16-782
Planning & Decision-making in Robotics

*Planning Representations/Search Algorithms: RRT, RRT-Connect, RRT**

Maxim Likhachev

Robotics Institute

Carnegie Mellon University

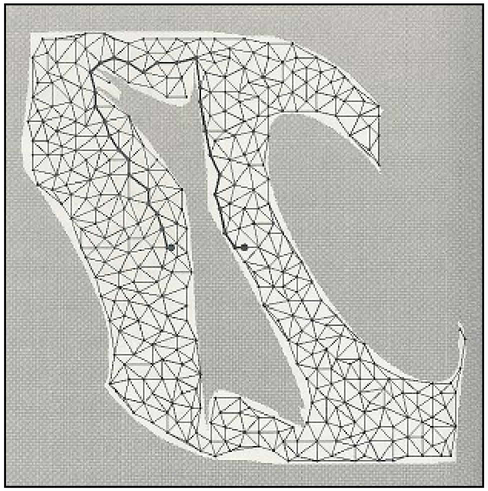
Probabilistic Roadmaps (PRMs)

Great for problems where a planner has to plan many times for different start/goal pairs (step 1 needs to be done only once)

Not so great for single shot planning

Step 1. Preprocessing Phase: Build a roadmap (graph) $\mathcal{G}$ which, hopefully, should be accessible from any point in C_{free}

Step 2. Query Phase: Given a start configuration q_I and goal configuration q_G, connect them to the roadmap $\mathcal{G}$ using a local planner, and then search the augmented roadmap for a shortest path from q_I to q_G



Rapidly Exploring Random Trees (RRTs) [LaValle, '98]

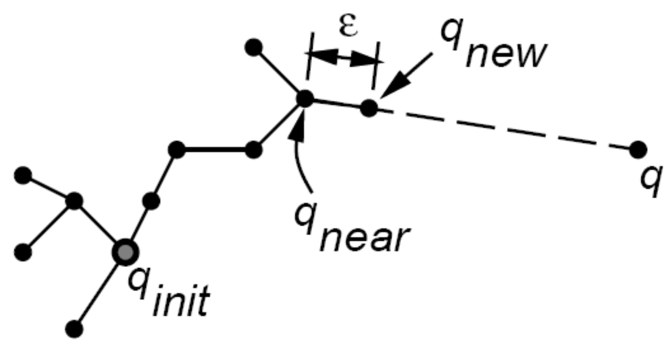
No preprocessing step: starting with the initial configuration q_I build the graph (actually, tree) until the goal configuration g_G is part of it

Very effective for single shot planning

Rapidly Exploring Random Trees (RRTs) [LaVallle, ’98]

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

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



EXTEND *operation*

borrowed from “RRT-Connect: An Efficient Approach to Single-Query Path Planning” paper by J. Kuffner & S. LaValle

Rapidly Exploring Random Trees (RRTs) [LaValle, '98]

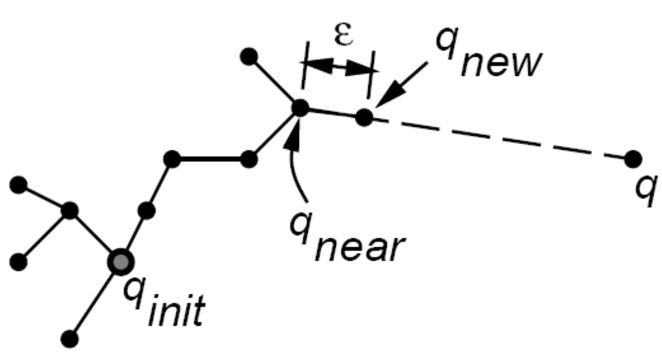
Path to the goal is a path in the tree from q_{init} to the vertex closest to goal

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

selects closest vertex in the tree

moves by at most ε from q_{near} towards q

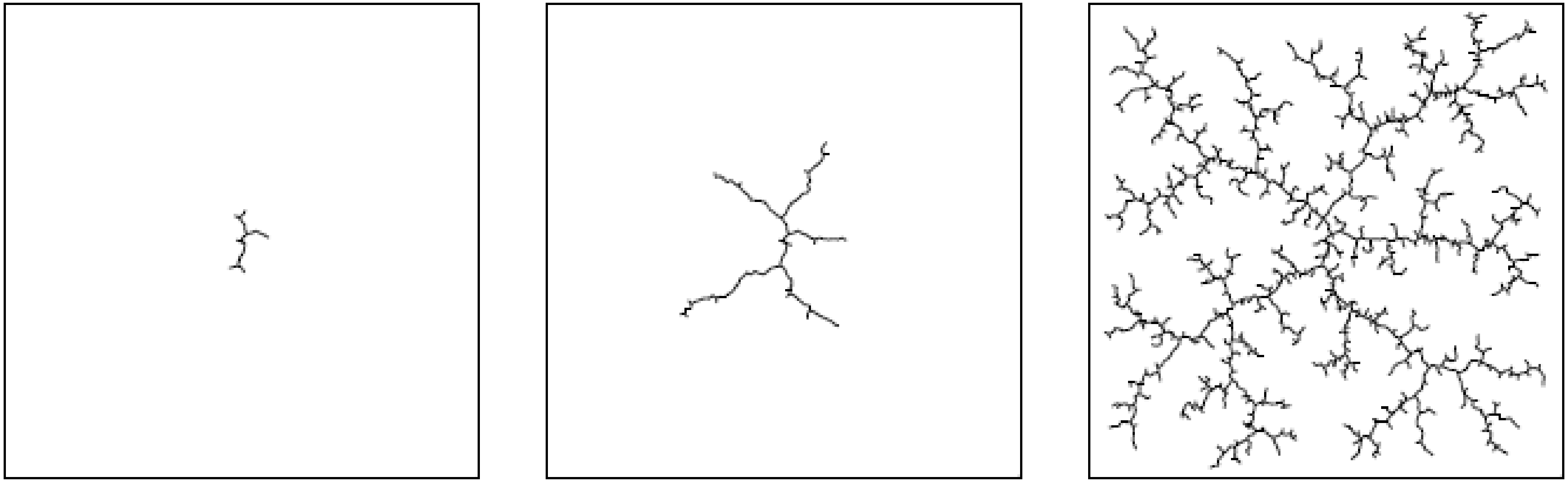
```
EXTEND(T, q)
    q_near ← NEAREST_NEIGHBOR(q, T);
    if NEW_CONFIG(q, q_near, q_new) then
        T.add_vertex(q_new);
        T.add_edge(q_near, q_new);
        if q_new = q then
            Return Reached;
        else
            Return Advanced;
    Return Trapped;
```



EXTEND *operation*

borrowed from "RRT-Connect: An Efficient Approach to Single-Query Path Planning" paper by J. Kuffner & S. LaValle

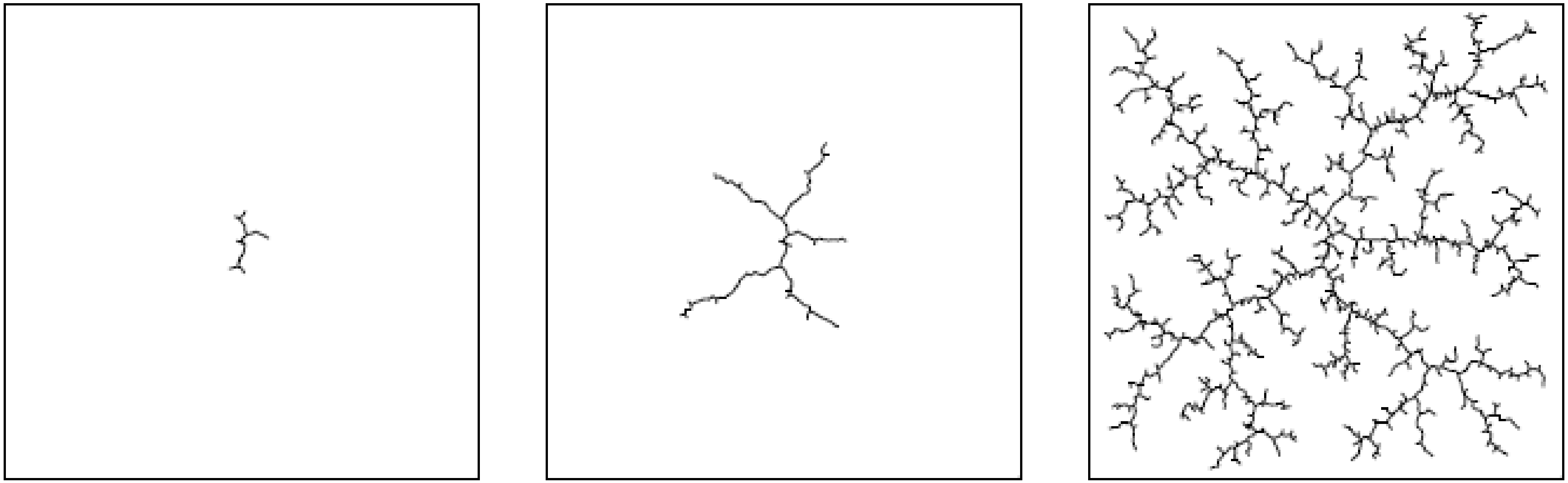
Rapidly Exploring Random Trees (RRTs) [LaValle, '98]



- RRT provides uniform coverage of space

borrowed from "RRT-Connect: An Efficient Approach to Single-Query Path Planning" paper by J. Kuffner & S. LaValle

Rapidly Exploring Random Trees (RRTs) [LaValle, '98]

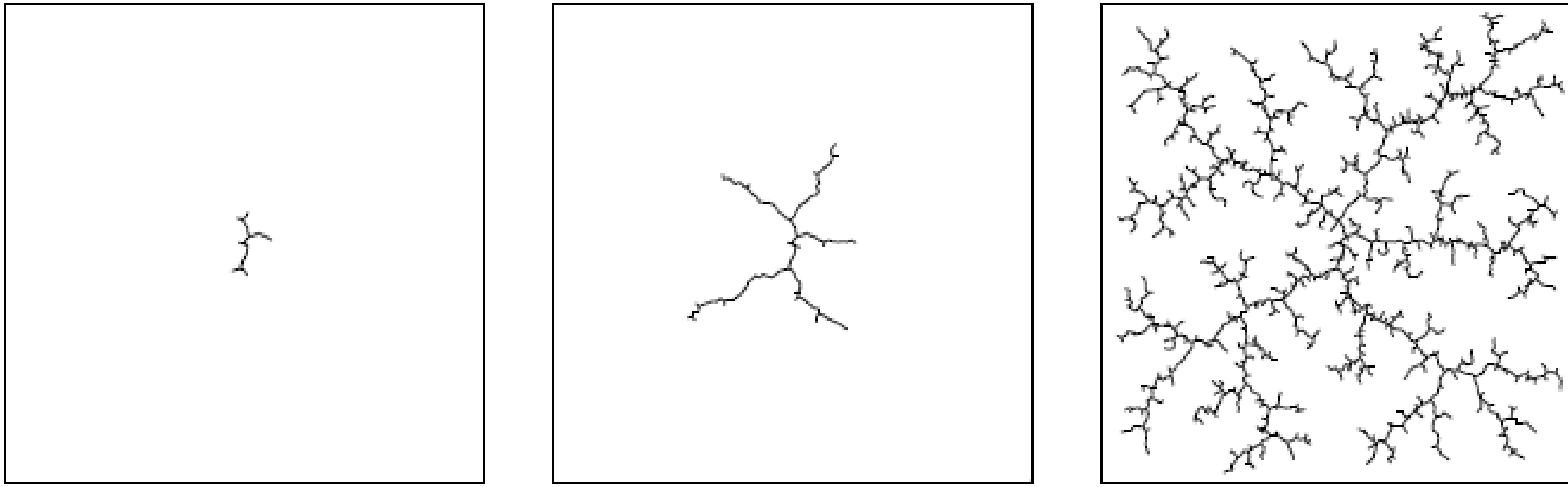


- RRT provides uniform coverage of space

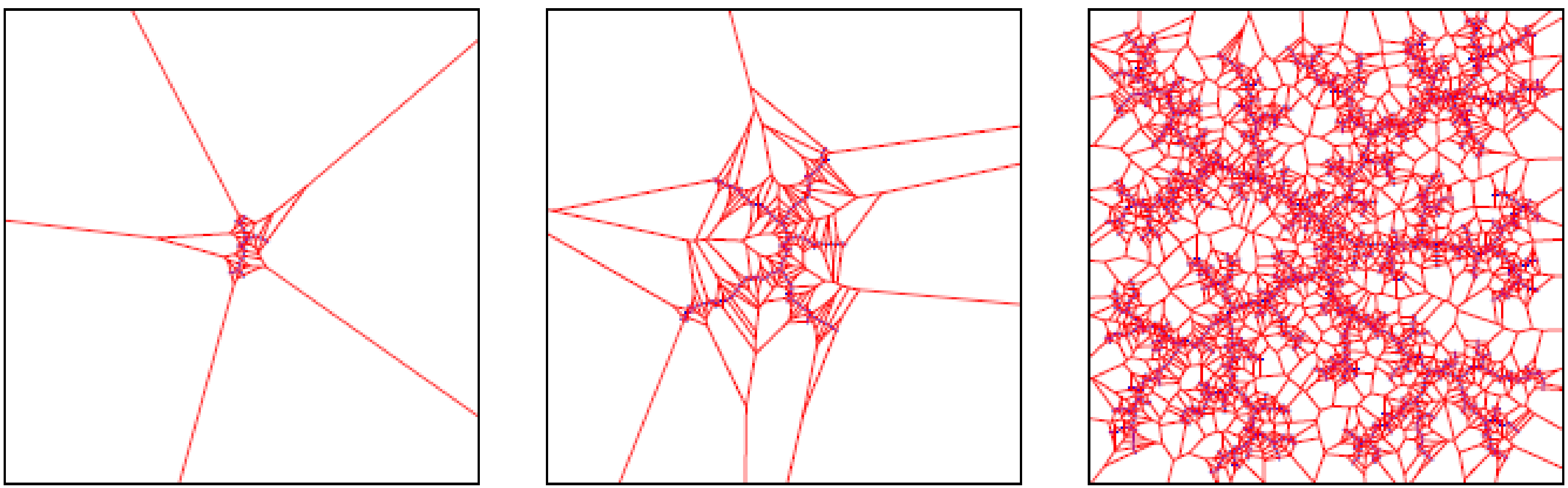
Pros/cons?

borrowed from "RRT-Connect: An Efficient Approach to Single-Query Path Planning" paper by J. Kuffner & S. LaValle

Rapidly Exploring Random Trees (RRTs) [LaValle, '98]

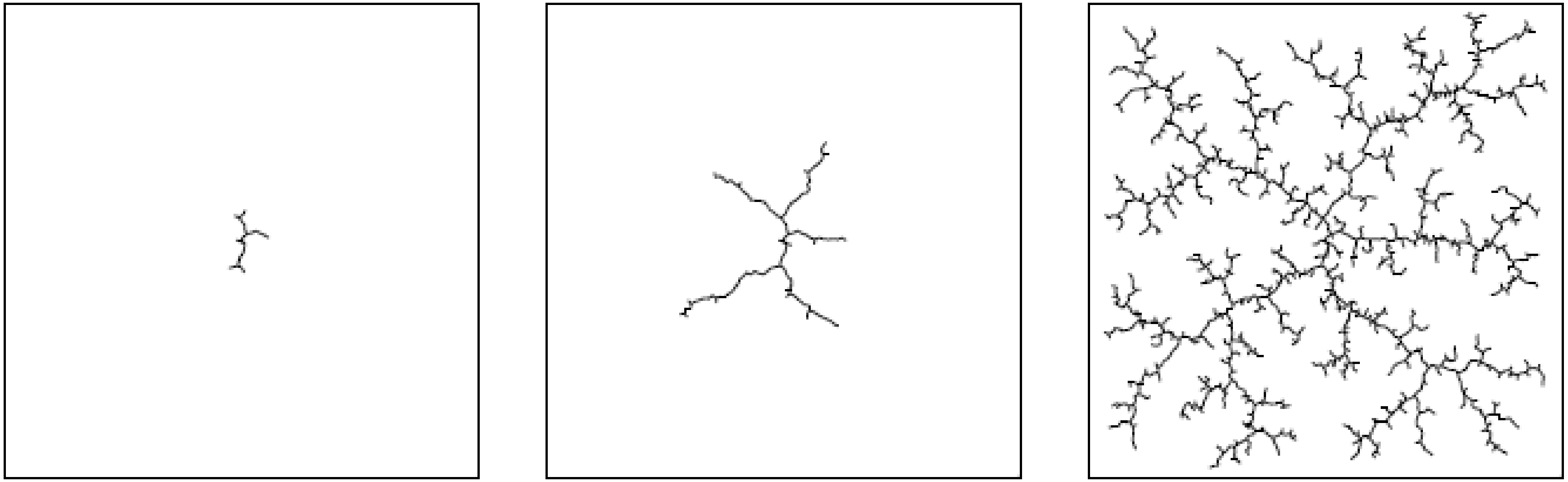


- Alternatively, the growth is always biased by the largest unexplored region

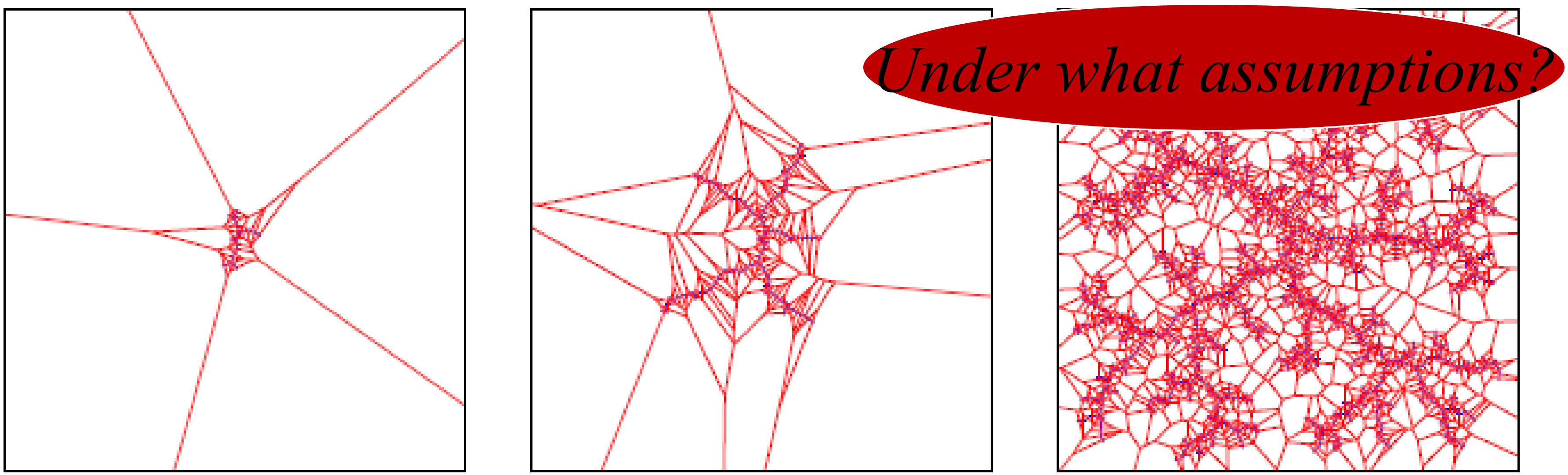


borrowed from "RRT-Connect: An Efficient Approach to Single-Query Path Planning" paper by J. Kuffner & S. LaValle

Rapidly Exploring Random Trees (RRTs) [LaValle, '98]



- Alternatively, the growth is always biased by the largest unexplored region



borrowed from "RRT-Connect: An Efficient Approach to Single-Query Path Planning" paper by J. Kuffner & S. LaValle

RRT-Connect [Kuffner & LaValle, '00]

Bi-directional growth of the tree

+

relax the ε constraint on the growth of the tree

RRT-Connect [Kuffner & LaValle, '00]

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

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

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

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

borrowed from "RRT-Connect: An Efficient Approach to Single-Query Path Planning" paper by J. Kuffner & S. LaValle

RRT-Connect [Kuffner & LaValle, ‘00]

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

tries to grow T_b to q_{new} that was just added to T_a

Why swap the trees?

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

CONNECT function grows the tree by more than just one ε

borrowed from “RRT-Connect: An Efficient Approach to Single-Query Path Planning” paper by J. Kuffner & S. LaValle

RRT-Connect [Kuffner & LaValle, '00]

- For any $q \in C_{free}$, $lim_{k \rightarrow \infty} P[d(q) < \varepsilon] = 1$, where $d(q)$ is a distance from configuration q to the closest vertex in the tree, and assuming C_{free} is connected, bounded and open
- RRT-Connect is probabilistically complete: *as # of samples approaches infinity, the algorithm is guaranteed to find a solution if one exists*

RRT-Connect [Kuffner & LaValle, '00]

- For any $q \in C_{free}$, $lim_{k \to \infty} P[d(q) < \varepsilon] = 1$, where $d(q)$ is a distance from configuration q to the closest vertex in the tree, and assuming C_{free} is connected, bounded and open
- RRT-Connect is probabilistically complete: *as # of samples approaches infinity, the algorithm is guaranteed to find a solution if one exists*

Is RRT-Connect asymptotically (as $k \to \infty$) optimal?

No, more on this later

RRT-Connect [Kuffner & LaValle, '00]

- For any $q \in C_{free}$, $\lim_{k\rightarrow\infty} P[d(q) < \varepsilon] = 1$, where $d(q)$ is a distance from configuration q to the closest vertex in the tree, and assuming C_{free} is connected, bounded and open

- RRT-Connect is probabilistically complete: *as # of samples approaches infinity, the algorithm is guaranteed to find a solution if one exists*

Applicability of RRT vs. RRT-Connect to kinodynamic planning?

Sampling-based approaches

Typical setup:

- Run PRM/RRT/RRT-Connect/…
- Post-process the generated solution to make it more optimal

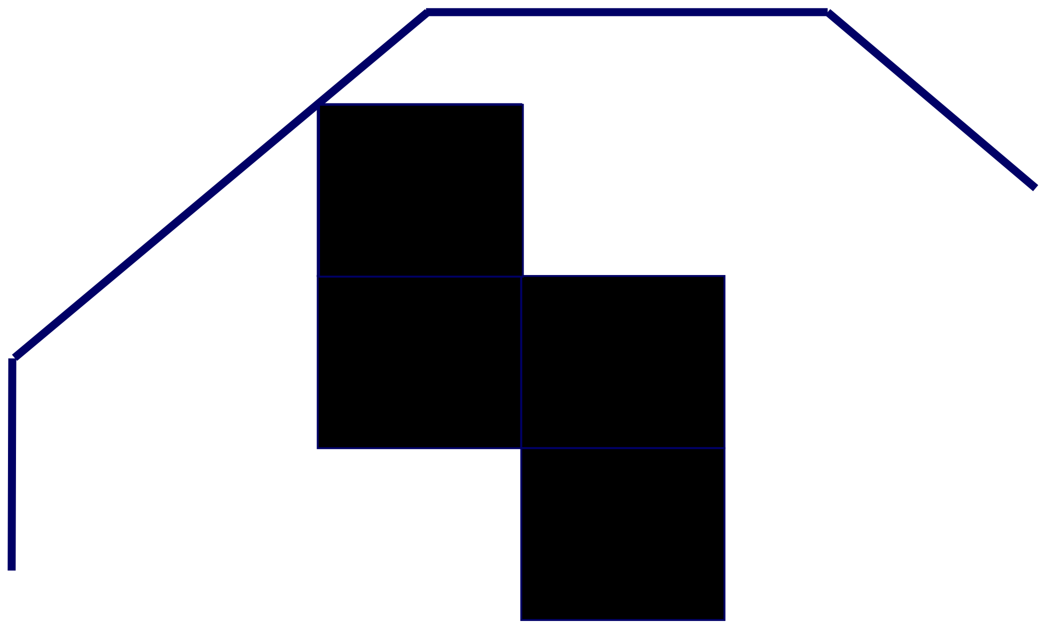
An important but often time-consuming step

Could also be highly non-trivial

Post-processing

Any ideas how to post-process it?

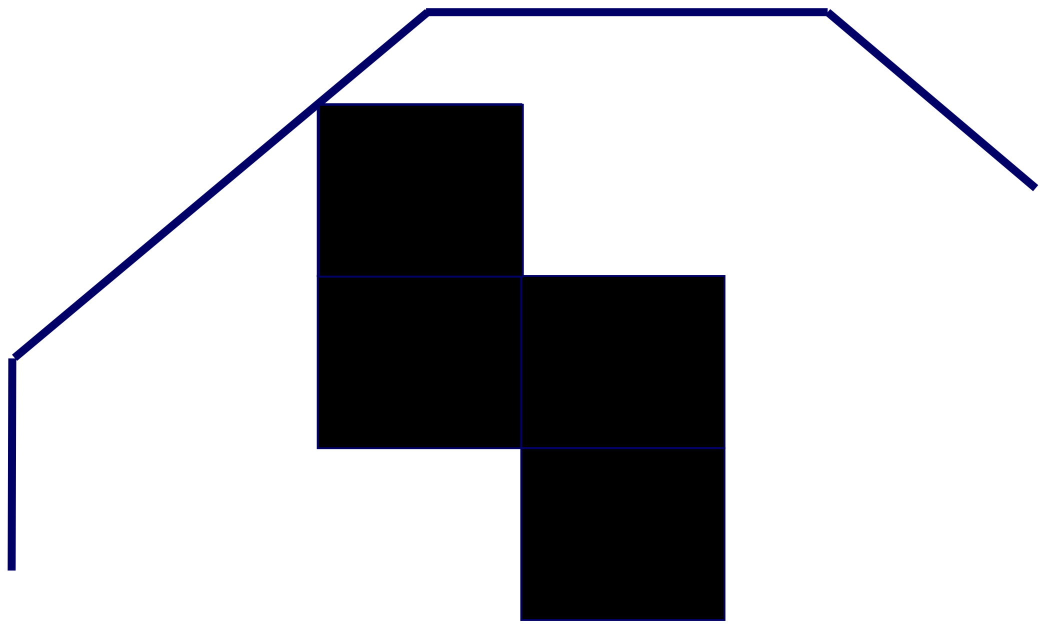
Consider this path generated by RRT or PRM or A on a grid-based graph:*



Simple Post-processing via Short-cutting

- Short-cutting a path consisting of a series of points

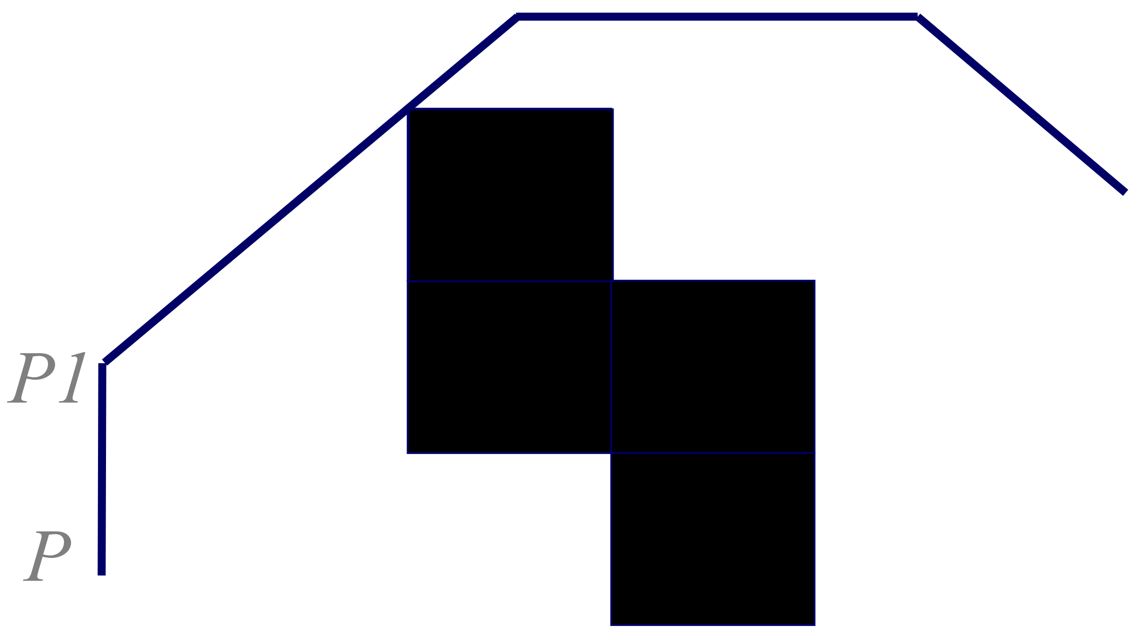
NewPath=[]; P=start point, P1 = point P+1 along the path
while P != goal point
while line segment [P,P1+1] is obstacle-free AND P1+1 < goal point
P1 = point P1+1 along the path;
NewPath+= [P,P1]; P = P1; P1 = point P+1 along the path;



Simple Post-processing via Short-cutting

- Short-cutting a path consisting of a series of points

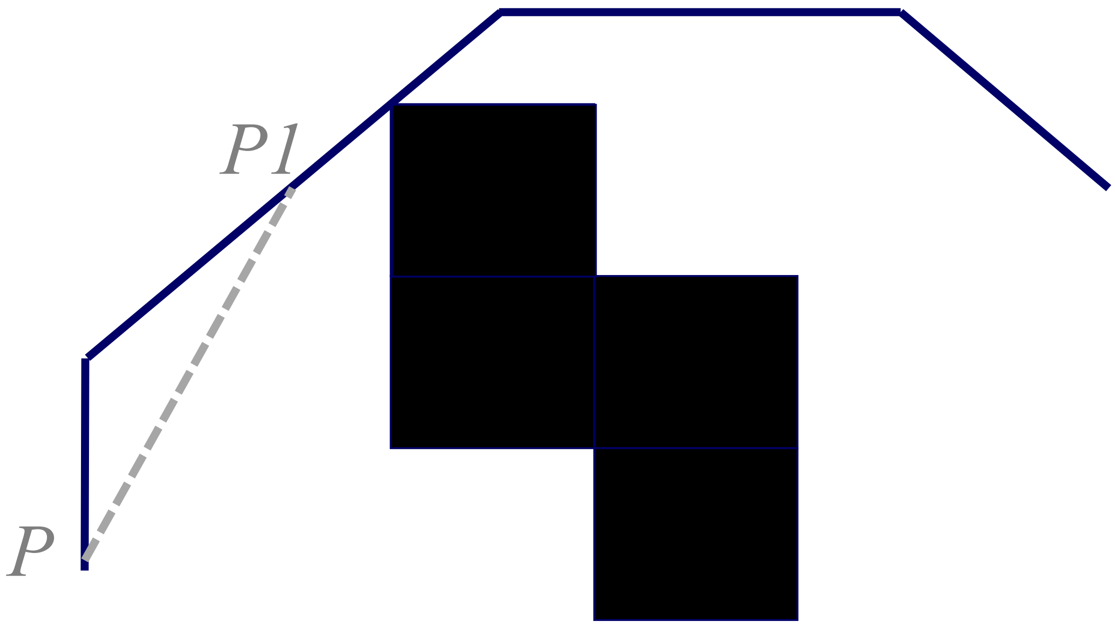
NewPath=[]; P=start point, P1 = point P+1 along the path
while P != goal point
while line segment [P,P1+1] is obstacle-free AND P1+1 < goal point
P1 = point P1+1 along the path;
NewPath+= [P,P1]; P = P1; P1 = point P+1 along the path;



Simple Post-processing via Short-cutting

- Short-cutting a path consisting of a series of points

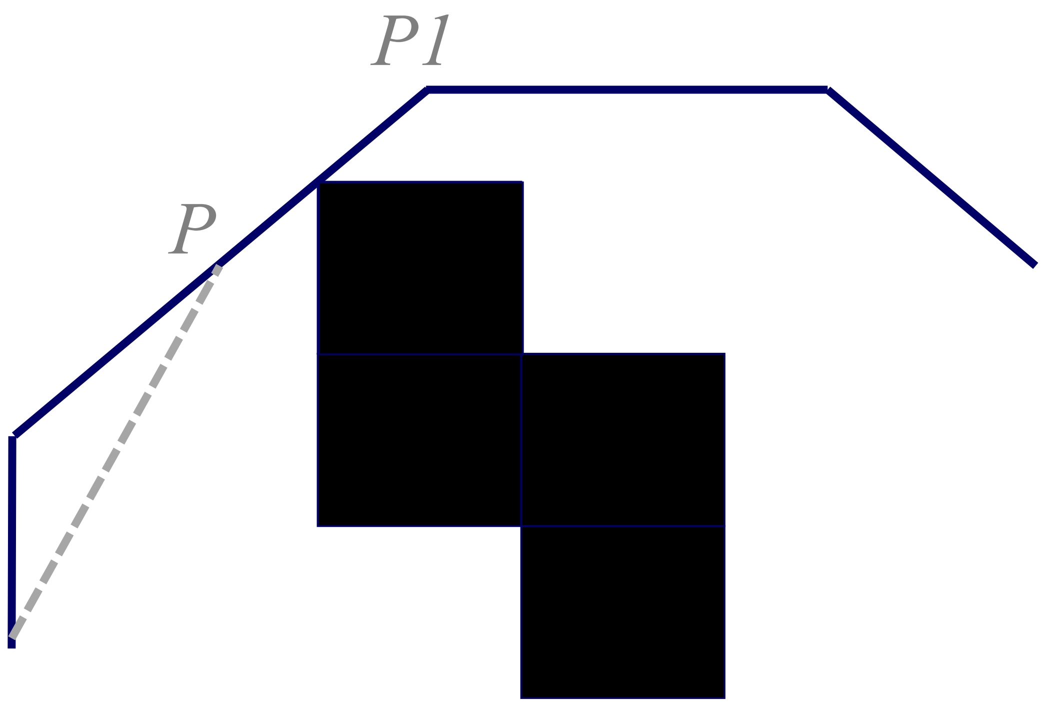
  NewPath=[]; P=start point, P1 = point P+1 along the path
  while P != goal point
  while line segment [P,P1+1] is obstacle-free AND P1+1 < goal point
  P1 = point P1+1 along the path;
  NewPath+= [P,P1]; P = P1; P1 = point P+1 along the path;



Simple Post-processing via Short-cutting

- Short-cutting a path consisting of a series of points

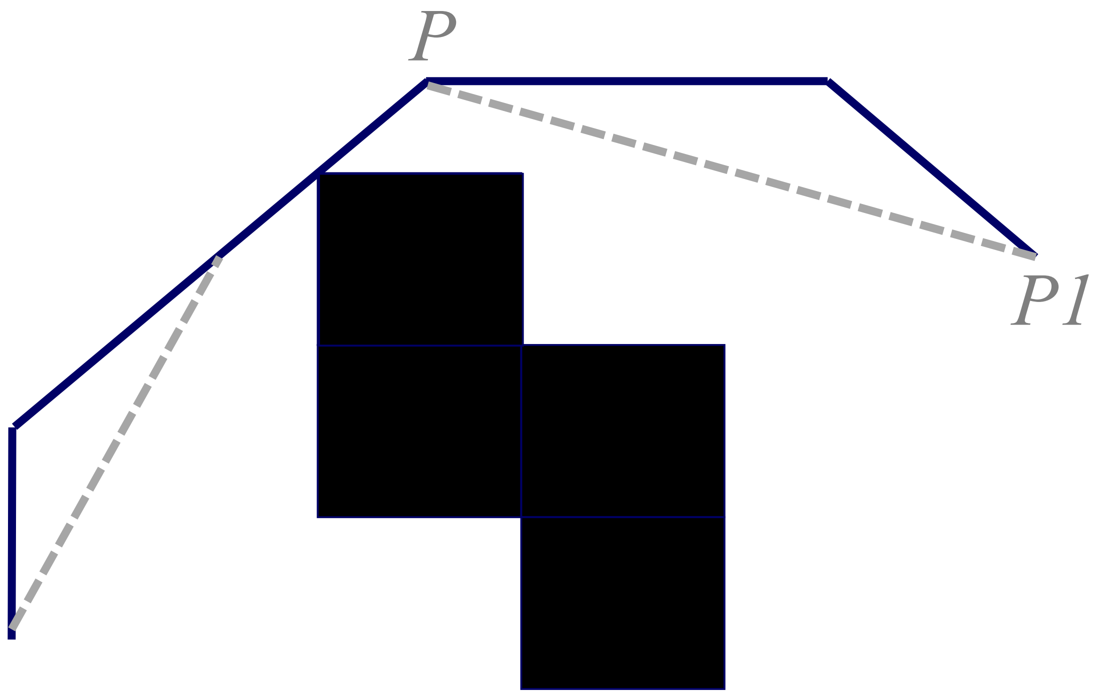
  NewPath=[]; P=start point, P1 = point P+1 along the path
  while P != goal point
  while line segment [P,P1+1] is obstacle-free AND P1+1 < goal point
  P1 = point P1+1 along the path;
  NewPath+= [P,P1]; P = P1; P1 = point P+1 along the path;



Simple Post-processing via Short-cutting

- Short-cutting a path consisting of a series of points

NewPath=[]; P=start point, P1 = point P+1 along the path
while P != goal point
    while line segment [P,P1+1] is obstacle-free AND P1+1 < goal point
        P1 = point P1+1 along the path;
    NewPath+= [P,P1]; P = P1; P1 = point P+1 along the path;



Simple Post-processing via Short-cutting

- Short-cutting a path consisting of a series of points

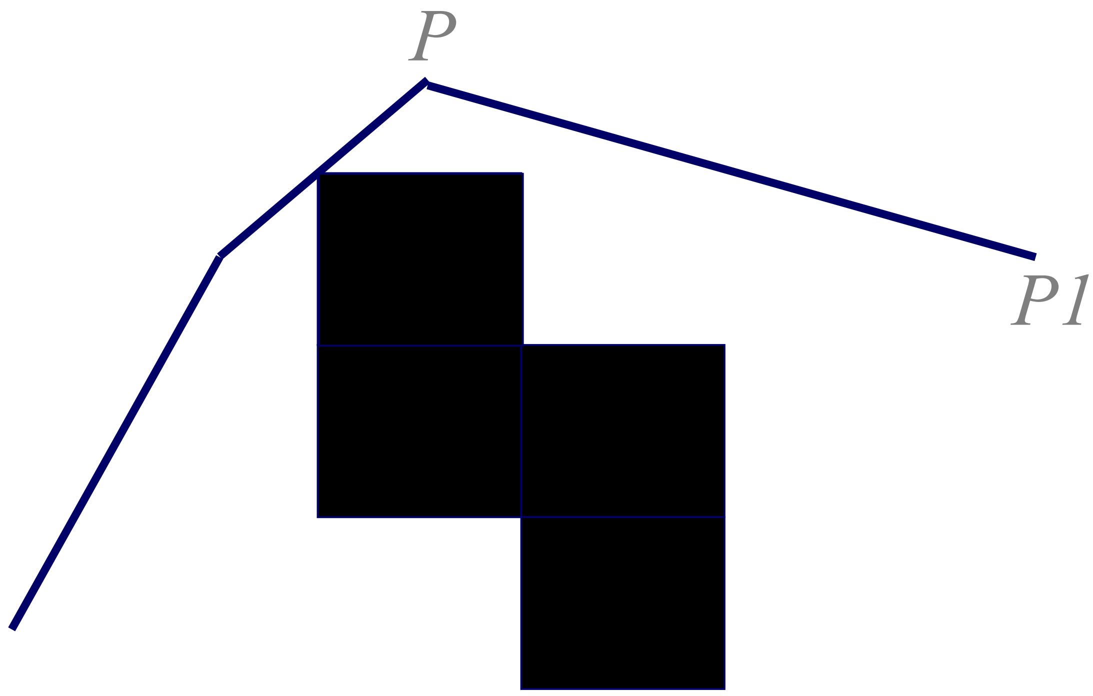
NewPath=[]; P=start point, P1 = point P+1 along the path

while P != goal point

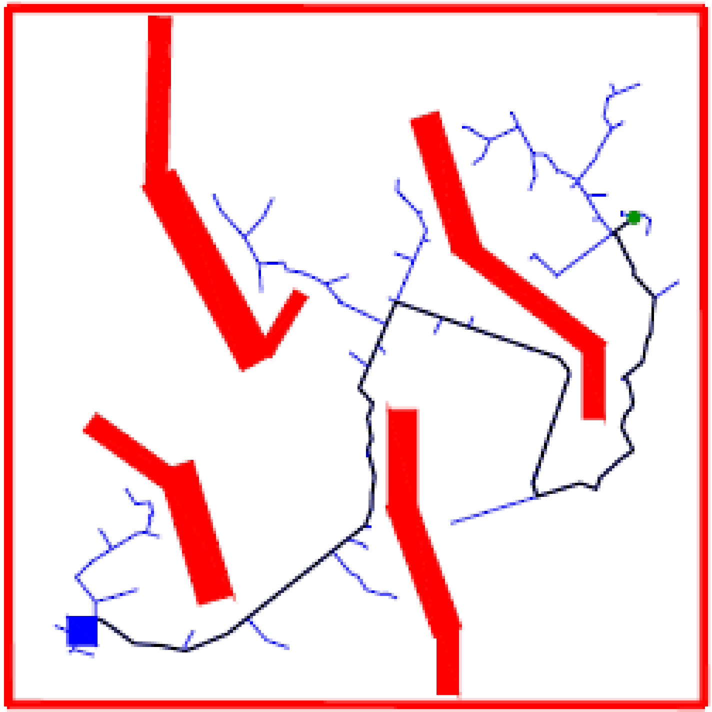
while line segment [P,P1+1] is obstacle-free AND P1+1 < goal point

P1 = point P1+1 along the path;

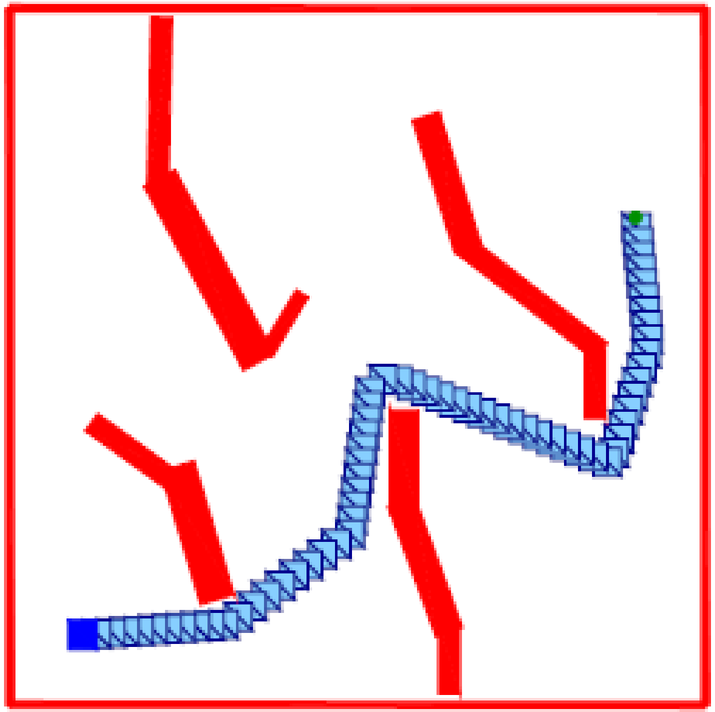
NewPath+= [P,P1]; P = P1; P1 = point P+1 along the path;



Examples of RRT in action



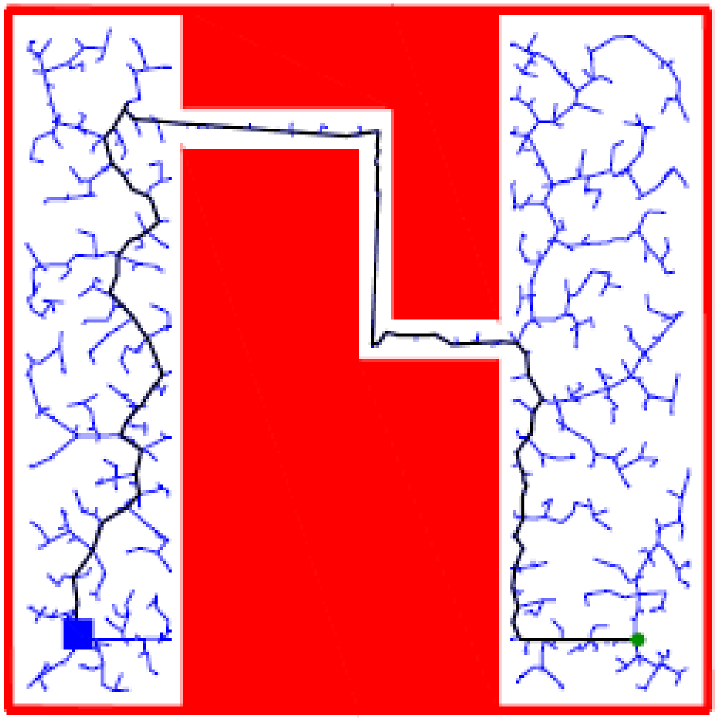
RRT-connect



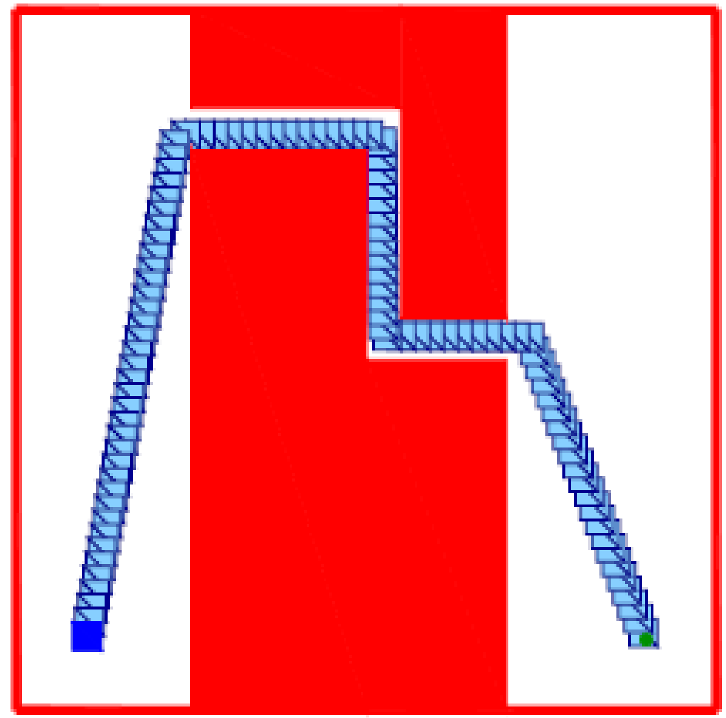
path after postprocessing

borrowed from "RRT-Connect: An Efficient Approach to Single-Query Path Planning" paper by J. Kuffner & S. LaValle

Examples of RRT in action



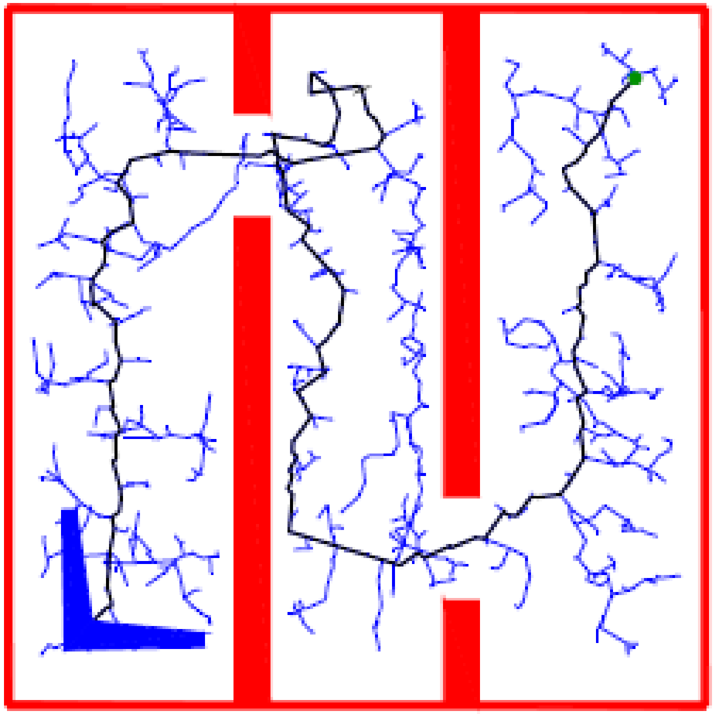
RRT-connect



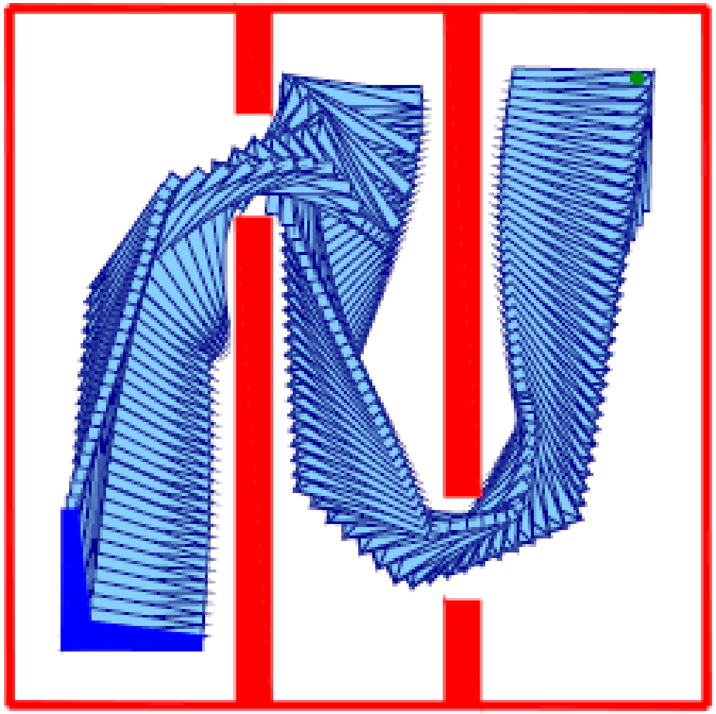
path after postprocessing

borrowed from "RRT-Connect: An Efficient Approach to Single-Query Path Planning" paper by J. Kuffner & S. LaValle

Examples of RRT in action



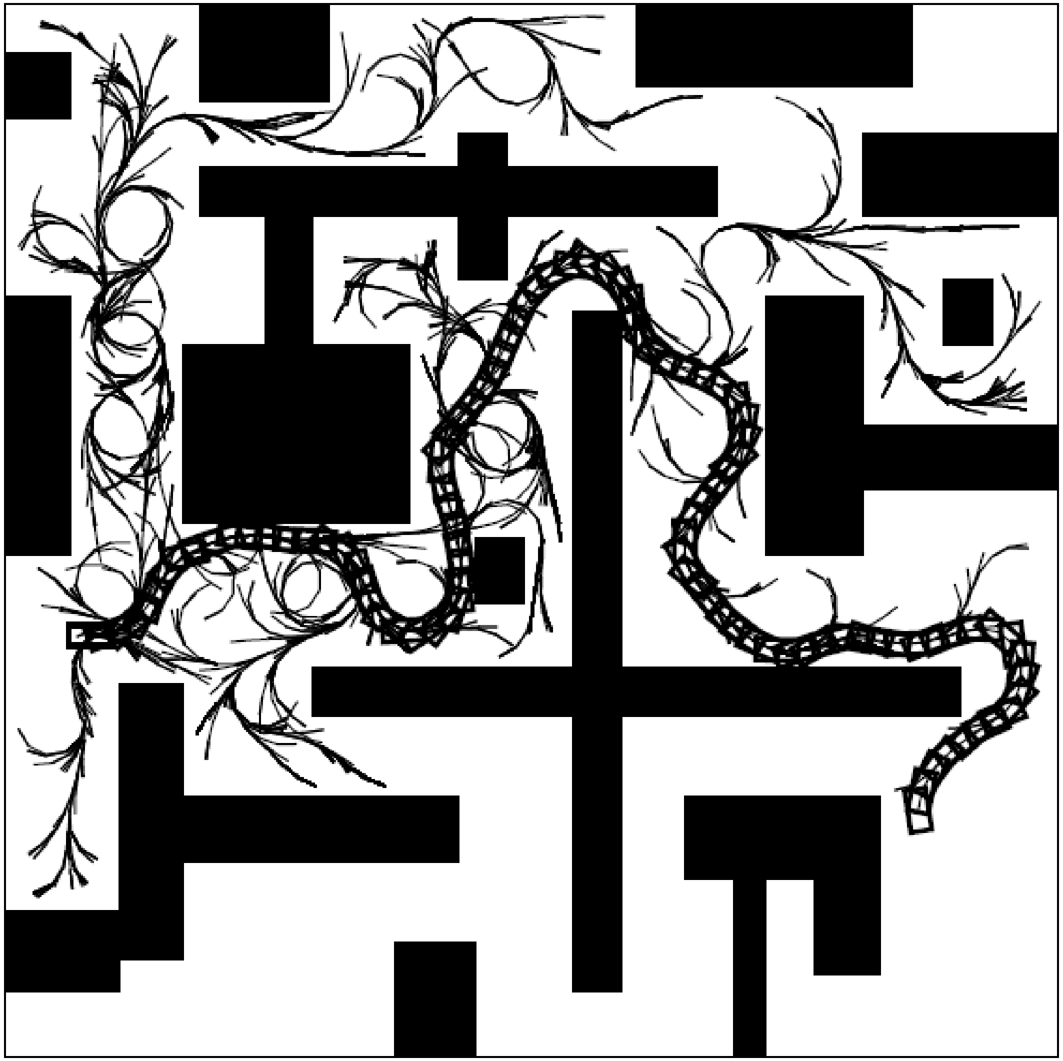
RRT-connect



path after postprocessing

borrowed from "RRT-Connect: An Efficient Approach to Single-Query Path Planning" paper by J. Kuffner & S. LaValle

Examples of RRT



5DOF kinodynamic planning for a car

borrowed from "Rapidly-Exploring Random Trees: A new tool for Path Planning" paper by S. LaValle

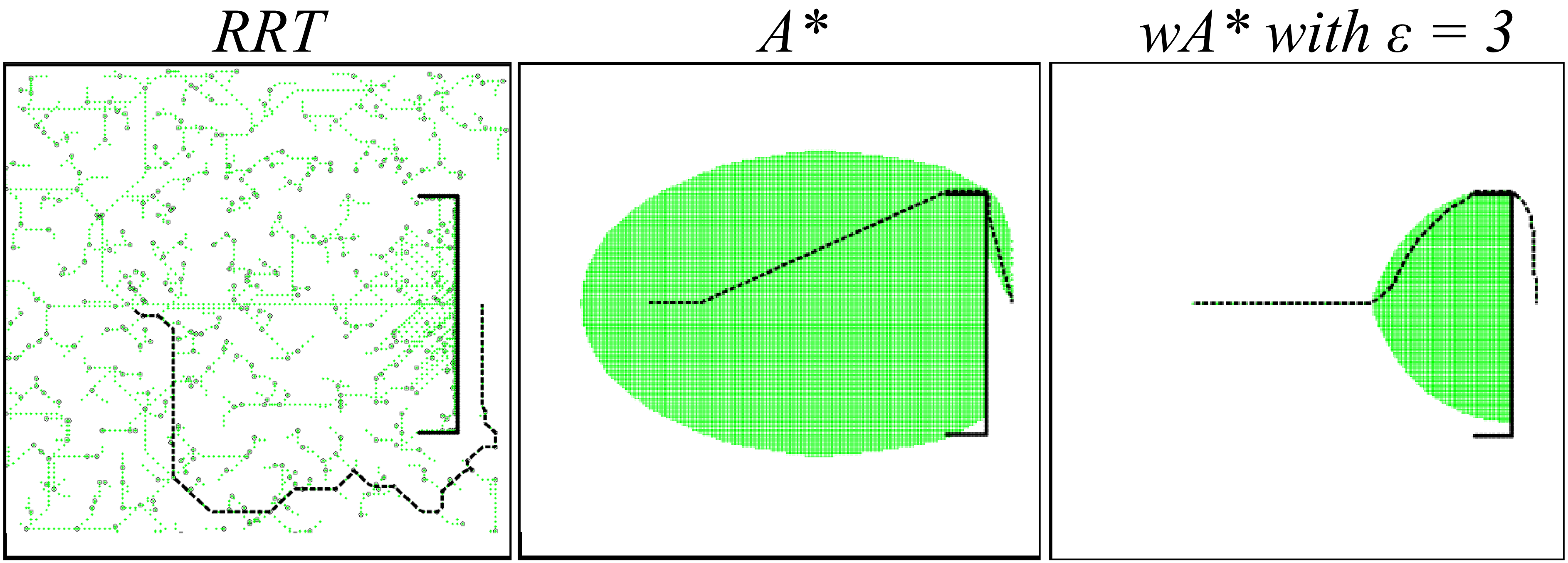
PRMs vs. RRTs

- PRMs construct a roadmap and then searches it for a solution whenever q_I, g_G are given
  - well-suited for repeated planning in between different pairs of q_I, g_G *(multiple queries)*

- RRTs construct a tree for a given q_I, q_G until the tree has a solution
  - well-suited for single-shot planning in between a single pair of q_I, g_G *(single query)*

  - There exist extensions of RRTs that try to reuse a previously constructed tree when replanning in response to map updates

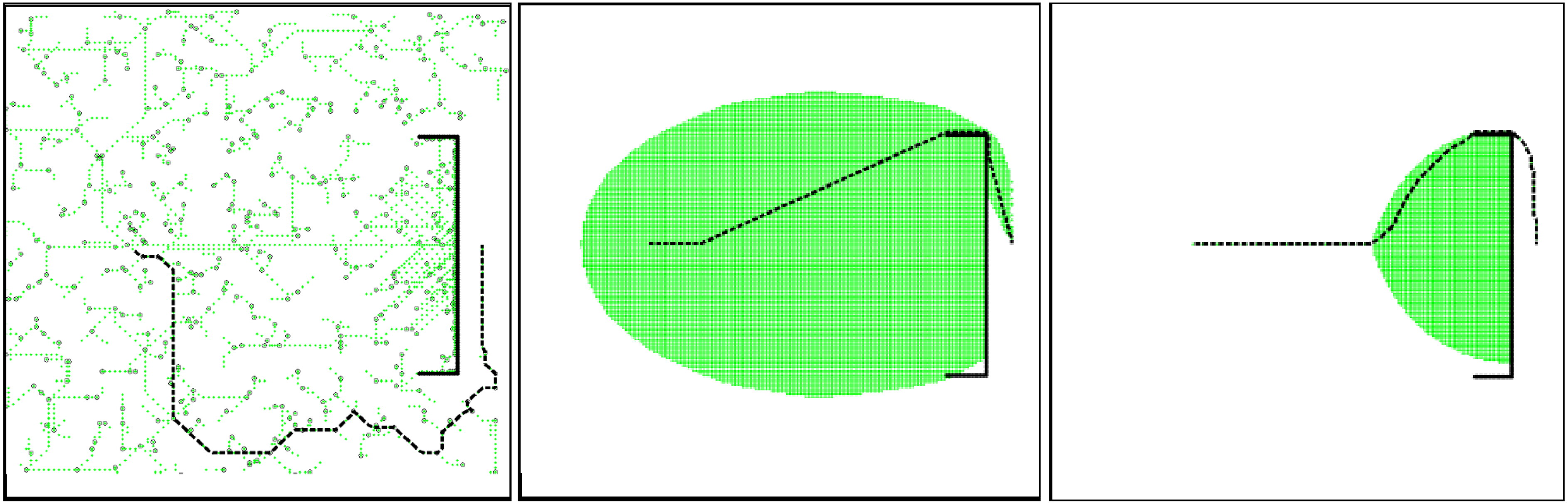
RRTs vs A*-based planning



- RRTs:
  - sparse exploration, usually little memory and computations required, works well in high-D
  - solutions can be highly sub-optimal, requires post-processing, which in some cases can be very hard to do, the solution is still restricted to the same homotopic class

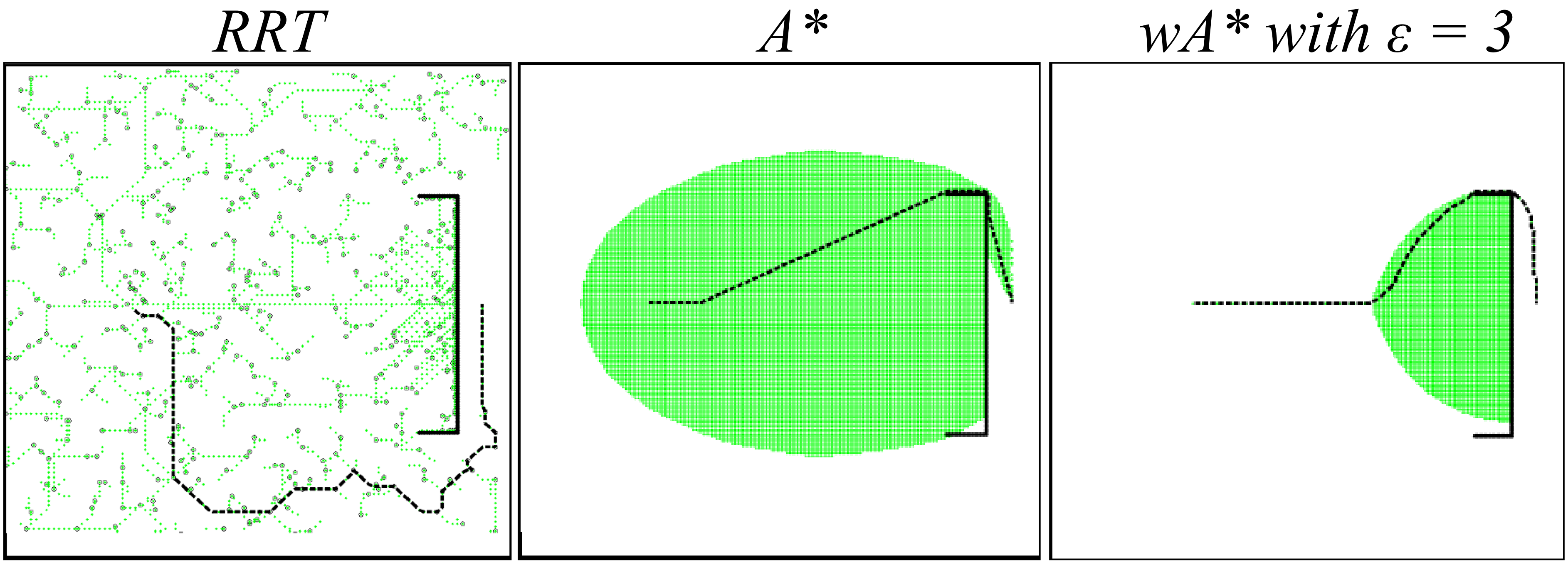
RRTs vs A*-based planning

RRT *A** *wA* with ε = 3*



- RRTs:
  - does not incorporate a (potentially complex) cost function
  - there exist versions (e.g., RRT*) that try to incorporate the cost function and converge to a provably least-cost solution in the limit of samples (but typically computationally more expensive than RRT)

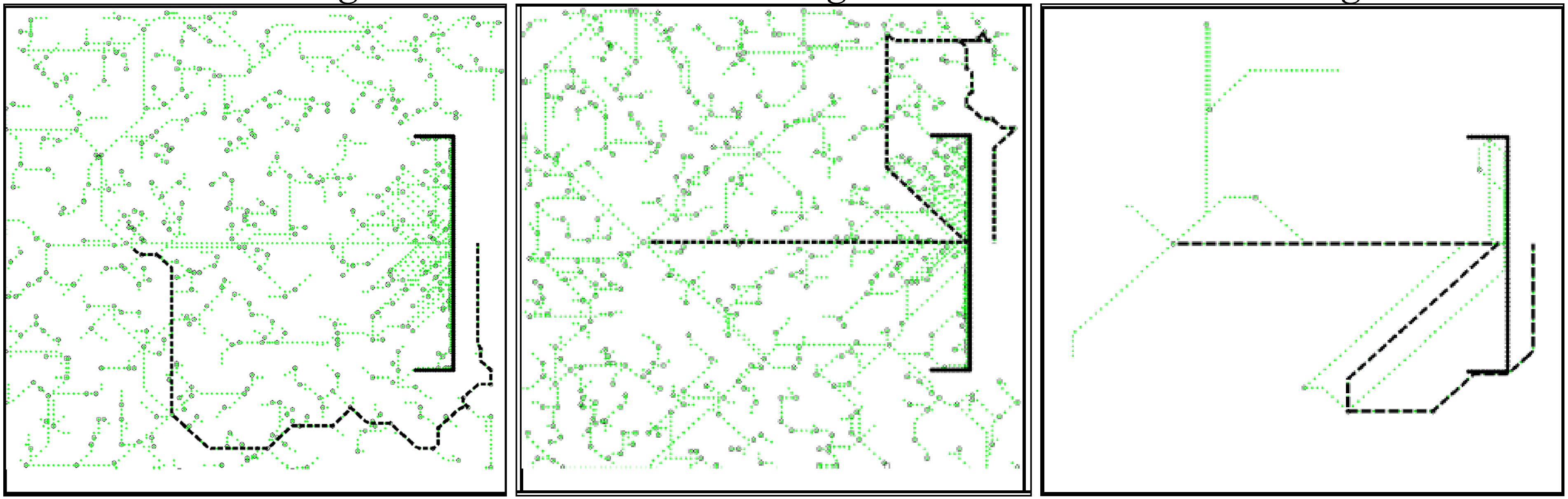
RRTs vs A*-based planning



- A* and weighted A* (wA*):
  - returns a solution with optimality (or sub-optimality) guarantees with respect to the discretization used
  - explicitly minimizes a cost function
  - requires a thorough exploration of the state-space resulting in high memory and computational requirements

Sampling in RRTs

RRT, P_g=0 *RRT, P_g=0.1* *RRT, P_g=0.5*

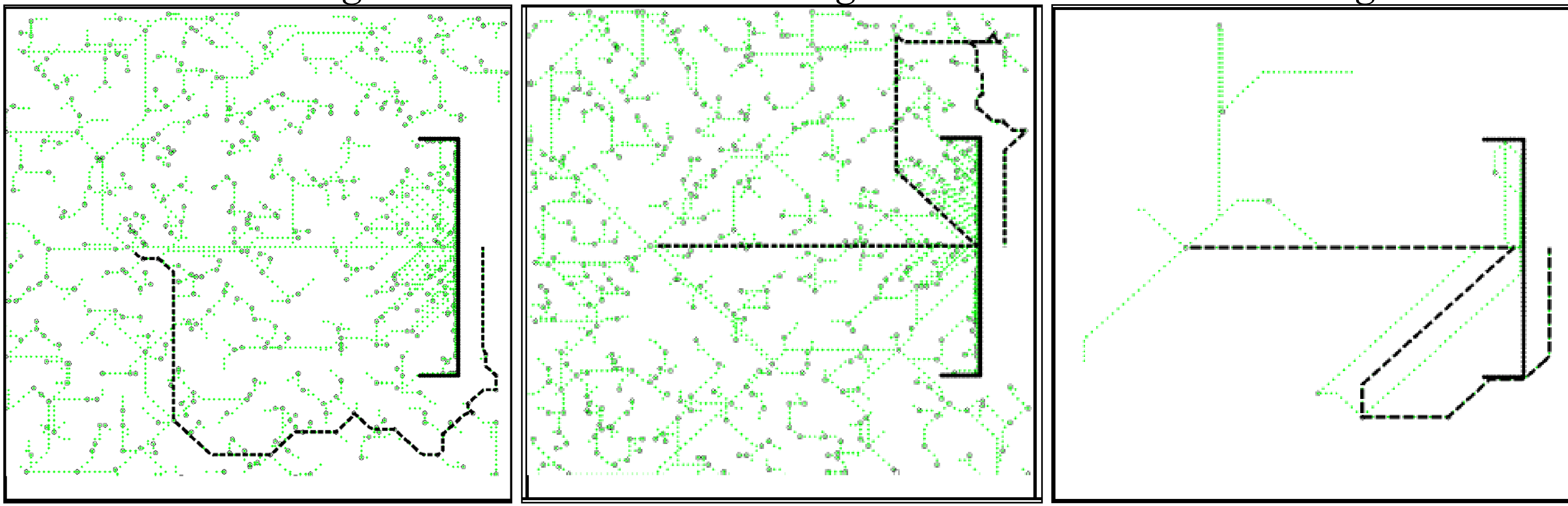


- Uniform: q_{rand} is a random sample in C_{free}

- Goal-biased: with a probability $(1\text{-}P_g)$, q_{rand} is chosen as a random sample in C_{free}, with probability P_g, q_{rand} is set to g_G

Sampling in RRTs

RRT, P_g=0 *RRT, P_g=0.1* *RRT, P_g=0.5*



- Uniform: q_{rand} is a random sample in C_{free}
- Goal-biased: with a probability $(1\text{-}P_g)$, q_{rand} is chosen as a random sample in C_{free}, with probability P_g, q_{rand} is set to g_G

Very useful!

RRT* [Karaman & Frazzoli, ‘06]

RRT

+

“re-wiring of nodes”

Properties of RRT again…

Is RRT
asymptotically (in the limit of the number of samples) complete?

Is RRT
asymptotically (in the limit of the number of samples) optimal?

Why?

RRT* [Karaman & Frazzoli, '06]

Main loop (same as in RRT):

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$; $i \leftarrow 0$;
while $i < N$ **do**
  $G \leftarrow (V, E)$;
  $x_{\text{rand}} \leftarrow \texttt{Sample}(i)$; $i \leftarrow i + 1$;
  $(V, E) \leftarrow \texttt{Extend}(G, x_{\text{rand}})$;

Extend(G,x) (same as in RRT + "re-wiring"):

$V' \leftarrow V$; $E' \leftarrow E$;
$x_{\text{nearest}} \leftarrow \texttt{Nearest}(G, x)$;
$x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x)$;
if $\texttt{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
  $V' \leftarrow V' \cup \{x_{\text{new}}\}$;
  $x_{\text{min}} \leftarrow x_{\text{nearest}}$;
  $X_{\text{near}} \leftarrow \texttt{Near}(G, x_{\text{new}}, |V|)$;
  for *all* $x_{\text{near}} \in X_{\text{near}}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{near}}, x_{\text{new}})$ **then**
      $c' \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$;
      if $c' < \texttt{Cost}(x_{\text{new}})$ **then**
        $x_{\text{min}} \leftarrow x_{\text{near}}$;
  $E' \leftarrow E' \cup \{(x_{\text{min}}, x_{\text{new}})\}$;
  for *all* $x_{near} \in X_{\text{near}} \setminus \{x_{\text{min}}\}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{new}}, x_{\text{near}})$ *and* $\texttt{Cost}(x_{\text{near}}) > \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}}))$ **then**
      $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E' \leftarrow E' \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$;
      $E' \leftarrow E' \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
return $G' = (V', E')$

borrowed from "Incremental Sampling-based Algorthms for Optimal Motion Planning" paper by S. Karaman & E. Frazzoli

RRT* [Karaman & Frazzoli, '06]

Main loop (same as in RRT):

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$; $i \leftarrow 0$;
while $i < N$ **do**
  $G \leftarrow (V, E)$;
  $x_{\text{rand}} \leftarrow \texttt{Sample}(i)$; $i \leftarrow i + 1$;
  $(V, E) \leftarrow \texttt{Extend}(G, x_{\text{rand}})$;

Extend(G,x) (same as in RRT + "re-wiring"):

$V' \leftarrow V$; $E' \leftarrow E$;
$x_{\text{nearest}} \leftarrow \texttt{Nearest}(G, x)$;
$x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x)$;
if $\texttt{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
  $V' \leftarrow V' \cup \{x_{\text{new}}\}$;
  $x_{\text{min}} \leftarrow x_{\text{nearest}}$;
  $X_{\text{near}} \leftarrow \texttt{Near}(G, x_{\text{new}}, |V|)$;
  for *all* $x_{\text{near}} \in X_{\text{near}}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{near}}, x_{\text{new}})$ **then**
      $c' \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$;
      if $c' < \texttt{Cost}(x_{\text{new}})$ **then**
        $x_{\text{min}} \leftarrow x_{\text{near}}$;
  $E' \leftarrow E' \cup \{(x_{\text{min}}, x_{\text{new}})\}$;
  for *all* $x_{near} \in X_{\text{near}} \setminus \{x_{\text{min}}\}$ **do**
    if $\texttt{ObstacleFree}(x_{\text{new}}, x_{\text{near}})$ *and* $\texttt{Cost}(x_{\text{near}}) > \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}}))$ **then**
      $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E' \leftarrow E' \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$;
      $E' \leftarrow E' \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
return $G' = (V', E')$

Re-wiring:
Checking if we can improve (re-wire) the cost of other nodes near the new node x_{new}

borrowed from "Incremental Sampling-based Algorthms for Optimal Motion Planning" paper by S. Karaman & E. Frazzoli

RRT* [Karaman & Frazzoli, '06]

Main loop (... "re-wiring"):

X_{near}: set of all vertices v in V s.t. they lie within radius r from x_{new}, where

$$r = \min\left(\left(\frac{\gamma}{\delta}\frac{\log|V|}{|V|}\right)^{1/d}, \ \mathcal{E}\right),$$

d – dimensionality of space, δ – volume of unit hyperball, γ – user defined constant

```
V ...
...
...
x_r...
(V, E) ← ...
```

```
X_near ← Near(G, x_new, |V|);
for all x_near ∈ X_near do
    if ObstacleFree(x_near, x_new) then
        c' ← Cost(x_near) + c(Line(x_near, x_new));
        if c' < Cost(x_new) then
            x_min ← x_near;
E' ← E' ∪ {(x_min, x_new)};
for all x_near ∈ X_near \ {x_min} do
    if ObstacleFree(x_new, x_near) and
    Cost(x_near) > Cost(x_new) + c(Line(x_new, x_near))
    then
        x_parent ← Parent(x_near);
        E' ← E' \ {(x_parent, x_near)};
        E' ← E' ∪ {(x_new, x_near)};
return G' = (V', E')
```

Re-wiring:

Checking if we can improve (re-wire) the cost of other nodes near the new node x_{new}

borrowed from "Incremental Sampling-based Algorthms for Optimal Motion Planning" paper by S. Karaman & E. Frazzoli

RRT* [Karaman & Frazzoli, '06]

Main loop ... *"re-wiring"):*

V
x_{ra}
$(V, E) \leftarrow$

$|V|)$;

);

X_{near}: set of all vertices v in V s.t. they lie within radius r from x_{new}, where

$$r = \min\left(\left(\frac{\gamma}{\delta}\frac{\log|V|}{|V|}\right)^{1/d}, \ \mathcal{E}\right),$$

d – dimensionality of space, δ – volume of unit hyperball, γ – user defined constant

RRT (unlike RRT) is asymptotically optimal:*
converges to an optimal solution in the limit of the number of samples

Checking if ...
the cost of other nodes ...
the new node x_{new}

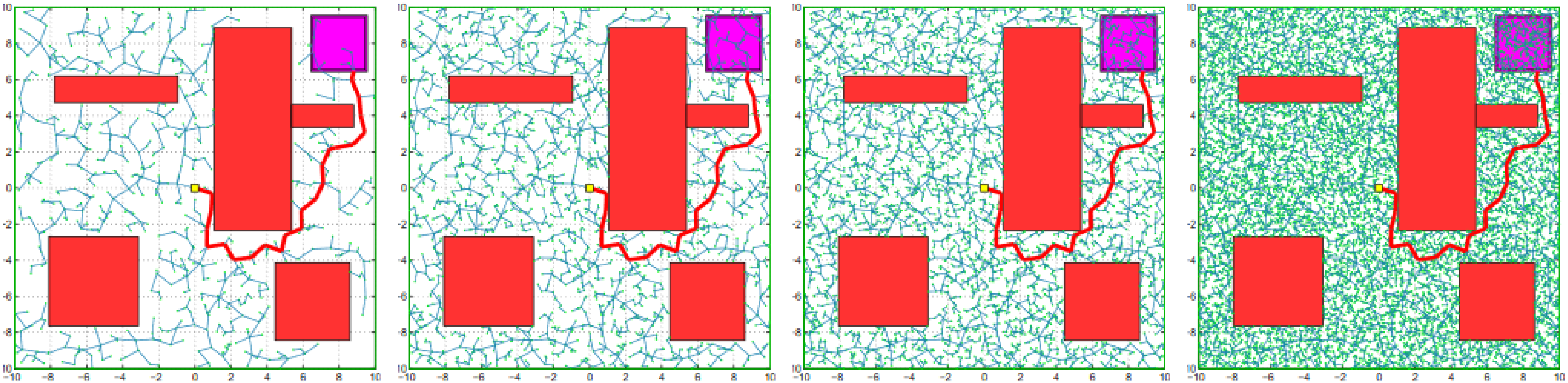
$\setminus \{x_{\min}\}$ **do**
if $\texttt{ObstacleFree}(x_{new}, x_{near})$ *and* $\texttt{Cost}(x_{near}) > \texttt{Cost}(x_{new}) + c(\texttt{Line}(x_{new}, x_{near}))$ **then**
$x_{parent} \leftarrow \texttt{Parent}(x_{near})$;
$E' \leftarrow E' \setminus \{(x_{parent}, x_{near})\}$;
$E' \leftarrow E' \cup \{(x_{new}, x_{near})\}$;

return $G' = (V', E')$

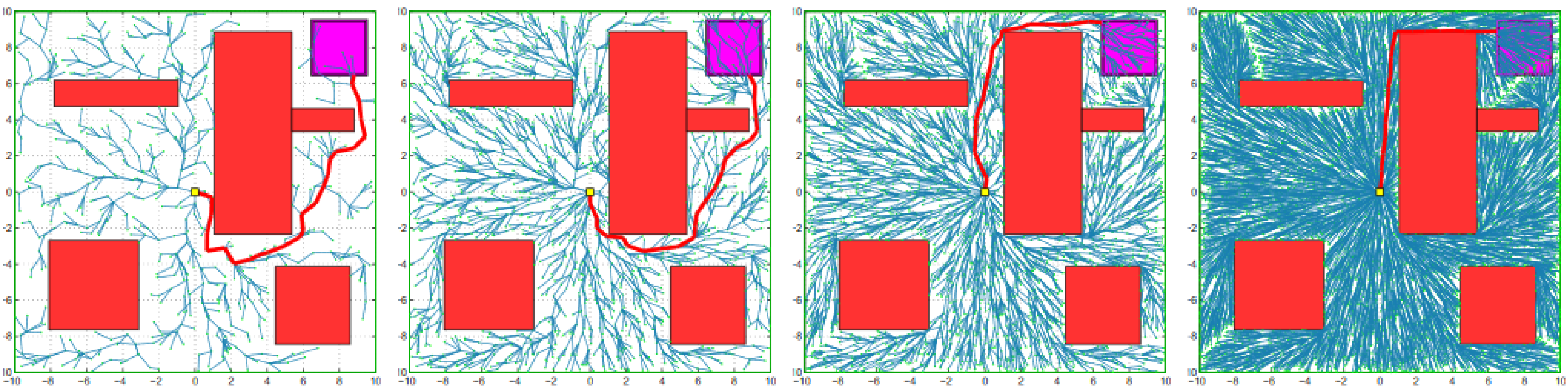
borrowed from "Incremental Sampling-based Algorthms for Optimal Motion Planning" paper by S. Karaman & E. Frazzoli

RRT vs RRT*

The growth of the RRT tree over time & its effect on the solution



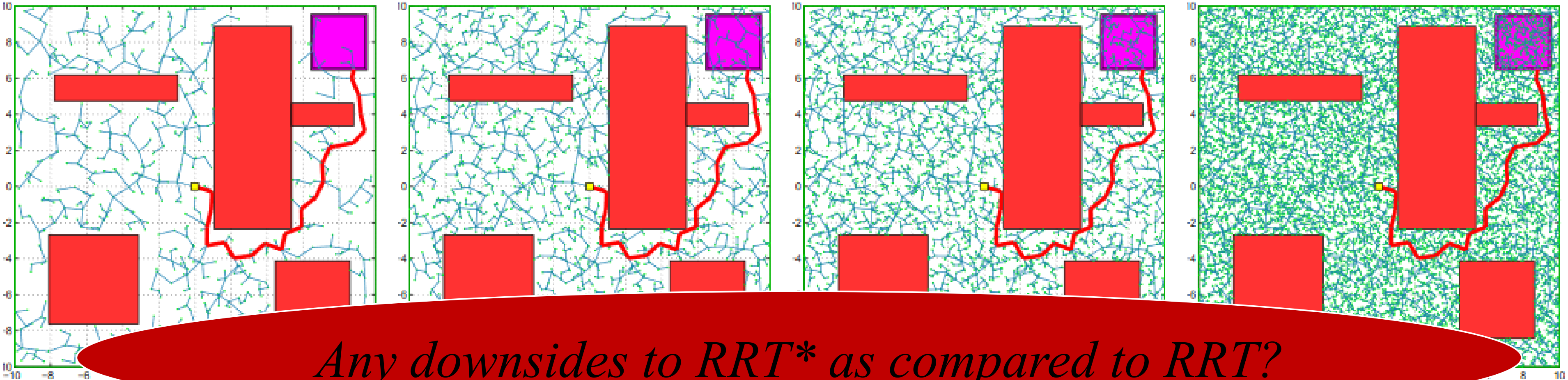
The growth of the RRT tree over time & its effect on the solution*



borrowed from "Incremental Sampling-based Algorthms for Optimal Motion Planning" paper by S. Karaman & E. Frazzoli

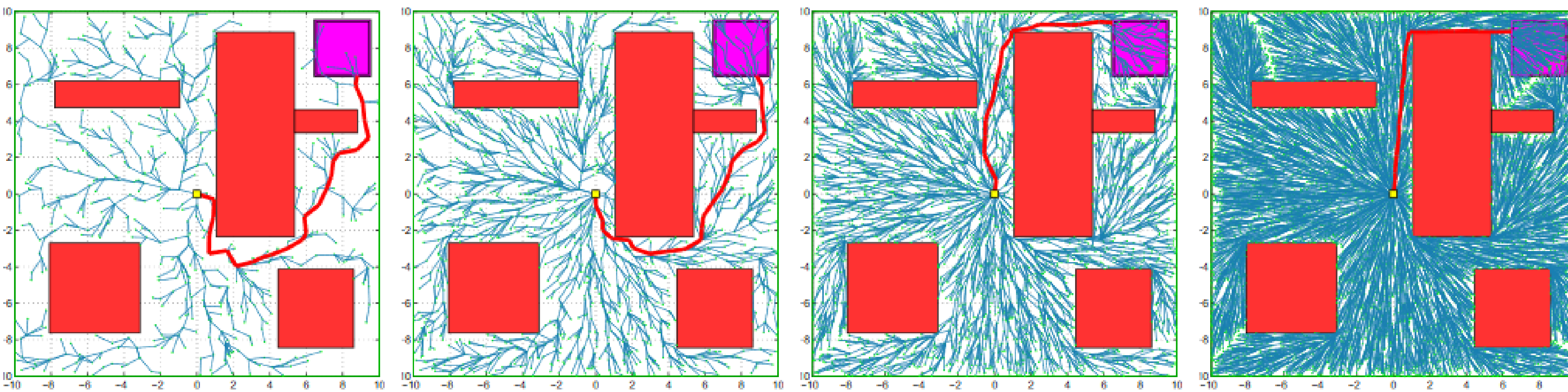
RRT vs RRT*

The growth of the RRT tree over time & its effect on the solution



Any downsides to RRT as compared to RRT?*

The growth of the RRT tree over time & its effect on the solution*



borrowed from "Incremental Sampling-based Algorthms for Optimal Motion Planning" paper by S. Karaman & E. Frazzoli

What You Should Know…

- Pros and Cons of RRT, PRM, RRT-Connect, RRT*
- How RRT, RRT-Connect and RRT* operate
- What guarantees RRT/RRT* provide
- Simple shortcutting algorithm