Lecture 2-2

Sample-based path finding

高飞

浙江大学 控制学院

采样类最短路径算法

- 高效地探索环境的拓扑结构，即可行区域的连接情况
- 不显示地构建构型空间及其边界
- 一般具有概率完备性
- 一般具有次优性或渐进最优性
- 可分为单查询（single-query）和多查询（multi-query）

Probabilistic Road Map

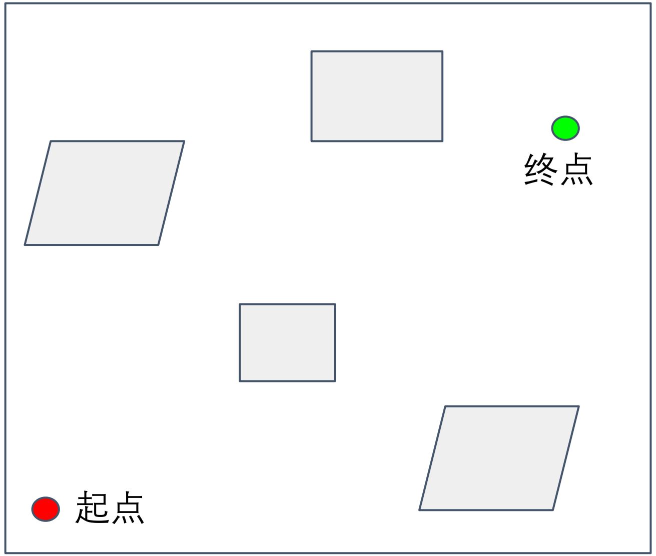
PRM是什么？

- 全称为Probabilistic Road Map
- 一种多查询（multi-query）算法
- 将规划分为两个阶段：

    构建阶段

    搜索阶段

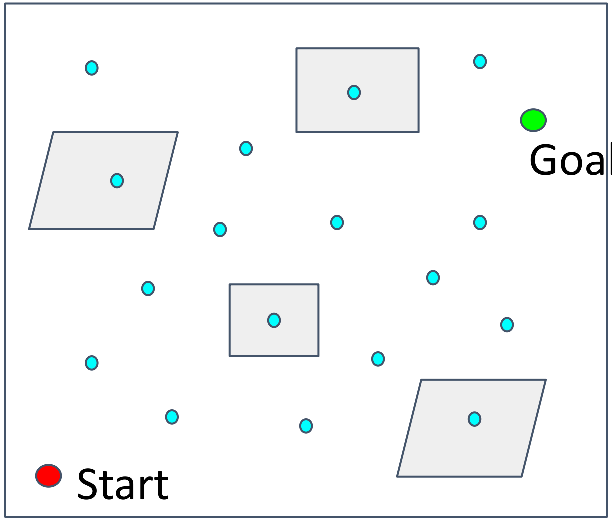
- 用相对少量的路标点和局部路径来构建一个连接图以得到可行区域的连接情况



Probabilistic Road Map（PRM）

构建阶段：构建一个表征环境连通情况的路标图

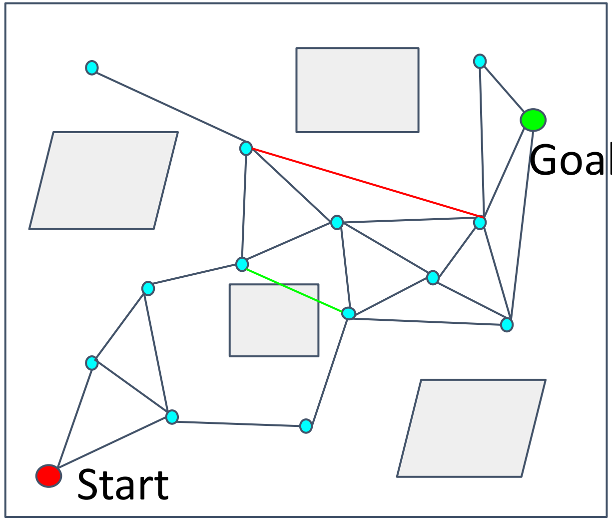
- 在工作空间采样N个点
- 删除和环境碰撞的点



Probabilistic Road Map（PRM）

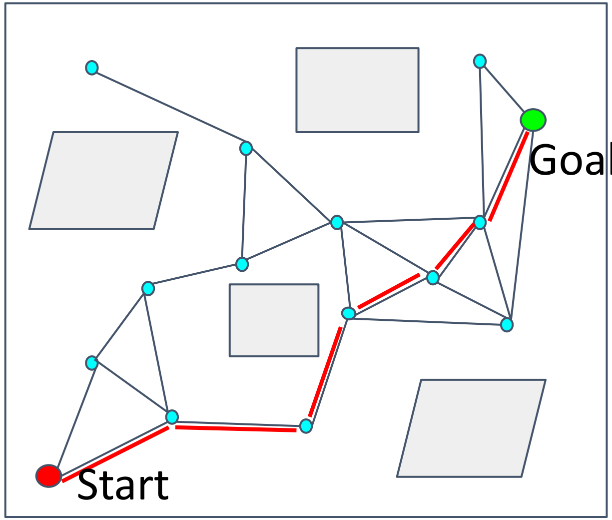
构建阶段：构建一个表征环境连通情况的路标图

- 连接近邻的节点
- 删除和环境碰撞的路径段



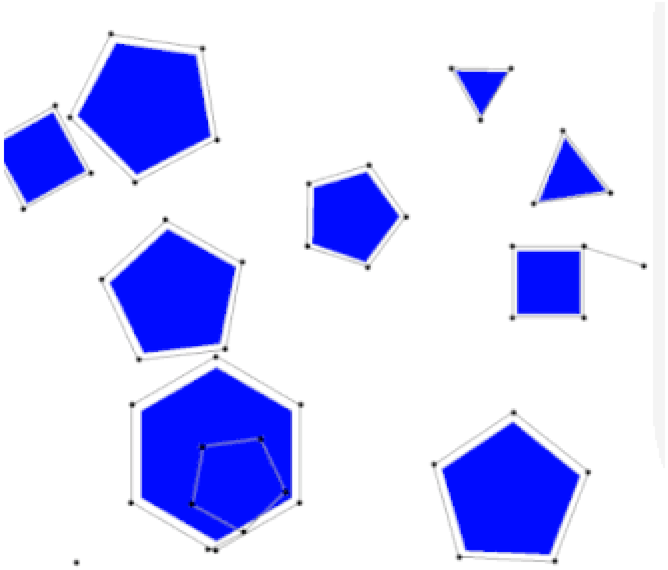
搜索阶段

- 在构建出的路标连接图中搜索一条起点到终点的路径（使用Dijkstra或者A*算法）
- 路标图可近似看作栅格图
- 可利用路标图进行多次搜索（multi-query）

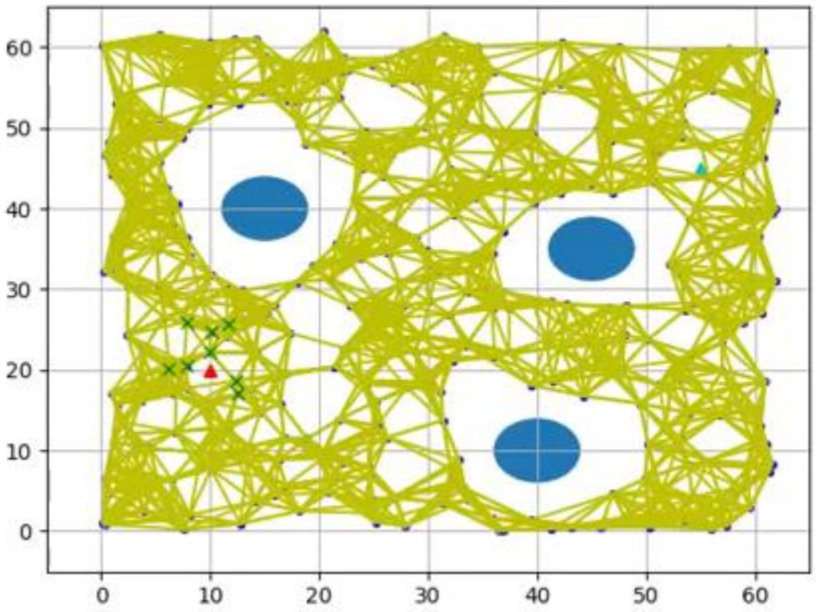


Probabilistic Road Map（PRM）

Learning phase[1]



Query phase[2]



[1] https://en.wikipedia.org/wiki/Probabilistic_roadmap
[2] https://www.youtube.com/watch?v=8DIn3sS_p4Q

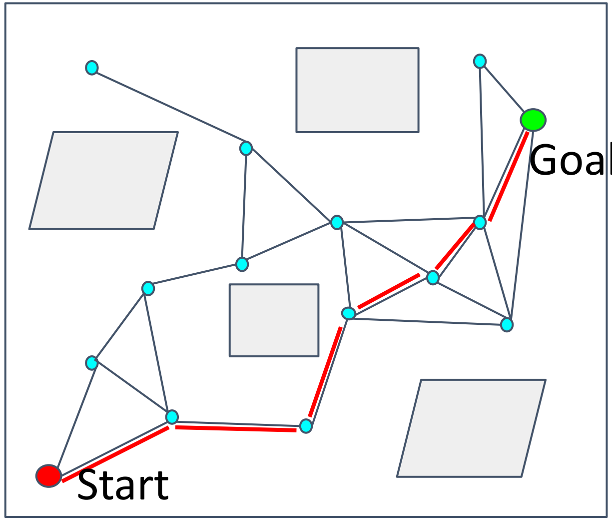
算法优劣

优势

- 概率完备性

劣势

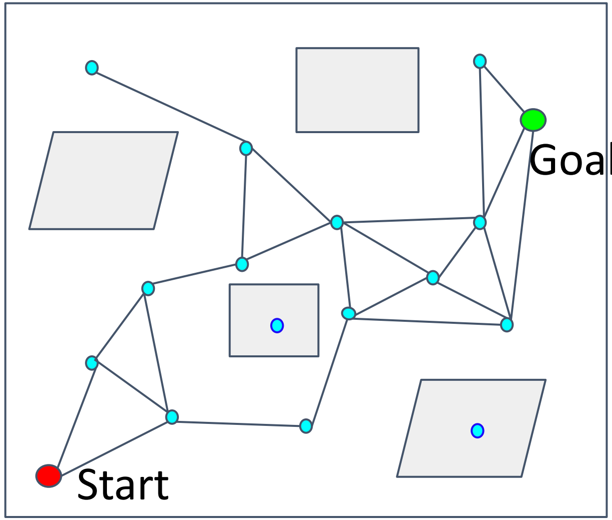
- 构建连接图没有专注于产生路径
- 产生大量低效无用的采样和路径连接
- 低效



Note：效率提升方法

- ➢ 懒惰(Lazy)的碰撞检查

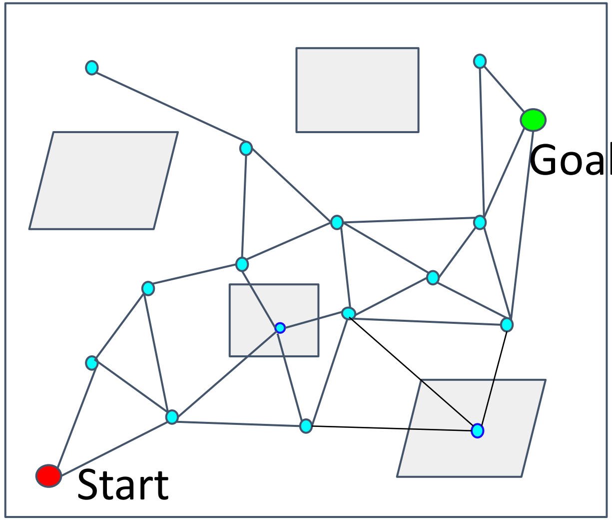
- 碰撞检查过程非常耗时，尤其是在复杂或高维环境中。



Probabilistic Road Map（PRM）

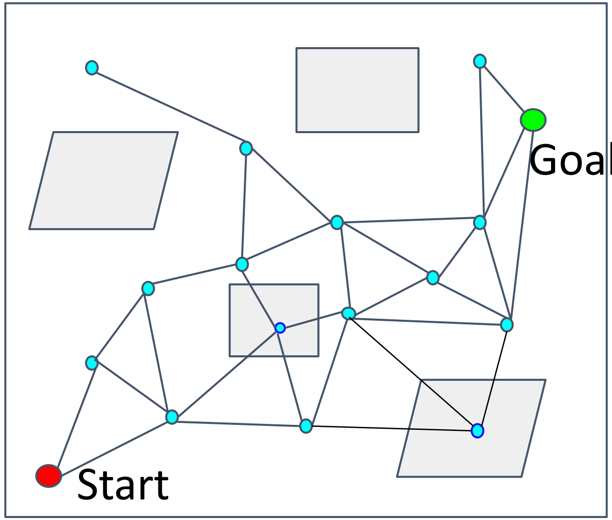
- ➢ 懒惰(Lazy)的碰撞检查

- 采样点并生成线段，不考虑碰撞（Lazy）。



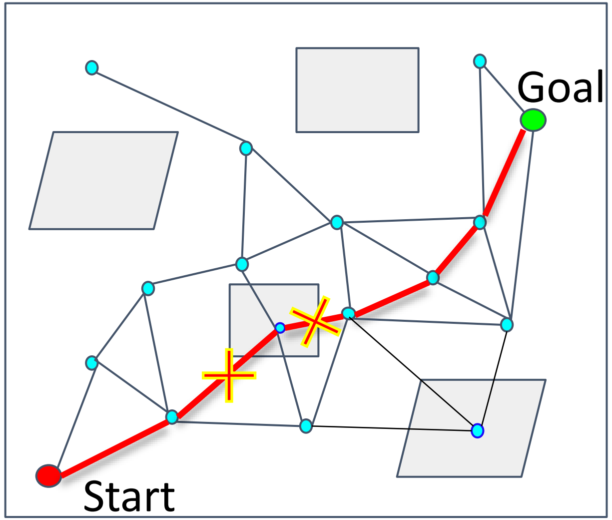
➢ 懒惰(Lazy)的碰撞检查

- 碰撞检查（如有必要）：
- 在未进行碰撞检查的情况下生成的路线图上查找路径

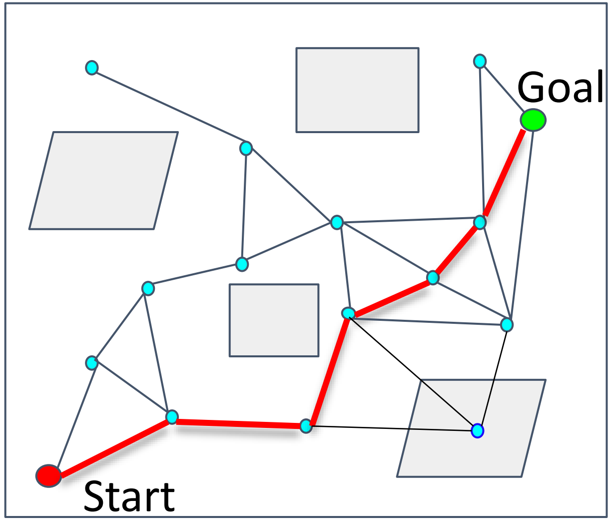


➢ 懒惰(Lazy)的碰撞检查

- 碰撞检查（如有必要）：
- 如果路径不是无碰撞的，则删除相应的边和节点。

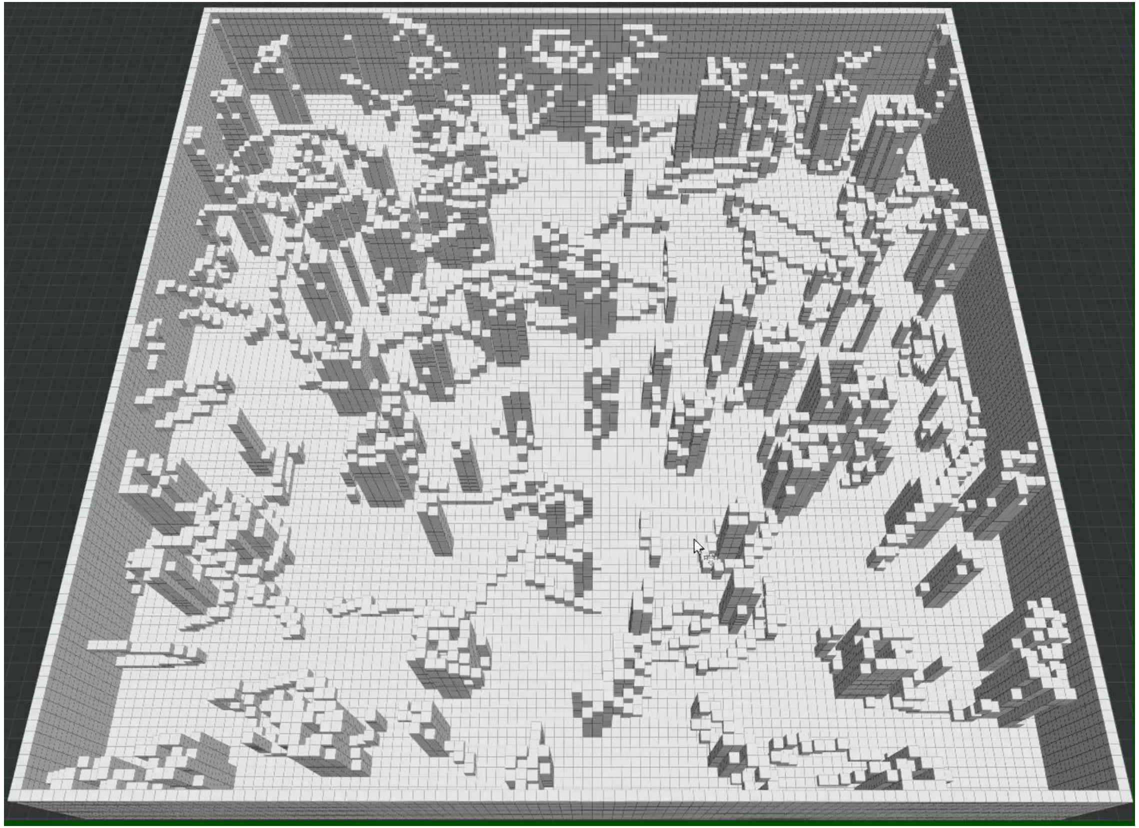


- 懒惰(Lazy)的碰撞检查



- 如果路径不是无碰撞的，则删除相应的边和节点。
- 重新开始路径查找。

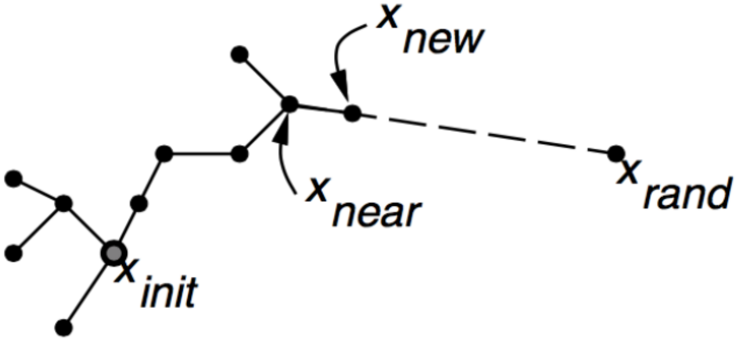
Rapidly-exploring Random Tree（RRT）



https://github.com/ZJU-FAST-Lab/sampling-based-path-finding

RRT是什么？

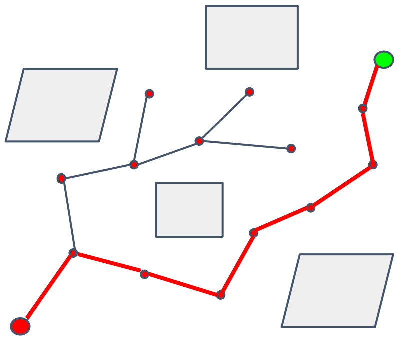
- 全称为Rapidly-exploring Random Trees
- 一种单查询（single-query）算法
- 通过在工作空间采样节点来构建一棵从起点到终点的树，随着采样增加，树从起点向终点生长



Rapidly-exploring Random Tree（RRT）

Algorithm 1: RRT Algorithm

Input: $\mathcal{M}$,x_{init},x_{goal}
Result: A path Γ from x_{init} to x_{goal}
$\mathcal{T}$.init();
for $i = 1$ *to* n **do**
  $x_{rand} \leftarrow Sample(\mathcal{M})$;
  $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
  $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
  $E_i \leftarrow Edge(x_{new}, x_{near})$;
  if *CollisionFree*($\mathcal{M}$, E_i) **then**
    $\mathcal{T}.addNode(x_{new})$;
    $\mathcal{T}.addEdge(E_i)$;
  if $x_{new} = x_{goal}$ **then**
    Success();



算法流程

- 在可行空间随机采样
- 找到当前树中离采样点最近的树节点
- 从最近的树节点“生长”出新的节点和树枝（路径）
- 如果此路径没有和环境发生碰撞，则将此节点和路径加到树中
- 重复n次采样，直到树生长到终点区域

Rapidly-exploring Random Tree（RRT）

Algorithm 1: RRT Algorithm

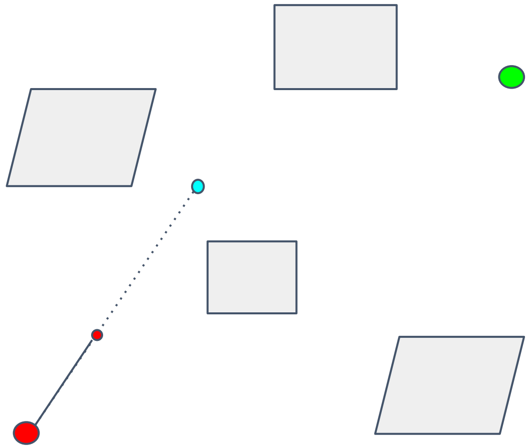
Input: $\mathcal{M}$,x_{init},x_{goal}

Result: A path Γ from x_{init} to x_{goal}

$\mathcal{T}$.init();

for $i = 1$ *to* n **do**

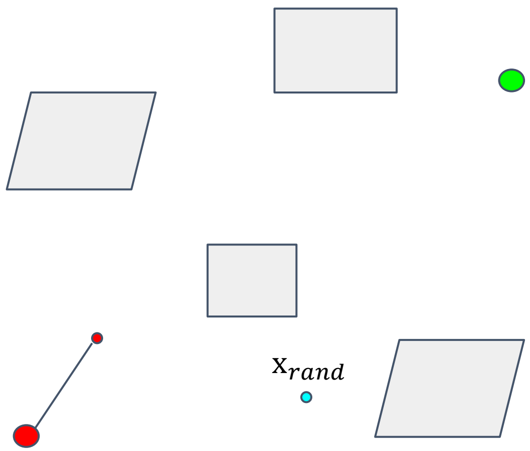
- $x_{rand} \leftarrow Sample(\mathcal{M})$;
- $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
- $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
- $E_i \leftarrow Edge(x_{new}, x_{near})$;
- **if** *CollisionFree*($\mathcal{M}$, E_i) **then**
  - $\mathcal{T}.addNode(x_{new})$;
  - $\mathcal{T}.addEdge(E_i)$;
- **if** $x_{new} = x_{goal}$ **then**
  - Success();



Rapidly-exploring Random Tree（RRT）

Algorithm 1: RRT Algorithm

Input: $\mathcal{M}$,x_{init},x_{goal}
Result: A path Γ from x_{init} to x_{goal}
$\mathcal{T}$.init();
for $i = 1$ *to* n **do**
  $x_{rand} \leftarrow Sample(\mathcal{M})$;
  $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
  $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
  $E_i \leftarrow Edge(x_{new}, x_{near})$;
  if *CollisionFree*($\mathcal{M}$, E_i) **then**
    $\mathcal{T}.addNode(x_{new})$;
    $\mathcal{T}.addEdge(E_i)$;
  if $x_{new} = x_{goal}$ **then**
    Success();



Rapidly-exploring Random Tree（RRT）

Algorithm 1: RRT Algorithm

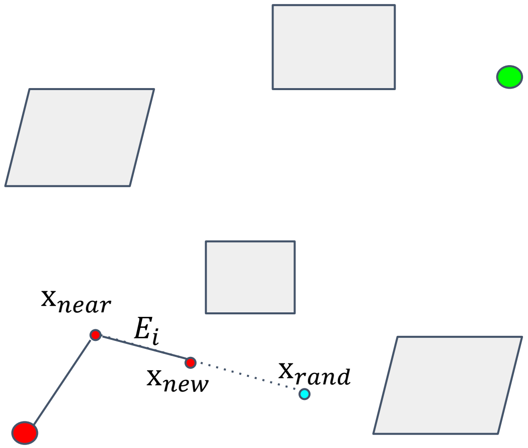
Input: $\mathcal{M}$,x_{init},x_{goal}

Result: A path Γ from x_{init} to x_{goal}

$\mathcal{T}$.init();

for $i = 1$ *to* n **do**

- $x_{rand} \leftarrow Sample(\mathcal{M})$;
- $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
- $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
- $E_i \leftarrow Edge(x_{new}, x_{near})$;
- **if** *CollisionFree*($\mathcal{M}$, E_i) **then**
  - $\mathcal{T}.addNode(x_{new})$;
  - $\mathcal{T}.addEdge(E_i)$;
- **if** $x_{new} = x_{goal}$ **then**
  - Success();



Rapidly-exploring Random Tree（RRT）

Algorithm 1: RRT Algorithm

Input: $\mathcal{M}$,x_{init},x_{goal}

Result: A path Γ from x_{init} to x_{goal}

$\mathcal{T}$.init();

for $i = 1$ *to* n **do**

  $x_{rand} \leftarrow Sample(\mathcal{M})$;

  $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;

  $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;

  $E_i \leftarrow Edge(x_{new}, x_{near})$;

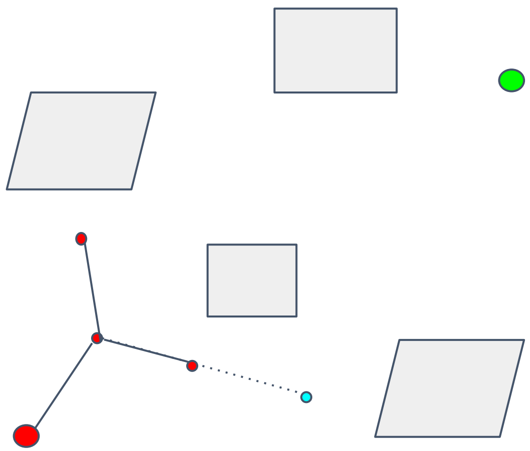
  if *CollisionFree*($\mathcal{M}$, E_i) **then**

    $\mathcal{T}.addNode(x_{new})$;

    $\mathcal{T}.addEdge(E_i)$;

  if $x_{new} = x_{goal}$ **then**

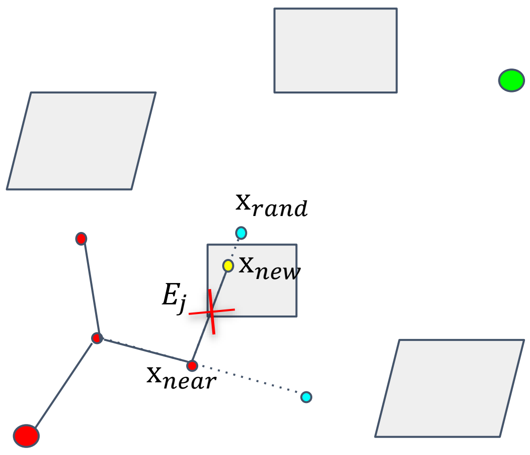
    Success();



Rapidly-exploring Random Tree（RRT）

Algorithm 1: RRT Algorithm

Input: $\mathcal{M}$,x_{init},x_{goal}
Result: A path Γ from x_{init} to x_{goal}
$\mathcal{T}$.init();
for $i = 1$ *to* n **do**
    $x_{rand} \leftarrow Sample(\mathcal{M})$;
    $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
    $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
    $E_i \leftarrow Edge(x_{new}, x_{near})$;
    if *CollisionFree*($\mathcal{M}$, E_i) **then**
        $\mathcal{T}.addNode(x_{new})$;
        $\mathcal{T}.addEdge(E_i)$;
    if $x_{new} = x_{goal}$ **then**
        Success();



Rapidly-exploring Random Tree（RRT）

Algorithm 1: RRT Algorithm

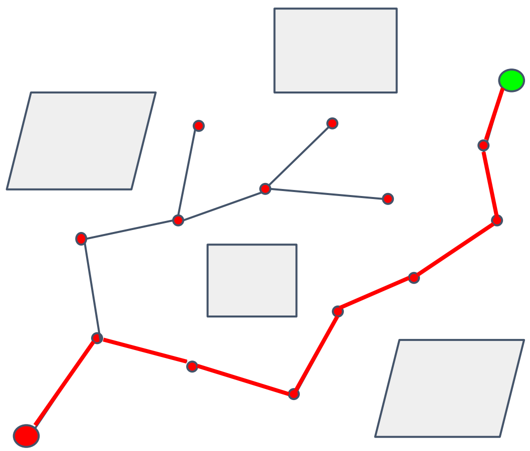
Input: $\mathcal{M}$,x_{init},x_{goal}

Result: A path Γ from x_{init} to x_{goal}

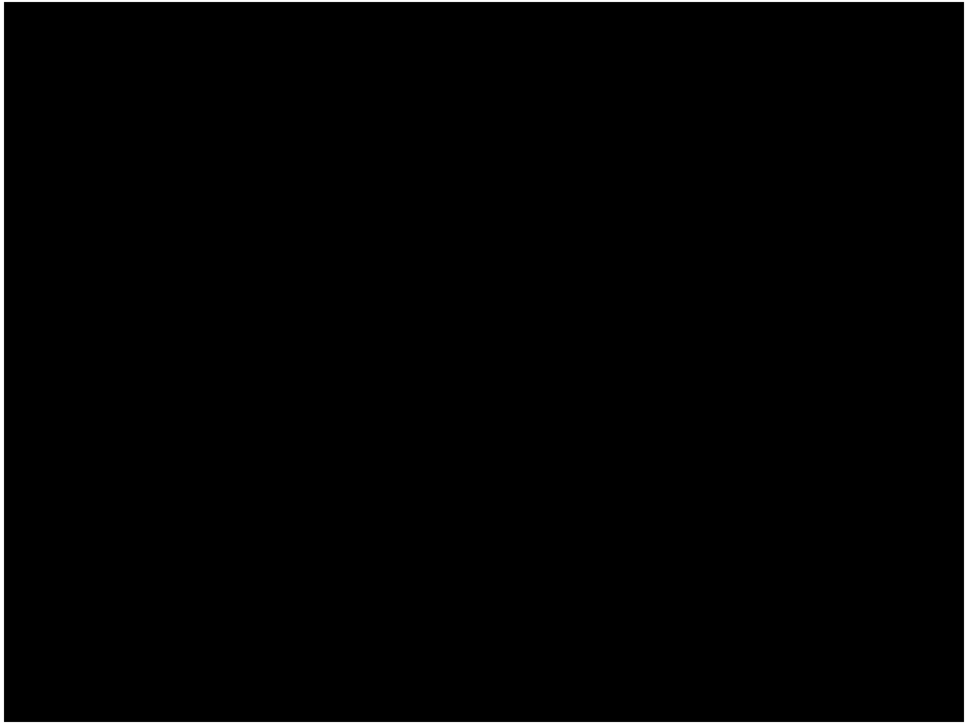
$\mathcal{T}$.init();

for $i = 1$ *to* n **do**

- $x_{rand} \leftarrow Sample(\mathcal{M})$;
- $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
- $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
- $E_i \leftarrow Edge(x_{new}, x_{near})$;
- **if** *CollisionFree*($\mathcal{M}$, E_i) **then**
  - $\mathcal{T}.addNode(x_{new})$;
  - $\mathcal{T}.addEdge(E_i)$;
- **if** $x_{new} = x_{goal}$ **then**
  - Success();



Rapidly-exploring Random Tree（RRT）



[1] https://www.youtube.com/watch?v=pOFtvZ_qVsA

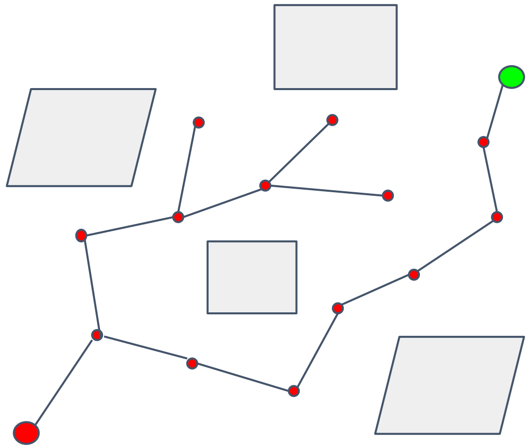
算法优劣

优势

- 致力于找到从起点到终点的路径
- 相比PRM更具目标导向性

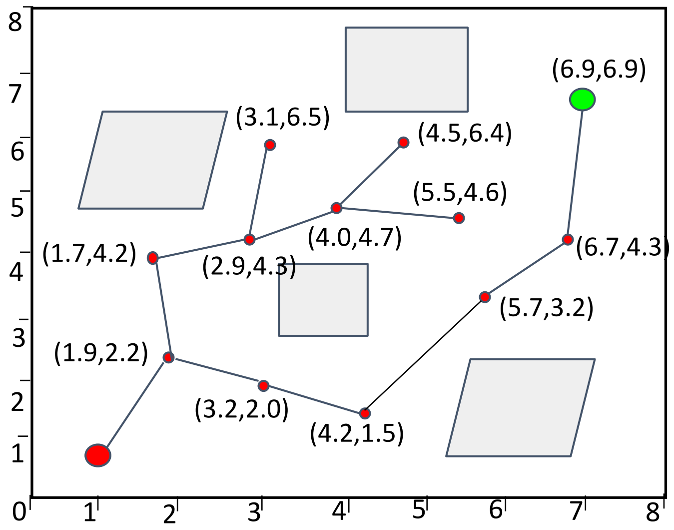
劣势

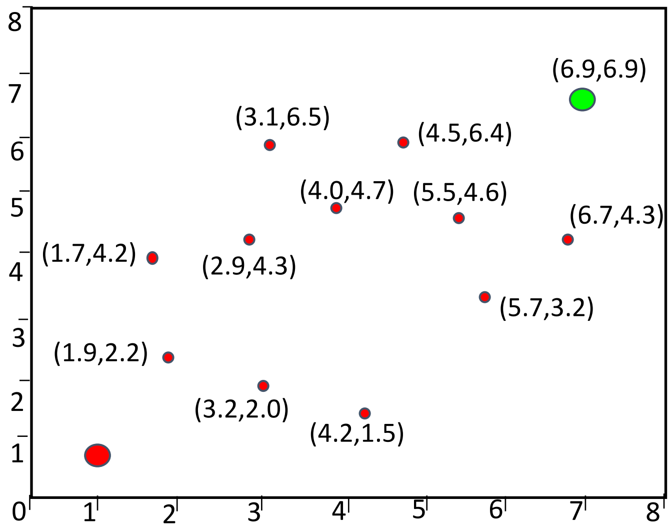
- 产生的路径非最优
- 不够高效



Note：效率提升方法

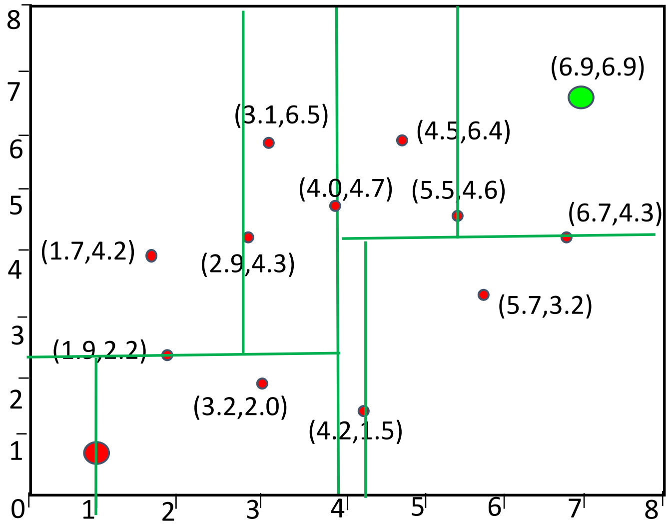
Kd-tree

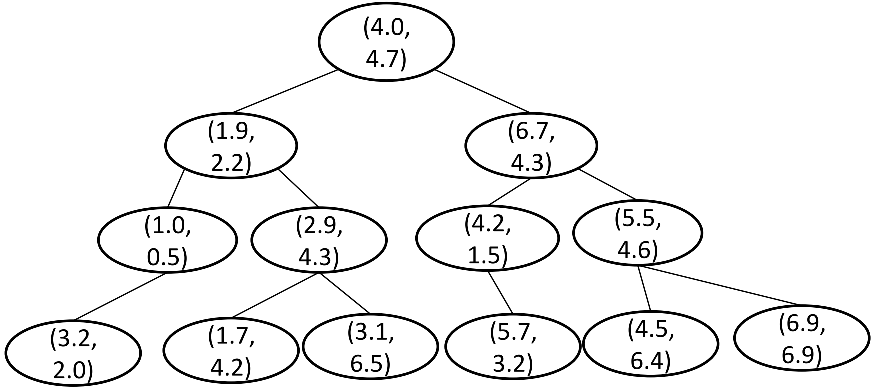




Note：效率提升方法

- Kd-tree

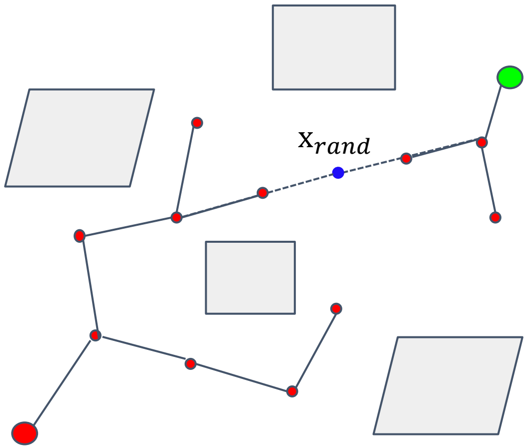




- Other variants: Spatial grid, hill climbing,etc

Note：效率提升方法

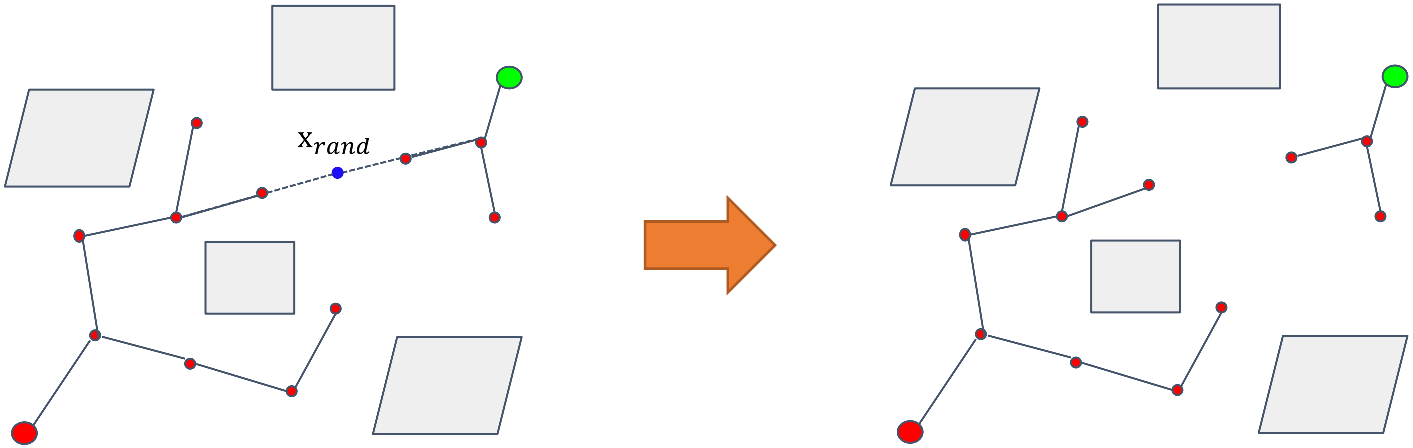
➢ 双向 RRT



- 从起点和目标点分别搜索树
- 查找连接两棵树的路径

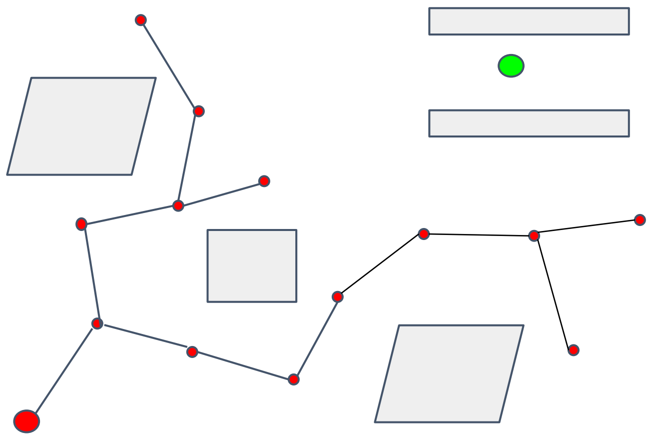
Note：效率提升方法

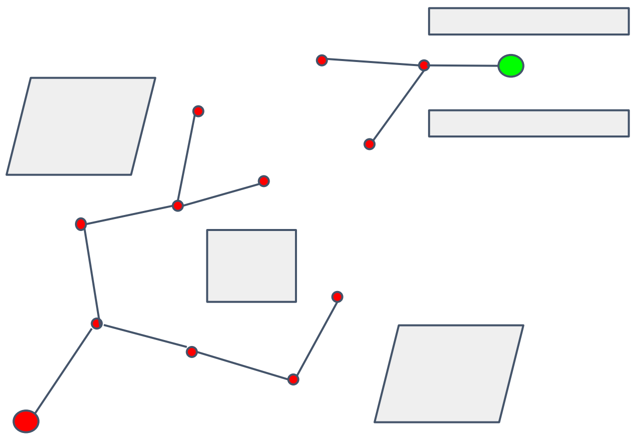
➢ 双向 RRT



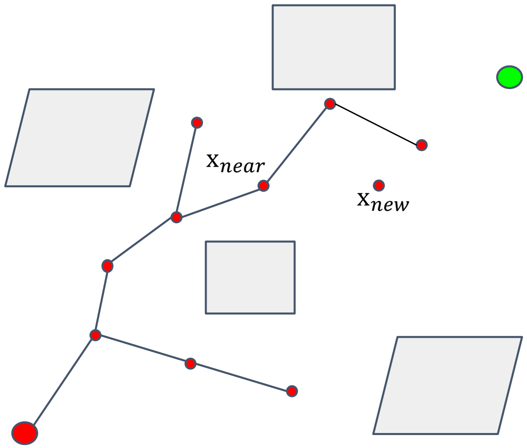
Note：效率提升方法

- 双向 RRT





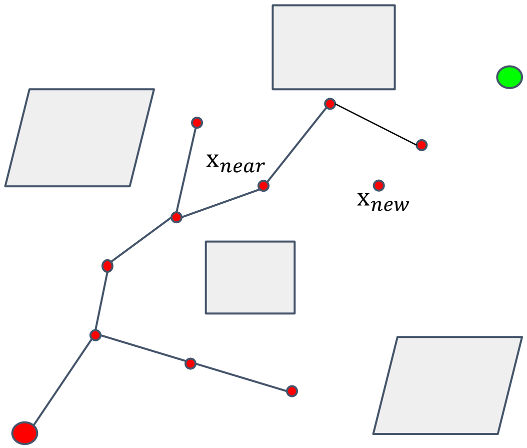
Optimal sampling-based path planning methods



RRT*是什么？

- 对RRT的改进
- 具备概率完备性和渐进最优性
- 通过在工作空间采样节点来构建一棵从起点到终点的树，随着采样增加，树从起点向终点生长，并且改进已有路径

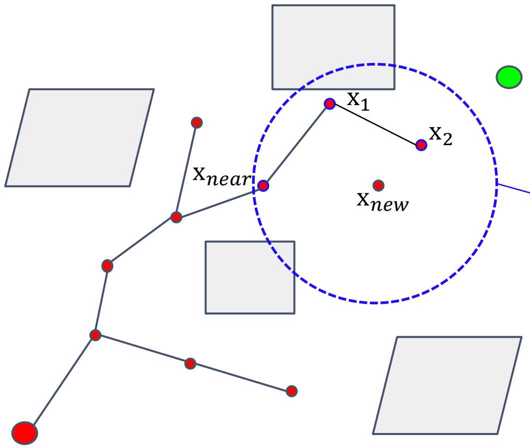
Rapidly-exploring Random Tree* （RRT * ）



Algorithm 2: RRT Algorithm

Input: $\mathcal{M}$,x_{init},x_{goal}
Result: A path Γ from x_{init} to x_{goal}
$\mathcal{T}$.init();
for $i = 1$ *to* n **do**
- $x_{rand} \leftarrow Sample(\mathcal{M})$;
- $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
- $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
- **if** *CollisionFree*(x_{new}) **then**
  - $X_{near} \leftarrow NearC(\mathcal{T}, x_{new})$;
  - $x_{min} \leftarrow ChooseParent(X_{near}, x_{near}, x_{new})$;
  - $\mathcal{T}.addNodEdge(x_{min}, x_{new})$;
  - $\mathcal{T}.rewire()$;

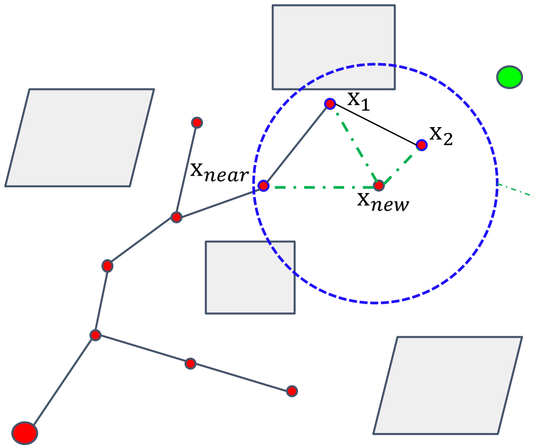
Rapidly-exploring Random Tree* （RRT * ）



Algorithm 2: RRT Algorithm

Input: $\mathcal{M}$,x_{init},x_{goal}
Result: A path Γ from x_{init} to x_{goal}
$\mathcal{T}$.init();
for $i = 1$ *to* n **do**
  $x_{rand} \leftarrow Sample(\mathcal{M})$;
  $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
  $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
  if *CollisionFree*(x_{new}) **then**
    $X_{near} \leftarrow NearC(\mathcal{T}, x_{new})$;
    $x_{min} \leftarrow ChooseParent(X_{near}, x_{near}, x_{new})$;
    $\mathcal{T}.addNodEdge(x_{min}, x_{new})$;
    $\mathcal{T}.rewire()$;

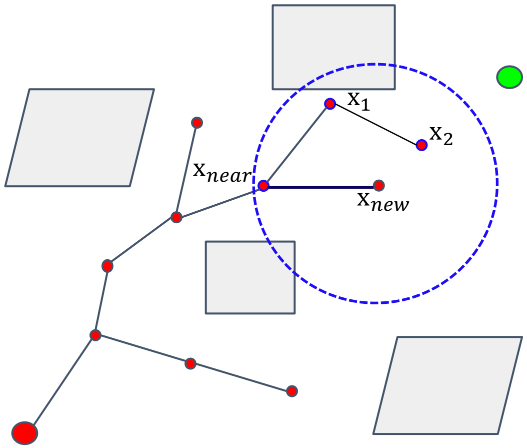
Rapidly-exploring Random Tree*（RRT * ）



Algorithm 2: RRT Algorithm

Input: $\mathcal{M}$,x_{init},x_{goal}
Result: A path Γ from x_{init} to x_{goal}
$\mathcal{T}$.init();
for $i = 1$ *to* n **do**
  $x_{rand} \leftarrow Sample(\mathcal{M})$;
  $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
  $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
  if *CollisionFree*(x_{new}) **then**
    $X_{near} \leftarrow NearC(\mathcal{T}, x_{new})$;
    $x_{min} \leftarrow ChooseParent(X_{near}, x_{near}, x_{new})$;
    $\mathcal{T}.addNodEdge(x_{min}, x_{new})$;
    $\mathcal{T}.rewire()$;

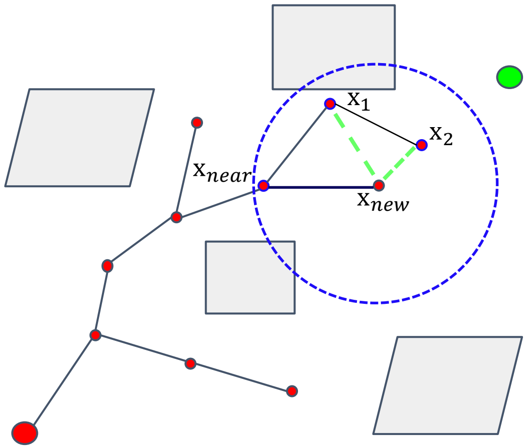
Rapidly-exploring Random Tree* （RRT * ）



Algorithm 2: RRT Algorithm

Input: $\mathcal{M}$,x_{init},x_{goal}
Result: A path Γ from x_{init} to x_{goal}
$\mathcal{T}$.init();
for $i = 1$ *to* n **do**
  $x_{rand} \leftarrow Sample(\mathcal{M})$;
  $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
  $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
  if *CollisionFree*(x_{new}) **then**
    $X_{near} \leftarrow NearC(\mathcal{T}, x_{new})$;
    $x_{min} \leftarrow ChooseParent(X_{near}, x_{near}, x_{new})$;
    $\mathcal{T}.addNodEdge(x_{min}, x_{new})$;
    $\mathcal{T}.rewire()$;

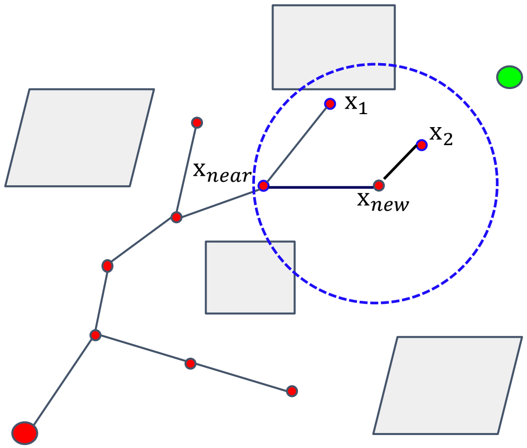
Rapidly-exploring Random Tree* （RRT * ）



Algorithm 2: RRT Algorithm

Input: $\mathcal{M}, x_{init}, x_{goal}$
Result: A path Γ from x_{init} to x_{goal}
$\mathcal{T}$.init();
for $i = 1$ *to* n **do**
  $x_{rand} \leftarrow Sample(\mathcal{M})$;
  $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
  $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
  if *CollisionFree*(x_{new}) **then**
    $X_{near} \leftarrow NearC(\mathcal{T}, x_{new})$;
    $x_{min} \leftarrow ChooseParent(X_{near}, x_{near}, x_{new})$;
    $\mathcal{T}.addNodEdge(x_{min}, x_{new})$;
    $\mathcal{T}.rewire()$;

Rapidly-exploring Random Tree* （RRT * ）



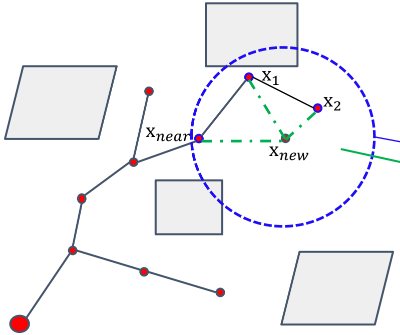
Algorithm 2: RRT Algorithm

Input: $\mathcal{M}$,x_{init},x_{goal}
Result: A path Γ from x_{init} to x_{goal}
$\mathcal{T}$.init();
for $i = 1$ *to* n **do**
  $x_{rand} \leftarrow Sample(\mathcal{M})$;
  $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
  $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
  if *CollisionFree*(x_{new}) **then**
    $X_{near} \leftarrow NearC(\mathcal{T}, x_{new})$;
    $x_{min} \leftarrow ChooseParent(X_{near}, x_{near}, x_{new})$;
    $\mathcal{T}.addNodEdge(x_{min}, x_{new})$;
    $\mathcal{T}.rewire()$;

Rapidly-exploring Random Tree*（RRT*）

改进之处

- 考虑邻近节点的历史路径长度，而非只是局部路径长度
- 选择父节点时考虑多个邻近节点



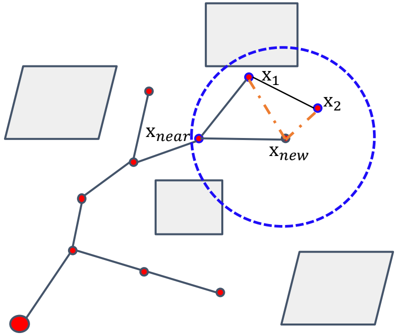
Algorithm 2: RRT*Algorithm

Input: $\mathcal{M}$,x_{init},x_{goal}
Result: A path Γ from x_{init} to x_{goal}
$\mathcal{T}$.init();
for $i = 1$ *to* n **do**
  $x_{rand} \leftarrow Sample(\mathcal{M})$;
  $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
  $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
  if $CollisionFree(x_{new})$ **then**
    $X_{near} \leftarrow NearC(\mathcal{T}, x_{new})$;
    $x_{min} \leftarrow ChooseParent(X_{near}, x_{near}, x_{new})$;
    $\mathcal{T}.addNodEdge(x_{min}, x_{new})$;
    $\mathcal{T}.rewire()$;

Rapidly-exploring Random Tree* （RRT * ）

改进之处

- 考虑邻近节点的历史路径长度，而非只是局部路径长度
- 重连接操作，改进局部最优解



Algorithm 2: RRT *Algorithm

Input: $\mathcal{M}$,x_{init},x_{goal}

Result: A path Γ from x_{init} to x_{goal}

$\mathcal{T}$.init();

for $i = 1$ *to* n **do**

  $x_{rand} \leftarrow Sample(\mathcal{M})$;

  $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;

  $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;

  if *CollisionFree*(x_{new}) **then**

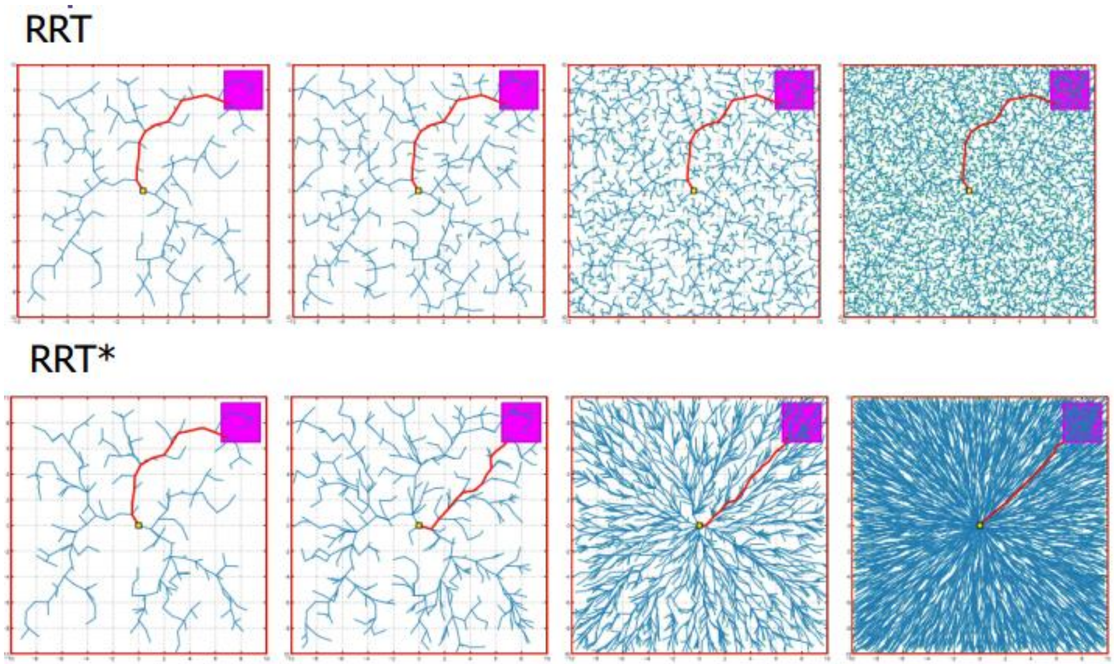
    $X_{near} \leftarrow NearC(\mathcal{T}, x_{new})$;

    $x_{min} \leftarrow ChooseParent(X_{near}, x_{near}, x_{new})$;

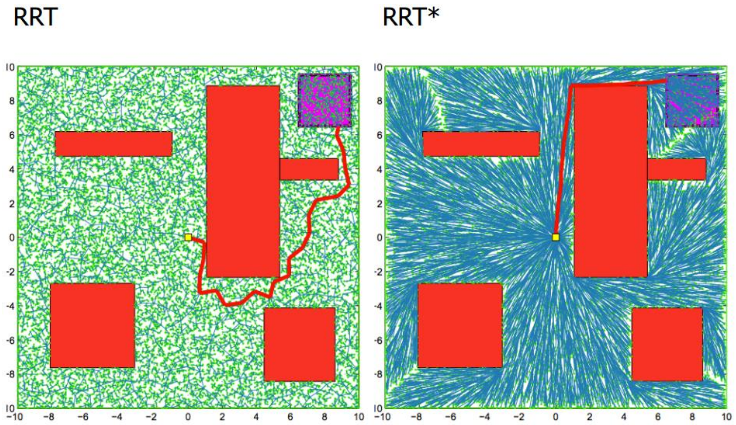
    $\mathcal{T}.addNodEdge(x_{min}, x_{new})$;

    $\mathcal{T}.rewire()$;

RRT vs RRT*

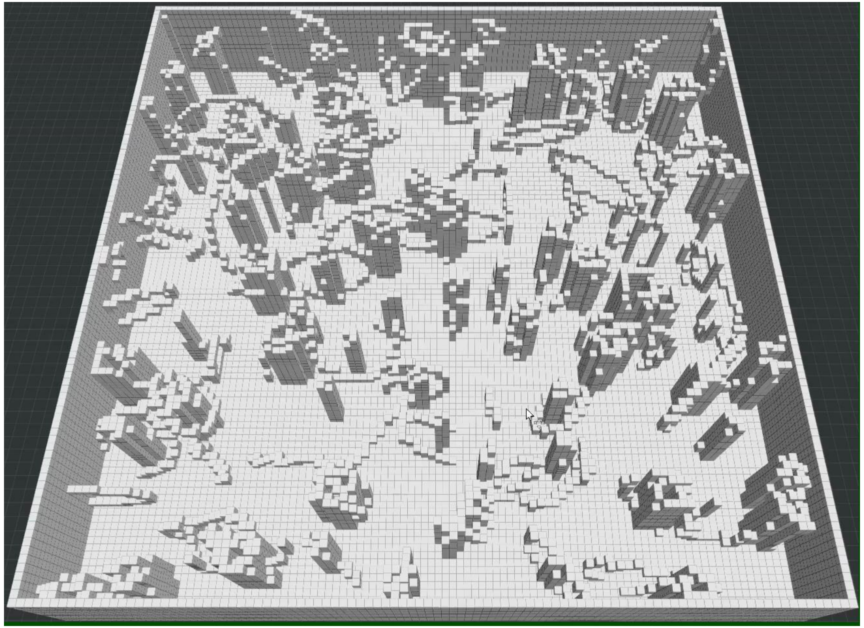


Source: Karaman and Frazzoli

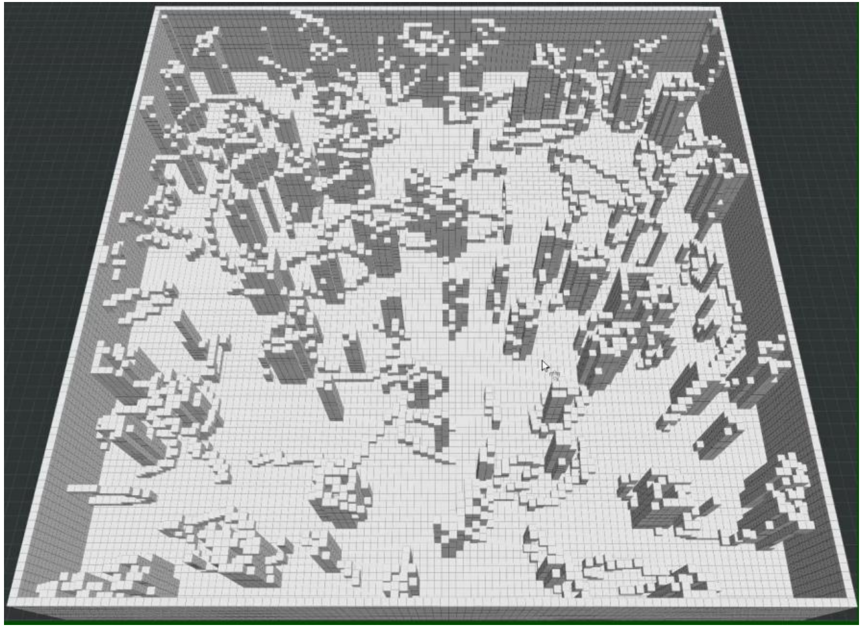


RRT vs RRT*

RRT:



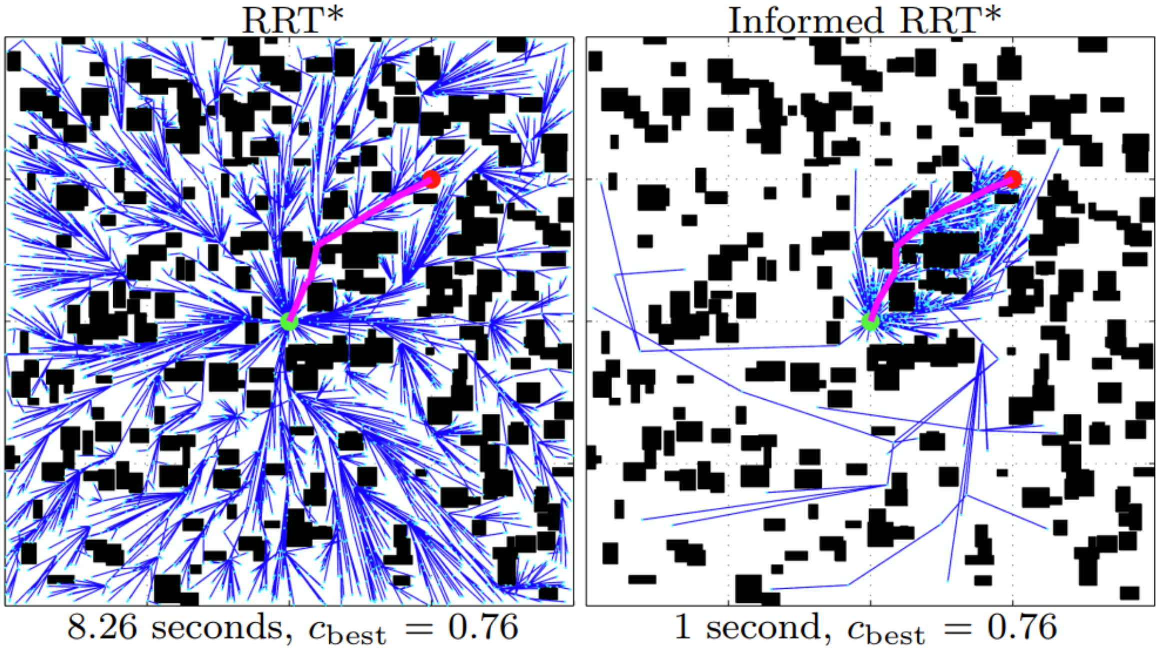
RRT*:



https://github.com/ZJU-FAST-Lab/sampling-based-path-finding

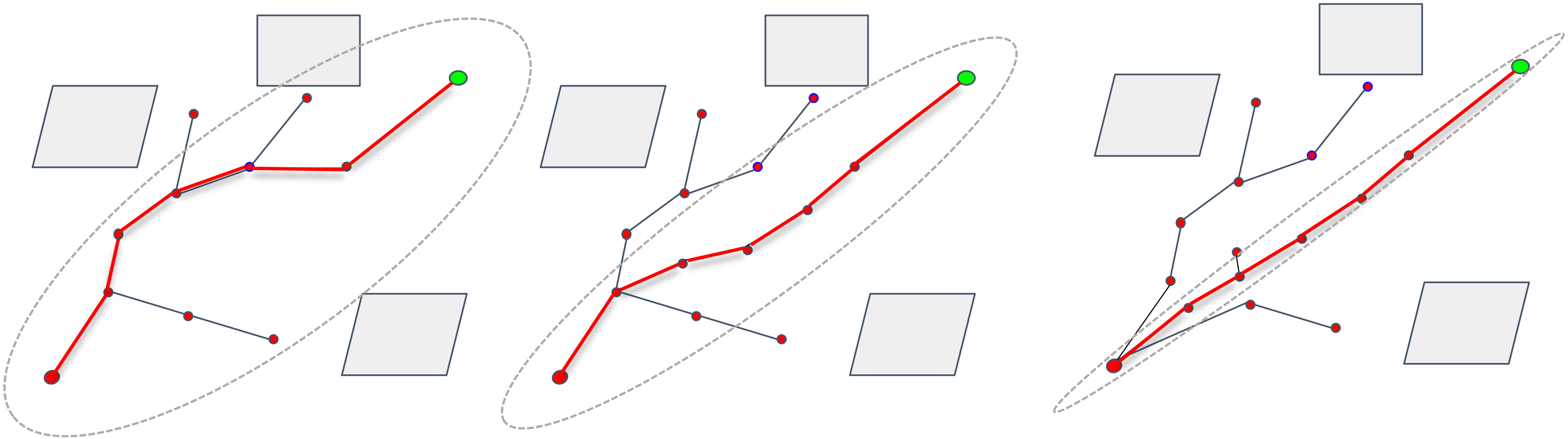
Advanced Sampling-based Methods

Informed RRT*

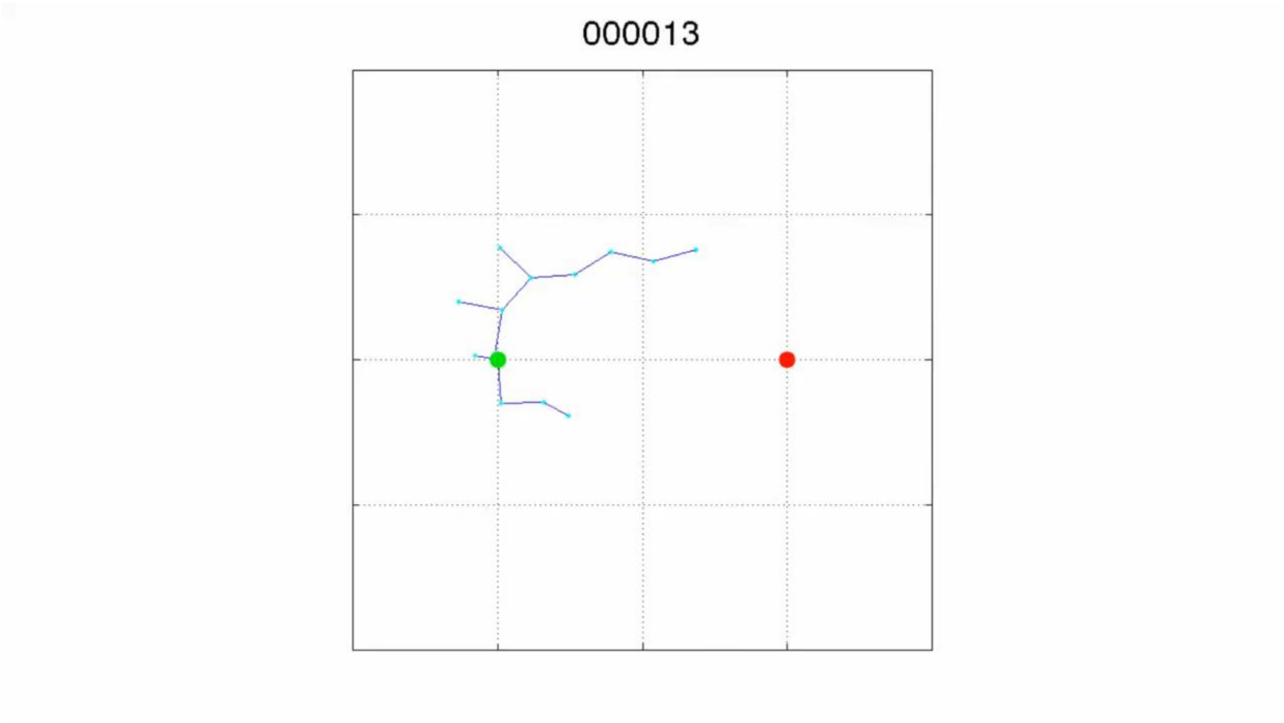


Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic

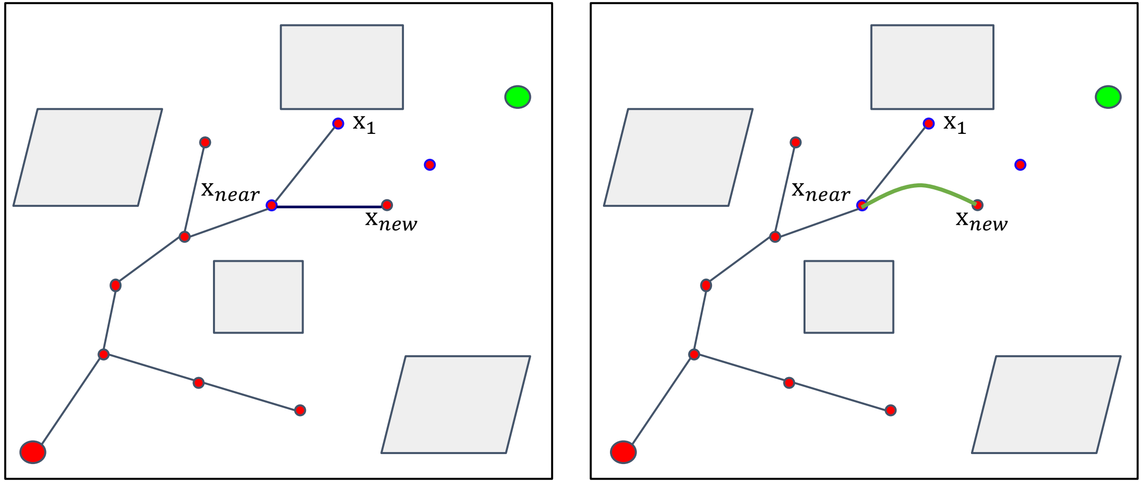
Informed RRT*



Informed RRT*



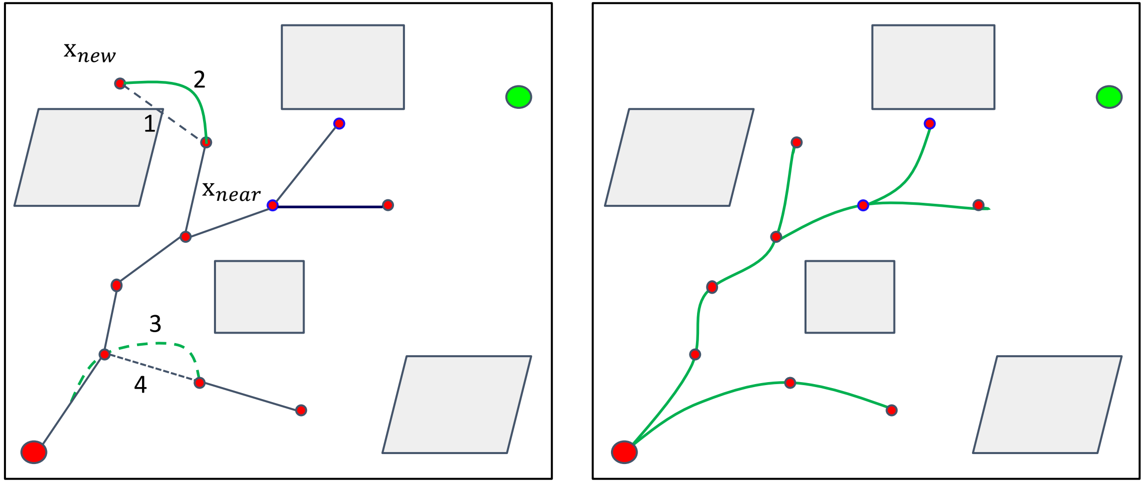
Kinodynamic-RRT*



更改函数以适应机器人导航中的运动或其他约束。

Kinodynamic RRT*: Optimal Motion Planning for Systems with Linear Differential Constraints

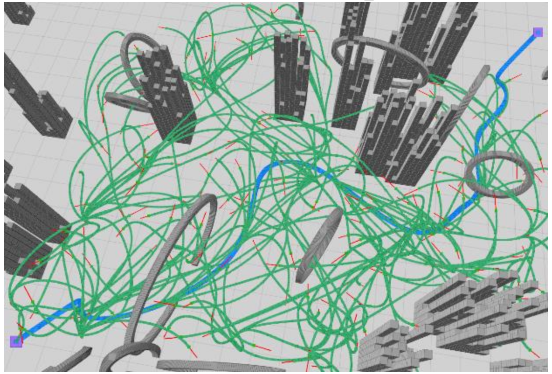
Kinodynamic-RRT*



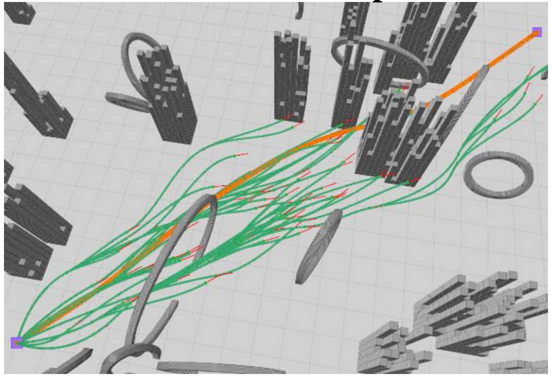
更改函数以适应机器人导航中的运动或其他约束。

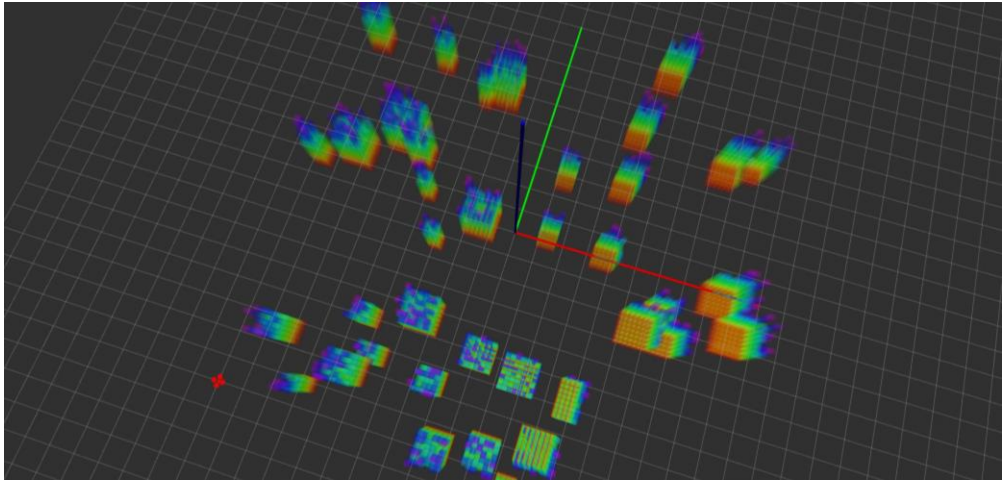
Smart Kinodynamic RRT*

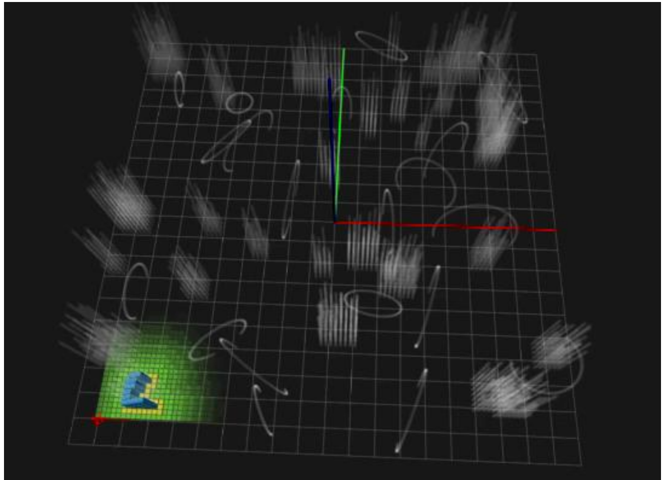
Random Sample



Guided Sample

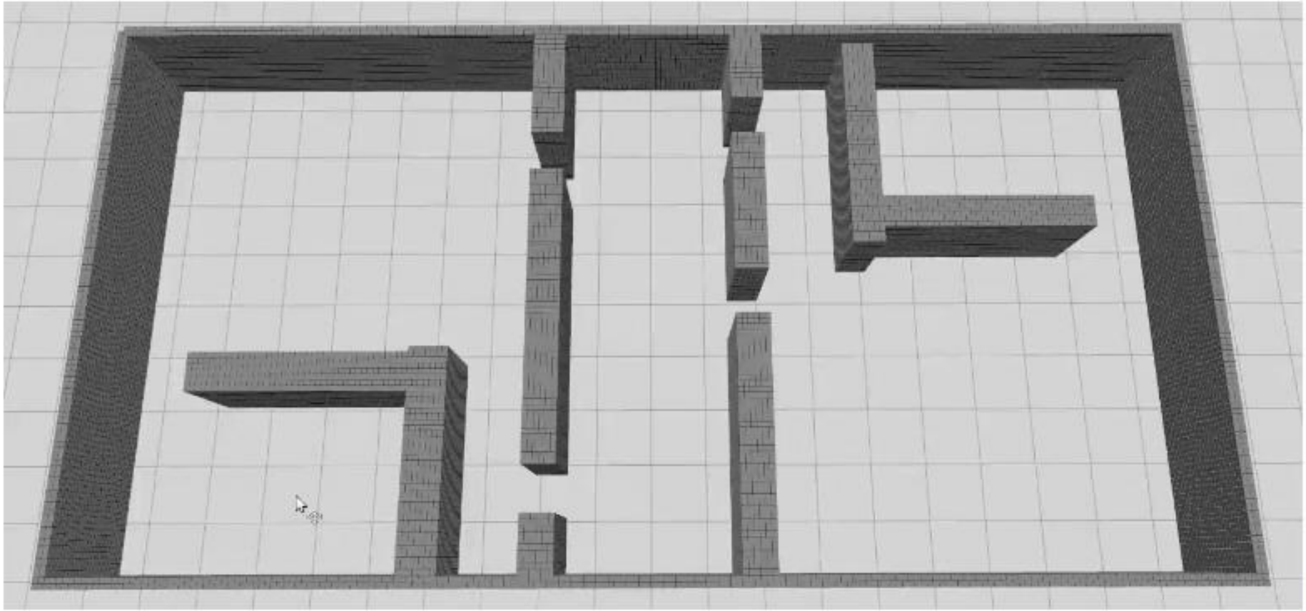






Smart Kinodynamic RRT*

bidirectional sampling

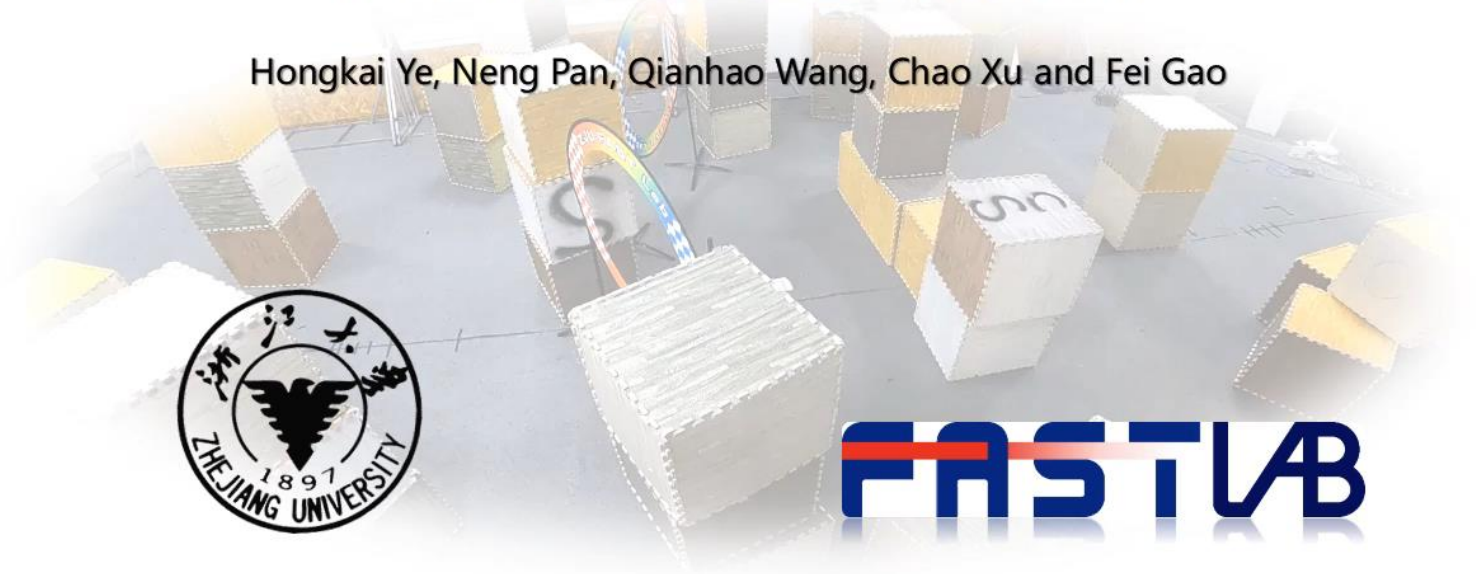


0.5x speed

The bidirectional tree growing process.

Efficient Sampling-based Kinodynamic Planning with Regional Optimization and Bidirectional Search for Multirotors

Hongkai Ye, Neng Pan, Qianhao Wang, Chao Xu and Fei Gao



STD-Trees: Spatio-temporal Deformable Trees for Multirotors Kinodynamic Planning

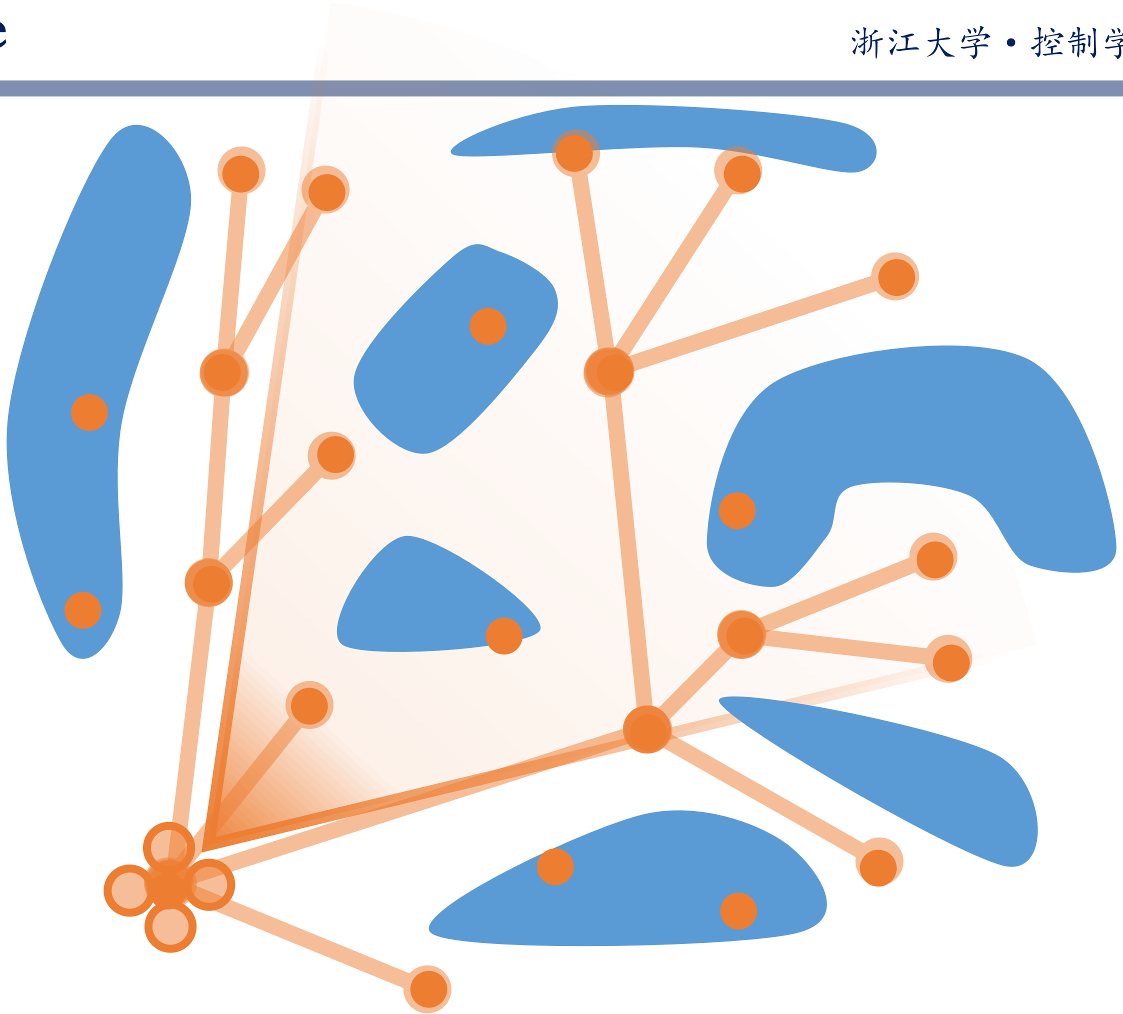
Hongkai Ye, Chao Xu and Fei Gao

Forward Spanning Tree

→采样

- 采样
- 碰撞检测
- 从近到远连接节点

  → 无需重连接



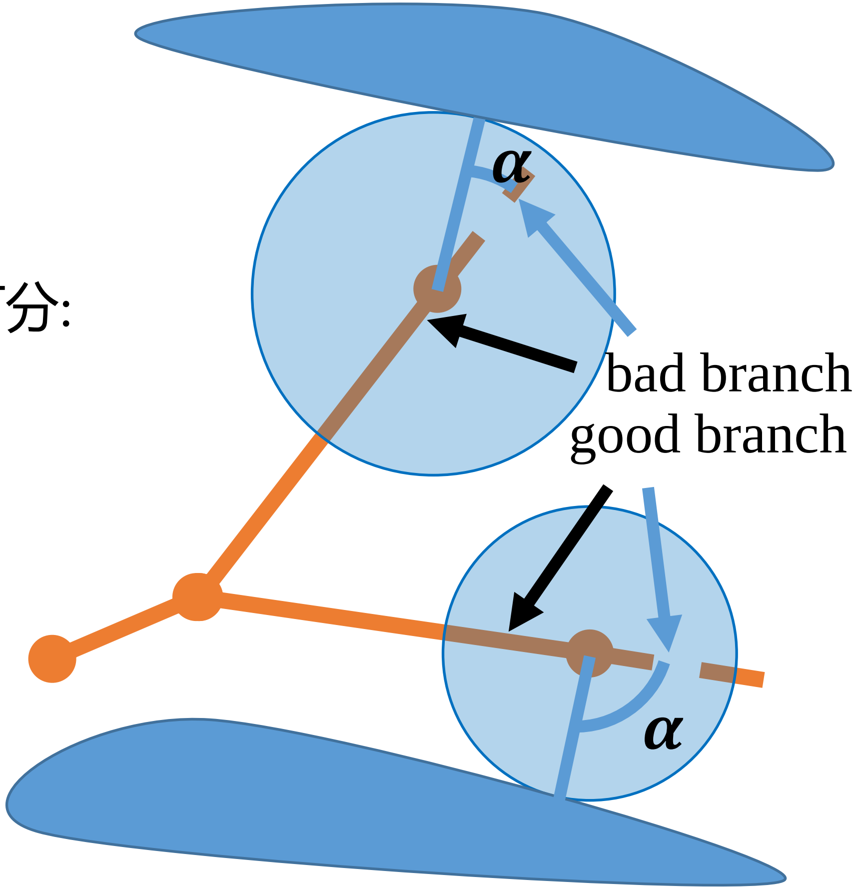
→打分

- 对子节点依据角度α进行0-1打分:

  1表示延伸到自由空间

  0 表示将会产生碰撞

- 计算父节点得分



Forward Spanning Tree

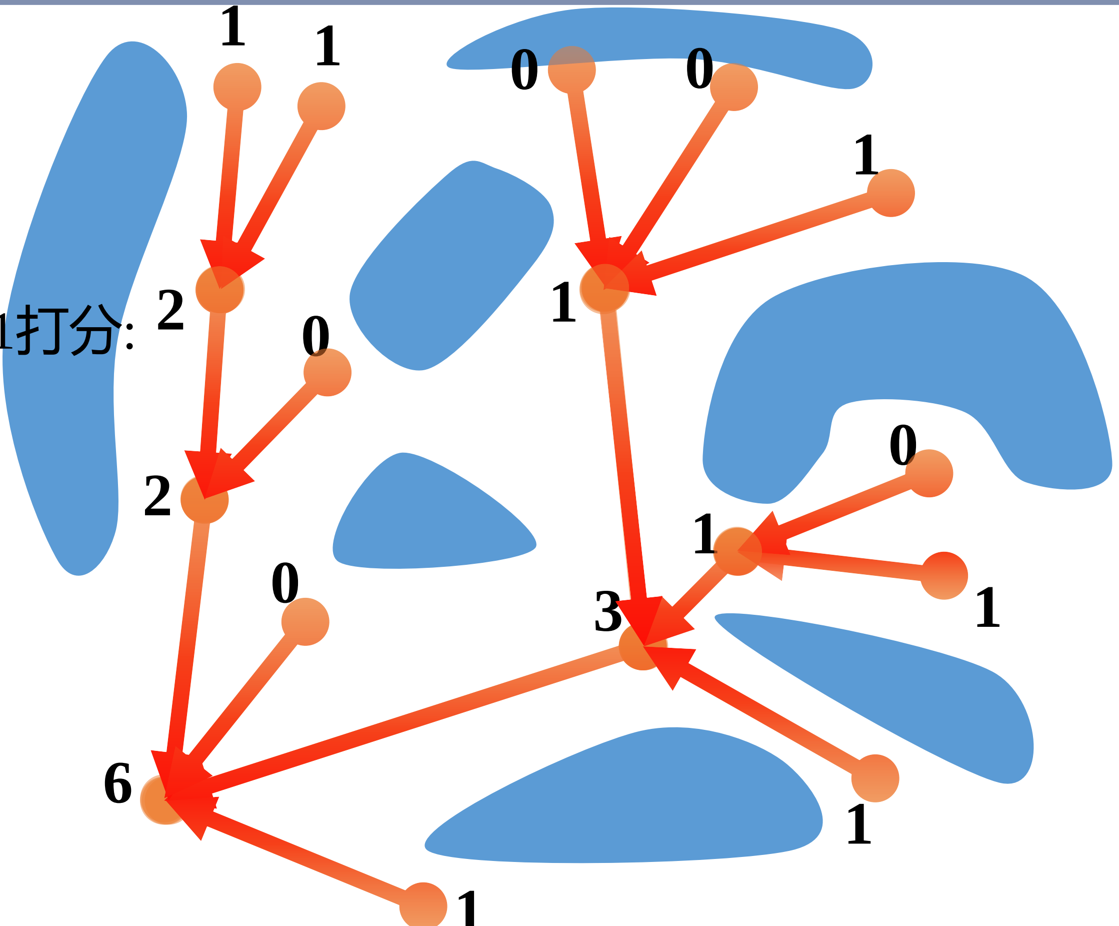
→打分

- 对子节点依据角度α进行0-1打分:

  1表示延伸到自由空间

  0 表示将会产生碰撞

- 计算父节点得分



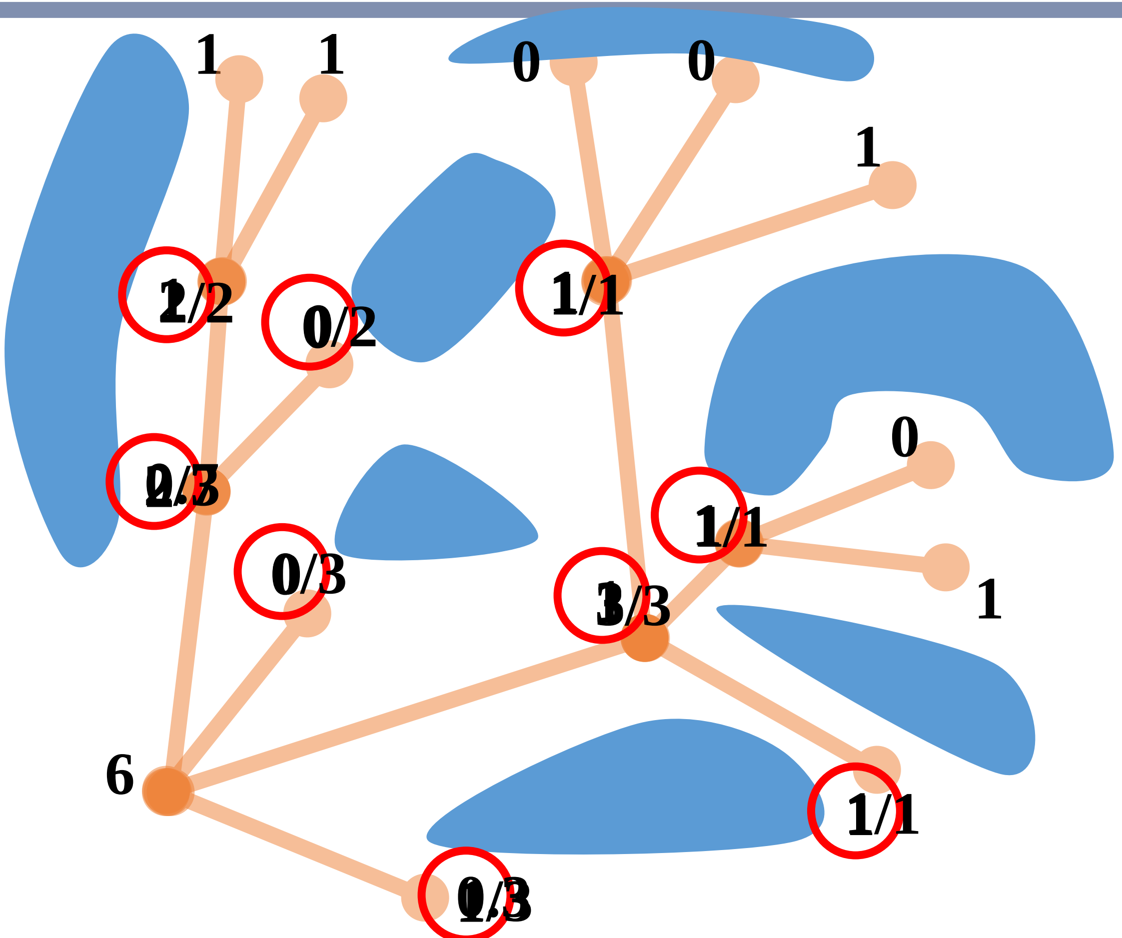
Forward Spanning Tree

→修剪

修剪权重=
其得分÷其兄弟节点的最大

↓ 广度优先遍历

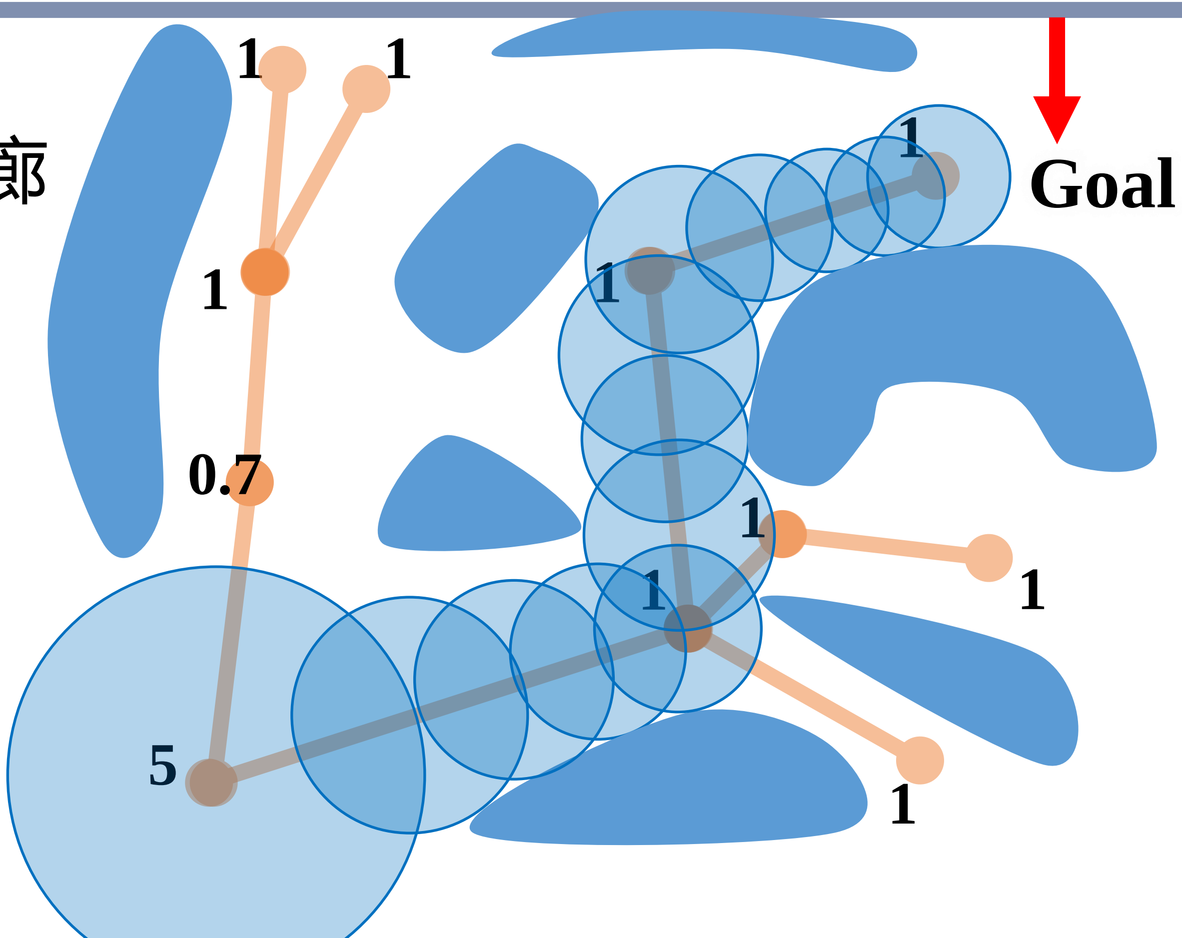
提取无碰撞空间框架!



Forward Spanning Tree

→生成安全飞行走廊

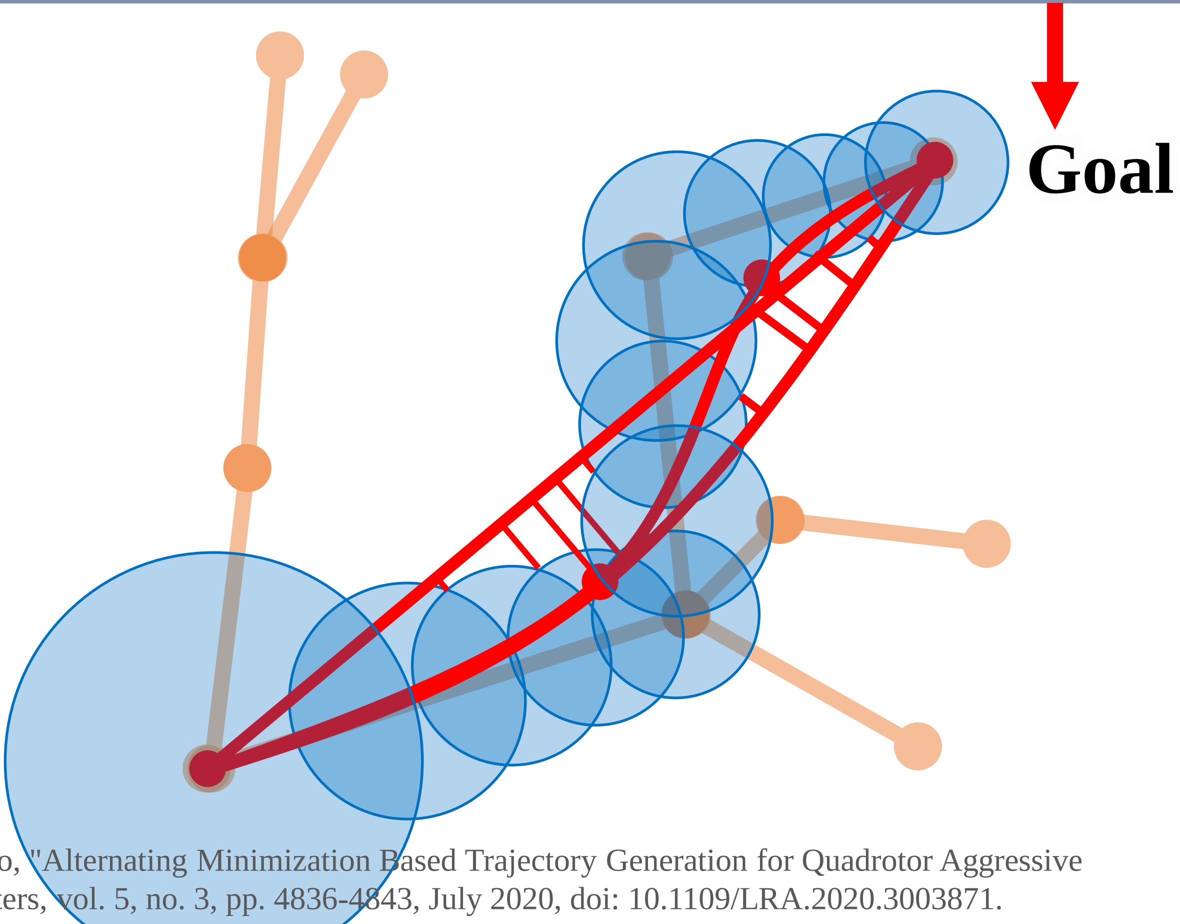
- 找到最短的分支
- 生成球形安全飞行走廊



Forward Spanning Tree

→轨迹优化

- 迭代路径点插入
- 迭代优化 [1]



[1] Z. Wang, X. Zhou, C. Xu, J. Chu and F. Gao, "Alternating Minimization Based Trajectory Generation for Quadrotor Aggressive Flight," in IEEE Robotics and Automation Letters, vol. 5, no. 3, pp. 4836-4843, July 2020, doi: 10.1109/LRA.2020.3003871.


Outdoor Experiment
Trajectory
FST
SFC
Goal changed!
Goal
Goal
1

Thanks for Listening!

Flying Autonomous Robotics（FAR）

FASTLAB
Field Autonomous System & compuTing