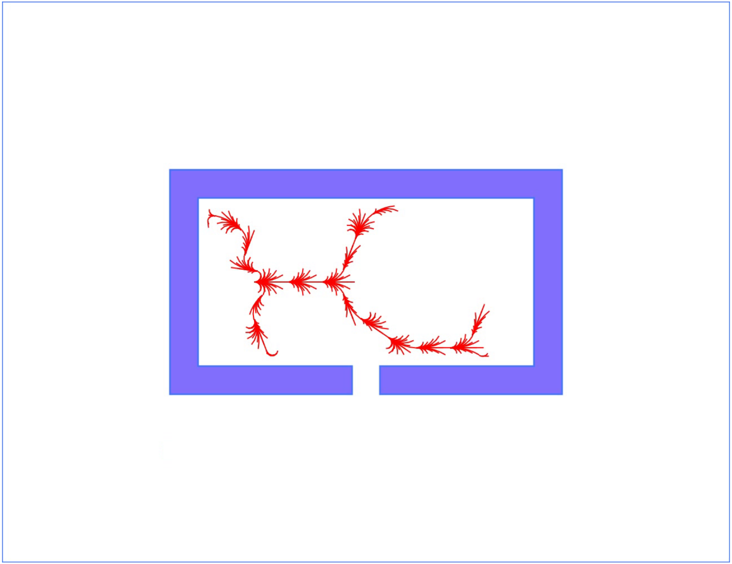
Motion planning: sampling-based planners I

Vojtěch Vonásek

Department of Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague

Planning for "car-like" vehicle



Motion/path planning

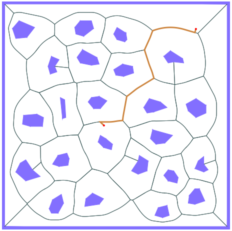
- Finding of collision-free trajectory/path for a robot
- Formulation using the configuration space $\mathcal{C}$
- C is continuous $\rightarrow$ conversion to a discrete representation (graph) $\rightarrow$ graph search
- Combinatorial path planning
  - Require an explicit representation of $\mathcal{C}_{obs}$
  - For point/disc robots (if $\mathcal{C}$ is sames as $\mathcal{W}$)
  - Visibility graphs, Voronoi diagrams, . . .

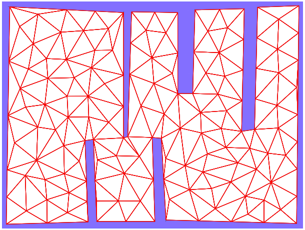
Motion planning

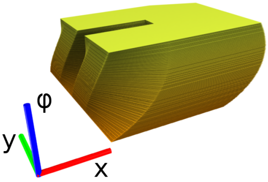
Continuous problem
$\mathcal{C}$-space specification, q_{init}, q_{goal}

Discretization of $\mathcal{C}$
Explicit construction random/deterministic sampling of $\mathcal{C}$

Graph search
Dijkstra, A*, D*, . . .





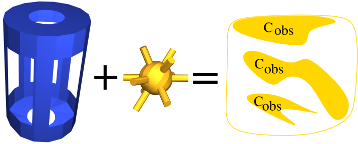


Configuration space

- Configuration space $\mathcal{C}$ has as many dimensions as DOFs of the robot
- Obstacles $\mathcal{C}_{\text{obs}}$ are given implicitly!

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

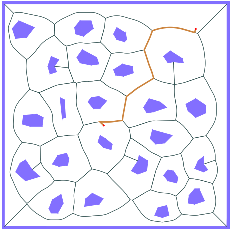
- $\mathcal{C}_{\text{obs}}$ depends both on robot and obstacles!

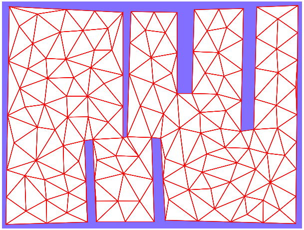


- Generally, explicit geometry/shape of $\mathcal{C}_{\text{obs}}$ is not available
- Problem of enumerating configurations in $\mathcal{C}_{\text{obs}}$
- Problem of enumerating “surface” configurations of $\mathcal{C}_{\text{obs}}$

Problem of enumerating “surface” configurations of $\mathcal{C}_{\text{obs}}$

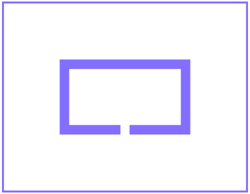
- We cannot generally/easy/fast say what are surface/boundary configurations of $\mathcal{C}_{\text{obs}}$
- This precludes combinatorial path planners (e.g., Visibility Graphs, Voronoi diagrams, Cell-decompositions, . . .) to be used for high-dimensional $\mathcal{C}$-space
    - they require surface/boundary of $\mathcal{C}_{\text{obs}}$

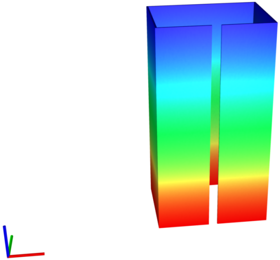




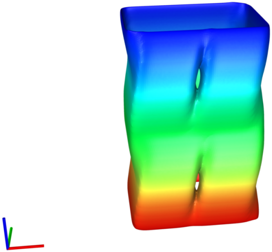
Configuration space: example I

- Map: 1000×700 units
- Robot: rectangle $20 \times a$ units
- $q = (x, y, \varphi)$
- $\mathcal{C}$ visualized for $0 \leq \varphi < 2\pi$
- $\varphi = 0 \rightarrow$  $\leftarrow \varphi = 2\pi$





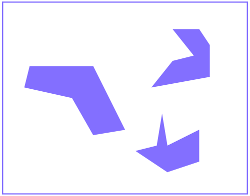
$a = 1$

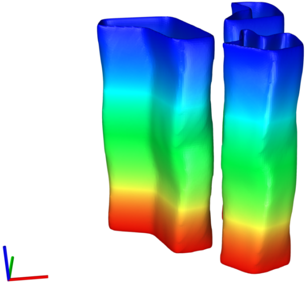


$a = 100$

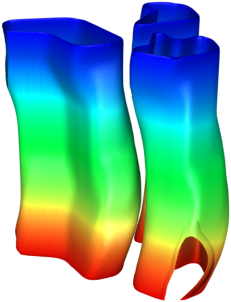
Configuration space: example II

- Map: 2000×1600 units
- $q = (x, y, \varphi)$
- $\mathcal{C}$ visualized for $0 \leq \varphi < 2\pi$
- $\varphi = 0 \rightarrow$  $\leftarrow \varphi = 2\pi$





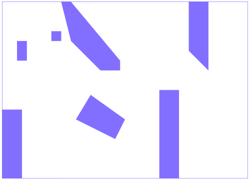
$\mathcal{A}$: rectangle 20×100 units

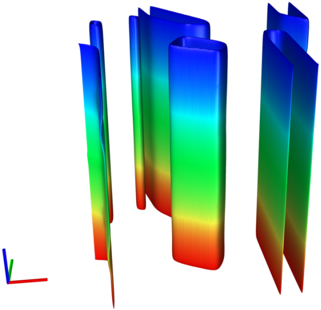


$\mathcal{A}$: equilateral triangle, side 100 units
(right-bottom "hole" caused by rendering clip)

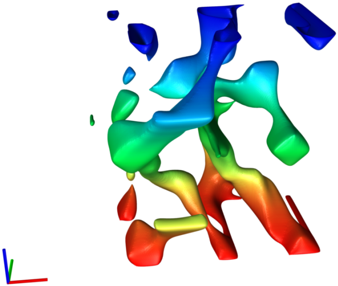
Configuration space: example III

- Map: 5000 $\times$ 3000 units
- $q = (x, y, \varphi)$
- $\mathcal{C}$ visualized for $0 \leq \varphi < 2\pi$
- $\varphi = 0 \rightarrow$ [colour scale] $\leftarrow \varphi = 2\pi$





$\mathcal{A}$: rectangle 20 $\times$ 100 units



$\mathcal{A}$: “u”-robot

Why is search in $\mathcal{C}$-space challenging

- $\mathcal{C}$-space is usually high-dimensional in practical applications
  - Discretization not reasonable due to memory/time limits
- Non-trivial mapping between the shape of robot $\mathcal{A}$ and obstacles $\mathcal{O}$
  - Simple obstacles in $\mathcal{W}$ may be quite complex in $\mathcal{C}$
- Narrow passages (we will discuss later)

Early methods (combinatorial path planners)

- Designed for 2D/3D workspaces for point robots, complete, optimal (some), deterministic
- Limited only to special cases
- In late 1980s, these methods have became impractical

But general path/planning requires search in $\mathcal{C}$-space!

- If you are desperate, flip a coin $\rightarrow$ randomization!

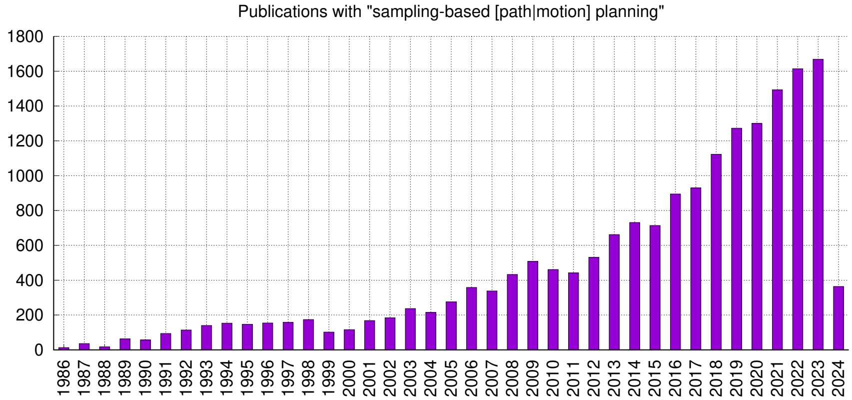
- Dijkstra's algorithm, 1959
- A*, 1968
- Configuration space, 1983
- Era of combinatorial planning, 1980s–1990s
- First planners using randomization, early 1990s
- Probabilistic roadmaps (PRM), 1995
- Rapidly-exploring Random Tree (RRT), 1998

☛ Dijkstra, E. W. "A Note on Two Problems in Connexion with Graphs", Numerische Mathematik 1, no. 1 (December 1959): 269–71.

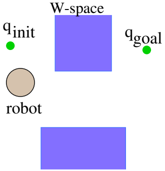
☛ P. E. Hart, N. J. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," in IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100-107, July 1968,

☛ Lozano-Perez, "Spatial Planning: A Configuration Space Approach," in IEEE Transactions on Computers, vol. C-32, no. 2, pp. 108-120, Feb. 1983,

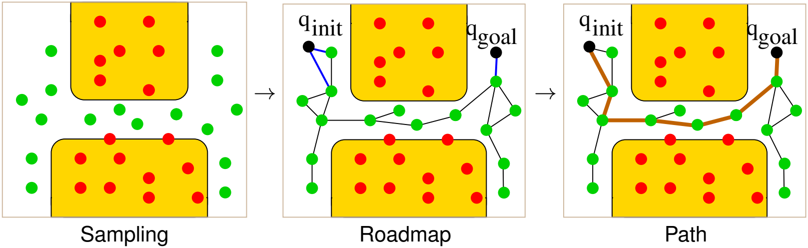
Milestones in motion planning



- $\mathcal{C}$ is randomly sampled
- Each sample is a configuration $q \in \mathcal{C}$
- The samples are classified as free ($q \in \mathcal{C}_{\text{free}}$) or non-free ($q \in \mathcal{C}_{\text{obs}}$) using collision detection



- Free samples are stored and connected, if possible, by a “local planner”
- Result of sampling-based planning is a “roadmap” — graph
- The roadmap is the discretized image of $\mathcal{C}_{\text{free}}$
- Graph-search in the roadmap

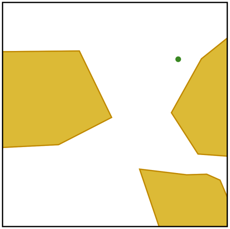


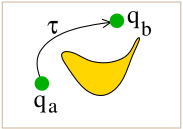
- Sampling-based planning can solve any problem formulated using $\mathcal{C}$-space

✓ Robots of arbitrary shapes
  - Robot shape is considered in collision detection
  - Collision detection is used as a "black-box"
  - Single-body or multi-body robots allowed

✓ Robots with many-DOFs
  - Because the search is realized directly in $\mathcal{C}$-space
  - Dimension of $\mathcal{C}$ is determined by the DOFs

✓ Kinematic, dynamic and task constraints can be considered
  - It depends on the employed local planner





- Given configurations $q_a \in \mathcal{C}_{\text{free}}$ and $q_b \in \mathcal{C}_{\text{free}}$, the local planner attempts to find a path τ:

$$\tau : [0,1] \rightarrow \mathcal{C}_{\text{free}}$$

such that $\tau(0) = q_a$ and $\tau(1) = q_b$

τ must be collision free!

Control-theory approach: special cases

- We can assume that q_a and q_b are "near" without obstacles
- Two-point boundary value problem (BVP)
- Local planner is designed as a controller
- But problems are with obstacles!

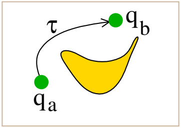
Generally:

- The definition of "local planning" is same as motion planning

→ same complexity as motion planning!

Exact local planners

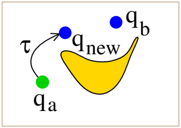
- For certain systems, BVP can be solved analytically
- Example: car-like without backward motions $\rightarrow$ Dubins car



Exact local planner

Approximate local planners

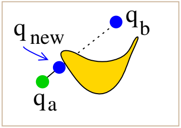
- Path τ connects q_a with q_{new} that is near-enough from q_b
- Computation e.g. using forward motion model and integration over time Δt



Approximate

Straight-line local planners

- Connects q_a and q_b by line-segment
- Check the collisions of the line-segment
- Connect q_a with the first contact configuration q_{new} or with q_b if no collision occurs
- Suitable for systems without kinematic/dynamic constraints

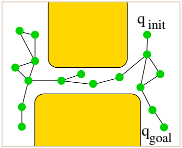


Straight-line

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE | MRS MULTI-ROBOT SYSTEMS GROUP

Multi-query methods

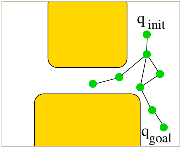
- Can find paths between multi start/goal queries
- Requires to build a roadmap covering whole $\mathcal{C}_{\text{free}}$
- Probabilistic Roadmaps (PRM) + many derivates
- ✓ good for frequent planning and replanning
- ✗ sometimes slower construction



Multi-query roadmap

Single-query methods

- The roadmap is built only to answer a single start/goal query
- The sampling of $\mathcal{C}$ terminates if the query can be answered
- Tree-based planners: Rapidly-exploring Random Trees (RRT), Expansive-space Tree (EST) + their variants
- ✓ Practically faster for single-query
- ✗ Any subsequent planning requires novel search of $\mathcal{C}$
- ✗ Slow for multi-query planning



Single-query roadmap

Probabilistic Roadmaps (PRM)

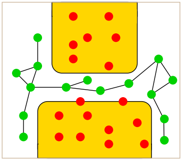
- Two-phase method: learning phase and query phase

Learning phase

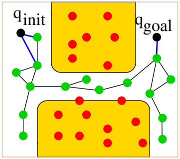
- Random samples are generated in $\mathcal{C}$
- Samples are classified as free/non-free; free samples are stored
- Each sample is connected to its near neighbors by a local planner
- Final roadmap may contain cycles

Query phase:

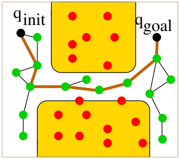
- Answers path/motion planning from $q_{\text{init}} \in \mathcal{C}_{\text{free}}$ to $q_{\text{goal}} \in \mathcal{C}_{\text{free}}$
- q_{init} and q_{goal} are connected to their nearest neighbors in the roadmap (using local planner)
- Graph-search of the roadmap



Learning phase



Query phase



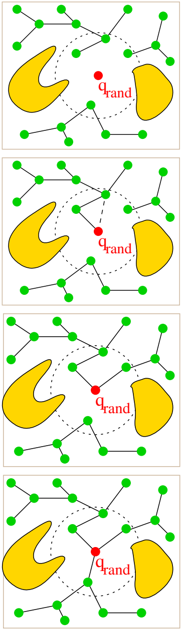
Path

☛ L. E. Kavraki, P. Svestka, et al., "Probabilistic roadmaps for path planning in high-dimensional configuration spaces,". IEEE Trans. on Robotics and Automation, 12(4), 1996.

Original PRM

- Simultaneous sampling + roadmap expansion
- q_{rand} is connected to each graph component only once
- Roadmap is a tree structure

```
V = ∅; E = ∅                                   // vertices and edges
G = (V, E)                                     // empty roadmap
while |V| < n do
    q_rand = generate random sample in C
    if q_rand is collision-free then
        G.addVertex(q_rand)
        foreach q ∈ V.neighborhood*(q_rand) do
            if not G.sameComponent(q_rand, q) ∧ connect(q_rand,q)
              then
                G.addEdge(q_rand, q)
```



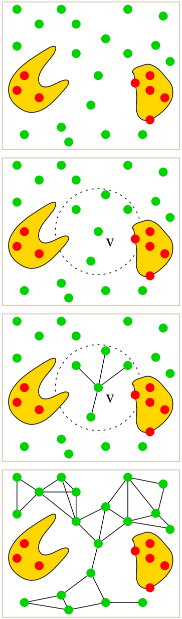
- *neighborhood** returns q by increasing distance from q_{rand}

☛ L. E. Kavraki, P. Svestka, et al., "Probabilistic roadmaps for path planning in high-dimensional configuration spaces,". IEEE Trans. on Robotics and Automation, 12(4), 1996.

Simplified PRM (sPRM)

- Separate sampling and roadmap connection
- Each node is connected to its nearest neighbors
- Roadmap can contains cycles

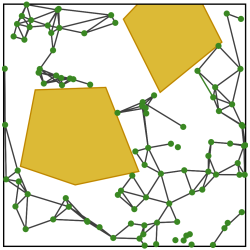
$V = \emptyset; E = \emptyset$ // vertices and edges
while $|V| < n$ **do** // generating n collision-free samples
    q_{rand} = generate random sample in $\mathcal{C}$
    if q_{rand} *is collision-free* **then**
        $V = V \cup \{q_{\text{rand}}\}$
foreach $v \in V$ **do** // connecting samples to roadmap
    $V_n = V$.neighborhood(v)
    foreach $u \in V_n, u \neq v$ **do**
        if *connect*(u, v) **then** // local planner
            $E = E \cup \{(u, v)\}$
$G = (V, E)$ // final roadmap



☛ S. Karaman, and E. Frazzoli. "Sampling-based algorithms for optimal motion planning." The international journal of robotics research 30.7 (2011): 846-894.

- Behavior of sPRM is mostly influenced by implementation of *V.neighborhood*

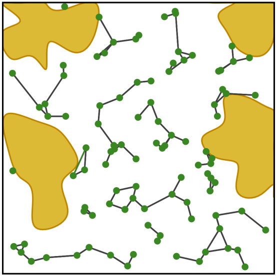
k-nearest sPRM (aka *k*-sPRM)



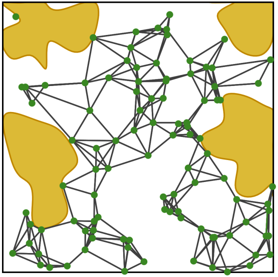
- *V.neighborhood* provides k nearest neighbors from each node of the roadmap
- Probabilistically complete if $k \neq 1$
- Is not asymptotically optimal
- Usually $k = 15$

Variable radius sPRM

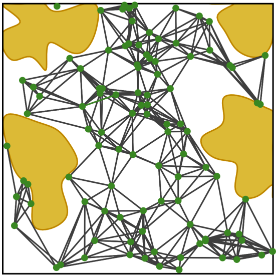
- *V.neighborhood* returns nearest neighbors of q_{rand} within a radius r
- The choice of r influences completeness and optimality of sPRM
- Most important — PRM* planner



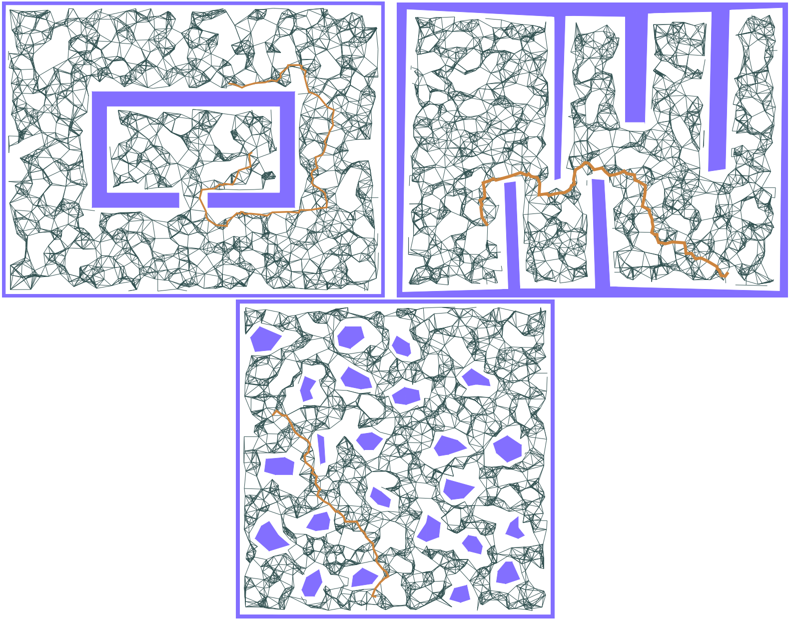
$k = 1$



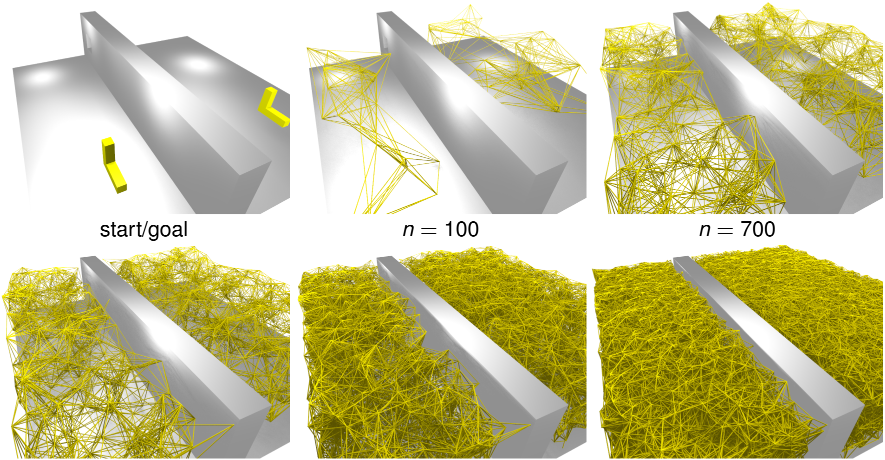
$k = 3$



$k = 5$



sPRM example 3D $\mathcal{W}$

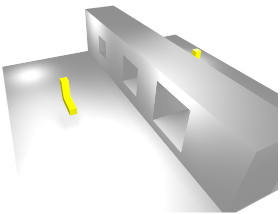


start/goal | $n = 100$ | $n = 700$

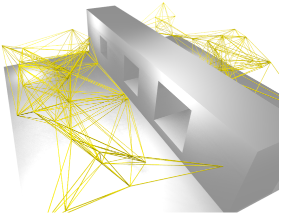
$n = 1500$ | $n = 8000$ | $n = 50000$

The wall contains one window, but no path found with 50k samples

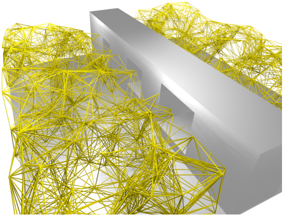
sPRM example 3D $\mathcal{W}$



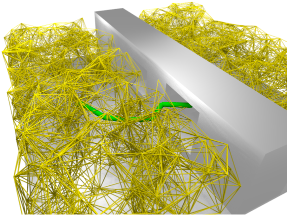
start/goal



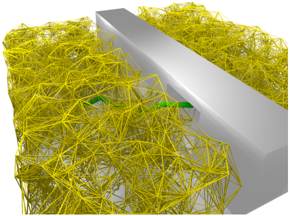
$n = 100$



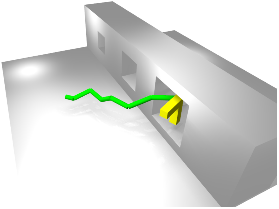
$n = 1000$



$n = 2100$



$n = 4100$



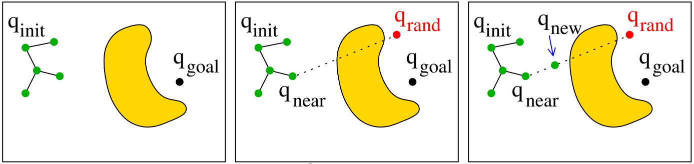
solution

Rapidly-exploring Random Tree (RRT)

- Incremental search of $\mathcal{C}$
- Collision-free configurations are stored in tree $\mathcal{T}$
- $\mathcal{T}$ is rooted at q_{init}
- Tree is expanded towards random samples q_{rand}
- The search terminates if tree is close enough to q_{goal}, or after I_{max} iterations

---

initialize tree $\mathcal{T}$ with q_{init}
for $i = 1, \ldots, I_{max}$ **do**
  q_{rand} = generate randomly in $\mathcal{C}$
  q_{near} = find nearest node in $\mathcal{T}$ towards q_{rand}
  q_{new} = localPlanner from q_{near} towards q_{rand}
  if *canConnect*($q_{\text{near}}, q_{\text{new}}$) **then**
    $\mathcal{T}$.addNode(q_{new})
    $\mathcal{T}$.addEdge($q_{\text{near}}, q_{\text{new}}$)
    if $\varrho(q_{\text{new}}, q_{\text{goal}}) < d_{goal}$ **then**
      return path from q_{init} to q_{goal}



Tree | Sampling | Tree extension

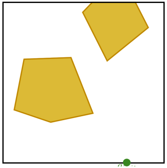
☛ LaValle:, S. M. Rapidly-exploring random trees: a new tool for path planning". Technical report, Iowa State University, 1998

Rapidly-exploring Random Tree (RRT)

- Incremental search of $\mathcal{C}$
- Collision-free configurations are stored in tree $\mathcal{T}$
- $\mathcal{T}$ is rooted at q_{init}
- Tree is expanded towards random samples q_{rand}
- The search terminates if tree is close enough to q_{goal}, or after I_{max} iterations

---

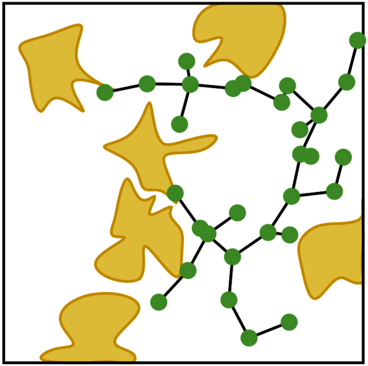
initialize tree $\mathcal{T}$ with q_{init}
for $i = 1, \ldots, I_{max}$ **do**
    q_{rand} = generate randomly in $\mathcal{C}$
    q_{near} = find nearest node in $\mathcal{T}$ towards q_{rand}
    q_{new} = localPlanner from q_{near} towards q_{rand}
    if *canConnect*$(q_{\text{near}}, q_{\text{new}})$ **then**
        $\mathcal{T}$.addNode(q_{new})
        $\mathcal{T}$.addEdge$(q_{\text{near}}, q_{\text{new}})$
        if $\varrho(q_{\text{new}}, q_{\text{goal}}) < d_{goal}$ **then**
            return path from q_{init} to q_{goal}



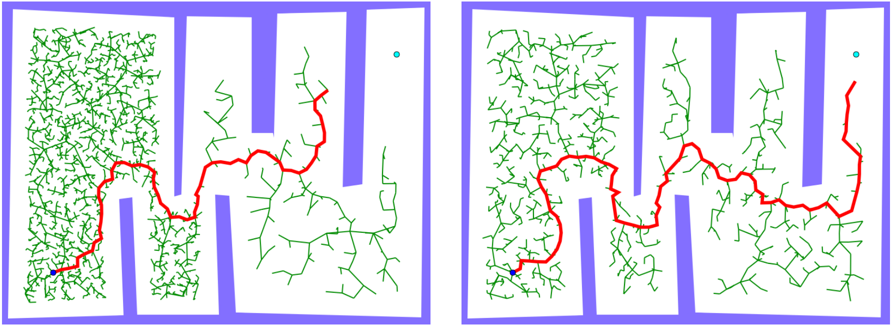
☛ LaValle:, S. M. Rapidly-exploring random trees: a new tool for path planning". Technical report, Iowa State University, 1998

RRT example in 2D $\mathcal{W}$

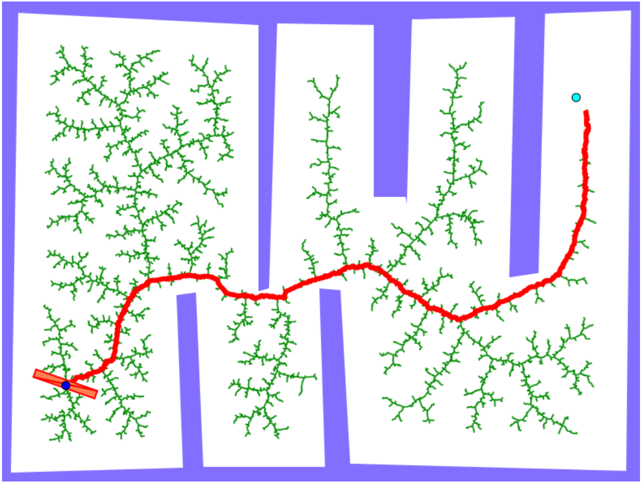
- 2D $\mathcal{C}$

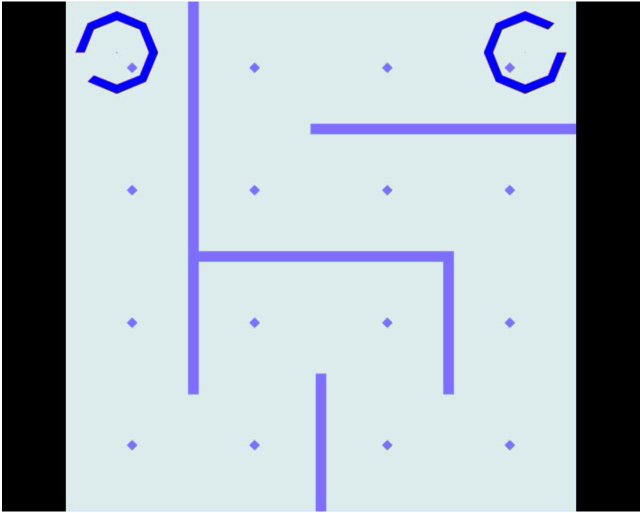


RRT example in 2D $\mathcal{W}$

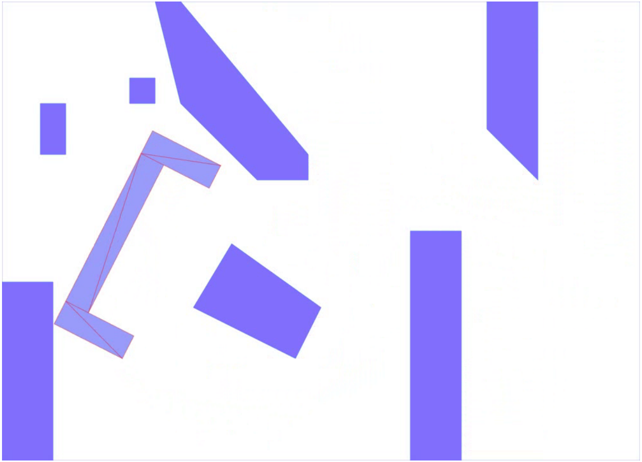


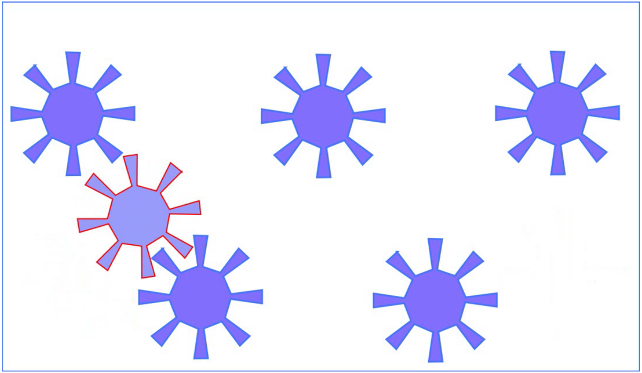
- 2D robot, rotation allowed $\rightarrow$ 3D $\mathcal{C}$
- Why the tree does not "touch" the obstacles?



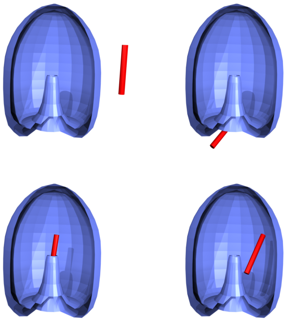


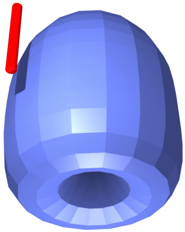
RRT example in 2D $\mathcal{W}$





RRT example in 3D $\mathcal{W}$

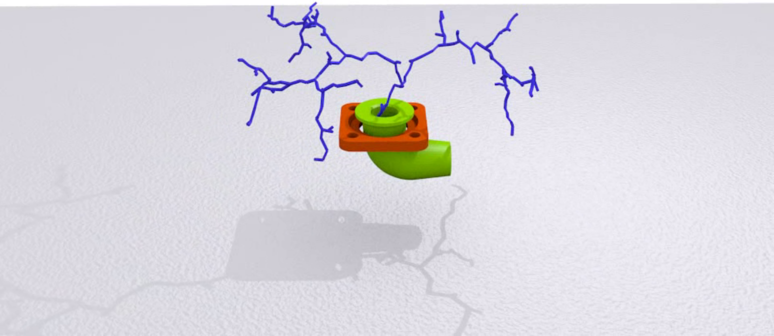




- 3D Bugtrap benchmark

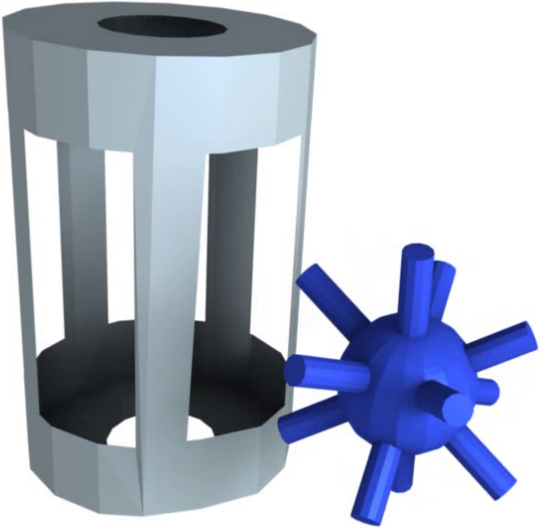
  `parasol.tamu.edu/groups/amatogroup/benchmarks/`
- 3D robot in 3D space $\rightarrow$ 6D $\mathcal{C}$

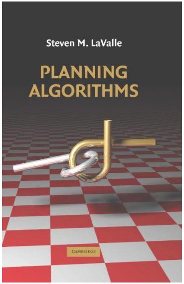
RRT example in 3D $\mathcal{W}$



- 3D Flange benchmark
  parasol.tamu.edu/groups/amatogroup/benchmarks/
- 3D robot in 3D space $\rightarrow$ 6D $\mathcal{C}$

RRT example in 3D $\mathcal{W}$




Steven M. LaValle
PLANNING ALGORITHMS

RRT: tree expansion types

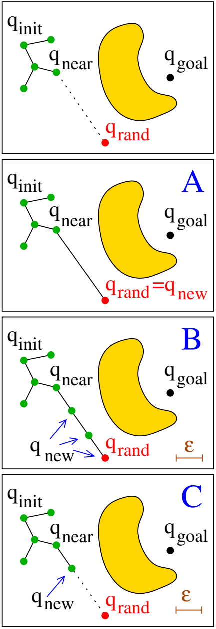
Straight-line expansion: make the line-segment S from q_{near} to q_{rand}

Variants:

A If S is collision-free, expand the tree only by $q_{\text{new}} = q_{\text{rand}}$
- Creates long segments, fast exploration of $\mathcal{C}$
- Requires nearest-neighbor search to consider point-segment distance
- Requires connection in the middle of line-segment

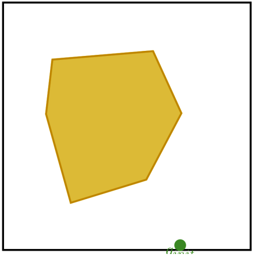
B If S is collision-free, discretize S and expand the tree by all points on S
- Most used, enables fast nearest-neighbor search

C Find configuration $q_{\text{new}} \in S$ at the distance ε from q_{near}. Expand tree by q_{new} if it's collision-free
- Basic RRT, slower growth than B
- Enables fast nearest-neighbor search

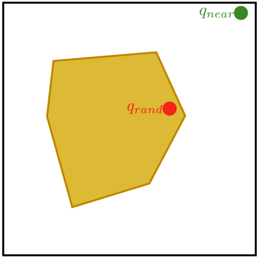


RRT: tree expansion types

- Examples for point robot in 2D $\mathcal{C}$
- Variant A

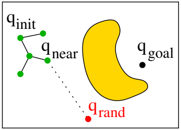


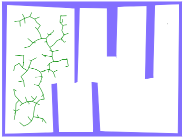
- Variant C

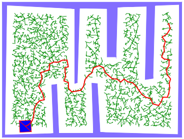


RRT: properties

- RRT builds a tree $\mathcal{T}$ of collision-free configurations
- $\mathcal{T}$ is rooted at q_{init}
- $\mathcal{T}$ is without cycles
- Path from q_{init} to q_{goal}:
  - Find nearest node $q'_{\mathrm{goal}} \in \mathcal{T}$ towards q_{goal}
  - Start at q'_{goal} and follow predecessors to q_{init}
- Existing $\mathcal{T}$ can answer queries starting at q_{init}
  - if goal is not in/near current $\mathcal{T}$, $\mathcal{T}$ is further grown
- Non-optimal
- Probabilistically complete



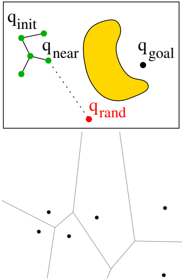


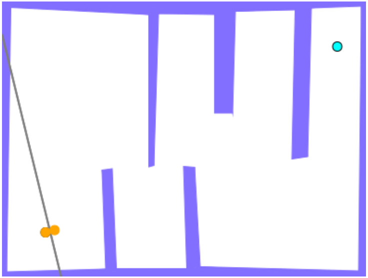


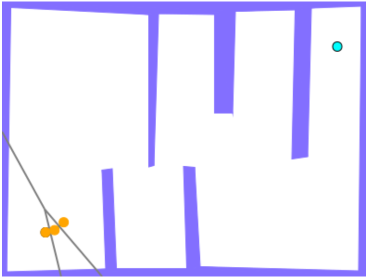
- Why the tree does not grow to itself?
- Why does it “rapidly” explore the $\mathcal{C}$-space?

  . . . because of Voronoi bias!

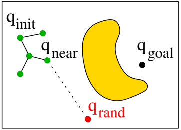
- RRT prefers to expand $\mathcal{T}$ towards unexplored areas of $\mathcal{C}$
- This is caused by **Voronoi bias**:
  - q_{rand} is generated **uniformly** in $\mathcal{C}$
  - $\mathcal{T}$ is expanded from **nearest** node in $\mathcal{T}$ **towards** q_{rand}
  - The probability that a node $q \in \mathcal{T}$ is selected for the expansion is proportional to the area/volume of it's Voronoi cell
- Voronoi bias is implicit (caused by the nearest-rule selection)



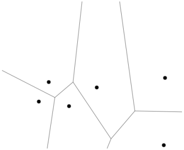


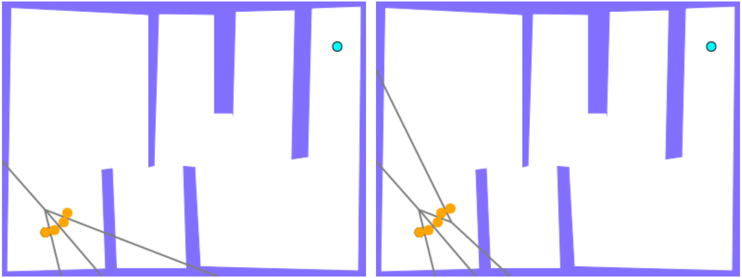


RRT: Voronoi bias I

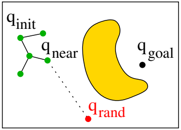


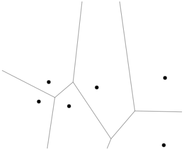
- RRT prefers to expand $\mathcal{T}$ towards unexplored areas of $\mathcal{C}$
- This is caused by **Voronoi bias**:
  - q_{rand} is generated **uniformly** in $\mathcal{C}$
  - $\mathcal{T}$ is expanded from **nearest** node in $\mathcal{T}$ **towards** q_{rand}
  - The probability that a node $q \in \mathcal{T}$ is selected for the expansion is proportional to the area/volume of it's Voronoi cell
- Voronoi bias is implicit (caused by the nearest-rule selection)

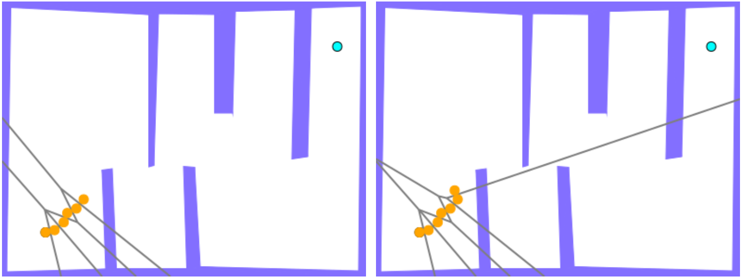




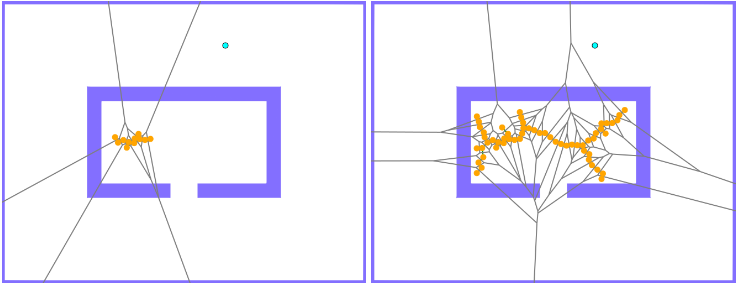
- RRT prefers to expand $\mathcal{T}$ towards unexplored areas of $\mathcal{C}$
- This is caused by **Voronoi bias**:
  - q_{rand} is generated **uniformly** in $\mathcal{C}$
  - $\mathcal{T}$ is expanded from **nearest** node in $\mathcal{T}$ **towards** q_{rand}
  - The probability that a node $q \in \mathcal{T}$ is selected for the expansion is proportional to the area/volume of it's Voronoi cell
- Voronoi bias is implicit (caused by the nearest-rule selection)







- Nearest-neighbors/Voronoi bias do not respect obstacles!
- If a node having large Voronoi cells is near an obstacle → tree expansion is blocked at this node

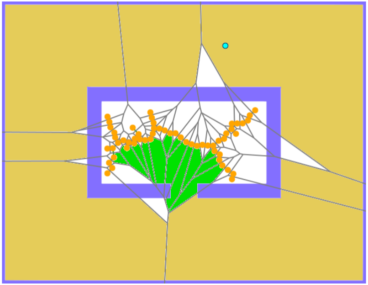


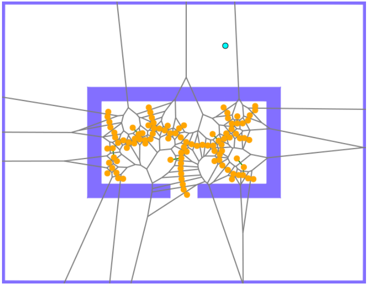
iteration 10, tree size 10 — iteration 70, tree size ∼ 60

- Tree grows well until iteration 70
- Yellow: areas with high prob. of being selected for expansion
- Green: areas that show be selected for expansion so the tree can escape the obstacle
- The tree does not expand much until iteration 300!

RRT: Voronoi bias II

- Nearest-neighbors/Voronoi bias do not respect obstacles!
- If a node having large Voronoi cells is near an obstacle → tree expansion is blocked at this node


iteration 70, tree size ∼ 60


iteration 300, tree size ∼ 100

- Tree grows well until iteration 70
- Yellow: areas with high prob. of being selected for expansion
- Green: areas that show be selected for expansion so the tree can escape the obstacle
- The tree does not expand much until iteration 300!

Expansive-space tree (EST)

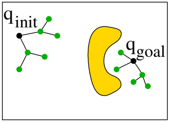
- Builds two trees $\mathcal{T}_i$ and $\mathcal{T}_g$ (from q_{init} and q_{goal})
- Weight $w(q)$ is computed for each configuration q
- Nodes are selected for expansion with probability $w(q)^{-1}$
- Expansion of one tree $\mathcal{T}$:

---

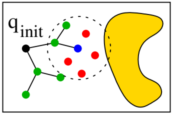
q' = select node from $\mathcal{T}$ with probability $w(q)^{-1}$
Q = k random points around q' : $Q = \{q \in \mathcal{C}_{\mathrm{free}} \mid \varrho(q, q') < d\}$
foreach $q \in Q$ **do**
    $w(q)$ = compute weight of the sample q
    if *rand*() $< w(q)^{-1}$ **and** *connect*(q, q') **then**
        $\mathcal{T}$.addNode(q)
        $\mathcal{T}$.addEdge(q', q)

- $w(q)$ is the number of nodes in $\mathcal{T}$ around q
- Both $\mathcal{T}_i$ and $\mathcal{T}_g$ grow until they approach each other
- Trees are connected using local planner between their nearest nodes

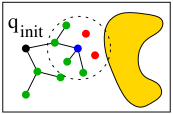
☛ D. Hsu, J.-C. Latomber et al. Path planning in expansive configuration spaces. Int. Journal of Comp. Geometry and Applications, 9(4-5), 1999



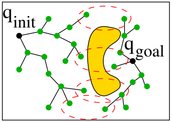
$\mathcal{T}_i$ and $\mathcal{T}_g$



q', samples Q

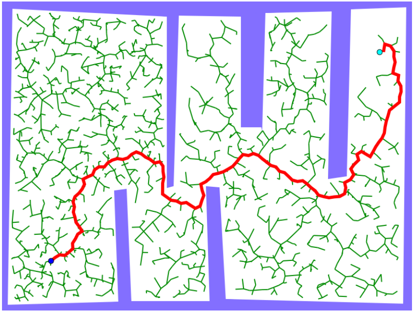


connected,ignored

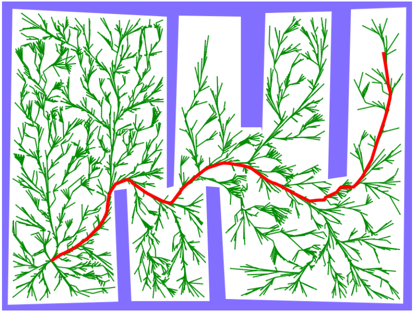


pairs for tree connection

- PRM/RRT/EST do not consider any optimality criteria
- Only sPRM is asymptotically optimal
- PRM* and RRT* are new planners for which asymptotic optimality was proven



RRT



RRT*

☛ S. Karaman, and E. Frazzoli. "Sampling-based algorithms for optimal motion planning." The international journal of robotics research 30.7 (2011): 846-894.

PRM*: overview

- PRM* is an improved version of sPRM
- PRM* uses "optimal" radius $r(n)$ for searching the nearest neighbors depending on the actual number of nodes n:

$$r(n) = \gamma_{PRM} \left(\frac{\log(n)}{n} \right)^{\frac{1}{d}}$$

$$\gamma_{PRM} > \gamma^*_{PRM} = 2 \left(1 + \frac{1}{d} \right)^{\frac{1}{d}} \left(\frac{\mu(\mathcal{C}_{\text{free}})}{\zeta_d} \right)^{\frac{1}{d}}$$

- d is the dimension of $\mathcal{C}$
- $\mu(\mathcal{C}_{\text{free}})$ is the volume of $\mathcal{C}_{\text{free}}$
- ζ_d is the volume of the unit ball in the $d-$dimensional Euclidean space
- r decays with n
- r depends also on the problem instance! — why?

PRM* algorithm

- Same as for sPRM, just the line 7 is changed to:
  $V_n = V.neighborhood(v, r(n))$, where $n = |V|$

- Variant of PRM* that uses k-nearest neighbors definitions

$$k = k_{PRM} \log(n)$$

$$k_{PRM} > k^*_{PRM} = e\left(1 + \frac{1}{d}\right)$$

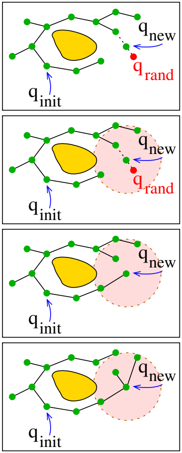
- The constant k^*_{PRM} depends only on d and not on the problem instance (compare it to γ^*_{PRM})
- $k_{PRM} = 2e$ is a valid choice for all problem instances

k-nearest PRM* algorithm (aka k-PRM*)

- Same as for sPRM, just the line 7 is changed to:

  V_n = $k-$nearest neighbors from V, $k = k_{PRM} \log(n)$

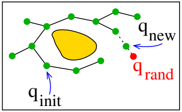
- Optimal version of RRT
- For each node, a cost of the path from q_{init} to that node is established
- RRT* has improved tree expansion and nearest-neighbor search
- Tree expansion by node q_{new}
  - Parent of q_{new} is optimized to minimize cost at q_{new}
  - After q_{new} is connected to tree, node it its vicinity are "rewired" via q_{new} if it improves their cost
- Nearest-neighbor search
  - Number of nearest-neighbors varies similarly to PRM*



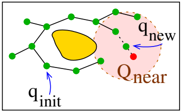
☛ S. Karaman, and E. Frazzoli. "Sampling-based algorithms for optimal motion planning." The international journal of robotics research 30.7 (2011): 846-894.

RRT*: algorithm

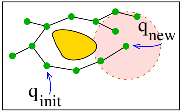
1 initialize tree $\mathcal{T}$ with q_{init}
2 **for** $i = 1, \ldots, I_{max}$ **do**
3   q_{rand} = generate randomly in $\mathcal{C}$
4   q_{near} = find nearest node in $\mathcal{T}$ towards q_{rand}
5   q_{new} = localPlanner from q_{near} towards q_{rand}
6   **if** q_{new} *is collision-free* **then**
7     $Q_{near} = \mathcal{T}$.neighborhood(q_{new}, r)
8     $\mathcal{T}$.addNode(q_{new})   `// new node to tree`
9     $q_{best} = q_{\text{near}}$ `// best parent of` q_{new} `so far`
10     $c_{best} = cost(q_{\text{near}}) + cost(line(q_{\text{near}}, q_{\text{new}}))$
11   **foreach** $q \in Q_{near}$ **do**
12       $c = cost(q) + cost(line(q, q_{\text{new}}))$
13   **if** *canConnect*(q, q_{new}) **and** $c < c_{best}$ **then**
14         $q_{best} = q$ `// new parent of` q_{new} `is` q
15         $c_{best} = c$ `// its cost`
16     $\mathcal{T}$.addEdge(q_{best}, q_{new})   `// tree connected to` q_{new}
17   **foreach** $q \in Q_{near}$ **do**   `// rewiring`
18       $c = cost(q_{\text{new}}) + cost(line(q_{\text{new}}, q))$
19   **if** *canConnect*(q_{new}, q) **and** $c < cost(q)$ **then**
20   change parent of q to q_{new}



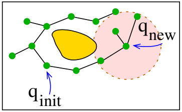
lines 3–5



line 7



lines 10–16



lines 17–20

- $cost(line(q_1, q_2))$ is cost of path from q_1 to q_2 (path by the local planner)
- $cost(q), q \in \mathcal{T}$ is cost of the path from q_{init} to q (path in $\mathcal{T}$)
- nearest neighbors Q_{near} are searched within radius r depending on the number of nodes n in the tree:

$$r = min\left\{\gamma^*_{RRT}\left(\frac{\log(n)}{n}\right)^{\frac{1}{d}}, \eta\right\}$$

$$\gamma^*_{RRT} = 2\left(1 + \frac{1}{d}\right)^{\frac{1}{d}}\left(\frac{\mu(\mathcal{C}_{\text{free}})}{\zeta_d}\right)^{\frac{1}{d}}$$

- d is the dimension of $\mathcal{C}$
- $\mu(\mathcal{C}_{\text{free}})$ is the volume of $\mathcal{C}_{\text{free}}$
- ζ_d is the volume unit ball in the $d-$dimensional Euclidean space
- η is constant given by the used local planner
- r decays with n
- r depends also on the problem instance

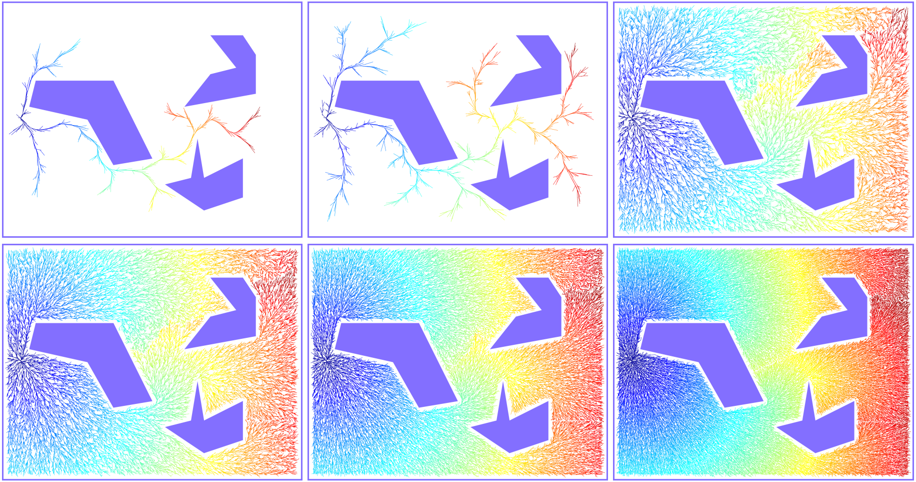
Alternative k-nearest RRT* (aka k-RRT*)

- k-nearest neighbors are selected for parent search and rewiring

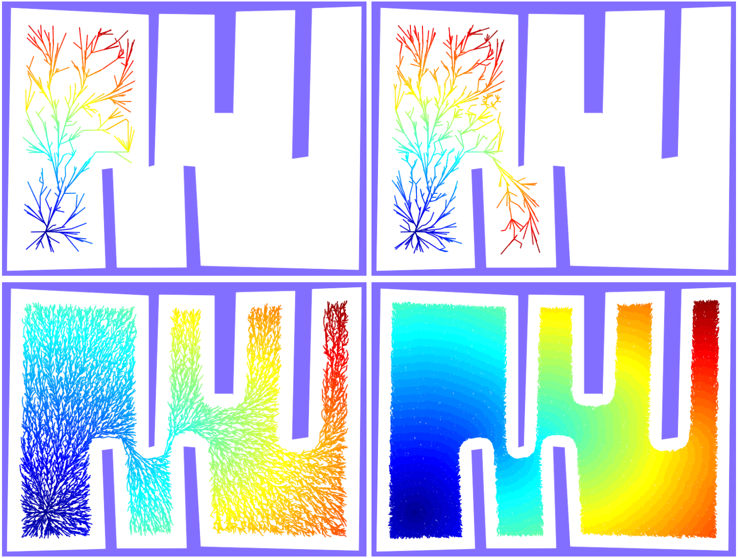
$$k = k_{RRT} \log(n)$$

$$k_{RRT} > k^*_{RRT} = e\left(1 + \frac{1}{d}\right)$$

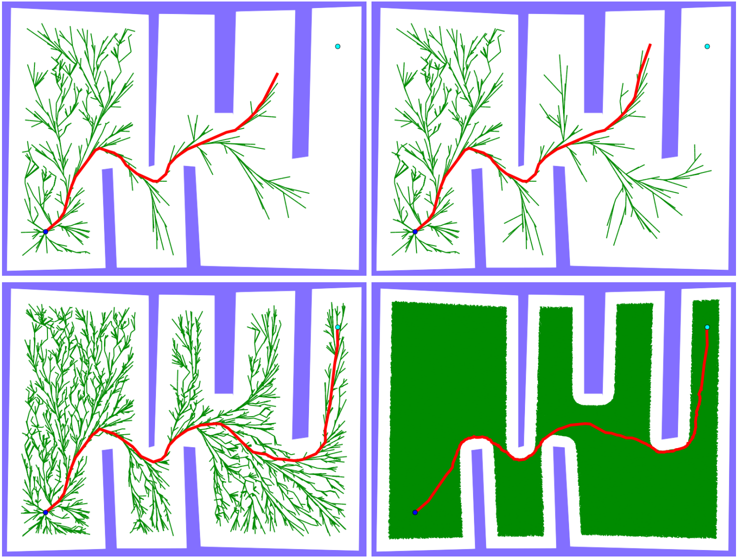
- n is the number of nodes in $\mathcal{T}$
- k-RRT* has same implementation as RRT* just line 7 is changed to Q_{near} = find k nearest neighbors in $\mathcal{T}$ towards q_{new}



Rectangle robot, rotation allowed $\rightarrow$ 3D $\mathcal{C}$



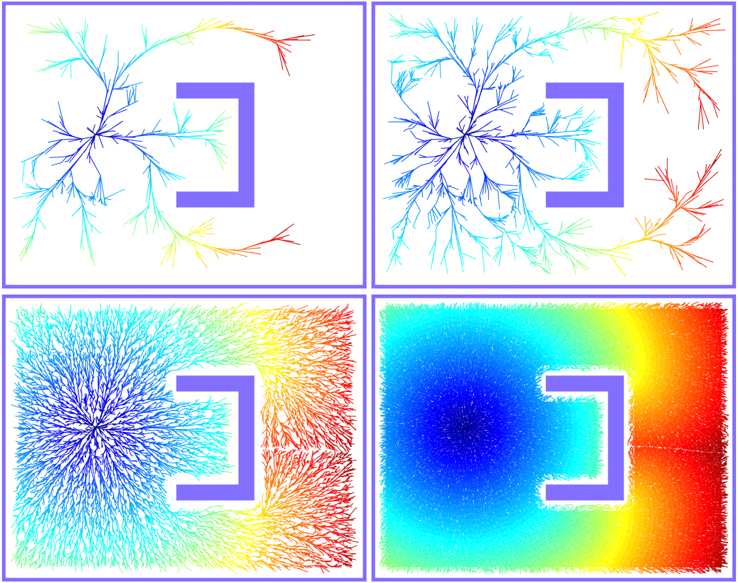
2D rectangle robot $\rightarrow$ 3D $\mathcal{C}$. The colormap shows the path length from q_{init}.
But is it really good?



2D rectangle robot $\rightarrow$ 3D $\mathcal{C}$

Depicted path demonstrates the slow convergence of the path quality

RRT*: example in 2D $\mathcal{W}$



Overview of sampling-based planners

Algorithm	**Probabilistic completeness**	**Asymptotic optimality**
RRT	Yes	No
PRM	Yes	No
sPRM	Yes	Yes
k-sPRM	No if $k = 1$	No
PRM* / k-PRM*	Yes	Yes
RRT* / k-RRT*	Yes	Yes

- If you don't need optimal solution, stay with RRT/PRM
- RRT is faster than RRT*
- RRT is way easier for implementation than RRT* (if we need an efficient implementation)
- Path quality of RRT can be improved by fast post-processing
- Asymptotic optimality is just asymptotic!

→ slow convergence of path quality

Lecture summary

- Sampling-based planning randomly samples $\mathcal{C}$
- Samples are classified as free/non-free, free samples are stored
- Multi-query vs. single-query planners
- PRM/RRT/EST and their optimal variants PRM* and RRT*