Motion planning II: sampling-based planners

Vojtěch Vonásek

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

Motion/path planning

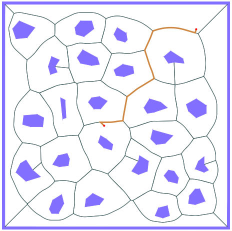
- Finding of collision-free trajectory/path for a robot
- Formulation using the configuration space $\mathcal{C}$
- C is continuous $\rightarrow$ conversion to a discrete representation (graph) $\rightarrow$ graph search
- Geometric-based methods (special cases)
  - Require an explicit representation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

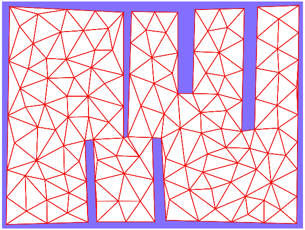
Motion planning

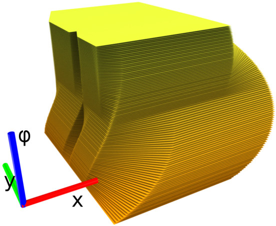
Continuous problem
$\mathcal{C}$-space specification, q_{init}, q_{goal}

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

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



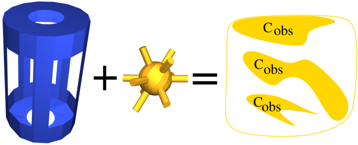




- Configuration space $\mathcal{C}$ has as many dimensions as DOFs of the robot
- Obstacles $\mathcal{C}_{\text{obs}}$ are given implicitly!

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

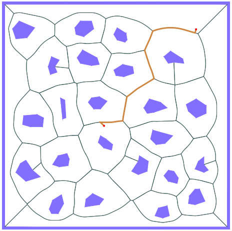
- $\mathcal{C}_{\text{obs}}$ depends both on robot and obstacles!

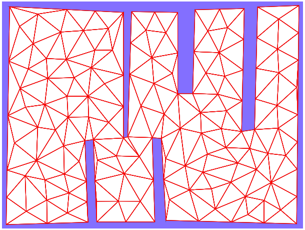


- Generally, explicit geometry/shape of $\mathcal{C}_{\text{obs}}$ is not available
- Problem of enumerating configurations in $\mathcal{C}_{\text{obs}}$
- Problem of enumerating "surface" configurations of $\mathcal{C}_{\text{obs}}$

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

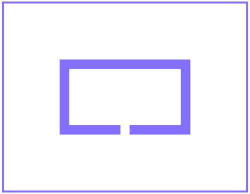
- We cannot generally/easy/fast say, what are surface/boundary configurations of $\mathcal{C}_{\text{obs}}$
- This precludes Visibility Graphs, Voronoi diagrams, Cell-decompositions to be used for high-dimensional $\mathcal{C}$-space
  - they require surface/boundary of $\mathcal{C}_{\text{obs}}$

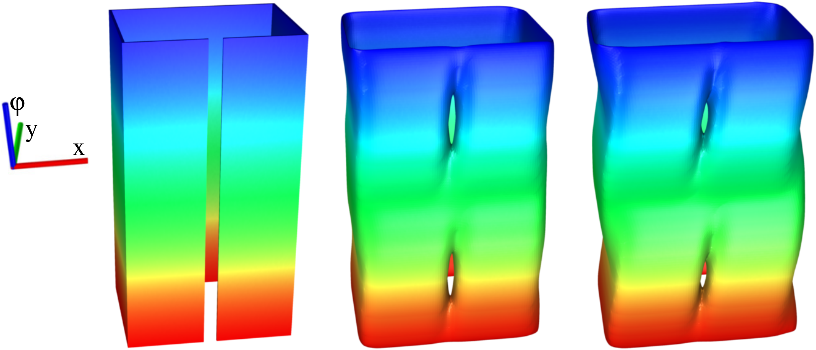




Configuration space: example I

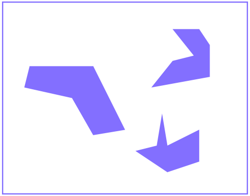
- Map: 1000×700 units
- Robot: rectangle $20 \times a$ units
- $q = (x, y, \varphi)$
- $\mathcal{C}$ visualized for $0 \leq \varphi < 2\pi$
- $\varphi = 0 \rightarrow$  $\leftarrow \varphi = 2\pi$

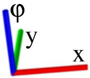


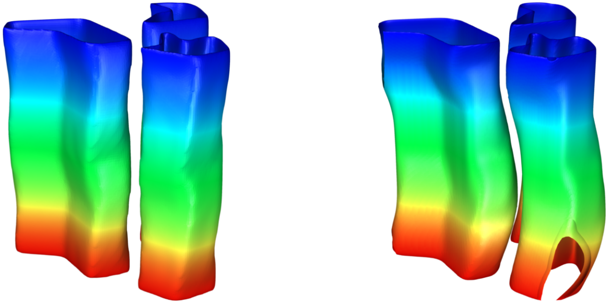


$a = 1$ $a = 60$ $a = 100$

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





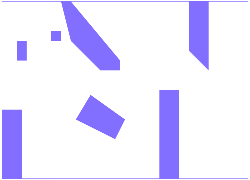


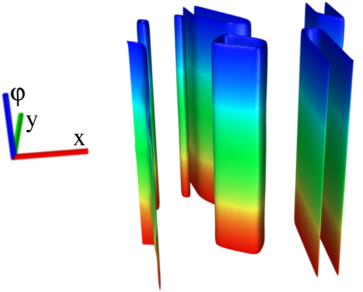
$\mathcal{A}$: rectangle 20×100 units

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

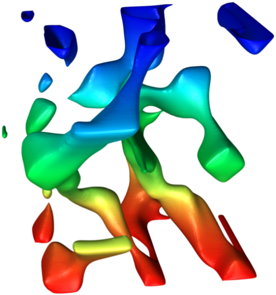
Configuration space: example III

- Map: 5000×3000 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



$\mathcal{A}$: “u”-robot

- Usually high-dimensional for 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)

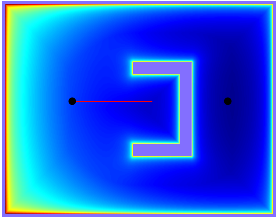
Early methods

- 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!

- Randomized path planner (RPP), 1991
  - Discrete workspace
  - Several potential fields for different control points of the robot
  - Gradient descend is performed for selected point
  - If goal is reached, algorithm terminates
  - Otherwise, different control point is selected and GD continues there
  - Escape from local minimum is performed by random walk



☛ J. Barraquand and J.-C. Latombe. Robot motion planning: a distributed representation approach. International Journal on Robotics Research, 10(6):628-649, 1991.

- ZZZ planner (1990)
  - Uses two planners: global and local
  - Global planner randomly places random goals in $\mathcal{C}_{\text{free}}$
  - Local planner uses potential field to connect these goals

☛ B. Glavina. Solving findpath by combination of goal-directed and randomized search. In IEEE International Conference on Robotics and Automation (ICRA), 1718-1723, 1990.

A bit of history III

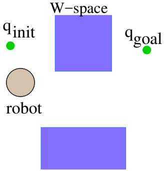
- Ariadne's clew algorithm (1998)
  - Two phase tree-based planner
  - Exploration phase: adds new configuration to tree rooted at q_{init}
  - Search phase: attempts to connect known (tree) configuration to q_{goal}
  - Both phases are solved using a genetic algorithm

☛ E. Mazer and J. M. Ahuactzin and P. Bessiere; The Ariadne's Clew Algorithm, Journal of Artificial Intelligence Research, vol 9, 1998, 295-316

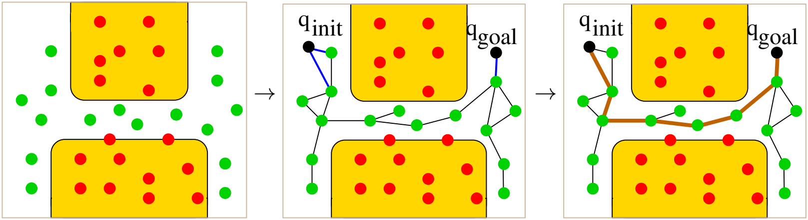
- Horsch planner (1994)
  - First roadmap-based approach: generate random samples in $\mathcal{C}_{\text{free}}$
  - Connect samples by straight-line if possible
  - If the roadmap is disconnected, random ray is shoot from one of its vertex
  - Contact configuration is added to the roadmap and connected with nearest neighbors

☛ Horsch, T. and Schwarz, F. and Tolle, H.; Motion planning with many degrees of freedom-random reflections at C-space obstacles; IEEE International Conference on Robotics and Automation (ICRA), 1994

Main idea:



- $\mathcal{C}$ 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 Roadmap Path

- 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

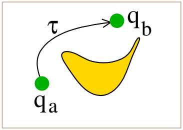
✓ Robots with many-DOFs

- 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

- Sampling-based planners rely on a “local planner”
- Given configurations $q_a \in \mathcal{C}_{\mathrm{free}}$ and $q_b \in \mathcal{C}_{\mathrm{free}}$, local planner attempts to find a path τ:



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

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

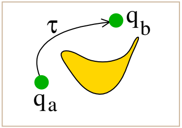
- The definition of “local planning” is same as motion planning

$\rightarrow$ same complexity as motion planning!

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

Exact local planners

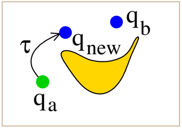
- For certain systems, BVP can be solved analytically
- Example: car-like without backward motions → Dubins car



Exact local planner

Approximate local planners

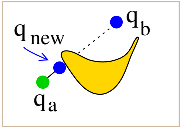
- Path τ connects q_a with q_{new} that is near-enough from q_b
- Computation e.g. using forward motion model and integration over time Δt



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



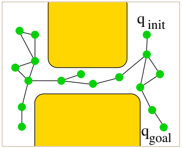
Straight-line

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

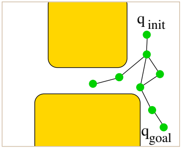
Single-query methods

- Roadmap is built only to answer a single start/goal query
- The search of $\mathcal{C}$ ends as soon as the query can be answered
- Rapidly-exploring Random Trees (RRT), Expansive-space Tree (EST) + their variants

✓ Practically faster for single-query

✗ Any subsequent planning requires novel search of $\mathcal{C}$

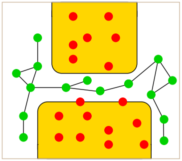
✗ Slow for multi-query planning



Single-query roadmap

Probabilistic Roadmaps (PRM)

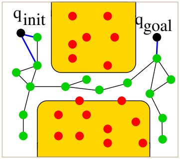
- Two-phase method: learning phase and query phase



Learning phase

Learning phase

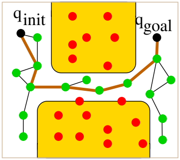
- Random samples are generated in $\mathcal{C}$
- Samples are classified as free/non-free; free samples are stored
- Each sample is connected to its near neighbors by a local planner
- Final roadmap may contain cycles



Query phase

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



Path

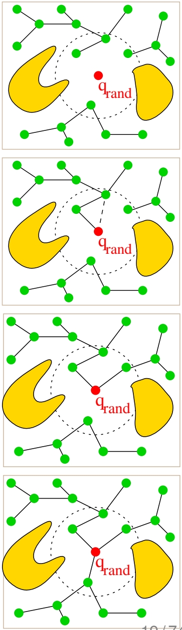
☛ L. E. Kavraki, P. Svestka, et al., "Probabilistic roadmaps for path planning in high-dimensional configuration spaces,". IEEE Trans. on Robotics and Automation, 12(4), 1996.

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}

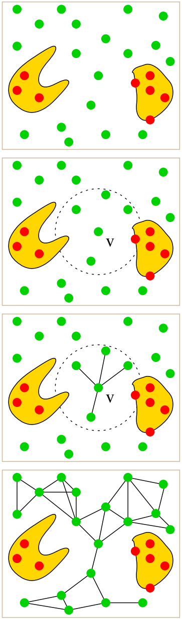


☛ L. E. Kavraki, P. Svestka, et al., "Probabilistic roadmaps for path planning in high-dimensional configuration spaces,". IEEE Trans. on Robotics and Automation, 12(4), 1996.

Simplified PRM (sPRM)

- Separate sampling and roadmap connection
- Each node is connected to it's nearest neighbors
- Roadmap can contains cycles
- Analysis of sPRM (completeness and optimality) is available

```
V = ∅; E = ∅                                   // vertices and edges
while |V| < n do  // generating n collision-free samples
    q_rand = generate random sample in C
    if q_rand is collision-free then
        V = V ∪ {q_rand}

foreach v ∈ V do                // connecting samples to roadmap
    V_n = V.neighborhood(v)
    foreach u ∈ V_n, u ≠ v do
        if connect(u, v) then                  // local planner
            E = E ∪ {(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 *V.neighborhood* function
- Several variants were proposed an analyzed

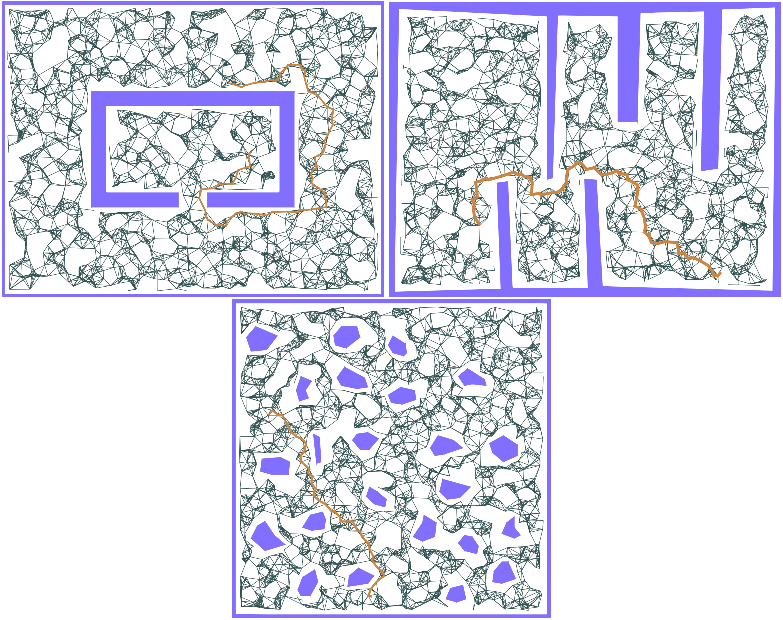
k-nearest sPRM (aka *k*-sPRM)

- *V.neighborhood* provides k nearest neighbors from q_{rand}
- Probabilistically complete if $k \neq 1$
- Is not asymptotically optimal
- Usually $k = 15$

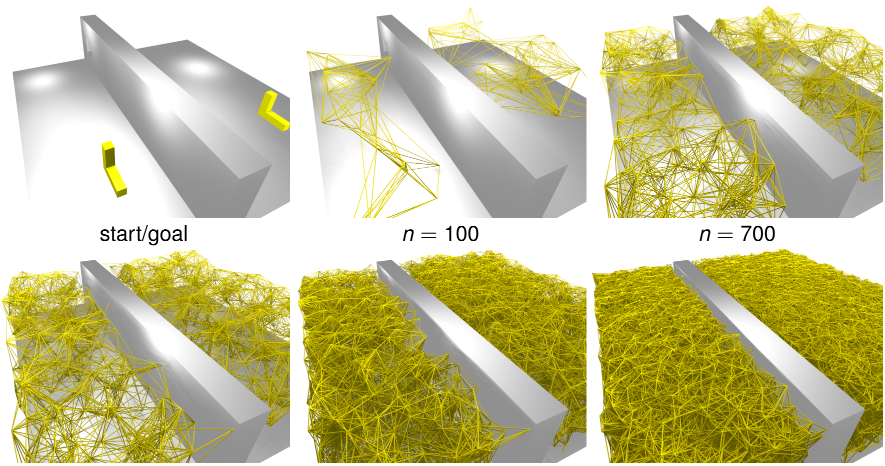
Variable radius sPRM

- *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

sPRM example 2D $\mathcal{W}$



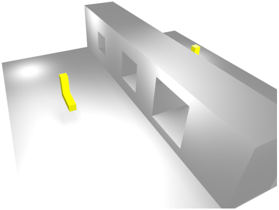
sPRM example 3D $\mathcal{W}$



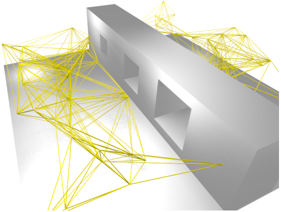
start/goal, $n = 100$, $n = 700$, $n = 1500$, $n = 8000$, $n = 50000$

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

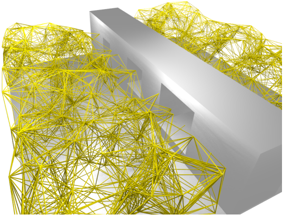
sPRM example 3D $\mathcal{W}$



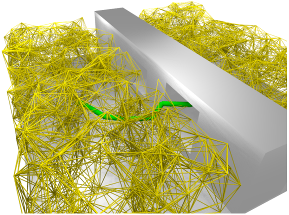
start/goal



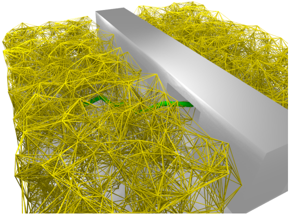
$n = 100$



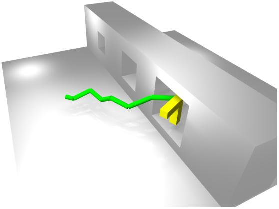
$n = 1000$



$n = 2100$



$n = 4100$

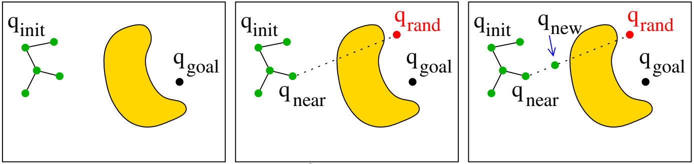


solution

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

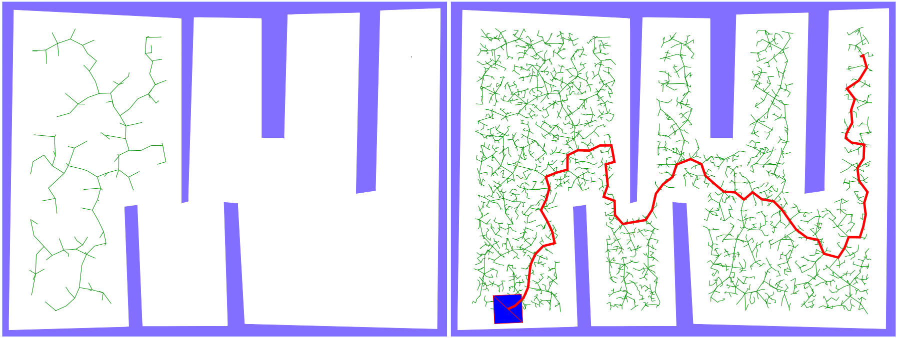
initialize tree $\mathcal{T}$ with q_{init}
for $i = 1, \ldots, I_{max}$ **do**
    q_{rand} = generate randomly in $\mathcal{C}$
    q_{near} = find nearest node in $\mathcal{T}$ towards q_{rand}
    q_{new} = localPlanner from q_{near} towards q_{rand}
    if *canConnect*($q_{\text{near}}, q_{\text{new}}$) **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

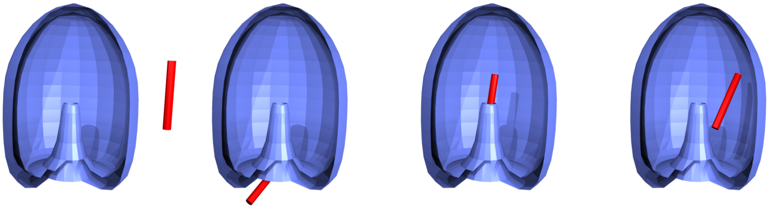
☛ LaValle:, S. M. Rapidly-exploring random trees: a new tool for path planning". Technical report, Iowa State University, 1998

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

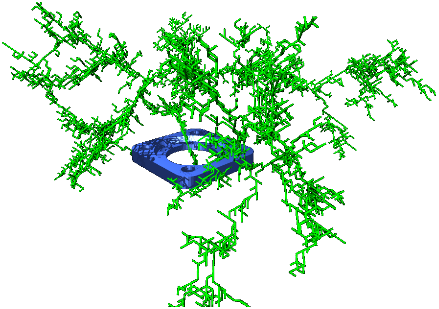


- 2D robot, rotation allowed $\rightarrow$ 3D $\mathcal{C}$
- Why the tree does not “touch” the obstacles?

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



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

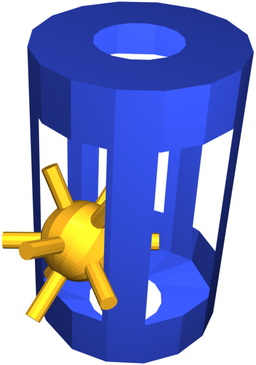


- 3D Flange benchmark

  `parasol.tamu.edu/groups/amatogroup/benchmarks/`

- 3D robot in 3D space $\rightarrow$ 6D $\mathcal{C}$

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



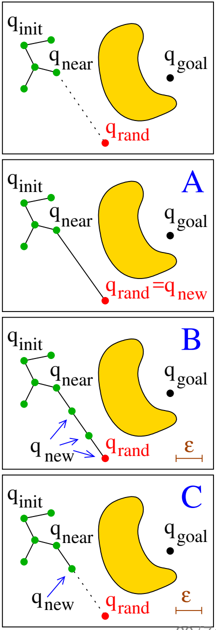
- Hedgehog in the cage
  `parasol.tamu.edu/groups/amatogroup/benchmarks/`
- First appereance in end of 19th century
- Popularization in books about youth by J. Foglar
- 3D robot, free-flying in 3D space $\rightarrow$ 6D $\mathcal{C}$
- Extremely difficult to solve (we will discuss later why)

RRT: tree expansion types

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

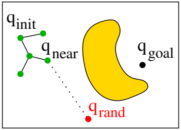
Variants:

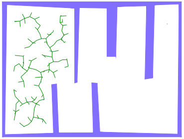
- **A** If S is collision-free, expand the tree only by $q_{\text{new}} = q_{\text{rand}}$
  - Creates long segments, fast exploration of $\mathcal{C}$
  - Requires nearest-neighbor search to consider point-segment distance
  - Requires connection in the middle of line-segment
- **B** If S is collision-free,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

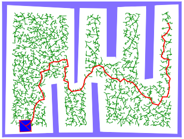


RRT: properties

- 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 $q'_{\text{goal}} \in \mathcal{T}$ 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

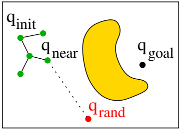


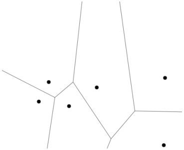




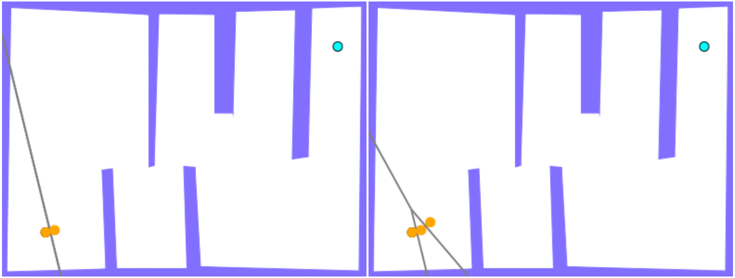
- Why the tree does not grow to itself?
- Why does it “rapidly” explore the $\mathcal{C}$-space?

. . . because of Voronoi bias!

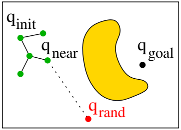


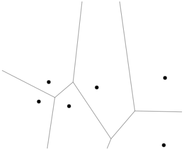


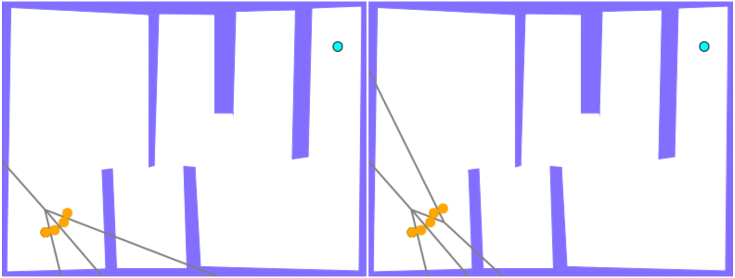
- RRT prefers to expand $\mathcal{T}$ towards unexplored areas of $\mathcal{C}$
- This is caused by **Voronoi bias**:
  - q_{rand} is generated **uniformly** in $\mathcal{C}$
  - $\mathcal{T}$ is expanded from **nearest** node in $\mathcal{T}$ **towards** q_{rand}
  - The probability that a node $q \in \mathcal{T}$ 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)

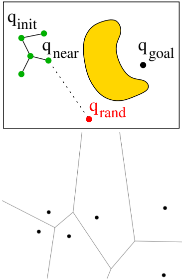


- RRT prefers to expand $\mathcal{T}$ towards unexplored areas of $\mathcal{C}$
- This is caused by **Voronoi bias**:
  - q_{rand} is generated **uniformly** in $\mathcal{C}$
  - $\mathcal{T}$ is expanded from **nearest** node in $\mathcal{T}$ **towards** q_{rand}
  - The probability that a node $q \in \mathcal{T}$ 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)

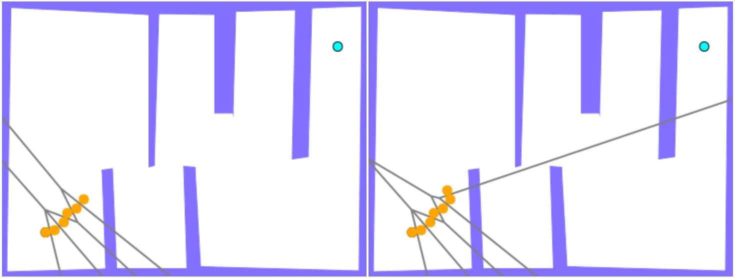






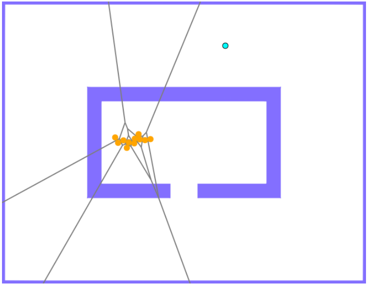


- RRT prefers to expand $\mathcal{T}$ towards unexplored areas of $\mathcal{C}$
- This is caused by **Voronoi bias**:
  - q_{rand} is generated **uniformly** in $\mathcal{C}$
  - $\mathcal{T}$ is expanded from **nearest** node in $\mathcal{T}$ **towards** q_{rand}
  - The probability that a node $q \in \mathcal{T}$ 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)

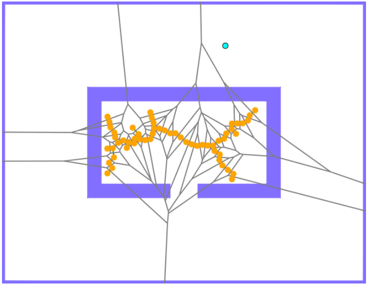


RRT: Voronoi bias II

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



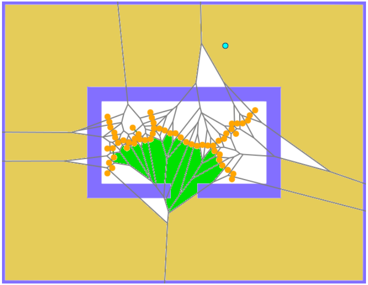
iteration 10, tree size 10



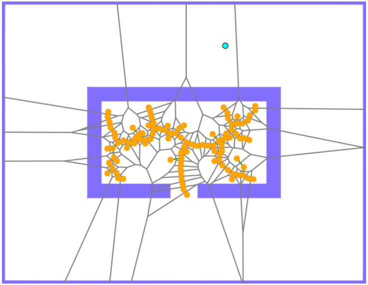
iteration 70, tree size $\sim$ 60

- 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!

- 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)

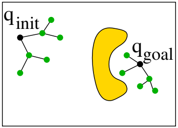
FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE — MRS MULTI-ROBOT SYSTEMS GROUP

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

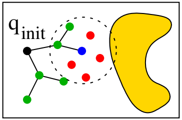
```
q' = select node from T with probability w(q)^-1
Q = k random points around q' :  Q = {q ∈ C | ϱ(q, q') < d}
foreach q ∈ Q do
    w(q) = compute weight of the sample q
    if rand() < w(q)^-1 and connect(q, q') then
        T.addNode(q)
        T.addEdge(q', q)
```

- $w(q)$ 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

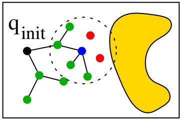
☛ D. Hsu, J.-C. Latomber et al. Path planning in expansive configuration spaces. Int. Journal of Comp. Geometry and Applications, 9(4-5), 1999



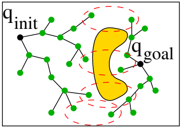
$\mathcal{T}_i$ and $\mathcal{T}_g$



q', samples Q

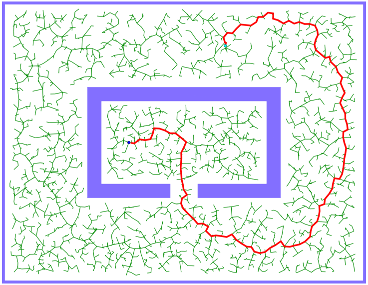


connected,ignored

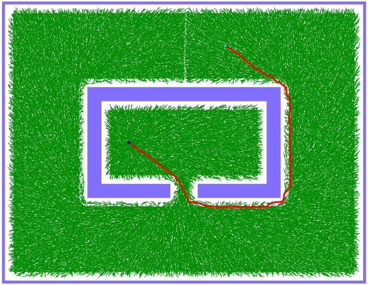


pairs for tree connection

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

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

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

- 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

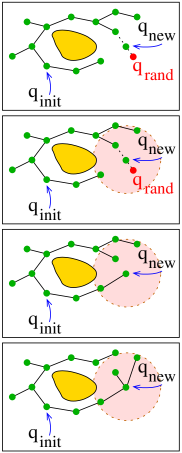
k-nearest PRM* algorithm (aka k-PRM*)

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

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

RRT*: overview

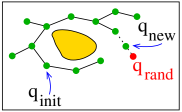
- Optimal version of RRT
- For each node, a cost of the path from q_{init} to that node is established
- RRT* has improved tree expansion and nearest-neighbor search
- Tree expansion by node q_{new}
  - Parent of q_{new} is optimized to minimize cost at q_{new}
  - After q_{new} is connected to tree, 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*



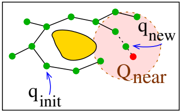
☛ S. Karaman, and E. Frazzoli. "Sampling-based algorithms for optimal motion planning." The international journal of robotics research 30.7 (2011): 846-894.

RRT*: algorithm

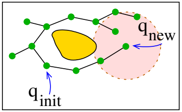

```
1  initialize tree T with q_init
2  for i = 1, ..., I_max do
3      q_rand = generate randomly in C
4      q_near = find nearest node in T towards q_rand
5      q_new = localPlanner from q_near towards q_rand
6      if q_new is collision-free then
7          Q_near = T.neighborhood(q_new, r)
8          T.addNode(q_new)                       // new node to tree
9          q_best = q_near              // best parent of q_new so far
10         c_best = cost(q_near) + cost(line(q_near, q_new))
11         foreach q ∈ Q_near do
12             c = cost(q) + cost(line(q, q_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         T.addEdge(q_best, q_new)      // tree connected to q_new
17         foreach q ∈ Q_near do                      // rewiring
18             c = cost(q_new) + cost(line(q_new, q))
19             if canConnect(q_new, q) and c < cost(q) then
20                 change parent of q to q_new
```



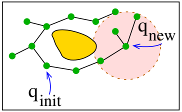
lines 3–5



line 7



lines 10–16



lines 17–20

- See next slide for explanation of functions/variables

RRT* with variable neighborhood

- $cost(line(q_1, q_2))$ 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*)

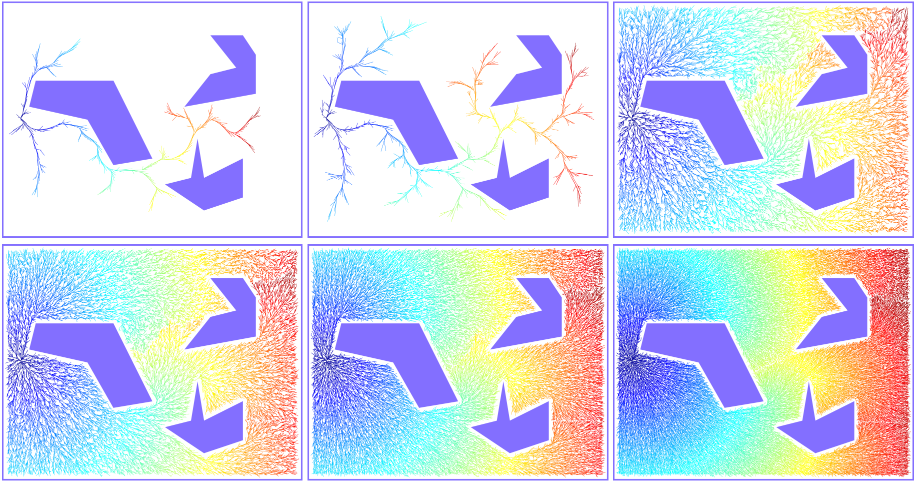
- k-nearest neighbors are selected for parent search and rewiring

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

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

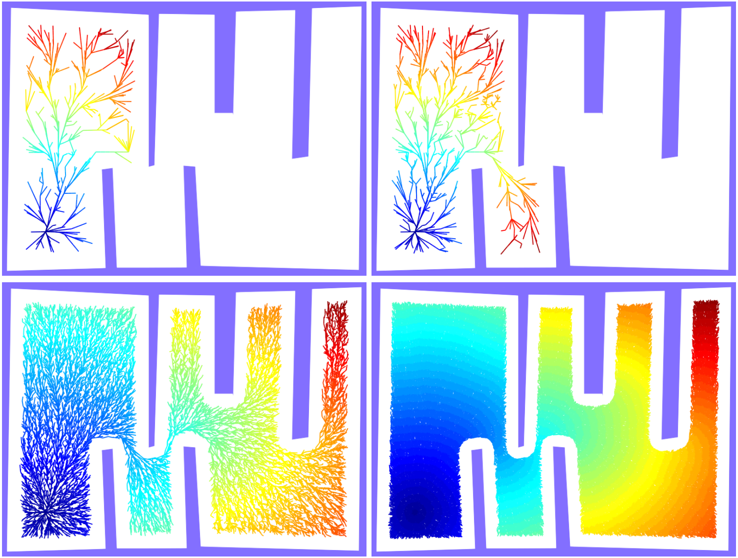
- n is the number of nodes in $\mathcal{T}$
- k-RRT* has same implementation as RRT* just line 7 is changed to Q_{near} = find k nearest neighbors in $\mathcal{T}$ towards q_{new}

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



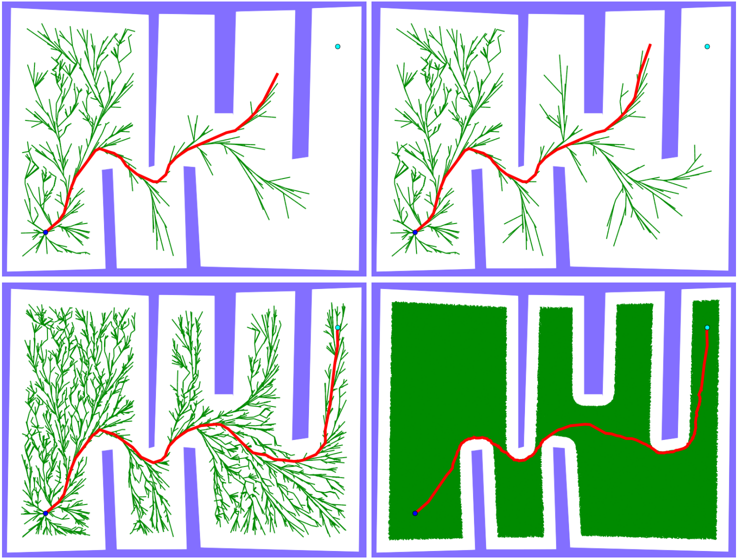
Rectangle robot, rotation allowed $\rightarrow$ 3D $\mathcal{C}$

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



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

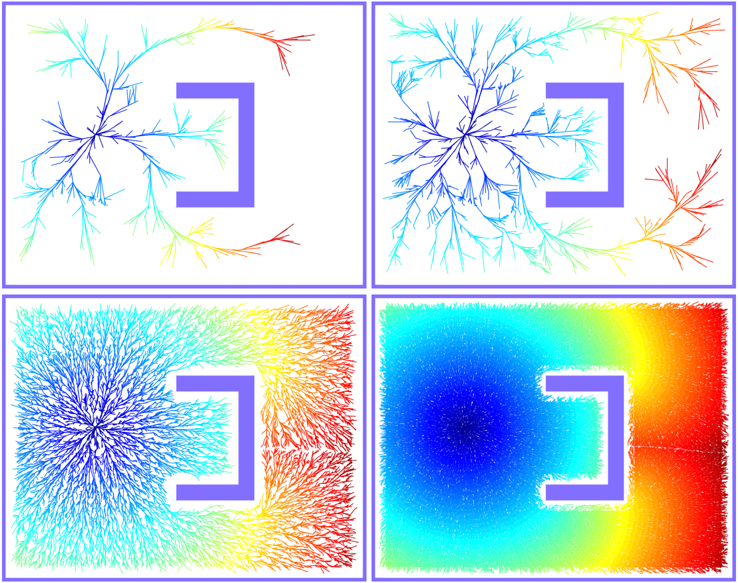
RRT*: example in 2D $\mathcal{W}$



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

Depicted path demonstrates the slow convergence of the path quality

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



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*