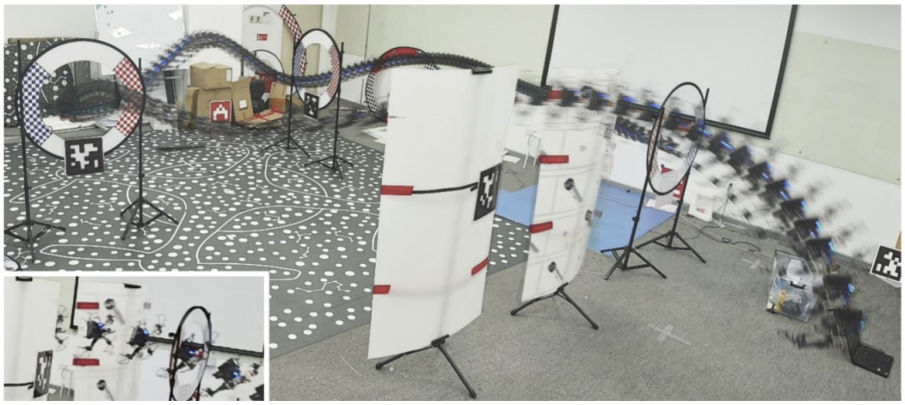
# 机器人运动规划讲义

## 1 引言

### 1.1 什么是运动规划？

在机器人领域，**运动规划（Motion Planning）**一般指：给定机器人的初始状态 $\boldsymbol{q}_s$ 和目标状态 $\boldsymbol{q}_g$，规划出适当的机器人的运动序列，使机器人能够从初始状态 $\boldsymbol{q}_s$ 运动到目标状态 $\boldsymbol{q}_g$。根据任务不同，运动过程会具有不同的约束，例如最基本的一个约束是在有障碍物的环境中，机器人需要无碰撞地从 $\boldsymbol{q}_s$ 运动到 $\boldsymbol{q}_g$。图1展示了两个需要运动规划的任务案例。



(a)

(b)

图 1: 机器人运动规划应用例子。(a) 无人机在障碍环境中飞行 [1]；(b) 人形机器人在障碍环境中进行操作 [2]。

最基础的运动规划（motion planning）问题可以简化为**路径规划（path planning）**问题，即生成一条从初始状态到目标状态的连续路径，而不考虑和时间的关系及机器人本身的

动力学，因此路径规划问题是一个几何问题（geometric problem），如图2。路径规划一般是完整运动规划的基础，受课时限制，本章主要介绍路径规划。

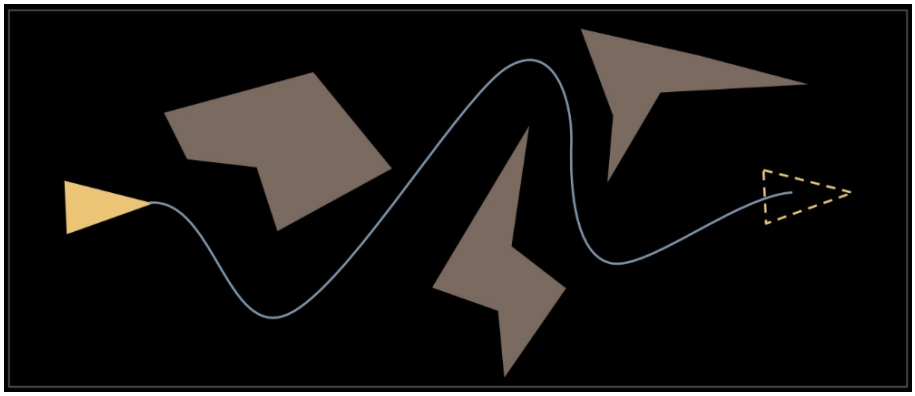


图 2: 路径规划问题的示例 [3]。

## 1.2 路径规划问题定义

首先，定义机器人的**构型空间（configuration space, C-space）**为 $\mathcal{C}$。一个路径规划问题需要有以下输入：

- 机器人的几何和运动学描述；
- 环境的几何描述；
- 机器人运动的约束；
- 机器人的初始状态；
- 机器人的目标状态（可以是不同的描述形式）；
- 对于最优规划问题，还需要有最优化的目标：例如使路径最短、消耗能量最少等。

一个**可行路径（feasible path）**指的是 C-space 中的一条以初始状态为起点、最终到达目标状态的不违反任何约束的连续曲线 $y(s) : [0, 1] \to \mathcal{C}$。如果制定了最优化的目标，那么得到的路径应该是**最优的**或者**近似最优的**（optimal path or near-optimal path）。

约束可以分很多种，如图3所示，其中包括：

- **局部约束**：每个路径点自身需要满足的约束，例如无碰撞约束等。
- **微分约束**：对于路径的导数（变化）的约束，例如喷气式飞机的转向约束。
- **全局约束**：对于路径整体的约束，例如无人机需要在电量耗尽前返回基地。

如果一个规划问题中既有局部约束，也有微分约束，则称其为 kinodynamic planning。需要注意的是，一些材料将局部约束称为 kinematic constraint，将微分约束称为 dynamic constraint，但这里的 kinematic 和 dynamic 与一般意义上的运动学和动力学的概念是不完全一致的。

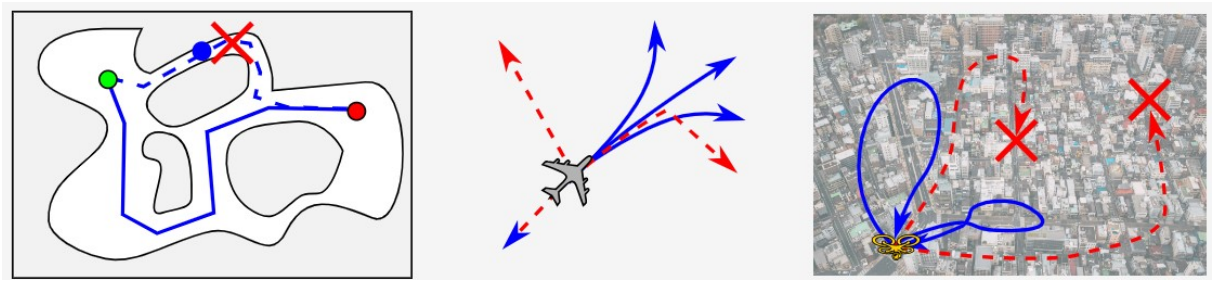


图 3: 三类约束的示意图 [4]。从左到右分别为局部约束、微分约束、全局约束。

### 1.3 规划算法的性能概念

一般来说，会使用以下几个概念来评价运动规划算法的性能：

- **完备性（completeness）**：完备性指对于可解的问题，算法能保证在有限时间内找到可行解；对于无解的问题，算法能够返回“无解”。
- **最优性（optimality）**：算法是能找到定义指标下的最优路径。
- **效率（efficiency）**：在实际问题中，算法需要多长时间来找到可行/最优路径?
- **泛化性（generality）**：算法能够求解哪类问题？不能求解哪类问题?

需要注意的是，对于不同的任务，以上指标的重要性和必要性是不同的，需要具体问题具体分析。

## 2 机器人构型空间

### 2.1 构型空间

我们定义机器人的构型空间（configuration space, C-space）为 $\mathcal{C}$。机器人的一个状态 $\boldsymbol{q}$ 对应 C-space 中的一个点，即 $\boldsymbol{q} \in \mathcal{C}$。另外，定义工作空间（workspace）为机器人所在的二维（$\mathbb{R}^2$）或三维空间（$\mathbb{R}^3$）。以下列举了几种机器人 C-space 的例子：

- 平面上的质点移动机器人：$\mathcal{C} = \mathbb{R}^2$，即二维坐标。
- 平面上的多边形移动机器人（考虑朝向）：$\mathcal{C} = SE(2) = \mathbb{R}^2 \times SO(2)$，即二维坐标 + 一维朝向，共三维。
- 三维空间中的多面体移动机器人（考虑姿态）：$\mathcal{C} = SE(3) = \mathbb{R}^3 \times SO(3)$，即二维坐标 + 三维朝向，共六维。
- 基座固定的具有 $n$ 个单自由度关节的机械臂：$\mathcal{C}$ 是 $SE(3)^n$ 的一个子集。因为每个关节只有一个自由度（即有五个约束），所以总的 $\mathcal{C}$ 的维度为 $6n - 5n = n$ 维。

其中 $SE(2), SE(3), SO(2), SO(3)$ 是李代数中的概念。一个生动的介绍可见 [5] 的第四章。这里可以简单理解为 $SO(2)$ 是二维平面上的朝向，$SE(2)$ 是二维平面上的位姿，$SO(3)$ 是三维空间中姿态，$SE(3)$ 是三维空间中的位姿。

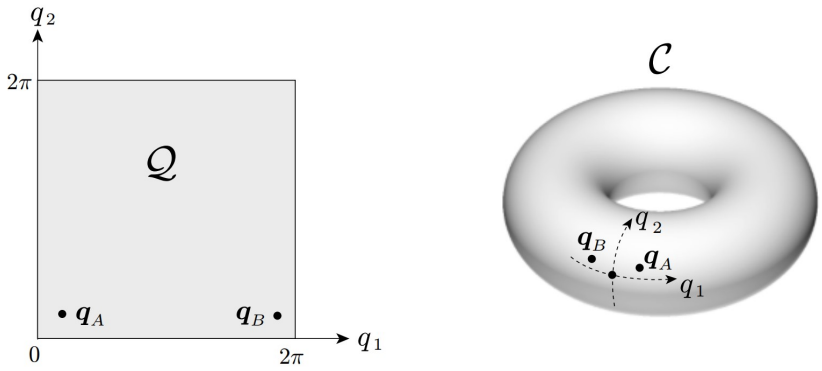



图 4: 平面二旋转关节机械臂的 C-space[6]。

需要注意的是，对于 $n$ 维的 $\mathcal{C}$，可以用一个向量 $\boldsymbol{q} \in \mathbb{R}^n$ 来表示一个状态，但是 $\mathcal{C}$ 并不一定等价于欧式空间 $\mathbb{R}^n$。这里举一个例子：考虑一个具有两个旋转关节（可无限旋转）的平面机械臂，其 C-space 看似可以表示为

$$\mathcal{Q} = \{\boldsymbol{q} = (q_1, q_2) : q_1 \in [0, 2\pi), q_2 \in [0, 2\pi)\} \tag{1}$$

如图4中左图所示。但是这种表示方式在拓扑上是不合理的。以图中 $\boldsymbol{q}_A$ 和 $\boldsymbol{q}_B$ 为例，它们所代表的机械臂状态在工作空间 $\mathcal{W}$ 中是非常接近的，但在 $\mathcal{Q}$ 中的距离较远。因此，$\mathcal{C}$ 更为合理的表示形式为图4中右图，其为 $\mathbb{R}^3$ 中的一个二维流形（manifold），构成一个环形（torus）形状。此处 $\mathcal{C}$ 正确的数学描述应为 $SO(2) \times SO(2)$。不过，对于关节最大旋转角度只有 $2\pi$ 的机械臂来说，$\mathcal{Q}$ 反而是正确的 C-space 描述形式。

## 2.2 障碍物

障碍物一般定义在工作空间 $\mathcal{W}$ 中。假设有 $p$ 个障碍物，记它们为 $\mathcal{O}_1, \cdots, \mathcal{O}_p$。假设它们的几何形状和位姿都已知。记机器人在状态为 $\boldsymbol{q}$ 时对应的在 $\mathcal{W}$ 中的几何形状为 $\mathcal{B}(\boldsymbol{q})$。在 $\mathcal{W}$ 中定义的障碍物 $\mathcal{O}_i$ 在构型空间 $\mathcal{C}$ 中的映像被称为 $\mathcal{C} - obstacle$，数学描述为：

$$\mathcal{CO}_i = \{\boldsymbol{q} \in \mathcal{C} : \mathcal{B}(\boldsymbol{q}) \cap \mathcal{O}_i \neq \emptyset\} \tag{2}$$

也就是说，$\mathcal{CO}_i$ 是 $\mathcal{C}$ 的一个子集，这个子集中的机器人状态 $q$ 会与障碍物 $\mathcal{O}_i$ 在工作空间中发生碰撞。考虑所有障碍物，则 $\mathcal{C}$-obstacles 空间被定义为

$$\mathcal{CO} = \bigcup_{i=1}^{p} \mathcal{CO}_i \tag{3}$$

其补集被称为自由构型空间（free configuration space），定义为：

$$\mathcal{C}_{\text{free}} = \mathcal{C} - \mathcal{CO} = \left\{ \boldsymbol{q} \in \mathcal{C} : \mathcal{B}(\boldsymbol{q}) \cap \left( \bigcup_{i=1}^{p} \mathcal{O}_i \right) = \emptyset \right\} \tag{4}$$

处于 $\mathcal{C}_{\text{free}}$ 中的机器人状态都是无碰撞的。无碰撞的路径规划即相当于在 $\mathcal{C}_{\text{free}}$ 中找到一条将初始状态 $\boldsymbol{q}_s$ 和目标状态 $\boldsymbol{q}_g$ 连接起来的（满足其它约束或目标的）路径。

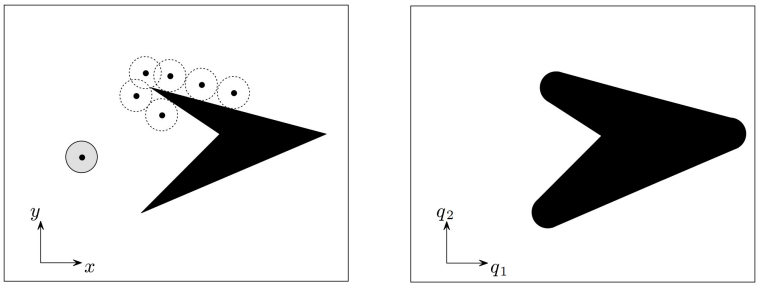



图 5: 一个平面圆形移动机器人的 $\mathcal{C}$-obstacles[6]。左图为 $\mathcal{W}$ 中的机器人和障碍物形态；右图为对应的 $\mathcal{C}$ 和 $\mathcal{CO}$。

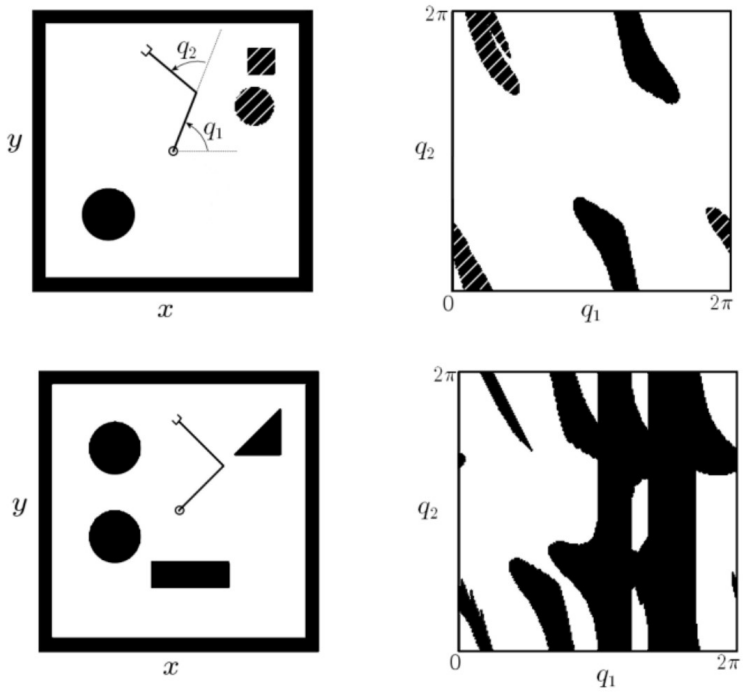



图 6: 一个平面二自由度机械臂的 $\mathcal{C}$-obstacles[6]。左图为 $\mathcal{W}$ 中的机器人和障碍物形态；右图为对应的 $\mathcal{C}$ 和 $\mathcal{CO}$。

图5展示了一个平面圆形移动机器人的 $\mathcal{C}$ 和 $\mathcal{CO}$，其构造是简单直观的。图6展示了一个平面二自由度机械臂的 $\mathcal{C}$ 和 $\mathcal{CO}$，可以看到，对于这种非常简单的机械臂，其 $\mathcal{C}$ 和 $\mathcal{CO}$ 已经是非常复杂、难以构造的了，甚至还有可能出现 $\mathcal{C}_{\text{free}}$ 不连通的情况（右下角图）。而对于三维空间中、更高自由度、几何形状更复杂的机械臂，其构型空间 $\mathcal{C}_{\text{free}}$ 几乎没有办法显示地计算得到。这使得对于高自由度机械臂的运动规划和对于移动机器人的运动规划具有不同的特点，常用的方法也有所不同。

对于机器人构型空间更详细的讲解，可以参看 [7] 的第四章。

# 3 构型空间中的路径规划算法

本节考虑最基本的运动规划问题：不考虑时间和机器人动力学，假设机器人运动只有局部约束，没有微分约束。此时规划问题简化为机器人构型空间中的路径规划问题，即在机器人的 $\mathcal{C}_{\text{free}}$ 找到一条连接 $\boldsymbol{q}_s$ 和 $\boldsymbol{q}_g$ 的曲线（路径）。

以下，我们以机器人的 $\mathcal{C}$ 是一个二维平面的情况为例，介绍几类经典的常用的路径规划算法。

## 3.1 基于搜索的算法

基于搜索（search-based）的路径规划算法是确定性算法，其基本思想为将 $\mathcal{C}_{\text{free}}$ 离散化为一个图（graph），之后在图上搜索得到路径。根据构建图的方式的不同，又可分为以下几类。

### 3.1.1 网格搜索

网格搜索（grid search）方法中，将 $\mathcal{C}_{\text{free}}$ 构建为图的方法非常直接，即将整个 $\mathcal{C}$ 网格化，如图7所示。其中，$\mathcal{C}$ 被离散化为二维的网格（图），每个顶点对应一个机器人状态 $\boldsymbol{q}$。假设 $\boldsymbol{q}_s$ 和 $\boldsymbol{q}_g$ 均处于网格顶点。将网格中被障碍物挡住的顶点和边从图中去掉，在剩下的图中进行图搜索，即可得到从 $\boldsymbol{q}_s$ 到 $\boldsymbol{q}_g$ 的路径。

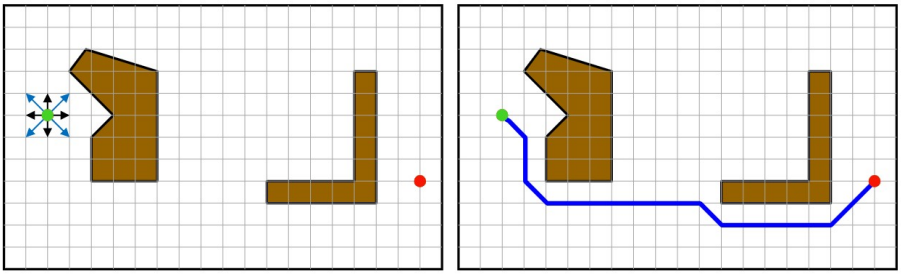


图 7: 基于网格搜索的路径规划方法示意 [4]。

网格化方法的一个缺点是网格的分辨率（resolution）对性能的影响较大。若分辨率太低（即网格边长太大），则通过狭窄通道的路径无法被找到。若分辨率太高，则图的规模会增大，使得规划时间开销增大。因此，基于网格搜索的概率方法不是完备的，但其是分辨率完备（resolution-complete）的，即：若解存在，在算法可以给定分辨率 $h$ 足够小的情况下，在有限时间内找到可行解。

### 3.1.2 可视图

可视图（visibility graph）是最早的路径规划算法之一。当在 $\mathcal{C}$ 中障碍物 $CO_i$ 均为多边形时，Visibility graph 方法是完备且最优的。其基本思想为：用 $\boldsymbol{q}_s$、$\boldsymbol{q}_g$ 及（多边形）$CO_i$ 的顶点共同构成图的顶点，将这些顶点间无碰撞的边全部添加进图中，该图被称为 visibility graph。之后，在该图中进行图搜索，即可得到（欧氏距离意义下的）最短路径。

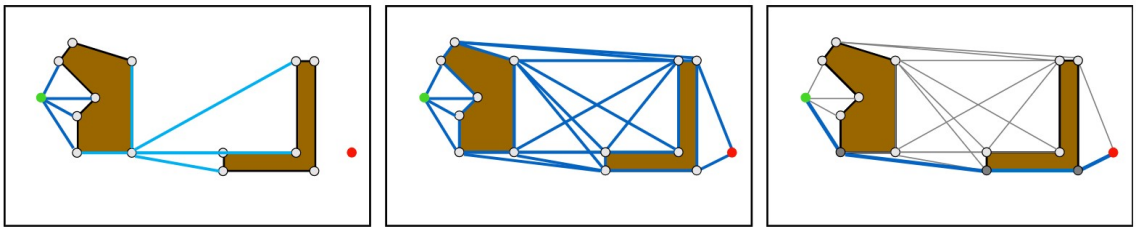


图 8: 基于可视图的路径规划方法示意 [4]。

### 3.1.3 图搜索算法

在构建好图后，使用图搜索算法即可得到可行路径。常用的图搜索方法包括广度优先算法、深度优先算法、Dijkstra 算法，A* 算法等。下文将对 Dijkstra 算法和 A* 算法进行介绍。

Dijkstra 算法是经典的图搜索算法，其使用了动态规划的思想，可以得到初始节点到图中任意节点的最短路径。已知图中每条边对应一个代价（距离是代价的一种），记图 $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$，所有节点组成 $\mathcal{V}$，所有边组成 $\mathcal{E}$。节点 $v_m$ 到 $\boldsymbol{v}_n$ 的边 $e_{mn}$ 的代价为 $c_{mn}$。算法在运行过程中，维护一个优先队列 $\mathcal{Q}$，并且对于每一个节点 $v_i$ 维护一个累积代价 $v_i.g$ 和一个前路径点 $v_m$.prior。阅读时请注意下标的含义：$v_s$ 和 $v_g$ 中的下表分表表示 start 和 goal。

**Algorithm 1** Dijkstra 算法

Initialize priority queue $\mathcal{Q} \leftarrow \{v_s\}$; $v_s.g \leftarrow 0$; $v_i.g \leftarrow \infty, v_i \in \mathcal{V} \setminus v_s$;
**while** $\mathcal{Q} \neq \emptyset$ **do**
    $v_n \leftarrow \mathcal{Q}$.pop(); // $v_n = \arg\min_{v_i} v_i.g$ $(v_i \in \mathcal{Q})$
    $v_n$.expanded = true;
    **if** $v_n = v_g$ **then**
        return true;
    **end if**
    **for** $v_m \in$ Neighbor$(v_n)$ **do**
        **if** $v_m$.expanded = false **and** $v_m.g > v_n.g + c_{nm}$ **then**
            **if** $v_m.g = \infty$ **then**
                $\mathcal{Q}$.push$(v_m)$;
            **end if**
            $v_m.g \leftarrow v_n.g + c_{nm}$;
            $v_m$.prior $\leftarrow v_n$;
        **end if**
    **end for**
**end while**
return false;

Dijkstra 算法的流程如算法1所示。当其运行完毕后，对于任意节点 $v_i$，只需递归式地读取其前节点 $v_i$.prior 直到 $v_s$，即可得到 $v_s$ 到 $v_i$ 的最短路径。在给定图中，该算法是完备

且最优的。

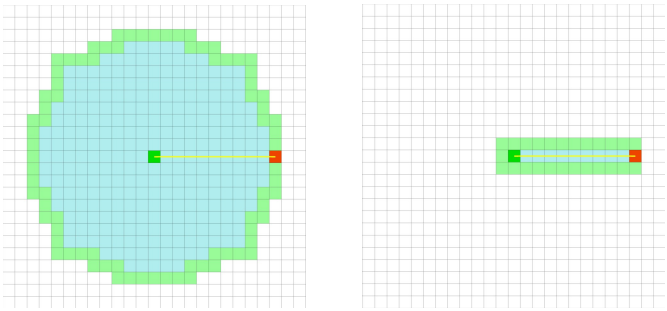
Dijkstra 算法平等地对待 $\mathcal{V} \setminus v_s$ 中的所有节点，没有利用 $v_g$ 的信息。当我们只需计算 $v_s$ 到 $v_g$ 的路径时，该算法的部分开销是多余的。A* 算法是 Dijkstra 算法的一个变种，其使用一种启发式的策略来更好地利用 $v_g$ 的信息来引导搜索。对于每个节点 $v_i$，除了从 $v_s$ 到 $v_i$ 的累积代价 $v_i.g$ 外，还维护一个从 $v_s$ 途径 $v_i$ 到达 $v_g$ 的最小估计代价 $v_i.f = v_i.g + h(v_i)$，其中 $h(v_i)$ 指从 $v_i$ 到 $v_g$ 的**估计**最小代价。A* 算法在搜索时以 $v_i.f$ 为指标，而 Dijkstra 算法中是以 $v_i.g$ 为指标。

**Algorithm 2** A* 算法

Initialize priority queue $\mathcal{Q} \leftarrow \{v_s\}$; $v_s.g \leftarrow 0$; $v_i.g \leftarrow \infty, v_i \in \mathcal{V} \setminus v_s$; Define $h(v_i)$;
**while** $\mathcal{Q} \neq \emptyset$ **do**
  $v_n \leftarrow \mathcal{Q}$.pop(); // $v_n = \arg\min_{v_i} v_i.f$ $(v_i \in \mathcal{Q})$
  $v_n$.expanded = true;
  **if** $v_n = v_g$ **then**
    return true;
  **end if**
  **for** $v_m \in$ Neighbor$(v_n)$ **do**
    **if** $v_m$.expanded = false **and** $v_m.g > v_n.g + c_{nm}$ **then**
      **if** $v_m.g = \infty$ **then**
        $\mathcal{Q}$.push$(v_m)$;
      **end if**
      $v_m.g \leftarrow v_n.g + c_{nm}$;
      $v_m$.prior $\leftarrow v_n$;
    **end if**
  **end for**
**end while**
return false;

A* 算法的流程如算法2所示，其中标红的部分为 A* 与 Dijkstra 算法不同的部分。为了保持算法的最优性，要求引入的 $h(\cdot)$ 是 admissible 的，即 $0 \leq h(v_i) \leq h^*(v_i), \forall v_i$，其中 $h^*(v_i)$ 是 $v_i$ 到 $v_g$ 真正的最小代价。使用 admissible 的启发函数可以保证算法找到最优路径，一个简单（可能过于简略）的证明可参考网址。举个例子，若机器人构型空间为欧式空间，且两节点间的边的代价就是两节点间的欧式距离，则 $h(v_i) = \|v_i.\boldsymbol{q} - v_g.\boldsymbol{q}\|_2$ 即为一个 admissible 的启发函数（其中 $v_i.\boldsymbol{q}$ 指节点 $v_i$ 对应的机器人状态向量），因为 $h^*(v_i)$ 不可能小于 $v_i$ 与 $v_g$ 间的欧氏距离。另外，选择的 $h(v_i)$ 越接近 $h^*(v_i)$（越紧），则 A* 算法的效率越高。当 $h(v_i)$ 不满足 admissible 时，不能保证找到最优解，但可能使算法更快地找到可行解。

图9展示了无障碍物的网格化后的二维欧式空间中的 Dijkstra 与 A* 算法的探索过程（从优先队列 $\mathcal{Q}$ 中取出的节点）对比示意图，其中可以很清晰地看出 Dijkstra 因为没有利用目标点的信息，所以其探索过程是向四周均匀展开的，而 A* 算法的探索过程则更有目标性。



(a) (b)

图 9: 无障碍物的二维欧式空间中的 Dijkstra 与 A* 算法的探索过程对比示意图 图片来源。绿色点为初始点，红色点为目标点。(a) Dijkstra 算法；(b)A* 算法（以欧式距离作为启发函数 $h(\cdot)$）。

### 3.1.4 小结

一般来说，基于搜索的算法适合于低维空间，例如二维或三维空间。另外，$\mathcal{C}_{\text{free}}$ 需要能够被显式地表示出来，以进行网格化或构建 visibility graph。因此，基于搜索的算法在移动机器人上应用较多，此时构型空间 $\mathcal{C}$ 与工作空间 $\mathcal{W}$ 相差不大，例如图5所示的例子。而对于高自由度的机械臂来说，基于搜索的算法就很难使用了，这就引出了下文将介绍的基于采样的路径规划算法。

除了本节介绍的网格搜索和 visibility graph 方法，还有一类经典的将低维的 $\mathcal{C}_{\text{free}}$ 离散化的方法，称为单元分解（cell decomposition）方法，对其详细的介绍可参考网址。

## 3.2 基于采样的算法

对于高自由度机械臂、多机器人等更高自由度的系统，其构型空间 $\mathcal{C}$ 的维度较高、且形态复杂。更重要的是，此时很难将障碍物在 $\mathcal{C}$ 中的映像 $\mathcal{CO}$ 显式地表示出来。本小结将介绍基于采样的路径规划算法，其不需要显式的 $\mathcal{CO}$ 的表达，而只需隐式的表达。这里隐式的表达指可行性查询（feasibility queries），其包括两个部分：一是检查一个机器人状态 $\boldsymbol{q}$ 是否是可行（feasible）的，即 $\boldsymbol{q}$ 是否属于 $\mathcal{CO}$；第二是两个状态之间的运动是否是可行的（visible）的，即 $\text{Path}(\boldsymbol{q}_1, \boldsymbol{q}_2) \bigcap \mathcal{CO}$ 是否为空（一般是两个状态之间的直线运动）。只要有能做出以上判断的方法，就可以实现基本的基于采样的路径规划算法了。在实际应用中，判断 $\text{Path}(\boldsymbol{q}_1, \boldsymbol{q}_2)$ 是否 visible 的一个简单方法是判断两状态间的以一定距离分布的插值点是否 feasible，从而将其转换为单点状态检测问题。

最常用的两类基于采样的算法是**概率路标图（probabilistic roadmaps, PRMs）**和**快速探索随机树（rapidly-exploring random trees, RRTs）**方法。

### 3.2.1 PRM 算法

PRM 算法的基本思想是使用随机采样的方式，建立 $\mathcal{C}_{\text{free}}$ 的近似路标图。与上一节中介绍的网格搜索等方法一样，其同样使用一个图来近似 $\mathcal{C}_{\text{free}}$，之后再使用图搜索方法得到可行路径。PRM 算法的流程如下（又如图10所示）：

1. 在 $\mathcal{C}$ 中随机采样 $N$ 个点；
2. 对采样得到的点进行 feasibility 检测；用其中 feasible 的点以及初始状态和目标状态建立一个路标图（roadmap），这些点即为 roadmap 的顶点（又称 milestones）；
3. 对每个顶点，选择它与它附近的顶点进行 visibility 检测；将其中 visible 的边添加进图中。
4. 在图上使用图搜索方法（例如 Dijkstra 或 A* 算法）得到可行路径。

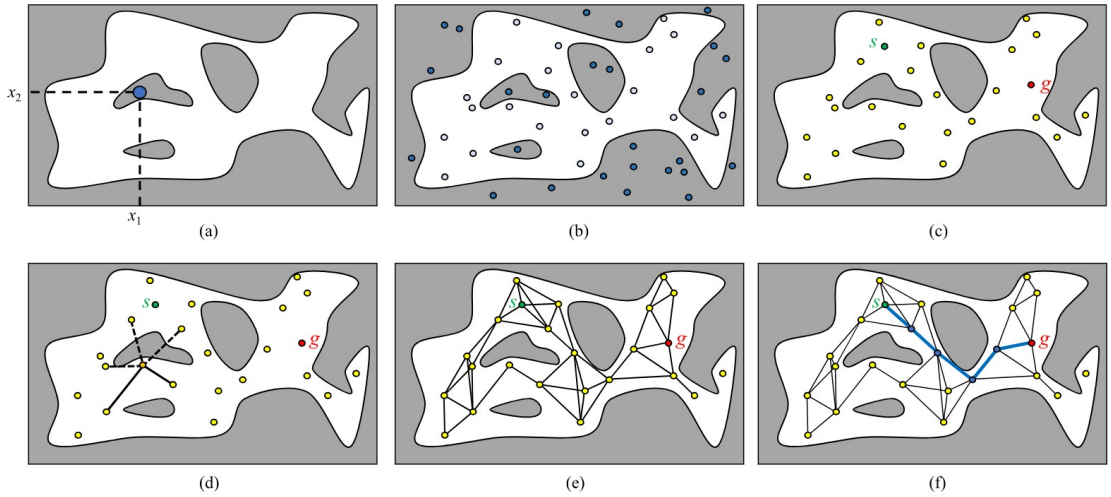



图 10: PRM 算法流程示意图 [4]。(a) 随机采样；(b)feasiblity 检测；(c) 保留 feasible 的点，添加初始和目标点；(d)visibility 检测；(d) 形成最终的路标图；(e) 进行图搜索。

可以看到，PRM 算法的思想非常简单直观，但其中有几个部分对算法实际效率影响很大。第一，如何进行随机采样？采样的分布是对 PRM 算法影响最大的因素；第二，如何找到一个路标点（顶点）附近的路标点？第一个问题引出了众多修改 PRM 中的采样分布以提升规划效率的工作（更多介绍可参考网址）。第二个问题相当于最近邻搜索问题，涉及到 k-d tree, FLANN[8] 等最近邻搜索算法。可以看到，在 PRM 算法中，建图的过程是主要的时间开销来源，当建好图后，可以很高效地得到图中任意两个节点之间的可行路径，所以其属于 multi-query 方法。

PRM 算法是一种概率性算法，其不是完备的，但是**概率完备的（probabilistically complete）**。概率完备指的是：如果问题有解，则随着采样点的增加，算法找到一个可行路径的概率逐渐趋近于 1。另外，在问题层面上，算法收敛的速度和 $\mathcal{C}_{\text{free}}$ 的 visibility 有关。更详细的分析可参考网址。

### 3.2.2 RRT 算法

PRM 算法属于 multi-query 算法，其在建图阶段并未考虑初始状态和目标状态。而在很多应用中，我们只需要得到初始状态到目标状态的路径即可，即 single-query。RRT 算法就属于 single-query 算法。其基本思想是：从初始状态开始 $\boldsymbol{q}_s$ 扩展一棵树 $\mathcal{T}$，逐渐（随机）探索整个空间，直到扩展到达目标节点 $\boldsymbol{q}_g$。一个基本的 RRT 算法的流程如算法3所示。

**Algorithm 3** RRT 算法

$\mathcal{T}$.init($\boldsymbol{q}_s$);
**for** $i = 1$ **to** max_iter **do**
  $\boldsymbol{q}_{\text{rand}} \leftarrow$ RandomConfig(); // 随机采样
  $\boldsymbol{q}_{\text{near}} \leftarrow$ Near($\mathcal{T}, \boldsymbol{q}_{\text{rand}}$); // 最近邻搜索
  $\boldsymbol{q}_{\text{new}} =$ Steer($\boldsymbol{q}_{\text{near}}, \boldsymbol{q}_{\text{rand}}$, step_size); // 扩展新节点
  **if** Feasible($\boldsymbol{q}_{\text{new}}$) **and** Visible($\boldsymbol{q}_{\text{near}}, \boldsymbol{q}_{\text{new}}$) **then**
    $\mathcal{T}$.AddVertex($\boldsymbol{q}_{\text{new}}$);
    $\mathcal{T}$.AddEdge($\boldsymbol{q}_{\text{near}}, \boldsymbol{q}_{\text{new}}$);
  **end if**
  **if** $\boldsymbol{q}_{\text{new}} = \boldsymbol{q}_g$ **then**
    return PathExtract($\mathcal{T}$);
  **end if**
**end for**

RRT 算法得到的是一棵树而不是一个图，所以每个节点都只有一个父节点。当算法运行成功后，从 $\boldsymbol{q}_g$ 开始，递归地提取父节点，直到 $\boldsymbol{q}_s$，即可提取出 $\boldsymbol{q}_s$ 到 $\boldsymbol{q}_g$ 的路径。图11展示了 RRT 中的 Steer() 步骤。

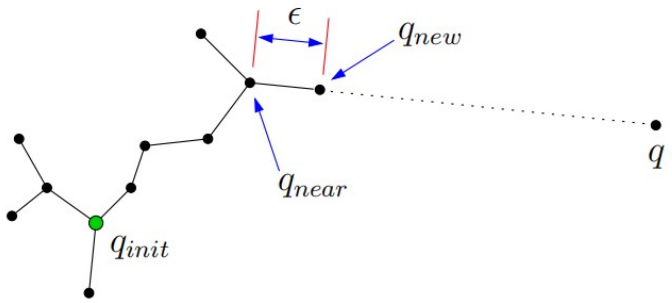



图 11: RRT 中的 Steer() 步骤示意图 [9]。Steer() 中，从当前 $\mathcal{T}$ 中离采样点 $\boldsymbol{q}$ 最近的 $\boldsymbol{q}_{\text{near}}$ 出发，向 $\boldsymbol{q}$ 的方向扩展 $\epsilon$ 步长，得到 $\boldsymbol{q}_{\text{new}}$。

如果在引导树扩展的节点一直是随机采样得到的，那么树就是向四周均匀扩展的，仍没有利用到 $\boldsymbol{q}_g$ 的信息，RRT 算法中常用 goal bias 的策略，即以一定概率 $p_{\text{goalbias}}$ 选择 $\boldsymbol{q}_g$ 替代 $\boldsymbol{q}_{\text{rand}}$ 来引导树扩展。根据经验，一般 $p_{\text{goalbias}}$ 选择 0.05 至 0.1 是比较合适的。

与 PRM 算法不同，RRT 算法并不会将随机采样得到的状态用在路径中，而只是用随机采样的状态来引导树扩展的方向。RRT 算法也是概率完备的，而且其算法实现简便，非常易于添加其它约束，所以应用十分广泛。不过 RRT 算法中很多细节会对性能产生较大影响，比如合适的采样策略、合适的距离度量、合适的扩展步长等。

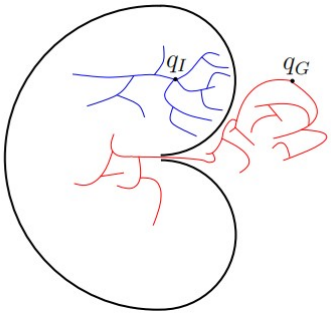



图 12: 双向 RRT 和 Bug trap 问题示意图 [7]（此处的 $\boldsymbol{q}_I$ 对应本文中的 $\boldsymbol{q}_s$，$\boldsymbol{q}_G$ 对应本文中的 $\boldsymbol{q}_g$）。

RRT 有非常多的变种和实现方式。双向 RRT（Bi-directional RRT，又名 RRT-Connect）是其中非常经典的一种，不同于基础 RRT 从 $\boldsymbol{q}_s$ 长出一棵树的操作，其从 $\boldsymbol{q}_s$ 和 $\boldsymbol{q}_g$ 同时长出两棵树，相向探索，从而提高扩展效率。Bi-directional RRT 算法的伪代码可见 [9]。图12展示了一个 bug trap 问题的例子，可以直观看出，在此种情况下双向 RRT 比单向 RRT 更容易找到可行路径。

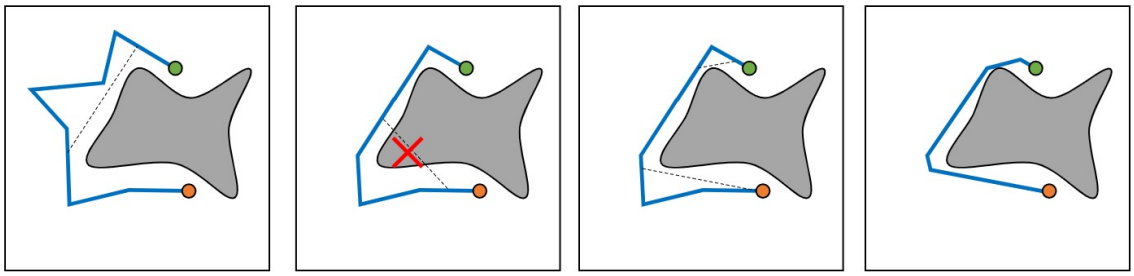


图 13: Shortcutting 方法示意图 [4]。

RRT 和 PRM 算法可以找到可行路径，路径质量往往较差（长度较长、路径较为曲折）。Shortcutting 算法是一种简单且高效的对找到的可行路径进行局部优化的方法。如图13所示，其基本思想为：每次迭代中，从当前的可行路径中随机选择两个点 $\boldsymbol{q}_i$ 和 $\boldsymbol{q}_j$，若 $\text{Visible}(\boldsymbol{q}_i, \boldsymbol{q}_j) = \text{true}$，将两点之间的原路径替换为直线，得到新的可行路径。如此进行很多次迭代后，即可有效缩短路径。需要注意的是，shortcutting 方法是一个局部优化算法，其得到的解的好坏取决于初值，也就是找到的初始可行路径。不过其非常高效，且优化得到的路径在最优性要求不是特别高的任务中一般是足够好用的，所以其应用十分广泛。

基础的 PRM 算法和 RRT 算法都是可行路径搜索算法，而不是最优路径搜索算法。概率性路径规划算法不具有严格最优性，但可能具有**渐进最优性（asymptotically optimal）**。

渐进最优性指：随着迭代次数增加，算法得到的路径的最小代价 $c(n)$ 逐渐逼近实际最优路径的代价 $c^*$；而对于概率性算法，渐进最优性指：算法得到的路径的最小代价 $c(n)$ 逐渐逼近实际最优路径的代价 $c^*$ 的概率为 1，即 $Pr\left(\lim_{n\to\infty} c(n) - c^* \neq 0\right) = 0$。PRM 的变种 PRM* 及 RRT 的变种 RRT* 在原始算法的基础上，考虑了路径的最优性，实现了渐进最优的路径规划。关于 PRM* 和 RRT* 的具体内容可参考网址和 [10]。此外，Informed RRT*[11] 是 RRT* 的一个重要变种，其通过去除冗余的探索空间，显著提高了 RRT* 的效率。需要注意的是，RRT*/PRM* 的计算开销显著高于 RRT/PRM，特别是在高自由度机械臂上更为明显，因此在一般的机械臂的路径规划中，RRT/PRM + shortcutting 的方式更为常用。

### 3.2.3 小结

基于采样的方法既可以用于低维空间，也可以用于高维空间，因此在无人车、无人机、高自由度机械臂等领域均应用十分广泛。另外，此类方法还可以非常方便地加入其它局部约束，例如实现末端位姿约束的机械臂规划；还可以加入微分约束来实现 kinodynamic planning。实际上，RRT 最早就是为了解决 kinodynamic planning 而提出的。虽然基于采样的方法实现简便，但其性能往往受到很多细节的影响，实际应用时往往需要调整超参数来达到最优的性能。另外，虽然有 RRT* 之类的最优规划方法，但基于采样的方法主要还是用于找到可行路径，而非严格最优路径。

## 3.3 非完备方法

前文提到的方法都具有某些意义下的完备性，除此以外，也有一些不具有完备性的方法，即不能理论保证在有解的情况下一定能找到可行路径，但这些非完备方法在某些场景下有自己的优势，应用也很广泛。

### 3.3.1 势场法

势场法（potential fields）是一种在线局部规划（online local planning）方法。从某种意义上，它也可以被认为是一种局部控制方法。势场法不是在机器人运动之前就规划出从 $\boldsymbol{q}_s$ 到 $\boldsymbol{q}_g$ 的完整路径，而是在运动的过程中，不断根据周围局部障碍物的情况，调整机器人当前的运动情况。因此，势场法的一个优点就是事先不需要知道全局的障碍物信息，而是在运动中在线地对环境变化做出反应。

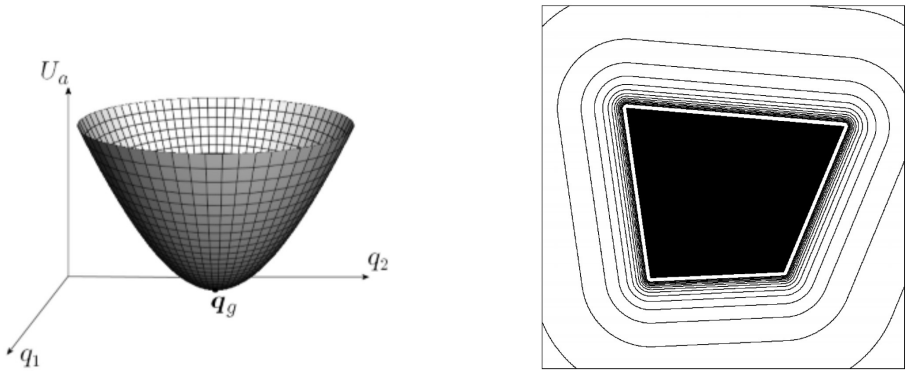
势场法的基本思想是将整个 $\mathcal{C}$ 构建为一个人工势场，理想情况下，$\boldsymbol{q}_g$ 处的势能最低，有障碍物（或其它违反约束）的状态处的势能都非常高，这样机器人一直沿势场的负梯度方向运动，就可以避开障碍物，到达 $\boldsymbol{q}_g$。

势场的设计方法如下。首先需要对 $\boldsymbol{q}_g$ 设计一个吸引势场（attractive potential）。对于任意状态 $\boldsymbol{q}$，设 $\boldsymbol{e}(q) = \boldsymbol{q}_g - \boldsymbol{q}$，则吸引势场可以被设计为（如图14(a)）：

$$U_a(\boldsymbol{q}) = \frac{1}{2} k_a \boldsymbol{e}^T(\boldsymbol{q}) \boldsymbol{e}(\boldsymbol{q}) = \frac{1}{2} k_a \|\boldsymbol{e}(\boldsymbol{q})\|^2 \tag{5}$$

其中 $k_a > 0$。易得负梯度方向为：

$$-\nabla U_a(\boldsymbol{q}) = k_a \boldsymbol{e}(\boldsymbol{q}) \tag{6}$$




(a) (b)

图 14: 人工势场示意图 [6]。(a) 吸引势场；(b) 障碍物附近的排斥势场。

另外，需要对障碍物设计排斥势场 (repulsive potential)。首先假设 $\mathcal{C}$-obstacles 可以被分解为数个凸的 $\mathcal{CO}_i, i = 1, \cdots, p$。对于每一个凸的 $\mathcal{CO}_i$，可以设计一个排斥势场 (如图14(b))：

$$U_{r,i}(\boldsymbol{q}) = \begin{cases} \frac{k_{r,i}}{2}\left(\frac{1}{\eta_i(\boldsymbol{q})} - \frac{1}{\eta_{0,i}}\right)^2 & \text{if } \eta_i(\boldsymbol{q}) \le \eta_{0,i} \\ 0 & \text{if } \eta_i(\boldsymbol{q}) > \eta_{0,i} \end{cases} \tag{7}$$

其中 $k_{r,i} > 0$，$\eta_i(\boldsymbol{q}) = \min_{q' \in \mathcal{CO}_i} \|\boldsymbol{q} - \boldsymbol{q}'\|$ 为 $\boldsymbol{q}$ 和 $\mathcal{CO}_i$ 间的最短距离，$\eta_{0,i}$ 表示了排斥势场的影响范围。可以看出，排斥势场在离 $\mathcal{CO}_i$ 的距离超过或等于 $\eta_{0,i}$ 时为零，离 $\mathcal{CO}_i$ 越近势场越大，当距离为零时趋近于无穷。可得排斥势场的负梯度方向为：

$$-\nabla U_{r,i}(\boldsymbol{q}) = \begin{cases} \frac{k_{r,i}}{\eta_i^2(\boldsymbol{q})}\left(\frac{1}{\eta_i(\boldsymbol{q})} - \frac{1}{\eta_{0,i}}\right)\nabla \eta_i(\boldsymbol{q}) & \text{if } \eta_i(\boldsymbol{q}) \le \eta_{0,i} \\ 0 & \text{if } \eta_i(\boldsymbol{q}) > \eta_{0,i} \end{cases} \tag{8}$$

所有障碍物的排斥势场基于每个障碍物的排斥势场之和：

$$U_r(\boldsymbol{q}) = \sum_{i=1}^{p} U_{r,i}(\boldsymbol{q}) \tag{9}$$

整个 $\mathcal{C}$ 上施加的总势场就是吸引势场和排斥势场的和：

$$U_t(\boldsymbol{q}) = U_a(\boldsymbol{q}) + U_r(\boldsymbol{q}) \tag{10}$$

图15展示了一个障碍环境中的总势场示意图。可以看到，$\boldsymbol{q}_g$ 是总势场的全局最优解，但势场中还存在很多局部最优解。在环境复杂时，机器人会很容易陷入局部最优解，这也是势场法最大的缺陷；另外，对于狭窄环境，很难设计合适的排斥势场。不过，势场法的优势在于只需局部信息，求解速度快，因此其适合用于在线规划，例如机械臂在运动过程中的实时动态（简单）障碍物躲避。

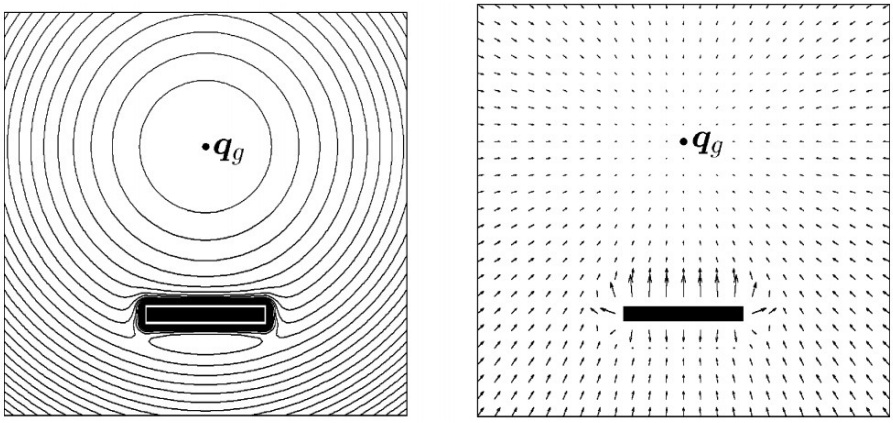



图 15: 一个障碍环境中的总势场示意图 [6]。左图：势场（等高线）；右图：势场的负梯度方向。

### 3.3.2 轨迹优化

轨迹优化（trajectory optimization）将规划问题描述成优化问题，其中路径就是优化变量，再使用优化求解器进行数值求解（某些简单问题可直接推导出解析解）。从控制的角度出发的最优控制问题在某种意义上可以认为是轨迹优化的一种。除了从零开始优化外，在实际应用中轨迹规划也常用于后端，即先用基于搜索或采样的方法得到一个可行的路径，再以路径作为变量初值，使用轨迹优化得到更好的满足更多约束的路径。一般在轨迹优化中需要考虑机器人全部的约束，包括机器人的动力学等。和一般的机器人学入门材料一样，本课程不对轨迹优化相关内容进行展开，感兴趣的同学可以阅读以下材料：

1. Russ Tedrake, "Chapter 10 Trajectory Optimization", Underactuated Robotics, MIT. 网址

2. Kelly, Matthew. "An introduction to trajectory optimization: How to do your own direct collocation." SIAM Review 59.4 (2017): 849-904. 网址

3. Malyuta, Danylo, et al. "Convex optimization for trajectory generation." 网址

# 4 对于多自由度机械臂的路径规划

面向不同种类的机器人的路径规划问题和方法均有各自的特点。本节将简要介绍面向多自由度机械臂的路径规划问题中的一些特殊的需求和方法。

首先，机械臂路径规划具有以下特点：1) $\mathcal{C}$ 维度高；2) $\mathcal{C}_{\text{free}}$ 难以描述；3) 需要考虑构型空间（configuration space）和任务空间（task space）：在机械臂上，构型空间一般指关节空间（joint-space），任务空间指笛卡尔空间（Cartesian space）。因此，基于搜索的方法往往无法得到使用；人工势场法和基于优化的方法往往只能用来处理一些环境相对简单的任务；对于复杂任务，基于采样的方法是应用最为广泛的，因此本节将重点介绍基于采样的方法。另外，对于不同的机械臂任务需求，也需要对规划方法进行一些特殊的设计。

## 4.1 关节空间目标的路径规划

如果机械臂路径规划的目标是定义在关节空间，即指定了 $\boldsymbol{q}_g$，则问题和上一节中讨论的构型空间中的路径规划是一致的。常用的方法是 PRM、RRT 及其多种变种；一般来说，Bidirectional RRT 的时间开销会比普通 RRT 要小很多。

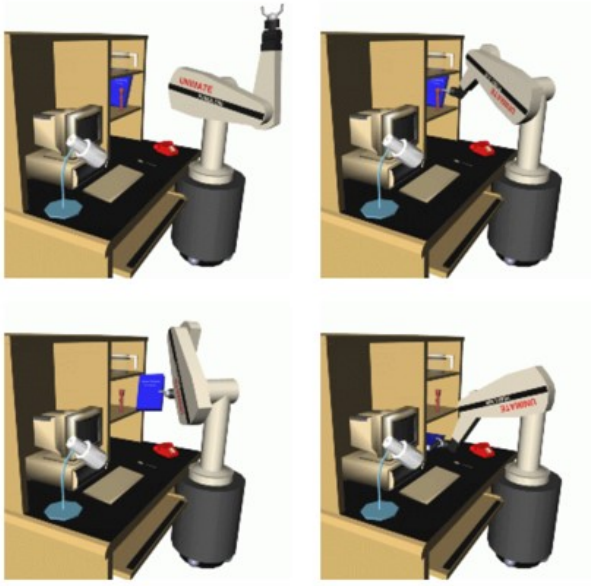


图 16: 基于 Bidirectional RRT 的机械臂路径规划及操作 [9]。

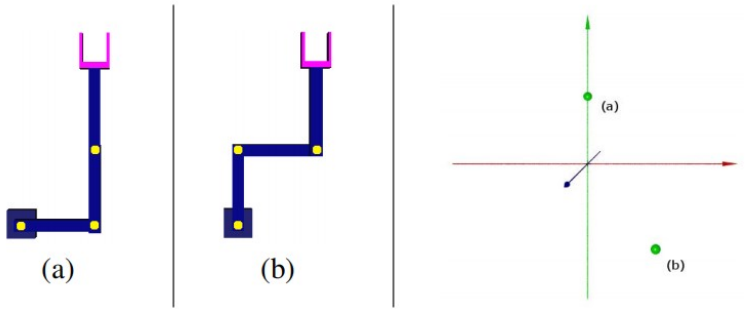
需要注意的是，在基于采样的方法中，距离度量的选择影响很大。定义 $n$ 自由度的关节空间为 $\mathbb{R}^n$（不考虑约束），一个最简单的方法是使用 $\mathbb{R}^n$ 中的欧氏距离作为度量，但这对于机械臂来说可能是不合适的，因为机械臂靠近基座的关节转 30 度和靠近末端的关节转 30 度对于机器臂整体运动的影响是差别很大的；一个更合适的方法是对不同关节加权，更靠近末端的关节赋予更小的权重。

另外，机械臂几何模型复杂，所以对其的碰撞检测往往也较为复杂，碰撞检测的时间开销在规划的总开销中的占比是很大的。碰撞检测是一个经典的但还在不断发展的领域，其中 FCL（Flexible Collision Library）是一个常用且好用的碰撞检测库（网址）。

## 4.2 任务空间目标的路径规划

有时，路径规划任务只制定了任务空间的目标，即只指定了机械臂末端的目标位姿。这在机械臂操作的应用中是非常常见的。一种最简单的方法是，先对于目标位姿使用逆运动学求解出对应的一个关节角度，然后再在关节空间中进行路径规划。但是，逆运动学的解不是唯一的，那么应该选择哪一个解呢？前文说过，因为障碍物、机械臂自碰撞等因素，机械臂的 $\mathcal{C}$ 是很复杂的，而且常常是不连通的。图17展示了一个例子，其中图17(a) 展示了对于一个三自由度平面机械臂，对于某一末端位姿存在两个不同的逆运动学解，这两个解分别对应 $\mathcal{C}$ 中的 a 点和 b 点；若环境中存在图17(b) 左图所示的障碍物，其对应的 $\mathcal{C}$-obstacles 如

图17(b) 右图；此时，受障碍物的影响，a 点和 b 点之间是无法连通的，如图17(c) 所示；因此，选择不合适的逆运动学会导致路径规划失败。




(a)


(b) (c)

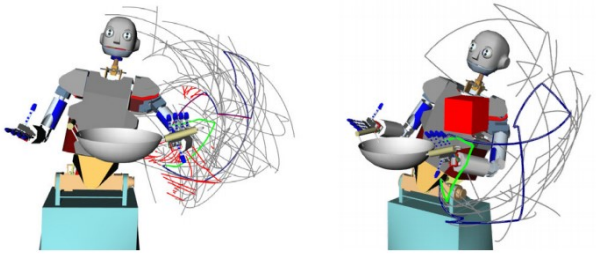
图 17: 选择不合适的逆运动学解作为路径规划目标的影响 [12]。(a) 对于一个三自由度平面机械臂，对于某一末端位姿存在两个不同的逆运动学解，分别对应 $\mathcal{C}$ 中的 a 点和 b 点；(b) 环境中存在障碍物，其对应的 $\mathcal{C}$-obstacles 会将 $\mathcal{C}_{\text{free}}$ 分隔为两个不连通的区域；(c)a 点和 b 点两个逆运动学解之间是无法连通的。

对于目标指定在任务空间中的路径规划任务，一类方法是在 RRT 树扩展时，基于机械臂雅克比矩阵引导机械臂末端向目标位姿运动，其基本思想如下：

- 在每次 RRT 迭代中，以概率 $p$ 按基本 RRT 方式扩展，此时不使用任何 goal bias。该步骤保证树的全局探索性和算法的概率完备性。
- 否则，则向目标位姿进行扩展：选择树中离目标位姿 $\boldsymbol{x}_g$ 最近的节点 $\boldsymbol{q}_{\text{near}}$（距离定义在 $SE(3)$ 中），基于机械臂雅克比矩阵的转置（或伪逆），向目标位姿进行扩展，即 $\boldsymbol{q}_{\text{new}} = \boldsymbol{q}_{\text{near}} + \boldsymbol{J}^T \Delta \boldsymbol{x}$，其中 $\Delta \boldsymbol{x}$ 为 $\boldsymbol{q}_{\text{new}}$ 对应的末端位姿与目标位姿的误差，$\boldsymbol{J}$ 为 $\boldsymbol{q}_{near}$ 处的 Jacobian。该步骤类似于一个任务空间的局部最优控制器，使机械臂局部最优地向目标位姿扩展。

更详细的算法细节可以参考 [13]。其特点在于不需要显式的逆运动学求解。

另一类方法则采用随机采样逆运动学解的方法。在每次 Bidirectional RRT 迭代中，其以一定概率随机采样目标位姿对应的一个逆运动学解，并将其添加进 $\mathcal{T}_g$（从目标节点长出的树）中。具体算法细节可参考 [14]。这样做的好处是可以使用 Bidirectional RRT，极大提高规划的速度，但缺点是需要非常高效的逆运动学求解器。



(a)

| | Without Obstacle Avg Runtime | With Obstacle Avg Runtime |
|---|---|---|
| $J^+$-RRT | 2 032 ms | 18 390 ms |
| IK-RRT | 140 ms | 480 ms |

(b)

图 18: 两类任务空间目标的路径规划方法的对比 [14]。其中 $J^+$-RRT 属于第一类不需要显式逆运动学求解的方法，IKRRT 属于第二类随机采样逆运动学解的方法。(a) 对于同一任务的 RRT 树扩展过程对比；(b) 规划时间开销对比。

第一类方法更适合简单的环境或机械臂逆运动学求解困难的情况，因为其不需要显式的逆运动学求解，而且其扩展方式是局部最优的。第二类方法更适合于环境复杂、但可以高效机械臂逆运动学求解的情况，因为其可以使用 Bidirectional RRT，从而极大提高复杂环境中的规划效率。图18展示了在某任务中两种方法的对比（来源于提出第二类方法的论文）。

## 4.3 位姿约束的路径规划

在很多机械臂操作任务中，机械臂末端的位姿是受到约束的，此时就不能仅在机械臂的构型空间中考虑路径规划问题，还需要考虑其在任务空间的位姿。图19展示了一些位姿约束的操作任务。


(a) (b) (c) (d)

图 19: 末端位姿受约束的任务案例 [15]。(a) 拉抽屉任务中，抽屉只允许在 x 方向上平动；(b) 开关门任务中，门只允许绕 z 轴转动；(c) 提水桶任务中，水桶不允许绕 x 轴和 y 轴的转动；(d) 对于闭链机构，在图示关节上只允许绕 z 轴的转动。

对于末端位姿受约束的机械臂，其构型空间 $\mathcal{C}$ 会变得更加复杂。如图20所示，此时的 $\mathcal{C}$ 是不受约束时的 $\mathcal{C}$ 的一个子空间，而根据约束维度的不同，受约束的 $\mathcal{C}$ 的维度可能会低于不受约束时的 $\mathcal{C}$ 的维度。实际上，受约束的 $\mathcal{C}$ 是 $\mathbb{R}^n$ 中的一个（或多个）流形。

前文中在处理碰撞约束时，一般采用的是否决法（rejection），即若采样或扩展得到的节点不满足碰撞约束时，则否决这个节点，重新采样或扩展。因为排除障碍物的 $\mathcal{C}_{\text{free}}$ 和原 $\mathcal{C}$ 是同维的，所以采样得到满足约束的状态的概率是存在的。但是，在 $\mathbb{R}^n$ 中采样采到低维流形上的概率为零，因此对于末端位姿约束的问题，不能简单采用否决法。

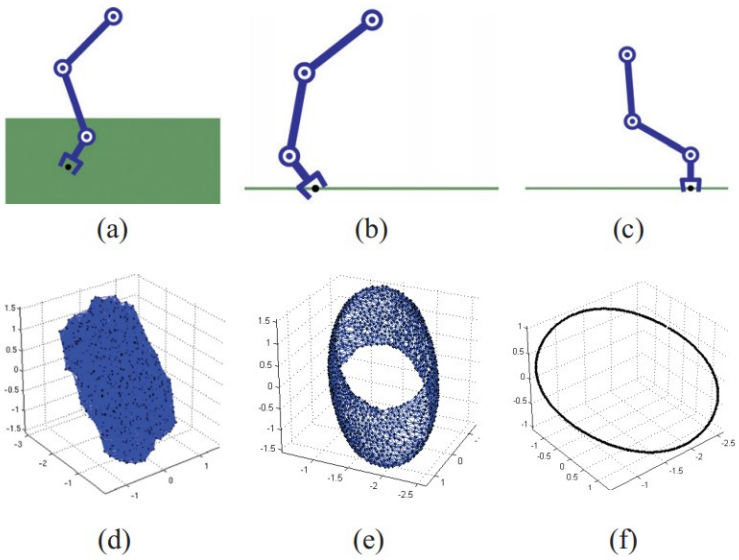



图 20: 末端位姿受约束的平面三自由度机械臂的构型空间 [16]。(a)(d) 末端位置被约束在一定平面区域内，此时其对应的 $\mathcal{C}$ 为 $\mathbb{R}^{\text{此}}$ 中的一个三维子空间；(b)(e) 末端位置被约束在一条直线上，此时对应 $\mathcal{C}$ 为 $\mathbb{R}^{\text{此}}$ 中的一个二维曲面；(c)(f) 末端位置被约束在一条直线上且朝向固定，此时对应 $\mathcal{C}$ 为 $\mathbb{R}^{\text{此}}$ 中的一条一维曲线。

解决末端位姿约束的路径规划的经典方法主要分两类。第一类是以任务空间为引导进行探索。此类方法在选择最近邻节点以末端位姿的距离作为标准。在树扩展时，首先在任务空间 $\mathcal{W}$ 中进行满足约束扩展，例如让末端只在 x 方向上运动，从而保证末端在 y 和 z 方向上的位置不变；然后再将 $\mathcal{W}$ 中的局部运动转为 $\mathcal{C}$ 中的局部运动。此类方法因为是在任务空间 $\mathcal{W}$ 中进行探索，所以不具有概率完备性，在机械臂需要经过奇异位形才能完成的任务中，此类方法往往无法找到可行路径。

第二类方法则在机械臂的构型空间中进行探索。其基本思想是：在普通 RRT 的基础上，将每次扩展得到的（不满足约束的）$\boldsymbol{q}_{\text{new}}$ 投影到约束流形上，从而保证每个新扩展的节点都满足位姿约束，如图21所示。其投影的方式是使用机械臂的雅克比矩阵，将机械臂迭代式地移动到满足位姿约束的状态下。具体算法可以参考 [17, 16, 15]。可以证明，此类方法具有概率完备性。

图22展示了两类方法在一个例子上的对比。大部分情况下，第二类方法都会有很好的性能表现，而且其十分通用，因此应用十分广泛。图23展示了此类方法的一些应用例子。

本课题组对上述机械臂路径规划问题和方法进行了更详细的总结和代码实现，可见https://github.com/Mingrui-Yu/arm_planning，欢迎参考和 star。

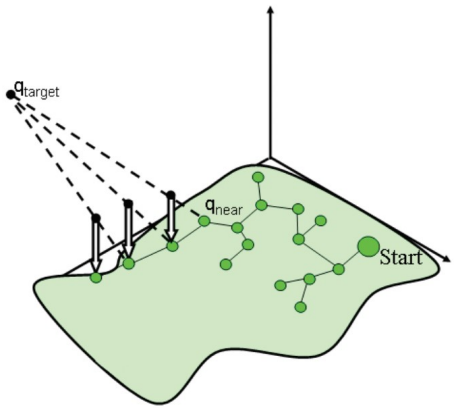



图 21: 使用状态投影进行约束扩展的示意图 [16]。从 $\boldsymbol{q}_{\text{near}}$ 开始，递进式地向 $\boldsymbol{q}_{\text{target}}$ 进行扩展，其中每一次扩展得到的节点都被投影到约束流形上。

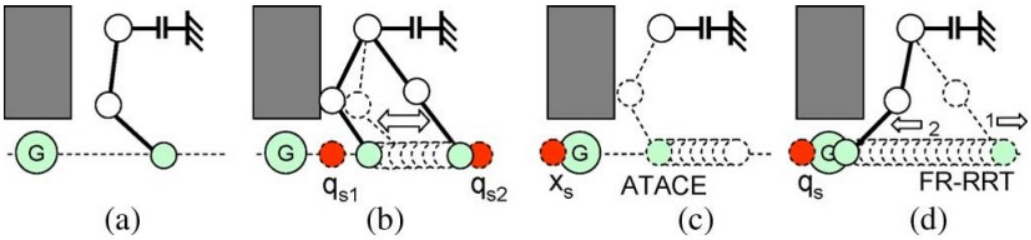



图 22: 两类约束路径规划算法的对比示意图 [15]。(a) 平面三自由度机械臂及其末端位置目标；(b) 假设两类方法都已探索到了图中黑色实线所示的构型；(c) 第一类方法选择任务空间中距离最近的节点，导致陷入局部最优；(d) 第二类方法考虑构型空间中的距离，最终可以让机械臂通过一个奇异位形到达目标位置。

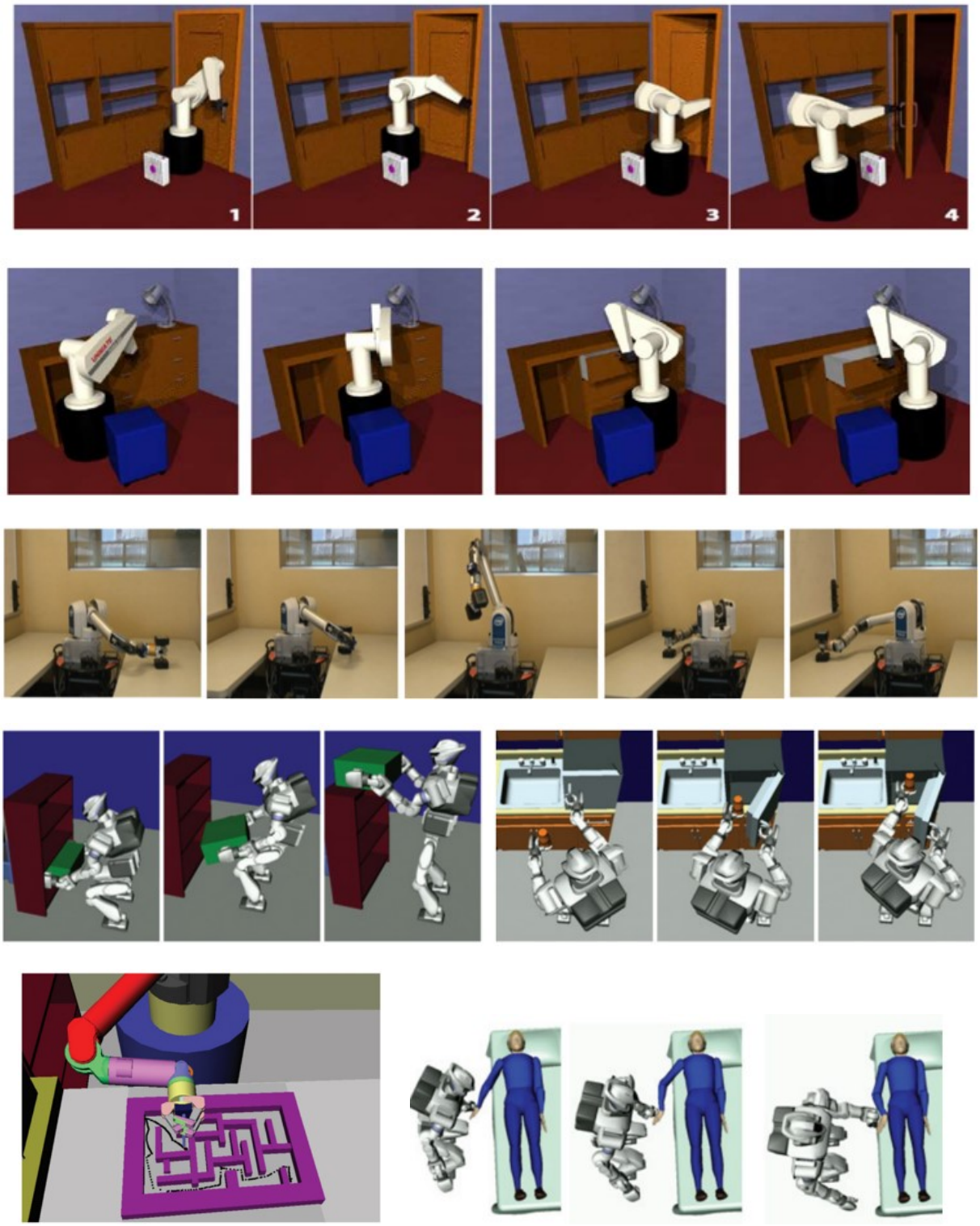



图 23: 第二类末端位姿受约束的路径规划算法的应用例子 [15, 16]。

# 5 本章总结

本章对机器人运动规划进行了初步的介绍。一般来说，运动规划（motion planning）、路径规划（path planning）、轨迹规划（trajectory planning）的含义略有不同：

1. 运动规划是比较广义的概念，机器人完成某个任务所需要的规划都可以算作运动规划；
2. 路径规划特指找到从初始状态到目标状态的路径 $y(s) : [0, 1] \to \mathcal{C}$，这里的路径一**般**是几何意义上的，与时间和机器人动力学无关；
3. 轨迹规划特指计算出机器人能实际执行的轨迹，每个轨迹点包含时间戳，需要考虑机器人的动力学约束。

受课时限制，本章主要对最基础的路径规划进行了初步的介绍。下面列举了一些本章没有介绍的内容及相应的可进一步阅读的学习材料：

1. 涉及微分约束的路径（轨迹）规划，即 kinodynamic planning：对于固定翼飞机、阿克曼转向的车辆（如小汽车）等，都需要使用 kinodynamic planning。感兴趣的同学可阅读网址和网址。
2. 涉及动态障碍物的路径（轨迹）规划：当环境中存在动态障碍物时，可能需要对障碍物的运动进行预测，也可能需要不确定环境下的规划。感兴趣的同学可阅读网址。
3. Time-optimal path parameterization（TOPP）问题，即时间最优的路径参数化（跟踪），其指的是对于已经实现规划好的无时间信息的路径，规划路径上每个点对应的时间，在满足各种约束的前提下，使路径被完成的耗时最短（或其它指标，例如能耗）。对于机械臂的 TOPP 问题可参考 [18, 19, 20, 21]。

# 6 相关课程资源

本章中大部分内容整理自以下相关学习资料，推荐同学们阅读原文进行更深入的学习。

相关书籍：

- Siciliano, “Robotics Modeling, Planning and Control”, 2009. 该书是机器人领域的经典教材，包括机器人从建模、规划、控制全方位的基础知识。其中第 12 章为 Motion Planning，篇幅较短，可用于初步了解机器人运动规划的概念和方法。
- Lavalle, “Planning Algorithms”, 2006. 该书是 Planning 领域最全面的教材，作者 Lavalle 是 RRT 及其多种变种的提出者。该书理论扎实、内容全面，适合用于极为深入地学习运动规划理论；但本书成书较早，无法包含近期的研究成果。该书可在线获取：http://lavalle.pl/planning/。

相关课程：

- “Robotic Systems”, UIUC, Keris Hauser, https://motion.cs.illinois.edu/RoboticSystems/. 其中第三章为 Motion Planning。该部分提供了详细的讲义，架构清晰，内容适中，易于阅读（非常推荐）。
- “Motion Planning”, University of Michigan, Dmitry Berenson, https://berenson.robotics.umich.edu/courses/winter2022motionplanning/index.html. 该课程完全聚焦机器人 motion planning，提供 slides，内容全面，链接了很多其它相关学习资料，且涉及一些较新的研究成果。
- “CS287 Advanced Robotics”, UCB, Pieter Abbeel, https://people.eecs.berkeley.edu/~pabbeel/cs287-fa19/. 其中 Lecture 10 为 Motion Planning。该课程提供了全部 slides 和录像。该课程对于 sampling-based planning （当前在机械臂运动规划问题上应用最为广泛）着墨较少，更多地讲解了 Markov decision process、optimization、reinforcement learning 等内容。

# 参考文献

[1] Z. Han, Z. Wang, N. Pan, Y. Lin, C. Xu, and F. Gao, "Fast-racing: An open-source strong baseline for SE(3) planning in autonomous drone racing," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 8631–8638, 2021.

[2] F. Burget, A. Hornung, and M. Bennewitz, "Whole-body motion planning for manipulation of articulated objects," in *2013 IEEE International Conference on Robotics and Automation*, pp. 1656–1662, 2013.

[3] D. Berenson, "Motion planning," 2022.

[4] H. Keris, "Robotic systems," 2022.

[5] 高翔, 张涛, 刘毅, and 颜沁睿, 视觉 *SLAM* 十四讲：从理论到实践. 电子工业出版社, 2017.

[6] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. London: Springer, 2009.

[7] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[8] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration.," *VISAPP (1)*, vol. 2, no. 331-340, p. 2, 2009.

[9] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 2, pp. 995–1001, IEEE, 2000.

[10] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[11] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed RRT: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *2014 IEEE/RSJ international conference on intelligent robots and systems*, pp. 2997–3004, IEEE, 2014.

[12] D. Bertram, J. Kuffner, R. Dillmann, and T. Asfour, "An integrated approach to inverse kinematics and path planning for redundant manipulators," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pp. 1874–1879, IEEE, 2006.

[13] M. V. Weghe, D. Ferguson, and S. S. Srinivasa, "Randomized path planning for redundant manipulators without inverse kinematics," in *2007 7th IEEE-RAS International Conference on Humanoid Robots*, pp. 477–482, IEEE, 2007.

[14] N. Vahrenkamp, D. Berenson, T. Asfour, J. Kuffner, and R. Dillmann, "Humanoid motion planning for dual-arm manipulation and re-grasping tasks," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2464–2470, IEEE, 2009.

[15] M. Stilman, "Global manipulation planning in robot joint space with task constraints," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 576–584, 2010.

[16] D. Berenson, S. Srinivasa, and J. Kuffner, "Task space regions: A framework for pose-constrained manipulation planning," *The International Journal of Robotics Research*, vol. 30, no. 12, pp. 1435–1460, 2011.

[17] D. Berenson, S. S. Srinivasa, D. Ferguson, and J. J. Kuffner, "Manipulation planning on constraint manifolds," in *2009 IEEE international conference on robotics and automation*, pp. 625–632, IEEE, 2009.

[18] D. Verscheure, B. Demeulenaere, J. Swevers, J. De Schutter, and M. Diehl, "Time-optimal path tracking for robots: A convex optimization approach," *IEEE Transactions on Automatic Control*, vol. 54, no. 10, pp. 2318–2327, 2009.

[19] Q.-C. Pham, "A general, fast, and robust implementation of the time-optimal path parameterization algorithm," *IEEE Transactions on Robotics*, vol. 30, no. 6, pp. 1533–1540, 2014.

[20] H. Pham and Q.-C. Pham, "A new approach to time-optimal path parameterization based on reachability analysis," *IEEE Transactions on Robotics*, vol. 34, no. 3, pp. 645–659, 2018.

[21] D. Kaserer, H. Gattringer, and A. Müller, "Nearly optimal path following with jerk and torque rate limits using dynamic programming," *IEEE Transactions on Robotics*, vol. 35, no. 2, pp. 521–528, 2018.