RBE550
Motion Planning
Overview

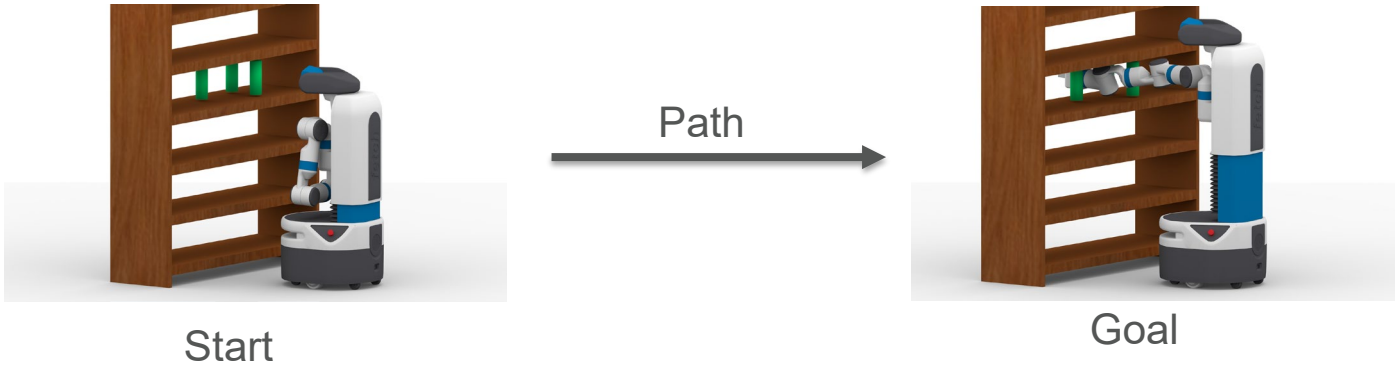
Constantinos Chamzas
https://www.cchamzas.com

Overview

- Motion Planning Definition
- Bug Algorithms
- Discrete Search/Roadmaps
- Configuration Space
- Sampling-Based Planners
- Asymptotically Optimal Planning
- Kinodynamic Planning
- Learning And Planning
- Task and Motion Planning
- Planning Under Uncertainty

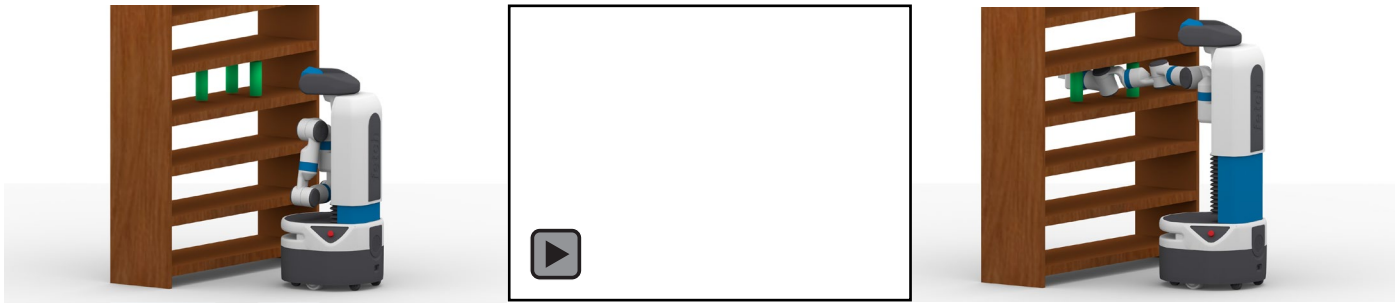
What is Motion Planning?

Research field that designs the trajectories for a robot to follow



What is Motion Planning?

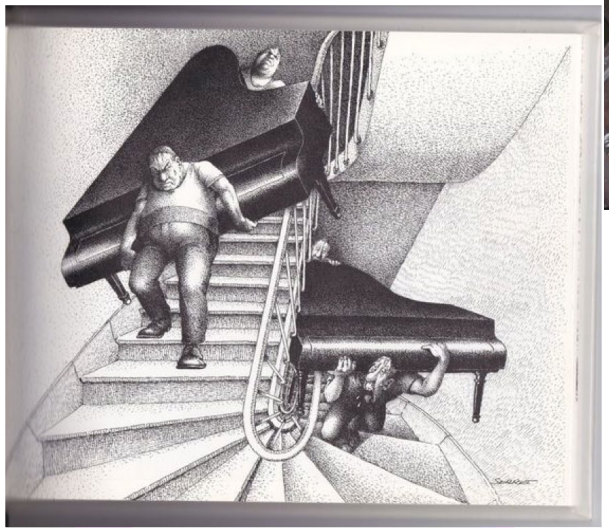
Research field that designs the trajectories for a robot to follow



Start

Goal

Motion Planning- The Piano's Movers Problem



Picture credit: https://www.pinterest.com/pin/178736678931496070/

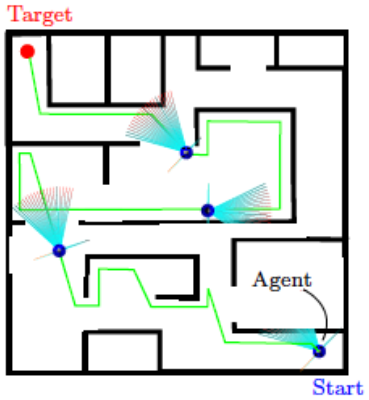
Motion Planning is Hard

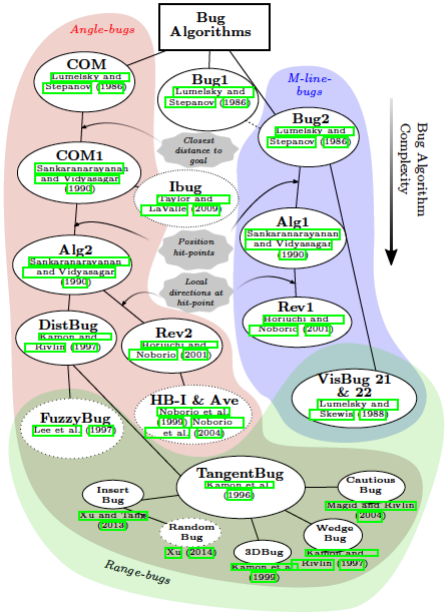
Problem	Complexity
Sofa Mover (3 DOF)	$O(n^{2+\varepsilon})$ not implemented
Piano Mover (6 DOF)	Polynomial – no practical algorithm known
n Disks in the Plane	NP-hard
n Link Planar Chain	PSPACE-Complete
Generalized Mover	PSPACE-Complete
Shortest Path for a Point in 3D	NP-hard
Curvature Constrained Point in 2D	NP-hard
Simplified Coulomb Friction	Undecidable

Overview

- Motion Planning Definition
- Bug Algorithms
- Discrete Search/Roadmaps
- Configuration Space
- Sampling-Based Planners
- Asymptotically Optimal Planning
- Kinodynamic Planning
- Learning And Planning
- Task and Motion Planning
- Planning Under Uncertainty

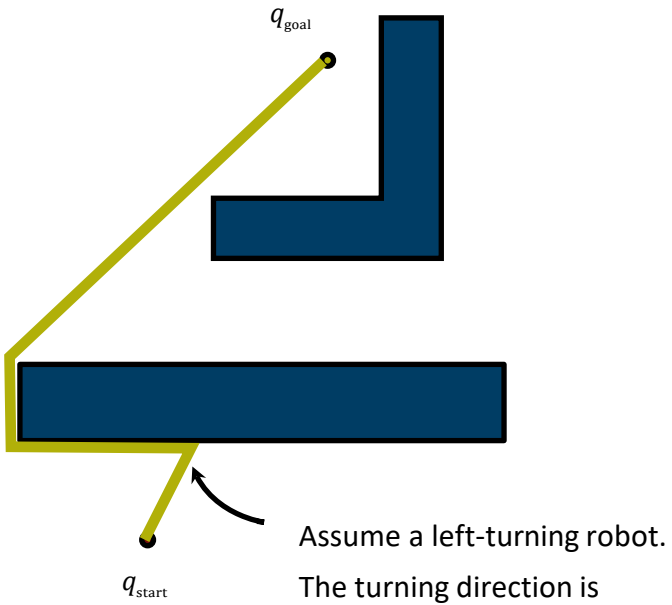
Bug Algorithms





From McGuire at al, 2018

Bug 0

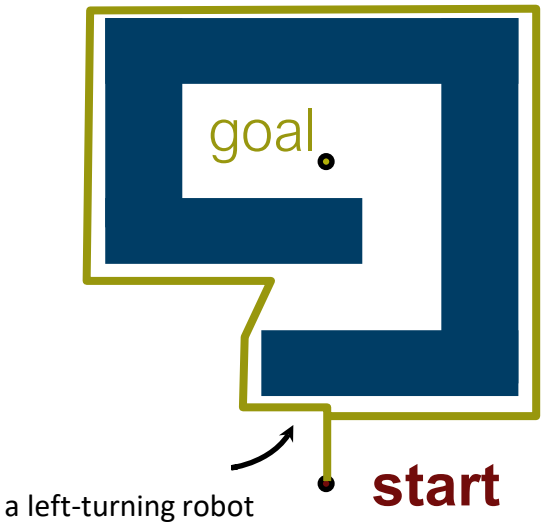


"Bug 0" algorithm

1. head toward goal
2. follow obstacles until you can head toward the goal again
3. continue

done?

What map will foil Bug 0



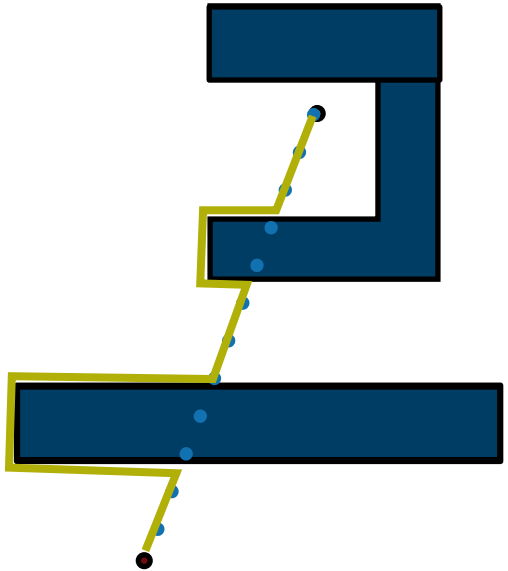
"Bug 0" algorithm

1. head toward goal
2. follow obstacles until you can head toward the goal again
3. continue

Bug 1 Algorithm

- **Assumptions:**
  - known direction to goal
  - otherwise local sensing walls/obstacles & encoders
  - Has memory
- **Strategy:**
  1. head toward goal
  2. if an obstacle is encountered, circumnavigate it and remember how close you get to the goal
  3. return to that closest point (by wall-following) and continue

Bug 2 Algorithm



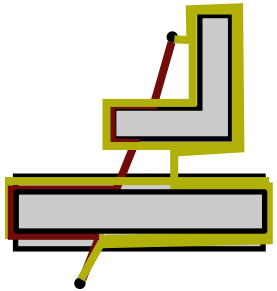
"Bug 2" algorithm

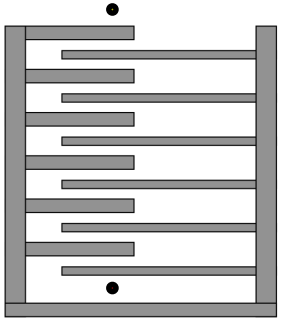
1. head toward goal on the m-line
2. if an obstacle is in the way, follow it until you encounter the m-line again closer to the goal
3. leave the obstacle and continue toward the goal

Better or worse than Bug1?

Bug 1 Vs Bug 2

- BUG 1 is an *exhaustive* search algorithm
  - it looks at all choices before committing
- BUG 2 is a *greedy* algorithm
  - it takes the first thing that looks better
- In many cases, BUG 2 will outperform BUG 1, but
- BUG 1 has a more predictable performance overall





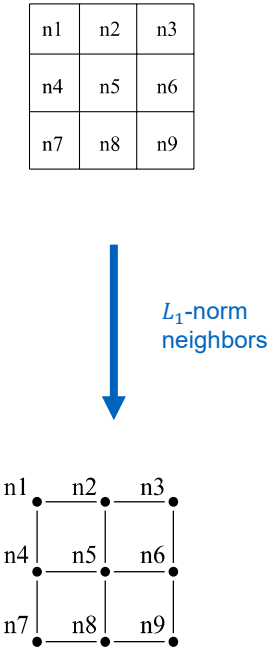
Limitations

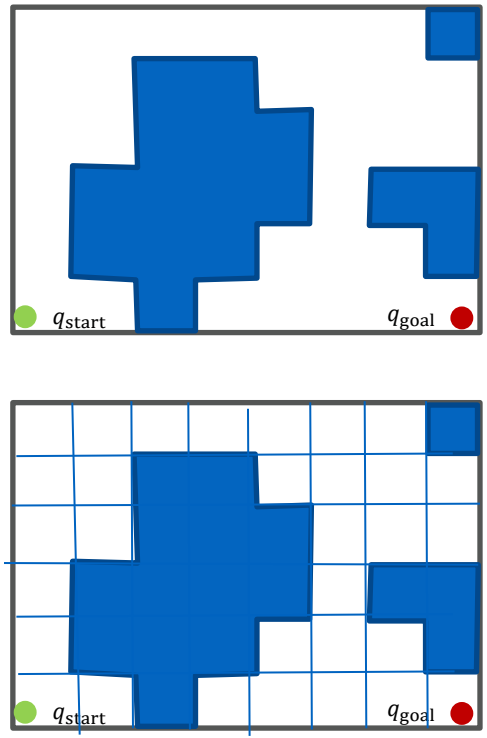
- Relies on knowing the boundary,
- Limited to 2D robots
- Highly non optimal and unsafe motions

Overview

- Motion Planning Definition
- Bug Algorithms
- Discrete Search/Roadmaps
- Configuration Space
- Sampling-Based Planners
- Asymptotically Optimal Planning
- Kinodynamic Planning
- Learning And Planning
- Task and Motion Planning
- Planning Under Uncertainty

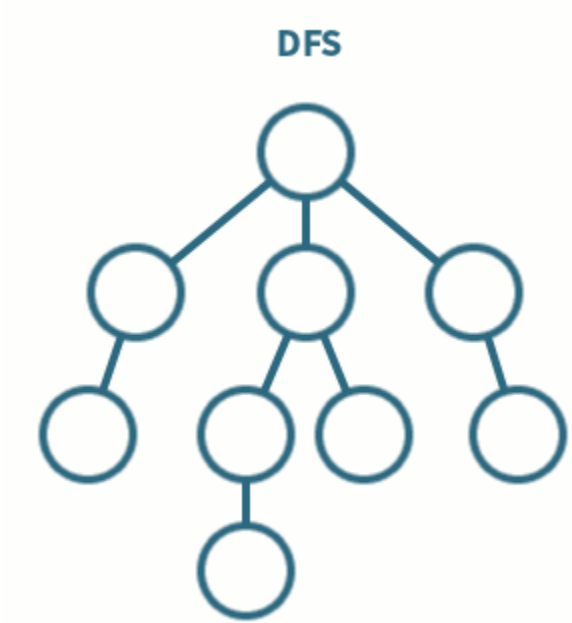
Discretization of Workspace Produces Graphs



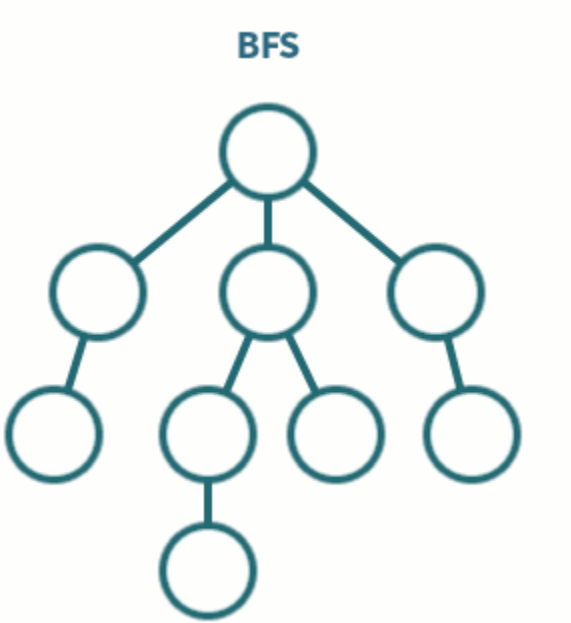


Graph Search Algorithms

- Depth-first search



- Breadth-first search



A* Algorithm

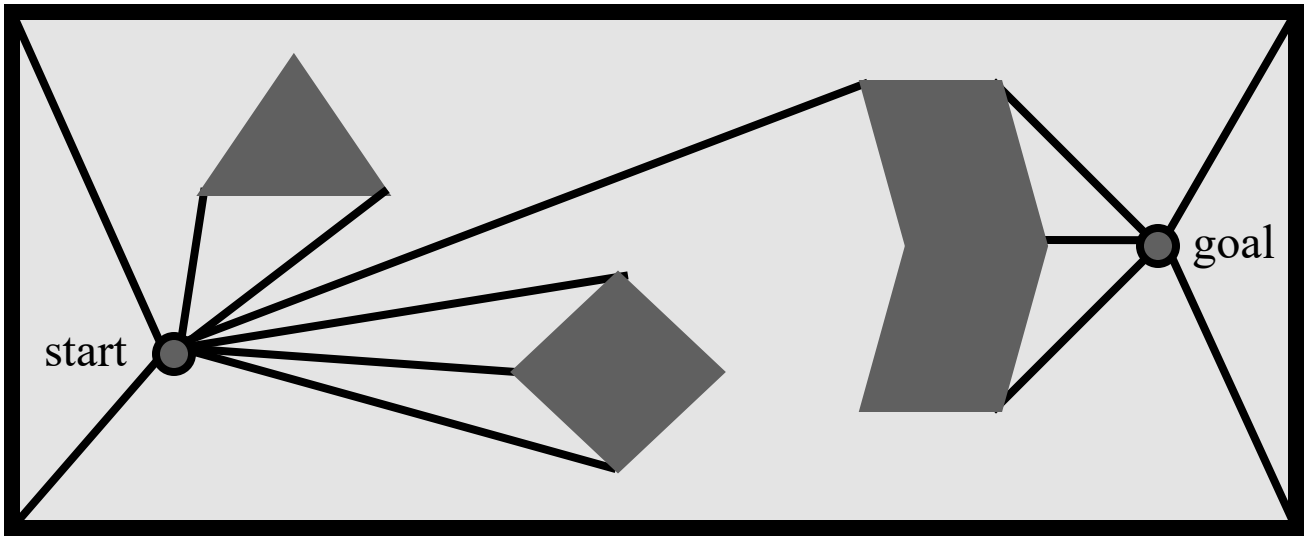
- A* algorithm searches a graph efficiently w.r.t. a given heuristic
  - Heuristic: estimates the cost to the goal node
  - Good heuristic → efficient search
  - Bad heuristic → inefficient search

Limitations

- We need to create a grid
- Does not scale due to the course of dimensionality
- If resolution is not “fine” enough we will not be able to navigate through cluttered obstacles.

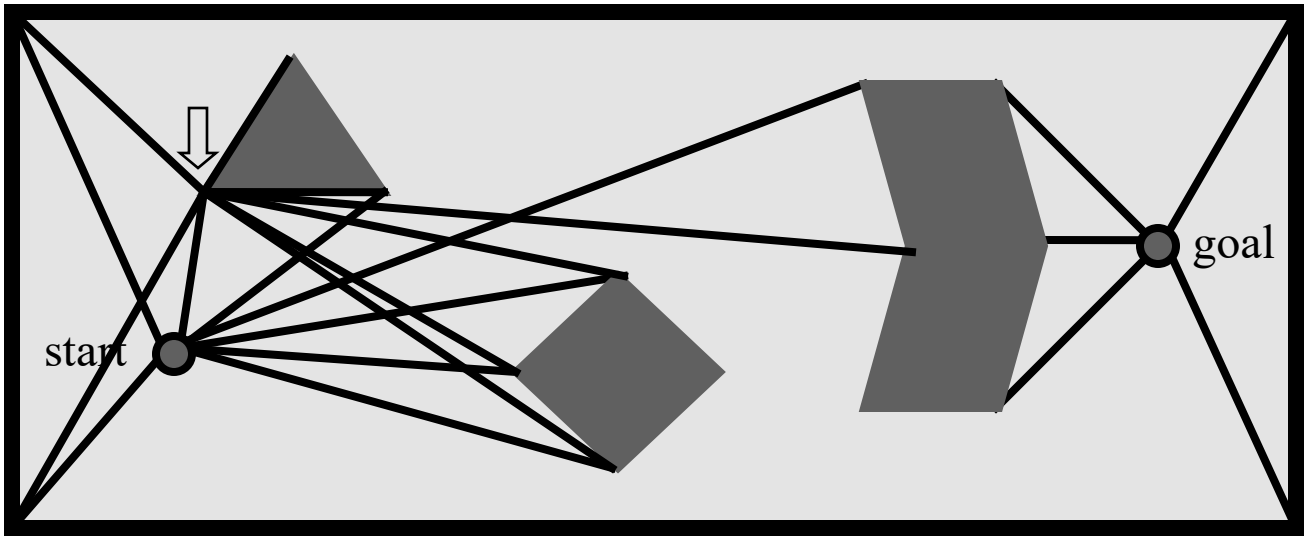
The Visibility Graph in Action (Part 1)

- First, draw lines of sight from the start and goal to all “visible” vertices and corners of the world.



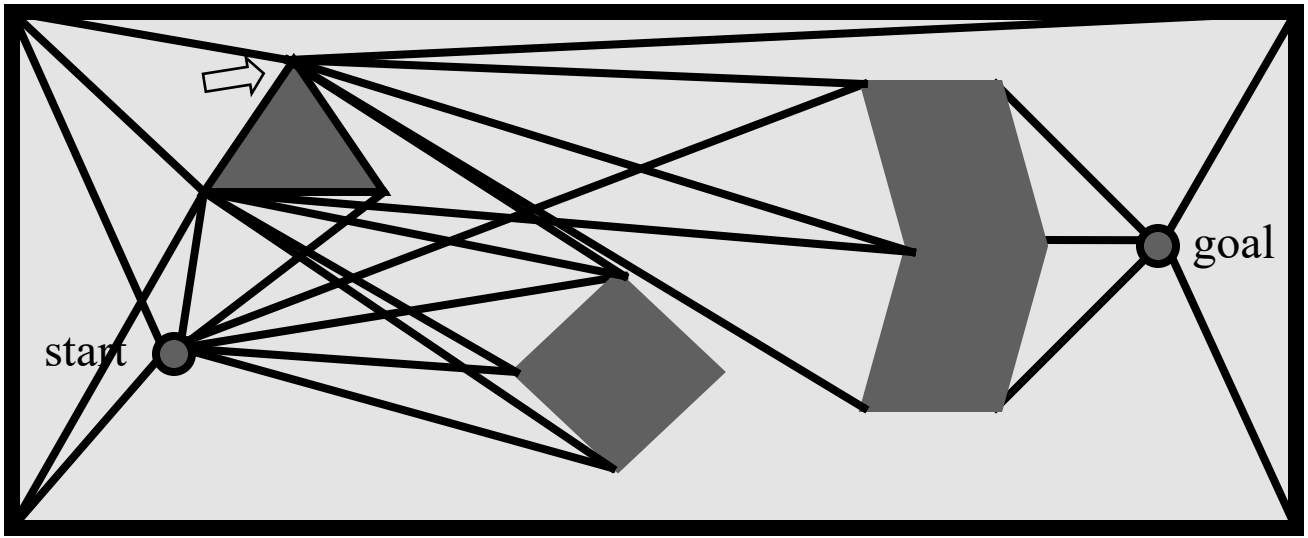
The Visibility Graph in Action (Part 2)

- Second, draw lines of sight from every vertex of every obstacle like before. Remember lines along edges are also lines of sight.



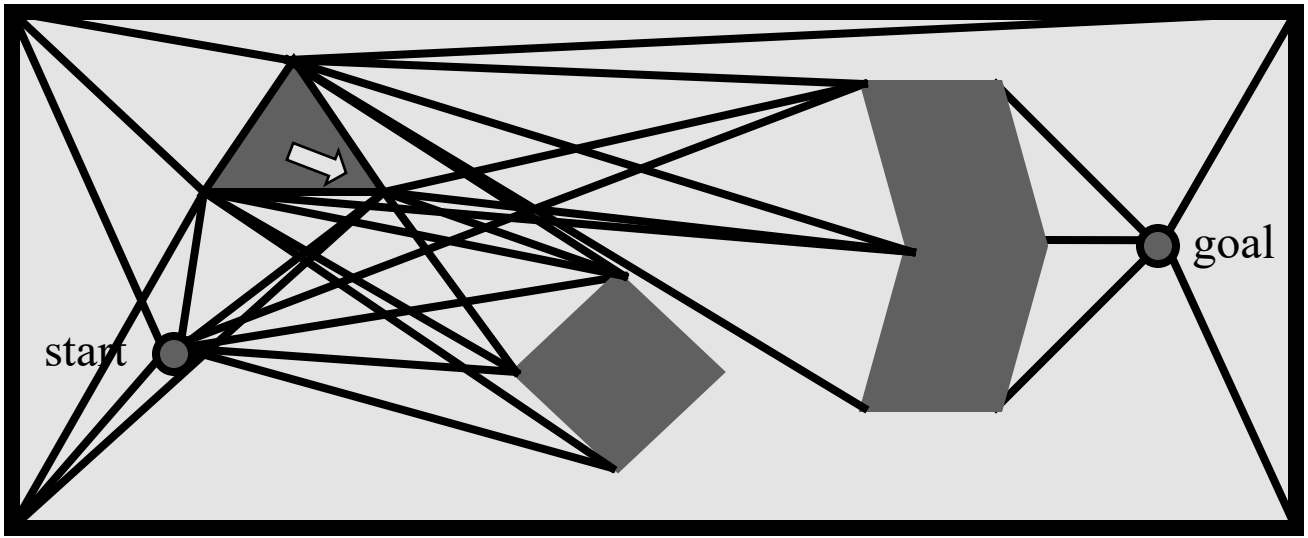
The Visibility Graph in Action (Part 3)

- Second, draw lines of sight from every vertex of every obstacle like before. Remember lines along edges are also lines of sight.



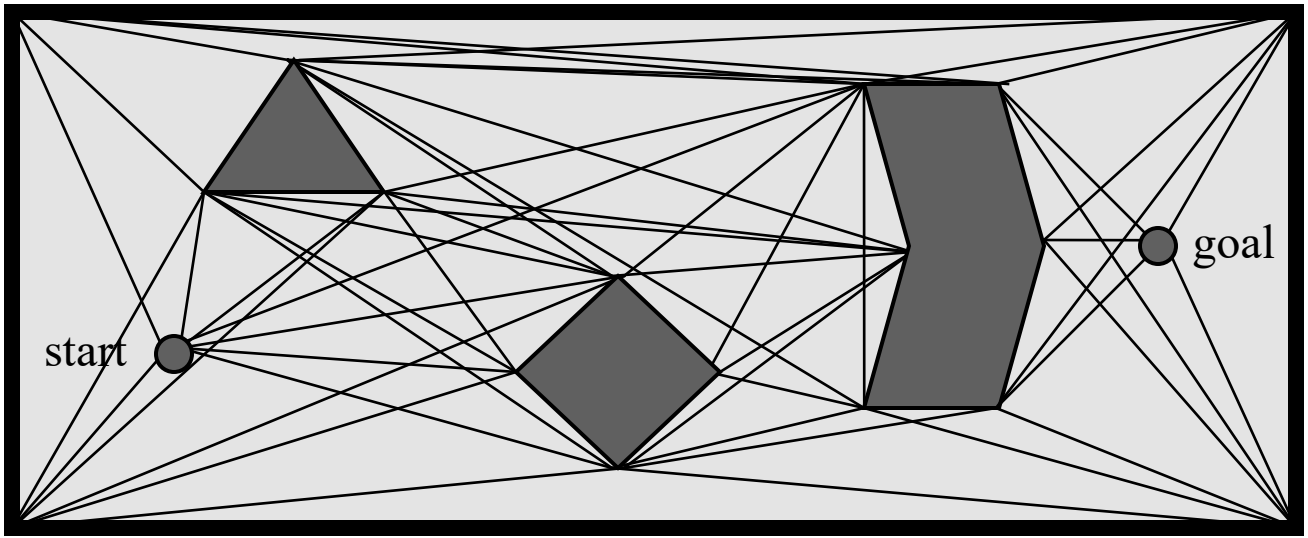
The Visibility Graph in Action (Part 4)

- Second, draw lines of sight from every vertex of every obstacle like before. Remember lines along edges are also lines of sight.

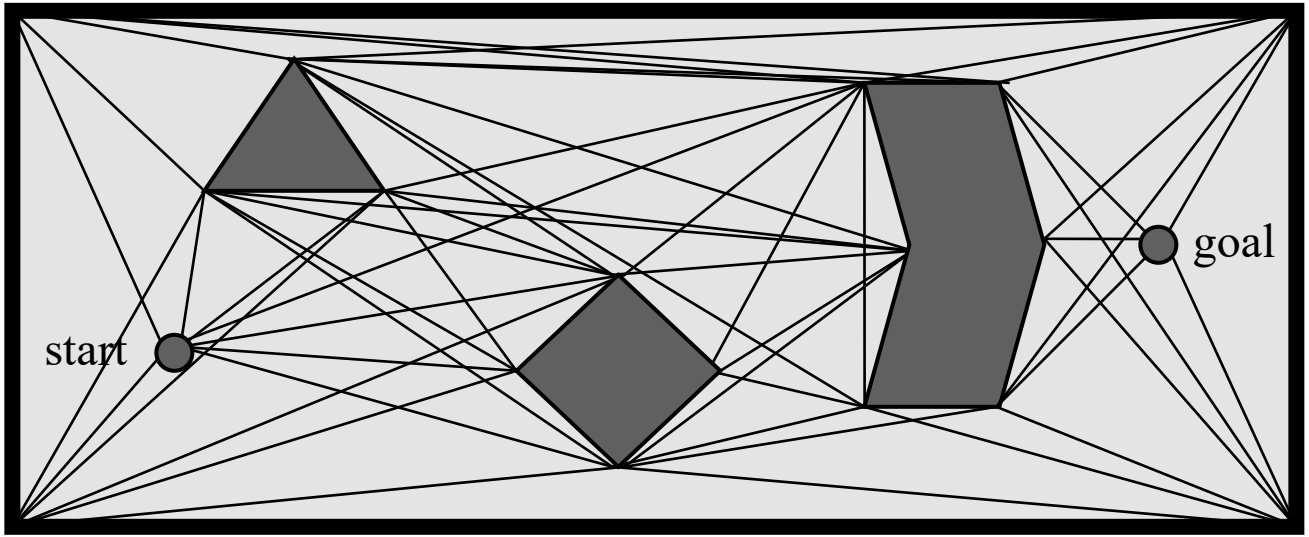


The Visibility Graph (Done)

- Repeat until you're done.



We can just run A* on this gragh



This graph can be search using A* and solves the problem continuously

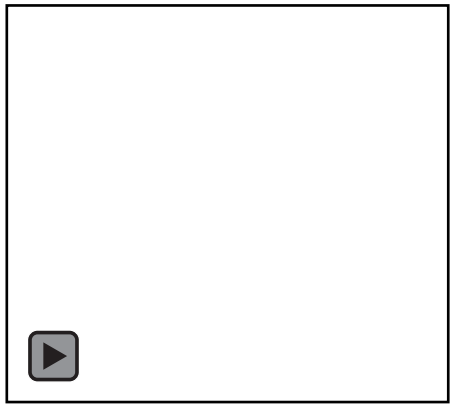
Limitations

- We achieved optimal paths in continuous spaces but only point robots so far …

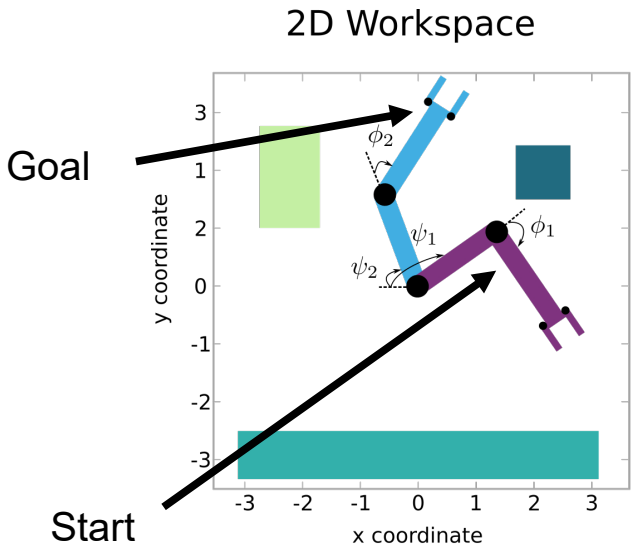
Overview

- Motion Planning Definition
- Bug Algorithms
- Discrete Search/Roadmaps
- Configuration Space
- Sampling-Based Planners
- Asymptotically Optimal Planning
- Kinodynamic Planning
- Learning And Planning
- Task and Motion Planning
- Planning Under Uncertainty

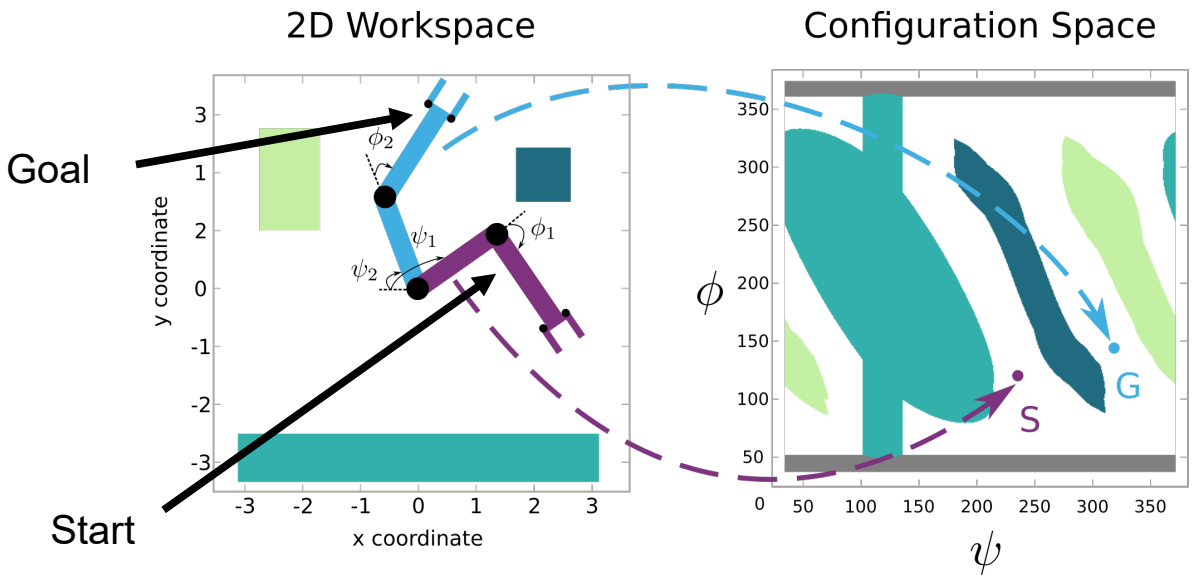
A Simple 2-link Articulated Robot



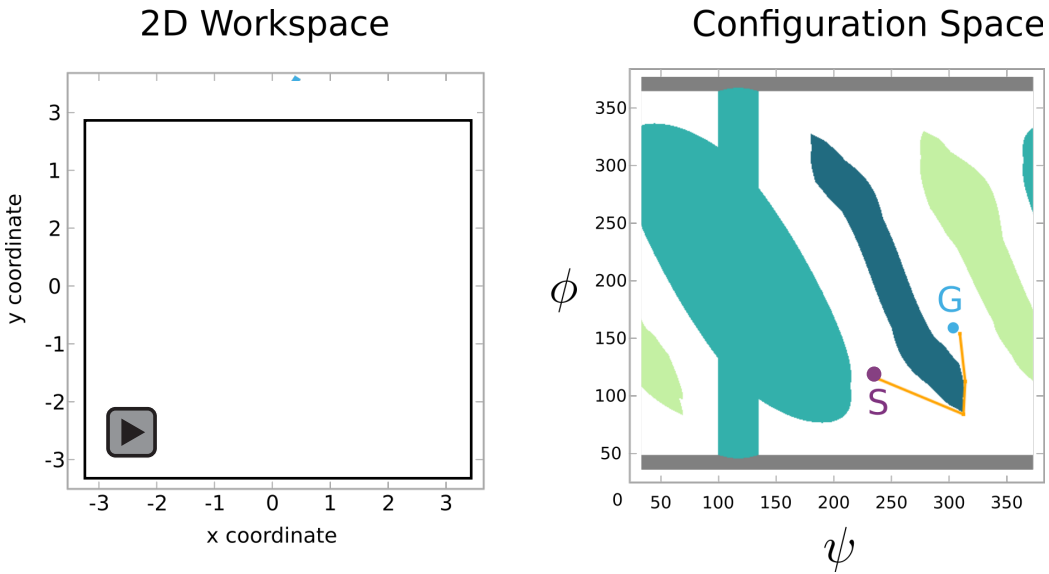
Big Idea: Configuration Space



Big Idea: Configuration Space

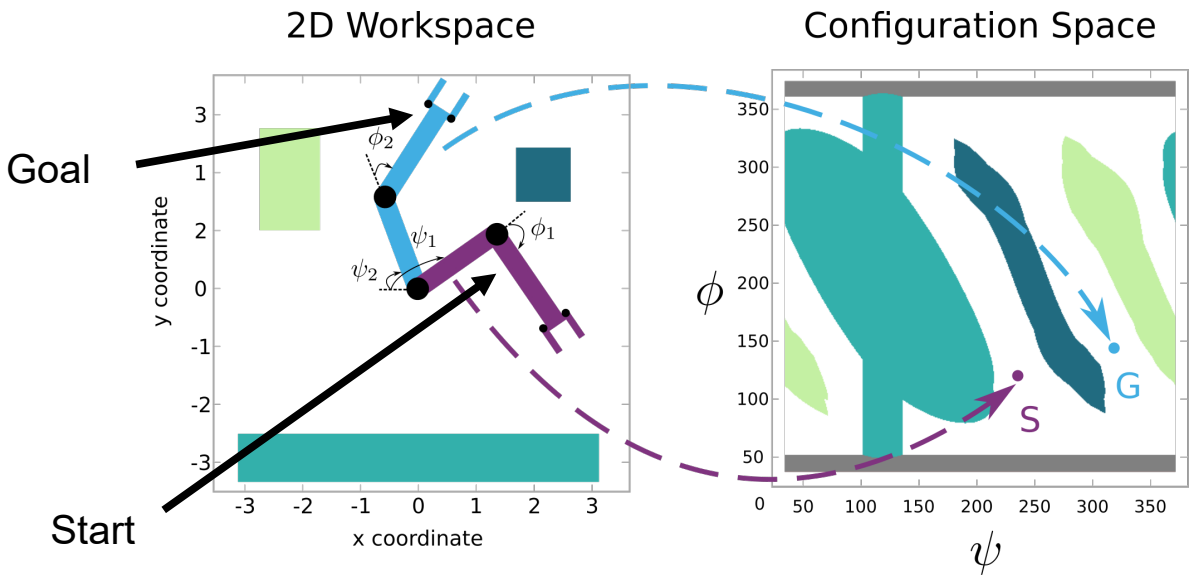


Path in Configuration space



Now we can use point Planning Algorithms to plan in the C-Space!

Workspace vs Configuration Space

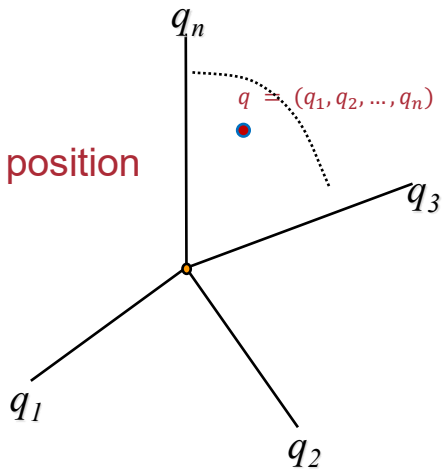


Configuration Space

- The configuration of a moving object is a specification of the position of every point on the object.

- Usually a configuration is expressed as a vector of position & orientation parameters:

$$q = (q_1, q_2, \dots, q_n)$$

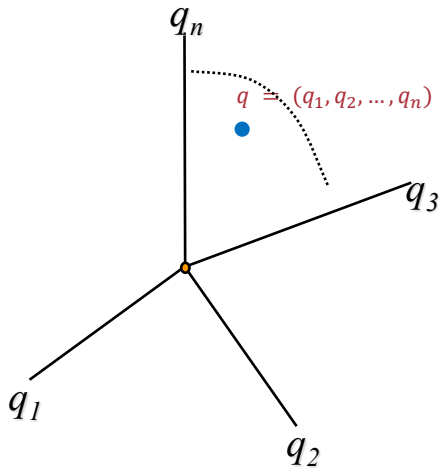


- The configuration space C is the set of all possible configurations.

- A configuration is a point in C

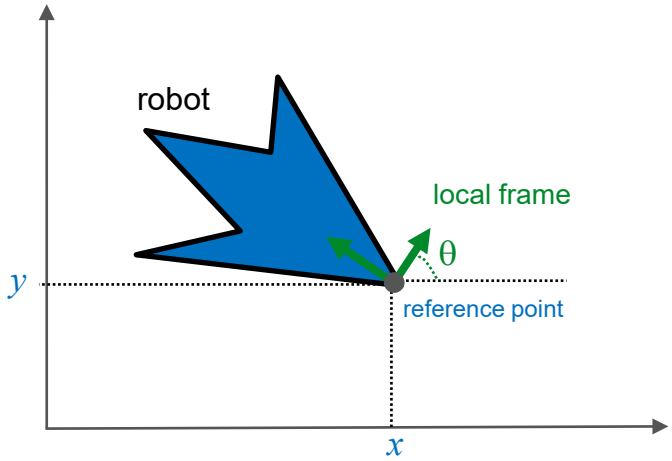
Configuration Space

- The dimension of a configuration space:
  - The minimum number of parameters needed to specify the configuration of the object completely.
  - also called the number of degrees of freedom (DoF) of a moving object.



Configuration Space

- Example: rigid body in 2-D workspace



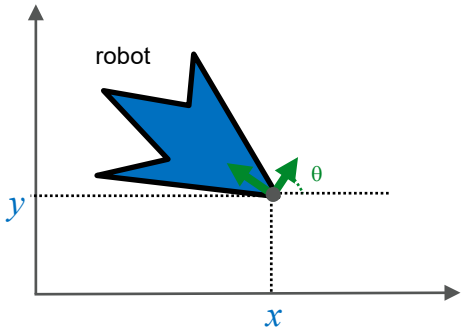
  - 3-parameter specification:

$$q = (x, y, \theta) \text{ with } \theta \in [0, 2\pi)$$

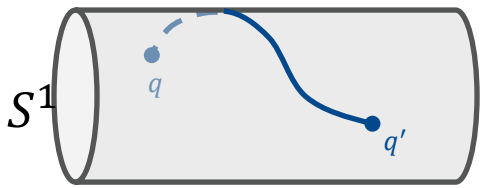
  - 3-D configuration space (3 DoF)

Topology of Configuration Space

- The topology of C is usually not that of a Cartesian space $\mathbb{R}^n$
- Here the robot **can only move in the x direction** and rotate $C = \mathbb{R} \times S^1$



3-D cylinder



$\mathbb{R} \times S^1$

(note this is not $\mathbb{R}^2 \times S^1$)

Configuration Space

- Example: rigid body in 3-D
  - How many parameters in C-space?

$$q = (x, y, z, q')$$

  - Parametrization of orientations by matrix:

$$q' = (r_{11}, r_{12}, \ldots, r_{32}, r_{33})$$

where $r_{11}, r_{12}, \ldots, r_{33}$ are the elements of rotation matrix

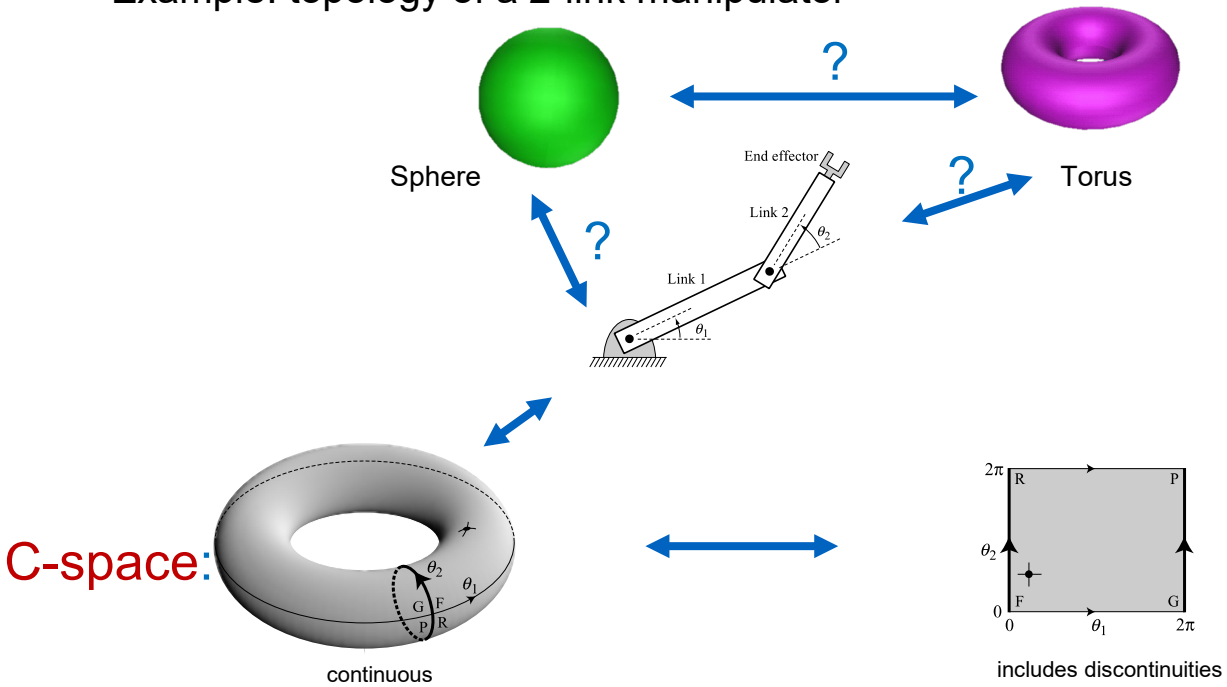
$$R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$$

with

- $r_{1i}^2 + r_{2i}^2 + r_{3i}^2 = 1$ for all i
- $r_{1i} r_{1j} + r_{2i} r_{2j} + r_{3i} r_{3j} = 0$ for all $i \neq j$
- $\det(R) = +1$

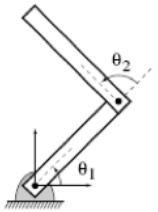
2-link manipulator robot Topology

- Example: topology of a 2-link manipulator



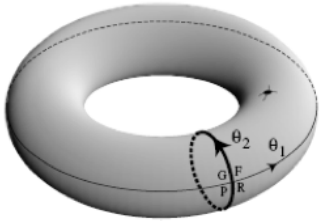
- Topology is the intrinsic character of the space

Articulated Robot

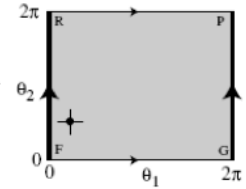


(a)

$$T = S^1 \times S^1$$



(b)



(c)

$(\theta_1, \theta_2) \in \mathbb{R}^2$,

problems at $\theta_i = \{0, 2\pi\}$.

Topology of Some C-Space

- Examples of some common robots

Type of robot	Representation of C-Space
Mobile robot translating in the plane	$\mathbb{R}^2$
Mobile robot translating and rotating in the plane	$SE(2)$ or $\mathbb{R}^2 \times S^1$
Rigid body translating in 3-D	$\mathbb{R}^3$
A spacecraft	$SE(3)$ or $\mathbb{R}^3 \times SO(3)$
An n-joint revolute arm	T^n
A planar mobile robot with an attached n-joint arm	$SE(2) \times T^n$

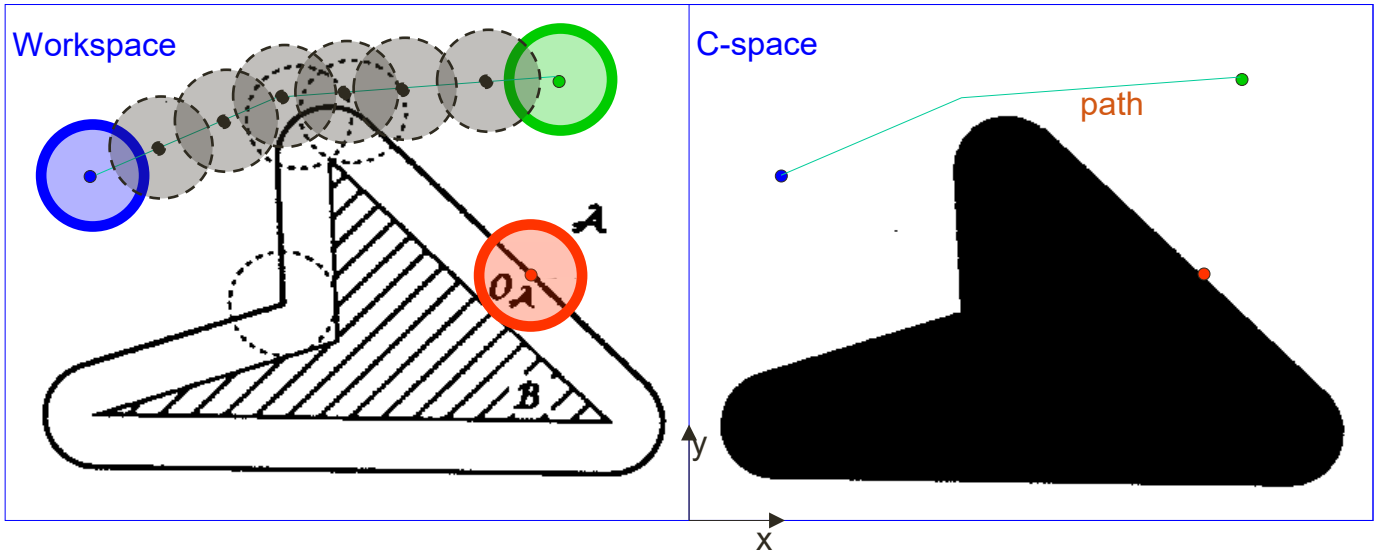
- Note that:
- $S^1 \times S^1 \times \cdots \times S^1 = T^n$, n-dimensional torus
- $S^1 \times S^1 \times \cdots \times S^1 \neq S^n$, n-dimensional sphere
- $S^1 \times S^1 \times S^1 \neq SO(3)$
- $SE(2) \neq \mathbb{R}^3$
- $SE(3) \neq \mathbb{R}^6$

Limitations

- Now we can use point algorithms to move any robot if we know the topology of its configuration space
- But what about avoiding obstacles?

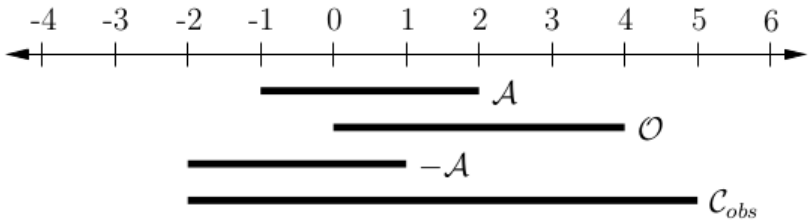
Obstacles in the Configuration Space

- 2-D disc robot (translation only)



- Configuration: $q = (x, y)$ coordinates of robot's center
- configuration space $C = \mathbb{R}^2$
- free space C_{free} = the set of collision-free configurations

C-Space Obstacle in a 1D Case



$A = [-1, 2]$ $O = [0\,, 4]$

$-A = [-2, 1]$

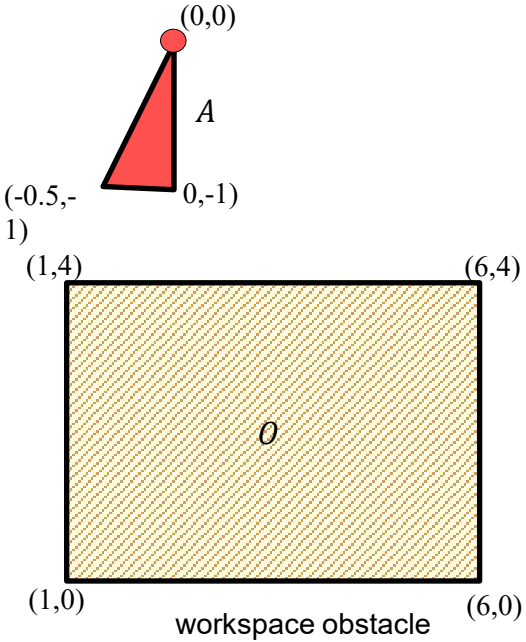
$C = O \ominus A = O + (-A) =$ $[-2, 5]$

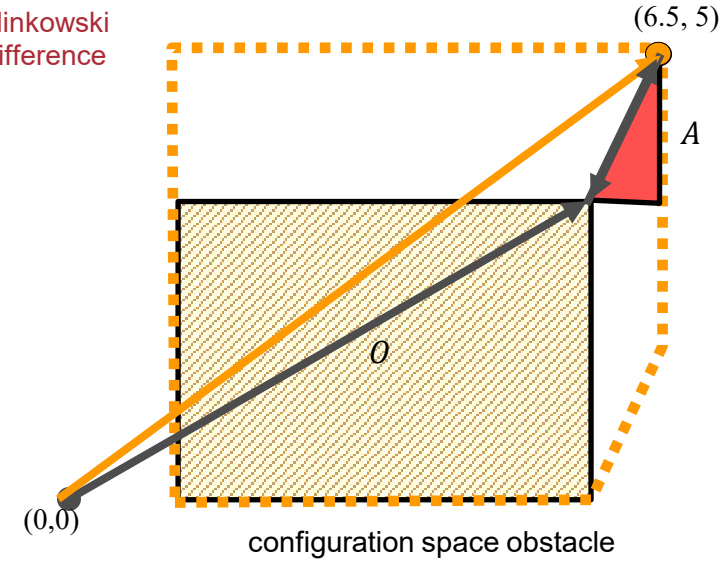
C-Space obstacles

- If O is an obstacle in the workspace and A is a moving object, then the C-space obstacle corresponding to O is

$$O \ominus A$$

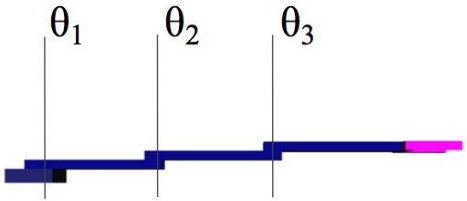
Minkowski difference





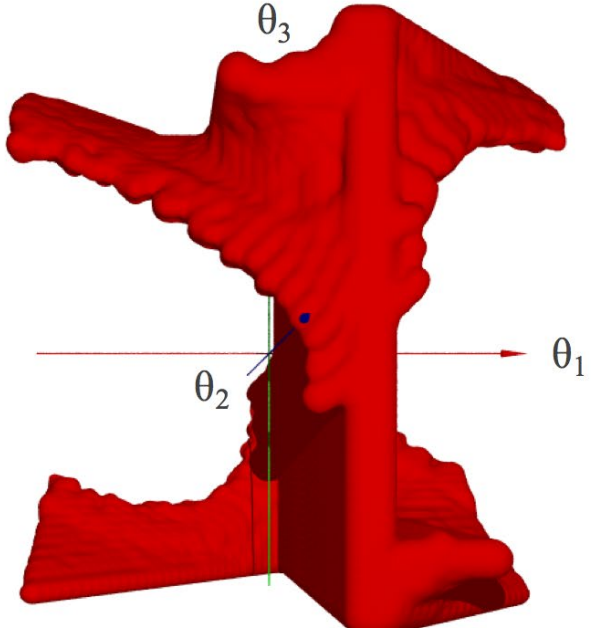
Articulated Robot in 2D workspace

- 3-link arm



side view

top view

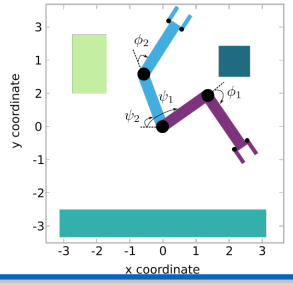


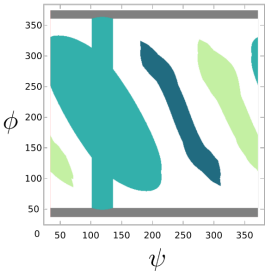
High-Dof Robots configuration space

Workspace

Configuration Space

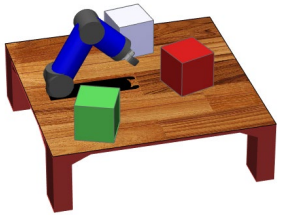
2-Dof Robot



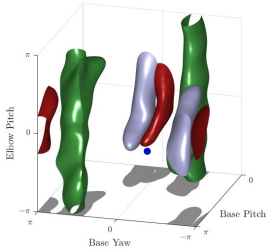


2 Dimensions

3-Dof Robot

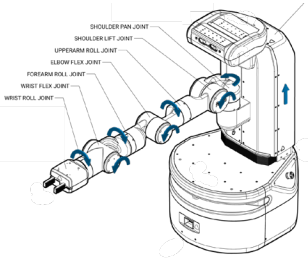


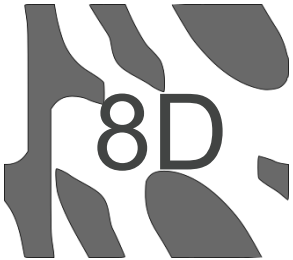
Das et al, *TRO* 2020



3 Dimensions

8-Dof Robot





8 Dimensions

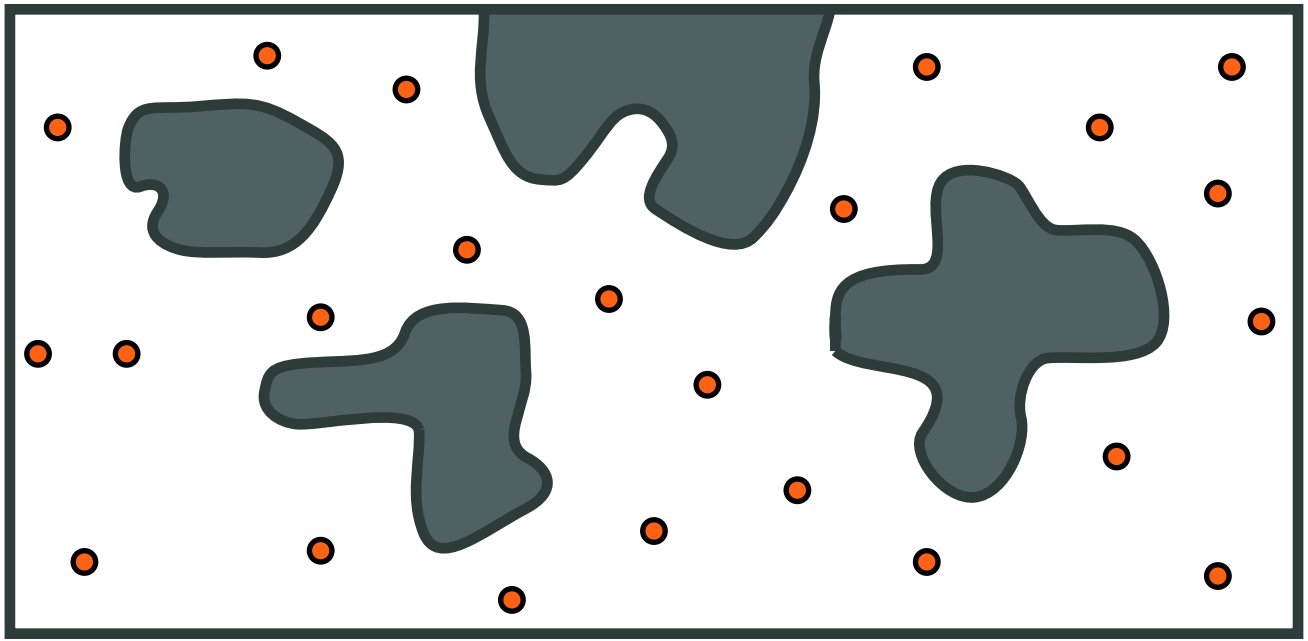
Limitations

- How can we create this high-dimensional configuration obstacles?

Overview

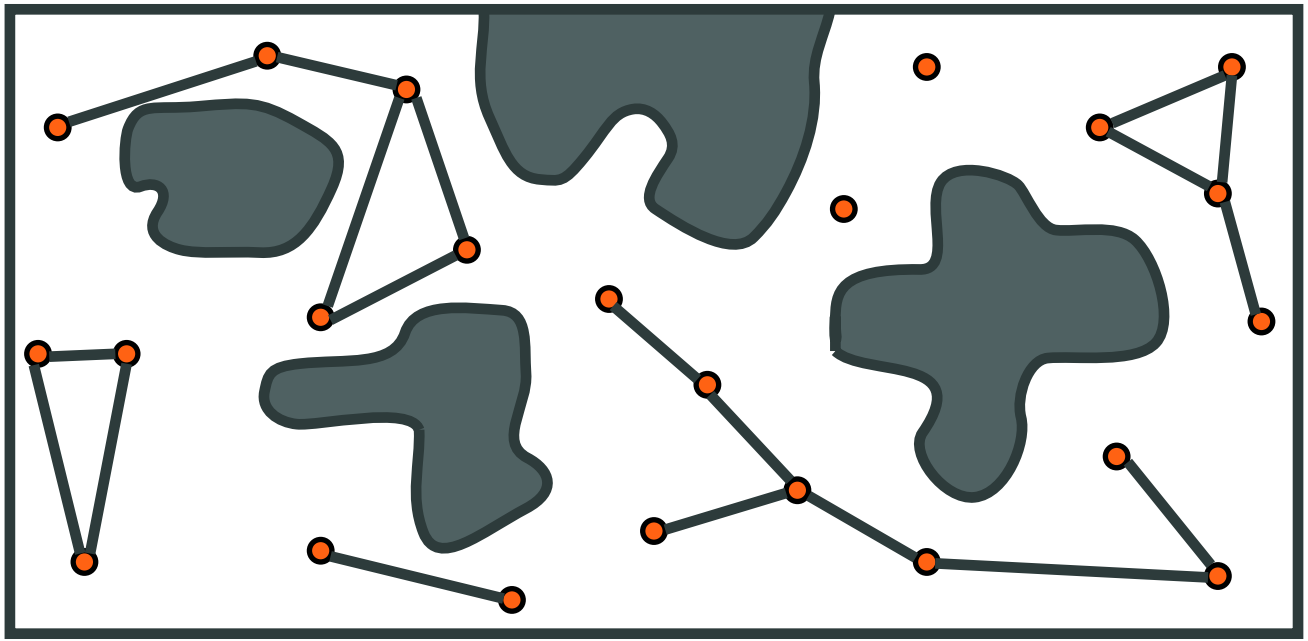
- Motion Planning Definition
- Bug Algorithms
- Discrete Search/Roadmaps
- Configuration Space
- Sampling-Based Planners
- Asymptotically Optimal Planning
- Kinodynamic Planning
- Learning And Planning
- Task and Motion Planning
- Planning Under Uncertainty

Random Sampling!



: **nodes,** random states

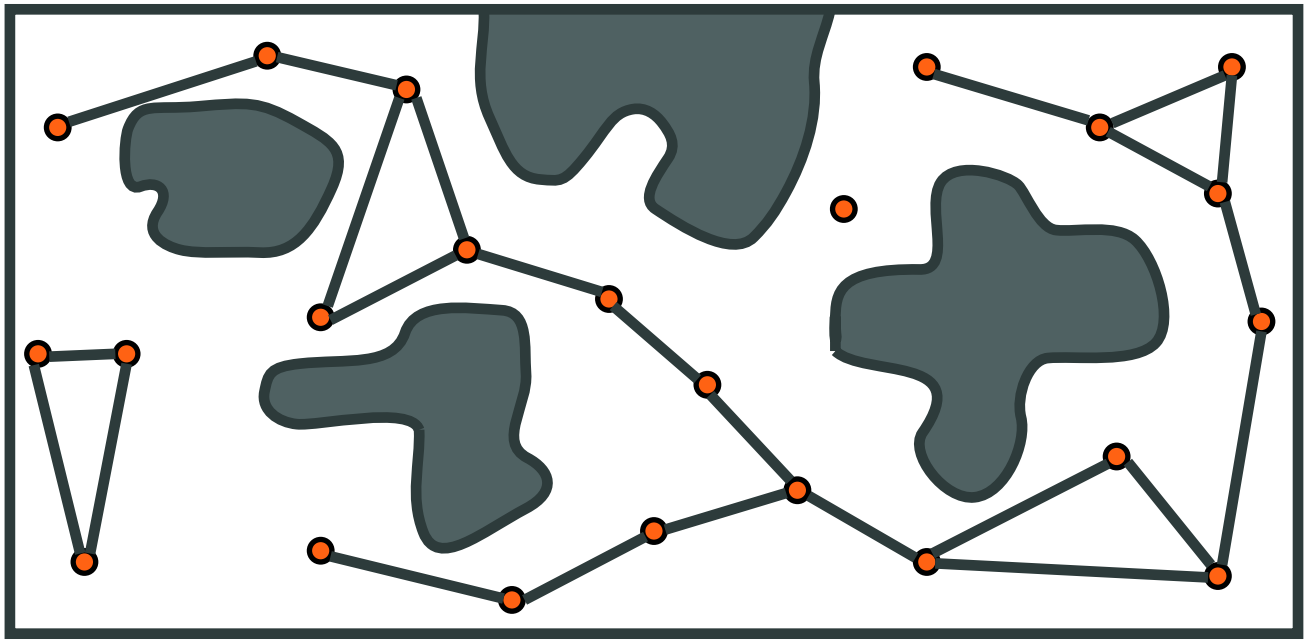
Operation of PRM



: **nodes,** random states

:edges, paths computed by local planner

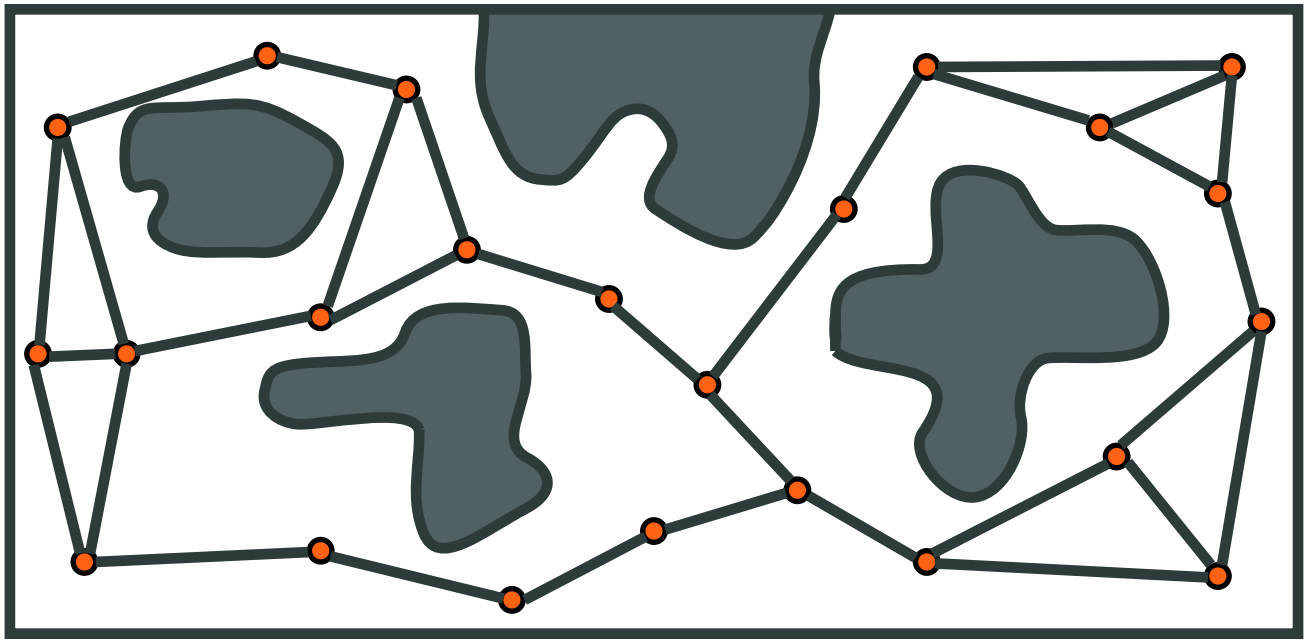
Operation of PRM



: **nodes,** random states

:edges, paths computed by local planner

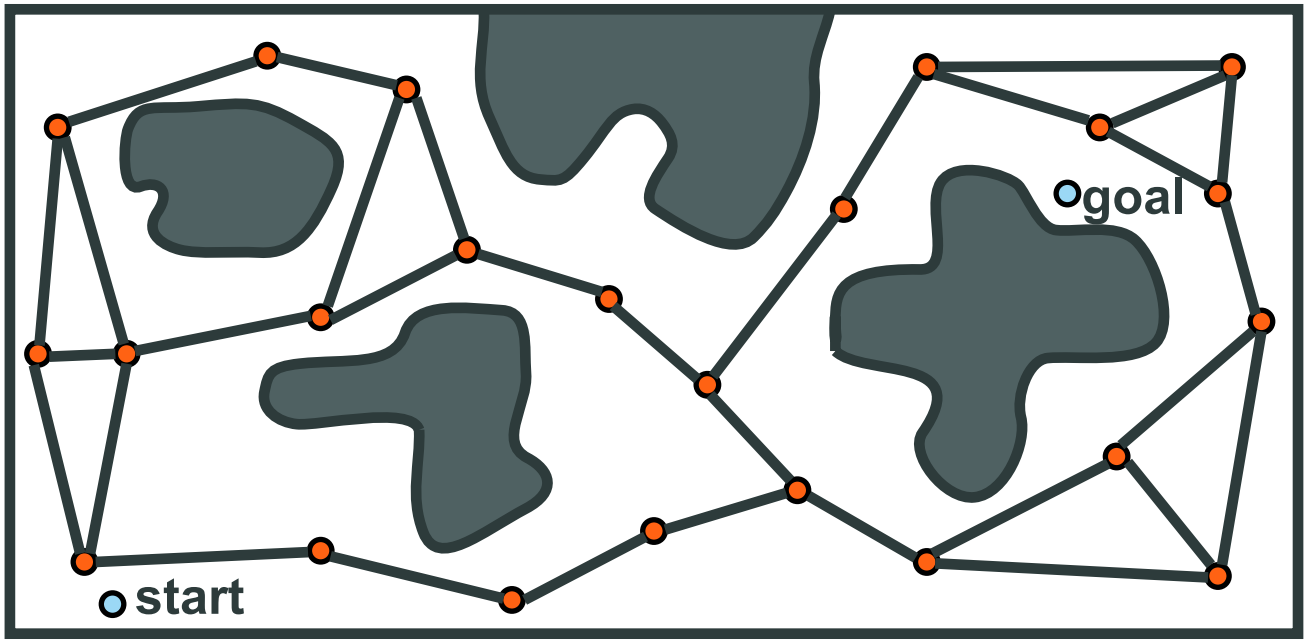
Operation of PRM



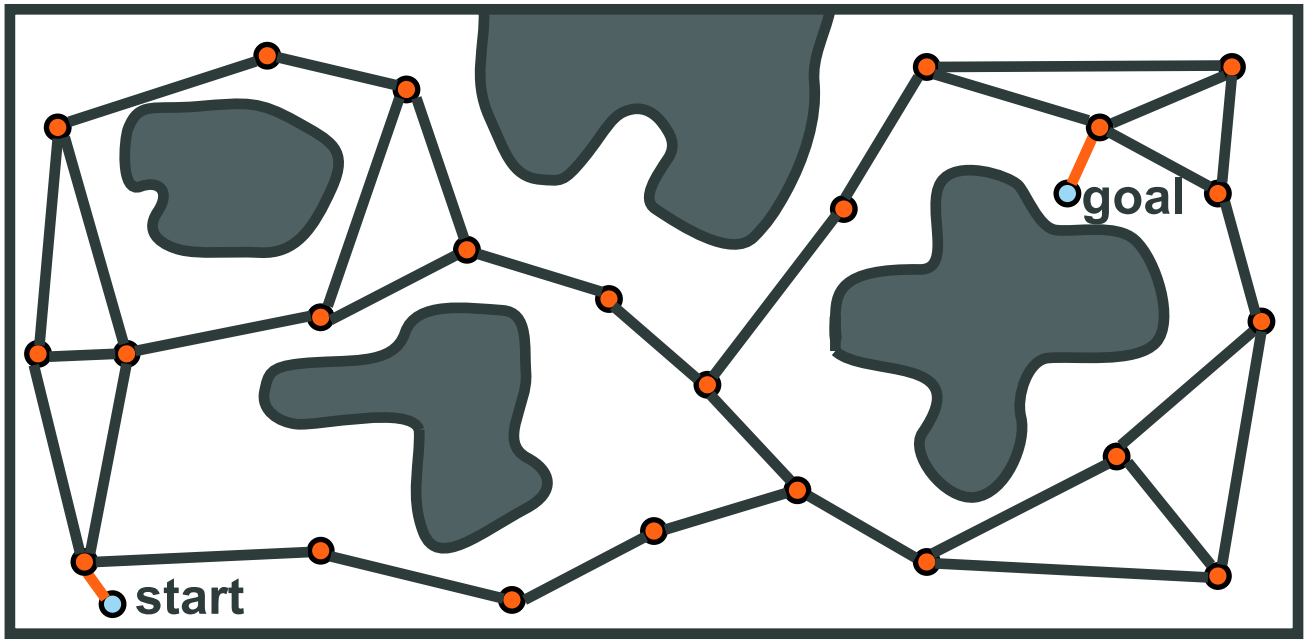
: **nodes,** random states

:edges, paths computed by local planner

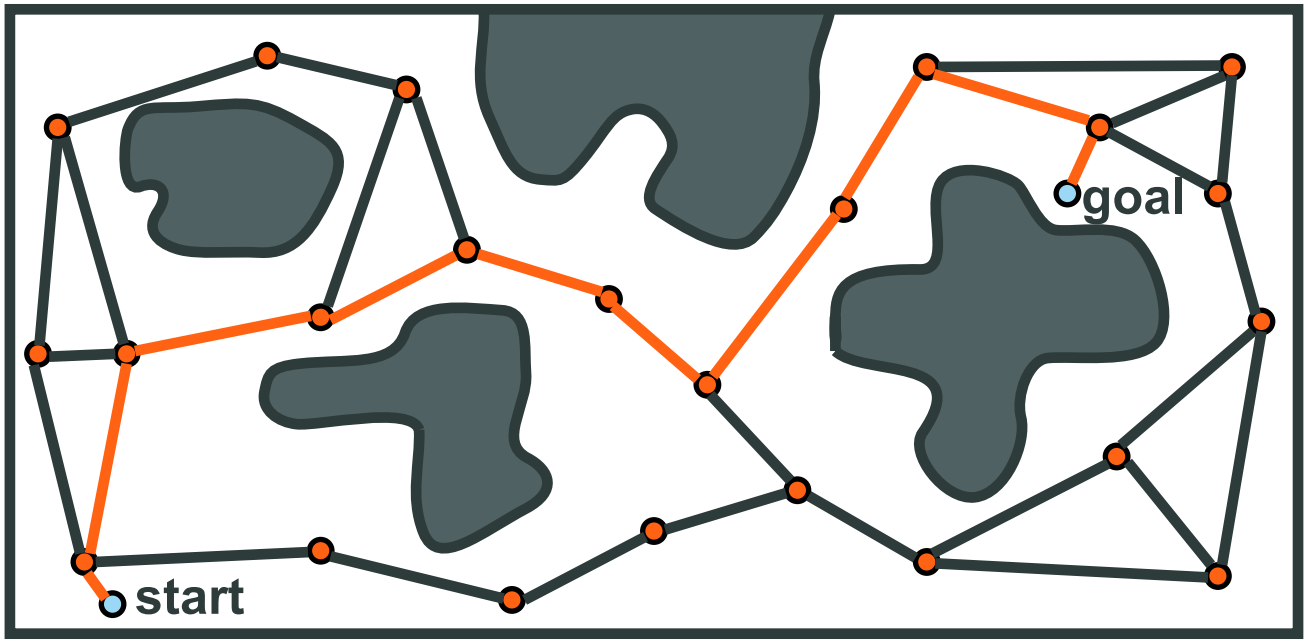
Answering a Query with PRM



Answering a Query with PRM



Answering a Query with PRM



Probabilistic Completeness

- Tradeoff between efficiency and completeness

- **Probabilistic completeness definition:**
  - Suppose $a, b \in Q_{free}$ can be connected by a path in Q_{free}. A planner is probabilistically complete if

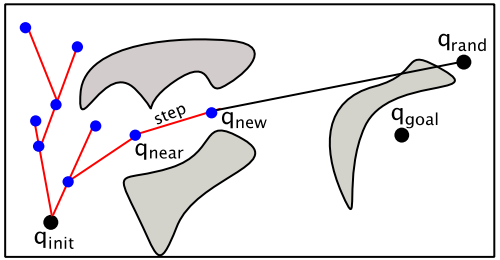
$$\lim_{N \to \infty} \Pr\big((a,b)\ \ Failure\big) = 0$$

Probability that PRM fails to answer query (a, b) with N samples

What if we only have a single-Query?

Rapidly-exploring Random Tree (RRT)

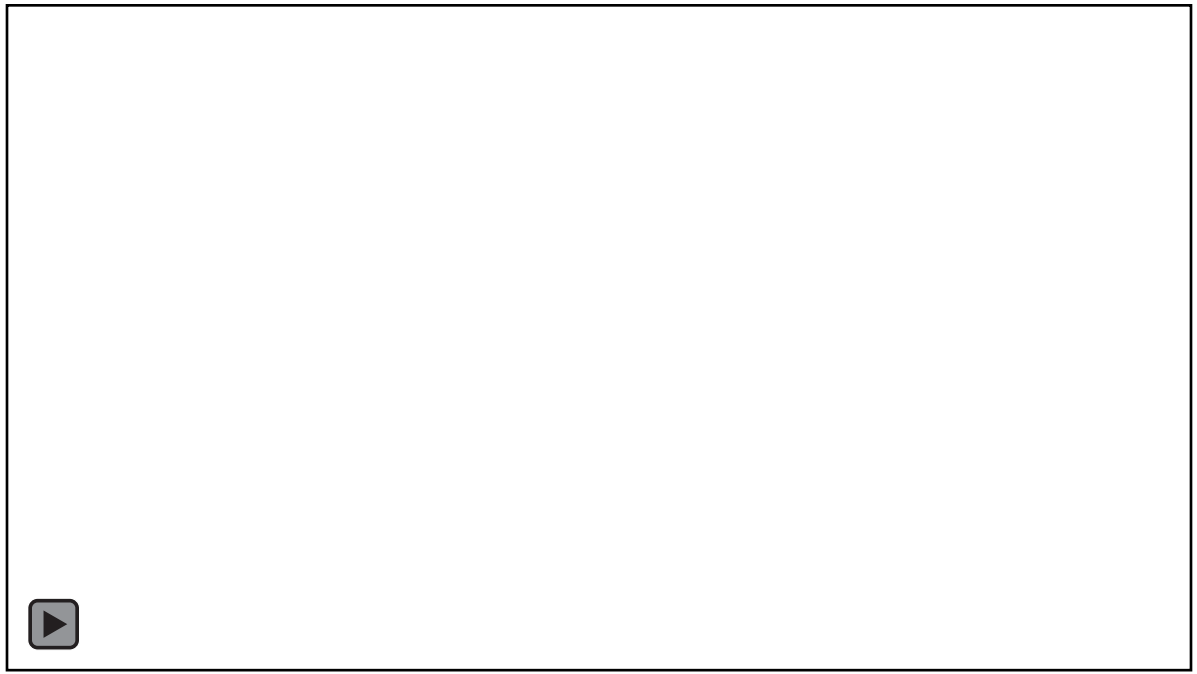
- **Idea:** Pull the tree toward random samples in the configuration space



RRT (q_{init}, q_{goal})
//initialize tree
1: $T \leftarrow$ create tree rooted at q_{init}
2: **while** solution not found **do**
// select configuration from tree
3: $q_{rand} \leftarrow$ generate a random sample
4: $q_{near} \leftarrow$ nearest configuration in T to q_{rand} according to distance ρ
// add new branch to tree from selected configuration
5: $path \leftarrow$ generate path (not necessarily collision free) from q_{near} to q_{rand}
6: **if** IsSubpathCollisionFree$(path, 0, step)$ **then**
7: $q_{new} \leftarrow path(step)$
8: add configuration q_{new} and edge (q_{near}, q_{new}) to T
//check if a solution is found
9: **if** $\rho(q_{new}, q_{goal}) \approx 0$ **then**
10: **return** solution path from root to q_{new}

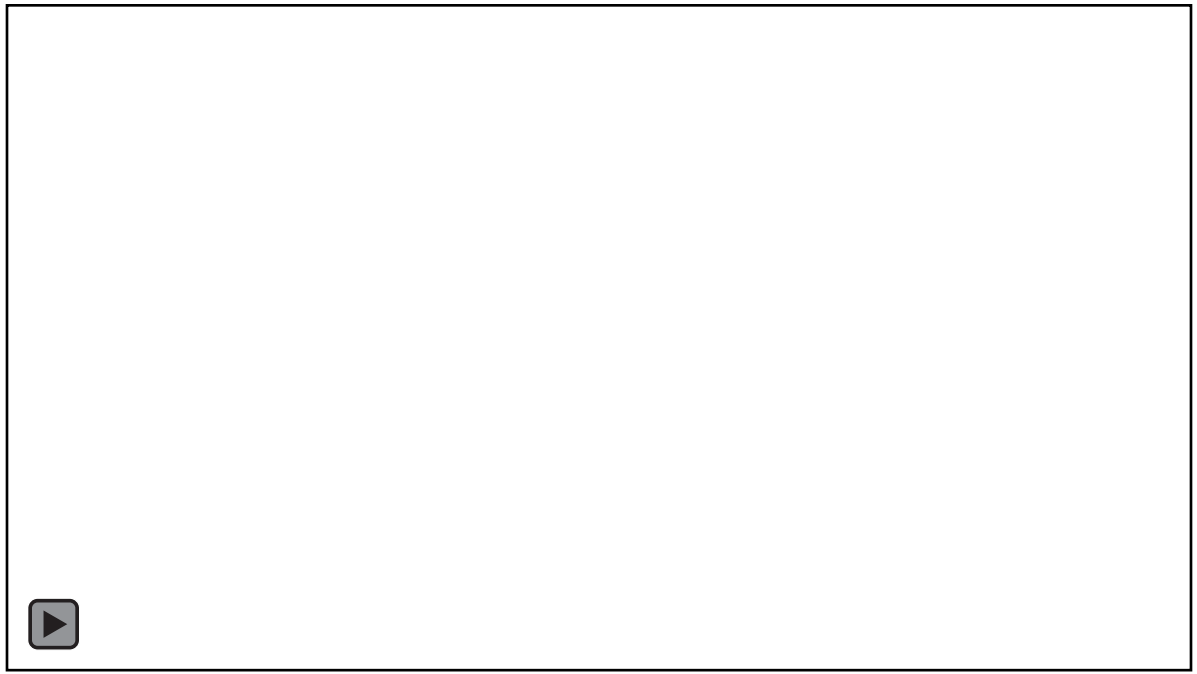
- RRT relies on nearest neighbors and distance metric $\rho : Q \times Q \leftarrow \mathbb{R}^{\geq 0}$
- RRT adds Voronoi bias to tree growth space

RRT-Expansion Step



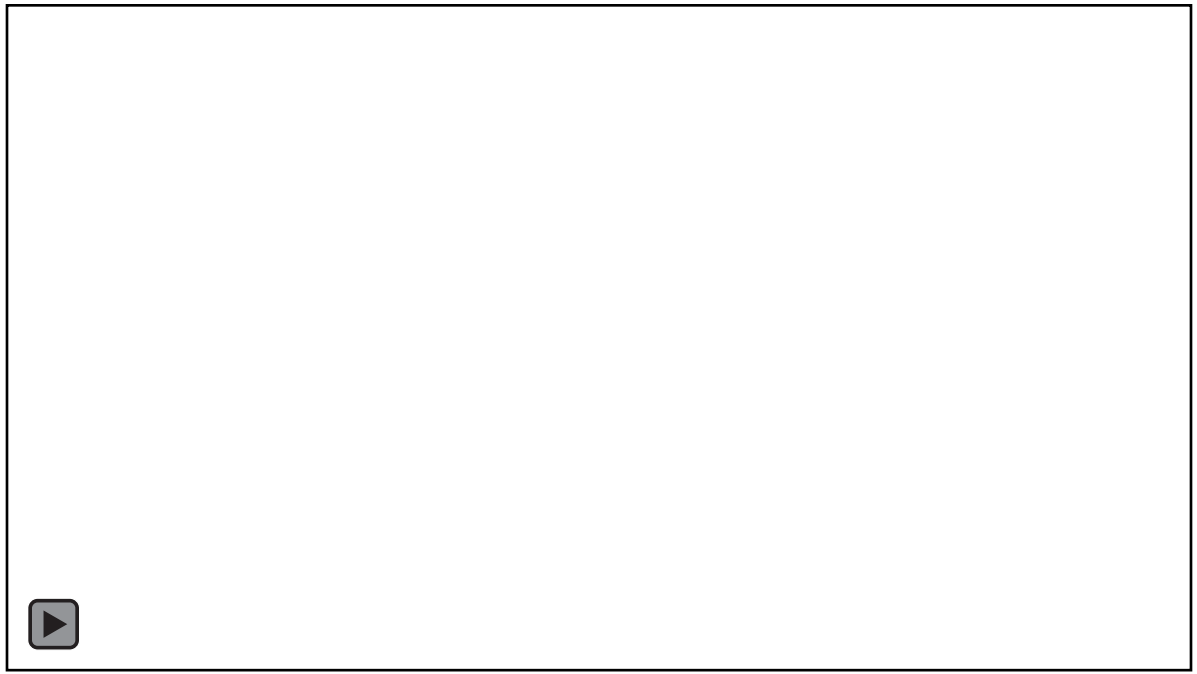
Video Credit: **Peter Murray**

RRT- Selecting node to extend



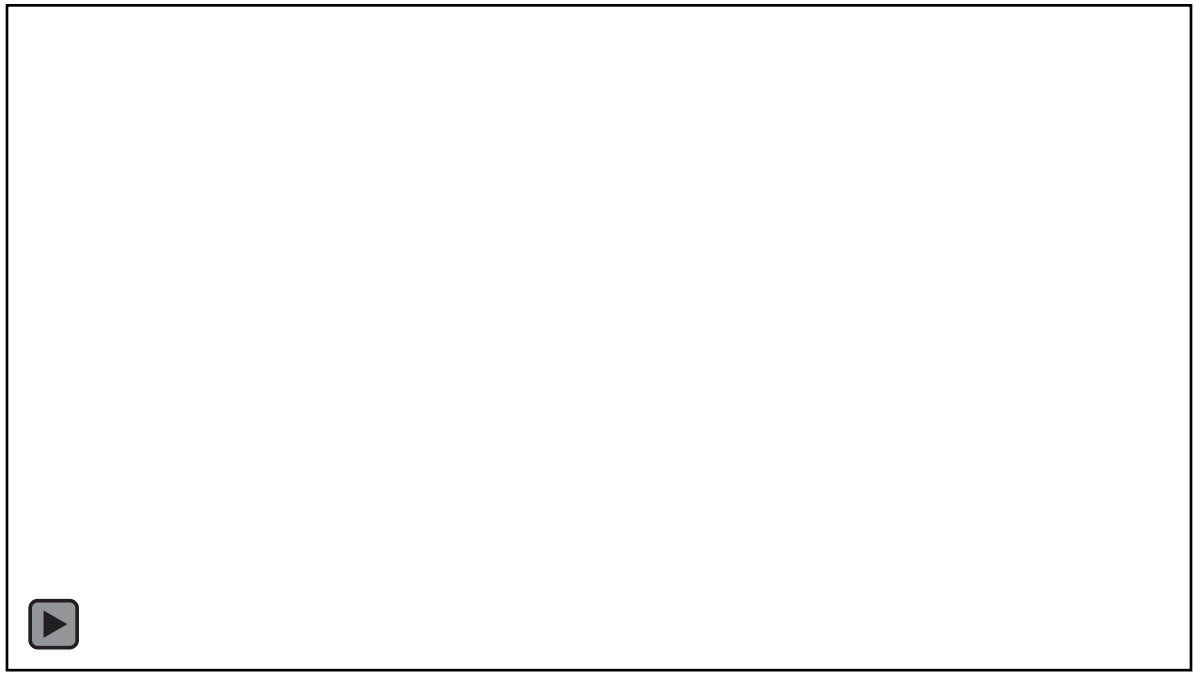
Video Credit: **Peter Murray**

RRT- Growth Illustration



Video Credit: **Peter Murray**

RRT- Voronoi -Bias



Video Credit: **Peter Murray**

Goal-Bias RRT-Improvements

Aspects for Improvement

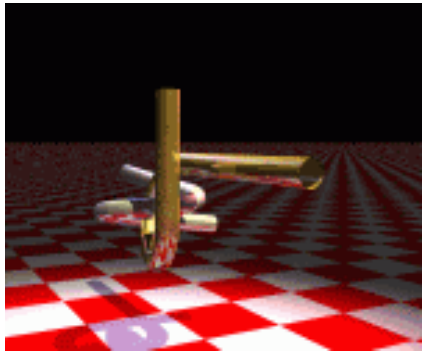
- $\mathrm{BasicRRT}$ does not take advantage of q_{goal}
- Tree is pulled towards random directions based on the uniform sampling of Q
- In particular, tree growth is not directed towards q_{goal}

Suggested Improvements in the Literature

- Introduce goal-bias to tree growth (known as $\mathrm{GoalBiasRRT}$)
- q_{rand} is selected as q_{goal} with probability p
- q_{rand} is selected based on uniform sampling of Q with probability $1 - p$
- Probability p is commonly set to ≈ 0.05

RRT Examples: The Alpha Puzzle

- VERY hard 6DOF motion planning problem (long, winding narrow passage)
- "*In 2001, it was solved by using a balanced bidirectional RRT, developed by James Kuffner and Steve LaValle. There are no special heuristics or parameters that were tuned specifically for this problem. On a current PC (circa 2003), it consistently takes a few minutes to solve*" –RRT website
- RRT became famous in large part because it was able to solve this puzzle



RRT-Connect

```
BUILD_RRT(q_init)
1  T.init(q_init);
2  for k = 1 to K do
3      q_rand ← RANDOM_CONFIG();
4      EXTEND(T, q_rand);
5  Return T
```

```
EXTEND(T, q)
1  q_near ← NEAREST_NEIGHBOR(q, T);
2  if NEW_CONFIG(q, q_near, q_new) then
3      T.add_vertex(q_new);
4      T.add_edge(q_near, q_new);
5      if q_new = q then
6          Return Reached;
7      else
8          Return Advanced;
9  Return Trapped;
```

Figure 2: The basic RRT construction algorithm.

```
CONNECT(T, q)
1  repeat
2      S ← EXTEND(T, q);
3  until not (S = Advanced)
4  Return S;
```

```
RRT_CONNECT_PLANNER(q_init, q_goal)
1  T_a.init(q_init); T_b.init(q_goal);
2  for k = 1 to K do
3      q_rand ← RANDOM_CONFIG();
4      if not (EXTEND(T_a, q_rand) =Trapped) then
5          if (CONNECT(T_b, q_new) =Reached) then
6              Return PATH(T_a, T_b);
7      SWAP(T_a, T_b);
8  Return Failure
```

Figure 5: The RRT-Connect algorithm.

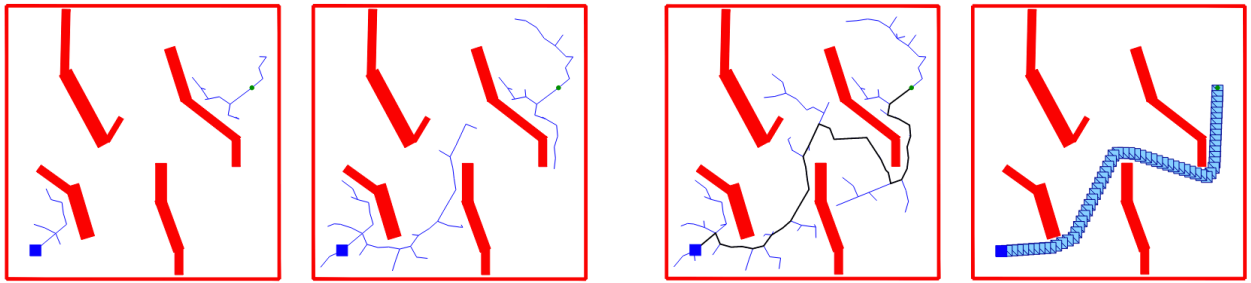


Figure 6: Growing two trees towards each other.

Expansive-Space Tree (EST)

- **Idea:** Push the tree frontier in the free configuration space

- EST relies on a probability distribution to guide tree growth
- EST associates a weight $w(q)$ with each tree configuration q
- $w(q)$ is a running estimate on importance of selecting q as the tree configuration from which to add a new tree branch
  - $w(q) = \frac{1}{1+\deg(q)}$
  - $w(q)$ = 1/(1+ number of neighbors near q)
  - combination of different strategies

What if the problem is challenging?

- Biased Sampling,
  - Bridge Sampling
  - Gaussian Sampling
  - Importance Sampling

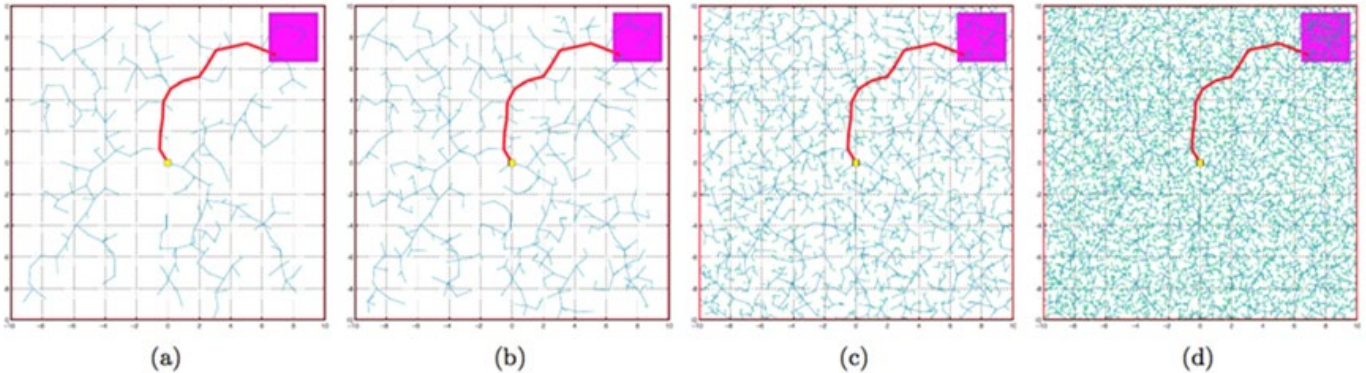
Limitations

- But what about the optimal solution?

Overview

- Motion Planning Definition
- Bug Algorithms
- Discrete Search/Roadmaps
- Configuration Space
- Sampling-Based Planners
- Asymptotically Optimal Planning
- Kinodynamic Planning
- Learning And Planning
- Task and Motion Planning
- Planning Under Uncertainty

More samples do not improve the solution



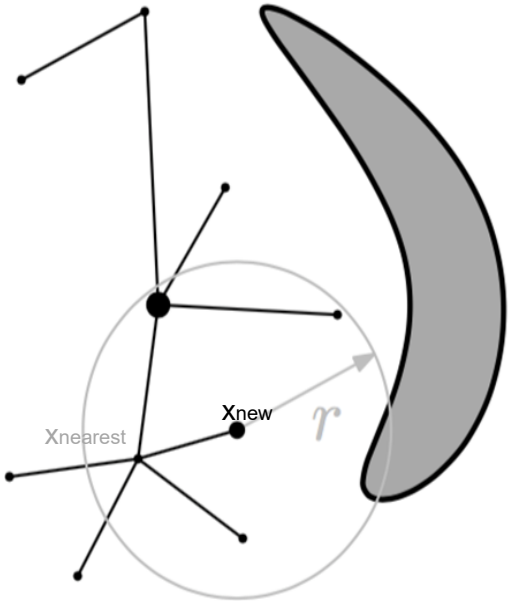
RRT* Extend

ExtendRRT*(G,x):

$V' \leftarrow V$; $E' \leftarrow E$;
$x_{\text{nearest}} \leftarrow \texttt{Nearest}(G, x)$;
$x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x)$;
if $\texttt{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
  $V' \leftarrow V' \cup \{x_{\text{new}}\}$;
  $x_{\min} \leftarrow x_{\text{nearest}}$;
  $X_{\text{near}} \leftarrow \texttt{Near}(G, x_{\text{new}}, |V|)$;
  for *all* $x_{\text{near}} \in X_{\text{near}}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{near}}, x_{\text{new}})$ **then**
      $c' \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$;
      if $c' < \texttt{Cost}(x_{\text{new}})$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$;
  $E' \leftarrow E' \cup \{(x_{\min}, x_{\text{new}})\}$;
  for *all* $x_{near} \in X_{\text{near}} \setminus \{x_{\min}\}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{new}}, x_{\text{near}})$ *and*
    $\texttt{Cost}(x_{\text{near}}) >$
    $\texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}}))$ **then**
      $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E' \leftarrow E' \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$;
      $E' \leftarrow E' \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
return $G' = (V', E')$

Karaman, Frazzoli. 2010.

Graphic from Gammell, 2016. Thesis.



RRT* Extend

define $\texttt{Near}(G, x, n)$ to be the set of all vertices within the closed ball of radius r_n centered at x, where

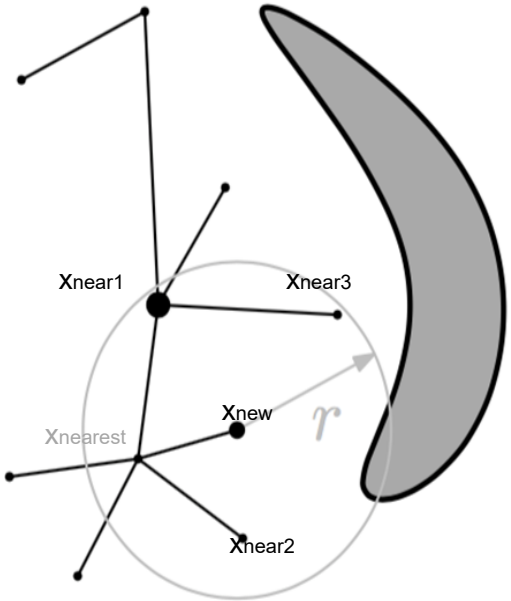
$$r_n = \min\left\{\left(\frac{\gamma}{\zeta_d}\frac{\log n}{n}\right)^{1/d}, \eta\right\},$$

ExtendRRT*(G,x):

$V' \leftarrow V$; $E' \leftarrow E$;
$x_{\text{nearest}} \leftarrow \texttt{Nearest}(G, x)$;
$x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x)$;
if $\texttt{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
  $V' \leftarrow V' \cup \{x_{\text{new}}\}$;
  $x_{\min} \leftarrow x_{\text{nearest}}$;
  $X_{\text{near}} \leftarrow \texttt{Near}(G, x_{\text{new}}, |V|)$;
  for *all* $x_{\text{near}} \in X_{\text{near}}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{near}}, x_{\text{new}})$ **then**
      $c' \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$;
      if $c' < \texttt{Cost}(x_{\text{new}})$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$;
  $E' \leftarrow E' \cup \{(x_{\min}, x_{\text{new}})\}$;
  for *all* $x_{near} \in X_{\text{near}} \setminus \{x_{\min}\}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{new}}, x_{\text{near}})$ *and*
    $\texttt{Cost}(x_{\text{near}}) >$
    $\texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}}))$ **then**
      $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E' \leftarrow E' \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$;
      $E' \leftarrow E' \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
return $G' = (V', E')$

Karaman, Frazzoli. 2010.

Graphic from Gammell, 2016. Thesis.



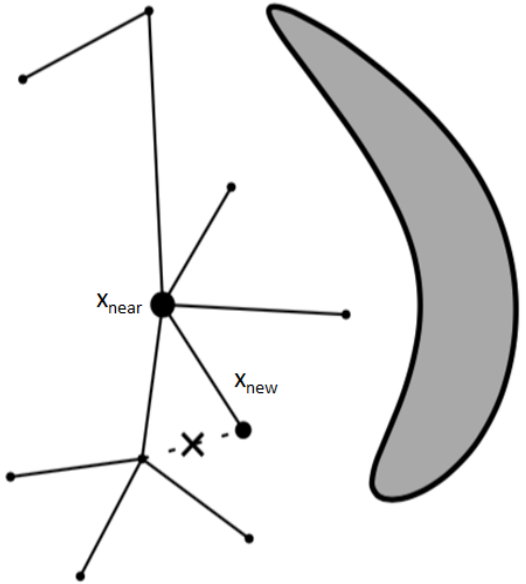
RRT* Extend

ExtendRRT*(G,x):

$V' \leftarrow V$; $E' \leftarrow E$;
$x_{\text{nearest}} \leftarrow \texttt{Nearest}(G, x)$;
$x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x)$;
if $\texttt{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
  $V' \leftarrow V' \cup \{x_{\text{new}}\}$;
  $x_{\min} \leftarrow x_{\text{nearest}}$;
  $X_{\text{near}} \leftarrow \texttt{Near}(G, x_{\text{new}}, |V|)$;
  for *all* $x_{\text{near}} \in X_{\text{near}}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{near}}, x_{\text{new}})$ **then**
      $c' \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$;
      if $c' < \texttt{Cost}(x_{\text{new}})$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$;
  $E' \leftarrow E' \cup \{(x_{\min}, x_{\text{new}})\}$;
  for *all* $x_{near} \in X_{\text{near}} \setminus \{x_{\min}\}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{new}}, x_{\text{near}})$ *and*
    $\texttt{Cost}(x_{\text{near}}) >$
    $\texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}}))$ **then**
      $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E' \leftarrow E' \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$;
      $E' \leftarrow E' \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
return $G' = (V', E')$

Karaman, Frazzoli. 2010.

Graphic from Gammell, 2016. Thesis.

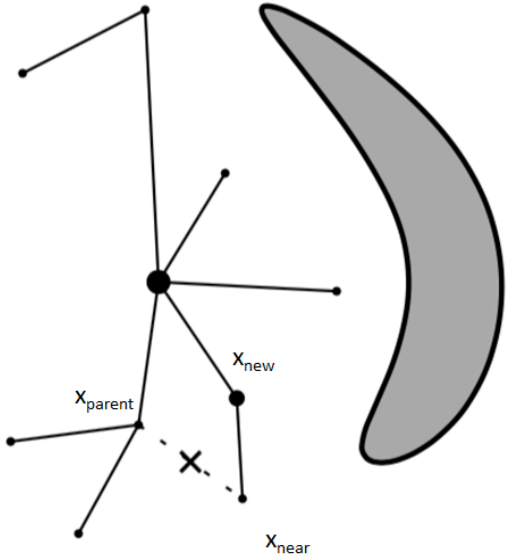


RRT* Extend

$V' \leftarrow V$; $E' \leftarrow E$;
$x_{\text{nearest}} \leftarrow \texttt{Nearest}(G, x)$;
$x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x)$;
if $\texttt{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
  $V' \leftarrow V' \cup \{x_{\text{new}}\}$;
  $x_{\min} \leftarrow x_{\text{nearest}}$;
  $X_{\text{near}} \leftarrow \texttt{Near}(G, x_{\text{new}}, |V|)$;
  for *all* $x_{\text{near}} \in X_{\text{near}}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{near}}, x_{\text{new}})$ **then**
      $c' \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$;
      if $c' < \texttt{Cost}(x_{\text{new}})$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$;
  $E' \leftarrow E' \cup \{(x_{\min}, x_{\text{new}})\}$;
  for *all* $x_{near} \in X_{\text{near}} \setminus \{x_{\min}\}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{new}}, x_{\text{near}})$ *and*
    $\texttt{Cost}(x_{\text{near}}) >$
    $\texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}}))$ **then**
      $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E' \leftarrow E' \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$;
      $E' \leftarrow E' \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
return $G' = (V', E')$

Karaman, Frazzoli. 2010.

Graphic from Gammell, 2016. Thesis.

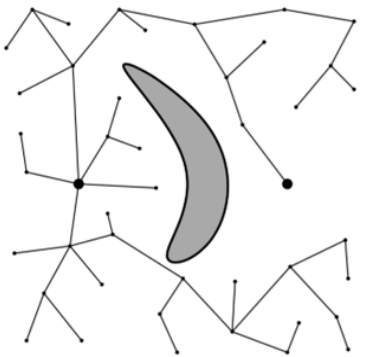


RRT* Extend vs RRT Extend

$V' \leftarrow V$; $E' \leftarrow E$;
$x_{\text{nearest}} \leftarrow \texttt{Nearest}(G, x)$;
$x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x)$;
if $\texttt{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
  $V' \leftarrow V' \cup \{x_{\text{new}}\}$;
  $x_{\min} \leftarrow x_{\text{nearest}}$;
  $X_{\text{near}} \leftarrow \texttt{Near}(G, x_{\text{new}}, |V|)$;
  for *all* $x_{\text{near}} \in X_{\text{near}}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{near}}, x_{\text{new}})$ **then**
      $c' \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$;
      if $c' < \texttt{Cost}(x_{\text{new}})$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$;
  $E' \leftarrow E' \cup \{(x_{\min}, x_{\text{new}})\}$;
  for *all* $x_{near} \in X_{\text{near}} \setminus \{x_{\min}\}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{new}}, x_{\text{near}})$ *and*
    $\texttt{Cost}(x_{\text{near}}) >$
    $\texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}}))$ **then**
      $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E' \leftarrow E' \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$;
      $E' \leftarrow E' \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
return $G' = (V', E')$

Karaman, Frazzoli. 2010.

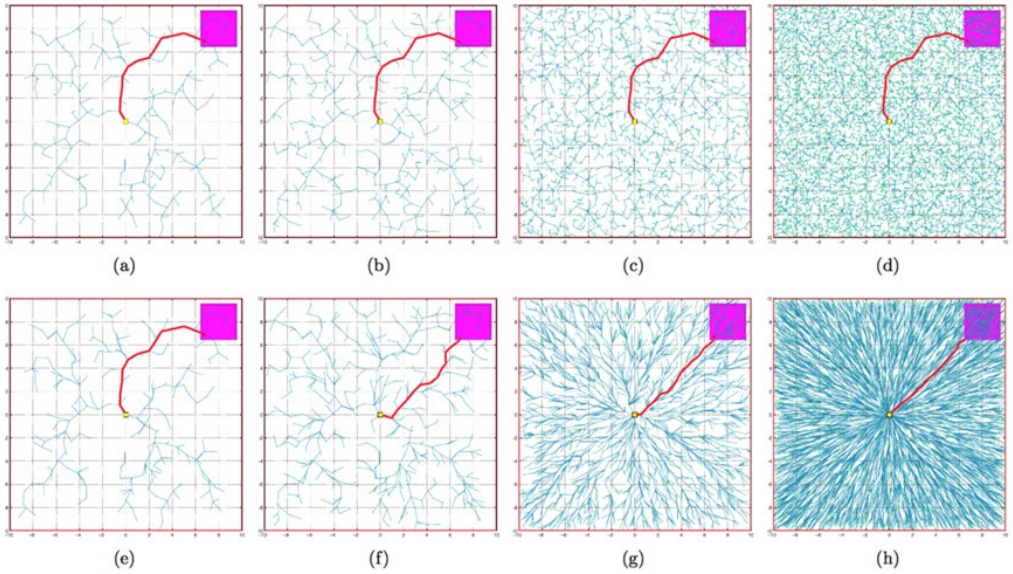
$V' \leftarrow V$; $E' \leftarrow E$;
$x_{\text{nearest}} \leftarrow \texttt{Nearest}(G, x)$;
$x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x)$;
if $\texttt{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
  $V' \leftarrow V' \cup \{x_{\text{new}}\}$;
  $E' \leftarrow E' \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$;
return $G' = (V', E')$



RRT*

RRT

RRT*

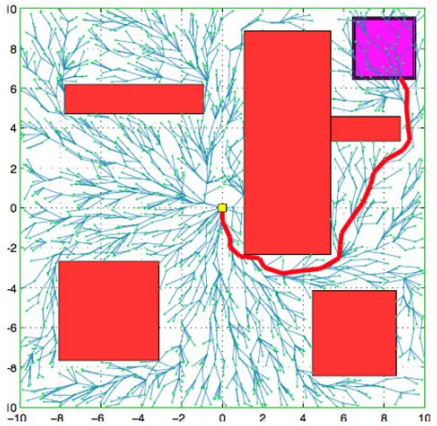


Karaman, Frazzoli. 2010.

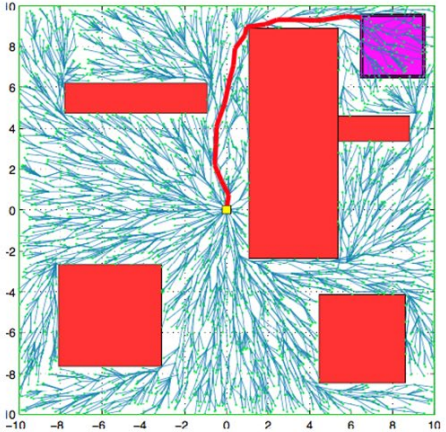
RRT*

RRT*

2500 Iterations



5000 Iterations



Karaman, Frazzoli. 2010.

PRM*

- "Simple" PRM

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

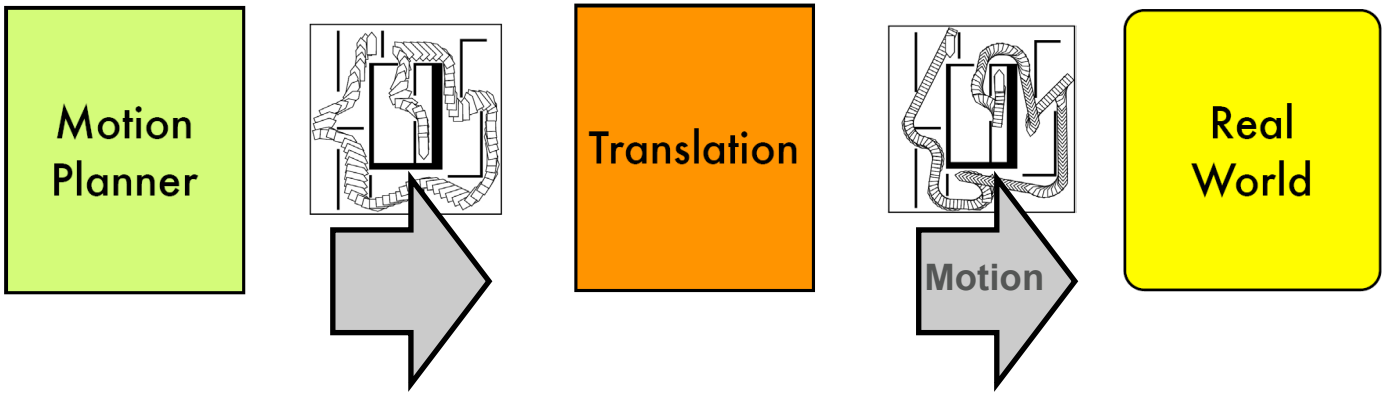
- PRM*
  - Radius shrinks with more vertices
  - One change, and its' optimal!

$V \leftarrow \{x_{\text{init}}\} \cup \{\texttt{SampleFree}_i\}_{i=1,\dots,n}$; $E \leftarrow \emptyset$;
foreach $v \in V$ **do**
  $U \leftarrow \texttt{Near}(G = (V, E), v, \gamma_{\text{PRM}}(\log(n)/n)^{1/d}) \setminus \{v\}$;
  foreach $u \in U$ **do**
    if $\texttt{CollisionFree}(v, u)$ **then** $E \leftarrow E \cup \{(v, u), (u, v)\}$
return $G = (V, E)$;

Overview

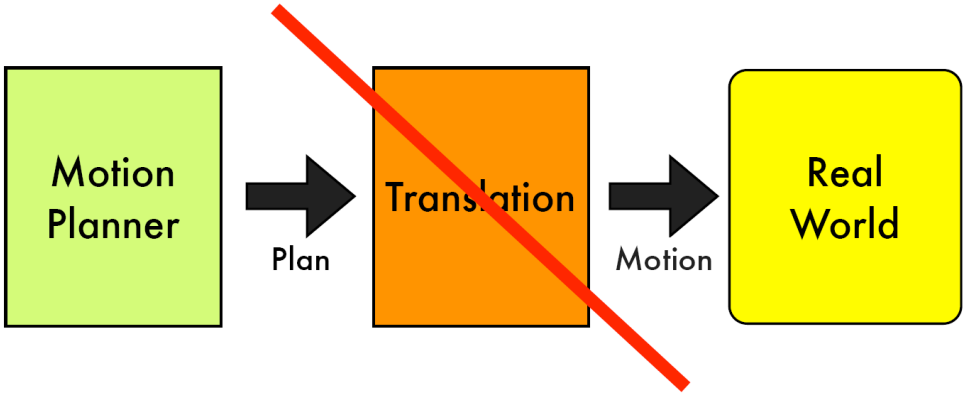
- Motion Planning Definition
- Bug Algorithms
- Discrete Search/Roadmaps
- Configuration Space
- Sampling-Based Planners
- Asymptotically Optimal Planning
- Kinodynamic Planning
- Learning And Planning
- Task and Motion Planning
- Planning Under Uncertainty

Decoupled Motion Planning Paradigm



- Finds a geometrically feasible plan with a planner that is approximately correct
- Satisfy any remaining physical constraints using a translation step
- Execute in the real world

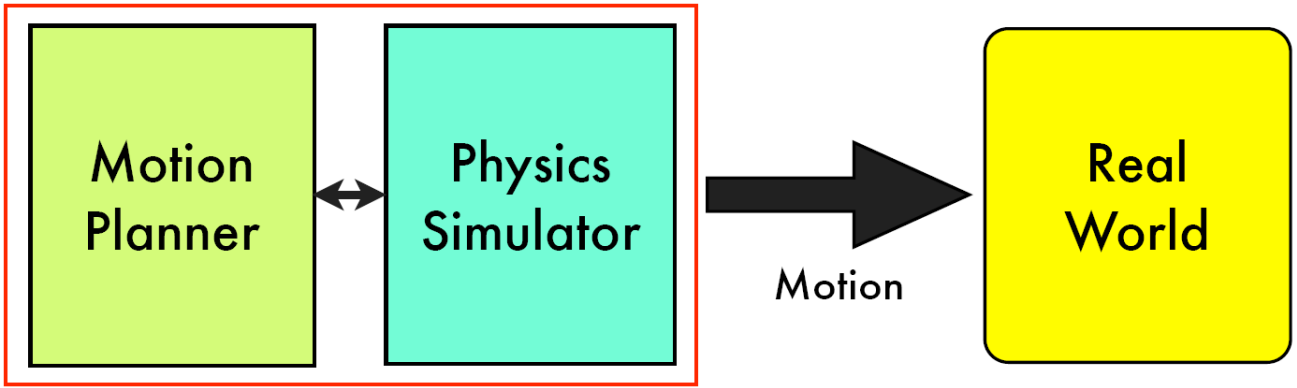
Shortcomings of Decoupled Motion Planning



- Motions are often low quality or inadmissible

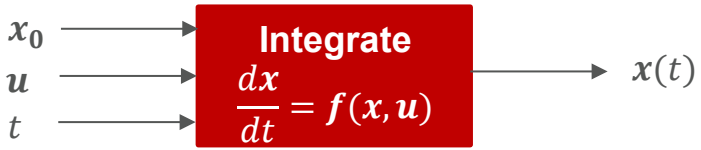
- Translating “collision-free” paths to physical motion is hard

Another approach (Native Approach)



- Reason directly about physical constraints in a simulation

Motion Planning with Physics-based Simulations

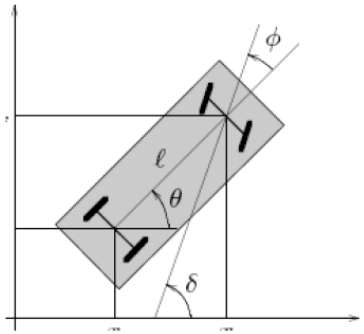


- Physics engines can be used to simulate robot motions
- Physics engines provide greater simulation accuracy
- Physics engines can take into account friction, gravity, and interactions of the robot with objects in the environment
- Existing tools

BULLET PHYSICS LIBRARY

OPEN DYNAMICS ENGINE™

Simple Car VS Real World Car



simple Car:

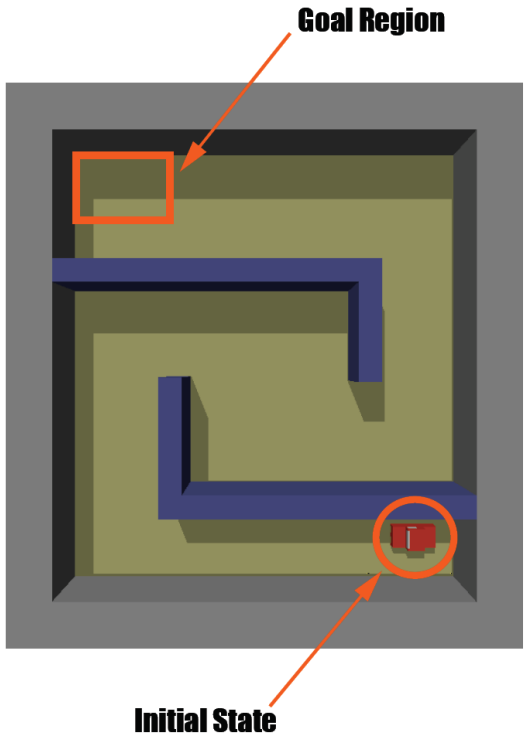
$$\dot{x} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix}$$

$$= \begin{pmatrix} u_v \cos\theta \\ u_v \sin\theta \\ \dfrac{u_v \tan u_\phi}{L} \end{pmatrix}$$

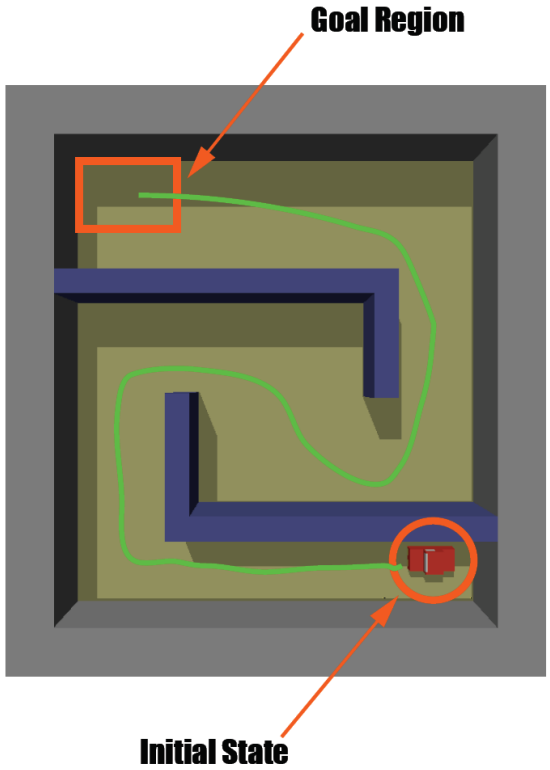
Real Car:

- A real car is even more complex
- Turning is controlled by steering
- Car motion is determined by tire contact
- Skidding, slip and other behavior

Example Problem

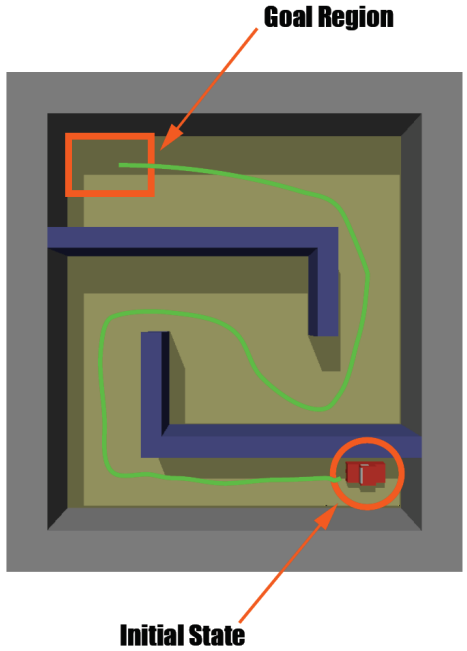


Example Problem



This is a Search Problem

- State space is enormous, continuous and hard to discretize
- Solution paths are usually “deep”
- There are many solutions



Moving between two states (without obstacles)

- **Two-Point Boundary Value Problem (BVP):**
  - Find a control sequence to take system from state X_I to state X_G while obeying kinematic constraints.
  - How to solve this problem?

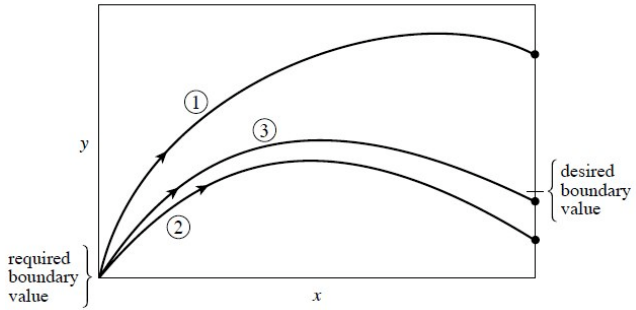
Shooting Method

- “Shoot” out trajectories in different directions until a trajectory of the desired boundary value is found.
  - System

$$\frac{d\mathbf{y}}{dx} + \mathbf{f}(x, \mathbf{y}) = 0$$

  - Boundary condition

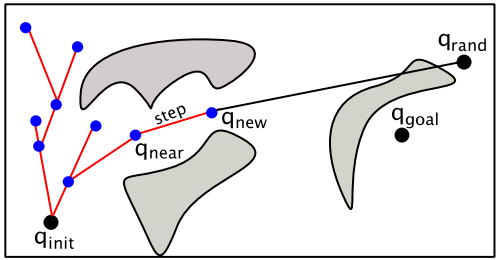
$$y(0) = 0,\ y(1) = 1$$



  - Convert to an initial boundary problem: Pick initial guess, iteratively get closer to goal

Kinodynamic Motion Planning with Tree-Based Methods

Tree-based Approaches



RRT

1: $\mathrm{T} \leftarrow$ create tree rooted at x_0

2: **While** solution not found **do**

\\ select state from tree

3: $x_{rand} \leftarrow \mathrm{StateSample}()$

4: $x_{near} \leftarrow$ nearest state in T to x_{rand} according to distance ρ

\\ add new branch to tree from selected state

5: $\lambda \leftarrow \mathrm{GenerateLocalTrajectory}(x_{near}, x_{rand})$

6: **if** $\mathrm{IsSubTrajectoryValid}(\lambda, 0, step)$ **then**

7: $x_{new} \leftarrow \lambda(step)$

8: add configuration x_{new} and edge (x_{near}, x_{new}) to T

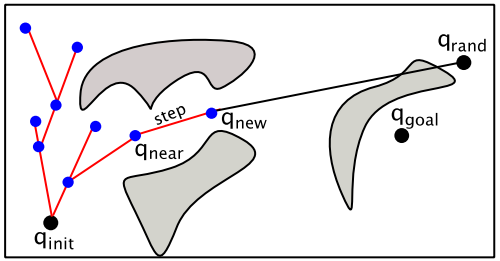
\\ check if a solution is found

9: **if** $\rho(x_{new}, x_{goal}) \approx 0$ **then**

10: **return** solution trajectory from root to x_{new}

Kinodynamic Motion Planning with Tree-Based Methods

Implementation of Tree-based Approaches

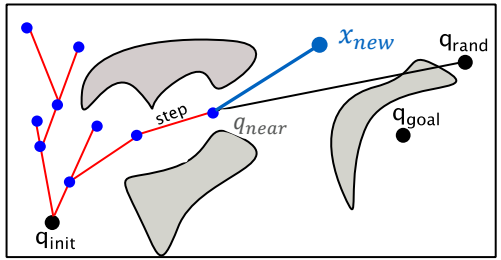


- SampleState()
  - random values for state components
- $\rho: X \times X \rightarrow \mathbb{R}^{\geq 0}$
  - Distance metric between states
- $\lambda \leftarrow \text{GenerateLocalTrajectory}(x_{near}, x_{rand})$
  - does it not create the same two-boundary value problems as in PRM?
  - is it necessary to connect to x_{rand}?
  - does it suffice to just come close to x_{rand}?
- $\text{IsSubTrajectoryValid}(\lambda, 0, steps)$
  - Incremental approach

Kinodynamic Motion Planning with Tree-Based Methods

Idea for avoiding the 2-point Boundary

Problem: Rather than computing a trajectory from x_{near} to x_{rand} compute a trajectory that starts at x_{near} and extends toward x_{rand}

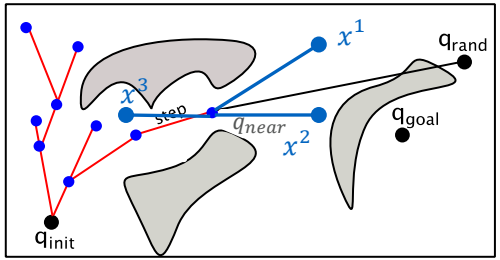


- Approach 1 – extend according to random control
  - Sample random control u in U
  - Integrate equations of motions when applying u to x_{near} for Δt units of time, i.e.,

$$\lambda \leftarrow x(t) = x_{near} + \int_0^{\Delta t} f(x(\tau), u)\, d\tau$$

Kinodynamic Motion Planning with Tree-Based Methods

Idea for avoiding the 2-point Boundary Problem: Rather than computing a trajectory from x_{near} to x_{rand} compute a trajectory that starts at x_{near} and extends toward x_{rand}

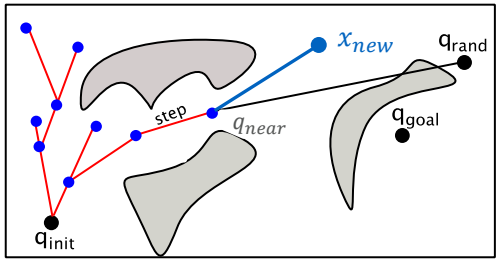


- Approach 2 – find the best-out-of-many random controls

1: **for** $i = 1, \ldots, m$ **do**
2: $u \leftarrow$ sample random control in U
3: $\lambda \leftarrow x(t) = x_{near} + \int_0^{\Delta t} f(x(\tau), u)\, d\tau$
4: $\mathrm{d_i} \leftarrow \rho(x_{rand}, \lambda_i(\Delta t)$
5: **return** λ_i with minimum d_i

Kinodynamic Motion Planning with Tree-Based Methods

Idea for avoiding the 2-point Boundary Problem: Rather than computing a trajectory from x_{near} to x_{rand} compute a trajectory that starts at x_{near} and extends toward x_{rand}



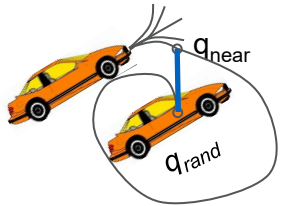
- Approach 2 – find the best-out-of-many random controls

1: **for** $i = 1, \ldots, m$ **do**
2: $u \leftarrow$ sample random control in U
3: $\lambda \leftarrow x(t) = x_{near} + \int_0^{\Delta t} f(x(\tau), u)\, d\tau$
4: $\mathrm{d_i} \leftarrow \rho(x_{rand}, \lambda_i(\Delta t)$
5: **return** λ_i with minimum d_i

RRTs and Distance Metrics

- Hard to define *d*, the distance metric
  - Mixing velocity, position, rotation ,etc.

How do you pick a good q_{near}?



Configurations are close according to Euclidian metric, but actual distance is large

Overview

- Motion Planning Definition
- Bug Algorithms
- Discrete Search/Roadmaps
- Configuration Space
- Sampling-Based Planners
- Asymptotically Optimal Planning
- Kinodynamic Planning
- Learning And Planning
- Task and Motion Planning
- Planning Under Uncertainty

Machine Learning and Motion Planning

Training/Learning Phase

Testing/Inference Phase

Biasing the Sampling Distribution

- Theoretical Insights supporting biased sampling

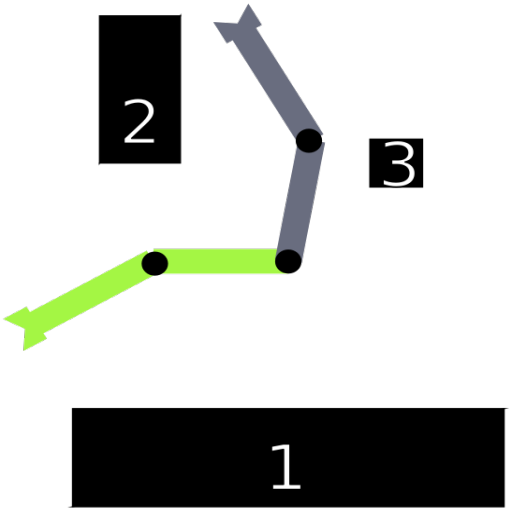
Algorithm 1: General Sampling Based Planner

Input : Number of iterations N
Output : Graph structure G

```
1 while i ≤ N or solutionFound() do
2 |   x ~ Uniform()
3 |   update(G, x)
4 end
5 return G
```

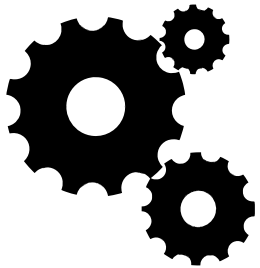
Learning a biased sampling distribution

Input



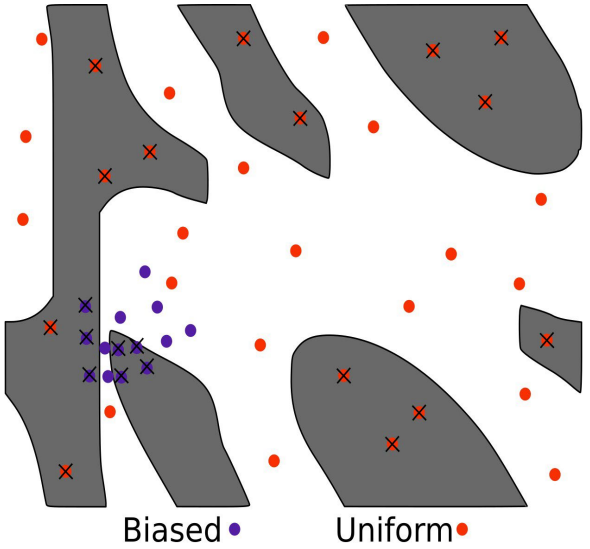
Workspace W

Learning Model



$Map: [W] \rightarrow X$

Output



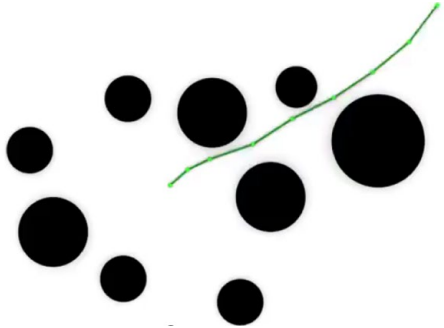
Sampling Dist. $P(X|W)$

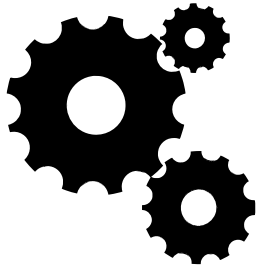
Retrieval of Database Samples

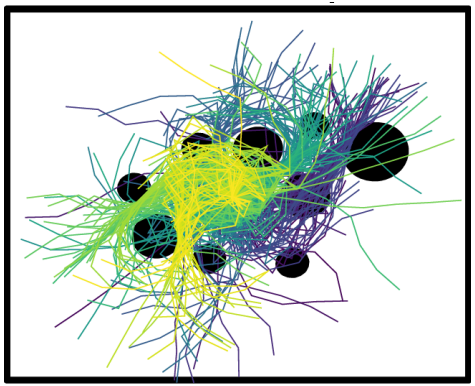
Input

Learning Model

Output







Workspace W

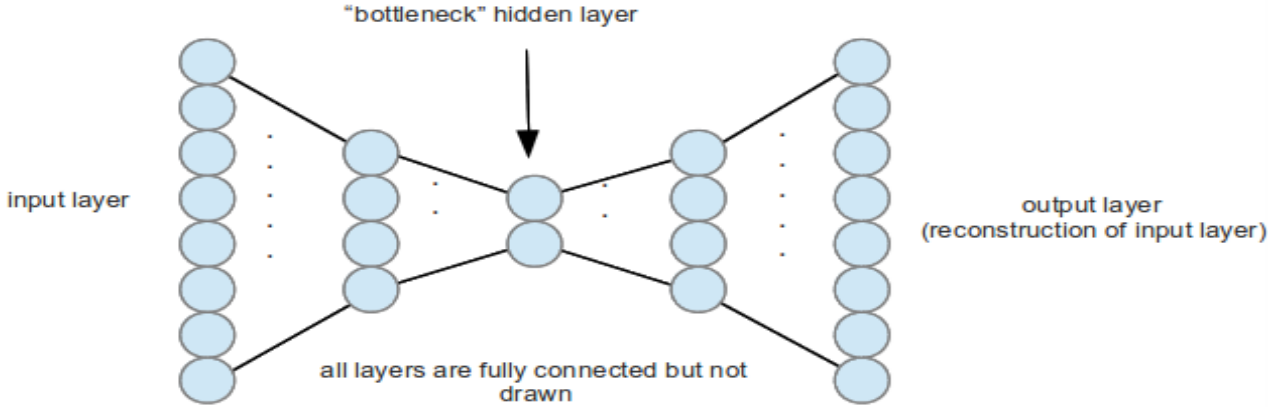
$Map: [W] \rightarrow X$

P(X|W)

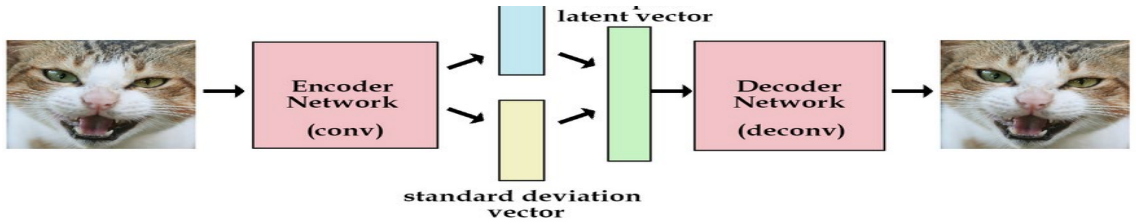
Variational Autoencoders

How to learn a sampling distribution with a neural network?

Autoencoder

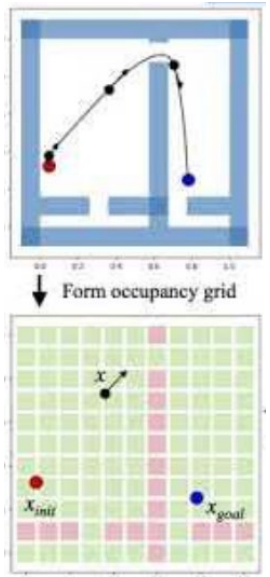


Variational Autoencoder



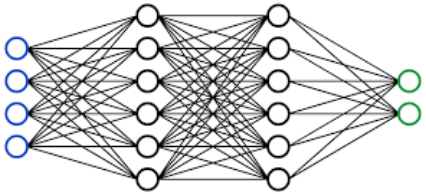
A neural network baseline

Input

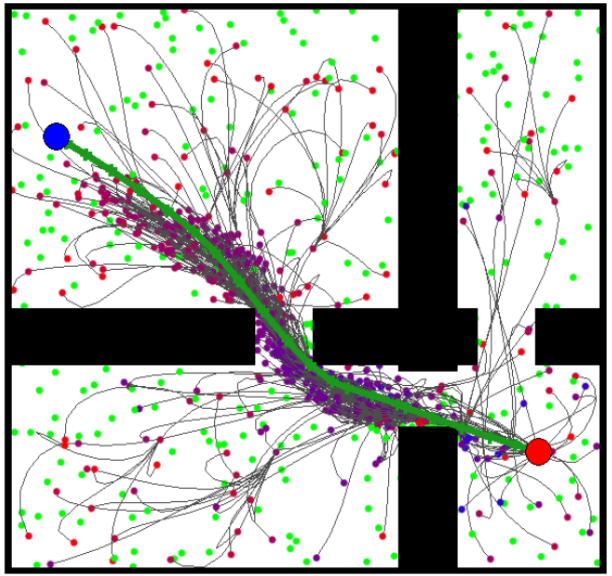


Learning Model

Conditional Variational Autoencoder (CVAE)



Output



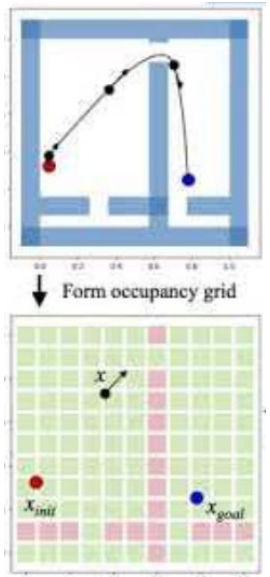
[10] Ichter et al ``Learning sampling distributions for robot motion planning'', *ICRA*, 2018.

A neural network baseline (CVAE)

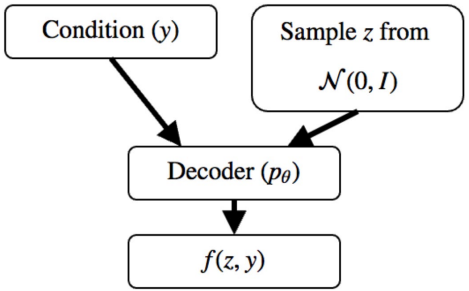
Input

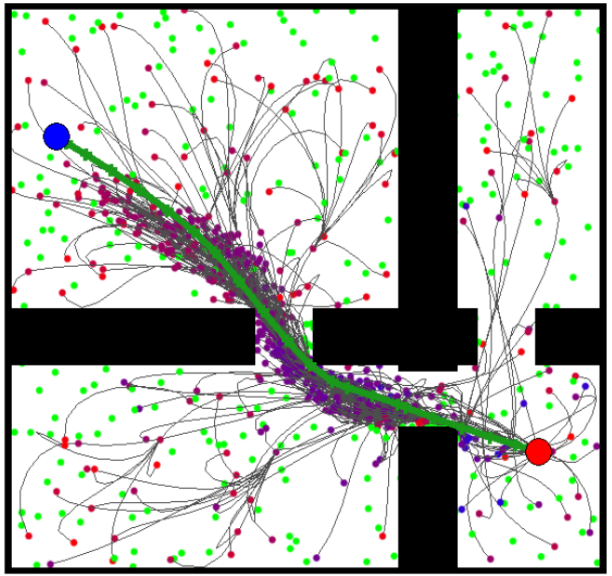
Learning Model

Output



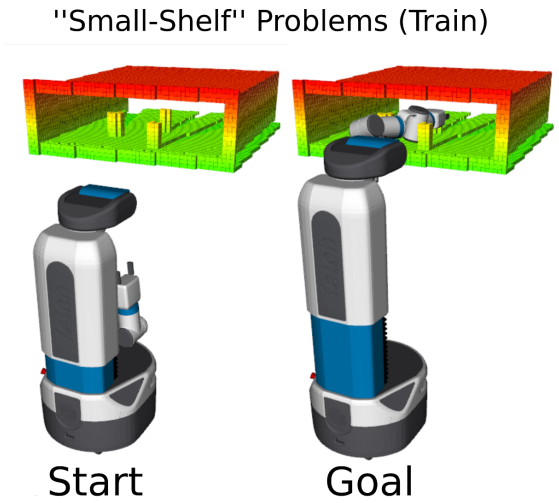
Conditional Variational Autoencoder (CVAE)

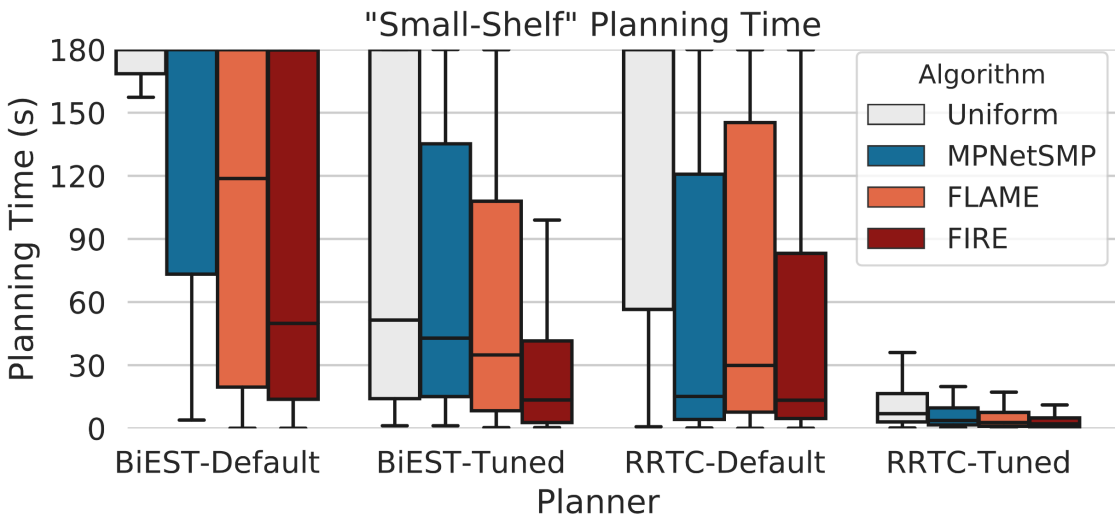




[10] Ichter et al ``Learning sampling distributions for robot motion planning'', *ICRA*, 2018.

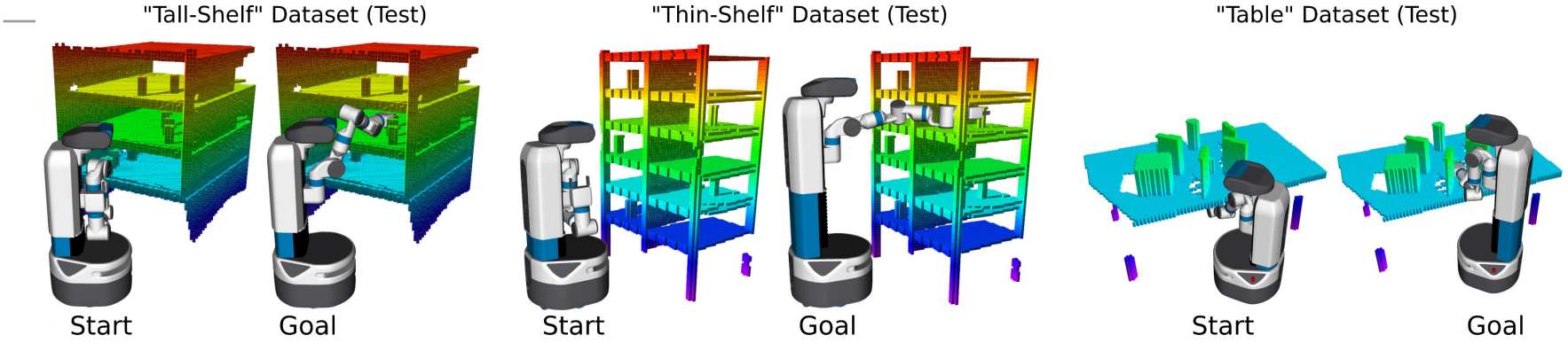
Generalizing within the training class of problems

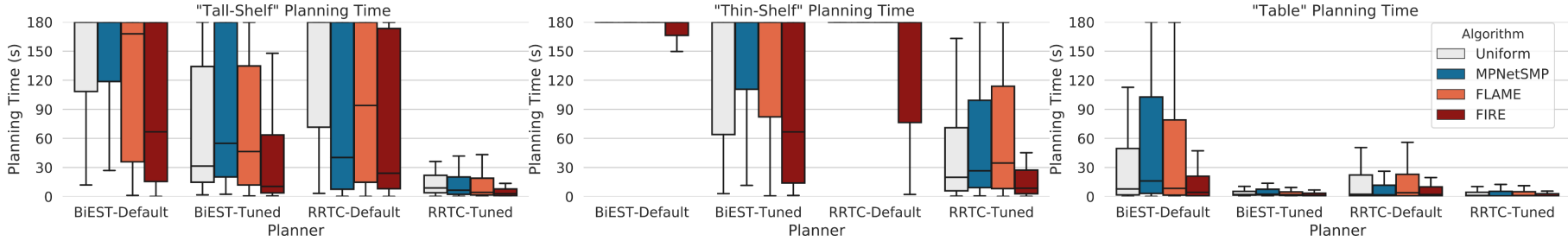




[12] Chamzas et al "Learning to Retrieve Relevant Experiences for Motion Planning", *ICRA*, 2022

Generalizing outside the training class of problems





[12] Chamzas et al "Learning to Retrieve Relevant Experiences for Motion Planning", *ICRA*, 2022

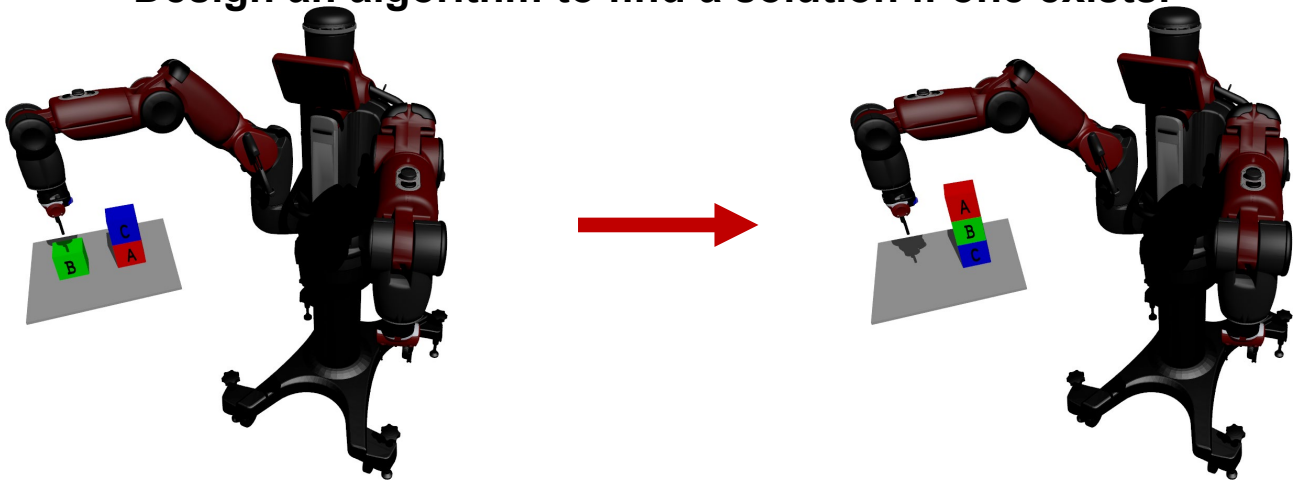
Overview

- Motion Planning Definition
- Bug Algorithms
- Discrete Search/Roadmaps
- Configuration Space
- Sampling-Based Planners
- Asymptotically Optimal Planning
- Kinodynamic Planning
- Learning And Planning
- Task and Motion Planning
- Planning Under Uncertainty

Task and Motion Planning

Informal		General Case
Tell robot where everything is	›	Initial state
Tell it where everything needs to end up	›	Goal state(s)
Let the robot figure it out	›	Transitions between states

Design an algorithm to find a solution if one exists.



Planning Domain Definition Language

- Planning Domain Definition Language (PDDL) is a way to express a task planning problem
- Use prefix notation:
  - `(+ 1 2)` returns `3`; `(+ 1 2 (* 3 4))` returns `15`
- Define:
  - Predicates: `on(?x,?y)`, `clear(?x)`, etc.
  - Actions: `stack(?x,?y)`, etc.
  - Initial state: `(ontable a)`, etc.
  - Goal state: `(on a b)`, `(on b c)`, `(ontable a)`

Planning Domain Definition Language

Operators

```
(define (domain blocks)
  (:predicates (on ?x ?y) (ontable ?x)
               (clear ?x) (handempty) (holding ?x))
  (:action pick-up  :parameters (?x)
           :precondition (and (clear ?x) (ontable ?x)
                              (handempty))
           :effect (and (not (ontable ?x))(not (clear ?x))
                        (not (handempty)) (holding ?x)))
  (:action put-down  :parameters (?x)
           :precondition (holding ?x)
           :effect (and (not (holding ?x)) (clear ?x)
                        (handempty)(ontable ?x)))
  (:action stack  :parameters (?x ?y)
           :precondition (and (holding ?x) (clear ?y))
           :effect (and (not (holding ?x))(not (clear ?y))
                        (clear ?x)(handempty) (on ?x ?y)))
  (:action unstack  :parameters (?x ?y)
           :precondition (and (on ?x ?y) (clear ?x)
                              (handempty))
           :effect (and (holding ?x) (clear ?y)
                        (not (clear ?x))
                        (not (handempty))
                        (not (on ?x ?y)))))
```

Facts

```
(define
 (problem sussman-anomaly)
  (:domain blocks)
  (:objects a b c)
  (:init (on c a)
         (ontable a)
         (ontable b)
         (clear c)
         (clear b)
         (handempty))
  (:goal (and (on b c)
              (on a b))))
```

Grounding on First-Order Logic

- Rely on the fact that there are finite:
  - Constants
  - Objects
  - locations
  - steps
- Construct a list of things that could be true or false for each time step
  - `At(Obj?, Loc?) → PLATE_2_AT_LOC_3`
  - `At(Obj?, Loc?) → PLATE_1_AT_LOC_5`
  - Etc.

Graph Search Approach

- Search state space to find a sequence of actions from start to goal
- Find successor states by looking at all actions that can be applied from a given state:
  - this means preconditions are met
- Search until we reach the goal:
  - this search space is typically very large
  - general heuristic search for task planning is an open research problem
  - use heuristic search
    - e.g., A*, Dijkstra's, FF, ...

Task Planning Approaches

- **Graph approach (employed so far)**
  - From PDDL description build a graph
  - Apply heuristic search algorithms
    - Search in graph implied by logical description
    - Heuristics are essential for good performance

- **Another approach that can be very efficient**
  - Reduce planning to a SAT problem instance (Boolean logic)
  - Solve with off-the-shelf solver
  - Avoid relying on task planning heuristics

SAT-Plan Encoding

- We want to feed a problem into the SAT solver
  - If the problem is SAT,
    - We get an assignment we can convert to a task plan
  - If the problem is UNSAT,
    - it means there is no task plan within that time horizon
- We need to eliminate quantifiers ∀ and ∃
  - Quantifiers are used with variables
  - so we will ground variables and time steps

SAT-Plan Encoding

1. (Initial State) Need to be in the start state at step 0:
    - Start State: `start-step-0`
2. (Goal State) Need to get to the goal by step k (a no-op can be used if we get there too early):
    - Goal State: `goal-step-k`
3. (Operator Encodings) Applying an operator at step i implies preconditions and, on the next step, effects
    - `op-step-i` → `pre(op)-step-i` ∧ `eff(op)-step-(i+1)`

SAT-Plan Encoding

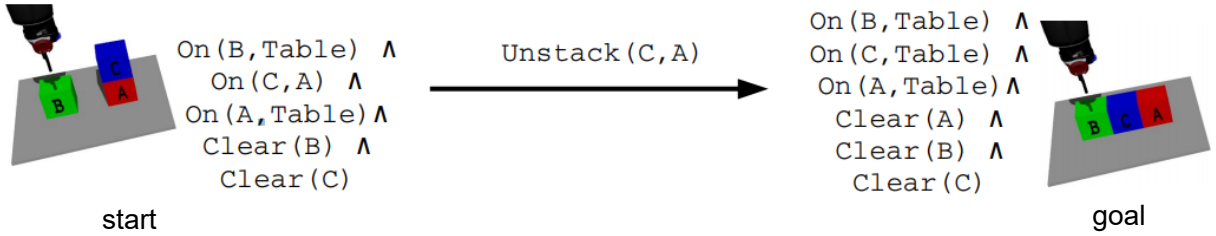
- 4. (Frame Axioms) Nothing changes unless we change it:
  - Frame Axioms: `(P-step-i == P-step-(i+1)) ∨ (op-P-0 ∨ … ∨ op-P-n)`
- 5. (Complete Exclusion Axion) Can take only one action per step:
  - Operator Exclusion: `op-0 → ((¬ op-1) ∧ … ∧ (¬ op-n))`
- We ground all possible operators and predicates:
  - Relations: `rel_0-arg_0, …, rel_0-arg_K, …, rel_M-arg_0, …, rel_M-arg_K`
  - Operators: `op_0-arg_0, …, op_0-arg_K, …, op_M-arg_0, …, op_M-arg_K`

SAT-Plan Encoding

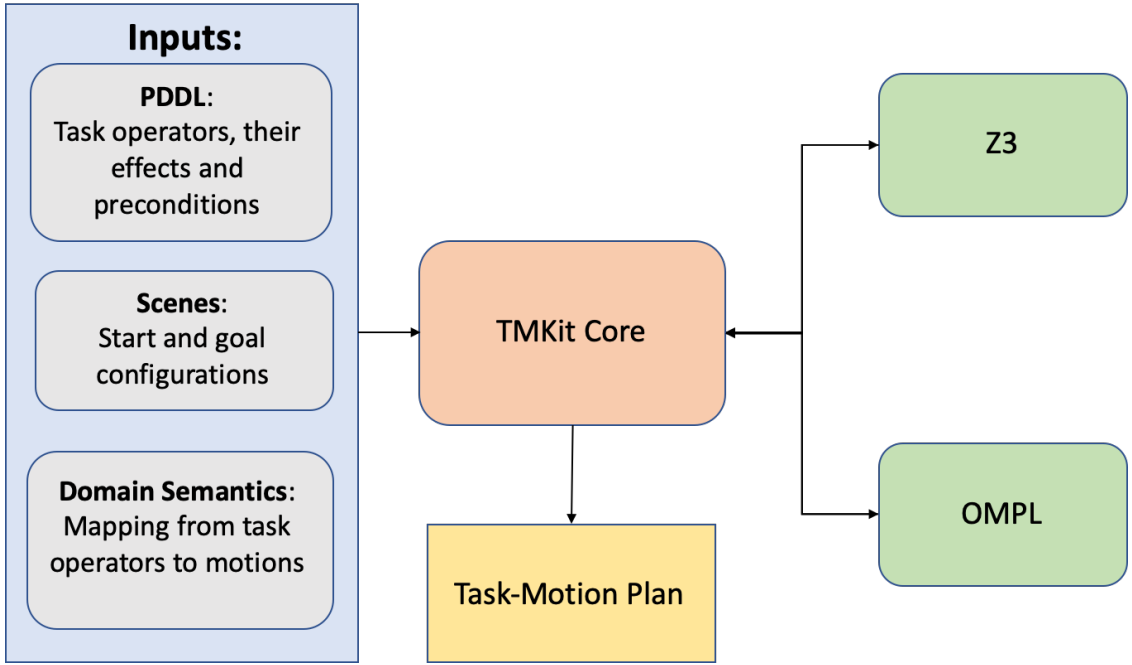
- Assume a value k for the number of steps in the solution
- Construct a huge formula with is the conjunction of 1-5 above
- A solution plan exists if and only if the resulting Boolean formula is satisfiable
- Give formula to a SAT solver
- Get the values that need to be assigned truth values and these will indicate the steps than need to be taken (steps is indicated in the encoding)

SAT-Plan Encoding

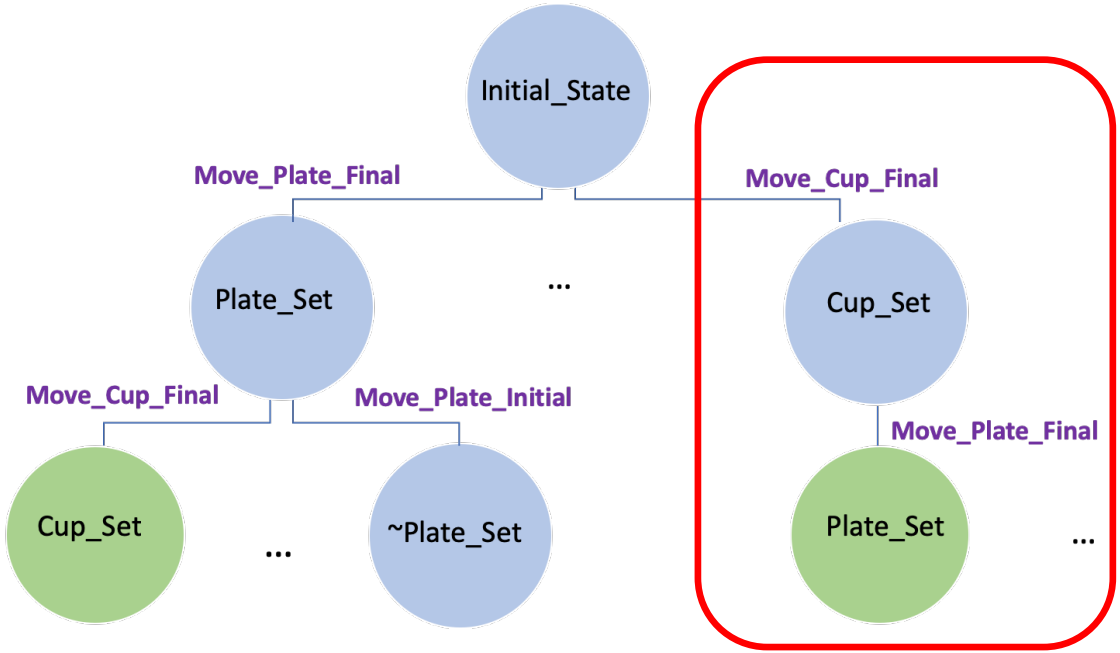
- Start State: `start-step-0`
- Goal State: `goal-step-k`
- Operator:
  - `Unstack_C_A_step_0 →  pre_unstack_C_A_step_0 ∧ eff_unstack_C_A_step_1`
- Frame axiom:
  - `(on_C_A_step_0 == on_C_A_step_1) ∨ (unstack_C_A_step_0) ∨ …`
- Operator exclusion:
  - `Unstack_C_A_step_0 → (¬ stack_C_B_step_0 ) ∧ (¬ stack_B_C_step_0) ∧ …`



Case Study – TMKit architecture



Task and Motion Planning



Initial_State:

¬ Plate_Set

¬ Cup_Set

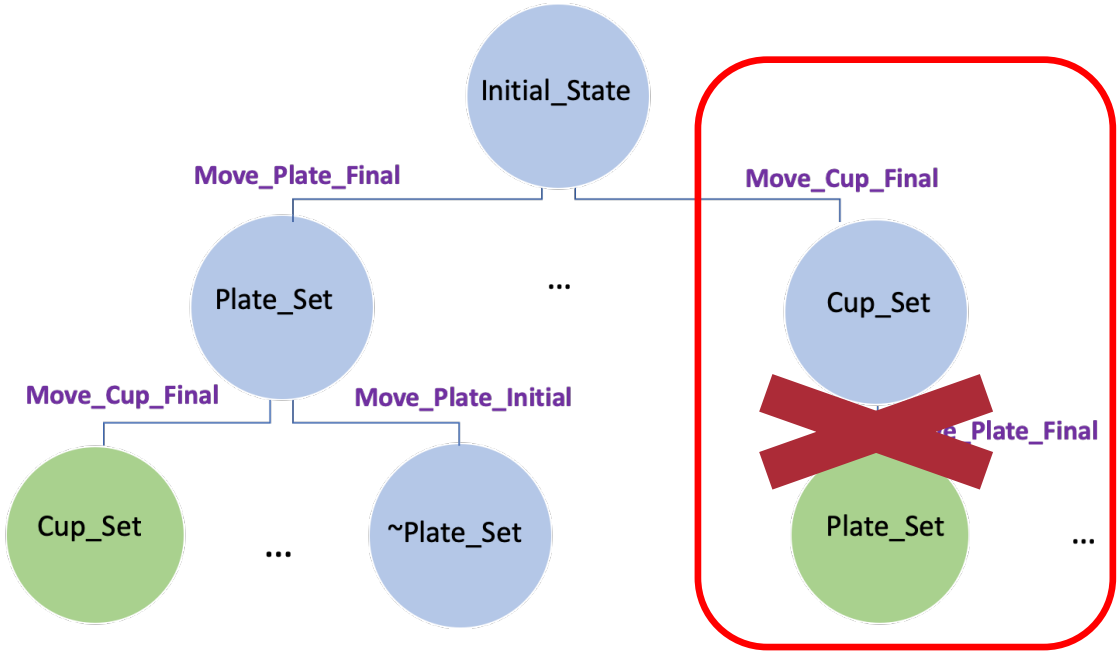
Plate_Initial

Cup_Initial

Hand_Free

...

Task and Motion Planning



Initial_State:

¬ Plate_Set

¬ Cup_Set

Plate_Initial

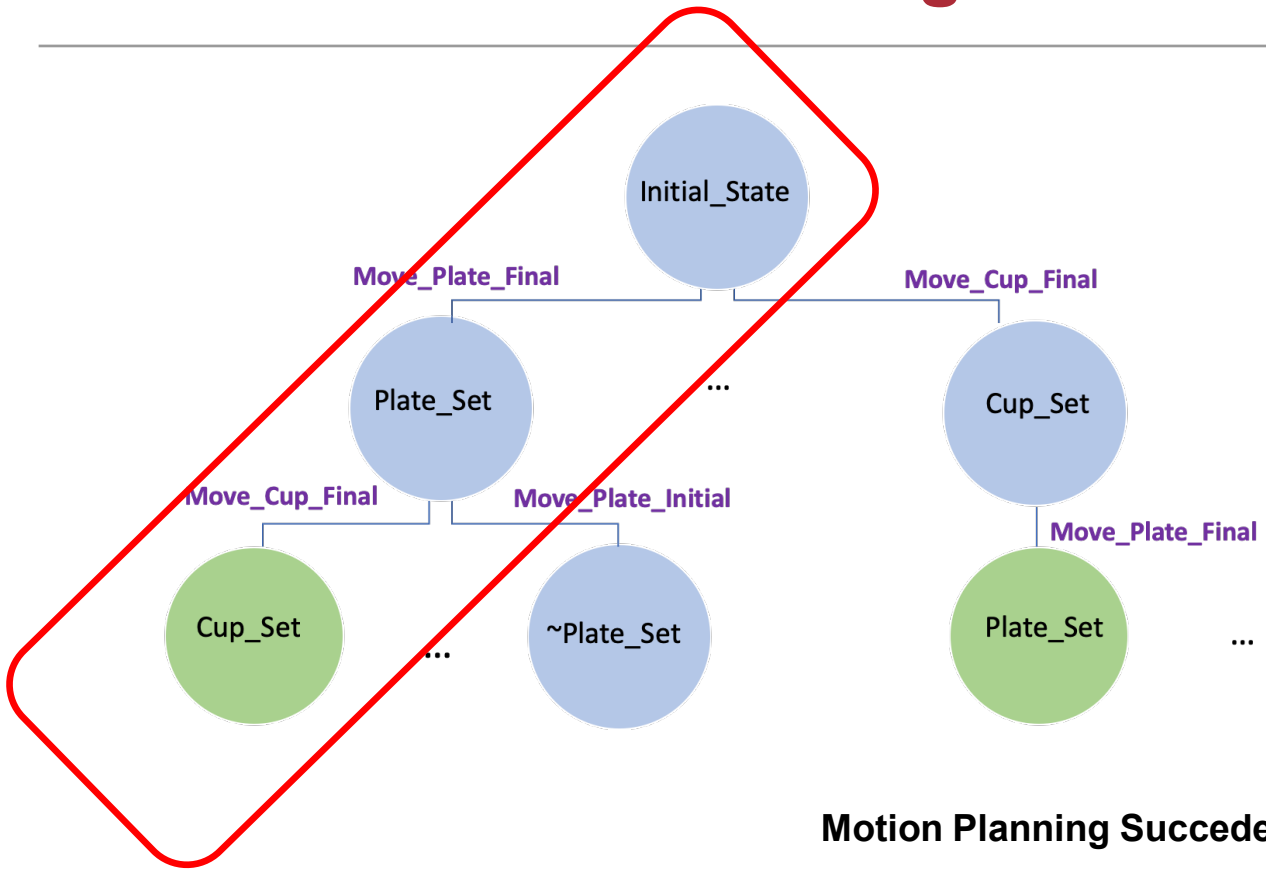
Cup_Initial

Hand_Free

...

Motion Planning Failed!

Task and Motion Planning



Initial_State:

¬ Plate_Set

¬ Cup_Set

Plate_Initial

Cup_Initial

Hand_Free

...

Motion Planning Succeded!

Overview

- Motion Planning Definition
- Bug Algorithms
- Discrete Search/Roadmaps
- Configuration Space
- Sampling-Based Planners
- Asymptotically Optimal Planning
- Kinodynamic Planning
- Learning And Planning
- Task and Motion Planning
- Planning Under Uncertainty

Source of Uncertainty

- **What are the sources of uncertainty**

- **Sensing (State)**
  - Where is the robot?
  - Where are the obstacles?
  - What is the map of the environment?

- **Motion (Actions)**
  - How will objects in the environment move?
  - Given a command action, where will the robot end up?

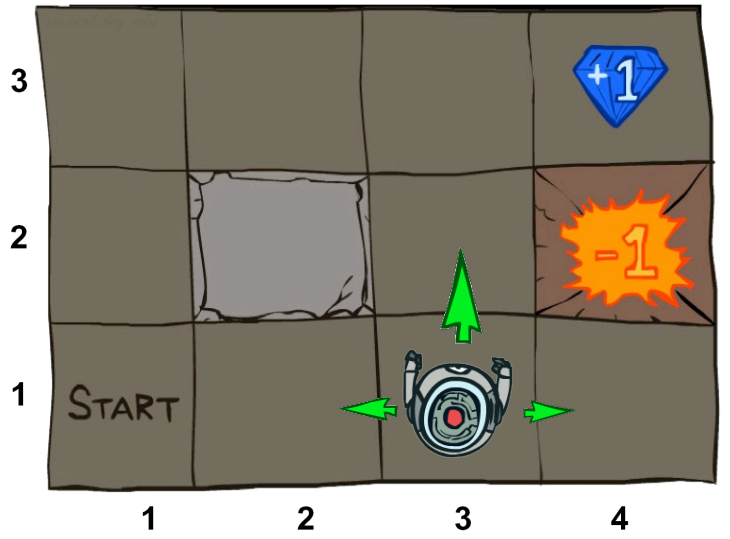
Markov Decision Processes

An MDP is defined by:

A set of states $s \in S$

A set of actions $a \in A$

- A transition function $T(s, a, s')$
  - Probability that a from s leads to s', i.e., $P(s'| s, a)$
  - Also called the model or the dynamics
- A reward function $R(s, a, s')$
  - Sometimes just R(s) or R(s')
- A start state
- Maybe a terminal state

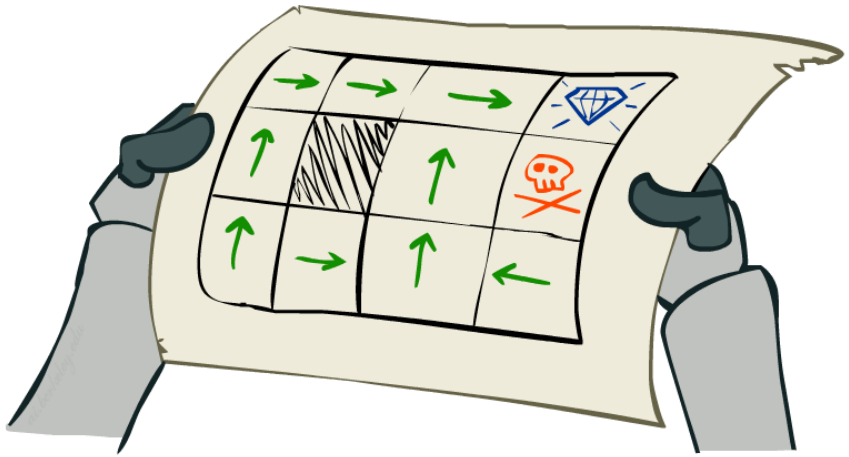


Policies

In deterministic single-agent search problem
of actions, from start to a goal

For MDPs, we want a policy $\pi: S \rightarrow A$

- A policy π gives an action for each state
- An optimal policy π^* is one that maximizes expecte
- An explicit policy defines a reflex agent
- It computed the action for a single state only



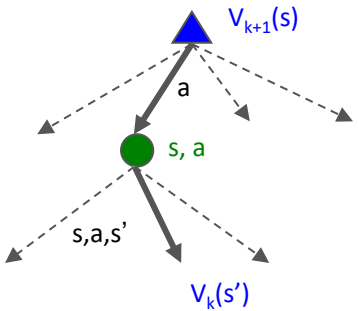
Optimal policy when R(s, a, s') = -0.03
for all non-terminals s

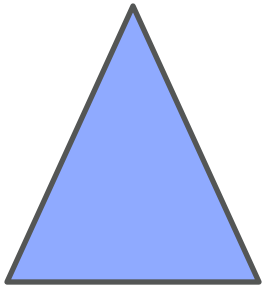
Value Iteration

- Start with $V_0(s) = 0$: no time steps left means an expected reward sum of zero

- Given vector of $V_k(s)$ values, do one step from each state:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

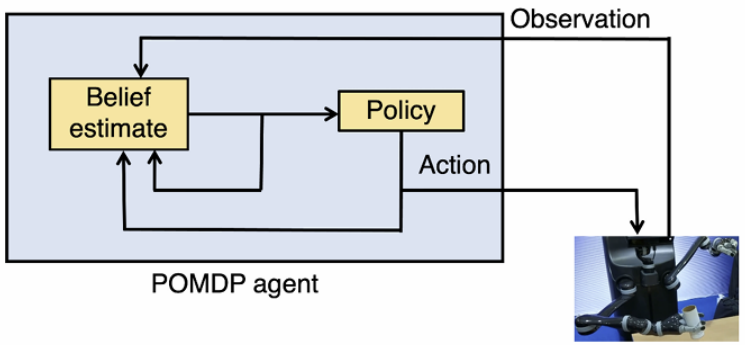




- Repeat until convergence
- Complexity of each iteration: $O(S^2A)$
- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do

POMDP Definition

- 6-tuple model:
  - States: S
  - Actions: A
  - Transition function: $T(s, a, s')$ or $P(s'|s, a)$
    - Represents the non-deterministic effects of a
  - Reward: $R(s, a, s')$
  - Observation space: O
  - Observation function: $Z(s', a, o)$ or $P(o|s', a)$
    - Represent errors and noise in measurement and perception



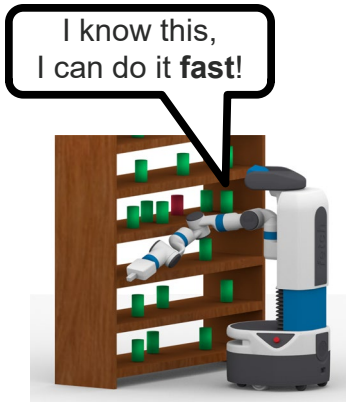
Overview

- Motion Planning Definition
- Bug Algorithms
- Discrete Search/Roadmaps
- Configuration Space
- Sampling-Based Planners
- Asymptotically Optimal Planning
- Kinodynamic Planning
- Learning And Planning
- Task and Motion Planning
- OMPL and Other Tools

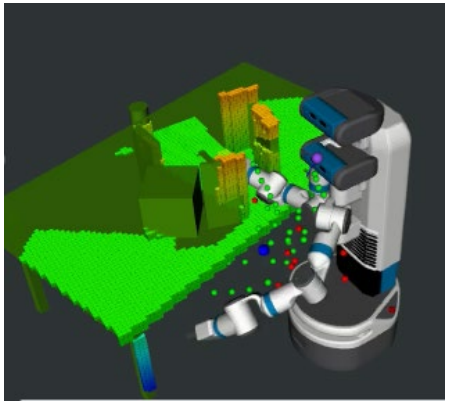
ELPIS Lab

Efficient Learning and Planning for Intelligent Systems

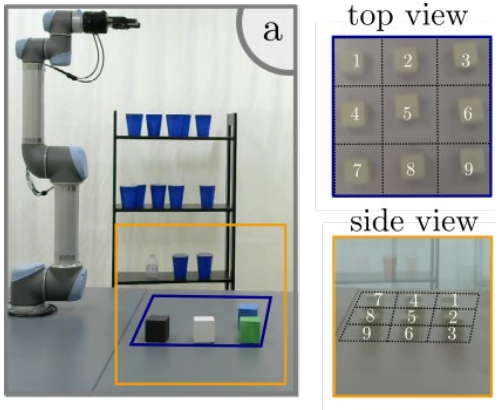
Research Directions/Projects



Leaning For Planning Efficiency

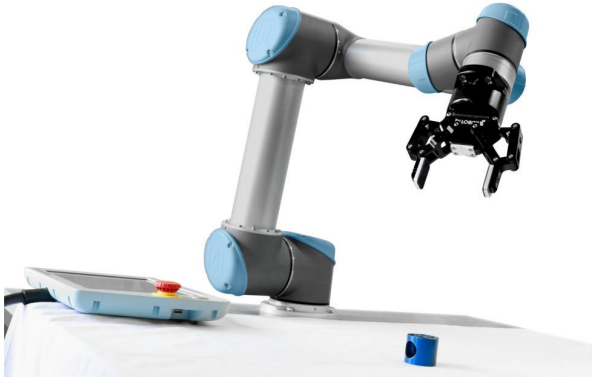


Planning Robustly with Uncertainty



Vision-Based Planning

Available Hardware



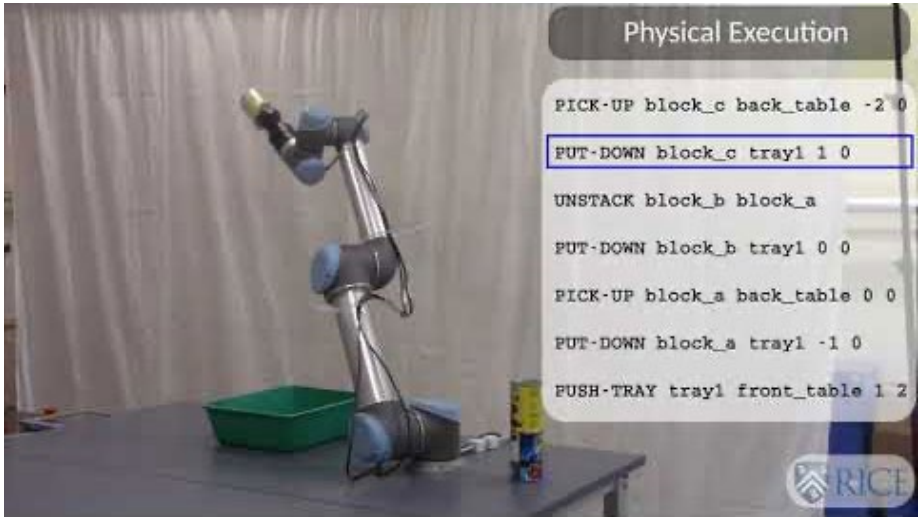
Universal Robotics UR-10
6-Dof Industrial Manipulator

Deep Learning Workstation:
4xA6000 Nvidia GPU, 28-core Intel Xeon

… and more coming soon !

Planned Projects

Self-Supervised Learning Setup



Task and Motion Planning Setup

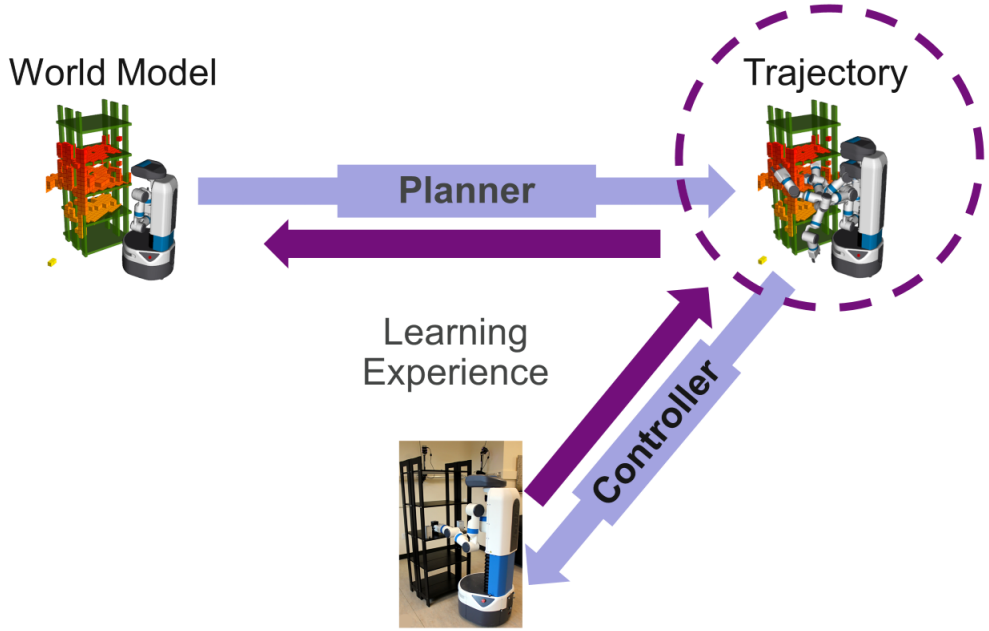
Future Work on the Intersection of Planning and Control

Kinodynamic manipulation

- Non prehensile manipulation
- Kinodynamic TAMP

Interleaving plan/sense actions for manipulation

- POMDP planning
- Ensure safety



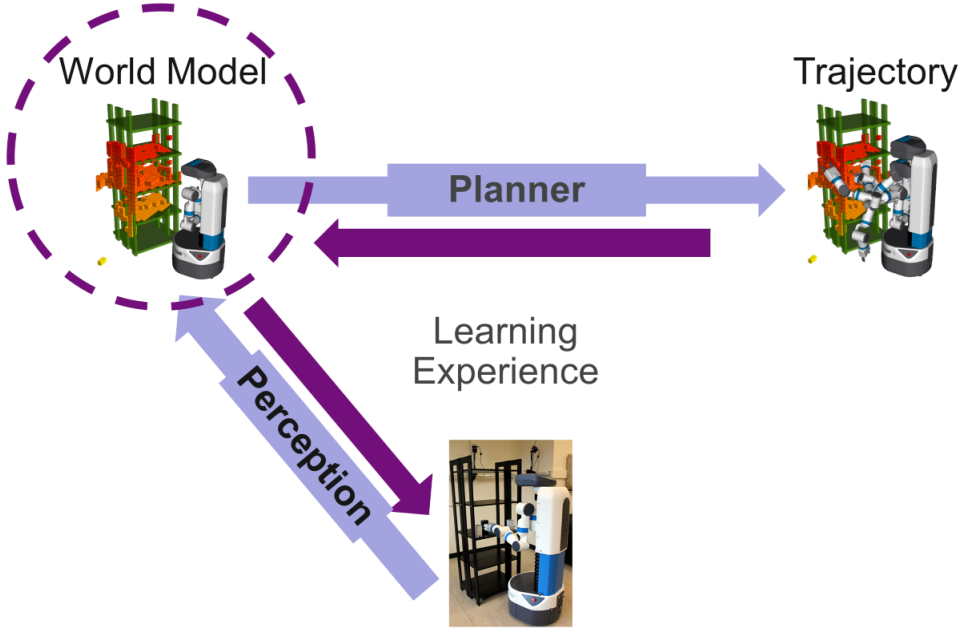
Future Work on the Intersection of Planning and Perception

Learn symbolic logic abstractions

- Image to symbols
- Learn through self-supervision

Learn forward dynamic models of environment objects

- Predict future environment states
- Model deformable/articulated objects



Next weeks Presentation Schedule!

- Groups 9 (Inclusive) to 19 will present on Monday (Dec 10)

- Groups 1-8 will present on Thursday (Dec 13)