



## 第四讲 路径规划2

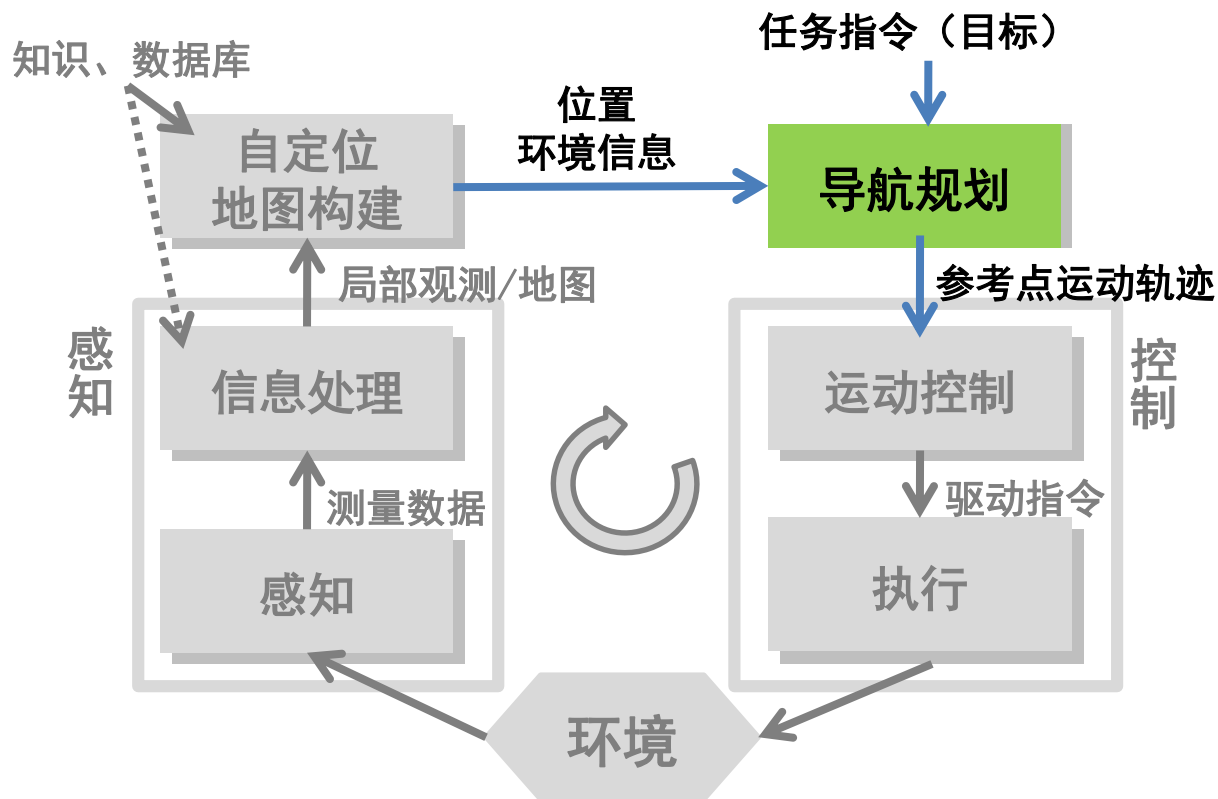
王越

浙江大学 控制科学与工程学院

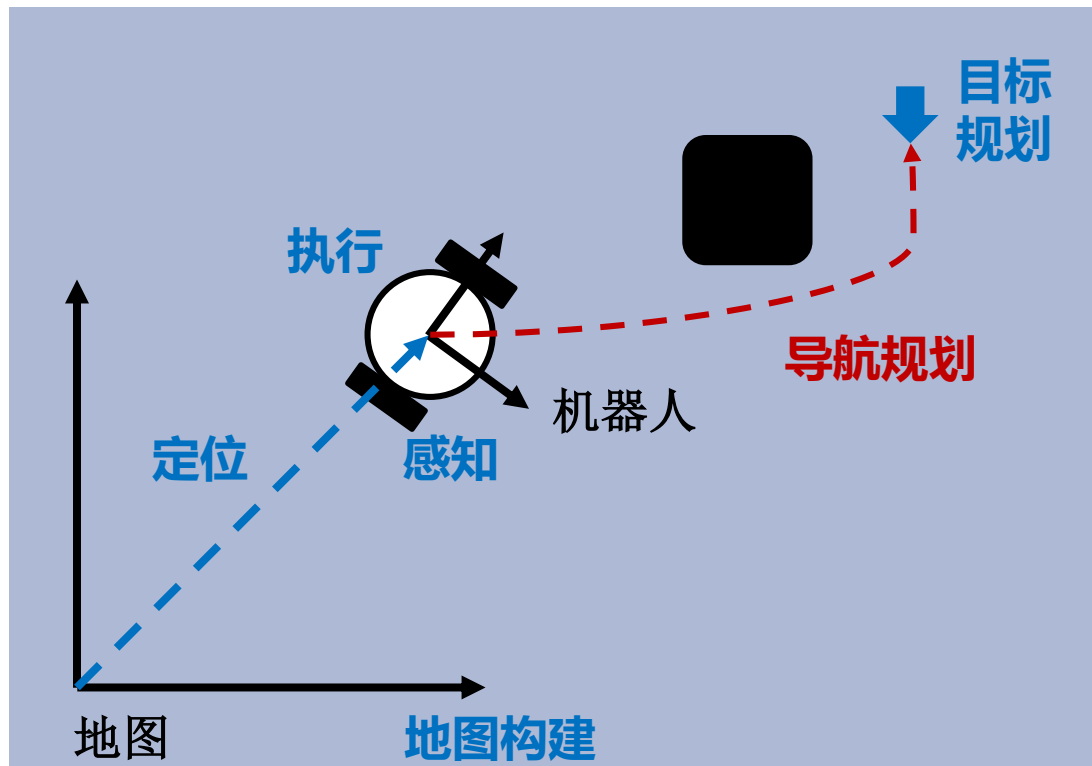


## 导航规划简介

# 自主移动机器人一般架构



# 导航规划



在给定环境的全局或局部知识以及一个或者一系列目标位置的情况下,使机器人能够根据知识和传感器感知信息高效可靠地到达目标位置

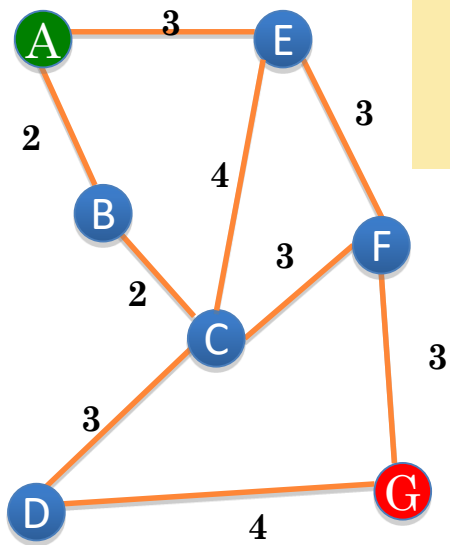


# DIJKSTRA算法

- 优先级：深度搜索（先进后出），广度搜索（先进先出）
  - 其他优先级形式？
- 设置两个集合S和T，S存放已经找到最短路径的顶点，T是尚未确定路径的顶点集合，同时也描述了起始点经过集合S中顶点到该点的最短路径及长度。
  - 每次更新时，从T中找出路径最短的点加入到集合S，T中顶点最短路径及长度则根据加入点作为中间点后起始点到该点距离是否减小来决定是否更新



# DIJKSTRA算法



- 设置两个集合S和T，S存放已经找到最短路径的顶点，T是尚未确定路径的顶点集合，同时也描述了起始点经过集合S中顶点到该点的最短路径及长度。
- 每次更新时，从T中找出路径最短的点加入到集合S，T中顶点最短路径及长度则根据加入点作为中间点后起始点到该点距离是否减小来决定是否更新

$$S=\{A \rightarrow A=0\}$$

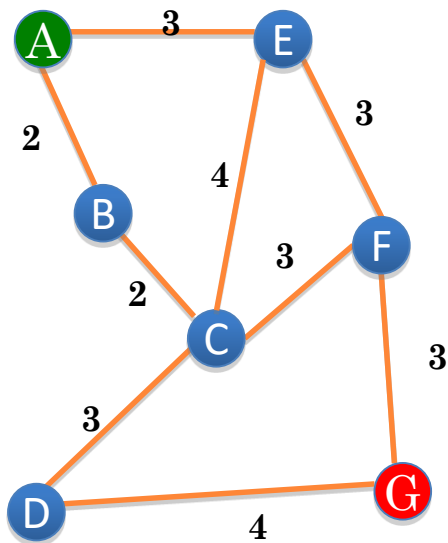
$$T=\{A \rightarrow B=2, A \rightarrow C=\infty, A \rightarrow D=\infty, A \rightarrow E=3, A \rightarrow F=\infty, A \rightarrow G=\infty\}$$



$$S=\{A \rightarrow A=0, A \rightarrow B=2\}$$

$$T=\{A \rightarrow C=4(\text{路径为ABC}), A \rightarrow D=\infty, A \rightarrow E=3, A \rightarrow F=\infty, A \rightarrow G=\infty\}$$

# DIJKSTRA算法



问题规模大时搜索耗时

↓

$S=\{A \rightarrow A=0, A \rightarrow B=2, A \rightarrow E=3\}$

$T=\{A \rightarrow C=4(\text{路径为ABC}), A \rightarrow D=\infty, A \rightarrow F=6(\text{路径为AEF}), A \rightarrow G=\infty\}$

↓

$S=\{A \rightarrow A=0, A \rightarrow B=2, A \rightarrow E=3, A \rightarrow C=4(\text{路径为ABC})\}$

$T=\{A \rightarrow D=7(\text{路径为ABCD}), A \rightarrow F=6(\text{路径为AEF}), A \rightarrow G=\infty\}$



.....



# 启发式搜索算法：A\*

- 基于优先级定义的广度优先搜索
- 根据**启发式评估函数**在连通图中寻找最优路径
  - 当选择下一个探索结点时，通过启发式评估函数进行评估，选择路径代价最小的结点作为下一步探索结点而跳转其上

评估函数  $f(n) = g(n) + h(n)$

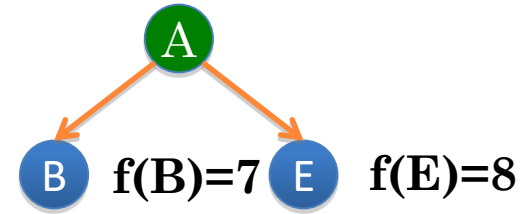
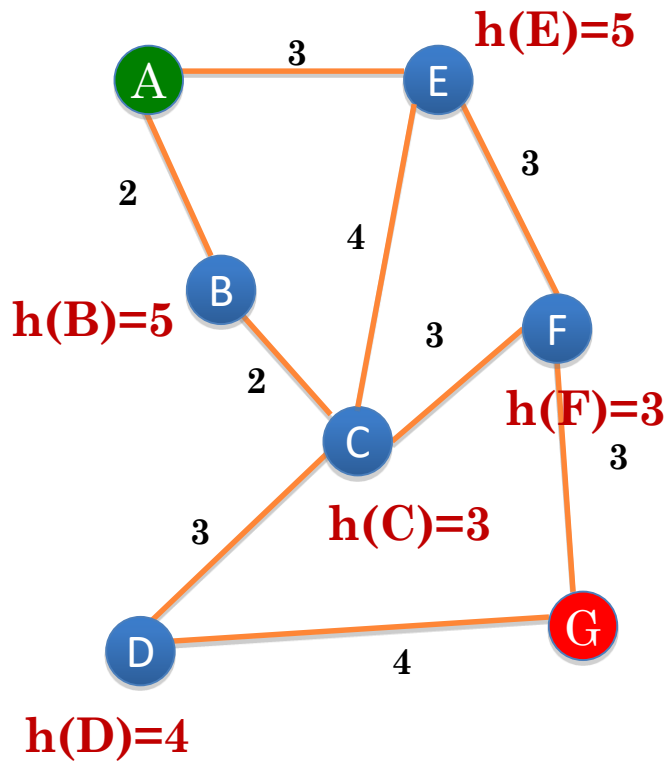
n 表示节点

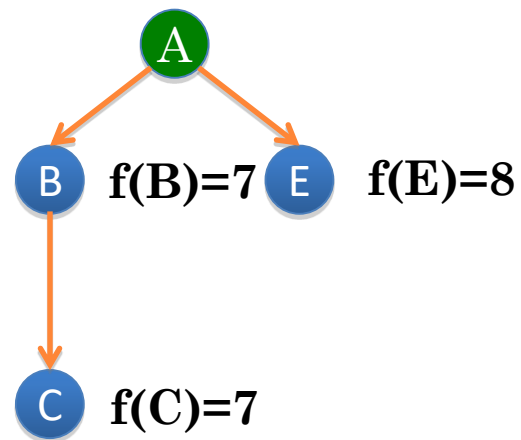
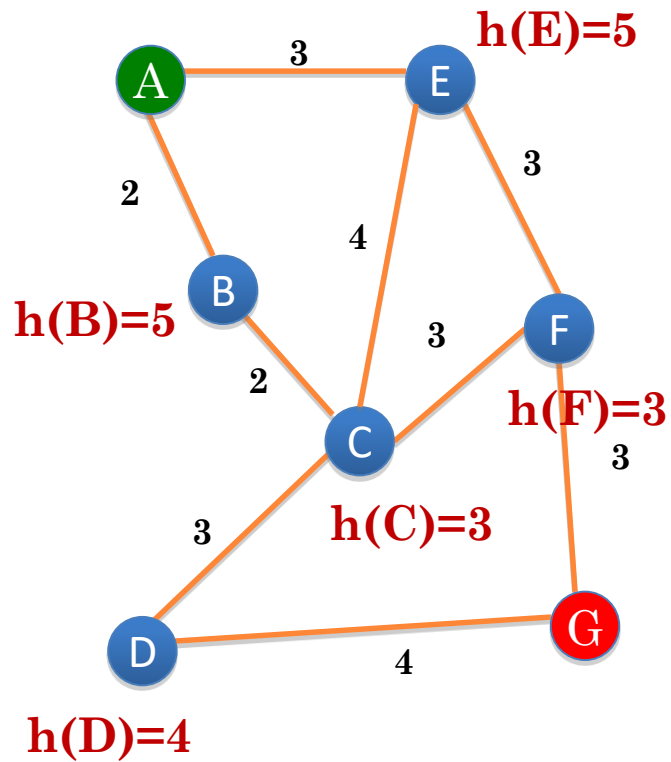
$g(n)$  表示从起始点到节点 的实际代价

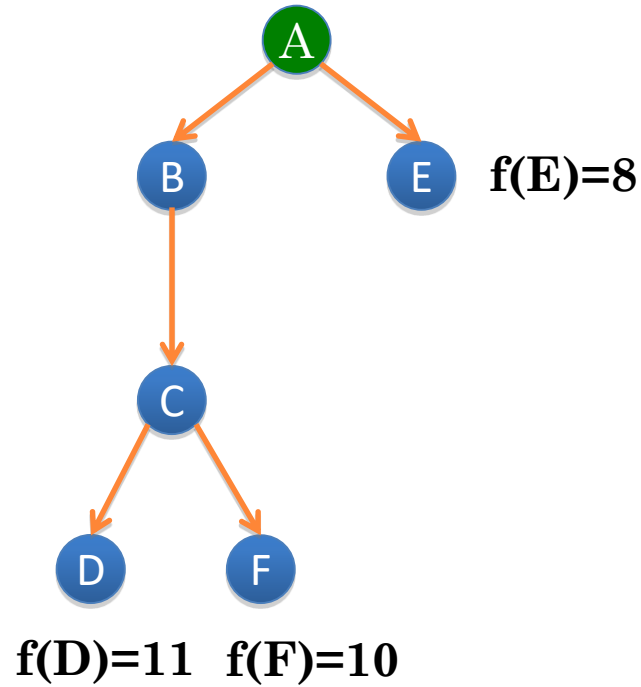
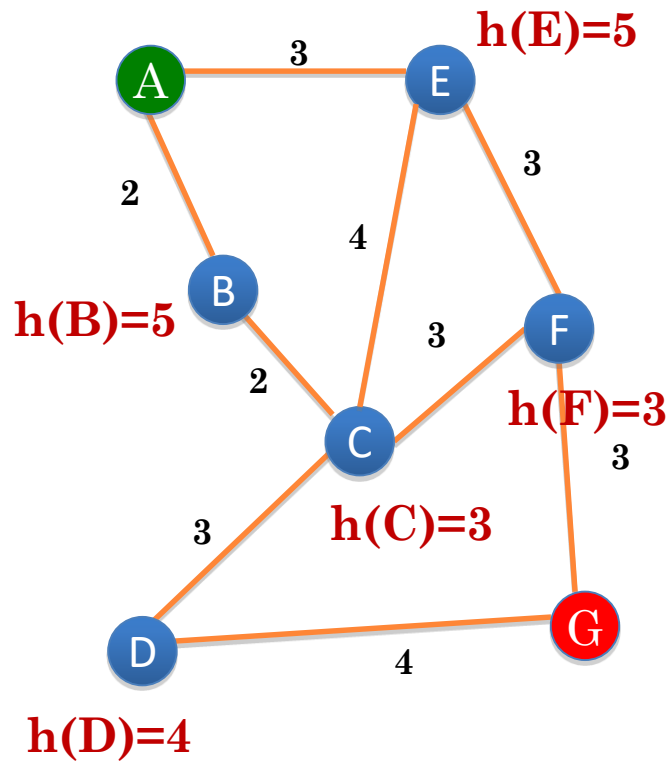
$h(n)$  为从节点 到目标点的最佳路径的估计代价

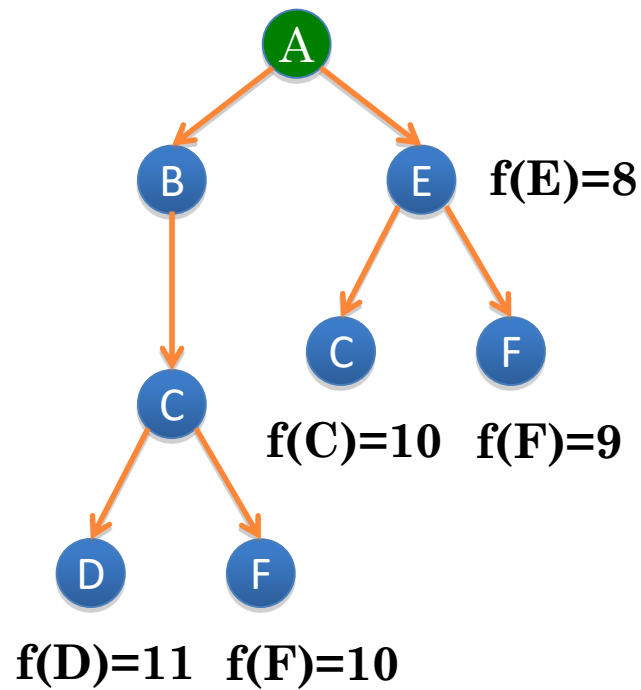
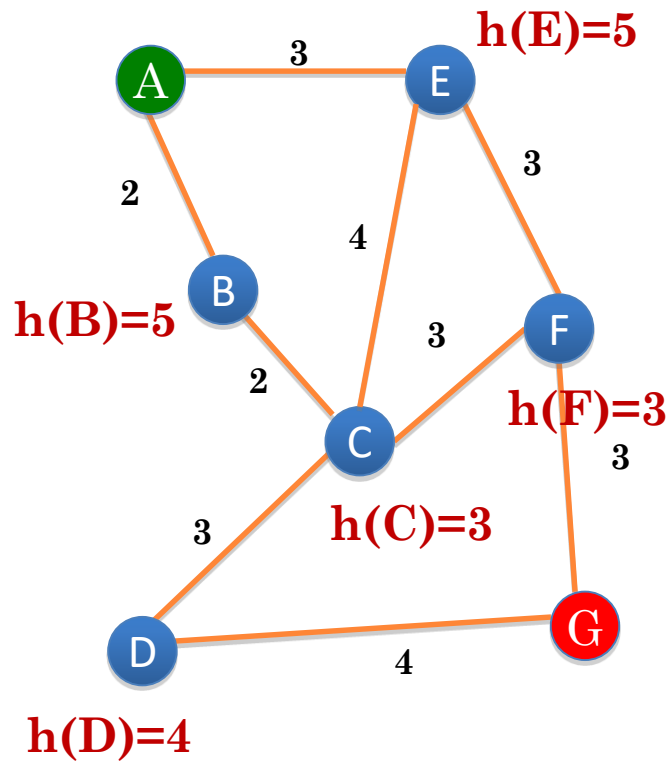


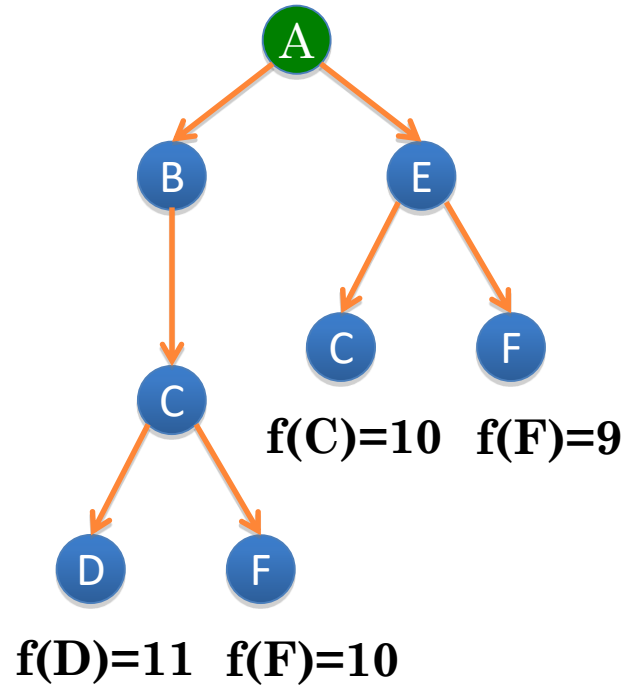
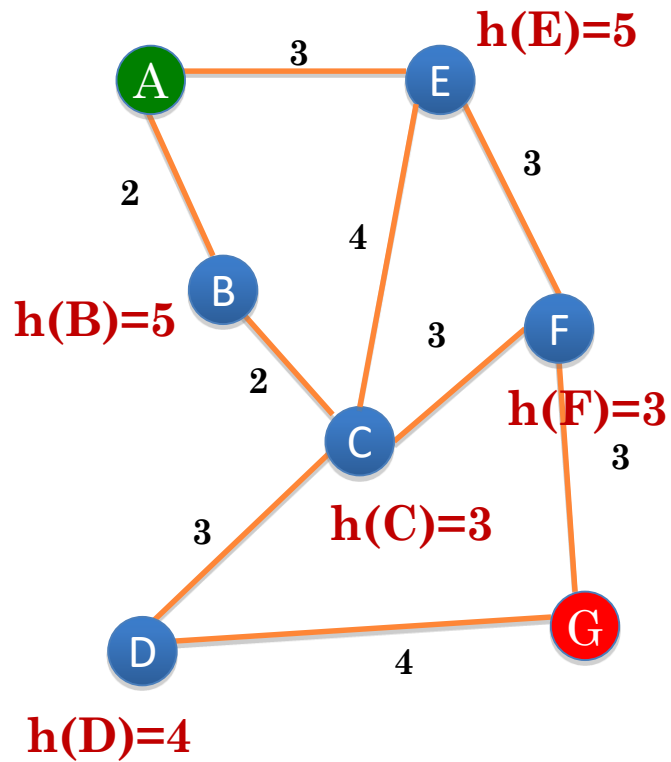


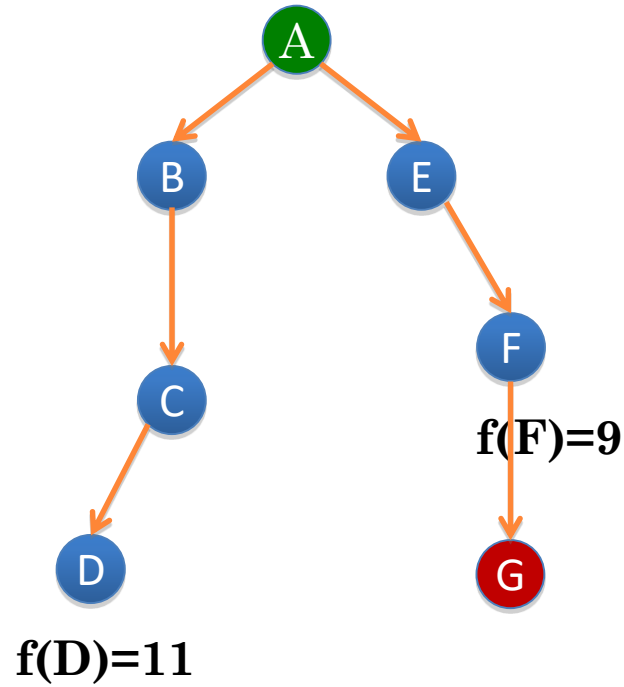
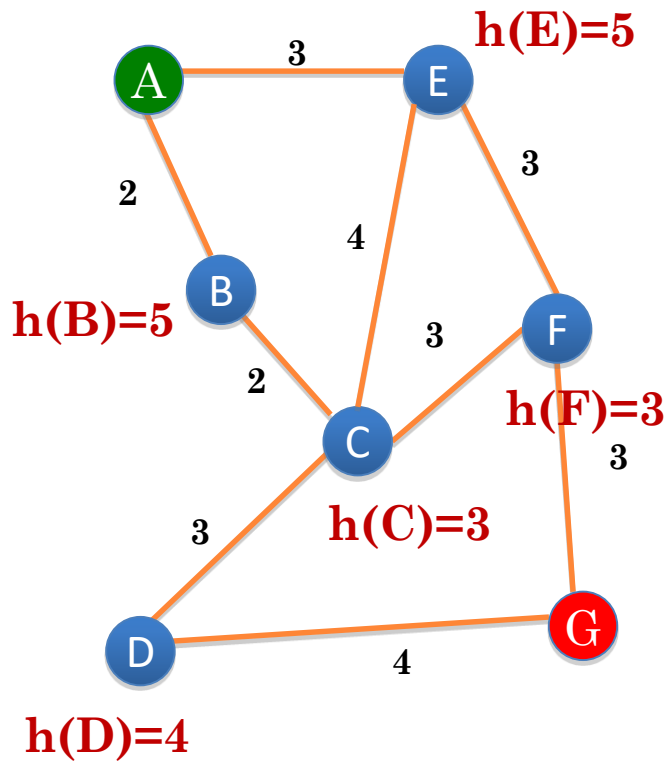


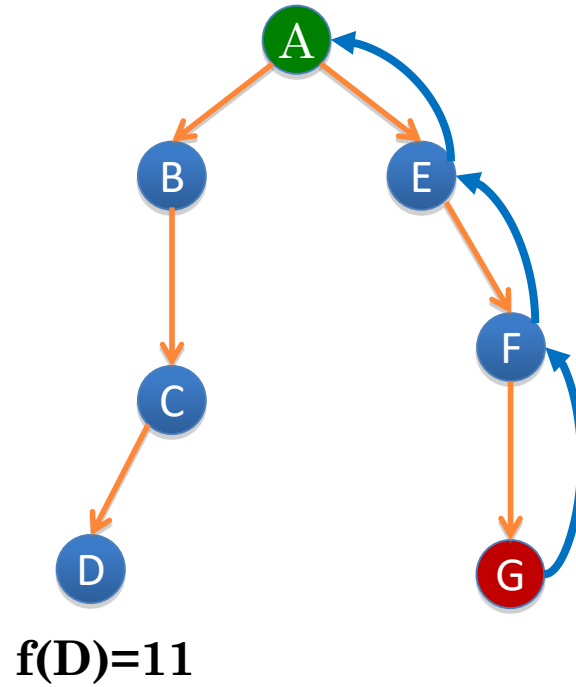
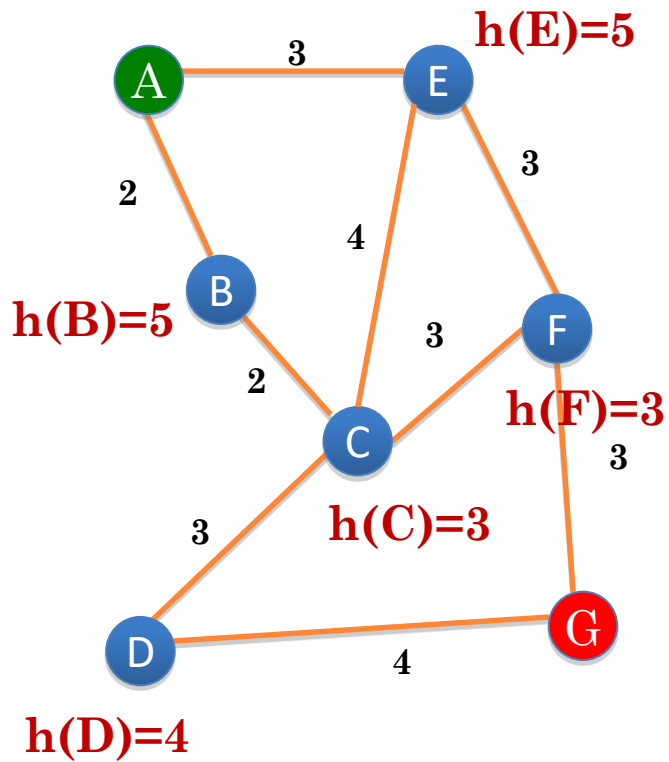










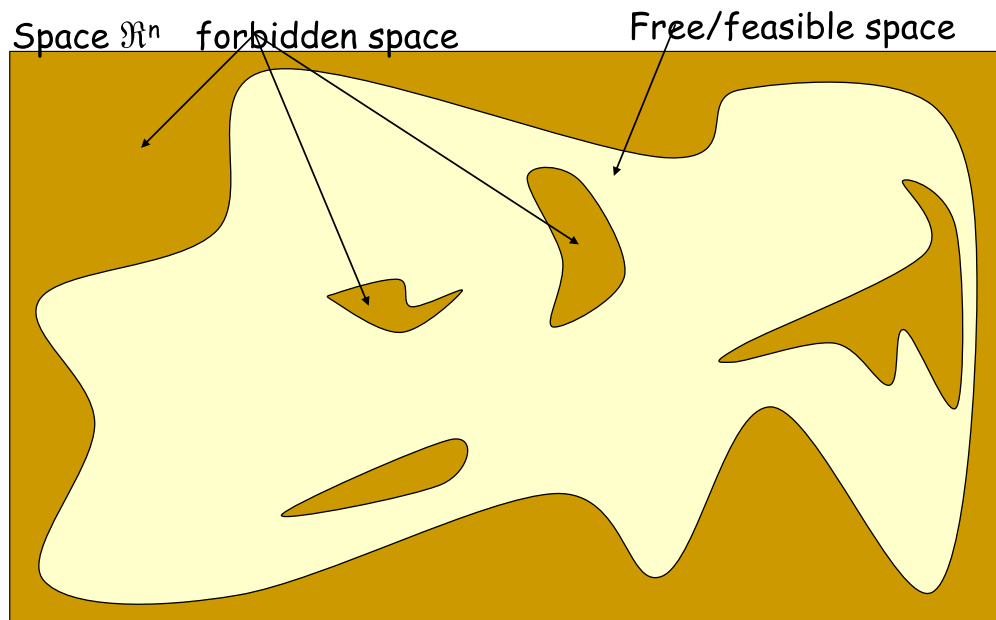




## 3.4 概率完备的连通图构建



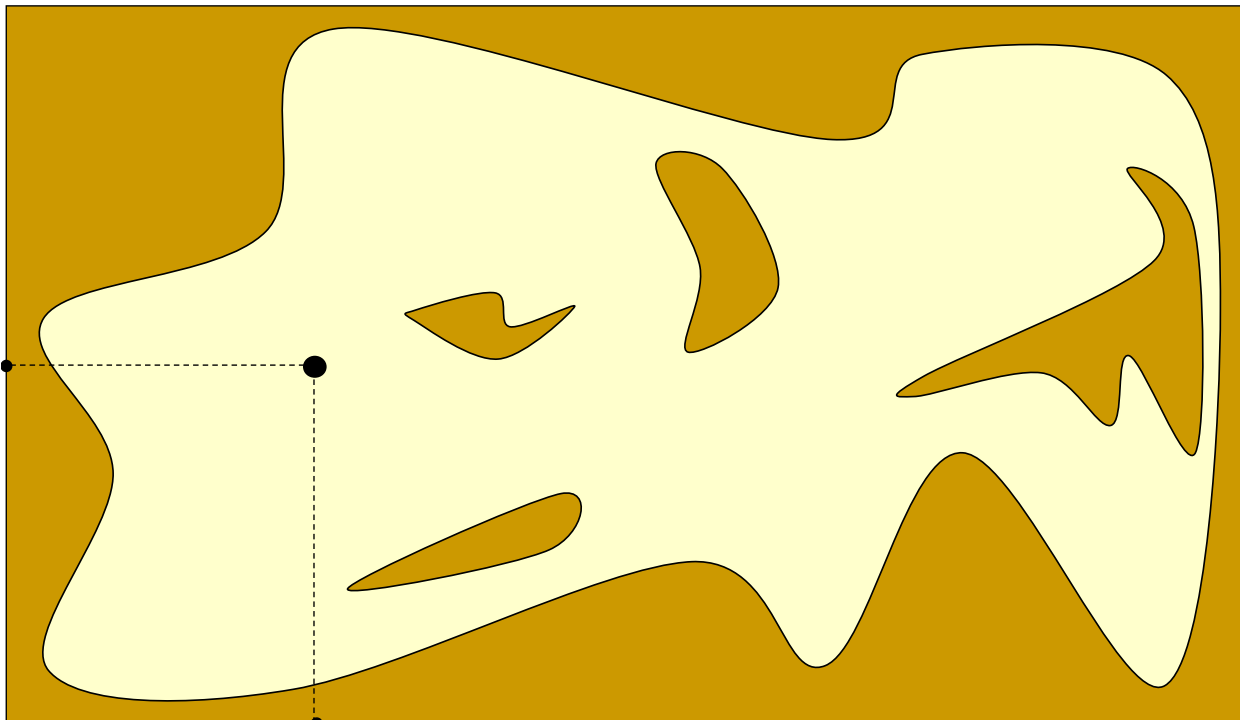
# 1. PRM(Probabilistic Roadmap)



**基本思想：**  
通过随机采样和碰撞检测找到自由位形空间中的路径点和无碰路径，构建连通图

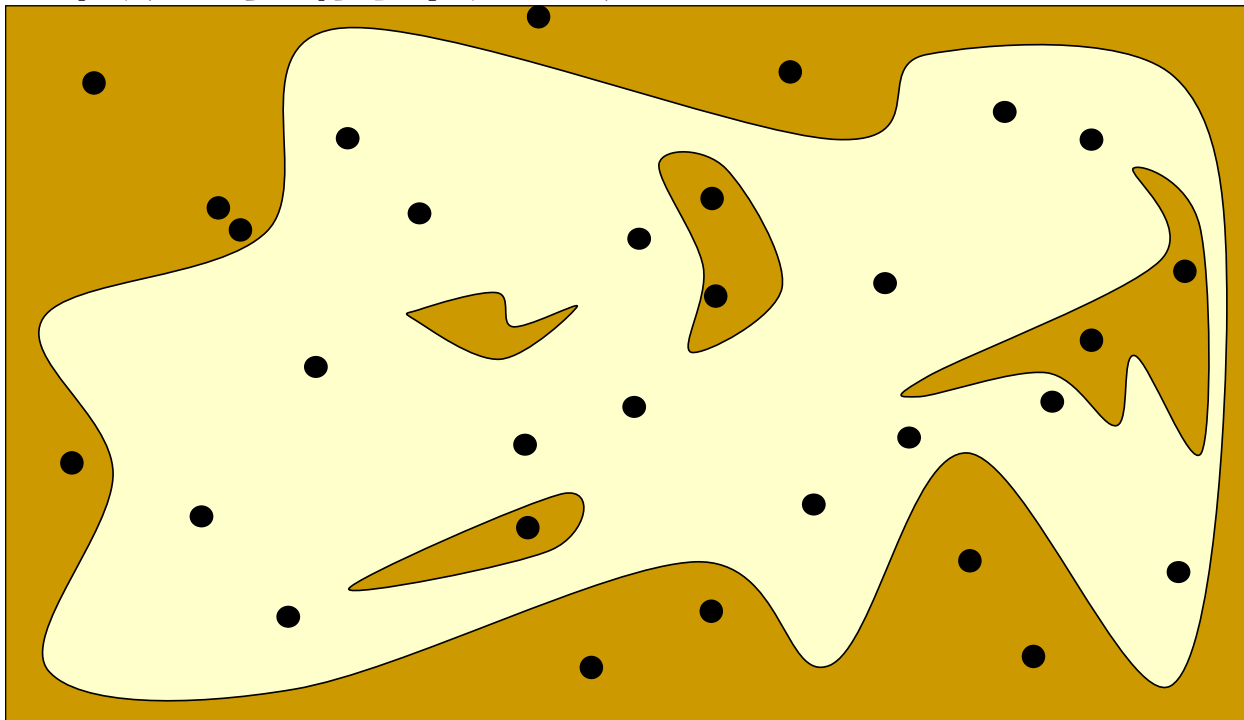
# 1. PRM(Probabilistic Roadmap)

在位形空间坐标系中随机取点



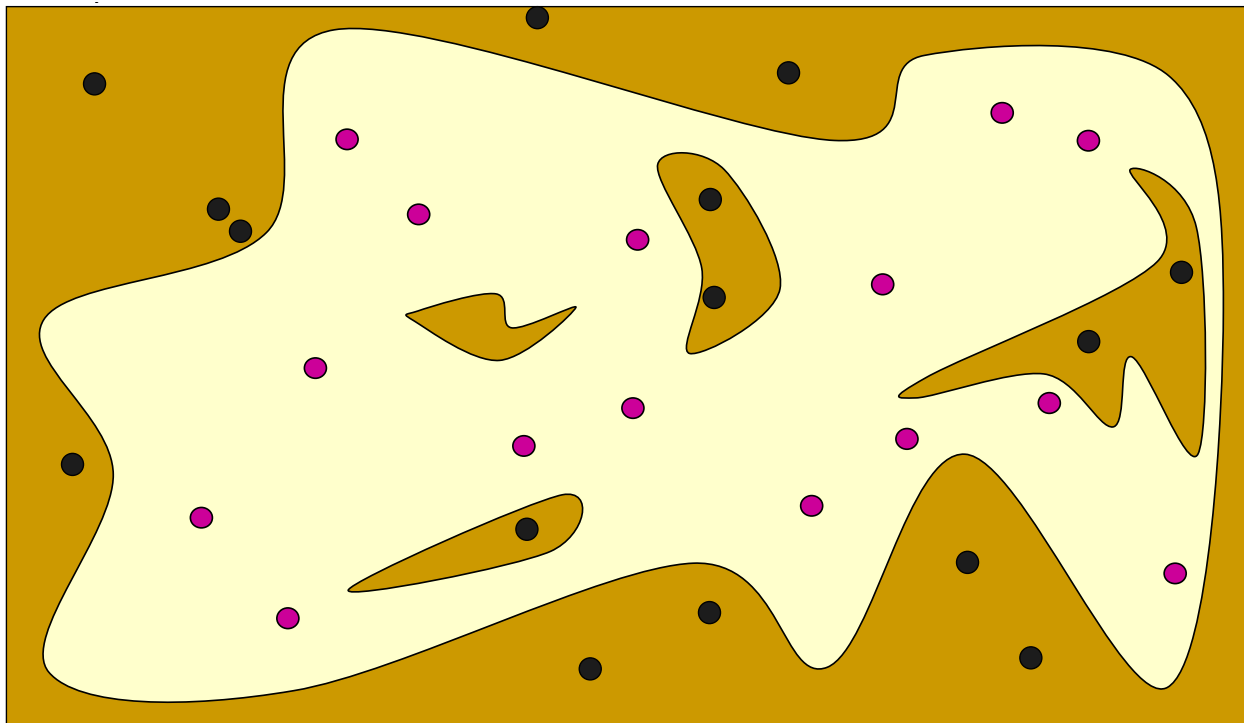
# 1. PRM(Probabilistic Roadmap)

在位形空间坐标系中随机取点



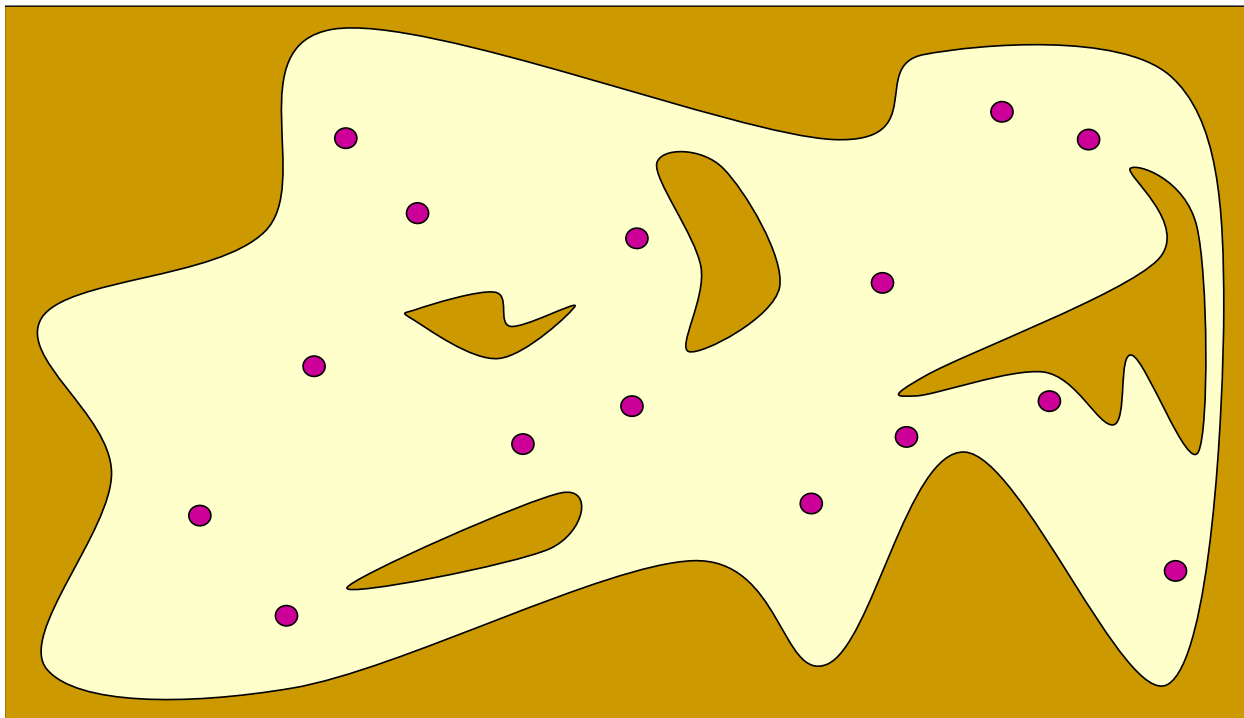
# 1. PRM(Probabilistic Roadmap)

对采样的姿态进行碰撞检测



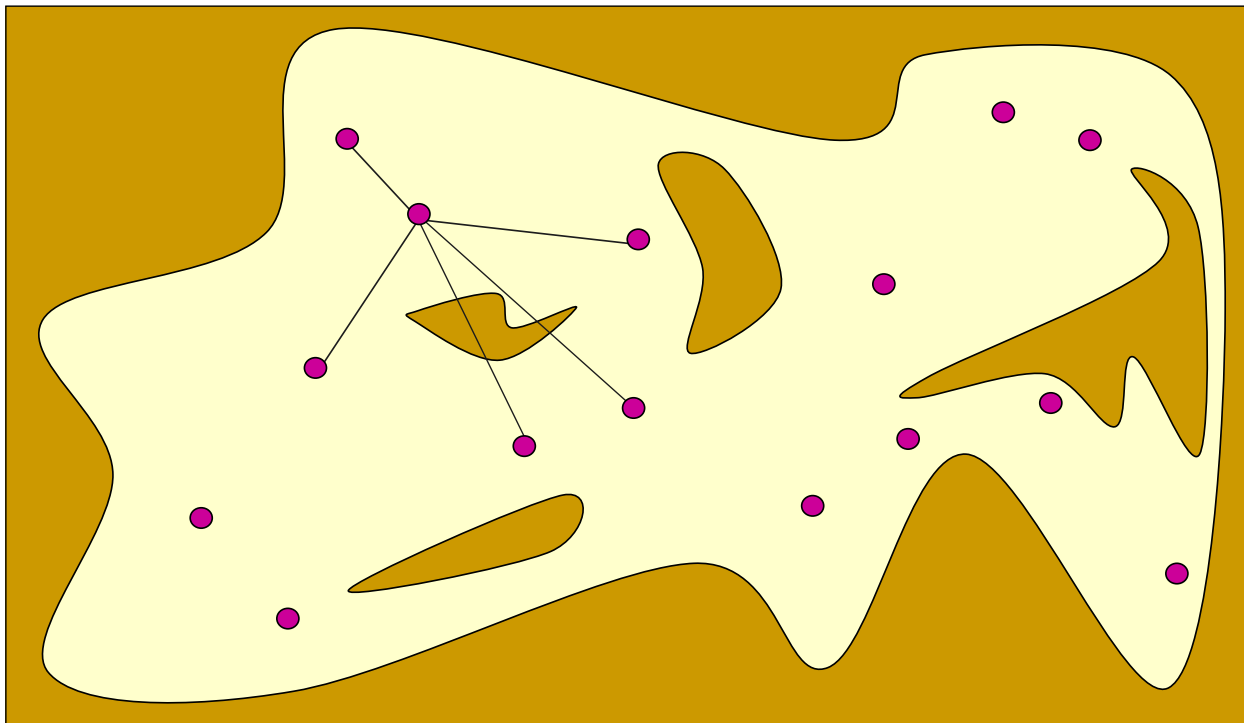
# 1. PRM(Probabilistic Roadmap)

无碰撞姿态成为图节点



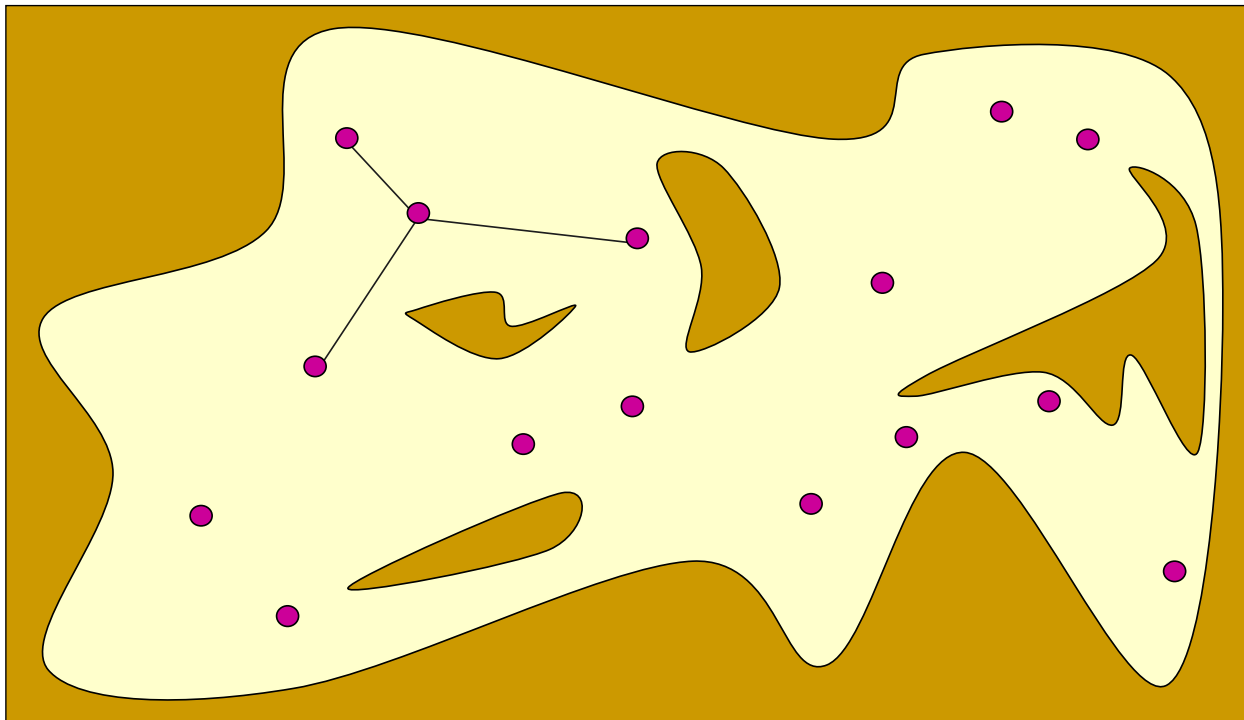
# 1. PRM(Probabilistic Roadmap)

每个图节点和其最近相邻的 $k$ 个节点直线连接



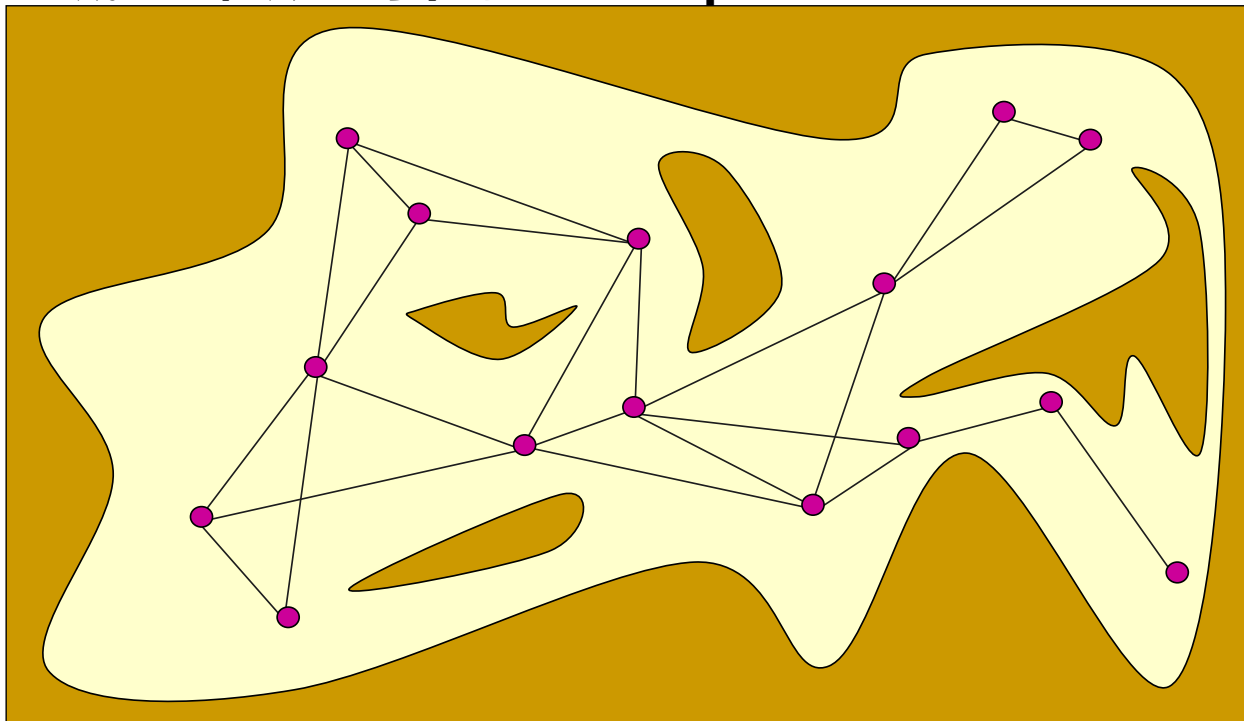
# 1. PRM(Probabilistic Roadmap)

保留无碰路径为图的边



# PRM(Probabilistic Roadmap)

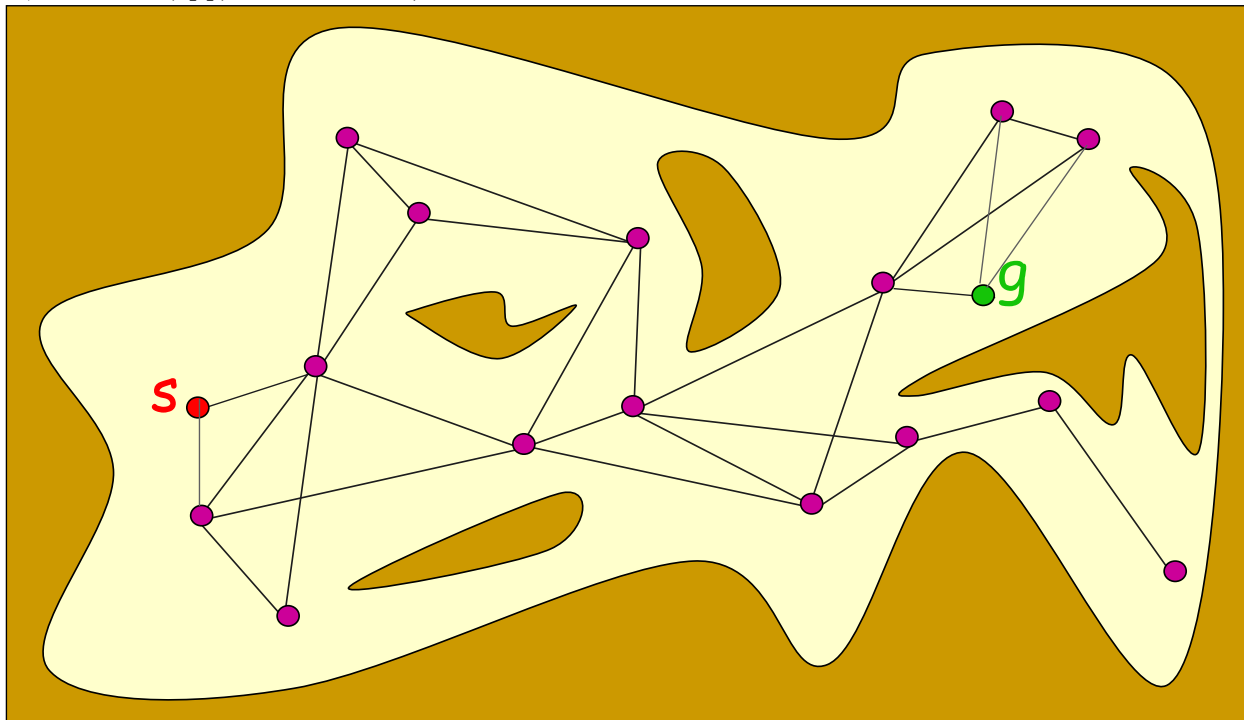
构成自由位形空间中的Roadmap





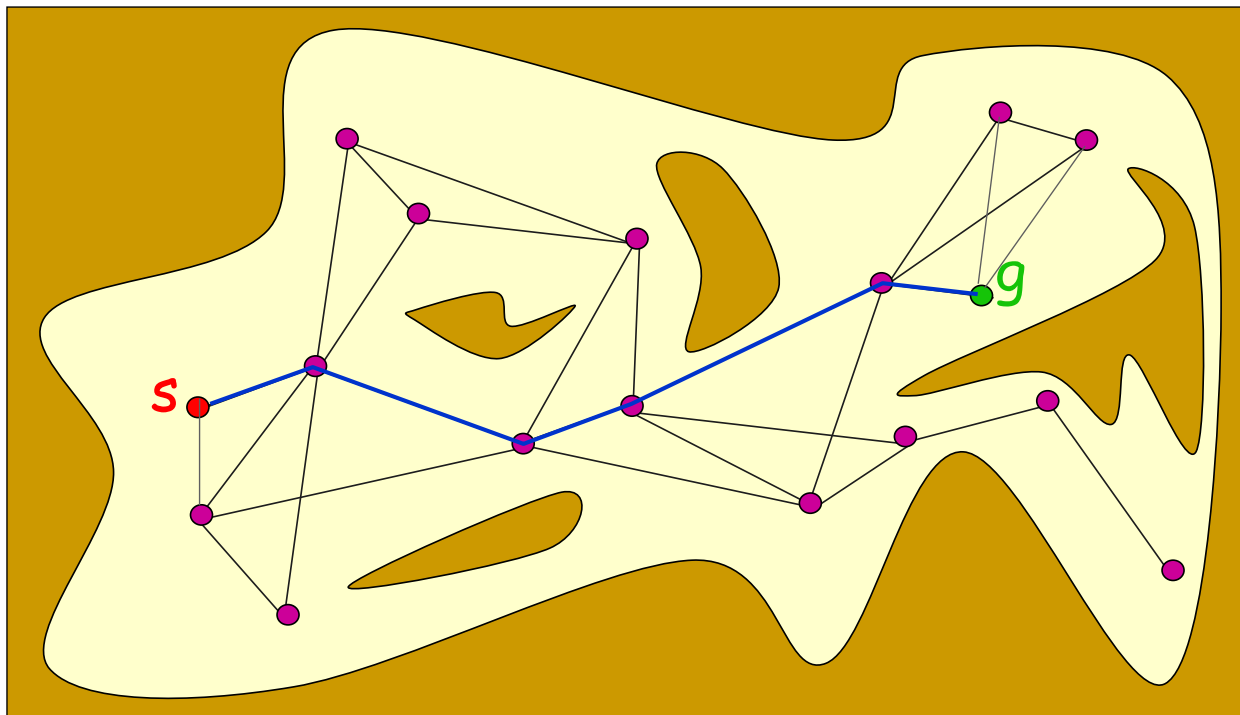
# 1. PRM(Probabilistic Roadmap)

加入起始点和终止点



## 1. PRM(Probabilistic Roadmap)

## 在PRM中搜索一条从起始点到终止点的路径



# PRM需要考虑的几个问题

- 随机位形选择      通常采用均匀随机采样方式
- 寻找最近邻点      可以采用KD树方法加速
- 生成局部路径      在一定范围内直接连接图节点
- 检查路径无碰      可以增量式取点或者二分法取点，判断点是否在障碍物区域内



# PRM

## ○ 优点：

- 简化了对环境的解析计算，可以快速构建得到行车图
- 适用于高维度自由位形空间中的规划
- 是一个近似完备的路径规划方法

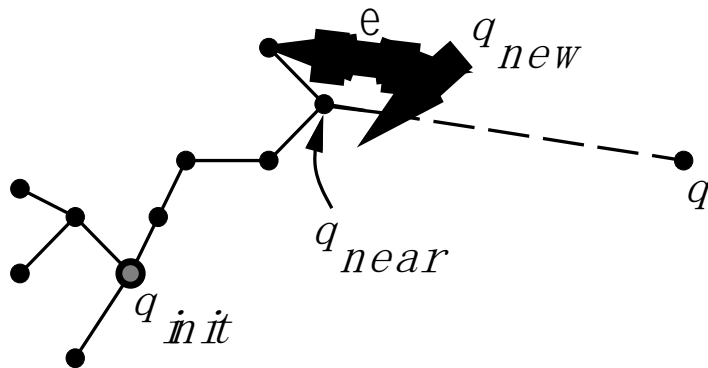
## ○ 缺点：

- 对自由空间连通性表达的完整性依赖于采样次数
- 从算法通用性上来讲难以评估需要多少时间做充分采样
- 不考虑机器人执行的可行性



## 2. RRT(Rapid-Exploring Random Tree)

- 1998年由美国爱荷华州立大学Steven M.Lavalle教授提出
- 基本思想：
  - 连通图采用树的形式，以起始点作为树的根节点
  - 采用在空间中随机采样、连接树中最近节点的方式拓展树
  - 考虑机器人的运动执行能力
  - 通过树结构可以直接回溯得到路径



## 2. RRT(Rapid-Exploring Random Tree)

---

### Algorithm 1: RRT Algorithm

---

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

**Result:** A path  $\Gamma$  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();

---



## 2. RRT(Rapid-Exploring Random Tree)

---

### Algorithm 1: RRT Algorithm

---

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

**Result:** A path  $\Gamma$  from  $x_{init}$  to  $x_{goal}$

$\mathcal{T}.init();$       以运动规划初始状态  $x_{init}$  为根节点，建立搜索树

**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();

---

## 2. RRT(Rapid-Exploring Random Tree)

---

### Algorithm 1: RRT Algorithm

---

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

**Result:** A path  $\Gamma$  from  $x_{init}$  to  $x_{goal}$

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

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

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

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

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

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

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

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

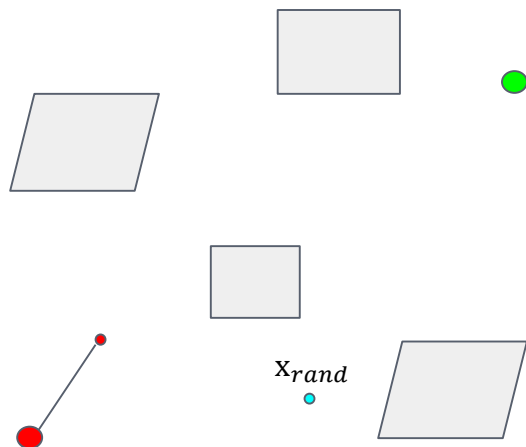
$\mathcal{T}.addEdge(E_i);$

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

**Success** $();$

---

在可行空间中随机采样





## 2. RRT(Rapid-Exploring Random Tree)

---

### Algorithm 1: RRT Algorithm

---

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

**Result:** A path  $\Gamma$  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**

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

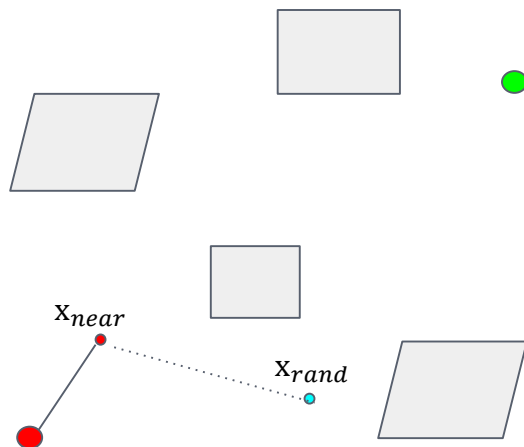
$\mathcal{T}.addEdge(E_i);$

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

        Success();

---

找到树中离采样点最近的树节点



## 2. RRT(Rapid-Exploring Random Tree)

---

### Algorithm 1: RRT Algorithm

---

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

**Result:** A path  $\Gamma$  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**

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

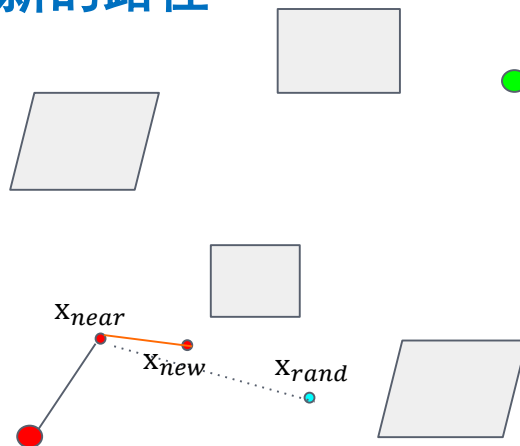
$\mathcal{T}.addEdge(E_i);$

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

        Success();

---

根据机器人执行能力生成新节点  
和新的路径



## 2. RRT(Rapid-Exploring Random Tree)

---

### Algorithm 1: RRT Algorithm

---

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

**Result:** A path  $\Gamma$  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**

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

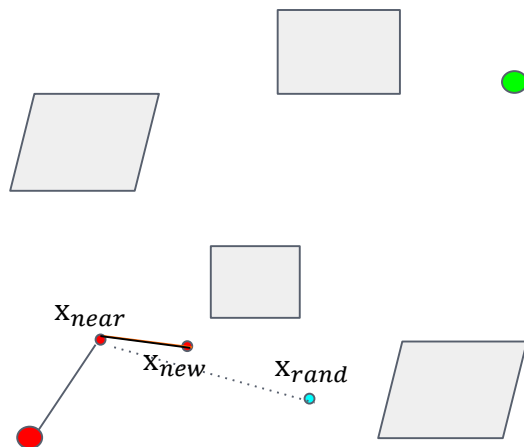
$\mathcal{T}.addEdge(E_i);$

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

        Success();

---

根据无碰撞检测加入树中



## 2. RRT(Rapid-Exploring Random Tree)

---

### Algorithm 1: RRT Algorithm

---

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

**Result:** A path  $\Gamma$  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**

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

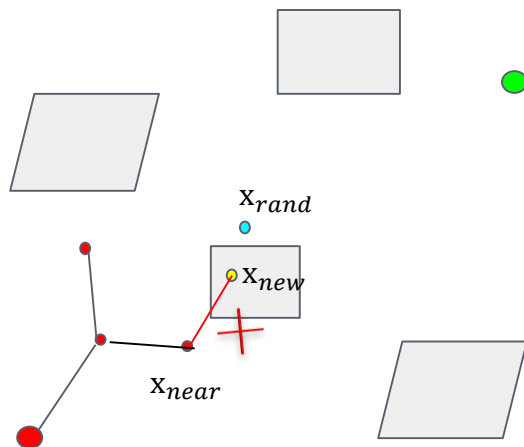
$\mathcal{T}.addEdge(E_i);$

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

        Success();

---

不断重复该过程



## 2. RRT(Rapid-Exploring Random Tree)

---

### Algorithm 1: RRT Algorithm

---

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

**Result:** A path  $\Gamma$  from  $x_{init}$  to  $x_{goal}$

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

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

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

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

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

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

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

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

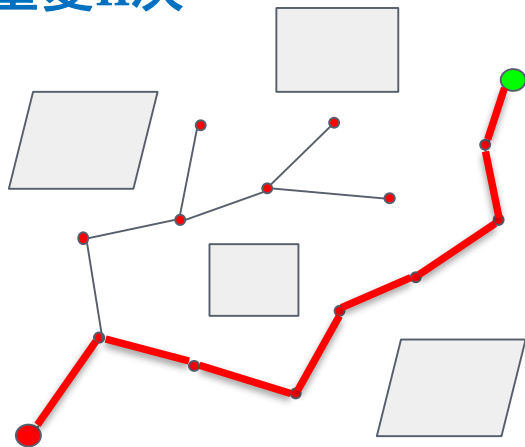
$\mathcal{T}.addEdge(E_i);$

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

        Success();

---

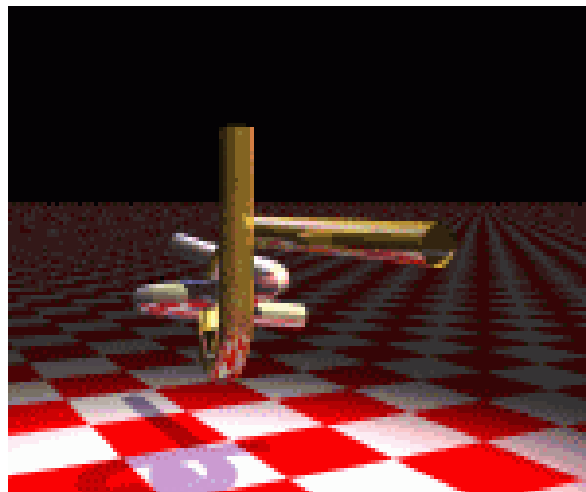
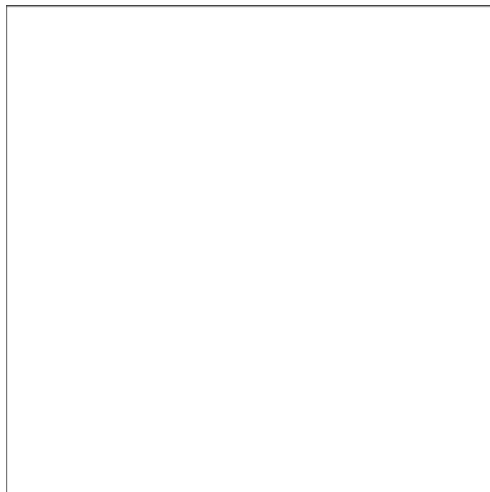
直到树节点生长到重点区域，或者重复n次



## 2. RRT(Rapid-Exploring Random Tree)

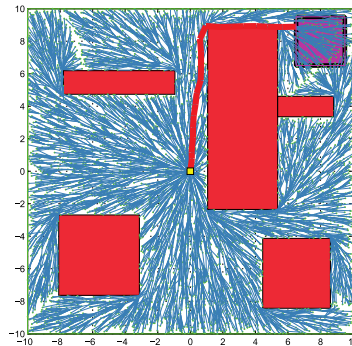
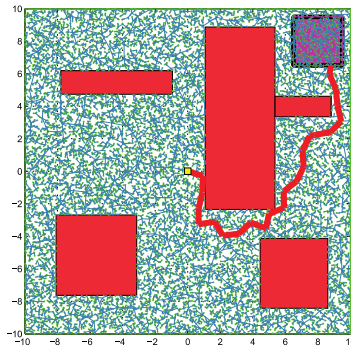
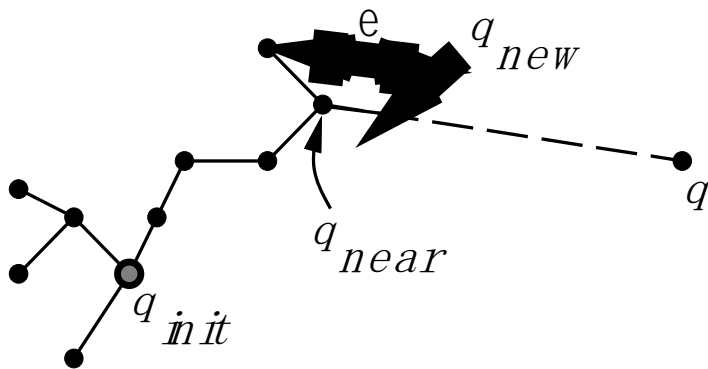
### ○ 影响规划收敛速度的三个步骤

- 随机状态的采样
- 在搜索树中查找与随机状态距离最近的节点
- 新生成节点的防碰撞检测



# RRT改进2

- 针对问题：随机性小步扩展导致路径曲折，成本高
- RRT\*：为实现渐近最优，考虑路径成本
  - 寻找树中新节点邻域内到新节点路径最短的节点，建立连接，加入树集合
  - 对树中新节点邻域内节点进行判断，如果从新节点到该节点形成的路径优于现有树中路径，则将该节点父节点修改为新节点



# RRT改进2

- 针对问题：随机性小步扩展导致路径曲折，成本高

---

## Algorithm 2: RRT\*Algorithm

---

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

**Result:** A path  $\Gamma$  from  $x_{init}$  to  $x_{goal}$

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

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

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

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

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

**if**  $\text{CollisionFree}(x_{new})$  **then**

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


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

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

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

---

为实现渐近最优，  
新节点加入时根据  
路径成本进行树结  
构优化





# RRT改进2

- 针对问题：随机性小步扩展导致路径曲折，成本高

---

## Algorithm 2: RRT\*Algorithm

---

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

**Result:** A path  $\Gamma$  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**

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

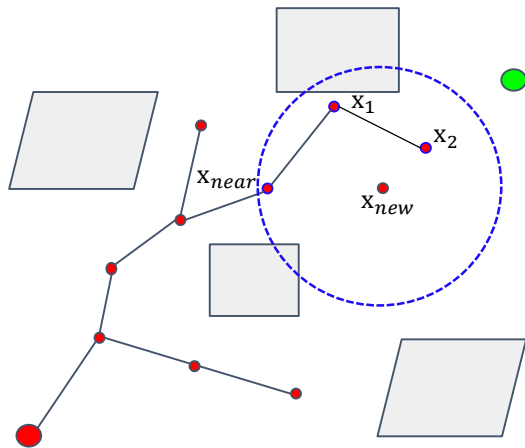
$x_{min} \leftarrow ChooseParent(X_{near}, x_{near}, x_{new});$

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

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

---

选择多个邻节点



# RRT改进2

- 针对问题：随机性小步扩展导致路径曲折，成本高

---

## Algorithm 2: RRT\*Algorithm

---

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

**Result:** A path  $\Gamma$  from  $x_{init}$  to  $x_{goal}$

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

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

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

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

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

**if**  $\text{CollisionFree}(x_{new})$  **then**

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

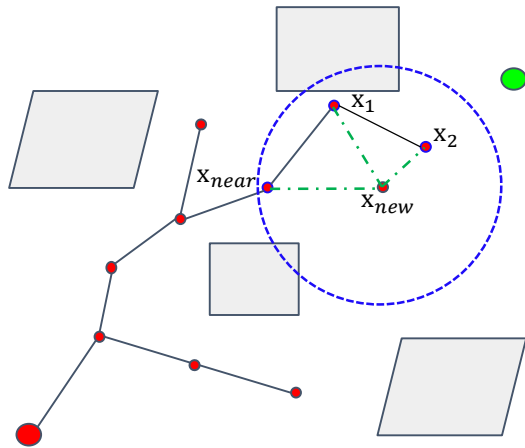
$x_{min} \leftarrow \text{ChooseParent}(X_{near}, x_{near}, x_{new});$

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

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

---

评估各邻节点作为父节点下的总路径长度



# RRT改进2

- 针对问题：随机性小步扩展导致路径曲折，成本高

---

## Algorithm 2: RRT\*Algorithm

---

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

**Result:** A path  $\Gamma$  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**

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

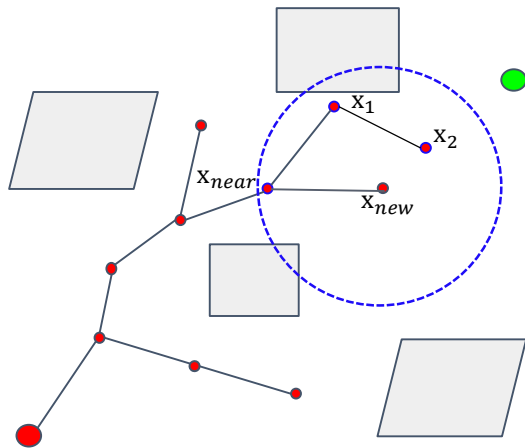
$x_{min} \leftarrow ChooseParent(X_{near}, x_{near}, x_{new});$

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

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

---

根据总最短路径选择父节点



# RRT改进2

- 针对问题：随机性小步扩展导致路径曲折，成本高

---

## Algorithm 2: RRT\*Algorithm

---

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

**Result:** A path  $\Gamma$  from  $x_{init}$  to  $x_{goal}$

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

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

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

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

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

**if**  $\text{CollisionFree}(x_{new})$  **then**

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

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

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

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

---

优化邻节点路径，如果从新节点到该节点形成的路径优于现有树中路径，则将该节点父节点修改为新节点



# RRT改进2

- 针对问题：随机性小步扩展导致路径曲折，成本高

---

## Algorithm 2: RRT\*Algorithm

---

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

**Result:** A path  $\Gamma$  from  $x_{init}$  to  $x_{goal}$

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

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

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

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

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

**if**  $\text{CollisionFree}(x_{new})$  **then**

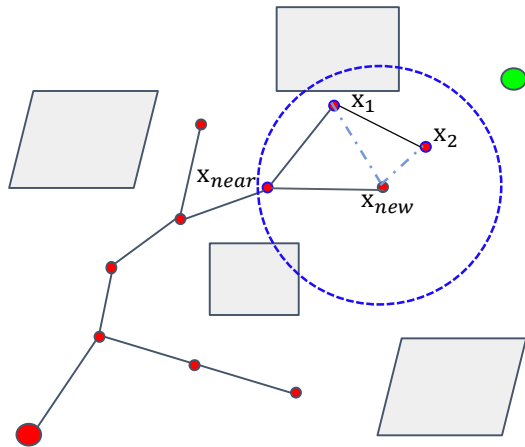
$X_{near} \leftarrow \text{NearC}(\mathcal{T}, x_{new});$

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

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

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

---



# RRT改进2

- 针对问题：随机性小步扩展导致路径曲折，成本高

---

## Algorithm 2: RRT\*Algorithm

---

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

**Result:** A path  $\Gamma$  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**

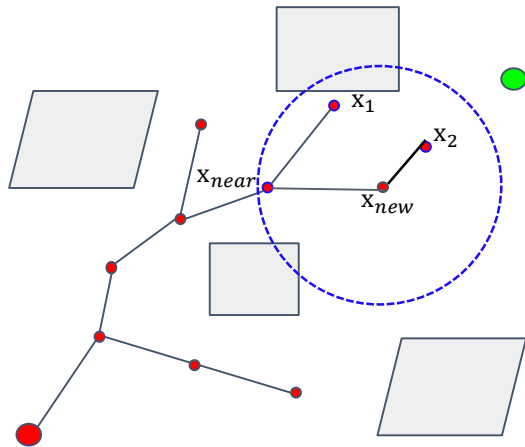
$X_{near} \leftarrow NearC(\mathcal{T}, x_{new});$

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

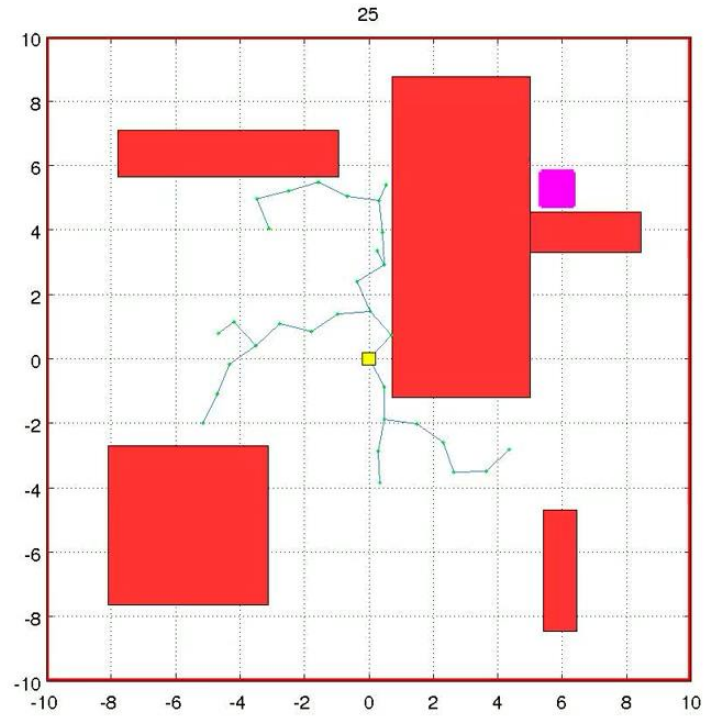
$\mathcal{T}.addNodeEdge(x_{min}, x_{new});$

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

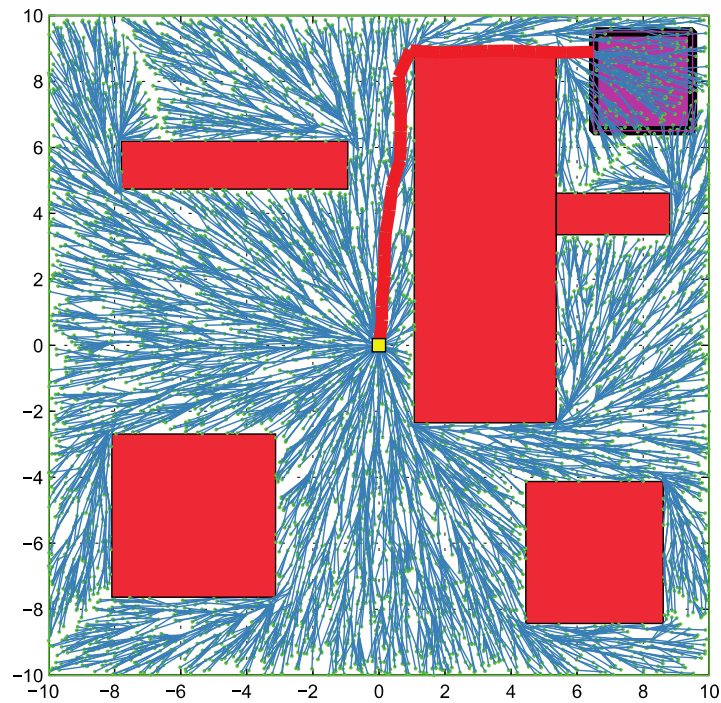
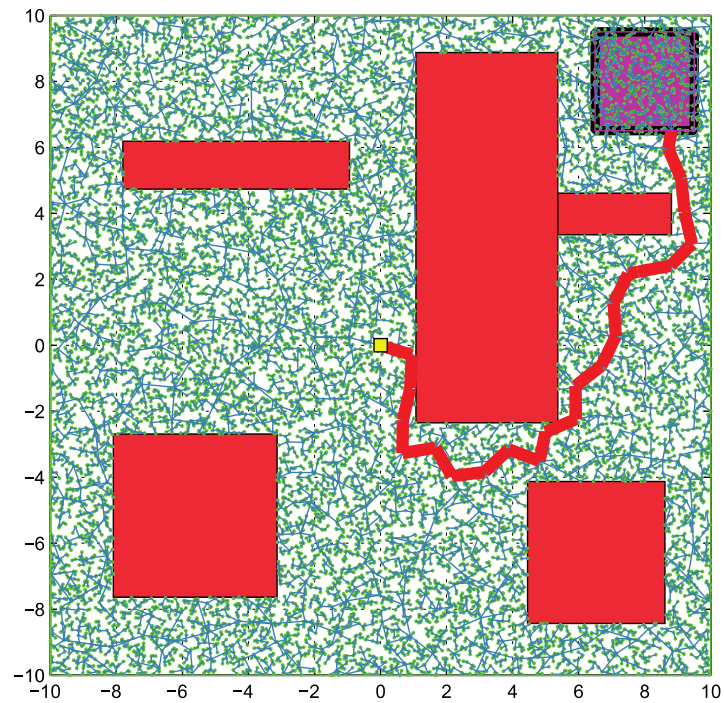
---



# RRT\*



# RRT\*





# 分辨率完备与概率完备方法比较

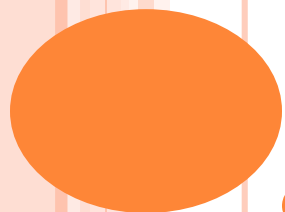
## ○ 空间离散采样：

- 分辨率完备是基于解析计算的姿态空间分解
- 概率完备是基于随机采样生成连通图或者树

## ○ 位形空间连通性表示：

- 分辨率完备完全表达了自由空间的连通性，高维情况下计算负担重
- 概率完备方法是近似表达了连通性，但计算快速，只需要计算单个机器人姿态是否存在碰撞，其效率与碰撞检测模块效率相关





**END!**

