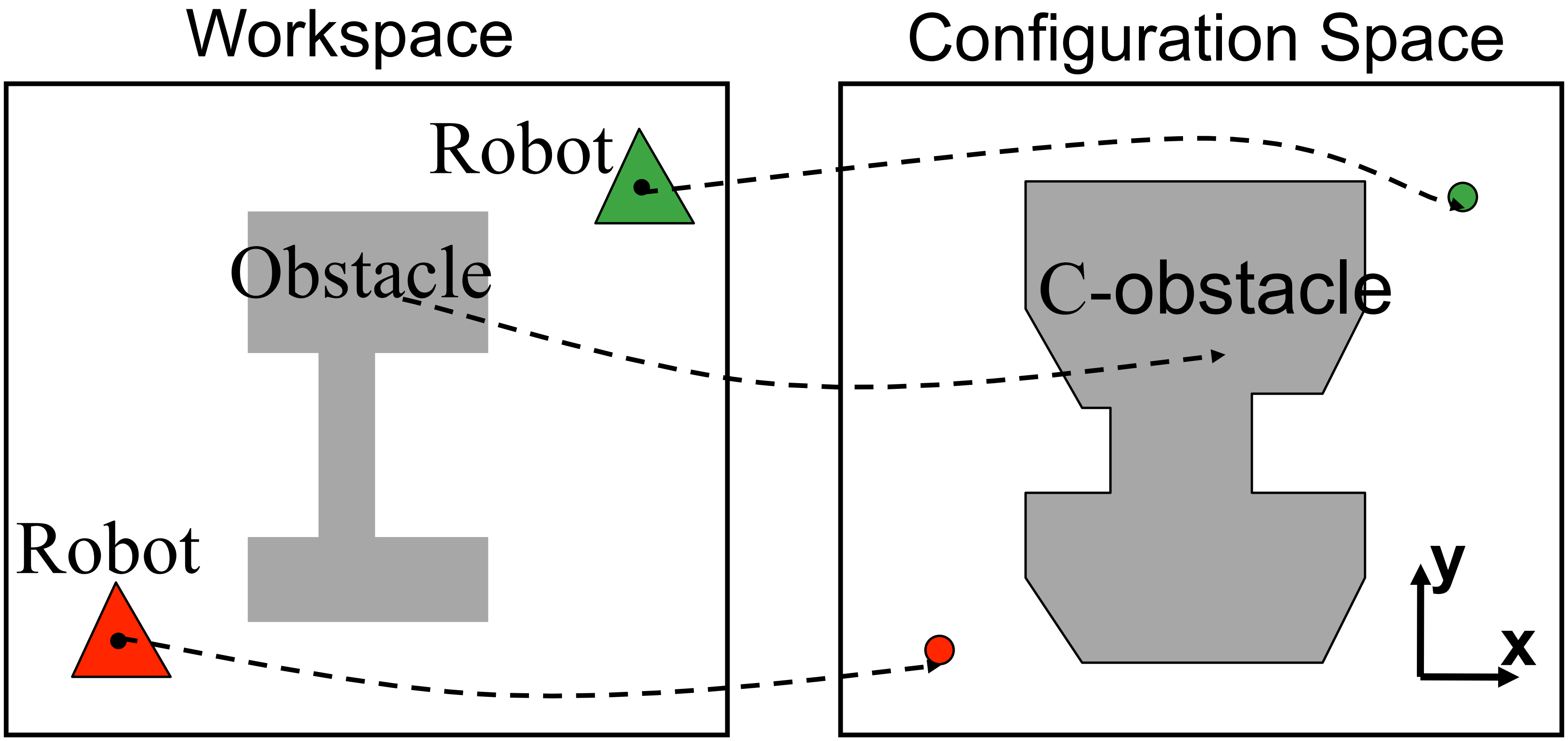
CS425 GAME PROGRAMMING

Path Planning

Configuration Space

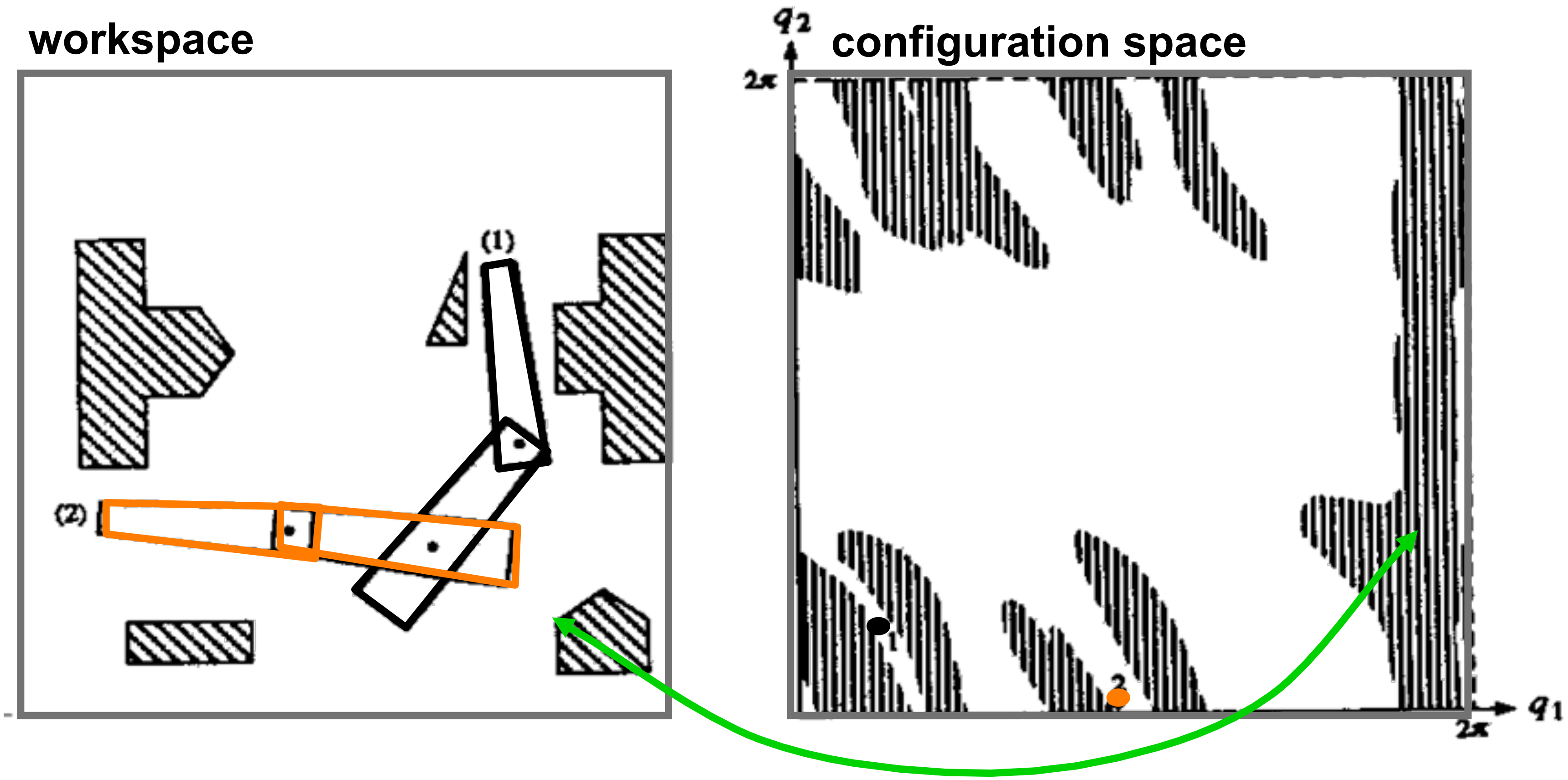
- Convert rigid robots, articulated robots, *etc.* into points
- Apply algorithms for moving points

Configuration Space



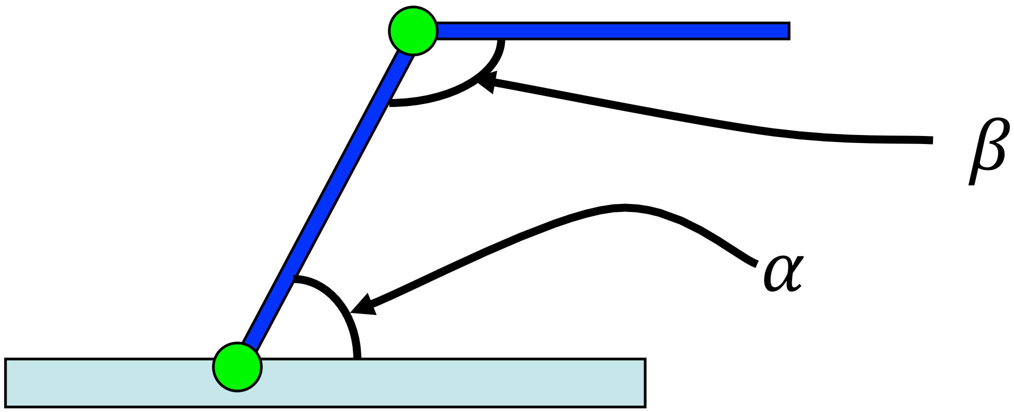
- C-obstacle is a polygon.

Configuration Space



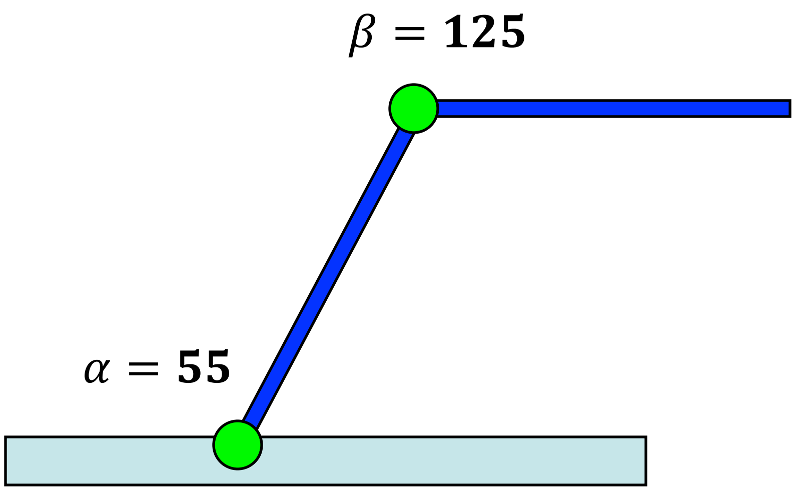
Workspace

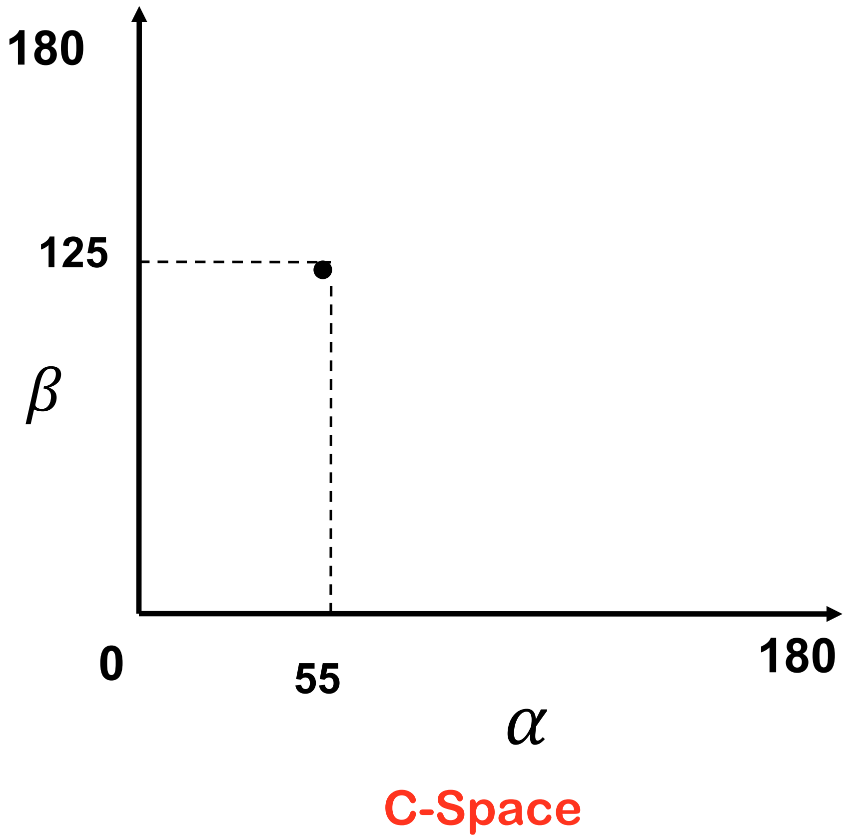
Degree of freedom (DOF)



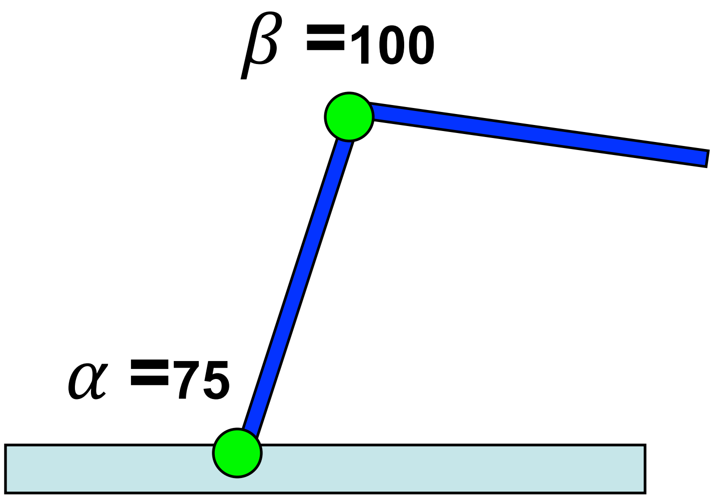
Configuration Space

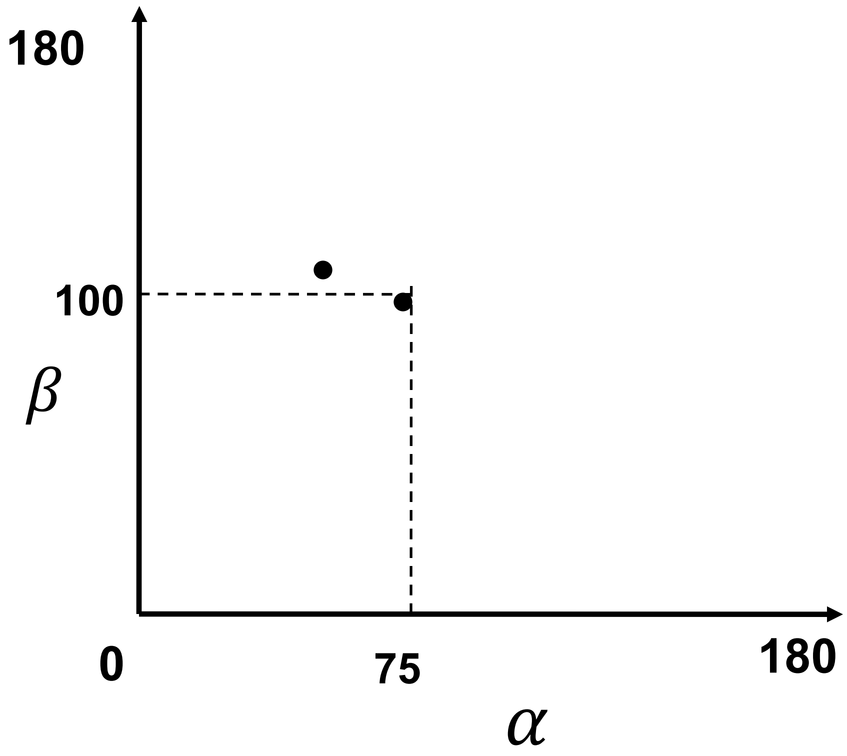
C-Space





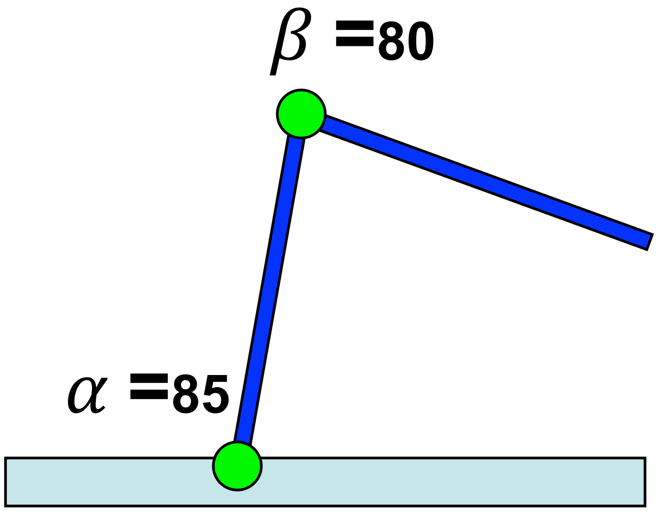
C-Space

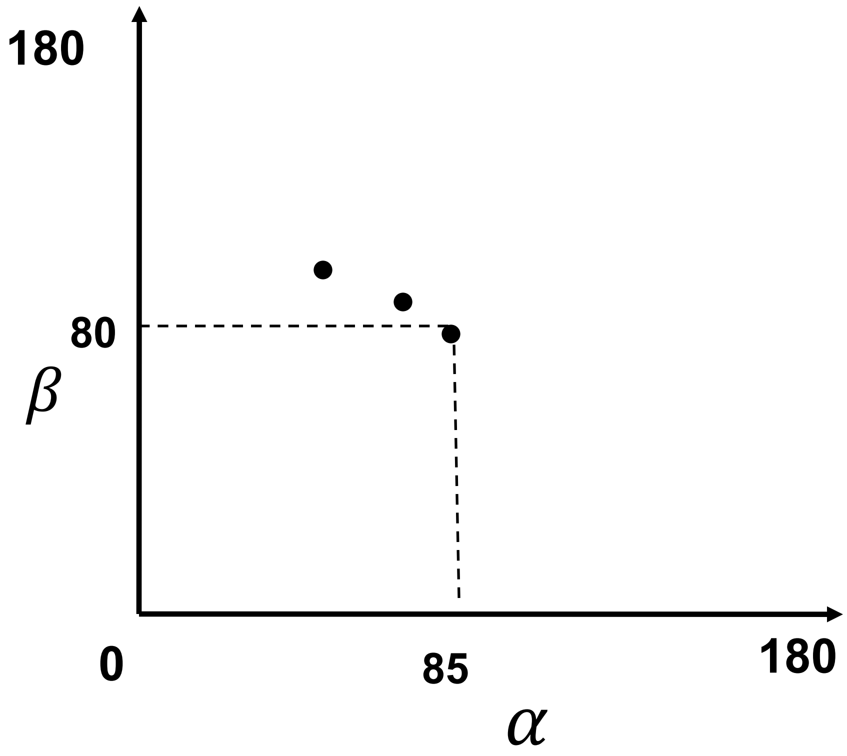




C-Space

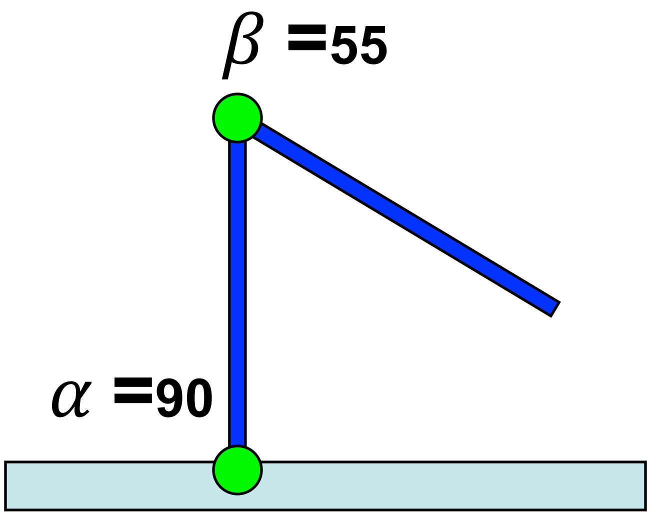
C-Space

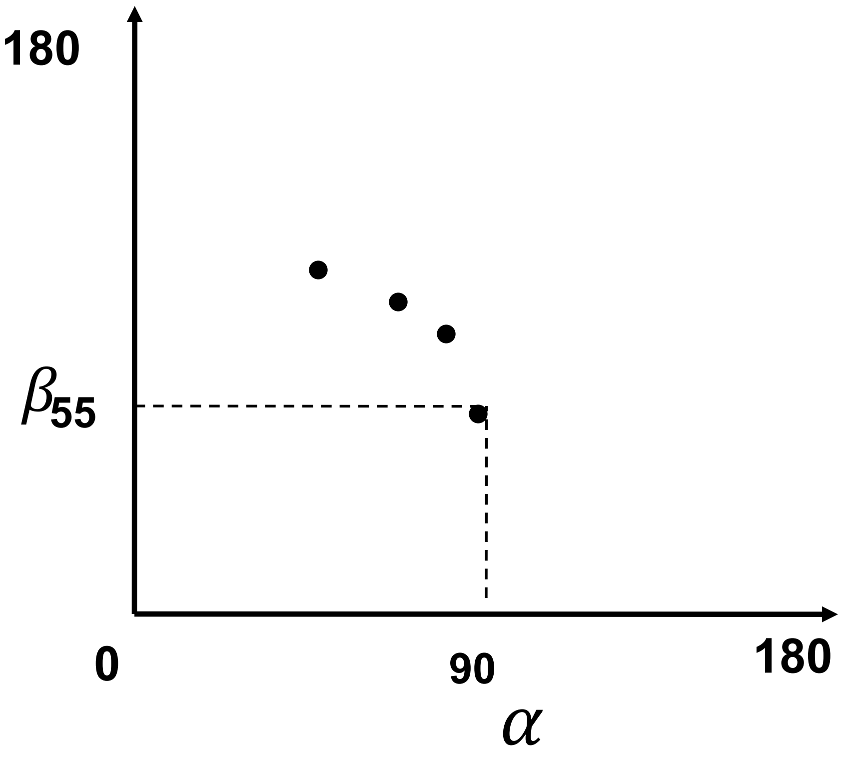




C-Space

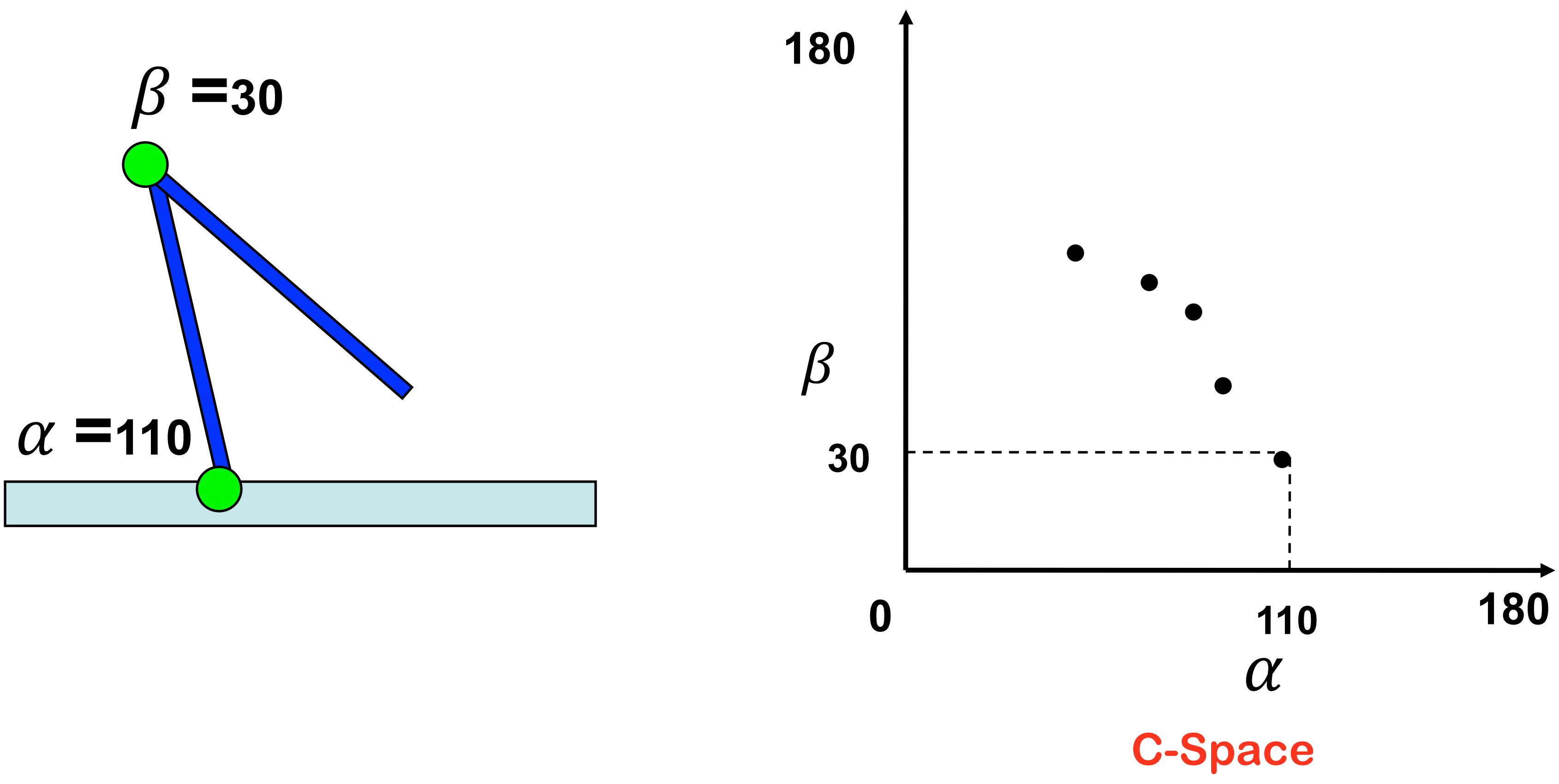
C-Space



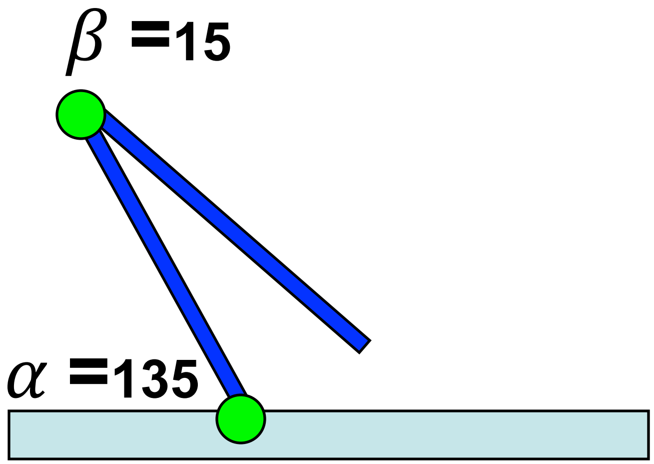


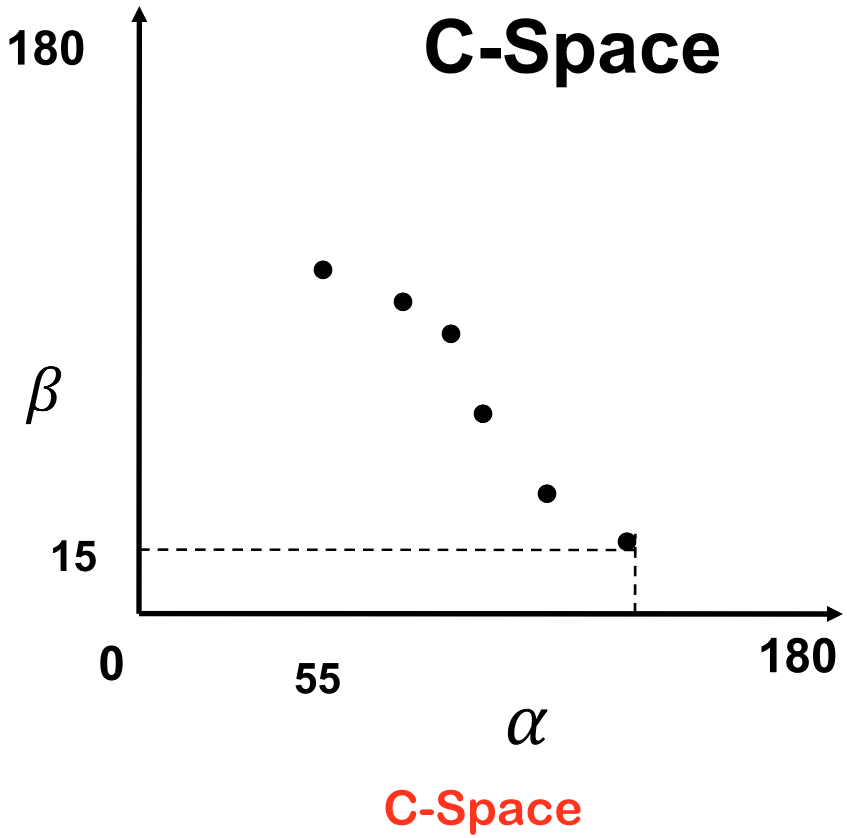
C-Space

C-Space

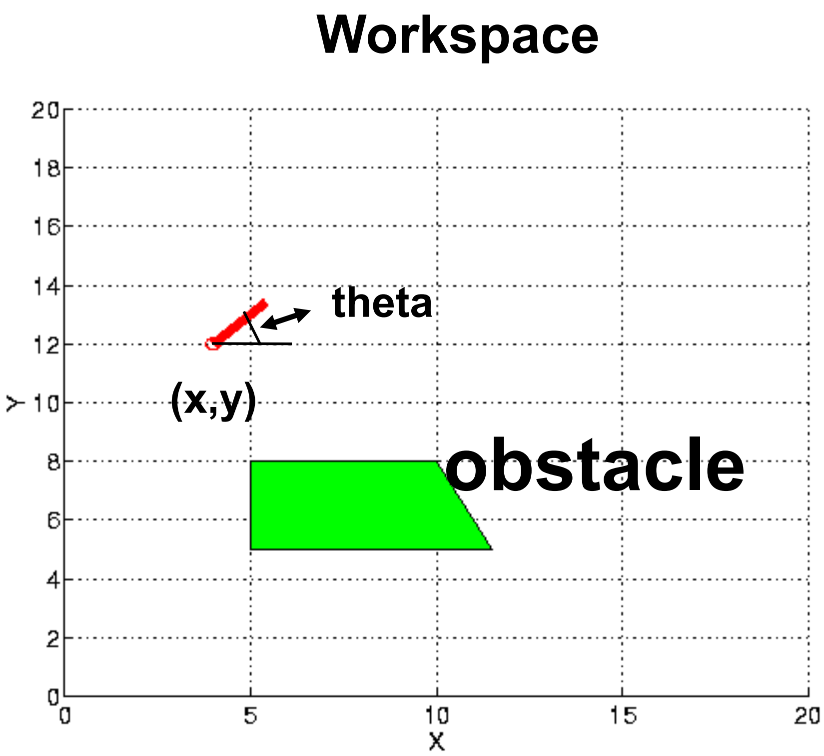


C-Space

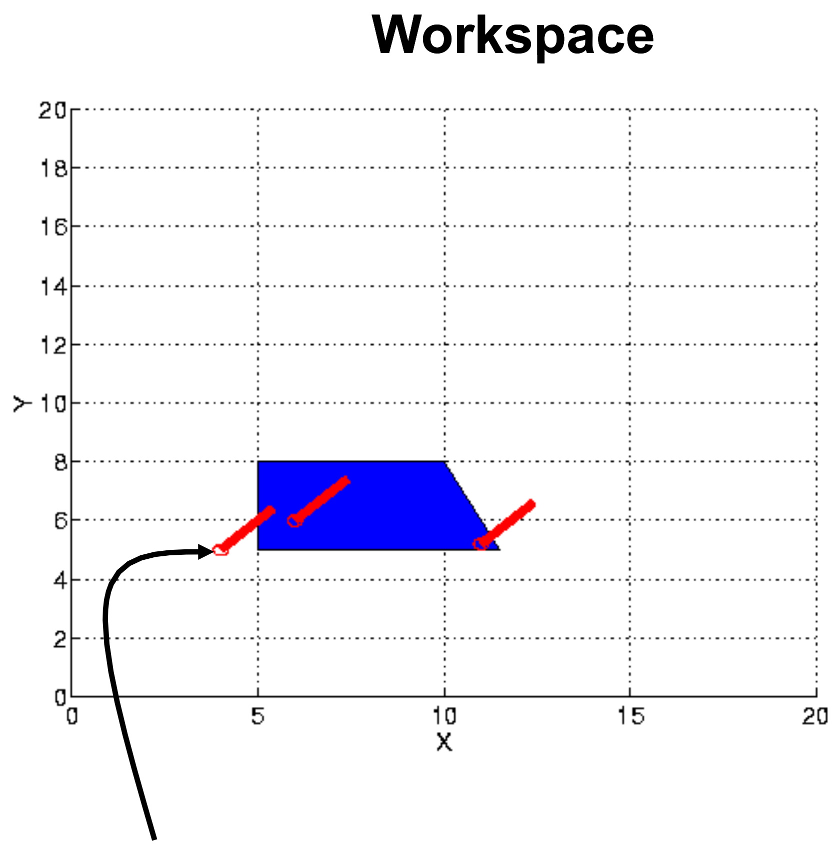




Workspace Obstacle

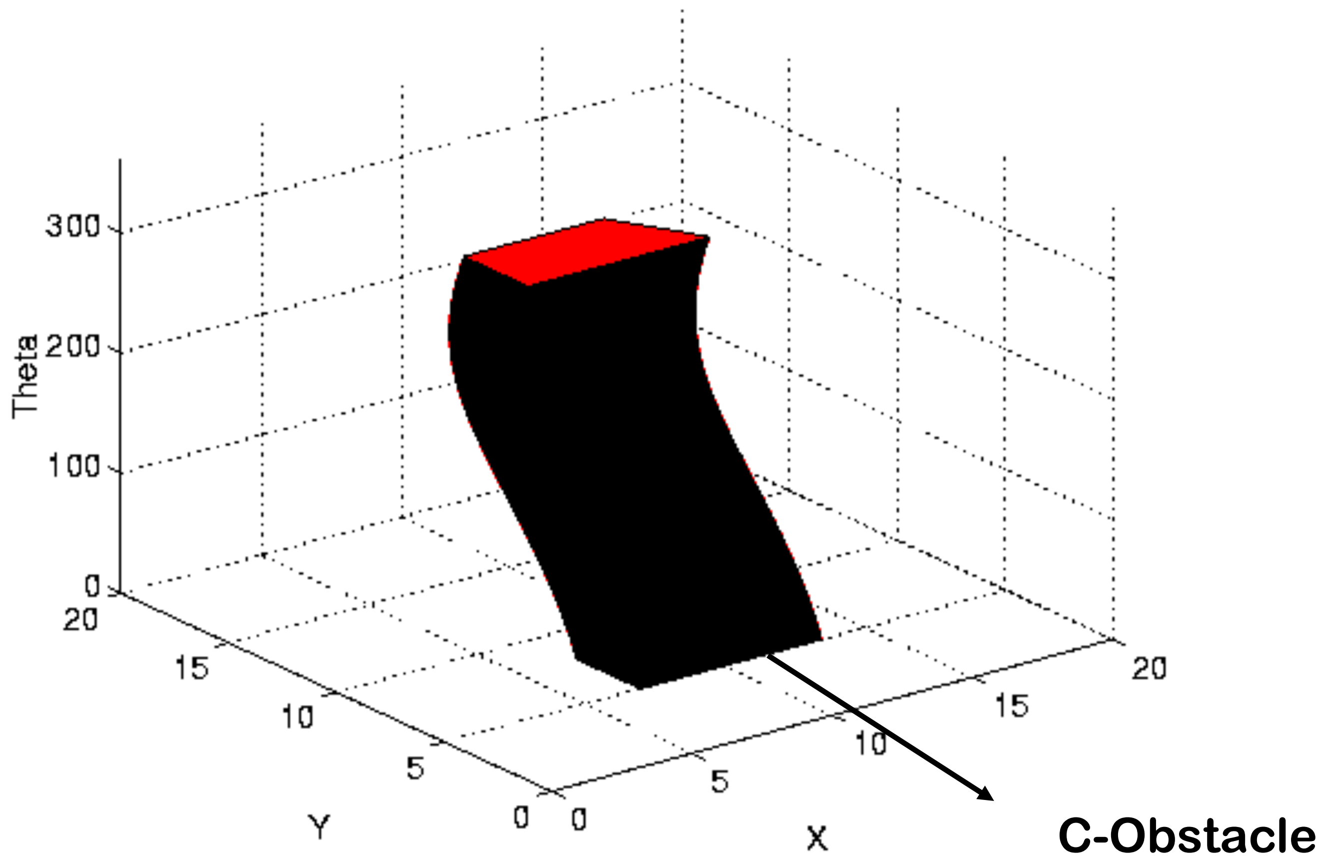


Configuration (x,y,theta)



(4,5,45)

C-Space Obstacle

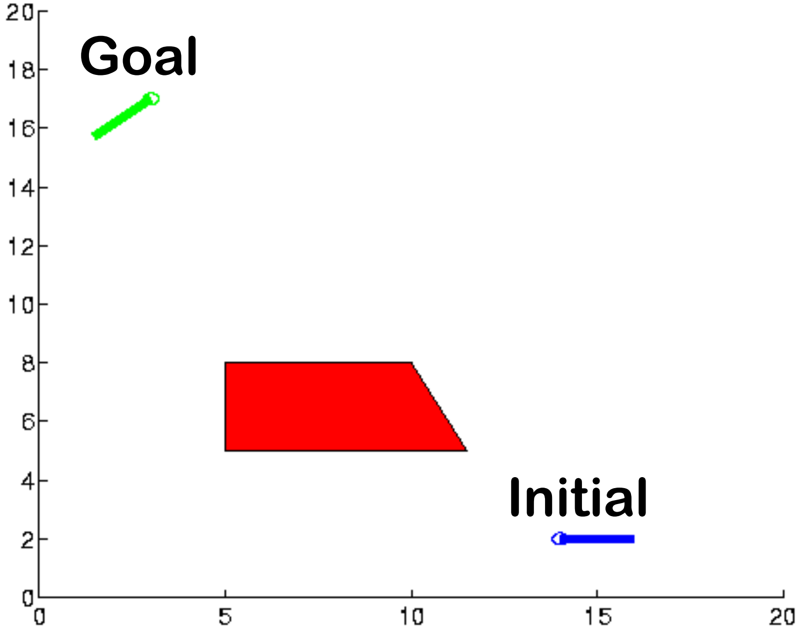


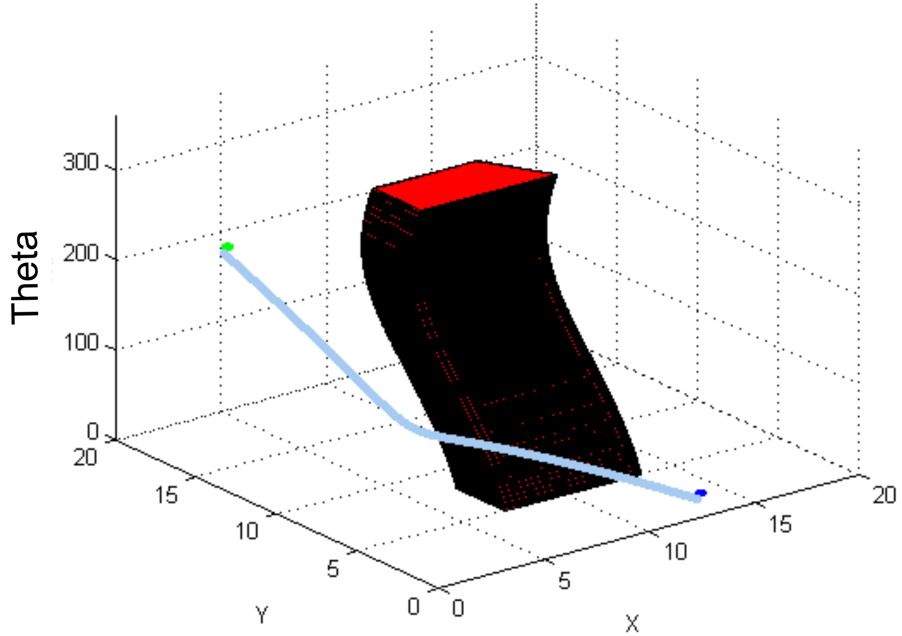
Really look like this???

Finding a Path

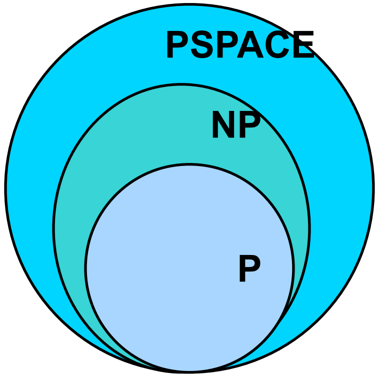
Find a path in workspace for a robot

Find a path in C-space for a point





The Complexity of Motion Planning



General motion planning problem is PSPACE-hard [Reif 79, Hopcroft et al. 84 & 86]
PSPACE-complete [Canny 87]

The best deterministic algorithm known has running time that is exponential in the dimension of the robot's C-space [Canny 86]

- C-space has high dimension - 6D for rigid body in 3-space
- simple obstacles have complex C-obstacles → impractical to compute explicit representation of freespace for more than 4 or 5 dof

So … attention has turned to *randomized algorithms*

Probabilistic Methods

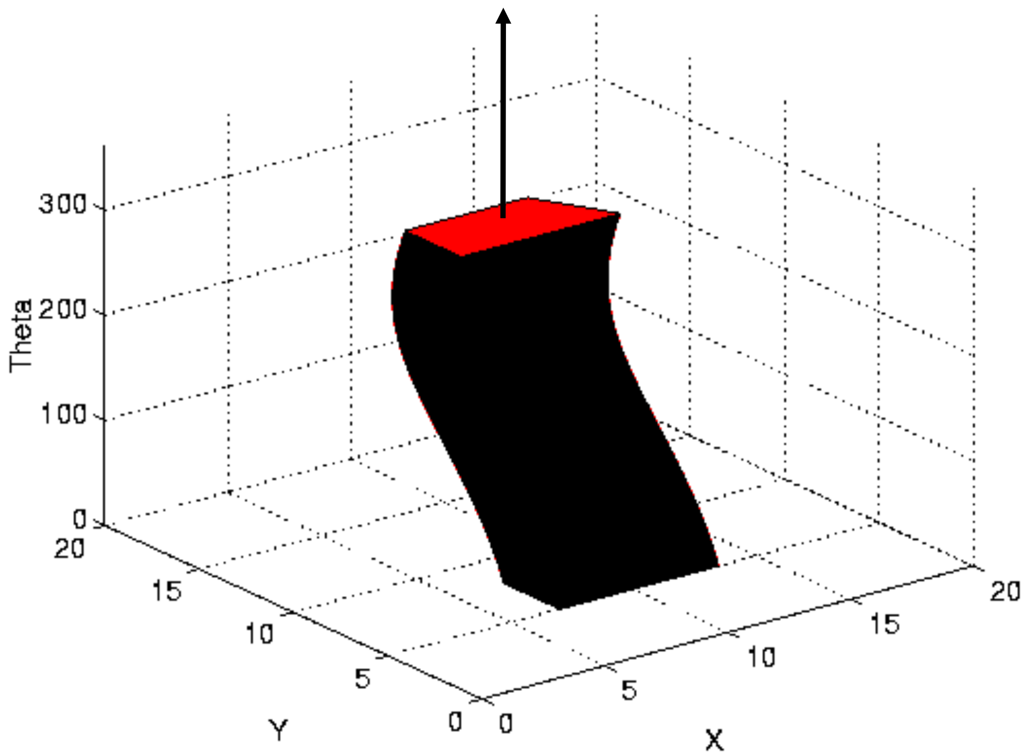
- Avoid computing C-obstacles
  - Too difficult to compute efficiently
- Idea: Sacrifice completeness to gain simplicity and efficiency
- Probabilistic Methods
  - Graph based
  - Tree based

Probabilistic Roadmap Method

[Kavraki, Svestka, Latombe,Overmars 1996]

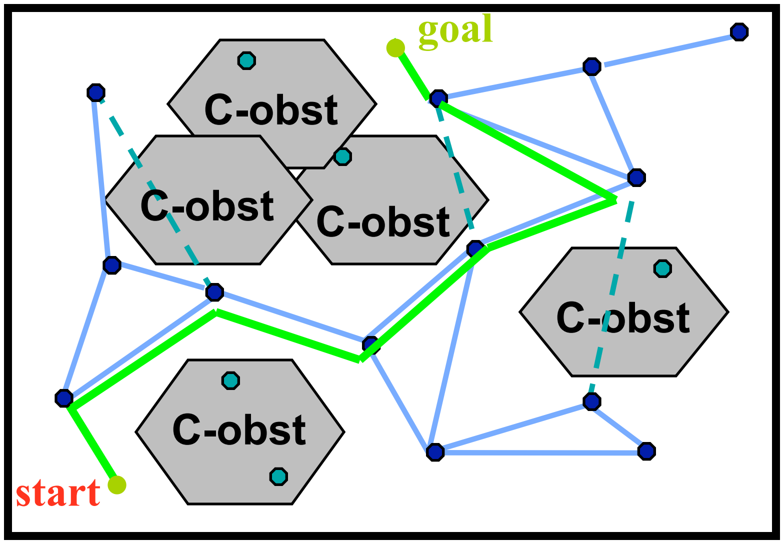
unknown

Theta



Probabilistic Roadmap Method

C-space



Roadmap Construction (Pre-processing)

1. Randomly generate robot configurations (nodes)
   - discard nodes that are invalid
2. Connect pairs of nodes to form **roadmap**
   - simple, deterministic *local planner* (e.g., straightline)
   - discard paths that are invalid

Query processing

1. Connect *start* and *goal* to roadmap
2. Find path in roadmap between *start* and *goal*
   - regenerate plans for edges in roadmap

Probabilistic Roadmap Method

Algorithm 1: PRM (preprocessing phase)

$V \leftarrow \emptyset$; $E \leftarrow \emptyset$;
for $i = 0, \ldots, n$ **do**
    $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
    $U \leftarrow \texttt{Near}(G = (V, E), x_{\text{rand}}, r)$;
    $V \leftarrow V \cup \{x_{\text{rand}}\}$;
    foreach $u \in U$, *in order of increasing* $\|u - x_{\text{rand}}\|$, **do**
        if x_{rand} *and* u *are not in the same connected component of* $G = (V, E)$ **then**
            if $\texttt{CollisionFree}(x_{\text{rand}}, u)$ **then** $E \leftarrow E \cup \{(x_{\text{rand}}, u), (u, x_{\text{rand}})\}$;
return $G = (V, E)$;

Probabilistic Roadmap Method

Algorithm 2: sPRM

$V \leftarrow \{x_{\text{init}}\} \cup \{\texttt{SampleFree}_i\}_{i=1,\ldots,n}$; $E \leftarrow \emptyset$;

foreach $v \in V$ **do**

  $U \leftarrow \texttt{Near}(G = (V, E), v, r) \setminus \{v\}$;

  foreach $u \in U$ **do**

    if $\texttt{CollisionFree}(v, u)$ **then** $E \leftarrow E \cup \{(v, u), (u, v)\}$

return $G = (V, E)$;

Probabilistic Methods

- Avoid computing C-obstacles
  - Too difficult to compute
- Sacrifice completeness to gain simplicity and efficiency - probabilistic complete!
- Probabilistic Methods
  - Graph based
  - Tree based - single-shot planners!

Rapidly-Exploring Random Tree (RRT)

- RRTs: Rapidly-exploring Random Trees

  Rapidly-exploring random trees: Progress and prospects. S. M. LaValle and J. J. Kuffner. In *Proceedings Workshop on the Algorithmic Foundations of Robotics*, 2000.)

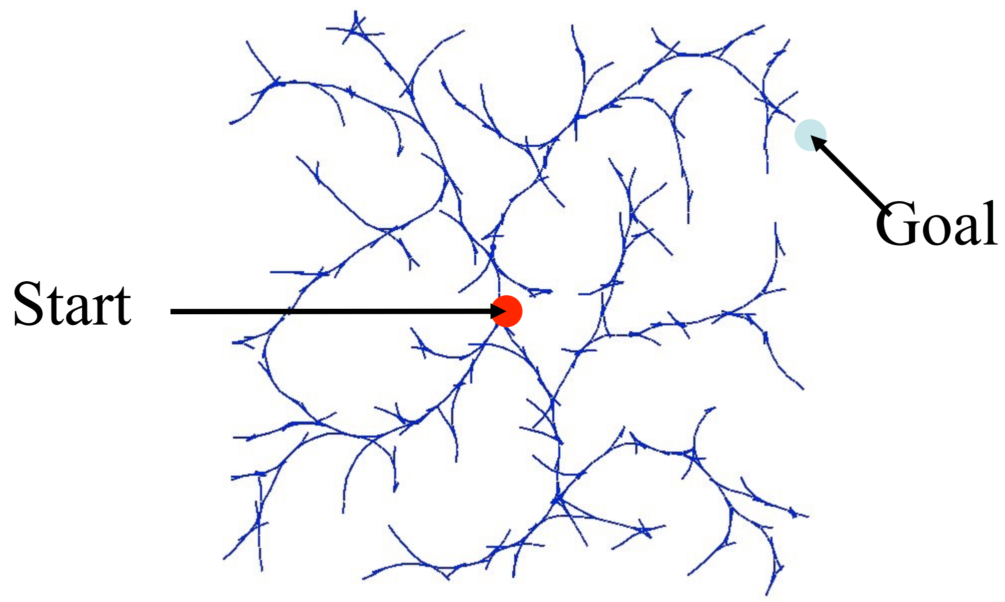
  – Incrementally builds the roadmap tree

- Extends to more advanced planning techniques

  – Integrates the control inputs to ensure that the kinodynamic constraints are satisfied

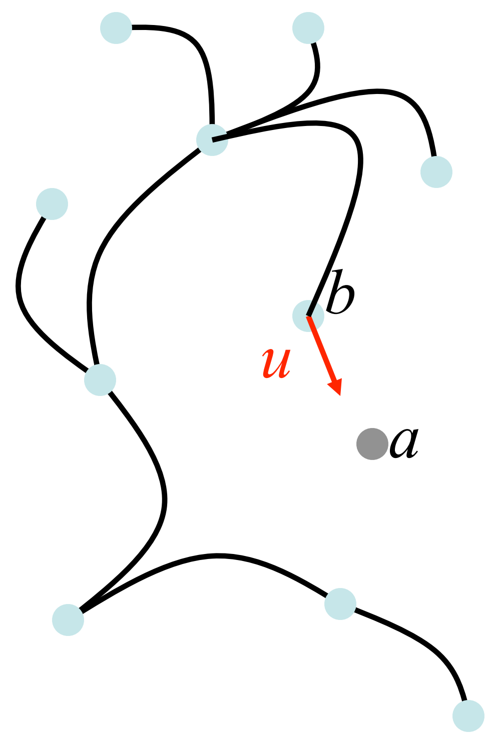
How it Works

- Build a rapidly-exploring random tree in state space (X), starting at s_{start}
- Stop when tree gets sufficiently close to s_{goal}



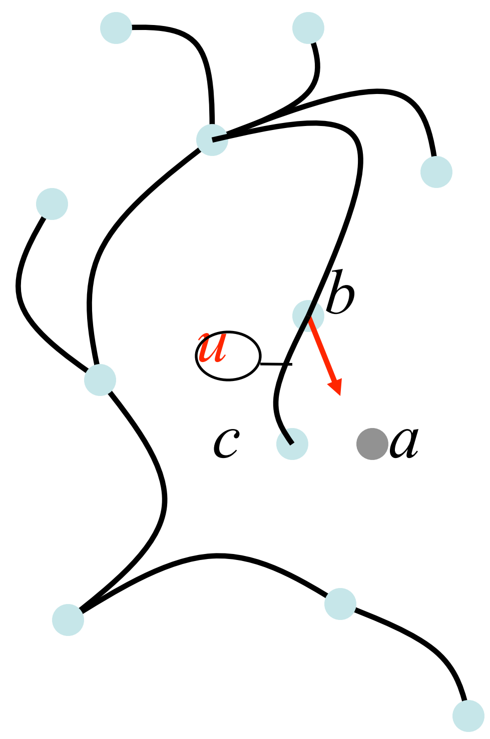
Building an RRT

- To extend an RRT:
  - Pick a random point *a* in *X*
  - Find *b*, the node of the tree closest to *a*
  - Find control inputs *u* to steer the robot from *b* to *a*



Building an RRT

- To extend an RRT (cont.)
  - Apply control inputs *u* for time δ, so robot reaches *c*
  - If no collisions occur in getting from *a* to *c*, add *c* to RRT and record *u* with new edge



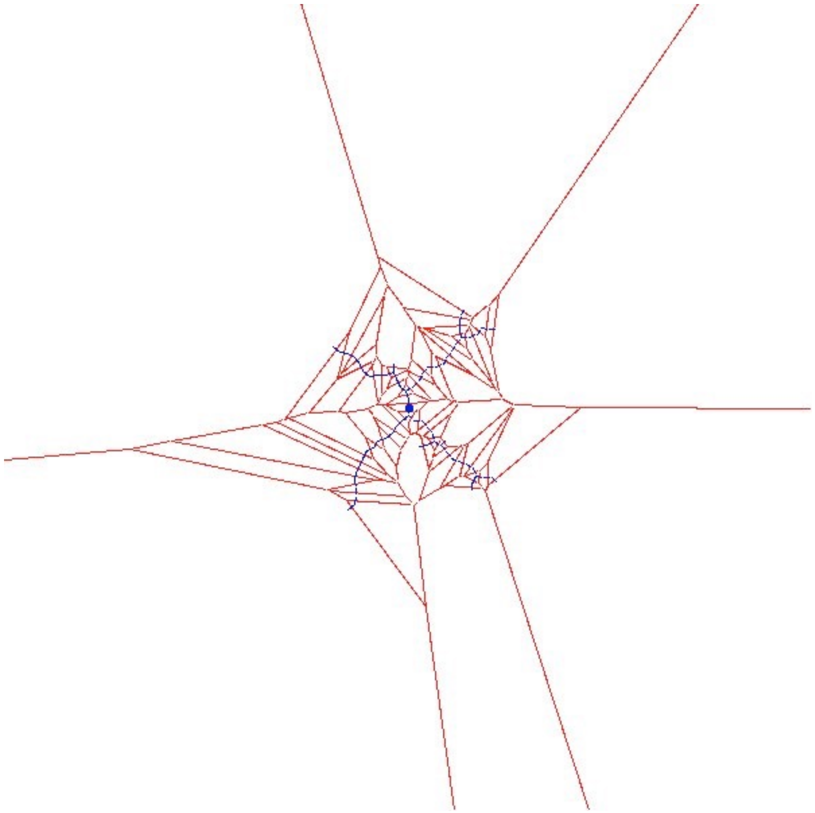
Executing the Path

Once the RRT reaches s_{goal}

- Backtrack along tree to identify edges that lead from s_{start} to s_{goal}
- Drive robot using control inputs stored along edges in the tree

Principle Advantage

- RRT quickly explores the state space:
  - Nodes most likely to be expanded are those with largest Voronoi regions

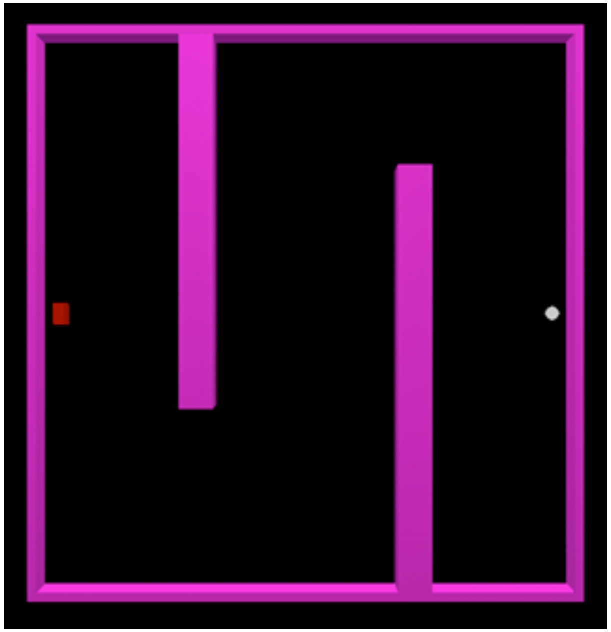


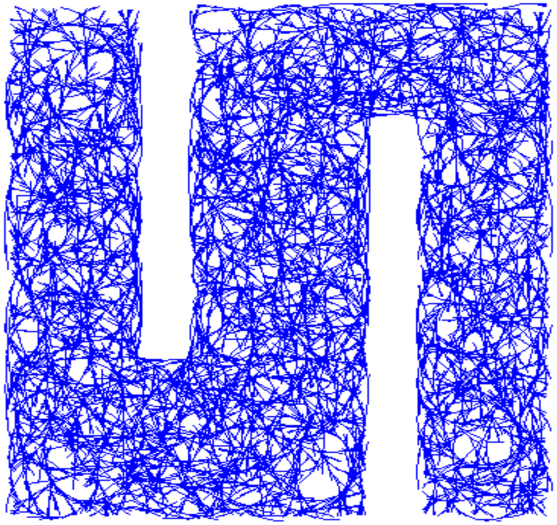
RRT

Algorithm 3: RRT

$V \leftarrow \{x_{\text{init}}\};\ E \leftarrow \emptyset;$
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i;$
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}});$
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})\ ;$
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $V \leftarrow V \cup \{x_{\text{new}}\};\ E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\}\ ;$
return $G = (V, E);$

Problem of Simple RRT Planner

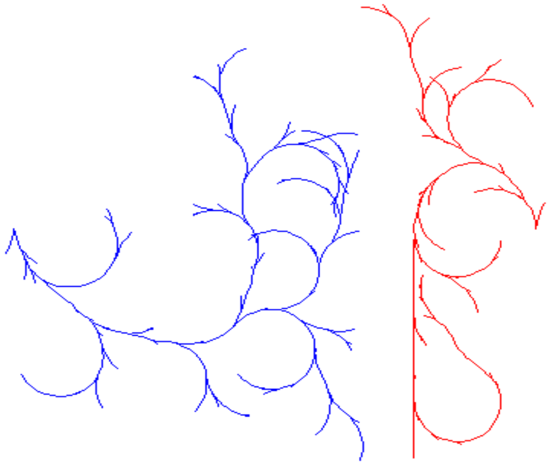




- Problem: ordinary RRT explores *X* uniformly
  - → slow convergence
- Solution: bias distribution towards the goal

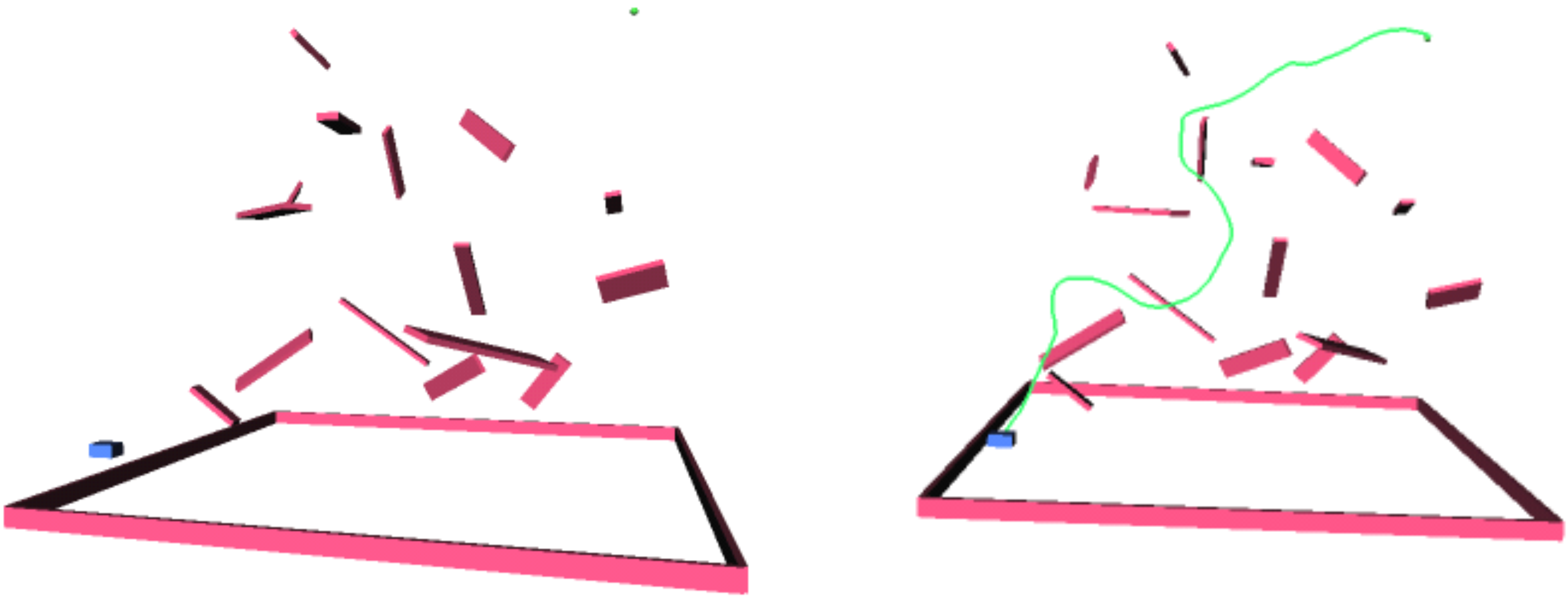
Bidirectional Planners

- Build two RRTs, from start and goal state

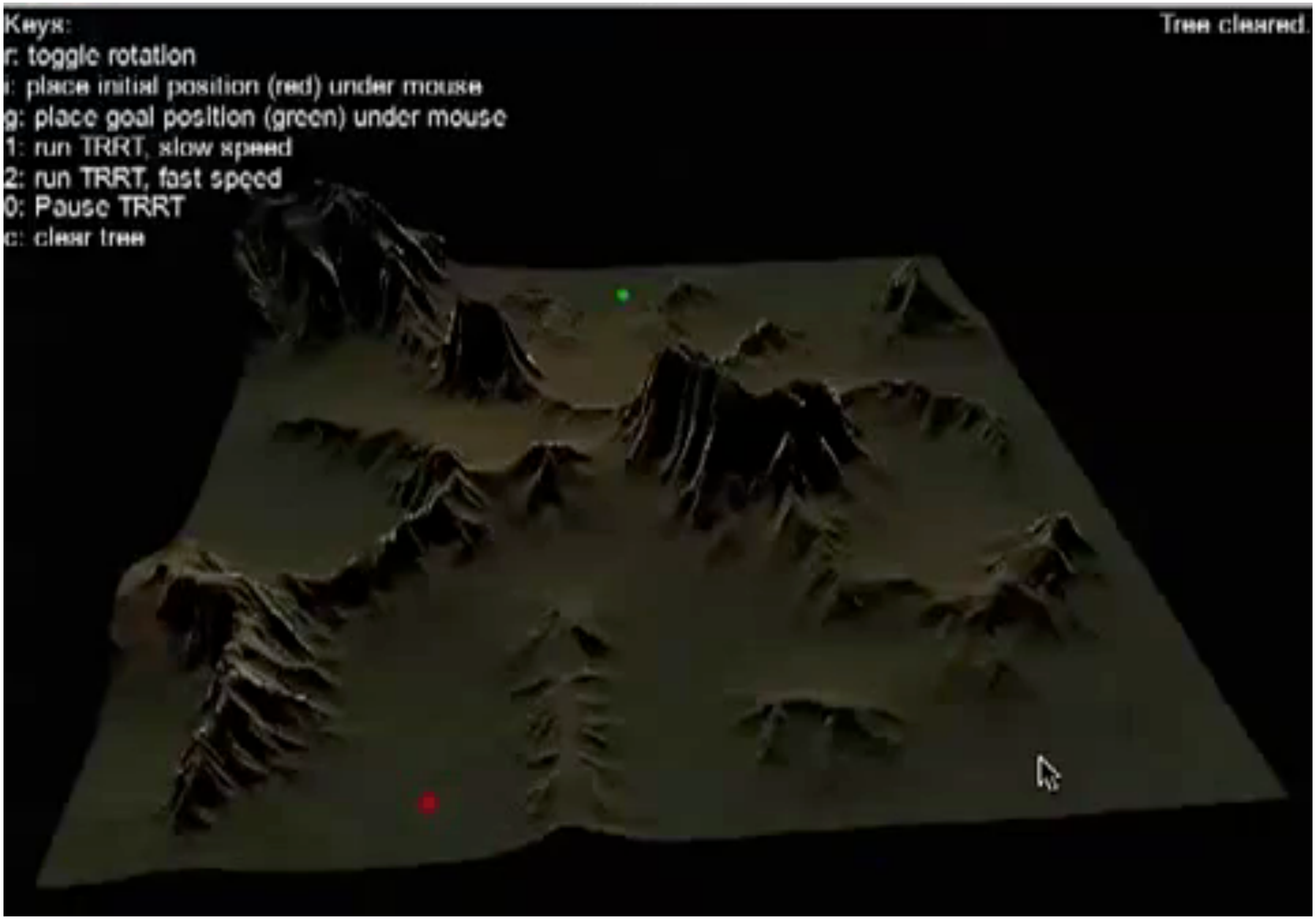


- Complication: need to connect two RRTs
  - local planner will not work (dynamic constraints)
  - **bias** the distribution, so that the trees meet

Bidirectional RRT Example



Consider 2.5D Terrain



Consider Moving Objects