RRT and Recent Advancements

Sung-Eui Yoon
(윤성의)

Course URL:
http://sglab.kaist.ac.kr/~sungeui/MPA

KAIST

Class Objectives

- **Understand the RRT technique and its recent advancements**

RRT-Connect: An Efficient Approach to Single-Query Path Planning

James Kuffner, Steven LaValle

ICRA 2000

# of citation: more 600

Initial slides from TaeJoon Kim

KAIST

Goal

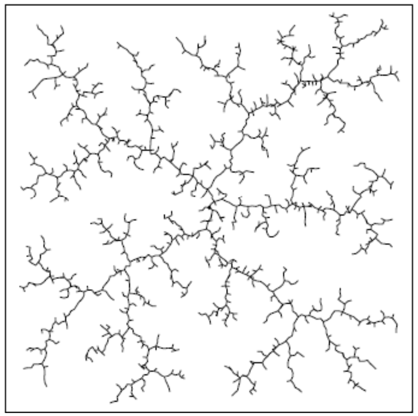
- Present an efficient randomized path planning algorithm for single-query problems
  - Converges quickly
  - Probabilistically complete
  - Works well in high-dimensional C-space

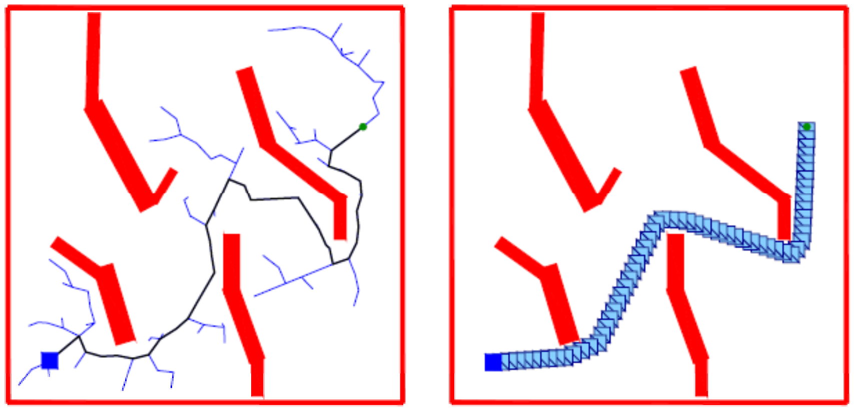
Motivation – Performance vs. Reliability

- Complete algorithms [Schwartz and M. Sharir 83, Canny 88]
  - Most reliable, needs high computational power
  - Only used to low-dimensional C-space
- Randomized potential field [Barraquand and Latombe 91]
  - Greedy & relaxation approach
  - Fast in many cases, but not in every case
- Probabilistic roadmap [Kavraki et al. 96]
  - Reliable, but needs preprocessing
  - Good for multiple-query problems

Approach

- **Design a simple, reliable, and fast algorithm for single-query problems**
  - **Use RRT (Rapidly-exploring Random Trees) [LaValle 98] for reliability**
  - **Develop a greedy heuristic to converge quickly**



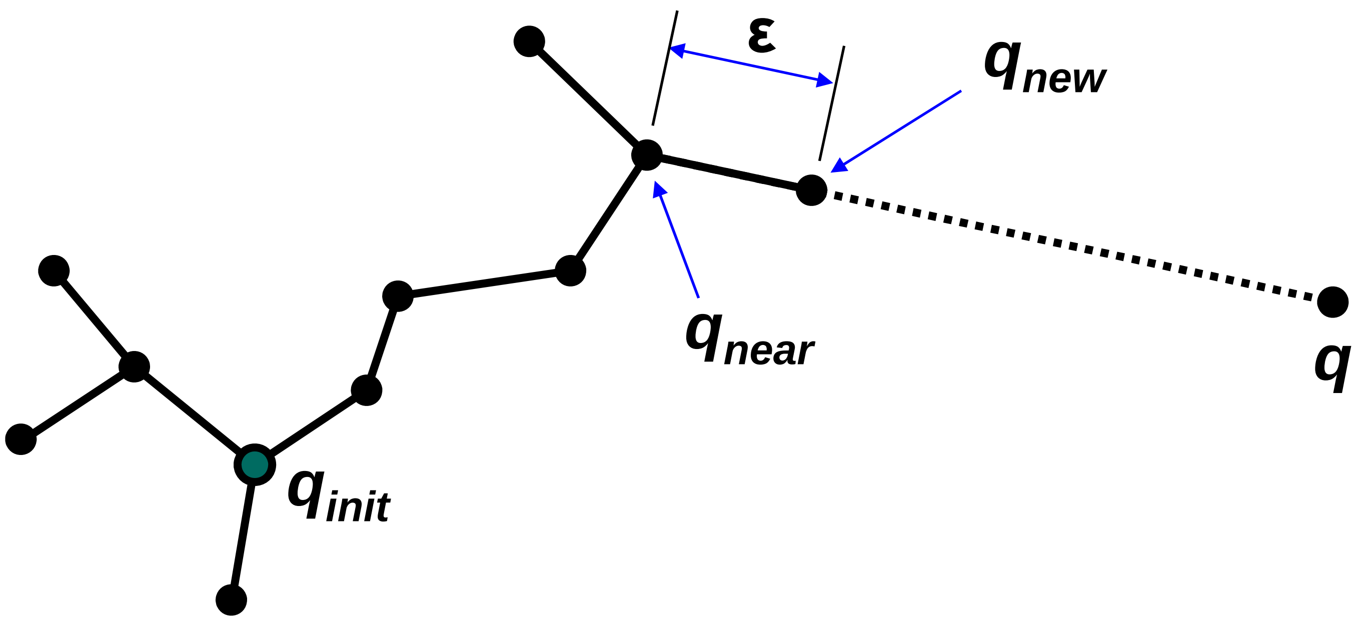


Rapidly-Exploring Random Tree

- **A growing tree from an initial state**

RRT Construction Algorithm

- **Extend a new vertex in each iteration**



Advantages of RRT

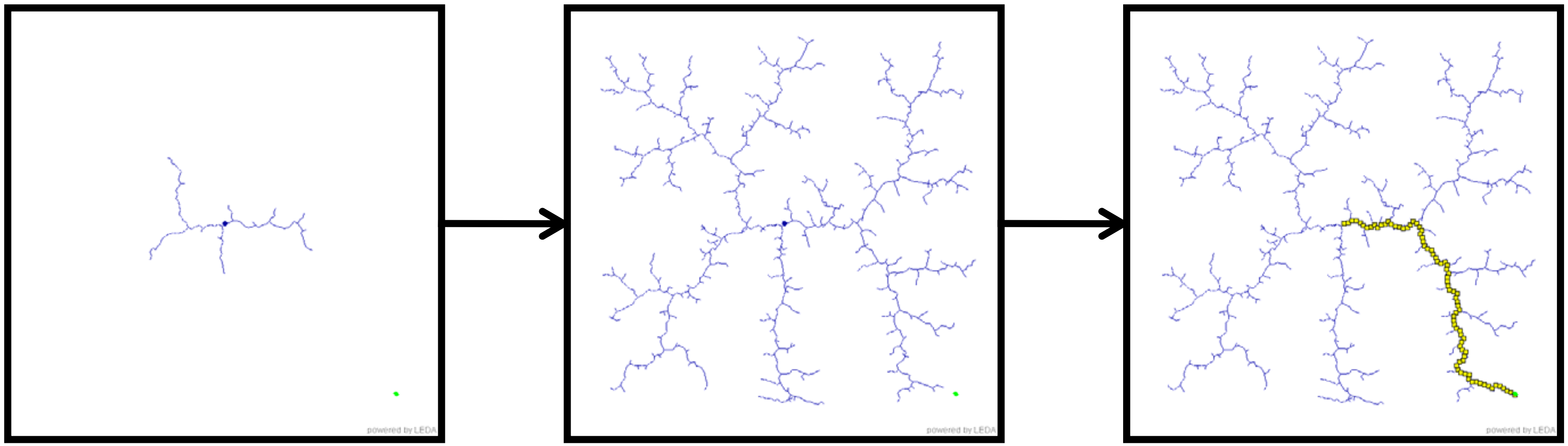
- Biased toward unexplored space
- Probabilistically complete
- Always connected
- Can handle nonholonomic constraints and high degrees of freedom

Outline

- Introduction
- Rapidly-exploring Random Tree
- Overview
- RRT-Connect Algorithm
- Demo
- Results
- Conclusion

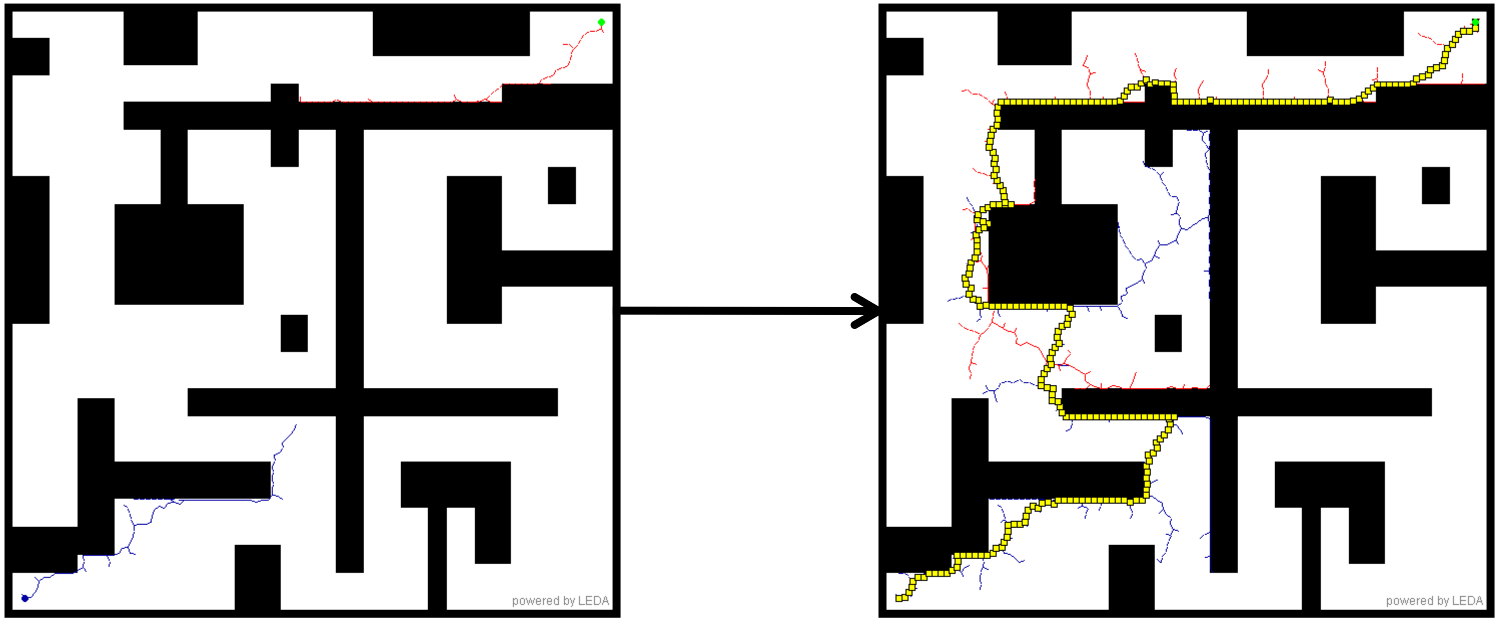
Overview – Planning with RRT

- Extend RRT until a nearest vertex is close enough to the goal state
- Probabilistically complete, but converge slowly



Overview – With Dual RRT

- Extend RRTs from both initial and goal states
- Find path much more quickly

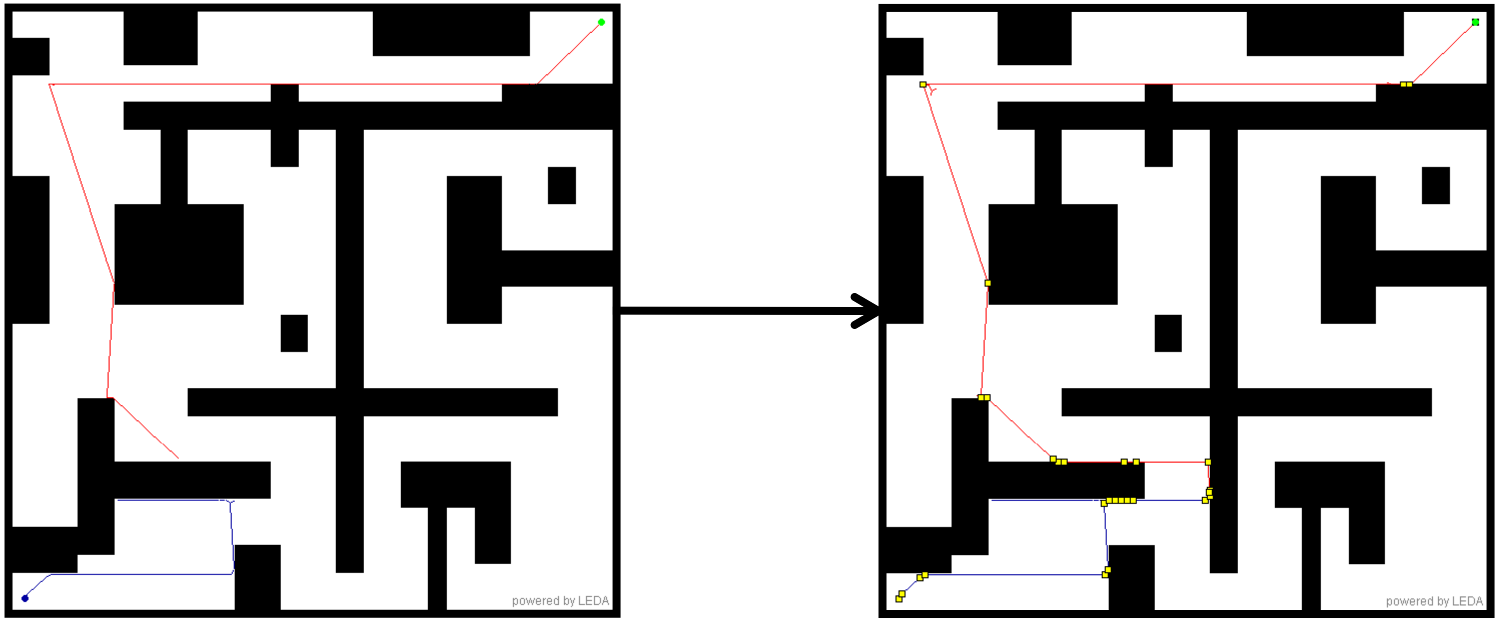


737 nodes are used

KAIST

Overview – With RRT-Connect

- **Aggressively connect the dual trees using a greedy heuristic**
- **Extend & connect trees alternatively**

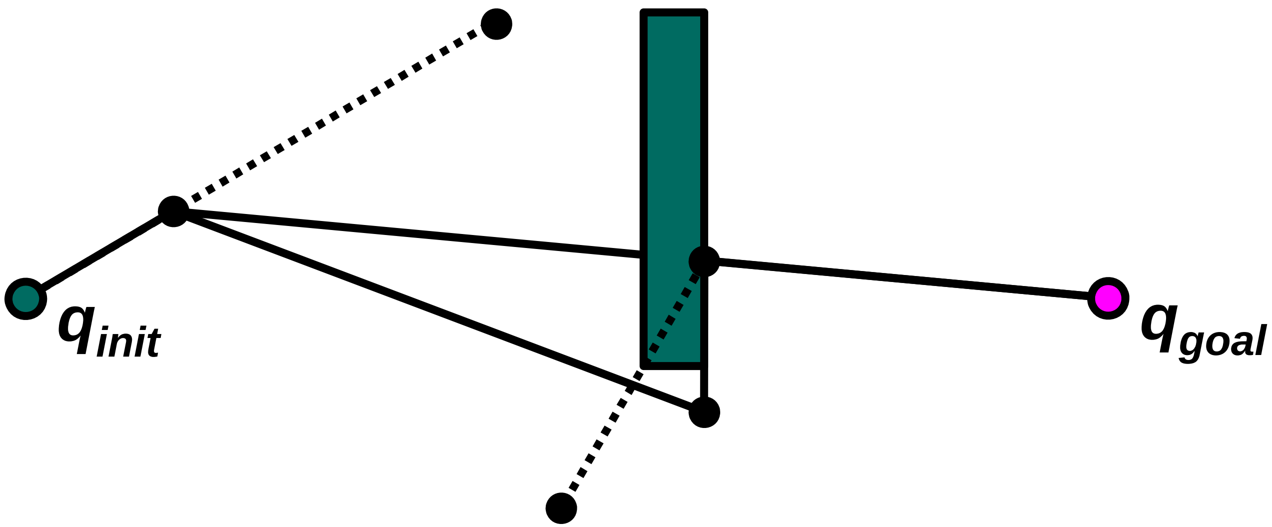


42 nodes are used

KAIST

RRT-Connect Algorithm

- Starting from both initial and goal states
- Extend a tree and try to connect the new vertex and another tree
- Alternatively repeat until two trees are actually connect

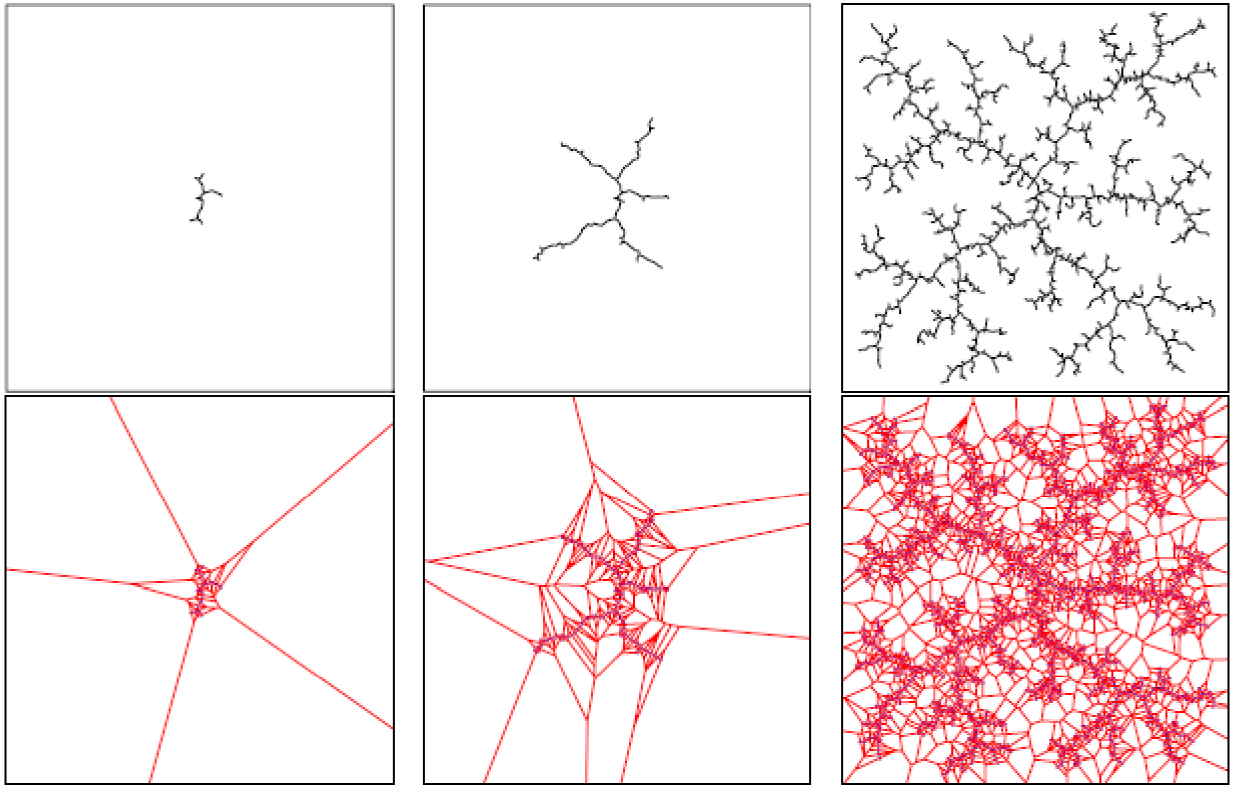


Variations of RRT-Connect

- Extend & Extend
  - Less aggressive, but works well on nonholonomic constrains
- Connect & Connect
  - Stronger greedy

Voronoi Region

- An RRT is biased by large Voronoi regions to rapidly explore, before uniformly covering the space



RRT Construction 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
```

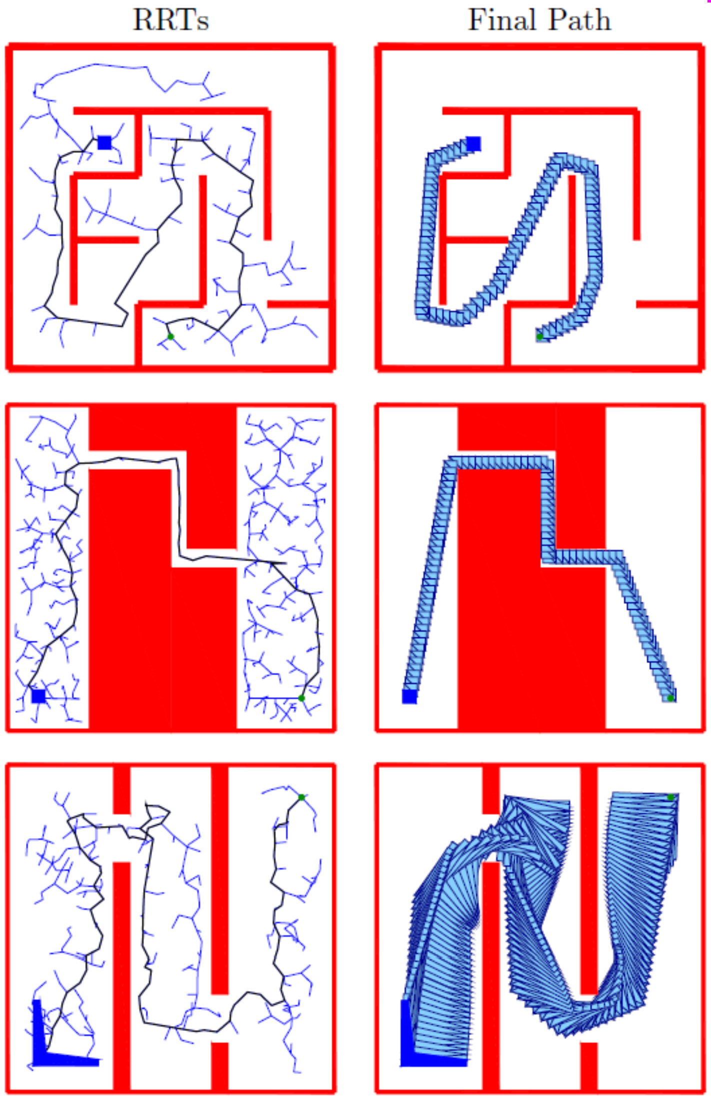
```
EXTEND(T, q)
    q_near ← NEAREST_NEIGHBOR(q, T);
    if NEW_CONFIG(q, q_near, q_new) then
        T.add_vertex(q_new);
        T.add_edge(q_near, q_new);
        if q_new = q then
            Return Reached;
        else
            Return Advanced;
    Return Trapped;
```

RRT Connect Algorithm

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

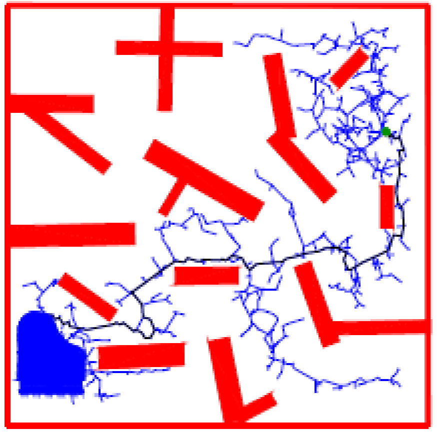
```
RRT_CONNECT_PLANNER(q_init, q_goal)
    T_a.init(q_init); T_b.init(q_goal);
    for k = 1 to K do
        q_rand ← RANDOM_CONFIG();
        if not (EXTEND(T_a, q_rand) = Trapped) then
            if (CONNECT(T_b, q_new) = Reached) then
                Return PATH(T_a, T_b);
        SWAP(T_a, T_b);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

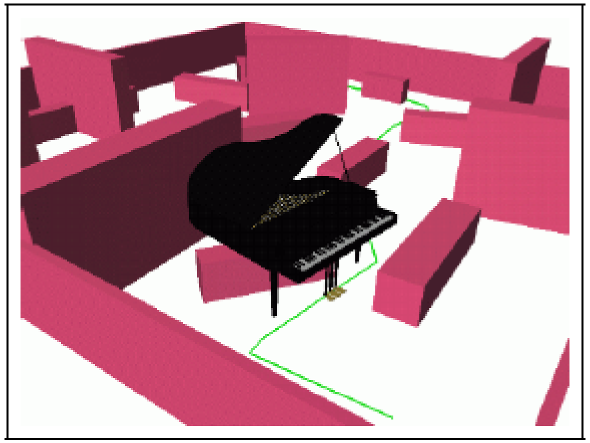
Results



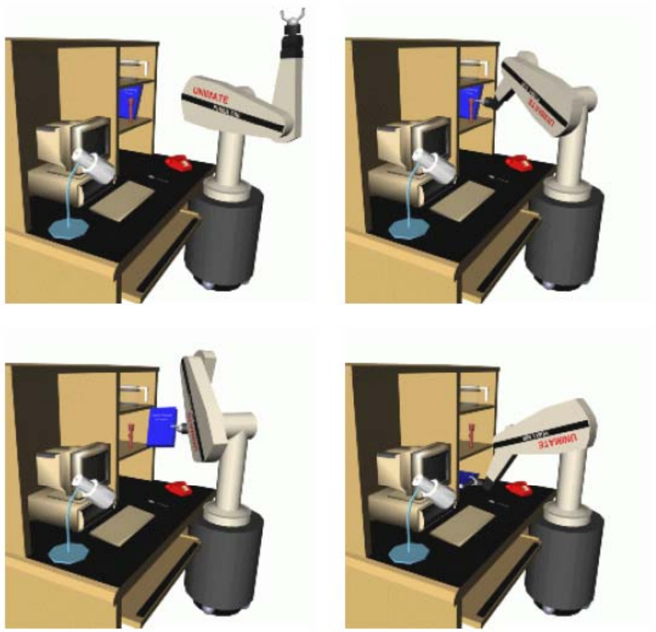
- 0.13s, 1.52s, and 1.02s on 270MHz
- Improves performance by a factor of three or four in uncluttered environments
- Slightly improves in very cluttered environments

Results





- Translations & rotations
- 12s



- 6-DOF
- 4s

Conclusions

- Reasonably balanced path planning between greedy exploration (as in a potential field) and uniform exploration (as in a probabilistic roadmap)
- Simple and practical method

- The huge performance improvements happen in relatively open spaces only
- Theoretical convergence ratio is not given

Randomized Kinodynamic Planning

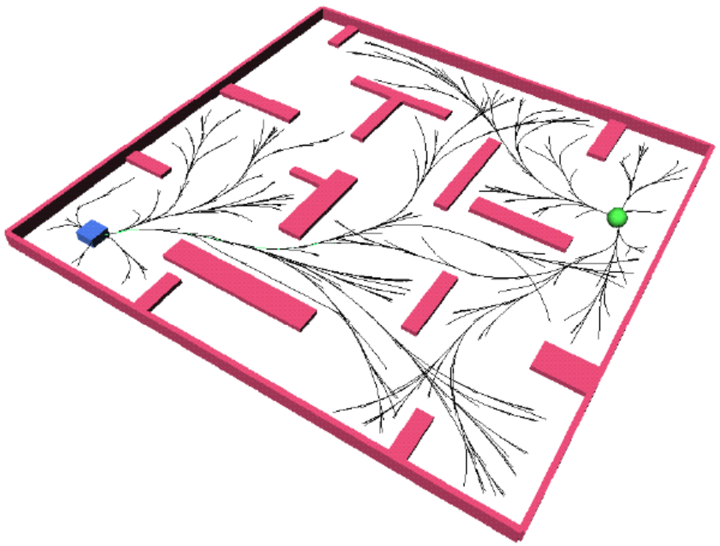
Steven LaValle   James Kuffner

ICRA 1999

# of citation: more than 400

Goal

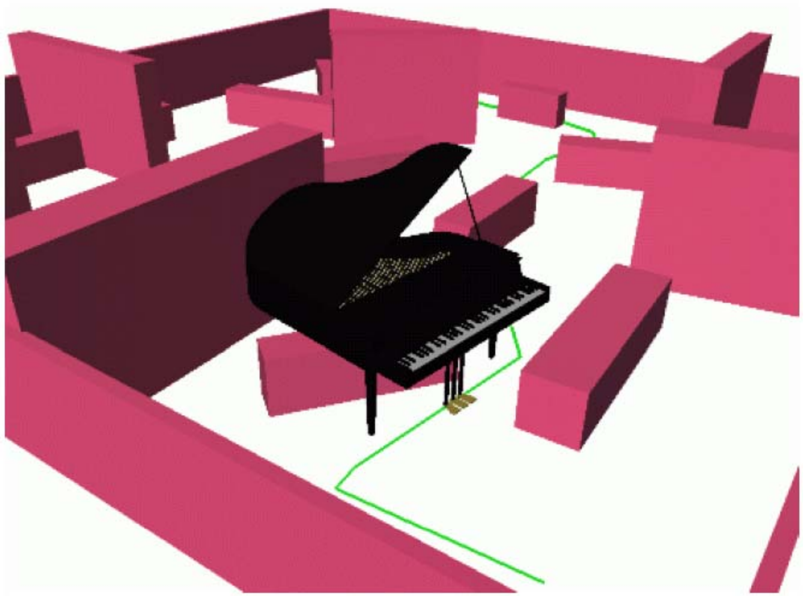
- **Present an efficient randomized path planning algorithm on the kinodynamic planning problem**

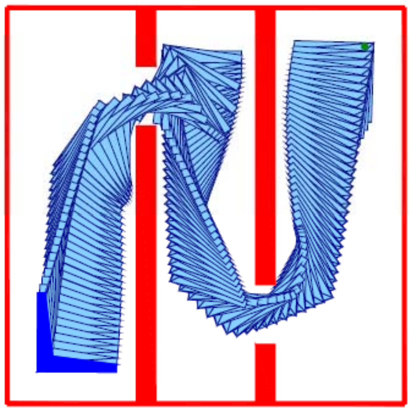


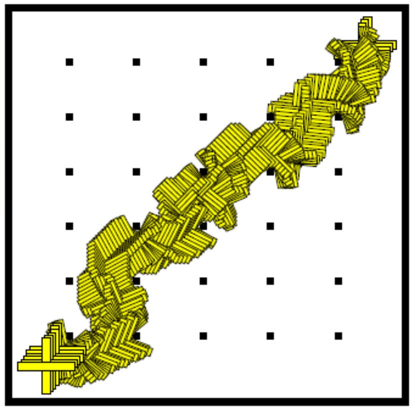
Holonomic Path Planning

GIVEN: $\mathcal{A}$ (robot), $\mathcal{C}$ (C-space), $f_i(q) \leq 0$ (obstacles), ...

FIND: Continuous path, τ, that satisfies $f_i(q)$ constraints





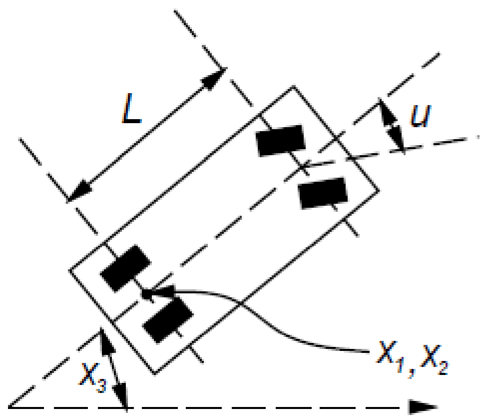


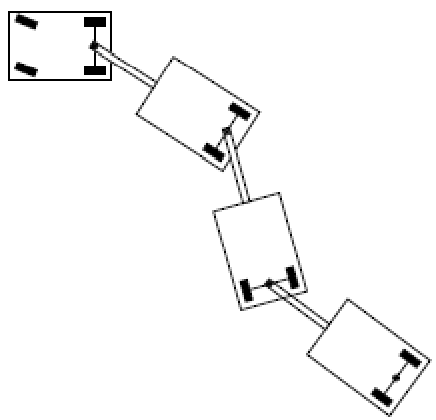
Nonholonomic Path Planning

ALSO GIVEN: $g_i(q, \dot{q}) \leq 0,\ g_i(q, \dot{q}) = 0,\ \ldots$

FIND: τ that satisfies $f_i(q),\ g_i(q, \dot{q})$ constraints

- **Consider kinematic constraints**



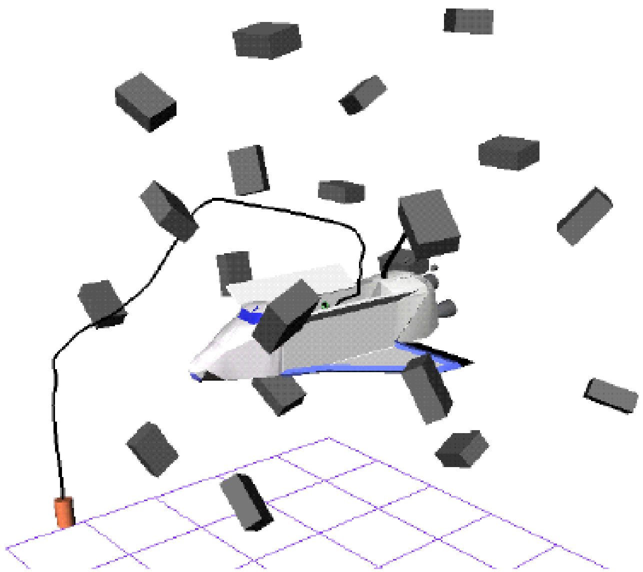


Kinodynamic Path Planning

ALSO GIVEN: $h_i(q, \dot{q}, \ddot{q}) \leq 0,\ h_i(q, \dot{q}, \ddot{q}) = 0,\ \ldots$

FIND: τ that satisfies $f_i(q),\ g_i(q, \dot{q}),\ h_i(q, \dot{q}, \ddot{q})$

- **Consider kinematic + dynamic constraints**



KAIST

Kinodynamic Path Planning

- Conventional planning: Decouple problems
  - Solve basic path planning
  - Find trajectory and controller that satisfies the dynamics and follows the path
  - [Bobrow et al. 85, Latombe 91, Shiller and Dubowsky 91]
- PSPACE-hard in general [Reif 79]

Outline

- Introduction
- Kinodynamic Planning
- Problem Formulation
- Randomized Kinodynamic Planning
- Rapidly-Exploring Random Trees (RRTs)
- Demo
- Results
- Conclusion

KAIST

State Space Formulation

- **Kinodynamic planning → 2n-dimensional state space**

  C denote the C-space

  X denote the state space

$$x = (q, \dot{q}), \text{ for } q \in C, x \in X$$

$$x = [q_1 \;\; q_2 \;\; \dots \;\; q_n \;\; \frac{dq_1}{dt} \;\; \frac{dq_2}{dt} \;\; \dots \;\; \frac{dq_n}{dt}]$$

Constraints in State Space

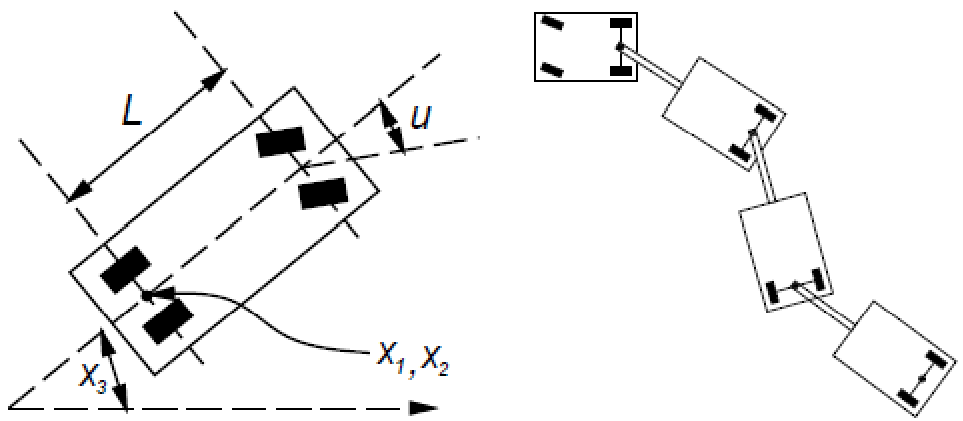
$$h_i(q,\dot{q},\ddot{q}) = 0 \text{ becomes } G_i(x,\dot{x}) = 0,$$

$$\text{for } i = 1,\ldots,m \text{ and } m < 2n$$

- **Constraints can be written in:**

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

$$u \in U, \quad U : \text{Set of allowable controls or inputs}$$



Solution Trajectory

- **Defined as a time-parameterized continuous path**

$$\tau : [0, T] \rightarrow X_{free}, \text{ satisfies the constraints}$$

- **Obtained by integrating** $\dot{x} = f(x, u)$
- **Solution: Finding a control function**

$$u : [0, T] \rightarrow U$$

Randomized Kinodynamic Planning

- Randomized potential fields
  - [Barraquand and Latombe 91, Challou et al. 95]
  - Set u which reduces the potential
  - Leads oscillations
  - Hard to design good potential fields
- Randomized roadmap
  - [Amato and Wu 96, Kavraki et al. 96]
  - Hard to connect two configurations (or states), except for specific environments [Svestka and Overmars 95, Reeds and Schepp 90, Bushnell et al. 95…]

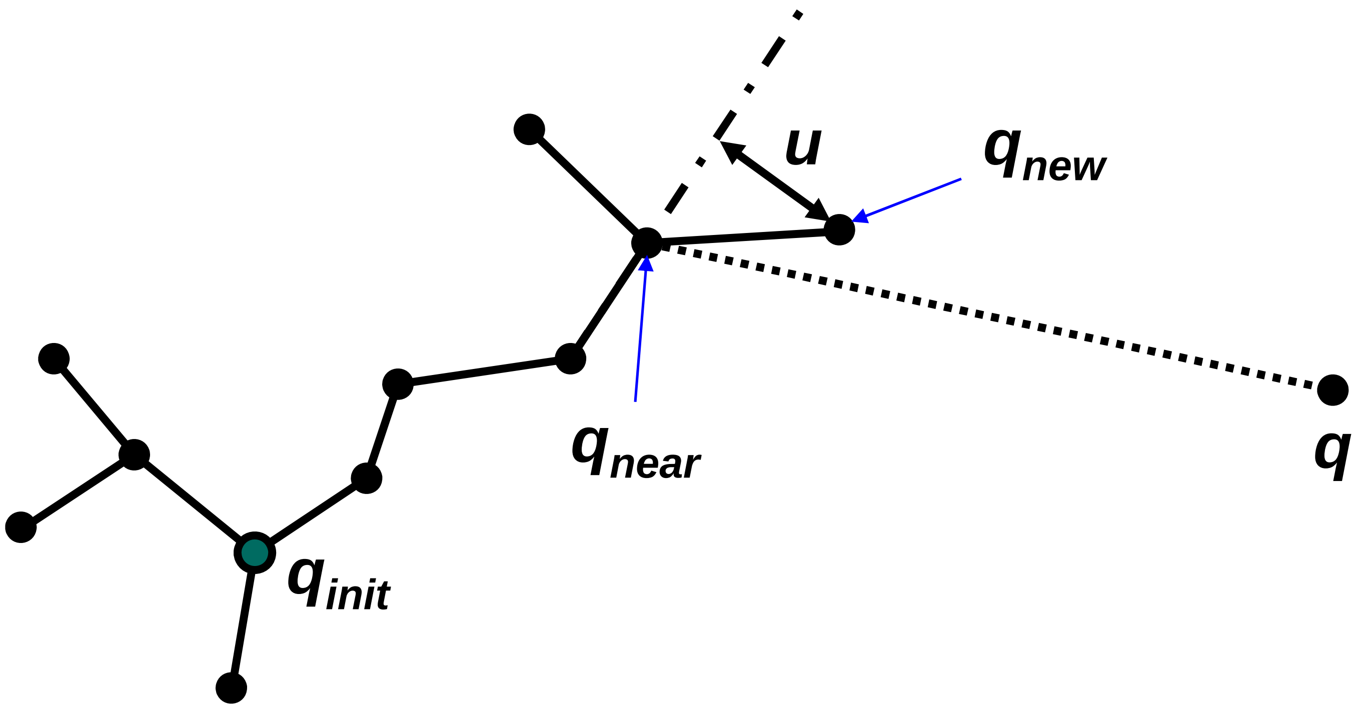
KAIST

Rapidly-Exploring Random Tree

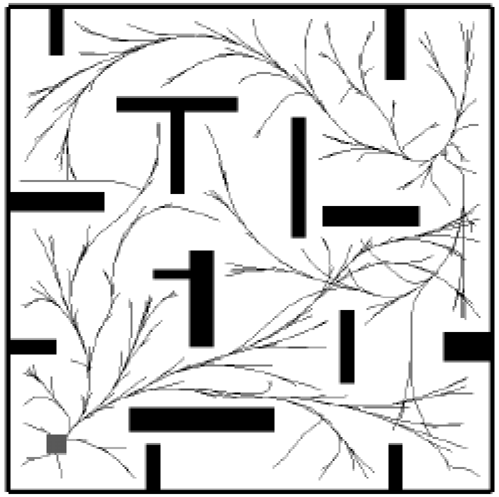
- A growing tree from initial state

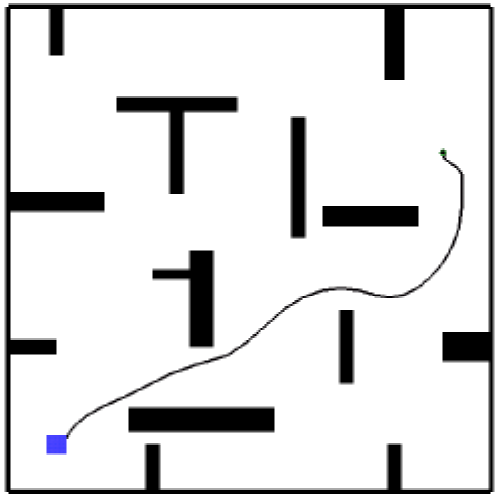
Rapidly-Exploring Random Tree

- **Extend a new vertex in each iteration**

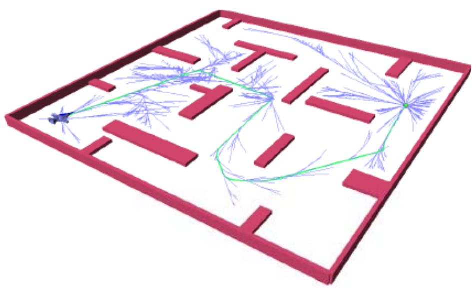


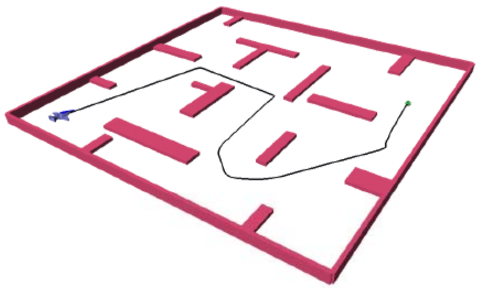
Results – 200MHz, 128MB





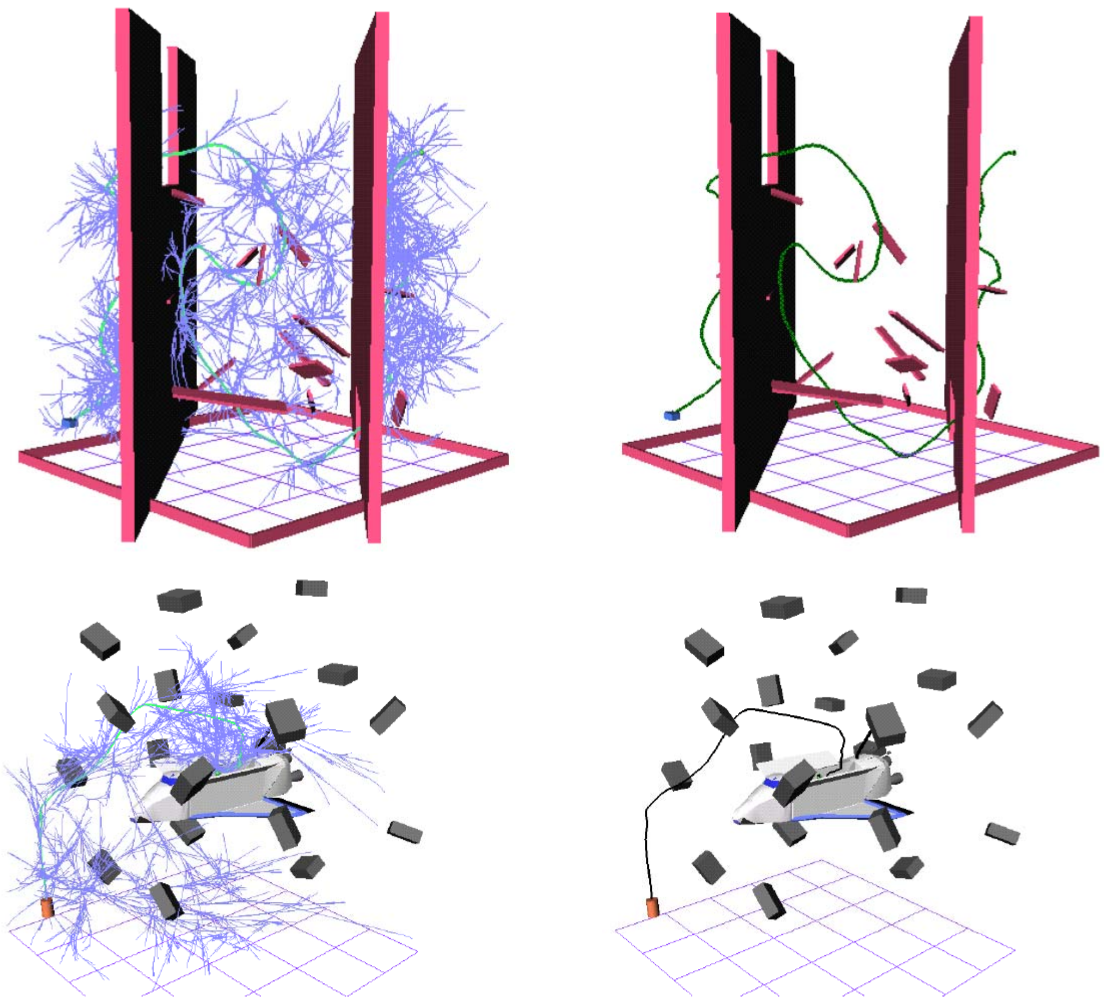
- Planar translating
  - X=4 DOF
- Four different controls: up, down, left, right forces
- 500~2,500 nodes
- 5~15sec





- Planar TR+RO
- X=6 DOF
- 13,600 nodes
- 4.2min

Results – 200MHz, 128MB



- 3D translating
- X=6 DOF
- 16,300 nodes
- 4.1min

- 3D TR+RO
- X=12 DOF
- 23,800 nodes
- 8.4min

Conclusions

- Take advantages from both randomized potential fields and roadmaps
  - “Drives forward” like potential fields
  - Quickly and uniformly explores like roadmaps
- Efficient and reliable method
  - Practical!

Dynamic-Domain RRTs: Efficient Exploration by Controlling the Sampling Domain

Anna Yershova Léonard Jaillet Thierry Siméon Steven M. LaValle

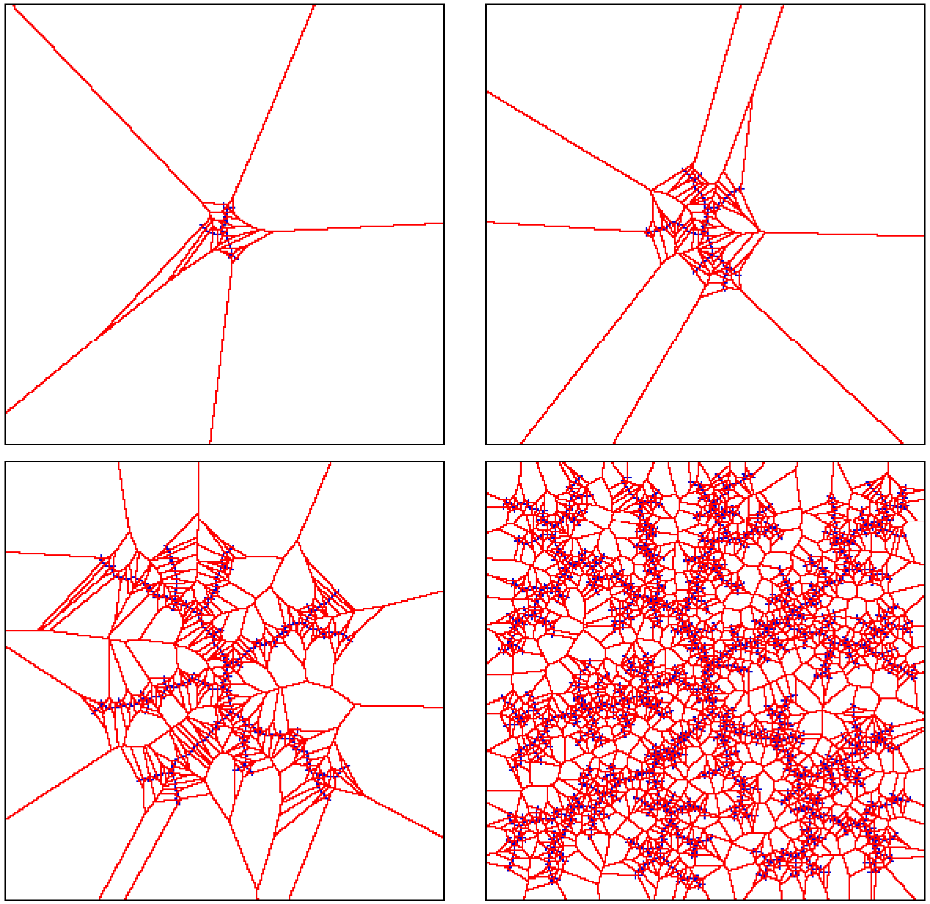
ICRA 05

Citation: more than 80

From author's slides

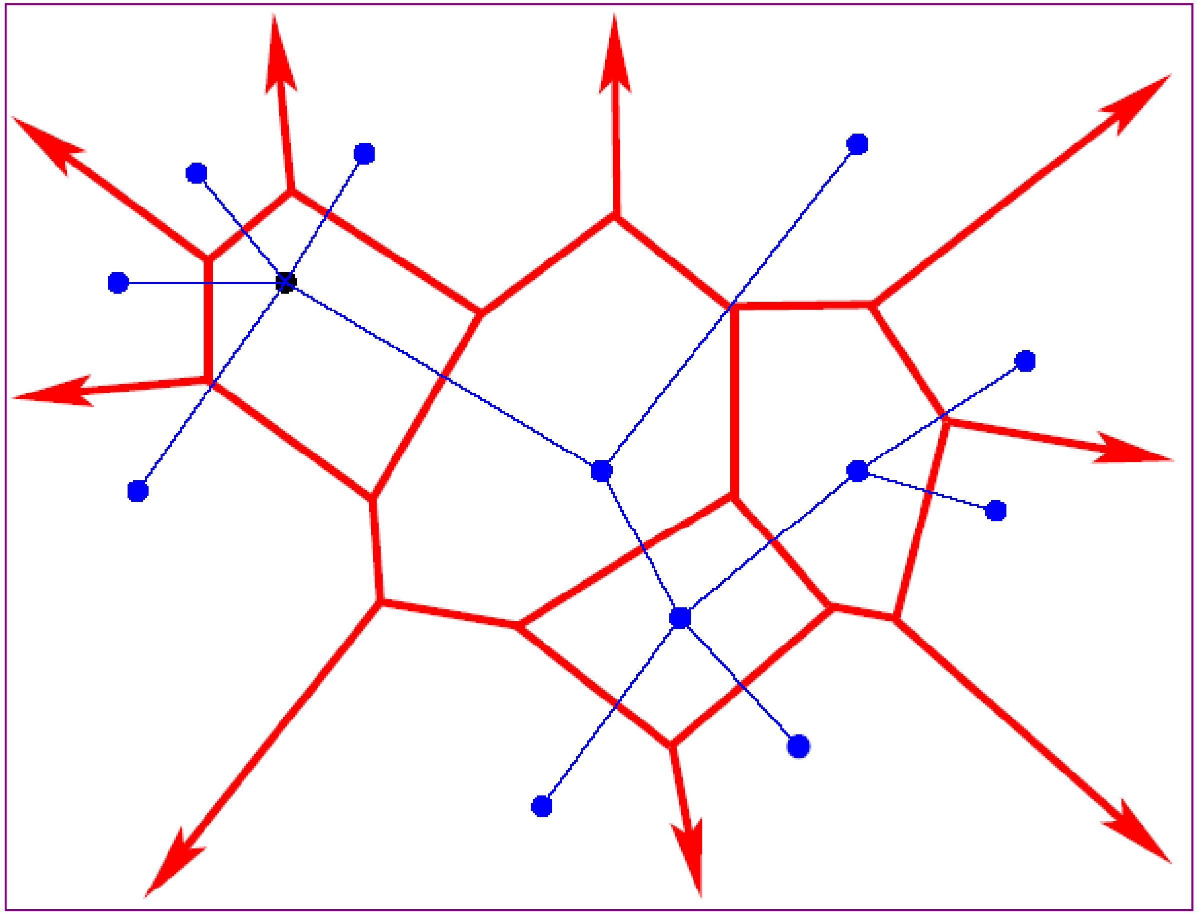
A Rapidly-exploring Random Tree (RRT)

Voronoi Biased Exploration

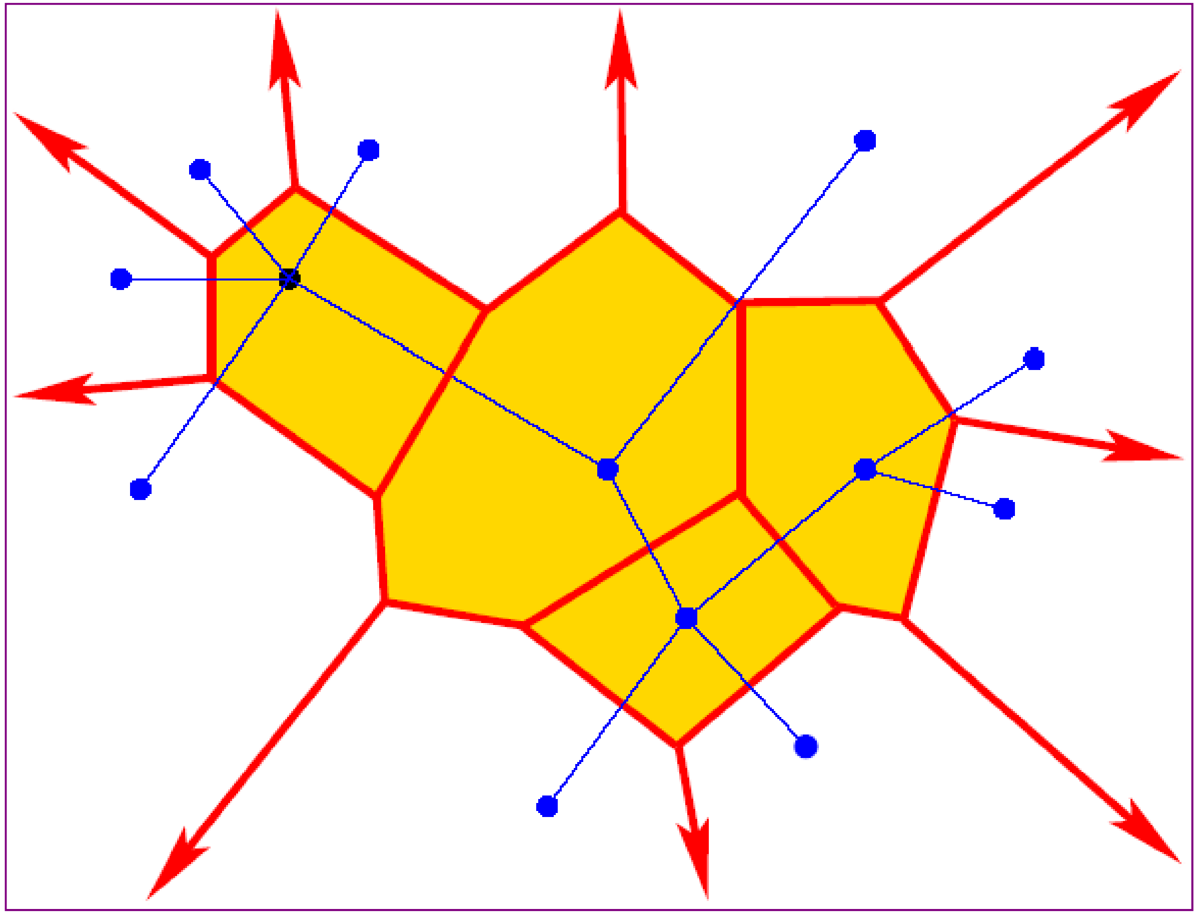


Is this always a good idea?

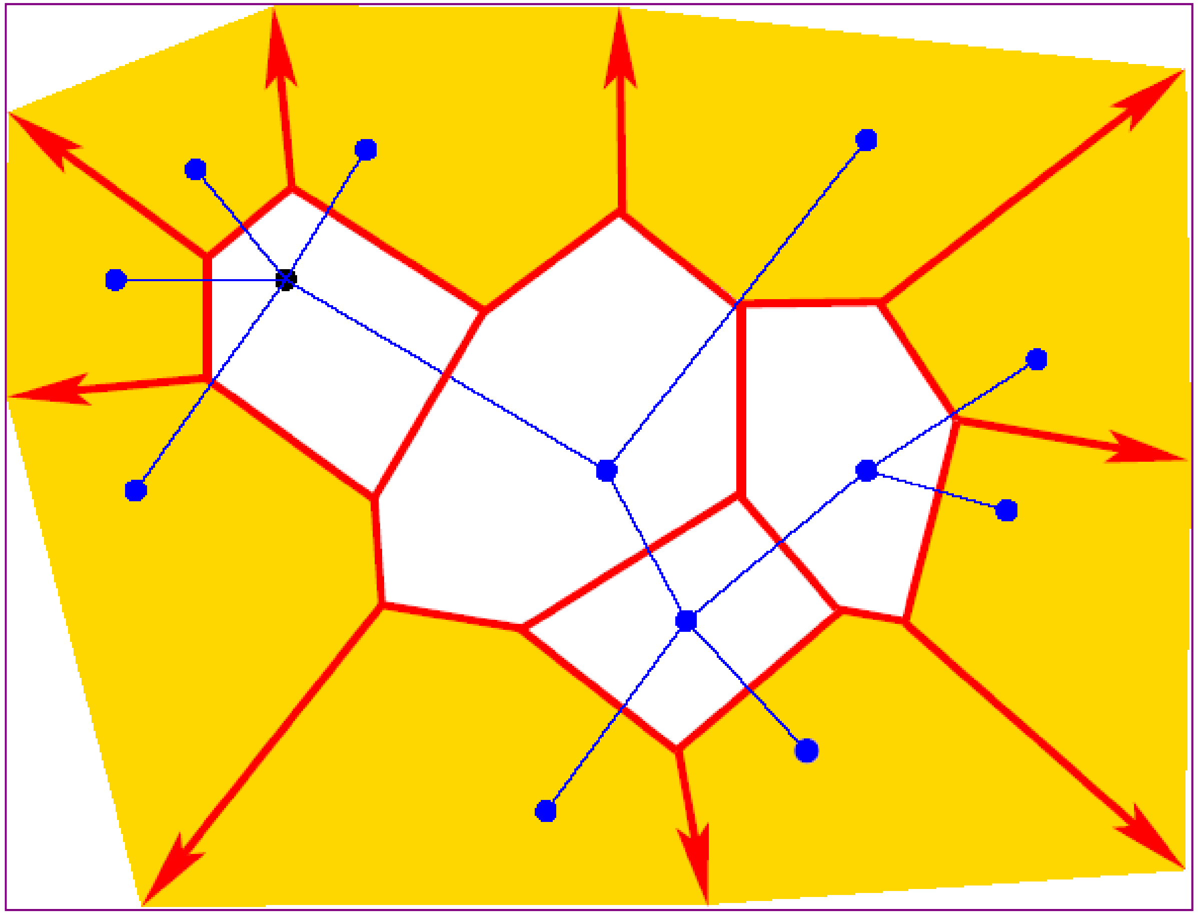
Voronoi Diagram in R^2



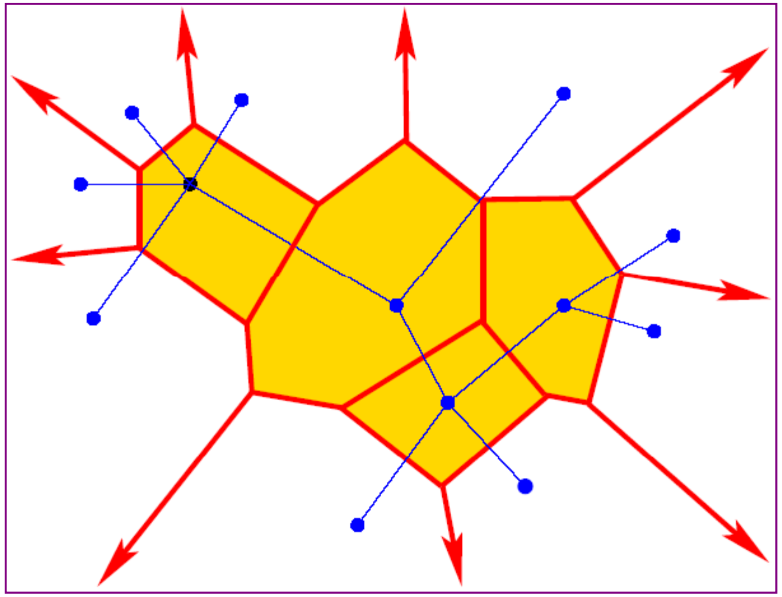
Voronoi Diagram in R^2



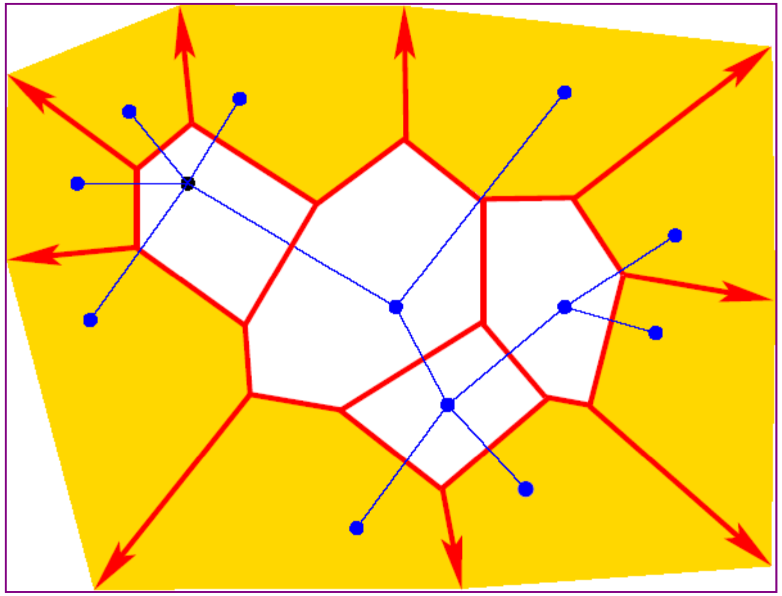
Voronoi Diagram in R^2



Refinement vs. Expansion



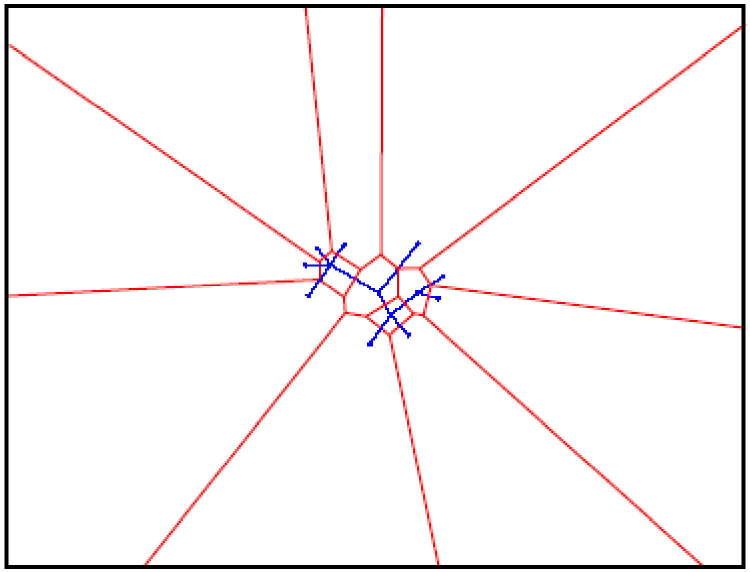
refinement



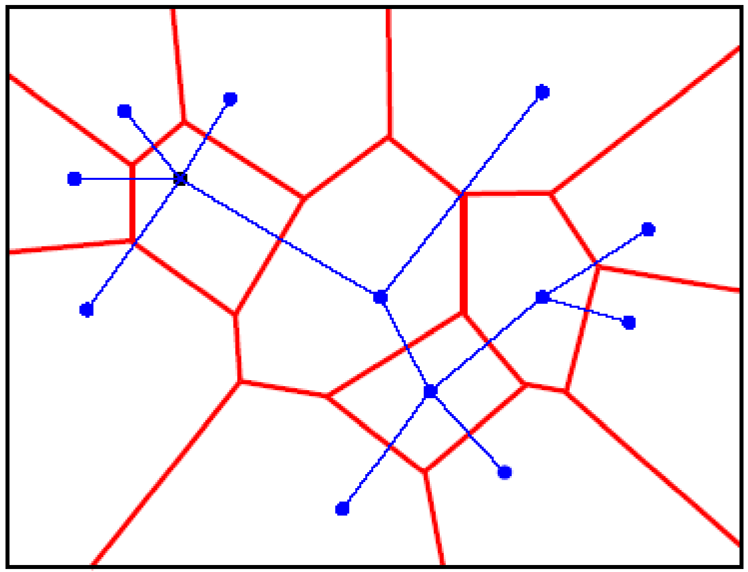
expansion

Where will the random sample fall? How to control the behavior of RRT?

Determining the Boundary



Expansion dominates



Balanced refinement and expansion

The tradeoff depends on the size of the bounding box

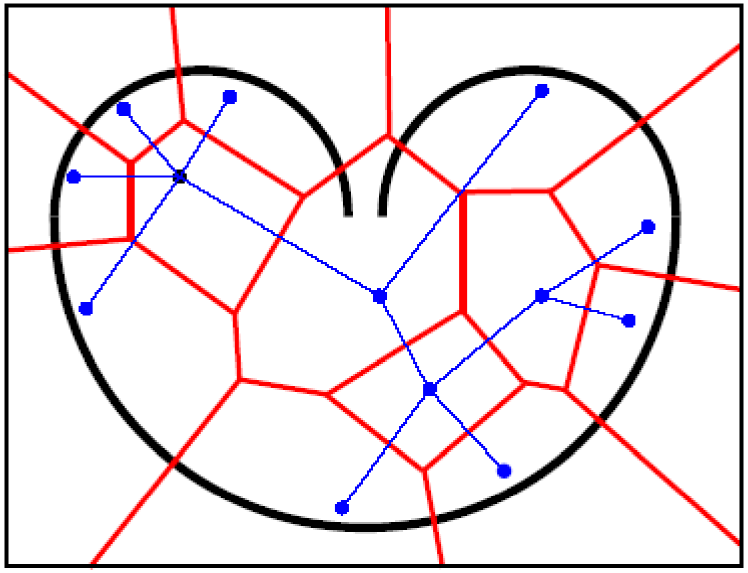
Controlling the Voronoi Bias

- Refinement is good when multiresolution search is needed
- Expansion is good when the tree can grow and not blocked by obstacles

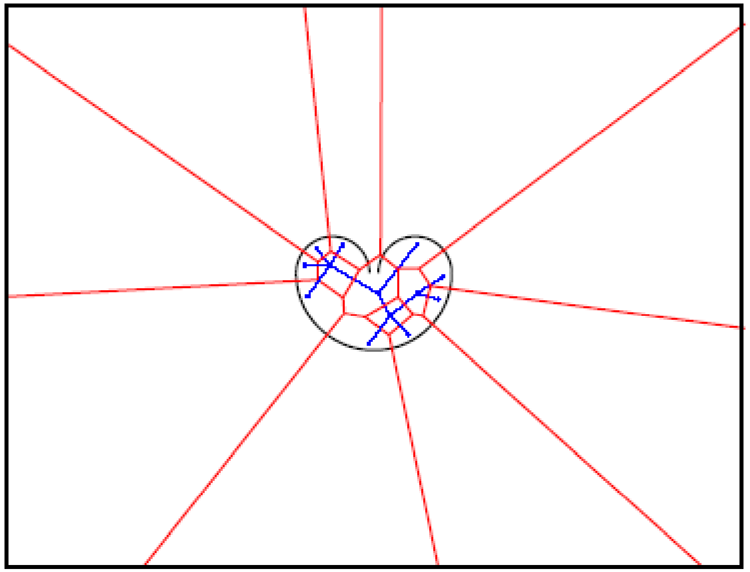
Main motivation:

- Voronoi bias does not take into account obstacles
- How to incorporate the obstacles into Voronoi bias?

Bug Trap



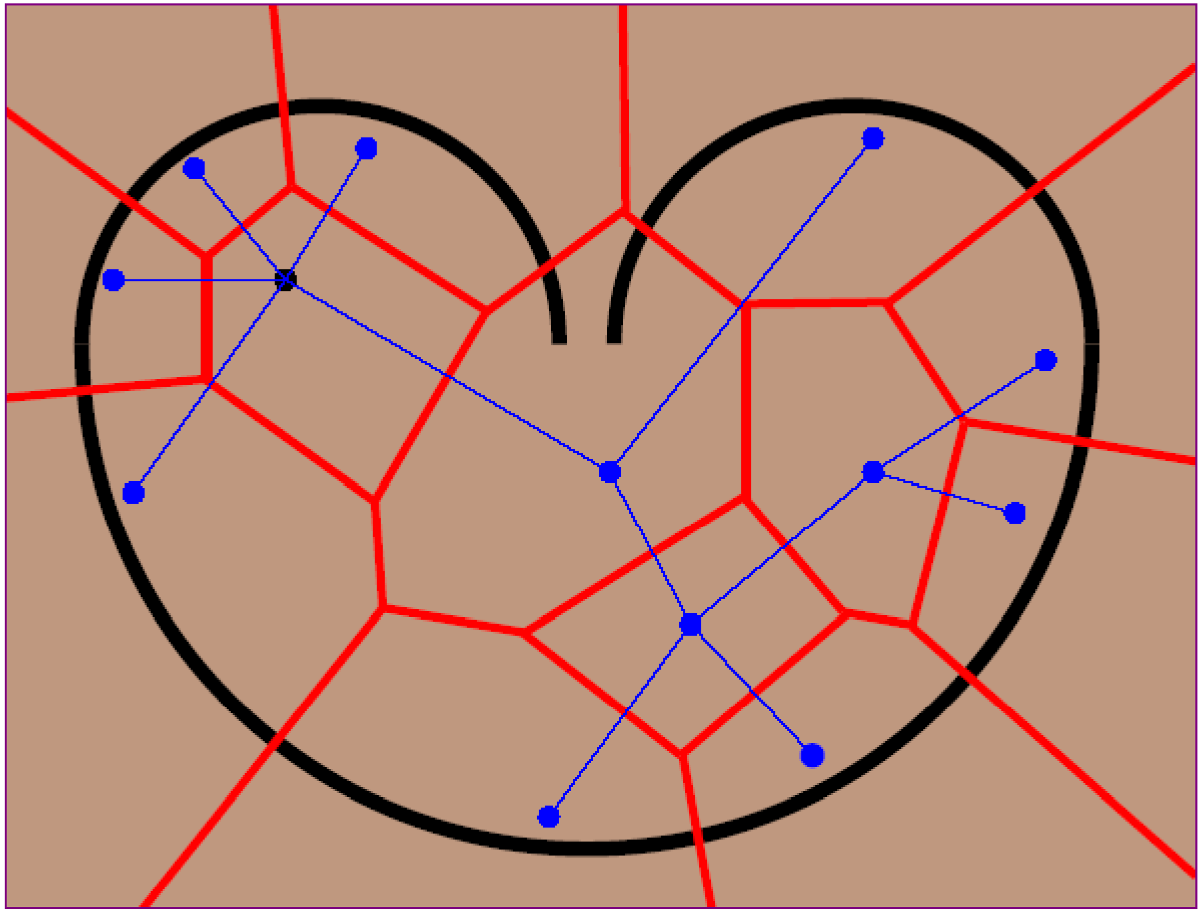
Small Bounding Box



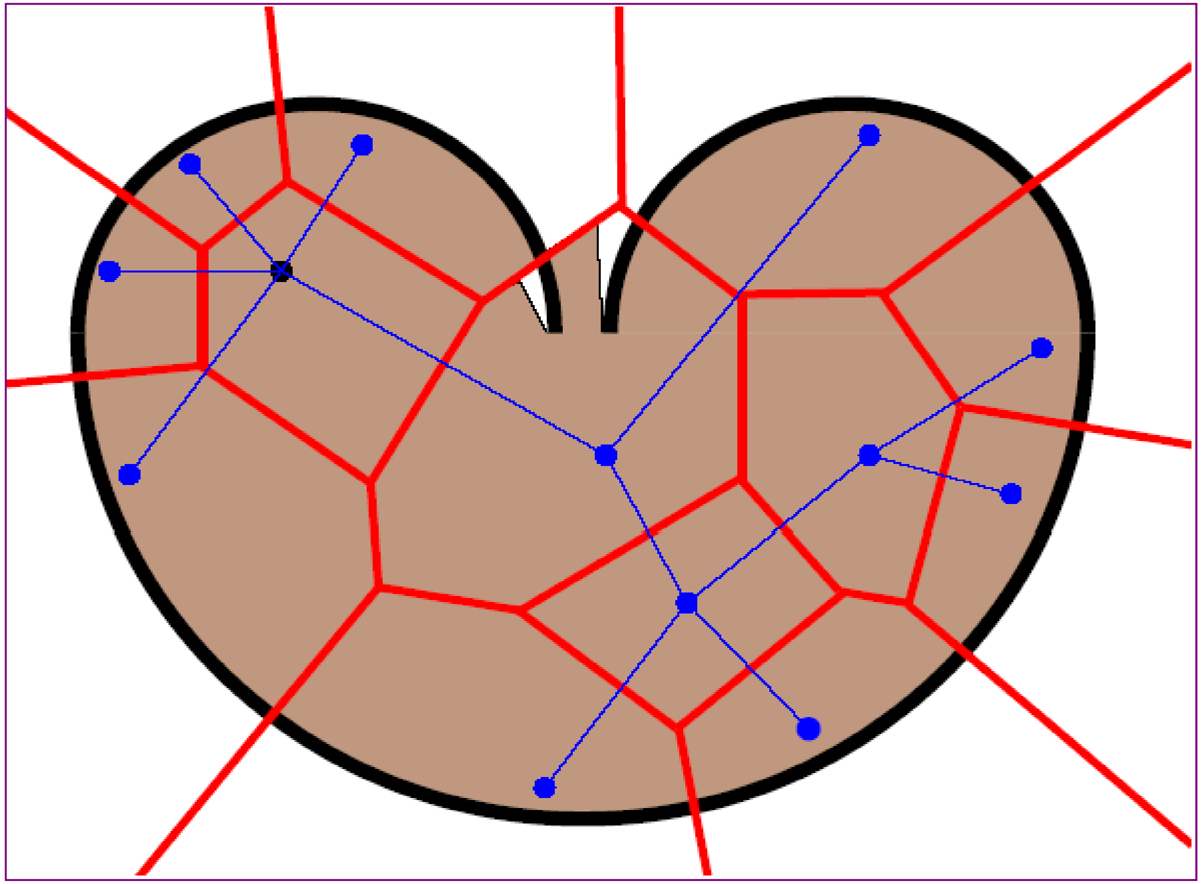
Large Bounding Box

Which one will perform better?

Voronoi Bias for the Original RRT

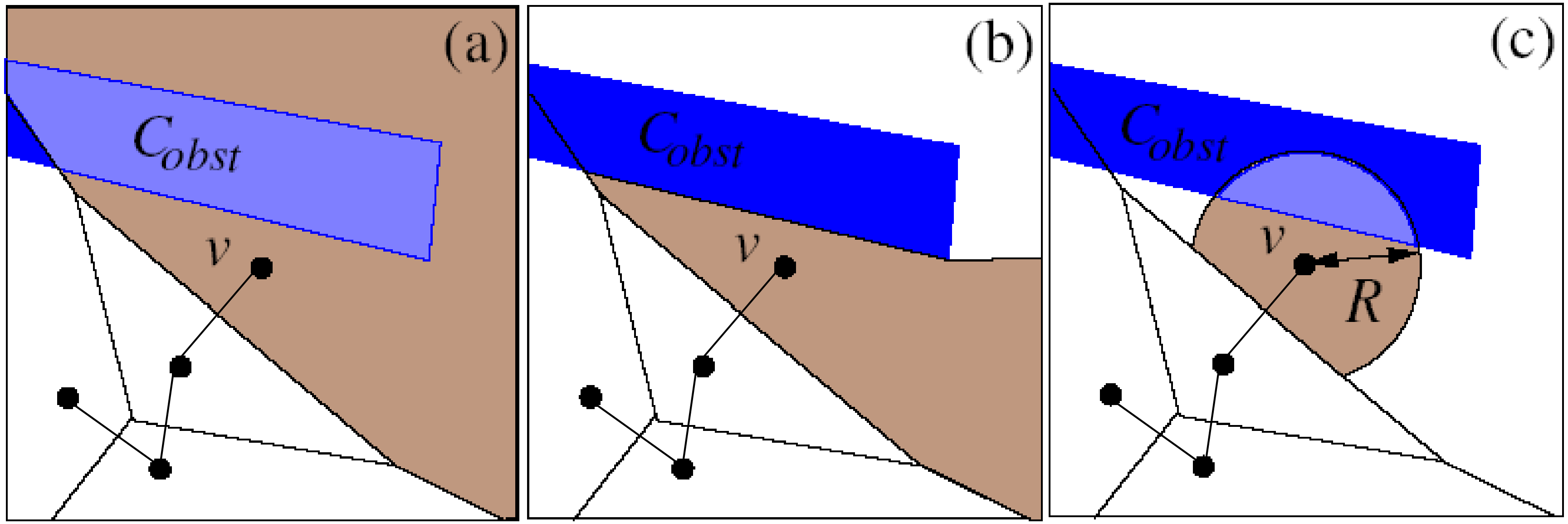


Visibility-Based Clipping of the Voronoi Regions



Nice idea, but how can this be done in practice?
Even better: Voronoi diagram for obstacle-based metric

A Boundary Node

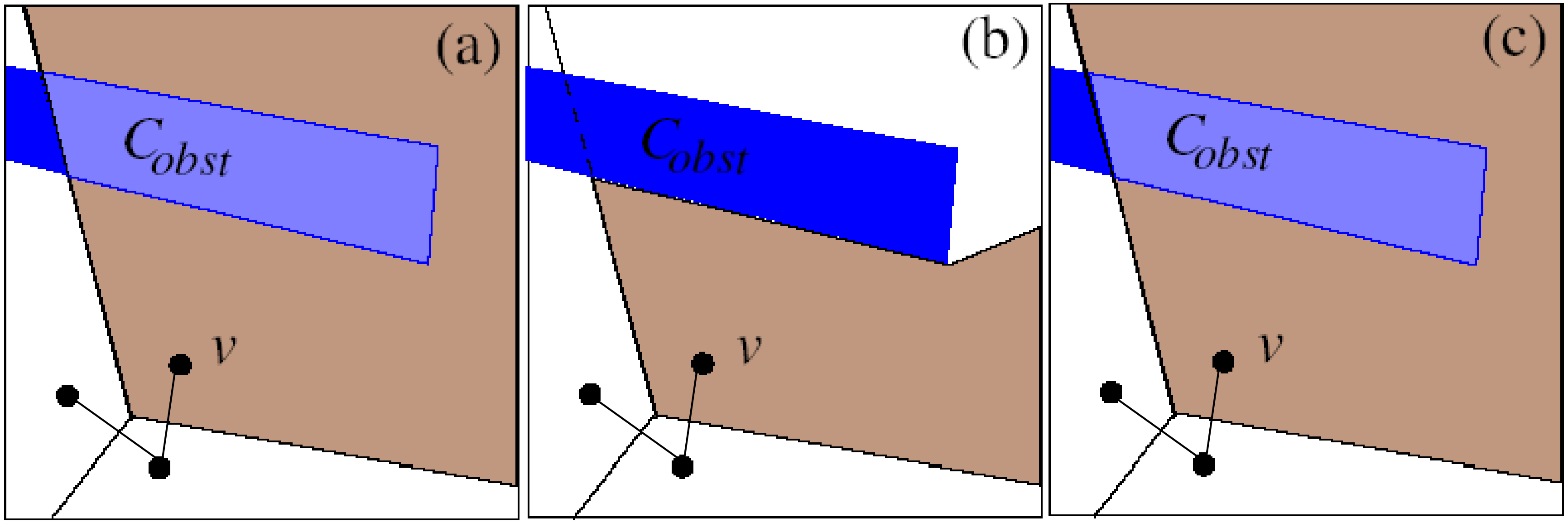


(a) Regular RRT, unbounded Voronoi region

(b) Visibility region

(c) Dynamic domain

A Non-Boundary Node

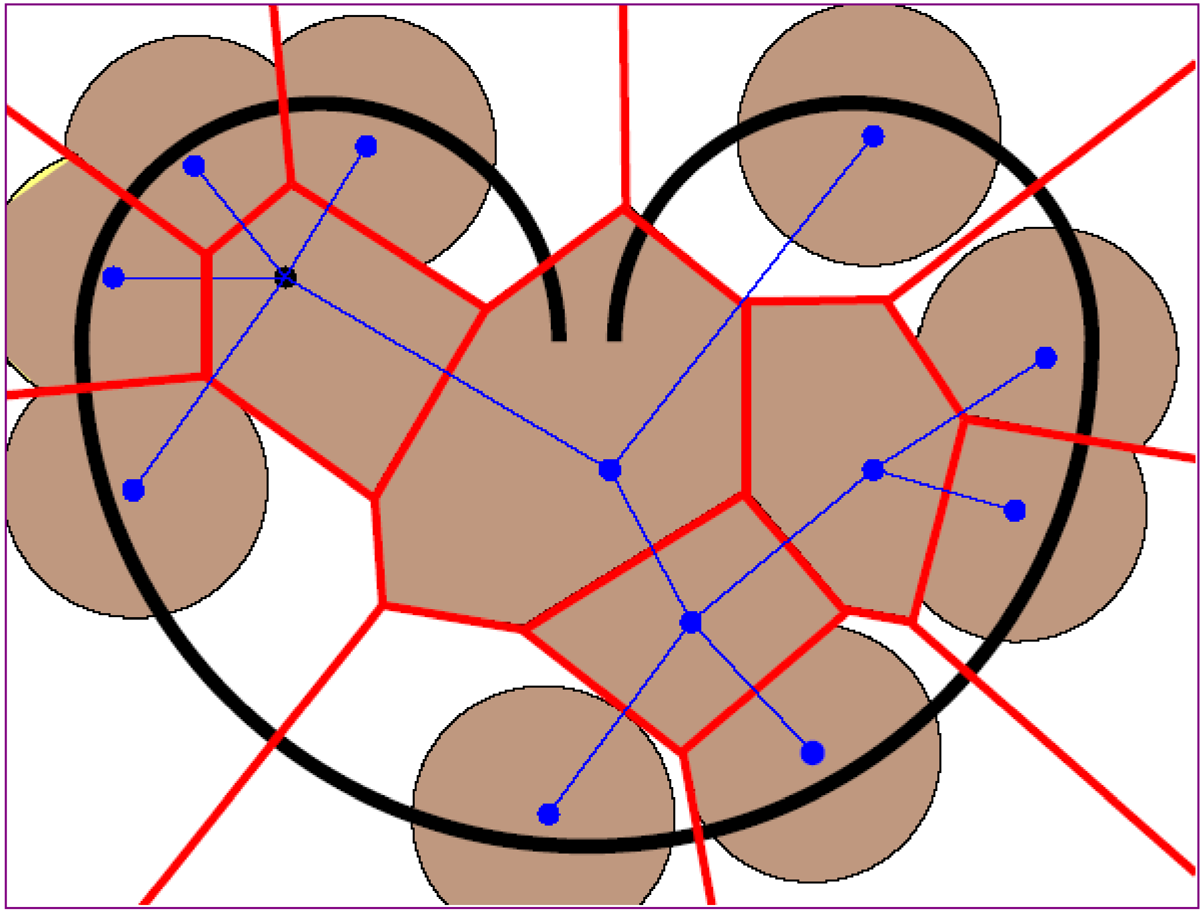


(a) Regular RRT, unbounded Voronoi region

(b) Visibility region

(c) Dynamic domain

Dynamic-Domain RRT Bias

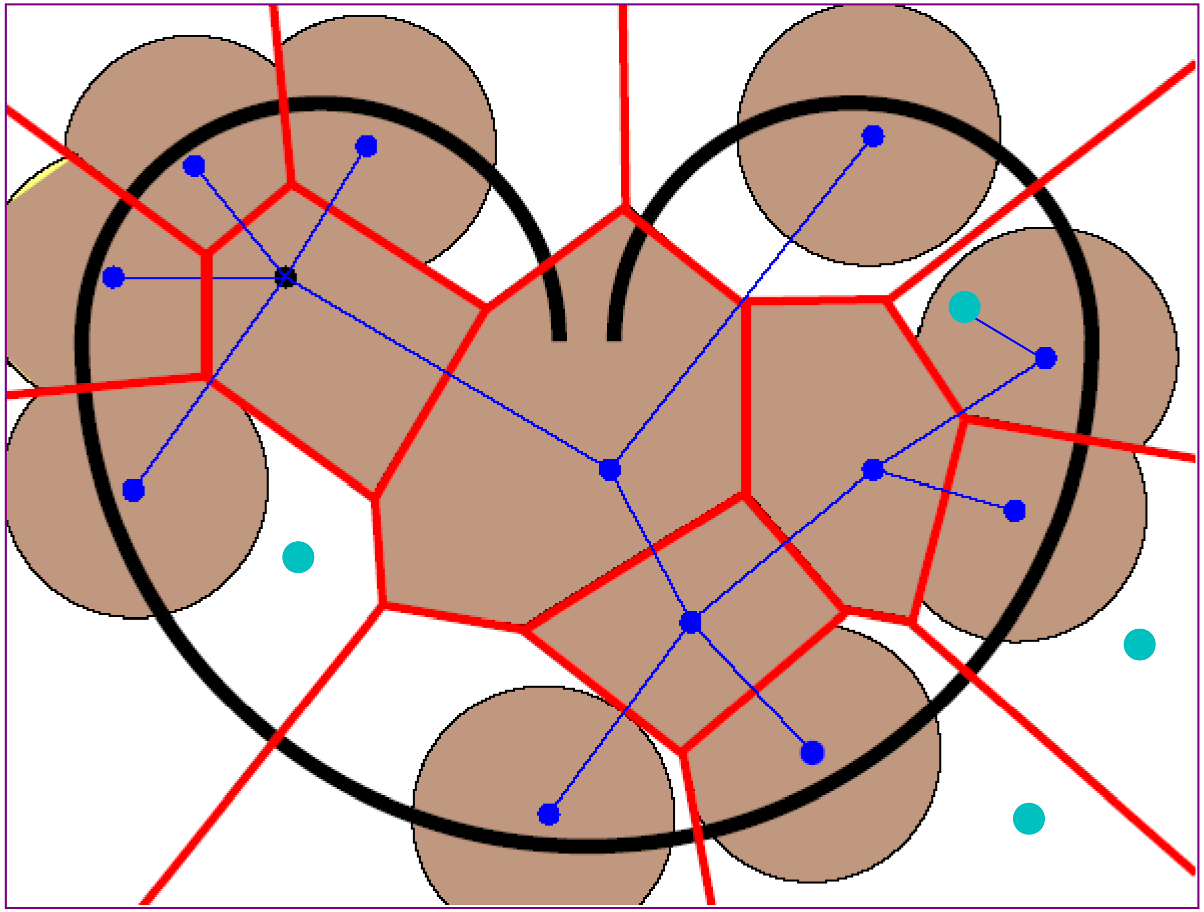


Dynamic-Domain RRT Construction

BUILD_DYNAMIC_DOMAIN_RRT(q_{init})

$\mathcal{T}.init(q_{init})$;
for $k = 1$ **to** K **do**
    repeat
        $q_{rand} \leftarrow$ RANDOM_CONFIG();
        $q_{near} \leftarrow$ NEAREST_NEIGHBOR($q_{rand}, \mathcal{T}$);
    until $dist(q_{near}, q_{rand}) < q_{near}$.radius
    if CONNECT($\mathcal{T}, q_{rand}, q_{near}, q_{new}$)
        q_{new}.radius $= \infty$;
        $\mathcal{T}$.add_vertex(q_{new});
        $\mathcal{T}$.add_edge(q_{near}, q_{new});
    else
        q_{near}.radius $= R$;
Return $\mathcal{T}$;

Dynamic-Domain RRT Bias



Tradeoff between nearest neighbor calls and collision detection calls

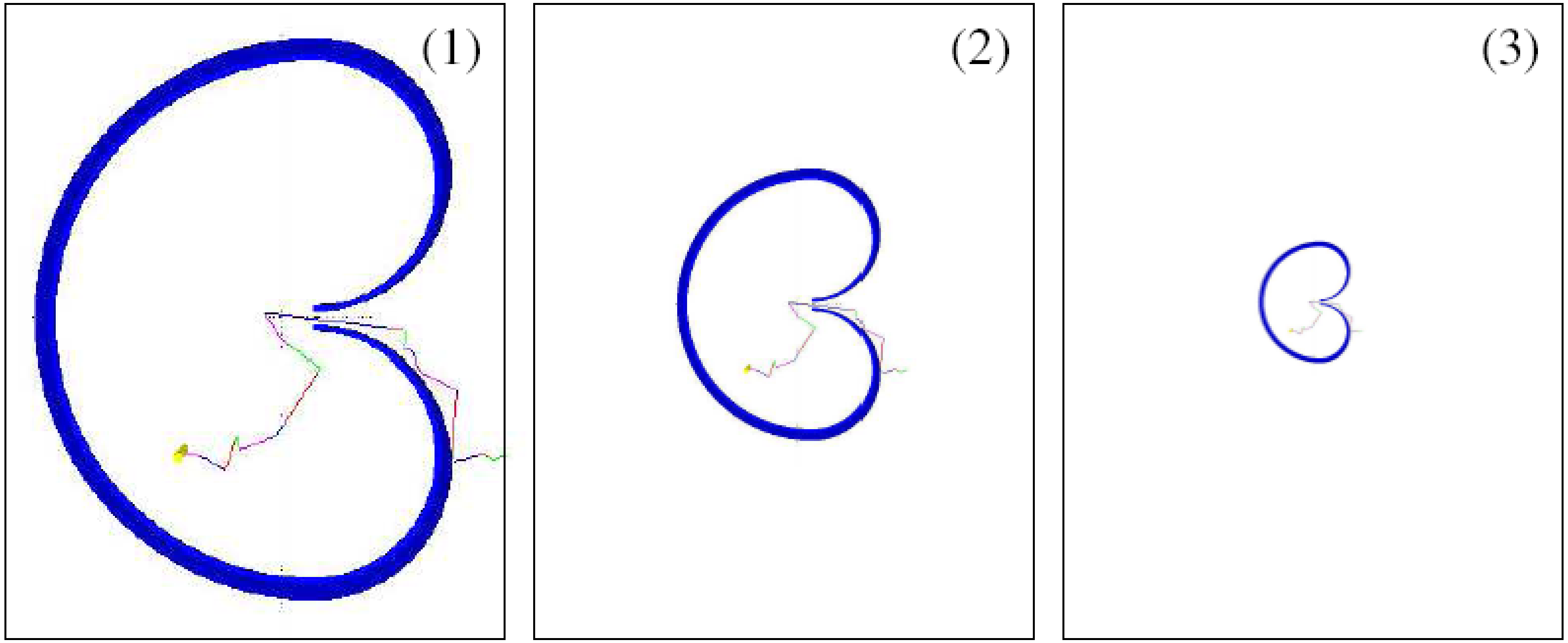
Experiments

- **333 Mhz machine**

Two kinds of experiments:

- **Controlled experiments for toy problems**
- **Challenging benchmarks from industry and biology**

Shrinking Bug Trap



Large Medium Small

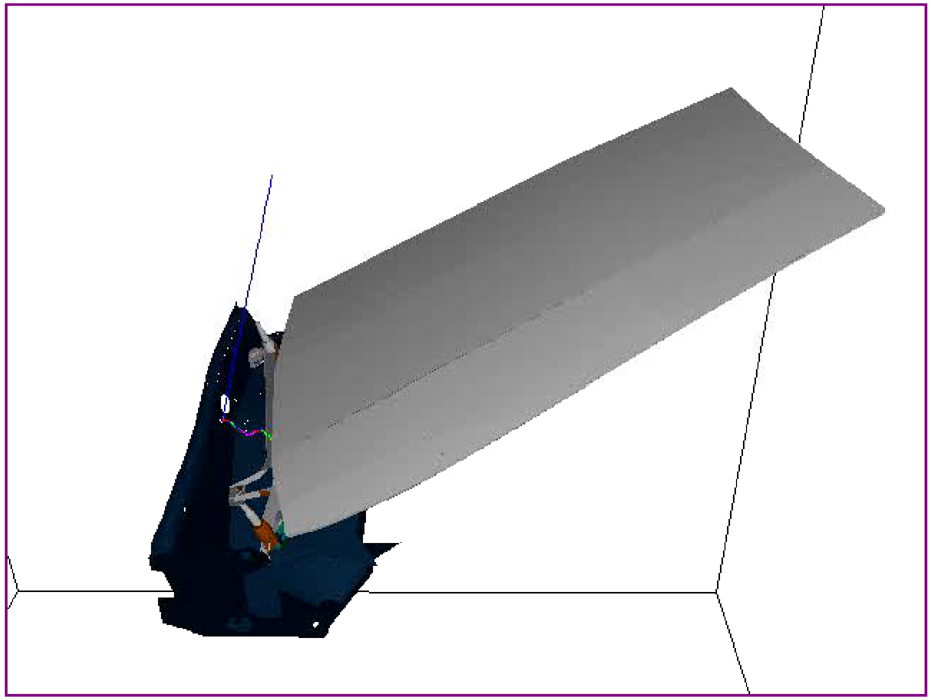
Shrinking Bug Trap

Trap Size	Statistic	Dynamic-Domain bi-RRT	bi-RRT
Large	time (1)	0.4 sec	0.1 sec
	no. nodes (1)	253	37
	CD calls (1)	618	54
Medium	time (2)	2.5 sec	379 sec
	no. nodes (2)	1607	6924
	CD calls (2)	3751	781530
Small	time (3)	1.6 sec	$>$ 80000 sec
	no. nodes (3)	1301	–
	CD calls (3)	3022	–

The smaller the bug trap, the better the improvement

KAIST

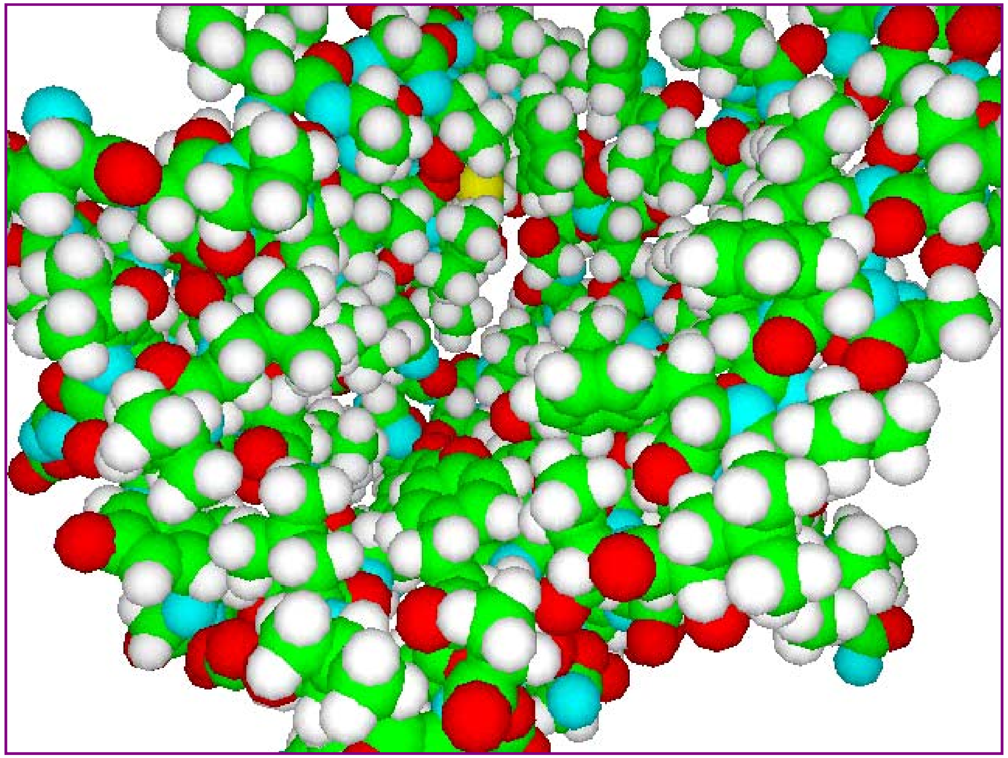
Wiper Motor (courtesy of KINEO)



- 6 dof problem
- CD calls are expensive

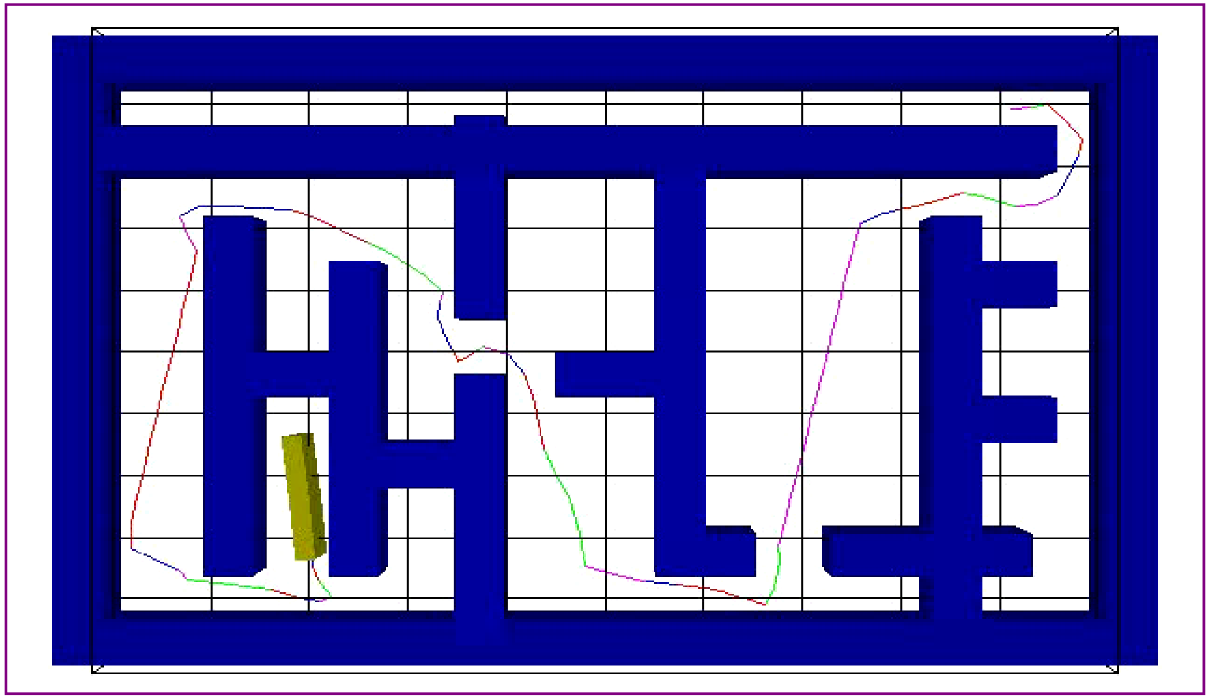
	Dynamic-Domain bi-RRT	bi-RRT
time	217 sec	$>$ 80000 sec
no. nodes	219	–
CD calls	30443	–

Molecule



	Dynamic-Domain bi-RRT	bi-RRT
time	70 sec	2926 sec
no. nodes	1358	428
CD calls	47710	1257055

Labyrinth



- 3 dof problem
- CD calls are not expensive

	Dynamic-Domain bi-RRT	bi-RRT
time	161 sec	237 sec
no. nodes	25483	20392
CD calls	604503	464137

Conclusions

- Controlling Voronoi bias is important in RRTs
- Provides dramatic performance improvements on some problems
- Does not incur much penalty for unsuitable problems

Work in Progress:

- There is a radius parameter; adaptive tuning is possible

Class Objectives were:

- Understand the RRT technique and its recent advancements

KAIST