16.410/413
Principles of Autonomy and Decision Making
Lecture 15: Sampling-Based Algorithms for Motion Planning

Emilio Frazzoli

Aeronautics and Astronautics
Massachusetts Institute of Technology

November 3, 2010

Reading: LaValle, Ch. 5
S. Karaman and E. Frazzoli, 2011

The Motion Planning problem

Get from point A to point B avoiding obstacles

The Motion Planning problem

Consider a dynamical control system defined by an ODE of the form

$$dx/dt = f(x, u), \qquad x(0) = x_{\text{init}}, \tag{1}$$

where x is the state, u is the control.

Given an obstacle set $X_{\text{obs}} \subset \mathbb{R}^d$, and a goal set $X_{\text{goal}} \subset \mathbb{R}^d$, the objective of the motion planning problem is to find, *if it exists*, a control signal u such that the solution of (1) satisfies $x(t) \notin X_{\text{obs}}$ for all $t \in \mathbb{R}_+$, and $x(t) \in X_{\text{goal}}$ for all $t > T$, for some finite $T \geq 0$. Return failure if no such control signal exists.

- Basic problem in robotics (and intelligent life in general).
- Provably very hard: a basic version (the Generalized Piano Mover's problem) is known to be PSPACE-hard [Reif, '79].

Mobility, Brains, and the lack thereof

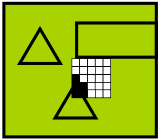
Image of sea squirts removed due to copyright restrictions. Please see: http://en.wikipedia.org/wiki/File:Sea-tulip.jpg.

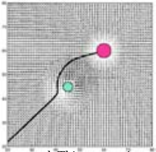
The **Sea Squirt**, or *Tunicate*, is an organism capable of mobility until it finds a suitable rock to cement itself in place. Once it becomes stationary, it digests its own cerebral ganglion, or “eats its own brain” and develops a thick covering, a “tunic” for self defense. [S. Soatto, 2010, R. Bajcsy, 1988]

Motion planning in practice

Many techniques have been proposed to solve such problems in practical applications, e.g.,

- **Algebraic planners**: Explicit representation of obstacles. Use complicated algebra (visibility computations/projections) to find the path. Complete, but impractical.
- **Discretization + graph search**: Analytic/grid-based methods do not scale well to high dimensions. Graph search methods (A^*, D^*, etc.) can be sensitive to graph size. Resolution complete.
- **Potential fields/navigation functions**: Virtual attractive forces towards the goal, repulsive forces away from the obstacles. No completeness guarantees, unless "navigation functions" are available—very hard to compute in general.





These algorithms achieve tractability by foregoing completeness altogether, or achieving weaker forms of it, e.g., resolution completeness.

Sampling-based algorithms

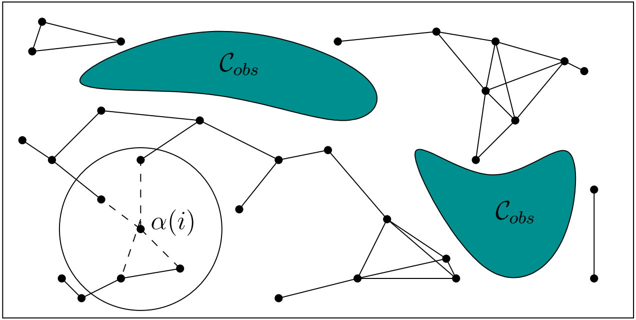
- A recently proposed class of motion planning algorithms that has been very successful in practice is based on (batch or incremental) sampling methods: solutions are computed based on samples drawn from some distribution. *Sampling algorithms retain some form of completeness, e.g., probabilistic or resolution completeness.*
- Incremental sampling methods are particularly attractive:
  - Incremental-sampling algorithms lend themselves easily to real-time, on-line implementation.
  - Applicable to very general dynamical systems.
  - Do not require the explicit enumeration of constraints.
  - Adaptively multi-resolution methods
    (i.e., make your own grid as you go along, up to the necessary resolution).

Probabilistic RoadMaps (PRM)

- Introduced by Kavraki and Latombe in 1994.
- Mainly geared towards "multi-query" motion planning problems.
- Idea: build (offline) a graph (i.e., the **roadmap**) representing the "connectivity" of the environment; use this roadmap to figure out paths quickly at run time.
- Learning/pre-processing phase:
  1. Sample n points from $X_{\mathrm{free}} = [0, 1]^d \setminus X_{\mathrm{obs}}$.
  2. Try to connect these points using a fast "local planner" (e.g., ignore obstacles).
  3. If connection is successful (i.e., no collisions), add an edge between the points.
- At run time:
  1. Connect the start and end goal to the closest nodes in the roadmap.
  2. Find a path on the roadmap.

First planner ever to demonstrate the ability to solve general planning problems in $>$ 4-5 dimensions!

Probabilistic RoadMap example



"Practical" algorithm:

- Incremental construction
- Connect points within a radius r, starting from "closest" ones.
- Do not attempt to connect points that are already on the same connected component of the PRM.

What kind of properties does this algorithm have? Will it find a solution if there is one? Is this an optimal solution? What is the complexity of the algorithm?

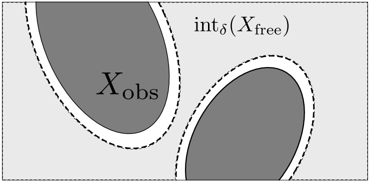
Probabilistic Completeness

Definition (Probabilistic completeness)

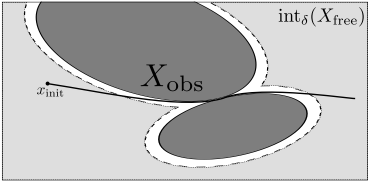
An algorithm ALG is probabilistically complete if, for any robustly feasible motion planning problem defined by $\mathcal{P} = (X_{\mathrm{free}}, x_{\mathrm{init}}, X_{\mathrm{goal}})$,

$$\lim_{N \to \infty} \Pr\left(ALG \text{ returns a solution to } \mathcal{P}\right) = 1.$$

- A "relaxed" notion of completeness
- Applicable to motion planning problems with a **robust solution**. A robust solution remains a solution if obstacles are "dilated" by some small δ.



Robust



NOT Robust

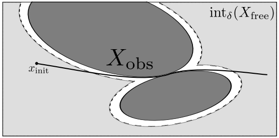
Asymptotic Optimality

Definition (Asymptotic optimality)

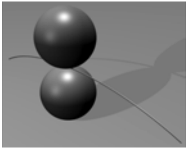
An algorithm ALG is asymptotically optimal if, for any motion planning problem $\mathcal{P} = (X_{\mathrm{free}}, x_{\mathrm{init}}, X_{\mathrm{goal}})$ and cost function c that admit a robust optimal solution with finite cost c^*,

$$\mathbb{P}\left(\left\{\lim_{i\to\infty} Y_i^{ALG} = c^*\right\}\right) = 1.$$

- The function c associates to each path σ a non-negative cost $c(\sigma)$, e.g., $c(\sigma) = \int_\sigma \chi(s)\, ds$.
- The definition is applicable to optimal motion planning problem with a **robust optimal solution**. A robust optimal solution is such that it can be obtained as a limit of robust (non-optimal) solutions.



Not robust



Robust

Complexity

- How can we measure complexity for an algorithm that does not necessarily terminate?
- Treat the number of samples as "the size of the input." (Everything else stays the same)
- Also, complexity per sample: how much work (time/memory) is needed to process one sample.
- Useful for comparison of sampling-based algorithms.
- Cannot compare with deterministic, complete algorithms.

Simple PRM (sPRM)

sPRM Algorithm

$V \leftarrow \{x_{\text{init}}\} \cup \{\texttt{SampleFree}_i\}_{i=1,\ldots,N-1}$; $E \leftarrow \emptyset$;
foreach $v \in V$ **do**
  $U \leftarrow \texttt{Near}(G = (V, E), v, r) \setminus \{v\}$;
  foreach $u \in U$ **do**
    if $\texttt{CollisionFree}(v, u)$ **then** $E \leftarrow E \cup \{(v, u), (u, v)\}$
return $G = (V, E)$;

- The simplified version of the PRM algorithm has been shown to be probabilistically complete. (No proofs available for the "real" PRM!)
- Moreover, the probability of success goes to 1 exponentially fast, if the environment satisfies certain "good visibility" conditions.
- New key concept: combinatorial complexity vs. "visibility"

Remarks on PRM

- sPRM is probabilistically complete and asymptotically optimal.
- PRM is probabilistically complete but NOT asymptotically optimal.
- Complexity for N samples: $\Theta(N^2)$.
- Practical complexity-reduction tricks:
  - ***k*-nearest neighbors**: connect to the k nearest neighbors. Complexity $\Theta(N \log N)$. *(Finding nearest neighbors takes* $\log N$ *time.)*
  - **Bounded degree**: connect at most k neighbors among those within radius r.
  - **Variable radius**: change the connection radius r as a function of N. How?

Rapidly-exploring Random Trees

- Introduced by LaValle and Kuffner in 1998.
- Appropriate for single-query planning problems.
- Idea: build (online) a tree, exploring the region of the state space that can be reached from the initial condition.
- At each step: sample one point from X_{free}, and try to connect it to the closest vertex in the tree.
- Very effective in practice, "Voronoi bias"

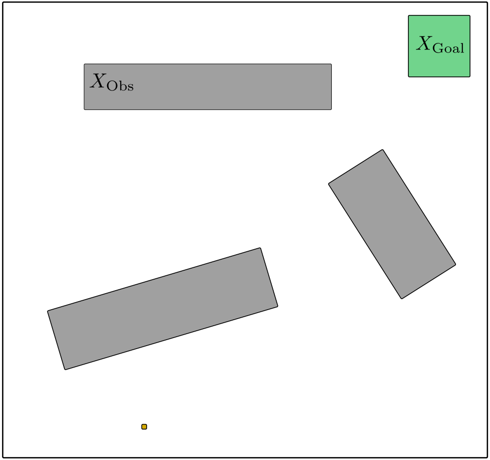
Rapidly-exploring Random Trees

RRT

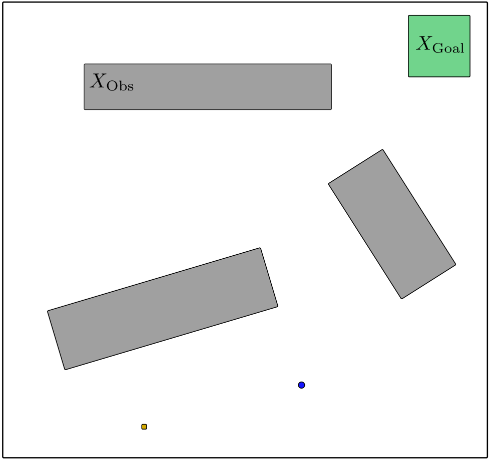
$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $V \leftarrow V \cup \{x_{\text{new}}\}$; $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$;
return $G = (V, E)$;

- The RRT algorithm is probabilistically complete.
- The probability of success goes to 1 exponentially fast, if the environment satisfies certain “good visibility” conditions.

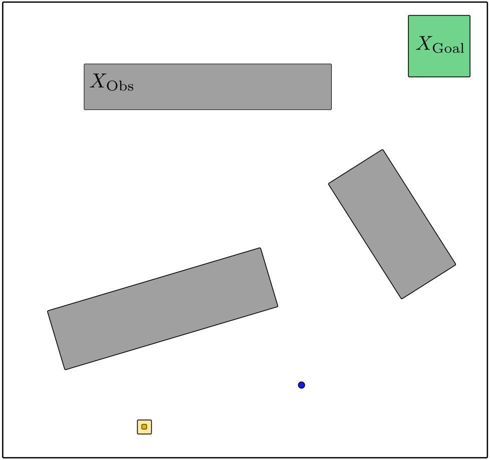
Rapidly-exploring Random Trees (RRTs)



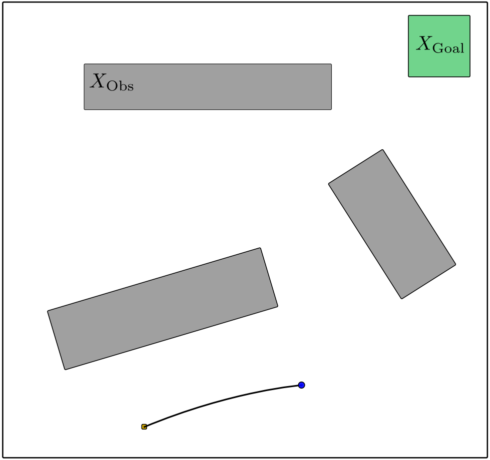
Rapidly-exploring Random Trees (RRTs)



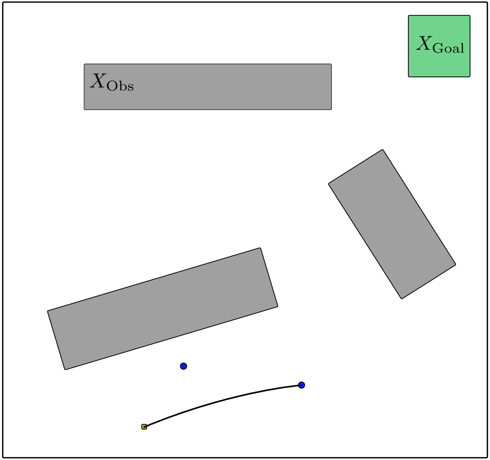
Rapidly-exploring Random Trees (RRTs)



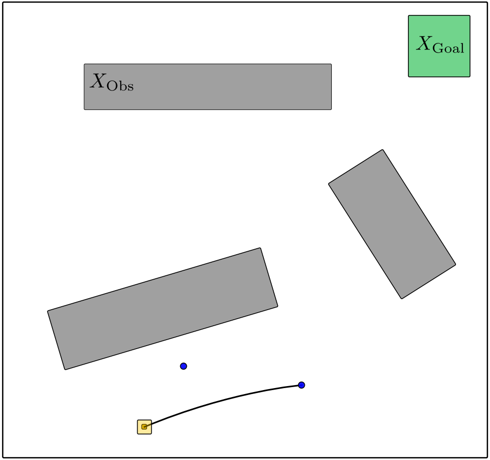
Rapidly-exploring Random Trees (RRTs)



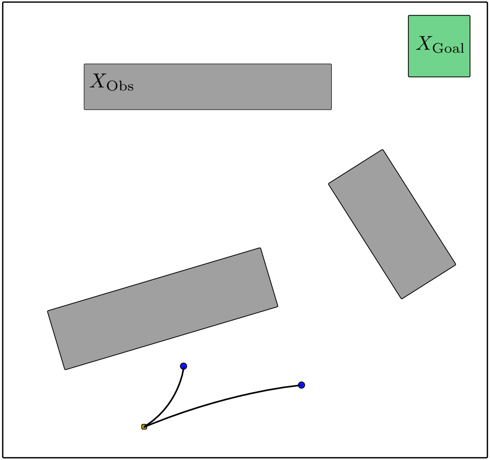
Rapidly-exploring Random Trees (RRTs)



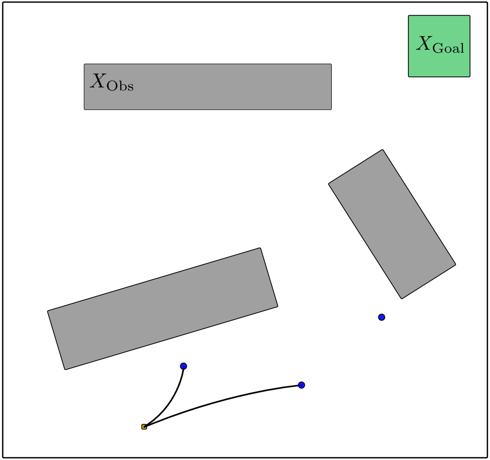
Rapidly-exploring Random Trees (RRTs)



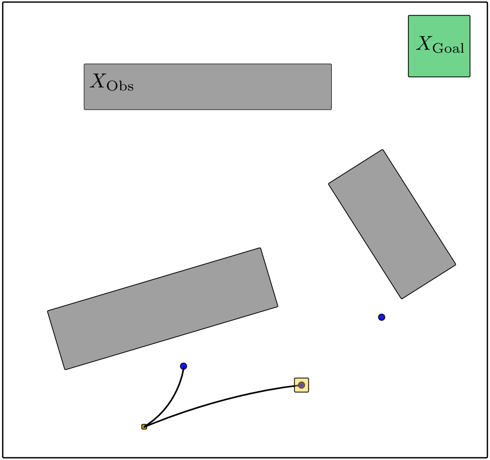
Rapidly-exploring Random Trees (RRTs)



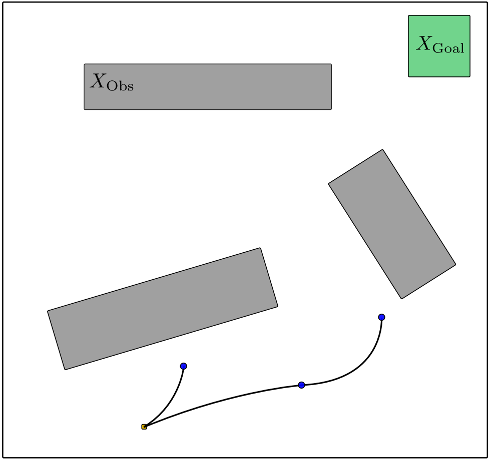
Rapidly-exploring Random Trees (RRTs)



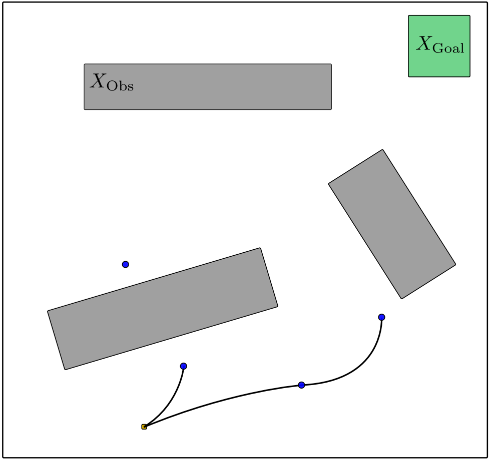
Rapidly-exploring Random Trees (RRTs)



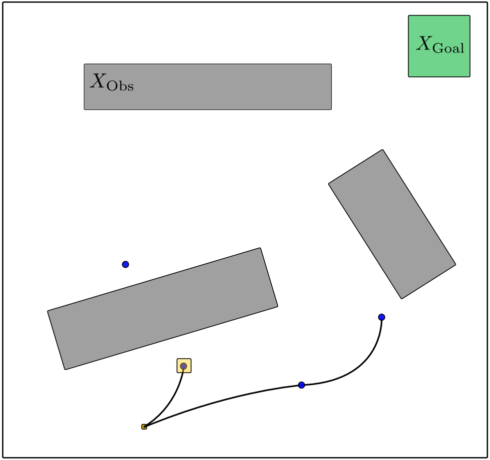
Rapidly-exploring Random Trees (RRTs)



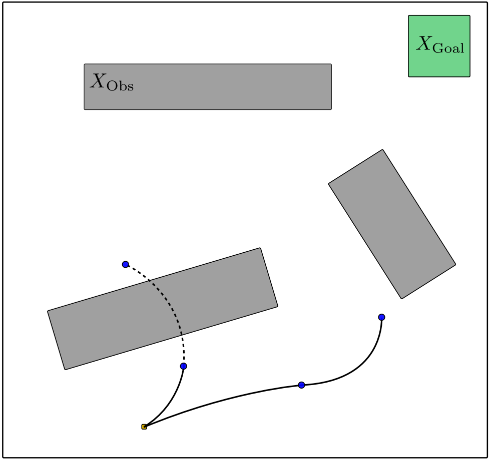
Rapidly-exploring Random Trees (RRTs)



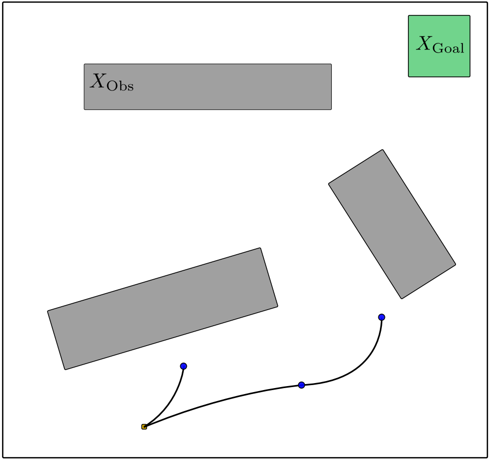
Rapidly-exploring Random Trees (RRTs)



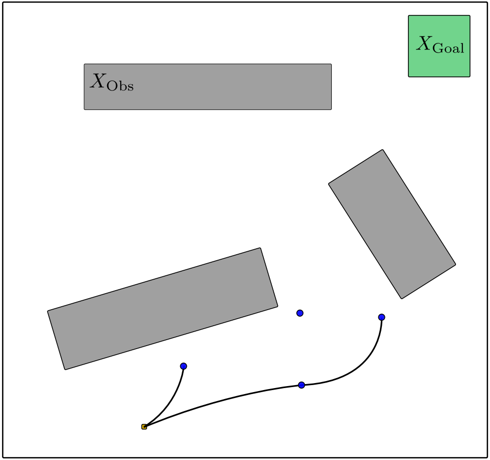
Rapidly-exploring Random Trees (RRTs)



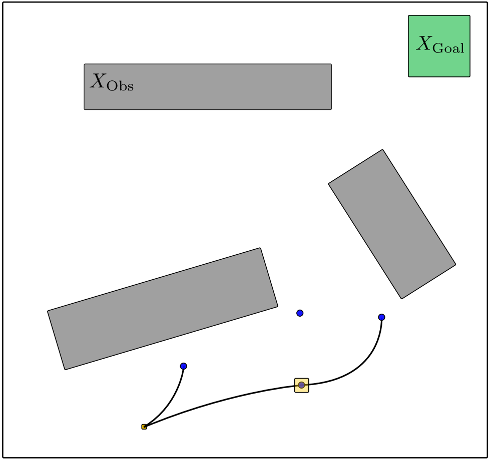
Rapidly-exploring Random Trees (RRTs)



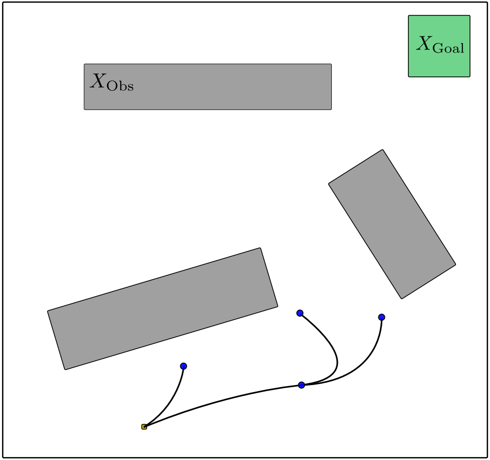
Rapidly-exploring Random Trees (RRTs)



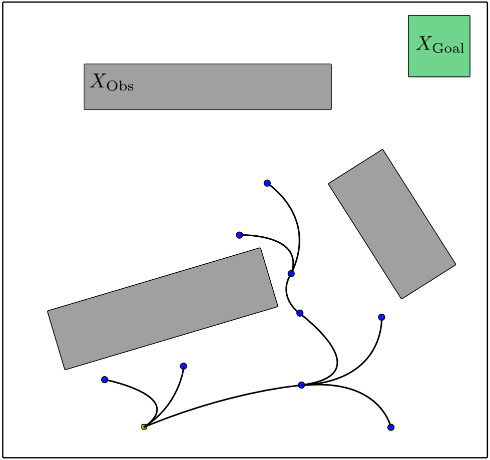
Rapidly-exploring Random Trees (RRTs)



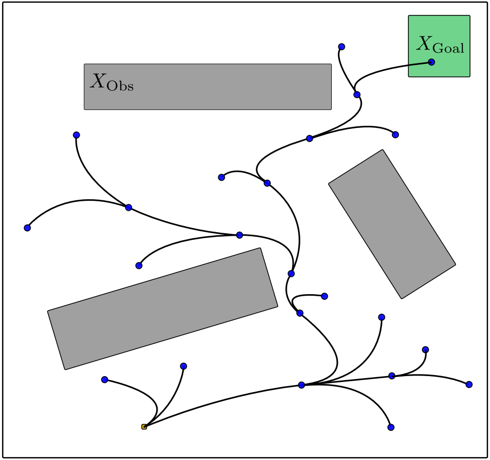
Rapidly-exploring Random Trees (RRTs)



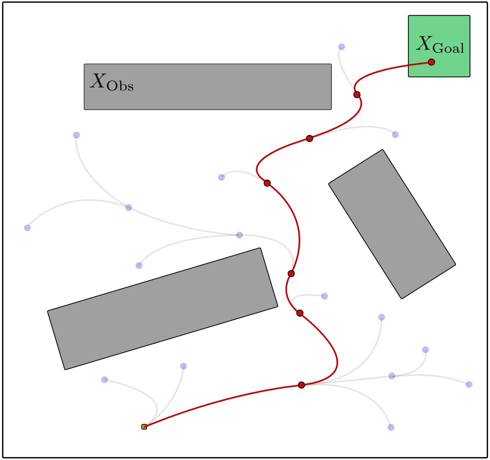
Rapidly-exploring Random Trees (RRTs)



Rapidly-exploring Random Trees (RRTs)



Rapidly-exploring Random Trees (RRTs)



Voronoi bias

Definition (Voronoi diagram)

Given n sites in d dimensions, the Voronoi diagram of the sites is a partition of $\mathbb{R}^d$ into regions, one region per site, such that all points in the interior of each region lie closer to that regions site than to any other site.

- Vertices of the RRT that are more "isolated" (e.g., in unexplored areas, or at the boundary of the explored area) have larger Voronoi regions—and are more likely to be selected for extension.

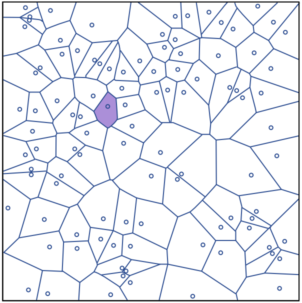
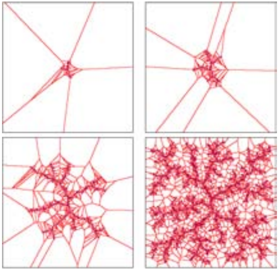


Image by MIT OpenCourseWare.



RRTs in action

- Talos, the MIT entry to the 2007 DARPA Urban Challenge, relied on an "RRT-like" algorithm for real-time motion planning and control.
- The devil is in the details: provisions needed for, e.g.,
  - Real-time, on-line planning for a safety-critical vehicle with substantial momentum.
  - Uncertain, dynamic environment with limited/faulty sensors.
- Main innovations [Kuwata, *et al.* '09]
  - **Closed-loop planning**: plan reference trajectories for an closed-loop model of the vehicle under a stabilizing feedback.
  - **Safety invariance**: Always maintain the ability to stop safely *within the sensing region*.
  - **Lazy evaluation**: the actual trajectory may deviate from the planned one, need to efficiently re-check the tree for feasibility.

The RRT-based P+C system performed flawlessly throughout the race.

Limitations of current incremental sampling methods

The MIT DARPA Urban Challenge code, as well as other incremental sampling methods, suffer from the following limitations:

- No characterization of the quality (e.g., "cost") of the trajectories returned by the algorithm.

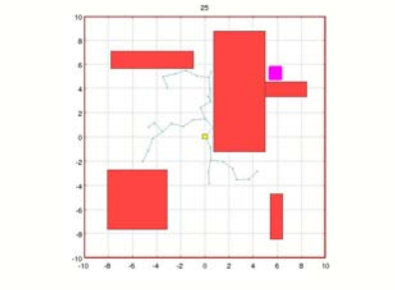
  Keep running the RRT even after the first solution has been obtained, for as long as possible (given the real-time constraints), hoping to find a better path than that already available.

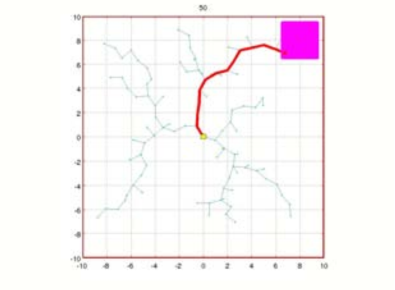
- No systematic method for imposing temporal/logical constraints, such as, e.g., the rules of the road, complicated mission objectives, ethical/deontic code.

  In the DARPA Urban Challenge, all logics for, e.g., intersection handling, had to be hand-coded, at a huge cost in terms of debugging effort/reliability of the code.

RRTs and optimality

- RRTs are great at finding feasible trajectories quickly...
- However, RRTs are apparently terrible at finding good trajectories.





- What is the reason for such behavior?

A negative result [K&F RSS'10]

- Let Y_n^{RRT} be the cost of the best path in the RRT at the end of iteration n.
- It is easy to show that Y_n^{RRT} converges (to a random variable), i.e.,

$$\lim_{n \to \infty} Y_n^{\mathrm{RRT}} = Y_\infty^{\mathrm{RRT}}.$$

- The random variable Y_∞^{RRT} is sampled from a distribution with zero mass at the optimum:

Theorem (Almost sure suboptimality of RRTs)

If the set of sampled optimal paths has measure zero, the sampling distribution is absolutely continuous with positive density in X_{free}, and $d \geq 2$, then the best path in the RRT converges to a sub-optimal solution almost surely, i.e.,

$$\Pr\left[Y_\infty^{\mathrm{RRT}} > c^*\right] = 1.$$

Some remarks on the negative result

- Intuition: RRT does not satisfy a necessary condition for asymptotic optimality, i.e., that the root node has infinitely many subtrees that extend at least a distance ϵ away from x_{init}.
- The RRT algorithm "traps" itself by disallowing new better paths to emerge.
- Heuristics such as
  - running the RRT multiple times [Ferguson & Stentz, '06]
  - running multiple trees concurrently,
  - deleting and rebuilding parts of the tree etc.

  work better than the standard RRT, but also result in almost-sure sub-optimality.
- **A careful rethinking of the RRT algorithm seems to be required to ensure (asymptotic) optimality.**

Rapidly-exploring Random Graphs (RRGs)

A new incremental sampling algorithm:

RRG algorithm

$V \leftarrow \{x_{\rm init}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $x_{\rm rand} \leftarrow \texttt{SampleFree}_i$;
  $x_{\rm nearest} \leftarrow \texttt{Nearest}(G = (V, E), x_{\rm rand})$;
  $x_{\rm new} \leftarrow \texttt{Steer}(x_{\rm nearest}, x_{\rm rand})$;
  if $\texttt{ObtacleFree}(x_{\rm nearest}, x_{\rm new})$ **then**
    $X_{\rm near} \leftarrow \texttt{Near}(G = (V, E), x_{\rm new}, \min\{\gamma_{\rm RRG}(\log(\text{card } V)/\text{ card } V)^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\rm new}\}$; $E \leftarrow E \cup \{(x_{\rm nearest}, x_{\rm new}), (x_{\rm new}, x_{\rm nearest})\}$;
    foreach $x_{\rm near} \in X_{\rm near}$ **do**
      if $\texttt{CollisionFree}(x_{\rm near}, x_{\rm new})$ **then** $E \leftarrow E \cup \{(x_{\rm near}, x_{\rm new}), (x_{\rm new}, x_{\rm near})\}$

return $G = (V, E)$;

- At each iteration, the RRG tries to connect to the new sample all vertices in a ball of radius r_n centered at it. (Or just default to the nearest one if such ball is empty.)
- in general the RRG builds graphs with cycles.

Properties of RRGs

Theorem (Probabilistic completeness)

Since $V_n^{\mathrm{RRG}} = V_n^{\mathrm{RRT}}$, for all n, it follows that RRG has the same completeness properties as RRT, i.e.,

$$\Pr\left[V_n^{\mathrm{RRG}} \cap X_{\mathrm{goal}} = \emptyset\right] = O(e^{-bn}).$$

Theorem (Asymptotic Optimality)

If the `Near` procedure returns all nodes in V within a ball of volume

$$Vol = \gamma \frac{\log n}{n}, \qquad \gamma > 2^d(1 + 1/d),$$

under some additional technical assumptions (e.g., on the sampling distribution, on the ϵ clearance of the optimal path, and on the continuity of the cost function), the best path in the RRG converges to an optimal solution almost surely, i.e.,

$$\Pr\left[Y_\infty^{\mathrm{RRG}} = c^*\right] = 1.$$

Computational Complexity [K&F RSS '10]

- At each iteration, the RRG algorithm executes $O(\log n)$ extra calls to `ObstacleFree` when compared to the RRT.
- However, the complexity of the `Nearest` procedure is $\Omega(\log n)$. *Achieved if using, e.g., a Balanced-Box Decomposition (BBD) Tree.*

Theorem: Asymptotic (Relative) Complexity

There exists a constant $\beta \in \mathbb{R}_+$ such that

$$\limsup_{i \to \infty} \mathbb{E}\left[\frac{OPS_i^{\mathrm{RRG}}}{OPS_i^{\mathrm{RRT}}}\right] \leq \beta$$

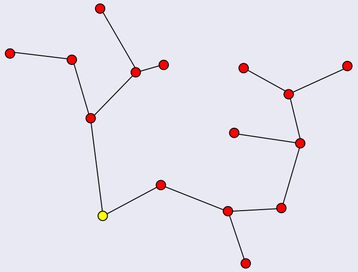
- In other words, the RRG algorithm has no substantial computational overhead over RRT, and ensures asymptotic optimality.

RRT*: A tree version of the RRG [K&F RSS '10]

- RRT algorithm can account for, e.g., non-holonomic dynamics, and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the `Steer` procedure to be **exact**. Exact steering methods are not available for general dynamical systems.

RRT* algorithm

- RRT* is a variant of RRG that essentially "rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRG and RRT.

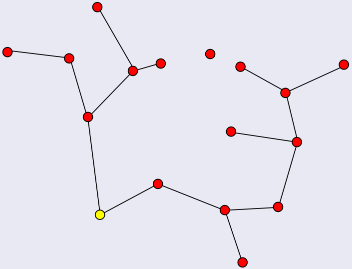


RRT*: A tree version of the RRG [K&F RSS '10]

- RRT algorithm can account for, e.g., non-holonomic dynamics, and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the `Steer` procedure to be **exact**. Exact steering methods are not available for general dynamical systems.

RRT* algorithm

- RRT* is a variant of RRG that essentially "rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRG and RRT.

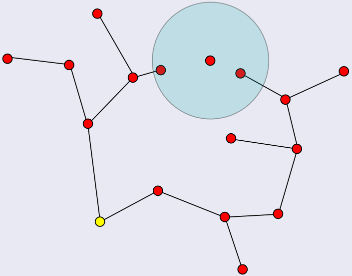


RRT*: A tree version of the RRG [K&F RSS '10]

- RRT algorithm can account for, e.g., non-holonomic dynamics, and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the `Steer` procedure to be **exact**. Exact steering methods are not available for general dynamical systems.

RRT* algorithm

- RRT* is a variant of RRG that essentially "rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRG and RRT.

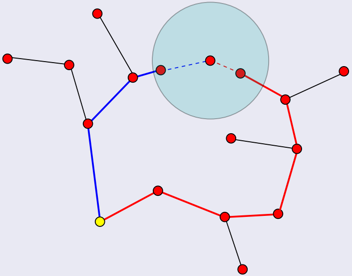


RRT*: A tree version of the RRG [K&F RSS '10]

- RRT algorithm can account for, e.g., non-holonomic dynamics, and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the `Steer` procedure to be **exact**. Exact steering methods are not available for general dynamical systems.

RRT* algorithm

- RRT* is a variant of RRG that essentially "rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRG and RRT.

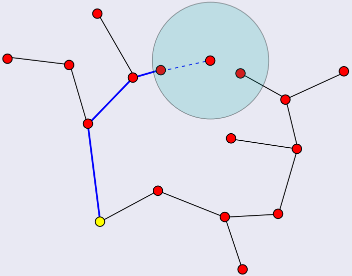


RRT*: A tree version of the RRG [K&F RSS '10]

- RRT algorithm can account for, e.g., non-holonomic dynamics, and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the `Steer` procedure to be **exact**. Exact steering methods are not available for general dynamical systems.

RRT* algorithm

- RRT* is a variant of RRG that essentially "rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRG and RRT.

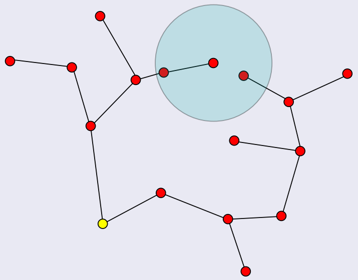


RRT*: A tree version of the RRG [K&F RSS '10]

- RRT algorithm can account for, e.g., non-holonomic dynamics, and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the `Steer` procedure to be **exact**. Exact steering methods are not available for general dynamical systems.

RRT* algorithm

- RRT* is a variant of RRG that essentially "rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRG and RRT.

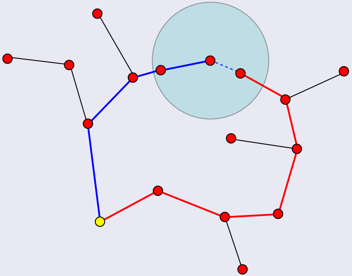


RRT*: A tree version of the RRG [K&F RSS '10]

- RRT algorithm can account for, e.g., non-holonomic dynamics, and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the `Steer` procedure to be **exact**. Exact steering methods are not available for general dynamical systems.

RRT* algorithm

- RRT* is a variant of RRG that essentially "rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRG and RRT.

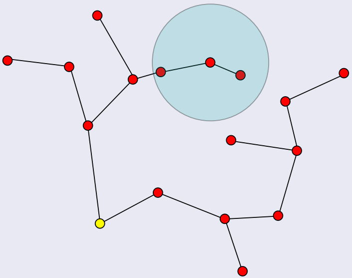


RRT*: A tree version of the RRG [K&F RSS '10]

- RRT algorithm can account for, e.g., non-holonomic dynamics, and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the `Steer` procedure to be **exact**. Exact steering methods are not available for general dynamical systems.

RRT* algorithm

- RRT* is a variant of RRG that essentially "rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRG and RRT.

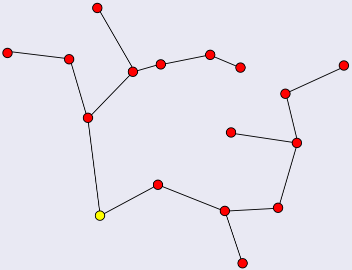


RRT*: A tree version of the RRG [K&F RSS '10]

- RRT algorithm can account for, e.g., non-holonomic dynamics, and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the `Steer` procedure to be **exact**. Exact steering methods are not available for general dynamical systems.

RRT* algorithm

- RRT* is a variant of RRG that essentially "rewires" the tree as better paths are discovered.
- After rewiring the cost has to be propagated along the leaves.
- If steering errors occur, subtrees can be re-computed.
- The RRT* algorithm inherits the asymptotic optimality and rapid exploration properties of the RRG and RRT.



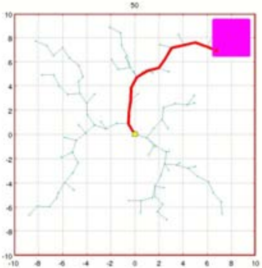
RRT* algorithm

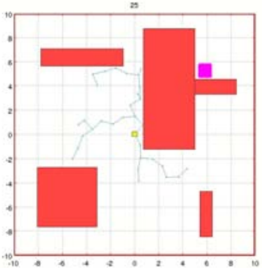
RRT*

$V \leftarrow \{x_{\rm init}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, N$ **do**
  $x_{\rm rand} \leftarrow \texttt{SampleFree}_i$;
  $x_{\rm nearest} \leftarrow \texttt{Nearest}(G = (V, E), x_{\rm rand})$;
  $x_{\rm new} \leftarrow \texttt{Steer}(x_{\rm nearest}, x_{\rm rand})$;
  if $\texttt{ObtacleFree}(x_{\rm nearest}, x_{\rm new})$ **then**
    $X_{\rm near} \leftarrow \texttt{Near}(G = (V, E), x_{\rm new}, \min\{\gamma_{RRG}(\log(\text{card } V)/ \text{card } V)^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\rm new}\}$;
    $x_{\rm min} \leftarrow x_{\rm nearest}$; $c_{\rm min} \leftarrow \texttt{Cost}(x_{\rm nearest}) + c(\texttt{Line}(x_{\rm nearest}, x_{\rm new}))$;
    foreach $x_{\rm near} \in X_{\rm near}$ **do** // Connect along a minimum-cost path
      if $\texttt{CollisionFree}(x_{\rm near}, x_{\rm new}) \wedge \texttt{Cost}(x_{\rm near}) + c(\texttt{Line}(x_{\rm near}, x_{\rm new})) < c_{\rm min}$ **then**
        $x_{\rm min} \leftarrow x_{\rm near}$; $c_{\rm min} \leftarrow \texttt{Cost}(x_{\rm near}) + c(\texttt{Line}(x_{\rm near}, x_{\rm new}))$
    $E \leftarrow E \cup \{(x_{\rm min}, x_{\rm new})\}$;
    foreach $x_{\rm near} \in X_{\rm near}$ **do** // Rewire the tree
      if $\texttt{CollisionFree}(x_{\rm new}, x_{\rm near}) \wedge \texttt{Cost}(x_{\rm new}) + c(\texttt{Line}(x_{\rm new}, x_{\rm near})) < \texttt{Cost}(x_{\rm near})$ **then** $x_{\rm parent} \leftarrow \texttt{Parent}(x_{\rm near})$;
      $E \leftarrow (E \setminus \{(x_{\rm parent}, x_{\rm near})\}) \cup \{(x_{\rm new}, x_{\rm near})\}$

return $G = (V, E)$;

RRT* experiment results

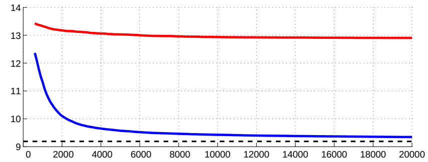




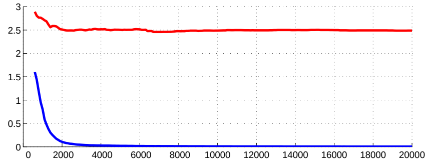
RRT* and RRT in simulations

Monte-Carlo simulation – 500 trials

Cost of the best path



Variance of the solution



- RRT is shown in RED, RRT* is shown in BLUE.

Relative computational cost

./figures/sim0time.pdf

Summary

- Key idea in RRG/RRT*: to combine optimality and computational efficiency, it is necessary to attempt connection to $\Theta(\log N)$ nodes at each iteration.
  - Reduce volume of the "connection ball" as $\log(N)/N$;
  - Increase the number of connections as $\log(N)$.

These principles can be used to obtain "optimal" versions of PRM, etc.:

Algorithm	Probabilistic Completeness	Asymptotic Optimality	Computational Complexity
sPRM	Yes	Yes	$O(N)$
k-nearest sPRM	No	No	$O(\log N)$
RRT	Yes	No	$O(\log N)$
PRM*	Yes	Yes	$O(\log N)$
k-nearest PRM*	Yes	Yes	$O(\log N)$
RRG	Yes	Yes	$O(\log N)$
k-nearest RRG	Yes	Yes	$O(\log N)$
RRT*	Yes	Yes	$O(\log N)$
k-nearest RRT*	Yes	Yes	$O(\log N)$

Conclusion

- Thorough and rigorous analysis of the optimality properties of incremental sampling-based motion planning algorithms.
- We show that state-of-the-art algorithms such as RRT converge to a NON-optimal solution almost-surely.
- We provide new algorithms (RRG and the RRT*), which almost-surely converge to optimal solutions while incurring no significant cost overhead wrt state-of-the-art-algorithms.
- Bibliographical reference: S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. Int. Journal of Robotics Research, 2011. To appear. Also available at http://arxiv.org/abs/1105.1186.
- Current Work:
  - Optimal motion planning with temporal/logic constraints (e.g., μ-calculus).
  - Anytime solution of PDEs (Eikonal equation, Hamilton-Jacobi-Bellman, etc.)
  - Anytime solution of differential games
  - Stochastic optimal motion planning (process + sensor noise)
  - Multi-agent problems.

16.410 / 16.413 Principles of Autonomy and Decision Making

Fall 2010