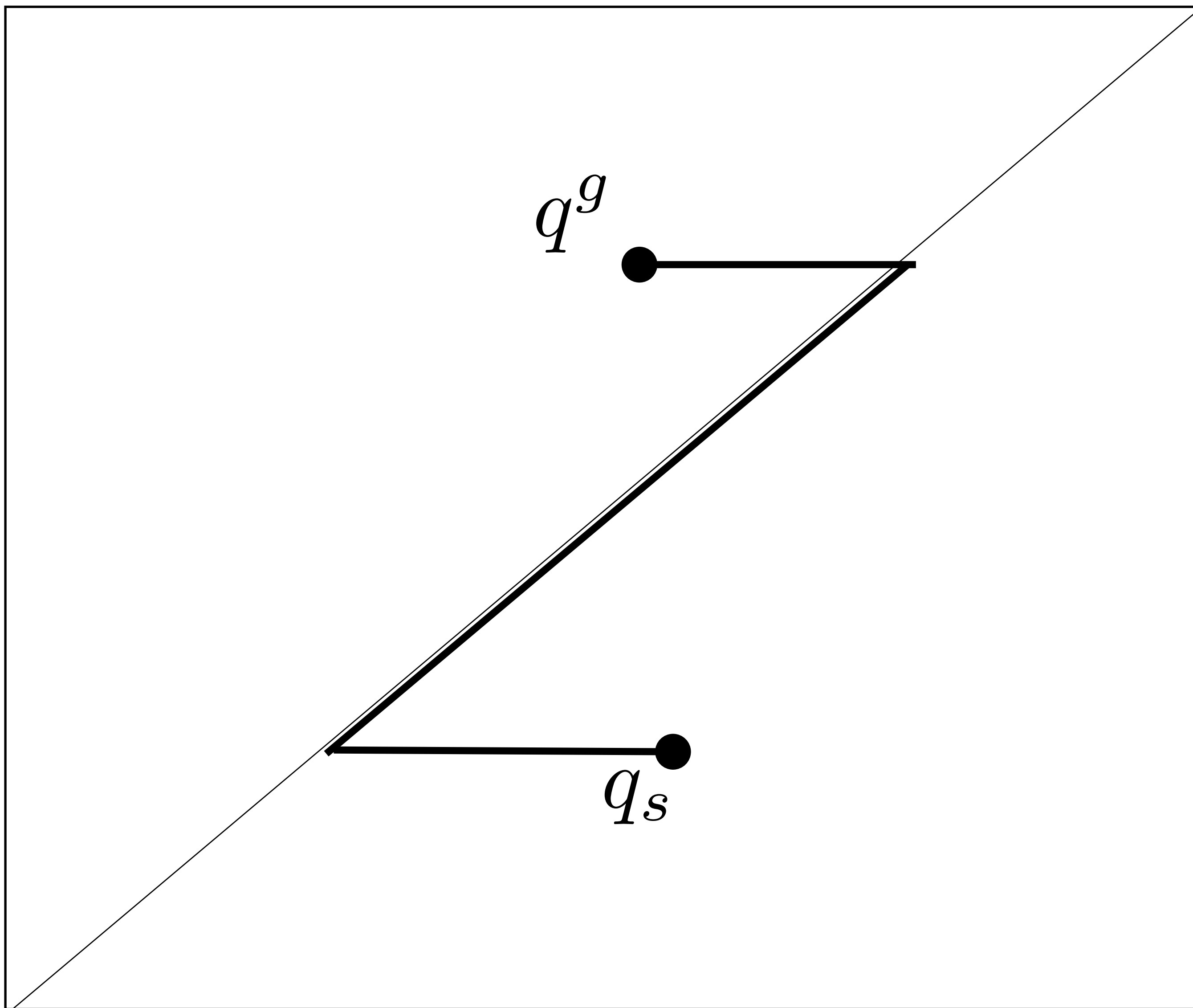
Example



Example

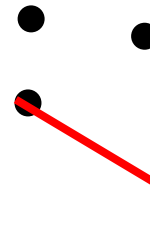


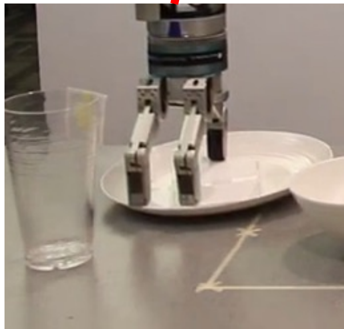
Example

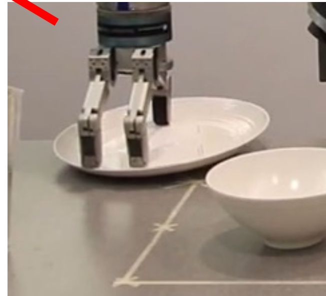


Manipulation Planning

- Sample configurations where mode transition can occur (valid grasp configuration with resting object)

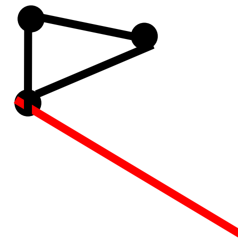


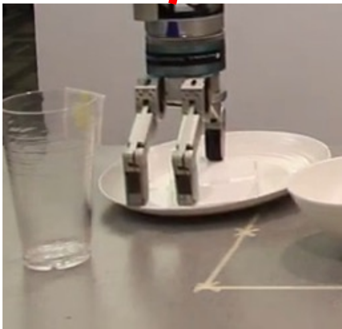


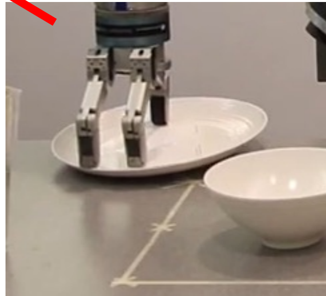


Manipulation Planning

- Compute sub-roadmaps

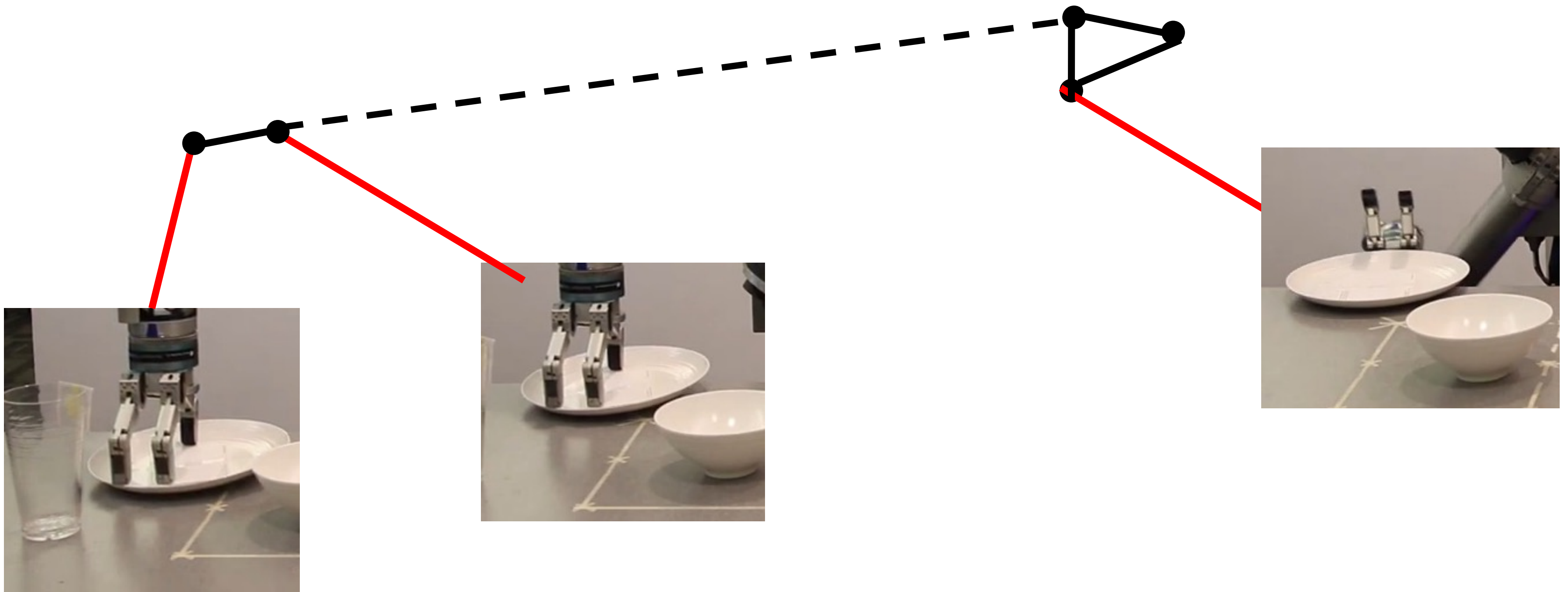




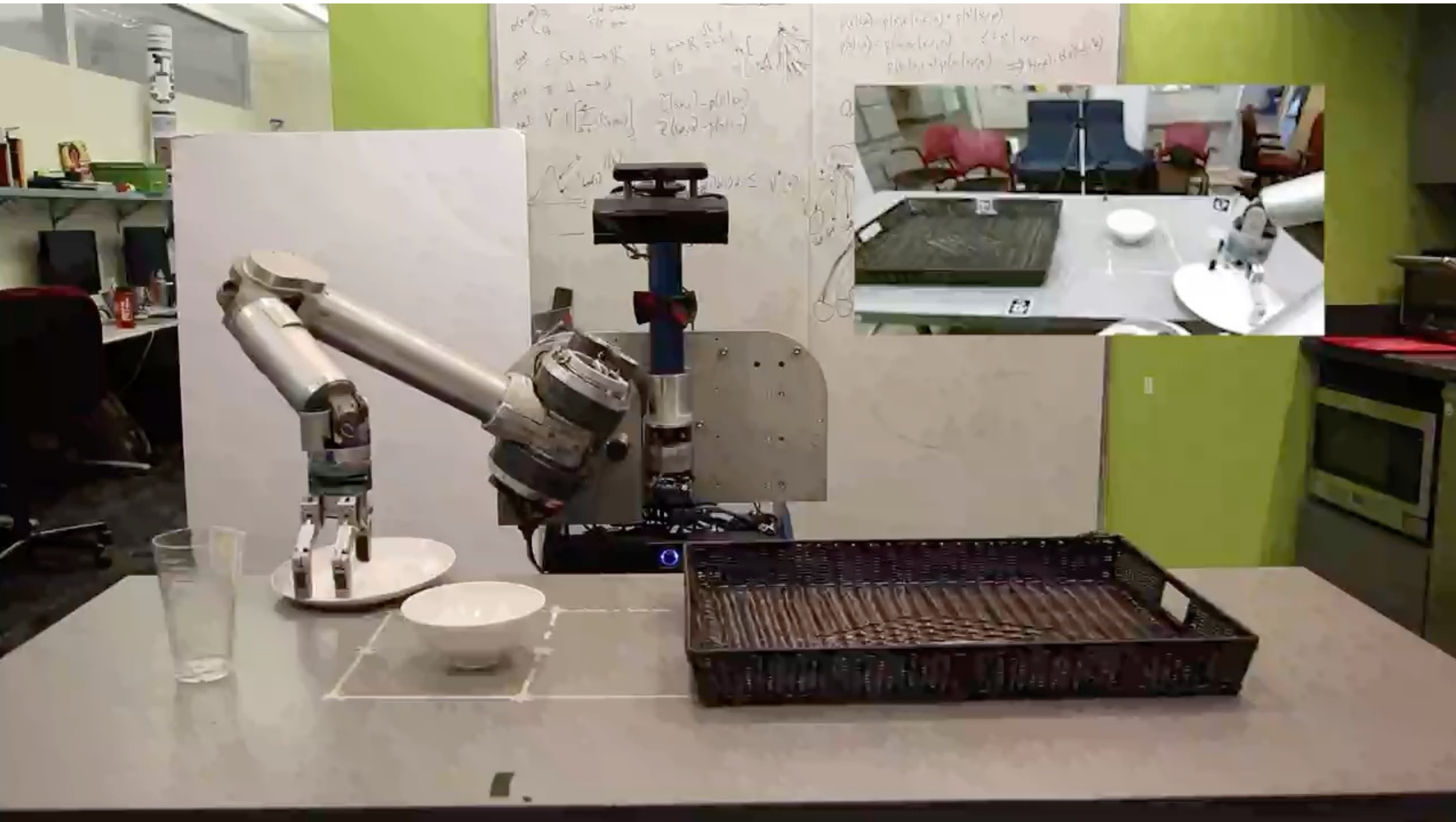


Manipulation Planning

- Attempt to connect each pair of vertices in the two roadmaps with transit or transfer paths

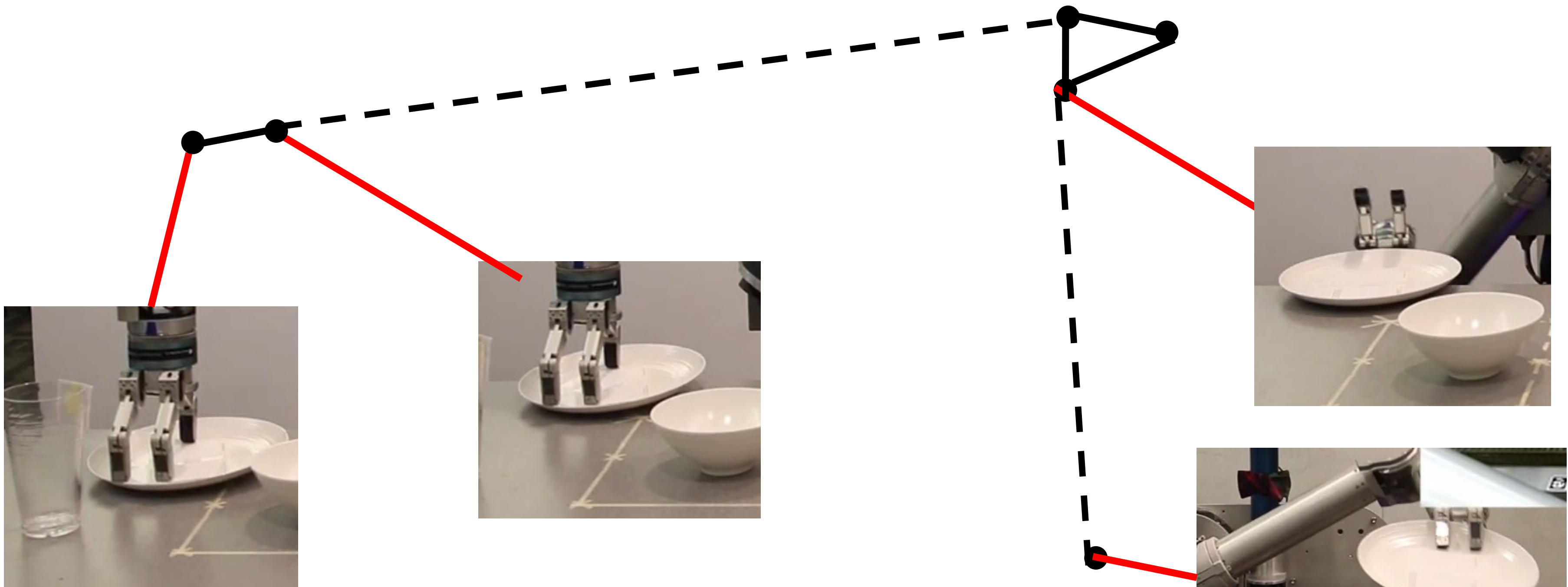


Manipulation Planning



Manipulation Planning

- Connect to goal configuration



Manipulation Planning

- Search graph