

机器人原理与应用

Principles and Applications of Robotics

Instructor: Dr. 阎华松(Huasong Min). Professor

Office Location: 钢铁楼1110(Room No. 1110, GANGTIE Building)

Class venue: Room No. F4201

Email: mhuasong@wust.edu.cn

Website: <https://github.com/mhuasong/Basics-of-Robotics-Theory-and-Technology>

Mobile:

Textbooks

➤ Textbook:

- **Robot Modeling and Control**, by M.W. Spong, S. Hutchinson, M. Vidyasagar (2005) (required)
机器人建模与控制 (中文译本)
- **Springer handbook of robotics**. Siciliano, Bruno, and Oussama Khatib, eds. Springer, 2016.
机器人手册 (中文译本)
- **Modelling and Control of Robot Manipulators** (Second Edition), L. Sciavicco and B. Siciliano, Springer-Verlag, London, 2000.
- **Robotics: Modelling Planning and Control**, B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo, Springer-Verlag, London, 2008.
- **Modern Robotics. Mechanics, Planning and Control**(现代机器人生学)



CONTENTS

The course is divided between the following areas

绪论

Introduction and Conceptual Problems

机器人系统分析基础

System Model of Robot

运动学

Robot Kinematics

动力学

Robot dynamics

机器人运动规划

Robot motion planning

机器人控制

Robot control

This is an introductory of robotics course, containing both fundamental as well as some more advanced concepts. It presents a broad overview of robotics with focus on manipulators rather than mobile robots, and includes robot kinematics, dynamics, planning and control.

The course is divided between the following areas:

Robotics Introduction

System Model of Robot

Robot kinematics

Robot dynamics

Robot control

Robot motion planning

...

运动规划



- 机器人位形空间表达方法
 - 位形空间障碍 C_{obs} 、到障碍物的**距离与碰撞检测**、图**Graph**和树**Tree**
- 轨迹生成算法（基于时间尺度）
 - 点到点轨迹生成(**线性插值**、**三次多项式插值**、**五次多项式插值**、**梯形加速与S型曲线**)
 - 具有中间路点的轨迹生成
 - 轨迹逼近(**三阶贝塞尔曲线**、**n阶贝塞尔曲线**、**B样条曲线**等)、严格经过路点(**Catmull-Rom曲线**)
- 位形空间离散化规划方法 (C-Space Discretizations)
 - 组合规划方法 (Combinatorial planning)
 - 四种常用的地图表示方法：可视图(Visibility graphs)、Voronoi图(Voronoi diagrams)、精确单元分解ECD(Exact cell decomposition)、近似单元分解ACD(Approximate cell decomposition)
 - 图搜索算法：Dijkstra算法、广度优先搜索、A⁺搜索、D⁺搜索等
 - 基于采样的规划方法 (Sampling-Based Planning)：PRM、RRT
 - 势场法(Potential Field Methods)
 - Dynamic Window Approach (DWA)
 - 运动轨迹平滑
 - 全局规划和局部规划



引言

机器人运动规划主要研究机器人在n维空间的运动及避障问题，包含路径规划和轨迹规划。

将运动规划问题表达为三个方面的问题：

- 机器人环境的表示方法，即机器人环境建模(位形表达方法:C-space、C-free、C-Obstacle)；
- 其次是如何在这个环境中寻找无碰撞的可行轨迹
(这个轨迹可能采用某种等式（本教程尽可能统一到时间尺度函数）直接生成，也可能是基于某种搜索算法得到）；
- 最后可能还需要对求解得到的轨迹进行处理与平滑等优化。

引言

运动规划的基本概念

完整机器人 vs 非完整机器人



复合机器人

在前面的运动学章节中，我们描述机器人运动空间的时候，基于是否有非完整运动学约束，定义了“非完整机器人”、“完整机器人”的概念。所谓“完整机器人”，指的是没有非完整运动学约束的机器人，反之，一个非完整机器人具有一个或多个非完整运动学约束。

完整的运动学约束可以被表示为位形变量的显函数，而非完整运动学约束则需要微分关系，通常是非可积的。如果机器人受非完整约束时，由于其不可积性，运动位形约束比较困难，无法通过位形变量的显函数直接控制，需要通过设计某种控制率 (control law) 来减小运动的位形误差。

引言

运动规划的基本概念

Introduction-Problem Formulation

目的(Goals)

- ★ 寻找一条无碰轨迹(Collision-free trajectories)
- ★ 最优轨迹(Ex: Robot should reach the goal location as fast as possible?)

◆ The **problem of motion planning** can be stated as follows. Given:

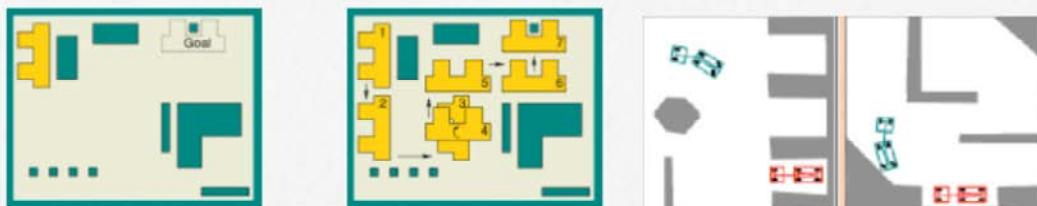
- A **start Configuration**(pose) of the robot
- A desired **goal Configuration**(pose)
- A geometric description of the **robot /Obstacle**(rigid?dot?)
- A geometric description of the **world** (**change to C-space C_{obs} C_{free}**)
- Find a path that moves the robot gradually from **start** to **goal** while **never touching** any **obstacle**
- Planning the configuration $\theta(t)$ 、 velocity $\dot{\theta}(t)$ 、 Acceleration $\ddot{\theta}(t)$ with time scaling



引言

运动规划的基本概念

Introduction-Problem Formulation



Motion planning is sometimes also called **piano mover's problem**

引言

运动规划的基本概念

Introduction-Problem Formulation

■ Path planning

- 所谓机器人路径，指机器人从初始位形到目标位形的位形序列，它只是在纯几何尺度上描述这个位形序列。
- 计算这个路径的策略称之为路径规划(path planning)。机器人的路径规划问题被描述成，已知机器人的几何描述与环境（机器人工作空间以及障碍物），求解机器人从初始位形运动到目标位形的一条路径，同时机器人在运动的过程中不会碰到任何障碍。

■ Trajectory = Path + assignment of time to points along the path

- 轨迹则是指机器人从初始位形到目标位形的位形序列和时间标度(time scaling)的组合，不仅包含几何尺度的描述，还包含时间约束。

■ Trajectory planning

path planning + design of linear and angular velocities(also acceleration)

- 轨迹则是指机器人从初始位形到目标位形的位形序列和时间标度(time scaling)的组合，不仅包含几何尺度的描述，还包含时间约束。
- 轨迹规划表示为针对机器人以某种前向速度（包括加速度）从初始位形运动到目标位形的路径求解，并且加入了时间约束。

■ Motion Planning (MP), a general term, either:

- Path planning, or
- Trajectory planning

■ Path planning < Trajectory planning

所谓机器人路径，指机器人从初始位形到目标位形的位形序列，它只是在纯几何尺度上描述这个位形序列。计算这个路径的策略称之为路径规划(path planning)。机器人的路径规划问题被描述成，已知机器人的几何描述与环境（机器人工作空间以及障碍物），求解机器人从初始位形运动到目标位形的一条路径，同时机器人在运动的过程中不会碰到任何障碍。

路径规划按其策略地图的适应范围，可分为全局路径规划和局部路径规划。得到规划路径后，需要我们根据实际的机器人运动学模型和约束，设计适当的路径跟随控制算法，将可行路径转化为可行轨迹，使机器人能鲁棒地跟随期望路径。

轨迹则是指机器人从初始位形到目标位形的位形序列和时间标度(time scaling)的组合，不仅包含几何尺度的描述，还包含时间约束。

轨迹规划表示为针对机器人以某种前向速度（包括加速度）从初始位形运动到目标位形的路径求解，并且加入了时间约束。同样的，得到规划轨迹后，需要我们根据实际的机器人运动学模型和约束，设计适当的轨迹跟踪控制算法，基于速度控制、位置控制以及航向角控制等方式，控制机器人实现期望的运动。

引言

运动规划的基本概念

运动规划问题表达为三个方面的问题



C-Space

首先是机器人环境的表示方法，即机器人环境建模。



Trajectory

其次是如何在这个环境中寻找无碰撞的可行轨迹（这个轨迹可能采用某种等式（本教程尽可能统一到时间尺度函数）直接生成，也可能是基于某种搜索算法得到）。



Smoothing

最后可能还需要对求解得到的轨迹进行处理与平滑等优化。

02

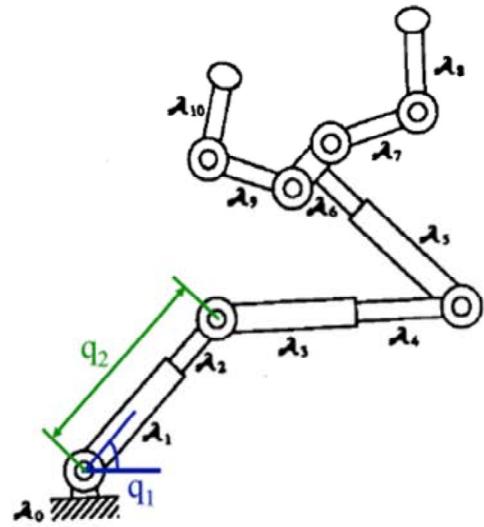
机器人位形空间表达方法

机器人环境建模，需要将实际环境的物理空间抽象成可以进行算法处理的数字模型空间。由于早期的移动机器人通常只考虑2D环境下的运动，所以机器人环境建模延续了计算机学科的建模方法，主要有栅格图、Voronoi图、拓扑图、概率图等表示方法；现代机器人学通常采用**位形**来描述机器人模型。位形空间C-Space采用完整精确的表达，还是离散化表达？

机器人位形空间表达方法

C-Space representation

多关节机器人位形



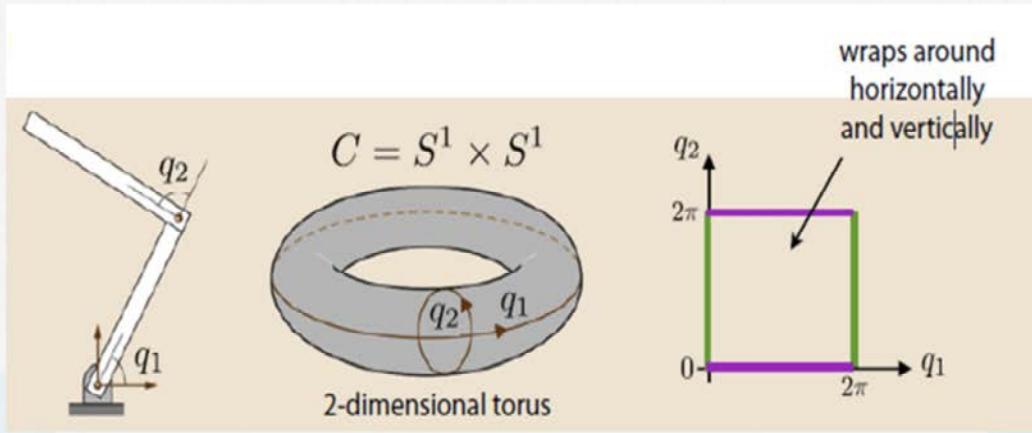
$$q = (q_1, q_2, \dots, q_{10})$$

关节变量对应唯一的位形（机器人正解是唯一的!）

机器人位形空间表达方法

C-Space representation

平面2R机械臂位形空间



包含所有可能机器人位形的n维空间定义为位形空间(C-Space)

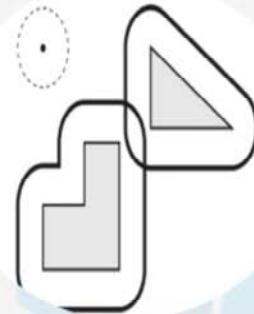
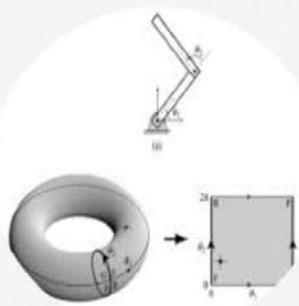
包含所有可能机器人位形的 n 维空间定义为位形空间(C-Space)。此空间的拓扑通常不是笛卡尔空间的拓扑，如图5.2所示，一个平面2R机械臂，其工作空间为一个环面(torus)，等效为两个一维球的笛卡尔积 $C = S^1 \times S^1$ ，而位形空间C-Space描述为一种拓扑流形，通过在 $q_1 = 0$ 和 $q_2 = 0$ 的地方将环面切割两次，将其置于平面得到的是一个 \mathbb{R}^2 内的正方形区域。

机器人位形空间表达方法

C-Space representation

位形空间C-Space

$$C = C_{free} \cup C_{obs}$$



C-Space

C_{free}

C_{obs}

包含所有可能机器人位形的n维空间定义为位形空间(C-Space)。

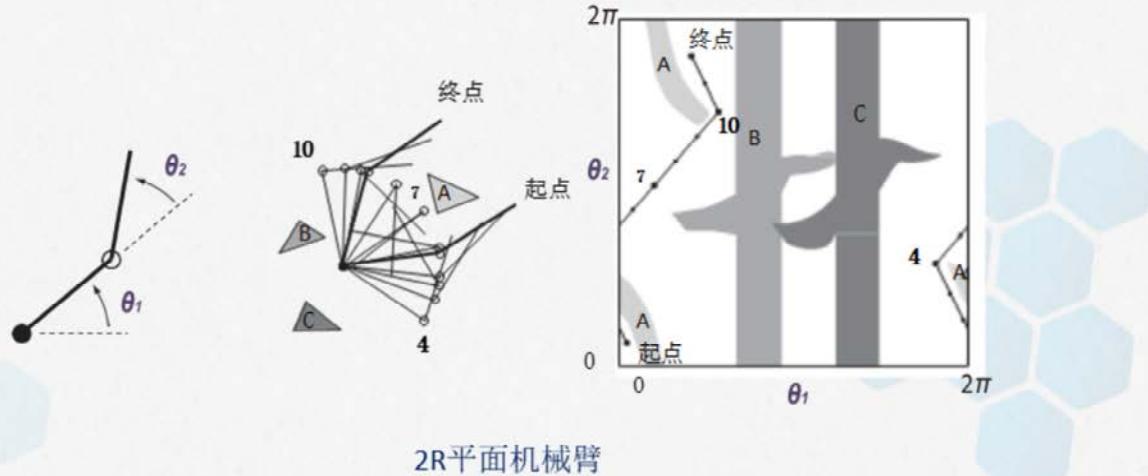
考虑机器人工作空间中的障碍物，我们将位形空间C划分为两部分，即自由空间 C_{free} 和障碍空间 C_{obs}

对于关节限位可被当作位形空间中的障碍物来处理。将机器人缩小成C空间中的一个点，相反将障碍物按一定规则进行适当膨胀放大。

机器人位形空间表达方法

C-Space representation

位形空间C-Space



如果障碍物将自由空间 C_{free} 划分为相互隔离的联通分支，并且起始位形 q_{start} 和目标位形 q_{goal} 不处在同一个联通分支内，那么这条轨迹就是一条无碰轨迹。

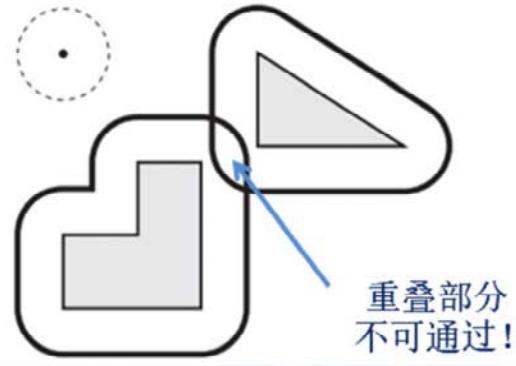
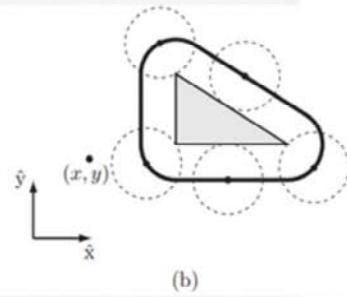
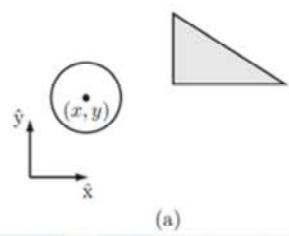
C空间障碍物的明确数学表示可能会十分复杂，因此我们很少使用其精确表示，但C空间障碍物的概念对于运动规划算法的理解非常重要，下面使用一些实例进行说明。

图5.3给出的2R平面机械臂，其位形为 $q = (\theta_1, \theta_2)$ ，在其工作空间中有障碍物A、B、C。图5.3(b)示意了工作空间的无碰撞自由路径，。图5.3(c)示意了C-空间的无碰撞自由路径。注意到，障碍物将自由空间 C_{free} 划分为3个联通分支。

机器人位形空间表达方法

C-Space representation

位形空间C-Space



圆形平面移动机器人

如图5.4(a)所示的平面圆形移动机器人（用圆表示）和障碍物（用灰色的三角形表示），C空间障碍物就是以机器人的尺寸（半径）为基础，绕障碍物对其进行一定的膨胀即可得到，障碍物几何尺寸被沿边沿增长了机器人的半径，同时机器人被当做为一个点进行处理，图图5.4(b)粗线外的任何位形 (x, y) 都是无碰撞的。

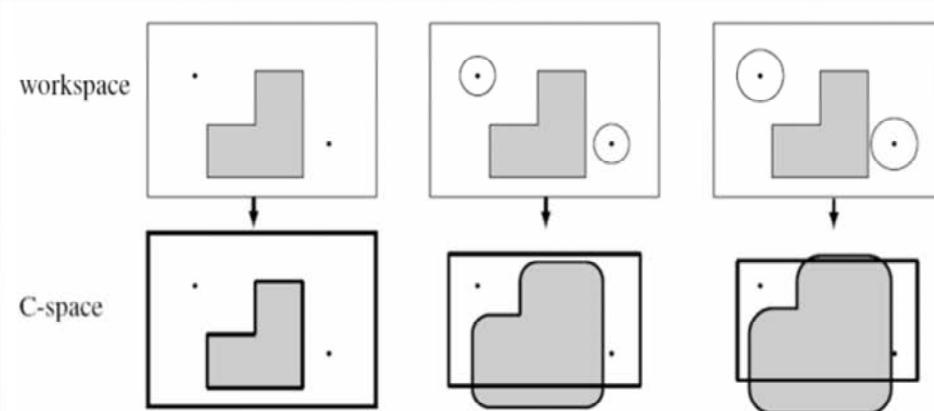
如图5.5所示，如果对工作空间中的障碍物进行膨胀后产生了重叠的部分，则表示机器人不能通过那个区域。

机器人位形空间表达方法

C-Space representation

位形空间C-Space

障碍物膨胀后C-Space与工作空间的关系



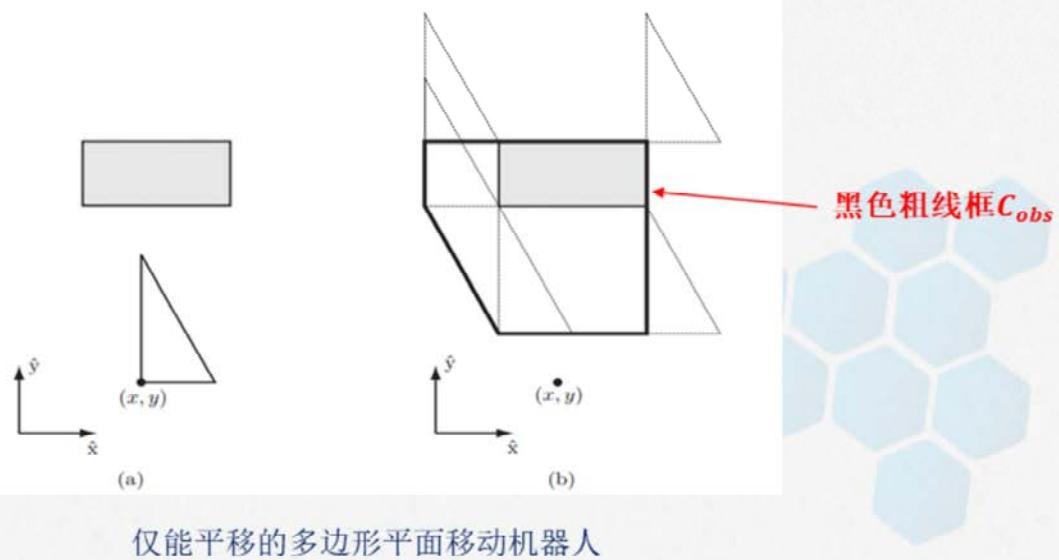
C-Space中的白色区域表示可通过的自由区域 C_{free}

C-Space与机器人实际形状和尺寸有关，从图5.6可以看出中间和右边的两个机器人由于尺寸（直径）太大而不能通过工作空间的某些区域（C-Space中的白色区域表示可通过的自由区域 C_{free} ）。

机器人位形空间表达方法

C-Space representation

位形空间C-Space



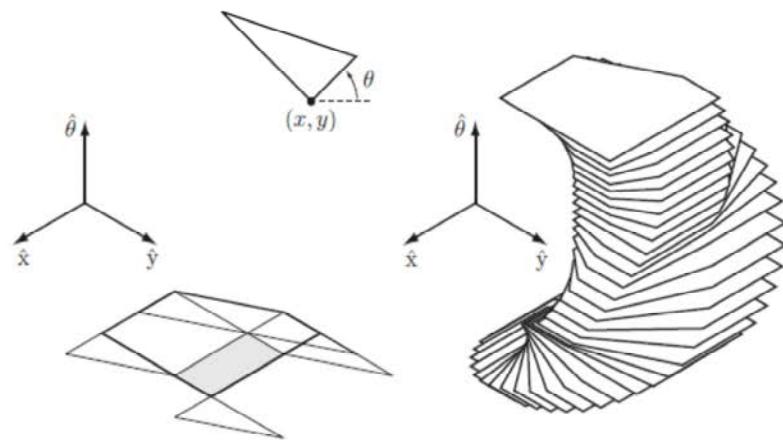
仅能平移的多边形平面移动机器人

如图5.7所示，机器人的形状也会影响C-Space。图5.7(a)中的直角三角形代表一个三角形状的机器人，其只能在平面内平移，其参考位置点在图中的 (x, y) 处。灰色矩形是工作空间内的障碍物，将机器人贴着障碍物边界滑行一圈，追踪参考点的位置可以画出一个黑色粗线框的多边形区域如(图5.7(b)所示)。这个区域就是该三角形机器人C-Space中的障碍区域。

机器人位形空间表达方法

C-Space representation

位形空间C-Space



能同时平移和旋转的平面多边形移动机器人

如图5.7所示，如果还允许机器人旋转，那么C-Space就是三维的，其C-Space由 $(x, y, \theta) \in \mathbb{R}^2 \times S^1$ 给出，三维C-空间障碍是角度为 $\theta \in [0, 2\pi)$ 的二维障碍物切片组成的并集，如图5.7右边所示。

从上面的例子中可知，即使是维度相对较低的C-Space，C-障碍物的精确表示也非常复杂。因此，我们实际很少对C-空间障碍进行精确表述。

到障碍物的距离

- 已知位形空间障碍物 B 和机器人位形 q , 定义 $d(q, B)$ 为机器人与障碍物之间的距离:

➤ $d(q, B) > 0$, 机器人与障碍物之间没有接触;

➤ $d(q, B) = 0$, 机器人与障碍物之间有接触;

➤ $d(q, B) < 0$, 机器人与障碍物之间发生了碰撞侵入。



机器人位形空间表达方法

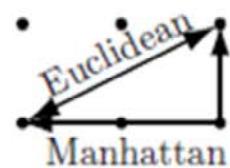
C-Space representation

到障碍物的距离

平面直角坐标系 $|AB| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ 欧氏距离

三维空间 $|AB| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$

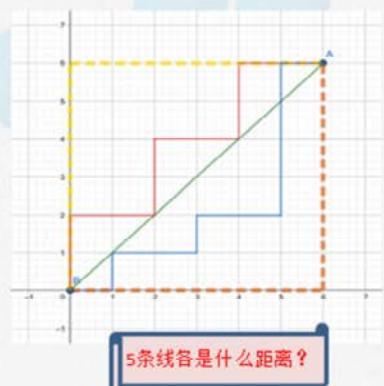
n 维空间 $\|\vec{AB}\| = \sqrt{\sum_{i=1}^n (x_{1i} - x_{2i})^2}$ $\vec{A}(x_{11}, x_{12}, \dots, x_{1n})$
 $\vec{B}(x_{21}, x_{22}, \dots, x_{2n})$



二维空间 $d(A, B) = |x_1 - x_2| + |y_1 - y_2|$ 曼哈顿距离

n 维空间 $d(A, B) = \sum_{i=1}^n |x_{1i} - x_{2i}|$

Chebyshev distance?



如图5.8所示，在A、B之间，黄线、橙线都表示曼哈顿距离，而红线、蓝线表示等价的曼哈顿距离，绿线表示欧氏距离。

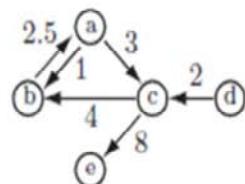
相比较曼哈顿距离与欧氏距离这两种几何距离，切比雪夫距离（Chebyshev distance）是向量空间中的一种度量，其定义二个点之间的距离为其各坐标数值差的最大值。

机器人位形空间表达方法

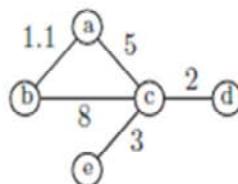
C-Space representation

图和树

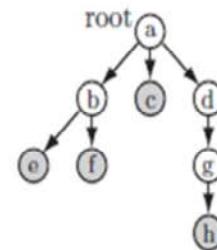
表达
C-空间 \rightarrow 图(graph)



(a)



(b)



(c)

加权有向图

加权无向图

树

在机器人运动规划中，我们可以将C-空间表达为一个图(graph)。一个图由 N 个节点和 E 条边组成，其中每条边 e 连接两个节点。节点表示位形，而节点 n_1 和 n_2 之间的边表示在不穿透障碍物或违反其他约束条件下从 n_1 和 n_2 的直接运动。

图可以是有向的，也可以是无向的，无向图的边是双向的，而有向图则仅允许沿一个方向运动，对于特点的两个节点而言，它们之间可以有两条边，允许沿相反方向行进。

图也可以是加权的或不加权的，在加权图中，每条边具有与遍历该边相关的正成本，而在不加权图中，每条边的成本相同。

树(tree)是一种特定的有向图：没有循环，且每个节点最多只能有一个父节点。树有一个没有父节点的根(root)节点和一些没有子节点(child)的叶子节点(leaf)。

图5.9为加权有向图、加权无向图和树的实例。将C-Space表达为图之后，我们可以采用计算机科学中的图搜索算法寻找机器人路径，进行运动规划。

03

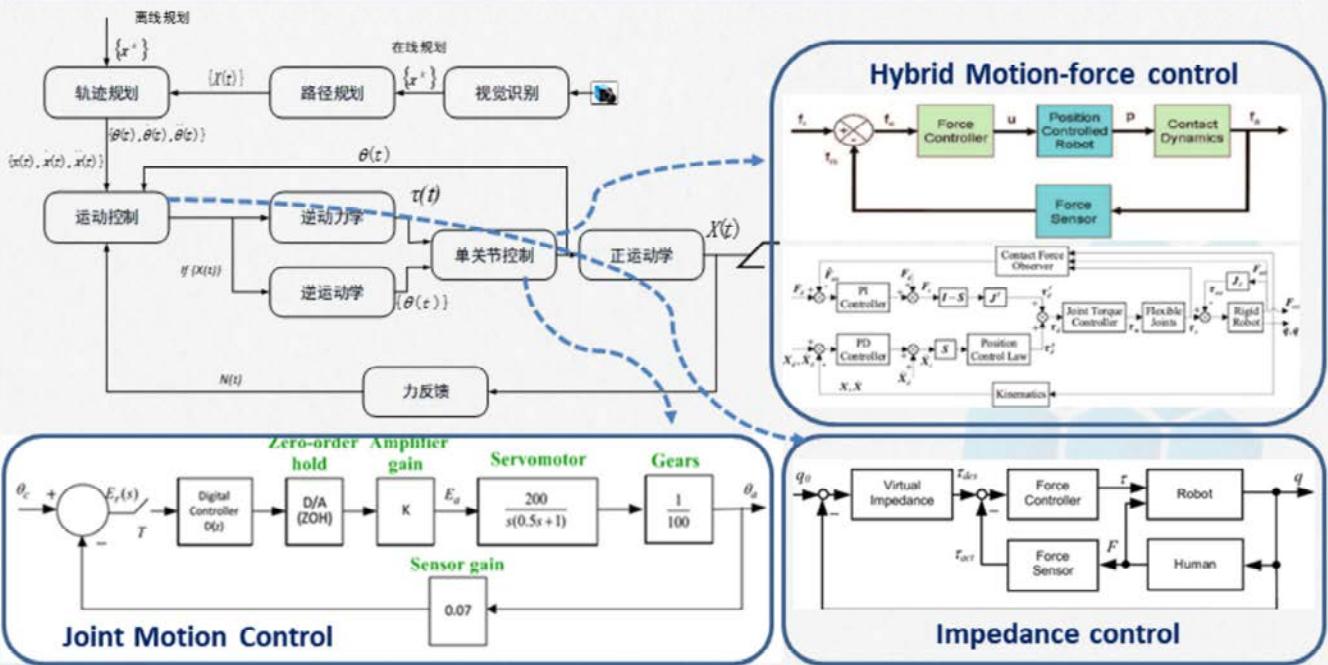
轨迹生成算法

早期机器人运动规划研究，基本集中在机械臂的轨迹生成算法方面，其采用直线或曲线公式直接生成轨迹，直接规划机械臂的位形、速度及加速度。

轨迹生成算法

Trajectory Generation

机器人运动规划与控制之间的关系



如图5.10所示，工业机械臂的作业路径 $\{x^k\} \in \mathbb{R}^k$ ，可以是离线预设好的，也可能由视觉等传感器获取。该路径将由轨迹规划算法进行求解，给出关节的位置、速度、加速度控制量（如果是笛卡尔空间轨迹规划，需要用运动学逆解转换到关节变量），由机器人控制器执行，如果需要控制精度，可以采取位置传感器进行反馈，执行位置闭环控制算法，如果对操作力或力矩有要求，可以采用力传感器，结合动力学执行力反馈控制。轨迹规划除了空间轨迹的平滑约束外，还需要满足对机器人的速度、加速度甚至加加速(jerk, 加速度的导数，反应了机器人冲击可能导致的振动)的约束。

轨迹规划的一般性问题

常见的机器人作业有两种：



点位作业 (PTP:
point-to-point motion)

连续路径作业 (**continuous-path motion**)，或者称为轮廓运动
(**contour motion**)

轨迹规划的一般性问题

工业机械臂最常用的轨迹规划方法有两种：

- 第一种是要求对于选定的轨迹节点（插值点）上的位姿、速度和加速度给出一组显式约束（例如连续性和光滑程度等），轨迹规划器从一类函数（例如n次多项式）选取参数化轨迹，对节点进行插值，并满足约束条件；
- 第二种方法则要求给出运动路径的解析式。

如果按机器人位形的表达方式不同，又可分为：

- 笛卡尔空间轨迹规划
- 关节空间轨迹规划。

无论哪种规划，都要求轨迹函数必须连续和平滑，使得工业机械臂的运动平稳。



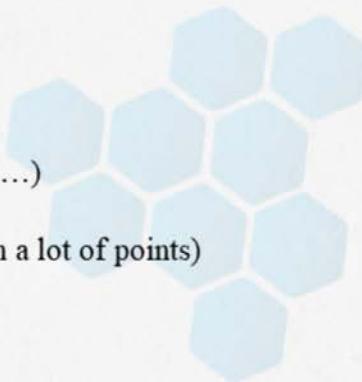
轨迹规划的一般性问题

□ Joint space

- Easy to go through via points
(Solve inverse kinematics at all path points)
- No problems with singularities
- Less calculations
- Can not follow straight line

□ Cartesian space

- We can track a shape
(for orientation : equivalent axes, Euler angles,...)
- More expensive at run time
(after the path is calculated need joint angles in a lot of points)
- Discontinuity problems



轨迹生成算法

Trajectory Generation

点到点轨迹生成

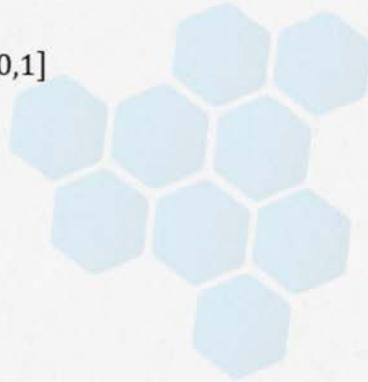
关节空间轨迹生成函数:

$$\theta(s) = \theta_{start} + s(\theta_{end} - \theta_{start}), s \in [0,1]$$

笛卡尔空间轨迹生成函数:

$$X(s) = X_{start} + s(X_{end} - X_{start}), s \in [0,1]$$

s为时间尺度 time scaling function



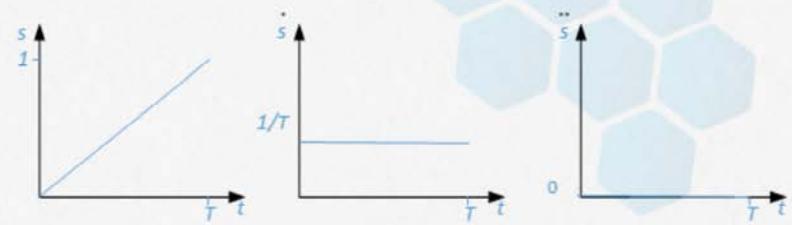


$$s(t) = a_0 + a_1 t$$

起始与终止位置约束 $s(0) = 0, s(T) = 1$

$$a_0 = 0, a_1 = \frac{1}{T}$$

$$\begin{aligned}\theta(t) &= \theta_{start} + \frac{t}{T}(\theta_{end} - \theta_{start}) \\ \dot{\theta}(t) &= \frac{1}{T}(\theta_{end} - \theta_{start}) \\ \ddot{\theta}(t) &= 0\end{aligned}$$



由上面三个位形、速度、加速度规划函数可知，等时间间隔的线性插值仅仅只是位形空间的等时间间隔离散，速度项为常数（平均速度），加速度项为零（代表无法解出）。由于这个时间尺度函数不能添加速度和加速度约束，其并没有规划速度和加速度，所以需要重新规划速度和加速度，保证在起始点速度为零，在终点处速度也为零，并且在中间段前后的速度和加速度连续。

轨迹生成算法

Trajectory Generation

点到点轨迹生成

线性插值



关节空间轨迹生成函数:

统一方法

$$\theta(s) = \theta_{start} + s(\theta_{end} - \theta_{start}), s \in [0,1]$$

笛卡尔空间轨迹生成函数:

$$X(s) = X_{start} + s(X_{end} - X_{start}), s \in [0,1]$$

s为时间尺度 time scaling function

特例: 线性插值

$$s(t) = a_0 + a_1 t$$

起始与终止位置约束 $s(0) = 0, s(T) = 1$

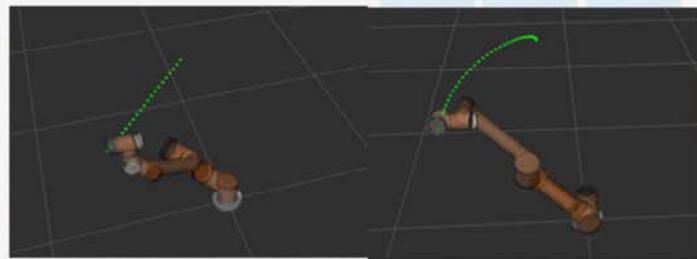
$$a_0 = 0, a_1 = \frac{1}{T}$$

位形、速度、加速度规划函数

$$\theta(t) = \theta_{start} + \frac{t}{T} (\theta_{end} - \theta_{start})$$

$$\dot{\theta}(t) = \frac{1}{T} (\theta_{end} - \theta_{start})$$

$$\ddot{\theta}(t) = 0(\infty)$$



在图5.11中, $s(t)$ 为一条直线, 所以又称直线规划, 实际如果是在笛卡尔空间规划, 末端轨迹是一条直线, 但如果在关节空间规划, 只能保证每个关节的变换是“直线”, 而机器人末端轨迹会是一条圆弧。

轨迹生成算法

Trajectory Generation

点到点轨迹生成

三次多项式插值



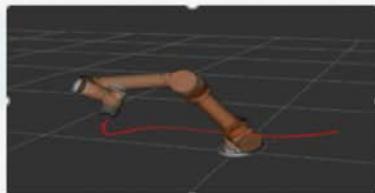
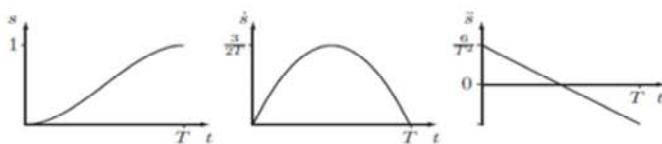
$$s(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

$$s = a_2 t^2 + a_3 t^3$$

四个约束条件：在起始处施加约束 $s(0) = \dot{s}(0) = 0$ ，在终点处施加约束 $s(T) = 1$ 、 $\dot{s}(T) = 0$

$$\ddot{s}(t) = a_1 + 2a_2 t + 3a_3 t^2$$

$$a_0 = 0, a_1 = 0, a_2 = \frac{3}{T^2}, a_3 = -\frac{2}{T^3}$$



$$\theta(t) = \theta_{start} + \left(\frac{3t^2}{T^2} - \frac{2t^3}{T^3} \right) (\theta_{end} - \theta_{start})$$

$$\dot{\theta}(t) = \left(\frac{6t}{T^2} - \frac{6t^2}{T^3} \right) (\theta_{end} - \theta_{start})$$

$$\ddot{\theta}(t) = \left(\frac{6}{T^2} - \frac{12t}{T^3} \right) (\theta_{end} - \theta_{start})$$

$$t = \frac{T}{2}; \quad \dot{\theta}_{max} = \frac{3}{2T} (\theta_{end} - \theta_{start})$$

在 $t = 0$ 和 $t = T$ 时

$$\dot{\theta}_{max} = \left| \frac{6}{T^2} (\theta_{end} - \theta_{start}) \right|, \dot{\theta}_{min} = - \left| \frac{6}{T^2} (\theta_{end} - \theta_{start}) \right|$$

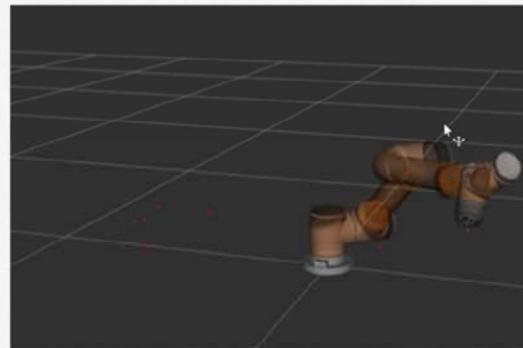
如果机器人的最大关节速度和最大关节加速度存在已知极限，即 $|\dot{\theta}| \leq \dot{\theta}_{limit}$ 、
 $|\ddot{\theta}| \leq \ddot{\theta}_{limit}$ ，则可以检查这些界限以查看所请求的运动时间 T 是否可行。当然据此
 也可以求解 T ，以便找到满足最严格速度或加速度约束的最小可能运行时间。

轨迹生成算法

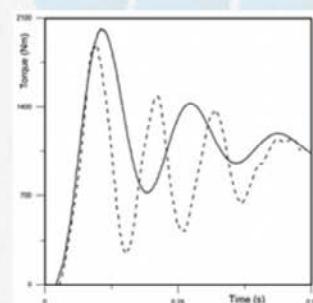
Trajectory Generation

点到点轨迹生成

三次多项式插值



由于三次多项式插值的时间尺度函数没有对端点路径施加约束，使得 $\ddot{s}(0) = \ddot{s}(T) = 0$ ，导致机器人在 $t = 0$ 和 $t = T$ 时刻会产生加速度的不连续跳跃，这意味着存在无限的**加加速**(**jerk**)，即加速度的导数，这可能会导致机器人产生振动。通过在端点处施加 $\ddot{s}(0) = \ddot{s}(T) = 0$ 的约束，采用五次多项式可以解决这个问题。



轨迹生成算法

Trajectory Generation

点到点轨迹生成

五次多项式插值



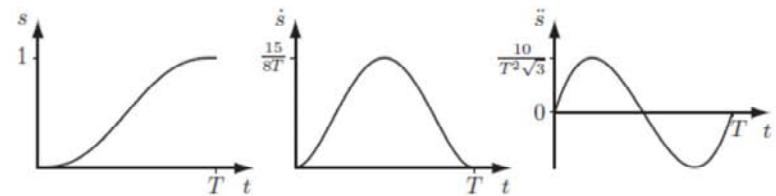
$$s(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5$$

$$\dot{s}(t) = a_1 + 2a_2 t + 3a_3 t^2 + 4a_4 t^3 + 5a_5 t^4$$

$$\ddot{s}(t) = 2a_2 + 6a_3 t + 12a_4 t^2 + 20a_5 t^3$$

$$s(0) = \dot{s}(0) = \ddot{s}(0) = 0 \quad , \quad s(T) = 1, \quad \dot{s}(T) = \ddot{s}(T) = 0$$

$$a_0 = 0, a_1 = 0, a_2 = 0, a_3 = \frac{10}{T^3}, a_4 = -\frac{15}{T^4}, a_5 = \frac{6}{T^5} \quad \rightarrow \quad s = a_3 t^3 + a_4 t^4 + a_5 t^5$$



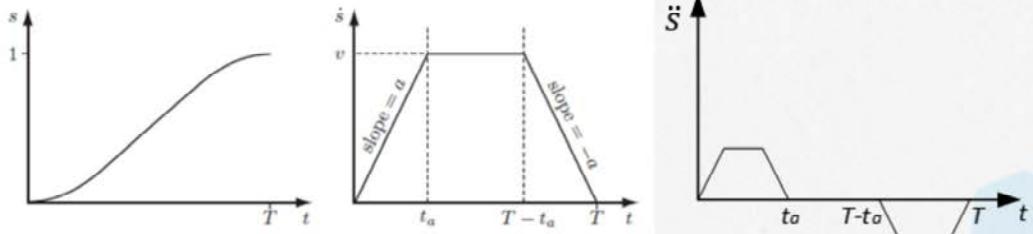
同上，将 $s = a_3 t^3 + a_4 t^4 + a_5 t^5$ 代入式 (5-6)，可以得到五次多项式下的位形、速度、加速度规划具有时间约束的函数 $\theta(t)$ 、 $\dot{\theta}(t)$ 、 $\ddot{\theta}(t)$ ，与三次多项式插值相比，五次多项式时间尺度所产生的最大速度更高、运动更平滑。其时间尺度图如图 5.14 所示。

轨迹生成算法

Trajectory Generation

点到点轨迹生成

梯形加减速曲线



梯形加减速时间尺度对应的 $s(t)$ 、 $\dot{s}(t)$

在加速阶段($0 \leq t \leq \frac{v}{a}$):

$$\ddot{s}(t) = a \quad (5-17)$$

$$\dot{s}(t) = at \quad (5-18)$$

$$s(t) = \frac{1}{2}at^2 \quad (5-19)$$

在恒速阶段($\frac{v}{a} < t \leq T - \frac{v}{a}$):

$$\ddot{s}(t) = 0 \quad (5-20)$$

$$\dot{s}(t) = v \quad (5-21)$$

$$s(t) = vt - \frac{1}{2}v^2 \quad (5-22)$$

在减速阶段($T - \frac{v}{a} < t \leq T$):

$$\ddot{s}(t) = -a \quad (5-23)$$

$$\dot{s}(t) = a(T - t) \quad (5-24)$$

$$s(t) = \frac{2avt - 2v^2 - a^2(t-T)^2}{2a} \quad (5-25)$$

梯形加减速的时间尺度方案虽然没有三项多项式平滑，但我们可以速度极限约束和加速度极限约束，求取最大速度 v 和加减速的斜率 a 。

$$\begin{aligned} |(\theta_{end} - \theta_{start})v| &\leq \dot{\theta}_{limit} \\ |(\theta_{end} - \theta_{start})a| &\leq \ddot{\theta}_{limit} \end{aligned}$$

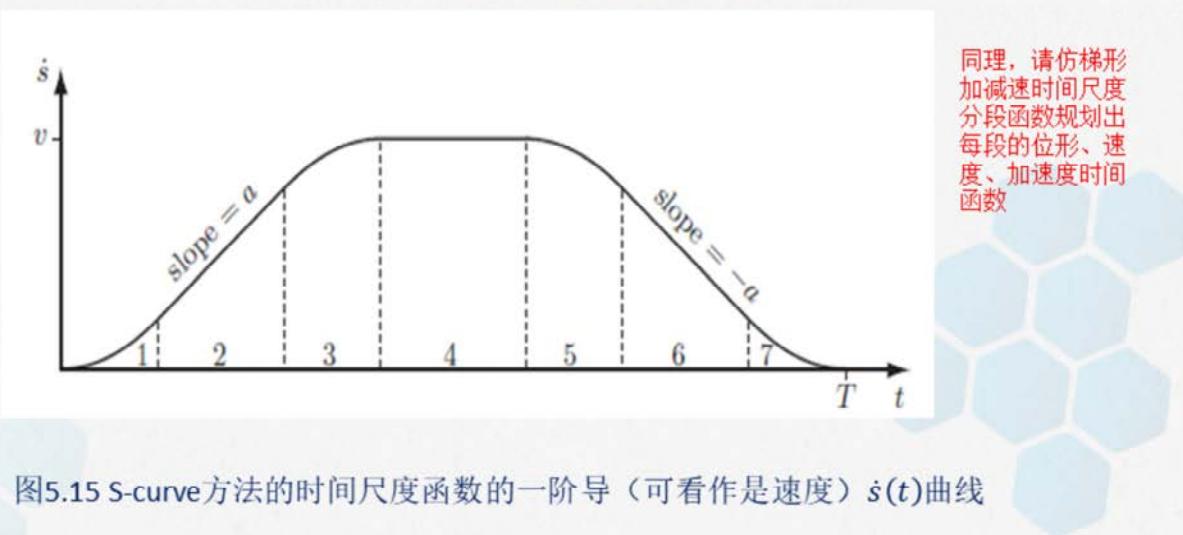


梯形加减速时间尺度函数 $s(t)$ 只能独立指定 v 、 a 、 T 中的两个参数。有三种选择：

- 选择 v 、 a ，为保证三段梯形曲线，需要满足 $v^2/a \leq 1$ （**问题1：如果 $v^2/a > 1$ ，会导
致什么结果？三段会变成两端的加速-减速控制（bang-bang）， $\dot{s}(t)$ 变为三角形**），
根据终止点的位形约束 $s(T) = 1$ ，代入公式 (5-25) 求得 $T = \frac{a+v^2}{va}$ ，如果 v 、 a 对应
最大的可能关节速度和加速度，则上式是梯形加速度方案下的运动最小可能时间。
- 选择 v 和 T ，为保证三段梯形曲线，需要满足 $2 \geq vT > 1$ （**问题2：如果 $vT < 1$ 或
 $vT > 2$ ，会导
致什么结果？三段会变成两端的加速-减速控制（bang-bang）或恒速，
 $\dot{s}(t)$ 变为三角形或一条直线，这也是不允许的**），施加同上一条的终止位形约束
 $s(T) = 1$ ，求得 $a = \frac{v^2}{vT-1}$ ，如果 v 是最高速度，那么在 a 为以最高速度能在 T 时间内
完成运动的加速度值（加减速的斜率）。
- 选择 a 和 T ，施加同上一条的终止位形约束 $s(T) = 1$ ，求得 $v = \frac{1}{2}(aT - \sqrt{a\sqrt{aT^2 - 4}})$ ，
为保证给定时间内能到达目标位形（即 v 有解），需要满足 $aT^2 \geq 4$ 。



S-curve时间尺度由7个阶段组成

图5.15 S-curve方法的时间尺度函数的一阶导（可看作是速度） $\dot{s}(t)$ 曲线

从以上的分析中，我们知道三次多项式时间尺度在起始和结束处会产生无限大的加速度，三段式的梯形加减速运动在 $t \in \{0, t_a, T - t_a, T\}$ 这四个过渡点处也会引起加速度的不连续跳变。在这种情况下，如果直接用时间尺度函数进行关节电机的伺服控制，由于伺服电机加速度突变会导致电机转矩的突变，导致关节力平衡瞬时失步而引起机械臂整体的短时抖动。

针对抖动的问题，我们可以借鉴电机控制的加减速优化算法进行轨迹规划的时间尺度函数优化。比如采用S型加减速算法引入S-curve作为时间尺度函数。S-curve时间尺度由7个阶段组成：

- ① 加加速恒定为 $d^3s/dt^3 = J$ 的阶段，直至达到期望加速度 $\ddot{s} = a$ ；
- ② 加速度恒定为 a 的阶段，直至接近期望速度 $\dot{s} = v$ ；
- ③ 加加速恒定为 $-J$ （负值）的阶段，直至加速度 \ddot{s} 在 $\dot{s} = v$ 时严格等于零；
- ④ 以恒定速度 v 滑行的阶段；
- ⑤ 加加速度恒定为 $-J$ （负值）的阶段；
- ⑥ 加速度恒定为 $-a$ 的阶段；
- ⑦ 加加速恒定为 J （正值）的阶段，直至加速度 \ddot{s} 和 \dot{s} 在位形终点处（ $s(T) = 1$ ）为零。

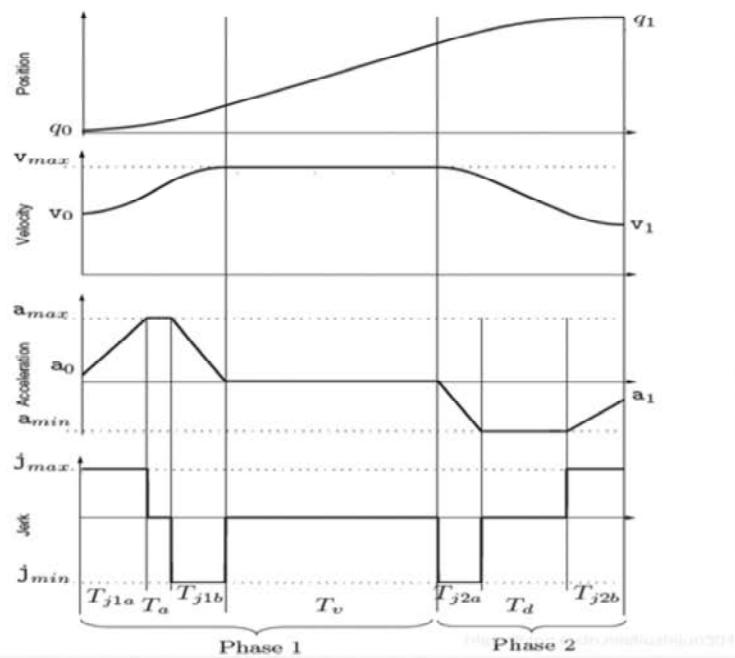
图5.15为S-curve方法的时间尺度函数的一阶导（可看作是速度） $\dot{s}(t)$ 曲线，同理，我们通过加速度约束、速度约束以及起始点处的位形约束计算出各个阶段时间尺度函数的二阶导、一阶导以及时间尺度完整方程（分别对应加速度、速度、位形相对于时间的分段规划函数）。

轨迹生成算法

Trajectory Generation

点到点轨迹生成

S形曲线



具有中间路点的轨迹生成

如果作业要求机器人在指定时间内通过一系列中间点(via point)，而并没有严格规定前后点之间的路径形状，那么需要约束的只是路径点上的关节速度问题。这个速度确定有三种方法：

- (1) 根据工具坐标系在直角坐标空间中的瞬时线速度和角速度来确定每个路径点的关节速度，但该方法工作量较大。
- (2) 为了保证每个路径点上的加速度连续，由控制系统按照此要求自动地选择路径点的速度；
- (3) 在直角坐标空间或关节空间中采用某种适当的启发式方法，由控制系统自动地选择路径点的速度。

具有中间路点的轨迹生成

采用时间尺度函数进行轨迹规划，简单的解决方法有两种：

- 直接采用前面的点到点的轨迹规划方法，只是将第一段的终止速度约束指定为非零的合理速度，后面段的起始速度等于前一段结束的终止速度即可。这个时间尺度函数的系数求取方法和前面介绍的分段式加减速方法相同。
- 采用一些高阶的曲线函数，逼近需要路径的控制点。例如采用贝塞尔曲线、B样条曲线等作为时间尺度函数。

第一种方法只是点到点规划的一种扩展，如果能较好地规划好路径点的速度，可以很容易推导实现。

第二种方法只是轨迹逼近路点，例如三阶贝塞尔曲线方程如公式

轨迹生成算法

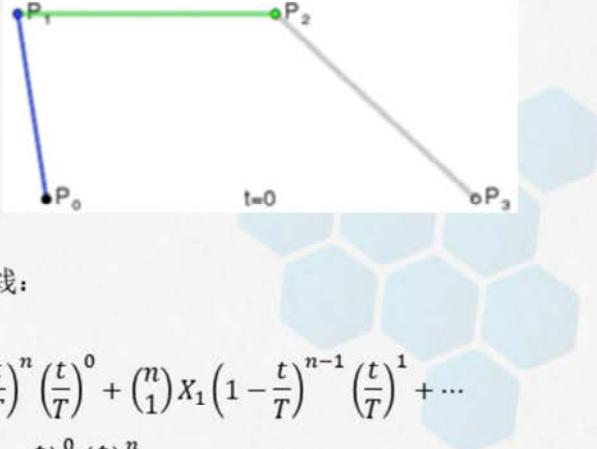
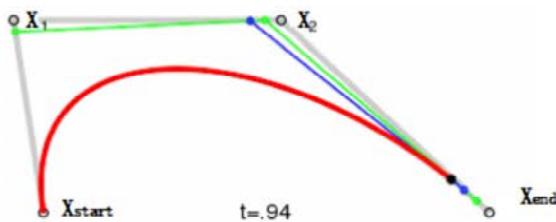
Trajectory Generation

具有中间路点的轨迹生成

不过路点 

三阶贝塞尔曲线

$$X(t) = X_{start} \left(1 - \frac{t}{T}\right)^3 + 3X_1 \frac{t}{T} \left(1 - \frac{t}{T}\right)^2 + 3X_2 \left(\frac{t}{T}\right)^2 \left(1 - \frac{t}{T}\right) + X_{end} \left(\frac{t}{T}\right)^3, t \in [0, T]$$



给定 $n + 1$ 个控制点，贝塞尔曲线一般形式为 n 次曲线：

$$\begin{aligned} X(t) &= \sum_{i=0}^n \binom{n}{i} X_i \left(1 - \frac{t}{T}\right)^{n-i} \left(\frac{t}{T}\right)^i = \binom{n}{0} X_0 \left(1 - \frac{t}{T}\right)^n \left(\frac{t}{T}\right)^0 + \binom{n}{1} X_1 \left(1 - \frac{t}{T}\right)^{n-1} \left(\frac{t}{T}\right)^1 + \dots \\ &\quad + \binom{n}{n-1} X_{n-1} \left(1 - \frac{t}{T}\right)^1 \left(\frac{t}{T}\right)^{n-1} + \binom{n}{n} X_n \left(1 - \frac{t}{T}\right)^0 \left(\frac{t}{T}\right)^n, t \in [0, T] \end{aligned}$$



B样条曲线为贝塞尔曲线的一种一般化。假设空间中有 $m+1$ 个控制点，则 n 次贝塞尔曲线可以写为：

$$X(t) = \sum_{i=0}^m X_i b_{i,n}(t/T), t \in [0, T]$$

上式中， m 为区间数， X_i 为位形控制点， $b_{i,n}(t/T)$ 为 n 次B样条基函数， $2 \leq n \leq m$ ，即当曲线有 $m+1$ 个控制点时，B样条基函数的次数可为2到 m 次，将多个贝塞尔曲线连接就可以得到B样条。



轨迹生成算法

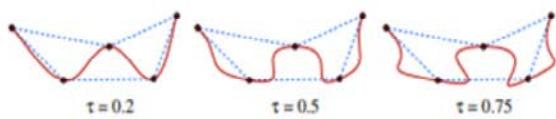
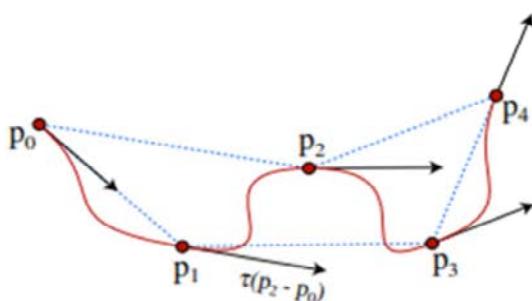
Trajectory Generation

具有中间路点的轨迹生成

严格过路点



Catmull-Rom曲线实际为三段三次多项式曲线组成



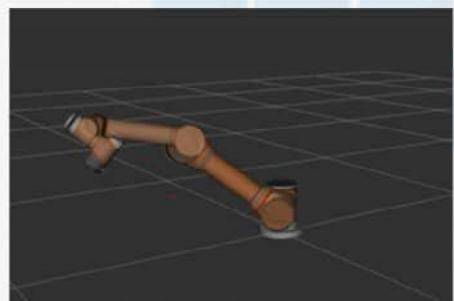
τ 的取值影响

$$\{2X_0 - X_1, X_0, X_1, X_2\}, \{X_0, X_1, X_2, X_3\}, \{X_1, X_2, X_3, 2X_3 - X_2\}$$

$$X_j(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3 \quad j = 1, 2, 3$$

其中j为三段三次曲线的编号:

$$\begin{aligned}c_0 &= X_{i-1} \\c_1 &= (-\tau)X_{i-2} + (\tau)X_i \\c_2 &= (2\tau)X_{i-2} + (\tau - 3)X_{i-1} + (3 - 2\tau)X_i + (-\tau)X_{i+1} \\c_3 &= (-\tau)X_{i-2} + (2 - \tau)X_{i-1} + (\tau - 2)X_i + (\tau)X_{i+1}\end{aligned}$$

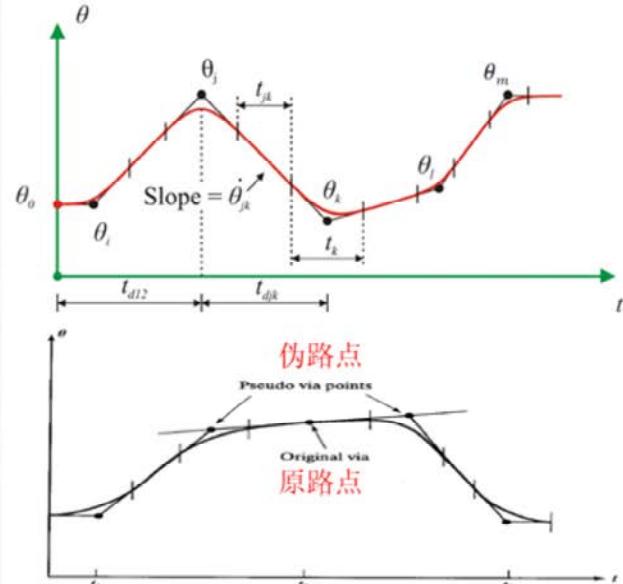
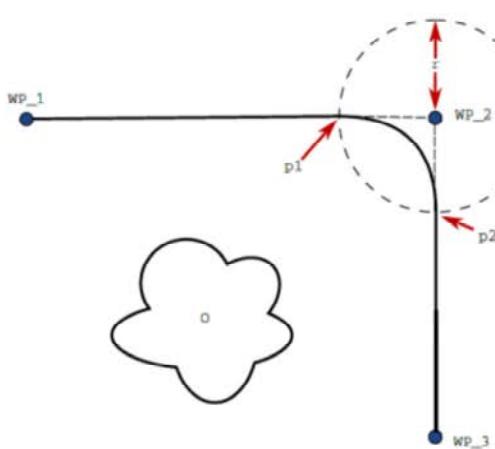


轨迹生成算法

Trajectory Generation

具有中间路点的轨迹生成

交融半径 r



另外，在实际的应用中，如果不要求机器人严格经过路点，也可简单规划机器人在路径点前后连续走两条直线轨迹，只是在两条直线过渡处设置一个转弯半径 r （交融半径），在过渡段规划一个半径为 r 的圆弧，或者采用抛物线过渡函数或者三次（五次）多项式函数进行插值。如果我们希望轨迹严格通过某个路点，也可以设置一些“伪路点”，而在抛物线插值方法的直线段严格经过原始路点。

在轨迹生成算法中，需要根据作业要求采取不同的轨迹规划算法，满足不同程度的作业轨迹、速度控制等要求。当然我们也可以采用时间尺度函数进行时间最优的求解，但需要注意加速度的连续性。

04

位形空间离散化规划方法

目前还没有任何一种单一的规划算法能适用于所有的运动规划问题。针对中的 C_{free} 和 C_{obs} ，其几何或拓扑的精确数学表达比较复杂，特别是高维空间或位形自由度较多的情况下，其数学描述难以推导。

C-Space离散化规划方法（C-Space Discretizations）有两种：

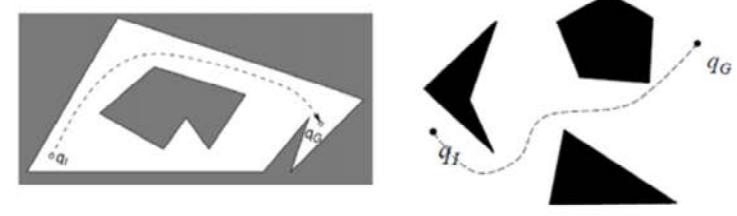
- **组合规划方法（Combinatorial planning）：** 判别 C_{free} 的连通性，将 C_{Free} 明确表征到图（graph）中，并使用搜索算法寻找路径；
- **基于采样的规划方法（Sampling-based planning）：** 使用碰撞检测来探查并逐步搜索C-Space得到规划路径。

位形空间离散化规划方法

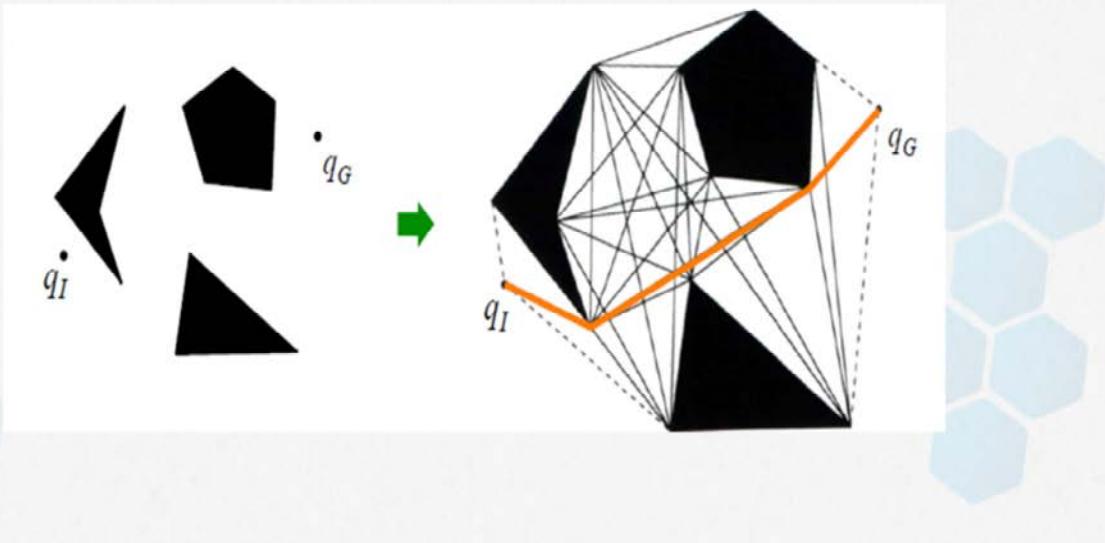
C-Space Discretizations

组合规划方法 (Combinatorial planning)

- 组合规划方法有四种常用的地图表示方法：可视图(Visibility graphs)、Voronoi图(Voronoi diagrams)、精确单元分解ECD(Exact cell decomposition)、近似单元分解ACD(Approximate cell decomposition);
- 它们都生成路线图(road map)，路线图处于 C_{free} 空间；
- 除了代表自由空间的空白处外， C_{free} 中每个 C_{obs} 图形顶点也表示为 C_{free} 中的一个位形，每条边也为可通过 C_{free} 的一条无碰路径。

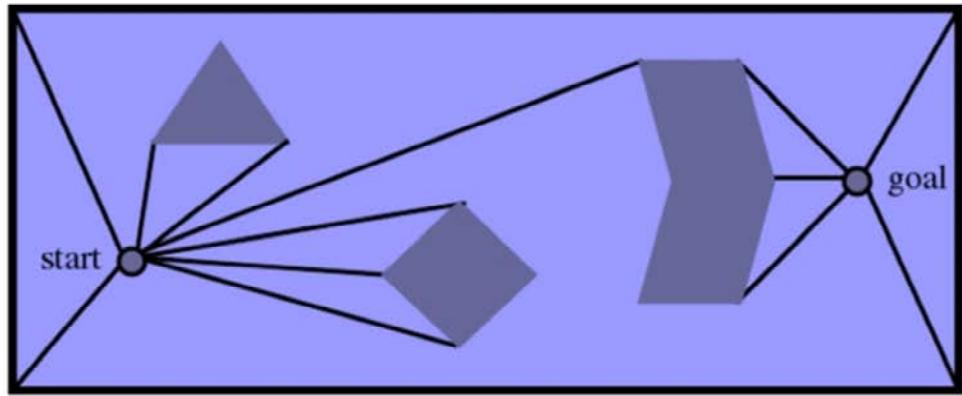


在这种情况下，我们将 C_{obs} 按前面介绍过的规则进行处理之后，机器人表示为一个不能旋转的点，它处于一个多边形的世界中。如图5.20所示，机器人为多变形障碍物世界中的一个点，组合路径规划方法需要找出一条在从起点 q_I 到目标点 q_G 的一条无碰路径。



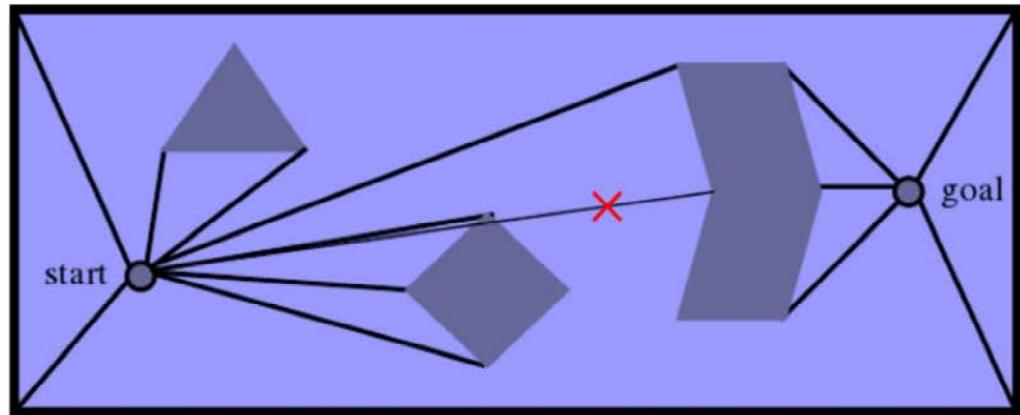
可视图法的思想是将可行的路径构造成通过 C_{obs} 顶点，从起点 q_I 到目标点 q_G 的一系列折线，然后再在所有可能的折线中，采用图搜索算法如Dijkstra最短路径算法寻找一条最优路径。如图5.21所示，这是一种最早的地图构建方法之一，该算法的时间复杂度为 $O(n^2 \log n)$ 。

在图5.21中，可视图为一种无向图，图中的可行路径节点由起始节点以及所有 C_{obs} 顶点组成，每两个节点连接的边不能穿过 C_{obs} ，即直线是“可视”的，其边长权重为欧几里得距离。

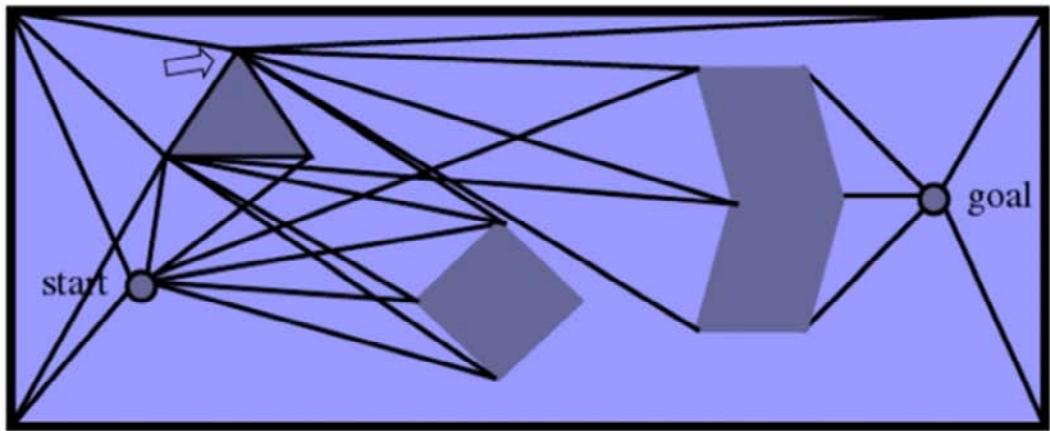


First, draw lines of sight from the start and goal to all “visible” vertices and corners of the world.

首先从起点和目标点向所有障碍物和环境的可视顶点或拐角画线

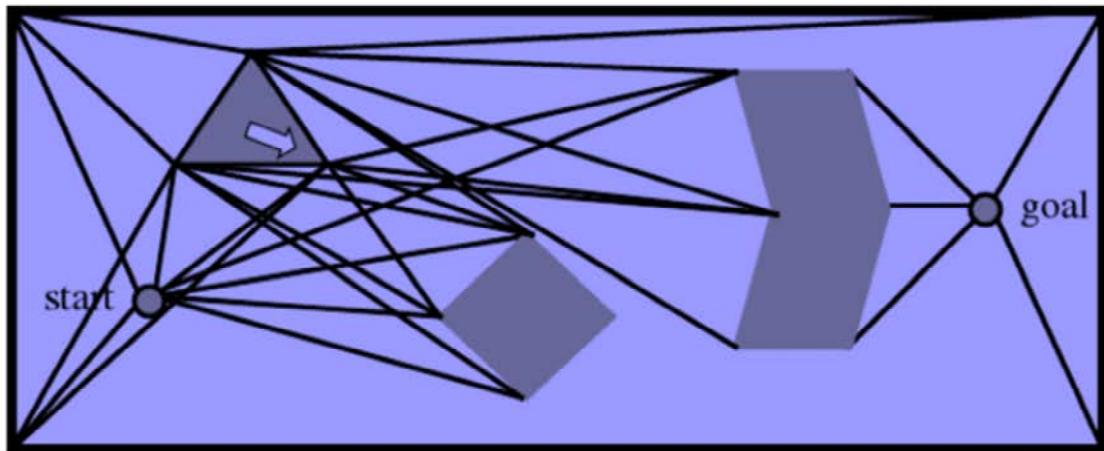


This is an error link, the obstacle middle vertex is ‘Invisible’ to start point



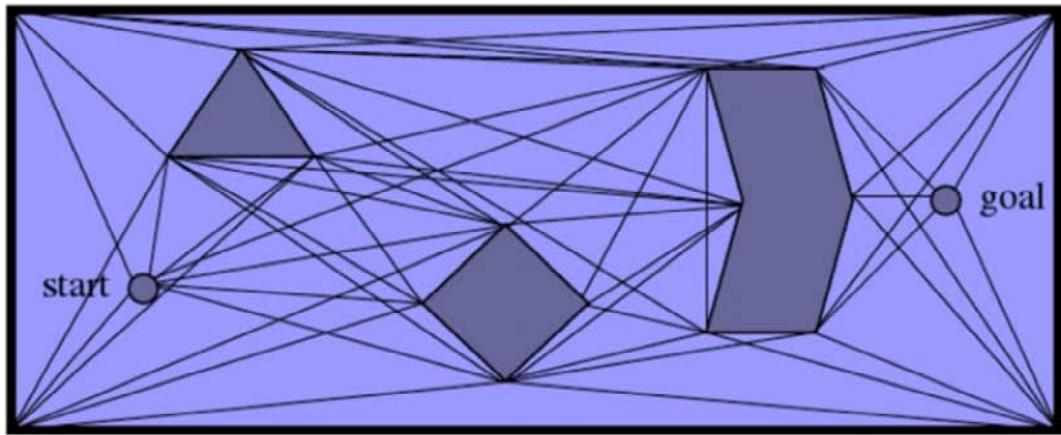
继续第二步，从每个障碍物的顶点向其余所有可见障碍物的顶点和环境的拐角画线，注意边线是可见的。

Continue, Second step , draw lines of sight from every vertex of every obstacle like before. Remember lines along edges are also lines of sight.



继续第二步，从每个障碍物的顶点向其余所有可见障碍物的顶点和环境的拐角画线，注意边线是可见的。

Continue, Second step , draw lines of sight from every vertex of every obstacle like before. Remember lines along edges are also lines of sight.



最后直至所有的顶点都与其余可视顶点连接，即可得到完整的可视图

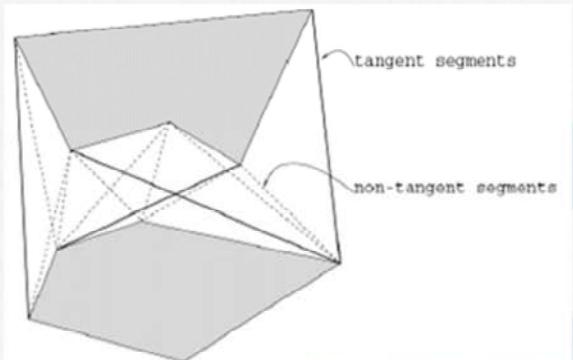
Repeat until you're done.

Since the map was in C-space, each line potentially represents part of a path from the start to the goal.

可视图法视移动机器人为一点，将机器人、目标点和多边形障碍物的各顶点进行组合连接，并保证这些直线均不与障碍物相交，这就形成了一张图，称为可视图。由于任意两直线的顶点都是可见的，从起点沿着这些直线到达目标点的所有路径均是运动物体的无碰路径。搜索最优路径的问题就转化为从起点到目标点经过这些可视直线的最短距离问题。运用优化算法，可删除一些不必要的连线以简化可视图，缩短搜索时间。该法能够求得最短路径，但假设忽略移动机器人的尺寸大小，使得机器人通过障碍物顶点时离障碍物太近甚至接触，并且搜索时间长。



- *Idea:* we don't really need all the edges in the visibility graph (even if we want shortest paths)
- *Definition:* Let L be the line passing through an edge (x,y) in the visibility graph G . The segment (x,y) is a **tangent segment** if L is tangent to CB at both x and y .
- Line segment is tangent if extending the line beyond each of the end points would not intersect the obstacles.

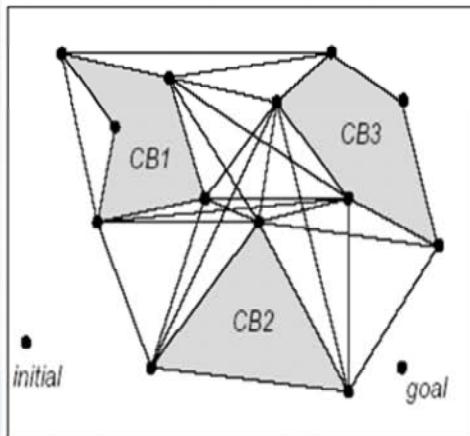


Visibility graph drawbacks

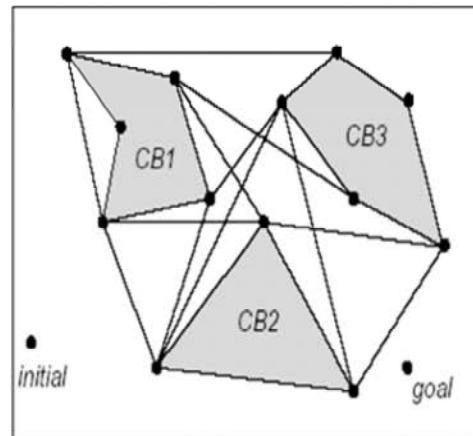
Visibility graphs do not preserve their optimality in higher dimensions.
In addition, the paths they find are “semi-free,” i.e. in contact with obstacles.
No clearance.



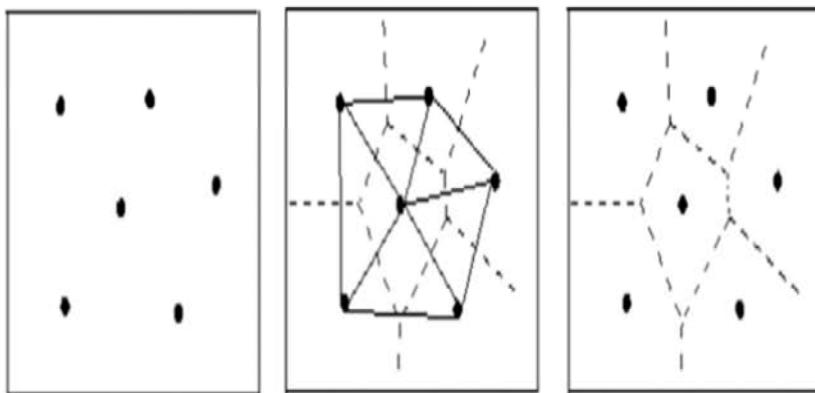
Visibility Graph



Reduced Visibility Graph

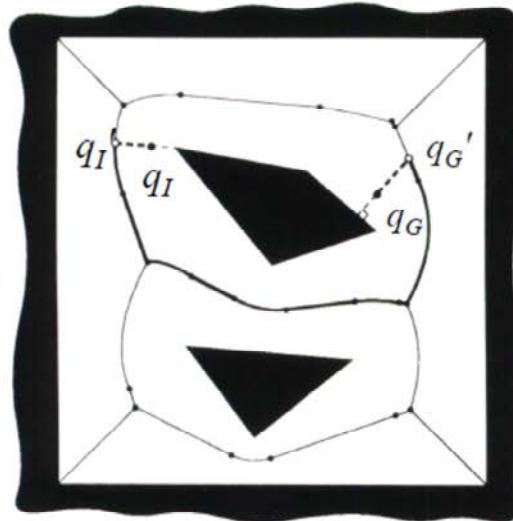


- It turns out we need only keep convex vertices of C-obstacle
- non-CB edges that are tangent segments



Voronoi图法是由俄国数学家Georgy Fedoseevich Voronoi建立的空间分割算法，狭义Voronoi图以一些离散点为基，生成一系列图元，每个图元到其对应基的距离比到其他基的距离都要短，而图元边界上的点到两个或者多个基的距离相等。在路径规划中，考虑到所规划路径的安全性，需要建立广义的Voronoi图，即Voronoi图上的节点到两个或多个障碍物的距离是相等的，也就是Voronoi图的节点是距离障碍物最远的点。

广义Voronoi图是对可视图法的一种改进，可视图法仅适用于多边形障碍物，对圆形障碍物失效。Voronoi图将生成元两两相连接，并且对连接线段作垂直平分线，若干条垂直平分线之间的交线形成一些多边形，这样就把一个整区域划分成了一些分区域，每一个分区域中都只包含一个生成元。如图5.22所示。



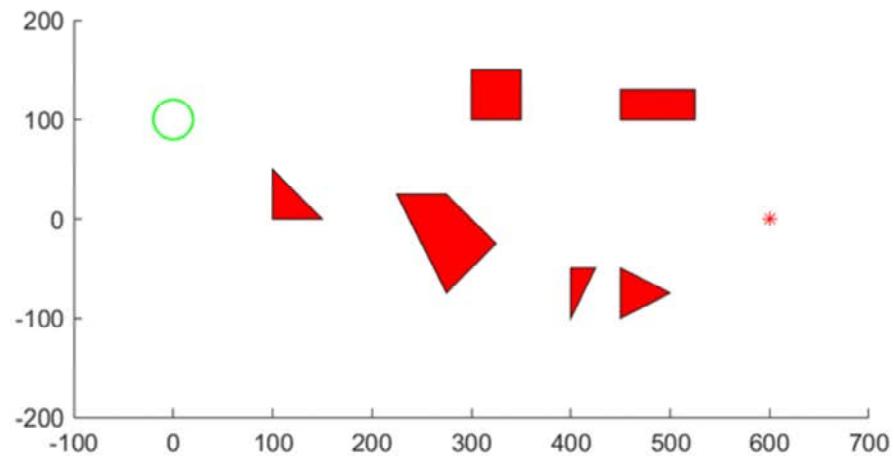
得到Voronoi图之后，确定起点 q_I 、目标点 q_G 到Voronoi子图的最近点，并将其连接，然后通过图搜索算法如Dijkstra算法寻找最优路径路径。

位形空间离散化规划方法

C-Space Discretizations

组合规划方法

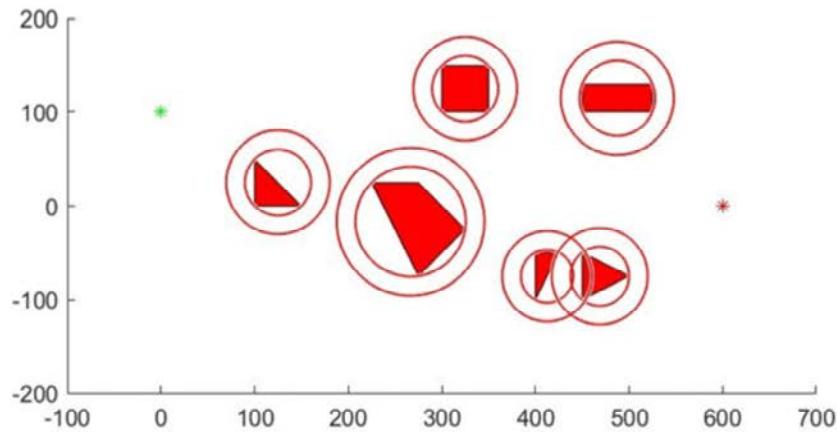
Voronoi图法



实例：
中国工

基于Voronoi图法的移动机器人路径规划[J].
参考，但这个论文有明显错误！）

初始化地图数据，其中红色色块为障碍物，绿色圆圈表示圆形运动体，它在起点的位置上，红色*表示目标点。



C_{obs} 膨胀方法明显错误！

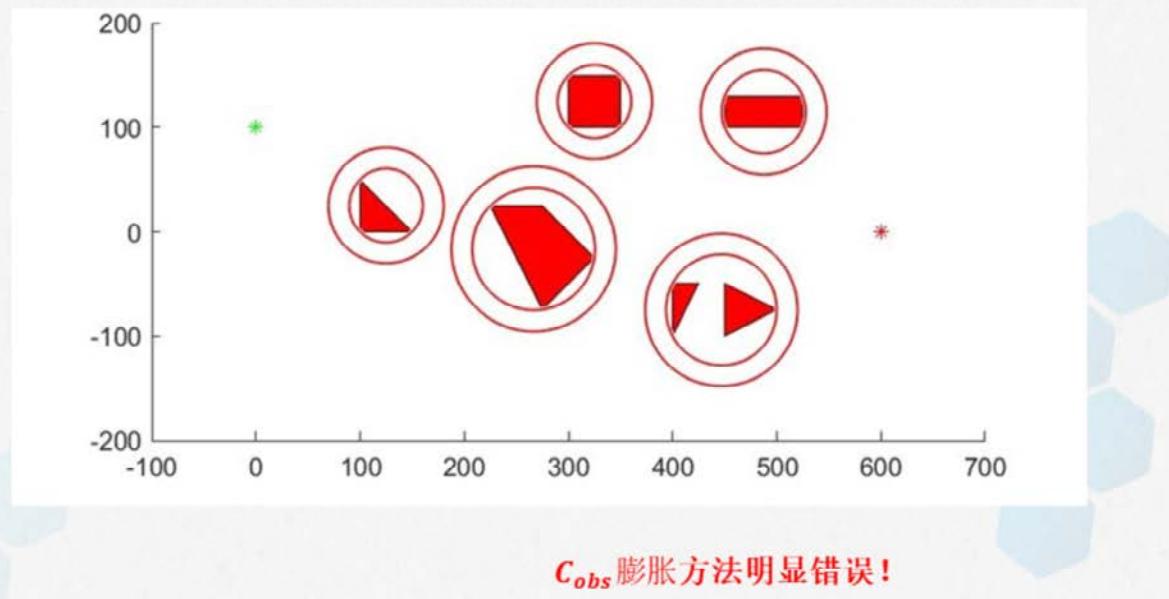
之后，得到障碍物的外接圆，并“增长”外接圆，此时与运动体可作为单点处理。

位形空间离散化规划方法

C-Space Discretizations

组合规划方法

Voronoi图法

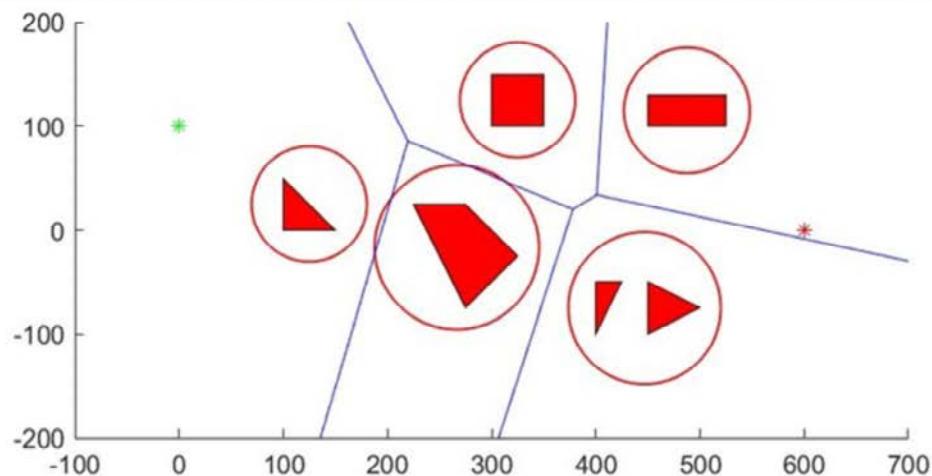


位形空间离散化规划方法

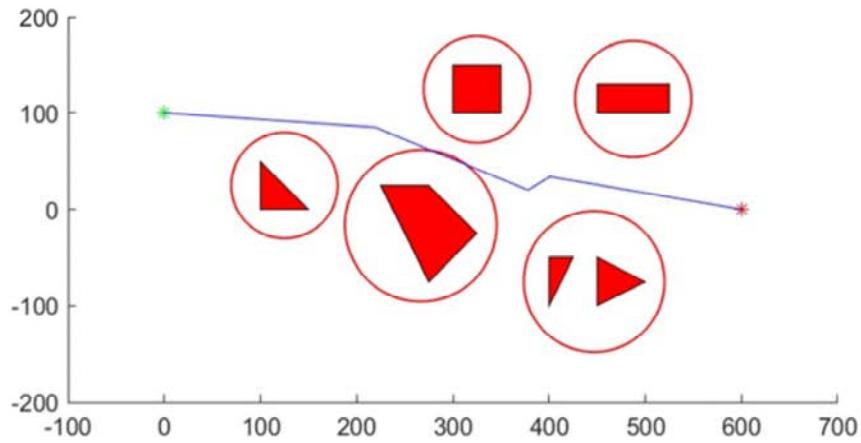
C-Space Discretizations

组合规划方法

Voronoi图法



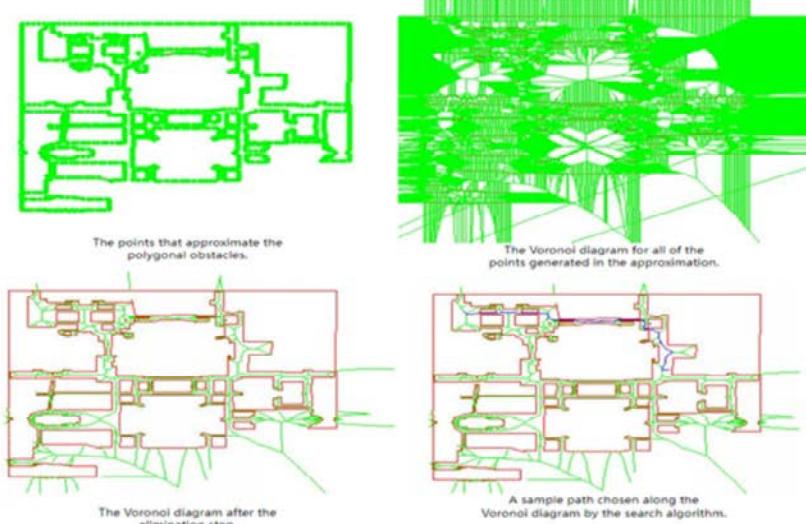
Voronoi图生成明显错误！



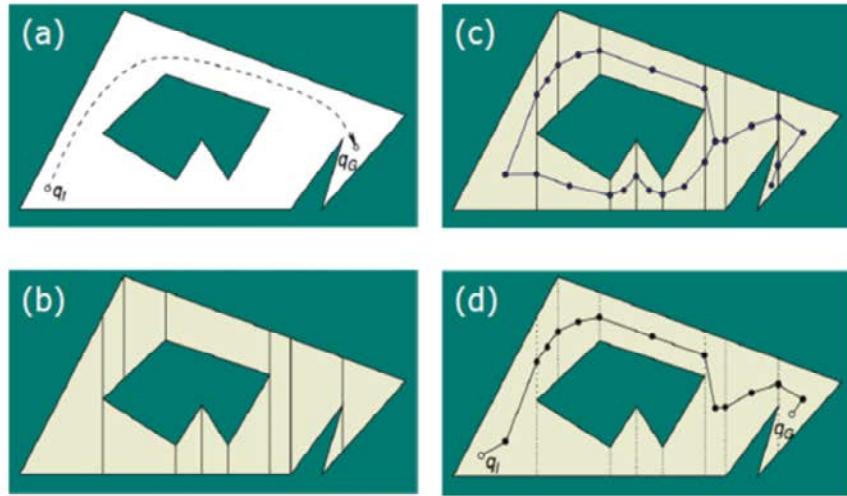
其实是应该得到一个路线图，然后再用图搜索算法寻找最优路径！



Robot Path Planning Using Generalized Voronoi Diagrams



https://www.cs.columbia.edu/~pblaer/projects/path_planner/



精确单元分解法ECD将 C_{free} 分解为互不重叠的单元栅格，构造连通图（connectivity graph）来表示邻接关系，然后通过图搜索算法如Dijkstra算法寻找最优路径路径。其中一种典型的算法实现步骤如下：

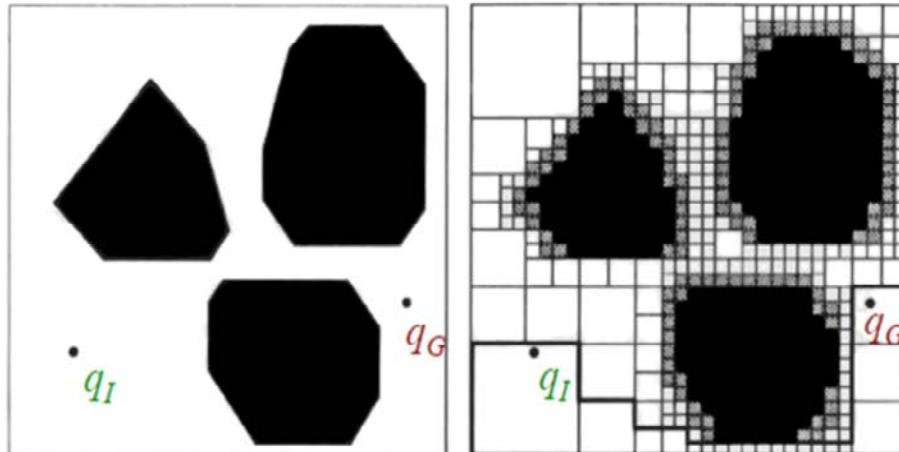
通过从每个 C_{obs} 多边形顶点向上和向下发射垂直光线（垂直线不能透过障碍物 C_{obs} ，将 C_{free} 分解为具有垂直边段的梯形；

在每个梯形的内部放置一个顶点，例如中心点；

在每个垂直线段中放置一个顶点；

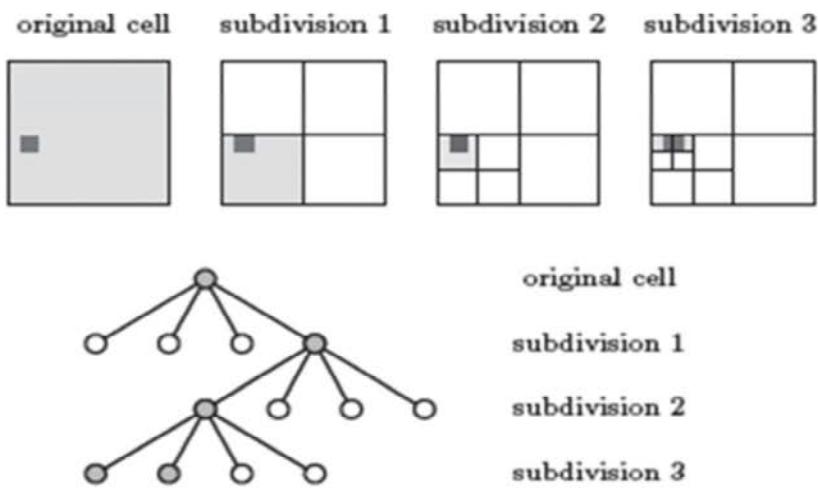
连接顶点；

寻找从起点 q_I 到目标点 q_G 的一条最优路径。



精确单元分解法ECD对于 C_{obs} 具有复杂几何形状的环境来说会导致算法效率的低下。而近似单元分解法ACD使用具有相同预定义的简单形状单元来对C-Space进行分解，如图5.24所示的栅格图（四叉树分解：Quadtree decomposition）。

这种分解如果采用4-连通栅格，可以根据罗盘的基点确定东南西北四个方向的运动，如果允许对角线运动，可定义8-连通栅格，如果东南西北方向的运动成本为1，则对角运动的成本可以是 $\sqrt{2}$ 。如果机器人只能沿轴线直行，则图搜索算法的距离应基于曼哈顿距离公式进行计算，而不是欧氏距离。

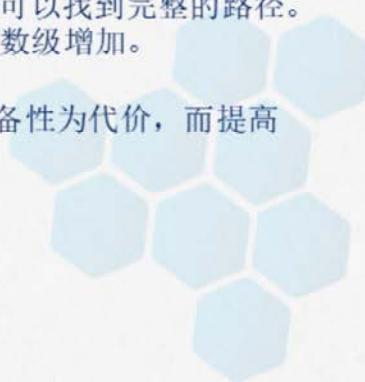


以网格点为中心的直线单元的任何部分接触到 C_{obs} ，该网格点就必须被标记为障碍物。近似单元分解法(ACD)栅格的数量 m 会随着分辨率 k 增加线性增长，而随机器人维度 n 的增加而呈指数级增长， $m = k^n$ 。分辨率高控制越精确，计算代价越大，分辨率降低，将会使C-Space空间的表达变粗略，可能会错过自由路径。一种替代方案是可以采用多分辨率网格进行分解，如图5.24所示，对 C_{obs} 进行细分为更小的单元，而远离 C_{obs} 的地方采用较粗分辨率的网格表示。原始网格单元的每个维度被划分为两半，从而又产生 n 维空间中的 2^n 个子单元。图5.24每个障碍单元被细分为四个单元。如果 $n = 3$ ，每个障碍单元被细分为 $2^n = 8$ 个子单元，该表示被称为八叉树(octree)分解。图5.25演示了原始单元中接触到障碍的网格四叉树细分过程。



小结

- 以上介绍的组合规划方法优点有：可以采用重复相同的简单计算（迭代搜索算法），数值上更稳定，更容易实施，如果路径存在就可以找到完整的路径。但如果C-Space维度增加，组合规划方法的复杂度会呈指数级增加。
- 针对这个问题，可使用基于采样的规划方法，以损失完备性为代价，而提高路径规划算法的执行效率。





组合规划法中可用到的图搜索算法

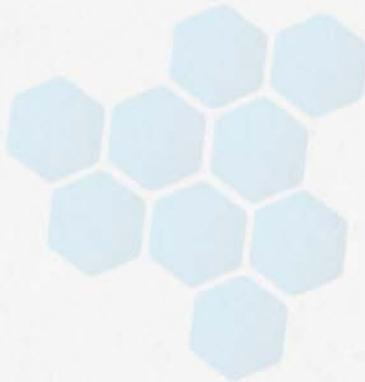
将自由空间表示为图之后，可以通过在图中搜索从起点到目标点的一条最优路径来进行运动规划。这些算法被称为启发式搜索算法，包括：

- Dijkstra算法；
- 广度优先搜索；
- A*搜索；
- D*搜索；
- ...





Edsger_Wybe_Dijkstra
(1930~2002)



Dijkstra算法是由荷兰计算机科学家Dijkstra于1959年提出的解决两点之间最短路径的一种算法。其算法思想为：

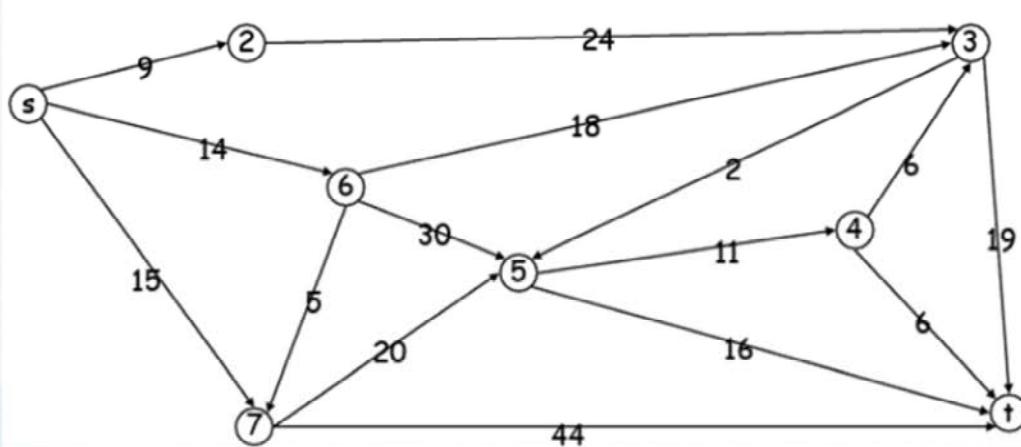
设 $G = (V, E)$ 为加权有向图，我们把图中顶点集合 V 分成两组，第一组为已求出最短路径的顶点集合，用 S 表示，初始时 S 中只有一个源点，以后每求得一条最短路径，就将其加入到集合 S 中，直到全部顶点都加入到 S 中，算法就结束了）。第二组为其余未确定最短路径的顶点集合（用 Q 表示），按最短路径长度的递增次序依次把第二组的顶点加入 S 中。在加入的过程中，总保持从源点 s 到 S 中各顶点的最短路径长度不大于从源点 s 到 Q 中任何顶点的最短路径长度。此外，每个顶点对应一个距离， S 中的顶点的距离就是从 s 到此顶点的最短路径长度， Q 中的顶点的距离，是从 s 到此顶点只包括 S 中的顶点为中间顶点的当前最短路径长度。



已知 $G = (V, E)$ 为加权有向图， S 表示最短路径的顶点集合， Q 表示其余未确定最短路径的顶点集合。

算法步骤：

- 1. 初始时， S 只包含源点，即 $S = \{v\}$ ， v 的距离为 0。 Q 包含除 v 外的其他顶点，即： $Q = \{\text{其余顶点 } u\}$ ，若 v 与 Q 中顶点 u 有边，则 $\langle u, v \rangle$ 正常有权值，若 u 不是 v 的有边连接点，则 $\langle u, v \rangle$ 权值为 ∞ ；
- 2. 从 Q 中选取一个与 v 的距离最小的顶点 k ，将 k 加入 S 中（该选定的距离就是 v 到 k 的最短路径长度）；
- 3. 以 k 为新考虑的中间点，修改 Q 中各顶点的距离；若从源点 v 到顶点 u 的距离（经过顶点 k ）比原来距离（不经过顶点 k ）短，则修改顶点 u 的距离值，修改后的距离值为顶点 k 的距离加上边的权值；
- 4. 重复步骤 2 和 3 直到所有顶点都包含在 S 中。



寻找一条从s到t的最短路径

For a given source node in the graph, the algorithm finds the shortest path between that node and every other.

Let the node at which we are starting be called the **initial node**.

known con:

- 1) A set of nodes in a directed graph: Q
- 2) Connect two points of weight: w
- 3) The connecting direction of the two points
- 4) Starting node number is s

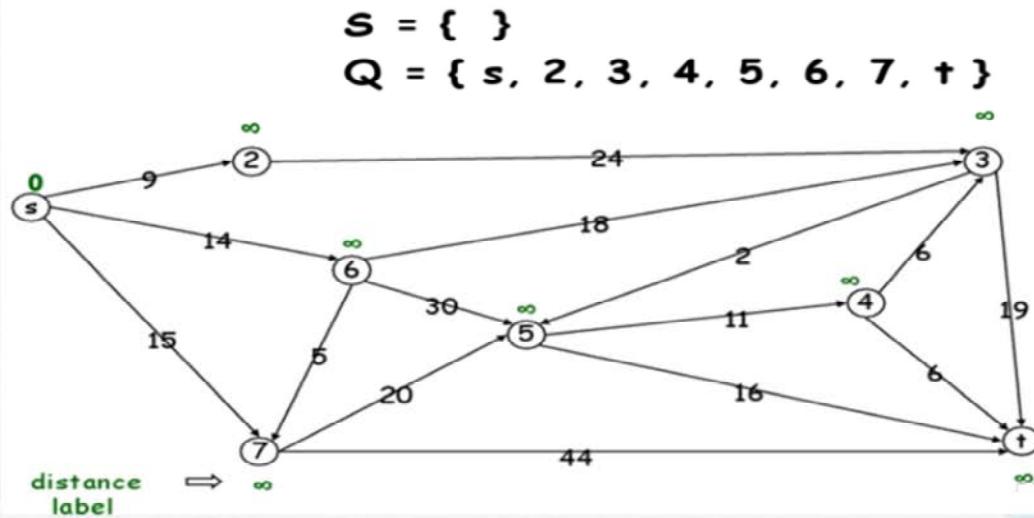
To find the shortest path between that start node and goal node

位形空间离散化规划方法

C-Space Discretizations

图搜索算法

Dijkstra算法

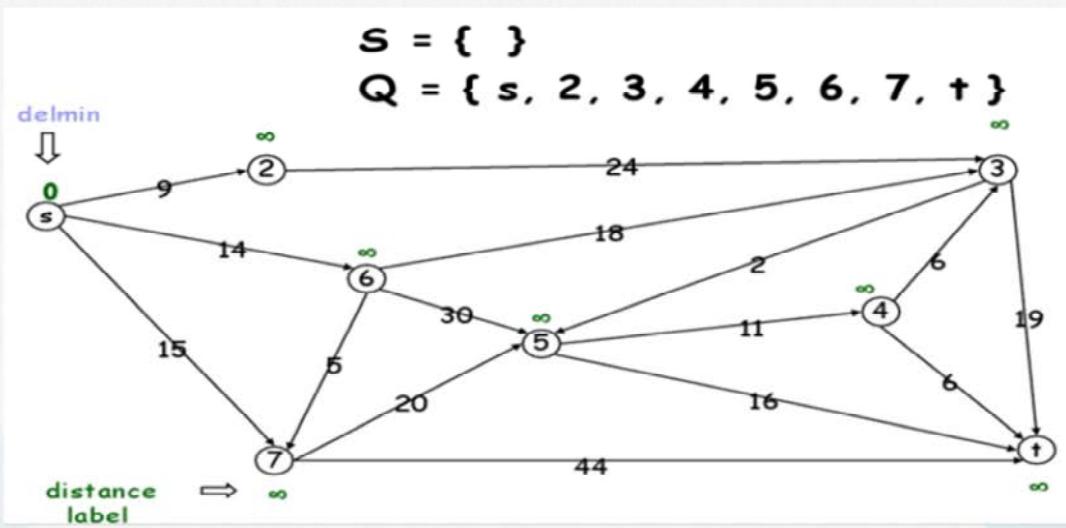


位形空间离散化规划方法

C-Space Discretizations

图搜索算法

Dijkstra算法

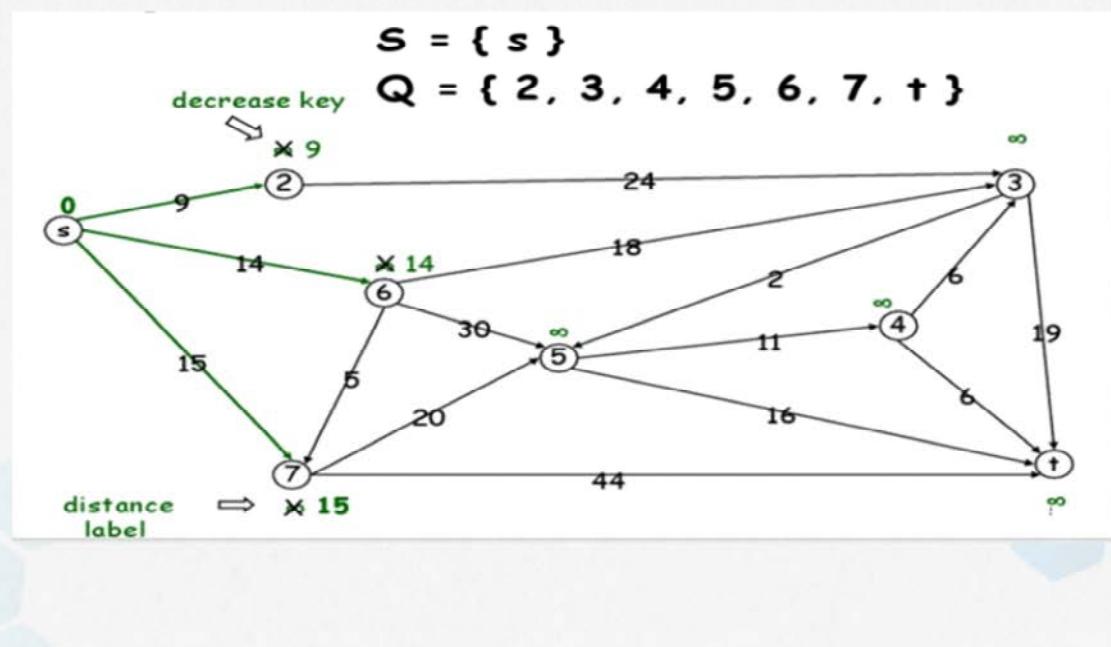


位形空间离散化规划方法

C-Space Discretizations

图搜索算法

Dijkstra算法

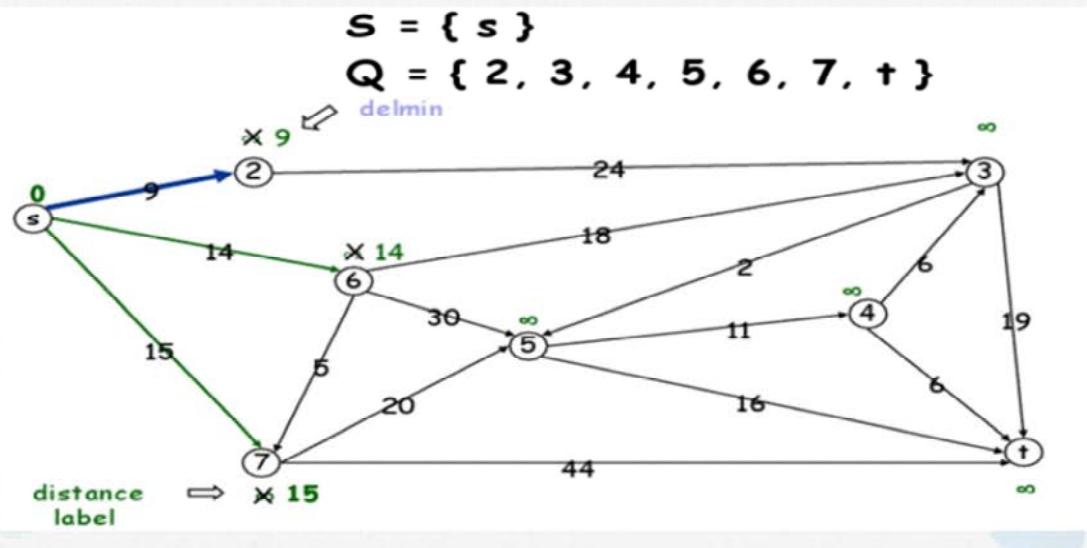


位形空间离散化规划方法

C-Space Discretizations

图搜索算法

Dijkstra算法



位形空间离散化规划方法

C-Space Discretizations

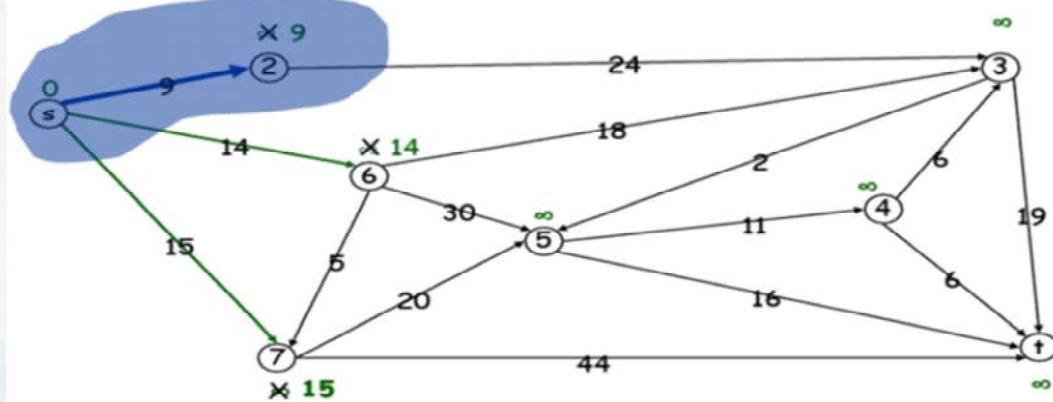
图搜索算法

Dijkstra算法



$$S = \{ s, 2 \}$$

$$Q = \{ 3, 4, 5, 6, 7, + \}$$



位形空间离散化规划方法

C-Space Discretizations

图搜索算法

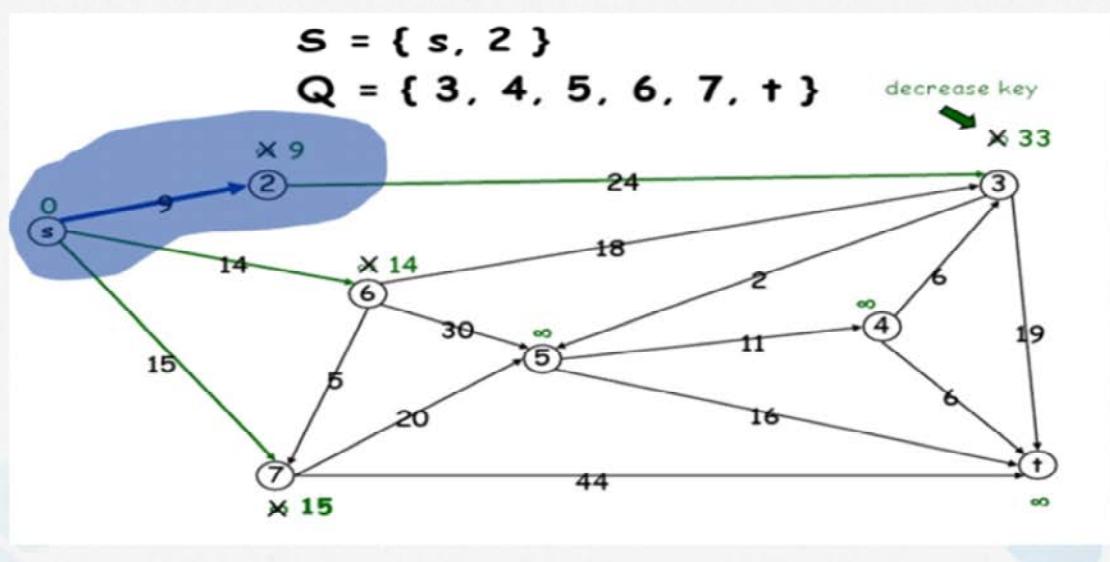
Dijkstra算法



$$S = \{ s, 2 \}$$

$$Q = \{ 3, 4, 5, 6, 7, t \}$$

decrease key

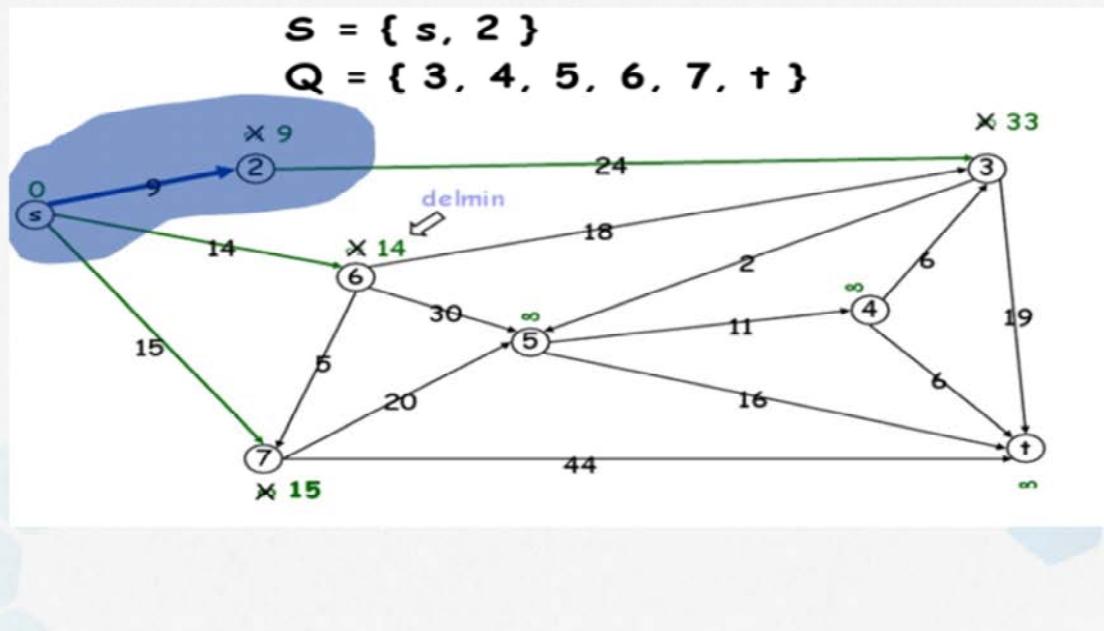


位形空间离散化规划方法

C-Space Discretizations

图搜索算法

Dijkstra算法



位形空间离散化规划方法

C-Space Discretizations

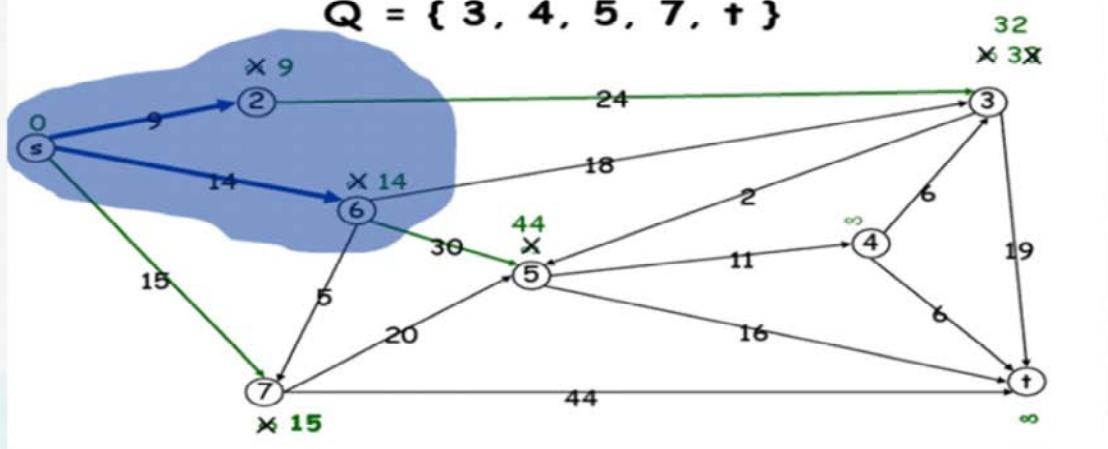
图搜索算法

Dijkstra算法



$$S = \{ s, 2, 6 \}$$

$$Q = \{ 3, 4, 5, 7, t \}$$

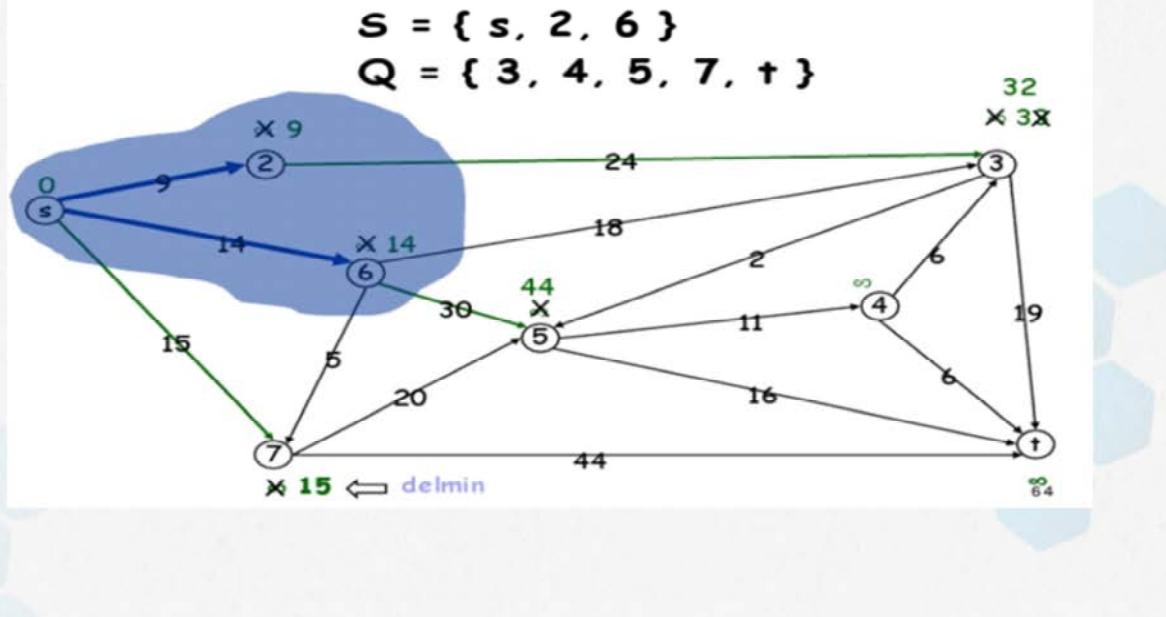


位形空间离散化规划方法

C-Space Discretizations

图搜索算法

Dijkstra算法

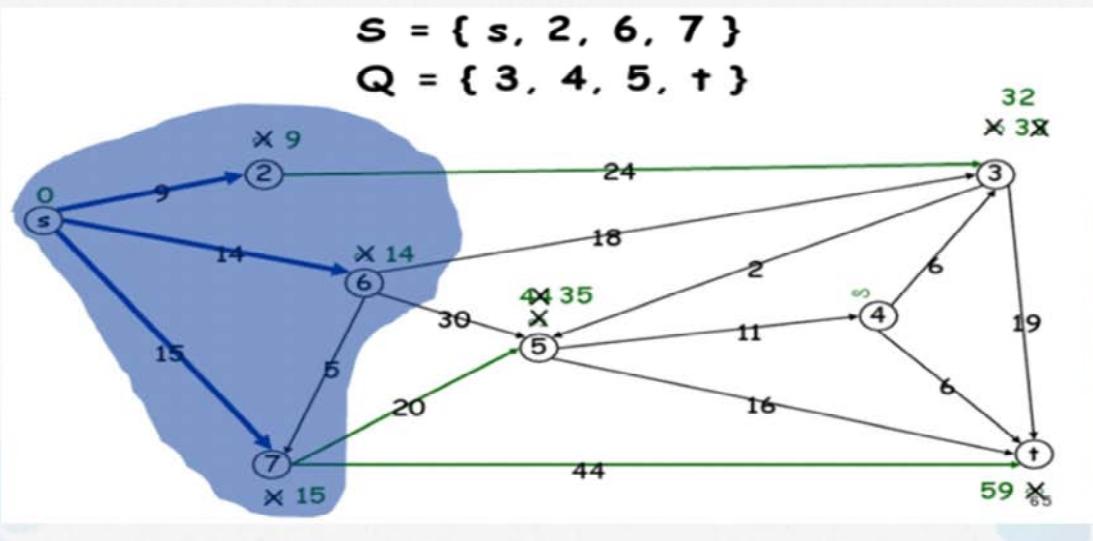


位形空间离散化规划方法

C-Space Discretizations

图搜索算法

Dijkstra算法



位形空间离散化规划方法

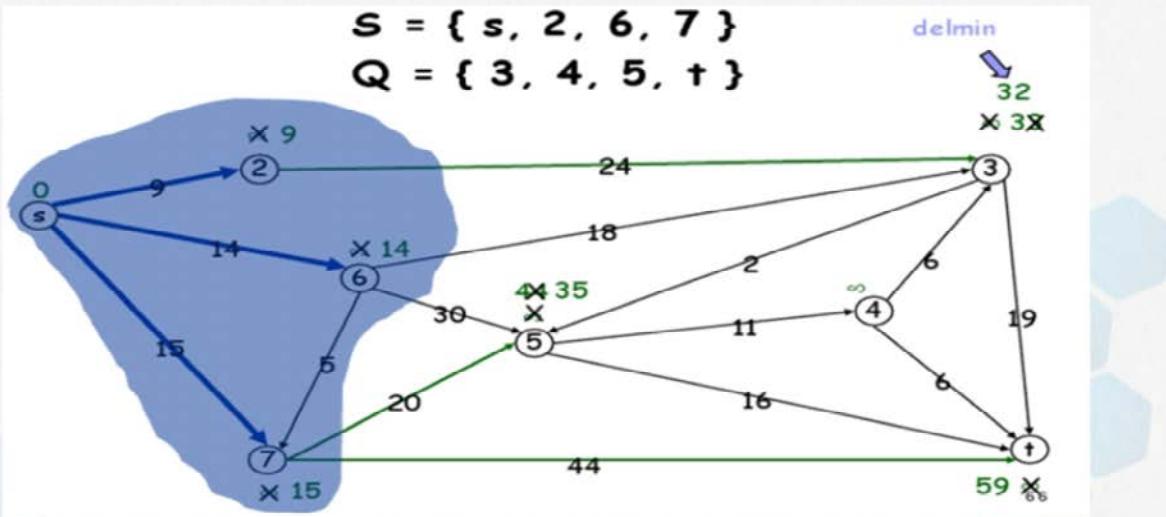
C-Space Discretizations

图搜索算法

Dijkstra算法



$$S = \{ s, 2, 6, 7 \}$$
$$Q = \{ 3, 4, 5, t \}$$



位形空间离散化规划方法

C-Space Discretizations

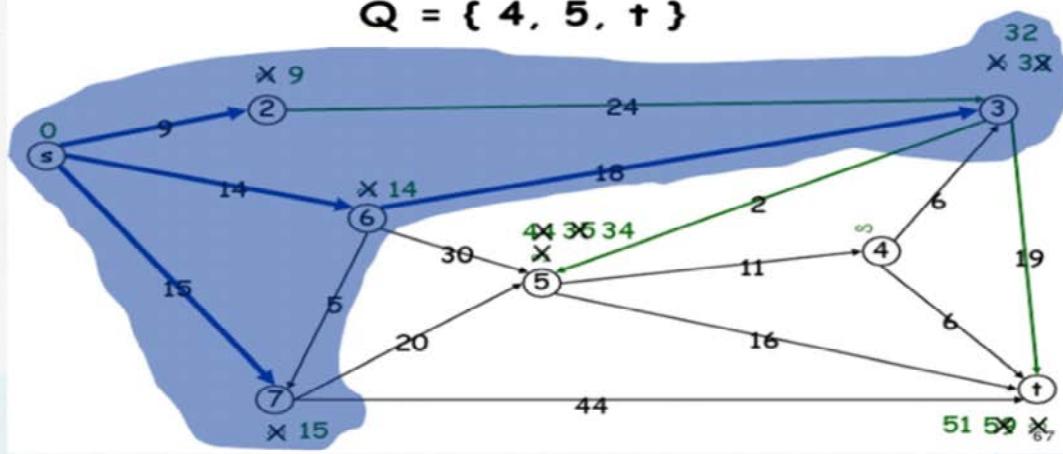
图搜索算法

Dijkstra算法



$$S = \{ s, 2, 3, 6, 7 \}$$

$$Q = \{ 4, 5, t \}$$



位形空间离散化规划方法

C-Space Discretizations

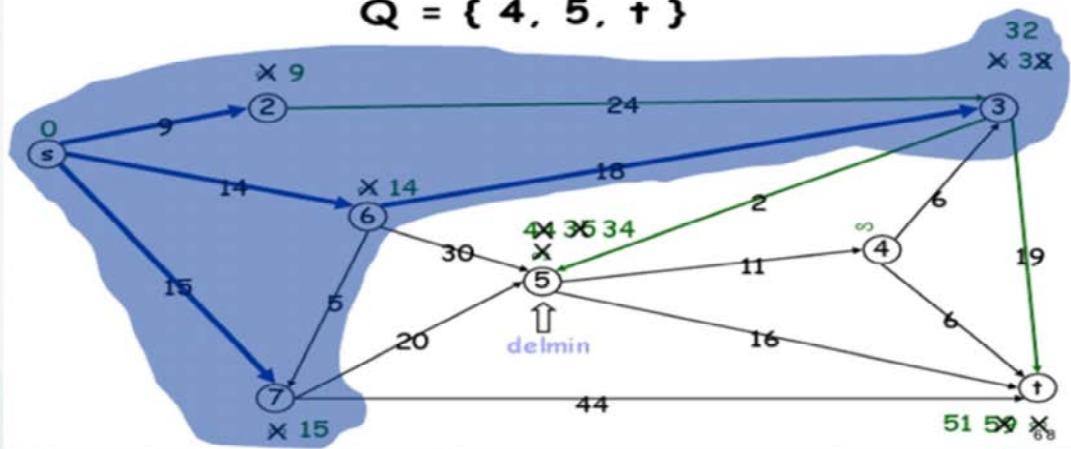
图搜索算法

Dijkstra算法



$$S = \{ s, 2, 3, 6, 7 \}$$

$$Q = \{ 4, 5, t \}$$



位形空间离散化规划方法

C-Space Discretizations

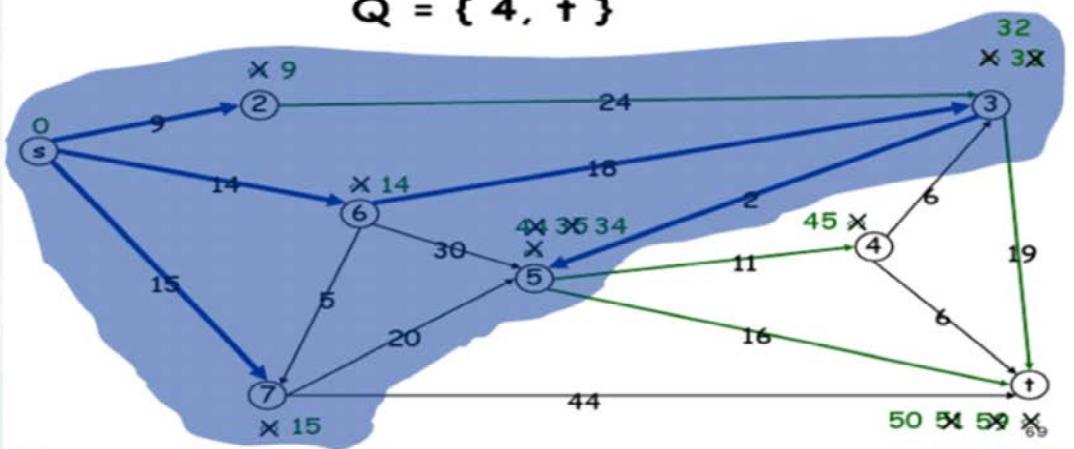
图搜索算法

Dijkstra算法



$$S = \{ s, 2, 3, 5, 6, 7 \}$$

$$Q = \{ 4, t \}$$



位形空间离散化规划方法

C-Space Discretizations

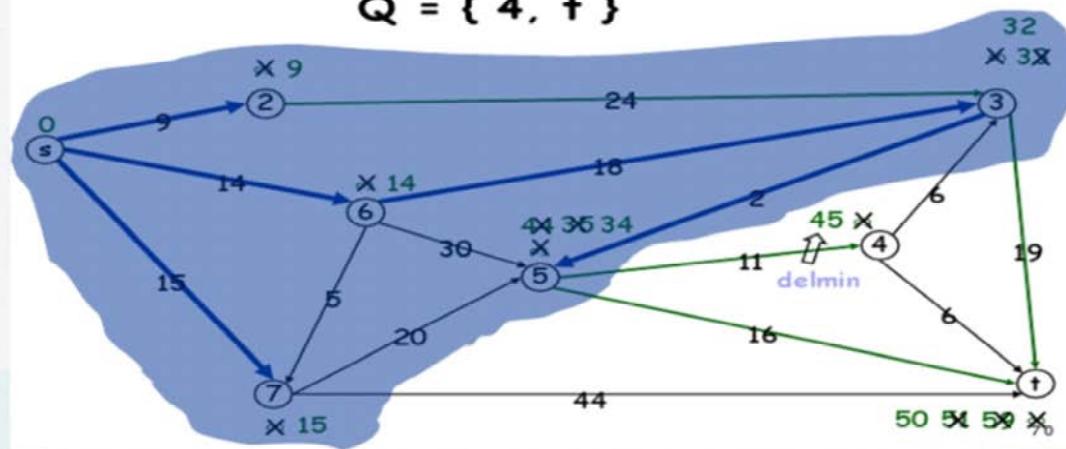
图搜索算法

Dijkstra算法



$$S = \{ s, 2, 3, 5, 6, 7 \}$$

$$Q = \{ 4, t \}$$



位形空间离散化规划方法

C-Space Discretizations

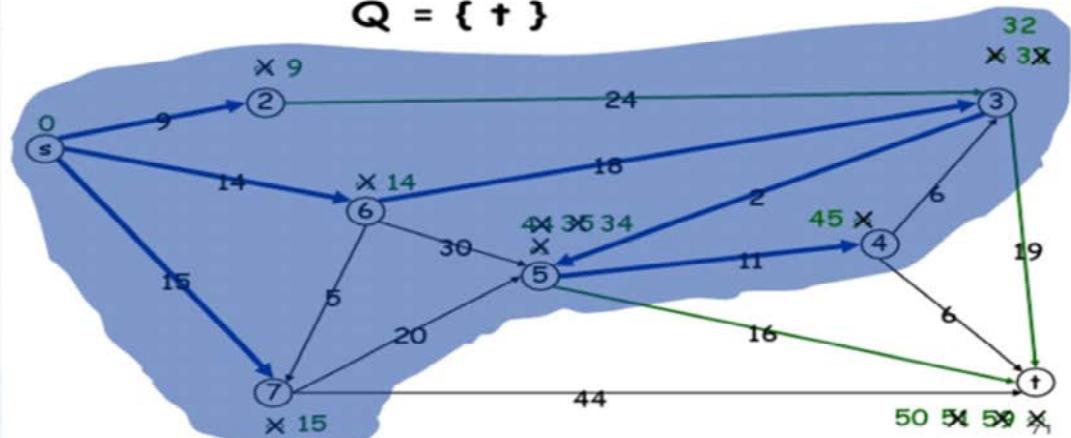
图搜索算法

Dijkstra算法



$$S = \{ s, 2, 3, 4, 5, 6, 7 \}$$

$$Q = \{ t \}$$



位形空间离散化规划方法

C-Space Discretizations

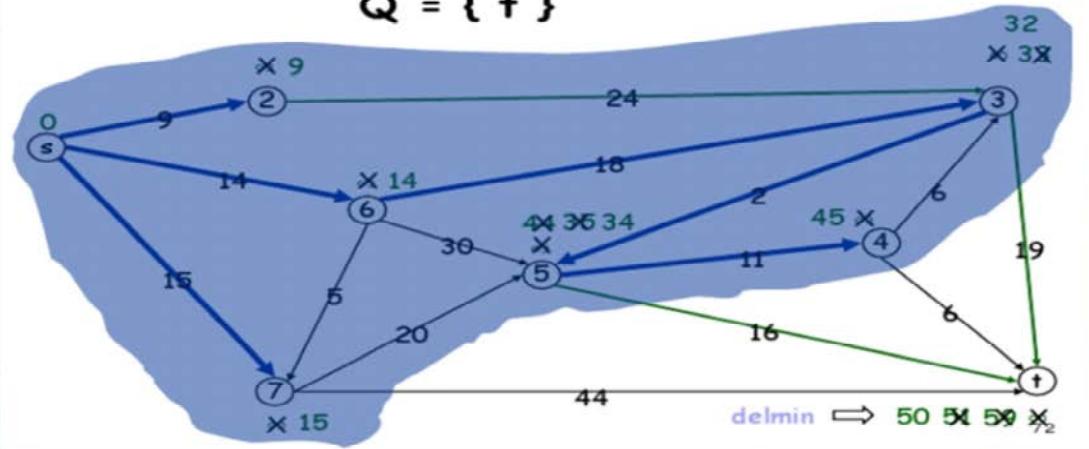
图搜索算法

Dijkstra算法



$$S = \{ s, 2, 3, 4, 5, 6, 7 \}$$

$$Q = \{ t \}$$



位形空间离散化规划方法

C-Space Discretizations

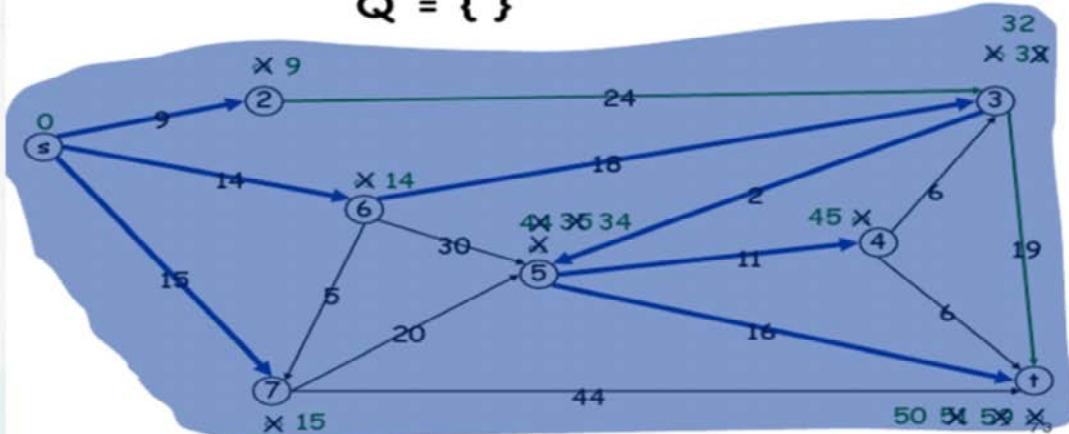
图搜索算法

Dijkstra算法



$$S = \{ s, 2, 3, 4, 5, 6, 7, t \}$$

$$Q = \{ \}$$



位形空间离散化规划方法

C-Space Discretizations

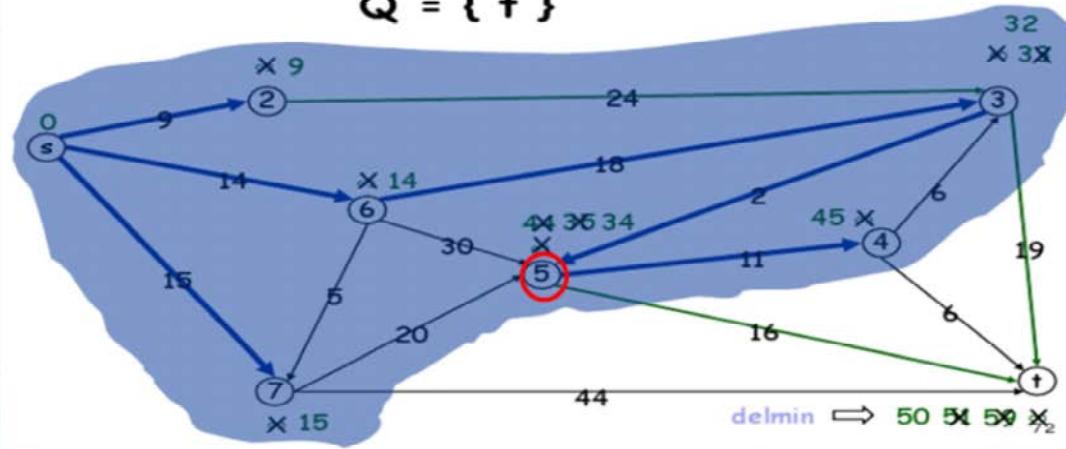
图搜索算法

Dijkstra算法



$$S = \{ s, 2, 3, 4, 5, 6, 7 \}$$

$$Q = \{ t \}$$



t的最近前节点是5

位形空间离散化规划方法

C-Space Discretizations

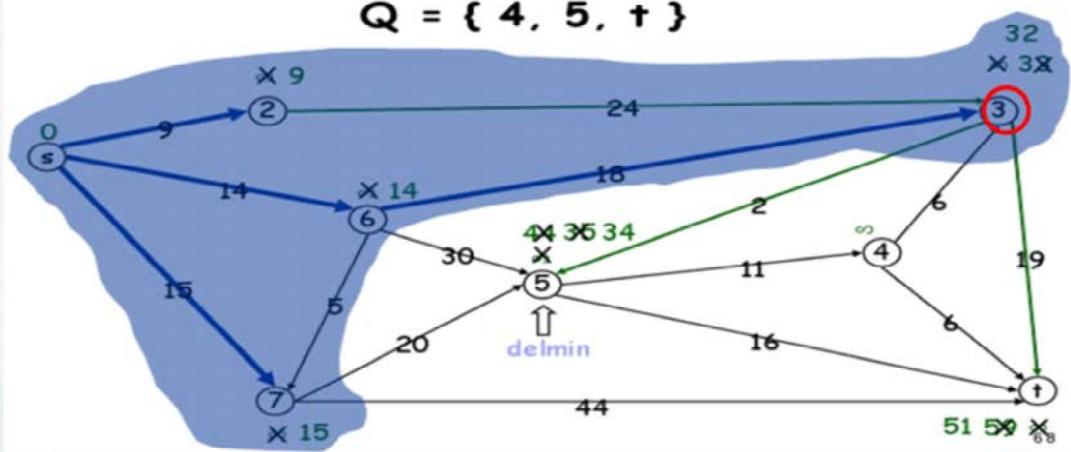
图搜索算法

Dijkstra算法



$$S = \{ s, 2, 3, 6, 7 \}$$

$$Q = \{ 4, 5, t \}$$



位形空间离散化规划方法

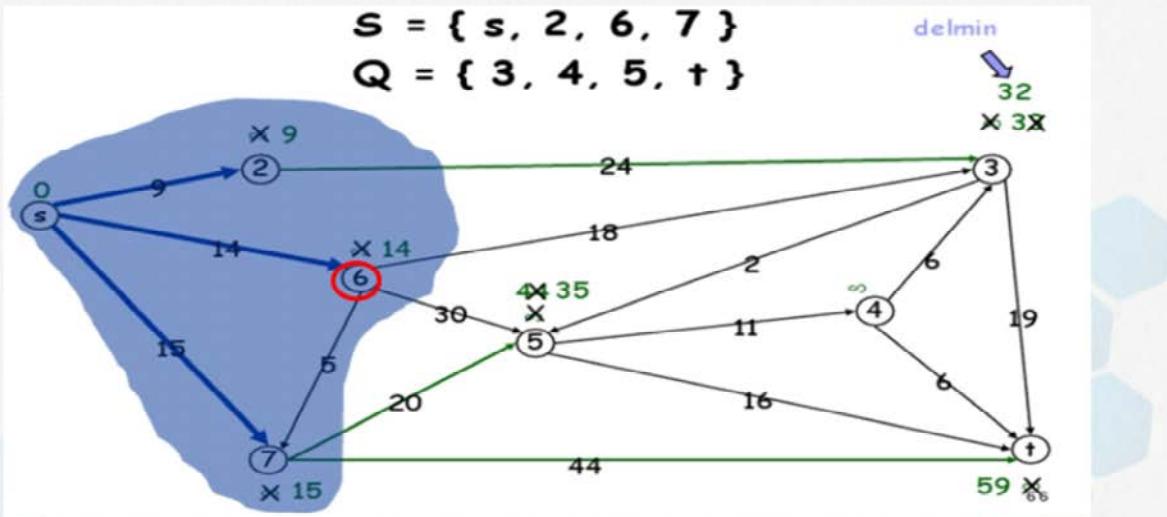
C-Space Discretizations

图搜索算法

Dijkstra算法



$$S = \{ s, 2, 6, 7 \}$$
$$Q = \{ 3, 4, 5, t \}$$



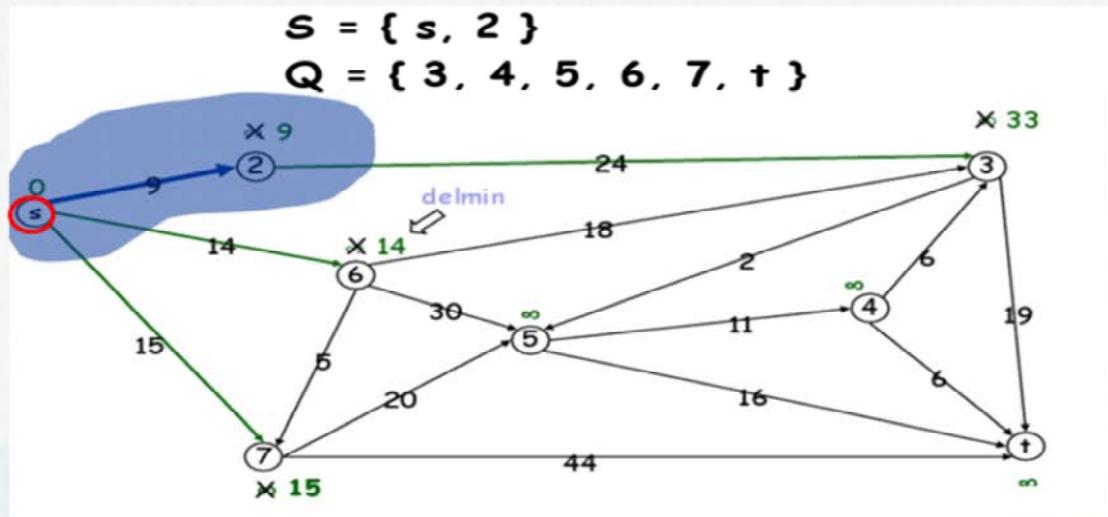
3的最近前节点是6

位形空间离散化规划方法

C-Space Discretizations

图搜索算法

Dijkstra 算法



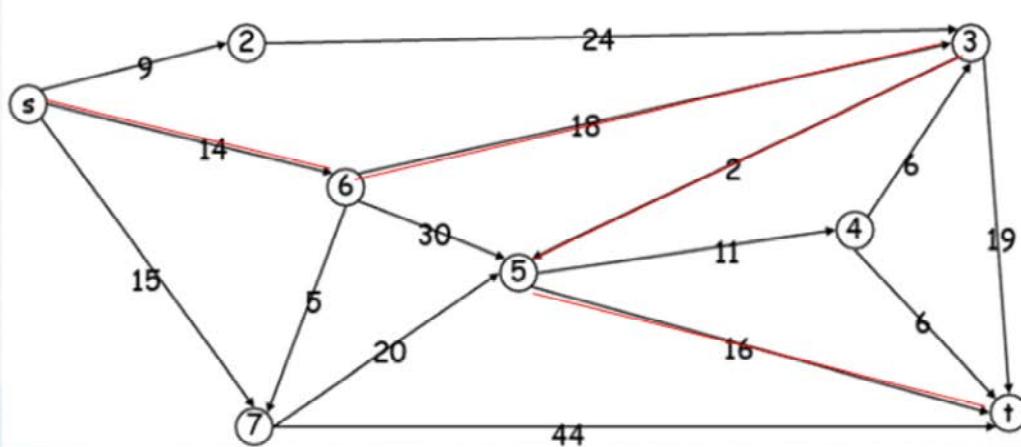
6的最近前节点是s

位形空间离散化规划方法

C-Space Discretizations

图搜索算法

Dijkstra算法



从s到t的最短路径: s,6,3,5,t $d(s,t)=50$



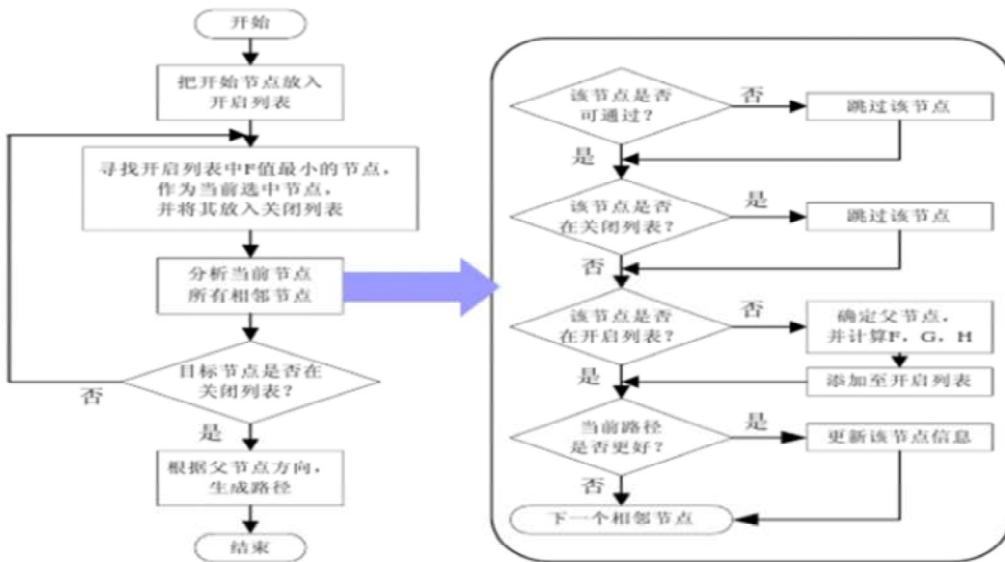
A*算法步骤:

- 1. 把起始格添加到开启列表。
- 2. 重复如下的工作:
 - a) 寻找开启列表中F值最低的格子。我们称它为当前格。
 - b) 把它切换到关闭列表。
 - c) 对相邻的8格中的每一个?
 - * 如果它不可通过或者已经在关闭列表中，略过它。反之如下。
 - * 如果它不在开启列表中，把它添加进去。把当前格作为这一格的父节点。记录这一格的F,G,和H值。
 - * 如果它已经在开启列表中，用G值为参考检查新的路径是否更好。更低的G值意味着更好的路径。如果是这样，就把这一格的父节点改成当前格，并且重新计算这一格的G和F值。如果你保持你的开启列表按F值排序，改变之后你可能需要重新对开启列表排序。
 - d) 停止，当你把目标格添加进了开启列表，这时候路径被找到，或者没有找到目标格，开启列表已经空了。这时候，路径不存在。
- 3. 保存路径。从目标格开始，沿着每一格的父节点移动直到回到起始格

在 Dijkstra 的基础上引入了启发式函数 $h(n)$

- 代价估计函数 $f(n)=g(n)+h(n)$
- $g(n)$ -从初始节点到节点n的累计成本；
- $h(n)$ -启发式函数，从节点n到目标节点的代价估计。

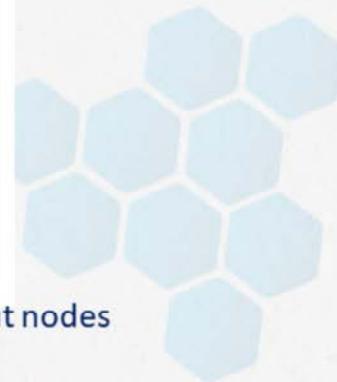
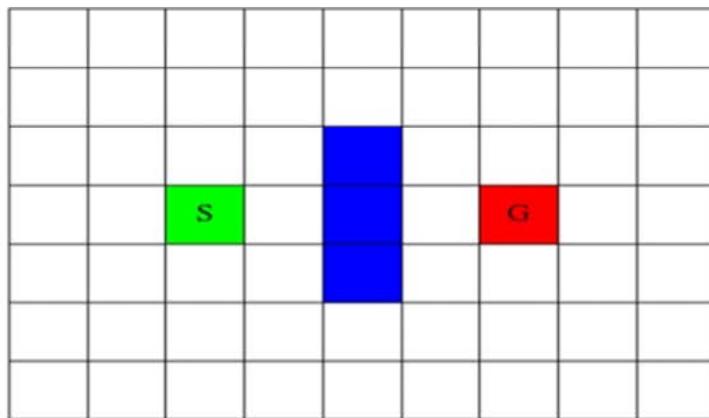
策略：每次访问 $f(n)$ 值最小的节点



40



First step: Environmental division



The search requires 2 lists to store information about nodes

- Open list (O) stores nodes for expansions
- Closed list (C) stores nodes which we have explored



第二步，生成“开启列表”：

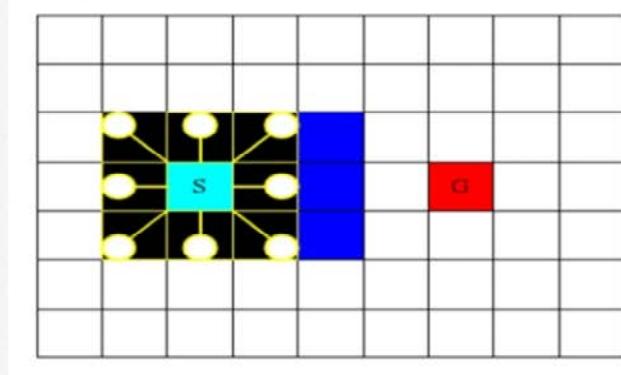
- 生成一个开启列表，并将 s 添加到开启列表；
- 寻找当前选中点（起点）周围所有可到达或可通过的方格，也把它们加入开启列表，为所有这些方格保存点 s 为“父方格”。





第三步，生成“关闭列表”：

- 从开启列表中删除节点s，把它加入到“关闭列表”。



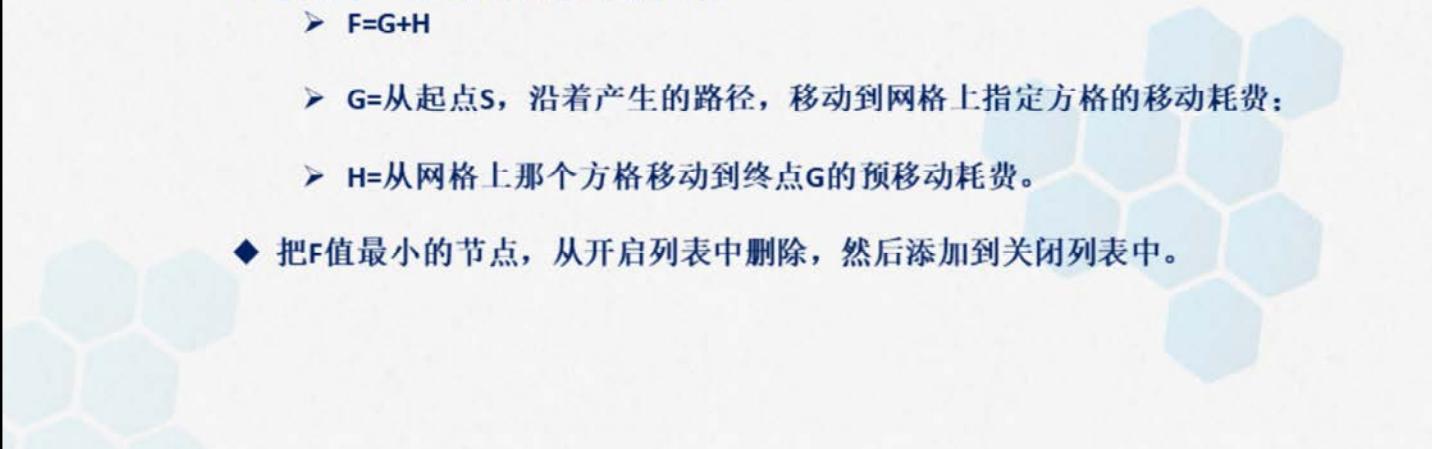
Generate a Closed list

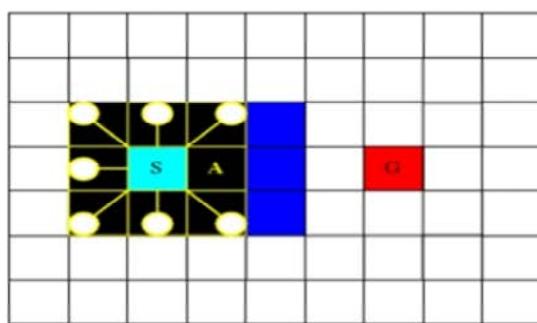
Delete S from opened list, add in closed list



第四步，更新“关闭列表”：

- ◆ 搜索下一个放入关闭列表的节点
 - $F=G+H$
 - G =从起点S，沿着产生的路径，移动到网格上指定方格的移动耗费；
 - H =从网格上那个方格移动到终点G的预移动耗费。
- ◆ 把F值最小的节点，从开启列表中删除，然后添加到关闭列表中。





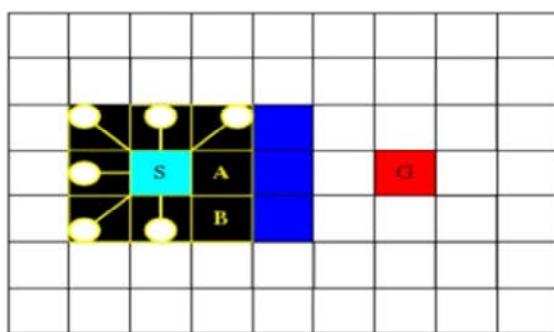
- G=10
- H=30
- F=G+H=40

位形空间离散化规划方法

C-Space Discretizations

图搜索算法

A*算法



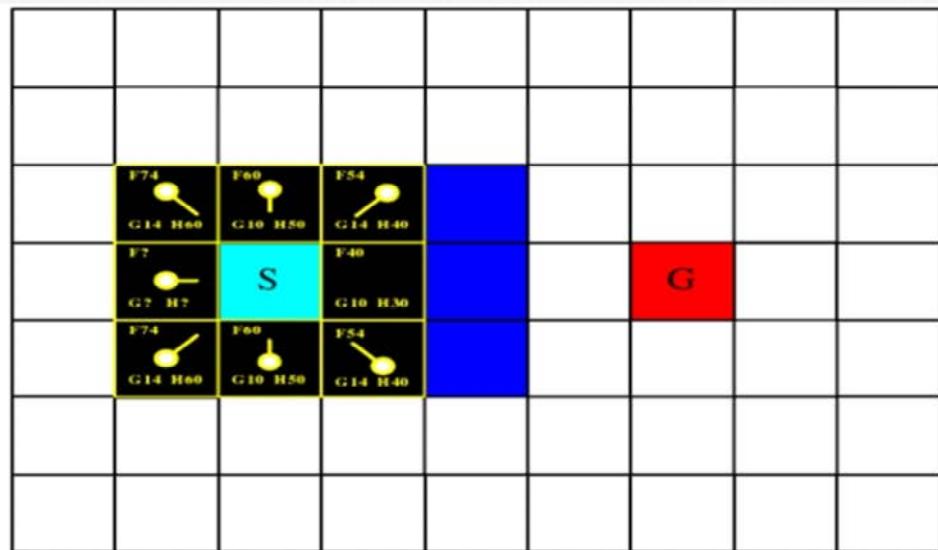
- G=14
- H=40
- F=G+H=54

位形空间离散化规划方法

C-Space Discretizations

图搜索算法

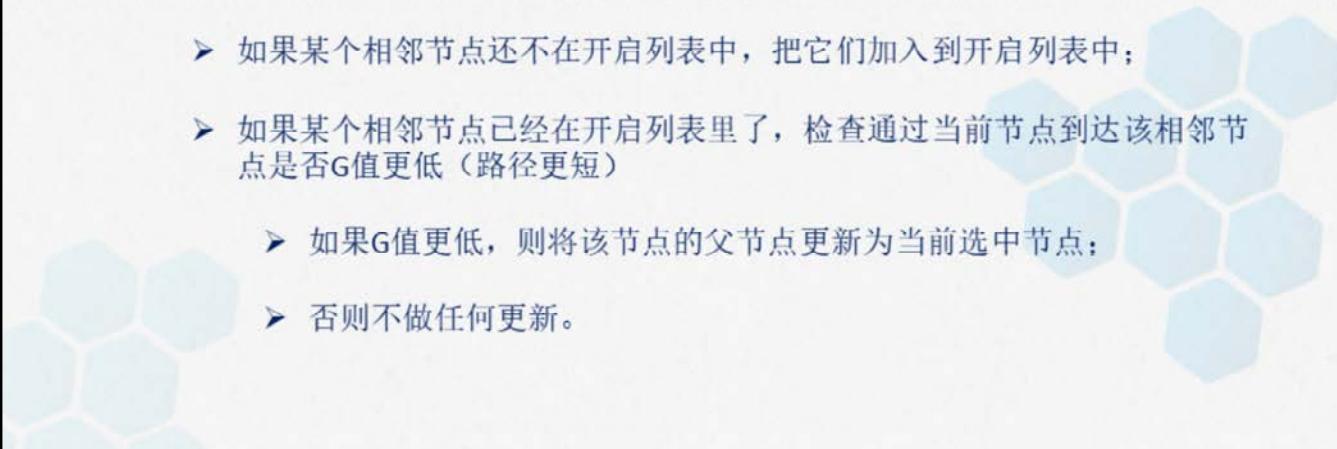
A*算法





第五步，更新“开启列表”

- 检查所有相邻节点；
- 如果某个相邻节点还不在开启列表中，把它们加入到开启列表中；
- 如果某个相邻节点已经在开启列表里了，检查通过当前节点到达该相邻节点是否G值更低（路径更短）
 - 如果G值更低，则将该节点的父节点更新为当前选中节点；
 - 否则不做任何更新。

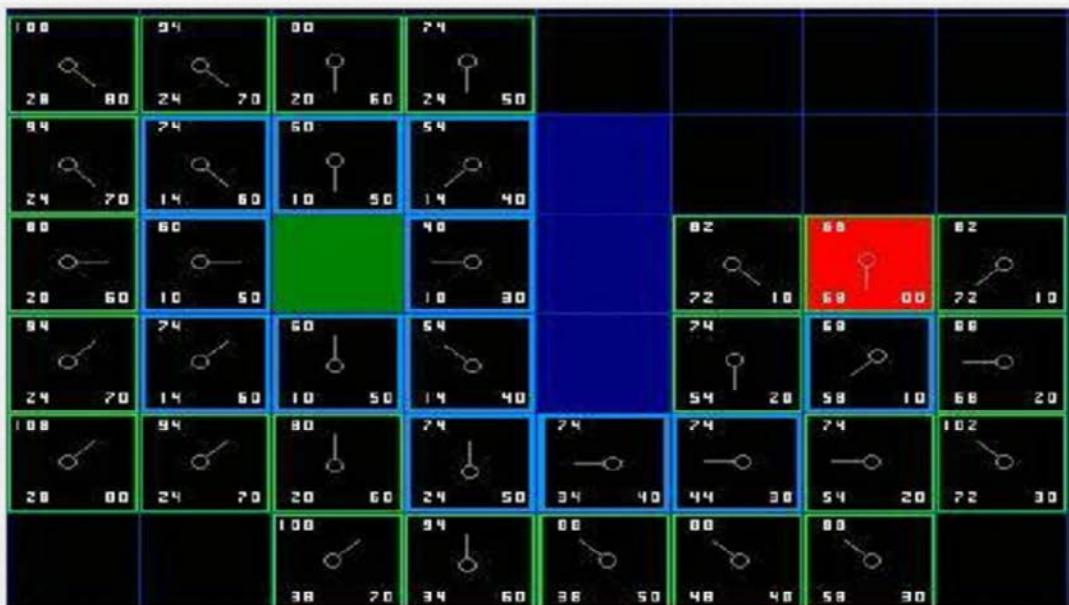


位形空间离散化规划方法

C-Space Discretizations

图搜索算法

A*算法





第六步，判断终点是否在开启列表

- 如果终点不在开启列表，则返回第四步重复；
- 如果终点已在开启列表中，则从终点开始，按照父节点方向，即可得到从起点到终点的路径。

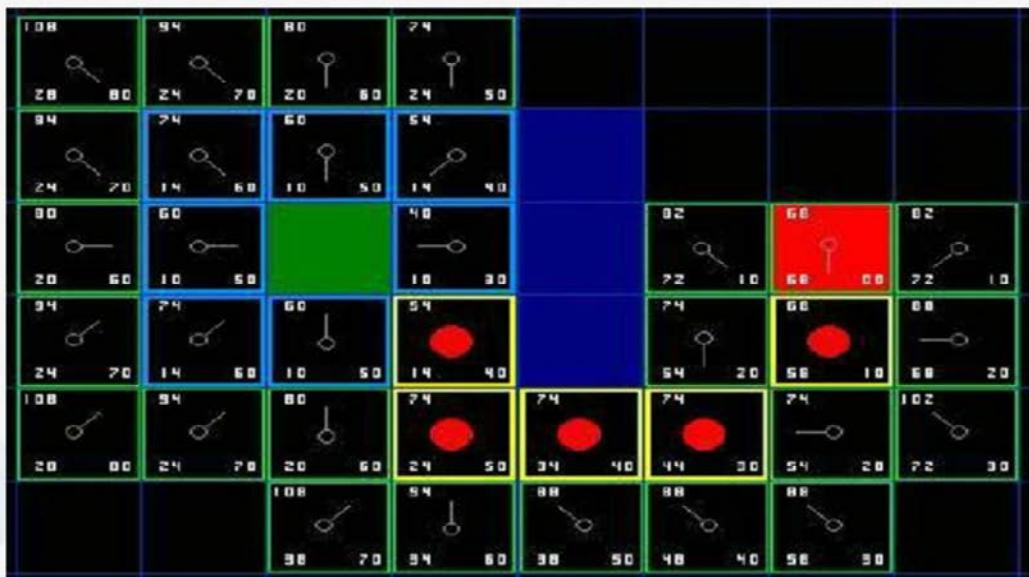


位形空间离散化规划方法

C-Space Discretizations

图搜索算法

A*算法





A*算法的特点

- A*算法是一种典型的启发式搜索算法;

$$\square F = G + H$$

- A*算法与广度优先及深度优先的联系:

- ✓ 当 $g(n)=0$ 时, 该算法类似DFS
- ✓ 当 $h(n)=0$ 时, 该算法类似于BFS

A*算法有待改进的地方:

- 路径的平滑性
- 搜索速度的快慢
- 三维地图?
- 非方形搜索区域?
- 不同的地形损耗
- 多个目标

.....

位形空间离散化规划方法

C-Space Discretizations

图搜索算法

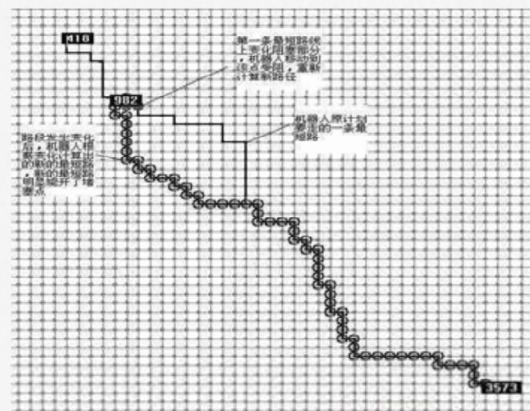
D*算法



- D*是动态A*（D-Star,Dynamic A Star）卡内基梅隆机器人的Stentz在1994和1995年两篇文章提出，主要用于机器人探路。曾经是火星探测器采用的寻路算法。

1. 先用Dijkstra算法从目标节点G向起始节点搜索。储存图中目标点到各个节点的最短路和该位置到目标点的实际值 h
2. 机器人沿最短路开始移动，在移动的下一节点没有变化时，无需计算，利用上一步Dijkstra计算出的最短路信息从出发点向后追溯即可，当在Y点探测到下一节点X状态发生改变，如堵塞。机器人首先调整自己在当前位置Y到目标点G的实际值 $h(Y)$ ， $h(Y)=X$ 到Y的新权值 $c(X,Y)+X$ 的原实际值 $h(X)$ 。X为下一节点(到目标点方向Y->X->G)，Y是当前点。k值取 h 值变化前后的最小。
3. 用A*或其它算法计算，这里假设用A*算法，遍历Y的子节点，点放入CLOSE，调整Y的子节点a的 h 值， $h(a)=h(Y)+Y$ 到子节点a的权重 $c(Y,a)$ ，比较a点是否存在于OPEN和CLOSE中

动态路径规划



Any-Angle A*, D* Lite.....

These are so many algorithms:

[D*](#) (1994)

[Focused D*](#) (1995)

[Dynamic SWSF-FP](#) (1996)

[LPA](#) (1997)

[LPA*/Incremental A*](#) (2001)

[D* Lite](#) (2002)

[SetA*](#) (2002)

[HPA*](#) (2004)

[Anytime D*](#) (2005)

[PRA*](#) (2005)

[Field D*](#) (2007)

[Theta*](#) (2007)

[HAA*](#) (2008)

[GAA*](#) (2008)

[LEARCH](#) (2009)

[BDDD*](#) (2009 - I cannot access this paper :|)

[Incremental Phi*](#) (2009)

[GFRA*](#) (2010)

[MTD*-Lite](#) (2010)

[Tree-AA*](#) (2011)



如果我们无法预先知道C-Space中的 C_{free} 和 C_{obs} ，机器人在探索 C_{free} 的时候，就会类似于在黑暗中摸索，那么这个时候唯一的光线就来自于碰撞检测算法，下面我们讨论几种基于概率的路径规划算法：

- 概率路线图（PRM Probabilistic road map）
[Kavraki et al, 1992]
- 快速搜索随机树（RRT Rapidly exploring random trees）
[Lavalle and Kuffner, 1999]

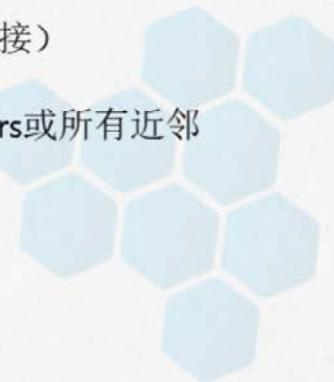




思路：从C-Space中随机抽取样本，如果在 C_{free} 中，则将它们声明为顶点，尝试使用局部规划器连接附近的顶点。

- ◆ 局部规划器：检测视线(line-of-sight)是否有碰撞（简单直接）
- ◆ k-nearest neighbors选择：指定半径内的k-nearest neighbors或所有近邻
- ◆ 将位形和连接添加到图(graph)中，直到路线图足够密集
- ◆ 采用图搜索算法寻找最优路径

Consists of two phases !



位形空间离散化规划方法

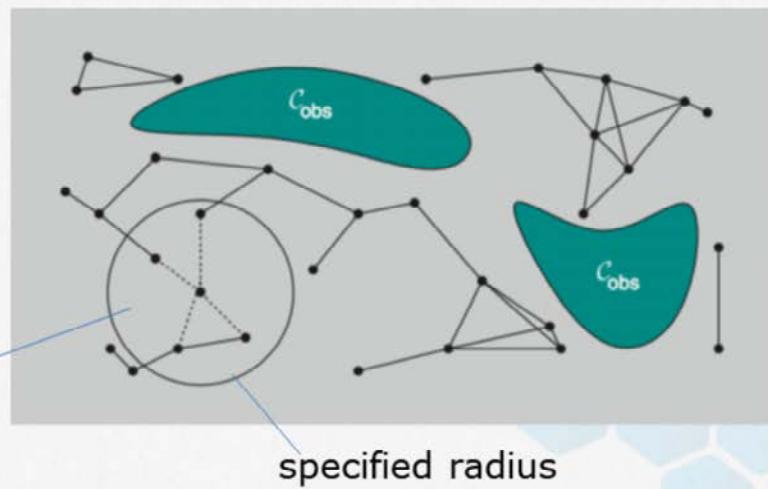
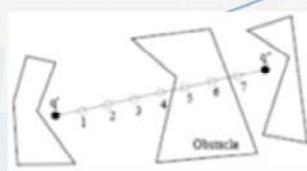
C-Space Discretizations

基于采样的方法

概率路线图 (PRM)



举例



**Algorithm** 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

```

- 基于随机采样技术的PRM法可以有效解决高维空间和复杂约束中的路径规划问题。
- PRM是一种基于图搜索的方法，它将连续空间转换成离散空间，再利用A*等搜索算法在路线图上寻找路径，以提高搜索效率。
- 这种方法能用相对少的随机采样点来找到一个解，对多数问题而言，相对少的样本足以覆盖大部分可行的空间，并且找到路径的概率为1（随着采样数增加，P（找到一条路径）指数的趋向于1）。
- 显然，当采样点太少，或者分布不合理时，PRM算法是不完备的，但是随着采用点的增加，也可以达到完备。所以PRM是概率完备且不最优的。

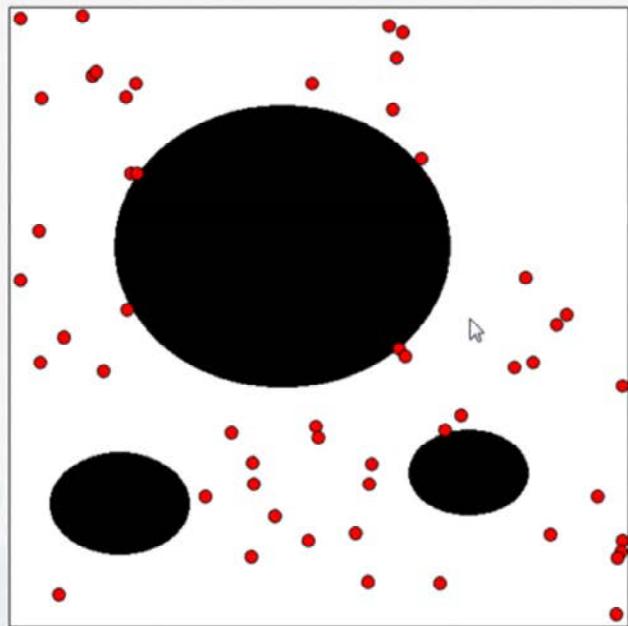
The probabilistic roadmap planner consists of two phases: a construction and a query phase. 学习阶段：在给定图的自由空间里随机撒点（自定义个数），构建一个路径网络图。查询阶段：采用图搜索算法（比如A*）寻找从起点到终点的路径。

位形空间离散化规划方法

C-Space Discretizations

基于采样的方法

概率路线图 (PRM)



- 学习阶段：在给定图的自由空间里随机撒点（自定义个数），构建一个路径网络图；
- 查询阶段：采用图搜索算法（比如A*）寻找从起点到终点的路径。

Pros:

- Probabilistically complete
- Do not construct C-space
- Apply easily to high-dim. C's
- PRMs have solved previously unsolved problems

Cons:

- Do not work well for some problems, narrow passages
- Not optimal, not complete

位形空间离散化规划方法

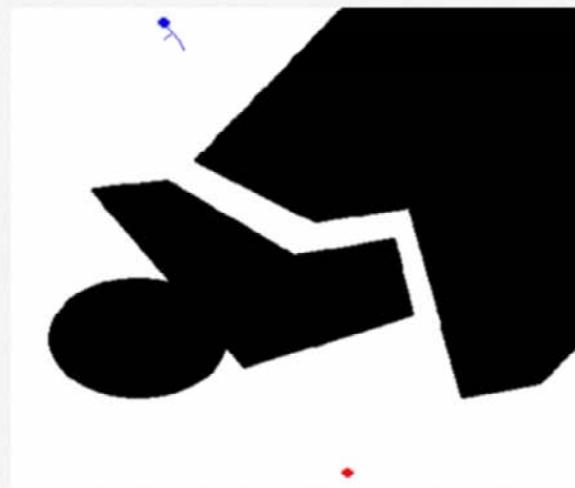
C-Space Discretizations

基于采样的方法

快速搜索随机树 (RRT)



思路：从初始位形 q_0 开始，积极探索C空间，通过状态空间的随机采样点，把搜索导向空白区域，从而寻找到一条从起始点到目标点的规划路径，探索出的区域为以 q_0 为根节点的一棵树(tree)。

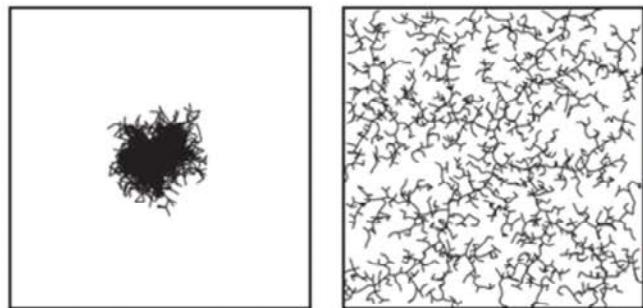




```

1 function BuildRRT(qinit, K, Δq)
2   T.init(qinit)
3   for k = 1 to K
4     qrand = Sample() -- chooses a random configuration
5     qnearest = Nearest(T, qrand) -- selects the node in the RRT tree that is closest to qrand
6     if Distance(qnearest, qgoal) < Threshold then
7       return true
8     qnew = Extend(qnearest, qrand, Δq) -- moving from qnearest an incremental distance in the direction of qrand
9     if qnew ≠ NULL then
10      T.AddNode(qnew)
11    return false
12
13
14 function Sample() -- Alternatively, one could replace Sample with SampleFree (by using a collision detection algorithm to reject samples in
C_obstacle
15   p = Random(0, 1.0)
16   if 0 < p < Prob then
17     return qgoal
18   elseif Prob < p < 1.0 then
19     return RandomNode()

```



RRT是一种多维空间中有效率的规划方法。它以一个初始点作为根节点，通过随机采样增加叶子节点的方式，生成一个随机扩展树，当随机树中的叶子节点包含了目标点或进入了目标区域，便可以在随机树中找到一条由从初始点到目标点的路径。

初始化时随机树T只包含一个节点：根节点 q_{init} 。首先Sample函数从状态空间中随机选择一个采样点 q_{rand} （4行）；

然后Nearest函数从随机树中选择一个距离 q_{rand} 最近的节点 $q_{nearest}$ （5行）；

最后Extend函数通过从 $q_{nearest}$ 向 q_{rand} 扩展一段距离，得到一个新的节点 q_{new} （8行）。如果 q_{new} 与障碍物发生碰撞，则Extend函数返回空，放弃这次生长，否则将 q_{new} 加入到随机树中。

重复上述步骤直到 $q_{nearest}$ 和目标点 q_{goal} 距离小于一个阈值，则代表随机树到达了目标点，算法返回成功（6~7行）。

为了使算法可控，可以设定运行时间上限或搜索次数上限（3行）。如果在限制次数内无法到达目标点，则算法返回失败。

为了加快随机树到达目标点的速度，简单的改进方法是：

在随机树每次的生长过程中，根据随机概率来决定 q_{rand} 是目标点还是随机点。

在Sample函数中设定参数Prob，每次得到一个0到1.0的随机值p，当 $0 < p < Prob$ 的时候，随机树朝目标点生长；

当 $Prob < p < 1.0$ 时，随机树朝一个随机方向生长。

上述算法的效果是随机采样点会“拉着”树向外生长，这样能更快、更有效地探索空间（The effect is that the nearly uniformly distributed samples “pull” the tree toward them, causing the tree to rapidly explore C-Space）。

随机探索也讲究策略，如果我们从树中随机取一个点，然后向着随机的方向生长，那么结果是什么样的呢？

见下图（Left: A tree generated by applying a uniformly-distributed random motion from a randomly chosen tree node does not explore very far. Right: A tree generated by the RRT algorithm using samples drawn randomly from a uniform distribution. Both trees have 2000 nodes）。

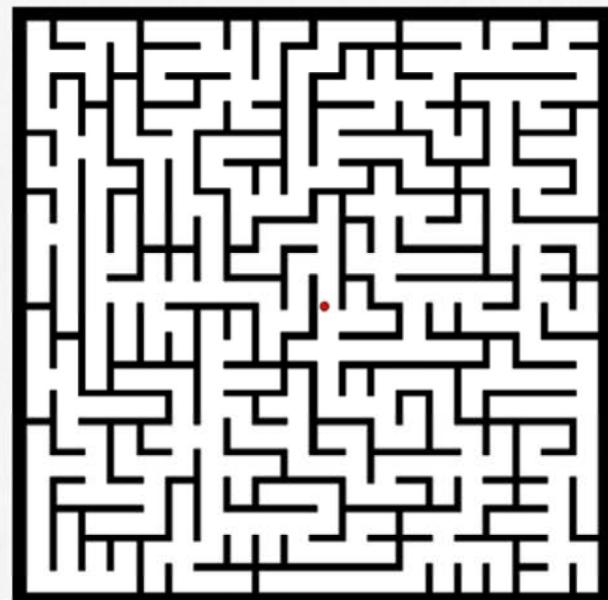
可以看到，同样是随机树，但是这棵树并没很好地探索空间。

位形空间离散化规划方法

C-Space Discretizations

基于采样的方法

快速搜索随机树（RRT）

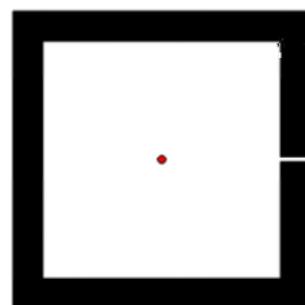
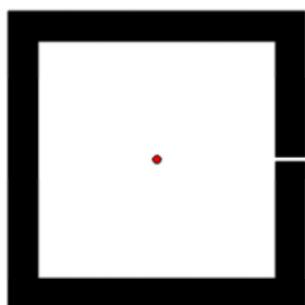


对环境类型不敏感，当C-空间包含大量障碍物或狭窄通道约束时，算法的收敛速度慢，效率会大幅下降

RRT算法也有一些缺点，它是一种纯粹的随机搜索算法对环境类型不敏感，当C-空间中包含大量障碍物或狭窄通道约束时，算法的收敛速度慢，效率会大幅下降



这种情况下如何改进?



RRT 的一个弱点是难以在有狭窄通道的环境找到路径。因为狭窄通道面积小，被碰到的概率高。下图展示的例子是 RRT 应对一个人为制作的很短的狭窄通道，有时 RRT 很快就找到了出路，有时则一直被困在障碍物里面：

05

势场法

前面讨论的所有技术都旨在将Cfree的连通图中寻找避碰路径，而势场法遵循不同的理念：将机器人抽象为一个粒子，在人工势场的作用下，机器人在排斥力和吸引力的合力作用下向目标点移动。

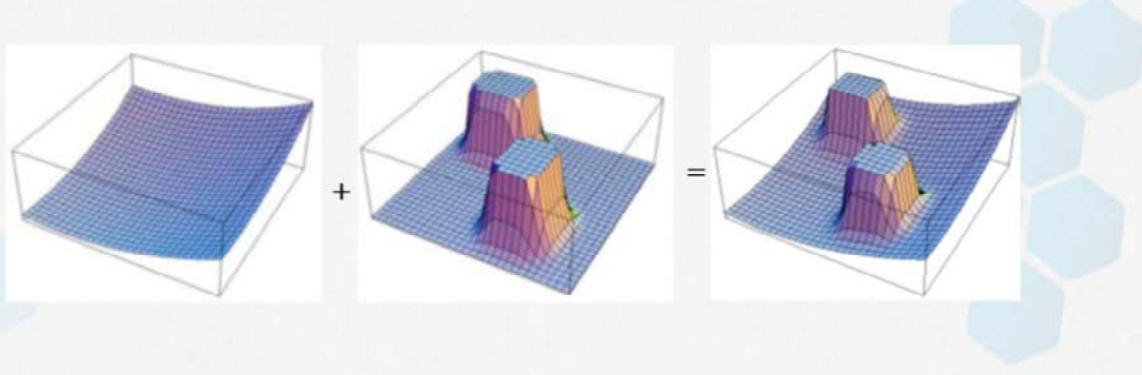
势场法

Potential Field Methods



障碍物对机器人施加排斥力，目标点对机器人施加吸引力合力形成势场，机器人移动就像球从山上滚下来一样，机器人在合力作用下向目标点移动。

$$U(q) = U_{att}(q) + U_{rep}(q) \quad \vec{F}(q) = -\vec{\nabla}U(q)$$



只需简单执行梯度下降
C-Space通常在用网格离散化

势场法

Potential Field Methods

人工势场法 (Artificial potential field method)



引力场:

$$U_{att}(q) = \frac{1}{2} k_{att} \cdot \rho_{goal}^2(q)$$

$$F_{att}(q) = -k_{att} \cdot (q - q_{goal})$$

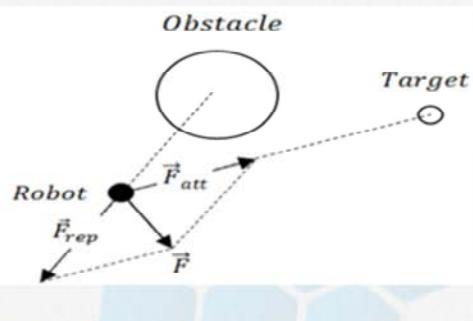
其中 k_{att} 是比例因子, $\rho_{goal}(q)$ 是位置 q 到目标点的直线距离 $\|q - q_{goal}\|$.

斥力场:

$$U_{rep}(q) = \begin{cases} \frac{1}{2} k_{rep} \left(\frac{1}{\rho(q)} - \frac{1}{\rho_r} \right)^2 & \text{if } \rho(q) \leq \rho_r \\ 0 & \text{if } \rho(q) \geq \rho_r \end{cases}$$

$$F_{rep}(q) = \begin{cases} k_{rep} \left(\frac{1}{\rho(q)} - \frac{1}{\rho_r} \right) \frac{1}{\rho^2(q)} \frac{q - q_{obstacle}}{\rho(q)} & \text{if } \rho(q) \leq \rho_r \\ 0 & \text{if } \rho(q) \geq \rho_r \end{cases}$$

机器人所受到的合力为: $F_{all} = F_{att} + F_{rep}$



该方法结构简单, 广泛用于实时避障和平滑的轨迹控制方面, 但存在局部极小问题、抖动问题及目标不可达问题, 容易产生死锁现象。

目标不可达的主要原因

当目标在障碍物的影响范围之内时, 整个势场的全局最小点并不是目标点。

局部极小的主要原因

机器人在向目标点和障碍物移动过程中, 引力不断减小, 斥力不断增大。因此, 必然存在一点使得机器人所受的斥力与引力恰好大小相等、方向相反。此时, 机器人所受合力为零, 机器人将停在障碍物前。

避免两者的改进算法

改进势力场函数, 使整个势场极小值仅在目标点出现

将APF同别的方法融合来克服人工势场的缺点

这部分的改进讲解见讲义

06

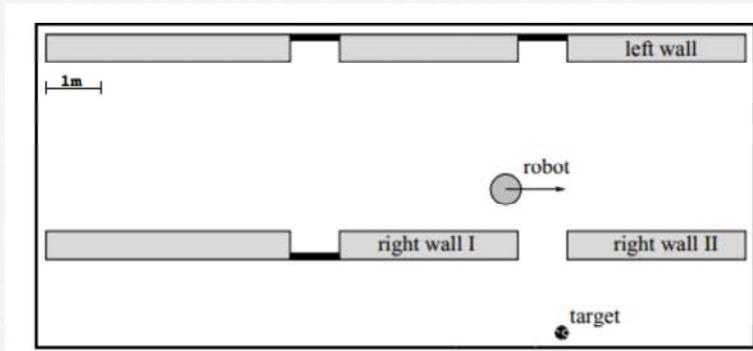
动态窗口法(DWA)

除了人工势场法之外，动态窗口法(Dynamic Window Approach,DWA)也可以作为局部规划方法。动态窗口法直接从机器人的动力学出发，对机器人速度和加速度施加约束，进行运动规划。它由两个主要部分组成：第一部分生成一个有效的搜索空间，一般这个搜索空间仅限于可以在短时间内到达且没有碰撞的安全圆弧轨迹，例如差速驱动机器人，其搜索空间为速度空间(v, ω)；第二部分在搜索空间中选择最佳解决方案，其优化目标一般是选择一个方向和速度，使机器人以最大的障碍物间隙距离到达目标。

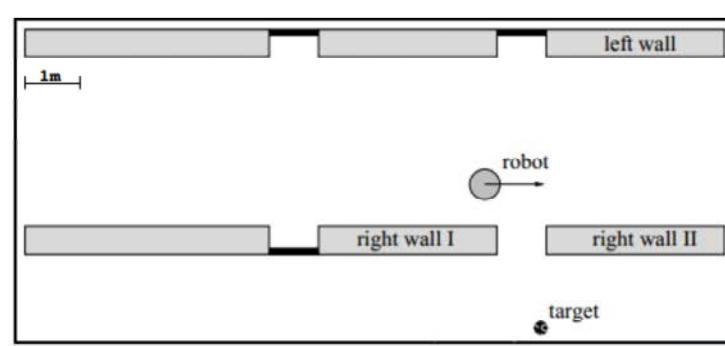
动态窗口法

Dynamic Window Approach

DWA(Dynamic Window Approach)



如图5.28所示，移动机器人在走廊中行进，机器人要到达目标位置，需要右转通过两面墙之间的缝隙。



障碍物速度约束公式

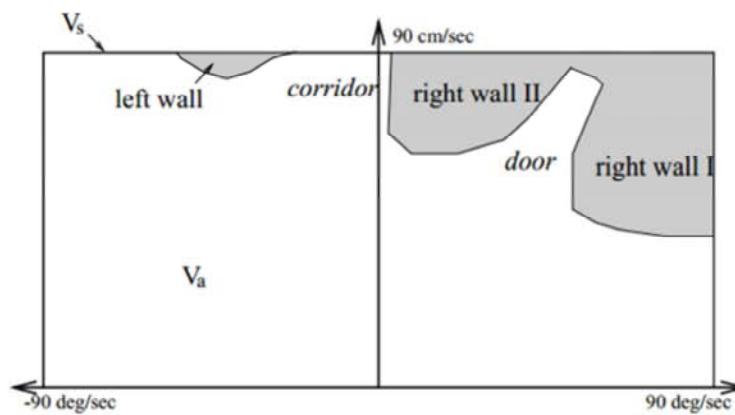
$$V_a = \{(v, \omega) | v \leq \sqrt{2 \cdot dist(v, \omega) \cdot \dot{v}_b} \wedge \omega \leq \sqrt{2 \cdot dist(v, \omega) \cdot \dot{\omega}_b}\}$$

假设该移动机器人的速度空间为 V_s , 最大线速度 $v_{max} = 90\text{cm/s}$, 最大角速度 $\omega_{max} = \pm 90\text{deg/s}$ 。首先我们对这个速度空间 $V_s(v, \omega)$ 进行障碍物约束。当机器人前方有障碍物时, 要保证当前的速度不要过大, 避免以最大加速度制停时机器人无法在障碍物前停下的情况发生。这个障碍物速度约束公式表达如公式(5-40)。

动态窗口法

Dynamic Window Approach

DWA(Dynamic Window Approach)



障碍物速度约束公式

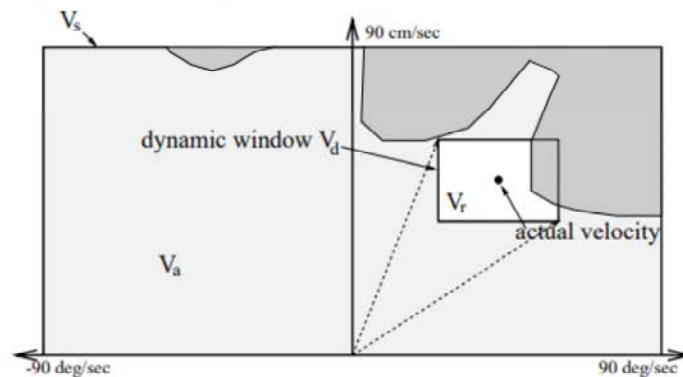
$$V_a = \{(v, \omega) | v \leq \sqrt{2 \cdot dist(v, \omega) \cdot \dot{v}_b} \wedge \omega \leq \sqrt{2 \cdot dist(v, \omega) \cdot \dot{\omega}_b}\}$$

其中， $dist(v, \omega)$ 为以该速度和角速度拟合出的圆弧上，机器人到障碍物的距离。经过障碍物约束之后的允许速度(Admissible Velocities)空间为 V_a ，如图5.29所示，为去掉灰色部分的其余空间。

动态窗口法

Dynamic Window Approach

DWA(Dynamic Window Approach)



动态窗口约束方程

$$V_d = \{(v, \omega) | v \in [v_a - \dot{v} \cdot t, v_a + \dot{v} \cdot t] \wedge \omega \in [\omega_a - \dot{\omega} \cdot t, \omega_a + \dot{\omega} \cdot t]\}$$

进一步进行动态窗口约束，限制搜索空间为短时间内可达的速度和角速度，约束方程为公式（5-41）。

搜索空间逐步缩小， $Space = V_s \cap V_a \cap V_d$ ， V_d 最终为一个动态窗口(图中小矩形内的白色部分)，表示速度的可达范围，如图5.30所示。

动态窗口法

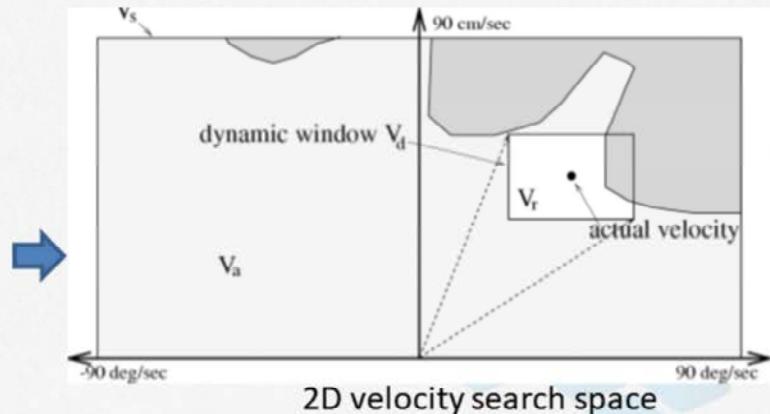
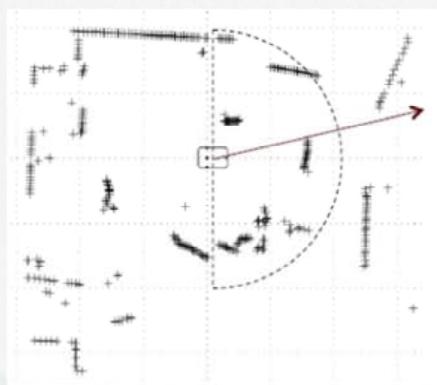
Dynamic Window Approach

DWA(Dynamic Window Approach) 如何选择(v , ω)?



已知：目标路径（一组途径点）、可对局部邻近区域进行距离扫描、动态约束；

控制：机器人无碰撞、安全、快速向目标移动。



V_s : 机器人的所有可能速度

V_a : 无障碍区速度

V_d : 给定加速度约束条件下一个时间范围内可达到的速度

$$Space = V_s \cap V_a \cap V_d$$

如何选择 (v , ω) ?

将问题作为动态窗口内目标函数的优化问题，搜索最大值

目标函数是一个启发式导航函数

该功能通过“在正确的方向上快速、安全地驾驶”来设置激励措施，最大限度减少运动耗费时间



轨迹评价目标函数

$$G(v, \omega) = \sigma(\alpha \cdot heading(v, \omega) + \beta \cdot dist(v, \omega) + \gamma \cdot vel(v, \omega))$$

如果应用到全局规划和局部规划结合：

假设已经从第一层的导航函数(NF1)算法中预先计算了目标距离，对(v, ω)-space进行约束：

启发式导航函数：

$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

最大速度

Rewards alignment to
NF1/A* gradient

Rewards large advances on
NF1/A* path

Comes in when goal region
reached

确定了速度搜索空间之后，对短时间内速度的轨迹进行模拟，并对轨迹进行评价，找到最优的轨迹，则对应的速度即为DWA规划的结果。轨迹评价目标函数见公式(5-42)。

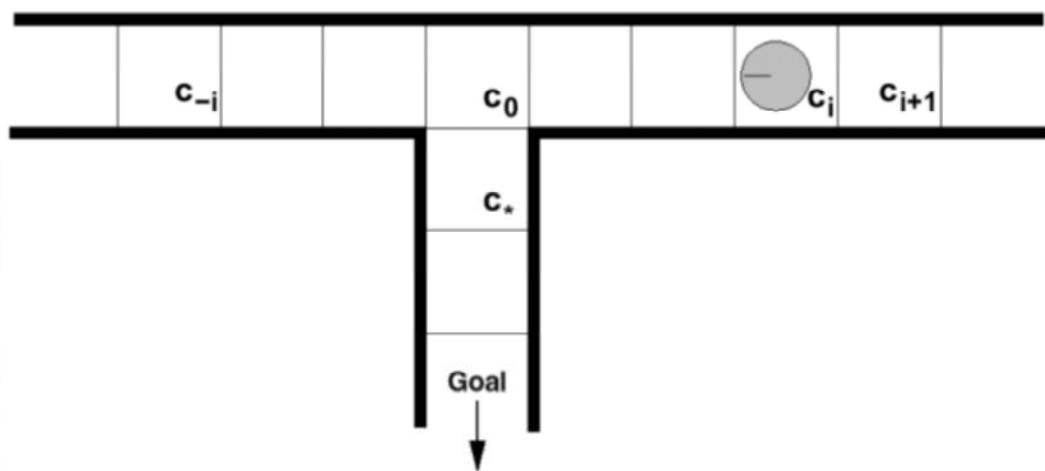
其中，评价目标函数的第一部分为朝向评价(target heading)，代表目标点的引力，权重为 α ， $heading(v, \omega)$ 为轨迹末端机器人的朝向和一根连线（机器人与目标点的连线）之间的夹角，这个夹角越小越好。第二部分为障碍物评价(间隙clearance)，代表障碍物产生的斥力，权重为 β ， $dist(v, \omega)$ 为机器人在当前轨迹上与最近的障碍物之间的间隙距离。如果在这条轨迹上没有障碍物，那就将其设定一个常数。第三部分为速度评价，代表对机器人的驱动。如果机器人自转到一个和目标对齐的朝向时，它可以原地不动就能满足前两项评价标准，这是需要避免的结果。为了克服机器人的这种“惰性”，引入速度评价 $vel(v, \omega)$ ，选择速度和角速度的幅度越大，该函数值越大，该函数将使机器人在合理范围内尽可能以更大的速度和角速度移动。

一般来说这三部分的评价不会直接进行相加，而需要进行归一化，也即每一项除以每一项的总和之后，变为一个小于1的百分比，然后再加权相加。其目的是为了避免某项评价值太大，导致其在评价目标函数中太占优势，引起评价目标函数的不连续。

动态窗口法

Dynamic Window Approach

DWA(Dynamic Window Approach)

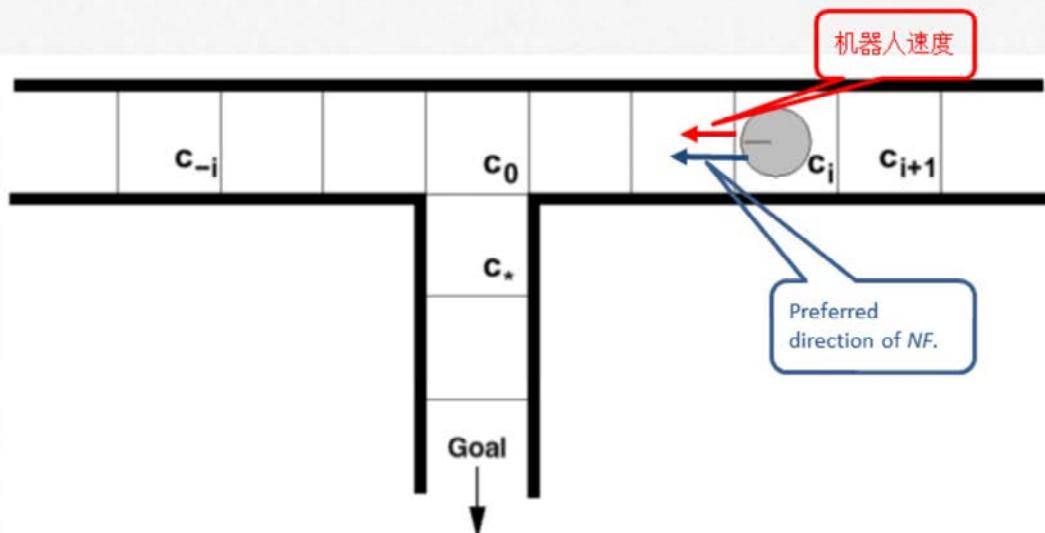


$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

动态窗口法

Dynamic Window Approach

DWA(Dynamic Window Approach)

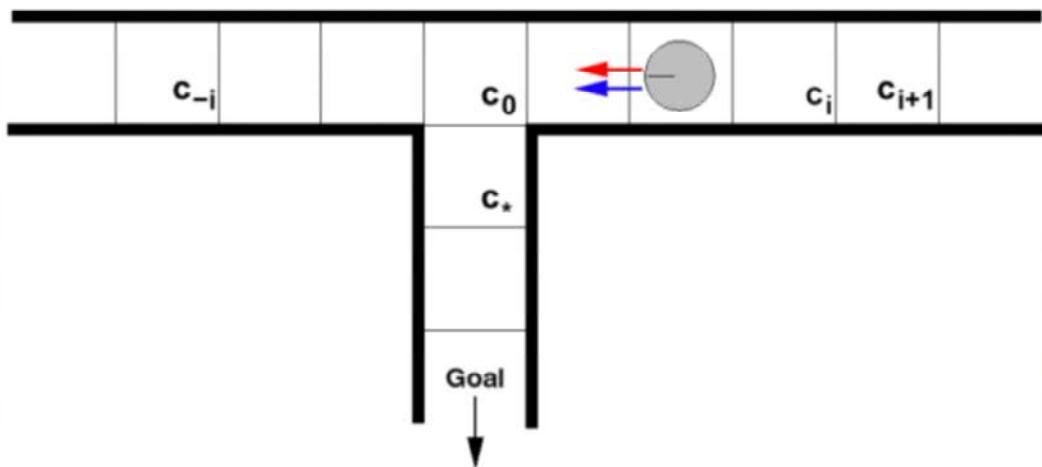


$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

动态窗口法

Dynamic Window Approach

DWA(Dynamic Window Approach)

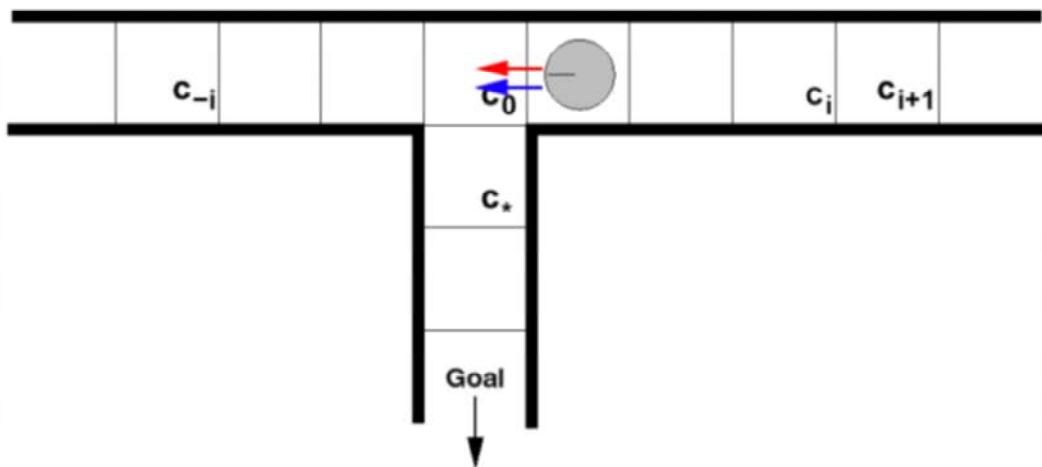


$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

动态窗口法

Dynamic Window Approach

DWA(Dynamic Window Approach)



$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

动态窗口法

Dynamic Window Approach

DWA(Dynamic Window Approach)

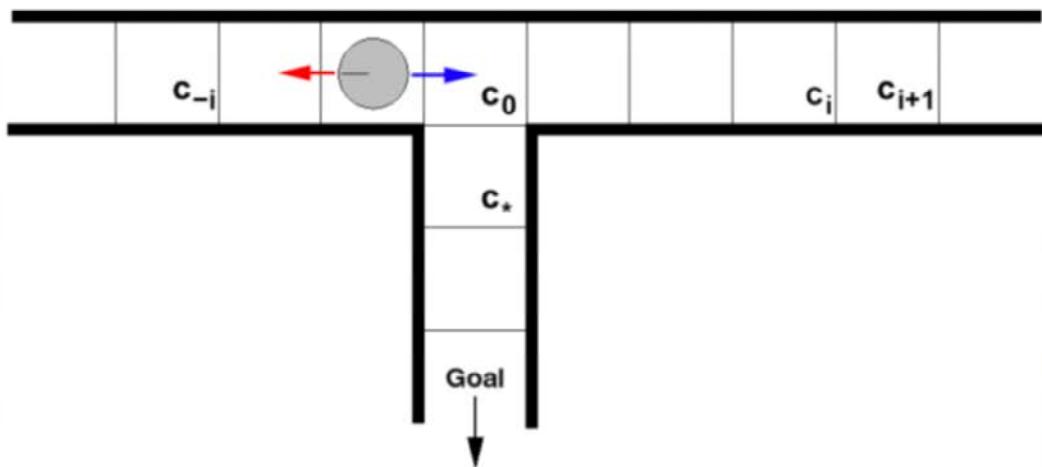


$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

动态窗口法

Dynamic Window Approach

DWA(Dynamic Window Approach)

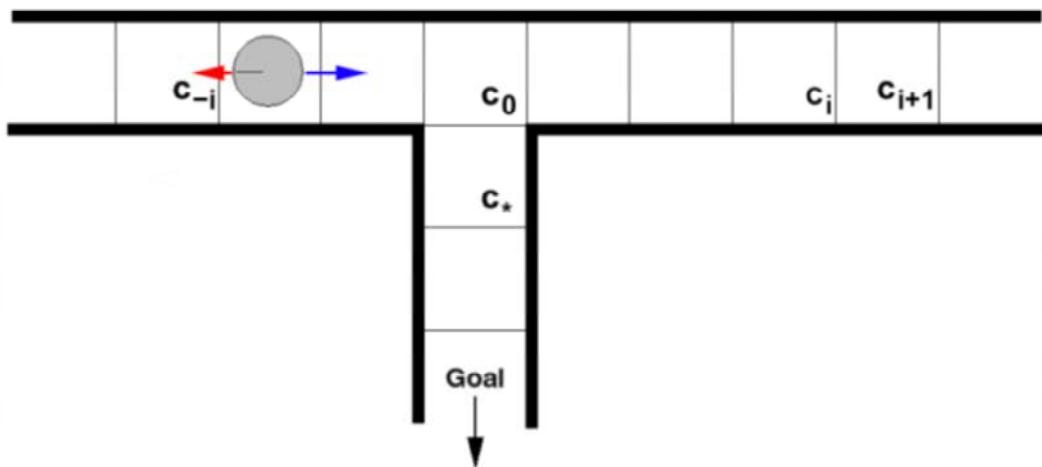


$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

动态窗口法

Dynamic Window Approach

DWA(Dynamic Window Approach)

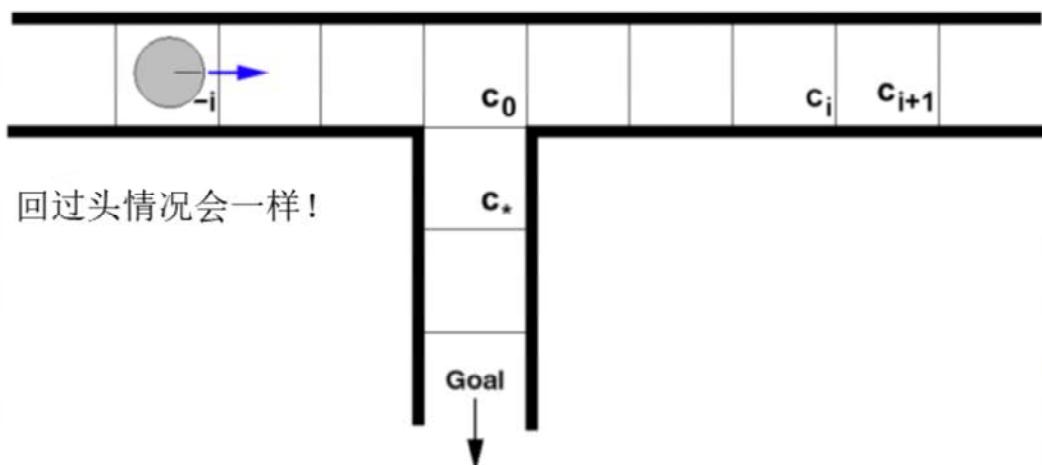


$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

动态窗口法

Dynamic Window Approach

DWA(Dynamic Window Approach)



$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

动态窗口法

Dynamic Window Approach

DWA(Dynamic Window Approach)

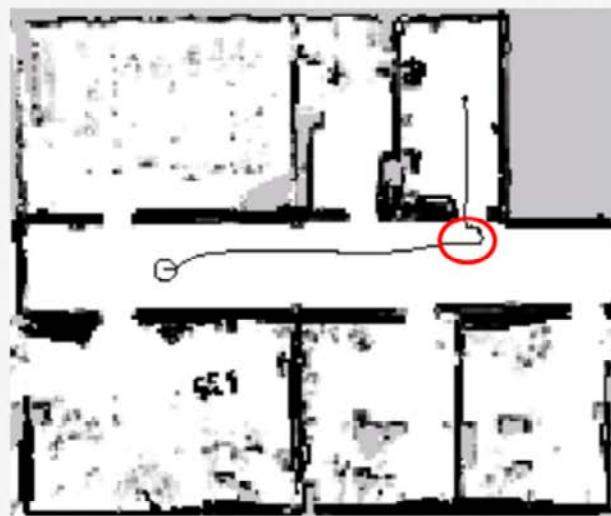


表 5.4 DWA 算法伪代码^④

```
BEGIN DWA(robotPose,robotGoal,robotModel)~  
    desiredV = calculateV(robotPose,robotGoal) //根据机器人的目标和位置  
    估算机器人的期望速度，例如：距离越近，所需速度越小，距离越大，所需速度越大。  
    laserscan = readScanner() //扫描空间。  
    allowable_v = generateWindow(robotV, robotModel) //得到线速度空间。  
    allowable_w = generateWindow(robotW, robotModel) //得到角速度空  
    间。  
    for each v in allowable_v //在线速度空间搜索。  
        for each w in allowable_w //在角速度空间搜索。  
            dist = find_dist(v,w,laserscan,robotModel) //计算空隙  
            find_dist(v,w)~  
            breakDist = calculateBreakingDistance(v) //计算停止距离。  
            if (dist > breakDist) //表示可以及时停止。  
                heading = hDiff(robotPose,goalPose, v,w) //机器人朝向评价。  
                clearance = (dist-breakDist)/(dmax - breakDist) //障  
                物评价(间隙 clearance)。  
                cost = costFunction(heading,clearance, abs(desired_v - v))  
//综合轨迹评价，根据公式(5-42)，将三部分的评价值归一化后相加。  
                if (cost > optimal)~  
                    best_v = v~  
                    best_w = w~  
                    optimal = cost~  
                set robot trajectory to best_v, best_w //确定最优轨迹的线速度、角速  
    度。  
END~
```



现实世界中的典型问题



机器人减速不够早，导致无法进入门口。

DWA法同样会发生局部最小值(local minima)问题，而且当机器人移动速度过快时，也会导致机器人错过转向，而无法进入狭窄通道的现象发生。

动态窗口法

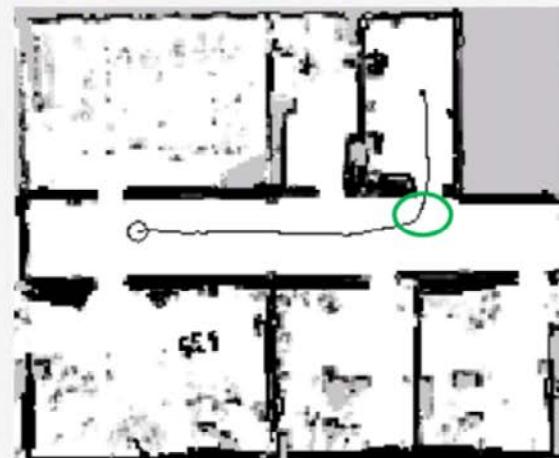
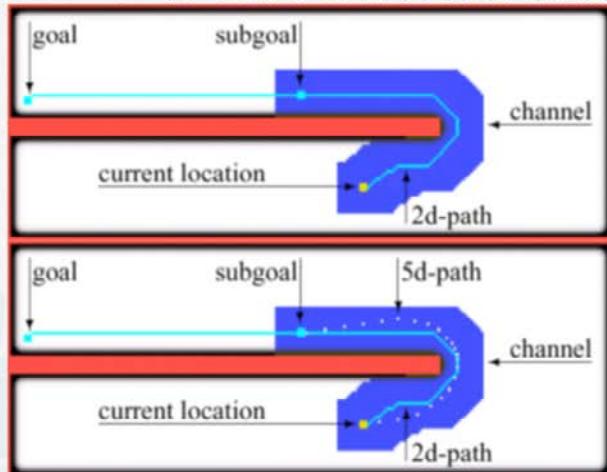
Dynamic Window Approach

DWA的替代算法：5D-Planning



在完整的 $\langle x, y, \theta, v, \omega \rangle$ -space位形空间中采用A*算法

- 问题：搜索空间太大，无法实时搜索
- 解决方案：将整个搜索空间限制为“channels”
- 在通道内的路径上选择一个subgoal
- 在“channel”5D空间中使用A*来查找一系列转向命令，以达到子目标





07

运动轨迹平滑

C-Space的离散化地图下进行组合规划方法以及基于采样的规划方法，可能导致机器人的运动变得急起急停，有时候必须对生成的运动轨迹进行处理以使其更平滑。

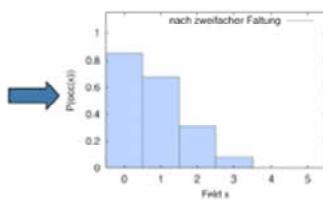
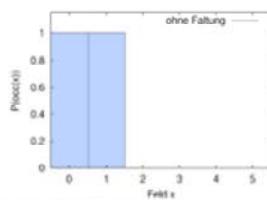
运动轨迹平滑

Nonlinear Optimization&Smoothing



例如，在栅格图中，采用卷积地图，让障碍物周围区域碰撞的概率大于零，类似于障碍物看起来比实际更大，那么在卷积地图中执行A*搜索算法的代价函数为：

$$f(n) = g(n)p_{occ}(n) + h(n) \quad p_{occ}(n) \text{ 为节点 } n \text{ 的概率}$$



1D example: cells before and after two convolution runs

使用卷积将地图M模糊化

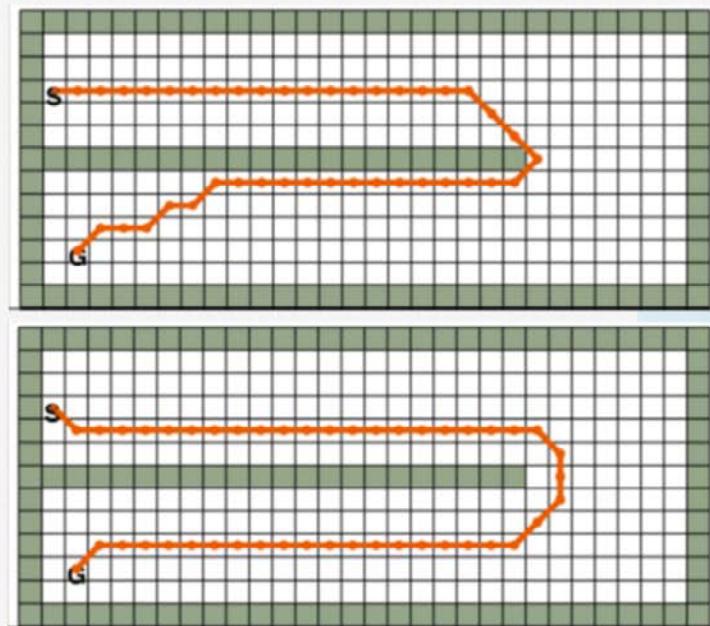
(例如用高斯核k) $(M * k)[i] = \sum_{j=-\infty}^{\infty} M[j] k[i-j]$



运动轨迹平滑

Nonlinear Optimization&Smoothing

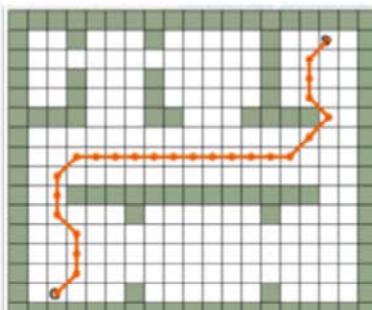
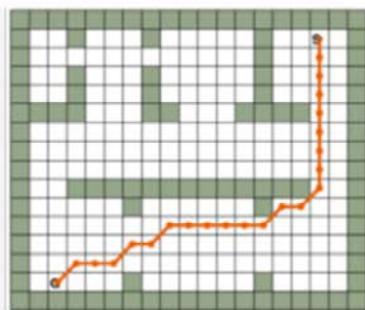
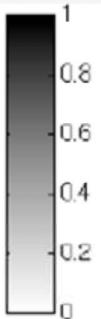
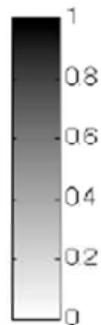
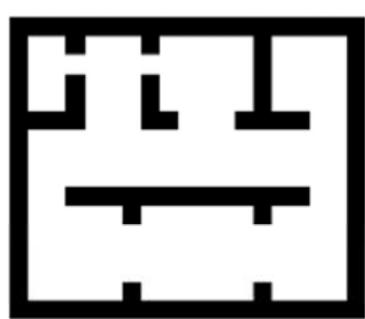
采用卷积地图对路径进行平滑



上式中， $p_{occ}(n)$ 为卷积地图中节点n的障碍物碰撞概率。在原始格栅地图中执行A*搜索算法找到的路径如图5.33(a)所示，采用卷积地图执行A*搜索算法找到的路径如图5.33 (b) 所示。

运动轨迹平滑

Nonlinear Optimization&Smoothing



08

全局规划与局部规划

我们是否需要为地图上的每个障碍物都重新规划整个路径?

如果机器人必须对不可预见的、快速移动的障碍物做出快速反应又该怎么办?

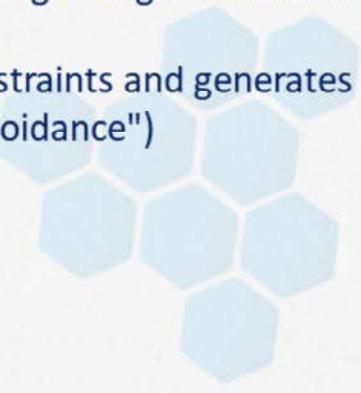
全局规划与局部规划

Global planner & Local planner



Integration into general motion planning?

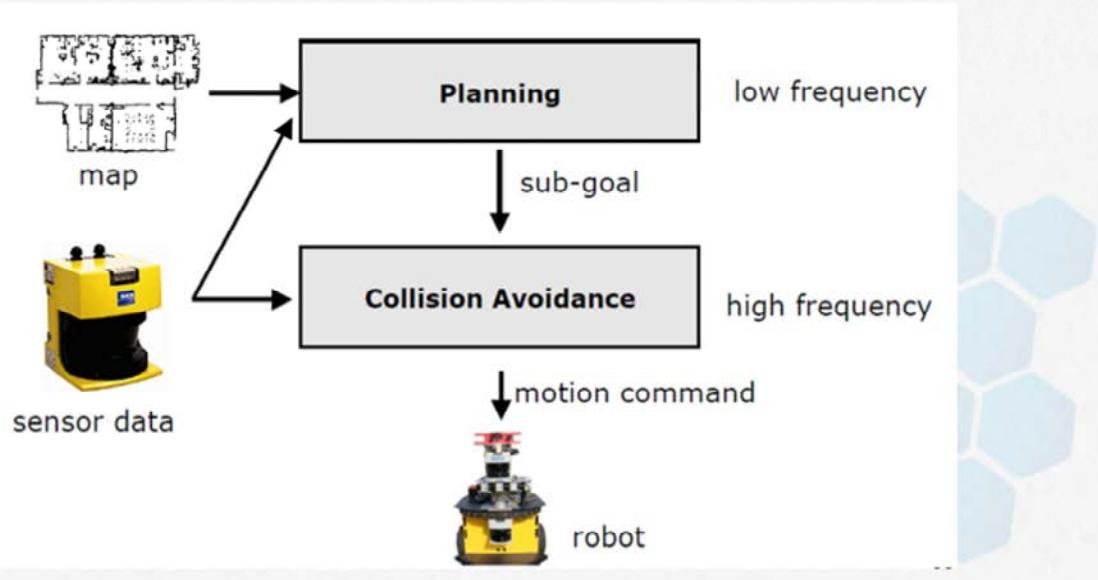
- It is common to subdivide the problem into a global and local planning task:
 - An approximate **global planner** computes paths ignoring the kinematic and dynamic vehicle constraints
 - An accurate **local planner** accounts for the constraints and generates (sets of) feasible local trajectories ("collision avoidance")
- What do we loose? What do we win?



全局规划与局部规划

Global planner & Local planner

Two-layered Architecture



全局规划与局部规划

Global planner & Local planner

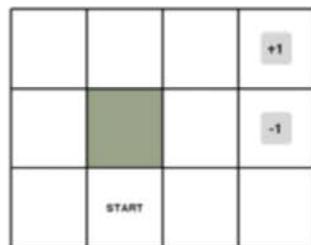


你有没有试着沿着一条路走的时候迷路过（比如沿百度地图之类）？

问题：路径执行固有的不确定性！如果机器人无法跟随，即使是最好的路径也毫无价值

原因：轨迹控制的不确定性、不完善的地图/动力学模型

你需要的不仅是**规划(planning)**，而可能还需要**决策(policy)**！



Markov Decision Process?
Dynamic Programming?

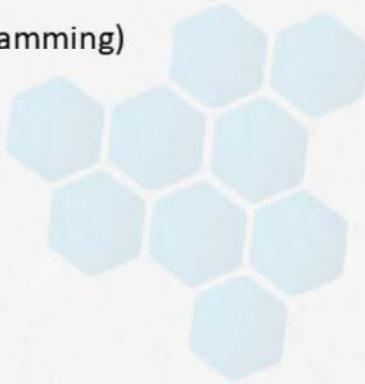
它的任务是达到+1标记的目标，避开标记为-1的单元

Markov Decision Process
动态规划(DP)



运动规划问题可表示为一般的非线性优化，带有等式约束或不等式约束，这样便可以用许多软件包来解决这些问题。可分为：

- 基于梯度的方法
 - 顺序二次规划 (SQP, Sequential Quadratic Programming)
- 非梯度方法
 - 模拟退火
 - Nelder-Mead 优化
 - 遗传算法 (genetic programming)

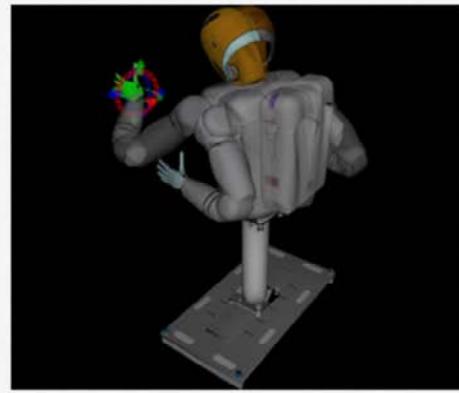


扩展：非线性优化

Nonlinear Optimization&Smoothing



For example: Track-IK



For a single IK solver request, TRAC-IK spawns two solvers, one running SQP-SS and one running KDL-RR. Once either finishes with a solution, both threads are immediately stopped, and the resulting solution is returned.

扩展阅读

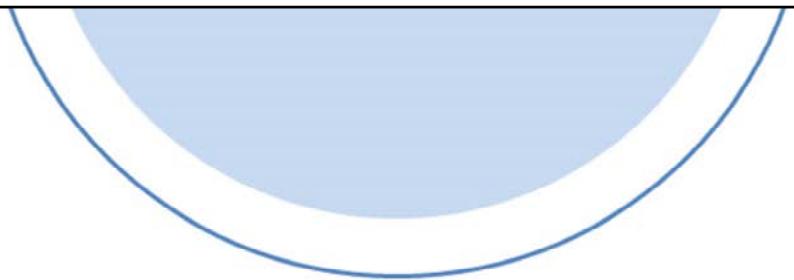


- 振动和运动规划的关系？轨迹生成算法能否解决哪些类型的振动？
- 碰撞检测算法（例如开源碰撞检测包FCL(Flexible Collision Library)）和运动规划的关系？
- 基于采样的规划方法使用样本来近似表示位形空间，均为非完整概率方法。这类方法在相对较短的时间内具有高效计算高维机器人路径的能力，这类PRM及RRT的改进算法在机器人领域得到了广泛的应用。然而随着位形空间维数的增加，会导致规划的路径质量变差，且规划时间过长，这类问题如何解决？
-

作业：

选择一种机器人，构建C-Space, C-free,C-obs，选择多种路径规划算法，进行编程仿真，分析算法优缺点，提出改进算法，并进行仿真对比实验，要求给出不同情况下的位形、速度、加速度以及Jerk图与分析。





THANK YOU

机器小学

Robotics