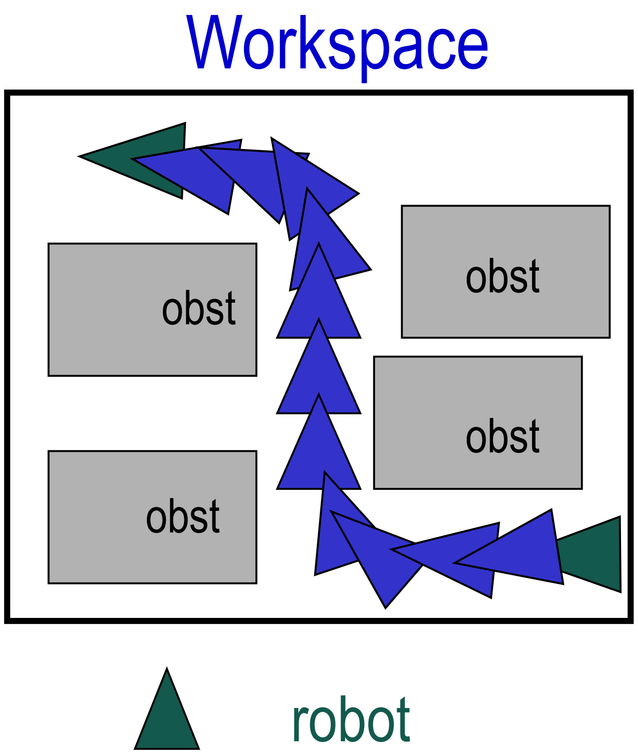
Randomized Graph Search

October 2, 2014

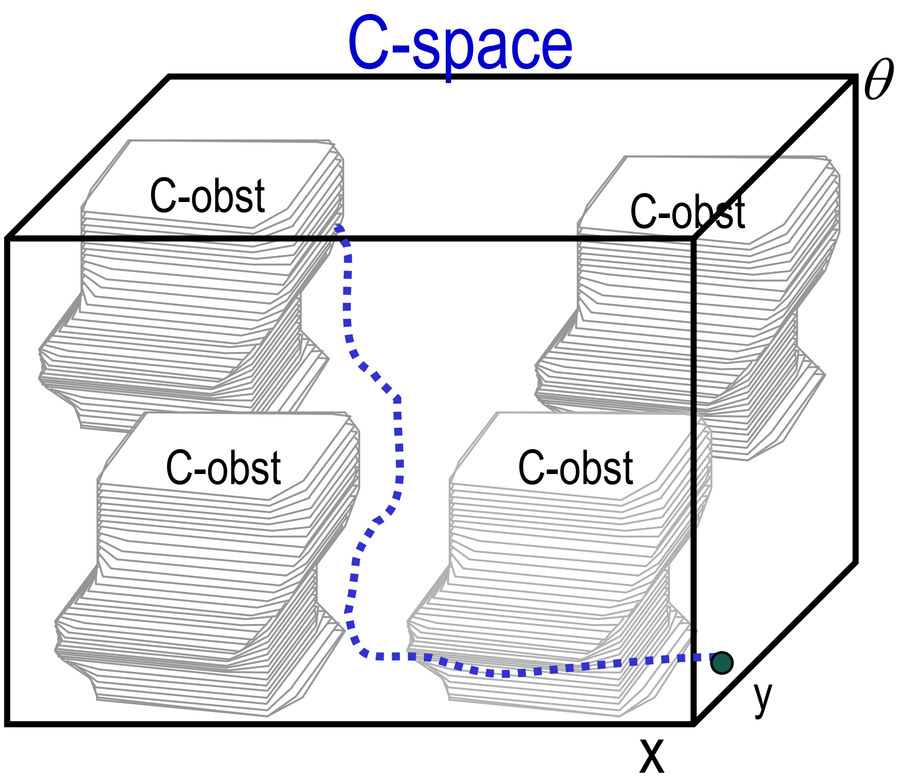
45 iterations 390 iterations

Motion Planning in C-space

Simple workspace obstacle transformed
into complicated C-obstacle!!



Path is swept volume



Path is 1D curve

(from Nancy Amato, Texas A&M Parasol Lab)

The Complexity of Motion Planning

Most motion planning problems of interest are PSPACE-hard
[Reif 79, Hopcroft et al. 84 & 86]

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 free space for more than 4 or 5 DOF

So … attention has turned to randomized algorithms which:

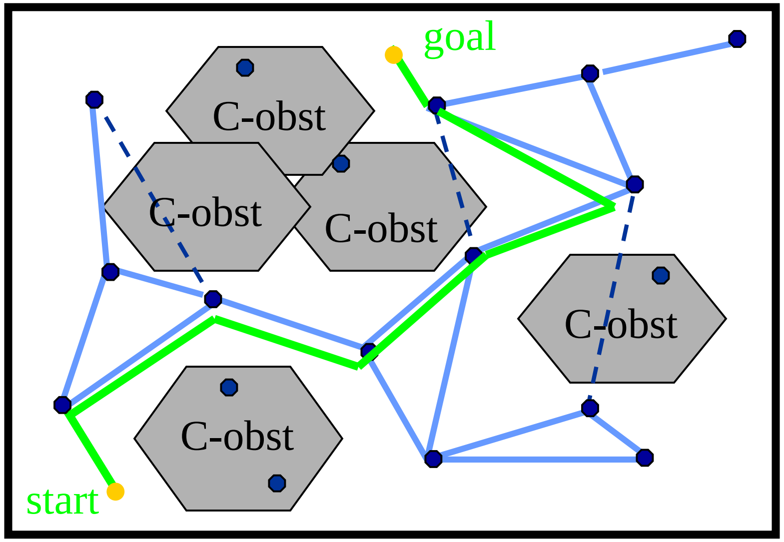
- trade full completeness of the planner
- for probabilistic completeness and a major gain in efficiency

(from Nancy Amato, Texas A&M Parasol Lab)

Probabilistic Roadmap Methods (PRMs)

[Kavraki, Svestka, Latombe,Overmars 1996]

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., straight line)
   - discard paths that are invalid

Query processing

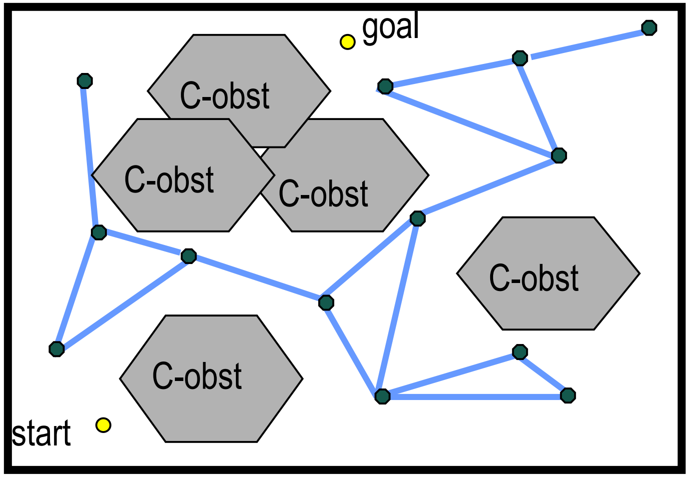
1. Connect start and goal to roadmap
2. Find path in roadmap between start and goal
   - regenerate plans for edges in roadmap

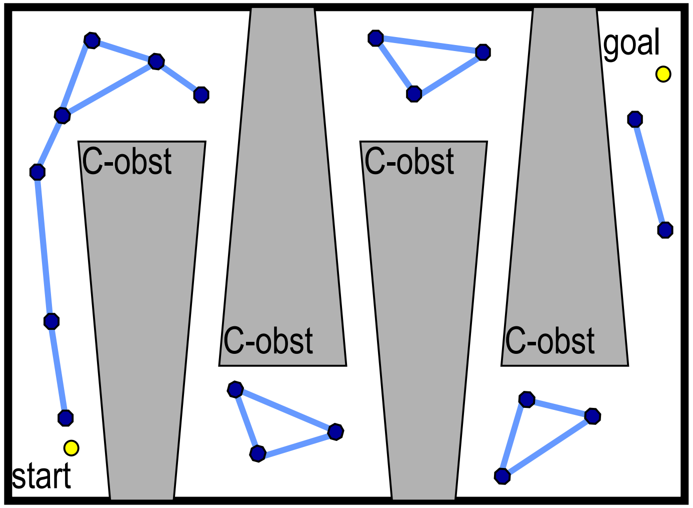
Primitives Required:

1. Method for Sampling points in C-Space
2. Method for `validating' points in C-Space

(from Nancy Amato, Texas A&M Parasol Lab)

PRMs: Pros & Cons





PRMs: The Good News

1. PRMs are probabilistically complete
2. PRMs apply easily to high-dimensional C-space
3. PRMs support fast queries w/ enough preprocessing

Many success stories where PRMs solve previously unsolved problems

PRMs: The Bad News

PRMs don't work as well for some problems:
– unlikely to sample nodes in narrow passages
– hard to sample/connect nodes on constraint surfaces

(from Nancy Amato, Texas A&M Parasol Lab)

Rapidly Exploring Random Trees (RRTs)

- Promoted by Steve Lavalle and James Kuffner
- Alternative to other randomized approaches
  - Probabilistic roadmap planner

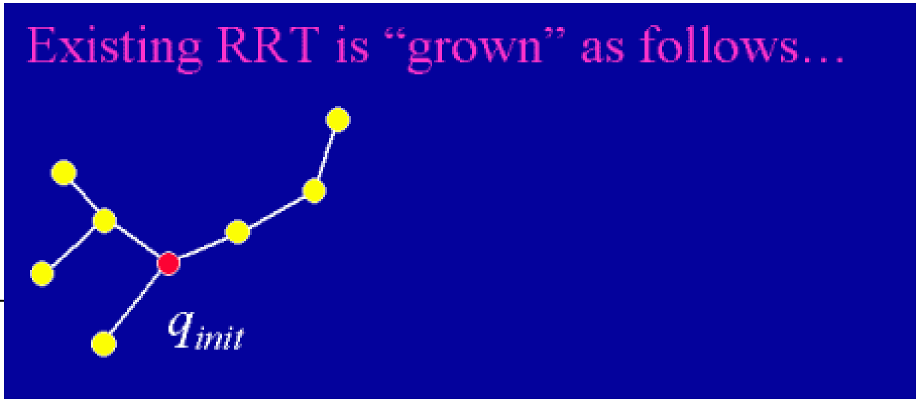
- RRT (rapidly exploring random trees) use Configuration Space
  - C : configuration space where q belongs to C and describes the position and orientation of a body place in the space.
  - C_{free} : set of configuration where the body does not collide with obstacles

The RRT Algorithm

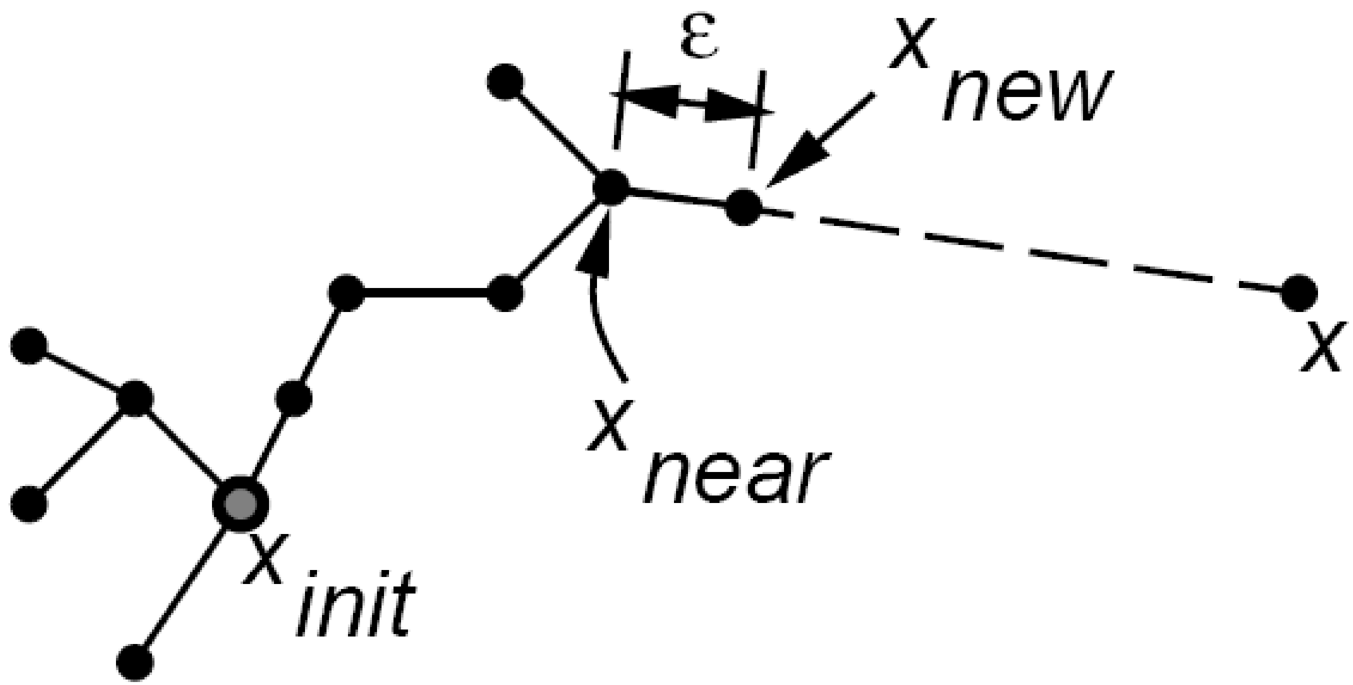
```
BUILD_RRT(q_init)
    T.init(q_init);
    for k = 1 to K do
        q_rand ← RANDOM_CONFIG();
        EXTEND(T, q_rand);
    Return T
```

```
EXTEND(T, q)
    q_near ← NEAREST_NEIGHBOR(q, T);
    if NEW_CONFIG(q, q_near, q_new) then
        T.add_vertex(q_new);
        T.add_edge(q_near, q_new);
        if q_new = q then
            Return Reached;
        else
            Return Advanced;
    Return Trapped;
```

- Start with initial (random) tree
- Select random configuration in free space
- The tree node closest to the selected random configuration is found
- An edge is "grown" toward the new configuration, taking into account the robot kinematic motion model

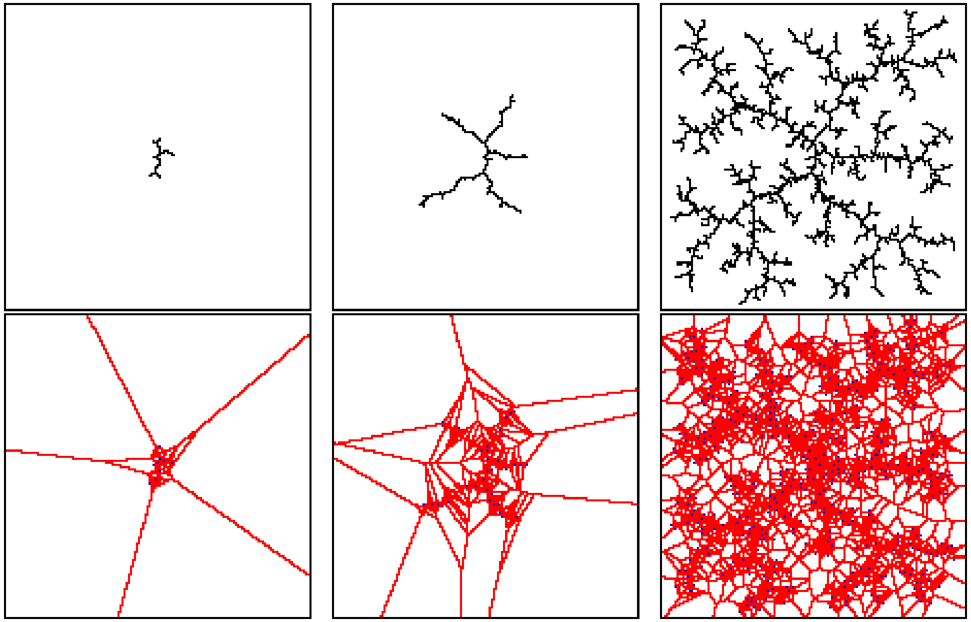


Basic Extend



Where is the “Rapid” in RRTs?

- Why are RRT’s rapidly exploring?
  - the probability of a node being selected for expansion is proportional to the area of its Voronoi region



- If just choose a vertex at random and extend, then it would act like random walk instead
  - Biased towards start vertex

Variation: RRT-Connect

- RRT-connect is a variation of RRT
  - grows two trees from both the source and destination until they meet
  - grows the trees towards each other (rather then towards random configurations)
  - the greediness becomes stronger by growing the tree with multiple epsilon steps instead of a single one

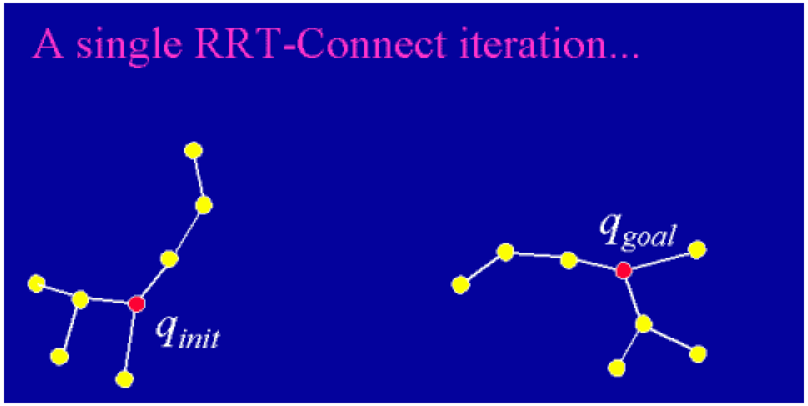
RRT-Connect Algorithm

CONNECT($\mathcal{T}, q$)

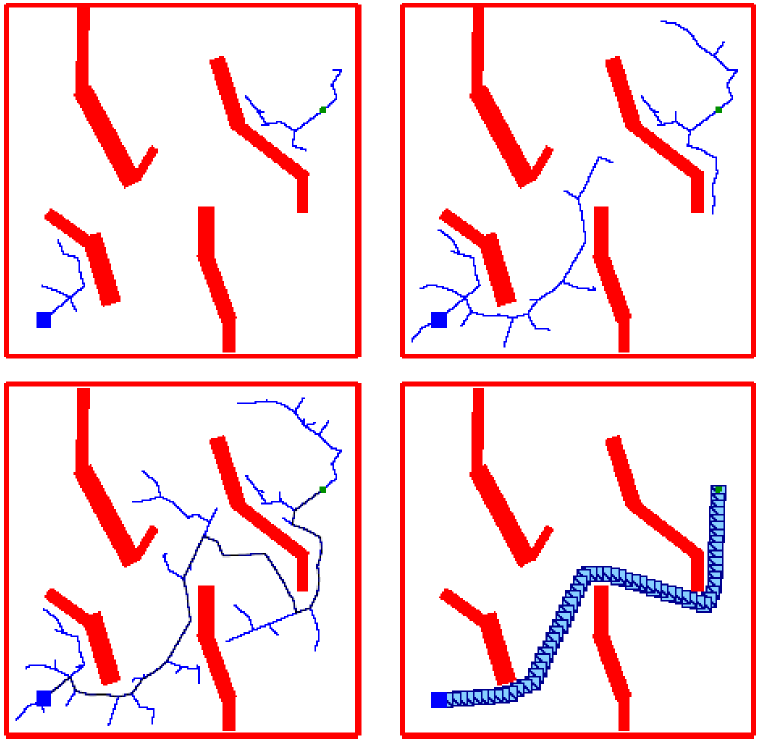
repeat
    $S \leftarrow$ EXTEND($\mathcal{T}, q$);
until not ($S =$ *Advanced*)
Return S;

RRT_CONNECT_PLANNER(q_{init}, q_{goal})

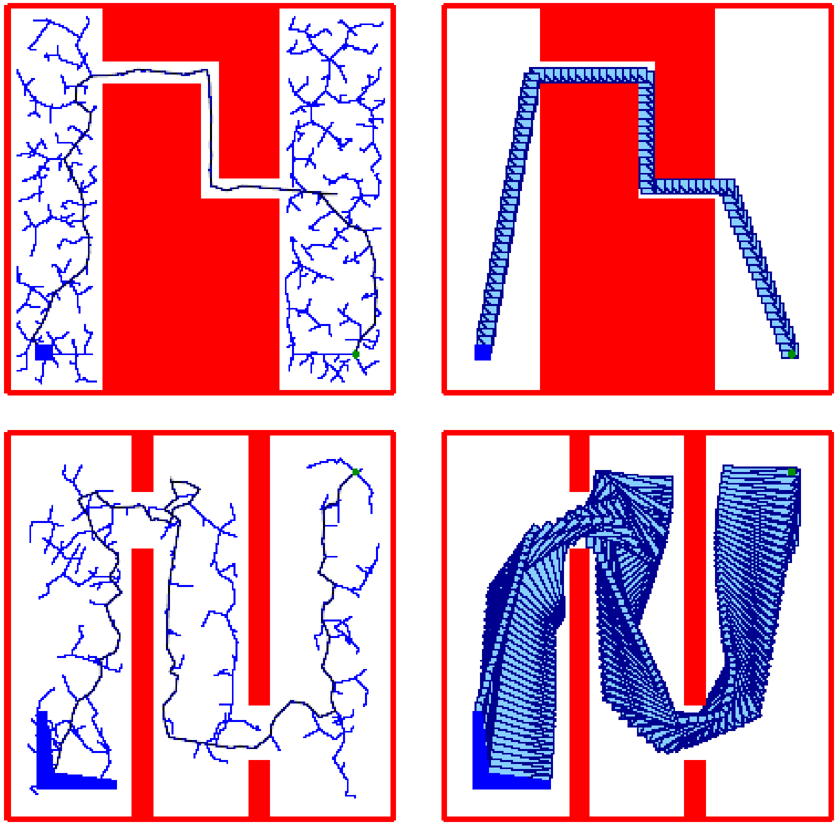
$\mathcal{T}_a$.init(q_{init}); $\mathcal{T}_b$.init(q_{goal});
for $k = 1$ **to** K **do**
    $q_{rand} \leftarrow$ RANDOM_CONFIG();
    if not (EXTEND($\mathcal{T}_a, q_{rand}$) $=$ *Trapped*) **then**
        if (CONNECT($\mathcal{T}_b, q_{new}$) $=$ *Reached*) **then**
            Return PATH($\mathcal{T}_a, \mathcal{T}_b$);
    SWAP($\mathcal{T}_a, \mathcal{T}_b$);
Return *Failure*



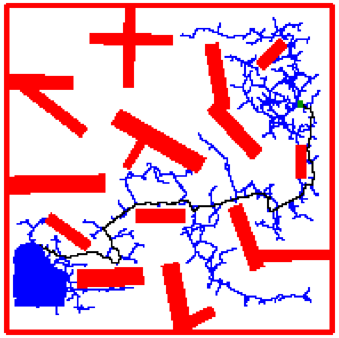
Examples

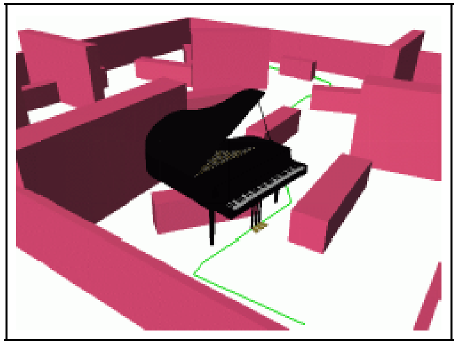


Examples

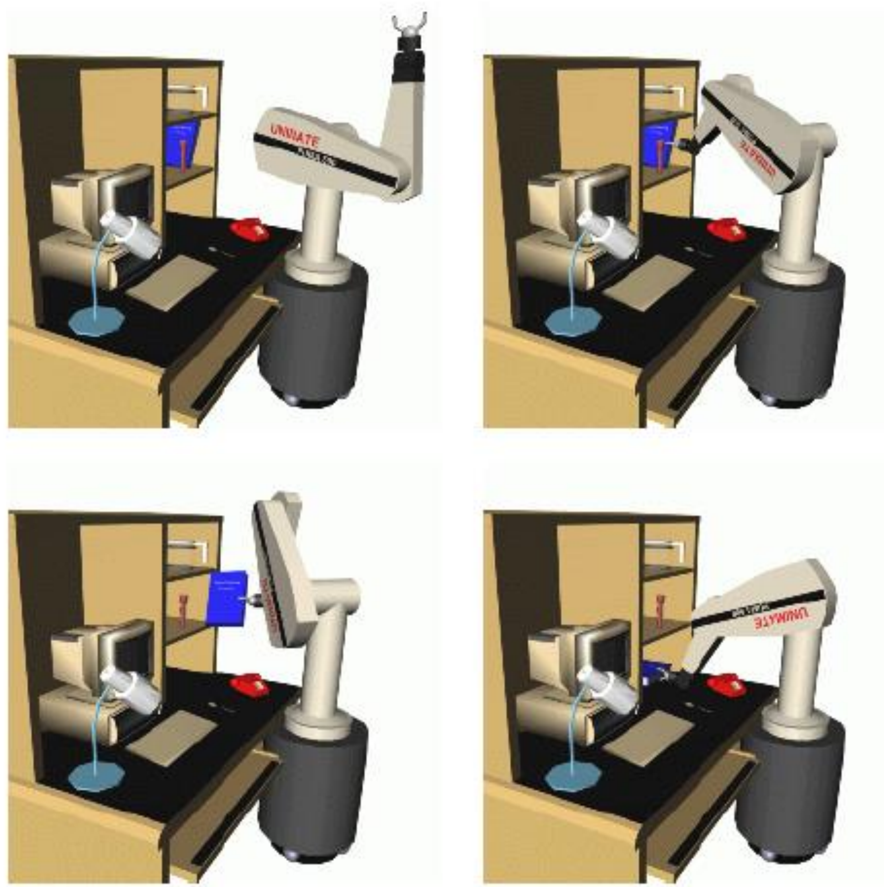


Examples





Examples



RRT-Connect Performance

- Much faster than common RRT methods for uncluttered environments and slightly faster in very cluttered environments
- 2D cases are solved in ≤ 1 second depending on the complexity of the situation
- 3D piano scene required 12 seconds
- 6 DOF robot arm required 4 seconds

RRT-Connect: Pros and Cons

- Improved version of RRT for faster convergence
- Finds paths in high dimensional spaces at interactive time rates
- Experiments showed it to be consistent
- Drawback: a lot of nearest neighbor searches are performed

Extension to Non-holonomic

- The new_state computation handles all the complicated part
- Given a state x and inputs u

$$\dot{x} = f(x, u)$$

- Integrate numerically to get new position

Examples

- http://msl.cs.uiuc.edu/rrt/gallery.html