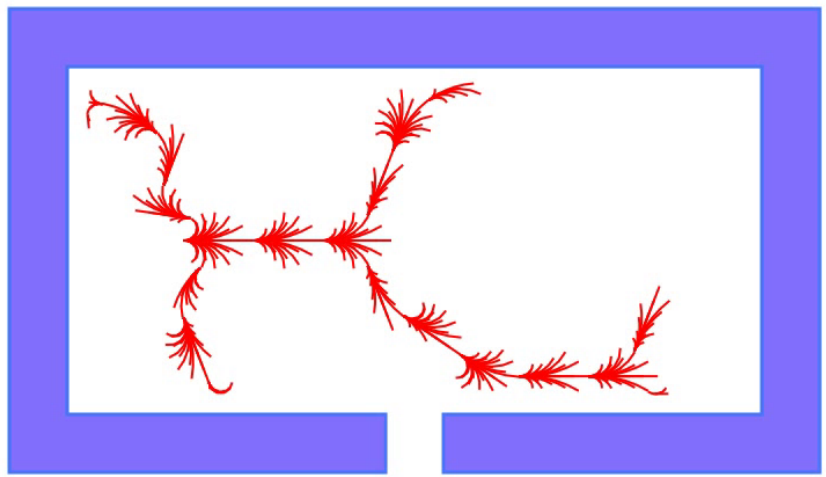
Motion planning: sampling-based planners I

Vojtěch Vonásek

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

Planning for “car-like” vehicle



Summary of the last lecture

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 $\mathcal{C}_{\text{obs}}$
  - For point/disc robots (if $\mathcal{C}$ is sames as $\mathcal{W}$)
  - Visibility graphs, Voronoi diagrams, . . .

Motion planning

Continuous problem

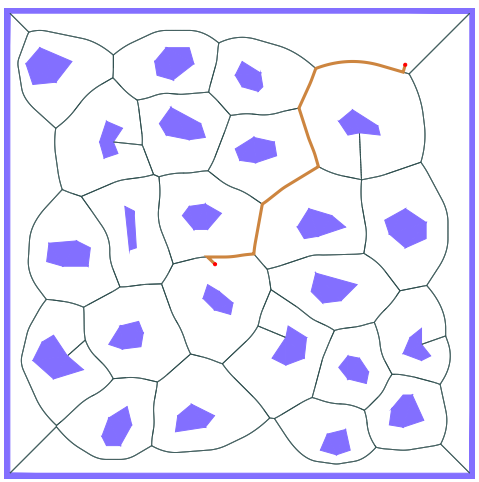
$\mathcal{C}$-space specification, q_{init}, q_{goal}

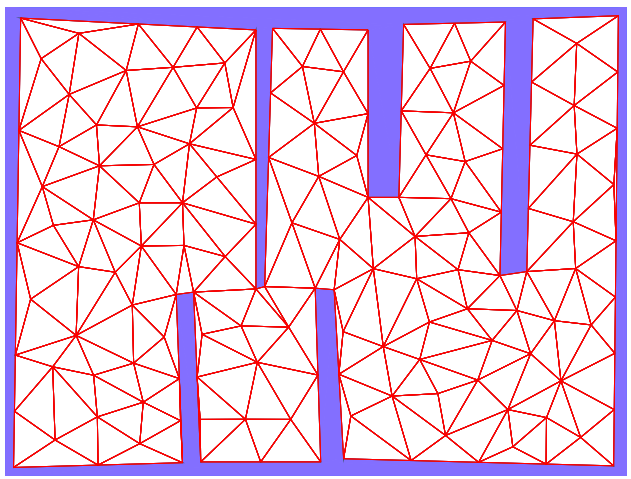
Discretization of $\mathcal{C}$

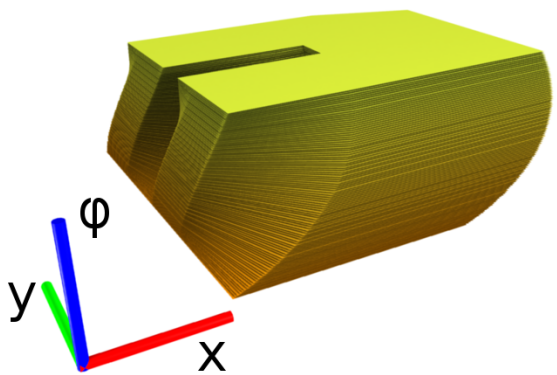
Explicit construction random/deterministic sampling of $\mathcal{C}$

Graph search

Dijkstra, A*, D*, . . .





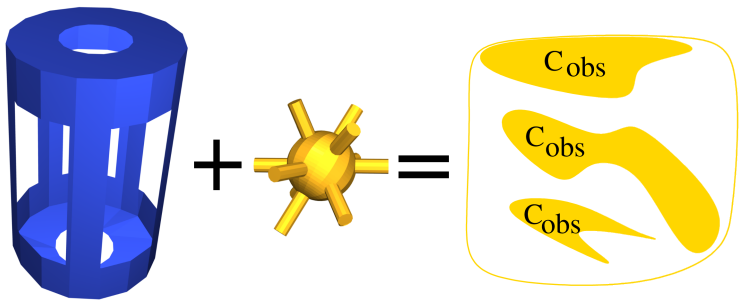


Configuration space

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

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

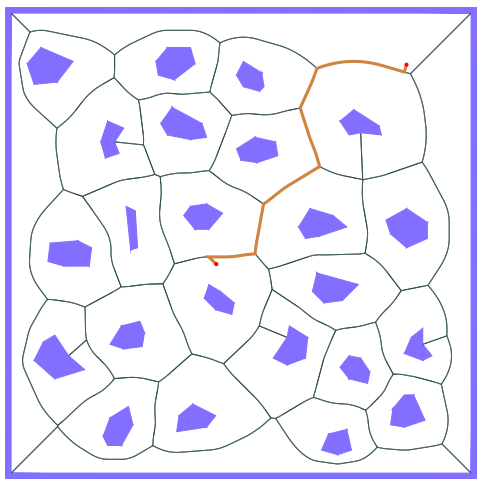
- $\mathcal{C}_{\text{obs}}$ depends both on robot and obstacles!

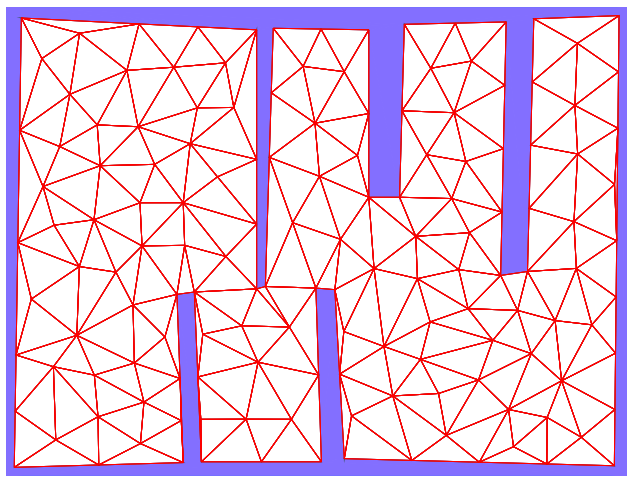


- Generally, explicit geometry/shape of $\mathcal{C}_{\text{obs}}$ is not available
- Problem of enumerating configurations in $\mathcal{C}_{\text{obs}}$
- Problem of enumerating “surface” configurations of $\mathcal{C}_{\text{obs}}$

Configuration space

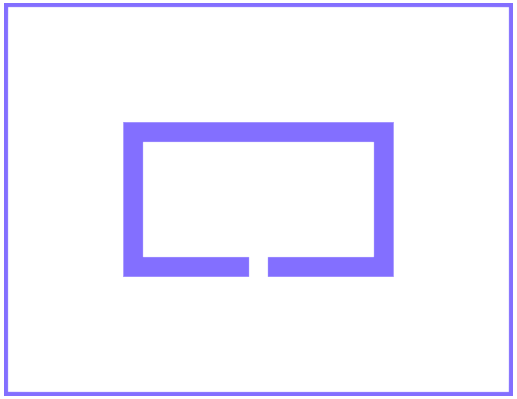
- We cannot generally/easy/fast say what are surface/boundary configurations of $\mathcal{C}_{\text{obs}}$
- This precludes combinatorial path planners (e.g., Visibility Graphs, Voronoi diagrams, Cell-decompositions, . . .) to be used for high-dimensional $\mathcal{C}$-space
  - they require surface/boundary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

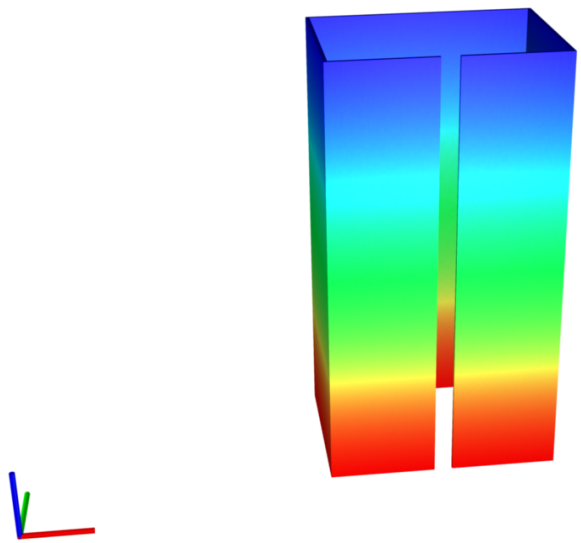




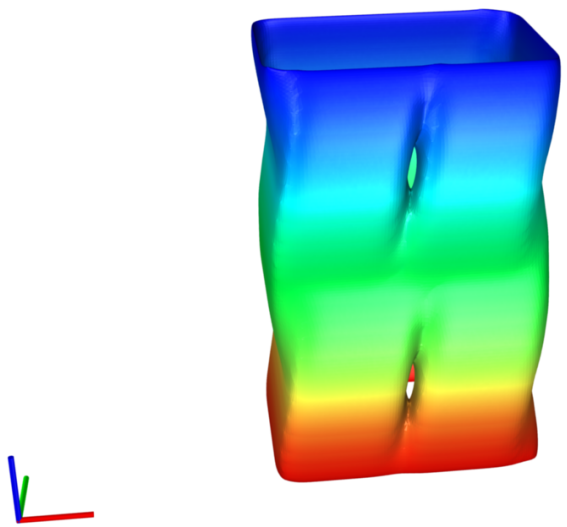
Configuration space: example I

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





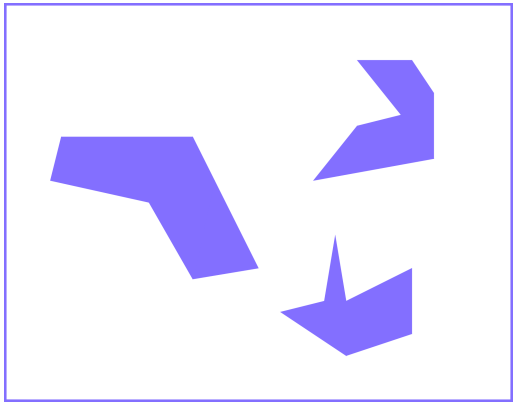
$a = 1$

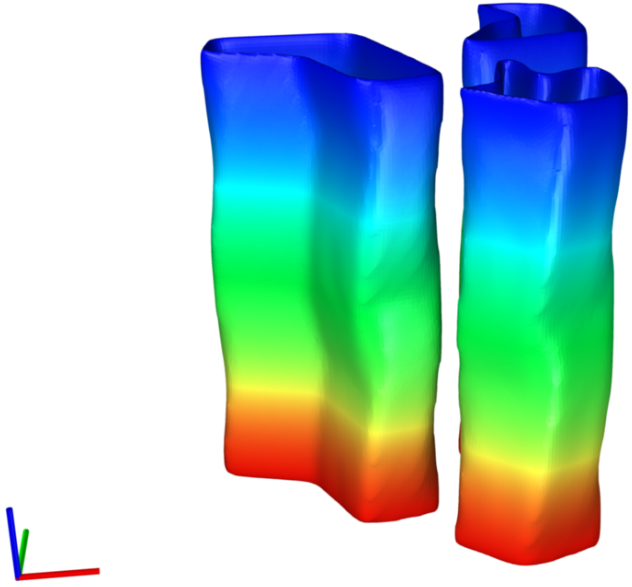


$a = 100$

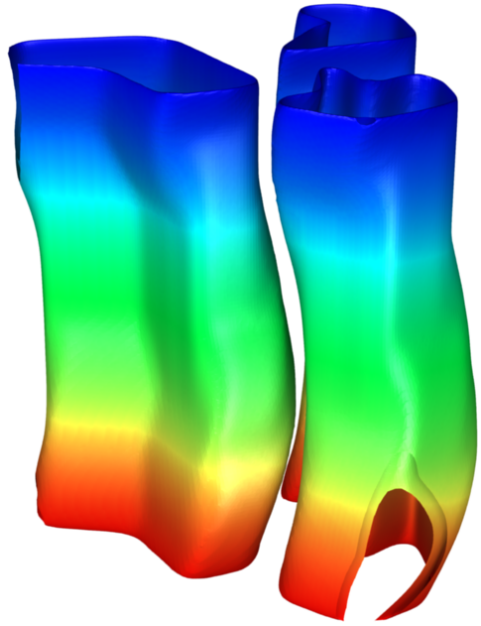
Configuration space: example II

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





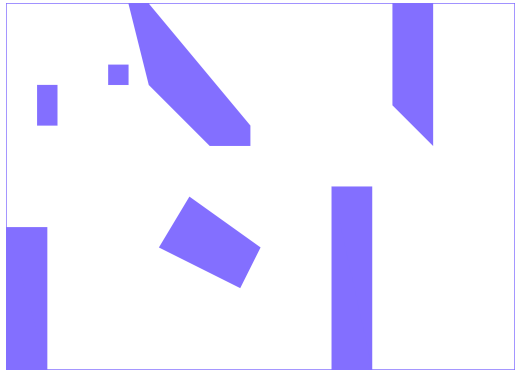
$\mathcal{A}$: rectangle 20 $\times$ 100 units

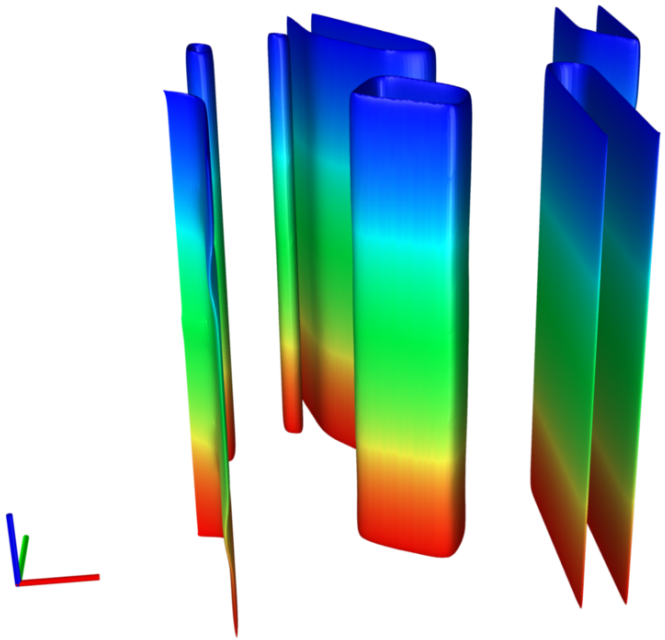


$\mathcal{A}$: equilateral triangle, side 100 units
(right-bottom “hole” caused by rendering clip)

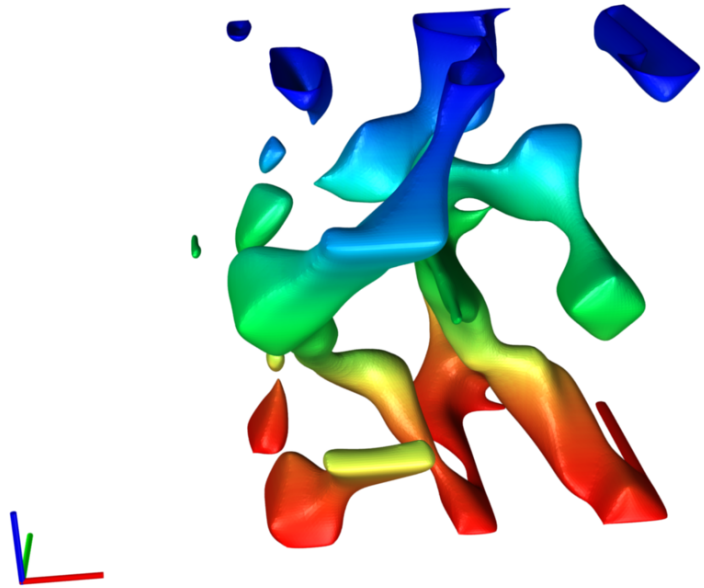
Configuration space: example III

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

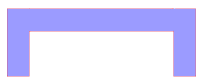




$\mathcal{A}$: rectangle 20 $\times$ 100 units



$\mathcal{A}$: “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!

Milestones in motion planning

- Dijkstra's algorithm[1], 1959
- A*[2], 1968
- Configuration space[3], 1983
- Era of combinatorial planning, 1980s–1990s
- First planners using randomization, early 1990s
- Probabilistic roadmaps (PRM)[4], 1995
- Rapidly-exploring Random Tree (RRT)[5], 1998
- Asymptotically optimal sampling-based planning (RRT*, PRM*)[6], 2011

[1] E. W. Dijkstra. "A Note on Two Problems in Connexion with Graphs". In: *Numerische Mathematik* 1.1 (Dec. 1959), pp. 269–271. DOI: 10.1007/BF01386390.

[2] P. E. Hart et al. "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

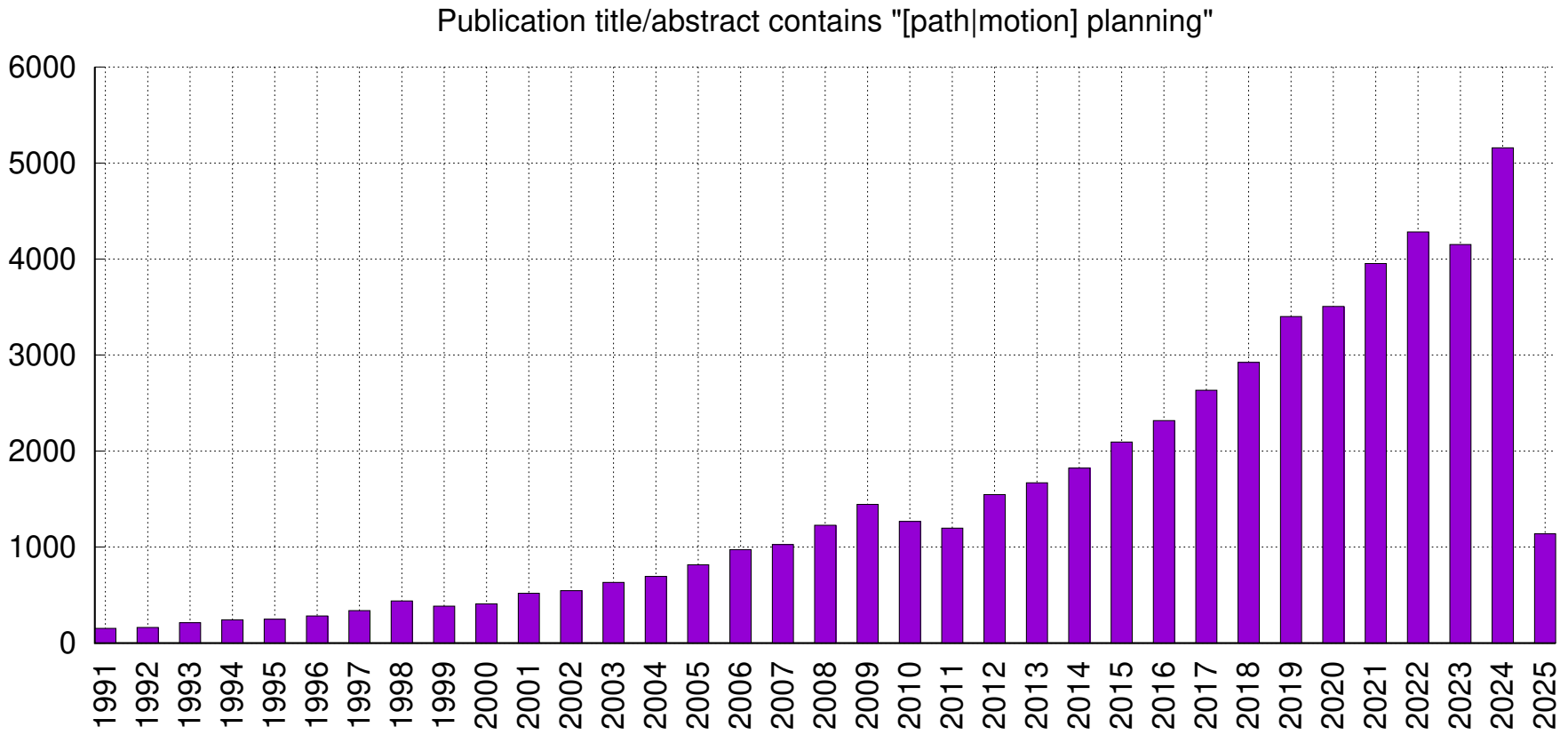
[3] Lozano-Perez. "Spatial Planning: A Configuration Space Approach". In: *IEEE Transactions on Computers* C-32.2 (1983), pp. 108–120. DOI: 10.1109/TC.1983.1676196.

[4] L. E. Kavraki et al. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: *IEEE ICRA* (1995), pp. 1555–1562.

[5] S. M. LaValle. "Rapidly-exploring random trees : a new tool for path planning". In: *The annual research report* (1998).

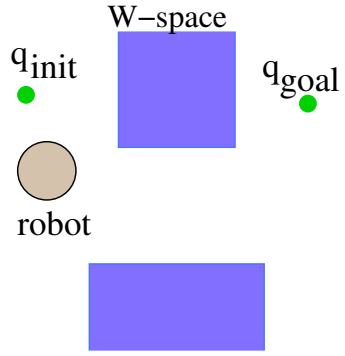
[6] S. Karaman and E. Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *The international journal of robotics research* 30.7 (2011), pp. 846–894.

Milestones in motion planning

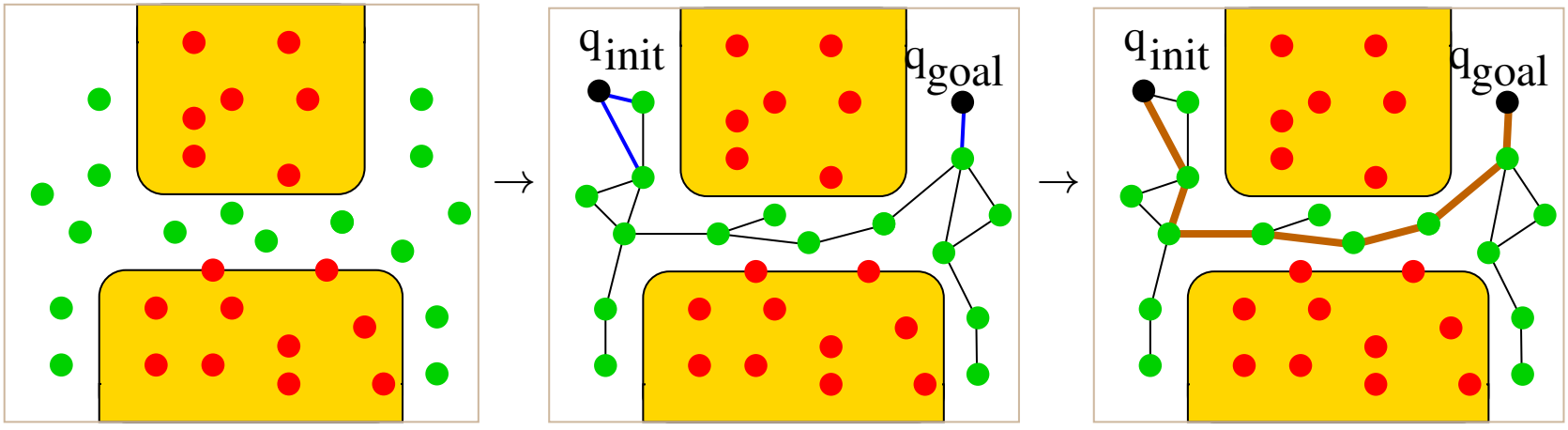


Sampling-based motion planning

Principle:



- $\mathcal{C}$ is randomly sampled (sample is $q \in \mathcal{C}$
- 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 a discretized image of $\mathcal{C}_{\text{free}}$
- Graph-search in the roadmap



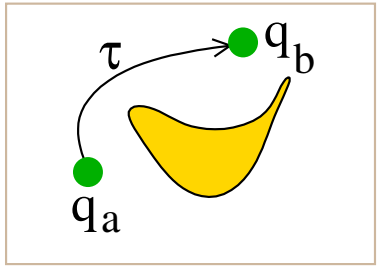
Sampling → Roadmap → Path

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:

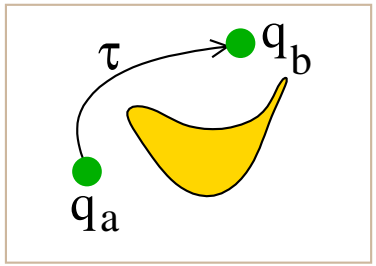
- The definition of "local planning" is same as motion planning

$\rightarrow$ same complexity as motion planning!

Local planners

Exact local planners

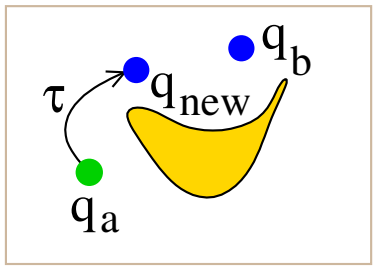
- For certain systems, BVP can be solved analytically
- Example: car-like without backward motions $\rightarrow$ Dubins car



Exact local planner

Approximate local planners

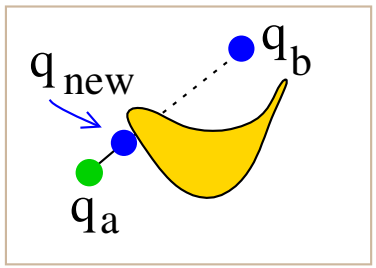
- Path τ connects q_a with q_{new} that is near-enough from q_b
- Computation e.g. using forward motion model and integration over time Δt



Approximate

Straight-line local planners

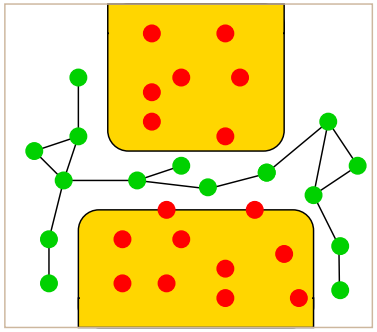
- Connects q_a and q_b by line-segment
- Check the collisions of the line-segment
- Connect q_a with the first contact configuration q_{new} or with q_b if no collision occurs
- Suitable for systems without kinematic/dynamic constraints



Straight-line

Learning phase

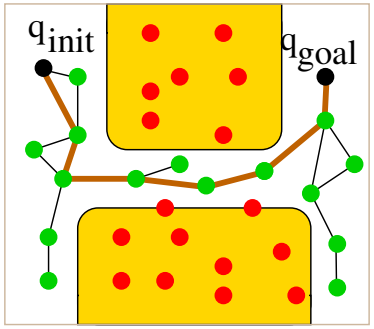
- Random samples are generated in $\mathcal{C}$
- Samples are classified as free/non-free; free samples are stored
- Each sample is connected to its near neighbors by a local planner
- Final roadmap may contain cycles



Learning 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
- More details in[7]



Query phase

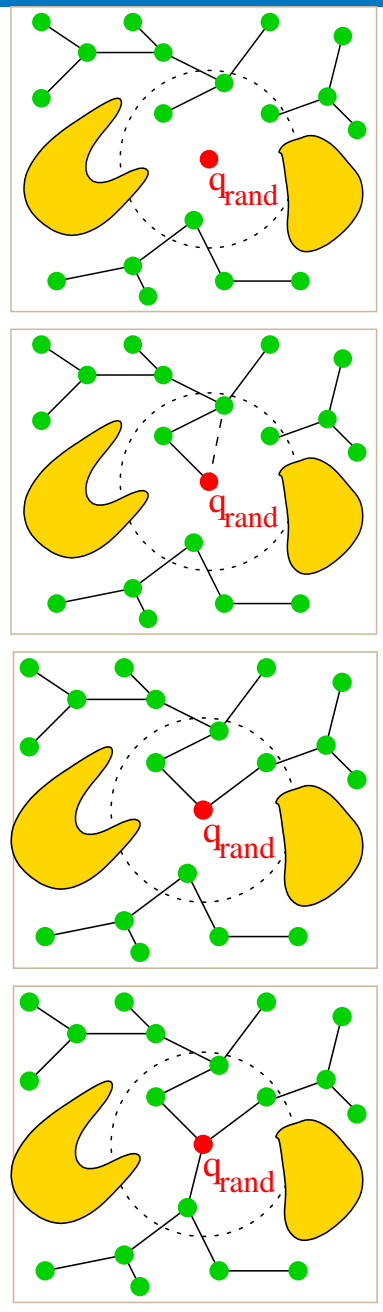
[7] L. E. Kavraki et al. “Probabilistic roadmaps for path planning in high-dimensional configuration spaces”. In: *IEEE ICRA* (1995), pp. 1555–1562.

Original PRM

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

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

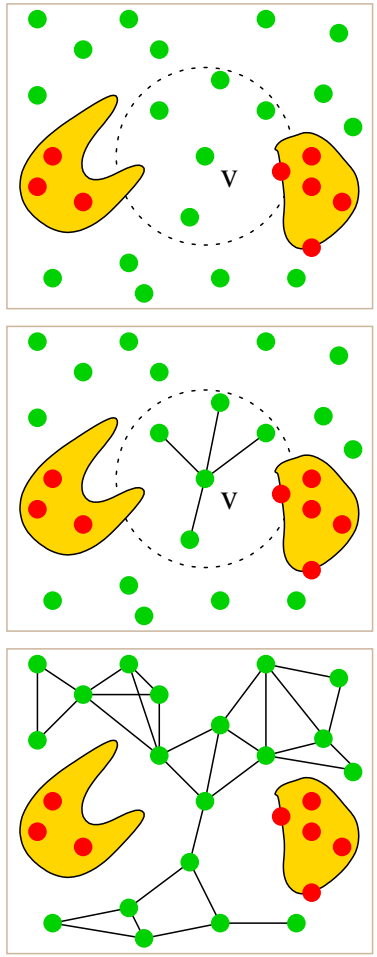
- *neighborhood** returns q by increasing distance from q_{rand}



Simplified PRM (sPRM)

- Separate sampling and roadmap connection
- Each node is connected to its nearest neighbors

```
1  V = ∅; E = ∅                                   // vertices and edges
2  while |V| < n do                  // generating n collision-free
     samples
3      q_rand = generate random sample in C
4      if q_rand is collision-free then
5          V = V ∪ {q_rand}
6  foreach v ∈ V do                  // connecting samples to roadmap
7      V_n = V.neighborhood(v)
8      foreach u ∈ V_n, u ≠ v do
9          if connect(u, v) then                  // local planner
10             E = E ∪ {(u, v)}
11 G = (V, E)                                     // final roadmap
```



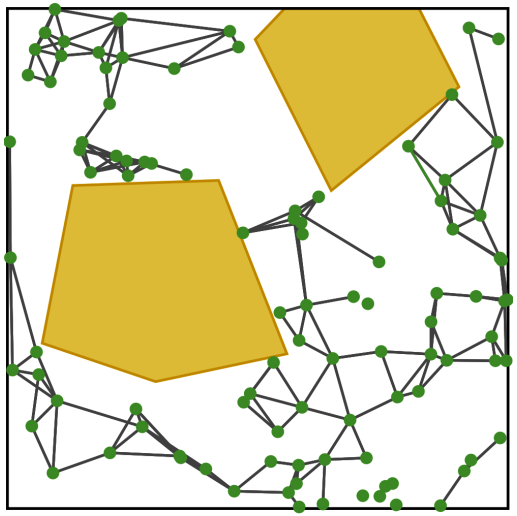
- Roadmap can contains cycles[8]

[8] S. Karaman and E. Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *The international journal of robotics research* 30.7 (2011), pp. 846–894.

sPRM: variants and properties

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

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

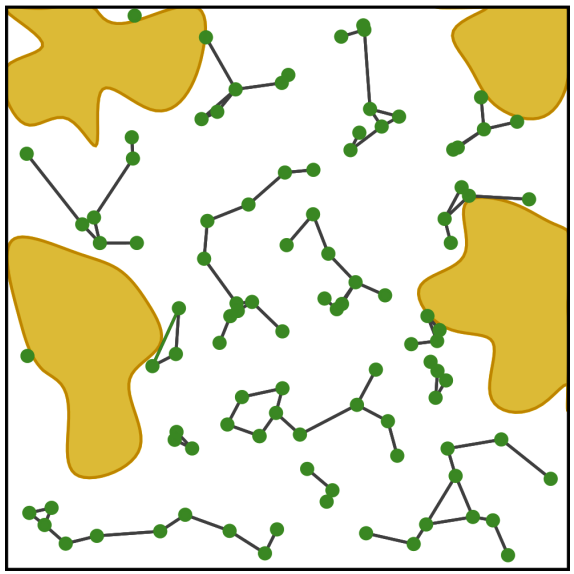


- *V*.*neighborhood* provides *k* nearest neighbors from each node of the roadmap
- Probabilistically complete if $k \neq 1$
- Is not asymptotically optimal
- Usually $k = 15$

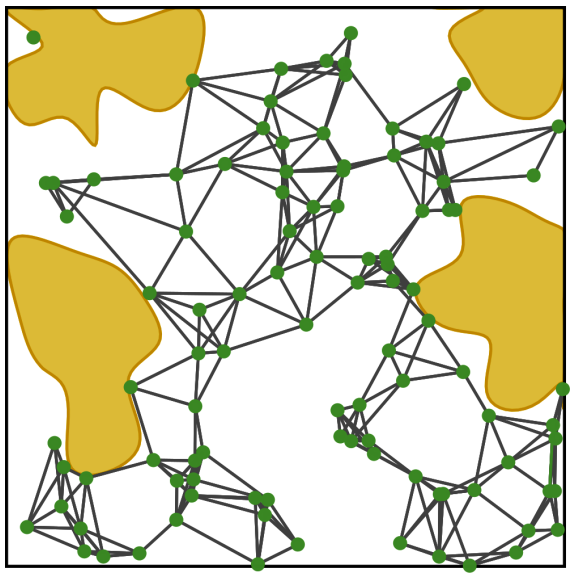
Variable radius sPRM

- *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

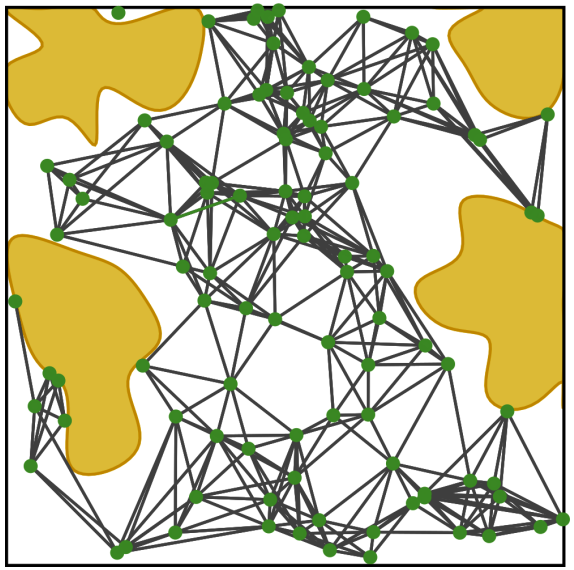
sPRM: influence of nearest neighbors



$k = 1$

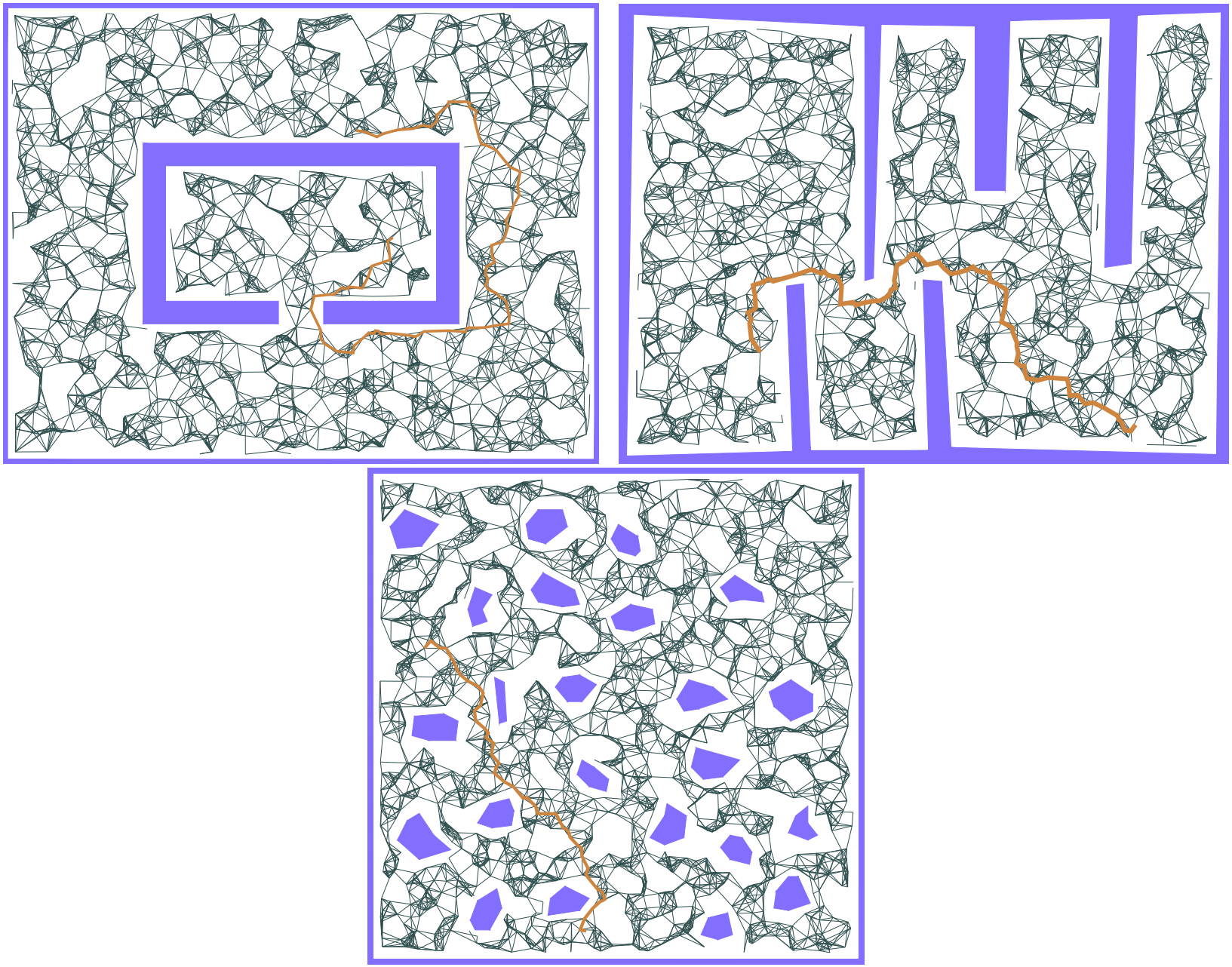


$k = 3$

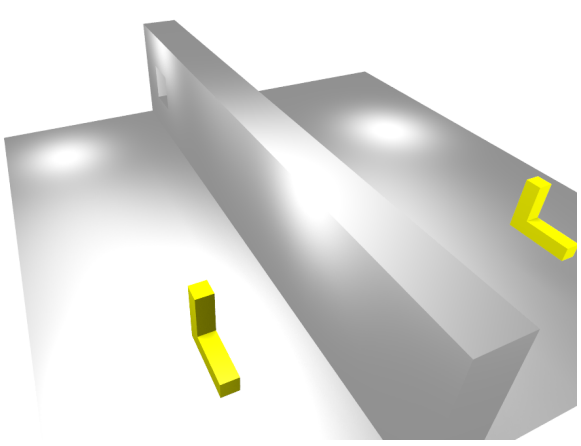


$k = 5$

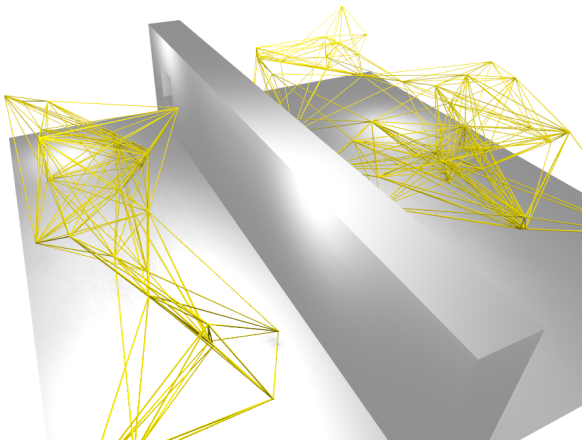
sPRM example 2D $\mathcal{W}$



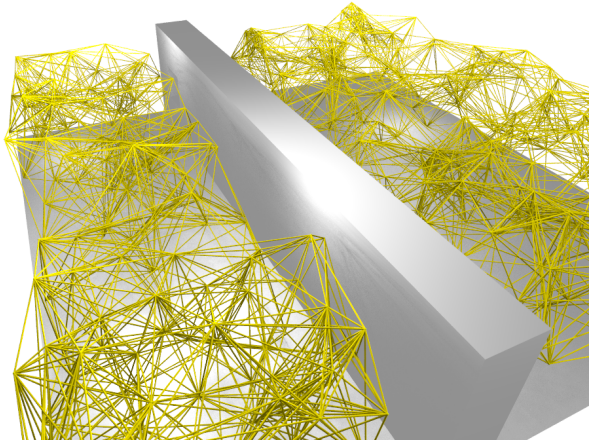
sPRM example 3D $\mathcal{W}$



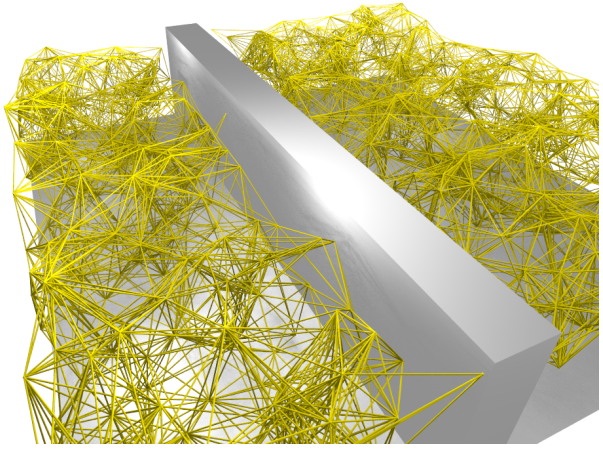
start/goal



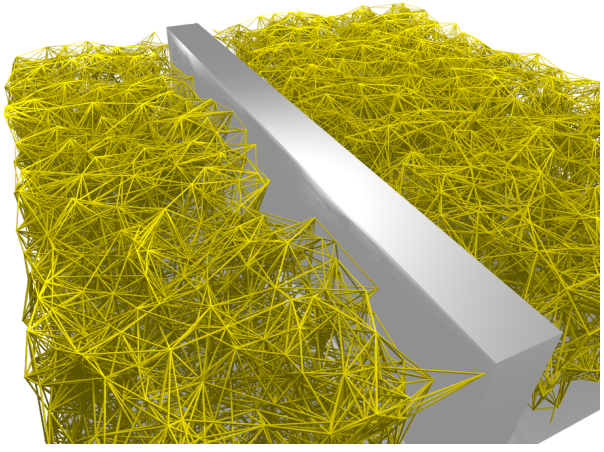
$n = 100$



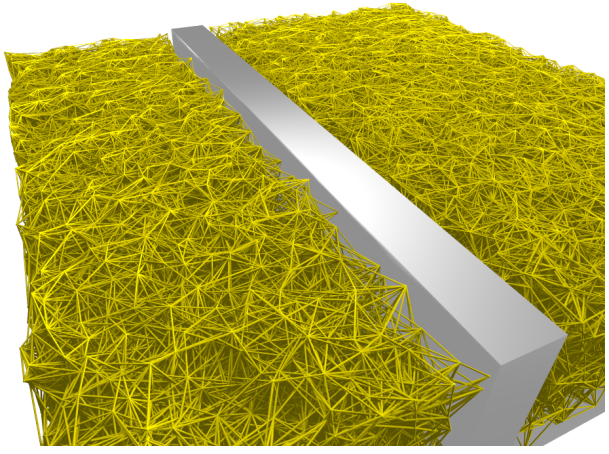
$n = 700$



$n = 1500$



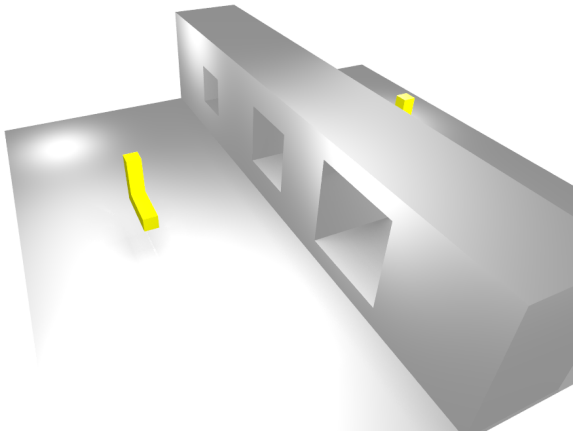
$n = 8000$



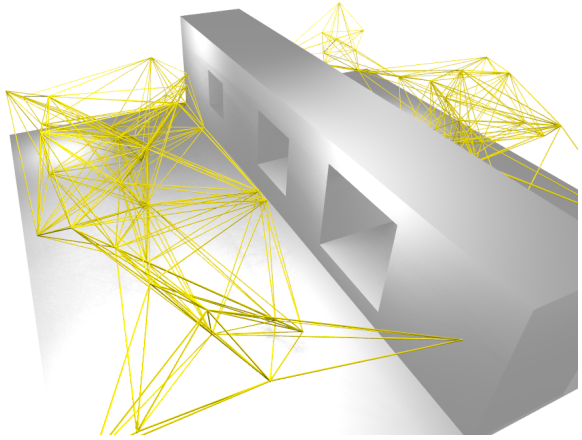
$n = 50000$

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

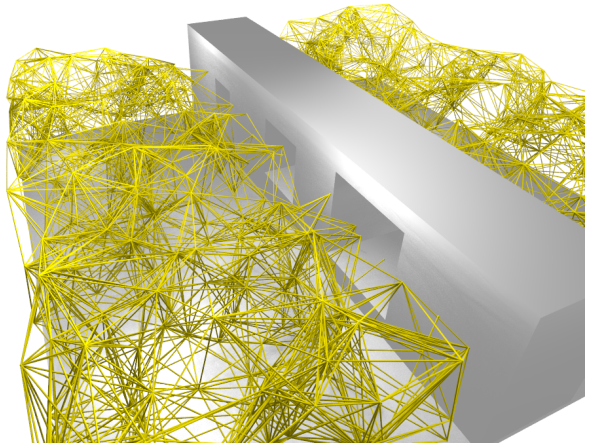
sPRM example 3D $\mathcal{W}$



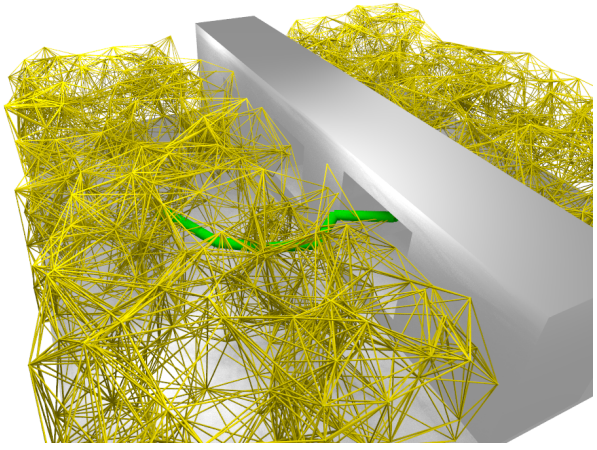
start/goal



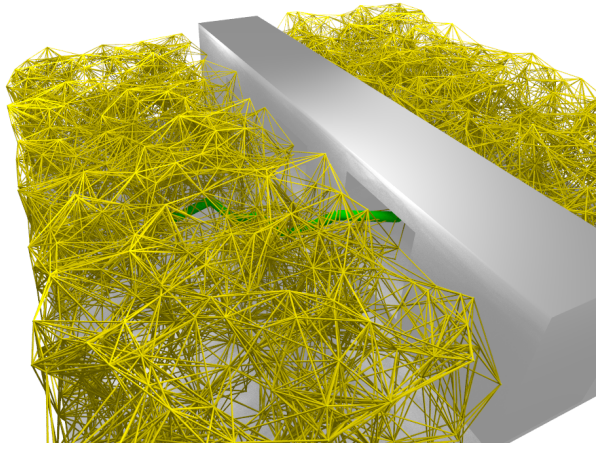
$n = 100$



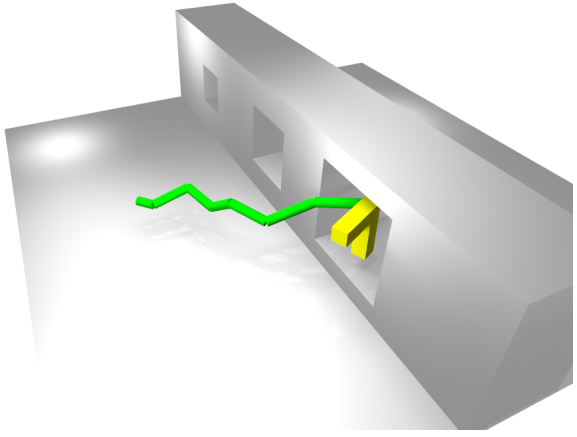
$n = 1000$



$n = 2100$



$n = 4100$

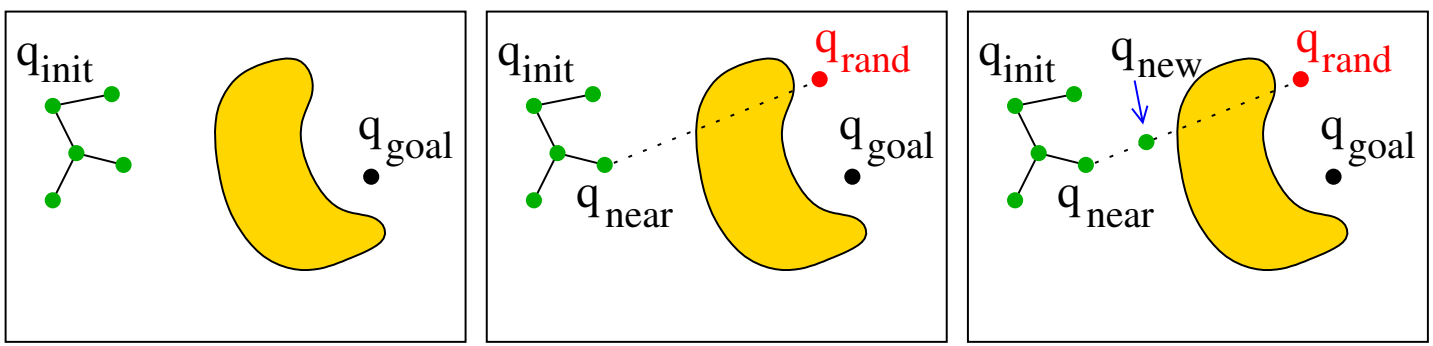


solution

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

```
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 canConnect(q_near, q_new) then
7          T.addNode(q_new)
8          T.addEdge(q_near, q_new)
9          if ϱ(q_new, q_goal) < d_goal then
10             return path from q_init to q_goal
```



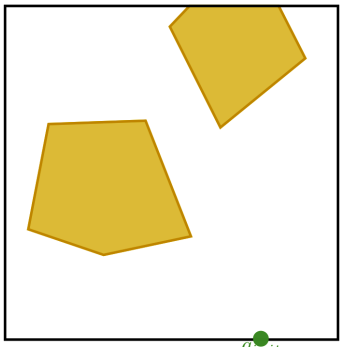
Tree | Sampling | Tree extension

(S. M. LaValle. "Rapidly-exploring random trees : a new tool for path planning". In: *The annual research report* [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

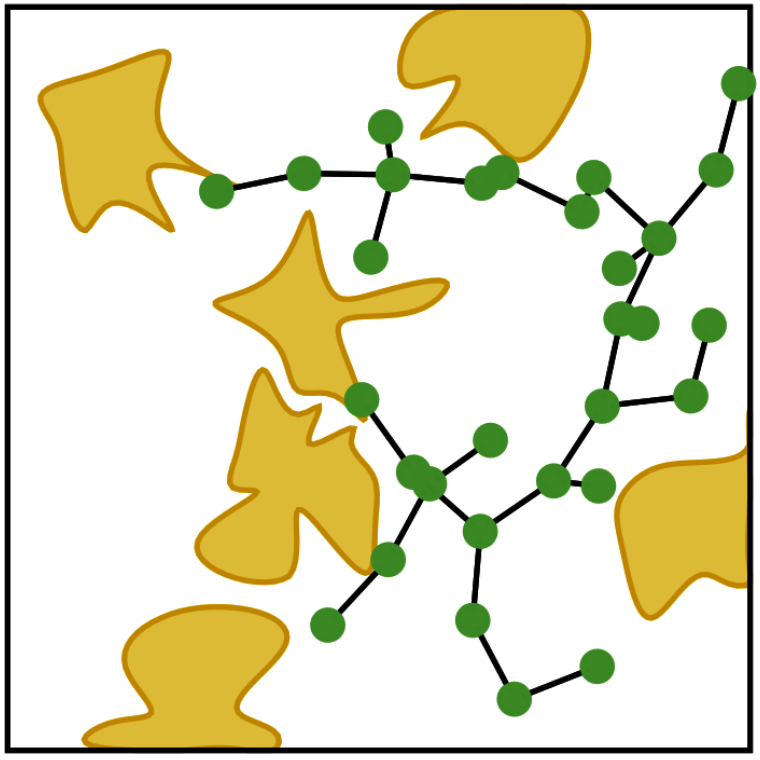
```
1  initialize tree T with q_init
2  for i = 1, ..., I_max do
3      q_rand = generate randomly in C
4      q_near = find nearest node in T towards
         q_rand
5      q_new = localPlanner from q_near towards
         q_rand
6      if canConnect(q_near, q_new) then
7          T.addNode(q_new)
8          T.addEdge(q_near, q_new)
9          if ϱ(q_new, q_goal) < d_goal then
10             return path from q_init to q_goal
```



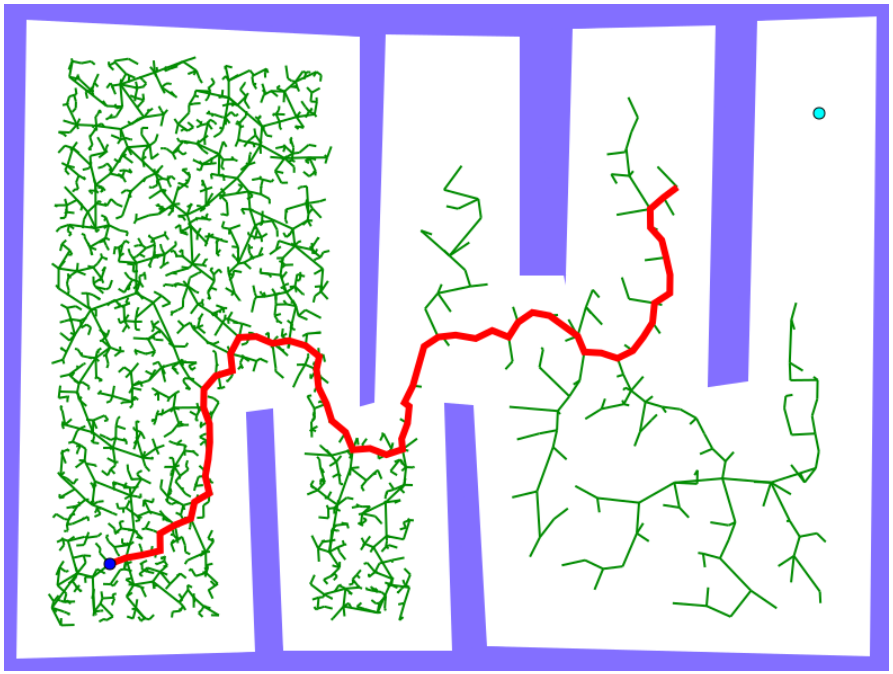
(S. M. LaValle. "Rapidly-exploring random trees : a new tool for path planning". In: *The annual research report* [1998])

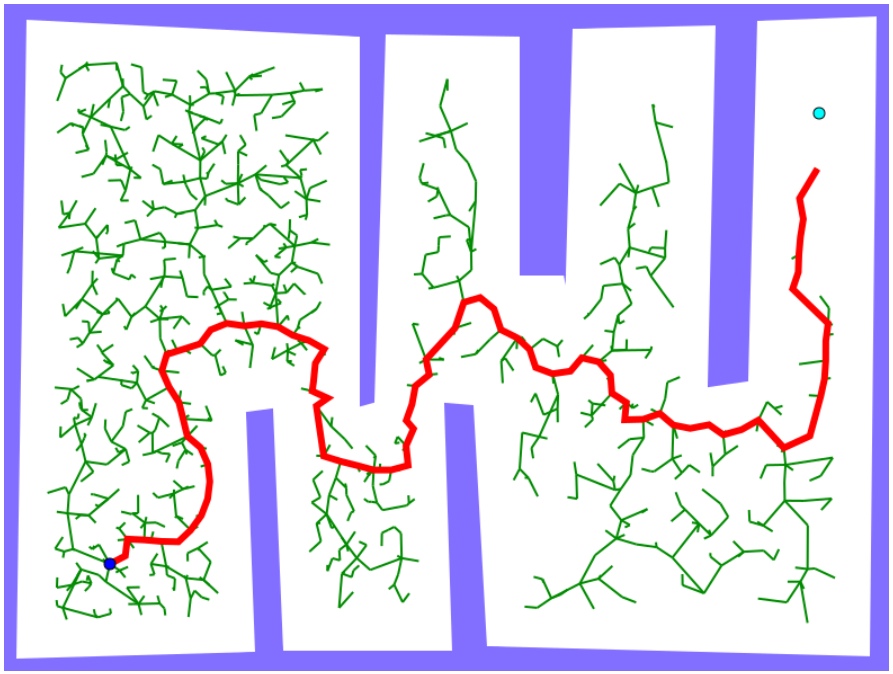
RRT example in 2D $\mathcal{W}$

- 2D $\mathcal{C}$



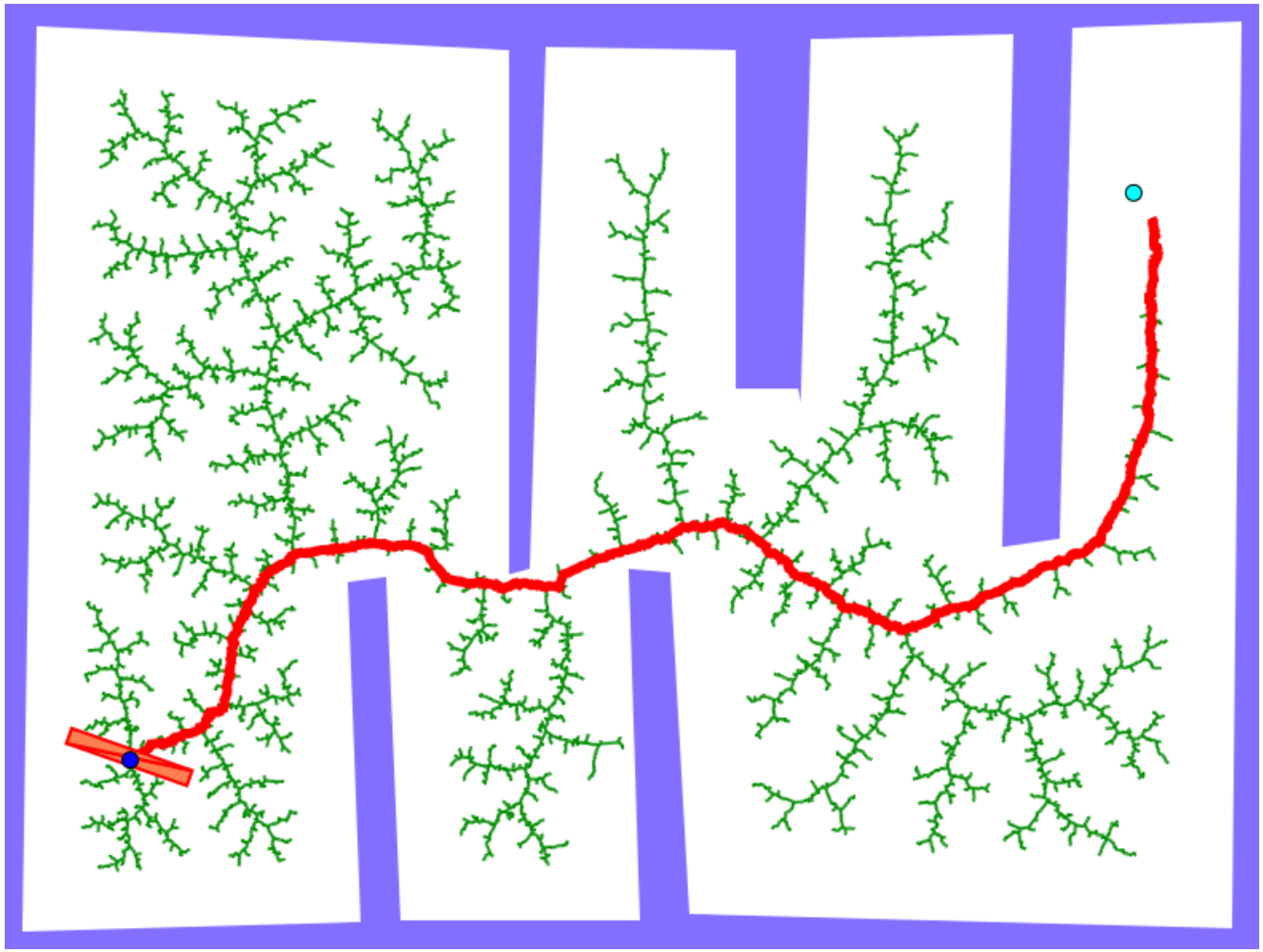
RRT example in 2D $\mathcal{W}$



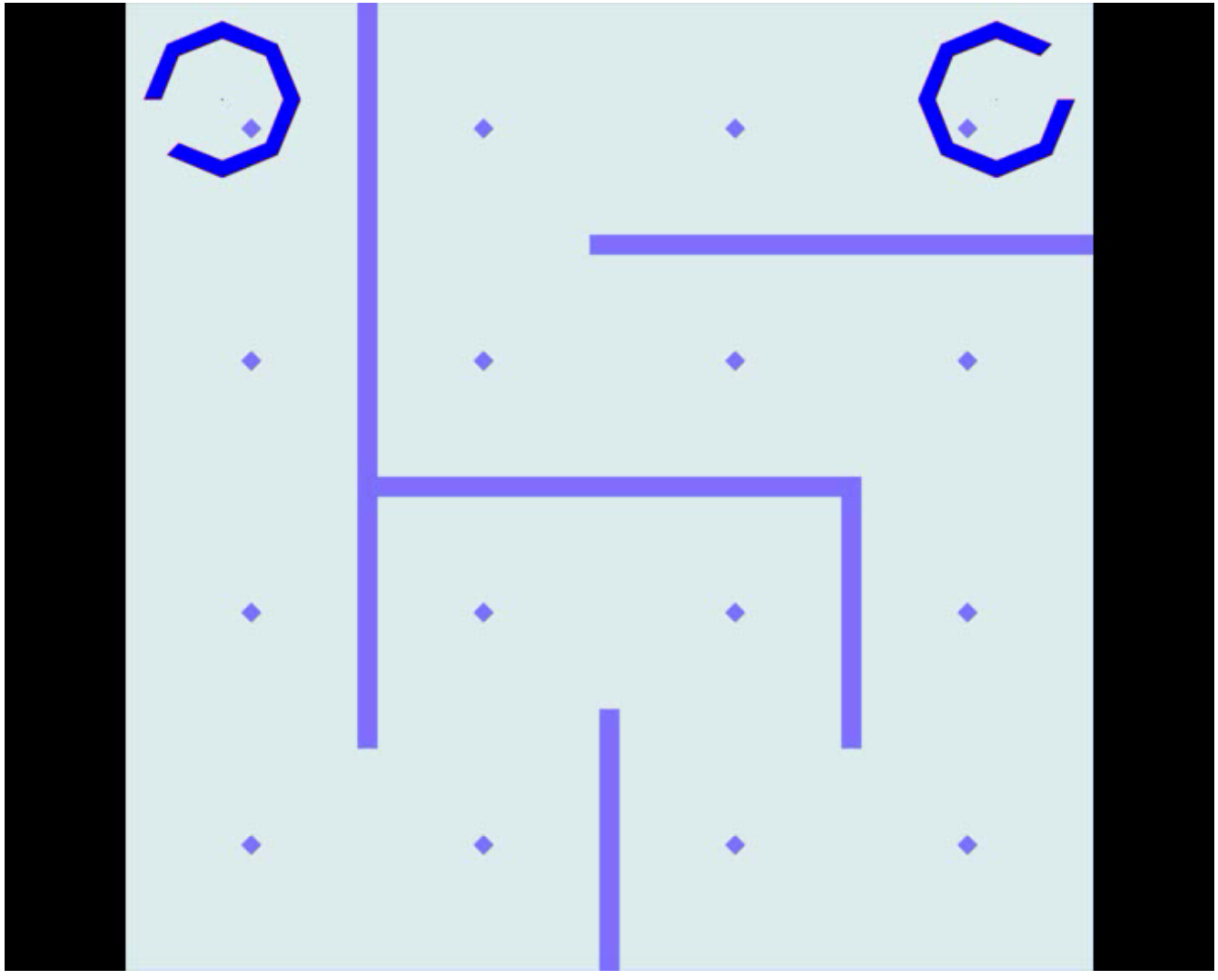


- 2D robot, rotation allowed $\rightarrow$ 3D $\mathcal{C}$
- Why the tree does not “touch” the obstacles?

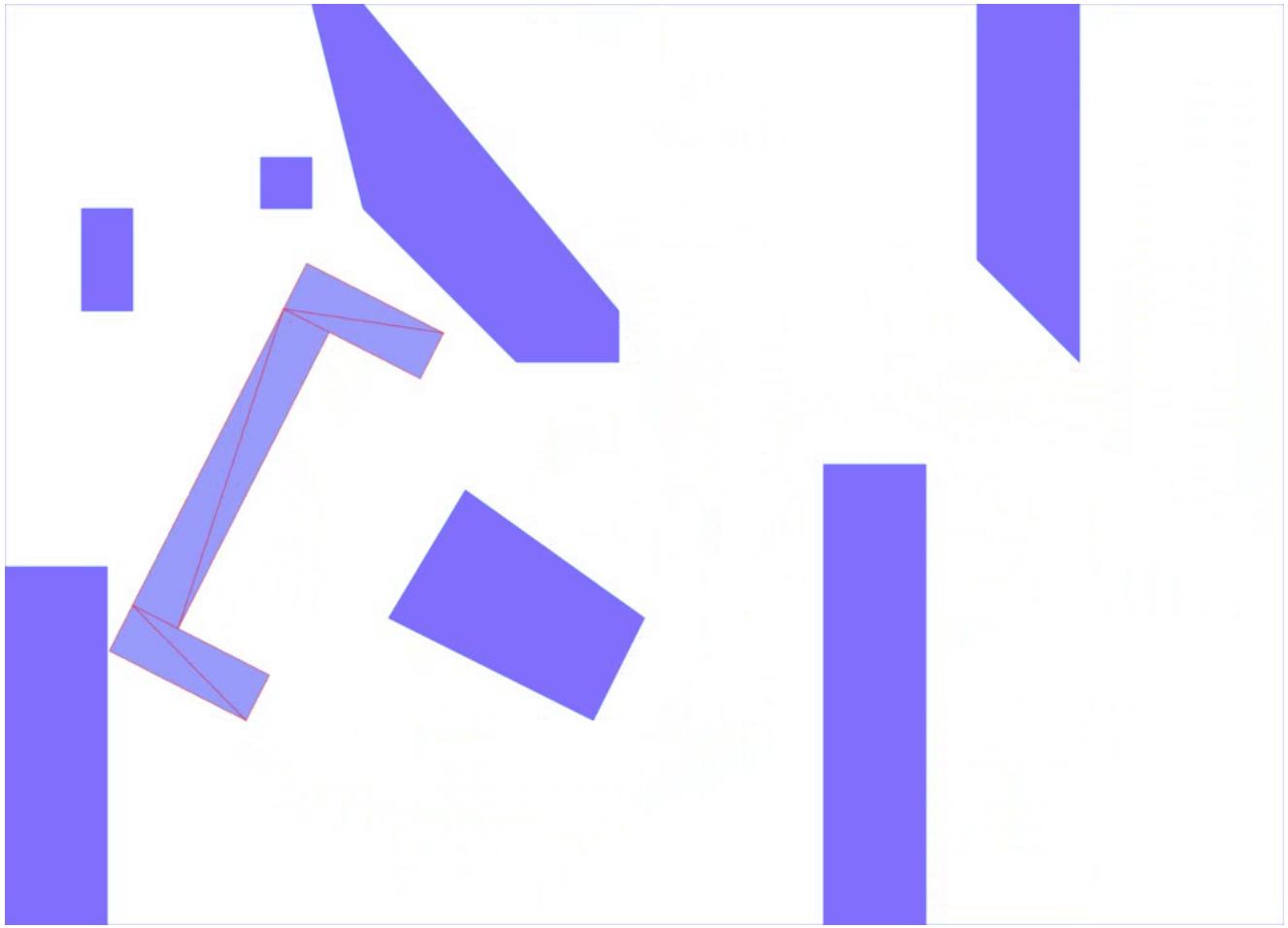
RRT example in 2D $\mathcal{W}$



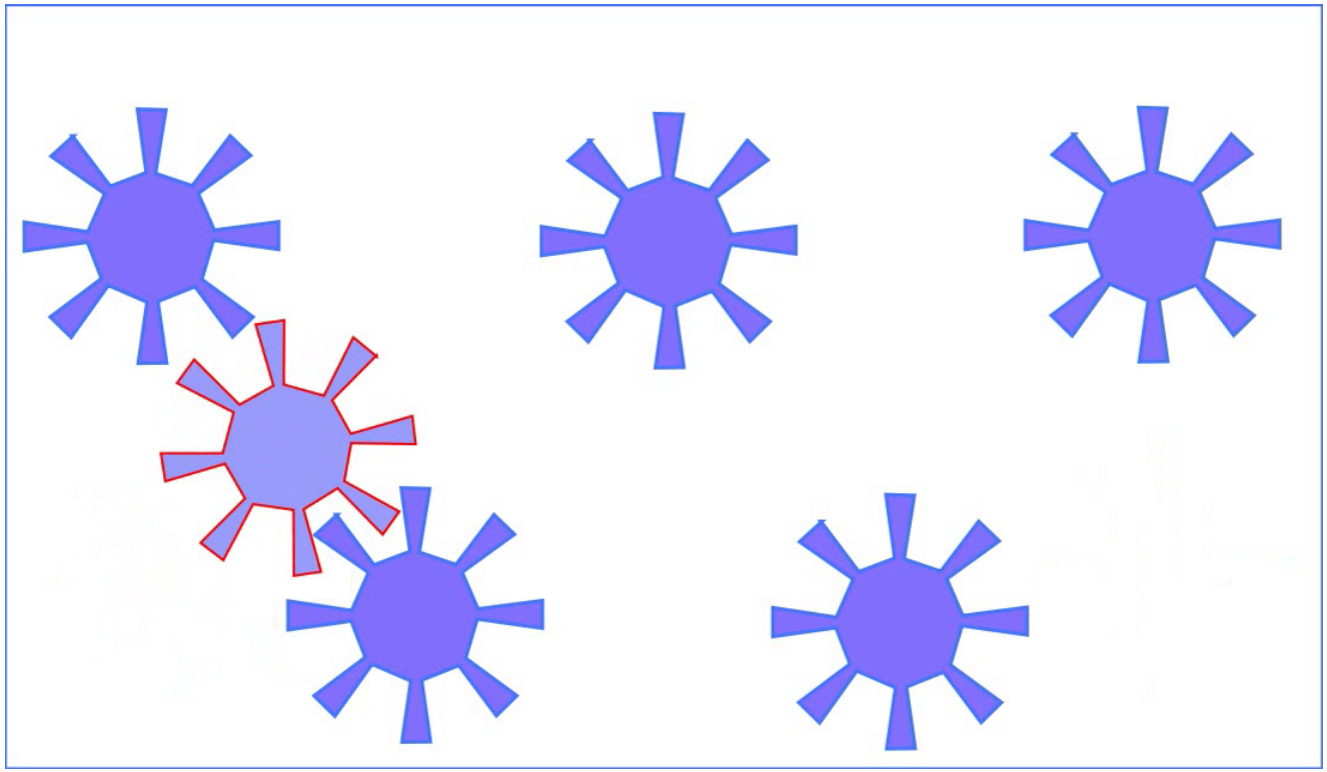
RRT example in 2D $\mathcal{W}$



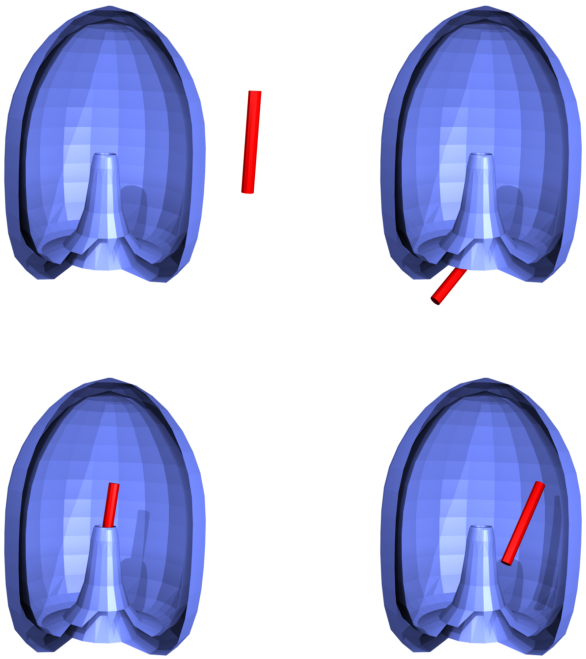
RRT example in 2D $\mathcal{W}$

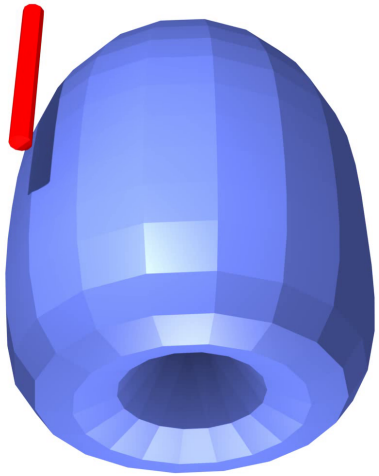


RRT example in 2D $\mathcal{W}$



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

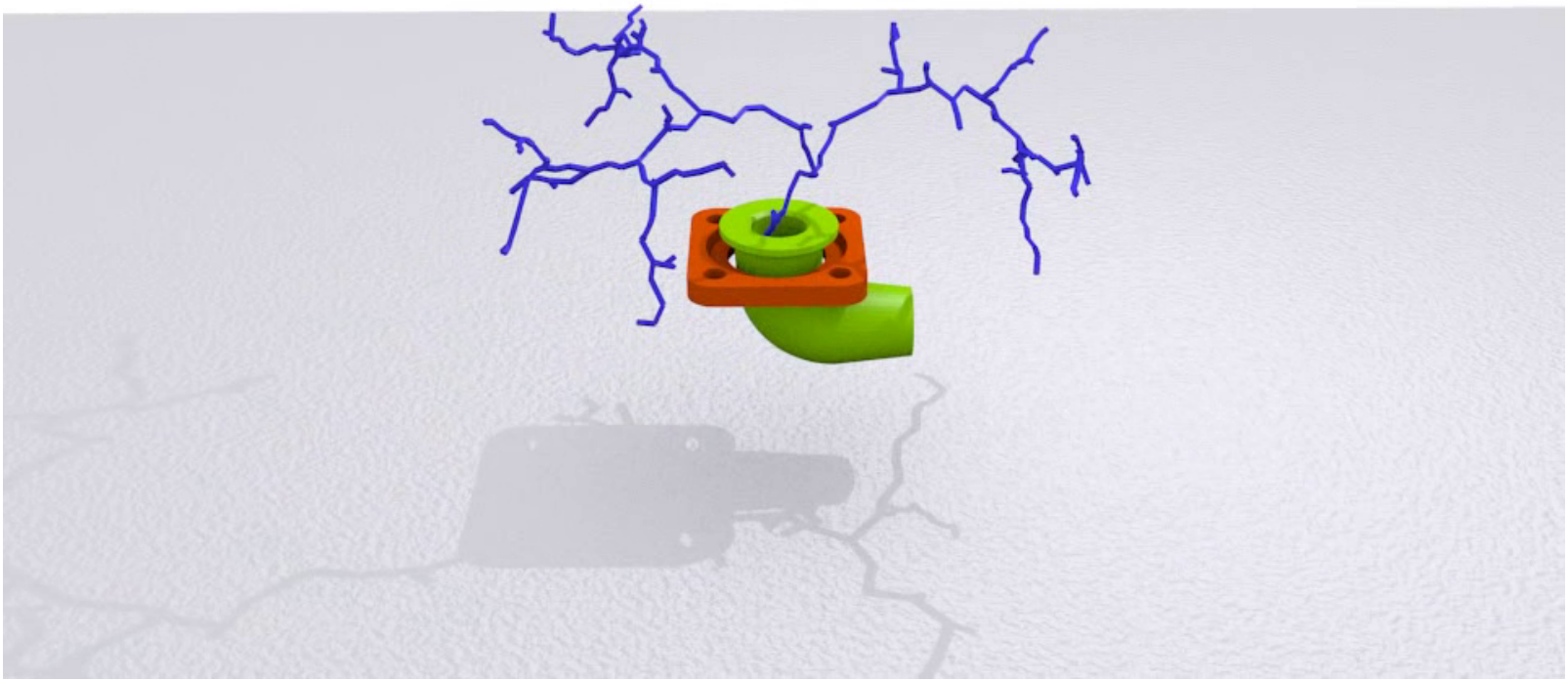




- 3D Bugtrap benchmark

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

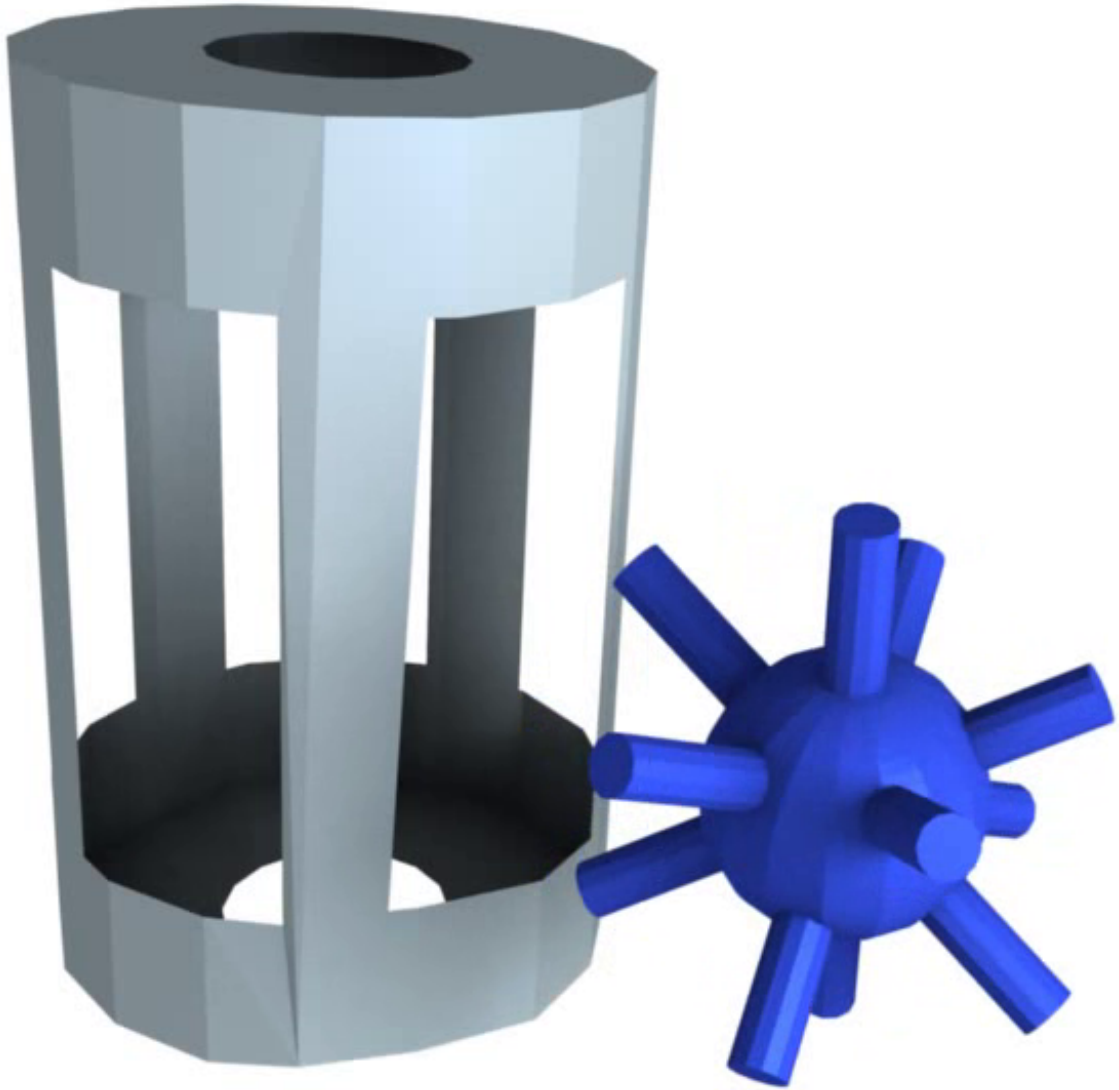
RRT example in 3D $\mathcal{W}$



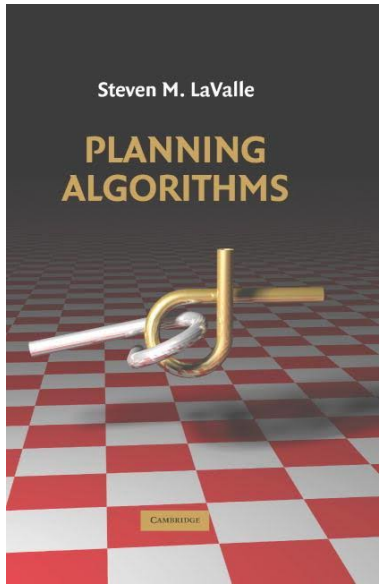
- 3D Flange benchmark

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

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



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



RRT: tree expansion types

Straight-line expansion for RRT

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

Variants:

A If S is collision-free, expand the tree only by q_{rand}

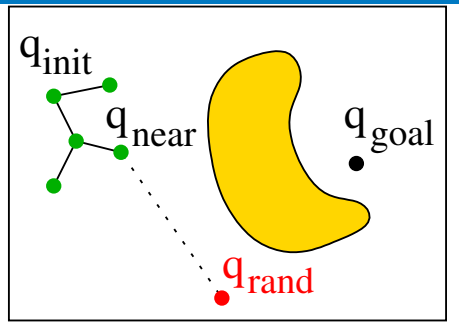
- Creates long segments, fast exploration of $\mathcal{C}$
- Requires nearest-neighbor search to consider point-segment distance
- Requires connection in the middle of line-segment

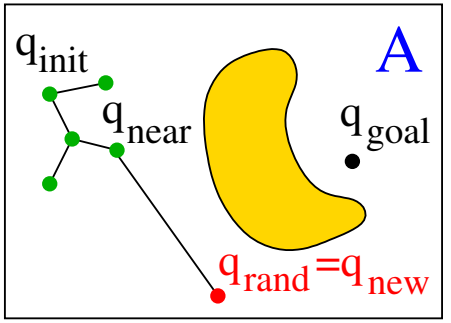
B If S is collision-free, discretize S and expand the tree by all points on S

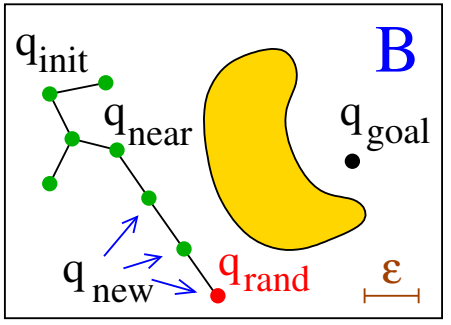
- Most used, enables fast nearest-neighbor search

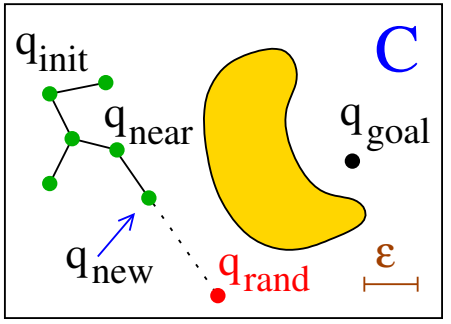
C Find configuration $q_{\text{new}} \in S$ at the distance ε from q_{near}. Expand tree by q_{new} if it's collision-free

- Basic RRT, slower growth than B
- Enables fast nearest-neighbor search



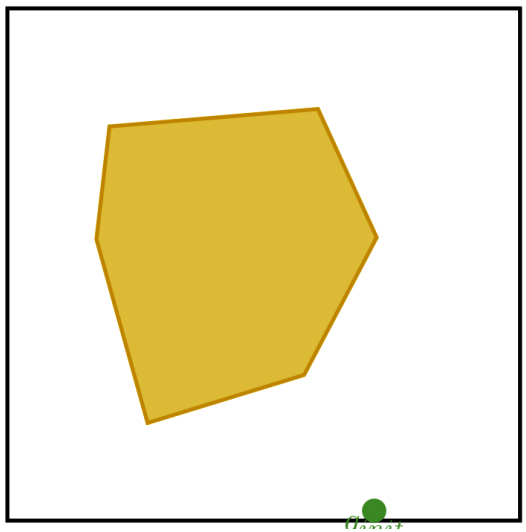




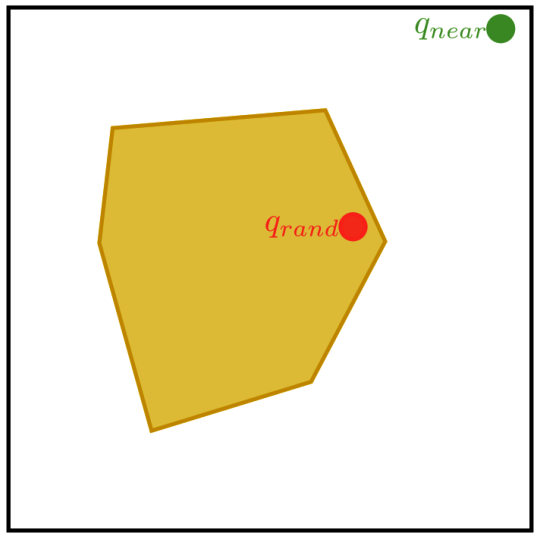


RRT: tree expansion types

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



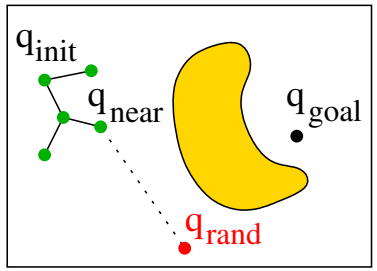
variant A

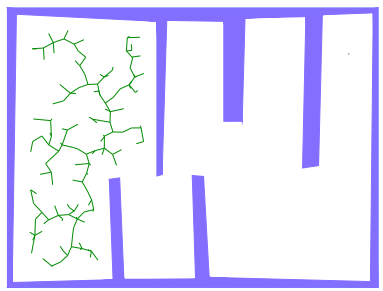


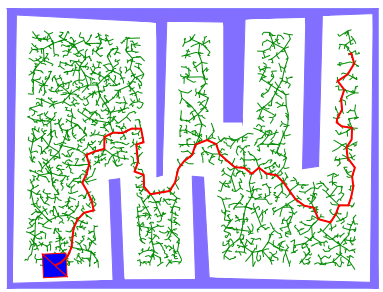
variant C

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





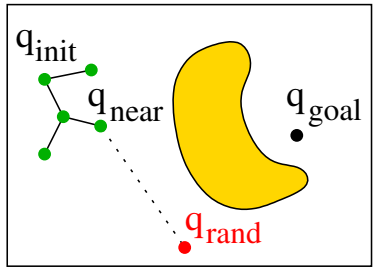


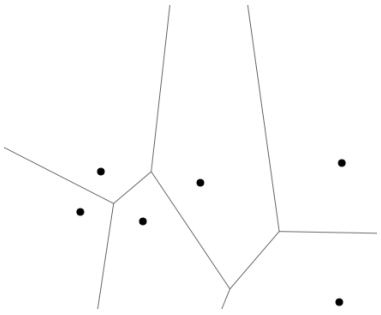
- Why the tree does not grow to itself?
- Why does it “rapidly” explore the $\mathcal{C}$-space?

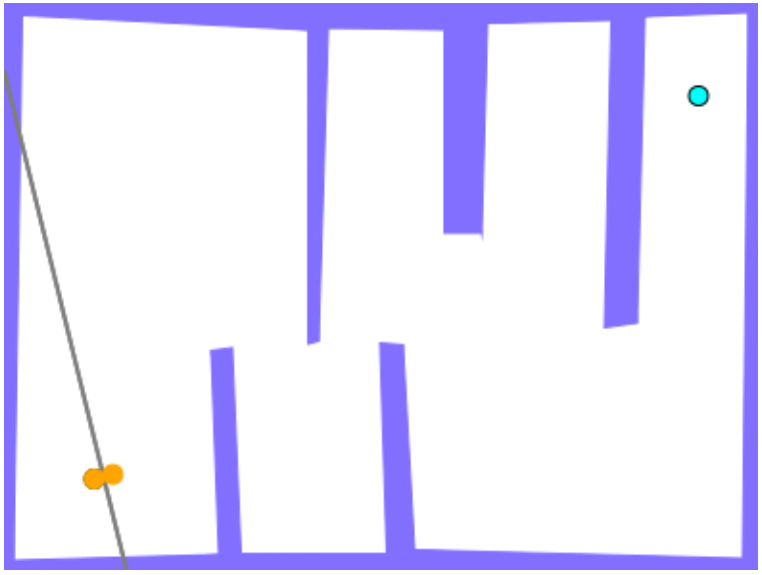
  . . . because of Voronoi bias!

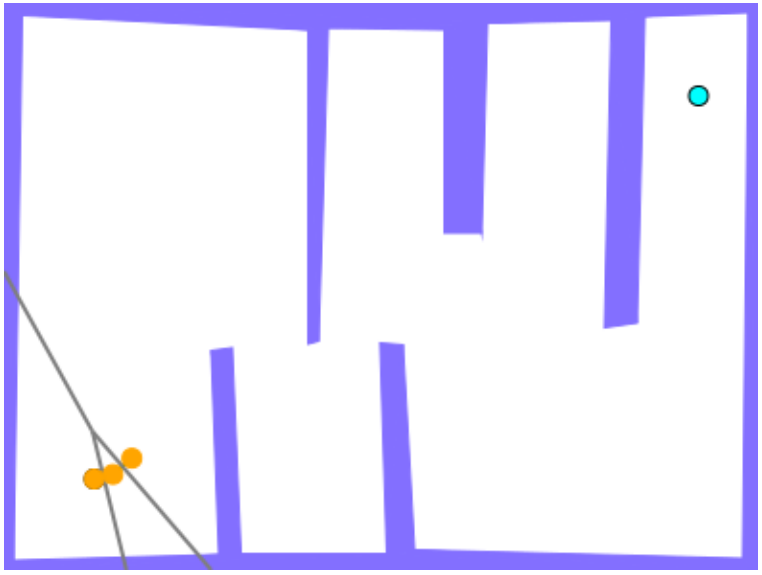
RRT: Voronoi bias I

- 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)



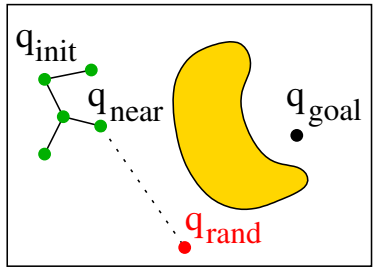


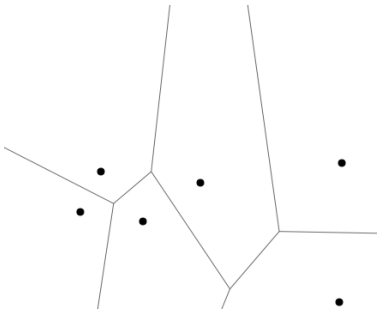


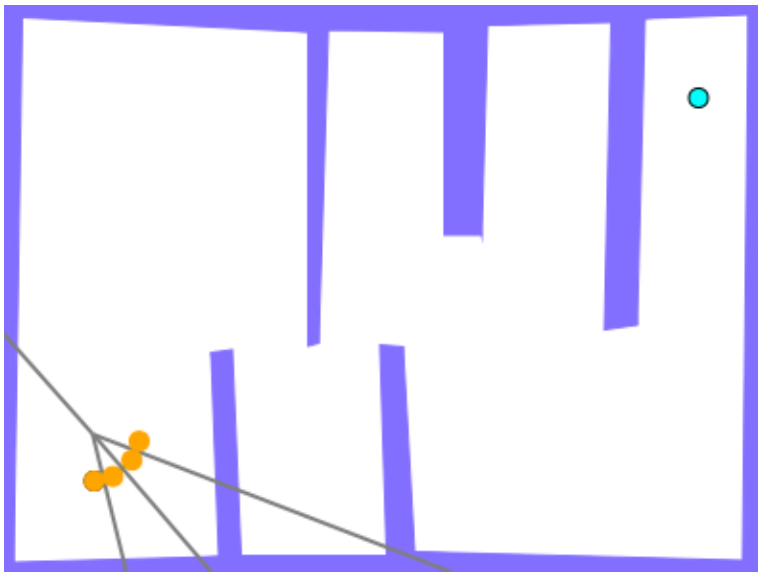


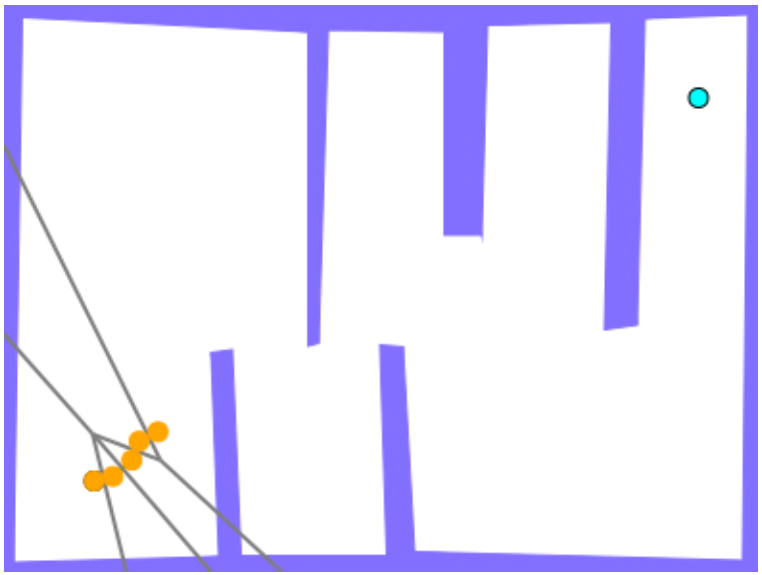
RRT: Voronoi bias I

- 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)

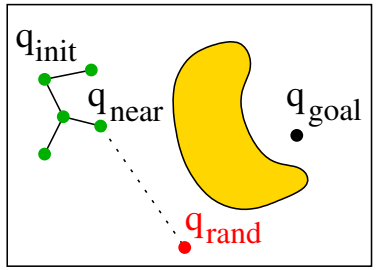




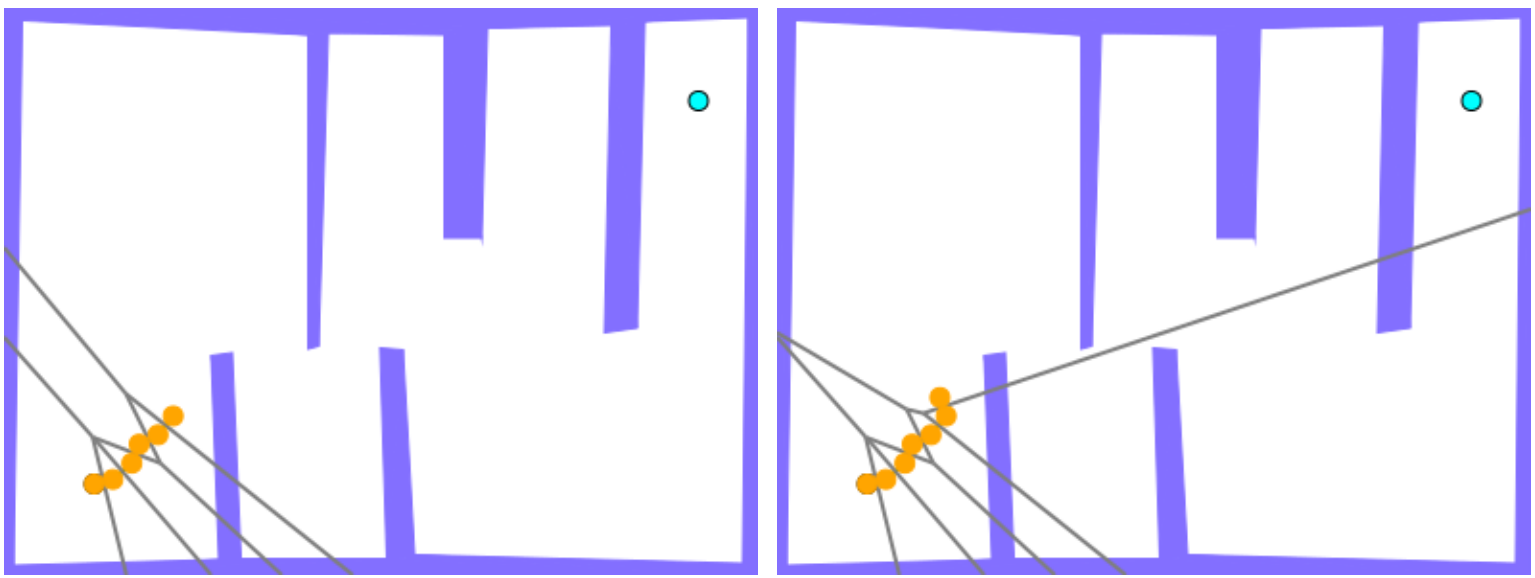




RRT: Voronoi bias I

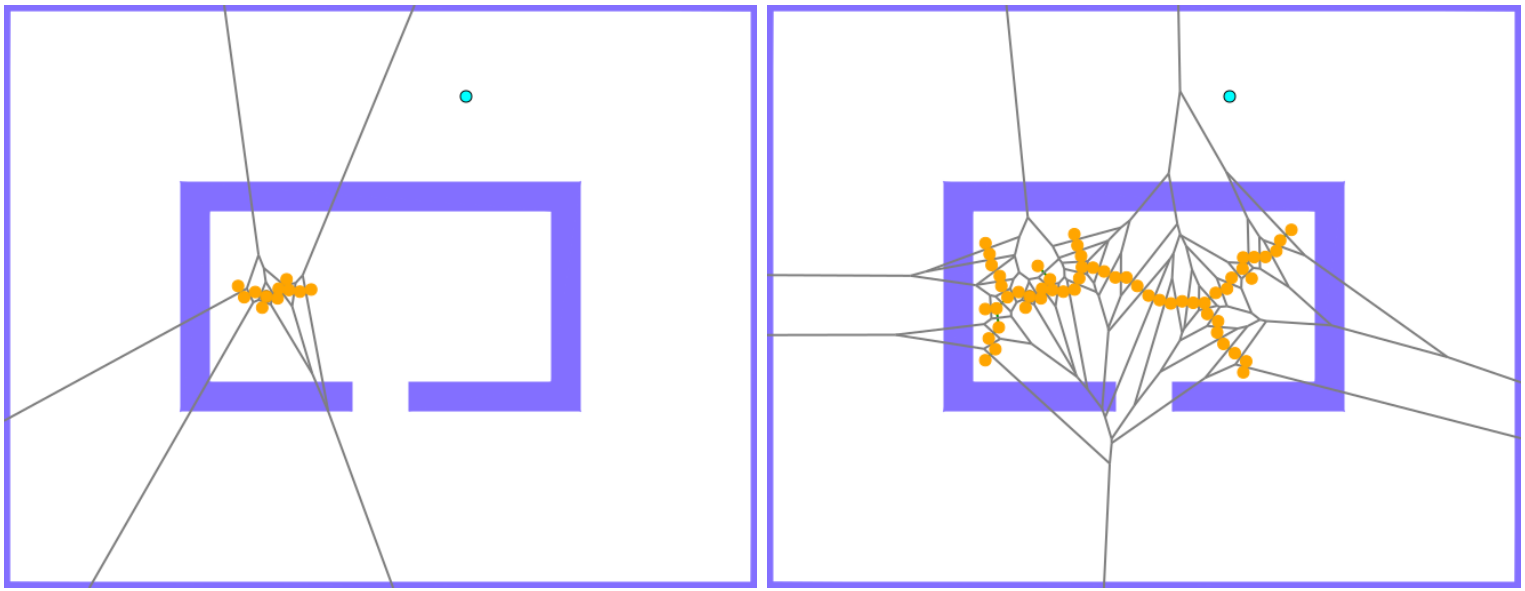


- RRT prefers to expand $\mathcal{T}$ towards unexplored areas of $\mathcal{C}$
- This is caused by **Voronoi bias**:
  - q_{rand} is generated **uniformly** in $\mathcal{C}$
  - $\mathcal{T}$ is expanded from **nearest** node in $\mathcal{T}$ **towards** q_{rand}
  - The probability that a node $q \in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

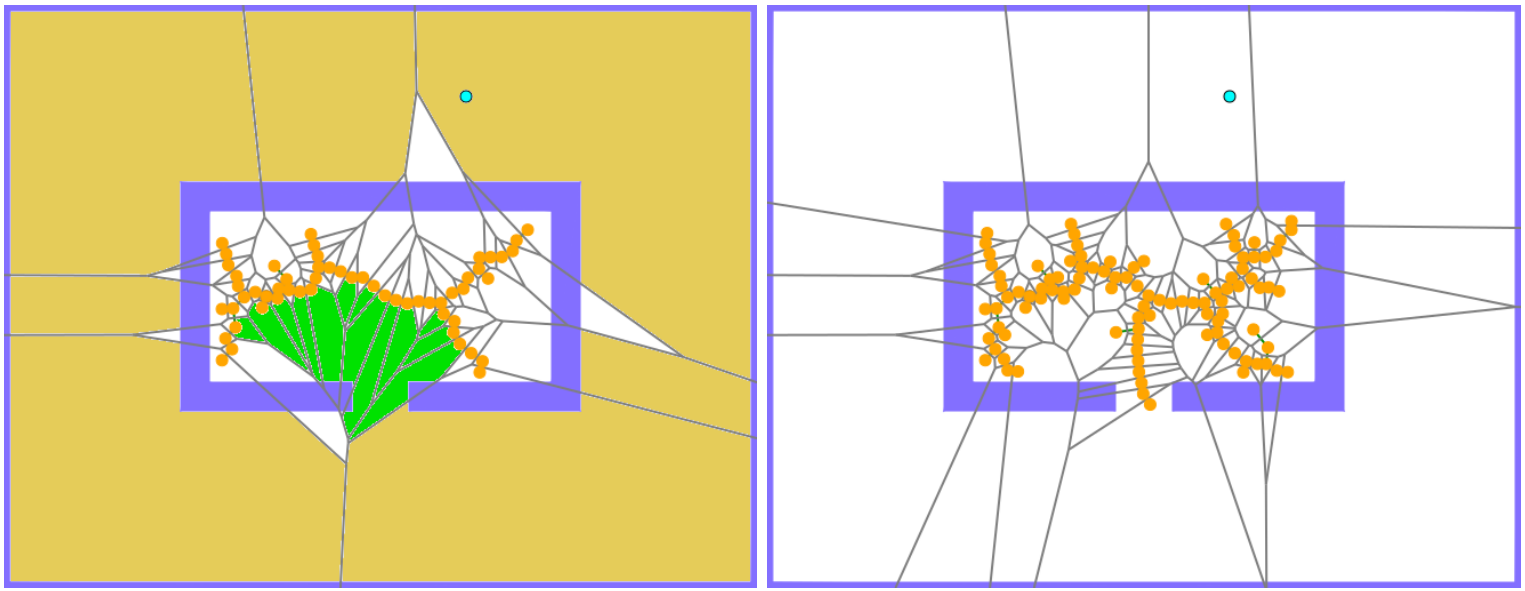


iteration 10, tree size 10 | iteration 70, tree size $\sim$ 60

- 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!

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 70, tree size $\sim$ 60 | iteration 300, tree size $\sim$ 100

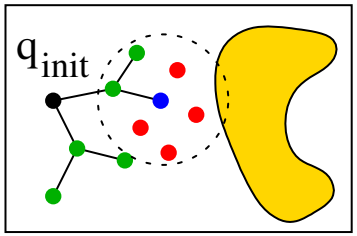
- Tree grows well until iteration 70
- Yellow: areas with high prob. of being selected for expansion
- Green: areas that show be selected for expansion so the tree can escape the obstacle
- The tree does not expand much until iteration 300!

Expansive-space tree (EST)

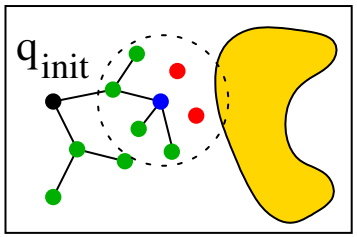
- Builds trees $\mathcal{T}_i$ (from q_{init}) and $\mathcal{T}_g$ (from q_{goal})
- Weight $w(q)$ is computed for each configuration q
- Nodes are expanded with probability $w(q)^{-1}$
- Expansion of one tree $\mathcal{T}$:

1 q' = select node from $\mathcal{T}$ with probability $w(q)^{-1}$
2 Q = k random points around
  $q' : Q = \{q \in \mathcal{C}_{\text{free}} \mid \varrho(q, q') < d\}$
3 **foreach** $q \in Q$ **do**
4 $w(q)$ = compute weight of the sample q
5 **if** *rand*() $< w(q)^{-1}$ **and** *connect*(q, q') **then**
6 $\mathcal{T}$.addNode(q)
7 $\mathcal{T}$.addEdge(q', q)

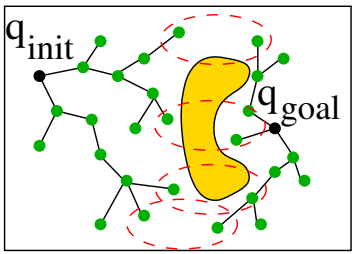
- $w(q)$ is the number of nodes in $\mathcal{T}$ around q
- $\mathcal{T}_i$ and $\mathcal{T}_g$ grow until they approach each other
- Trees are connected using local planner between their nearest nodes



q', samples Q



connected,ignored

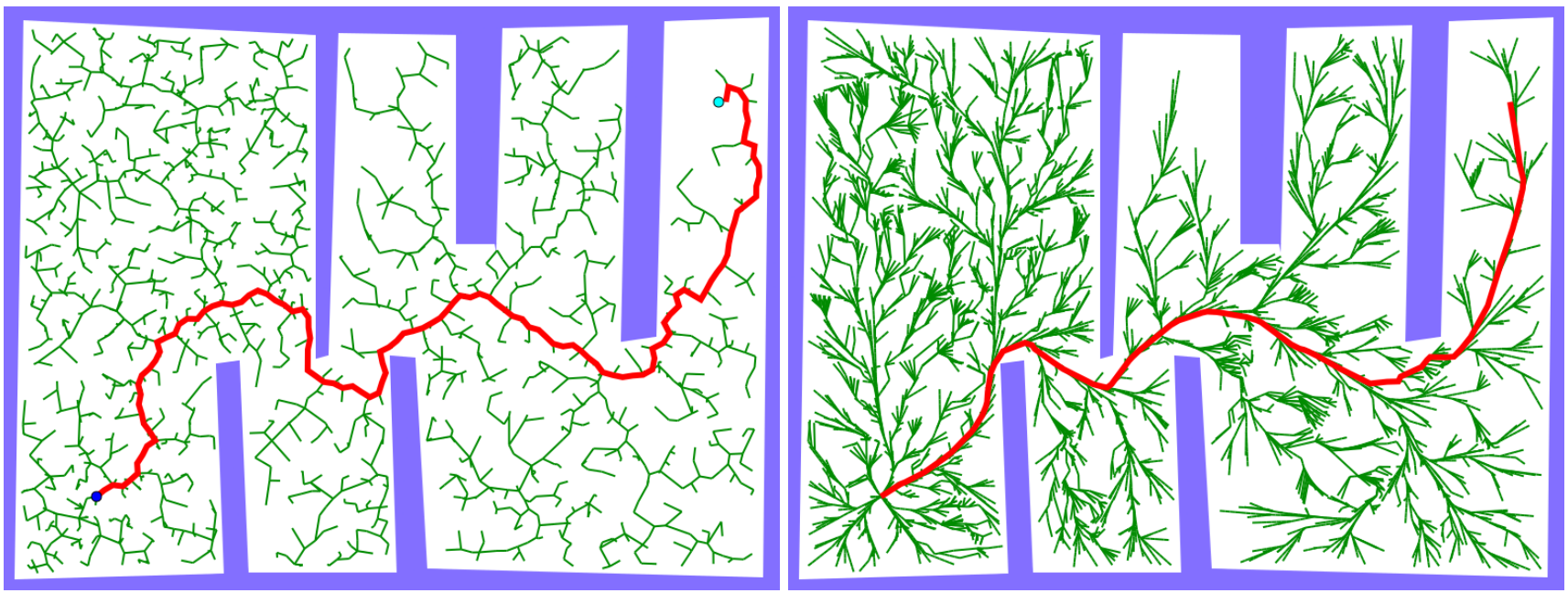


$\mathcal{T}_i$ and $\mathcal{T}_g$, pairs for connection

(D. Hsu et al. “Path planning in expansive configuration spaces”. In: *Proceedings of International Conference on Robotics and Automation*. Vol. 3. 1997, 2719–2726 vol.3. DOI: 10.1109/ROBOT.1997.619371)

Asymptotically optimal RRT* and PRM*

- 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[9]



RRT RRT*

[9] S. Karaman and E. Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *The international journal of robotics research* 30.7 (2011), pp. 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 $n = |V|$

k-nearest PRM*

- 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

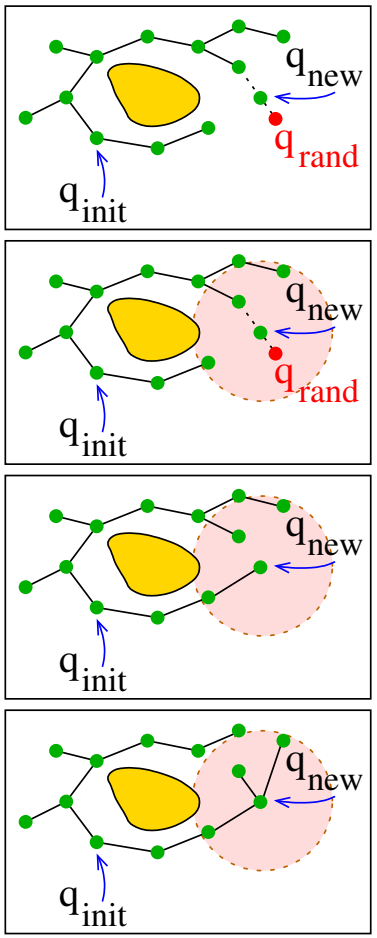
k-nearest PRM* algorithm (aka k-PRM*)

- 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

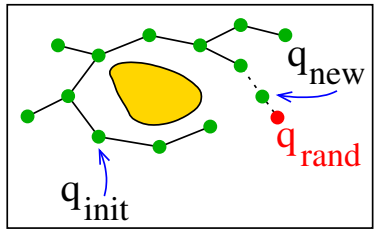
- Optimal version of RRT[10]
- For each node, a cost of the path from q_{init} to that node is established
- RRT* has improved tree expansion and nearest-neighbor search
- Tree expansion by node q_{new}
  - Parent of q_{new} is optimized to minimize cost at q_{new}
  - After q_{new} is connected to tree, 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*



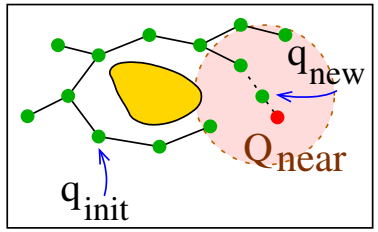
[10] S. Karaman and E. Frazzoli. “Sampling-based algorithms for optimal motion planning”. In: *The international journal of robotics research* 30.7 (2011), pp. 846–894.

RRT*: algorithm

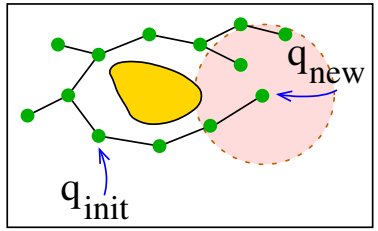

```
1  initialize tree T with q_init
2  for i = 1, ..., I_max do
3      q_rand = generate randomly in C
4      q_near = find nearest node in T towards q_rand
5      q_new = localPlanner from q_near towards q_rand
6      if q_new is collision-free then
7          Q_near = T.neighborhood(q_new, r)
8          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
```



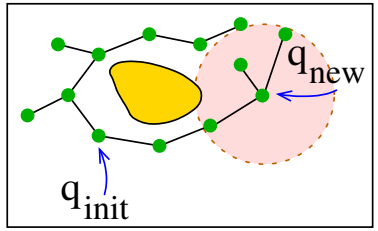
lines 3–5



line 7



lines 10–16



lines 17–20

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*)

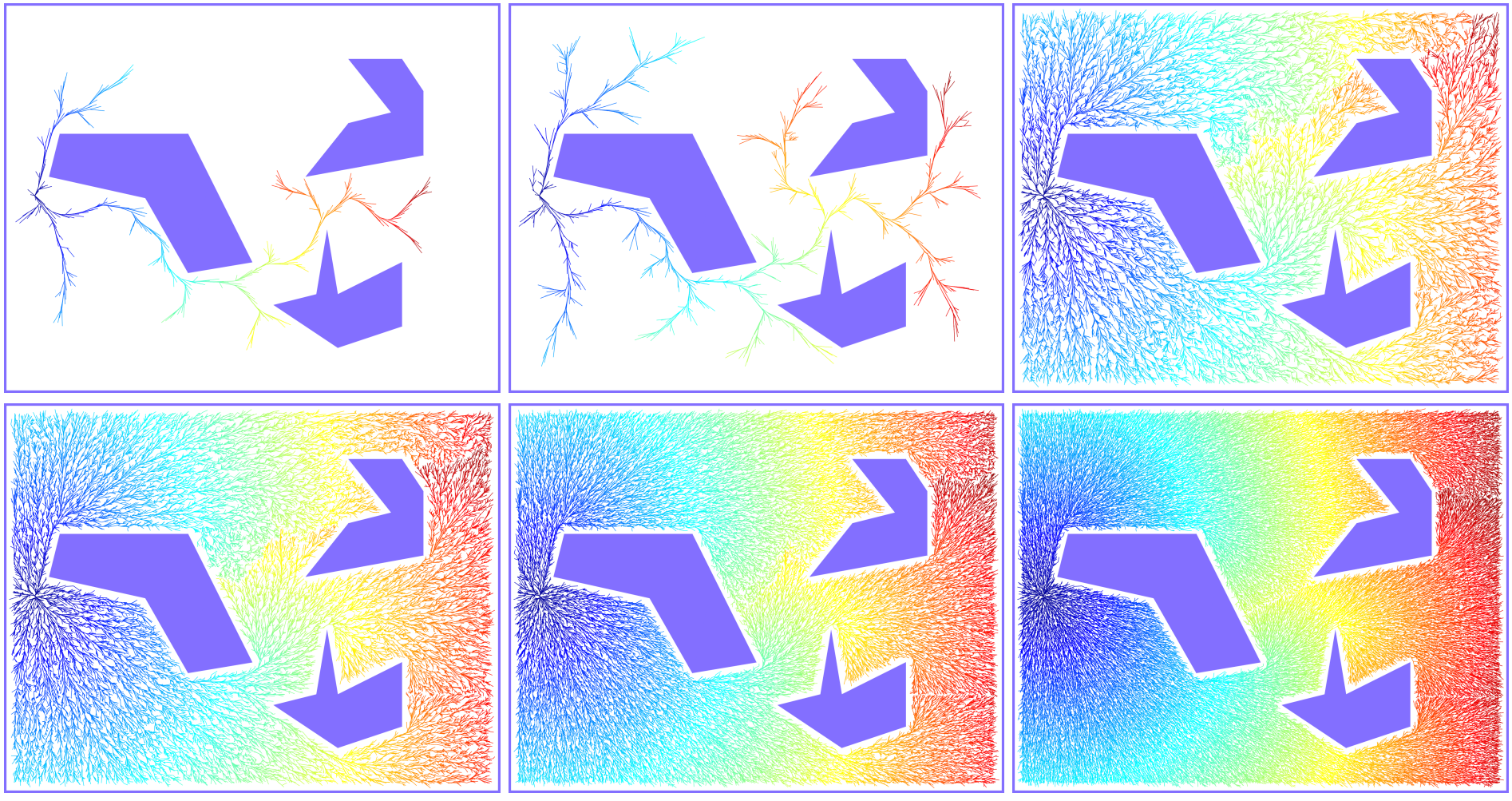
- k-nearest neighbors are selected for parent search and rewiring

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

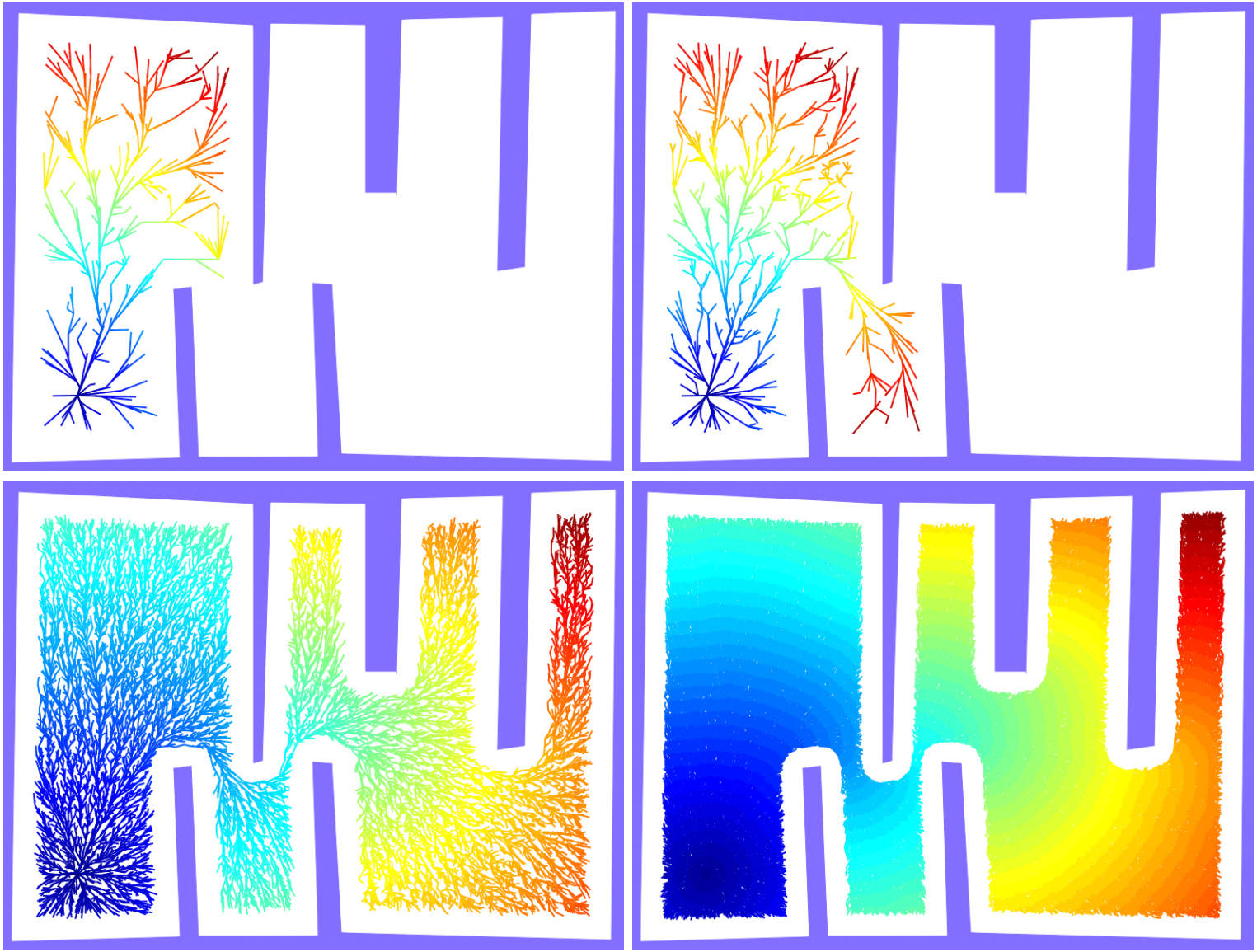
$$k_{RRT} > k^*_{RRT} = e\left(1 + \frac{1}{d}\right)$$

- 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 $\mathcal{W}$

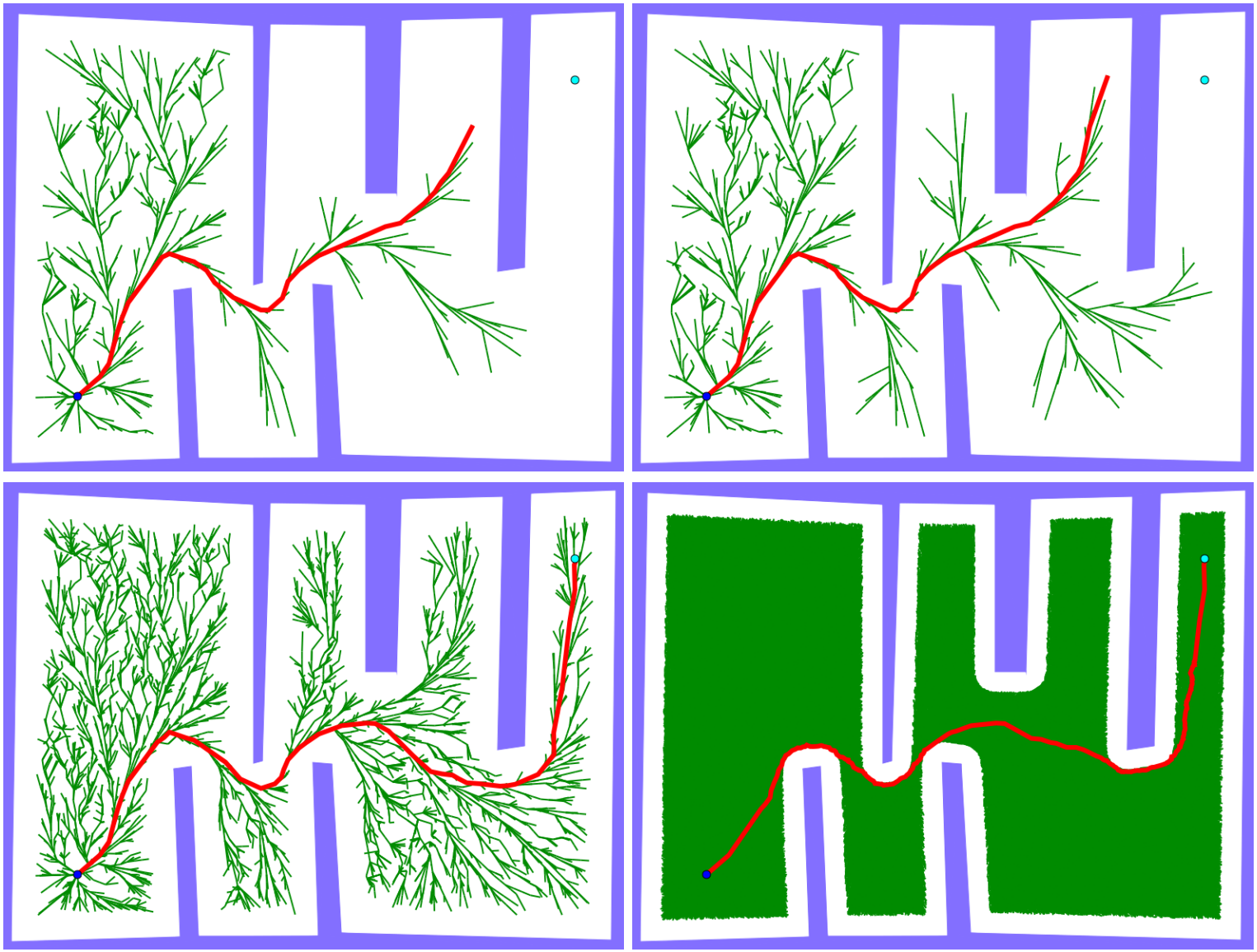


Rectangle robot, rotation allowed $\rightarrow$ 3D $\mathcal{C}$



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

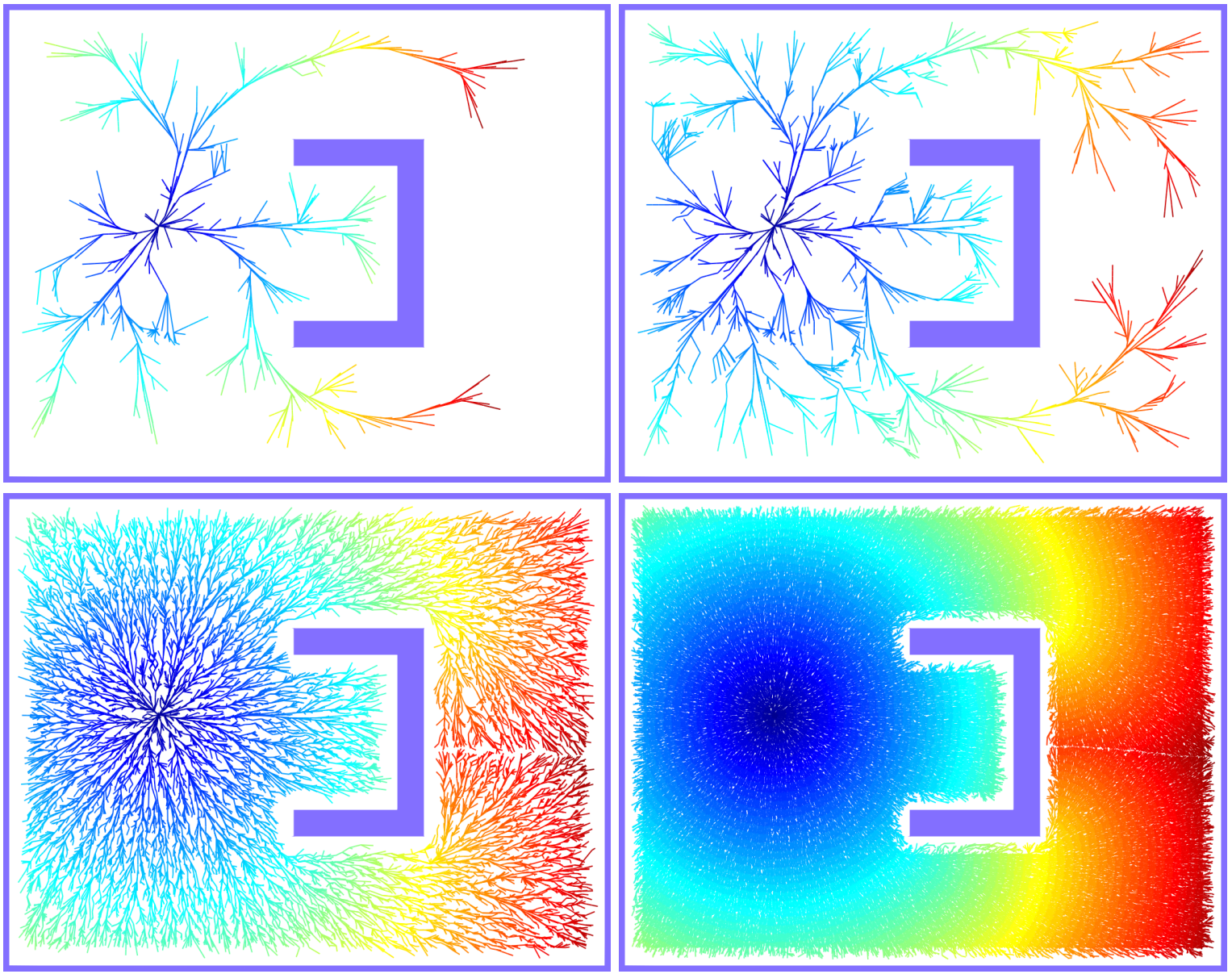
RRT*: example in 2D $\mathcal{W}$



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

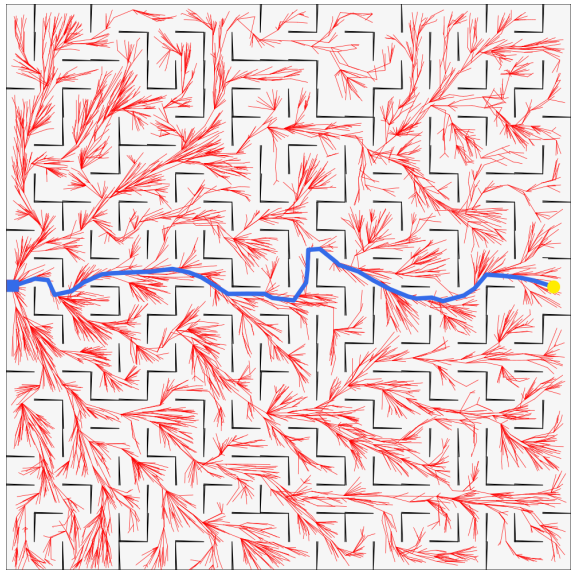
Depicted path demonstrates the slow convergence of the path quality

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



Informed RRT*: motivation

- RRT*samples the whole configuration space $\mathcal{C}$
- There are definitively samples that cannot help to improve the path[11]
- Their sampling is useless and increases runtime



RRT*

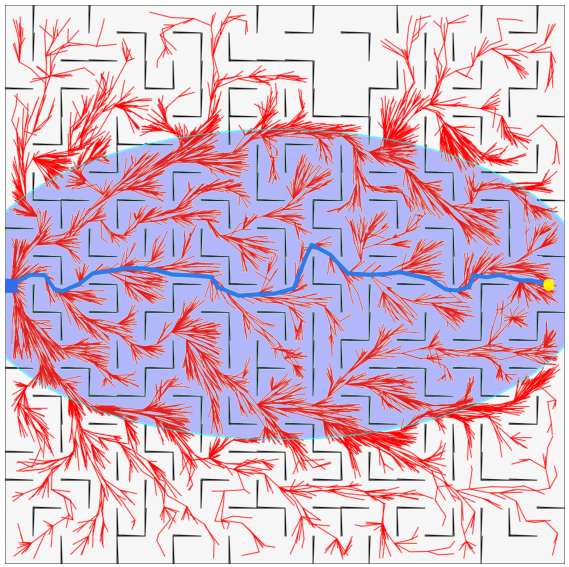
[11] J. D. Gammell et al. "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014, pp. 2997–3004. DOI: 10.1109/IROS.2014.6942976.

Informed RRT*

- Observation: a sample $q \in \mathcal{C}$ may improve the quality of the current solution only if path from it to start and goal is shorter (or equal) than the current solution[12]
- "Omniscient set" $\mathcal{O}$: all configurations that **can** improve the solution:

$$\mathcal{O} = \{q | \text{len}(P_{to}) + \text{len}(P_{from}) \leq \text{len}(P), q \in \mathcal{C}\}$$

- P_{to} is path from q_{init} to q
- P_{from} is path from q to q_{goal}
- P is the current best solution
- $\text{len}(\cdot)$ is the length of the path
- $\mathcal{O}$ is a prolate n-dimensional ellipsoid
- Sampling from $\mathcal{O}$ ensures asymptotic optimality

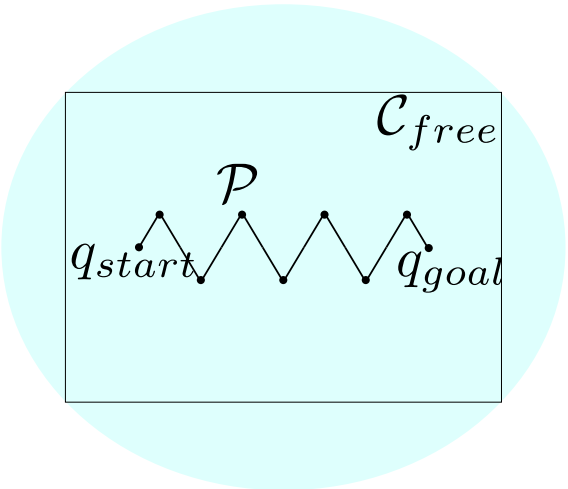


Informed RRT*

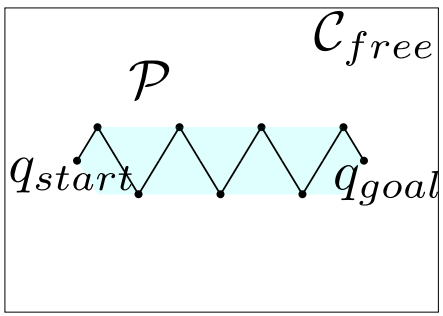
[12] J. D. Gammell et al. "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014, pp. 2997–3004. DOI: 10.1109/IROS.2014.6942976.

Partially informed RRT*: motivation

- Omniscient set of Informed RRT* may still be too large
- This happens when the current best solution is quite long
- We can shrink $\mathcal{O}$ even more[13]



Informed RRT*

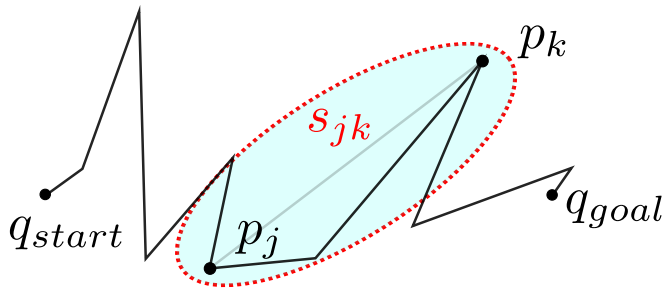


Ideal sampling space

[13] J. Kříž and V. Vonásek. "Asymptotically Optimal Path Planning With an Approximation of the Omniscient Set". In: *IEEE Robotics and Automation Letters* 10.4 (2025), pp. 3214–3221. DOI: 10.1109/LRA.2025.3540627.

Partially Informed RRT*

- Current best solution:
  $P = (q_1 = q_{\text{init}}, q_2, \ldots, q_{n-1}, q_n = q_{\text{goal}})$
- Its subpath is $P_{j,k} = (q_j, q_{j+1}, \ldots, q_k), q_i \in P$
- Local hyperellipsoid:



$$s_{j,k} = \{q \mid \|q - p_j\| + \|q - p_k\| \leq len(P_{j,k}), q \in \mathcal{C}\}$$

- Partially informed set

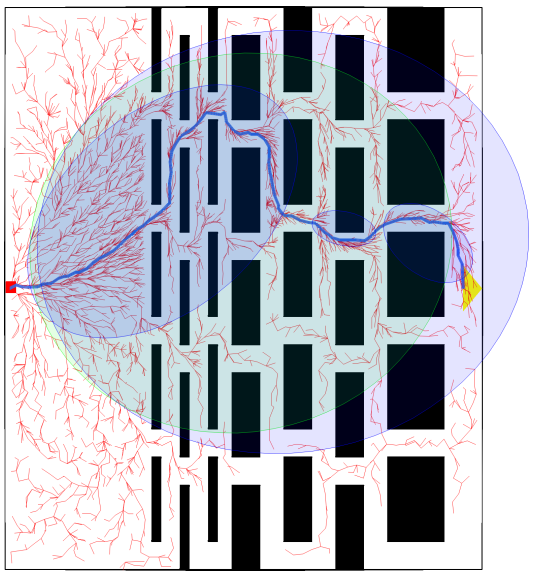
$$\mathcal{S} = \bigcup_{\substack{j,k \in \{1,\ldots,n\} \\ k>j\ ,\ k-j \geq c}} s_{j,k}$$

What is c for?

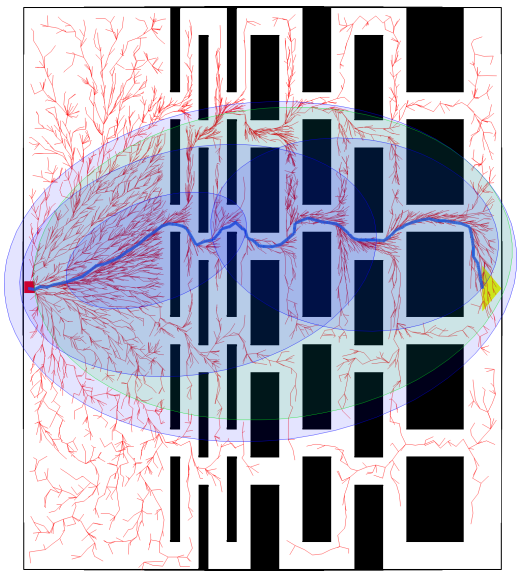
- It controls exploitation vs exploration
- Small c: we want to improve current solution (exploitation)
- Large c: we want to find new alternative paths

Partially informed sampling (sampling from $\mathcal{S}$)

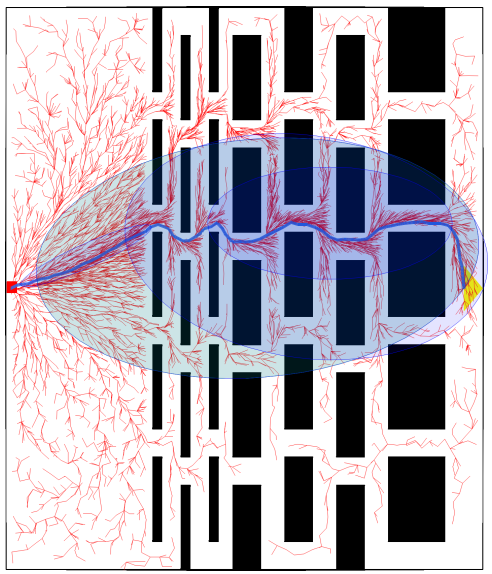
- Choose the random subpath length l from the range $[c, n]$
- Select begining of the path j, set $k = j + l$, construct $s_{j,k}$
- Draw random samples from $s_{j,k}$ (same as in Informed RRT*)
- Sampling from $\mathcal{S}$ ensures asymptotic optimality



11k samples



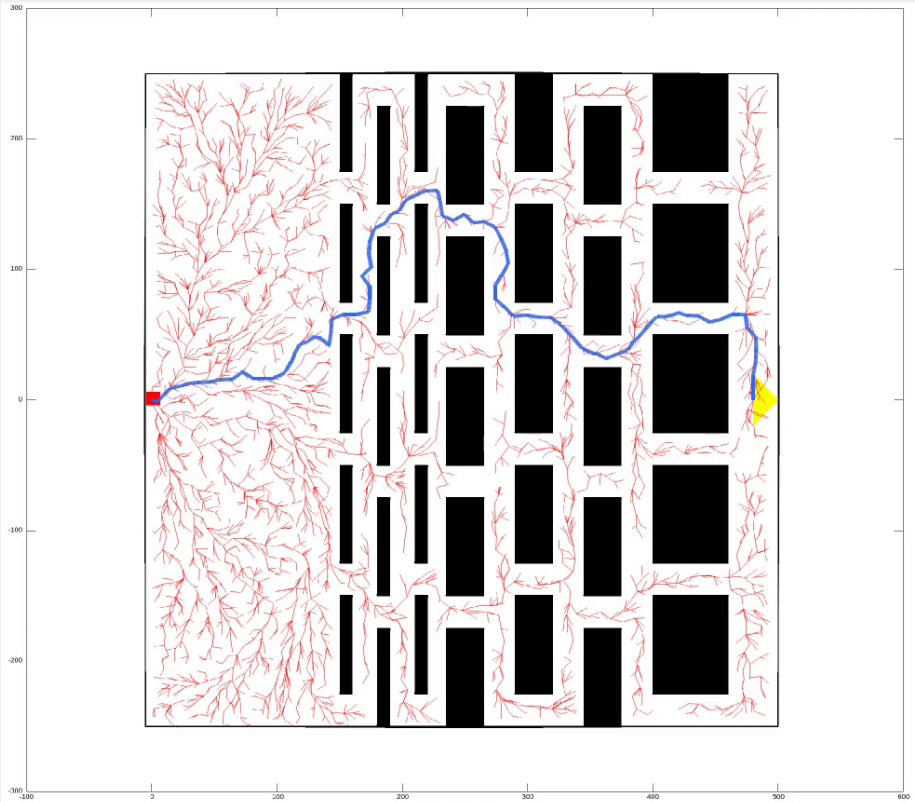
15k samples

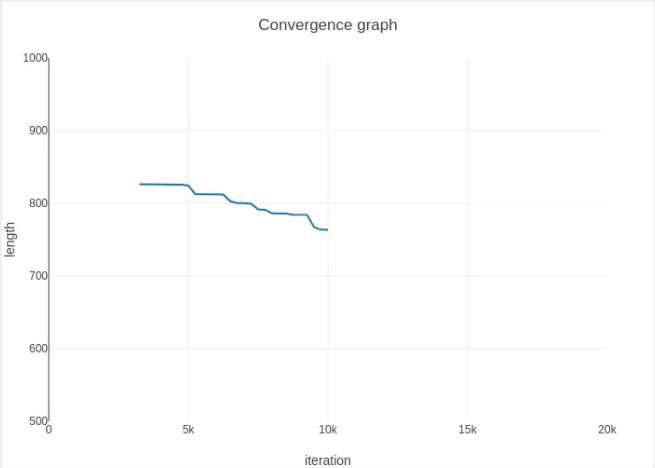


20k samples

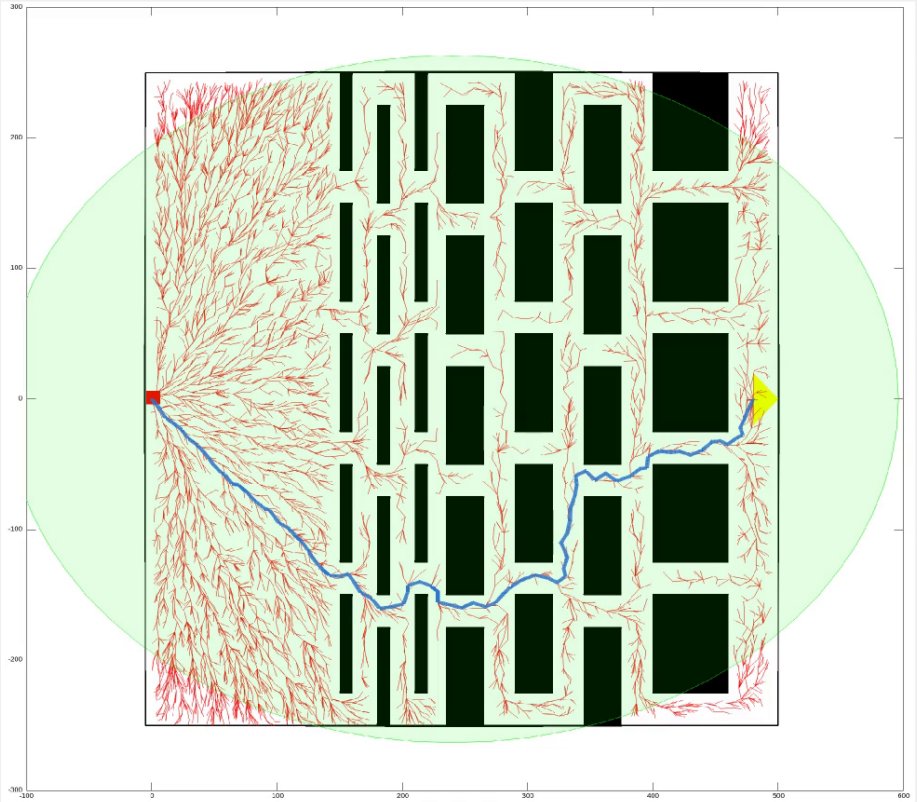
Basic RRT*

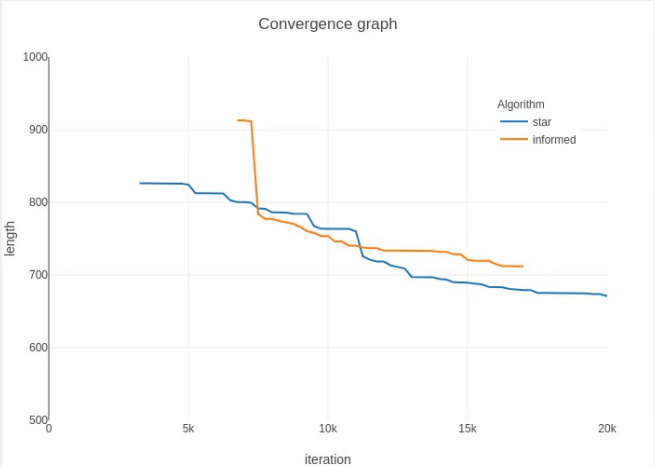
Motivation: slow convergence of RRT*-based planners





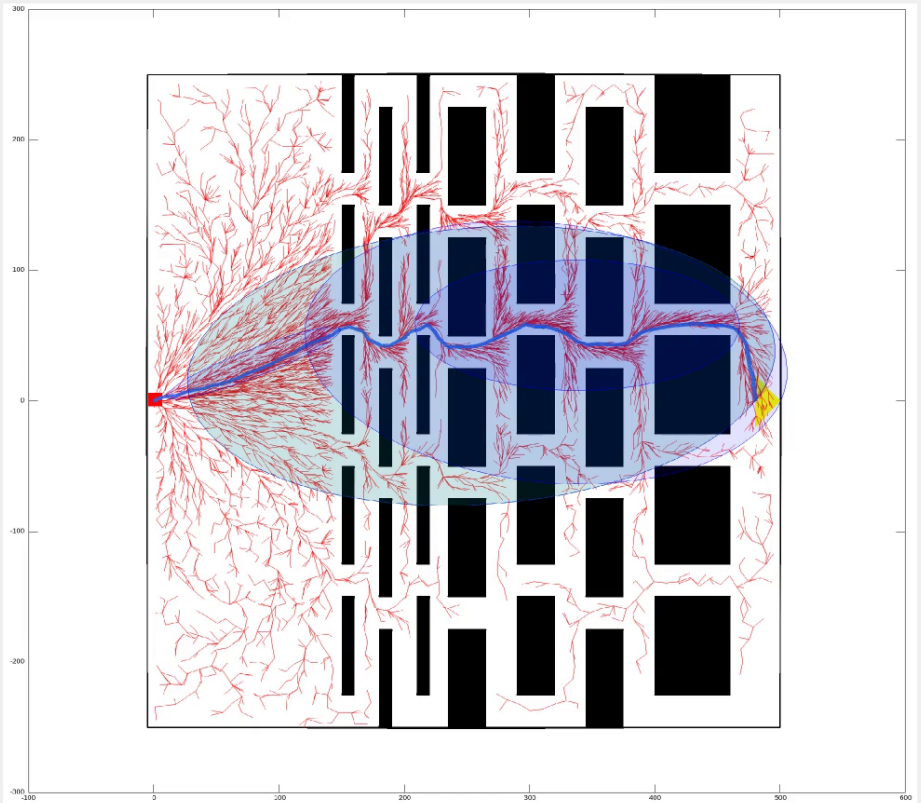
Informed RRT*: sampling from omniscient set

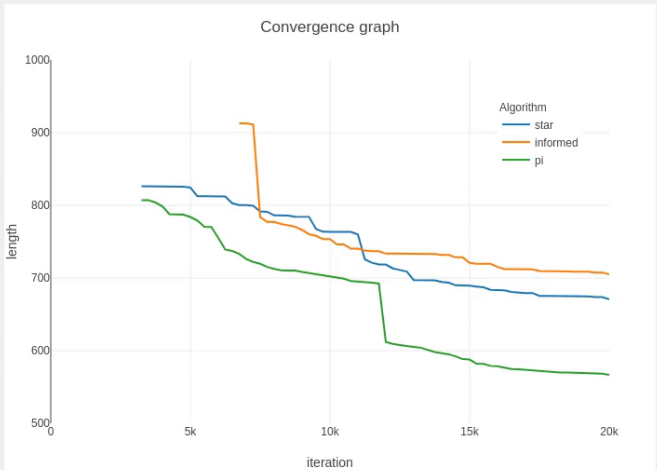




Comparison of RRT* variants

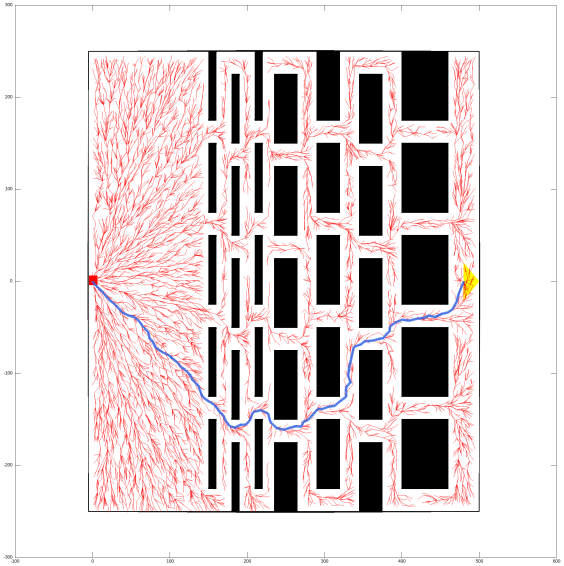
Locally informed space S_l + RRT*



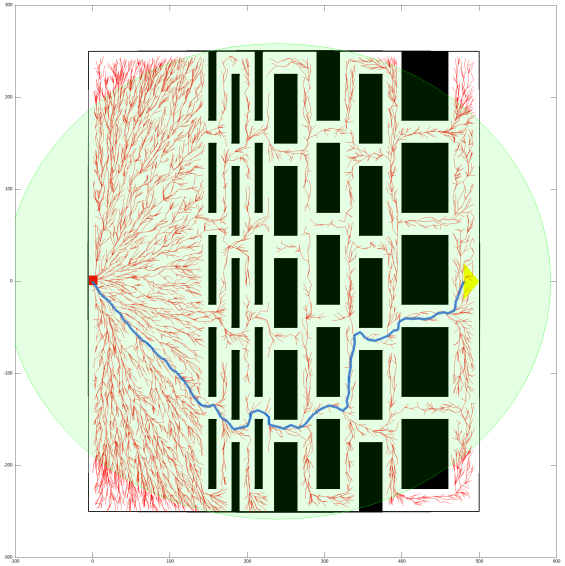


Comparison of RRT* variants

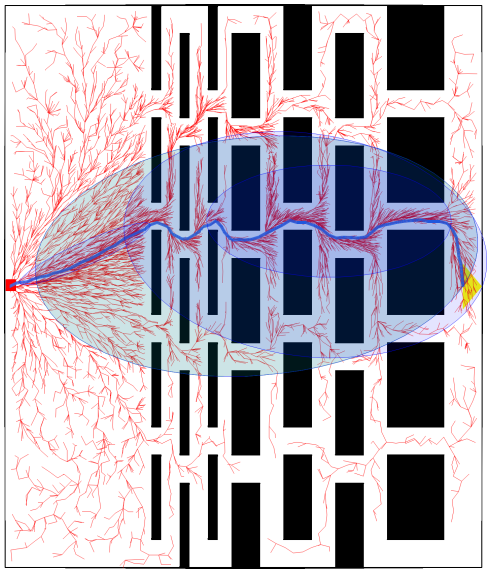
Note: paths at 20k samples



RRT*

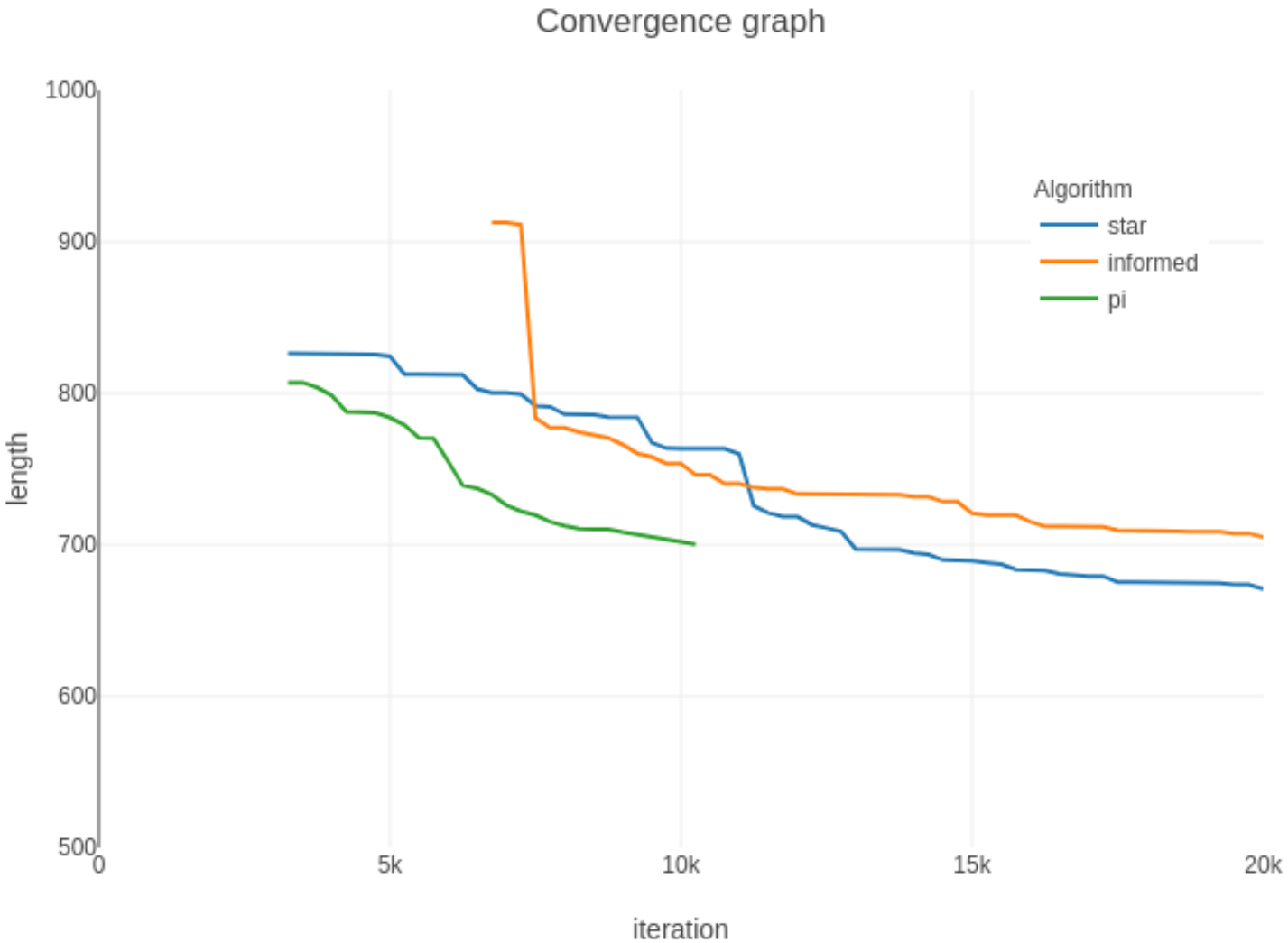


Informed RRT*



Partially informed RRT*

Comparison of RRT* variants



Algorithms properties

Algorithm	**Probabilistic completeness**	**Asymptotic optimality**
RRT	✓	✗
PRM	✓	✗
sPRM	✓	✓
k-sPRM	✗(if $k = 1$)	✗
PRM* / k-PRM*	✓	✓
RRT* / k-RRT*	✓	✓
Informed RRT*	✓	✓
Partially informed RRT*	✓	✓

- If you don't need optimal solution, use RRT/PRM
- RRT is faster than RRT*
- RRT is way easier to implement 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

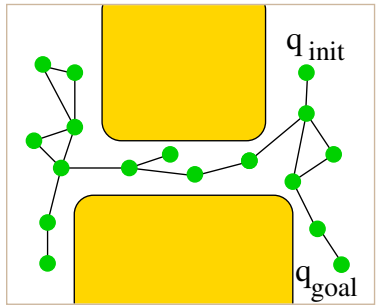
Single query vs. multi-query planning

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

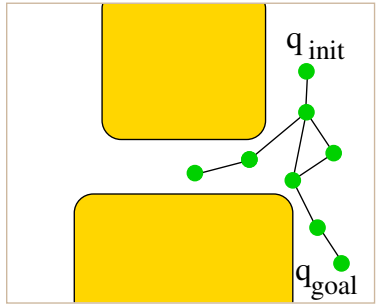
Single-query methods

- 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