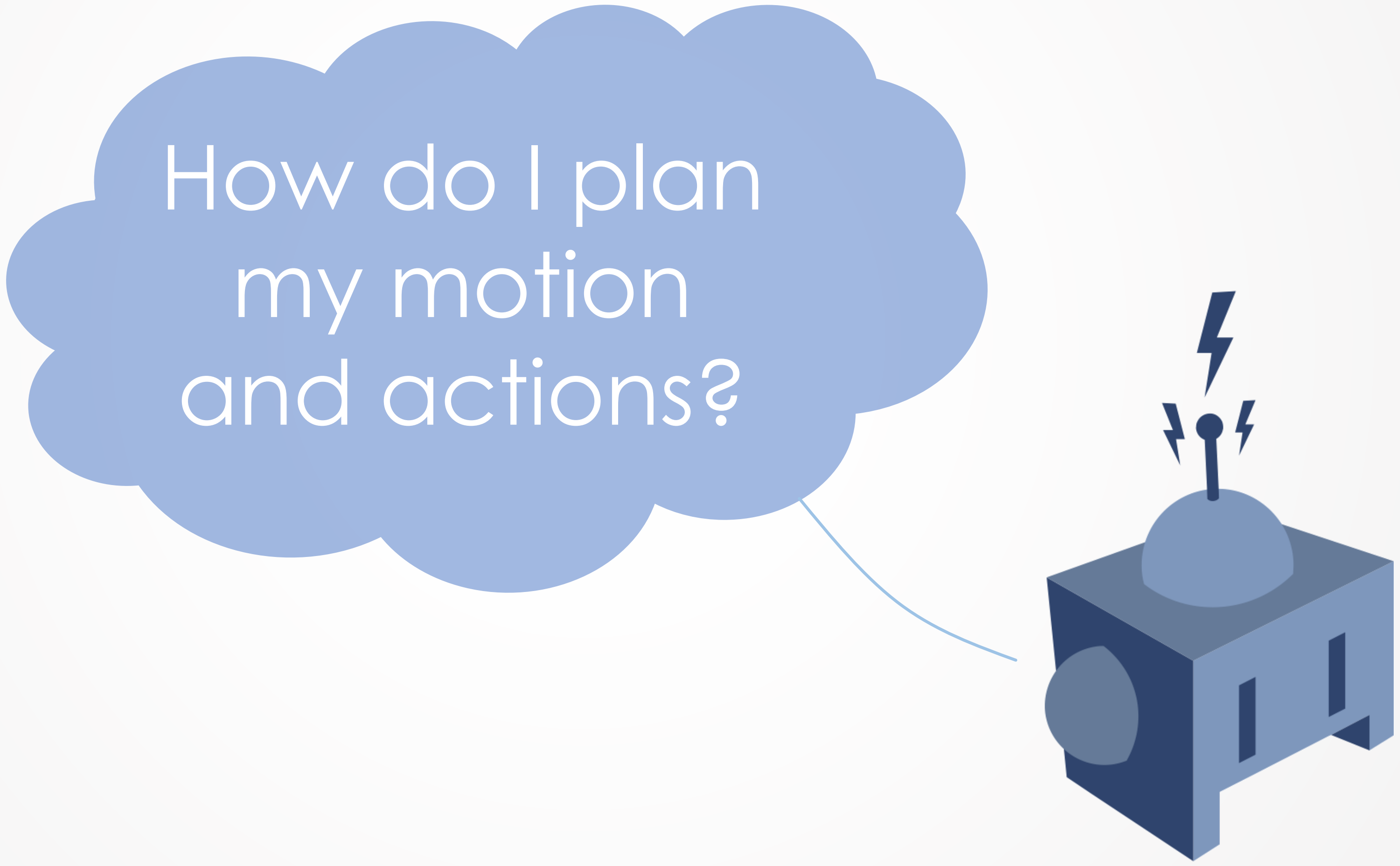
Drones Demystified!

K. Alexis, C. Papachristos, Autonomous Robots Lab, University of Nevada, Reno

A. Tzes, Autonomous Robots & Intelligent Systems Lab, NYU Abu Dhabi

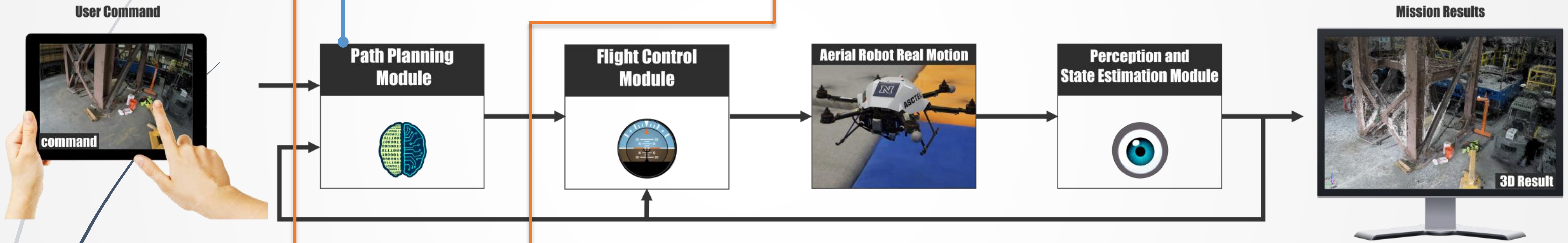
Drones Demystified!

Topic: Introduction to Path Planning


How do I plan my motion and actions?

The Aerial Robot Loop

- Path planning in order to compute the path the robot should follow to ensure safe navigation and execute the desired mission.



Section 4 of our course

The motion planning problem

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

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

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

- Basic problem in robotics
- Provably hard: a basic version of it (the Generalized Piano Mover's problem) is known to be PSPACE-hard.

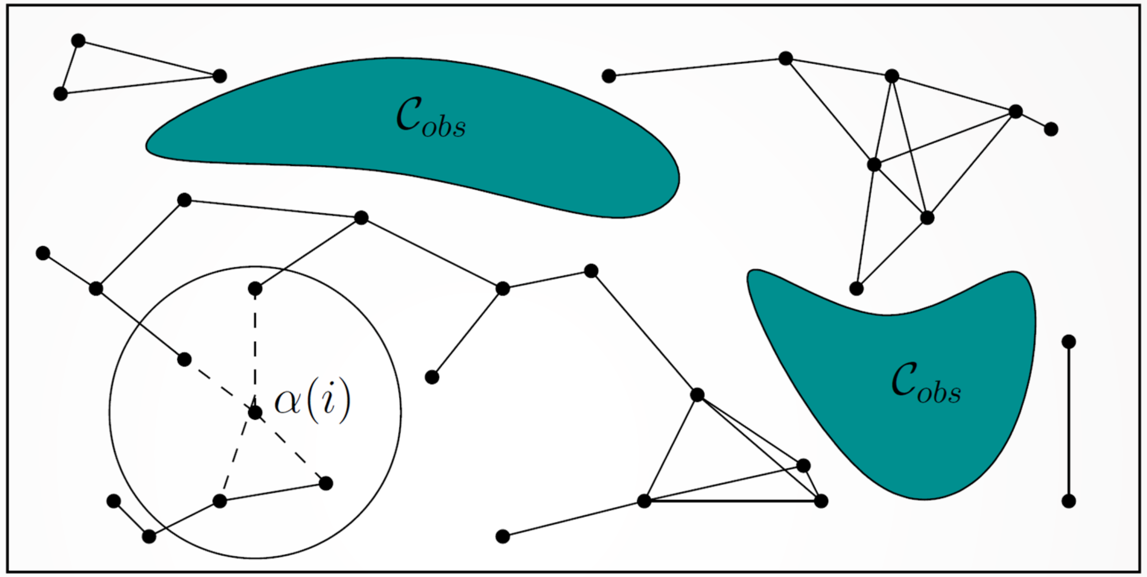
Sampling-based algorithms

- 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 generic 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 find paths quickly at run-time.
- **Learning/pre-processing phase:**
  - Sample n points from $X_{free} = [0,1]^d \backslash X_{obs}$.
  - Try to connect these points using a fast "local planner" (e.g. ignore obstacles).
  - If connection successful (i.e. no collisions), add an edge between the points.
- **At run-time:**
  - Connect the start and end goal to the closest nodes in the roadmap.
  - Find a path on the roadmap.
- *First planner ever to demonstrate the ability to solve generic planning problems in > 4-5 dimensions!*

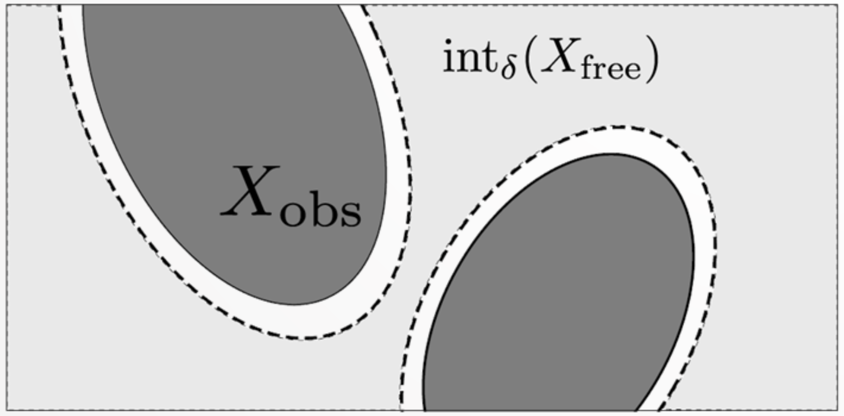
Probabilistic RoadMap example



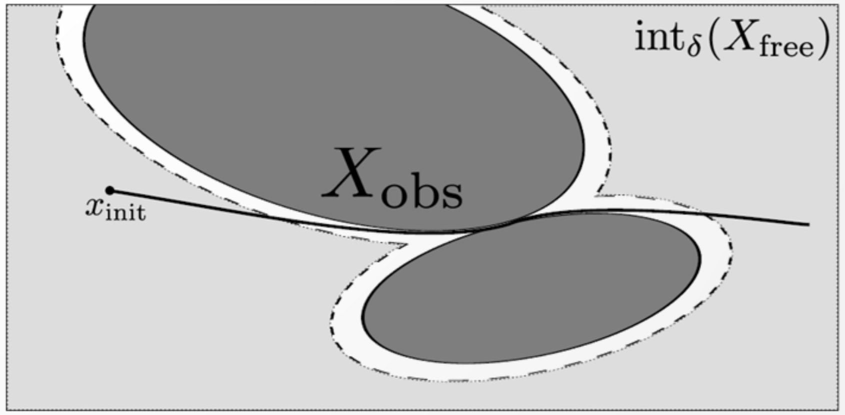
- **"Practical" algorithm:**
  - Incremental construction.
  - Connects points within a radius r, starting from "closest" ones.
  - Do not attempt to connect points that are already on the same connected component of the RPM.
- *What kind of properties does this algorithm have? Will it find a solution if there is one? Will that be 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 $P = (X_{free}, x_{init}, X_{goal})$, then:
$$\lim_{N\to\infty} \Pr(ALG\ returns\ a\ solution\ P) = 1$$
- A "relaxed" notion of completeness
- Applicable to motion planning problems with a robust solution. A robust solutions remains a valid solution even when the obstacles are "dilated" by small small δ.



robust



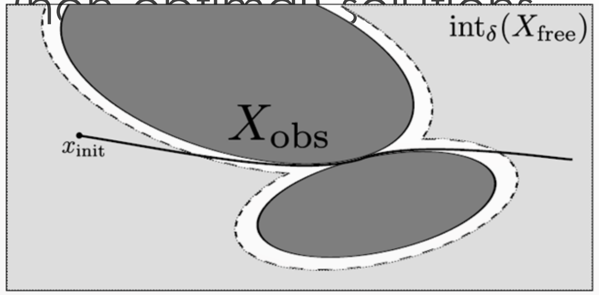
NOT robust

Asymptotic Optimality

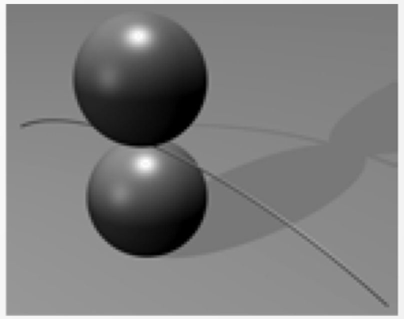
- Definition – Asymptotic Optimality:
- An algorithm ALG is asymptotically optimal if, for any motion planning problem defined by $P = (X_{free}, x_{init}, X_{goal})$ and function c that admit a robust optimal solution with finite cost c^*,

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

- The function c associates to each path σ a non-negative $c(\sigma)$, e.g. $c(\sigma) = \int_\sigma \mathrm{X}(s)ds$
- 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

Simple PRM (sPRM)

sPRM Algorithm

```
$V \leftarrow \{x_{init}\} \cup \{SampleFree_i\}_{i=1,,\ldots,N-1}; E \leftarrow 0;$
**foreach** $v \in V$ **do**:
    $U \leftarrow Near(G = (V,E), v, r)\backslash\{v\};$
    **foreach** $u \in U$ **do**:
        **if** $CollisionFree(v,u)$ **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.
- Moreover, the probability of success goes to 1 exponentially fast, if the environment satisfies "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: $O(N^2)$.
- Practical complexity-reduction tricks:
  - k-nearest neighbors: connect to the k nearest neighbors. Complexity $O(NlogN)$. (Finding nearest neighbors takes $logN$ time.)
  - Bounded degree: connect at most k nearest 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”.

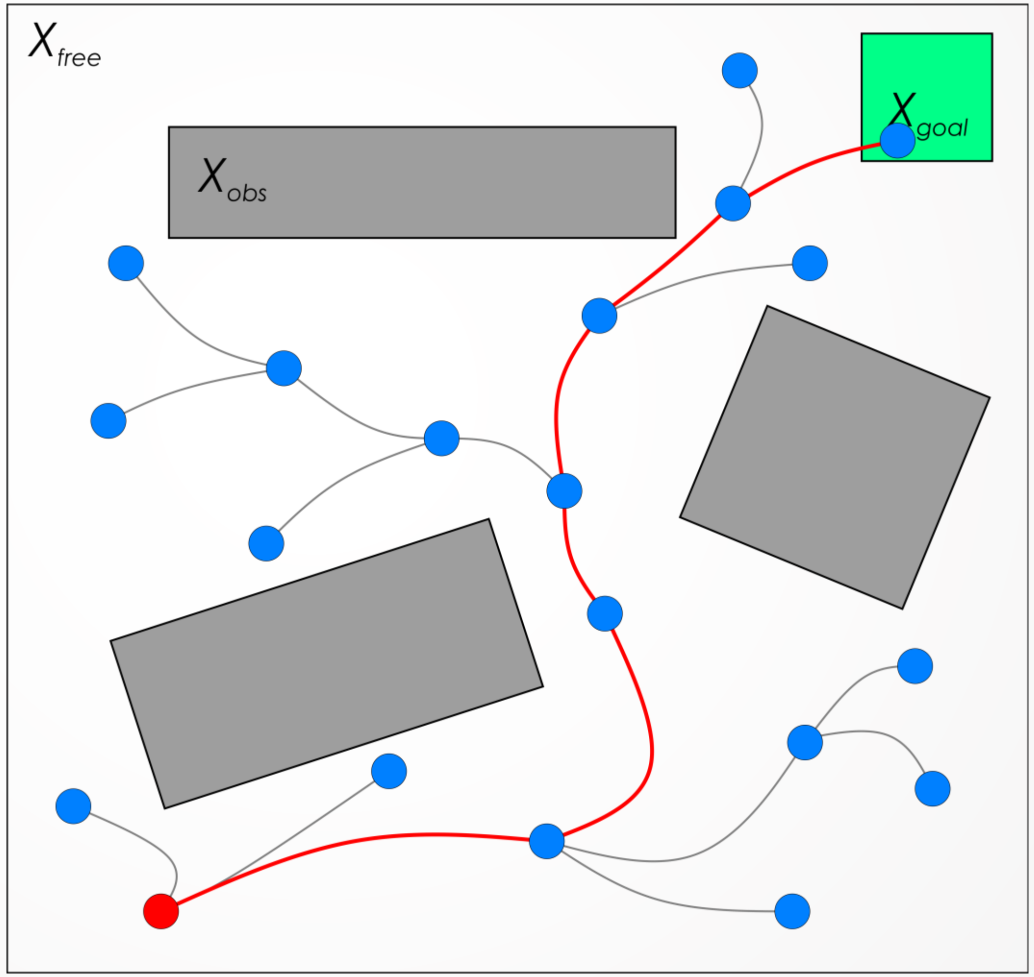
Rapidly-exploring Random Trees

RRT

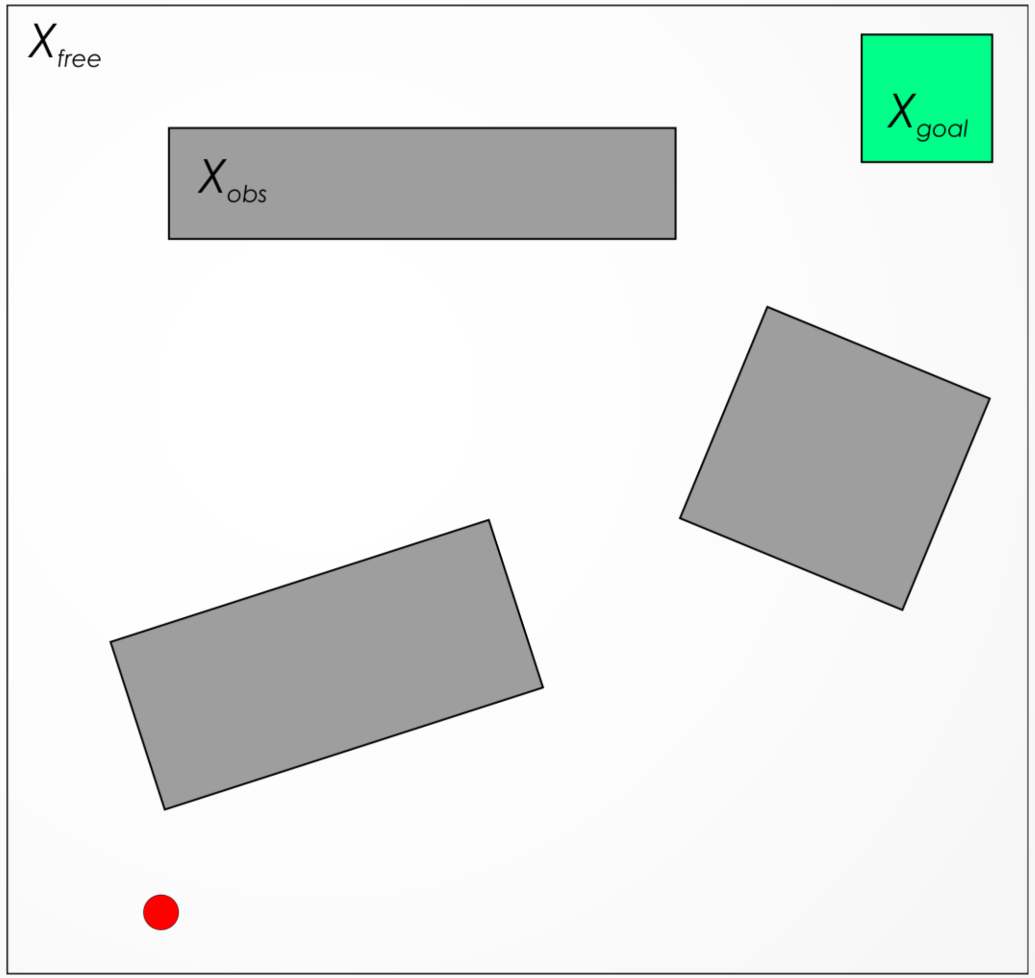
```
$V \leftarrow \{x_{init}\}; E \leftarrow 0;$
for i=1,…,N do:
    $x_{rand} \leftarrow SampleFree;$
    $x_{nearest} \leftarrow Nearest(G = (V, E), x_{rand});$
    $x_{new} \leftarrow Steer(x_{nearest}, x_{rand});$
    if $ObstacleFree(x_{nearest}, x_{new})$ then:
        $V \leftarrow V \cup \{x_{new}\}; E \leftarrow E \cup \{(x_{nearest}, x_{new})\};$
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.

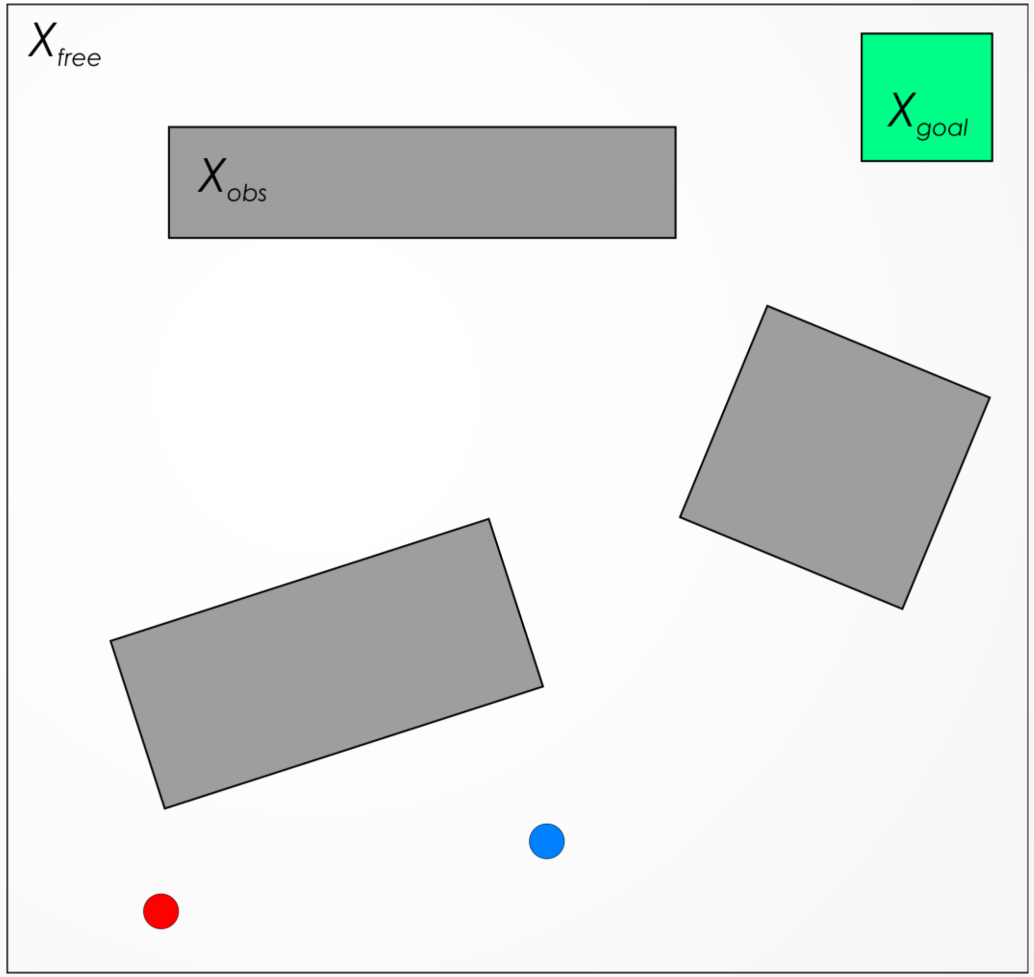
Rapidly-exploring Random Trees (RRTs)



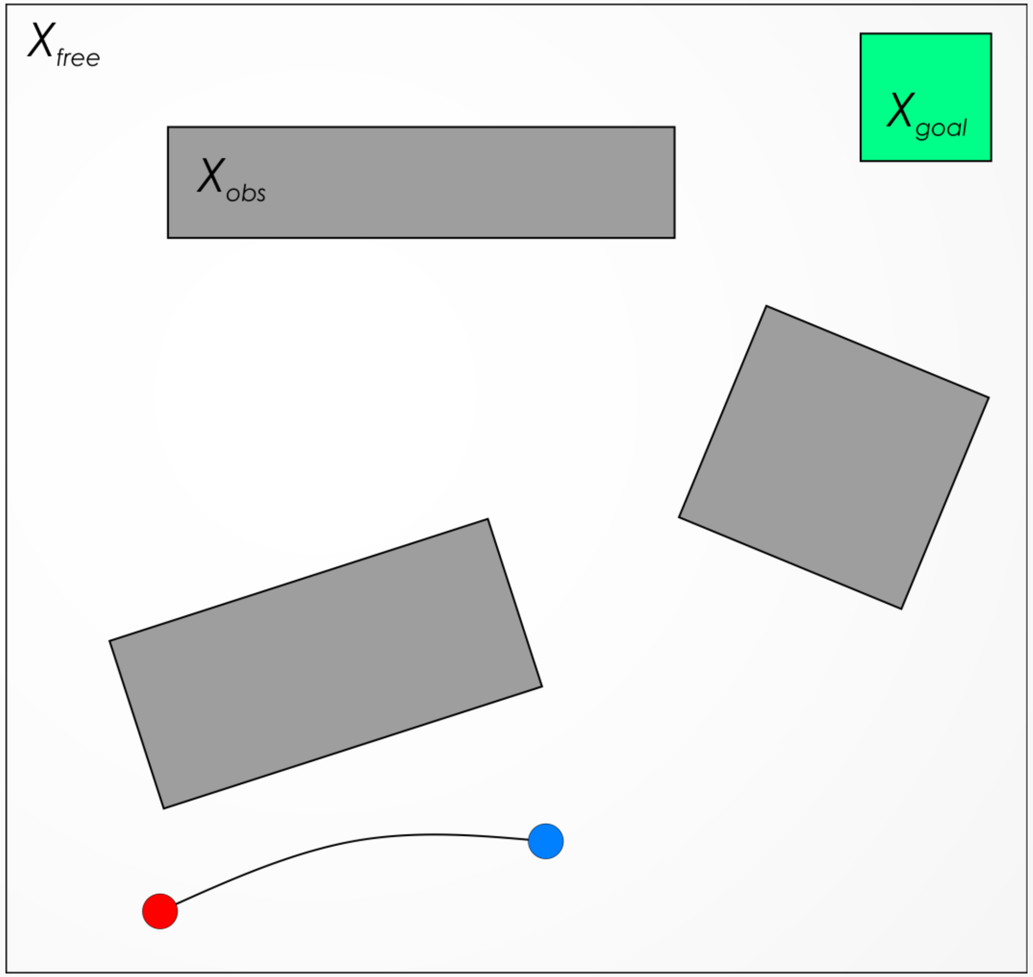
Rapidly-exploring Random Trees (RRTs)



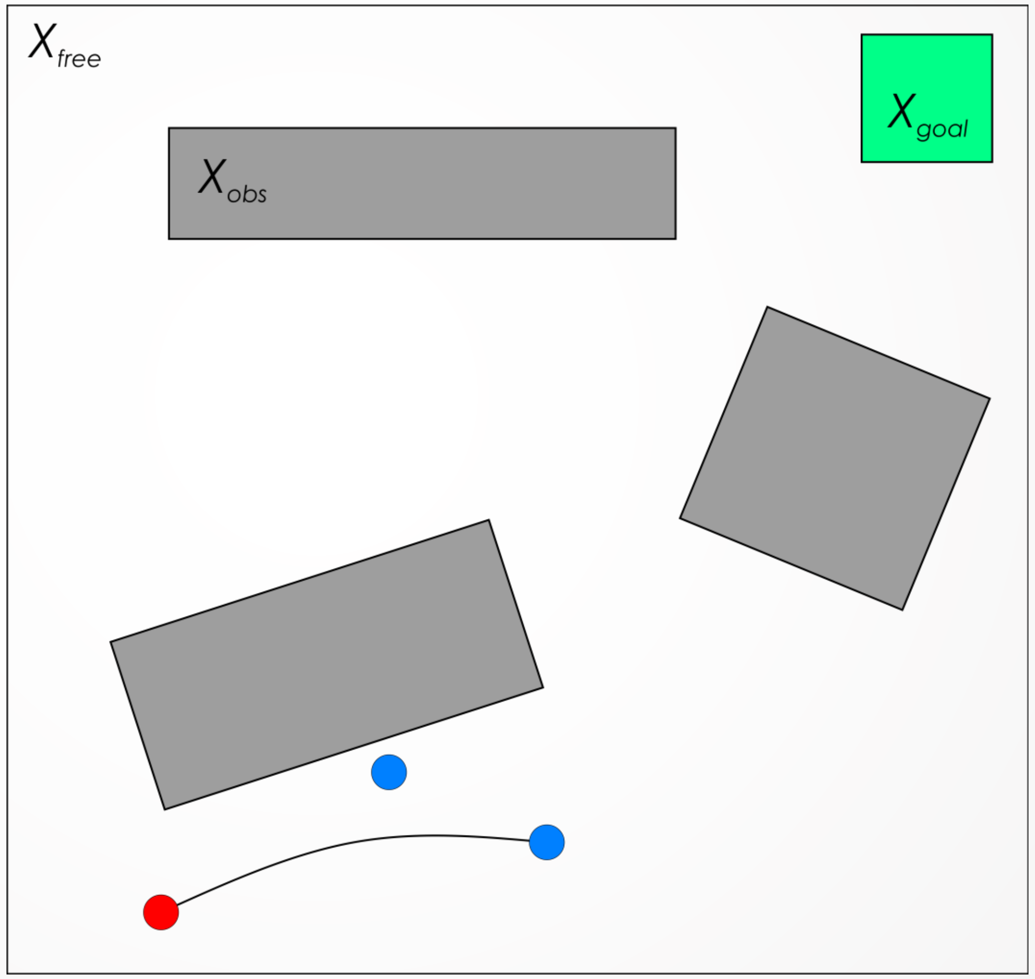
Rapidly-exploring Random Trees (RRTs)



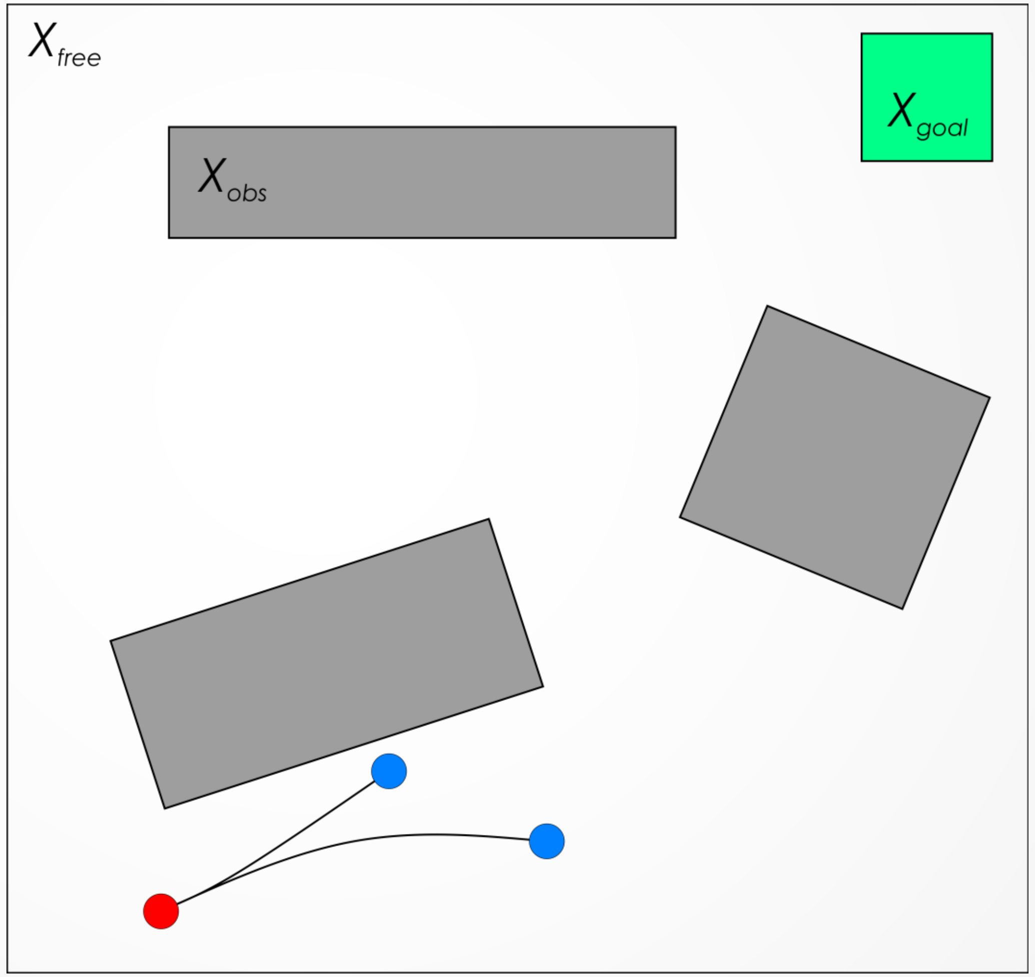
Rapidly-exploring Random Trees (RRTs)



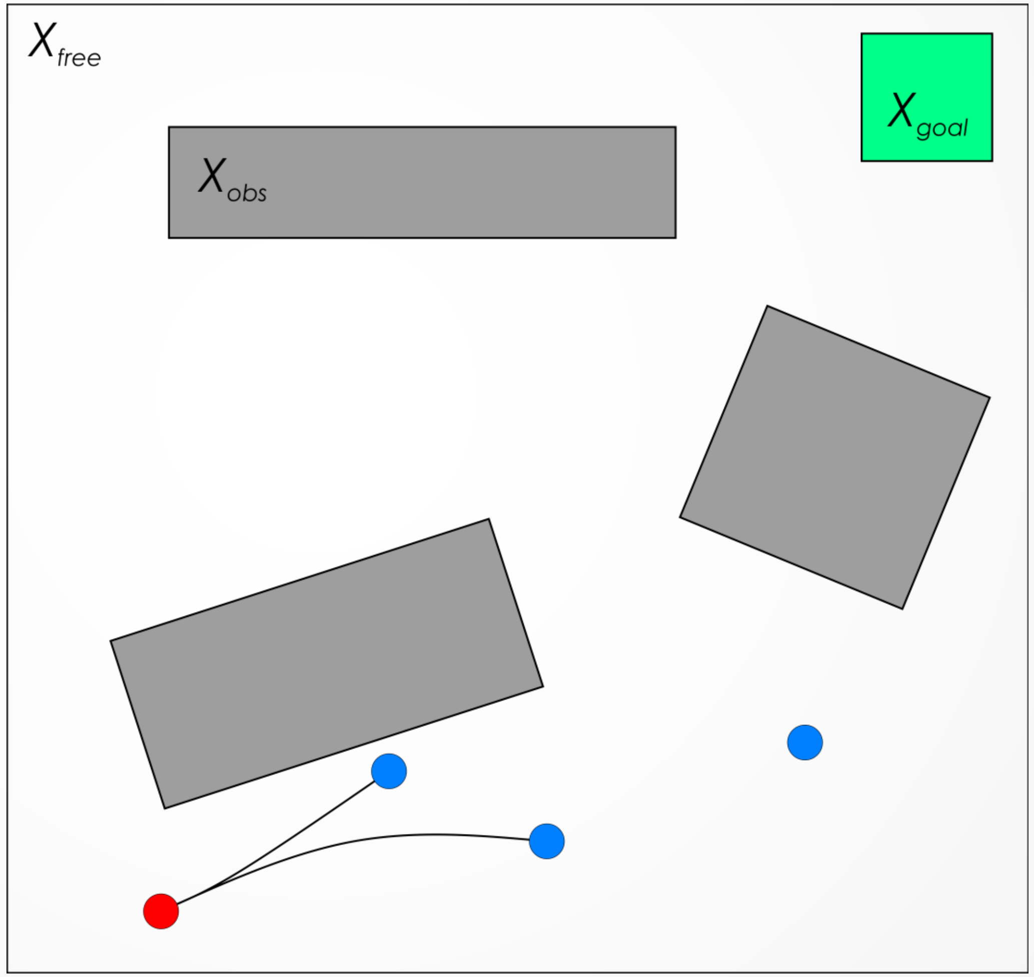
Rapidly-exploring Random Trees (RRTs)



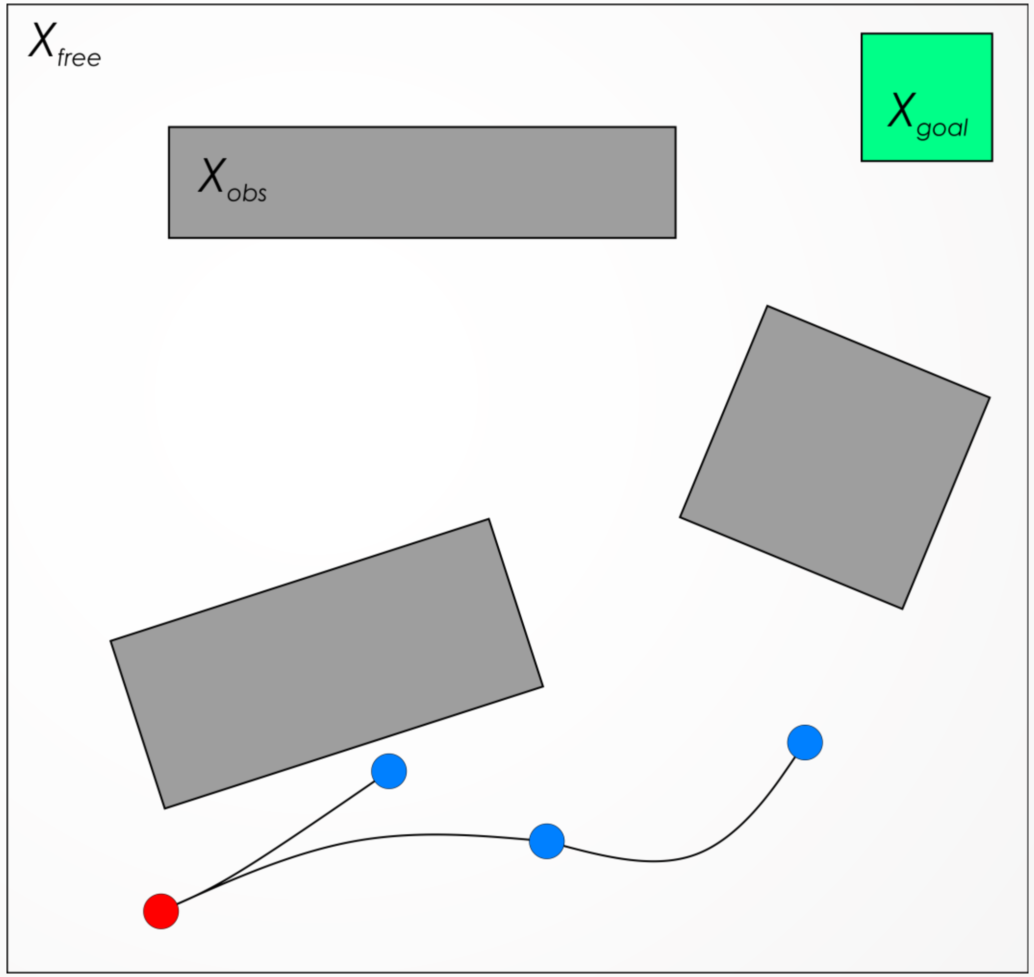
Rapidly-exploring Random Trees (RRTs)



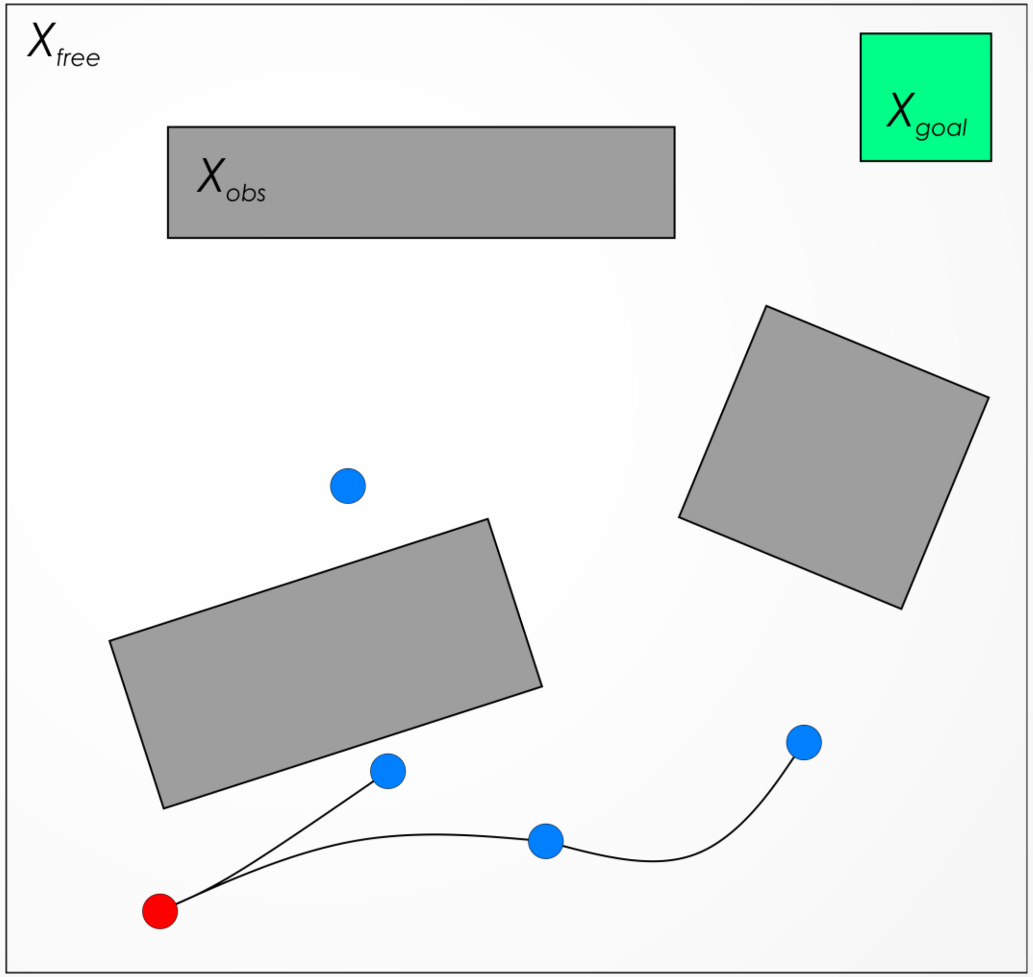
Rapidly-exploring Random Trees (RRTs)



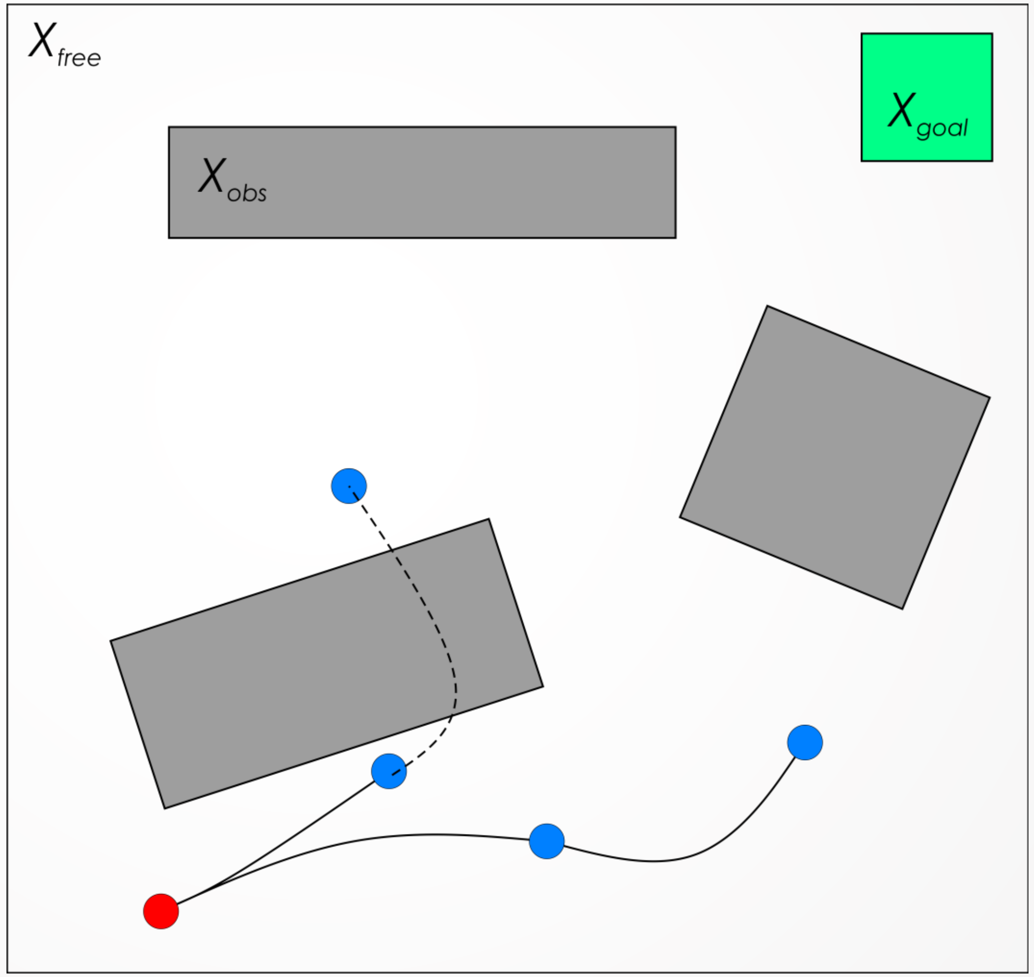
Rapidly-exploring Random Trees (RRTs)



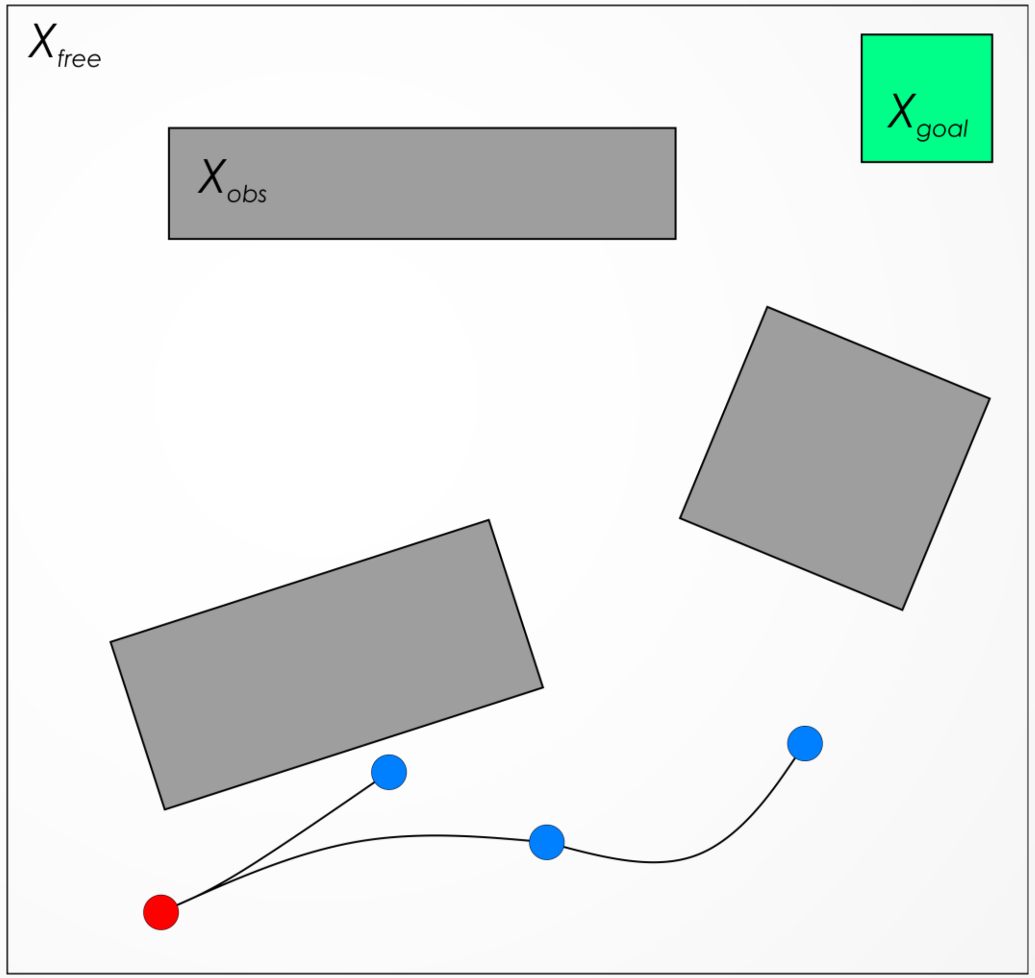
Rapidly-exploring Random Trees (RRTs)



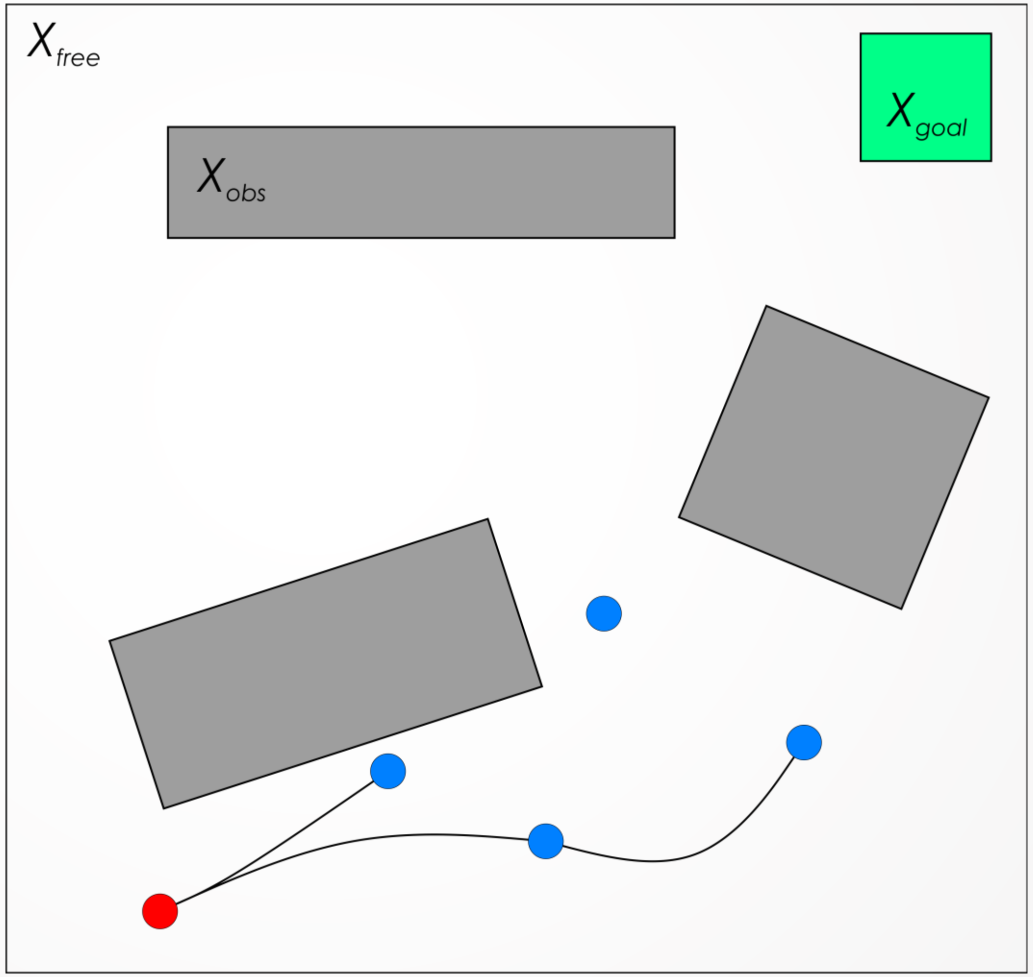
Rapidly-exploring Random Trees (RRTs)



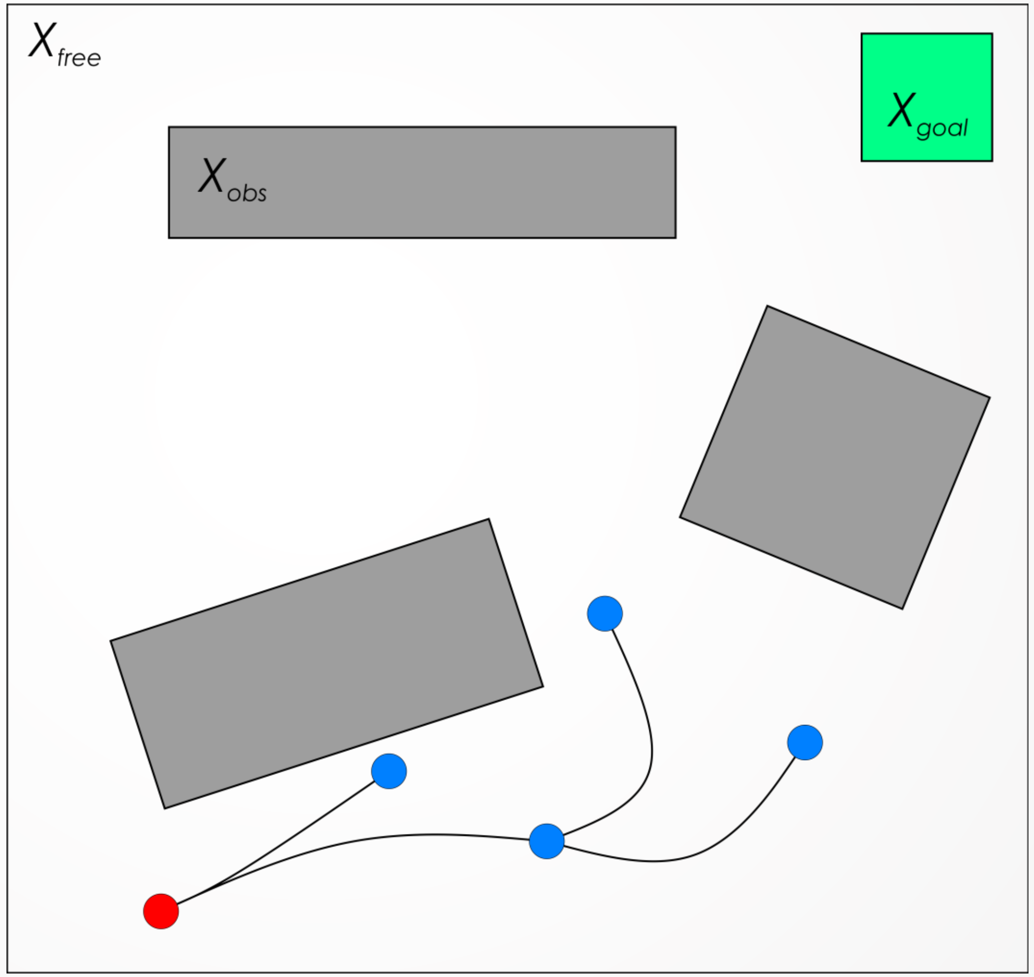
Rapidly-exploring Random Trees (RRTs)



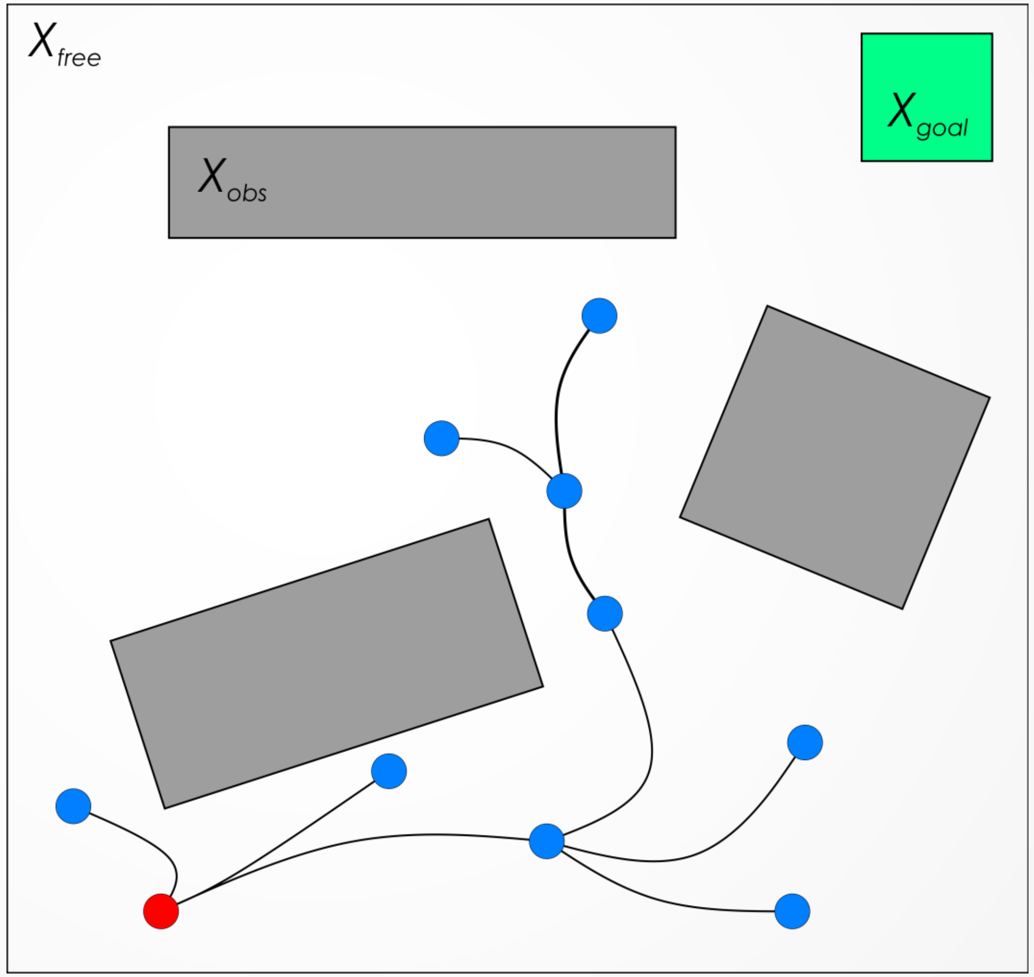
Rapidly-exploring Random Trees (RRTs)



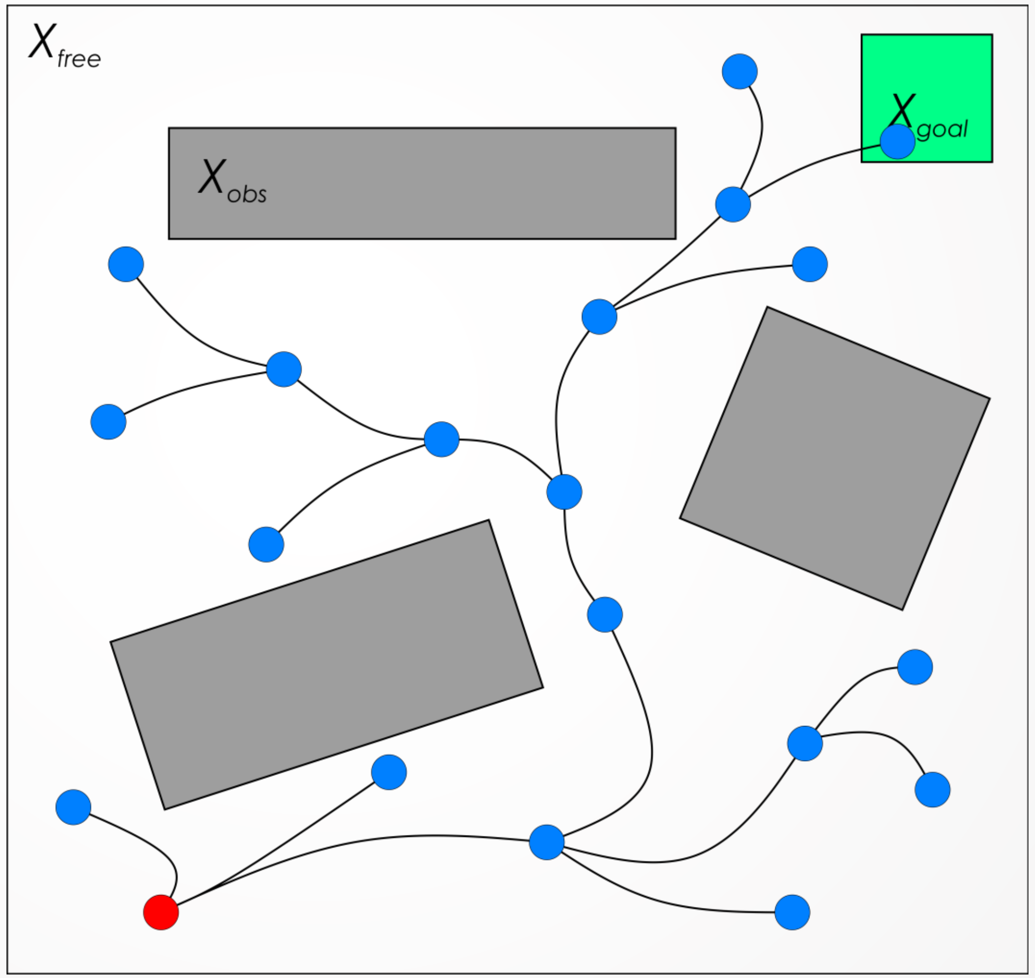
Rapidly-exploring Random Trees (RRTs)



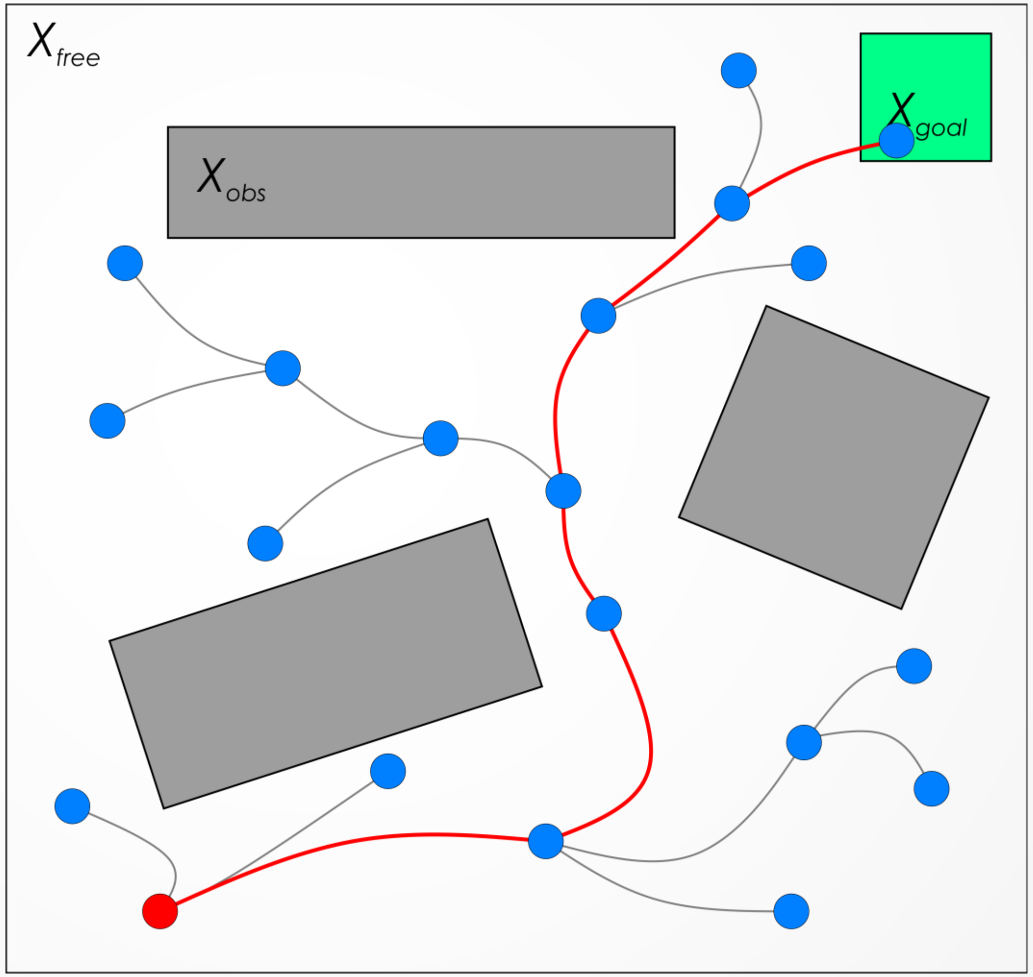
Rapidly-exploring Random Trees (RRTs)



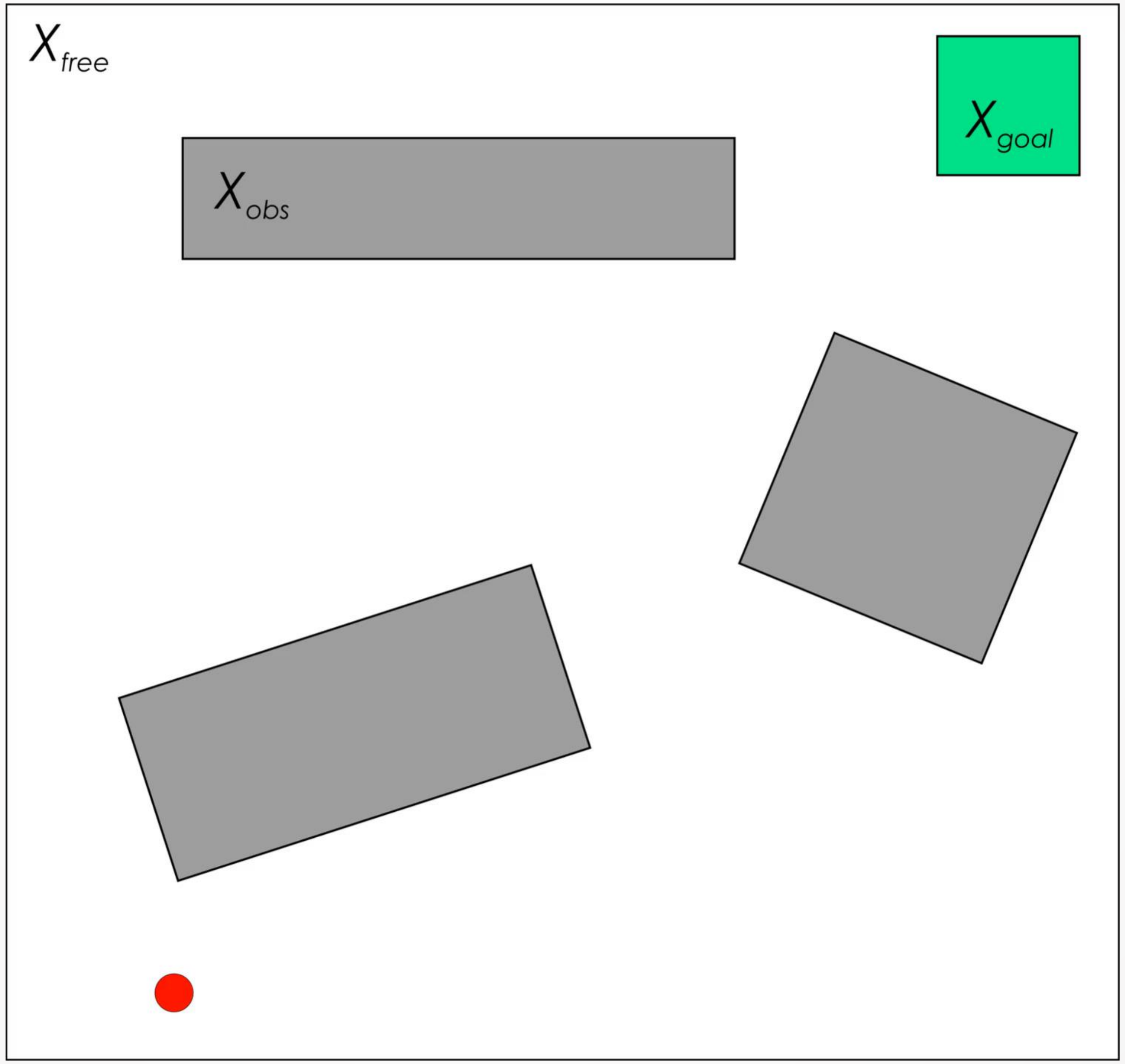
Rapidly-exploring Random Trees (RRTs)



Rapidly-exploring Random Trees (RRTs)



Rapidly-exploring Random Trees (RRTs)



RRTs in action

ETH zürich
Autonomous Systems Lab
Continuous-Time Trajectory Optimization for Online UAV Replanning
Helen Oleynikova, Michael Burri, Zachary Taylor, Juan Nieto, Roland Siegwart and Enric Galceran

Autonomous Robotic Aerial Tracking, Avoidance, and Seeking of a Mobile Human Subject
Christos Papachristos, Dimos Tzoumanikas, Kostas Alexis, and Anthony Tzes

- Great results for collision-avoidance of UAVs
- Integral component of several, higher-level algorithms (e.g. exploration and inspection)

Limitations of such incremental sampling methods

- 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 the one 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.

RRTs don’t have the best behavior

- 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.:
$$\lim_{n\to\infty} Y_n^{RRT} = Y_\infty^{RRT}$$
- The random variable Y_∞^{RRT} is sampled from a distribution with zero mass at the optimum:

Theorem – Almost sure suboptimality of RRTs

If a 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.:
$$\Pr[Y_\infty^{RRT} > c^*] = 1$$

Some remarks on that 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
  - Running multiple times concurrently
  - Deleting and rebuilding parts of the tree etc.

  Work better than the standard RRT, but cannot remove the sub-optimal behavior.

- **How can we do better?**

Rapidly-exploring Random Graphs (RRGs)

RRG Algorithm

```
$V \leftarrow \{x_{init}\}; E \leftarrow 0;$
for i=1,…,N do:
    $x_{rand} \leftarrow SampleFree;$
    $x_{nearest} \leftarrow Nearest(G = (V, E), x_{rand});$
    $x_{new} \leftarrow Steer(x_{nearest}, x_{rand});$
    if $ObstacleFree(x_{nearest}, x_{new})$ then:
        $X_{near} \leftarrow Near\left(G = (V, E), x_{new}, \min\{\gamma_{RRG}\left(\frac{\log(card\,V)}{card\,V}\right)^{1/d}, \eta\}\right);$
        $V \leftarrow V \cup \{x_{new}\}; E \leftarrow E \cup \{(x_{nearest}, x_{new}), (x_{nearest}, x_{new})\};$
        foreach $x_{near} \in X_{near}$ do:
            if $CollisionFree(x_{near}, x_{new})$ then $E \leftarrow E \cup \{(x_{near}, x_{new}), (x_{near}, x_{new})\};$
return $G = (V, E);$
```

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

Properties of RRGs

Theorem – Probabilistic completeness

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

$$\Pr\left[V_n^{RRG} \cap X_{goal} = 0\right] = O(e^{-bn})$$

Theorem – Asymptotic optimality

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

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

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.:

$$\Pr[Y_\infty^{RRG} = c^*] = 1$$

Computational complexity

- 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

$$\lim_{i \to \infty} sup\, E\left[\frac{OPS_i^{RRG}}{OPS_i^{RRT}}\right] \leq \beta$$

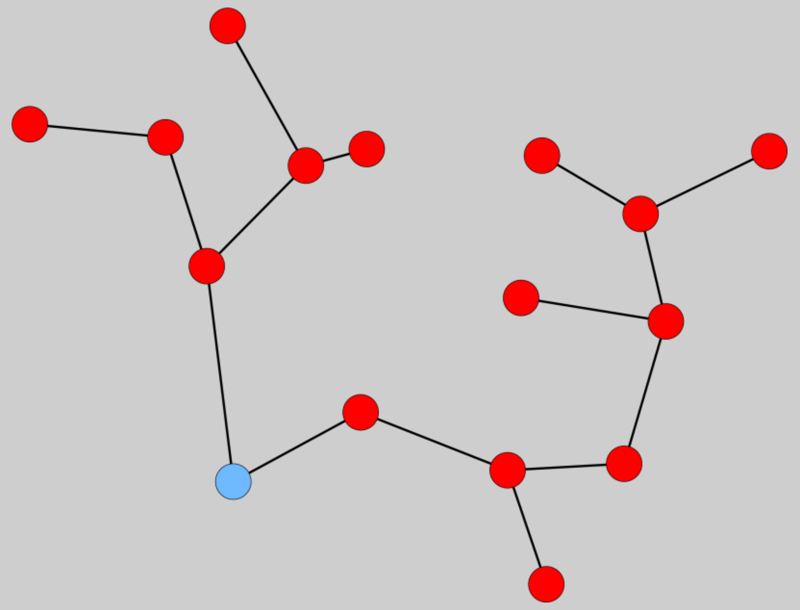
- In other words, the RRG algorithm has **no substantial computational overhead** over RRT, and ensures asymptotic optimality.

RRT*: A tree version of the RRG

- RRT algorithm can account for nonholonomic dynamics and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the *Steer* procedure has to be exact. Exact steering methods are not available for general dynamic 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.

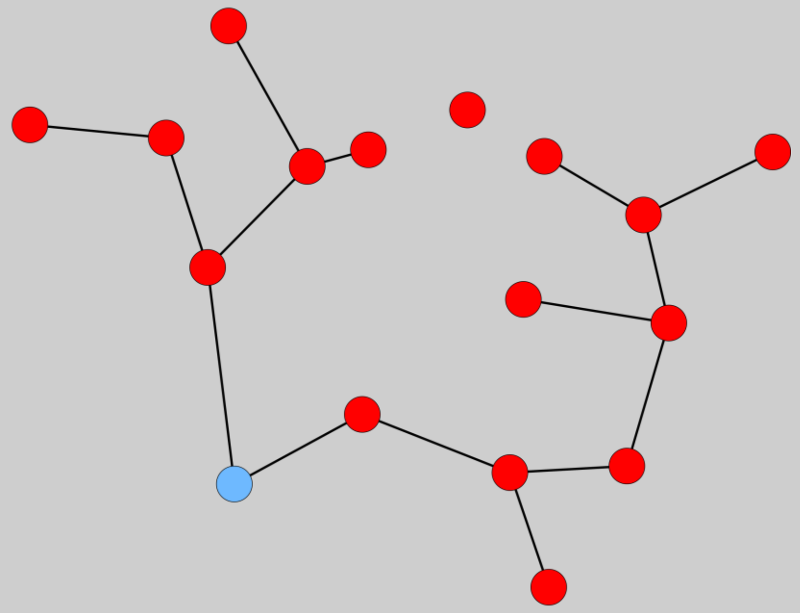


RRT*: A tree version of the RRG

- RRT algorithm can account for nonholonomic dynamics and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the $Steer$ procedure has to be exact. Exact steering methods are not available for general dynamic 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.

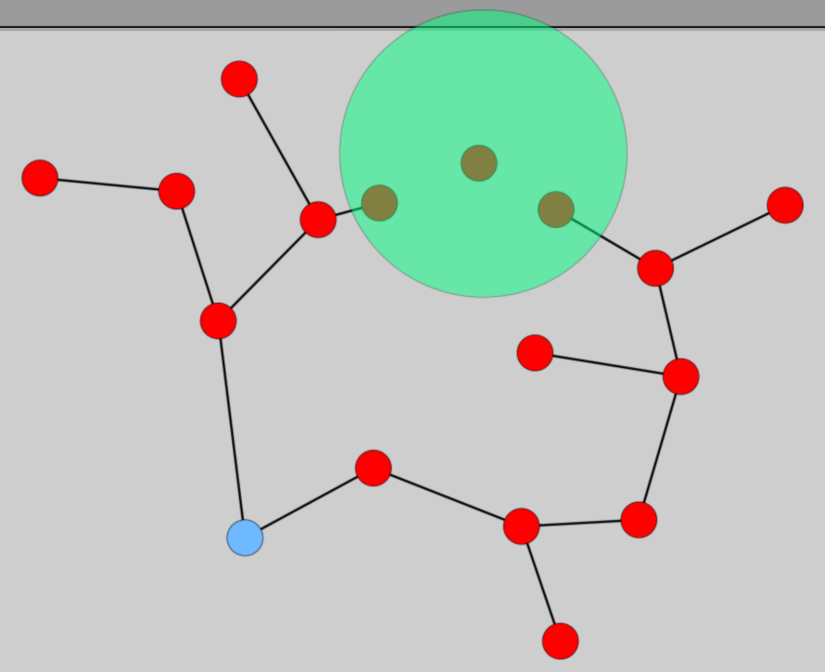


RRT*: A tree version of the RRG

- RRT algorithm can account for nonholonomic dynamics and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the $Steer$ procedure has to be exact. Exact steering methods are not available for general dynamic 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.

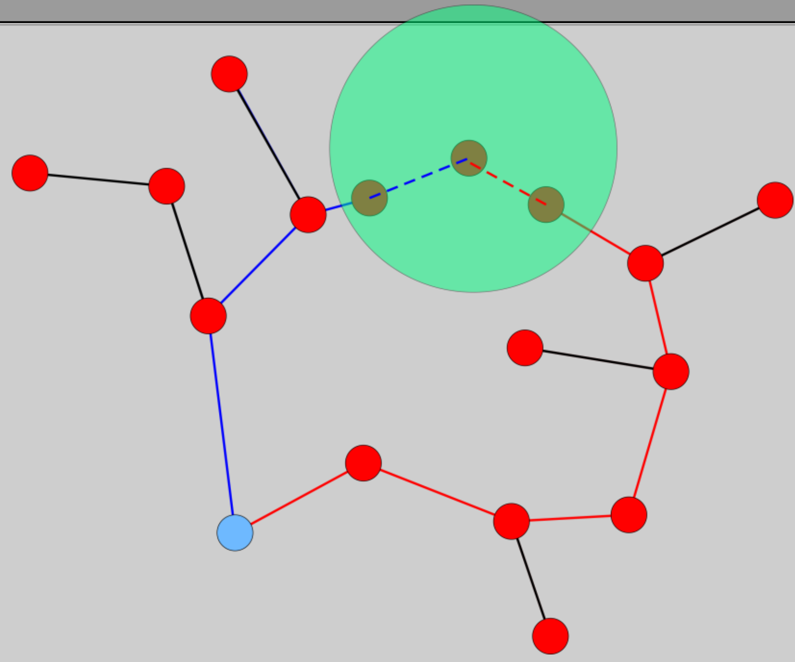


RRT*: A tree version of the RRG

- RRT algorithm can account for nonholonomic dynamics and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the Steer procedure has to be exact. Exact steering methods are not available for general dynamic 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.

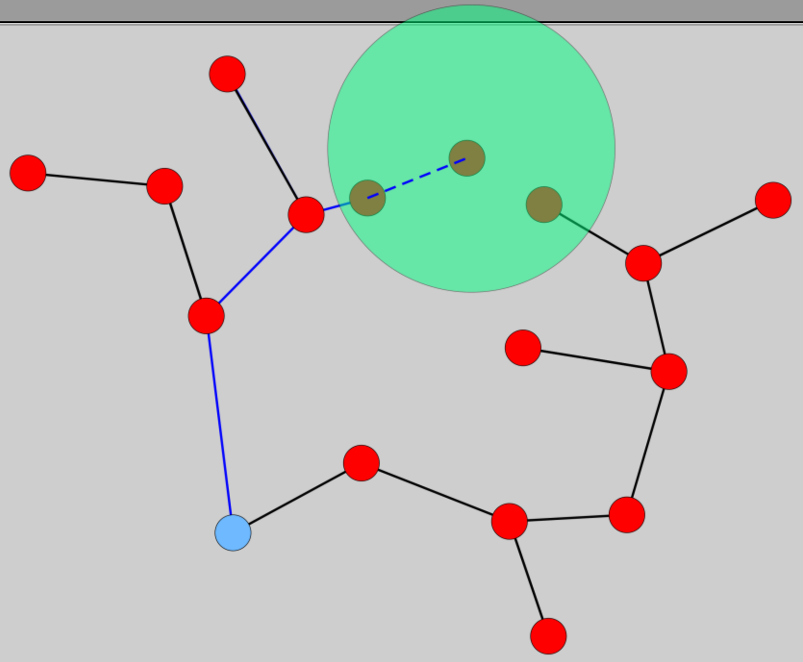


RRT*: A tree version of the RRG

- RRT algorithm can account for nonholonomic dynamics and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the Steer procedure has to be exact. Exact steering methods are not available for general dynamic 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.

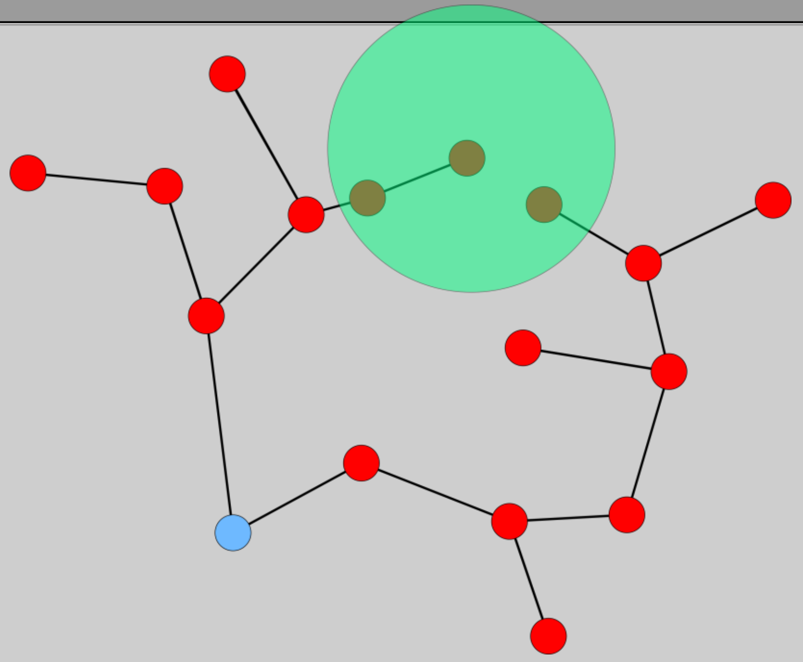


RRT*: A tree version of the RRG

- RRT algorithm can account for nonholonomic dynamics and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the $Steer$ procedure has to be exact. Exact steering methods are not available for general dynamic 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.

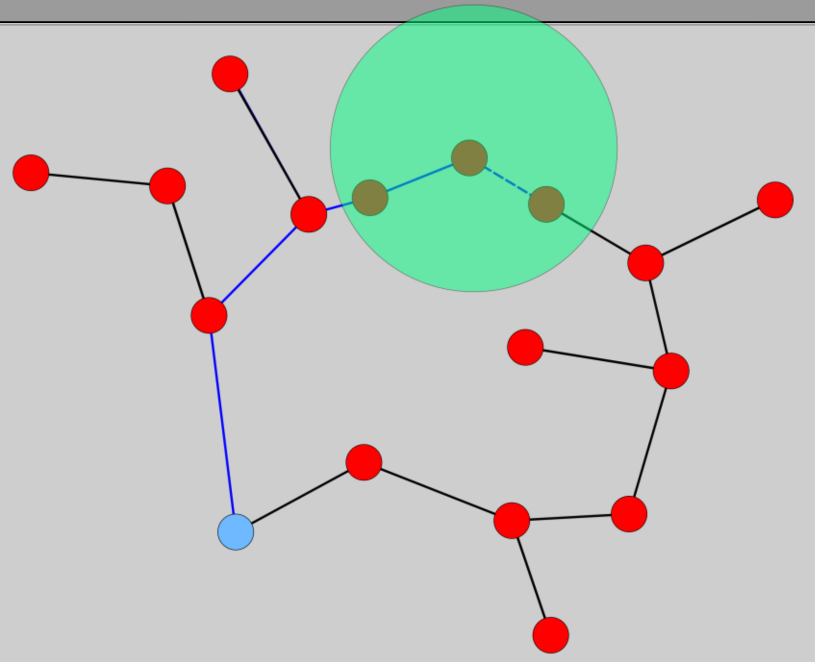


RRT*: A tree version of the RRG

- RRT algorithm can account for nonholonomic dynamics and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the $Steer$ procedure has to be exact. Exact steering methods are not available for general dynamic 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.

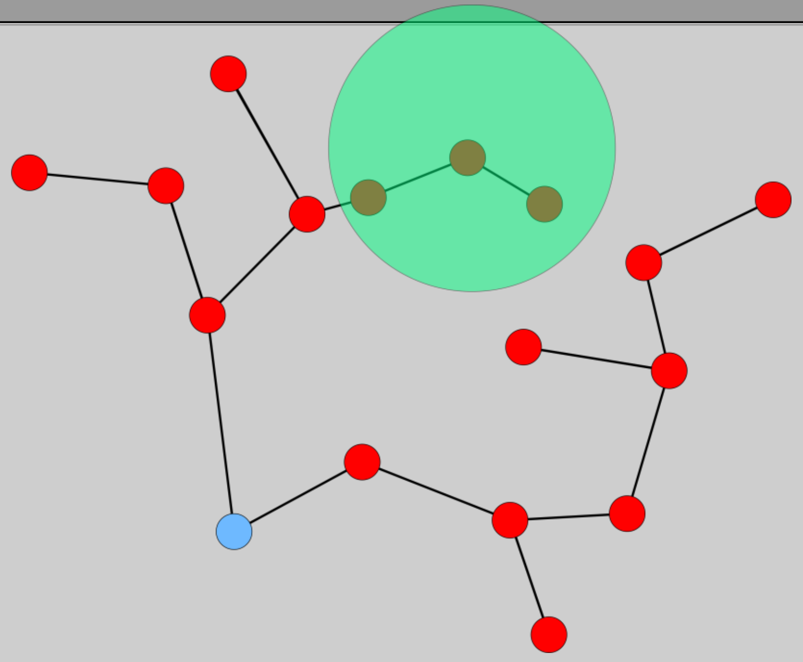


RRT*: A tree version of the RRG

- RRT algorithm can account for nonholonomic dynamics and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the $Steer$ procedure has to be exact. Exact steering methods are not available for general dynamic 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.

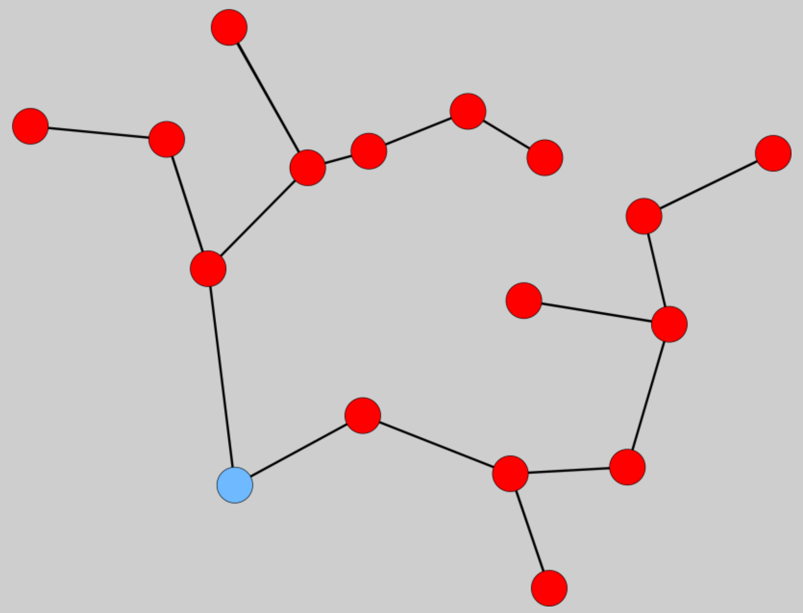


RRT*: A tree version of the RRG

- RRT algorithm can account for nonholonomic dynamics and modeling errors.
- RRG requires connecting the nodes exactly, i.e., the $Steer$ procedure has to be exact. Exact steering methods are not available for general dynamic 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.



Rapidly-exploring Random Tree-star (RRT*)

RRT* Algorithm

```
$V \leftarrow \{x_{init}\}; E \leftarrow 0;$
for i=1,…,N do:
    $x_{rand} \leftarrow SampleFree;$
    $x_{nearest} \leftarrow Nearest(G = (V, E), x_{rand});$
    $x_{new} \leftarrow Steer(x_{nearest}, x_{rand});$
    if $ObstacleFree(x_{nearest}, x_{new})$ then:
        $X_{near} \leftarrow Near\left(G = (V, E), x_{new}, \min\{\gamma_{RRG}\left(\frac{\log(card\ V)}{card\ V}\right)^{1/d}, \eta\}\right);$
        $V \leftarrow V \cup \{x_{new}\};$
        $x_{min} \leftarrow x_{nearest};\ c_{\min} \leftarrow Cost(x_{nearest}) + c(Line(x_{nearest}, x_{new}))$
        foreach $x_{near} \in X_{near}$ do:
            if $CollisionFree(x_{near}, x_{new}) \wedge Cost(x_{near}) + c\big(Line(x_{near}, x_{new})\big) < Cost(x_{near})$ then:
                $x_{parent} \leftarrow Parent(x_{near});$
                $E \leftarrow E \backslash \{(x_{parent}, x_{near})\} \cup \{(x_{new}, x_{near})\};$
return $G = (V, E);$
```

Code Examples and Tasks

- https://github.com/unr-arl/drones_demystified/tree/master/matlab/path-planning/rrt
- https://github.com/unr-arl/drones_demystified/tree/master/ROS/path-planning/structural-inspection
- https://github.com/unr-arl/drones_demystified/tree/master/ROS/path-planning/autonomous-exploration

Find out more

- http://www.autonomousrobotslab.com/holonomic-vehicle-bvs.html
- http://www.autonomousrobotslab.com/dubins-airplane.html
- http://www.autonomousrobotslab.com/collision-free-navigation.html
- http://www.autonomousrobotslab.com/structural-inspection-path-planning.html

- http://ompl.kavrakilab.org/
- http://moveit.ros.org/
- http://planning.cs.uiuc.edu/

- http://www.autonomousrobotslab.com/literature-and-links1.html

Thank you!

Please ask your question!