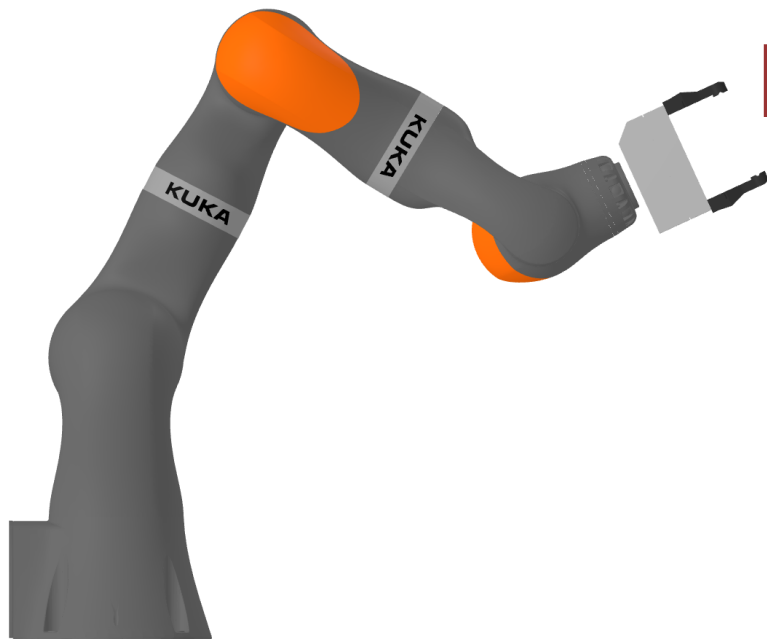




Motion Planning

Part I: Basics and Sampling

MIT 6.4210/2

Robotic Manipulation

Fall 2025, Lecture 9

Motion Planning

Most slides from:

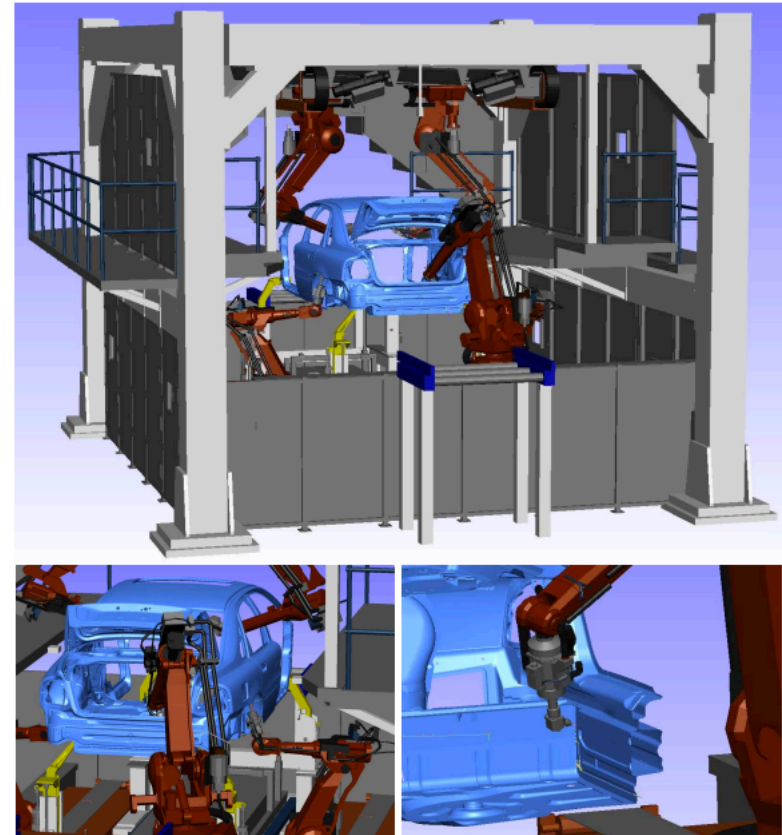
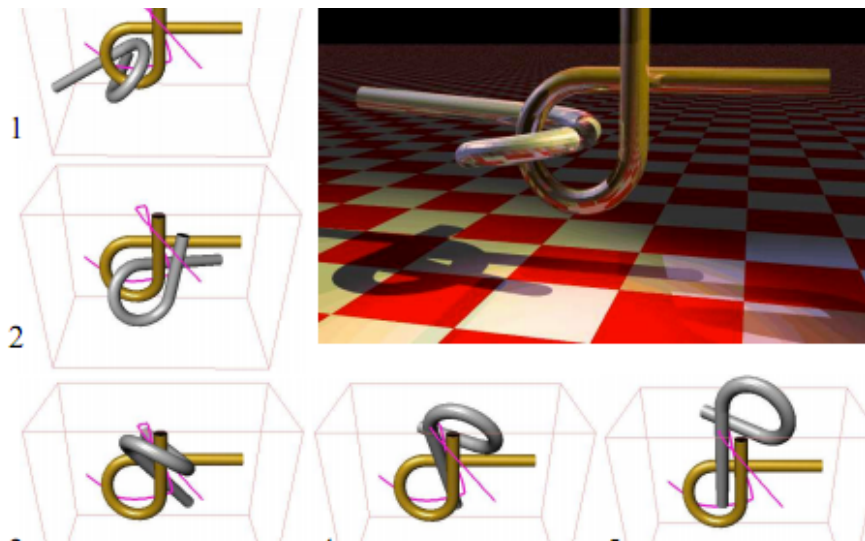
Katharina Muelling, CMU (16-843 Manipulation Algorithms)

Some slides from:

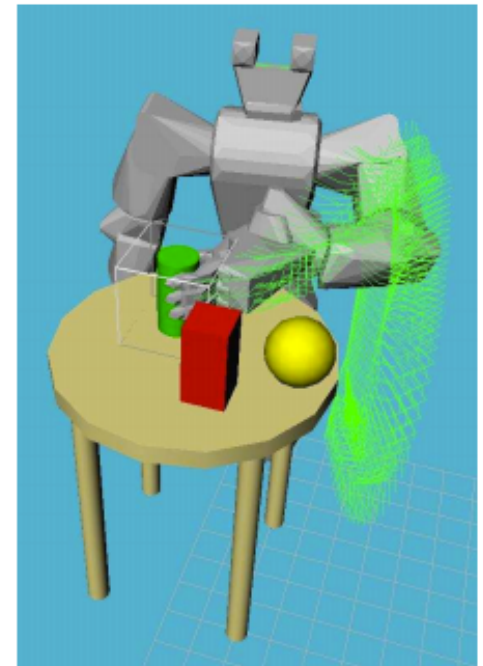
Pieter Abbeel (Berkeley)

Howie Choset (CMU)

Examples



Examples

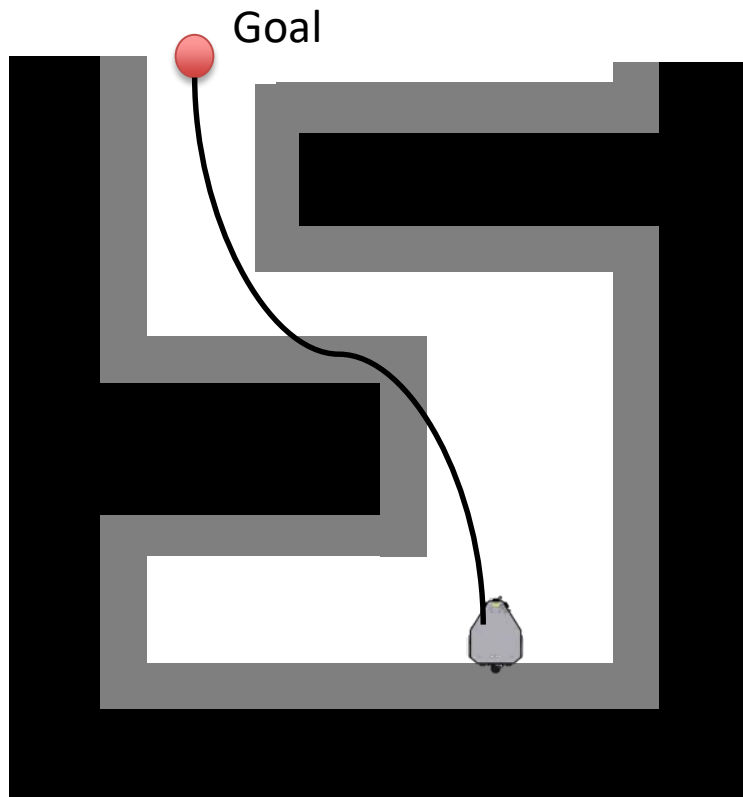


Examples



Problem Definition

Consider the problem of robot motion planning:



How to avoid collisions of parts of the robot with the environment if modeled as point?

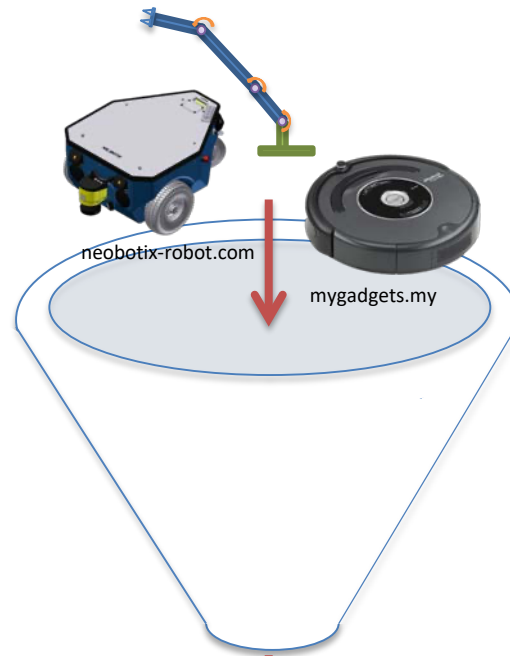
Robot:



neobotix-robot.com

Outline

Can we create a space in which all robots can be treated as points?



Single point representation

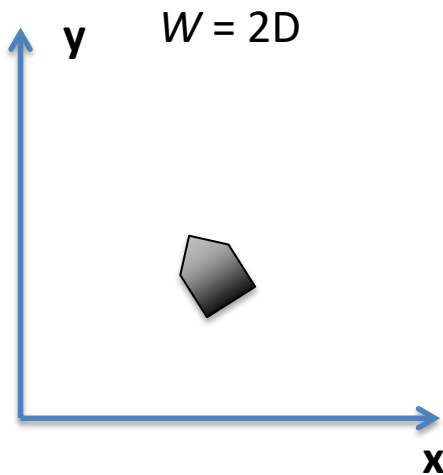
Configuration Space

Algorithms

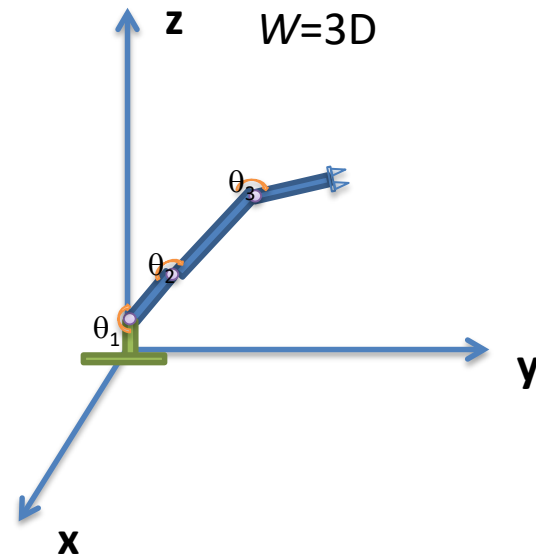
Configuration Space

Definitions

Work space: The world in which the robot lives and occupies space.
The set of reachable points by an end-effector.



3-parameter specification (x, y, θ)



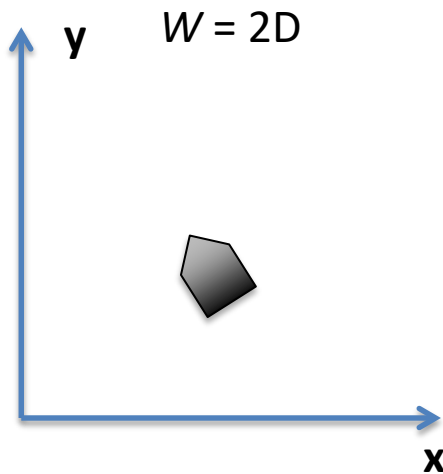
6-parameter specification $(x, y, z, \alpha, \beta, \gamma)$

Configuration Space

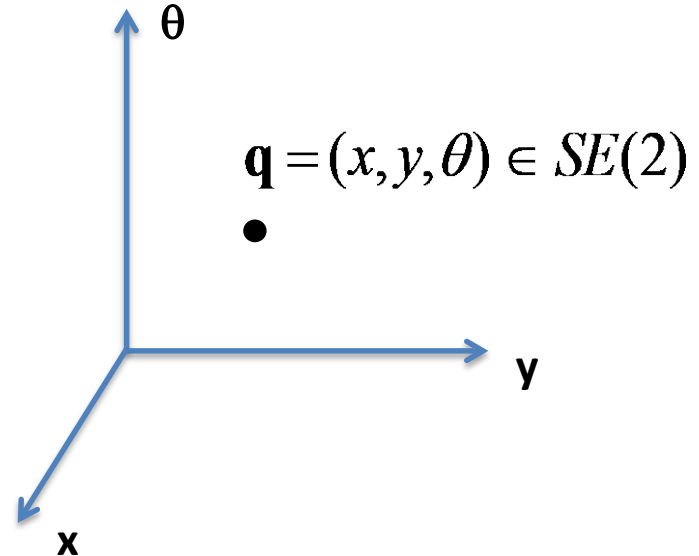
Definitions

Configuration:

The **configuration** of a robot is the sufficient and complete specification of the position of every point of the physical system



World space



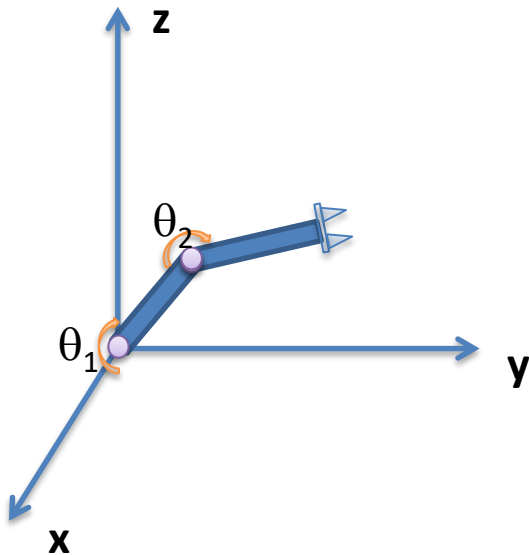
Configuration Space

Configuration Space

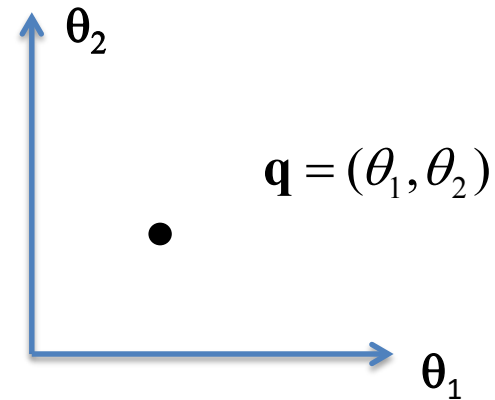
Definitions

Configuration:

The **configuration** of a robot is the sufficient and complete specification of the position of every point of the physical system



World space



Configuration Space

Configuration Space

Definitions

Configuration:

The **configuration** of a robot is the sufficient and complete specification of the position of every point of the physical system

The configuration $\mathbf{q}$ is usually expressed as a vector of length n , where n defines the degree of freedom of the robot

$$\mathbf{q} = [q_1, q_2, \dots, q_n]$$

Configuration Space:

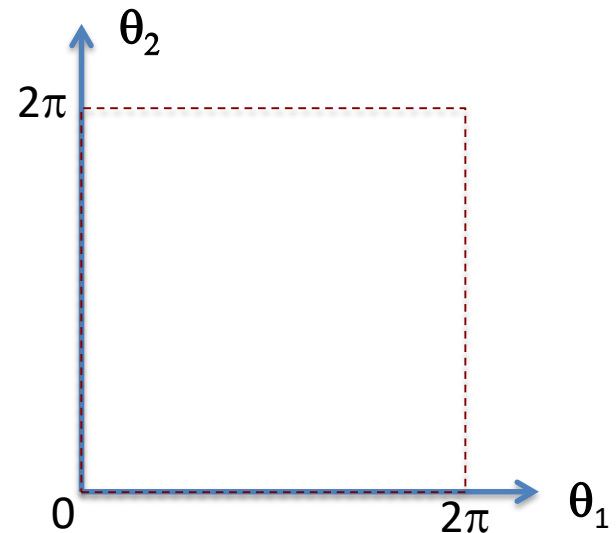
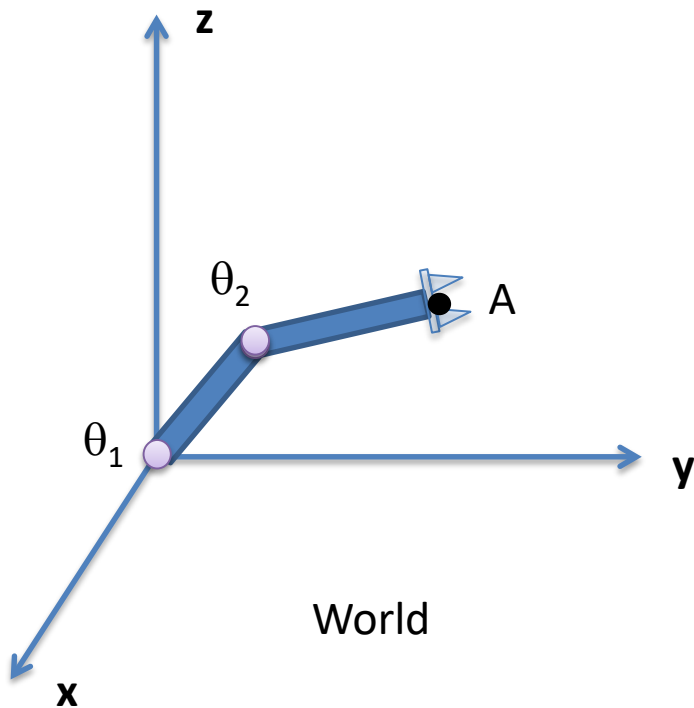
The set C of all possible configurations.

$$\mathbf{q} \in C$$

Read:

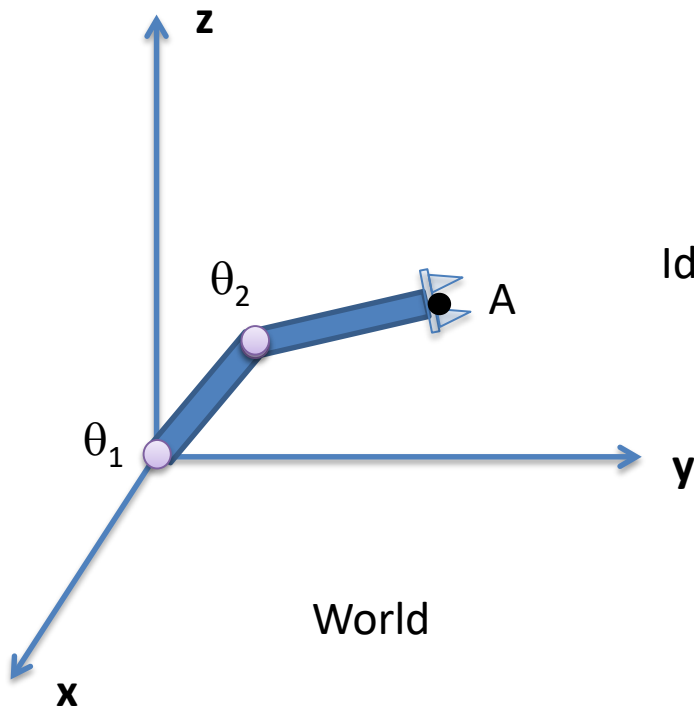
Spatial Planning: a Configuration Space
T. Lozano-Perez, 1980

Configuration Space

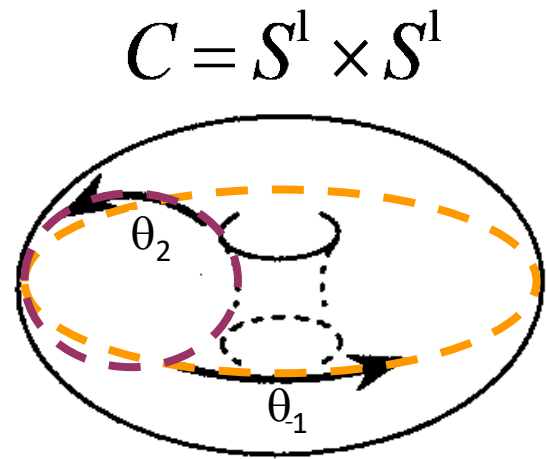


Topology of this space is in not necessarily the one of a Cartesian space.

Configuration Space



Identifying 0 and 2π



Configuration Space

Topology of this space is not necessarily the one of a Cartesian space.

Motion Planning

Motion planning:

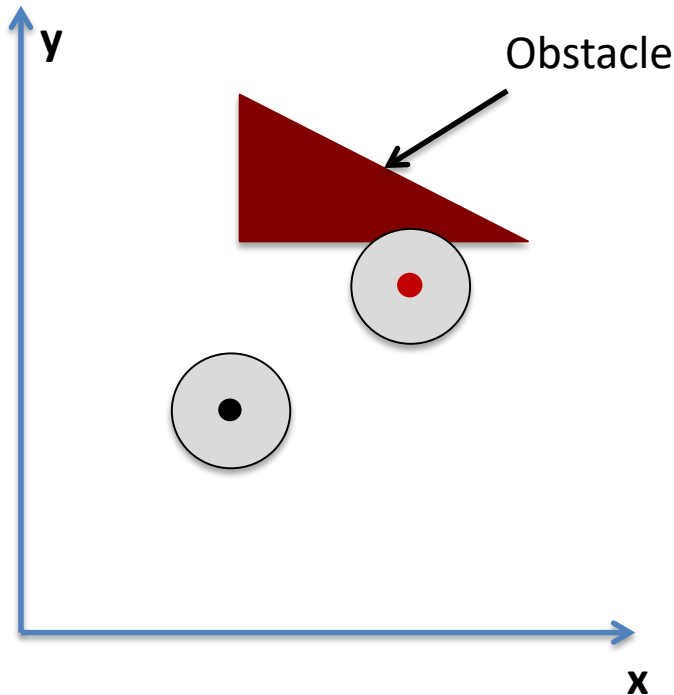
Instead of framing the motion planning problem as finding a path in the work space, we can define it as finding a path in the configuration space.

Problem:

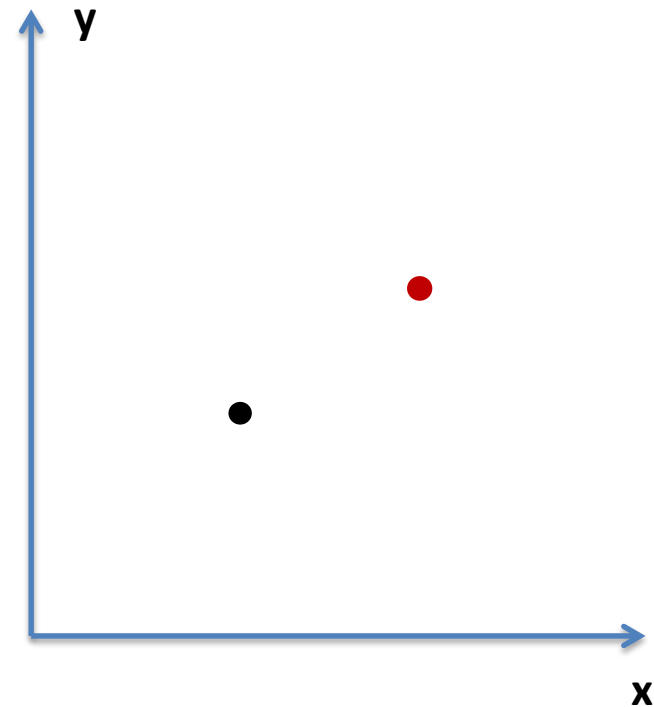
- 1) A World
- 2) An obstacle region $O \subset W$
- 3) A robot A defined in W : $A \subset W$
- 4) Find a collision free path from $\mathbf{q}_I \in C$ to $\mathbf{q}_G \in C$

Configuration Space Obstacles

Where do we put the obstacle in configuration space?



World



Configuration Space

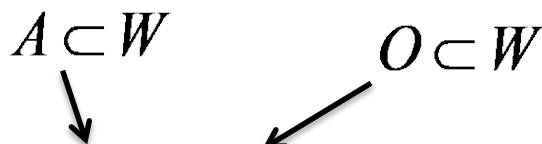
$$\mathbf{q} = (x, y) \in \mathbb{R}^2$$

Configuration Space Obstacles

Obstacle representation in configuration space

- Obstacles make certain configurations in configuration space impossible
- Remove the configurations in configuration space that cause the robot to collide with obstacles or cause some specific links of the robot to collide

Obstacle Region:

$$C_{obs} = \{q \in C \mid A(q) \cap O \neq \emptyset\}$$


The world contains
obstacle region O and
rigid robot A

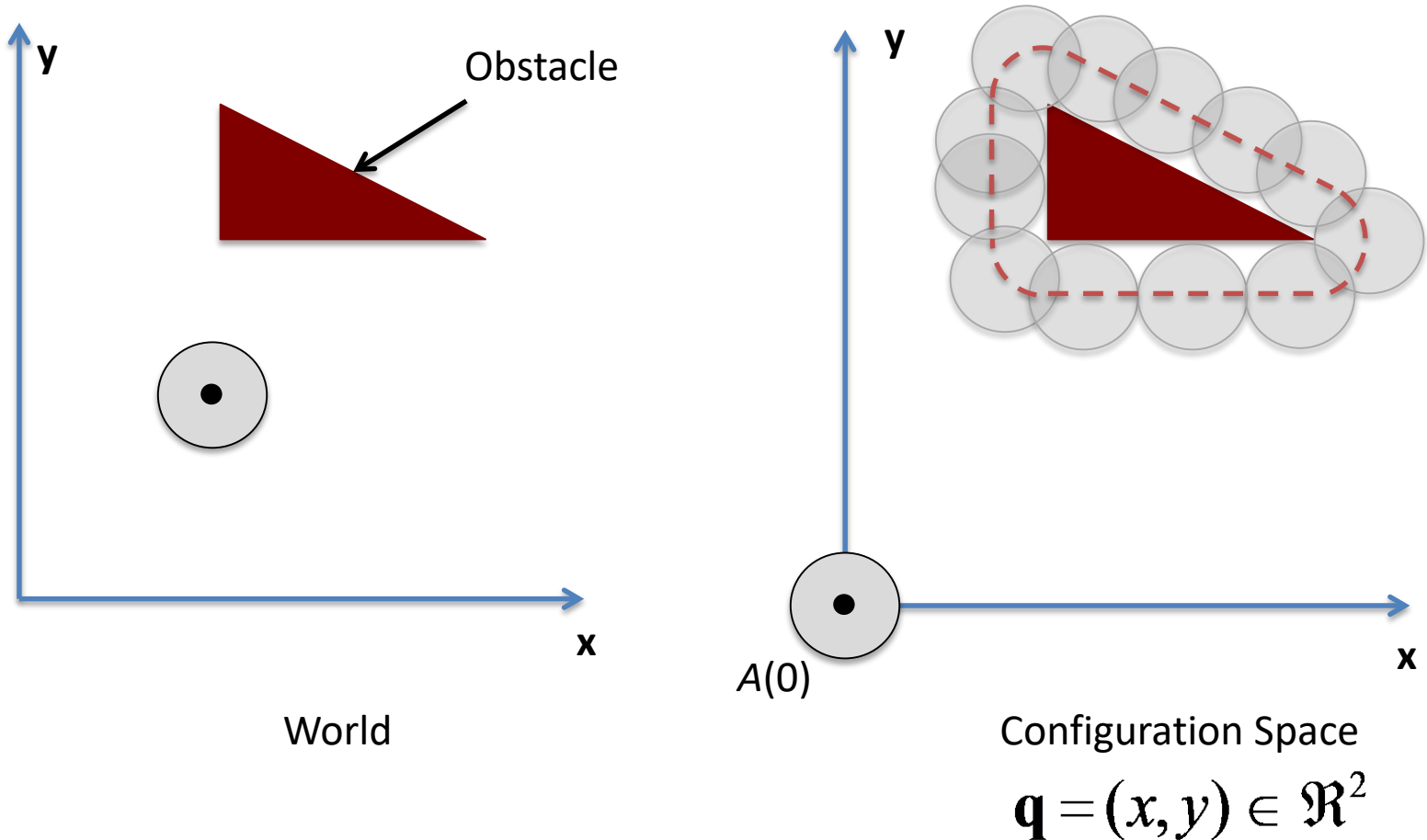
Free Space:

$$C_{free} = C \setminus C_{obs}$$

Note: C_{free} is open set. Sometimes need to consider closure of C_{free} .

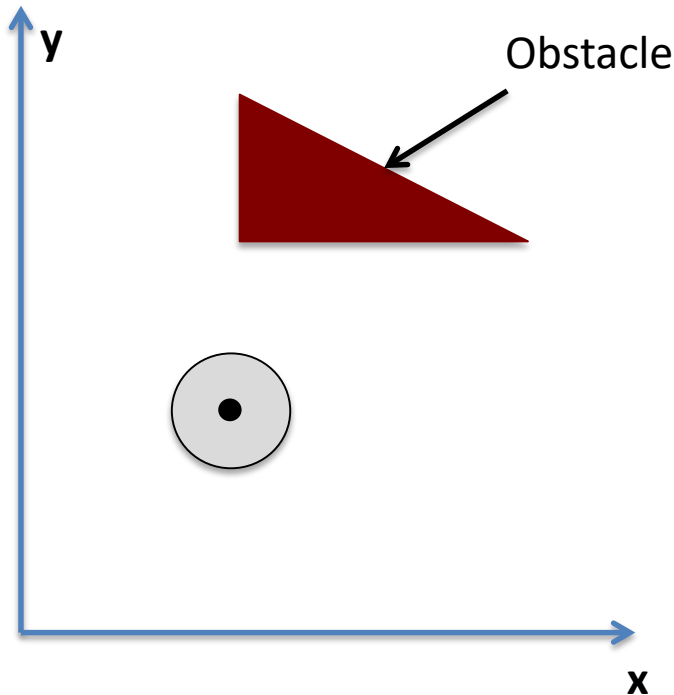
Configuration Space Obstacles

Where do we put the obstacle in configuration space?

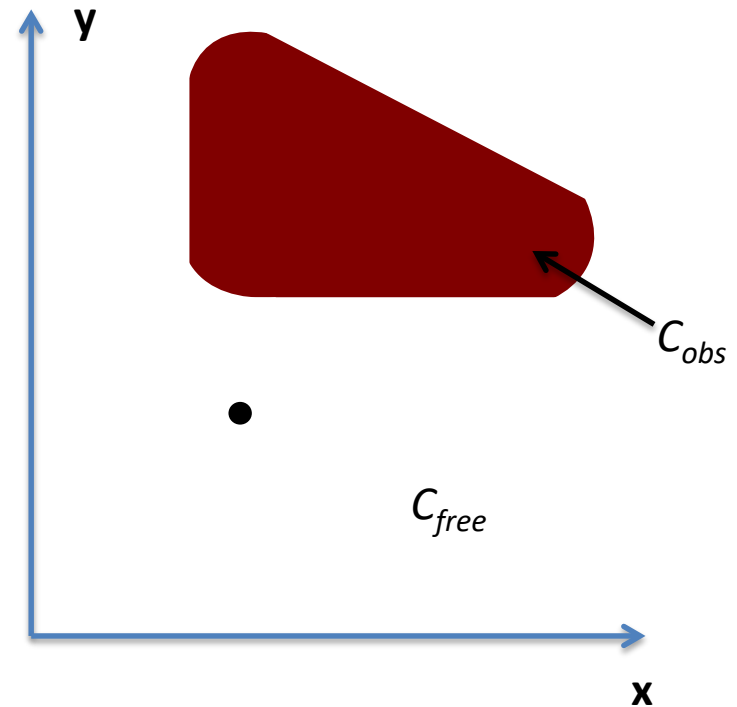


Configuration Space Obstacles

Where do we put the obstacle in configuration space?



World



Configuration Space

$$\mathbf{q} = (x, y) \in \mathbb{R}^2$$

Configuration Space Obstacles

Minkowski Difference

Formally we define the obstacle configuration space with the Minkowski difference, if

- $C = \mathbb{R}^n$, $n=1,2,3$
- Robot is rigid body

Minkowski Sum:

$$A \oplus B = \{a + b \mid a \in A, b \in B\}$$

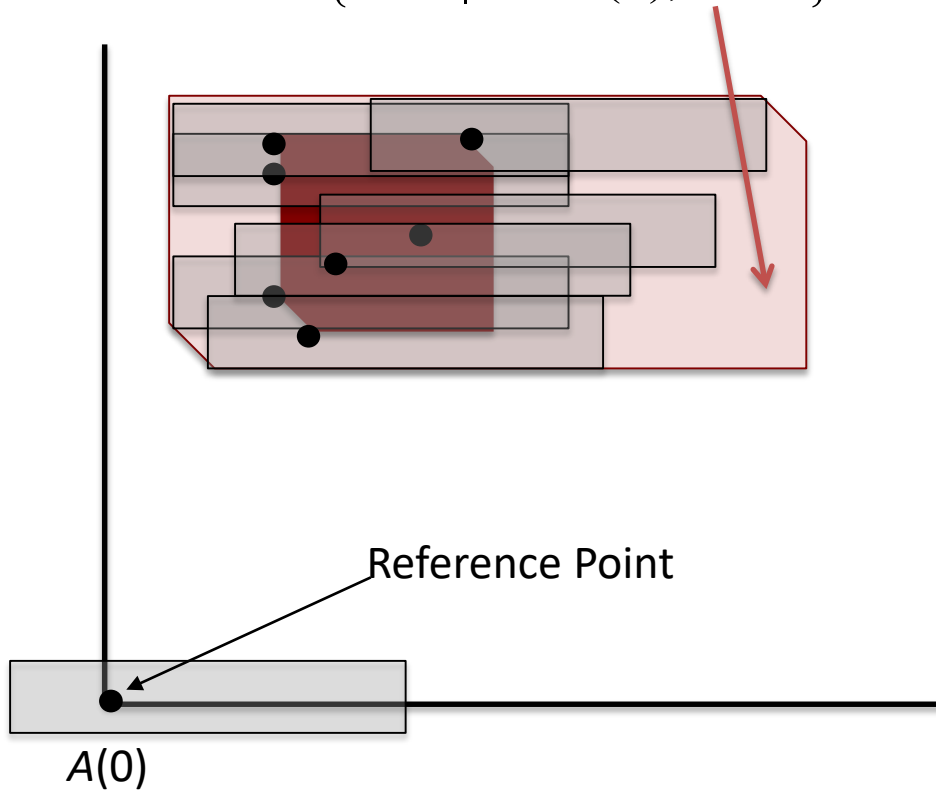
Minkowski Difference:

$$A \ominus B = \{a - b \mid a \in A, b \in B\}$$

Configuration Space Obstacles

Minkowski Difference

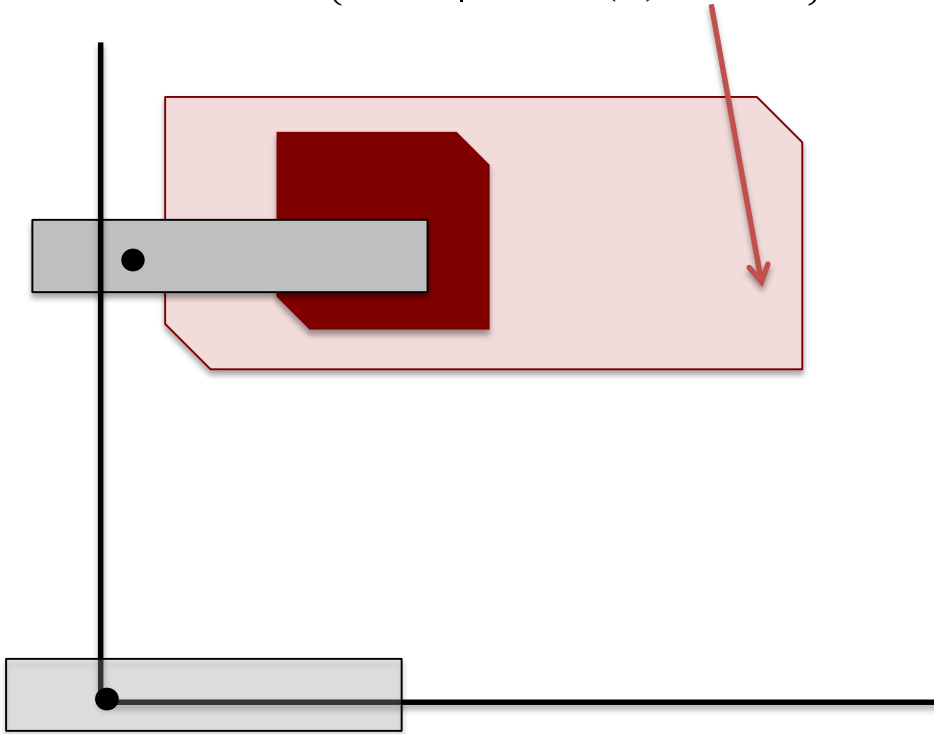
$$O \oplus A = \{o + a \mid a \in A(0), o \in O\}$$



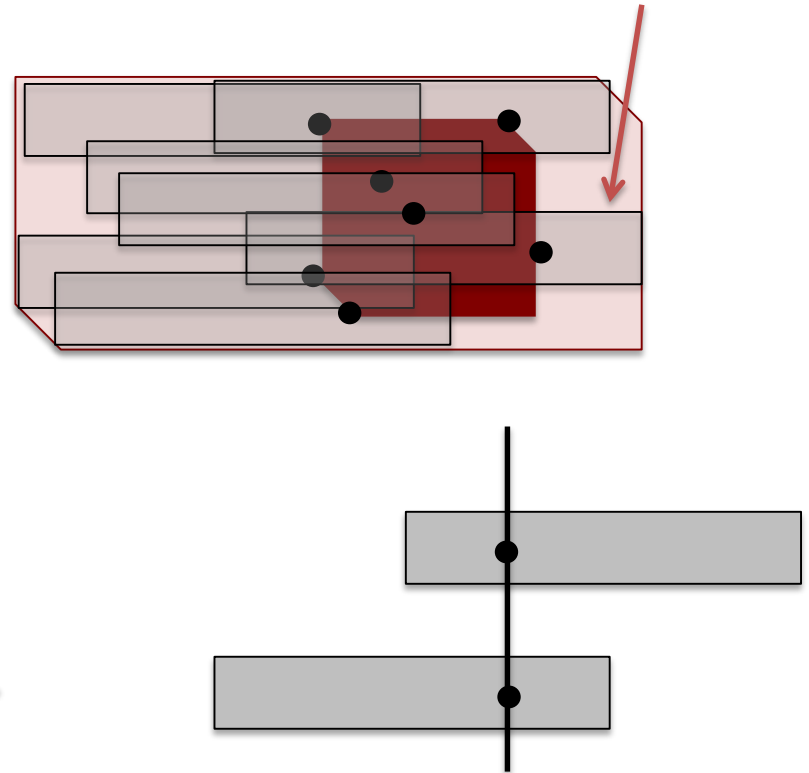
Configuration Space Obstacles

Minkowski Difference

$$O \oplus A = \{o + a \mid a \in A(0), o \in O\}$$



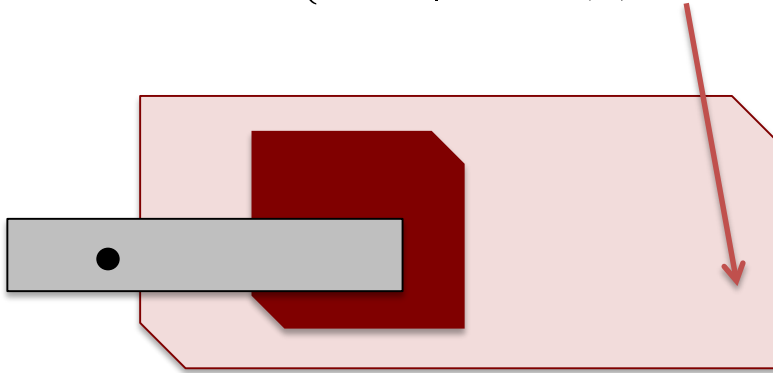
$$O \ominus A = \{o - a \mid a \in A, o \in O\}$$



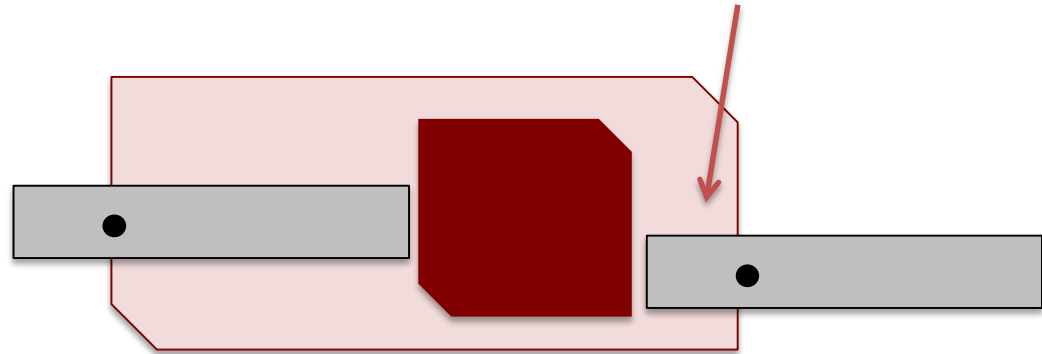
Configuration Space Obstacles

Minkowski Difference

$$O \oplus A = \{o + a \mid a \in A(0), o \in O\}$$



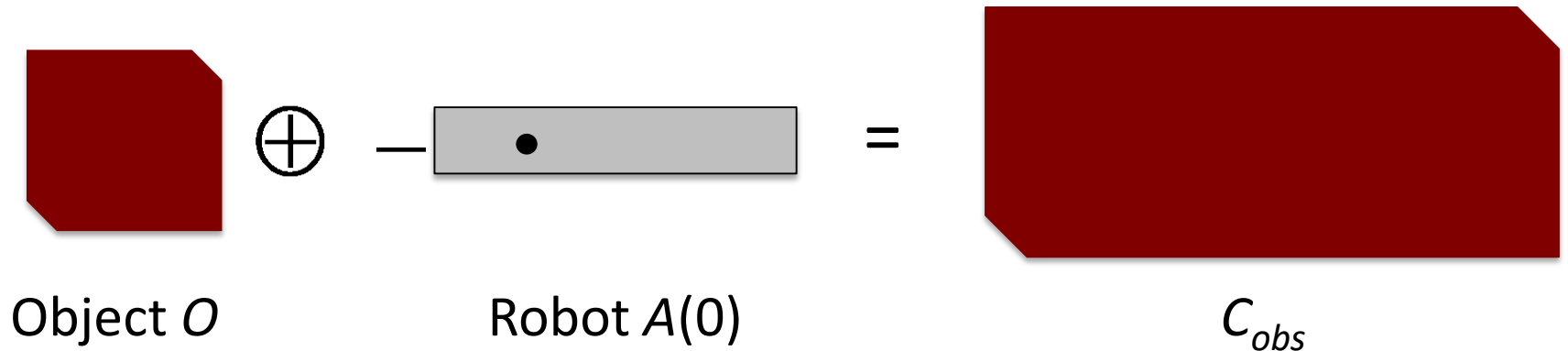
$$O \ominus A = \{o - a \mid a \in A, o \in O\}$$



Configuration Space Obstacles

Minkowski Difference

$$\begin{aligned}C_{obs} &= O \oplus -A(0) \\ &= O \oplus A(0)\end{aligned}$$



Gets difficult in high dimensions.

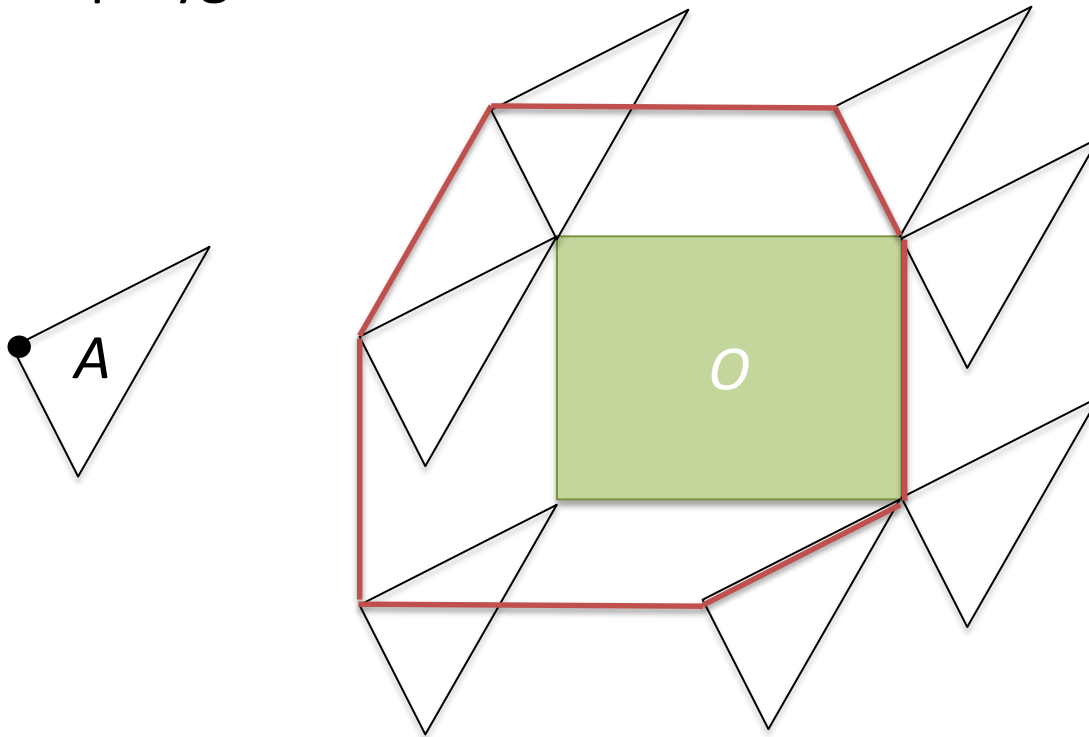
Configuration Space Obstacles

Alternatively:
Convex Hull of Vert(O) [-] Vert(A)

Star algorithm

2D Workspace, translation

C_{obs} : convex polygon

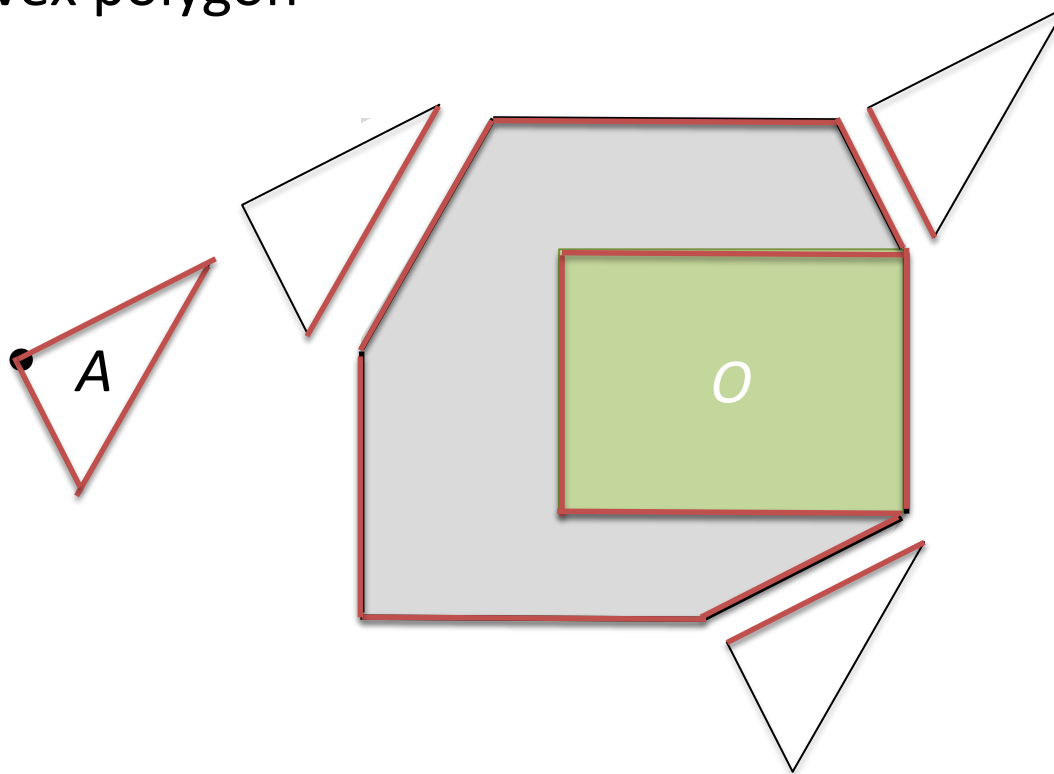


Configuration Space Obstacles

Star algorithm

2D Workspace, translation

C_{obs} : convex polygon



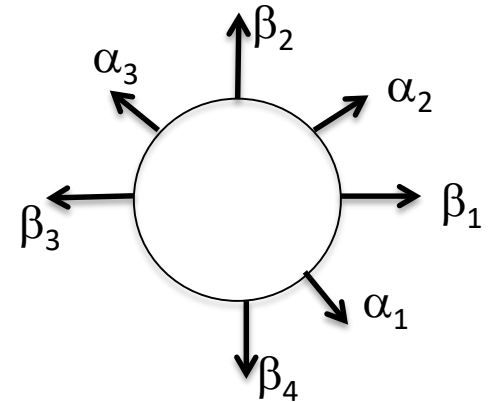
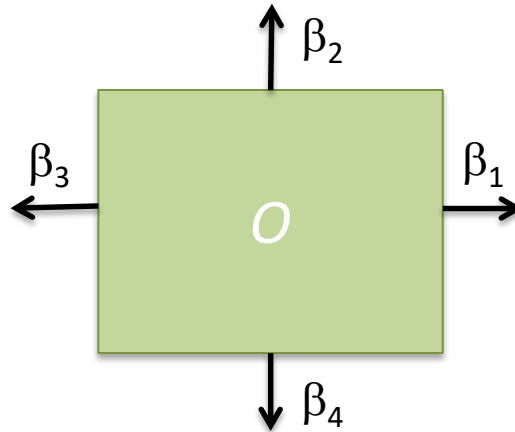
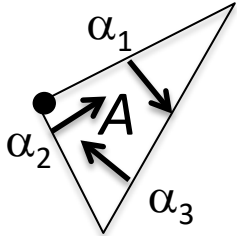
Key observation: every edge of C_{obs} is a translated edge from either A or O

Configuration Space Obstacles

Star algorithm

2D Workspace, translation

C_{obs} : convex polygon



Problem: Determine the order of these edges!

Let α_i denote the angles of the inward edge normals around A

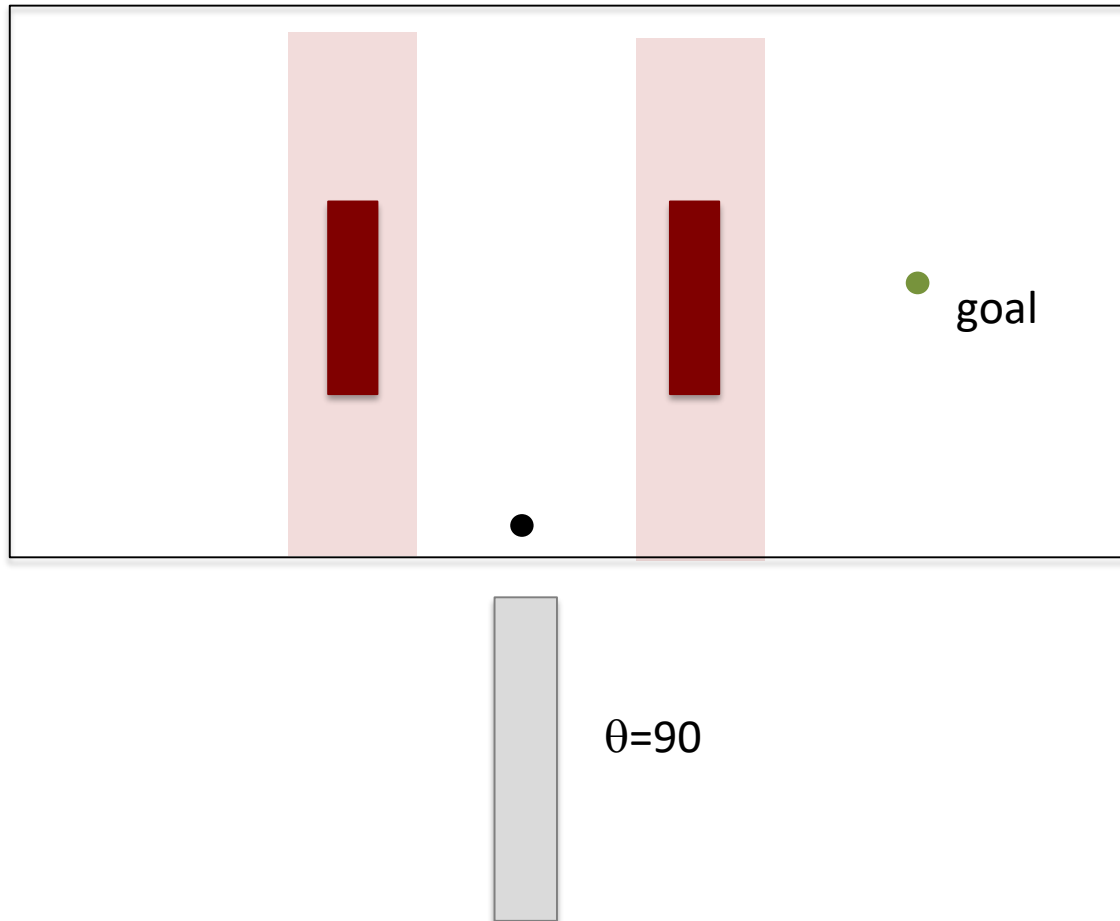
Let β_i denote the outward edge normals to O

Sort all angles in around S^1

Read:
Spatial Planning: a Configuration Space
T. Lozano-Perez, 1980

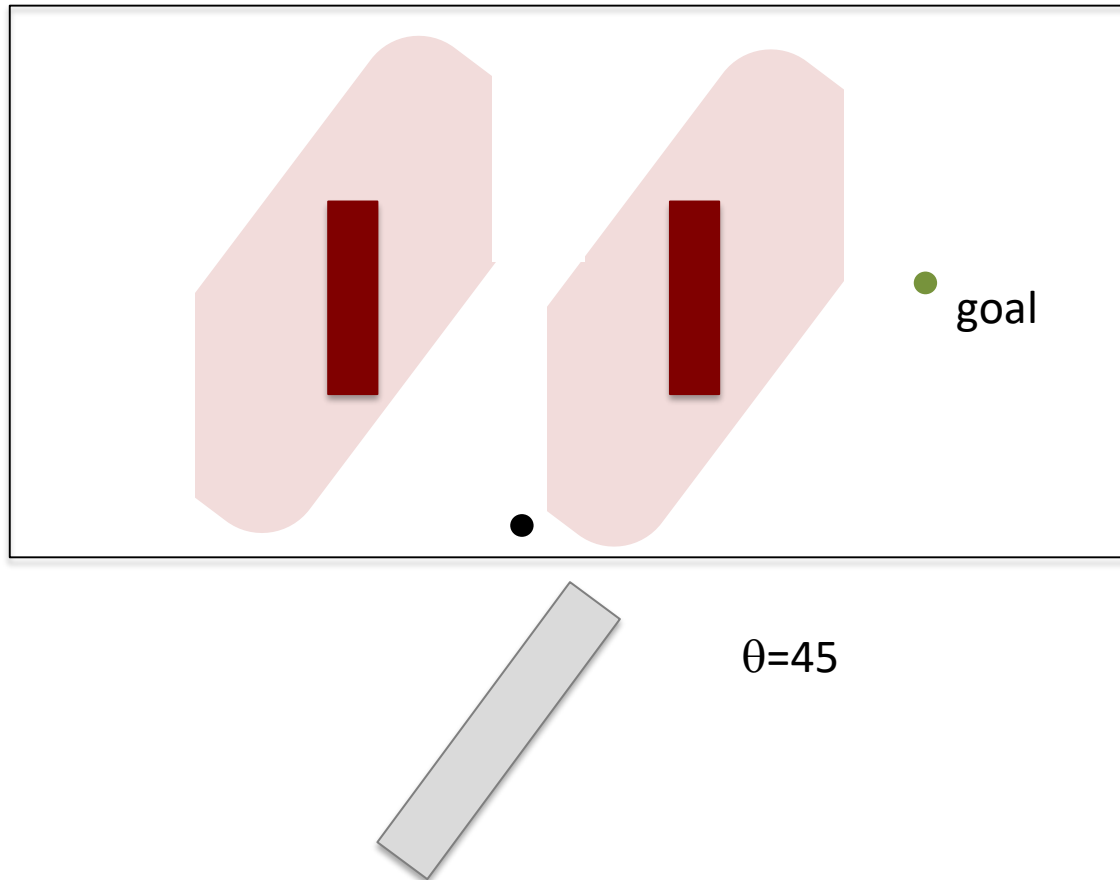
Configuration Space Obstacles

What happens when the robot translates *and* rotates in the plane?



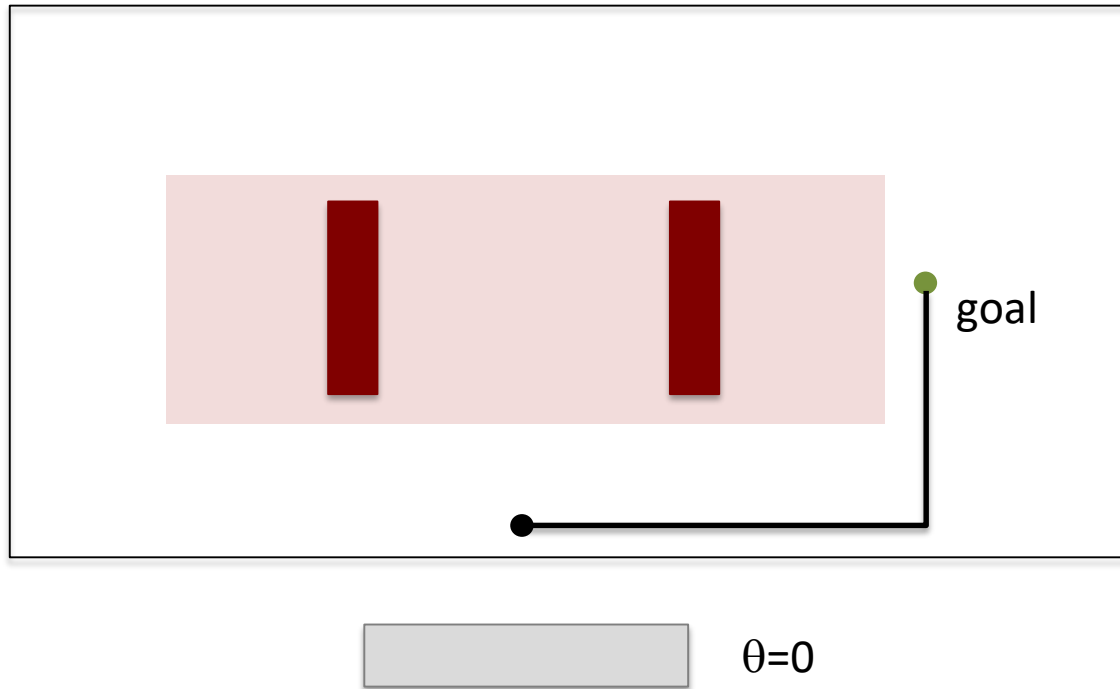
Configuration Space Obstacles

What happens when the robot translates *and* rotates in the plane?



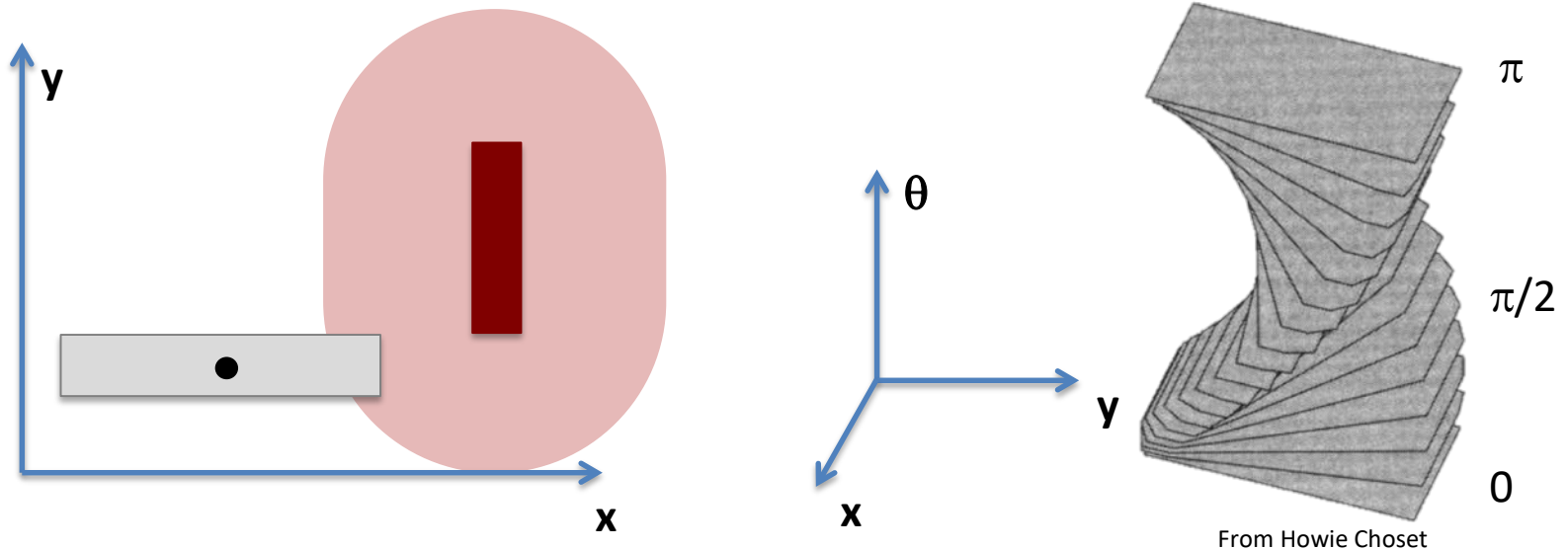
Configuration Space Obstacles

What happens when the robot translates *and* rotates in the plane?



Configuration Space Obstacles

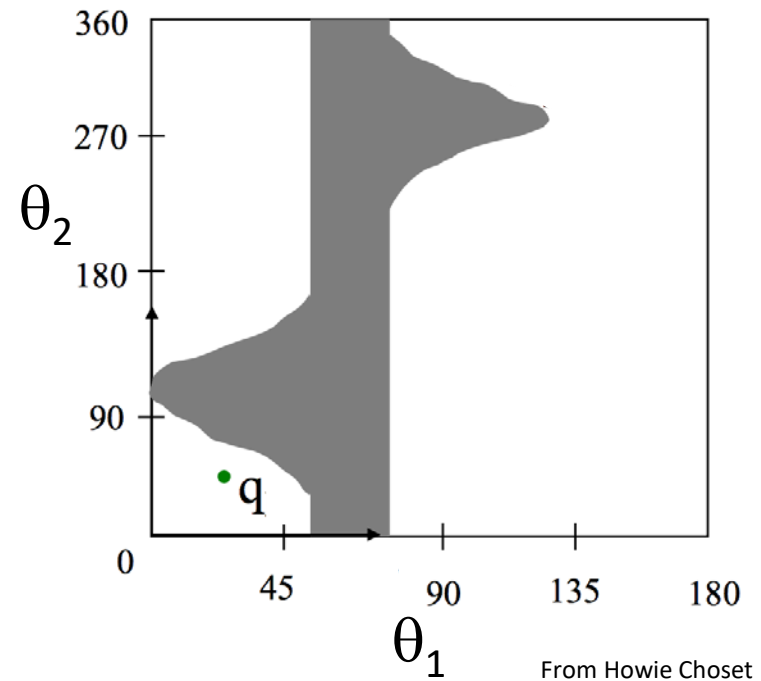
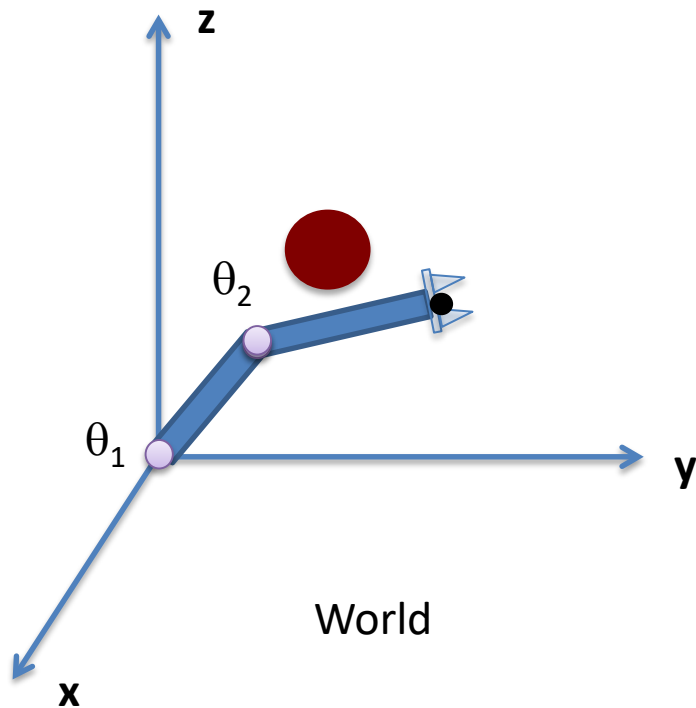
What happens when the robot translates *and* rotates in the plane?



Need to consider orientation, i.e., need to consider all possible contact interactions between all pairs of object and robot features!

Problem: Exponential run times!

Articulated Robots

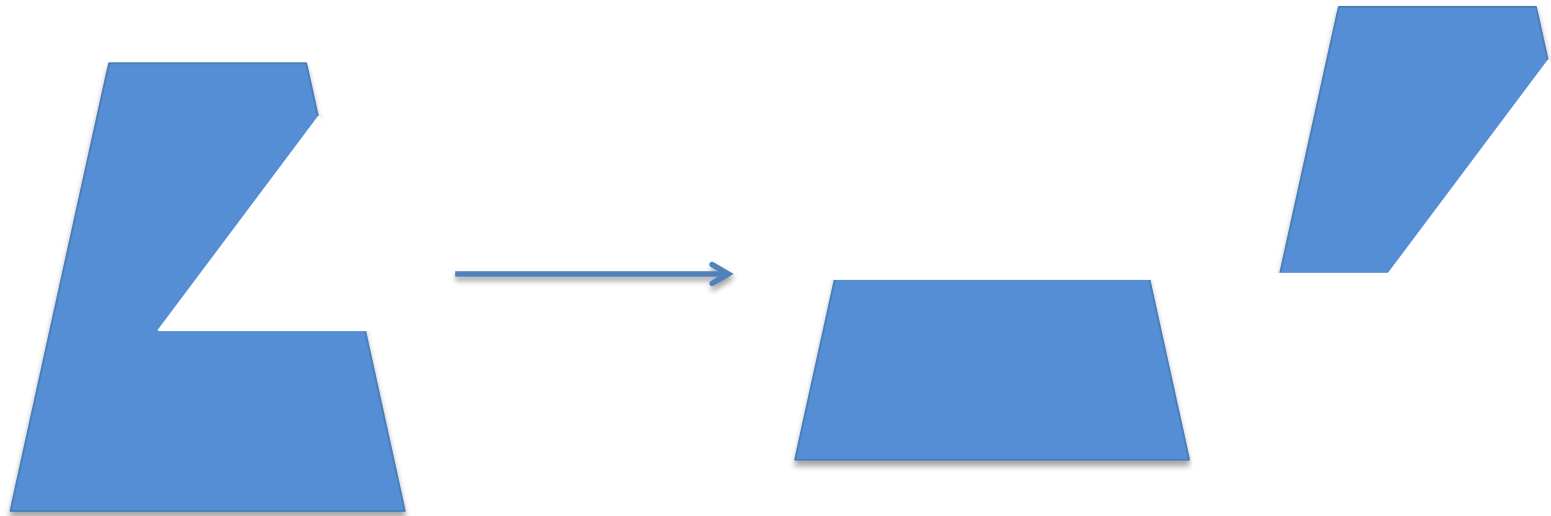


Configuration space

Non-Convex Case

$O \subset W$ is non-convex

Consider C_{obs} as the union of convex components of O



Motion Planning

Let $W = \mathbb{R}^m$ being the work space, $O \subset W$ the set of obstacles, $A(q)$ the robot in configuration $q \in C$

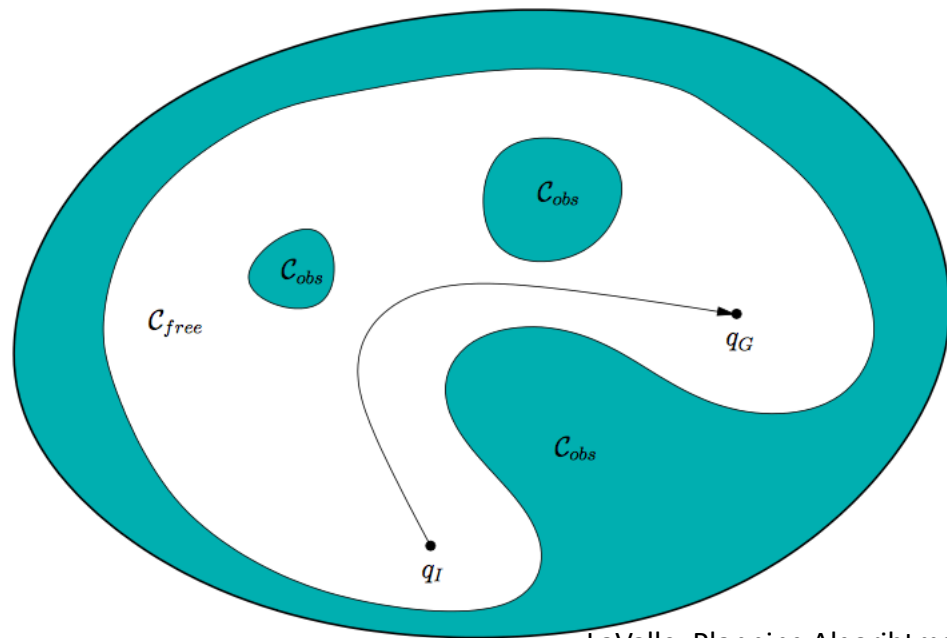
$$C_{obs} = \{q \in C \mid A(q) \cap O \neq \emptyset\}$$

$$C_{free} = C \setminus C_{obs}$$

Motion planning: find a continuous path

$$\tau : [0, 1] \rightarrow C_{free}$$

With $\tau(0) = q_I$, $\tau(1) = q_G$



LaValle, Planning Algorithms

Planning with robot being point in C-Space

Motion Planning

Continuous terrain needs to be discretized for path planning!

Except we'll see
optimization-based
approaches later

Two general approaches:

Combinatorial Planning

- Explicit characterization of C_{free}
- Capture connectivity of C_{free} into graph and do graph search to find solutions
- Often complete

Sampling based planning

- Sample in C-space
- Use collision detection to probe for collisions with obstacles
- Probabilistic complete

Completeness

Complete:

A motion planner always returns a solution if there exist one and returns a failure otherwise in bounded time.

Probabilistic Complete:

The probability of failing to find a path
The probability tends to zero as the number of sampled points increases. No time bound.

Road maps

Road map R , is a union of 1-D curves s.t. for all $\mathbf{q}_I \in C_{free}$ and $\mathbf{q}_G \in C_{free}$ there exist a path between $\mathbf{q}_I$ and $\mathbf{q}_G$ if:

- 1) Accessibility: There is a path to $\mathbf{q}_I \in C_{free}$ to a $\mathbf{q}' \in R$
- 2) Departability: There is a path from a $\mathbf{q}_G \in C_{free}$ to a $\mathbf{q}'' \in R$
- 3) Connectivity: There exist a path in R between $\mathbf{q}'$ and $\mathbf{q}''$

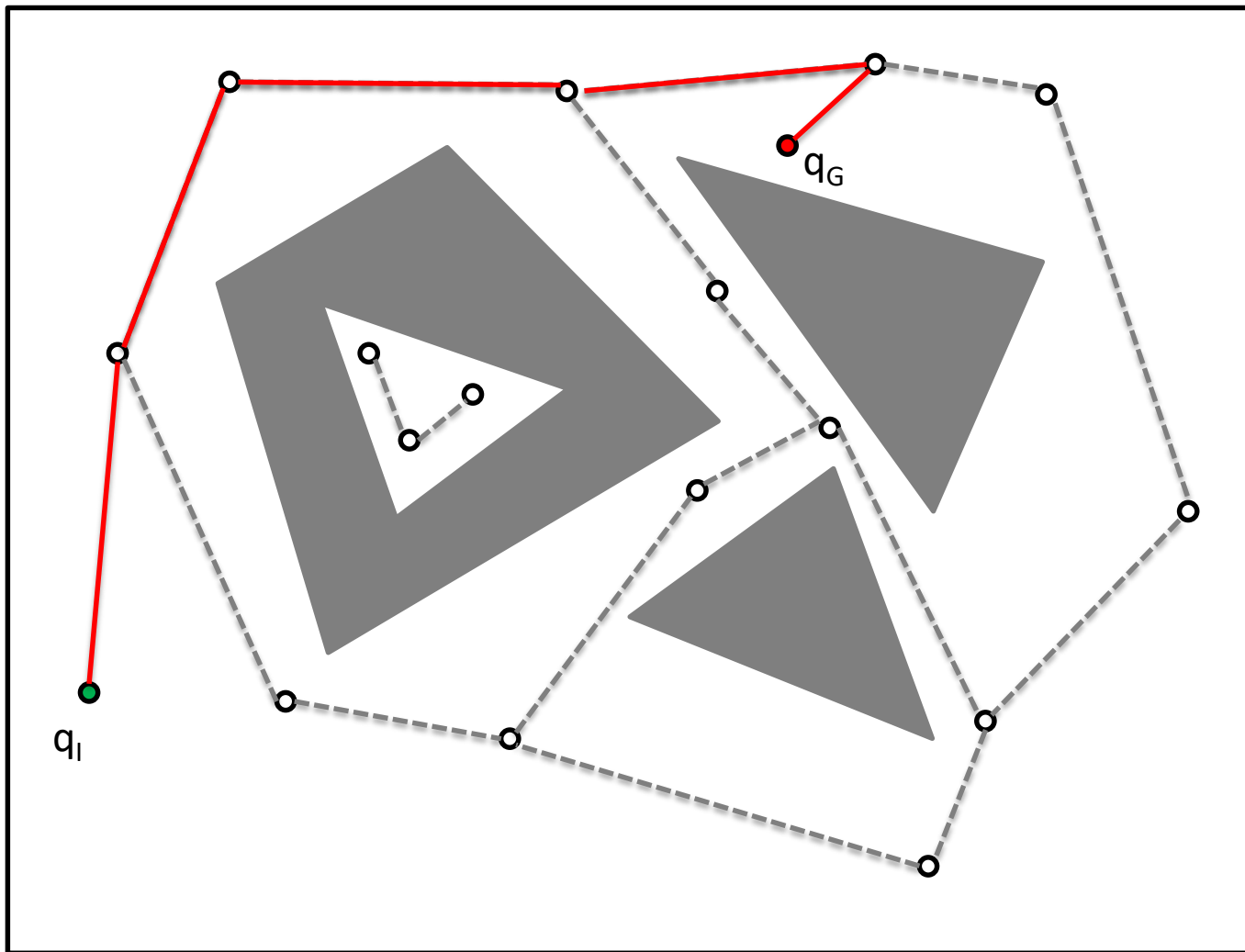
Building R :

- Nodes: $\mathbf{q} \in C_{free}$ (or its boundary)
- Edge between nodes: if there is a free path between the two nodes

Path Planning:

- 1) Connect $\mathbf{q}_I$ and $\mathbf{q}_G$ to road map to $\mathbf{q}'$ and $\mathbf{q}''$
- 2) Find path from $\mathbf{q}'$ and $\mathbf{q}''$ in R

Road maps



Combinatorial Planning

Main combinatorial planning techniques:

- Visibility graphs
- Voronoi diagrams
- Exact cell decomposition
- Approximate cell decomposition

Result: Graph, often road map

Require: Explicit C_{obs} representation

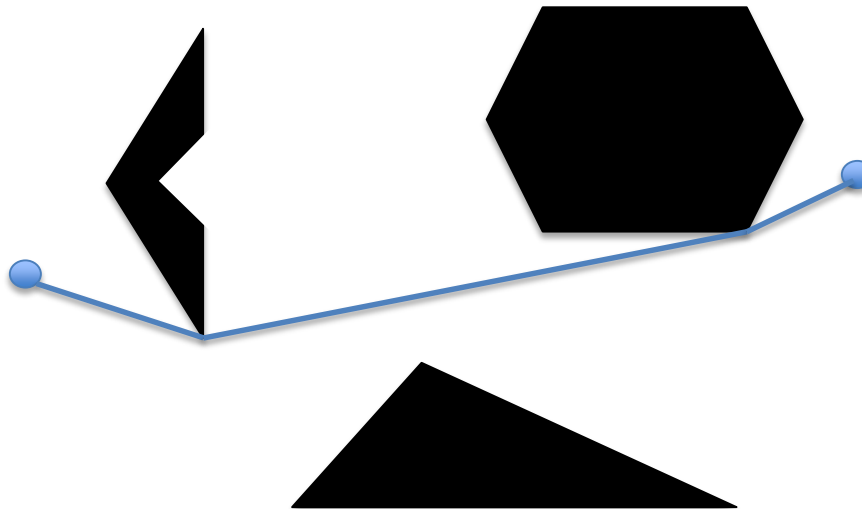
Visibility graph

Idea:

Construct a path as polygonal line connecting q_I and q_G through vertices of C_{obs}

Why?

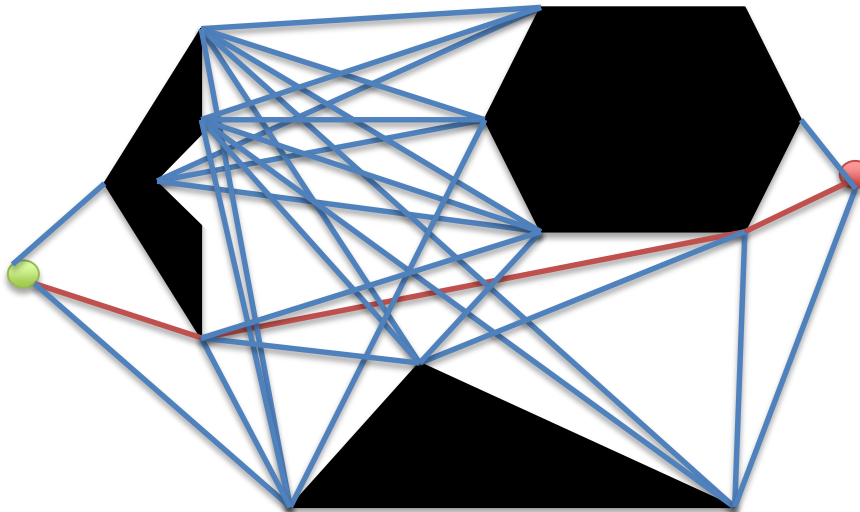
Any collision-free path can be transformed into a piecewise linear path that bends only at the obstacle vertices



Visibility graph

A graph defined by:

- Nodes: q_I, q_G , obstacle vertex
- Edges: either obstacle edge or edge between nodes that does not intersect with any obstacle

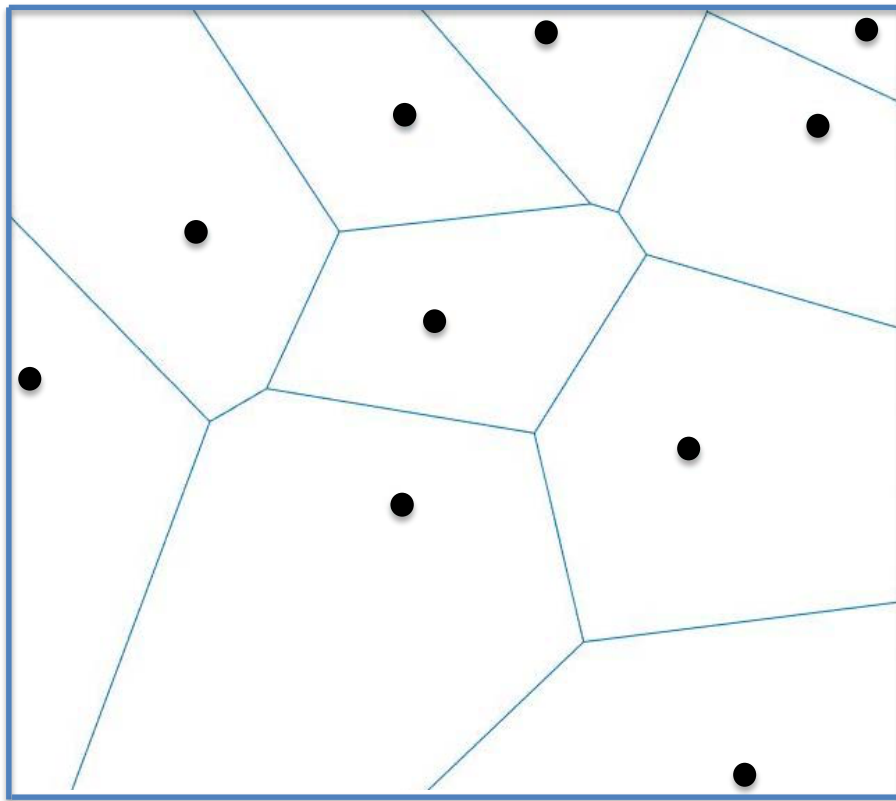


Runtime: $O(n^2 \log n)$

Shortest Path Roadmap

Voronoi Diagram

Idea: Divide the space into regions closest to a particular 'site'



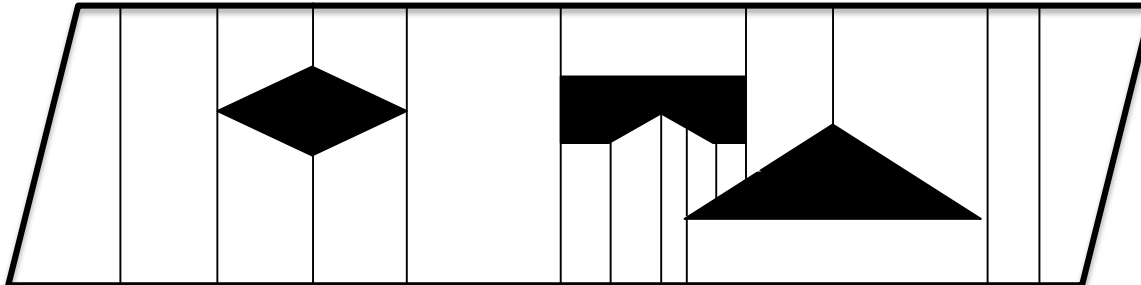
Cell Decomposition

Idea:

Decompose C_{free} into regions (cells) and create connectivity graph that represents adjacencies

Two types of approach:

- **Exact Cell Decomposition:** Union of cells corresponds to C_{free}
- **Approximate Cell Decomposition:** Decomposes C_{free} into cells with pre-defined shape; union of cells typically only approximates C_{free} space

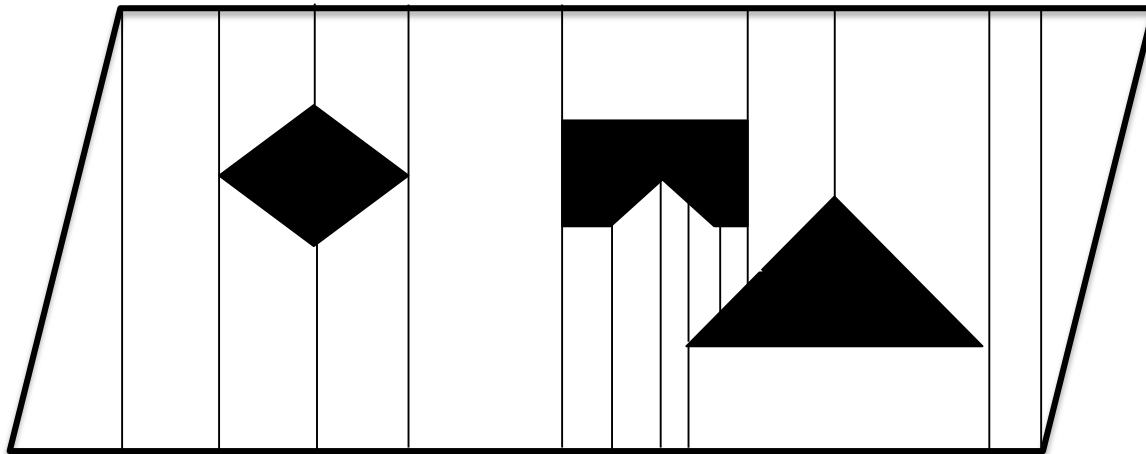


Exact Cell Decomposition

Example: Trapezoid Decomposition

Each cell is either triangle or trapezoid and is formed by:

- 1) Draw vertical line segments by shooting rays upward and downward from each polygonal vertex

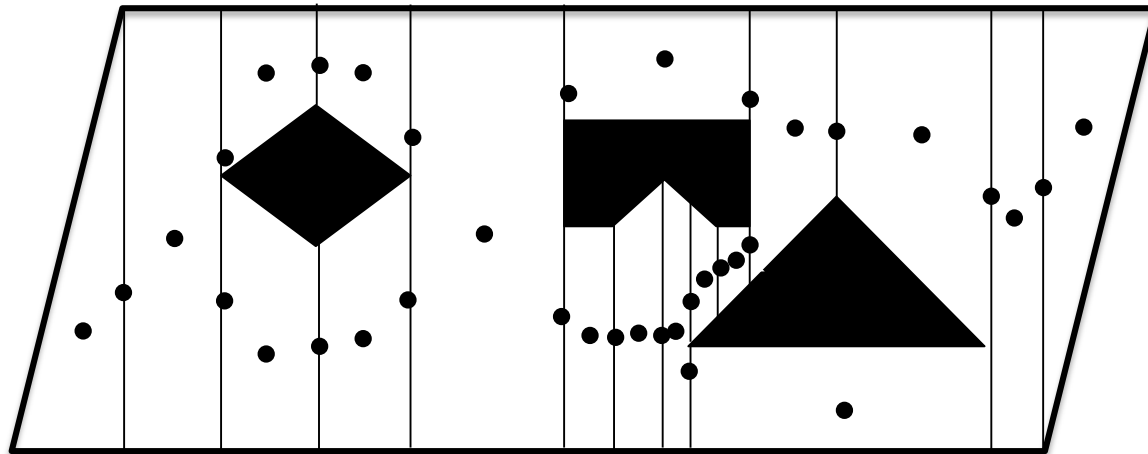


Exact Cell Decomposition

Example: Trapezoid Decomposition

Each cell is either triangle or trapezoid and is formed by:

- 1) Draw vertical line segments by shooting rays upward and downward from each polygonal vertex
- 2) Each cell corresponds to one node in connectivity graph



Exact Cell Decomposition

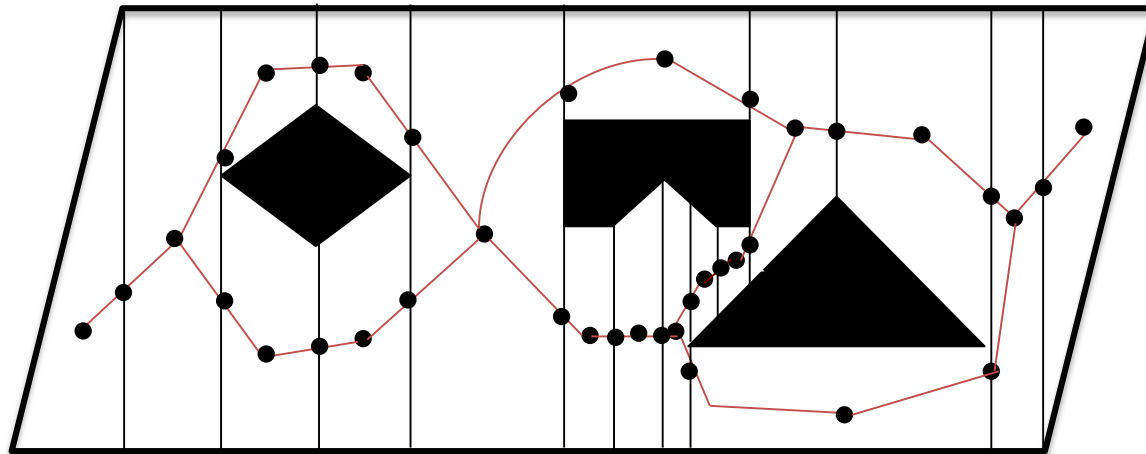
Example: Trapezoid Decomposition

Each cell is either triangle or trapezoid and is formed by:

- 1) Draw vertical line segments by shooting rays upward and downward from each polygonal vertex
- 2) Each cell corresponds to one node in connectivity graph
- 3) Nodes are connected if the corresponding cells share a common boundary

Nodes: Cells

Edges: between all adjacent cells



Run time best: $O(n \log n)$
N number of cells

Approximate Cell Decomposition

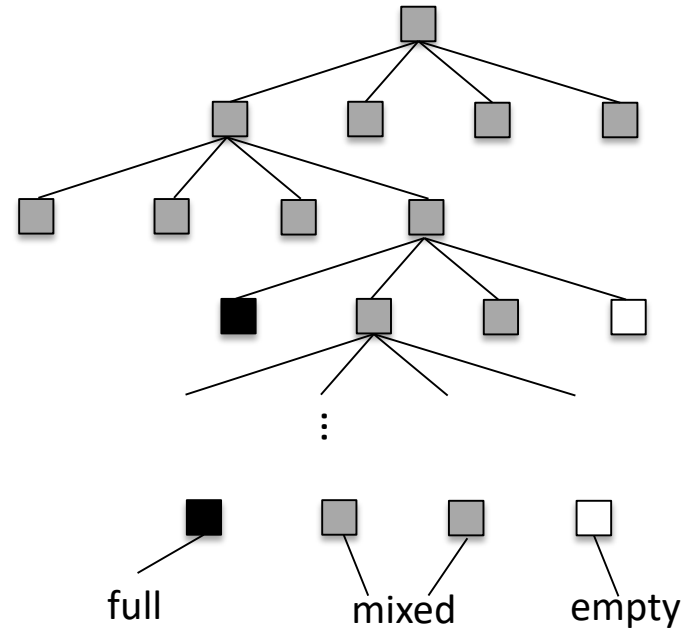
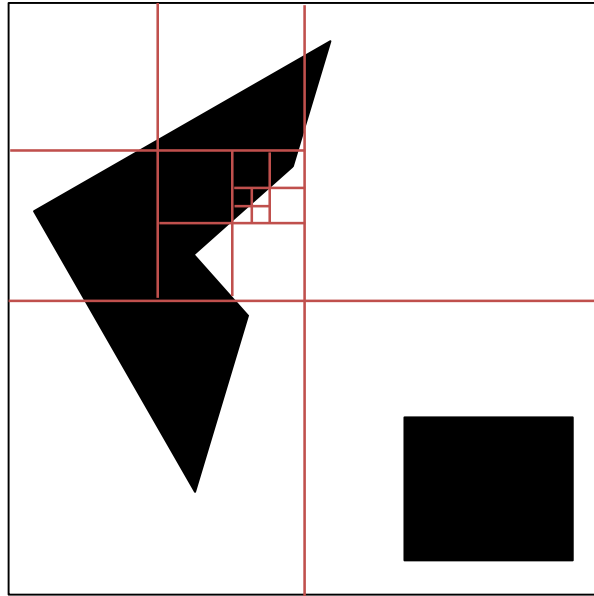
Exact cell decomposition can be inefficient for complex objects and problems

Idea: Use cells with the same simple predefined shape, e.g., rectangle, square

- Hierarchical space decomposition: Quadtree or Octree decomposition
- The union of those non-overlapping cells is contained in C_{free}

Approximate Cell Decomposition

Example: Quadtree decomposition



Pros:

- Iteration of repetitive and simple computation
- Simple to implement
- Can be made complete

Combinatorial Planning

Summary

Provably hard - PSPACE hard [Reif]
Canny's Roadmap algorithm is a complete algorithm for general motion planning problems. It is exponential in the degrees of freedom of the robot. But, no practical implementation exists.

Pros

Completeness



In finite time:

- Report failure if no solution exist
- Report solution, otherwise

Cons

Become intractable with increasing C-space dimensionality and/or object complexity



Combinatorial explosion in representing robot, obstacles and C_{obs} , especially with manipulators

How can we do better?

Sampling Based Planning

Sampling based planning algorithms:

- Probabilistic Road Maps (PRM)
- Rapidly Exploring Random Trees
- Derivates of above

More **efficient in most practical problems** but weaker guarantees:

- Probabilistic Complete: The probability tends to 0 that no solution exist if not found yet

Main planning approaches found in manipulation!

Sampling Based Planning

Sampling Based Planning

```
graph TD; A[Sampling Based Planning] --> B[Multi-Query]; A --> C[Single-Query];
```

Multi-Query

Two phases:

- 1) Build Roadmap
- 2) Search

Assumes fixed obstacle setting over multiple queries.

Probabilistic Road Maps

[Kavaki et al.: Probabilistic roadmaps for path planning in high dimensional configuration spaces; 1996]

Single-Query

Iteratively build tree with search.

Rapidly exploring Random Trees

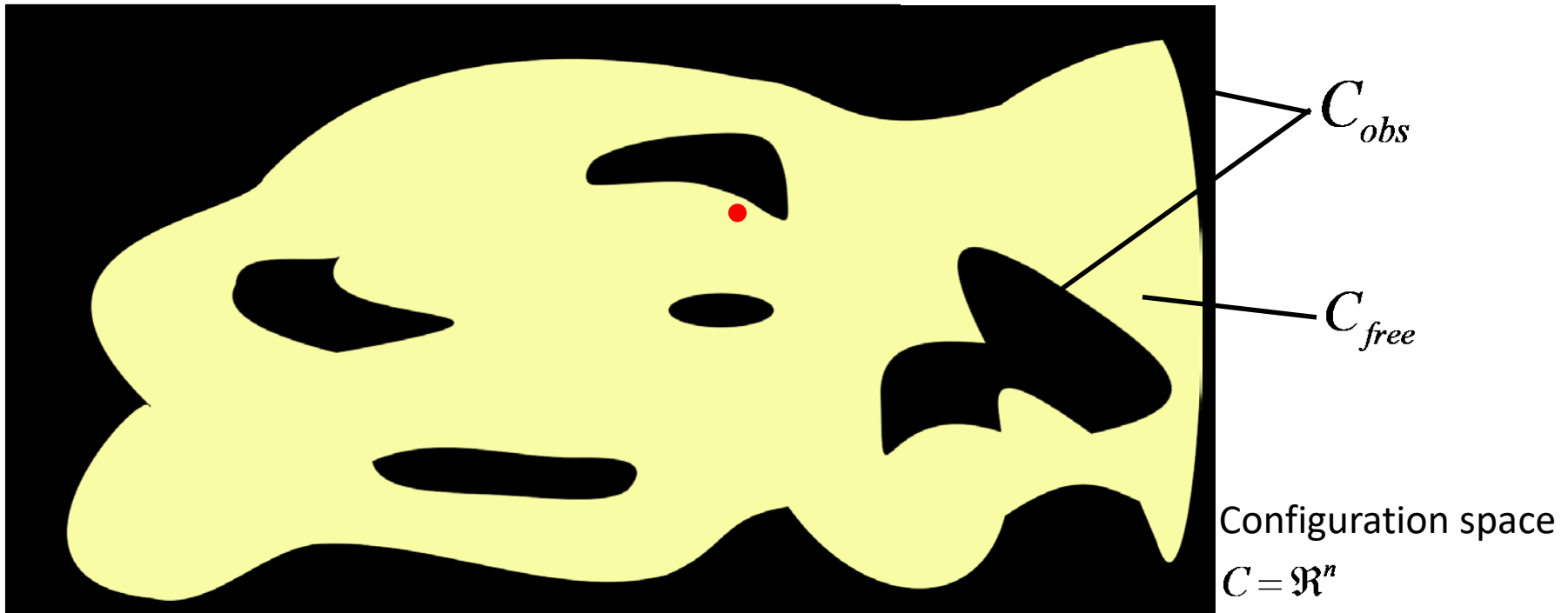
[Kuffner and LaValle; Rapidly exploring Random Trees; 2000]

Probabilistic Road Maps

$$G=(V,E) \quad V \leftarrow \emptyset, E \leftarrow \emptyset$$

Construction Step:

Generate a uniformly distributed robot configuration $\mathbf{q}_{new}$



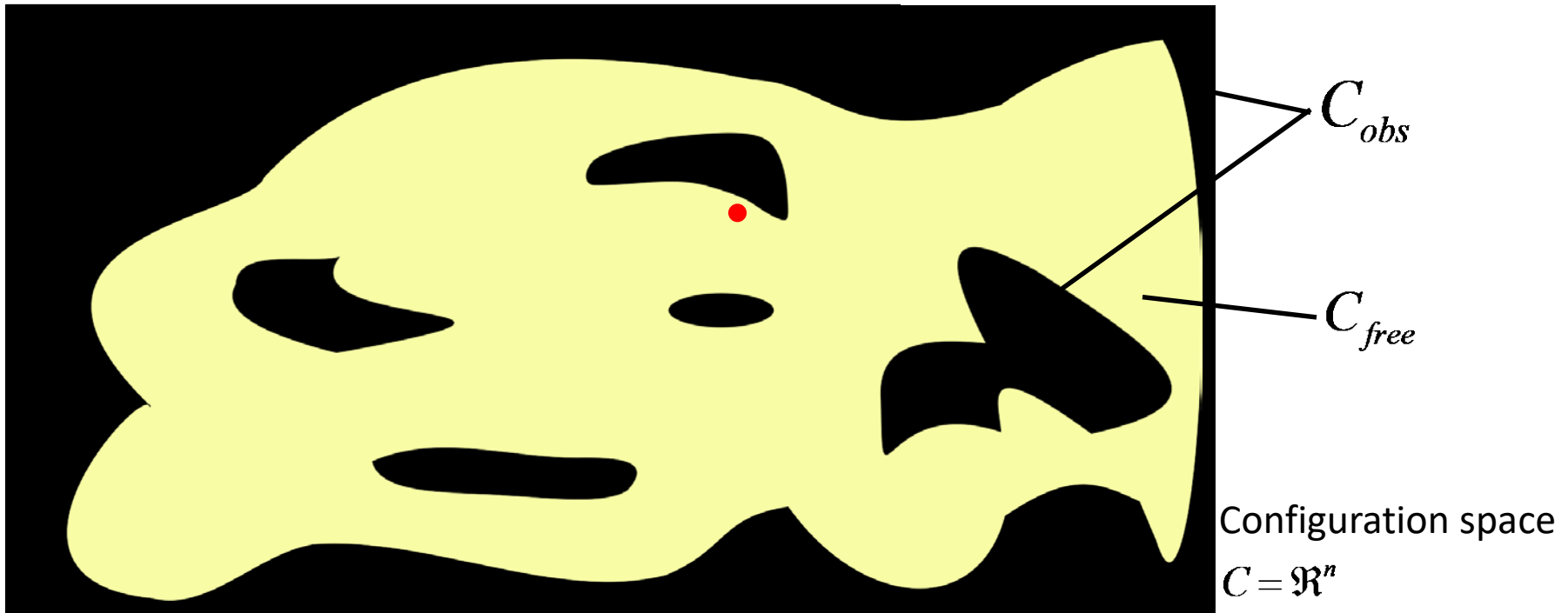
Probabilistic Road Maps

$$G=(V,E) \quad V \leftarrow \emptyset, E \leftarrow \emptyset$$

Construction Step:

$\mathbf{q}_{new} \leftarrow$ random sample in C
if $\mathbf{q}_{new} \in C_{free}$

Use collision checker to test if $\mathbf{q}_{new}$ is in C_{free}
or in C_{obs}



Probabilistic Road Maps

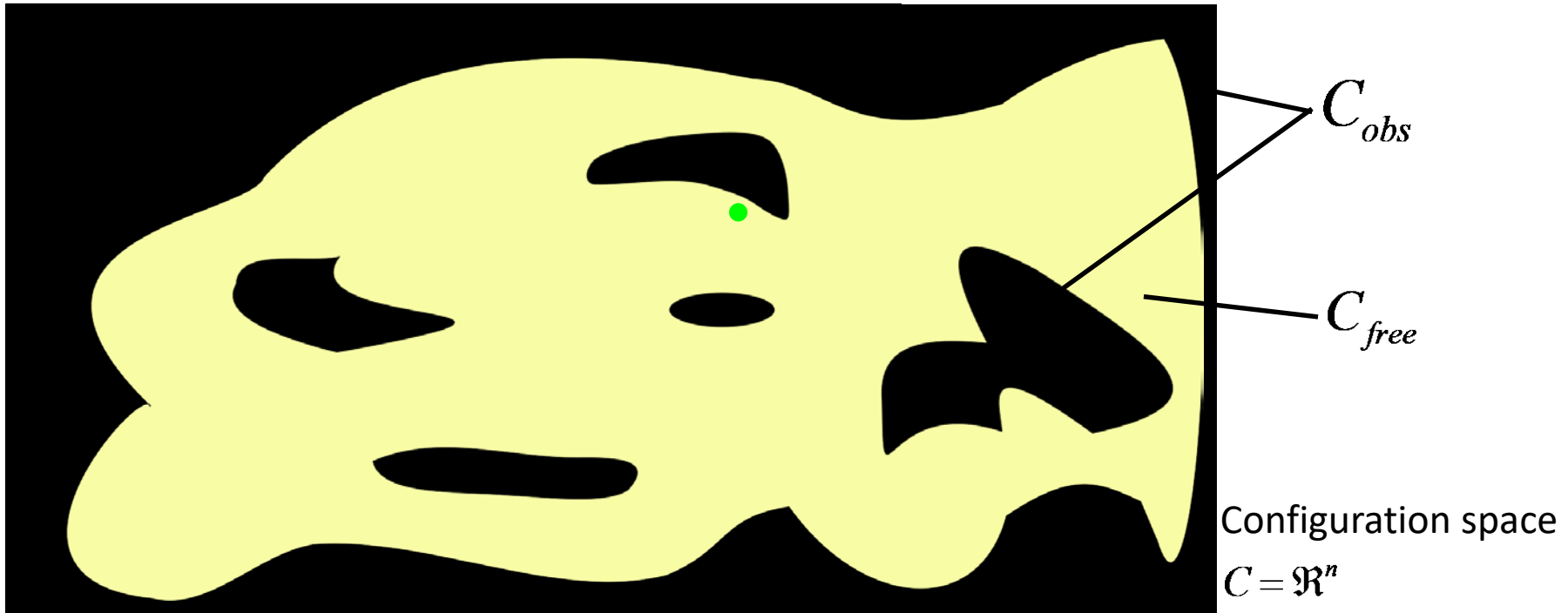
$$G=(V,E) \quad V \leftarrow \emptyset, E \leftarrow \emptyset$$

Construction Step:

$\mathbf{q}_{new} \leftarrow$ random sample in C

if $\mathbf{q}_{new} \in C_{free}$
add $\mathbf{q}_{new}$ to V

If $\mathbf{q}_{new} \in C_{free}$ add $\mathbf{q}_{new}$ to G ;
If not, sample new



Probabilistic Road Maps

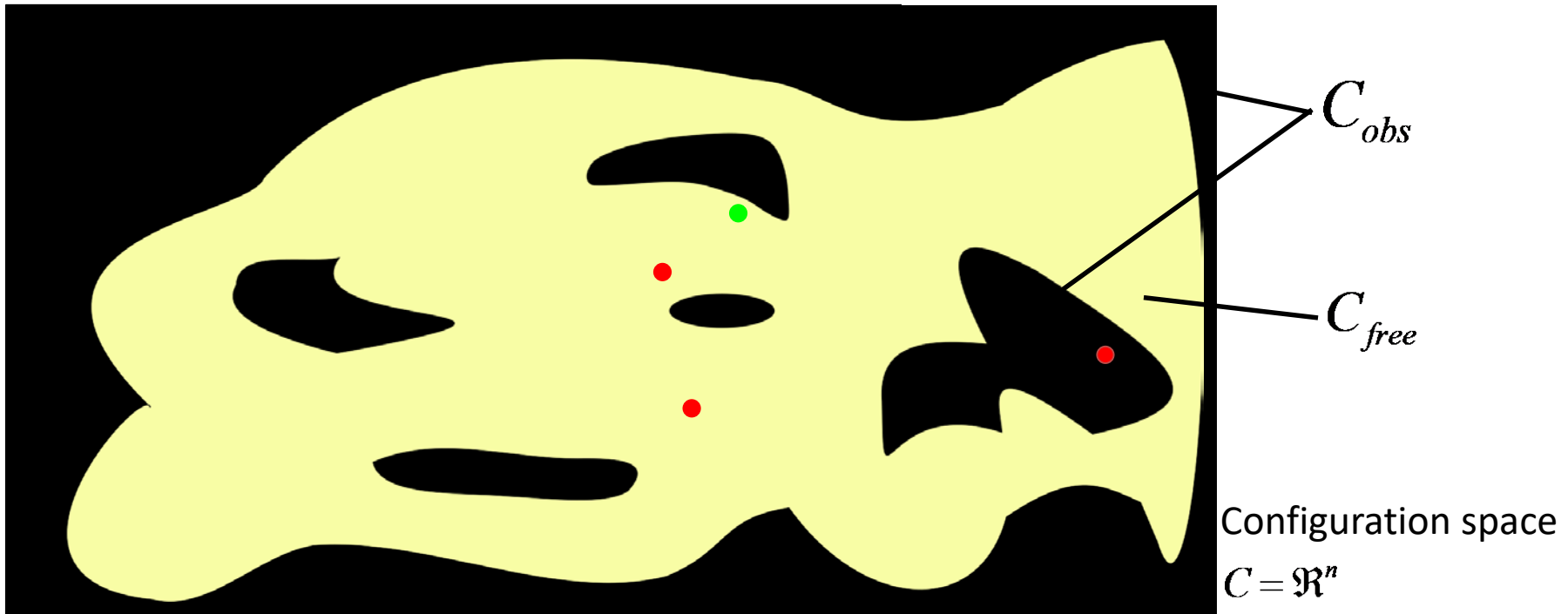
$$G=(V,E) \quad V \leftarrow \emptyset, E \leftarrow \emptyset$$

Construction Step:

$\mathbf{q}_{new} \leftarrow$ random sample in C

if $\mathbf{q}_{new} \in C_{free}$
add $\mathbf{q}_{new}$ to V

If $\mathbf{q}_{new} \in C_{free}$ add $\mathbf{q}_{new}$ to G ;
If not, sample new



Probabilistic Road Maps

$$G=(V,E) \quad V \leftarrow \emptyset, E \leftarrow \emptyset$$

Construction Step:

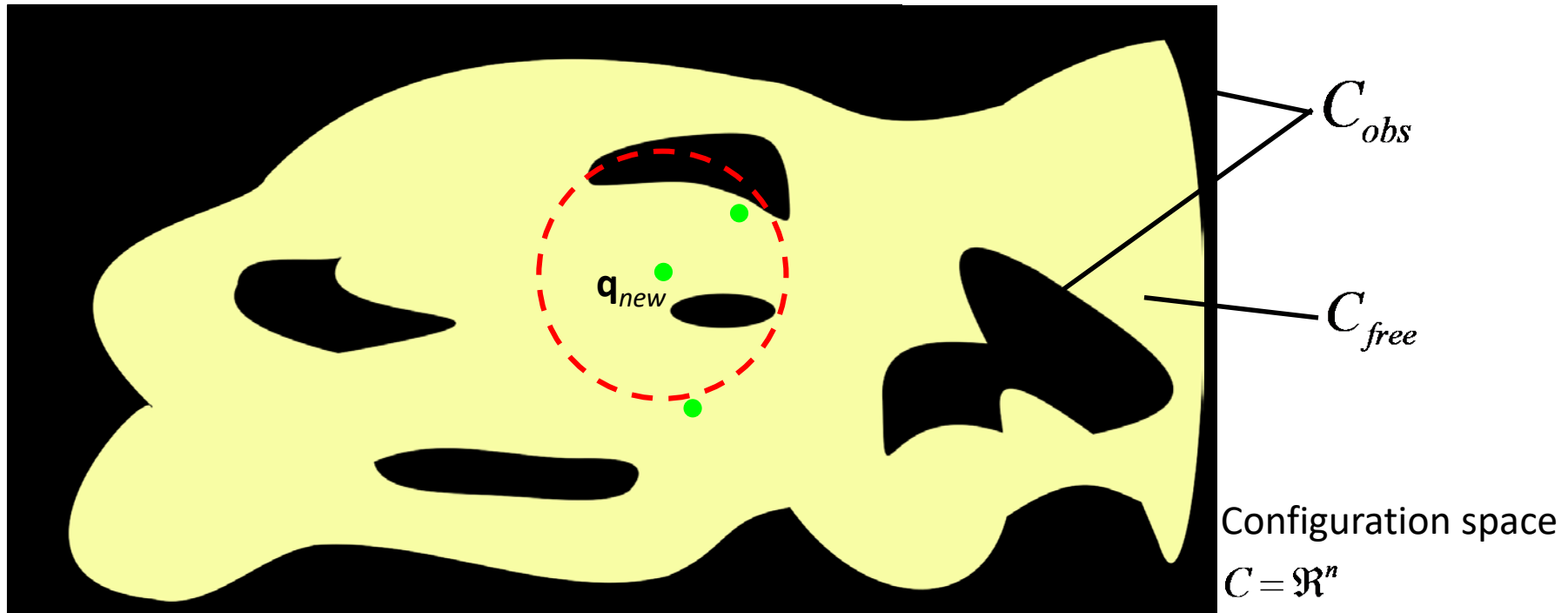
Find the set of all neighbors of $\mathbf{q}_{new}$

$\mathbf{q}_{new} \leftarrow$ random sample in C

if $\mathbf{q}_{new} \in C_{free}$

add $\mathbf{q}_{new}$ to V

for each $\mathbf{q} \in \text{Neighbor}(G, \mathbf{q}_{new})$



Probabilistic Road Maps

$$G=(V,E) \quad V \leftarrow \emptyset, E \leftarrow \emptyset$$

Construction Step:

Find the set of all neighbors of $\mathbf{q}_{new}$
and connect each neighbor for which a
path in C_{free} exist

$\mathbf{q}_{new} \leftarrow$ random sample in C

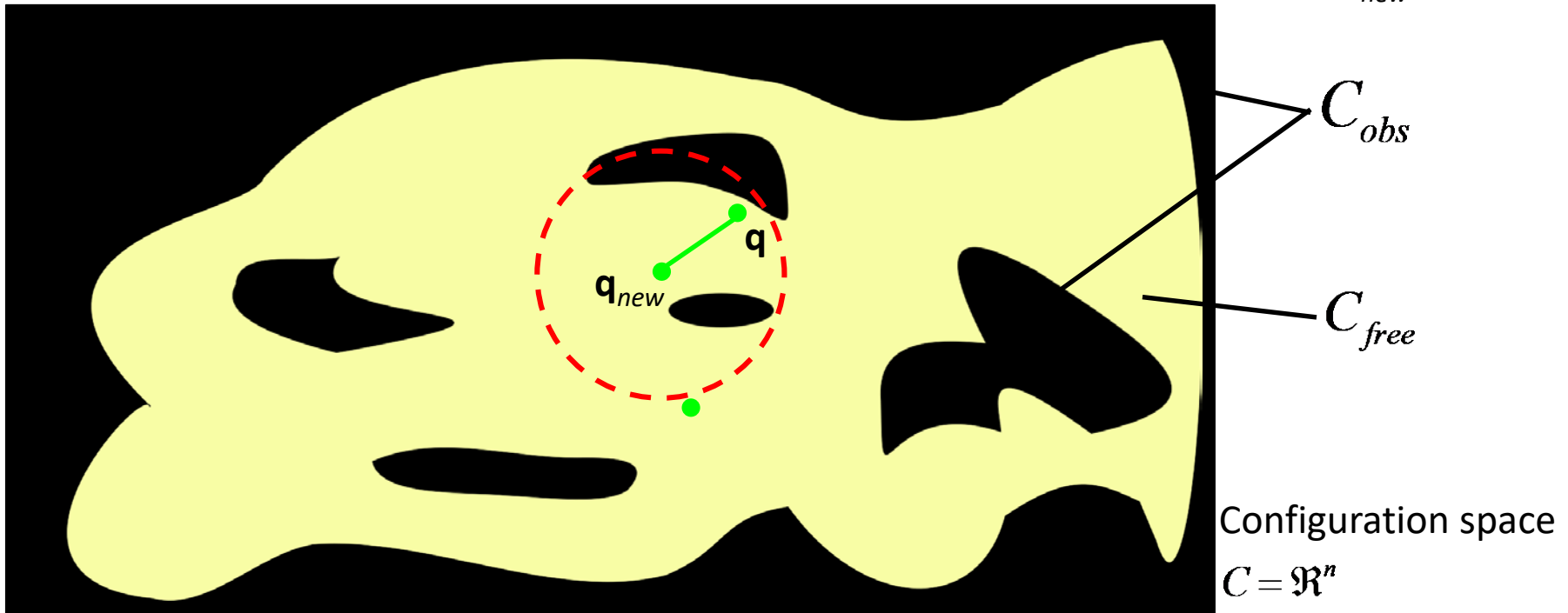
if $\mathbf{q}_{new} \in C_{free}$

add $\mathbf{q}_{new}$ to V

for each $\mathbf{q} \in \text{Neighbor}(G, \mathbf{q}_{new})$

If $\text{link}(\mathbf{q}, \mathbf{q}_{new})$

add edge $(\mathbf{q}_{new}, \mathbf{q})$ to E

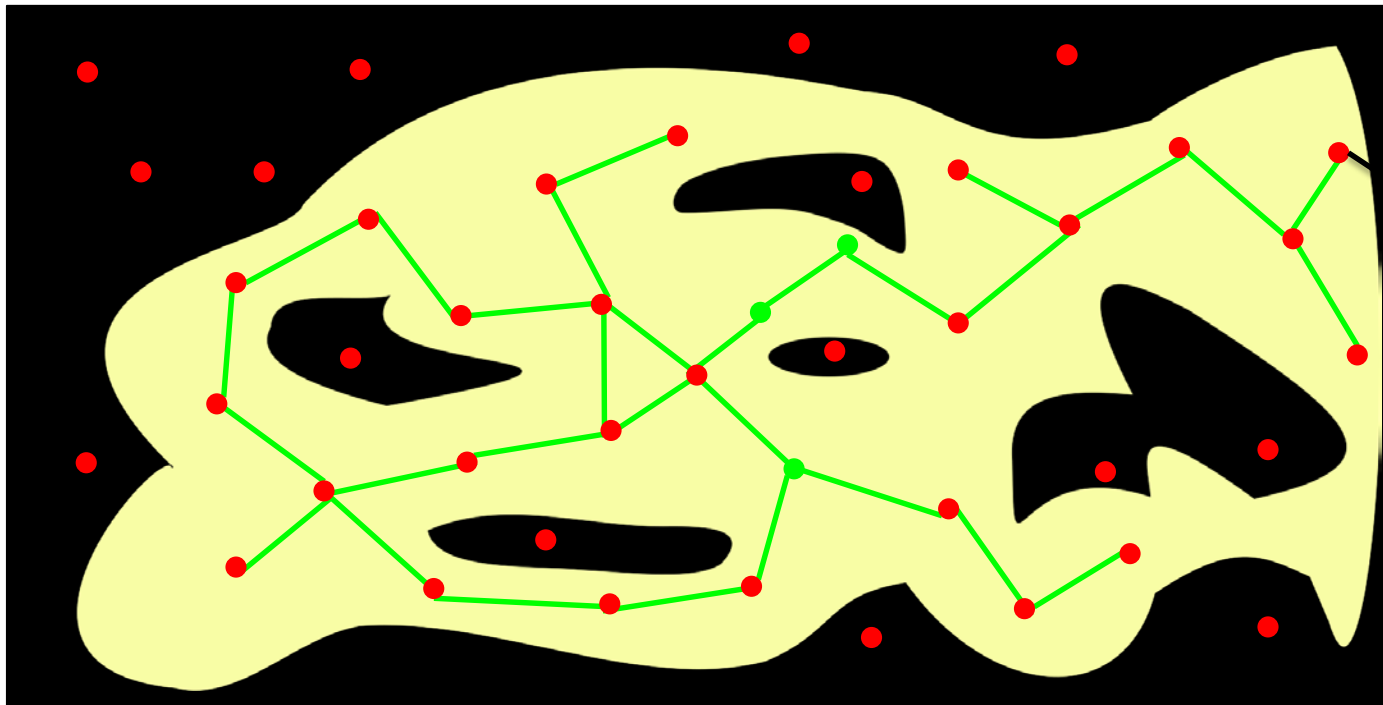


Probabilistic Road Maps

Construction Step:

```
G=(V,E)  V ← ∅, E ← ∅  
while |E| < N  
    qnew ← random sample in C  
    if qnew ∈ Cfree  
        add qnew to V  
        for each q ∈ Neighbor(G, qnew)  
            If link(q, qnew)  
                add edge (qnew, q) to E
```

Repeat



Probabilistic Road Maps

Construction Step:

Initialization:

$G=(V,E)$, $V=\{\}$, $E=\{\}$

What are the most difficult and time consuming steps?

Algorithm:

while $|E| < N$

$\mathbf{q}_{new} \leftarrow$ random sample in C

if $\mathbf{q}_{new} \in C_{free}$

$G.addVertex(\mathbf{q}_{new})$

for each $\mathbf{q} \in \text{Neighbor}(G, \mathbf{q}_{new})$

if $\text{link}(\mathbf{q}, \mathbf{q}_{new})$

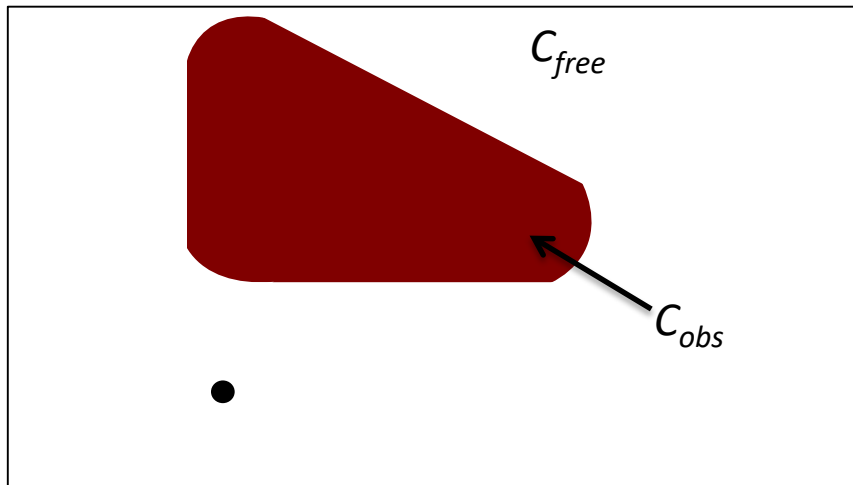
$G.addEdge(\mathbf{q}_{new}, \mathbf{q})$

Collision Checking

How do we check if a given configuration $\mathbf{q}$ is in collision with the objects in the environment?

In the past

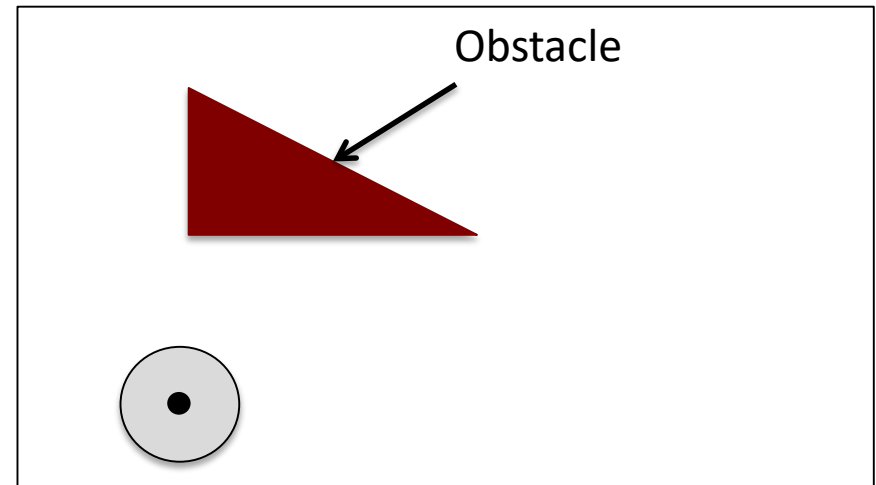
Computed C_{obs} a priori



Combinatorial motion planning in C_{free}

Now

Use collision checker



Collision Detection

True,
false ↓ ↑ $\mathbf{q}$

Sampling based motion planning in C

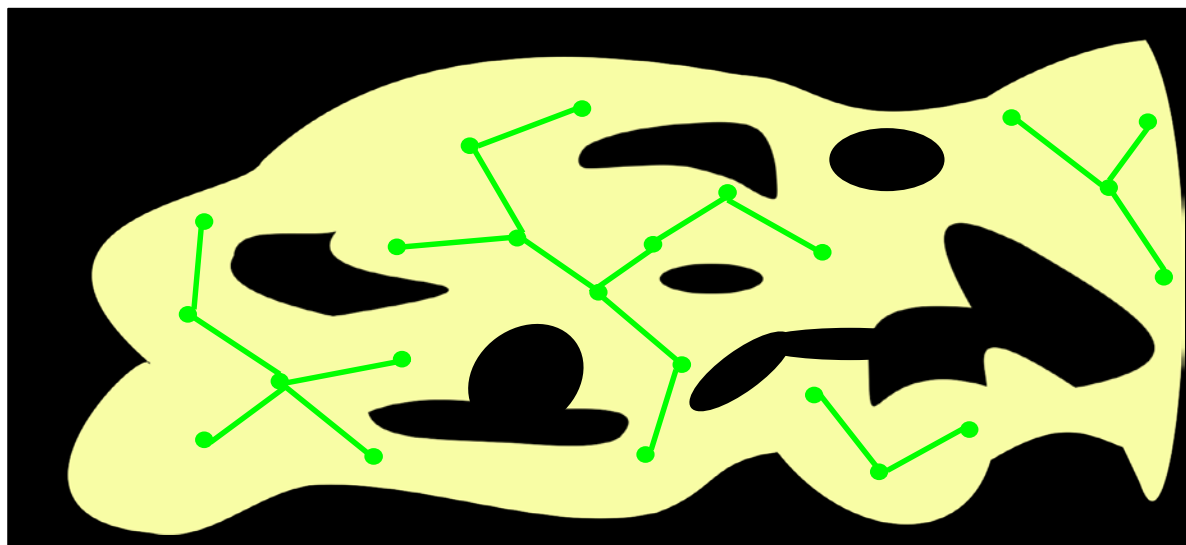
Probabilistic Road Maps

See Incremental Variant
of PRM described in
Hauser's Robotic
Systems Book (Chapter
10)

Expansion Step: Improve connectivity

In easy scenes with high number of sampled points: G is most likely well connected

But: in more complex scenes, the graph might be disconnected and not capture the connectivity of C_{free}



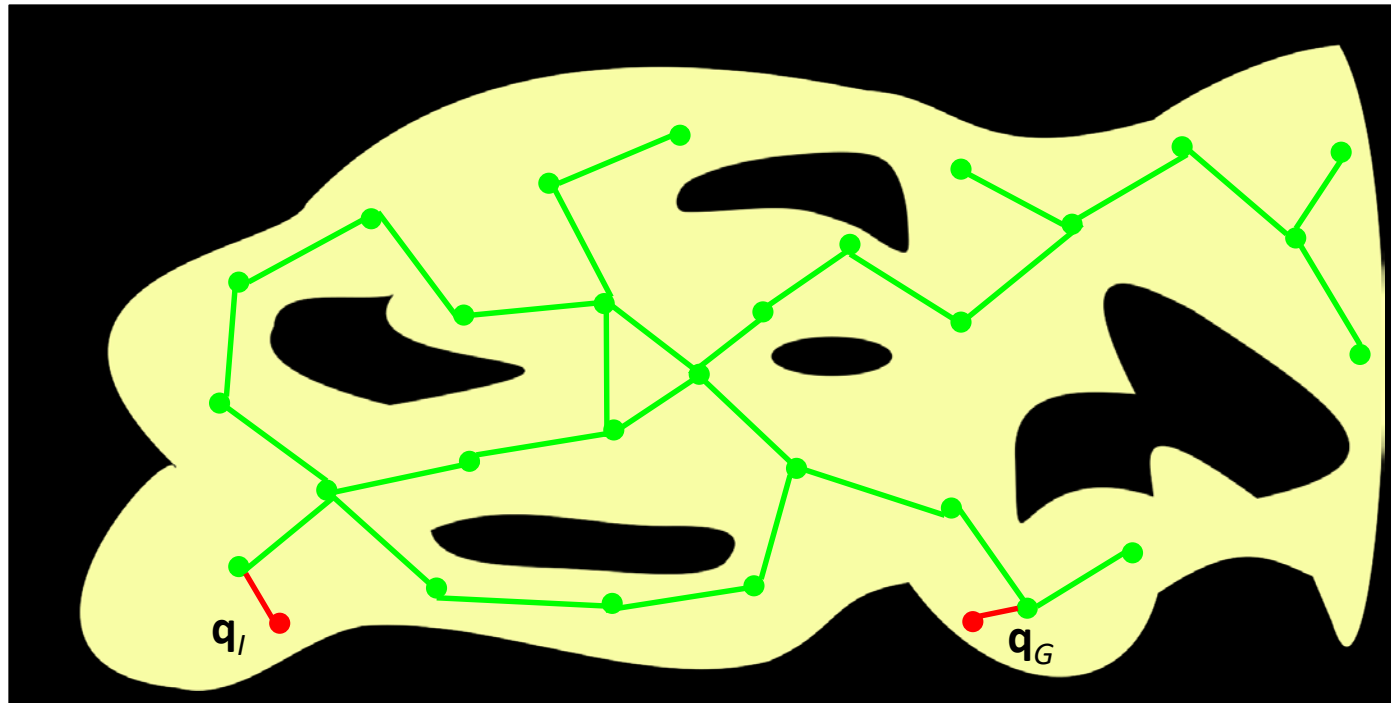
Use heuristics to improve connectivity!

Probabilistic Road Maps

Query Phase:

Path Planning: For a new pair of start ($\mathbf{q}_I$) and goal ($\mathbf{q}_G$) configuration

- 1) Connect $\mathbf{q}_I$ and $\mathbf{q}_G$ to road map to $\mathbf{q}'$ and $\mathbf{q}''$
- 2) Find path from $\mathbf{q}'$ and $\mathbf{q}''$ in G

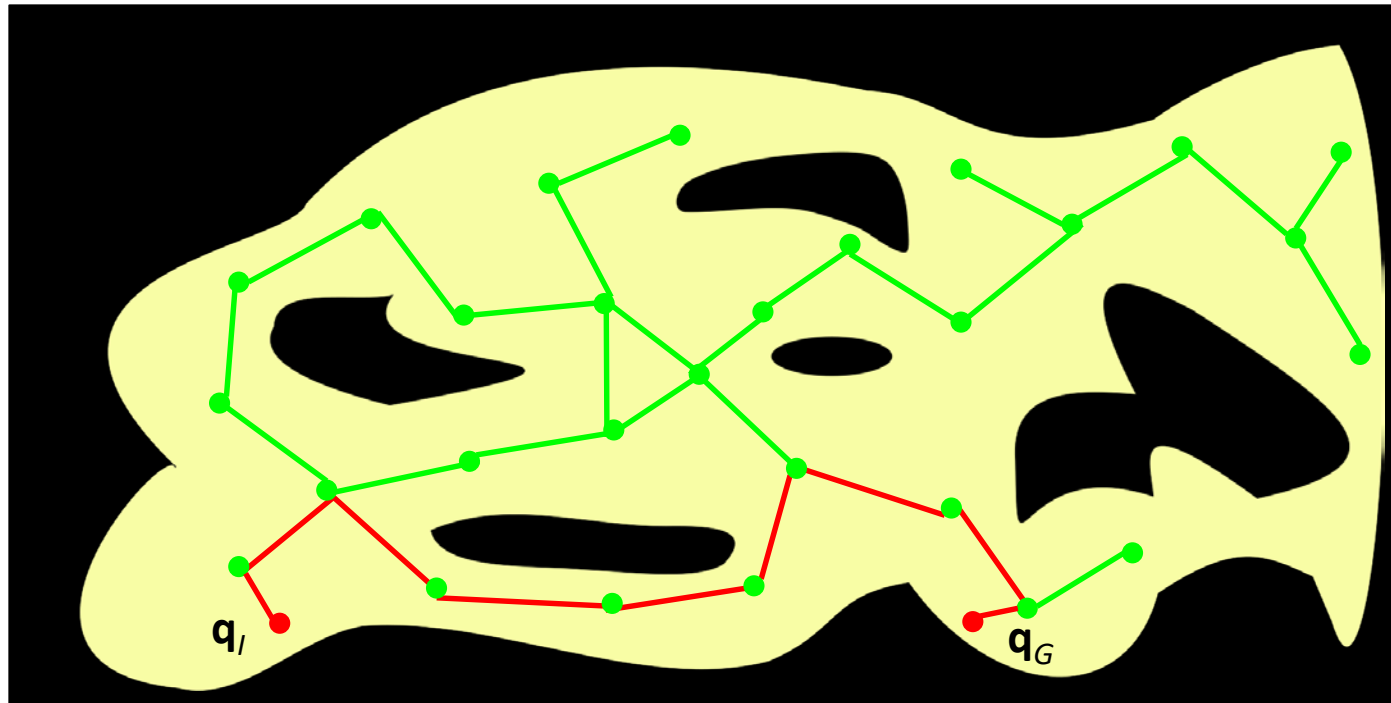


Probabilistic Road Maps

Query Phase:

Path Planning: For a new pair of start ($\mathbf{q}_I$) and goal ($\mathbf{q}_G$) configuration

- 1) Connect $\mathbf{q}_I$ and $\mathbf{q}_G$ to road map to $\mathbf{q}'$ and $\mathbf{q}''$
- 2) Find path from $\mathbf{q}'$ and $\mathbf{q}''$ in G

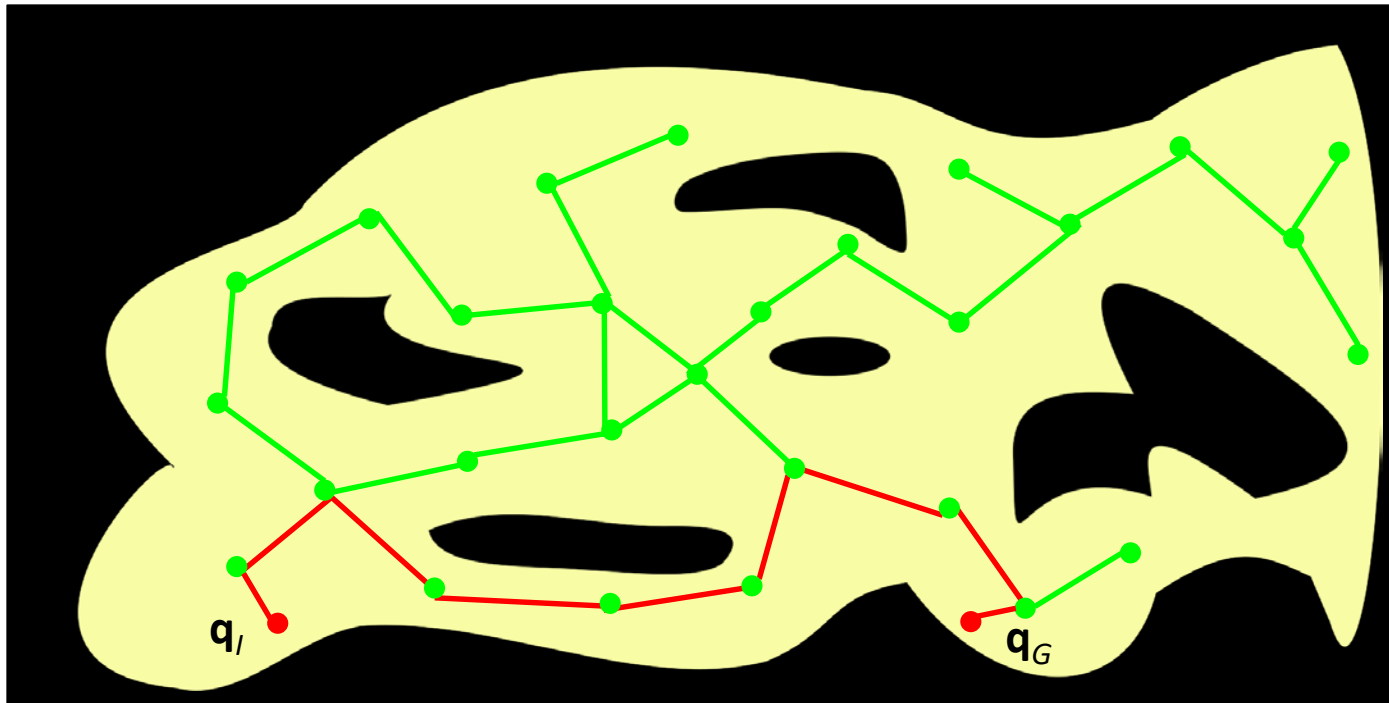


Probabilistic Road Maps

Query Phase:

Is this a good path to follow?

Randomized motion planners usually find jagged and longer path than necessary that are not suited for execution: -> need to smoothen

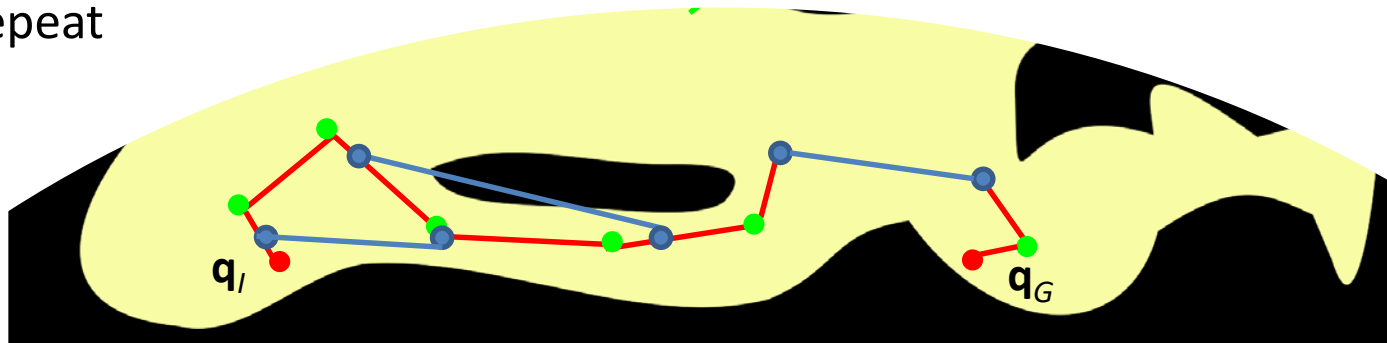


Probabilistic Road Maps

Path Smoothing:

Shortcutting:

- Along the found path: select randomly to points q_1 and q_2 and try to connect them
- If successful replace new path with old one.
- Repeat

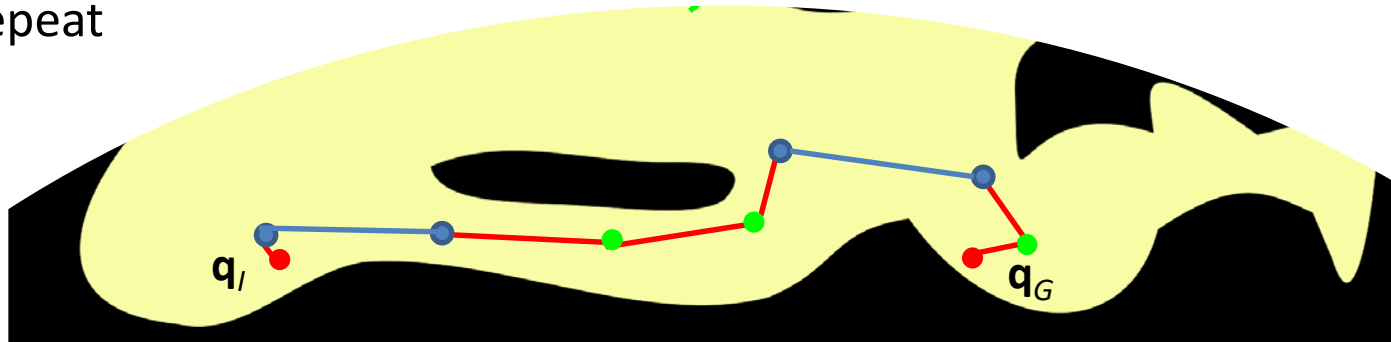


Probabilistic Road Maps

Path Smoothing:

Shortcutting:

- Along the found path: select randomly to points q_1 and q_2 and try to connect them
- If successful replace new path with old one.
- Repeat

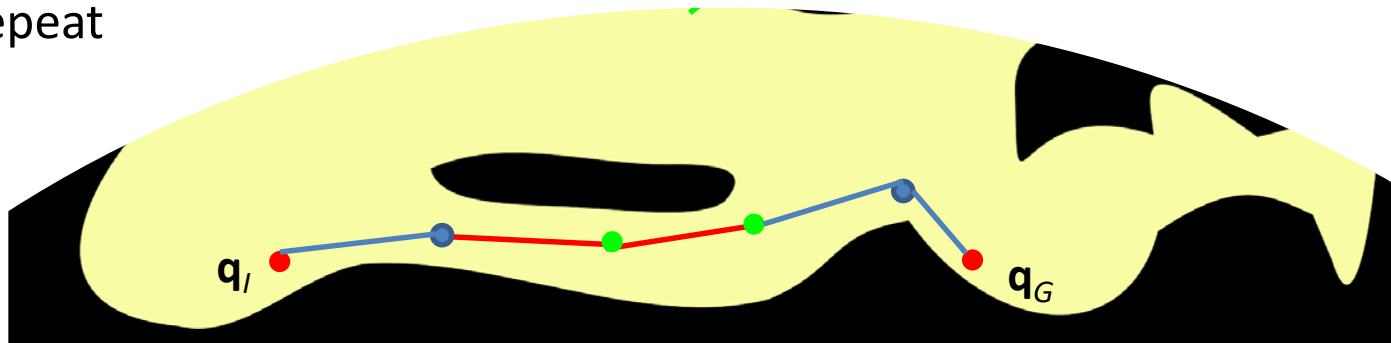


Probabilistic Road Maps

Path Smoothing:

Shortcutting:

- Along the found path: select randomly to points q_1 and q_2 and try to connect them
- If successful replace new path with old one.
- Repeat



Nonlinear optimization :

- Specify object function to optimize path

Probabilistic Road Maps

What are scenarios in which the PRM might fail?

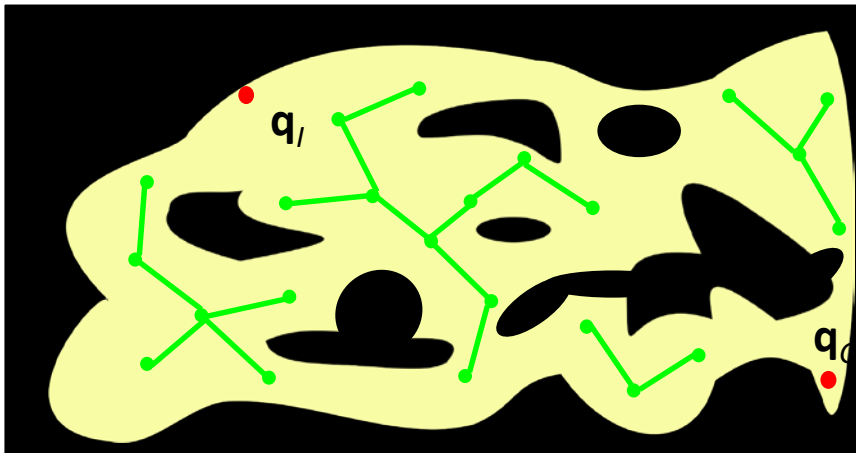
1. Accessibility or Departability constraint not fulfilled

2. Connectivity of C_{free} not correctly represented in G

Why?

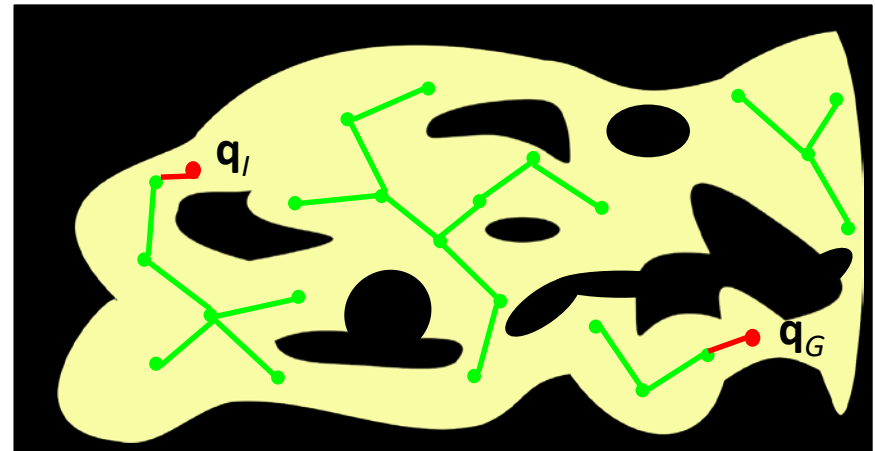
Coverage

Milestones should be distributed such that any point in C_{free} can connect to G



Connectivity

Connectivity of G must represent connectivity of C_{free}



Probabilistic Road Maps

Summary

Idea:

- Build a roadmap from sampled configuration space points
- Check if in C_{free} by using a collision checker, and
- search the roadmap to find a path.

Algorithm

```
G=(V,E)  V ← ∅, E ← ∅
while |E| < N
    qnew ← random sample in C
    if qnew ∈ Cfree
        add qnew to V
        for each q ∈ Neighbor(G, qnew)
            If link(q, qnew)
                add edge (qnew, q) to E
```

Properties:

- Provides feasible, but not optimal solution
- Works well on practical problems with fixed obstacle sets
- Probabilistic complete
- Problems with narrow passages
- Spending a lot of time of pre-processing to explore whole space to create road map

Single Query Planner

Sampling Based Planning

Multi-Query

Two phases:

- 1) Build Roadmap **long**
- 2) Search **short**

Assumes fixed obstacle setting over multiple queries.

Probabilistic Road Maps

[Kavaki et al.: Probabilistic roadmaps for path planning in high dimensional configuration spaces; 1996]

Single-Query

Iteratively build tree with search.

Rapidly exploring Random Trees

Kuffner and LaValle: Rapidly Exploring Random Trees; 2000



Interleave 1) and two 2) until path found

In a changing environment: Just interested once in the path and do not want to explore whole configuration space

Single Query Planner

Key Ideas

Idea 1:

- Add q_I and q_G to the milestones, connect them to G and stop sampling if they are in the same connected component

Idea 2:

- Keep track of q_I and q_G connected components
- Sample with bias to q_I and q_G

Idea 3:

- Make the graph a tree

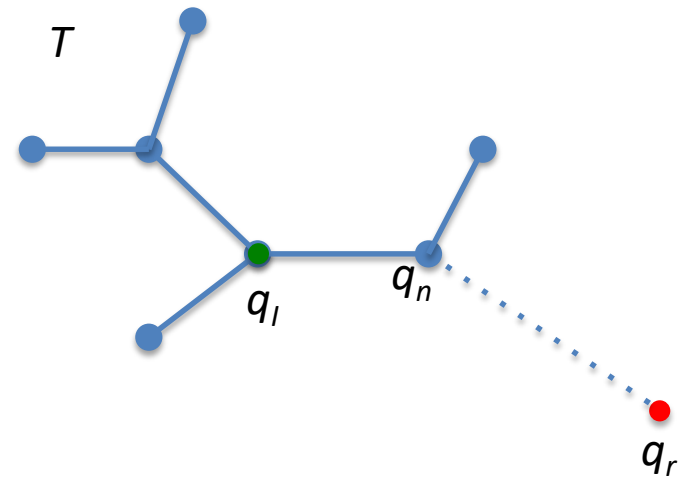
Rapidly-exploring Random Trees (RRT)

Algorithm

RRT($N, \Delta t, q_l$):

$T.\text{init}(q_l)$

For $k=1$ to N :



Rapidly-exploring Random Trees (RRT)

Algorithm

RRT($N, \Delta d, q_l$):

$T.\text{init}(q_l)$

For $k=1$ to N :

$q_r \leftarrow \text{random_sample_C}_{\text{free}}()$

$q_n \leftarrow \text{nearest_neighbor}(q_r, T)$

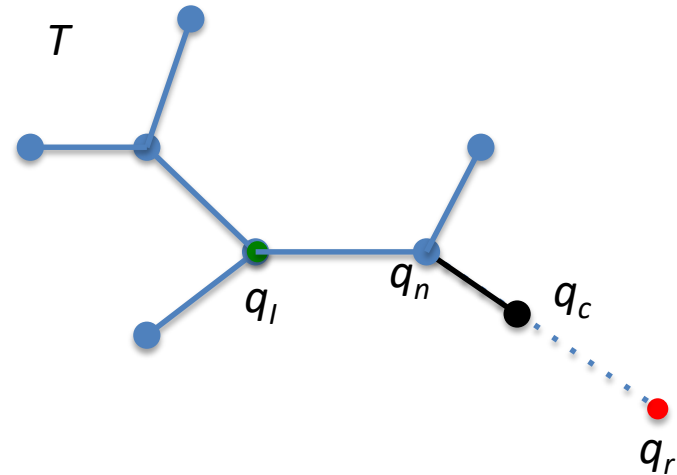
if $\text{new_state}(q_r, q_n, q_c, u, \Delta t)$

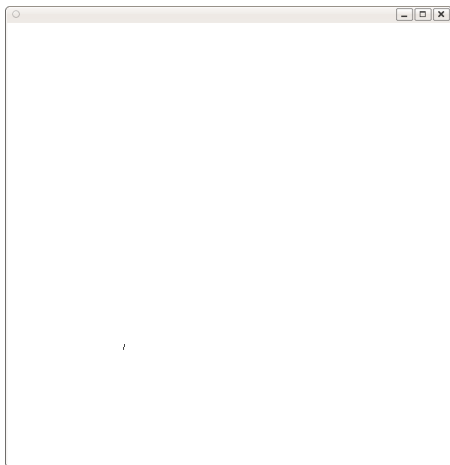
$T.\text{add_vertex}(q_c)$

$T.\text{add_edge}(q_n, q_c)$

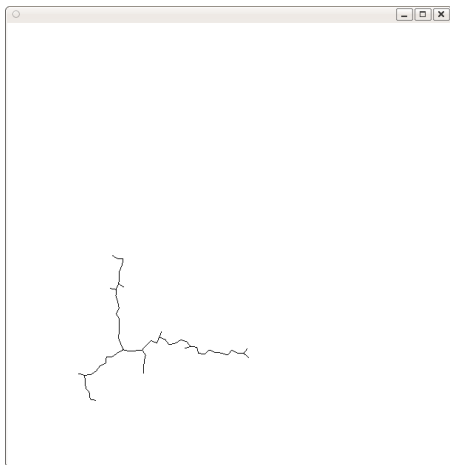
Extend(T, q_r)

Return T

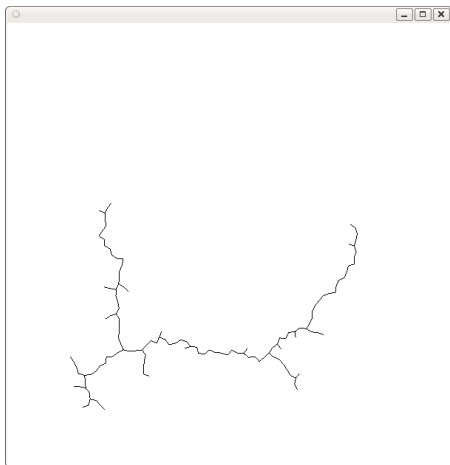




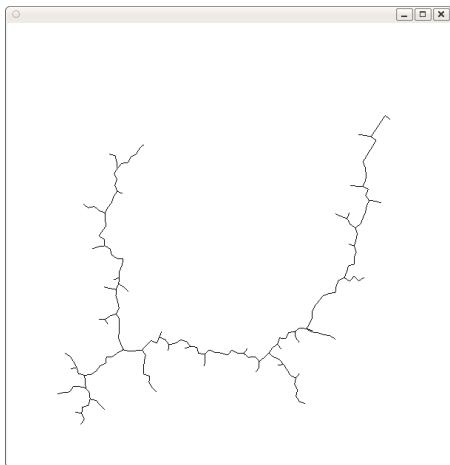
$$n = 1$$



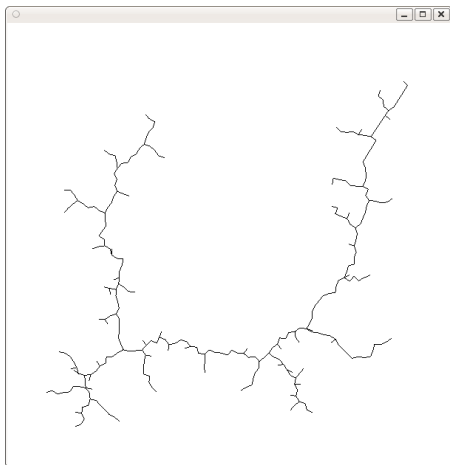
$$n = 100$$



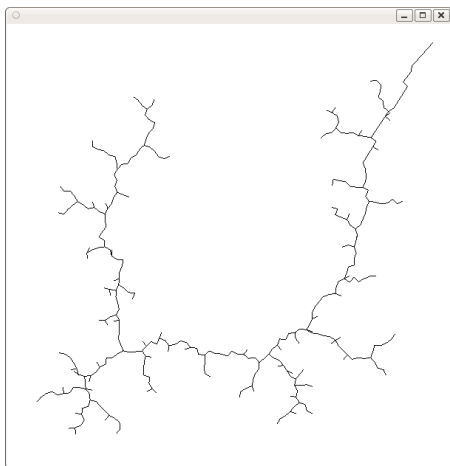
$n = 200$



$n = 300$



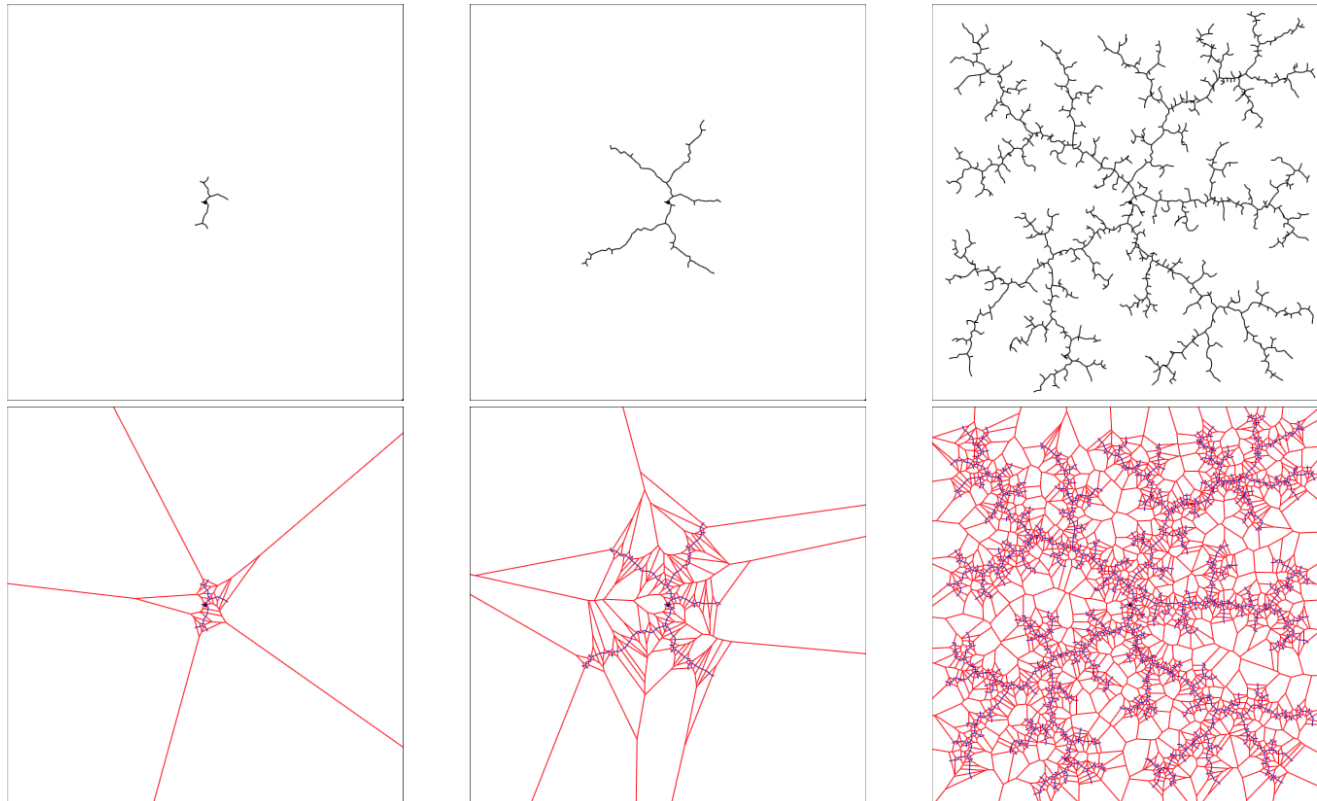
$n = 400$



$n = 500$

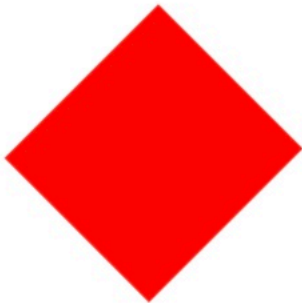
Rapidly-exploring Random Trees (RRT)

RRT is biased by large Voronoi regions



<http://msl.cs.uiuc.edu/~lavalley/papers/LavKuf01.pdf>

★ goal



start



Naive Sampling

Rapidly-exploring Random Trees (RRT)

Goal Bias

Idea: bias sampling towards the goal configuration to create quickly a path to q_G

How:

With probability p sample q_G instead of uniform sampled configuration

Choose small p (1 to 10 %)

If bias too strong: RRT becomes too greedy and gets stuck in local minima

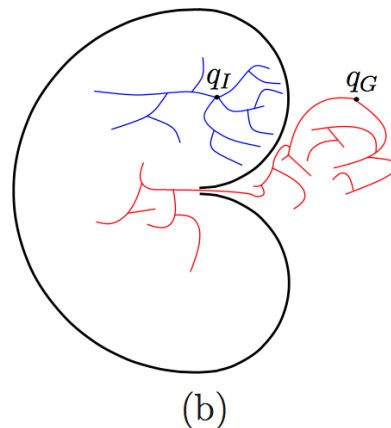
Rapidly-exploring Random Trees (RRT)

Balanced Bidirectional Search

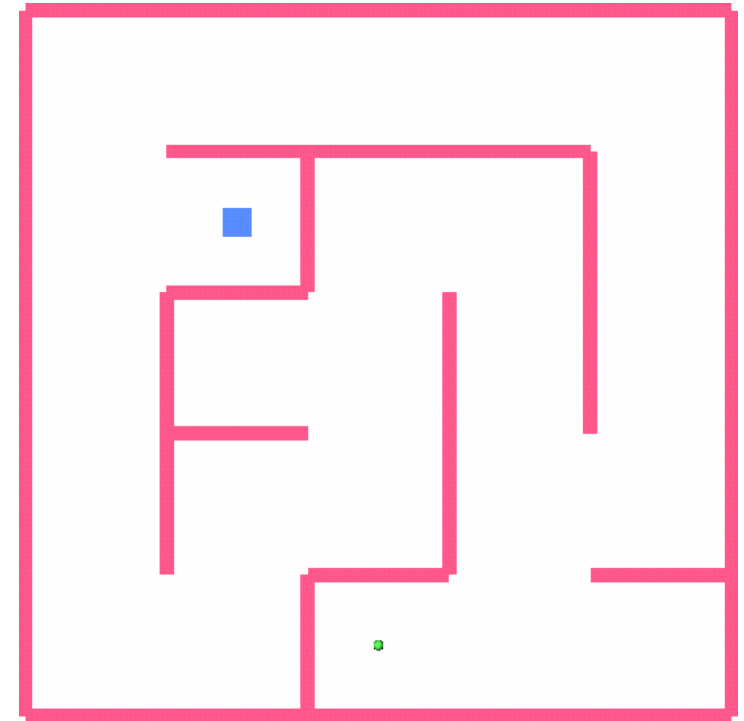
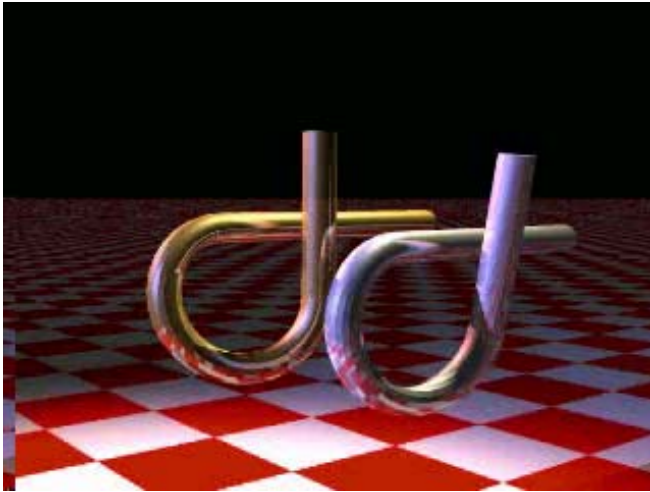
Sometimes it is easier to search from both directions

Idea:

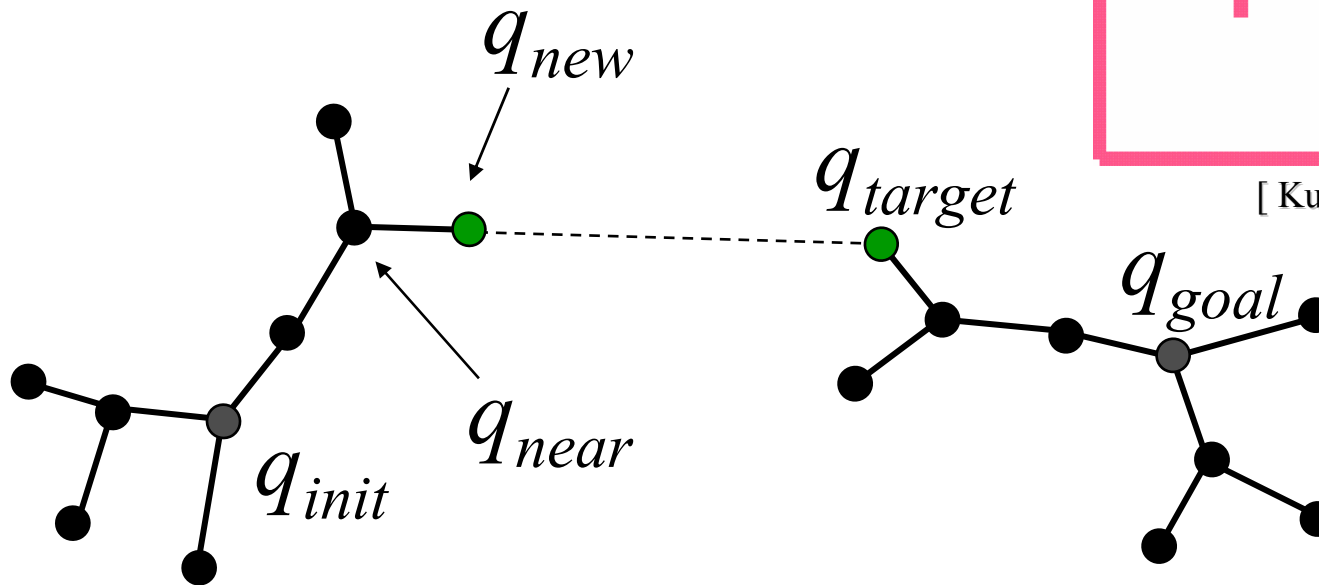
- Grow trees from both q_I and q_G
- Make sure both trees are balanced: switch between extending and connecting after a few iterations



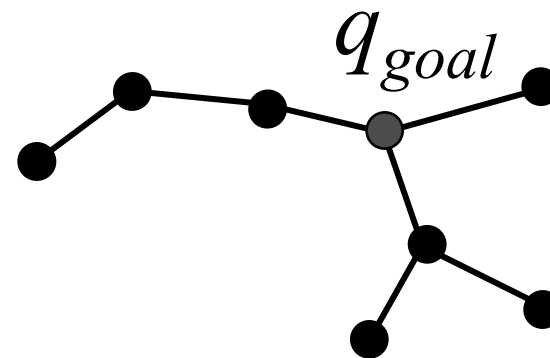
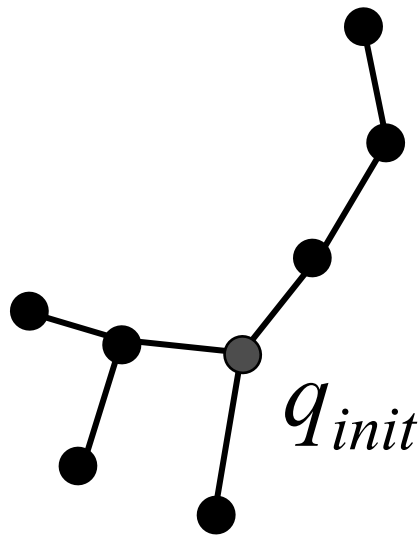
Grow two RRTs towards each other



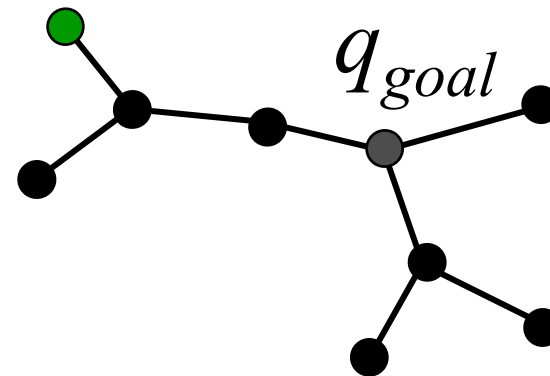
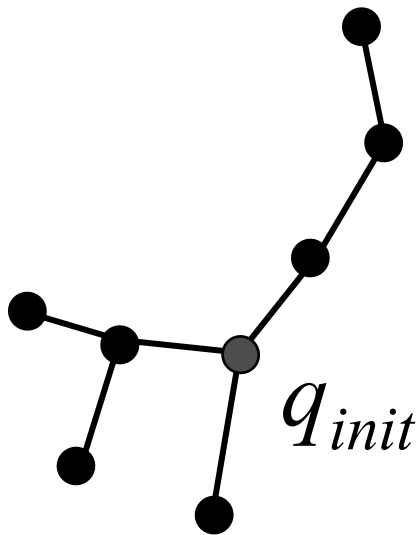
[Kuffner, LaValle ICRA '00]



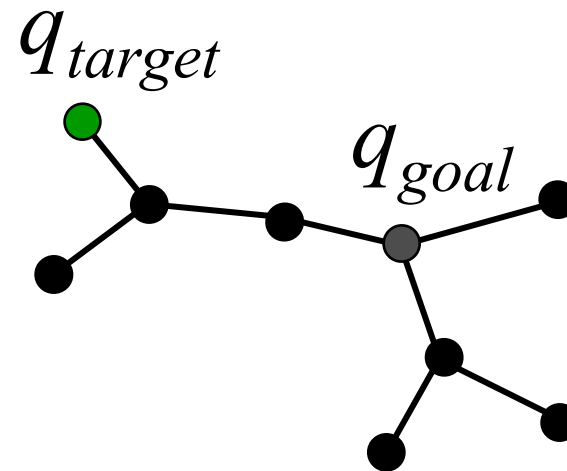
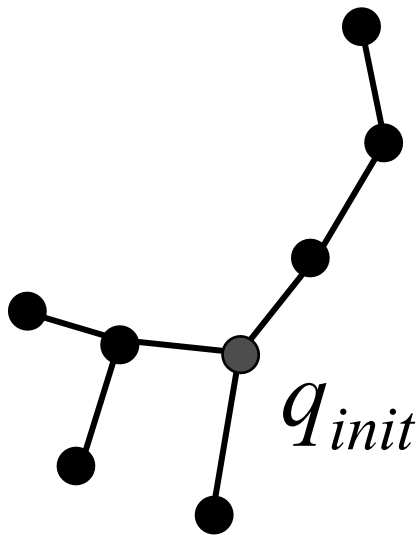
A single RRT-Connect iteration...



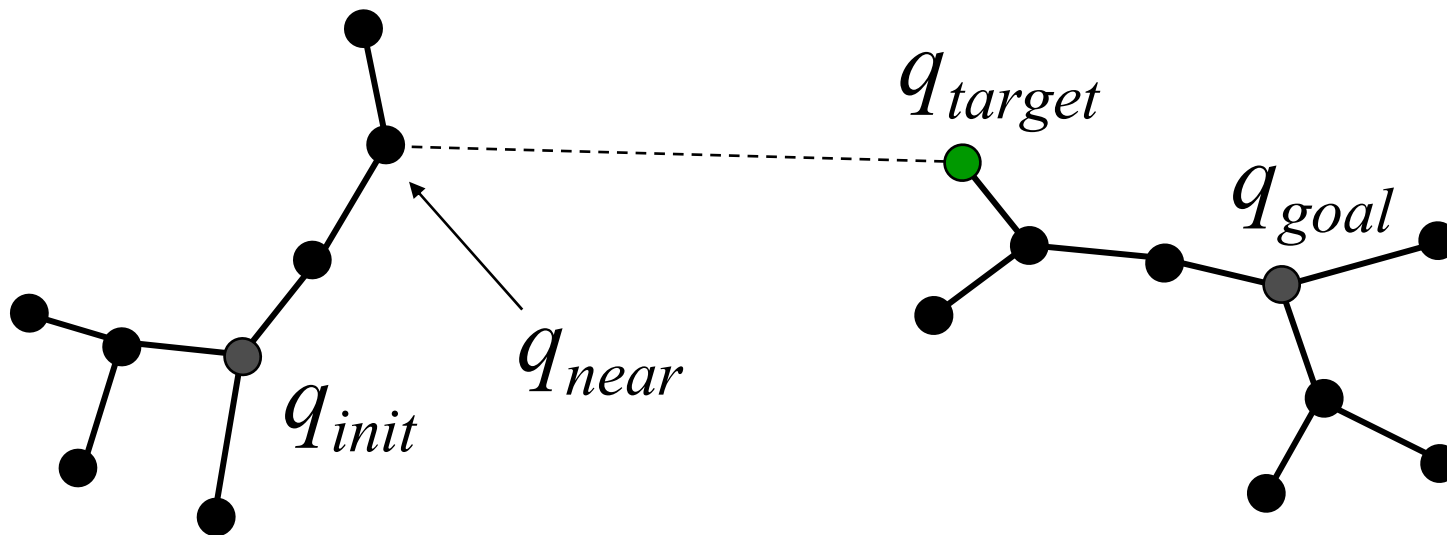
1) One tree grown using random target



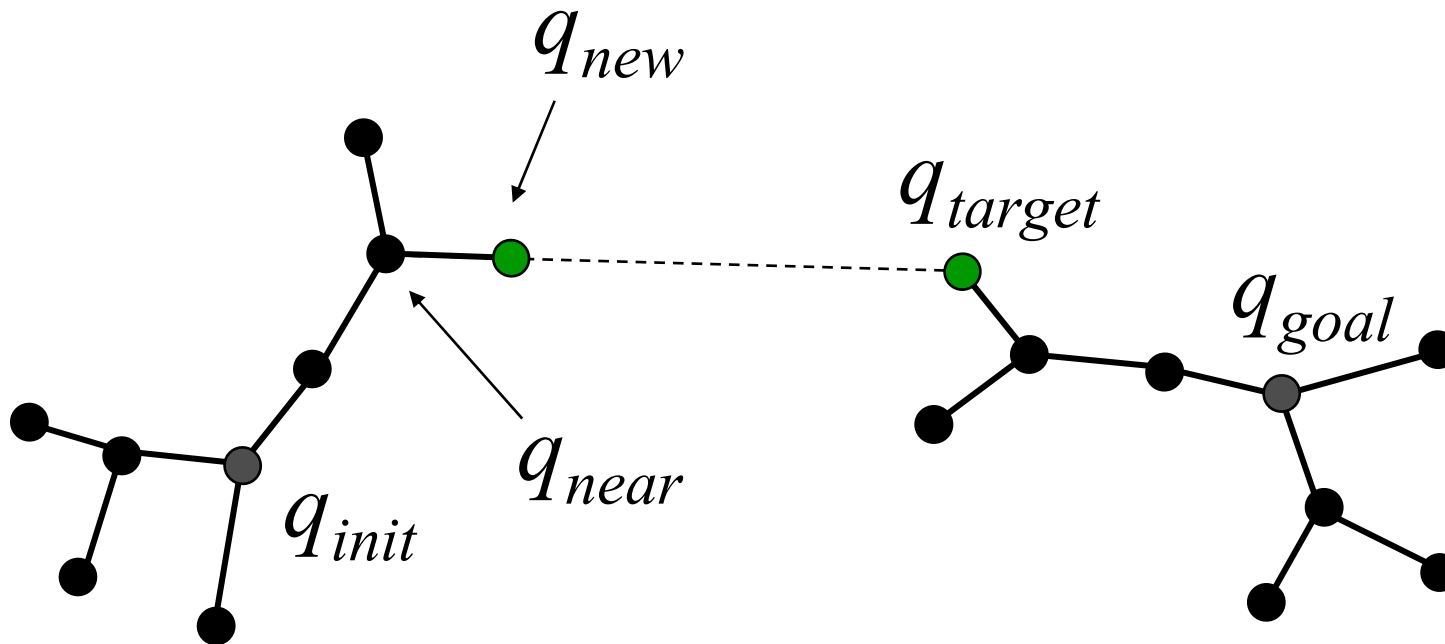
2) New node becomes target for other tree



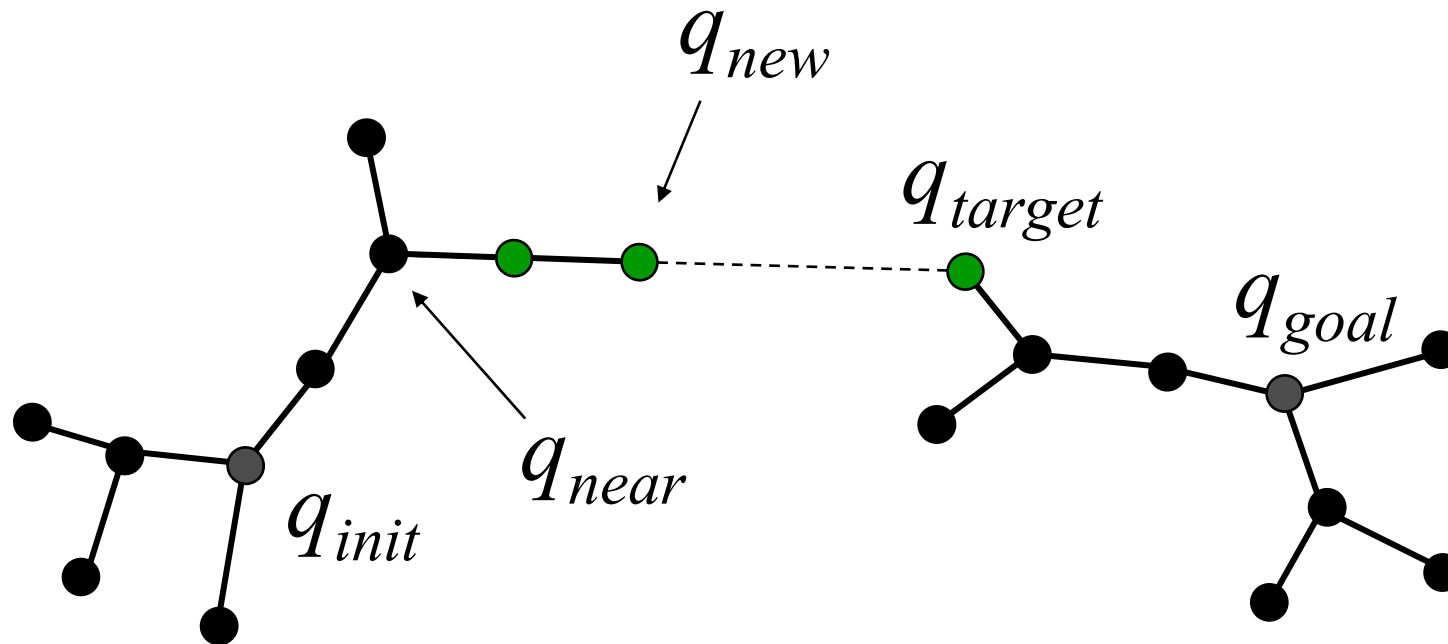
3) Calculate node “nearest” to target



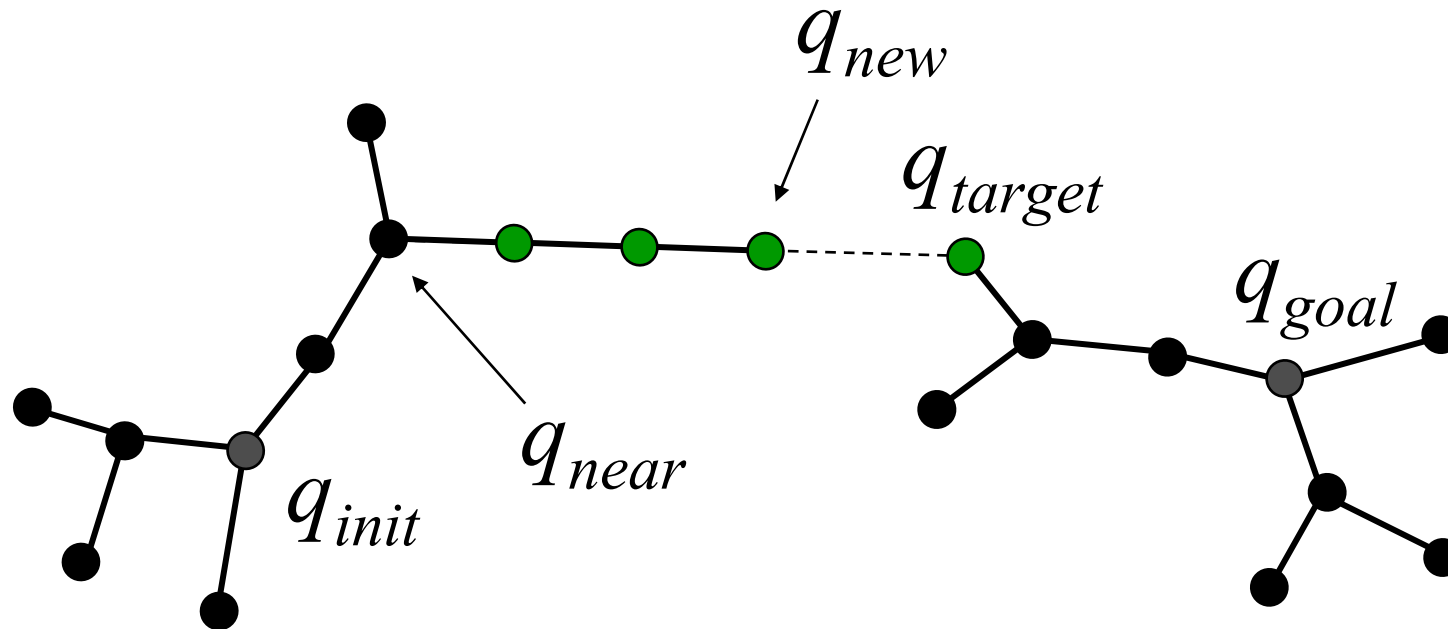
4) Try to add new collision-free branch



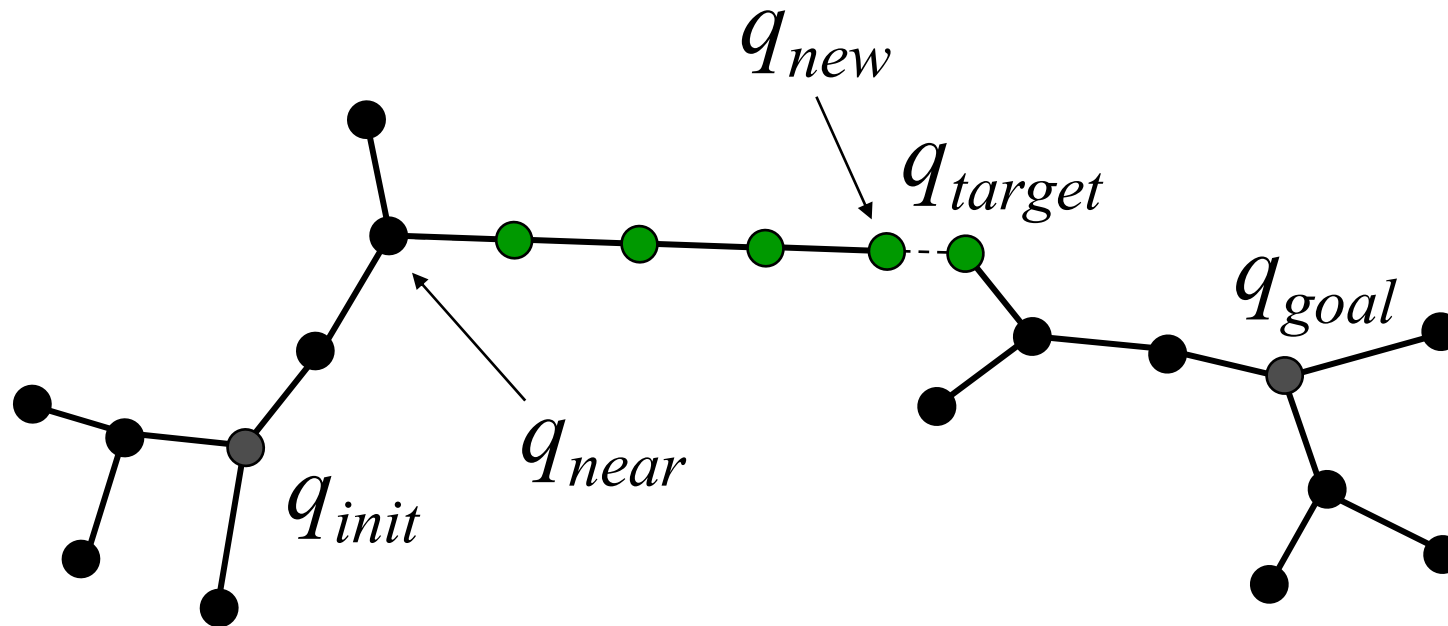
5) If successful, keep extending branch



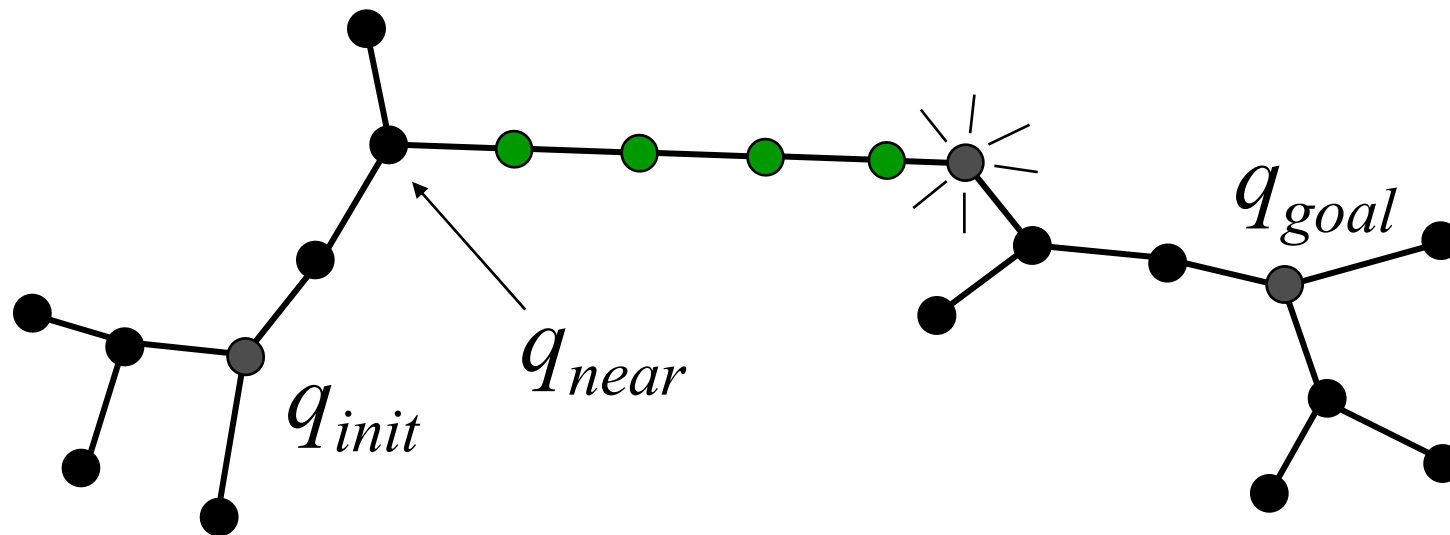
5) If successful, keep extending branch



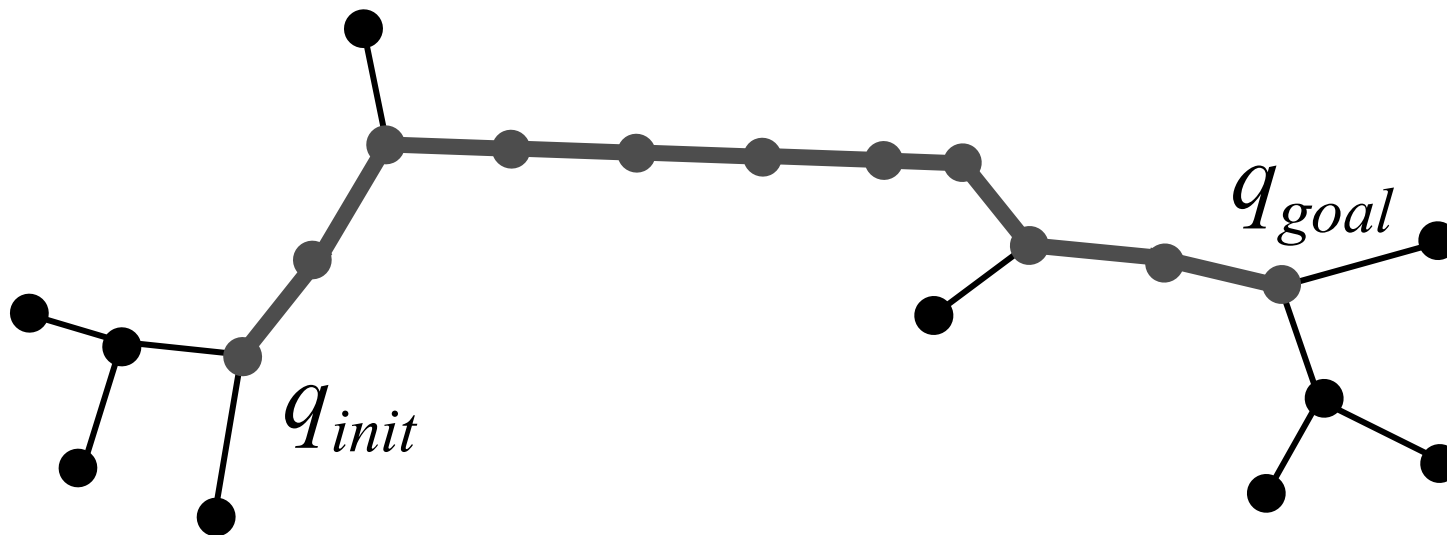
5) If successful, keep extending branch



6) Path found if branch reaches target

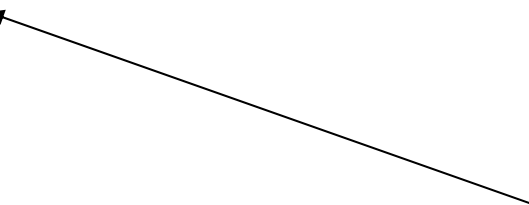


7) Return path connecting start and goal



Basic RRT-Connect

```
RRT_CONNECT ( $q_{init}, q_{goal}$ ) {  
     $T_a.init(q_{init}); T_b.init(q_{goal});$   
    for  $k = 1$  to  $K$  do  
         $q_{rand} = \text{RANDOM\_CONFIG}();$   
        if not ( $\text{EXTEND}(T_a, q_{rand}) = \text{Trapped}$ ) then  
            if ( $\text{EXTEND}(T_b, q_{new}) = \text{Reached}$ ) then  
                Return  $\text{PATH}(T_a, T_b);$   
             $\text{SWAP}(T_a, T_b);$   
    Return Failure;  
}
```



Instead of switching, use T_a as smaller tree. This helped James a lot

```

EXTEND( $\mathcal{T}, q$ )
1   $q_{near} \leftarrow \text{NEAREST\_NEIGHBOR}(q, \mathcal{T});$ 
2  if NEW_CONFIG( $q, q_{near}, q_{new}$ ) then
3       $\mathcal{T}.\text{add\_vertex}(q_{new});$ 
4       $\mathcal{T}.\text{add\_edge}(q_{near}, q_{new});$ 
5      if  $q_{new} = q$  then
6          Return Reached;
7      else
8          Return Advanced;
9  Return Trapped;

```

RRT – Non-holonomic robots

Algorithm

RRT($N, \Delta d, q_l$):

$T.\text{init}(q_l)$

For $k=1$ to N :

$q_r \leftarrow \text{random_sample_C}_{\text{free}}()$

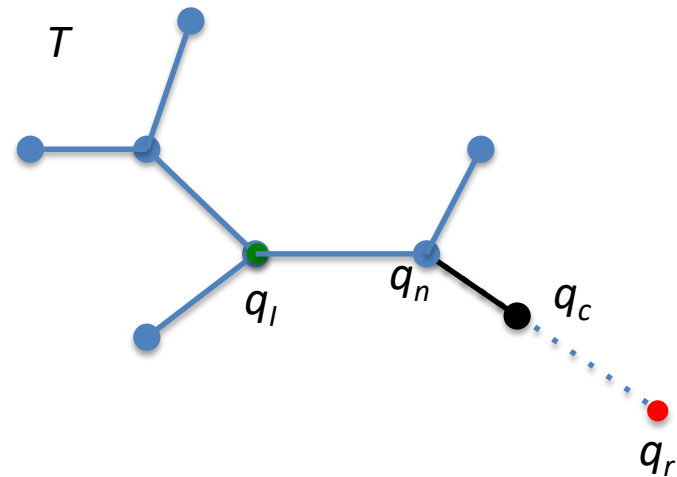
$q_n \leftarrow \text{nearest_neighbor}(q_r, T)$

→ if $\text{new_state}(q_r, q_n, q_c, u, \Delta t)$

$T.\text{add_vertex}(q_c)$

$T.\text{add_edge}(q_n, q_c, u)$

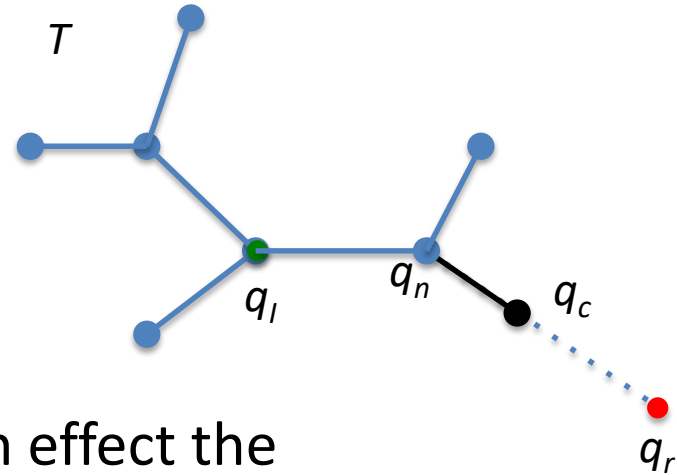
Return T



RRT – Non-holonomic robots

$\text{new_state}(q_r, q_n, q_c, u, \Delta t)$

Input/actions to reach
 q_c from q_n



U : set of controls or actions that can effect the state (configuration) of the robot

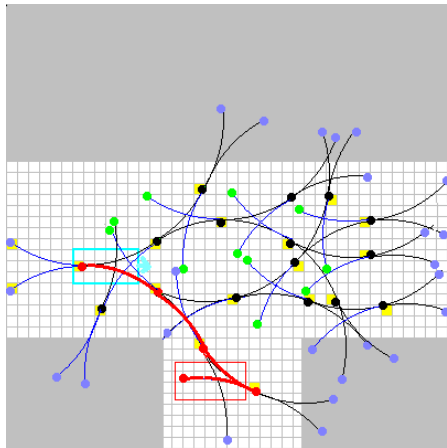
The function new_state makes a motion from q_n towards q_r by applying input $u \in U$ for some time Δt or until it reaches an obstacle



Incremental Simulator:

Given q_t , U , and $\Delta t \rightarrow \text{Compute } q(t+\Delta t)$

Control-based sampling for the car



- 1) Select a $q \in T$
- 2) Pick v , ϕ , and τ
- 3) Integrate motion from q
- 4) Add result if collision-free

Rapidly-exploring Random Trees (RRT)

Summary

Idea:

- Probe and explore the configuration space by incrementally expanding from initial configuration into open space
- Use tree instead of graph that is rooted at q_i

Pros:

- Easy to implement
- Balance between greedy search and exploration with goal bias
- Well suited for high-dimensional practical problems

Algorithm:

$T.\text{init}(q_i)$

For $k=1$ to N :

$q_r \leftarrow \text{random_sample_C}_{\text{free}}()$

$q_n \leftarrow \text{nearest_neighbor}(q_r, T)$

if $\text{new_state}(q_r, q_n, q_c, u, \Delta t)$

$T.\text{add_vertex}(q_c)$

$T.\text{add_edge}(q_n, q_c, q_i)$

Return T

} Extend(T, q_r)

Cons:

- Much more sensitive to metric
- Only feasible path, not optimal
- Path needs to be smoothed

A bit of history

Visibility graph in Cspace (Lozano-Perez and Wesley) 1977-99

Fig. 1.

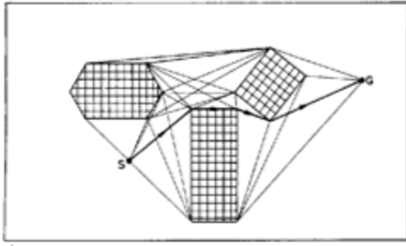


Fig. 3(a).

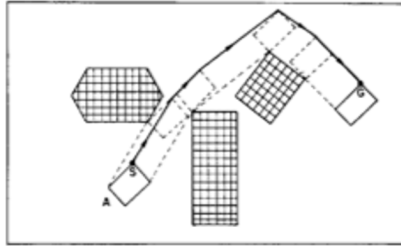


Fig. 2.

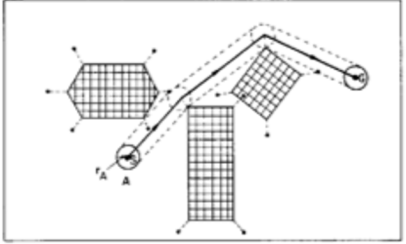
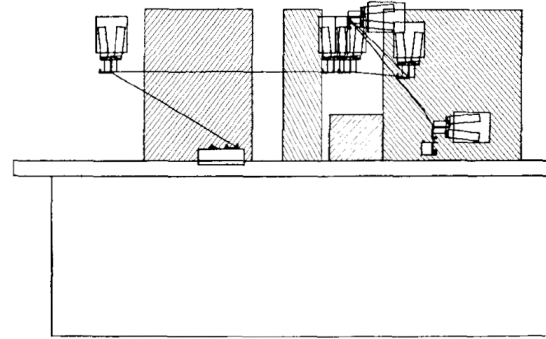
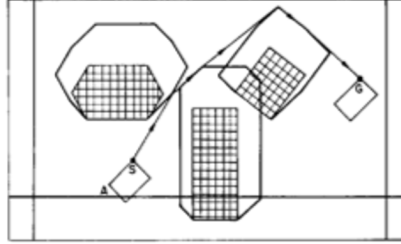
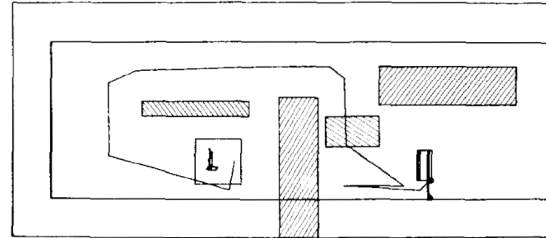


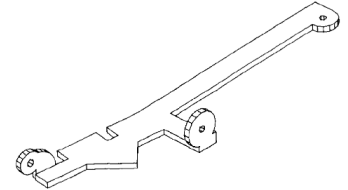
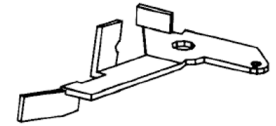
Fig. 3(b).



(a)



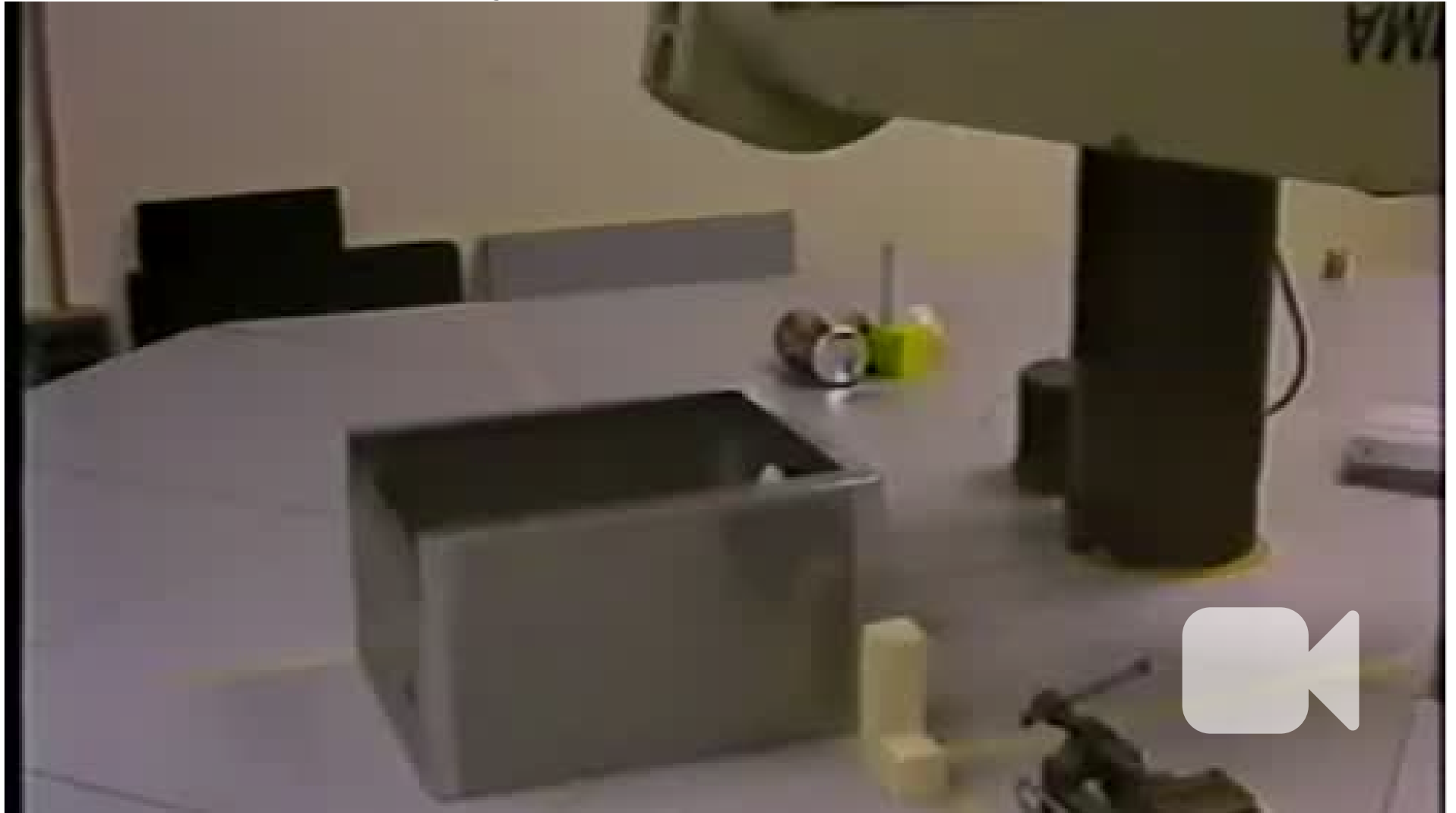
(b)



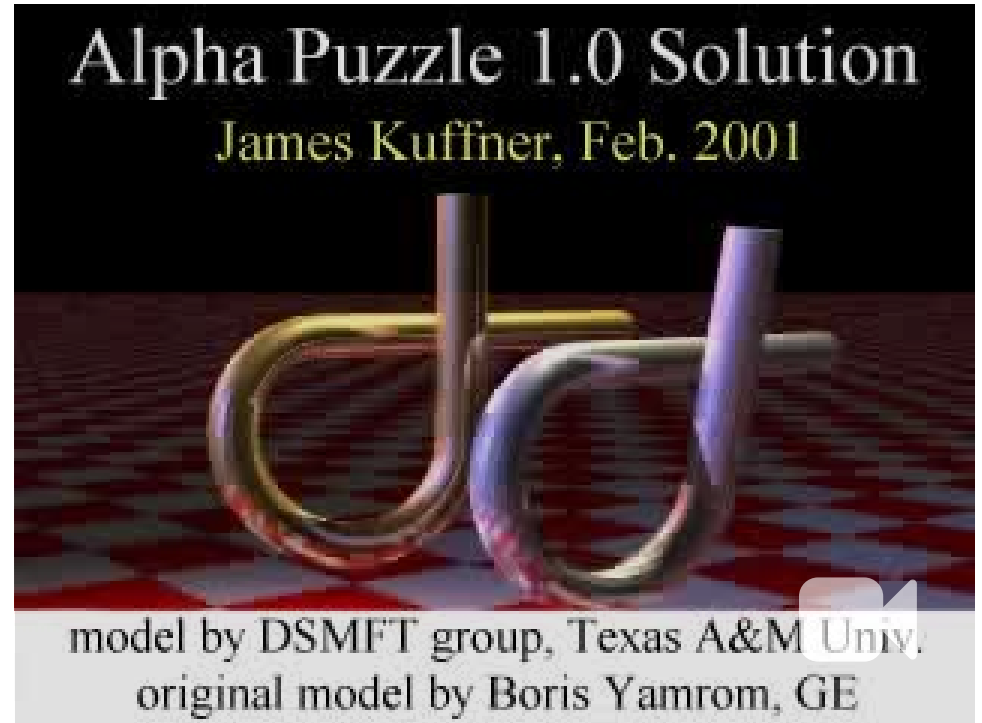
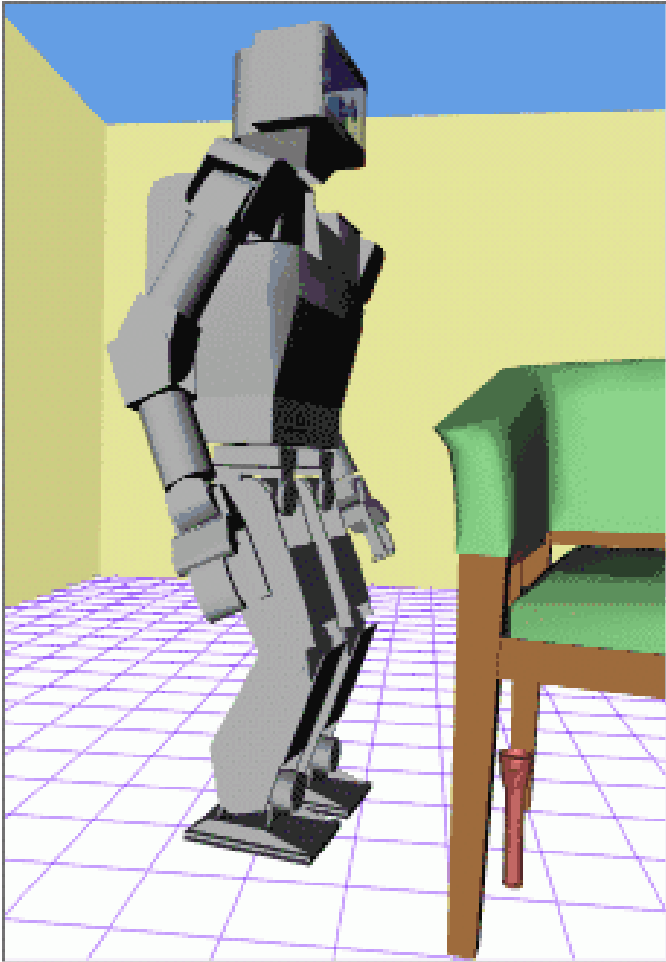
Obstacles in C-space
xyz continuous
discretized angles

Illustrations from 1980 paper 3

Handey (Lozano-Perez, et al) 1987-89



Sample-based (Barraquand, Kavraki, Latombe, LaValle, et al) 1990-2000



<http://www.kuffner.org/james/plan>



Robot Motion Planning on a Chip

Sean Murray, Will Floyd-Jones*, Ying Qi*, Daniel Sorin and George Konidakis
Duke Robotics

Departments of Computer Science and Electrical & Computer Engineering
Duke University, Durham NC 27708

[https://www.youtube.com/embed/u4snHh_S_Ao?
start=40&rel=0&enablejsapi=1](https://www.youtube.com/embed/u4snHh_S_Ao?start=40&rel=0&enablejsapi=1)

Motions in Microseconds via Vectorized Sampling-Based Planning

Wil Thomason[†], Zachary Kingston[†], and Lydia E. Kavraki

Abstract—Modern sampling-based motion planning algorithms typically take between hundreds of milliseconds to dozens of seconds to find collision-free motions for high degree-of-freedom problems. This paper presents performance improvements of more than 500x over the state-of-the-art, bringing planning times into the range of microseconds and scales into the range of kilohertz, without specialized hardware. Our key insight is how to exploit fine-grained parallelism within planning, providing generality-preserving algorithmic improvements to any such planner and significantly accelerating critical subroutines, such as forward kinematics and collision checking. We demonstrate our approach over a diverse set of challenging, realistic problems for complex robots ranging from 6 to 14 degrees-of-freedom. Moreover, we show our approach does not require high-power hardware by evaluating on a low-single-board computer. The planning speeds demonstrate it is fast enough to reside in the range of control frequencies, opening up new avenues of motion planning research.

I. INTRODUCTION

High degree-of-freedom (DoF) robots rely on *motion planning* to move in complex workspaces, either using sampling-based approximations [1–5] or numerical optimization. These planners are general and can solve realistic, challenging problems in hundreds of milliseconds to dozens of seconds on consumer CPUs. However, this level of performance is short—it is too slow for reactive operation in evolving environments and hampers algorithms for higher-level autonomy

is the use of *vector-oriented* state representations and planning primitives (e.g., forward kinematics and collision checking), which enable fine interleaving of parallel and

	System	Mean	Q1	Median	Q3	95%	Mean Simpl.	Succ.
Panda	PyBullet/OMPL	58.47	12.76	21.21	43.35	224.41	121.38	99.6%
	MoveIt/OMPL	23.51	14.28	16.58	22.54	63.08	46.95	100%
	VAMP (ARM)	0.70	0.18	0.29	0.62	2.73	1.03	100%
	VAMP	0.10	0.02	0.04	0.08	0.43	0.12	100%
Fetch	PyBullet/OMPL	3035.83	135.88	461.48	1544.34	10358.44	225.17	99.8%
	MoveIt/OMPL	788.68	94.11	243.80	666.56	3107.80	137.14	100%
	VAMP (ARM)	28.24	1.72	5.76	23.25	116.74	3.93	99.5%
	VAMP	6.25	0.25	1.12	4.92	25.16	0.52	99.5%
Baxter	PyBullet/OMPL	4309.19	881.31	1500.44	2711.61	13100.54	938.62	99.3%
	MoveIt/OMPL	668.80	192.23	362.34	757.69	2267.35	96.81	100%
	VAMP (ARM)	32.54	8.02	13.93	25.89	104.57	9.77	100%
	VAMP	6.68	1.22	2.32	4.68	22.97	1.22	100%

Table I. Planning time for RRT-Connect from Figs. 3 to 5. The mean, first quantile, median, third quantile, and 95% quantile are shown, along with mean simplification time and success rate. All times are in **milliseconds**.

CUDA-Accelerated Robot Motion Generation in Milliseconds with NVIDIA cuRobo

<https://www.youtube.com/embed/t2O9sj3yQMs?enablejsapi=1>

https://manipulation.csail.mit.edu/data/rrt_basic.html