



## 第三讲 路径规划

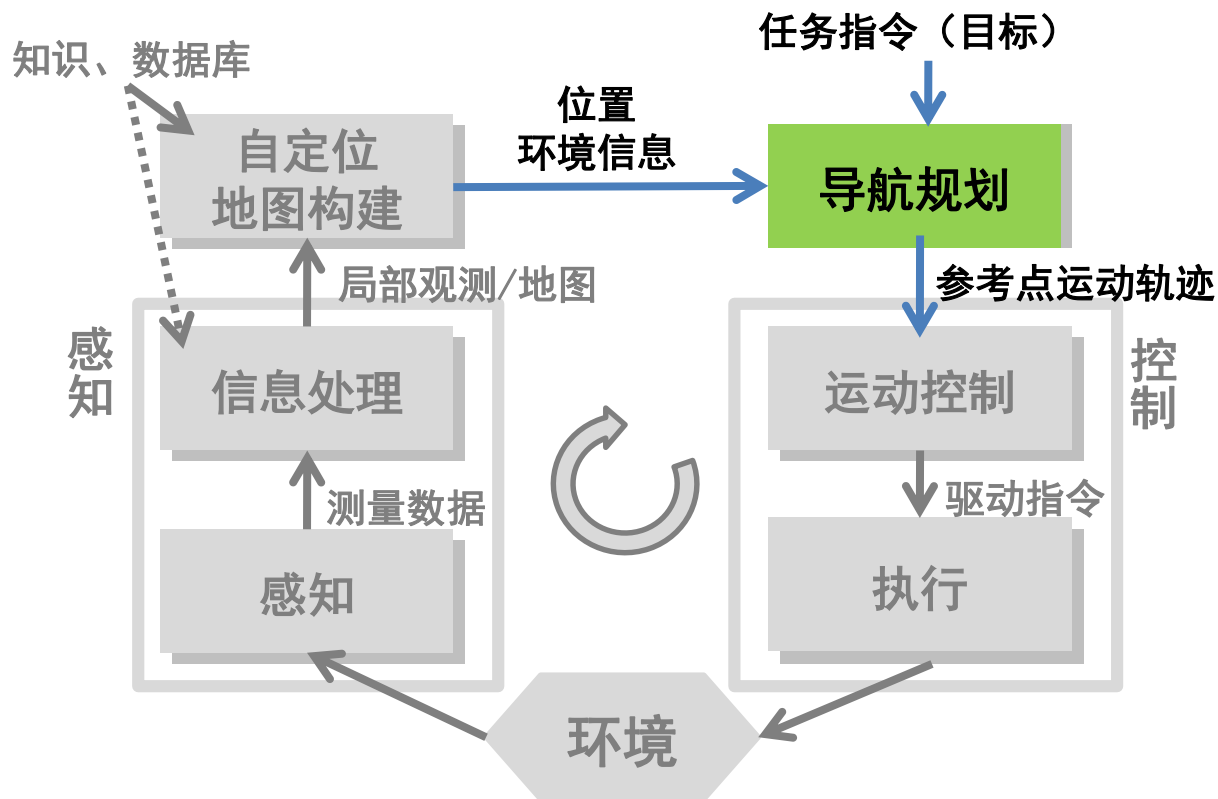
王越

浙江大学 控制科学与工程学院

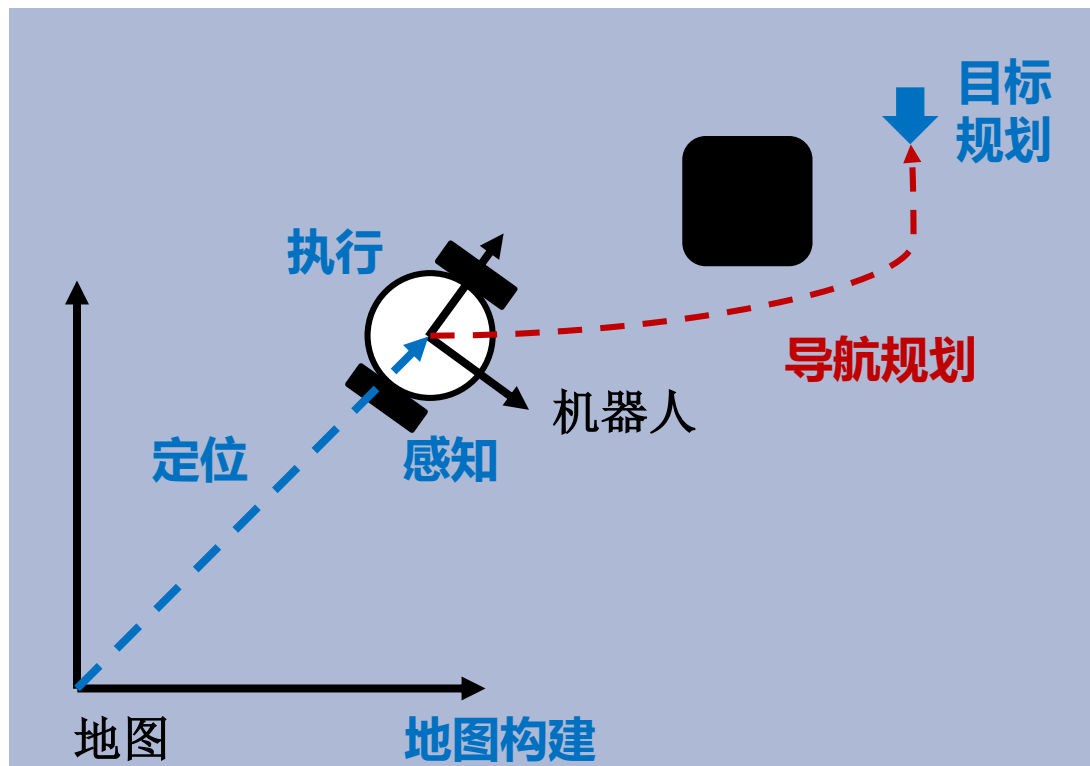


## 导航规划简介

# 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

# 导航规划类型

- 固定路径导引：
  - 有人工标识导引
- 无轨导航：
  - 有人工标识导引的无固定路径（无轨）导航
  - 无标识导引的自然无轨导航



# 导航规划类型1：有人工标识导引的固定路径导航



磁条导航



磁感应线导航



磁钉导航



二维码导航



AGV: Automatic Guided Vehicle 自动导引车

- 优点：技术成熟、稳定可靠、价格优惠
- 缺点：需要施工和维护、路线无法调整

## 导航规划类型2：有人工标识导引的无轨导航



激光反射板导航



- 优点：技术成熟、路径可调
- 缺点：需要施工和维护、价格昂贵



## 导航规划类型3：无人工标识导引的无轨导航



自然导航

- 优点：无需施工、路径可调、精确定位、室内外通用
- 缺点：算法复杂，环境变化影响定位可靠性和稳定性





# 导航规划问题

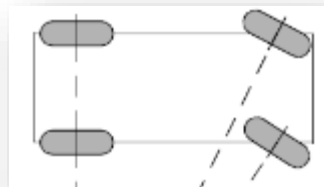
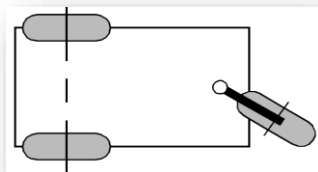
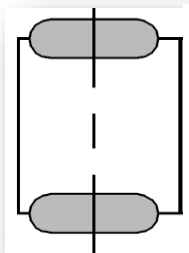
在给定环境的全局或局部知识以及一个或者一系列目标位置的条件下，使机器人能够根据知识和传感器感知信息高效可靠地到达目标位置

- 环境几何约束
- 机器人执行约束



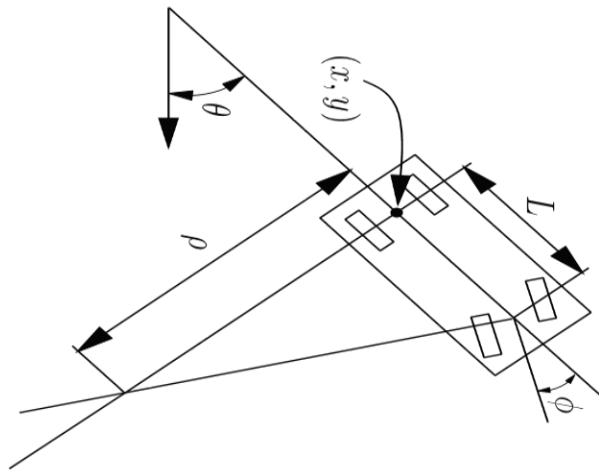
# 非完整机器人 NON-HOLOMIC

- 存在至少一个非完整运动学约束

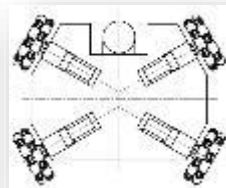
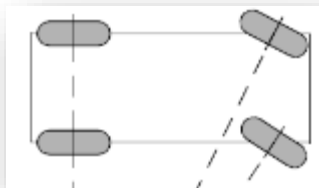
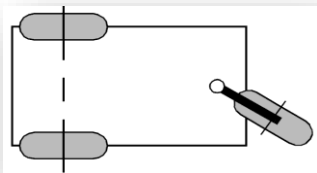
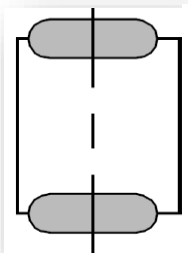


# 其他运动学约束

- 最大转弯半径
- 最大速度、最大加速度

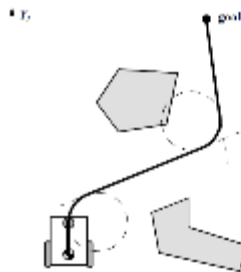
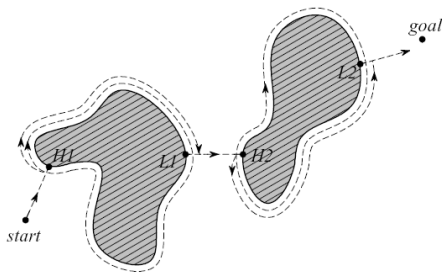
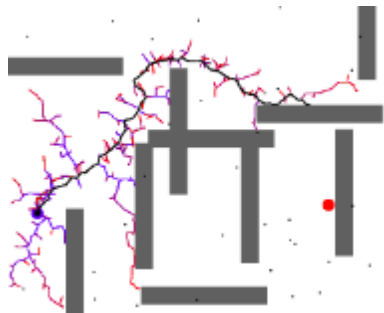


# 移动机器人形态丰富，运动学模型和约束各不相同

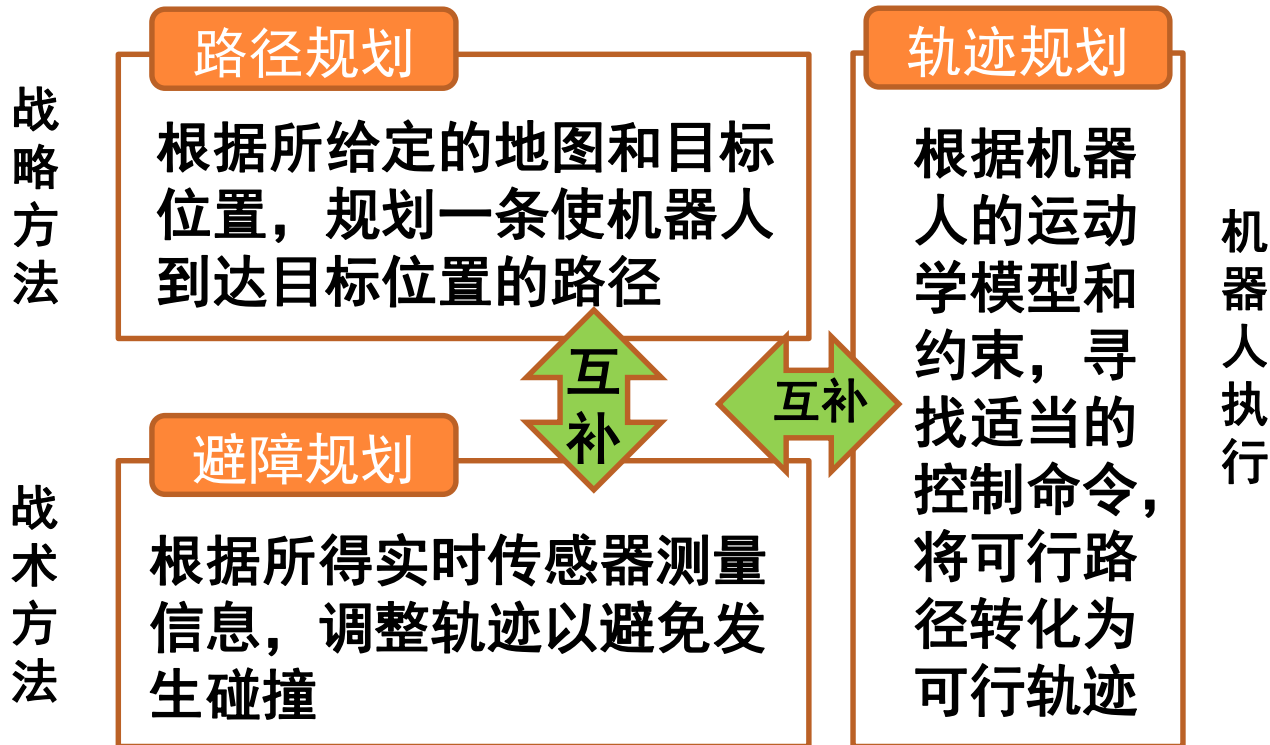


# 导航规划的主要研究内容

- **路径规划**：根据所给定的地图和目标位置，规划一条使机器人到达目标位置的路径（只考虑工作空间的几何约束，不考虑机器人的运动学模型和约束）
- **避障规划**：根据所得到的实时传感器测量信息，调整路径/轨迹以避免发生碰撞
- **轨迹生成**：根据机器人的运动学模型和约束，寻找适当的控制命令，将可行路径转化为可行轨迹。



# 路径规划、避障规划、轨迹规划三者关系

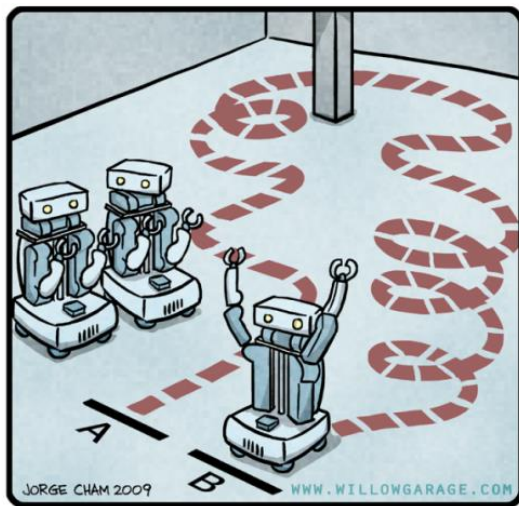




## 3.1 基本概念

# 路径规划

- 根据所给定的地图和目标位置，规划一条使机器人到达目标位置的路径

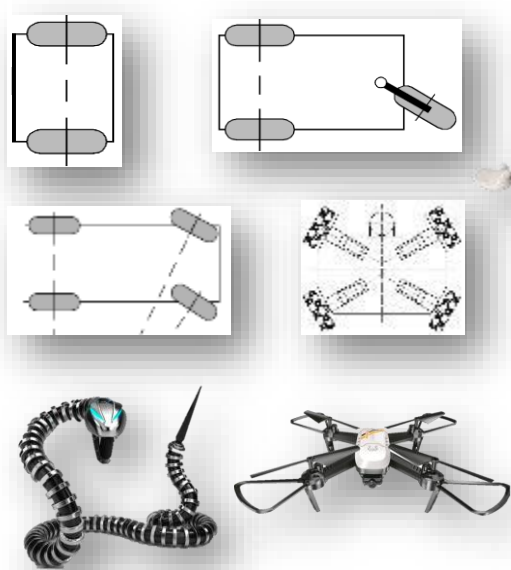


"HIS PATH-PLANNING MAY BE  
SUB-OPTIMAL, BUT IT'S GOT FLAIR."



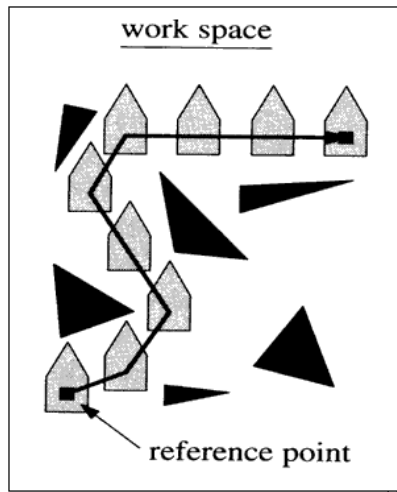
# 路径规划

- 简化：只考虑工作空间的几何约束，不考虑机器人的运动学模型和约束



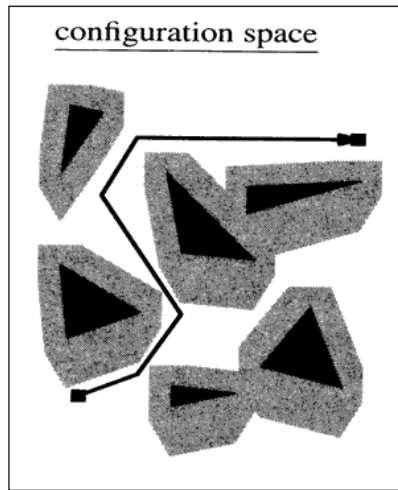
# 工作空间与位形空间(C-SPACE)

工作空间



工作空间：移动机器人上的参考点能达到的空间集合，机器人采用位置和姿态描述，并需考虑体积

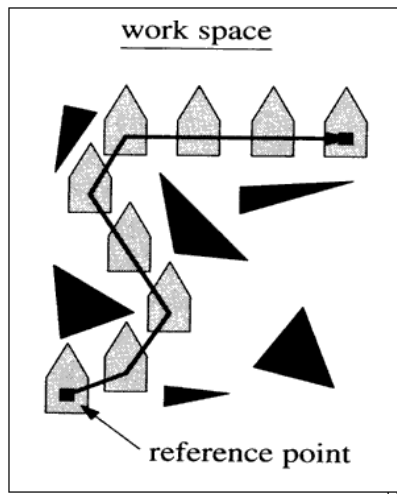
位形空间  
(Configuration Space)



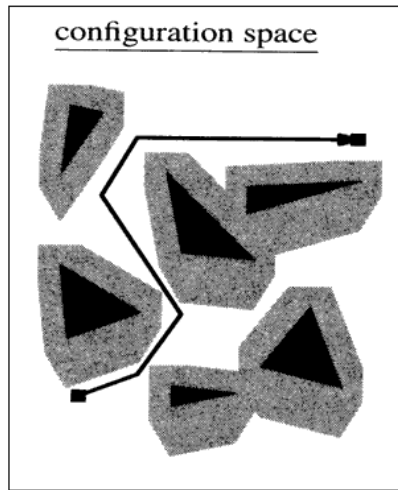
位形空间：机器人成为一个可移动点，不考虑姿态、体积和非完整运动学约束

# 工作空间与位形空间(C-SPACE)

工作空间



configuration space



位形空间  
(Configuration Space)

障碍物按机器人半径进行膨胀

→ 机器人成为一个点

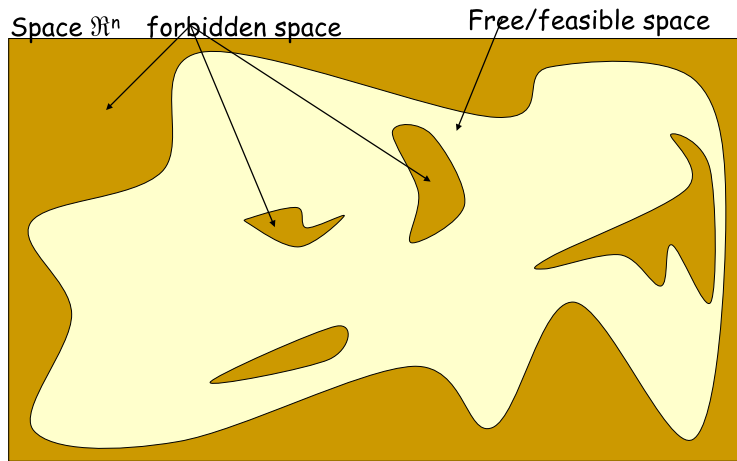
忽略非完整约束对姿态的限制

→ 机器人是完整的



# 位形空间

- 障碍物空间：不可行的位形集合
  - 在该空间中，机器人会与障碍物发生碰撞
- 自由空间：可行的位形集合
  - 在该空间中，机器人将无碰地安全移动

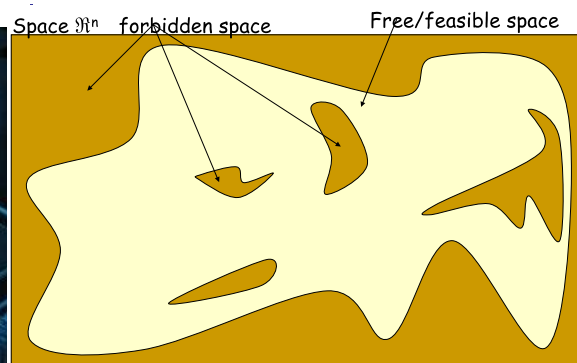
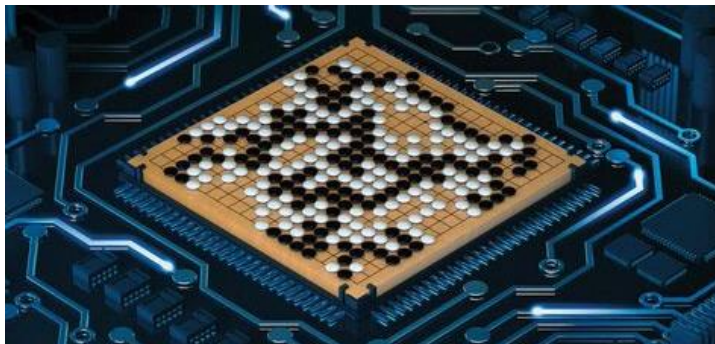


路径规划就是在自由位形空间中为机器人寻找一条路径，使其从初始位置运行到目标位置

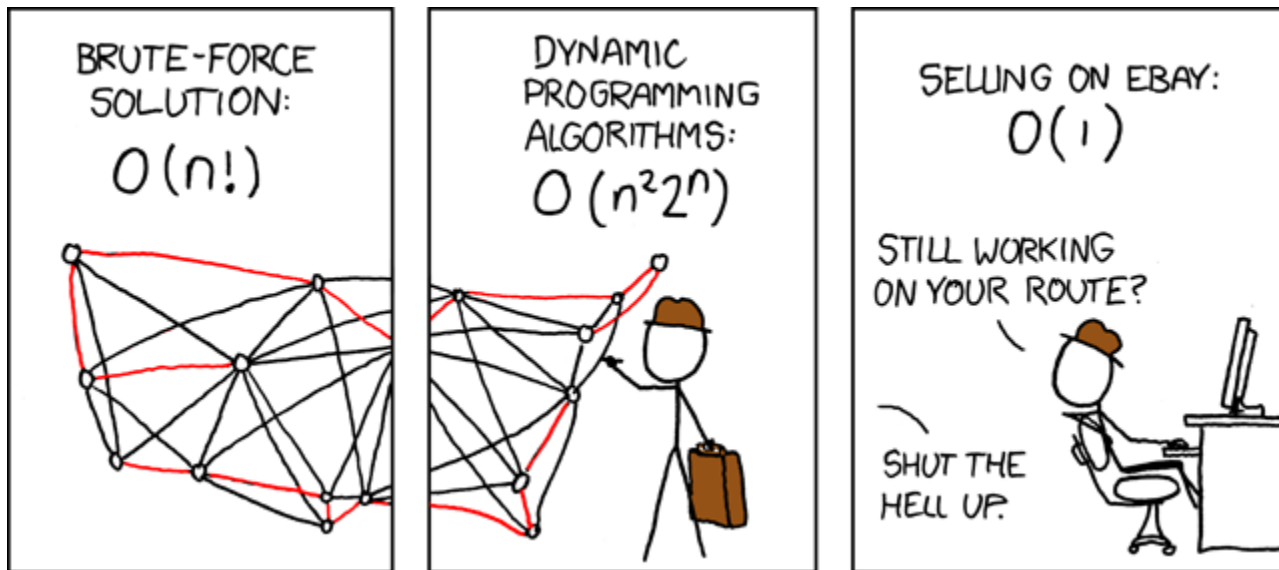


# 路径规划的完备性要求

- 完备性：当解存在时，能够在有限的时间内找到解
- 路径规划算法挑战：
  - 在连续空间内搜索，难以保证时间



# 一种常见的路径规划



拓扑连通图+最优路径搜索

路径规划的两个基本问题



# 拓扑连通图构建方法

- 基本思路：对空间作离散化
- **分辨率完备resolution completeness:**  
解析性离散化，确保获得可行解
  - 行车图法：基于障碍物几何形状分解姿态空间
  - 单元分解法：区分空闲单元和被占单元
  - 势场法：根据障碍物和目标对空间各点施加虚拟力
- **概率完备Probabilistic completeness:**  
基于概率进行随机采样离散化，使获得解的概率趋近于1
  - PRM (Probabilistic RoadMaps)
  - RRT (Rapid-Exploring Random Trees)



# 最优路径搜索方法

- 精确最优搜索法：深度优先法、宽度优先法
- 近似最优搜索法
  - 启发式搜索法：A\*，D\*
  - 准启发式搜索算法：退火、进化和蚁群优化等

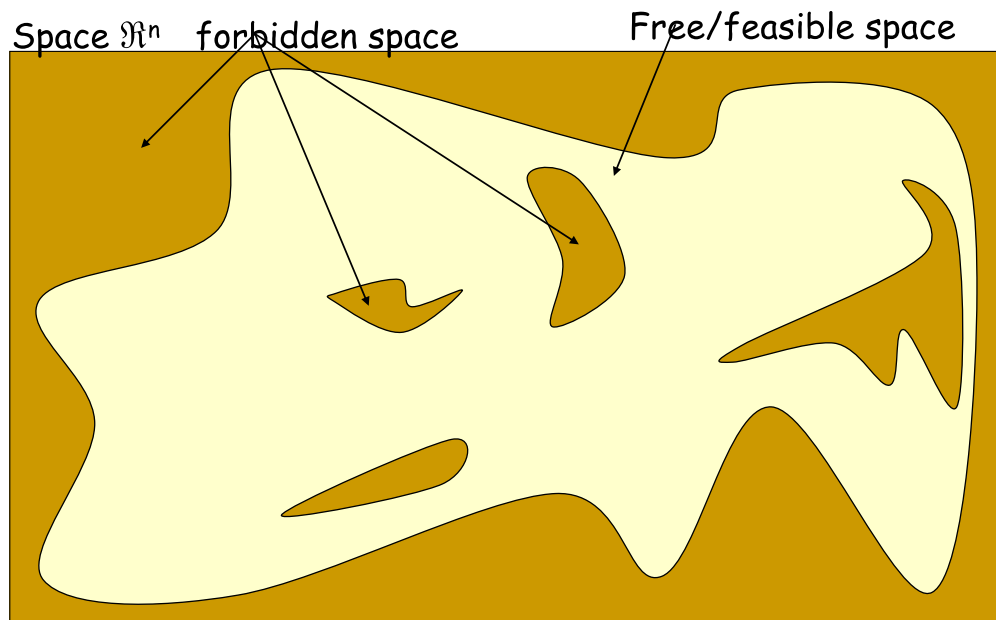






## 3.4 概率完备的连通图构建

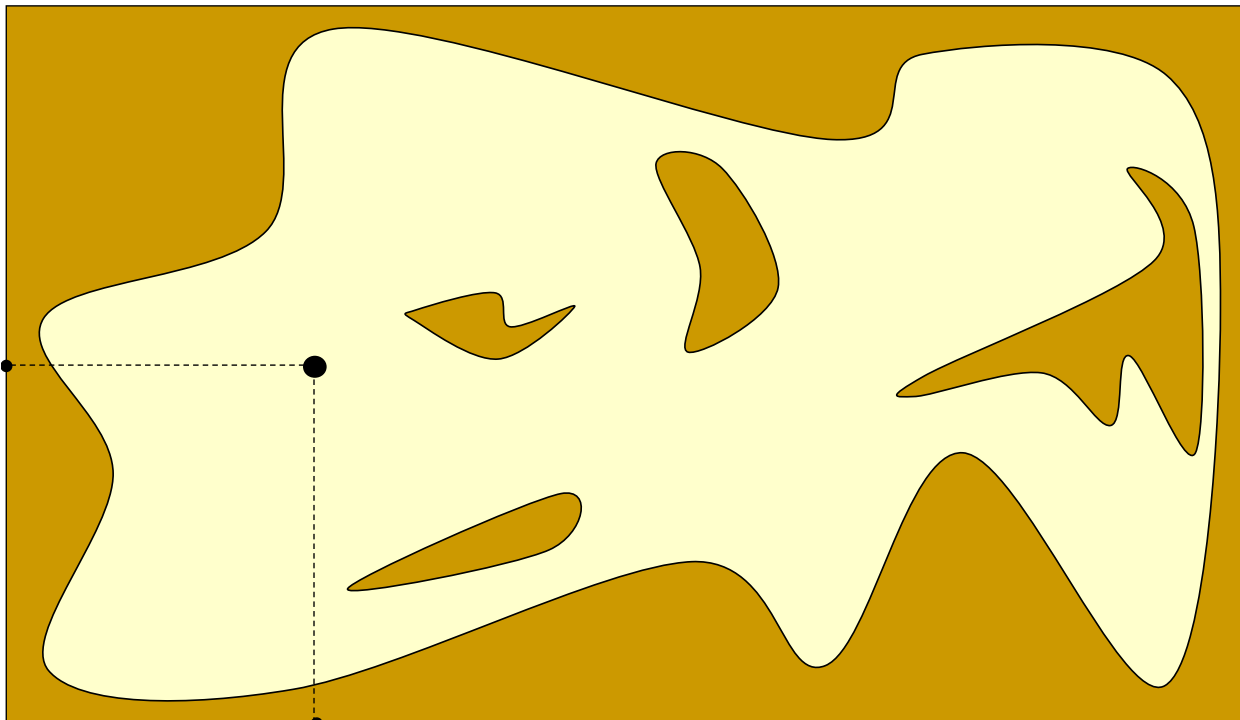
# 1. PRM(Probabilistic Roadmap)



**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

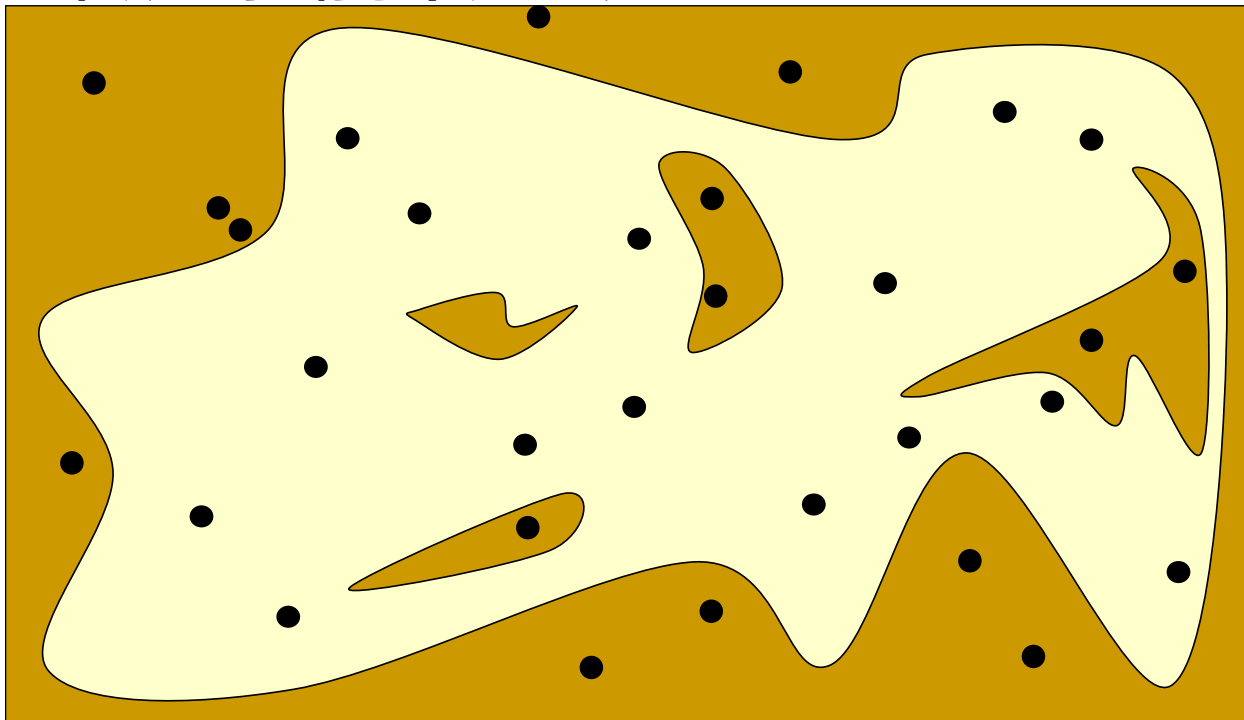
# 1. PRM(Probabilistic Roadmap)

在位形空间坐标系中随机取点



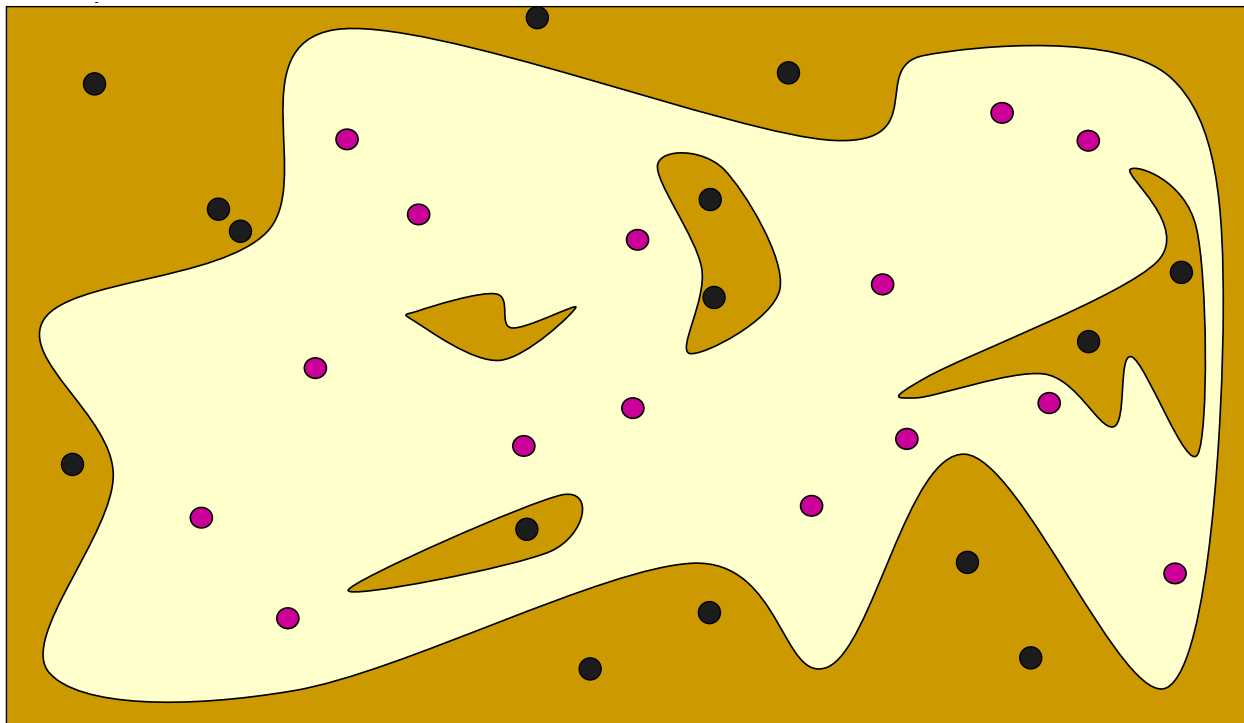
# 1. PRM(Probabilistic Roadmap)

在位形空间坐标系中随机取点



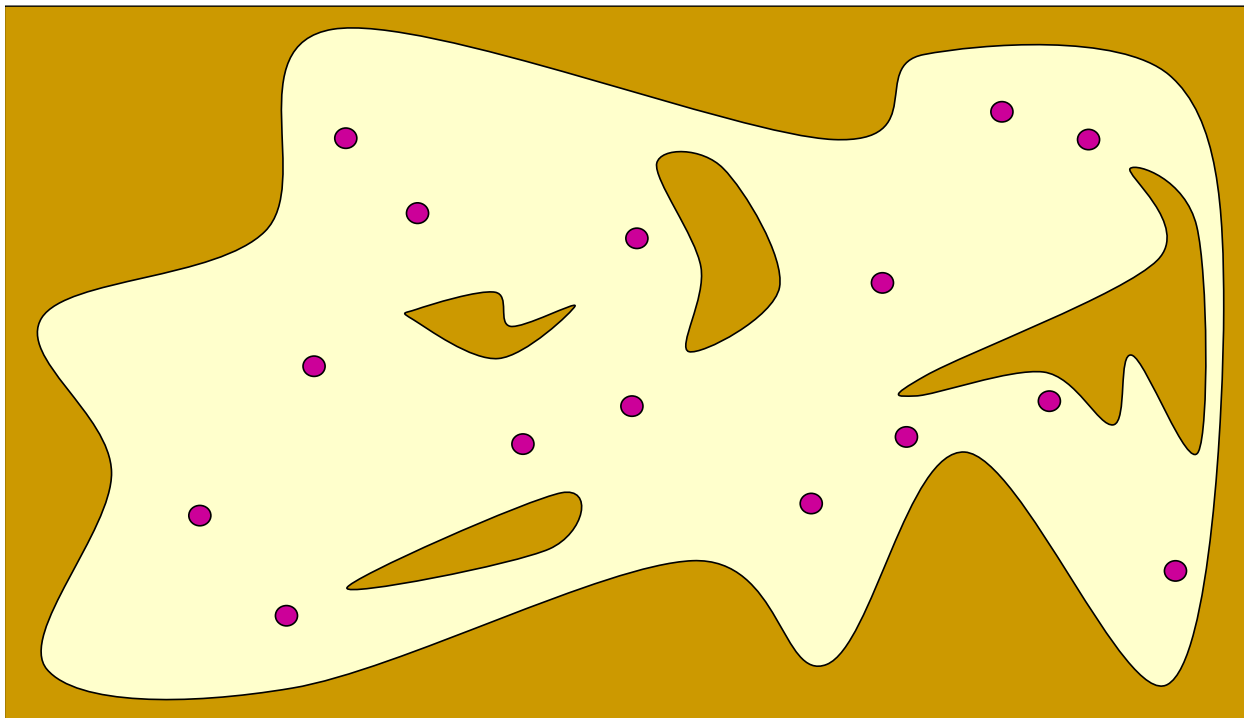
# 1. PRM(Probabilistic Roadmap)

对采样的姿态进行碰撞检测



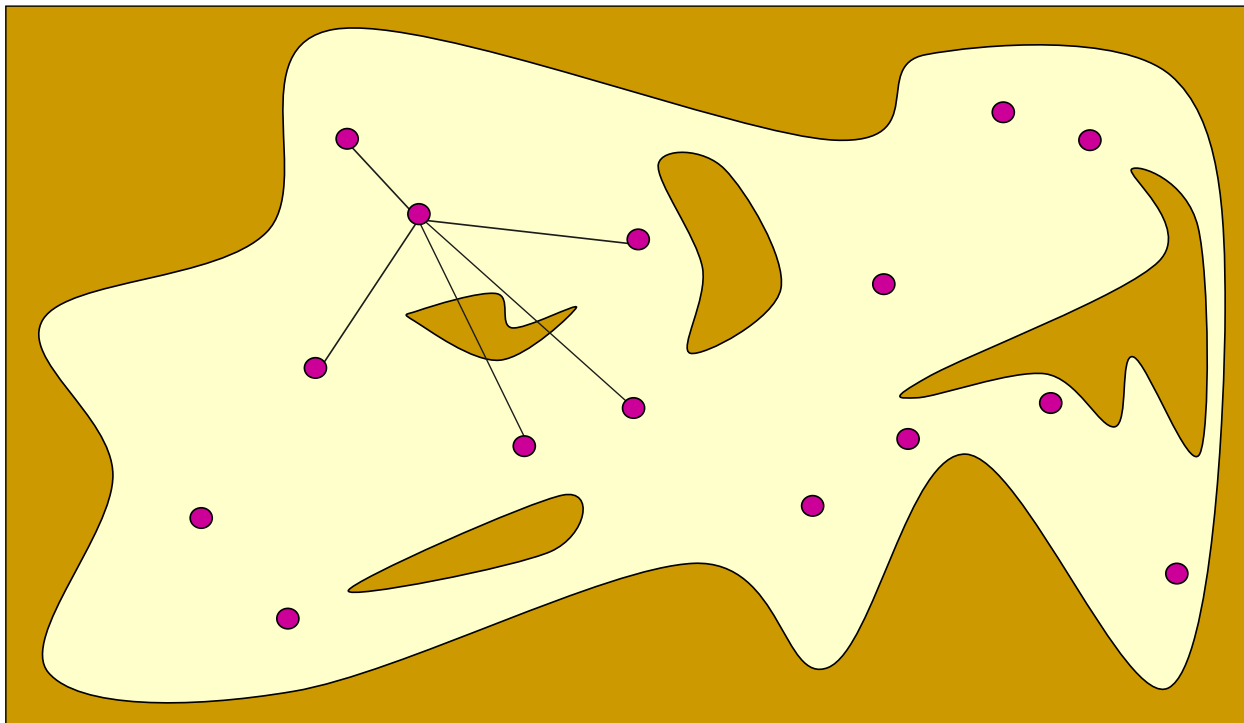
# 1. PRM(Probabilistic Roadmap)

无碰撞姿态成为图节点



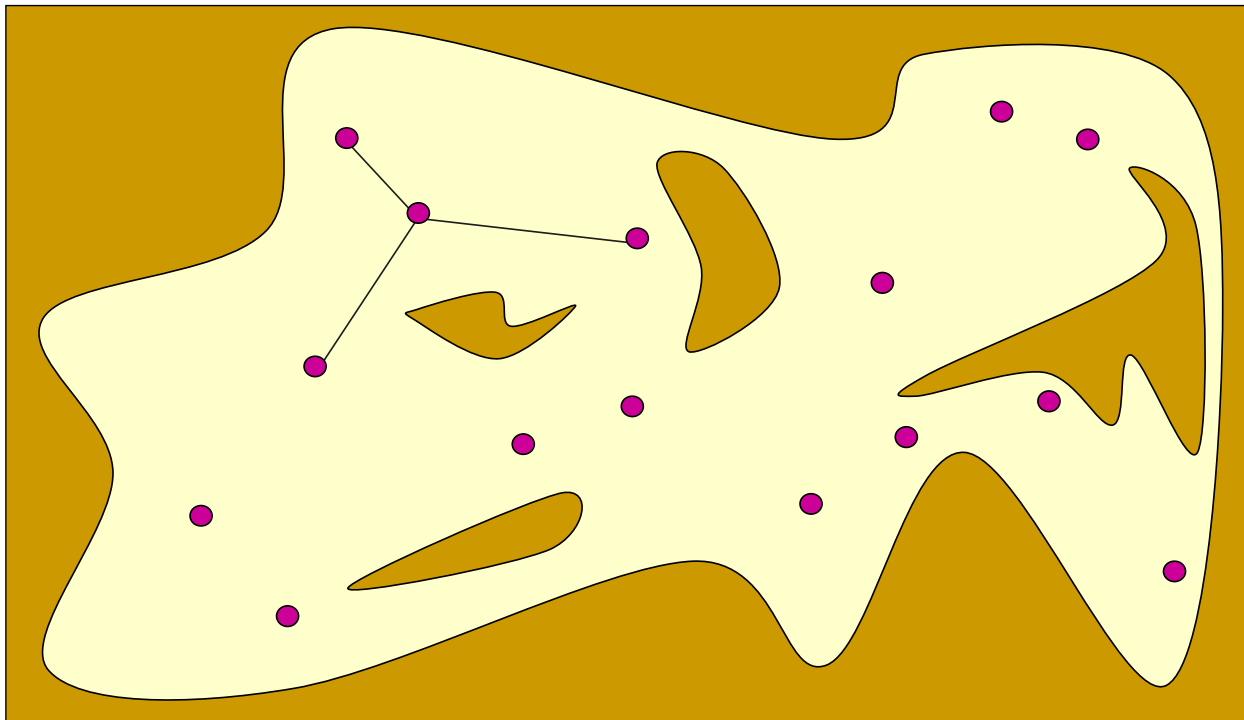
# 1. PRM(Probabilistic Roadmap)

每个图节点和其最近相邻的 $k$ 个节点直线连接



# 1. PRM(Probabilistic Roadmap)

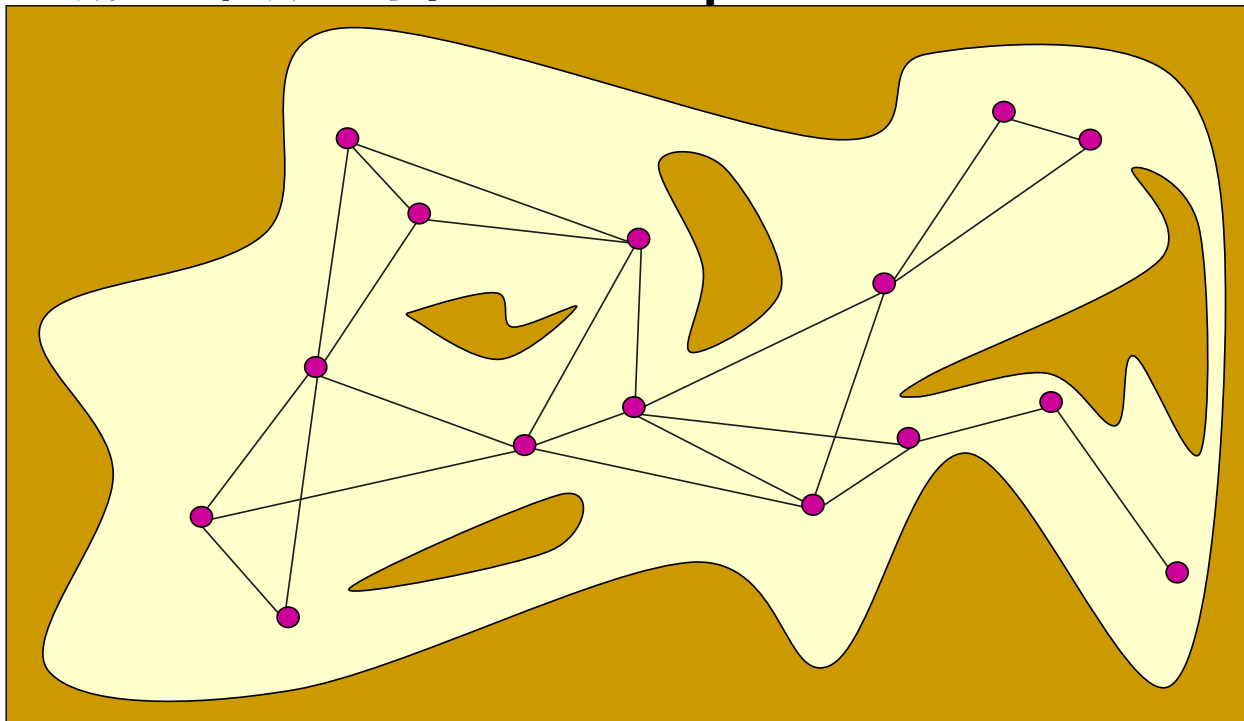
保留无碰路径为图的边





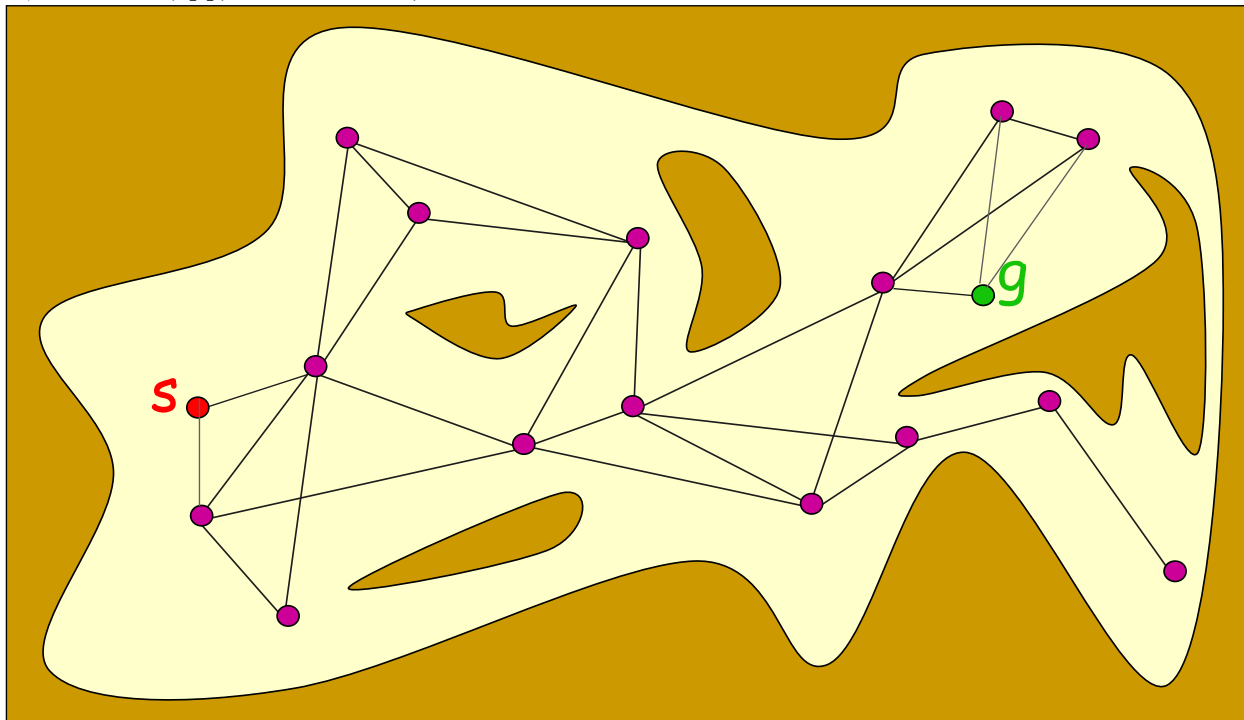
# PRM(Probabilistic Roadmap)

构成自由位形空间中的Roadmap



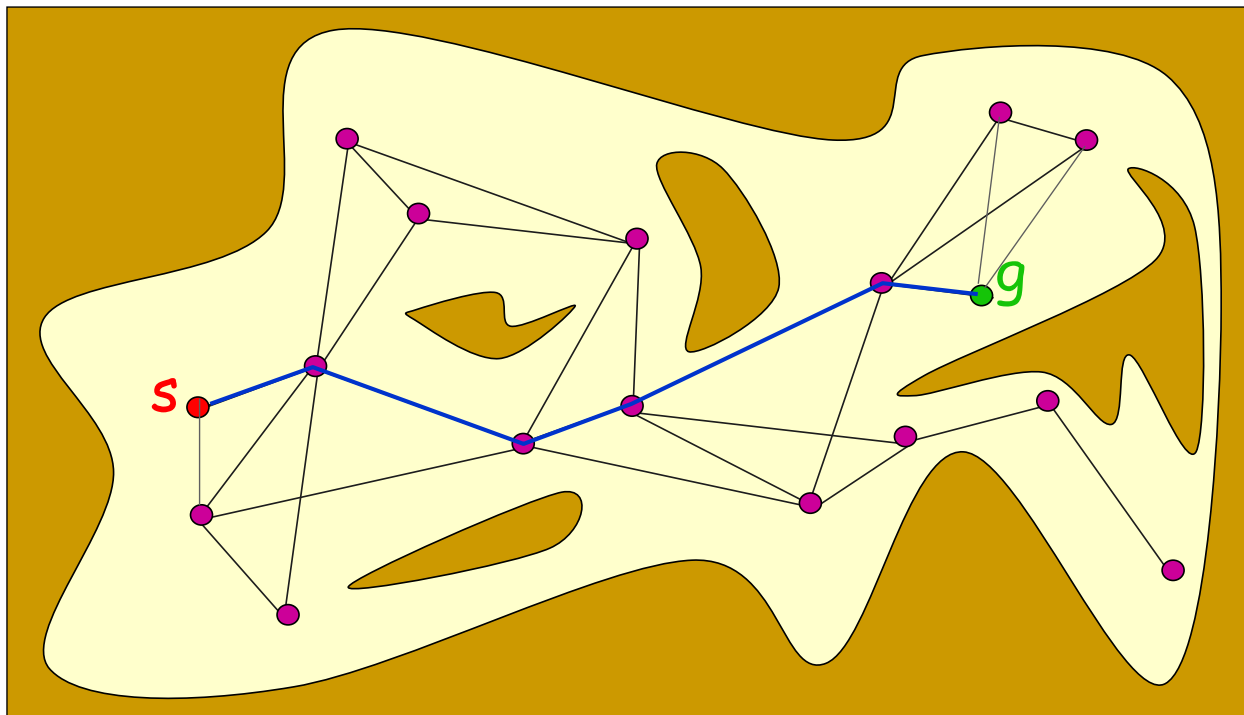
# 1. PRM(Probabilistic Roadmap)

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



---

**Algorithm 6** Roadmap Construction Algorithm

---

**Input:**

$n$  : number of nodes to put in the roadmap

$k$  : number of closest neighbors to examine for each configuration

**Output:**

A roadmap  $G = (V, E)$

---

```
1:  $V \leftarrow \emptyset$ 
2:  $E \leftarrow \emptyset$ 
3: while  $|V| < n$  do
4:   repeat
5:      $q \leftarrow$  a random configuration in  $\mathcal{Q}$ 
6:   until  $q$  is collision-free
7:    $V \leftarrow V \cup \{q\}$ 
8: end while
9: for all  $q \in V$  do
10:   $N_q \leftarrow$  the  $k$  closest neighbors of  $q$  chosen from  $V$  according to  $dist$ 
11:  for all  $q' \in N_q$  do
12:    if  $(q, q') \notin E$  and  $\Delta(q, q') \neq \text{NIL}$  then
13:       $E \leftarrow E \cup \{(q, q')\}$ 
14:    end if
15:  end for
16: end for
```

---

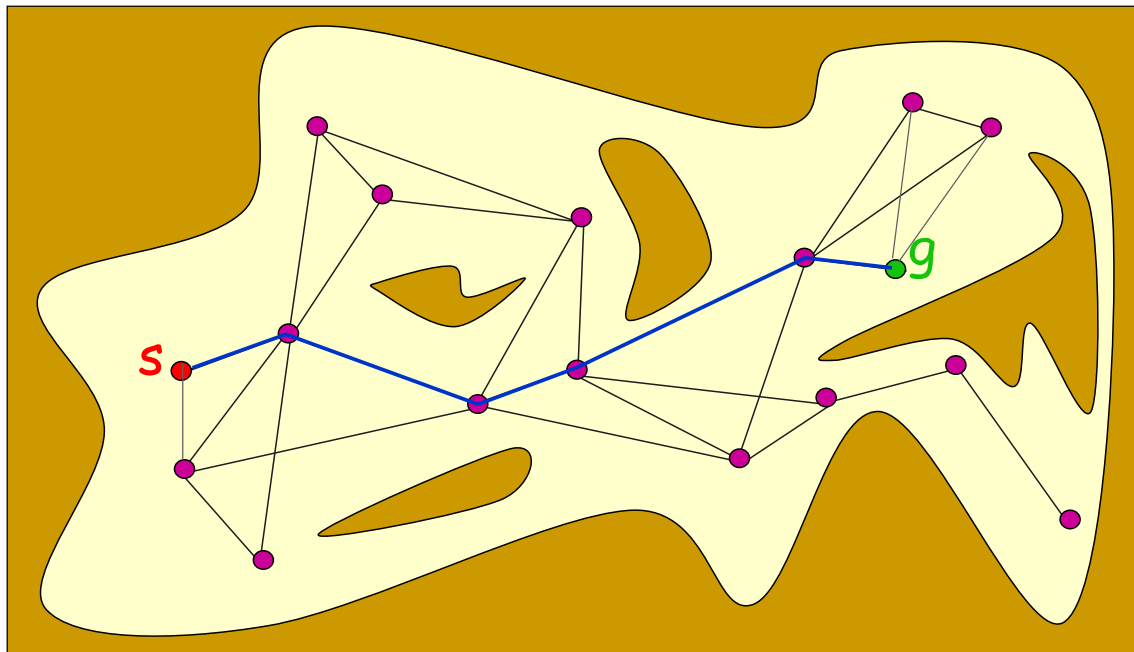
# PRM需要考虑的几个问题

- 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需要考虑的几个问题

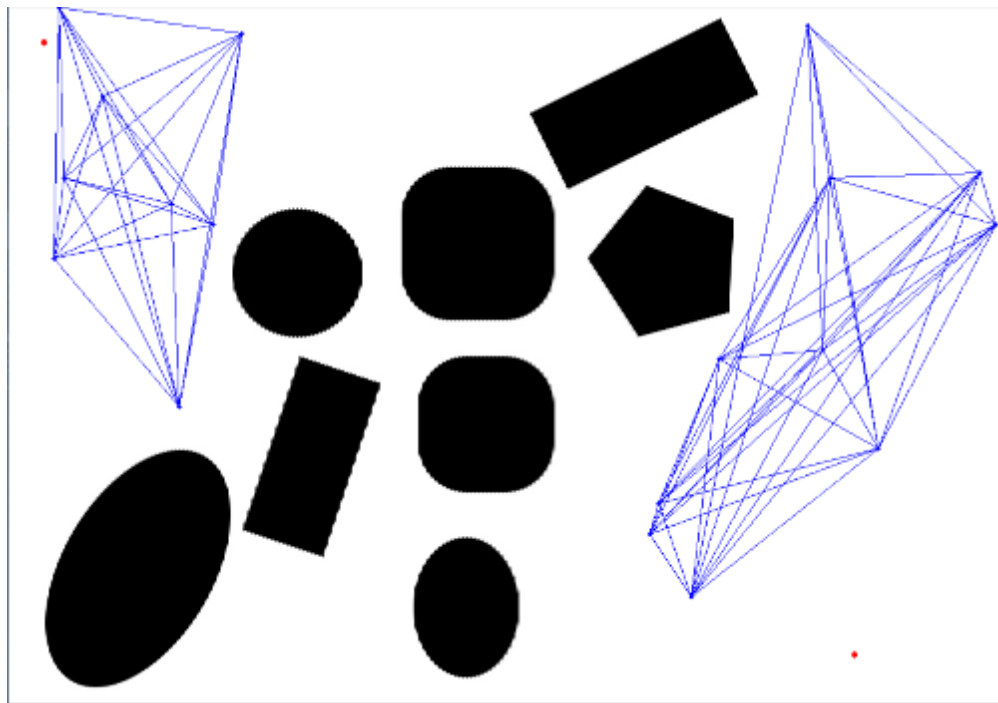
- 进行路径优化



# PRM需要考虑的几个问题

## ○ 非全联通情况

当环境复杂，存在狭窄通道等困难场景时，由于PRM对自由位形空间的近似覆盖，存在图非全联通的情况



问题：请查阅文献找出解决非全联通问题的解决方案

# 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：

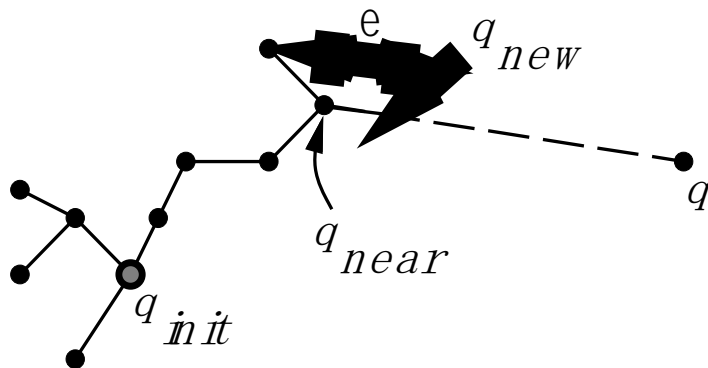
- 对自由空间连通性表达的完整性依赖于采样次数
- 从算法通用性上来讲难以评估需要多少时间做充分采样
- 不考虑机器人执行的可行性





## 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**

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

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

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

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

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

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

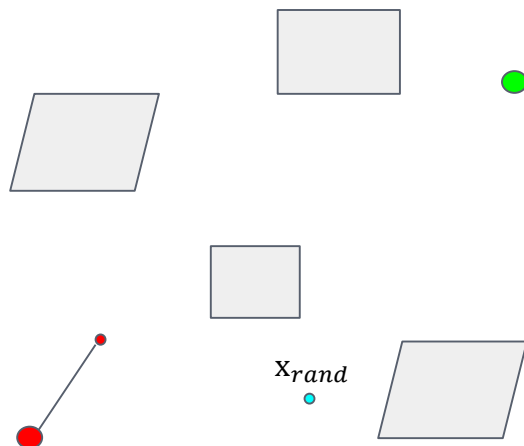
$\mathcal{T}.addEdge(E_i);$

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

        Success();

---

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**

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

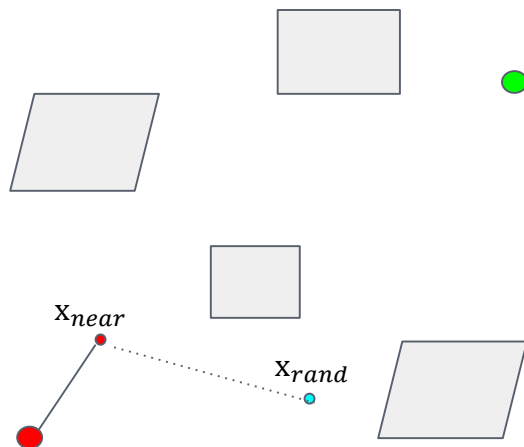
$\mathcal{T}.addEdge(E_i);$

**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**

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

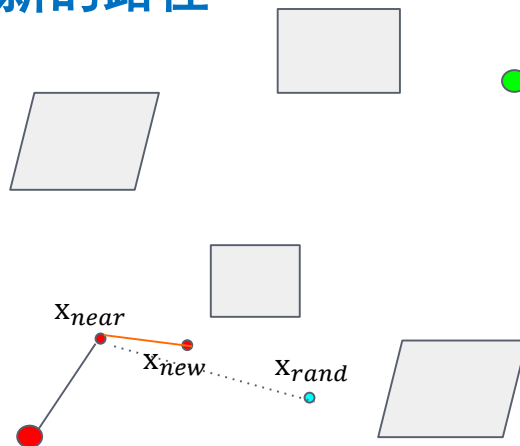
$\mathcal{T}.addEdge(E_i);$

**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**

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

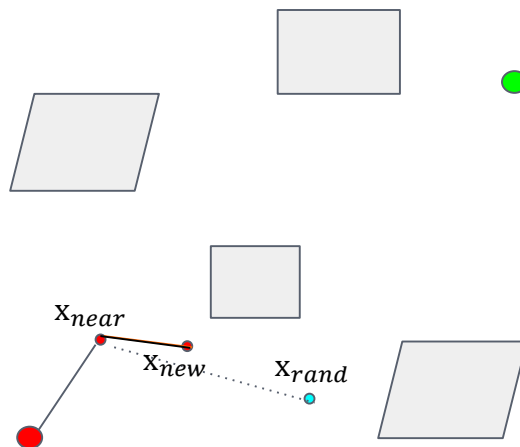
$\mathcal{T}.addEdge(E_i)$ ;

**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**

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

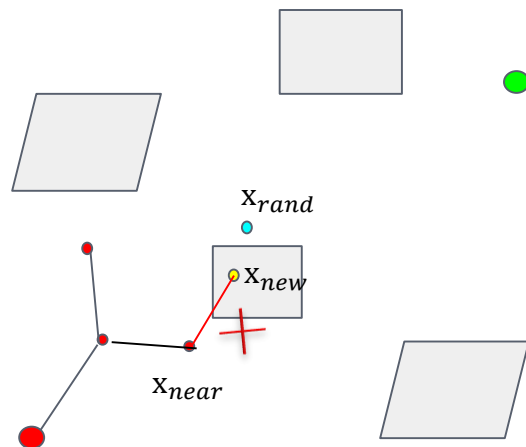
$\mathcal{T}.addEdge(E_i);$

**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**

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

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

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

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

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

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

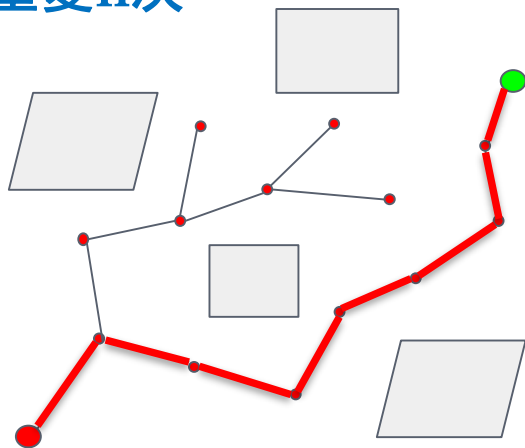
$\mathcal{T}.addEdge(E_i);$

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

        Success();

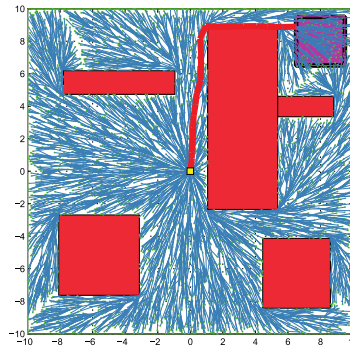
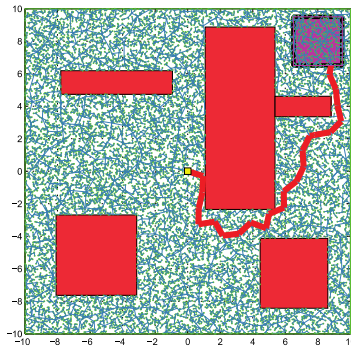
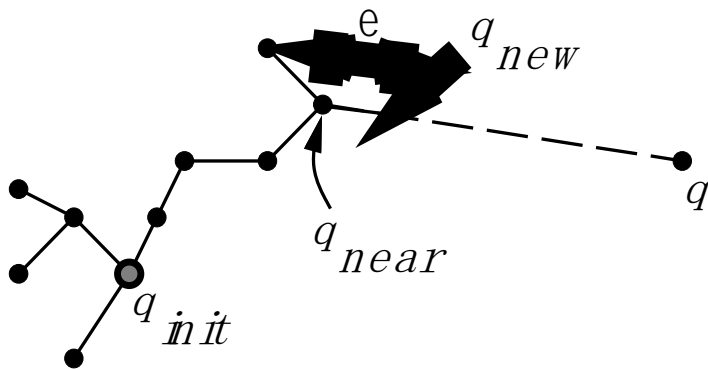
---

直到树节点生长到重点区域，或者重复n次



# RRT\*

- 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\*

- 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**

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


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

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

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

---

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\*

- 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**

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

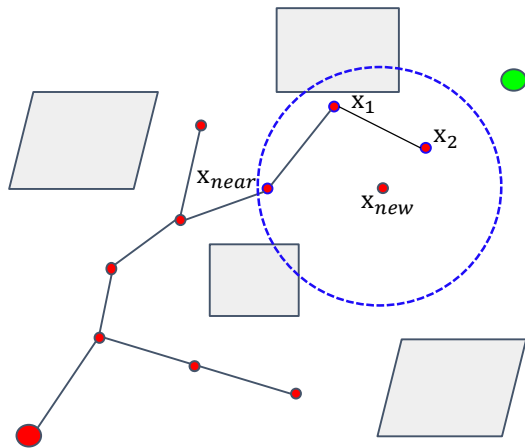
$x_{min} \leftarrow ChooseParent(X_{near}, x_{near}, x_{new});$

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

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

---

选择多个邻节点



# RRT\*

- 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**

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

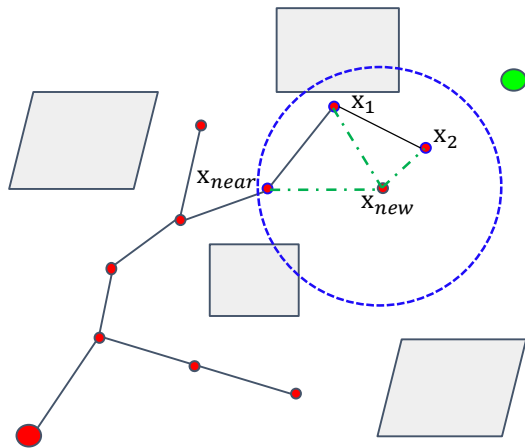
$x_{min} \leftarrow \text{ChooseParent}(X_{near}, x_{near}, x_{new});$

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

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

---

评估各邻节点作为父节点下的总路径长度



# RRT\*

- 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**

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

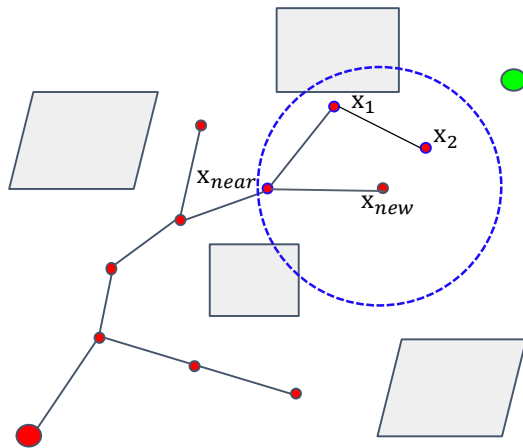
$x_{min} \leftarrow \text{ChooseParent}(X_{near}, x_{near}, x_{new});$

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

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

---

根据总最短路径选择父节点



# RRT\*

- 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\*

- 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**

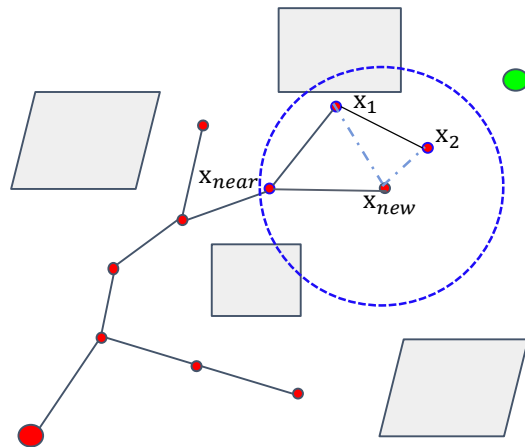
$X_{near} \leftarrow \text{NearC}(\mathcal{T}, x_{new});$

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

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

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

---





# RRT\*

- 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**

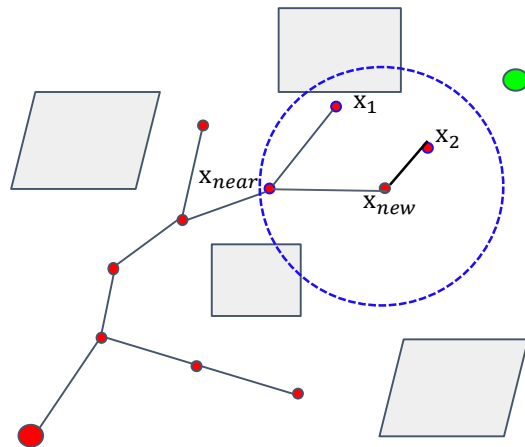
$X_{near} \leftarrow \text{NearC}(\mathcal{T}, x_{new});$

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

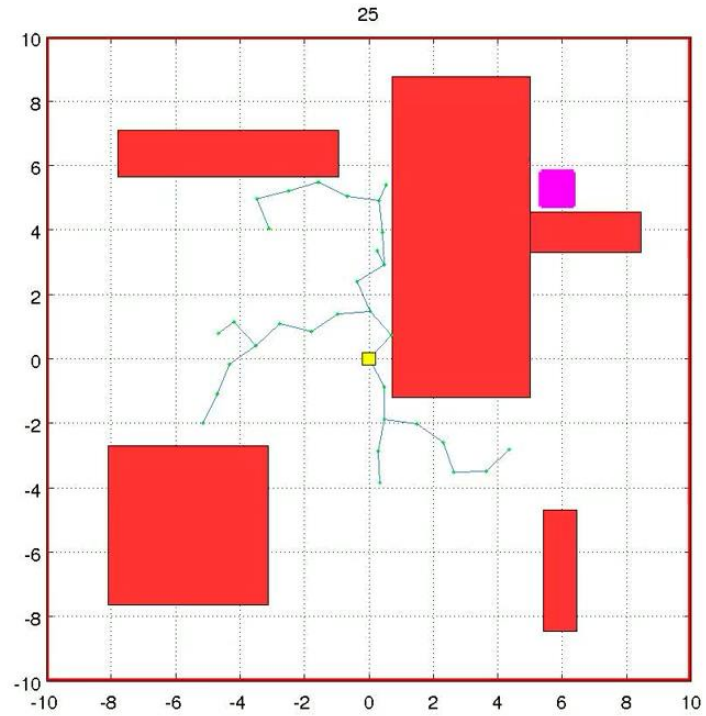
$\mathcal{T}.addNodeEdge(x_{min}, x_{new});$

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

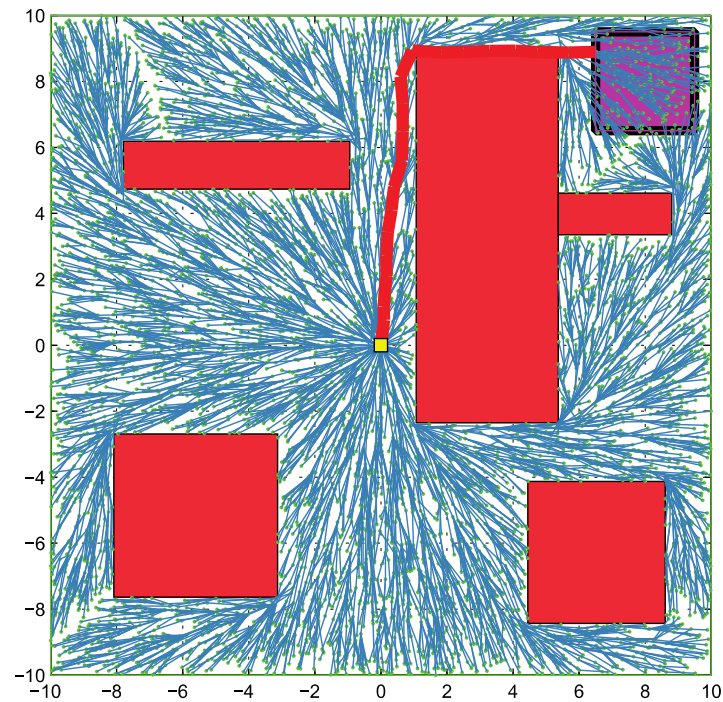
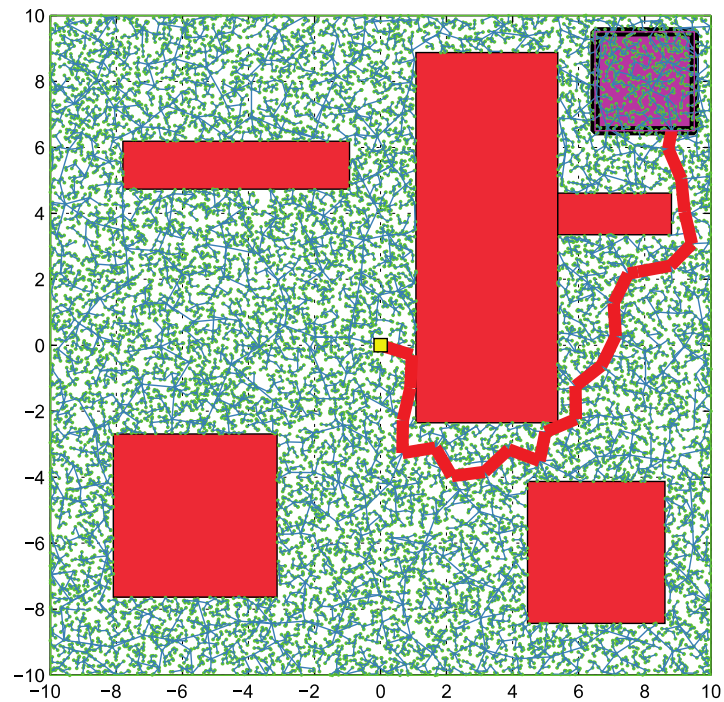
---



# RRT\*



# RRT\*



# 分辨率完备与概率完备方法比较

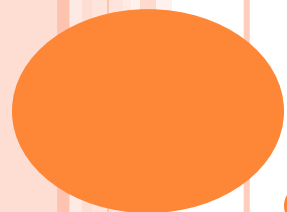
## ○ 空间离散采样：

- 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!**