CS7355
自动驾驶前沿技术

2024年4月24日

自动驾驶路径规划

任务规划（Mission Planning）

↓

行为规划（Behavior Planning）

↓

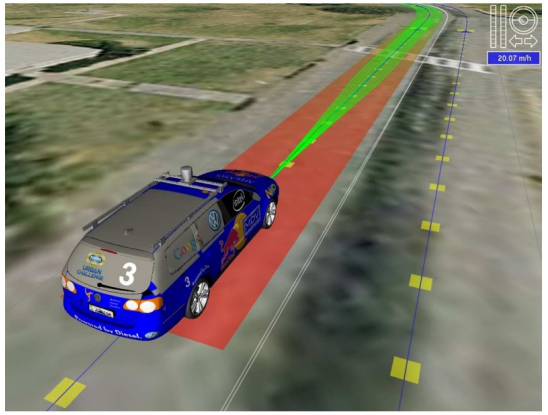
运动规划（Motion Planning）

路径规划（Path Planning） | 速度分布（Velocity Profile）

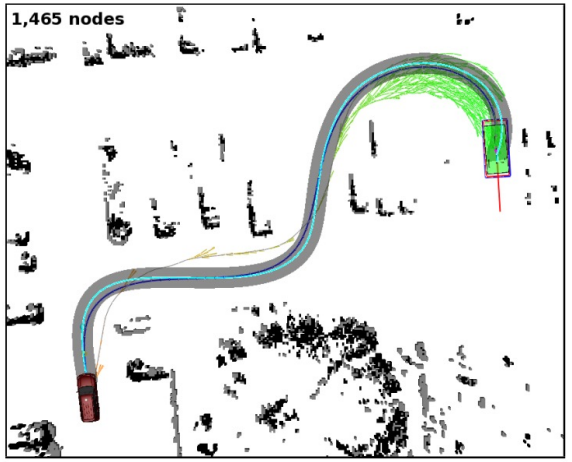
↓

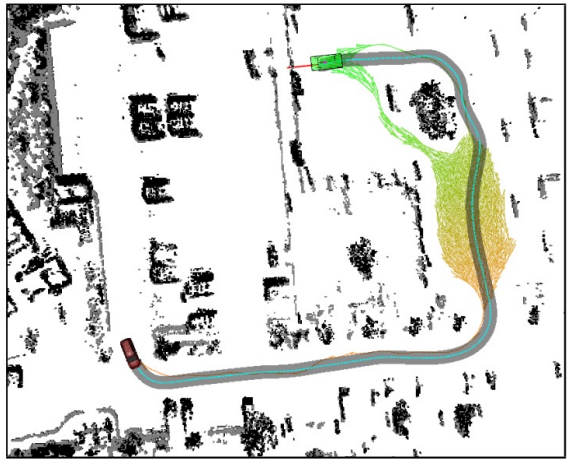
运动控制（Motion Control）

结构化场景 -> Conformal Lattice Planner

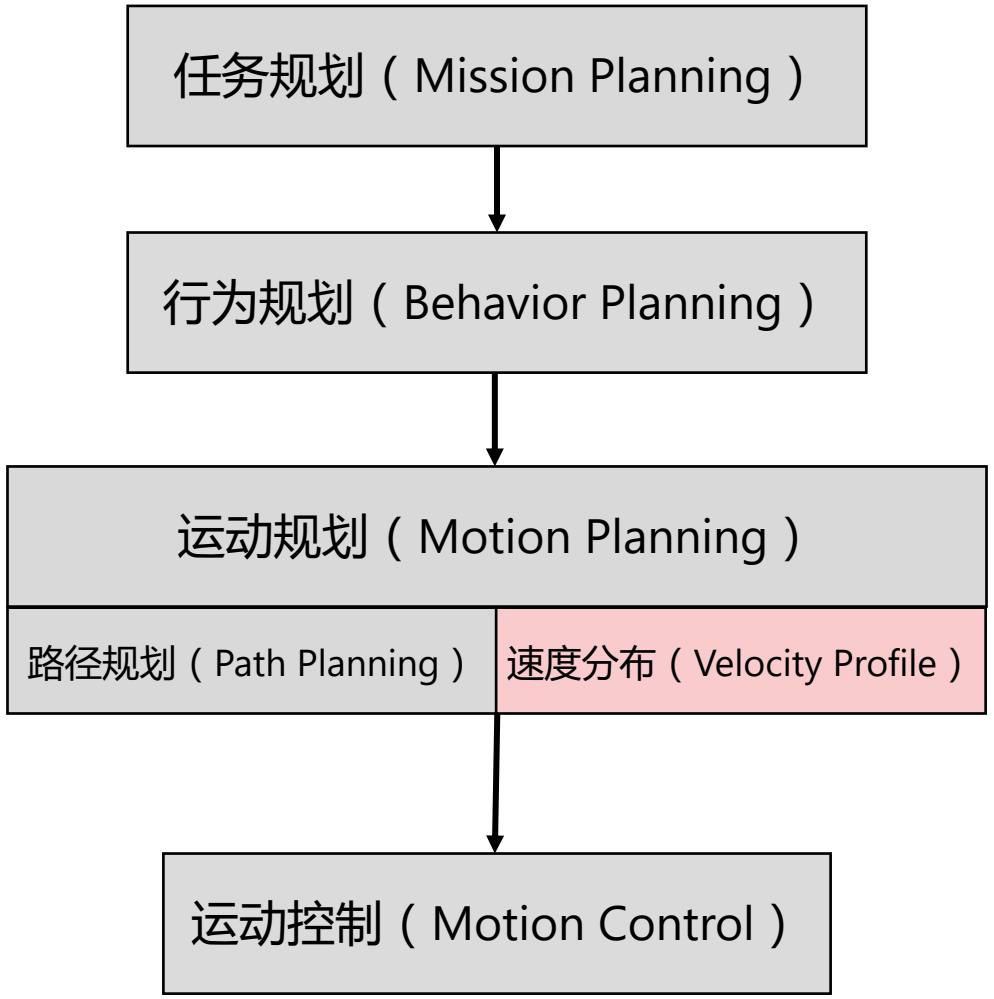


非结构化场景 -> Hybrid A* + 复合 heuristics

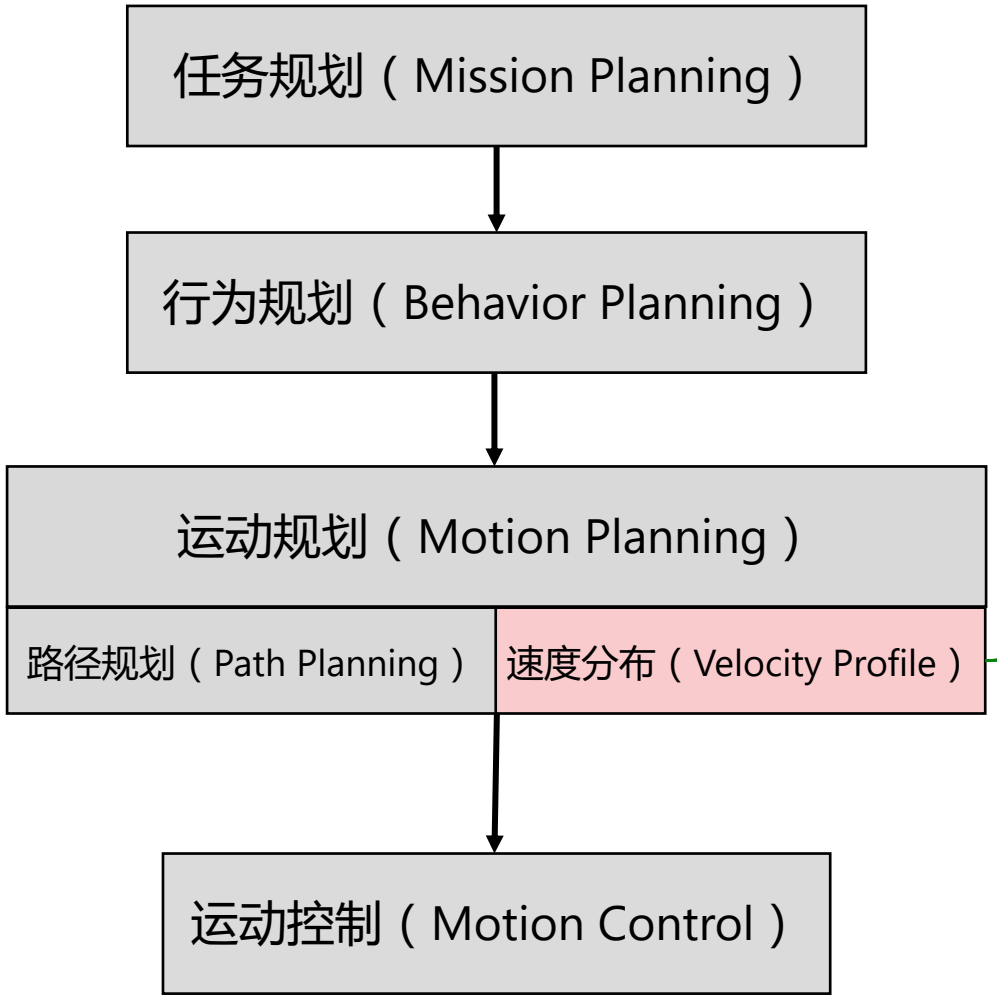




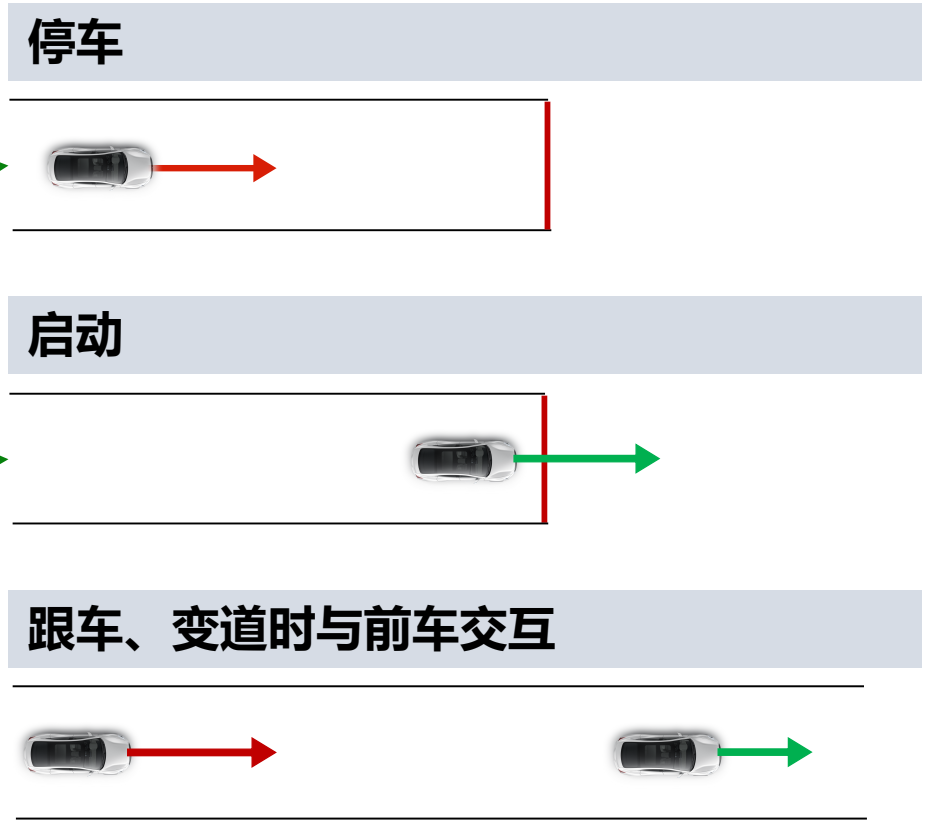
自动驾驶速度分布生成


任务规划（Mission Planning）
行为规划（Behavior Planning）
运动规划（Motion Planning）
路径规划（Path Planning）
速度分布（Velocity Profile）
运动控制（Motion Control）

自动驾驶速度分布生成



结构化场景



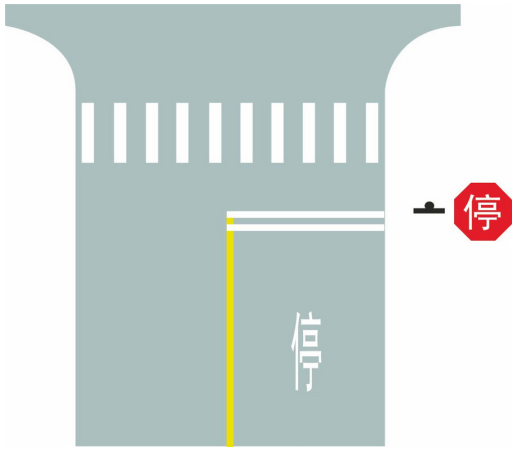
结构化场景中的目标速度 – 影响因素 I

- 参考速度 v_{ref}
  - 由用户指定：*“请开到 80 km/h”*
  - 受到道路限速的限制
  - 收到交通管制装置的约束

$v_{ref} \leq 60\ km/h$

$v_{ref} = 0\ km/h$

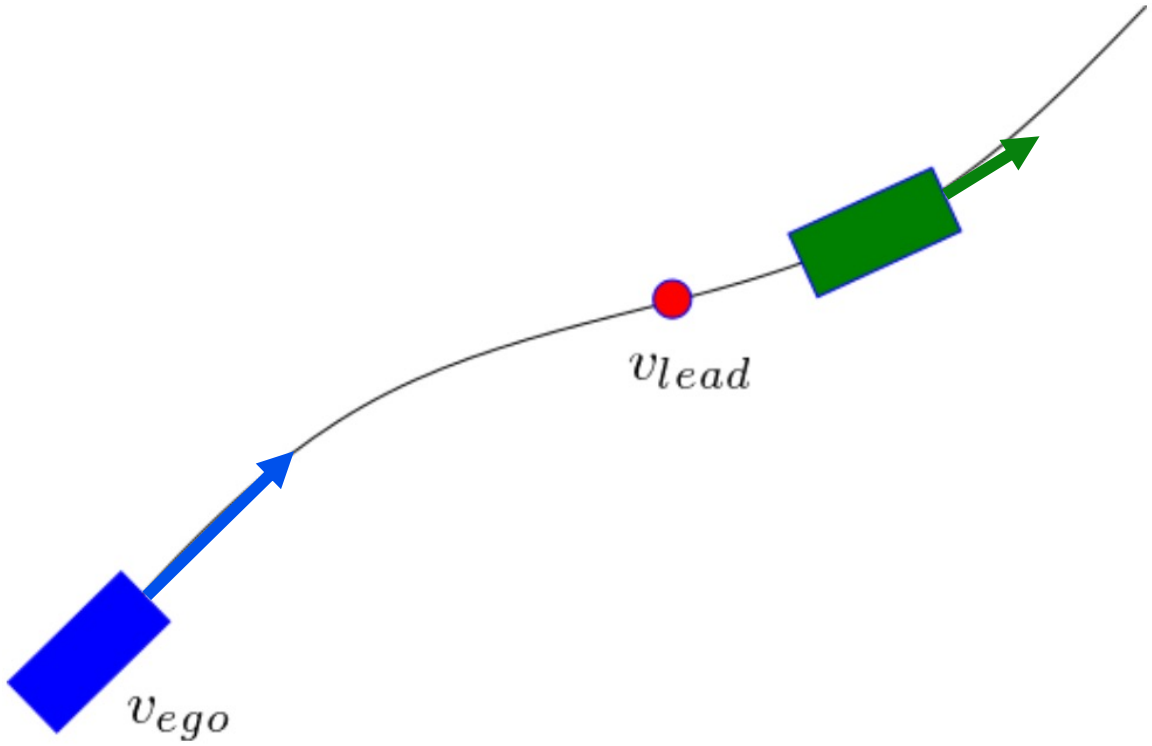
$v_{ref} = 0$（持续2s）



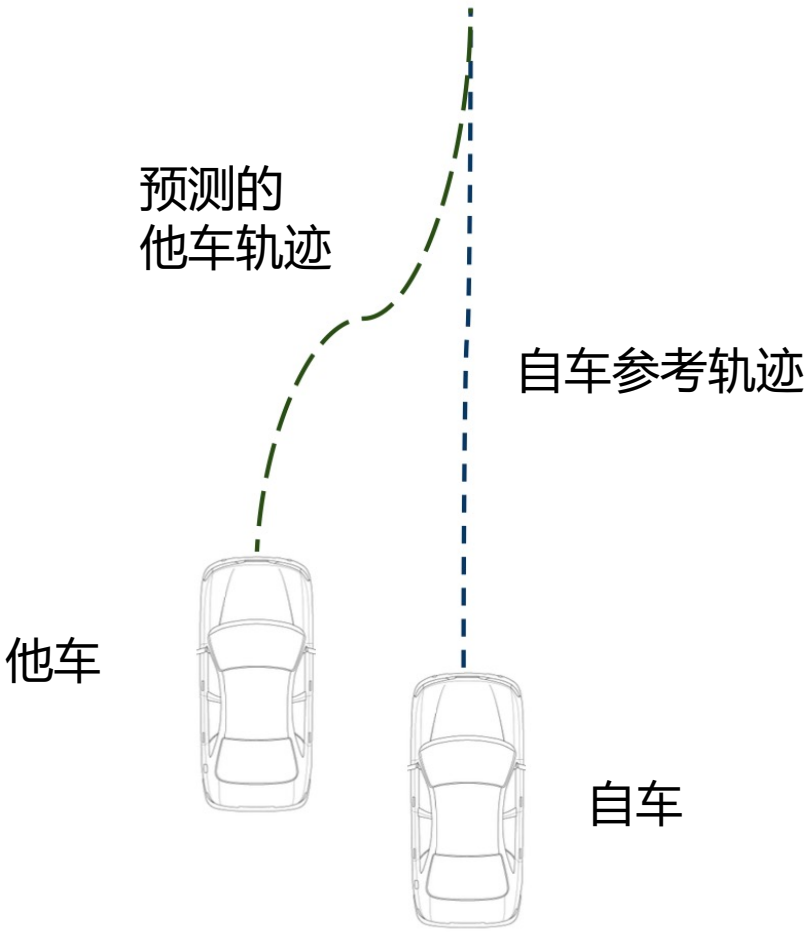
结构化场景中的目标速度 – 影响因素 II

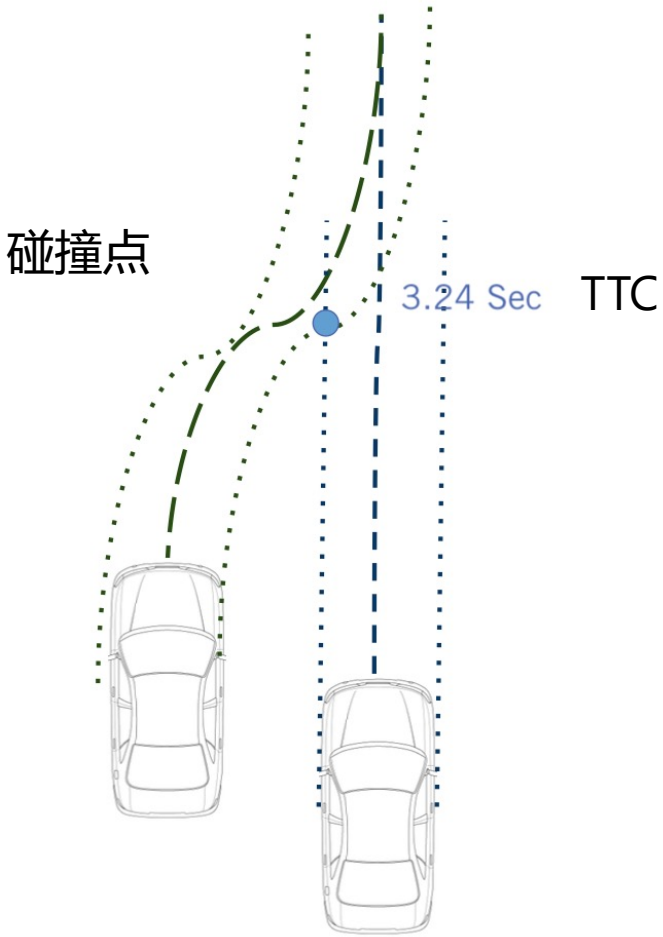
- 前车速度 v_{lead}
  - 在自车到达前车尾部附近时，自车速度必须降到前车速度相同或者之下

- 红点：
  - 碰撞位置（collision point）



碰撞点（Collision Point）与碰撞时间（Time-To-Collision）

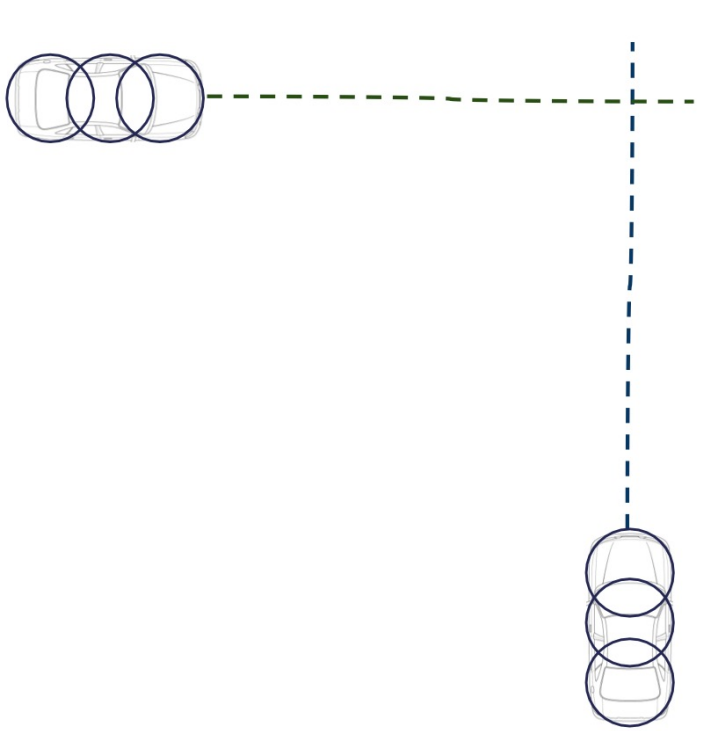


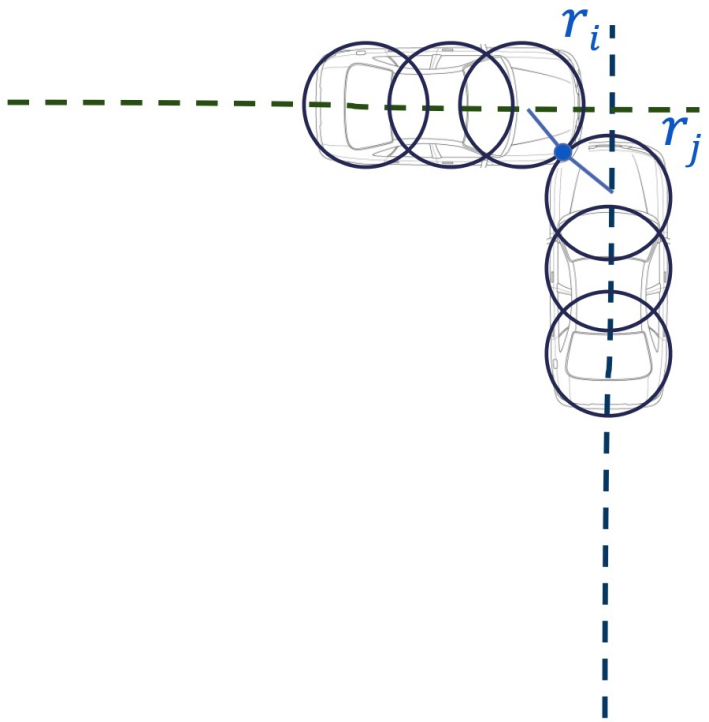


碰撞点（Collision Point）与碰撞时间（Time-To-Collision）

- 将车辆模拟成3个圆盘，按照预测轨迹向前模拟车的未来状态
- 在每一个时间步，进行碰撞检测
- 当碰撞发生时，记录碰撞时间、碰撞点

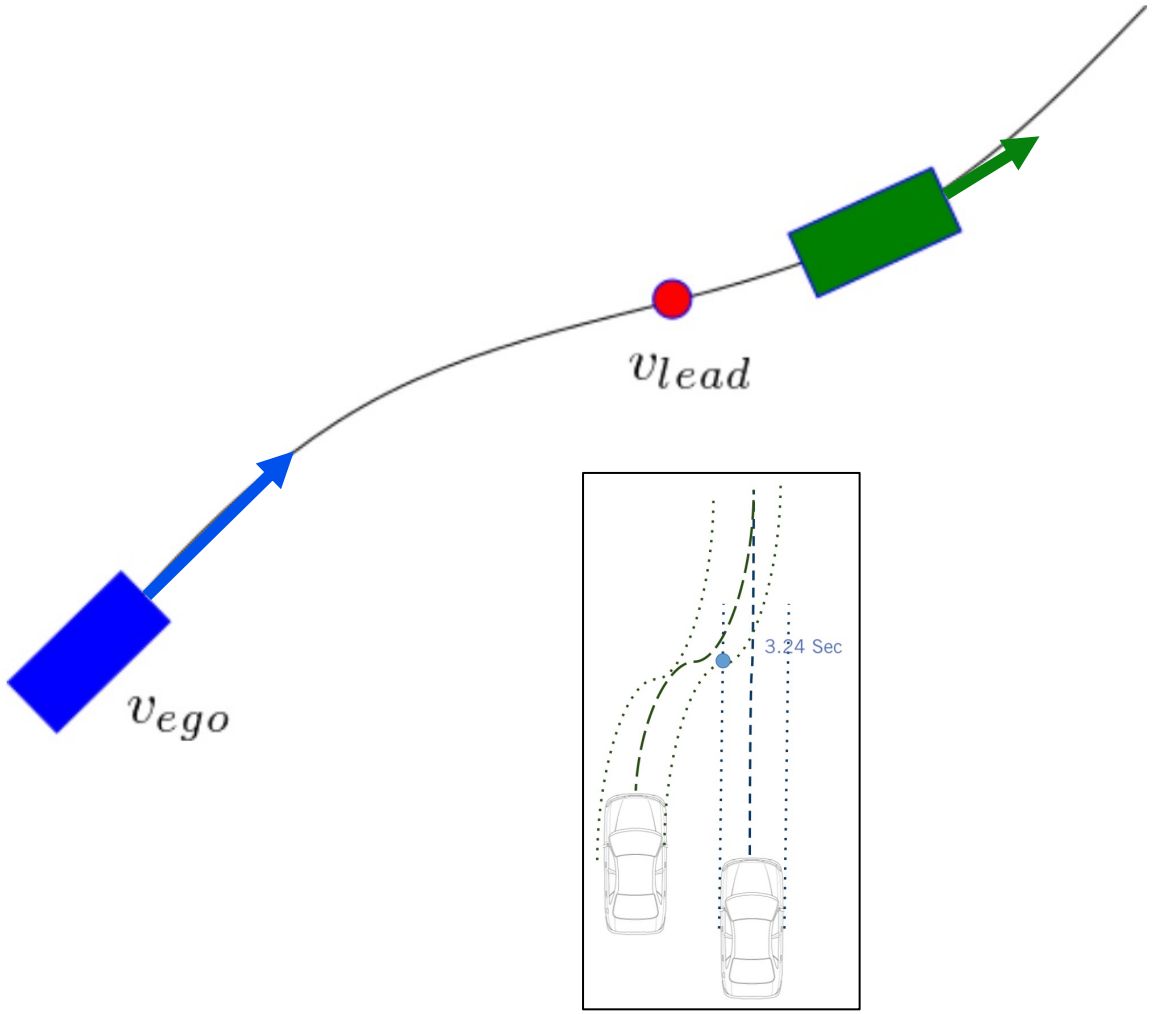
$$C_X = \frac{(x_i * r_j) + (x_j * r_i)}{(r_i + r_j)}, \qquad C_Y = \frac{(y_i * r_j) + (y_j * r_i)}{(r_i + r_j)}$$





结构化场景中的目标速度

- 前车速度 v_{lead}
  - 在到达路径上的**碰撞点（红点）**时，自车速度必须降到与前车速度相同或者之下

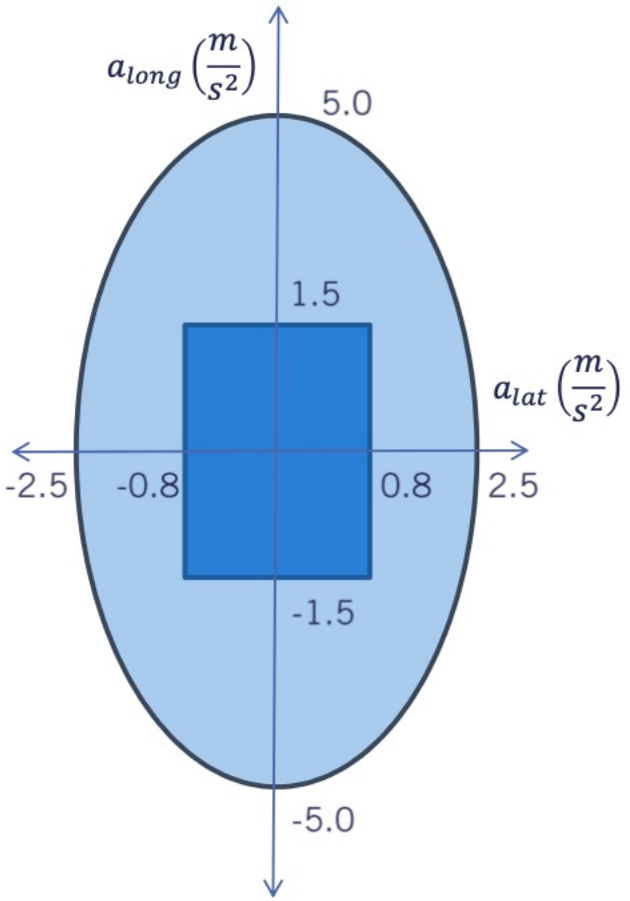


结构化场景中的目标速度 – 影响因素 III

- 运动学与动力学限制的最大速度 v_k
  - **摩擦力椭圆**：车辆加速度在椭圆内不会导致打滑
  - **舒适矩形**：车辆加速度在矩形内，乘客感到舒适
  - 横向加速度限制 -> 纵向速度限制：
    - 车辆横向加速度与速度、转弯半径的关系：

$$a_{lat} = \frac{v^2}{r}, \qquad a_{lat} \leq a_{lat_{max}}$$

    - 根据曲率定义： $\kappa = \frac{1}{r}$
    - 于是： $v^2 \leq \frac{a_{lat_{max}}}{\kappa}$



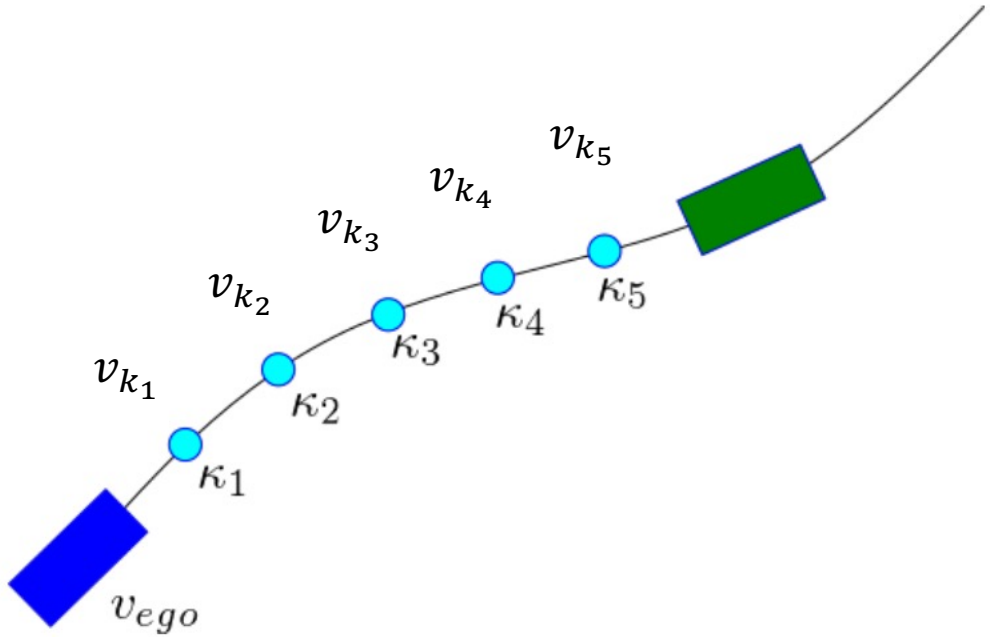
结构化场景中的目标速度分布

- 运动学与动力学限制的最大速度 v_k
  - 在**轨迹上任意一个点**，若当时的曲率是 κ_i，那么：

$$v_k \leq \sqrt{\frac{a_{lat}}{\kappa_i}}$$

  - 车辆在运动到第 i 个离散点时，必须满足：

$$v \leq v_{k_i}$$



结构化场景中的目标速度分布

- 结合以上三项因素，可以算出最终目标速度v_f：

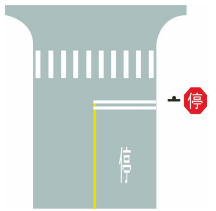
$$v_f = \min(v_{ref}, v_{lead}, v_k)$$

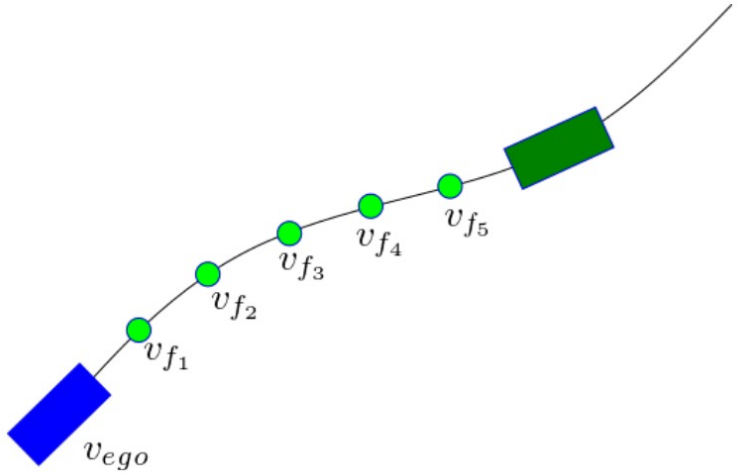
min

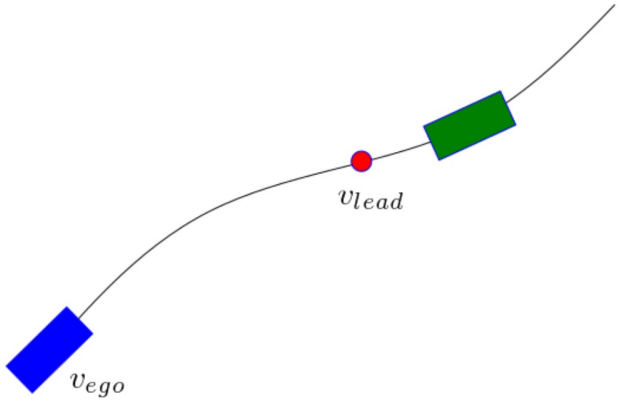
v_{ref}

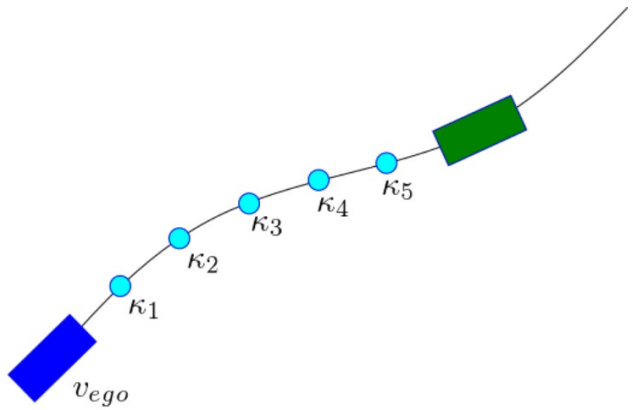
v_{lead}

v_k









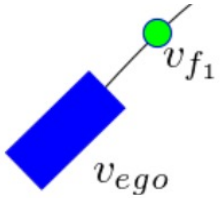
结构化场景中的速度曲线生成

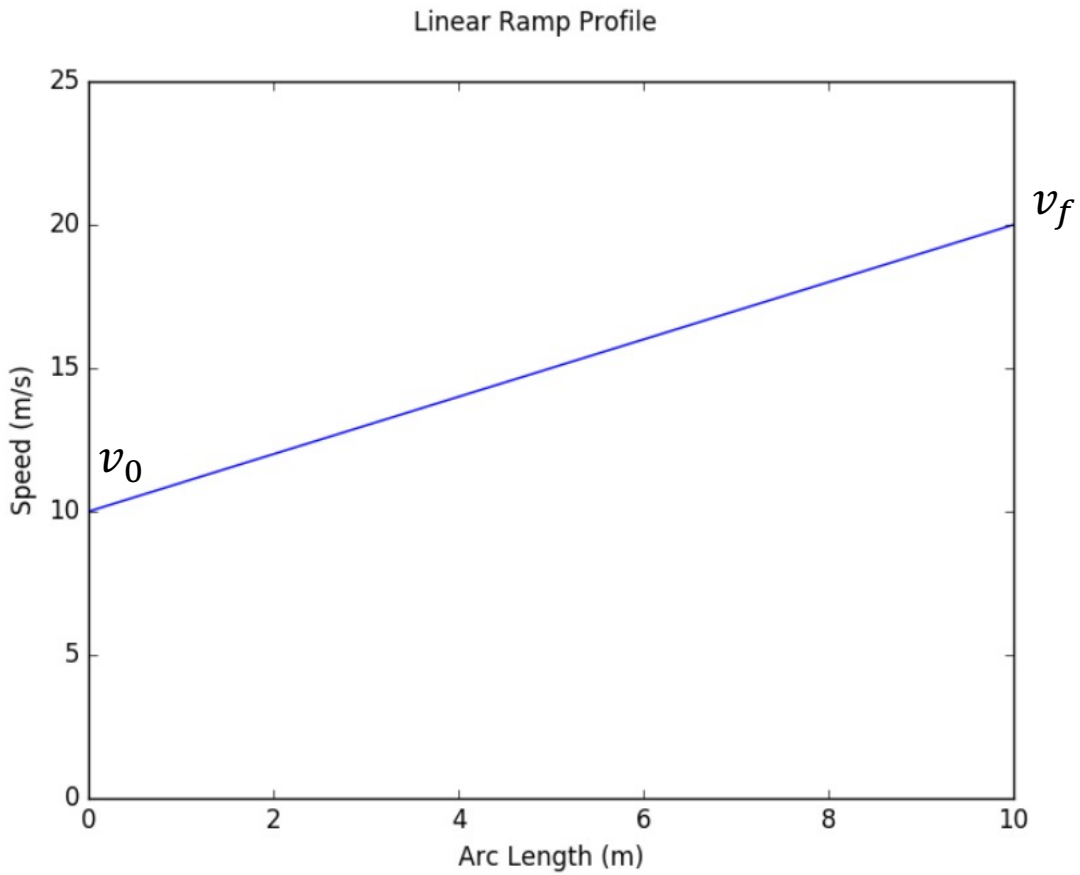
- 在当前路径段内，将速度从 v_0 变化到 v_f
- **方法1**：线性斜坡曲线（Linear Ramp Profile）
  - 将 v_0 与 v_f 沿着弧长维度线性插值
  - 需要的加速度可以计算为：

$$\frac{v_f^2 - v_0^2}{2s} = a$$

  - 若达到最大加速度限制，则不一定能达到 v_f：

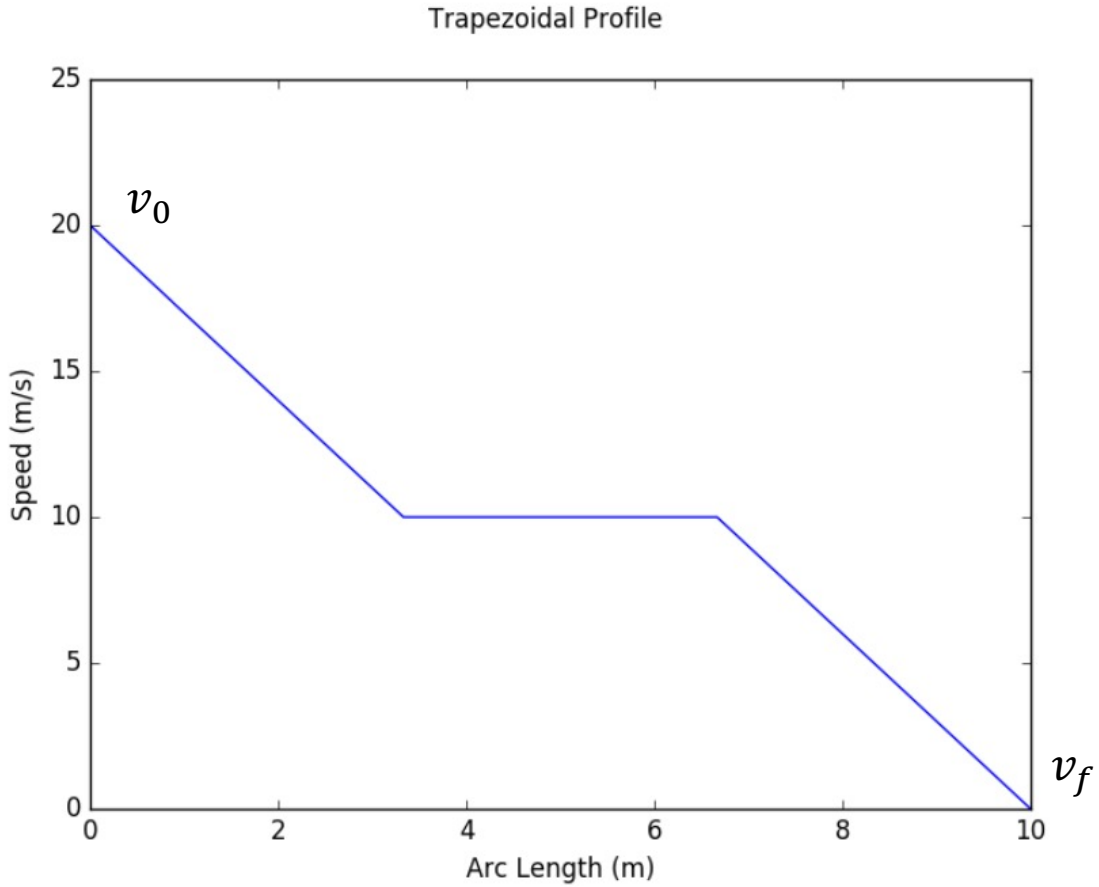
$$\sqrt{2as + v_0^2} = v_f$$





结构化场景中的速度曲线生成

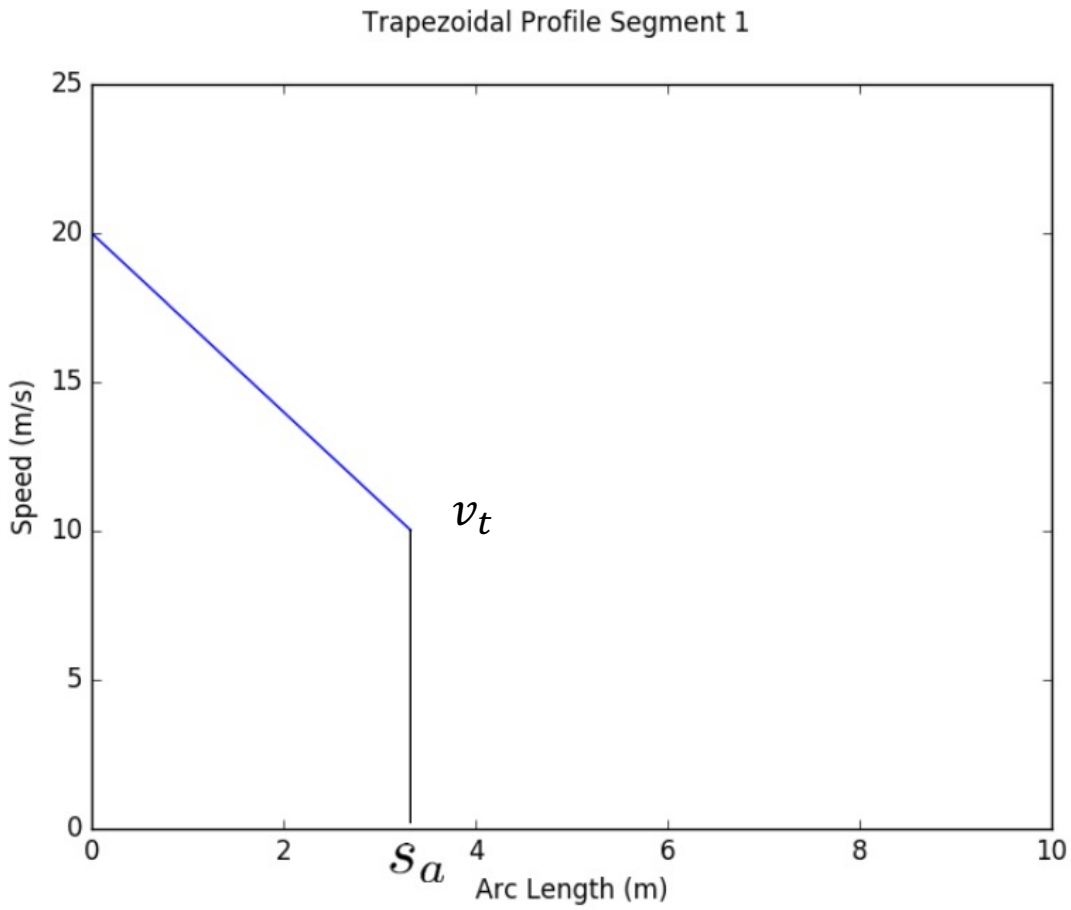
- 在当前路径段内，将速度从 v_0 变化到 v_f
- **方法2**：梯形曲线（Trapezoidal Profile）
  - 匀加/减速-匀速-匀加/减速
  - 停车时适用



结构化场景中的速度曲线生成

- 在当前路径段内，将速度从 v_0 变化到 v_f
- **方法2**：梯形曲线（Trapezoidal Profile）
  - **第一段**：执行匀加速度 s_0 达到临时速度 v_t

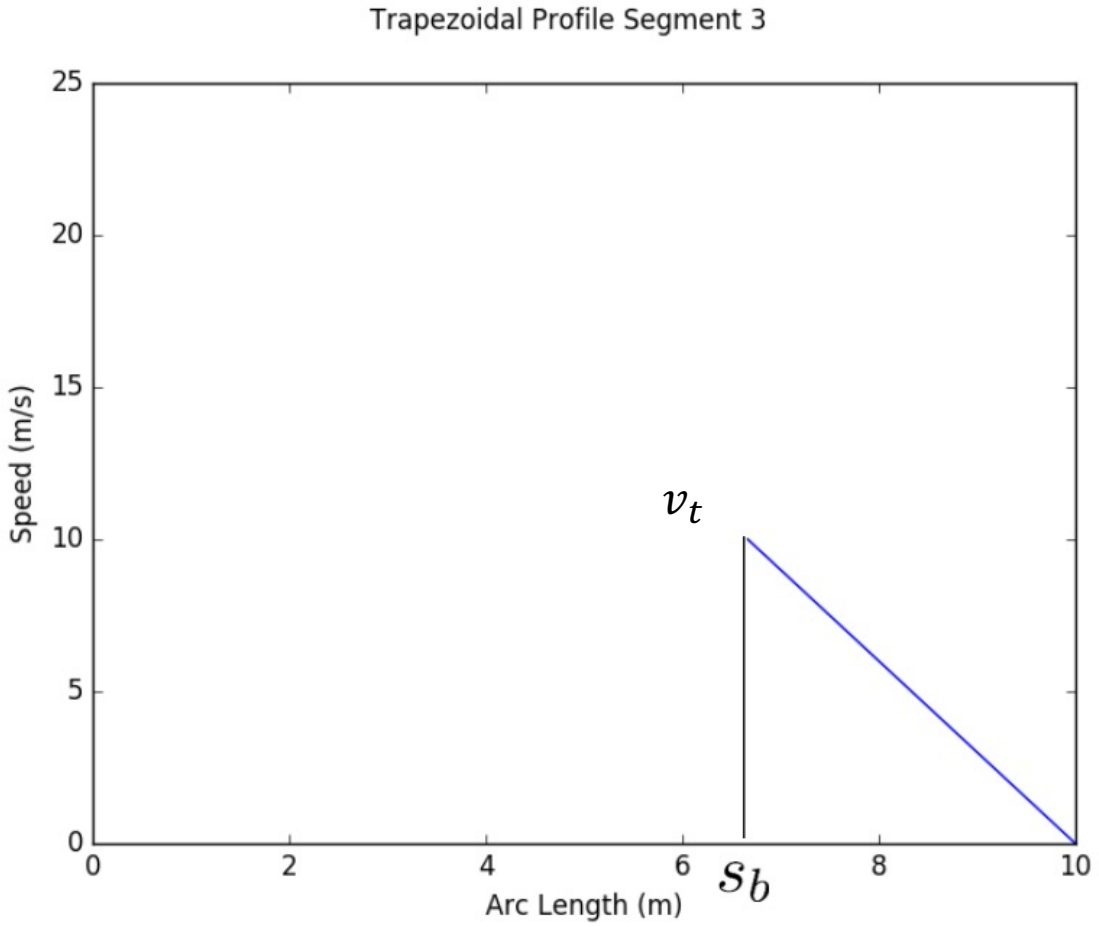
$$\frac{v_t^2 - v_0^2}{2a_0} = s_a$$



结构化场景中的速度曲线生成

- 在当前路径段内，将速度从 v_0 变化到 v_f
- **方法2**：梯形曲线（Trapezoidal Profile）
  - **第一段**：执行匀加速度 s_0 达到临时速度 v_t
  - **第三段**：执行匀加速度 s_0 达到最终速度 0

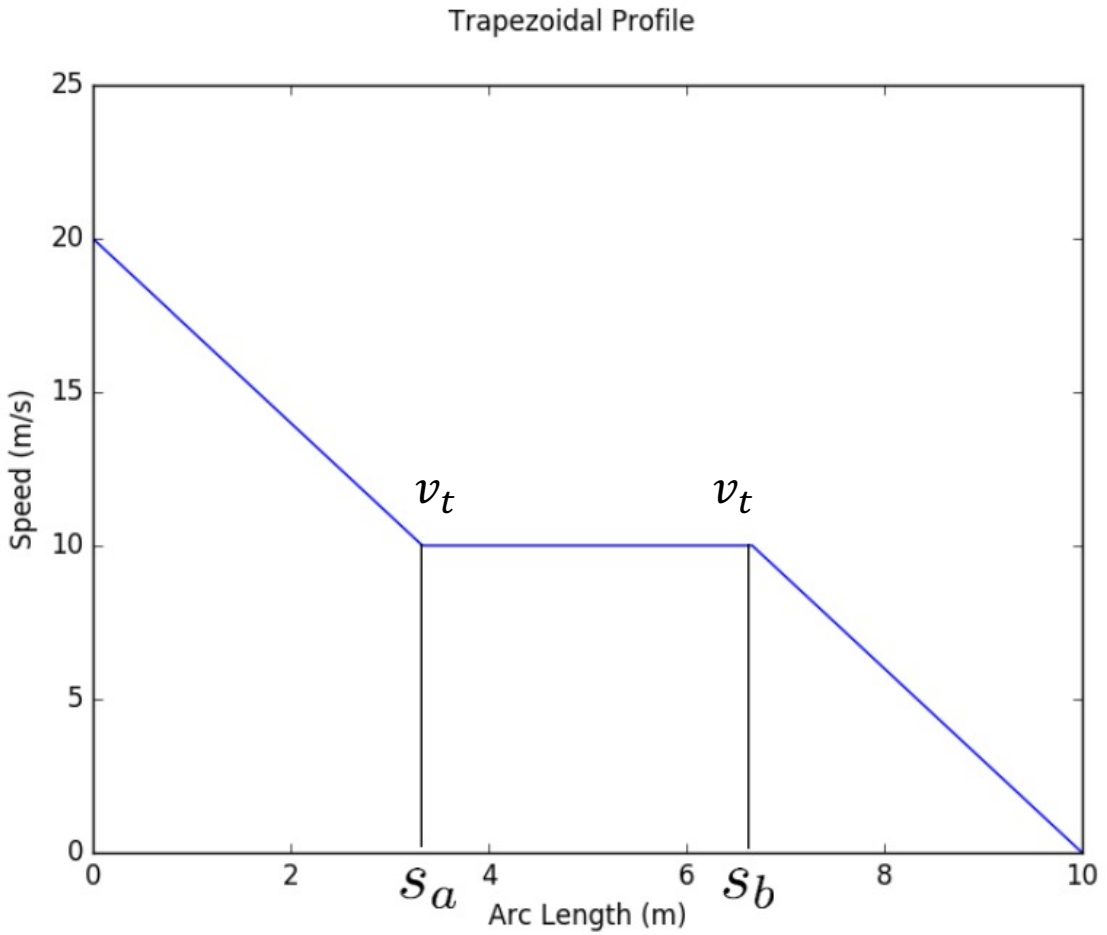
$$\frac{0-v_t^2}{2a_0} = s_f - s_b$$



结构化场景中的速度曲线生成

- 在当前路径段内，将速度从 v_0 变化到 v_f
- **方法2**：梯形曲线（Trapezoidal Profile）
  - **第一段**：执行匀加速度 s_0 达到临时速度 v_t
  - **第二段**：执行匀速运动
  - **第三段**：执行匀加速度 s_0 达到最终速度 0

$$v_{f_i} = \begin{cases} \sqrt{2a_0 s_i + v_i^2}, & s_i \le s_a \\ v_t, & s_a \le s_i \le s_b \\ \sqrt{2a_0(s_i - s_b) + v_i^2}, & s_b \le s_i \le s_f \end{cases}$$



$$\frac{v_t^2 - v_0^2}{2a_0} = s_a$$

$$\frac{0 - v_t^2}{2a_0} = s_f - s_b$$

结构化场景中的速度曲线

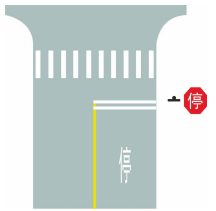
$$v_f = \min(v_{ref}, v_{lead}, v_k)$$

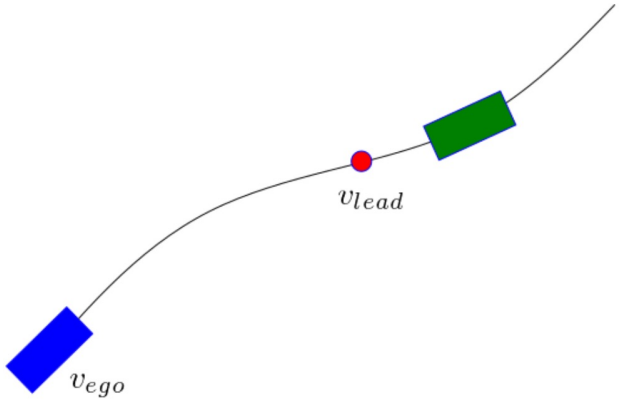
min

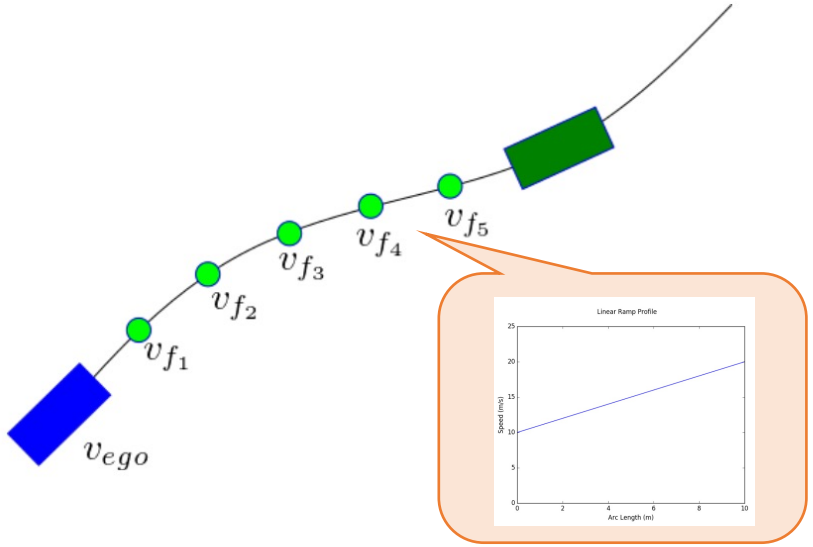
v_{ref}

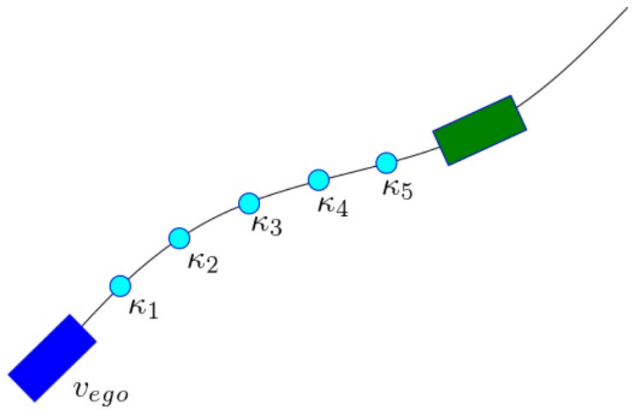
v_{lead}

v_k









自动驾驶速度分布生成

任务规划（Mission Planning）

行为规划（Behavior Planning）

运动规划（Motion Planning）

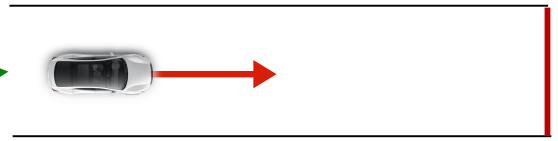
路径规划（Path Planning） | 速度分布（Velocity Profile）

运动控制（Motion Control）

结构化场景

非结构化场景

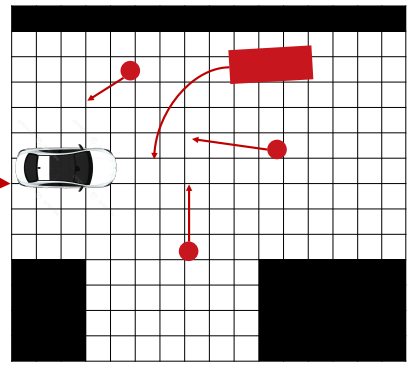
停车



启动

跟车、变道时与前车交互

复杂动态交互



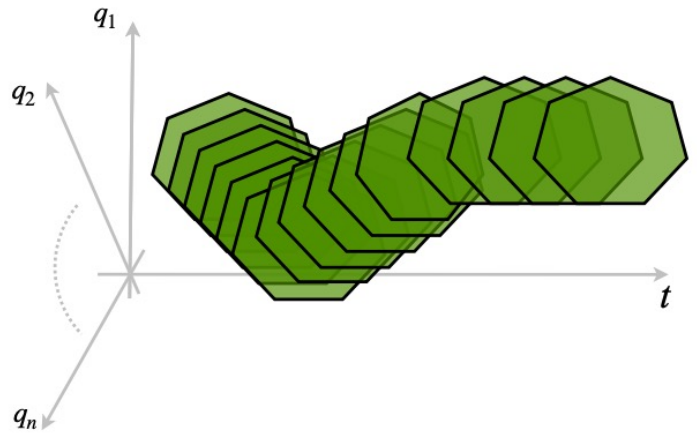
Recap：Hybrid A* 时空搜索

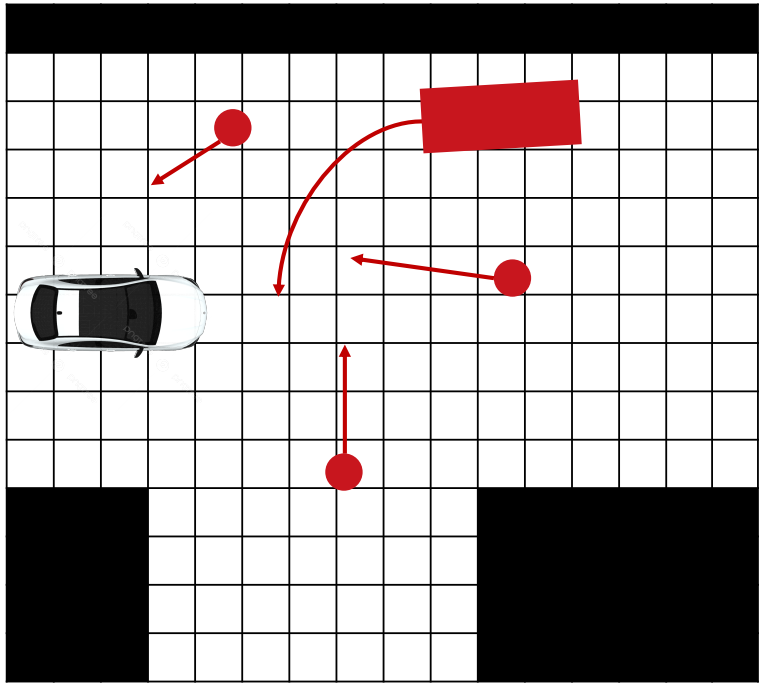
- **5D 时空栅格**：$< x, y, \theta, dir, t >$

速度！

- **自车轨迹扩展**：$< q', t+1 > = < f(q, u), t >$
  - 他人/车轨迹预测：$< q'_{exo}, t+1 >= m(h_{exo}, h, c, t)$
  - 碰撞检测：
    - 对于所有$t', < q', t' >$与$< q'_{exo}, t' >$的几何是否发生重叠？

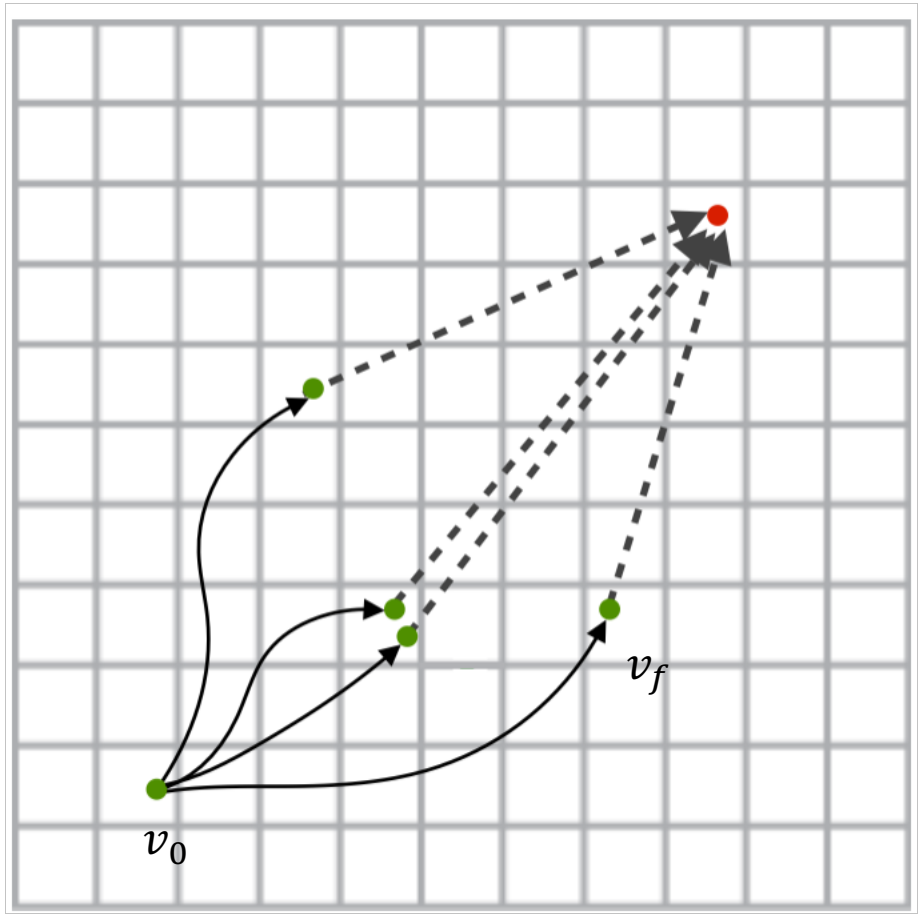
- 在搜索中如何引入速度信息？





Hybrid A* 时空搜索 + 速度控制

- **轨迹扩展控制输入**： $u = < \phi, v_f >$
  - ϕ = 方向盘角度 或 ϕ = 方向盘的变化率
  - v_f = 终点速度
  - v_f 也需要受到多方面的限制：
    - 交通限速 (<=60) 、红绿灯（ =0 ）
    - 曲率限制（与方向盘角度相关）



自动驾驶速度分布生成

任务规划（Mission Planning）

行为规划（Behavior Planning）

运动规划（Motion Planning）

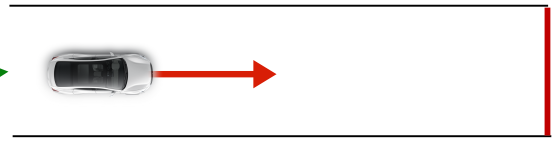
路径规划（Path Planning） | 速度分布（Velocity Profile）

运动控制（Motion Control）

结构化场景

非结构化场景

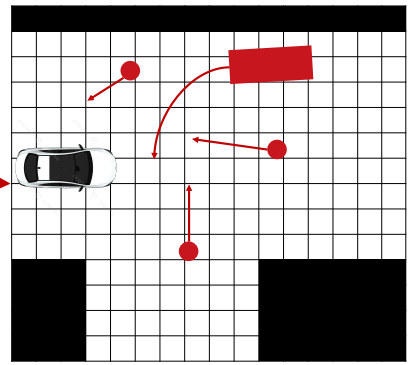
停车



启动

跟车、变道时与前车交互

其他复杂交互



自动驾驶运动规划实用方法

没有完备性、最优性等理论性质！

任务规划（Mission Planning）

↓

行为规划（Behavior Planning）

↓

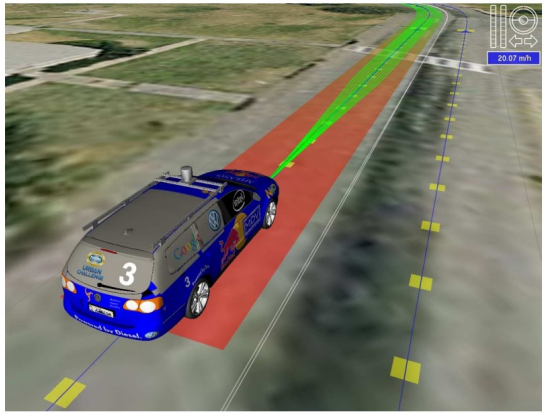
运动规划（Motion Planning）

路径规划（Path Planning） | 速度分布（Velocity Profile）

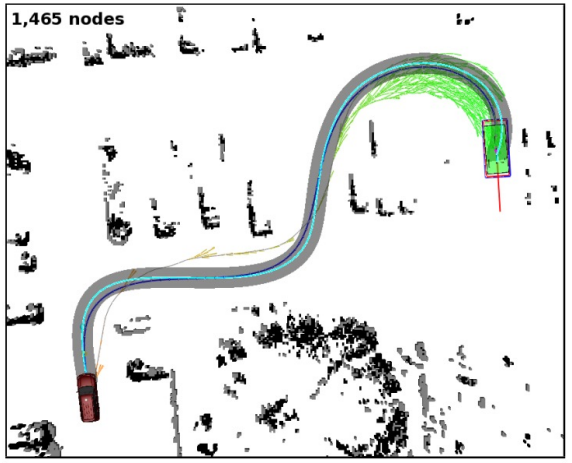
↓

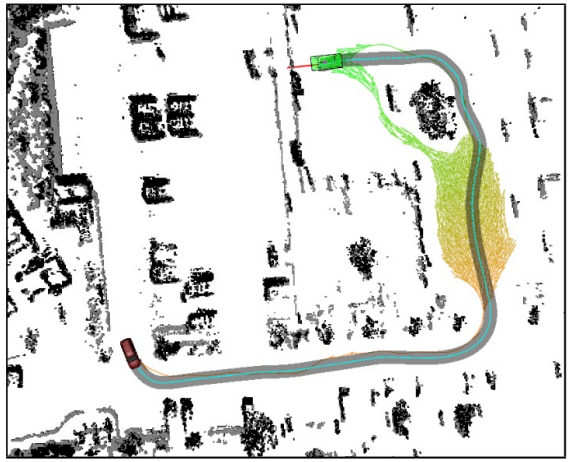
运动控制（Motion Control）

结构化场景 -> Conformal Lattice Planner



非结构化场景 -> Hybrid A* + 复合 heuristics + 速度输入





自动驾驶运动规划实用方法的理论性质

- **完备性**：当解存在时，算法可以保证在有限时间内找到解；当解不存在时，算法也可以在有限时间内报告不存在
  - Conformal lattice planner 不具备完备性
    - 无法考虑所有可能的行车轨迹
  - Hybrid A* 不具备完备性
    - 扩展时只考虑有限的、固定时长的控制输入
- **最优性**：当有解存在时，算法可以保证在有限时间内找到最优解
  - 最优性需建立在完备性的基础上
  - Conformal lattice planner 与 Hybrid A* 都不具备最优性

是否存在理论性质好的运动规划算法？

自动驾驶与机器人学

机器人学

系统架构

感知 | 控制

运动规划 | 策略规划

有限应用

自动驾驶技术

领域内拓展

机器人运动规划

看起来很不一样？

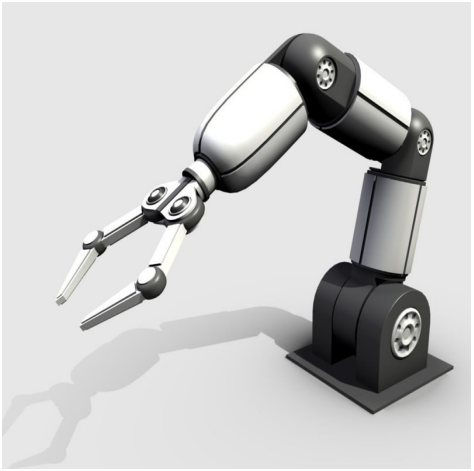
其实不然

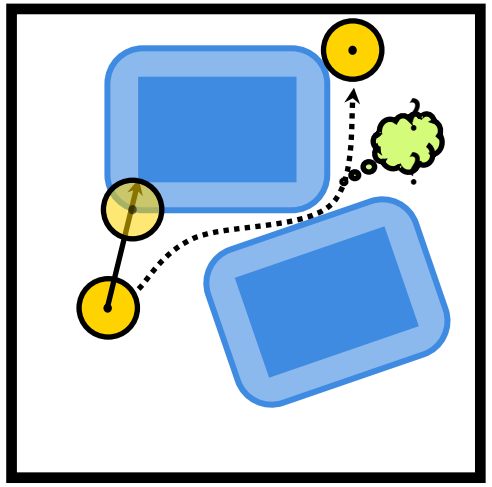
如何为所有机器人设计通用的运动规划算法？

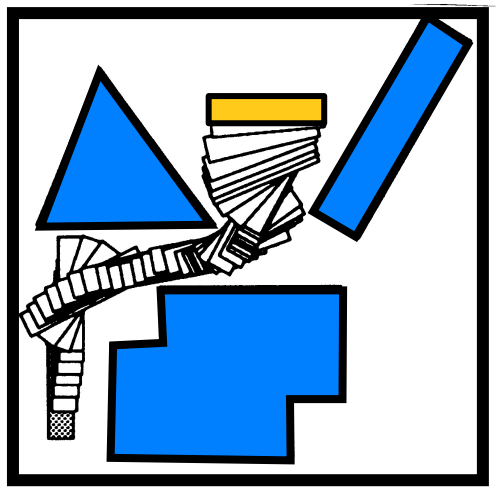
扫地机器人
是一个圆盘

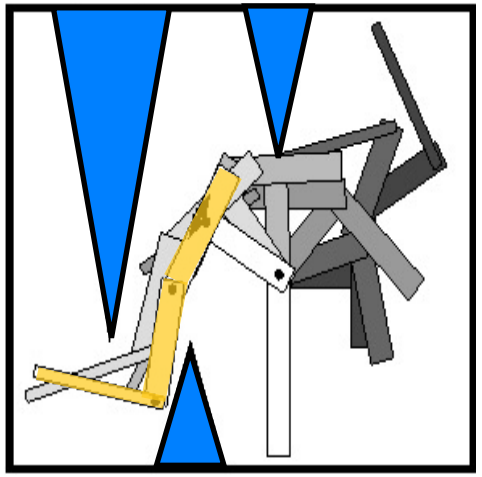
自动驾驶汽车
形状不完美对称

关节机器人
复杂的运动学结构



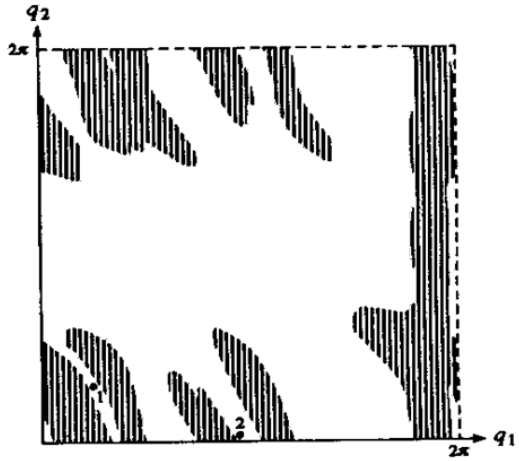






如何为所有机器人设计通用的运动规划算法？

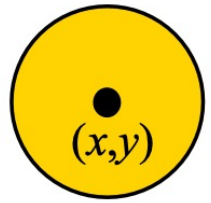
- 配置空间（Configuration Space）
  - 配置空间中所有机器人都可以被当做一个点



Tomas Lozano-Perez
Professor, MIT

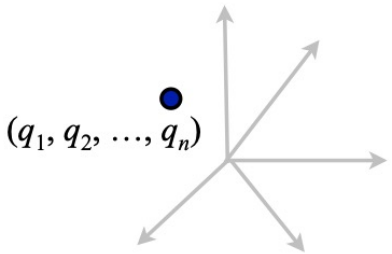
配置与配置空间

- 机器人的配置 (configuration)
  - 是一组能够完全确定机器人身上每个点的位置的参数 $q = (q_1, q_2, \ldots, q_n)$

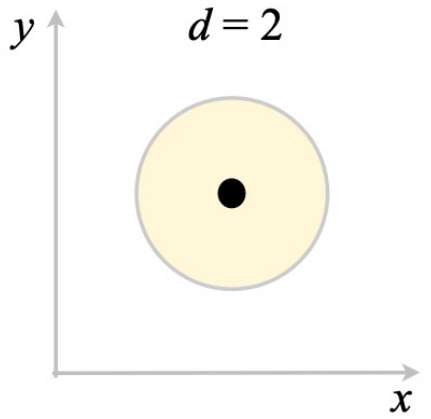


The coordinates for the disc center specify the configuration of a translation-only disc.

- 配置空间 C (Configuration space or C-space)
  - 是机器人所有可能配置构成的集合，也就是说，任何配置都是配置空间中的一个点

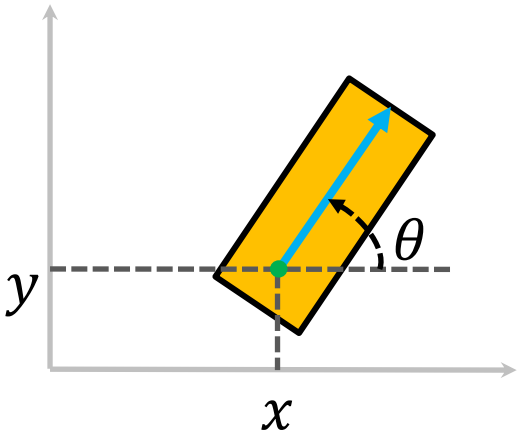


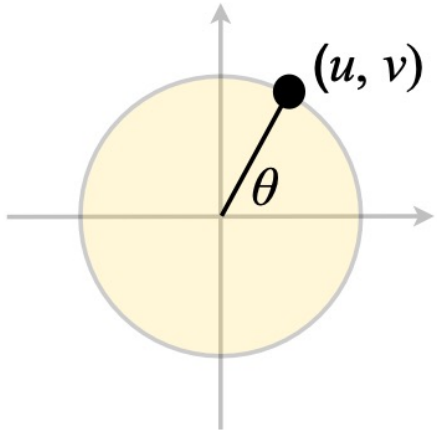
- 配置空间维度 d
  - 是可以确定机器人配置的最小参数量，也叫做机器人的自由度（Degrees of freedom, DOF）

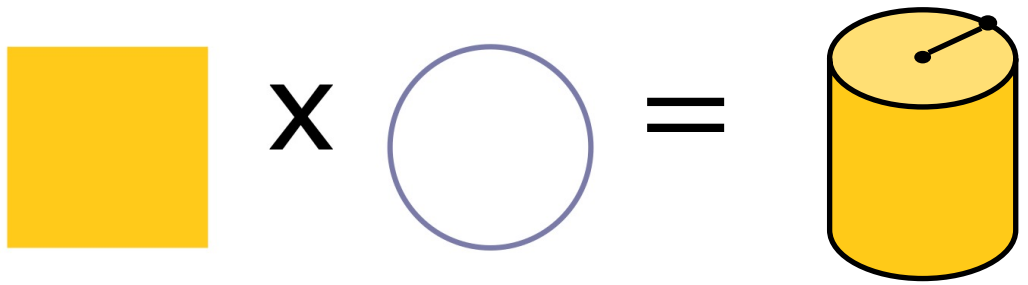


车辆配置空间

- 车辆的配置 (configuration)
  - $q = (x, y, \theta)$?
  - $q = (x, y, u, v)$?
- 车辆的配置空间维度 d
  - DOF = 3
- 配置空间 C (Configuration space or C-space)
  - 柱体 $C = R^2 \times S^1$

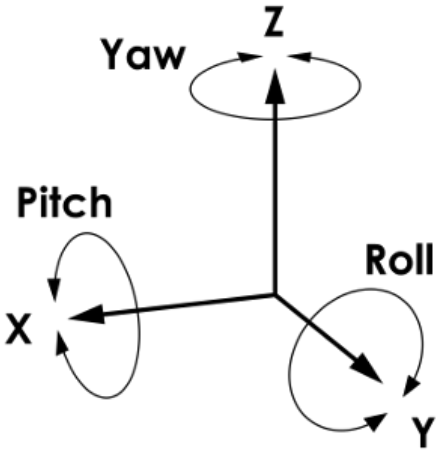






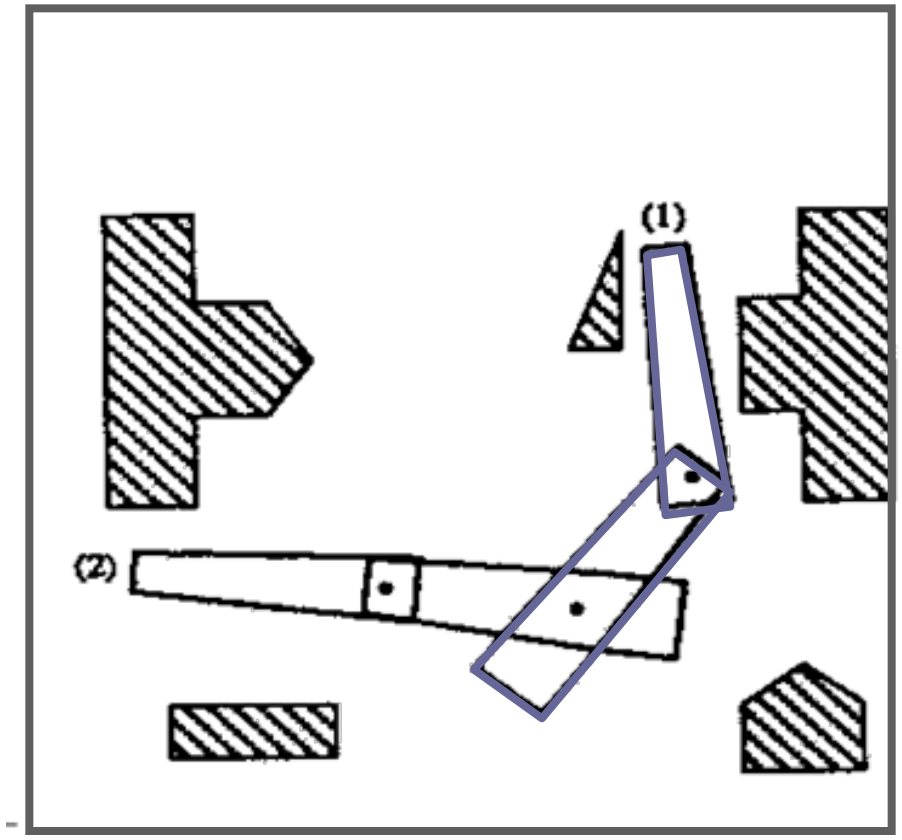
车辆的配置空间（非平坦路面）

- 车辆的配置 (configuration)
  - $q = (x, y, z, \theta_x, \theta_y, \theta_z)$
- 配置空间 C (Configuration space or C-space)
  - 3D刚体变换群：$C = SE(3) = R^3 \times SO(3)$
- 车辆的配置空间维度 d
  - DOF = 6

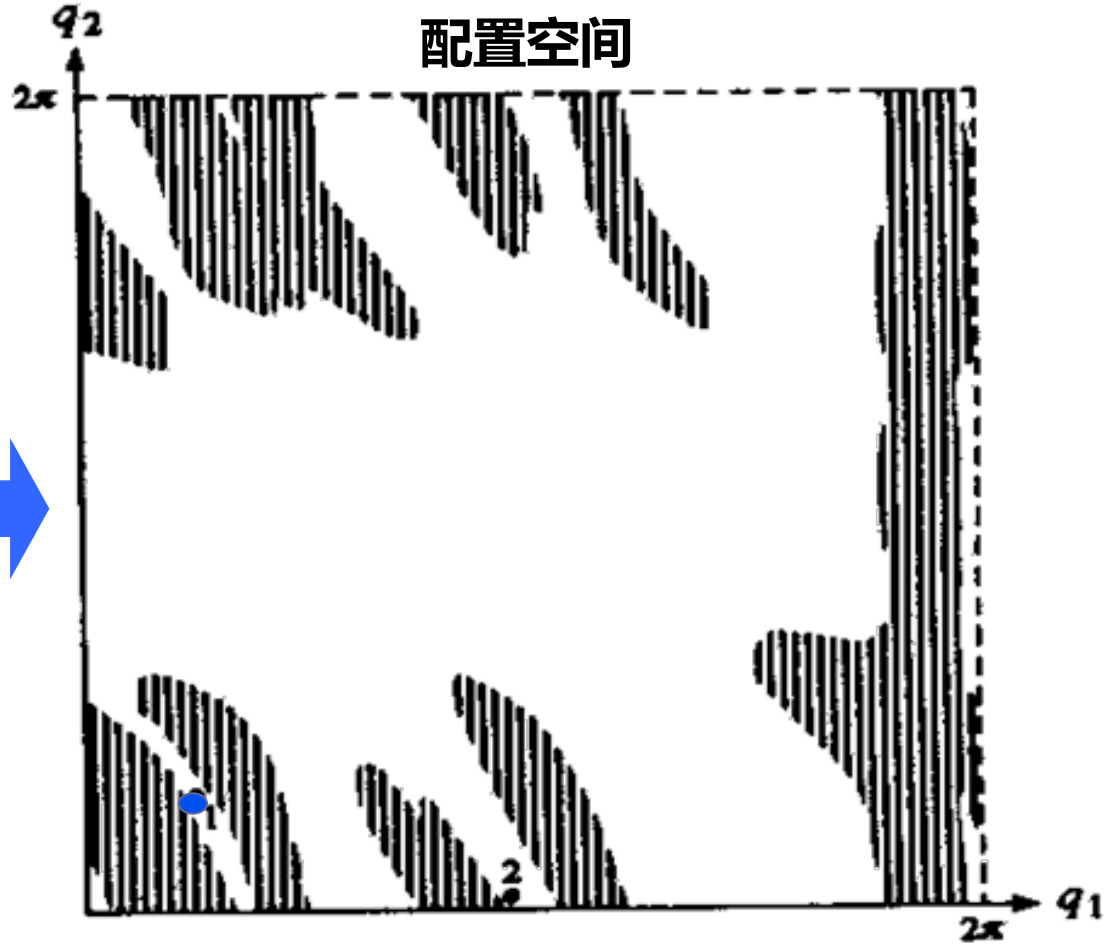


机器人如何成为一个点？

工作空间

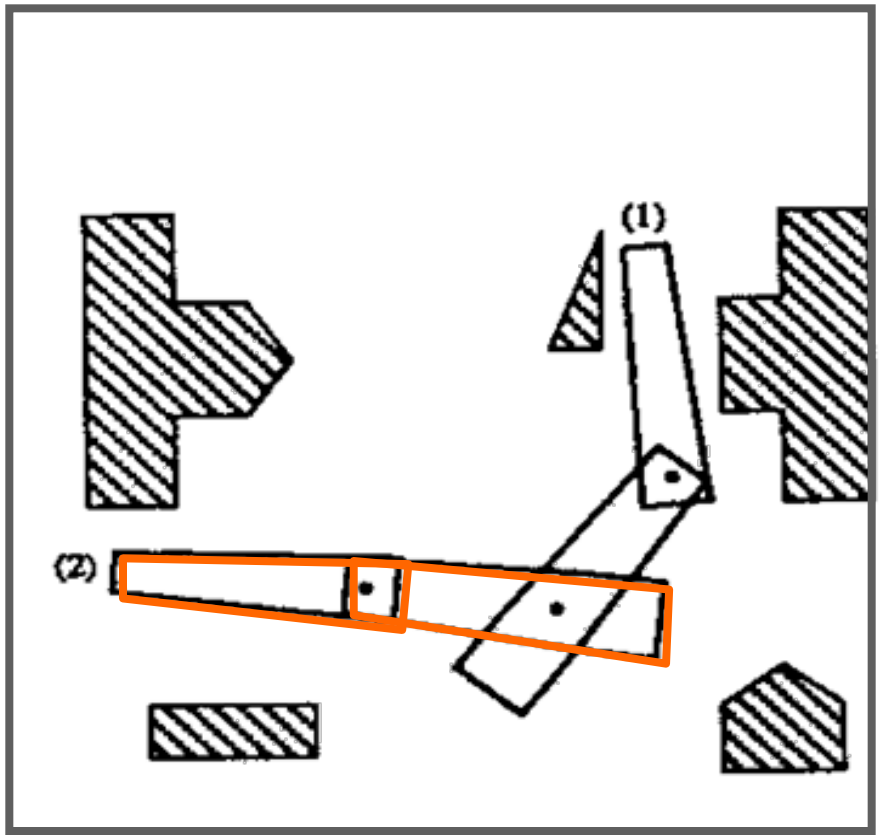


配置空间

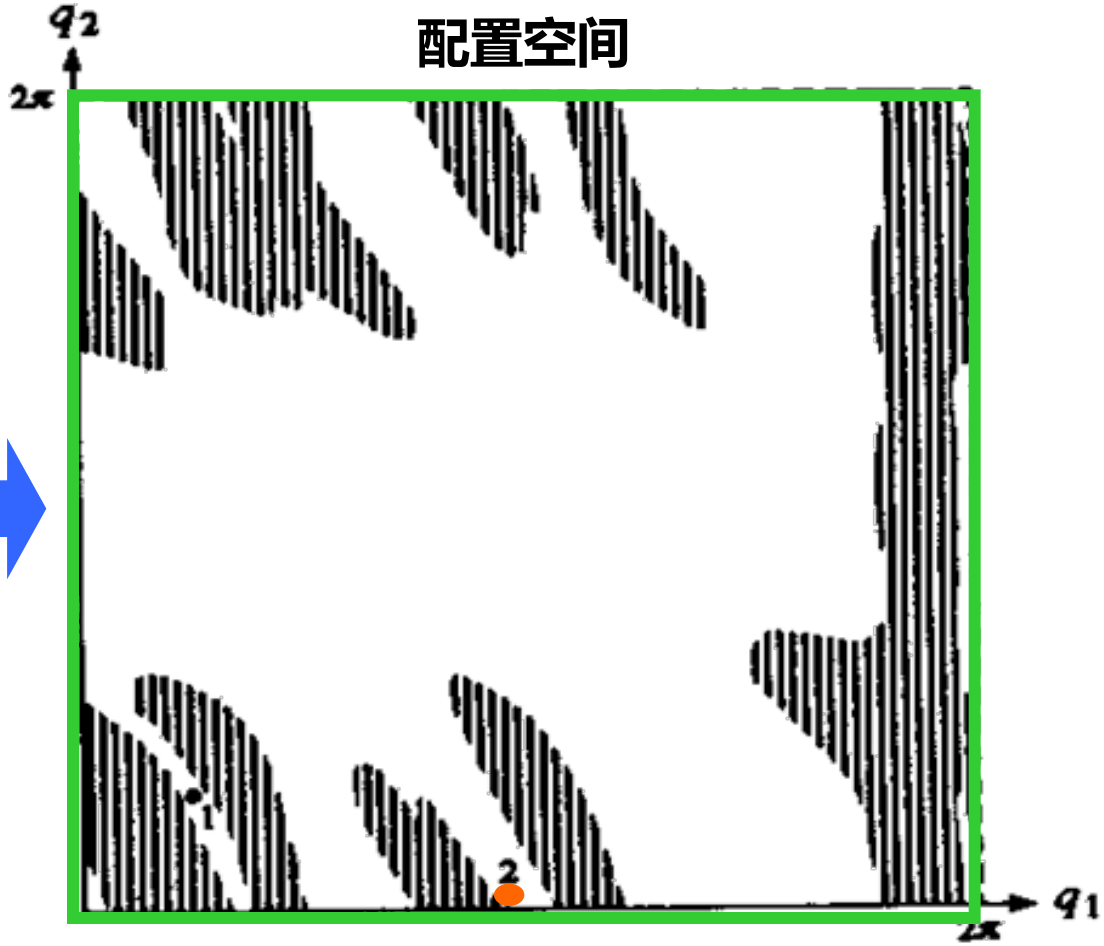


机器人如何成为一个点？

工作空间

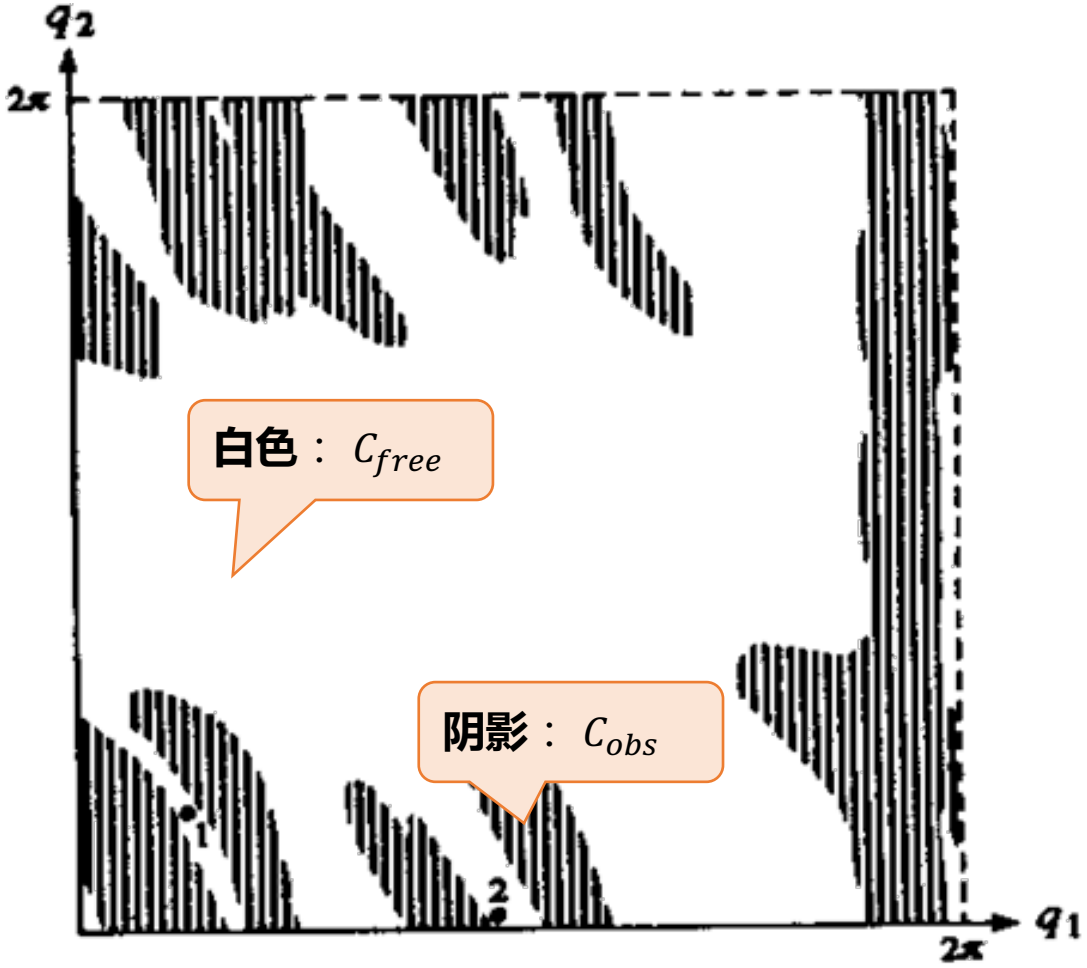


配置空间



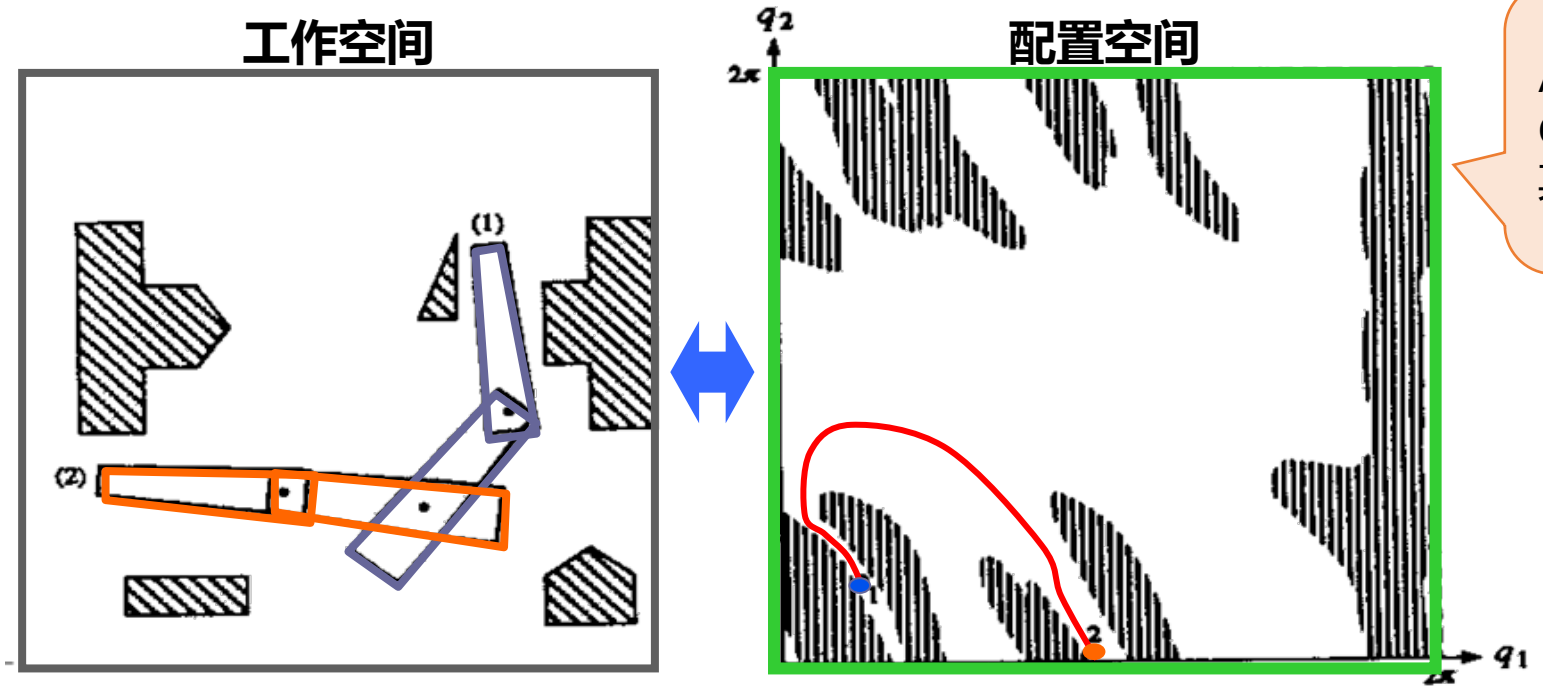
配置空间结构

- 给定一个配置 q，如果处于 q 的机器人不与工作空间中的任何障碍物相碰撞，则q被称为无碰撞的（collision free）
- 自由空间（free space，C_{free}）是所有无碰撞配置（collision-free configuration）的集合
- 配置空间障碍物（C-space obstacles，C_{obs}）是所有发生碰撞的配置集合



运动规划问题的通用定义

- 在 C_{free} 中，给定一个起始配置（start configuration）和终点配置（goal configuration），计算一条完全处于 C_{free} 中的连续路径
  - 输入 $\{q_s, q_g\}$（query）
  - 输出连续变换 $\tau: [0,1] \rightarrow C_{free}$



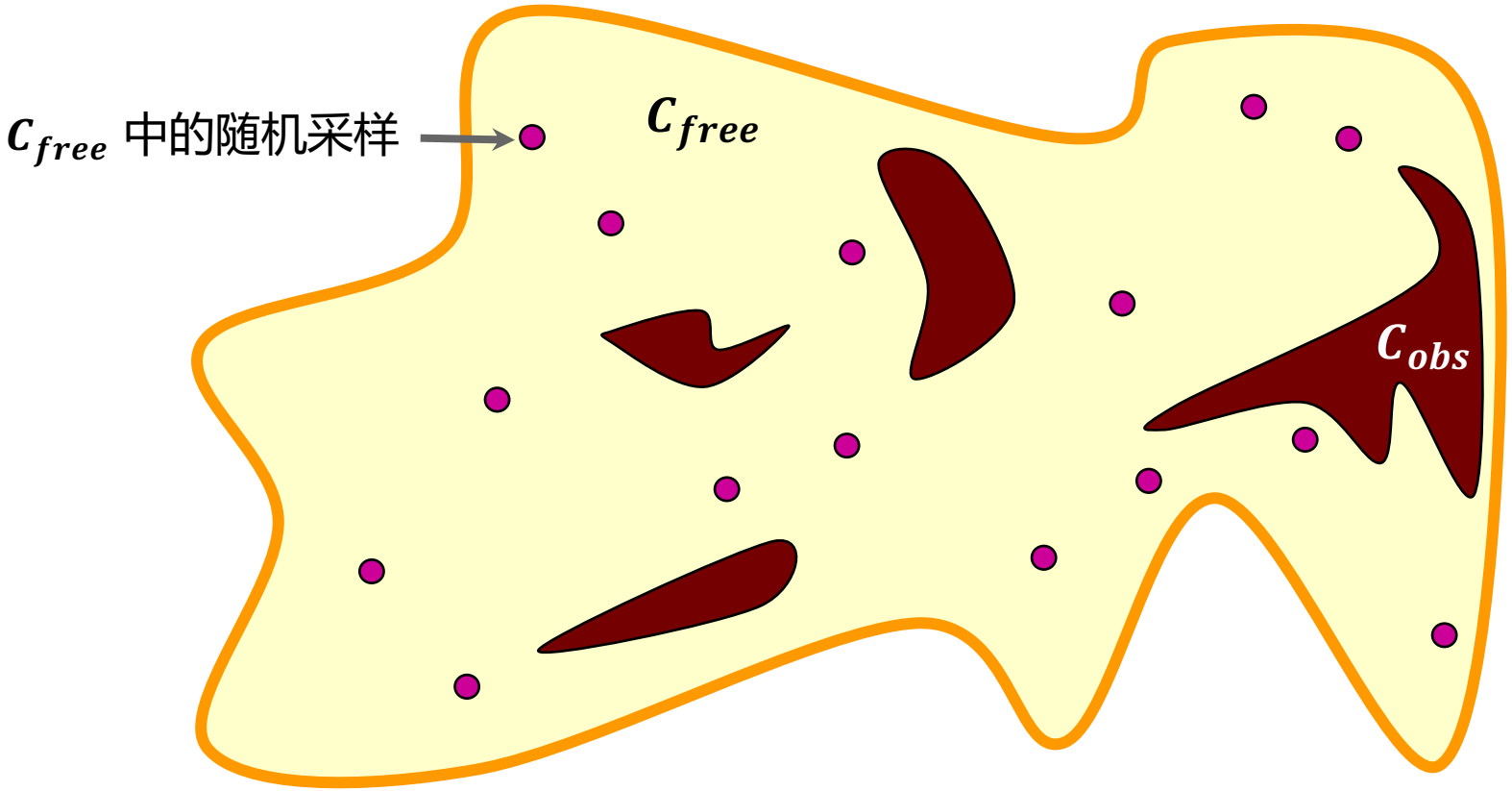
A*, D* Lite, Hybrid A* 都是通过将 C-space 离散化为栅格，然后在其中进行搜索

栅格搜索的计算复杂度

- 栅格搜索的计算时间通常随着**配置空间的维度**呈**指数增长**
- **例**：对于一个 d 维的配置空间，在每个维度上离散为 10 个值，就会有 10^d 个格子。在最坏情况下，需要搜索完所有格子才能达到最优解。
- **例**：在越野飞车的自动驾驶问题中，若将一个 $100m \times 100m \times 20m$ 的场地按照 $0.5m$ 的粒度离散化，将角度按 15 度离散化，则有 2.2×10^{10} 个格子。
- 栅格搜索算法通常不能在超过 3-DOF 的机器人系统中应用

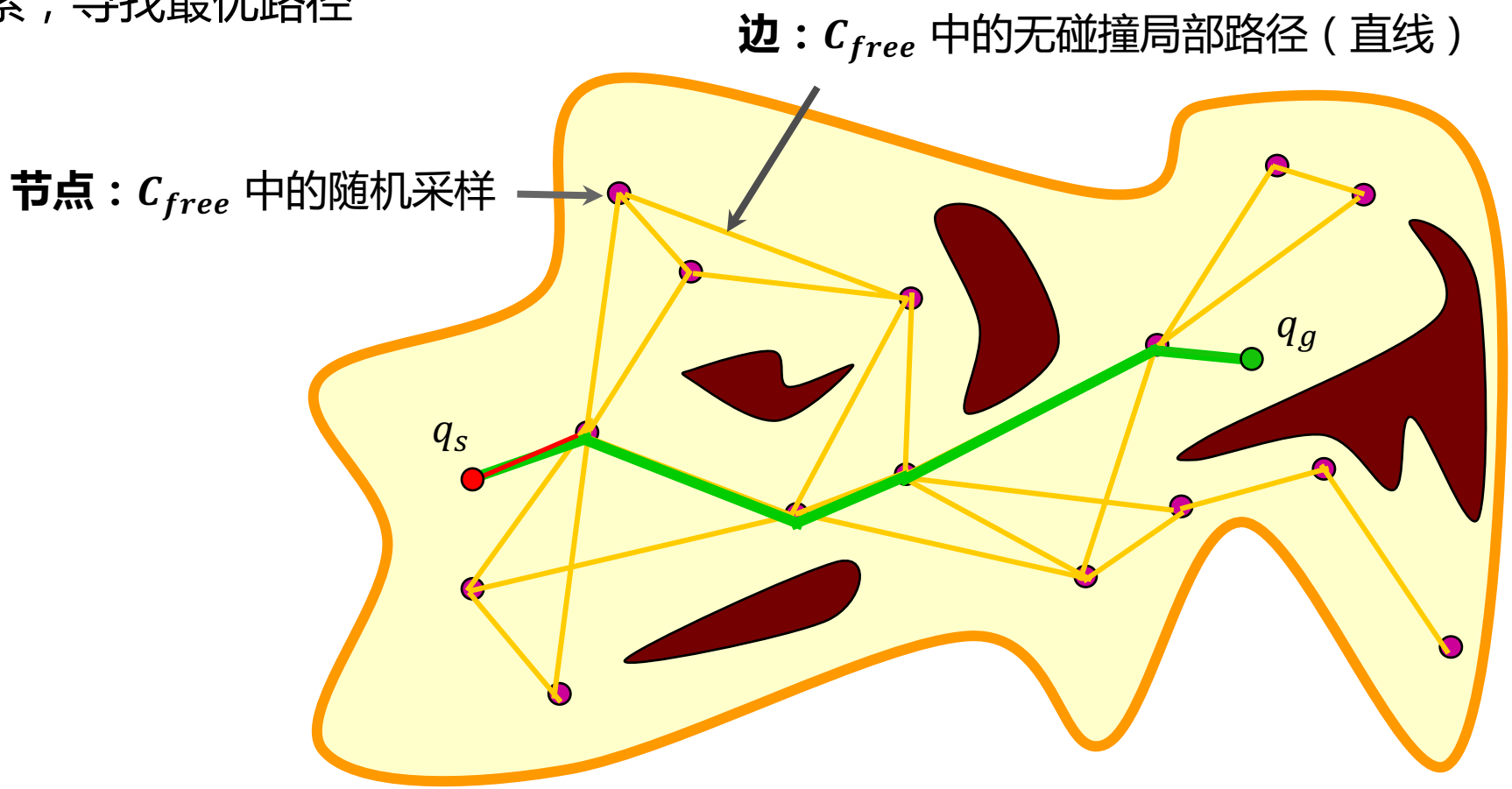
攻克高维空间中的运动规划

- 用随机采样近似表达自由空间（C_{free}）



概率道路图（Probabilistic Roadmap, PRM）

- 将随机采样通过局部路径互相连接，构造一个连通的道路图（roadmap）
- 对于给定 query $\{q_s, q_g\}$，首先将它们连接到PRM
- 在图中进行搜索，寻找最优路径



PRM 算法

```
Input:
    N: the number of roadmap nodes
    geometry of a robot and obstacles
Output:
  roadmap G = (V, E)
BasicPRM
1: V ← ∅ and E ← ∅.
2:  while |V| <= N
3:   q ← a configuration sampled uniformly at random from C.
4:    if Collision-Free(q) = TRUE then
5:        Add q to V.
6:      Nq ← a set of nodes in V that are close to q.
7:      for each q'∈ Nq, in order of increasing distance d(q,q')
8:        if LINK(q',q) = TRUE then
9:          Add an edge between q and q' to E.
10:return G = (V,E)
```

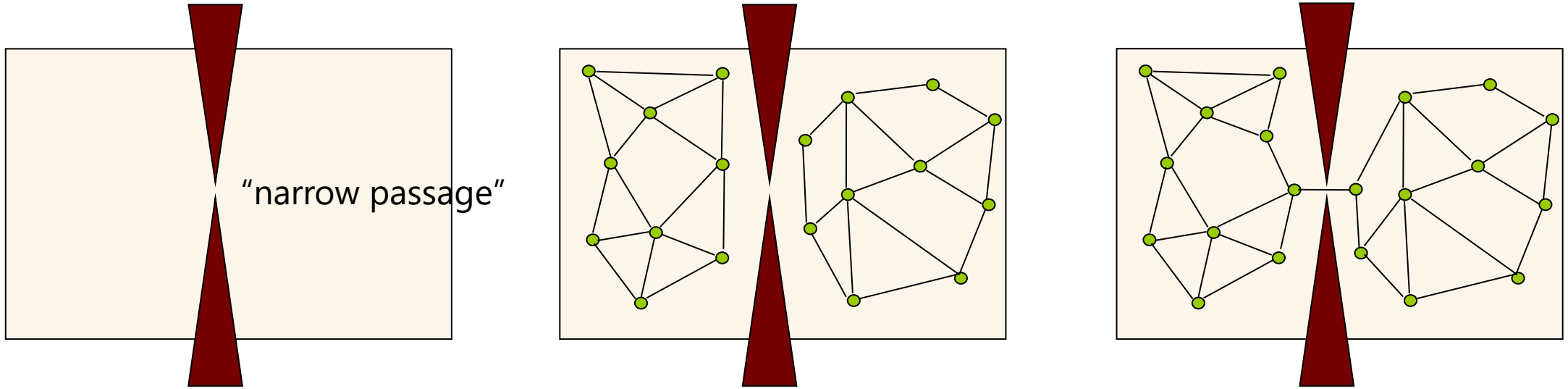
LINK:

- （holonomic）构造q到q'的直线路径
- （nonholonomic）求解边界值问题得到局部路径
- 对局部路径进行碰撞检测

LINK是PRM中最耗时的部分

PRM 的理论性质

- PRM 具有**概率完备性**：
  - 如果解存在，随着计算时间趋近于∞， PRM算法找到解的概率将趋近于1
  - 当有足够多的随机采样时，PRM有高概率找到可行解
- PRM 不擅长窄通道问题（ narrow passage problem ）



PRM + 重要性采样

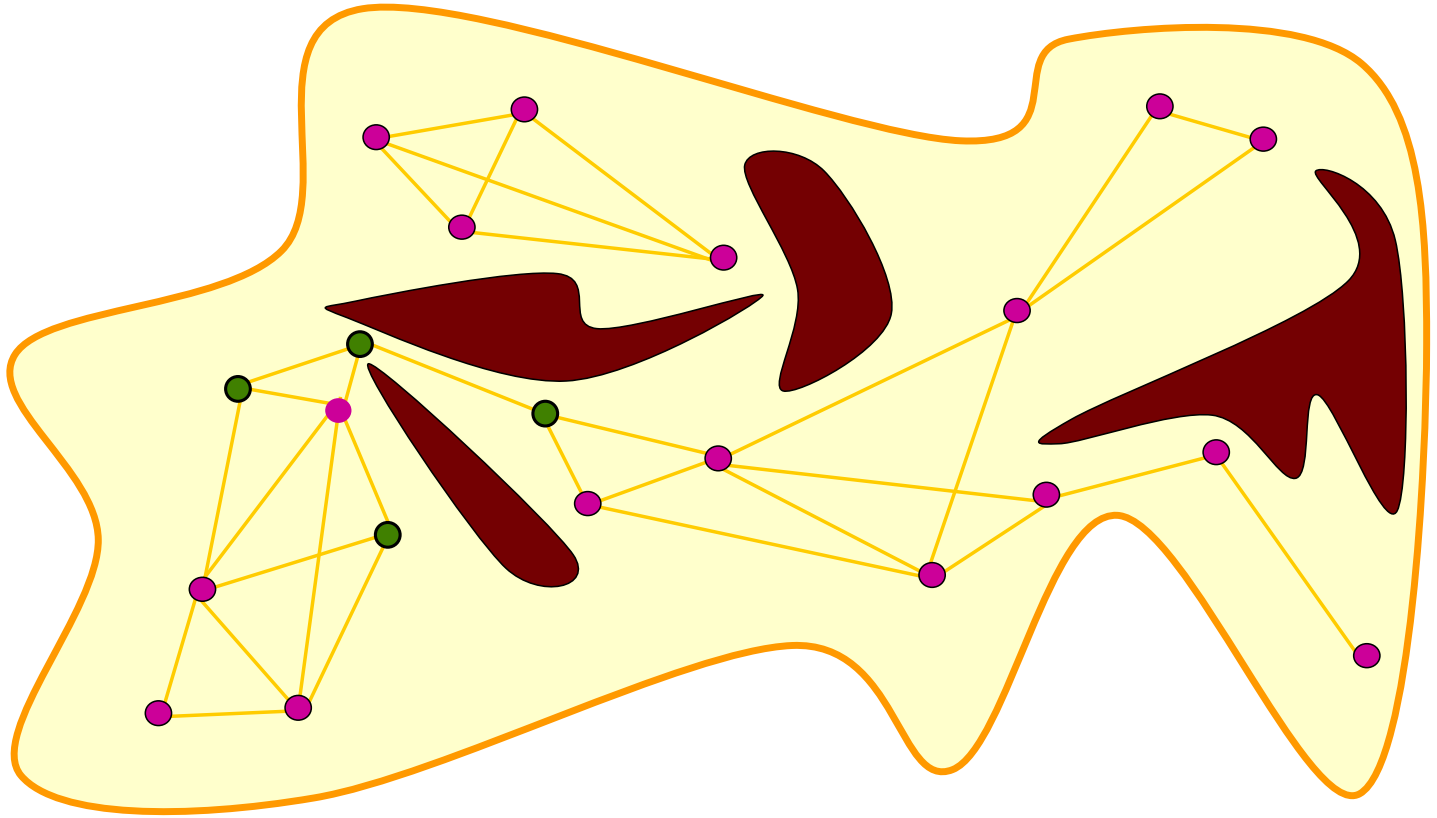
- 在连通性低的“困难区域”进行更多的采样
  - 对于roadmap中的每一个顶点，定义一个重要性权重（importance weight）

$$\pi(q) = \frac{1/(\deg(q)+1)}{\sum_{p \in V} 1/(\deg(p)+1)}$$

    - 其中 deg(q) 是顶点在 roadmap 中的连接度
  - 以概率 $\pi(q)$ 选取节点 q
  - 在节点 q 附近进行随机采样
  - 将新的采样点添加到 PRM

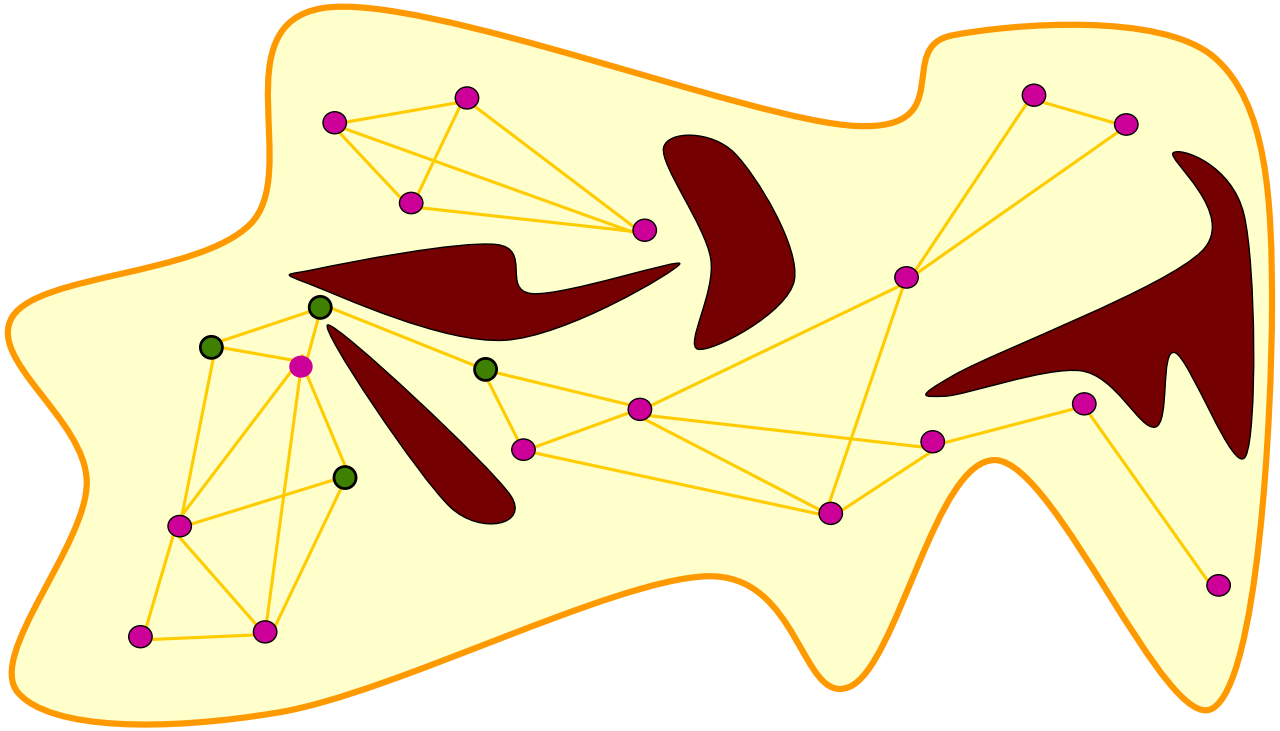
PRM + 重要性采样

- 在连通性低的“困难区域”进行更多的采样



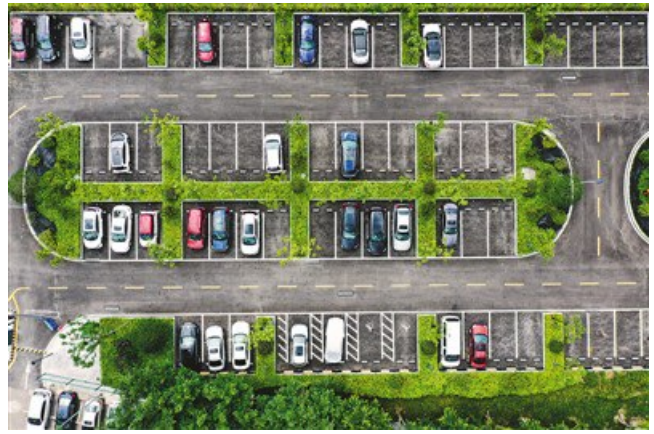
PRM 中的学习元素

- 随机采样-> 数据
- PRM -> 储存经验的容器
- 对于在相同环境的新问题，利用经验快速解决
- 当环境发生局部的变化，可以局部地更新 PRM，适应变化



PRM 适用与不适用场景

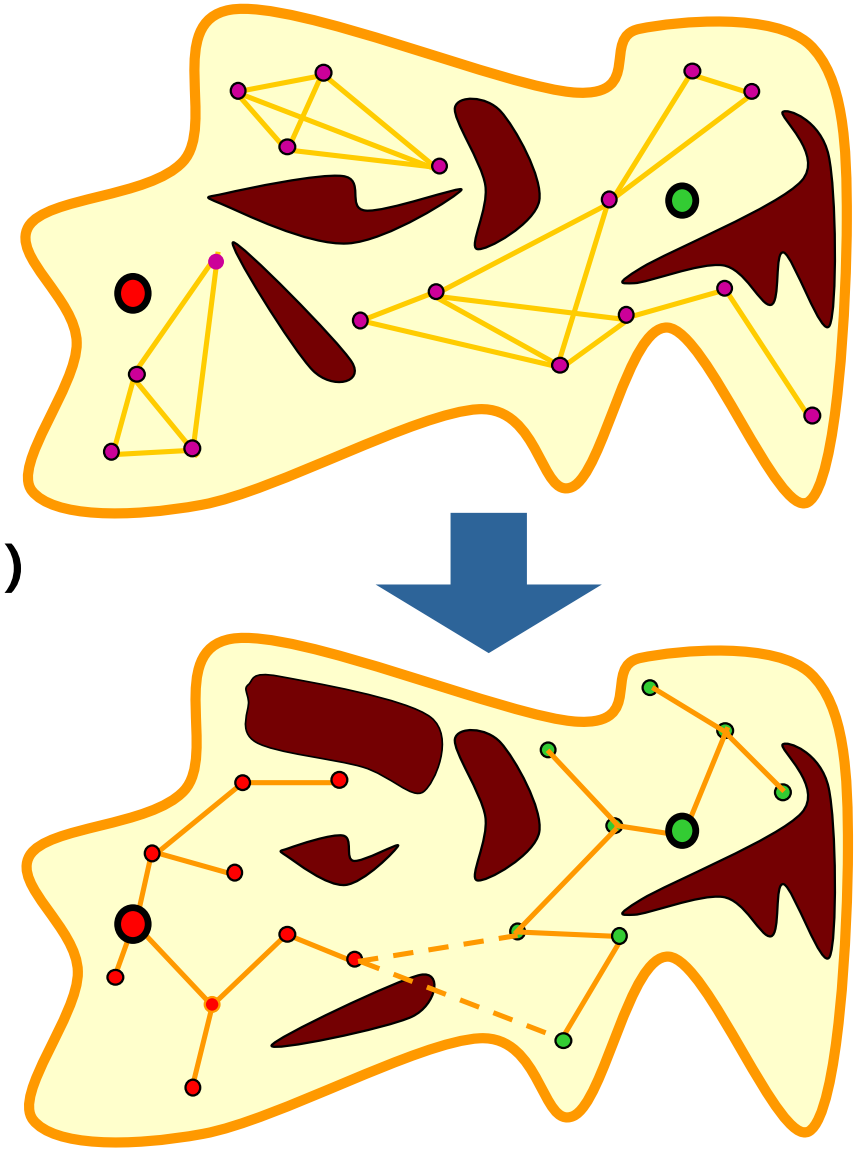
- 当机器人在同一环境中需要进行许多次运动规划时适用
  - **多询问运动规划（Multi-Query Planning）**
- 在变化度高的或有动态障碍物的环境中不适用
  - 机器人每次运动规划面临的环境都不一样
  - PRM 可重用性非常低
  - **单询问运动规划（Single-Query Planning）**



单询问运动规划（Single-Query Planning）

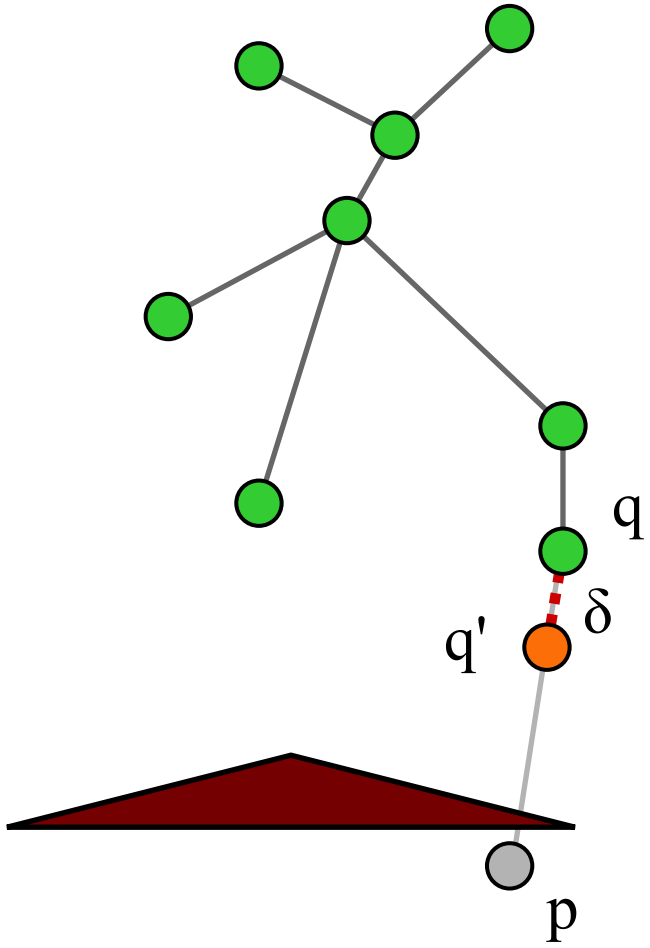
- 只考虑从 q_s（或者 q_g）“可到达”的配置
- 从 q_s 和 q_g 出发，分别扩展一颗树
  - 每次随机采样一个全局配置，扩展其中一颗搜索树
- 不断尝试连接（connect）两棵树，即连通起点与终点

=> 快速探索随机树（Rapidly-exploring Random Tree, RRT）



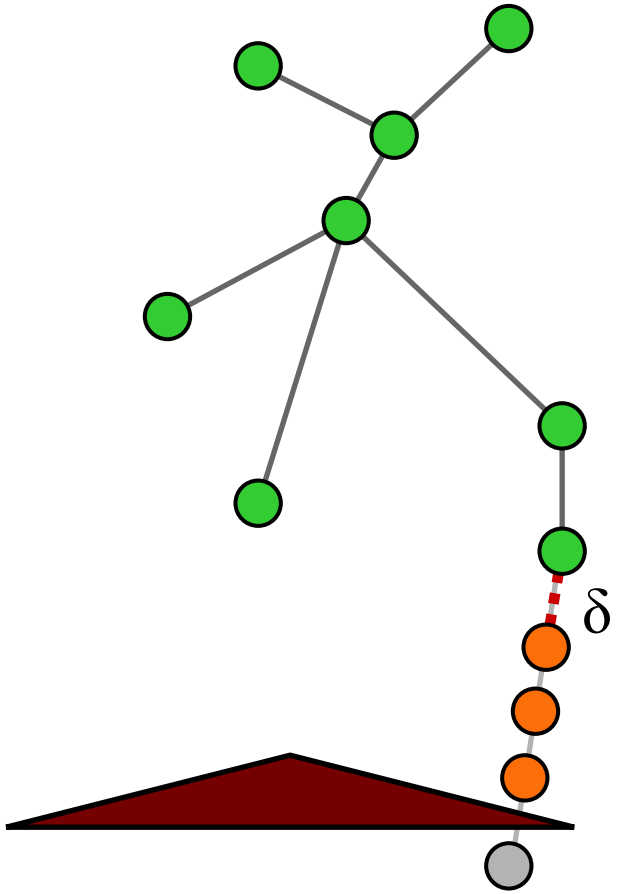
Rapidly-exploring Random Tree（RRT）

- **Extend:** RRT 从一个目标采样分布（Sampling Distribution，比如 C-space 中的均匀分布）中采样机器人配置，用以构建搜索树。在每次迭代中：
  - 随机采样一个无碰撞的配置 p
  - 在树中找到离 p 最近的顶点 q
  - 从顶点 q 出发，沿着向随机配置 p 的直线方向移动 δ 距离，到达新的顶点 q':
    - 若 q 到 q' 的直线路径无碰撞，将 q' 加入搜索树



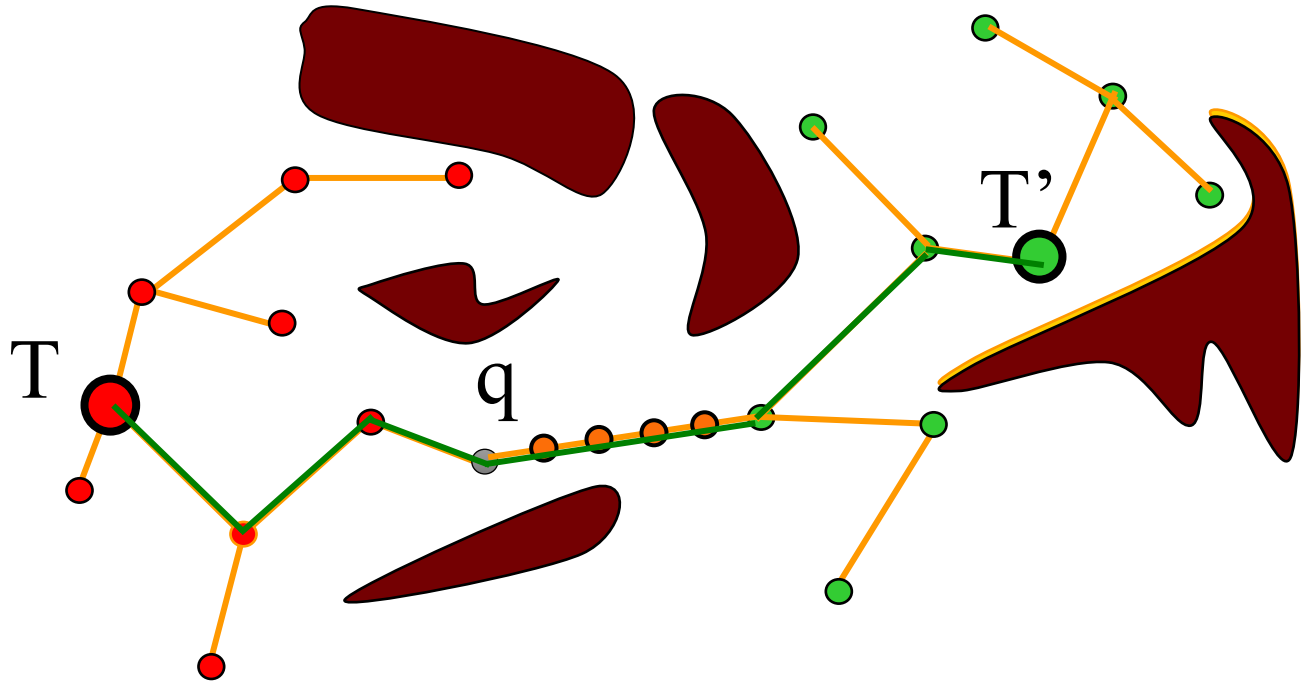
Rapidly-exploring Random Tree（RRT）

- **Greedy extend:** RRT 从一个目标采样分布（Sampling Distribution，比如C-space 中的均匀分布）中采样机器人配置，用以构建搜索树。在每次迭代中：
  - 随机采样一个无碰撞的配置 p
  - 在树中找到离 q 最近的顶点 q
  - 从顶点 q 出发，沿着向随机配置 p 的直线方向移动 δ 距离，到达新的顶点 q':
    - 若 q 到 q' 的直线路径无碰撞，将 q' 加入搜索树
    - **使 $q = q'$，重复以上扩展过程，直至发生碰撞或到达 p**



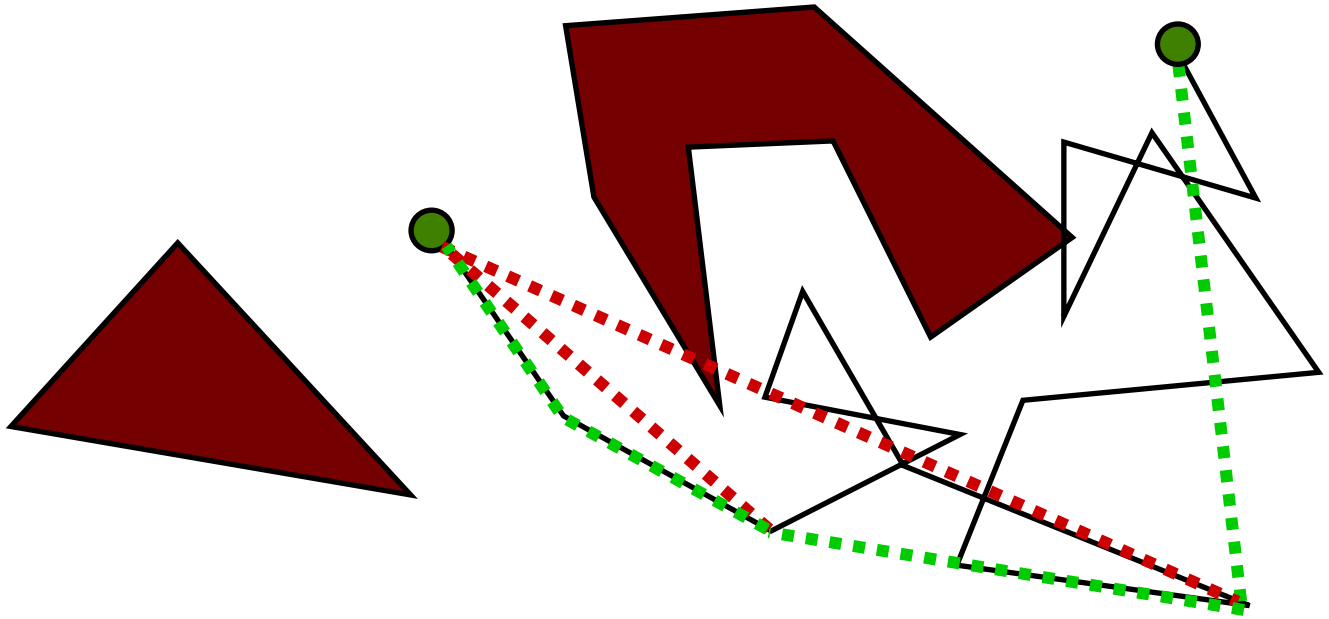
Rapidly-exploring Random Tree（ RRT ）

- **Connect:** 当搜索树 T 每次新扩展顶点 q， RRT 尝试将 q 作为目标，对另一棵树 T' 进行扩展：
  - 若 T' 成功连接到 q 则规划成功，返回连接起点与终点的路径



Rapidly-exploring Random Tree（ RRT ）

- **后处理 (shortcutting):** 沿着路径，在各个顶点不断寻找捷径（ shortcuts ），以将路径变得更平滑：
  - 对于每一个顶点 p_i, 尝试与 $goal$ 和 p_j $(j > i)$ 建立直线连接
  - 若成功，用 (p_i, p_j) 或 $(p_i, goal)$ 替代原本的 $\{p_i, \dots, p_j\}$, 或 $\{p_i, \dots, goal\}$ 路径

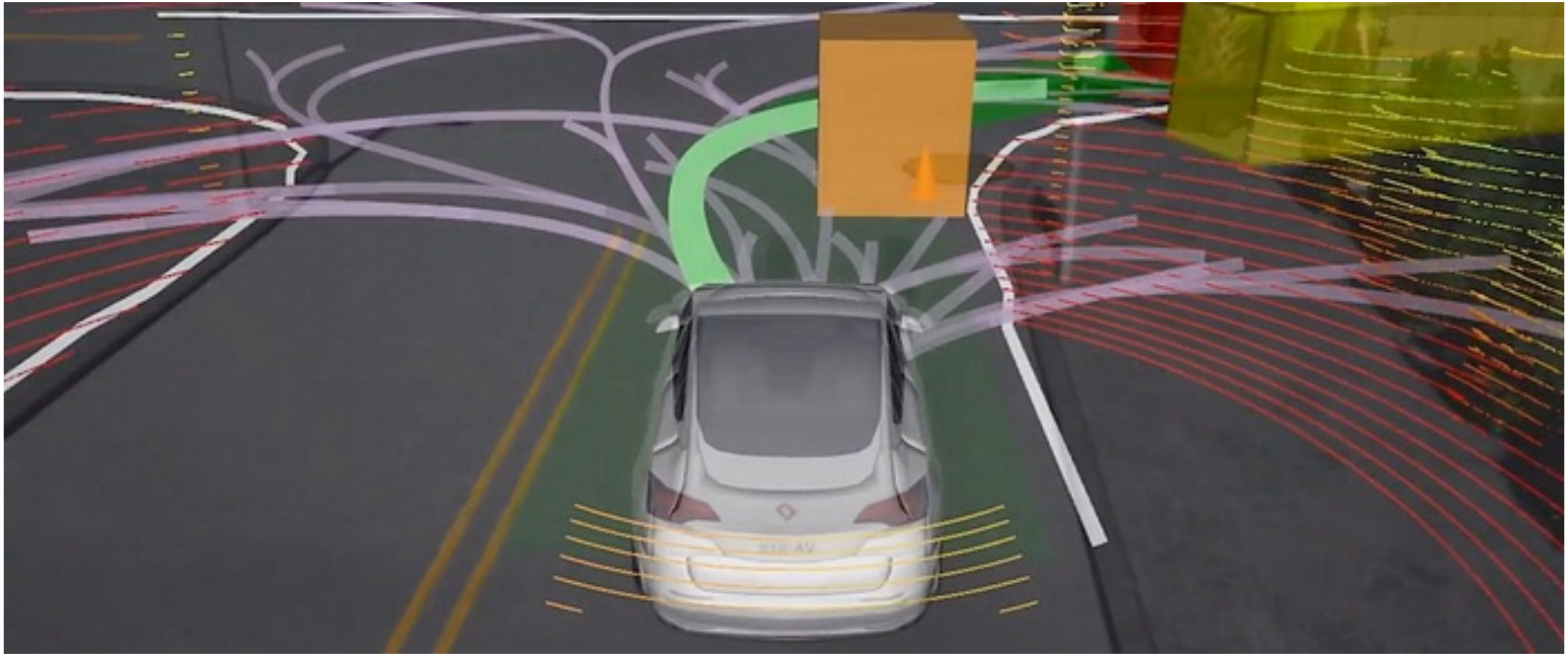


RRT 理论与实用性质

- **理论**：RRT 与 PRM 一样具有概率完备性
  - 如果解存在，随着计算时间趋近于 ∞， RRT 找到解的概率将趋近于 1
  - 当有足够多的随机采样时，RRT 有高概率找到可行解
- **实用**：RRT + shortcutting 通常能迅速地生成高质量路径

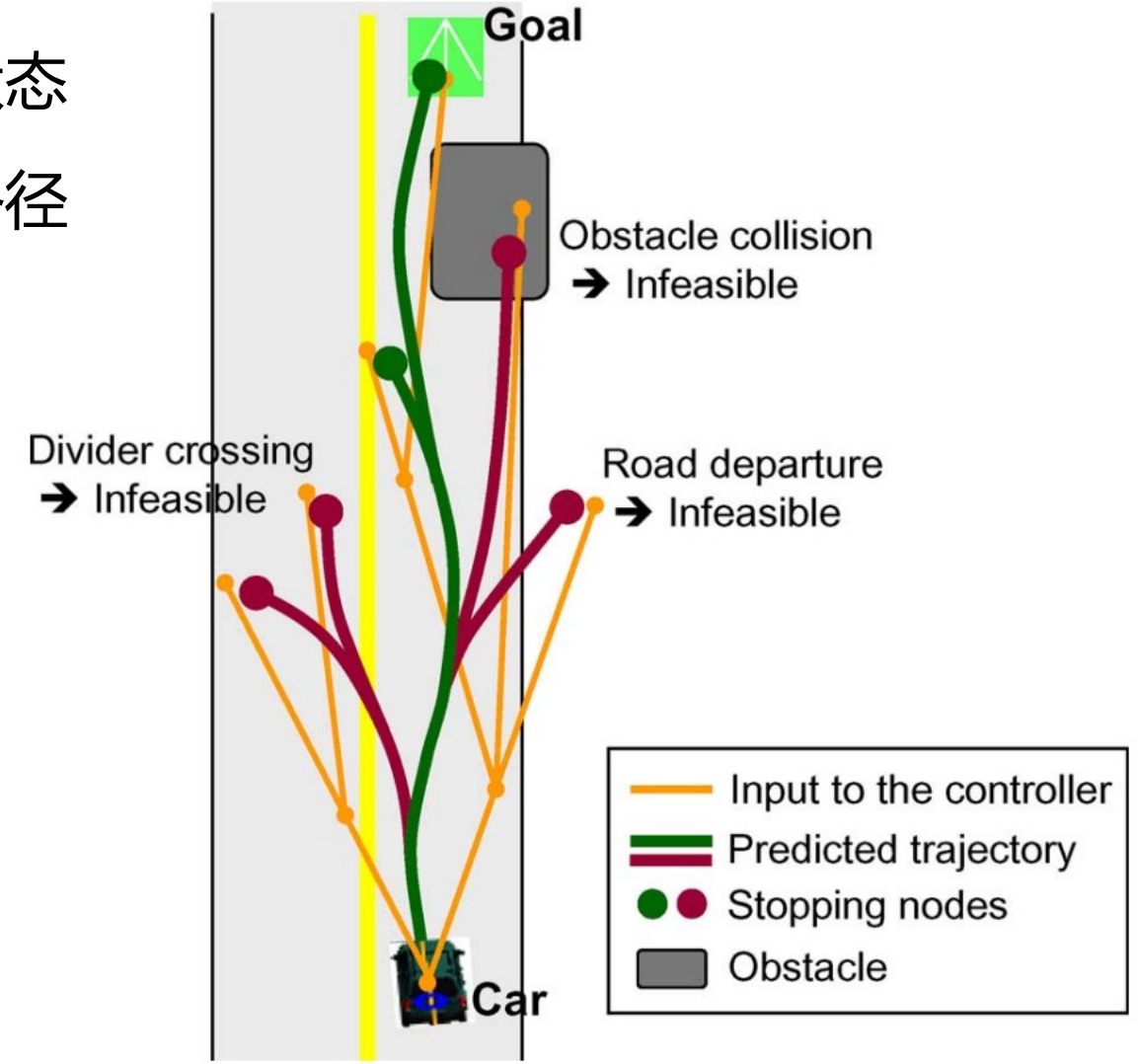
RRT 应用于自动驾驶

- 每条边应当是车辆可以执行的路径/轨迹



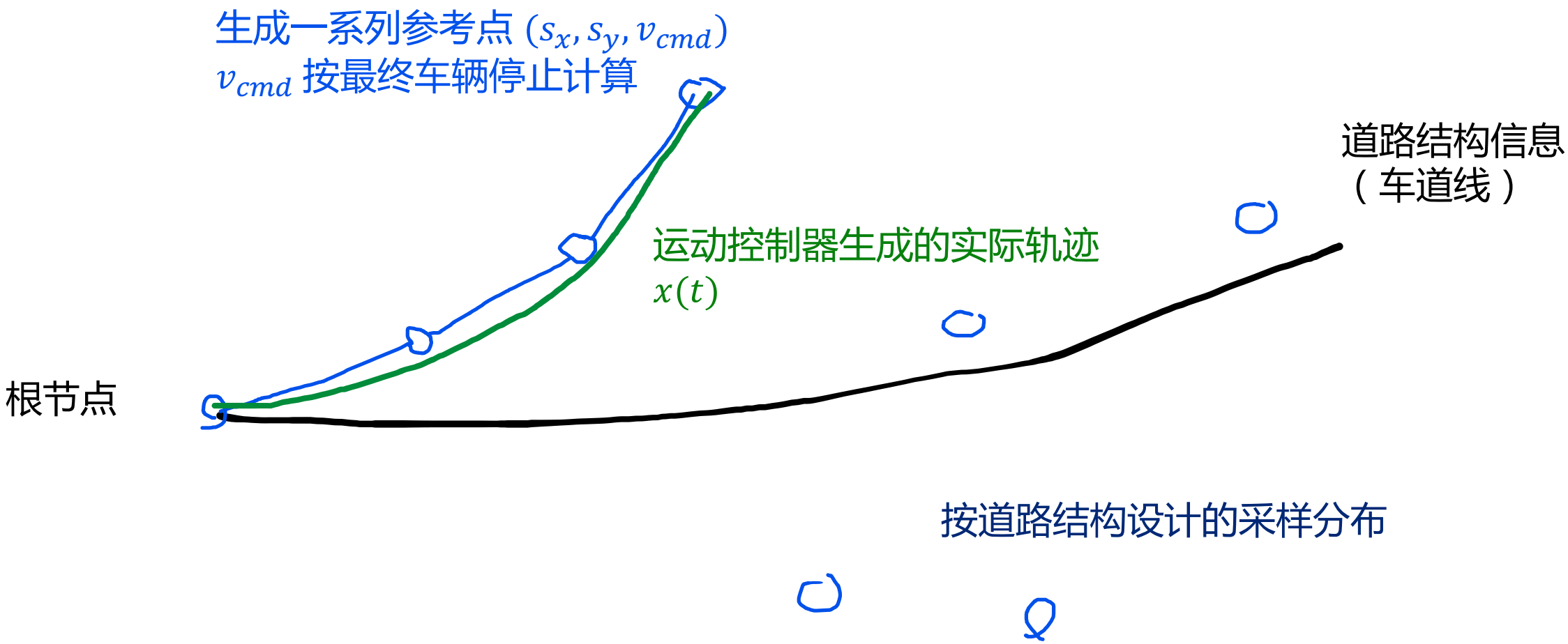
Closed-Loop RRT (CL-RRT)

- 采样控制（转向、速度），而不是采样状态
- 利用底层控制器和动力学模型生成扩展路径
- 直接规划轨迹，而不是路径



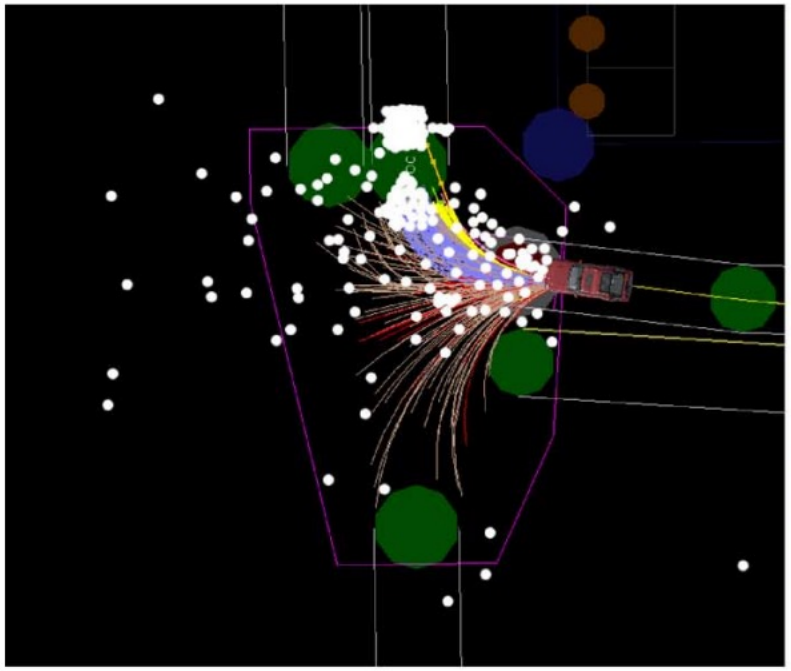
Closed-Loop RRT (CL-RRT)

- **采样策略**：根据道路结构进行偏置采样（ biased sampling ）

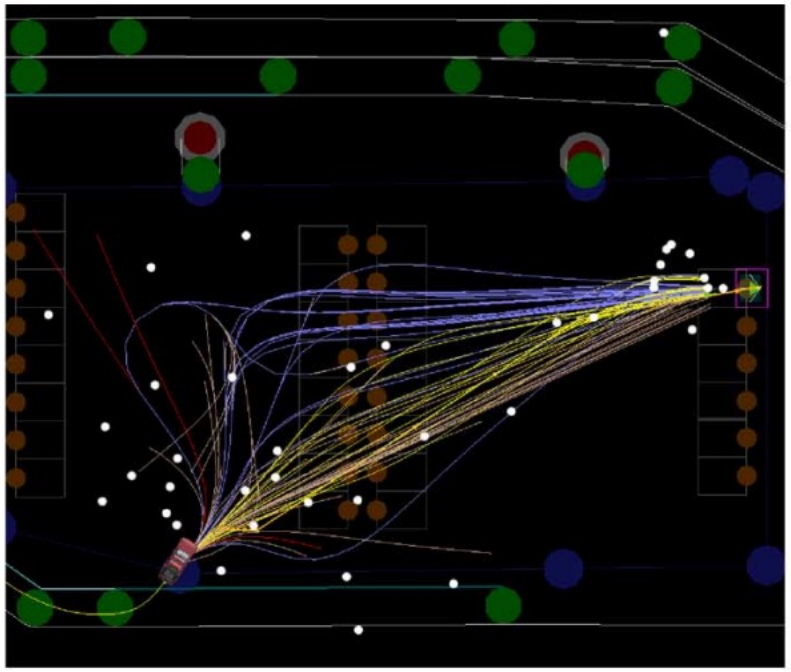


Closed-Loop RRT (CL-RRT)

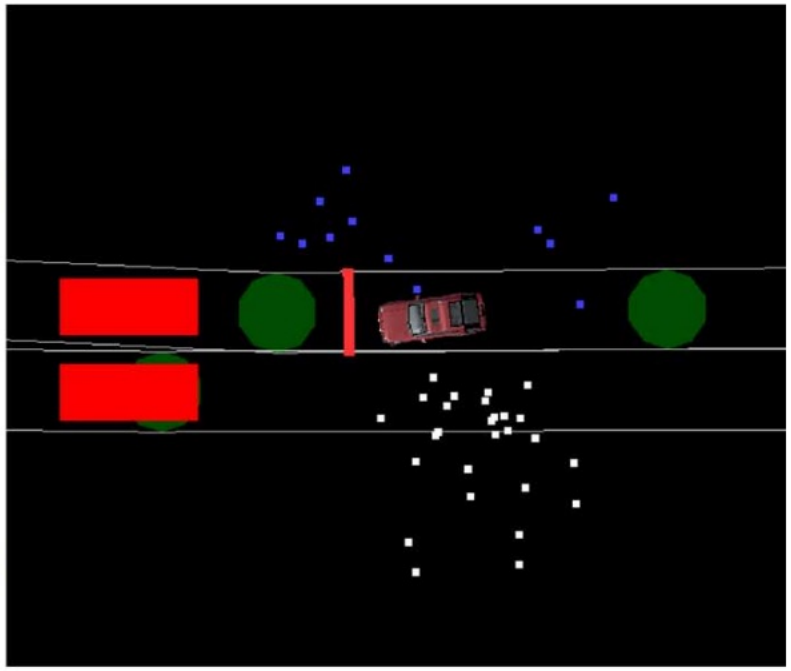
不同场景的采样分布：



十字路口：对整个路口区域用高斯分布覆盖



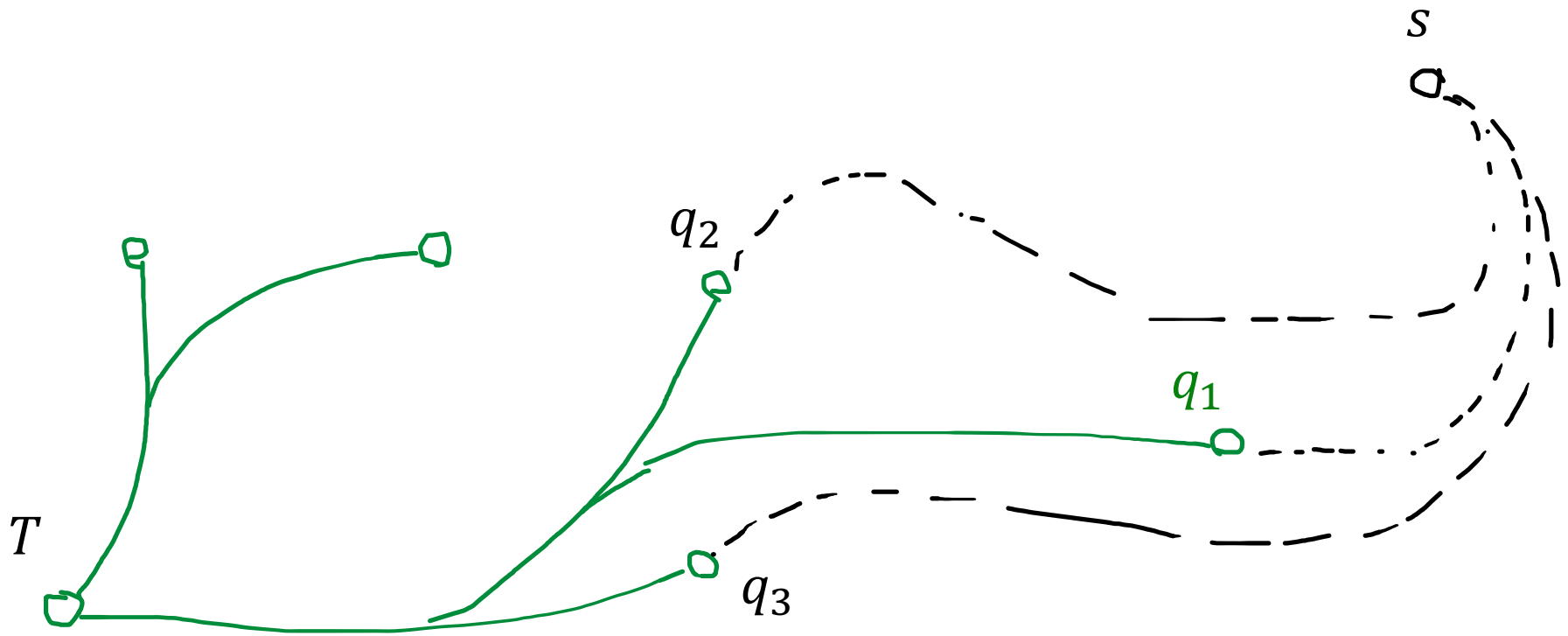
停车场：对车的前方区域与停车点的前方区域用高斯分布覆盖



单行道掉头：按照三点掉头法则，分阶段设计采样分布

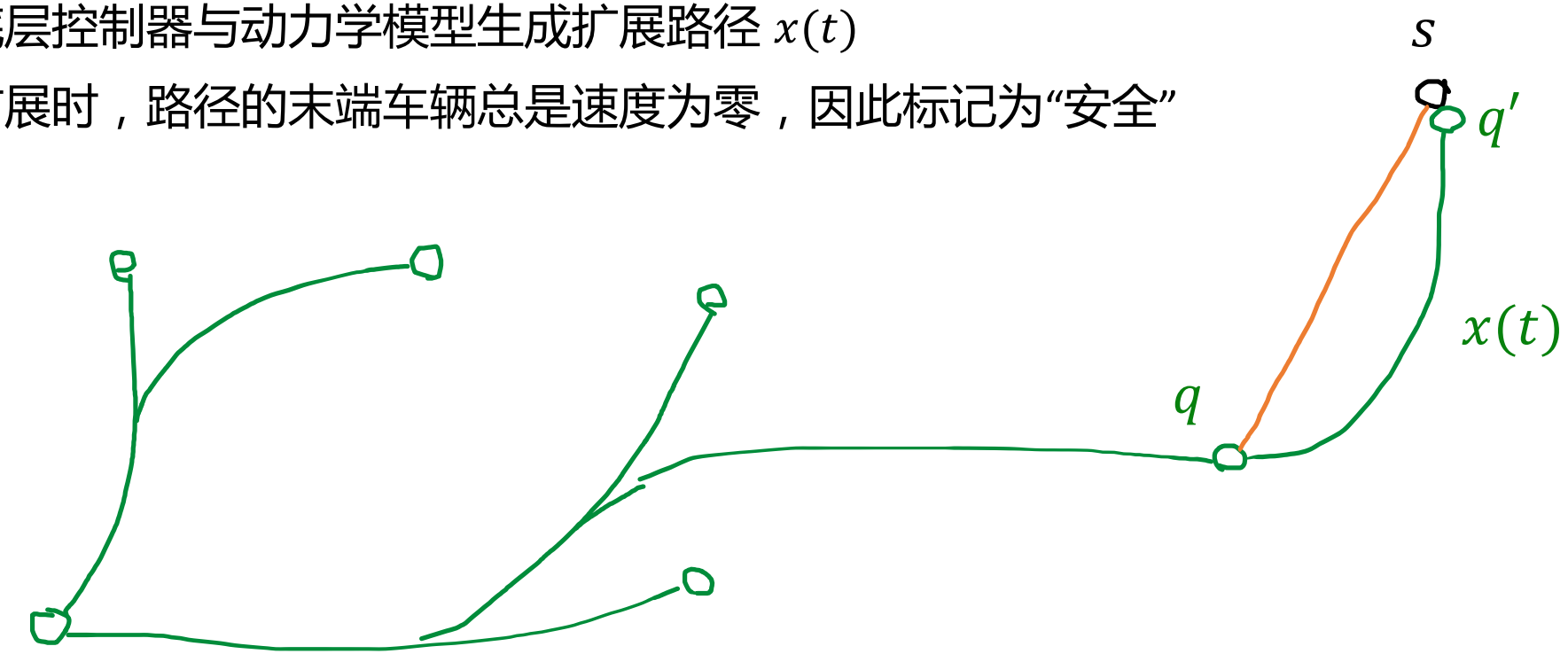
Closed-Loop RRT (CL-RRT)

- **扩展策略**：给定一个采样，根据 Dubins curve 的长度计算与树中顶点的距离，并给顶点排序



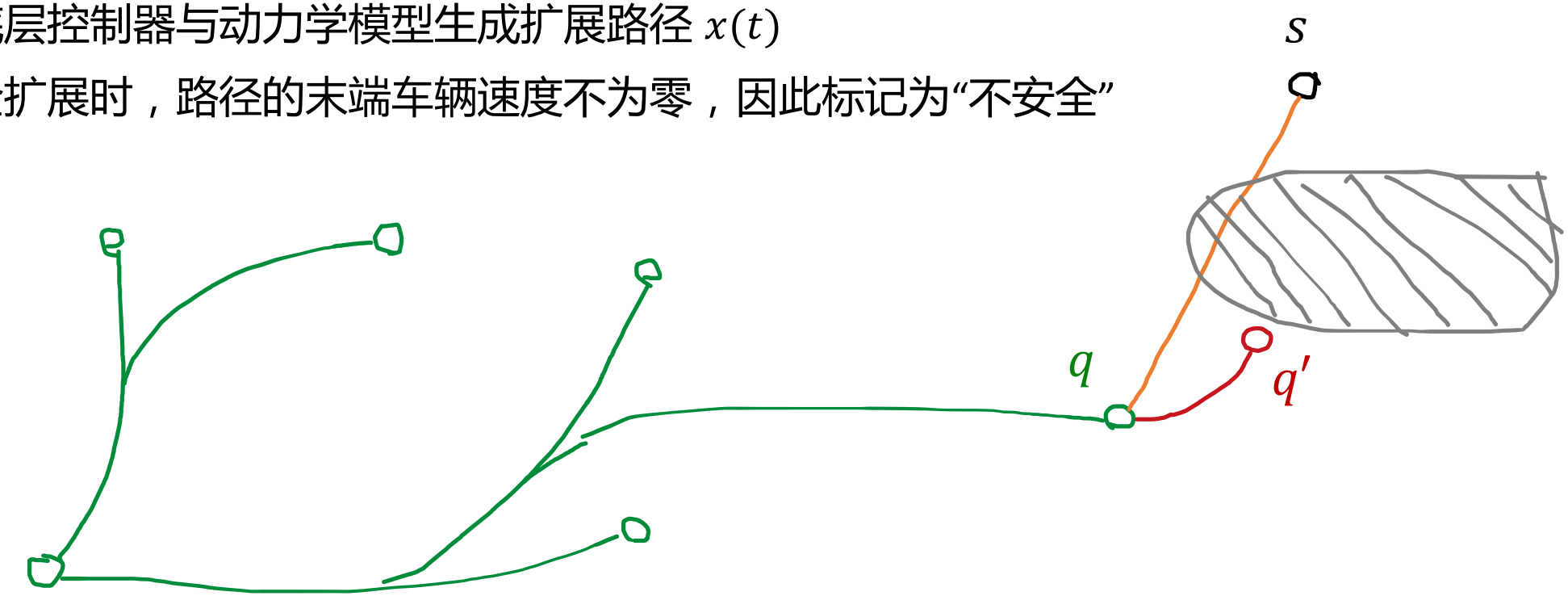
Closed-Loop RRT (CL-RRT)

- **扩展策略**：优先扩展最近距离的顶点
  - 通过连接 q 和 s 构造参考轨迹
  - 利用底层控制器与动力学模型生成扩展路径 $x(t)$
  - 完全扩展时，路径的末端车辆总是速度为零，因此标记为“安全”



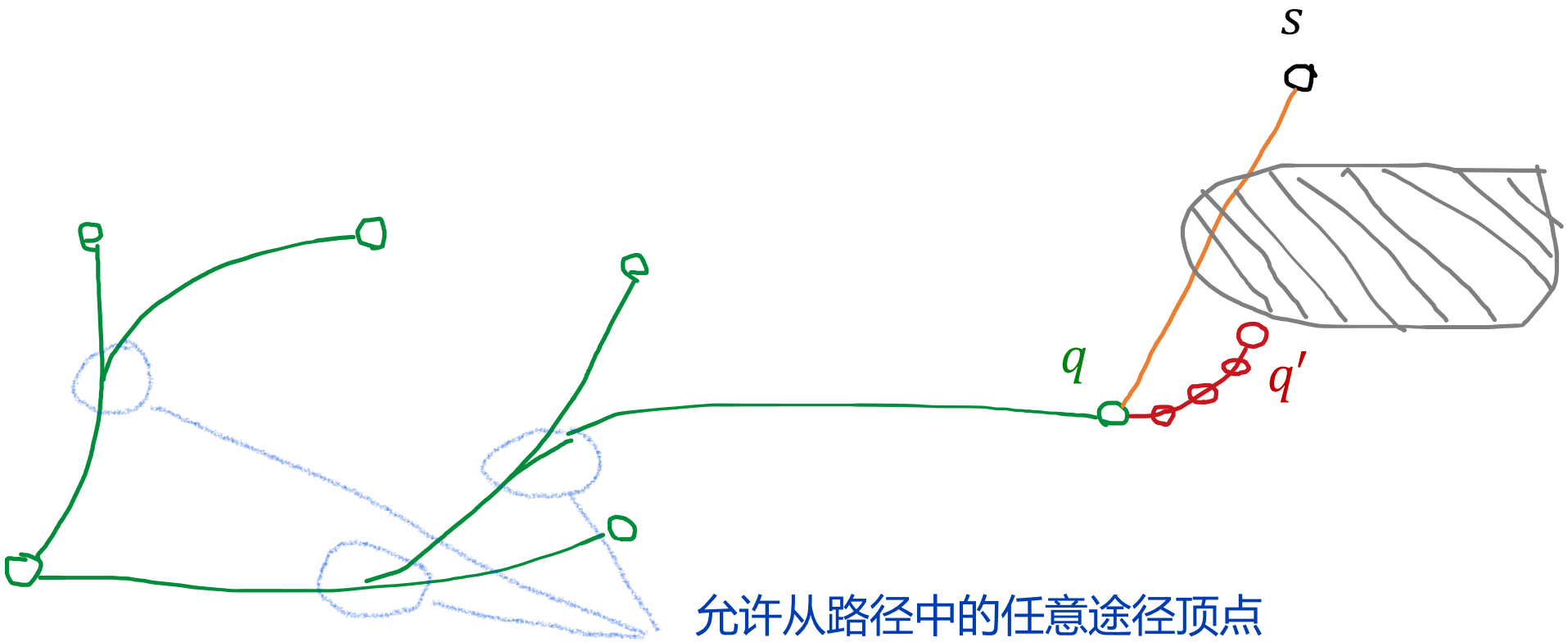
Closed-Loop RRT (CL-RRT)

- **扩展策略**：优先扩展最近距离的顶点
  - 通过连接 q 和 s 构造参考轨迹
  - 利用底层控制器与动力学模型生成扩展路径 $x(t)$
  - 不完全扩展时，路径的末端车辆速度不为零，因此标记为“不安全”



Closed-Loop RRT (CL-RRT)

- **扩展策略**：每次扩展时，会添加一串途径顶点，以允许从途径点扩展路径



Closed-Loop RRT (CL-RRT)

执行逻辑伪代码：

Algorithm 2 Execution loop of RRT.

1: **repeat**
2: Update the current vehicle states $\boldsymbol{x}_0$ and environmental constraints $\mathcal{X}_{\text{free}}(t)$
3: Propagate the states by the computation time and obtain $\boldsymbol{x}(t_0 + \Delta t)$
4: **repeat**
5: Expand_tree()
6: **until** time limit Δt is reached
7: Choose the best safe node sequence in the tree
8: **if** No such sequence exists **then**
9: Send E-Stop to controller and goto line 2
10: **end if**
11: Repropagate from the latest states $\boldsymbol{x}(t_0 + \Delta t)$ using the $\boldsymbol{r}$ associated with the best node sequence, and obtain $\boldsymbol{x}(t),\ t \in [t_1, t_2]$
12: **if** $\boldsymbol{x}(t) \in X_{\text{free}}(t)\ \forall t \in [t_1, t_2]$ **then**
13: Send the best reference path $\boldsymbol{r}$ to controller
14: **else**
15: Remove the infeasible portion and its children from the tree, and goto line 7
16: **end if**
17: **until** the vehicle reaches goal

预测下一时间步的状态（interleaving planning and execution）

RRT 树搜索

若有解，选择最优的"安全"路径

若无解，选择紧急停车

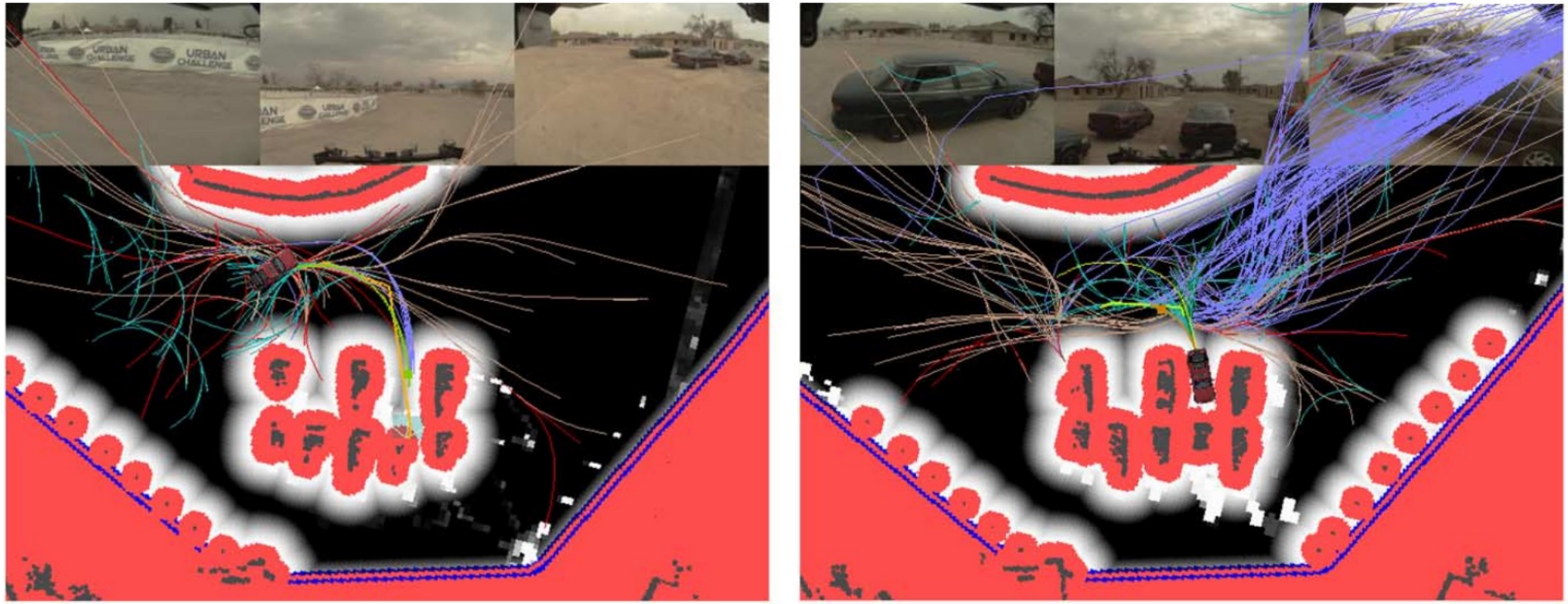
模拟最优控制序列，得到参考轨迹 x(t)

若无碰撞，给运动控制器执行

若有碰撞，在树中找其他最优路径 ***(why?)***

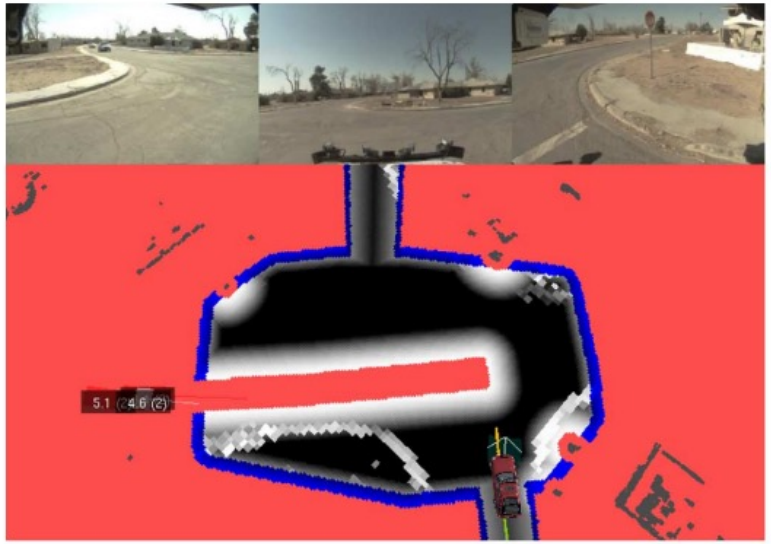
直到车辆到达终点

CL-RRT 效果演示 – 停车

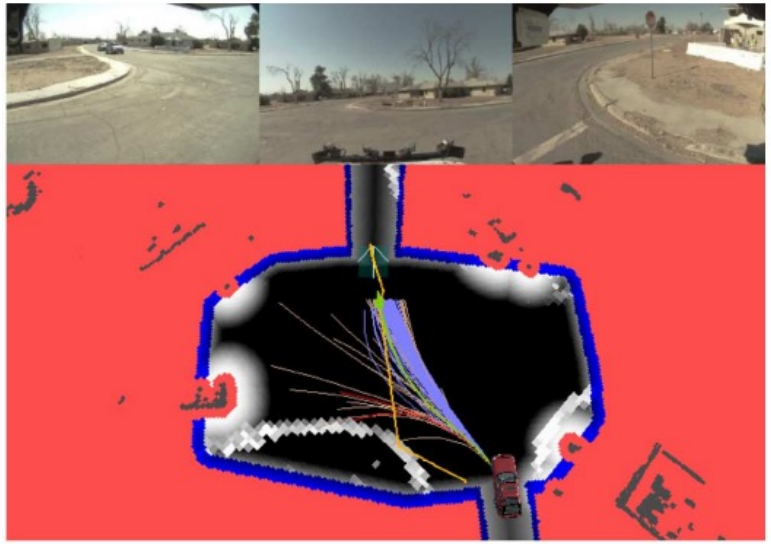


紫色：到达goal的前向轨迹
浅棕色：未到达goal的前向轨迹
蓝绿色：倒车轨迹
红色：不安全轨迹
绿色：最优轨迹

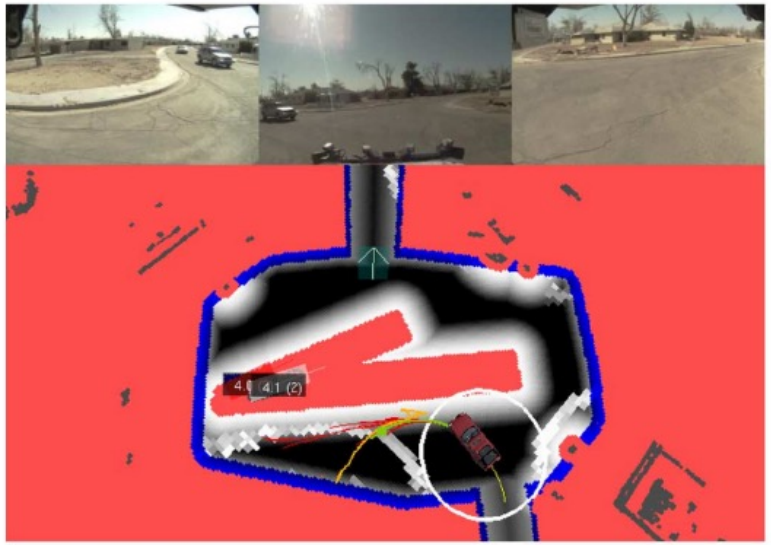
CL-RRT 效果演示 – 十字路口



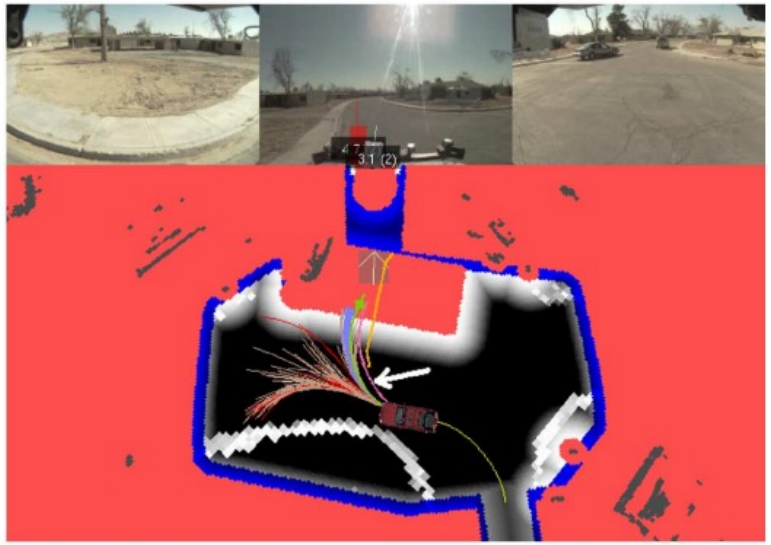
(a) Odin approaches the intersection, Talos stopped at t_0.



(b) Odin stops at intersection, Talos starts maneuver at t_1.



(c) Odin moves into intersection, Talos detected potential collision and executes avoidance maneuver at t_2.



(d) Odin clears intersection, Talos proceeds through intersection at t_3.

紫色：到达goal的前向轨迹
浅棕色：未到达goal的前向轨迹
蓝绿色：倒车轨迹
红色：不安全轨迹
绿色：最优轨迹

第九讲总结

- 自动驾驶路径规划算法
  - Hybrid A*
  - Conformal lattice planner
- 动态环境中的规划
  - 运动预测
  - 时空搜索
- 自动驾驶速度曲线生成
  - 目标速度分布
  - 线性、梯形
- 基于采样的运动规划算法
  - PRM
  - RRT
  - 自动驾驶：CL-RRT

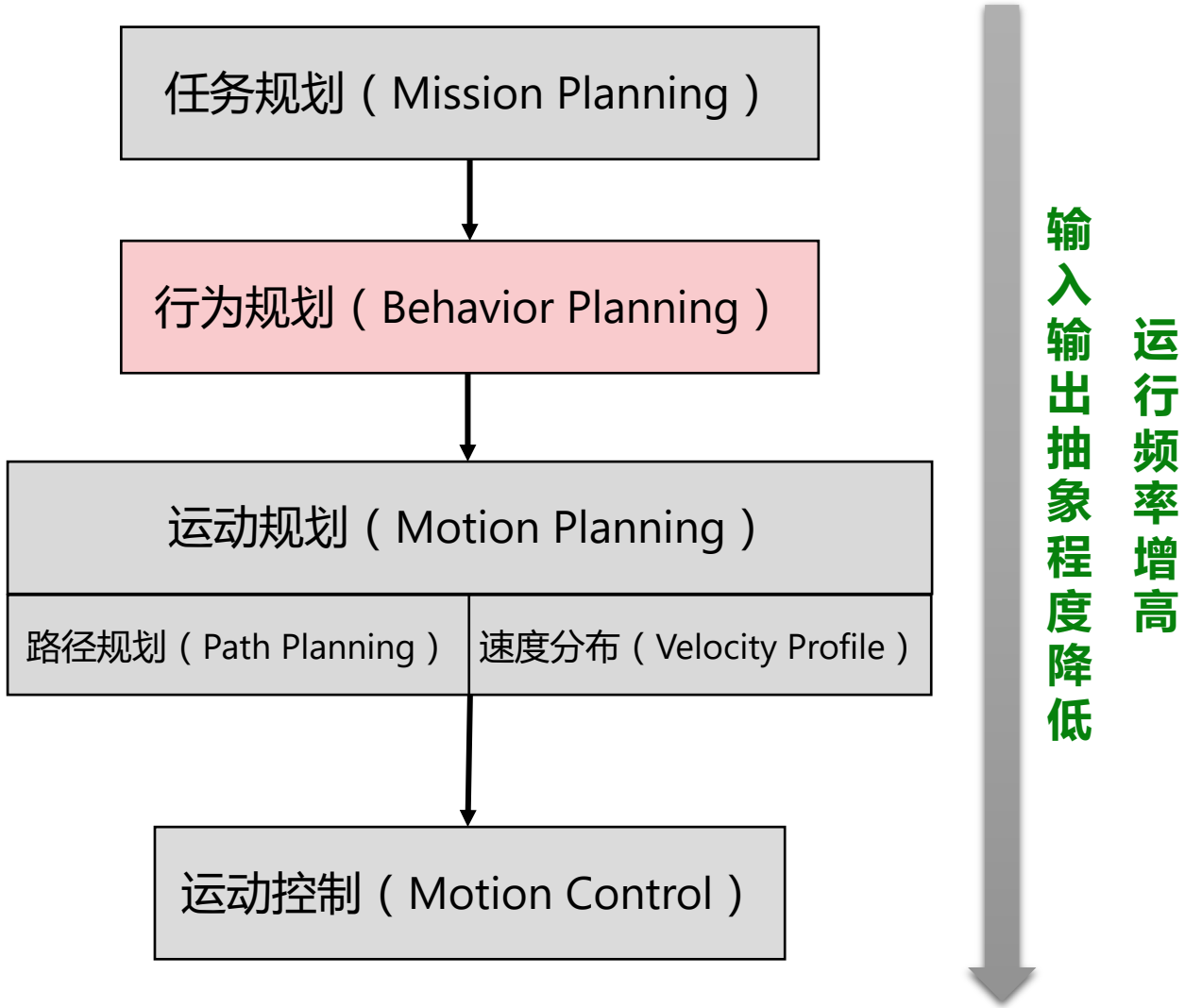
第九讲扩展阅读

- **Conformal Lattice Planner**：M. Mcnaughton, C. Urmson, J. M. Dolan, and J.-W. Lee, “Motion planning for autonomous driving with a conformal spatiotemporal lattice,” 2011 IEEE International Conference on Robotics and Automation, 2011.
- **螺旋线求解边界值问题**：A. Kelly and B. Nagy, “Reactive Nonholonomic Trajectory Generation via Parametric Optimal Control,” The International Journal of Robotics Research, vol. 22, no. 7, pp. 583–601, 2003.
- **行人运动预测 survey**：Rudenko, Andrey, Luigi Palmieri, Michael Herman, Kris M. Kitani, Dariu M. Gavrila, and Kai O. Arras. "Human motion trajectory prediction: A survey." The International Journal of Robotics Research 39, no. 8 (2020): 895-935.
- **交通运动预测（深度学习，MRTR）**：Gan, Yiqian, Hao Xiao, Yizhe Zhao, Ethan Zhang, Zhe Huang, Xin Ye, and Lingting Ge. "MGTR: Multi-granular transformer for motion prediction with lidar." arXiv preprint arXiv:2312.02409 (2023).
- **交通运动预测（约束优化，GAMMA）**：Luo, Yuanfu, Panpan Cai, Yiyuan Lee, and David Hsu. "Gamma: A general agent motion model for autonomous driving." IEEE Robotics and Automation Letters 7, no. 2 (2022): 3499-3506.

第九讲扩展阅读

- **PRM:** Kavraki, Lydia E., Petr Svestka, J-C. Latombe, and Mark H. Overmars. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces." IEEE transactions on Robotics and Automation 12, no. 4 (1996): 566-580.
- **RRT-connect**: An efficient approach to single-query path planning. J. J. Kuffner and S. M. LaValle. In Proceedings IEEE International Conference on Robotics and Automation, pages 995-1001, 2000.
- **RRT* 与 PRM***： Karaman, Sertac, and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning." The international journal of robotics research 30, no. 7 (2011): 846-894. Karaman, Sertac, and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning." The international journal of robotics research 30, no. 7 (2011): 846-894.
- **CL-RRT**: Kuwata, Yoshiaki, Justin Teo, Gaston Fiore, Sertac Karaman, Emilio Frazzoli, and Jonathan P. How. "Real-time motion planning with applications to autonomous urban driving." IEEE Transactions on control systems technology 17, no. 5 (2009): 1105-1118.
- **基于RRT的最新轨迹规划研究：**Shen, Congkai, Siyuan Yu, Bogdan I. Epureanu, and Tulga Ersal. "An Efficient Global Trajectory Planner for Highly Dynamical Nonholonomic Autonomous Vehicles on 3D Terrains." IEEE Transactions on Robotics (2023).
- **基于PRM + MPC 的最新运动规划研究**： de Groot, Oscar, Laura Ferranti, Dariu Gavrila, and Javier Alonso-Mora. "Topology-Driven Parallel Trajectory Optimization in Dynamic Environments." arXiv preprint arXiv:2401.06021 (2024).

层级式自动驾驶规划架构



第十讲：决策规划I

蔡盼盼 副教授

上海交通大学清源研究院

研究领域：机器人规划、机器人学习、自动驾驶

邮箱：Cai_panpan@sjtu.edu.cn

网站：https://cindycia.github.io/

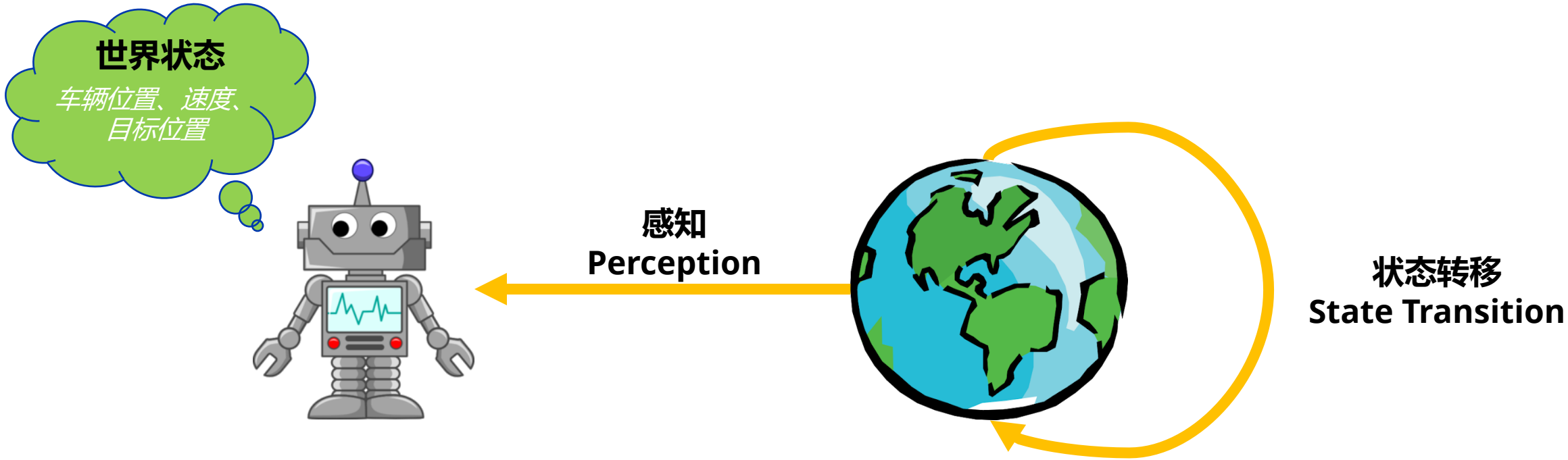
本讲材料与设计借鉴了David Hsu在NUS的课程 CS5478

确定性环境与随机环境

- **确定性环境：**
  - 可以准确预测环境动态（自车的行为结果与障碍物的运动）
- **随机环境：**
  - 无法准确预测环境动态（自车的行为结果与障碍物的运动）

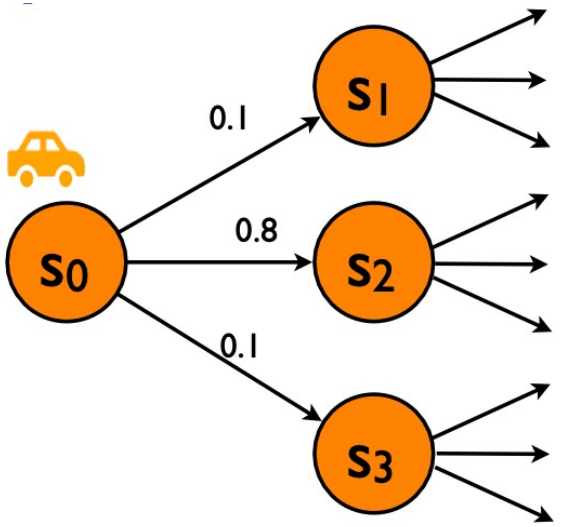
机器人与随机环境的交互

- **被动观察：**
  - 车辆停在十字路口，观察其他车辆和行人的行为（感知）



马尔科夫链（Markov Chain）

- **Markov chain** 包含两个基本元素：
  - S: 状态空间
    - 状态 s 可以表示场景中所有人、车所在的几何状态、行为状态
  - $T(s, s') = p(s'|s)$：状态转移函数
    - 表达状态转移的**随机性**：若当前世界处在状态 s, 下一步转移到状态 s′ 的概率是多少？



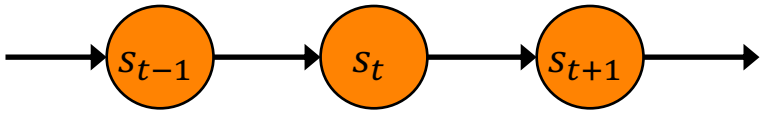
$P(s'|s)$:

	s_0	s_1	s_2	s_3
s_0	0	0.1	0.8	0.1
s_1	...			
s_2	...			
s_3	...			

马尔科夫链（Markov Chain）

- **假设**：
  1. 离散时间
     - 用离散时间步近似地表征连续时间
  2. 马尔科夫性（ Markovian ）
     - 当给定当前状态时，过去状态与未来状态为独立的随机变量
     - Markov chain的动态贝叶斯图蕴含了条件独立性假设
       - Causal chain 被阻隔

$$s_{t-1} \perp s_{t+1} \mid s_t$$

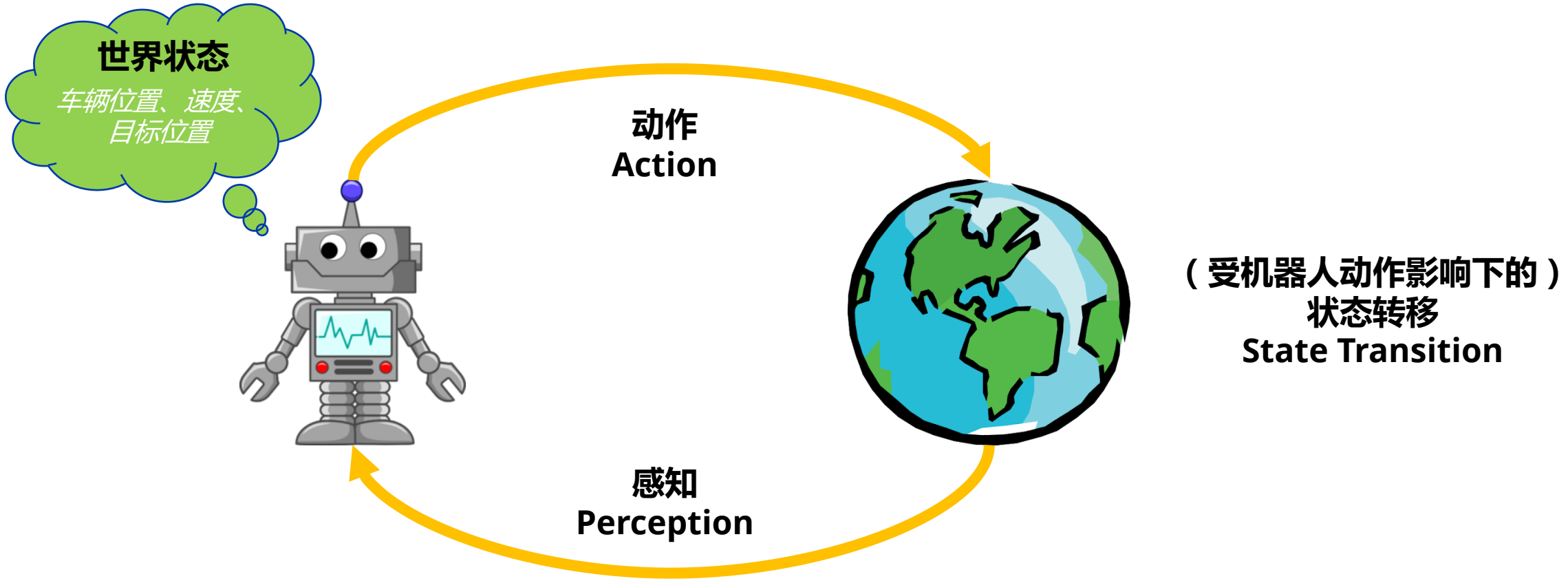


$P(s'|s)$:

	s_0	s_1	s_2	s_3
s_0	0	0.1	0.8	0.1
s_1	...			
s_2	...			
s_3	...			

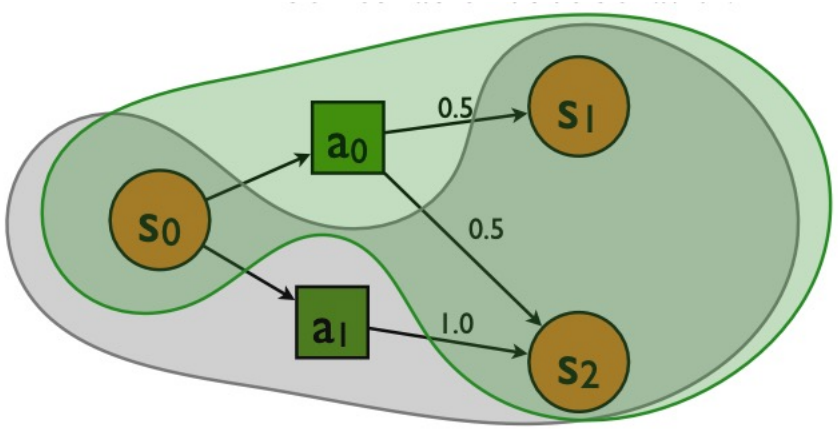
机器人与随机环境的交互

- **主动交互：**
  - 车辆观察其他车辆和行人的行为（感知），并找机会穿过十字路口（动作）



马尔科夫决策过程（Markov Decision Process）

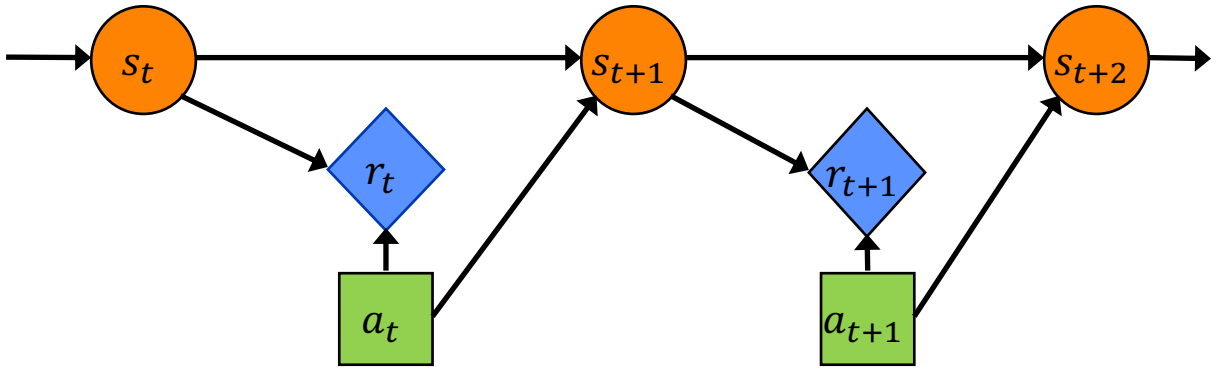
- **MDP 包含了一下四个元素：**
  - S: 状态空间
    - 状态 s 可以表示场景中所有人、车所在的几何、行为状态
  - A: 动作空间
    - 动作 a 可以表示自车的行为
    - 底层行为：方向盘角度、加速度
    - 高层行为：跟车、变道、避让，...
  - $T(s, a, s') = p(s'|s, a)$：状态转移函数
    - 表达机器人行为带来的不确定性结果：若当前世界处在状态 s, 机器人执行动作 a, 世界下一步转移到状态 s′ 的概率是多少？
  - $R(s, a)$ ：奖励函数
    - 表达机器人任务：若机器人在世界状态 s 执行动作 a, 所获得的即时奖励是多少？
    - 例：碰撞（-30000），到达终点（+100），变道（-4），...



马尔科夫决策过程（Markov Decision Process）

假设：

1. 离散时间： 用离散时间步近似地表征连续时间
2. 马尔科夫性（ Markovian）：当给定当前状态时，过去状态与未来状态为独立的随机变量
   - MDP 的动态贝叶斯图蕴含了独立性假设
     - Causal chain 通路被阻隔
     - Common Cause 通路被阻隔



发生了什么事？

导致了什么奖励/惩罚？

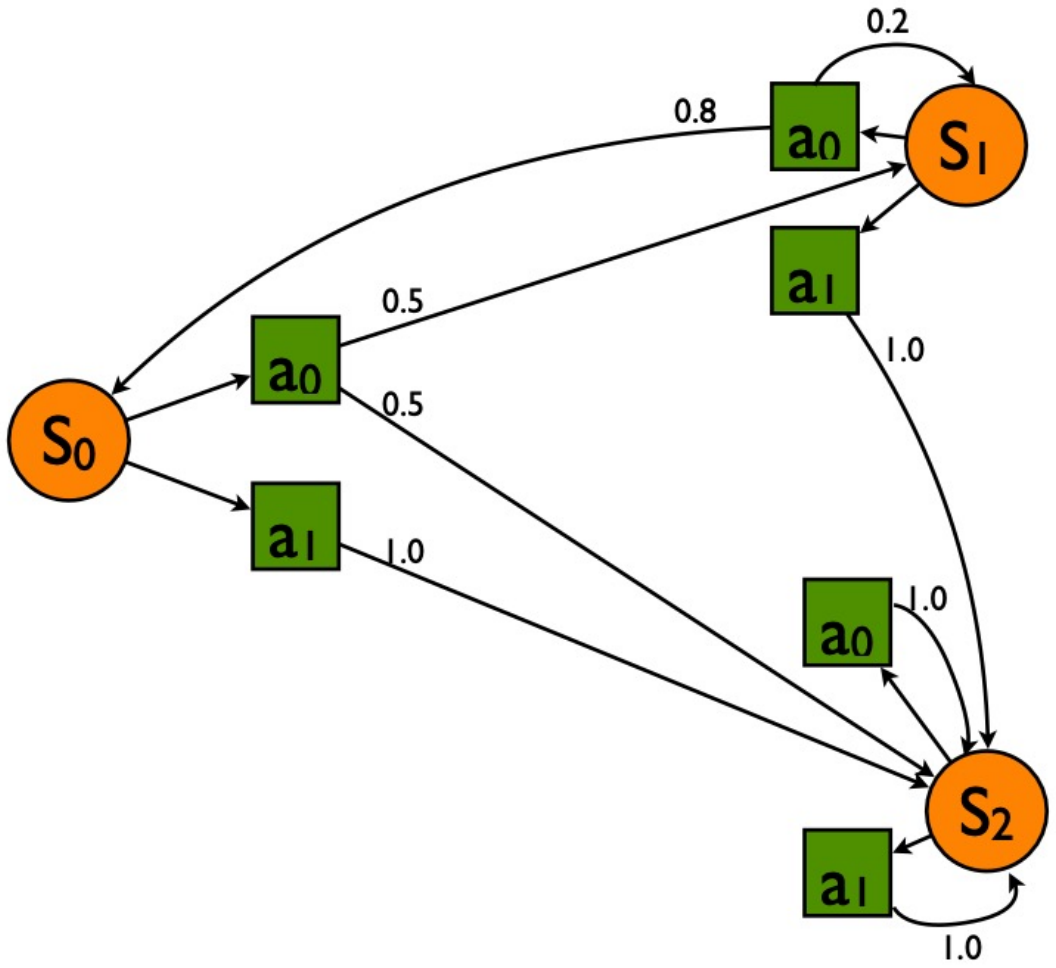
机器人应当如何决策？

$$s_{t-1} \perp s_{t+1} \mid s_t$$

$$s_{t-1} \perp r_t \mid s_t$$

$$s_{t+1} \perp r_t \mid s_t$$

MDP 状态空间图



s_2 是一个吸收状态（absorbing state）！

- 进入 s_2 后逃离的概率为0
- 在 s_1 执行 a_1 一定会进入 s_2
- 在 s_0 执行 a_1 也一定会进入 s_2
- 不断执行 a_0 在 s_0 与 s_1 之间循环？
  - 每次循环都有 0.8*0.5 的概率落入 s_2
  - 随着时间趋于无穷，落入 s_2 的概率将趋于1

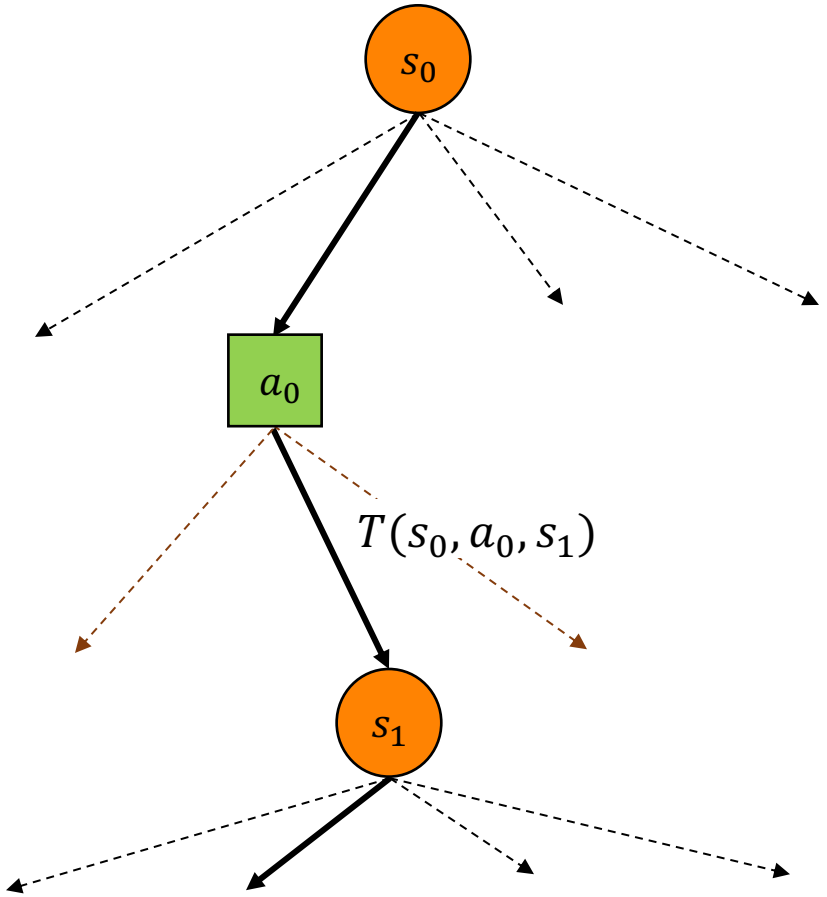
有限时域 MDP 规划（Finite horizon MDP planning）

- 找到一条动作序列，使得某个前瞻时域内的价值最大化
  - **时域（horizon）**：前瞻的时间步数
  - **价值（value）**：沿着动作序列的累积奖励的期望值

$$V = \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t R(s_t, a_t)\right]$$

对所有可能发生的轨迹求期望值

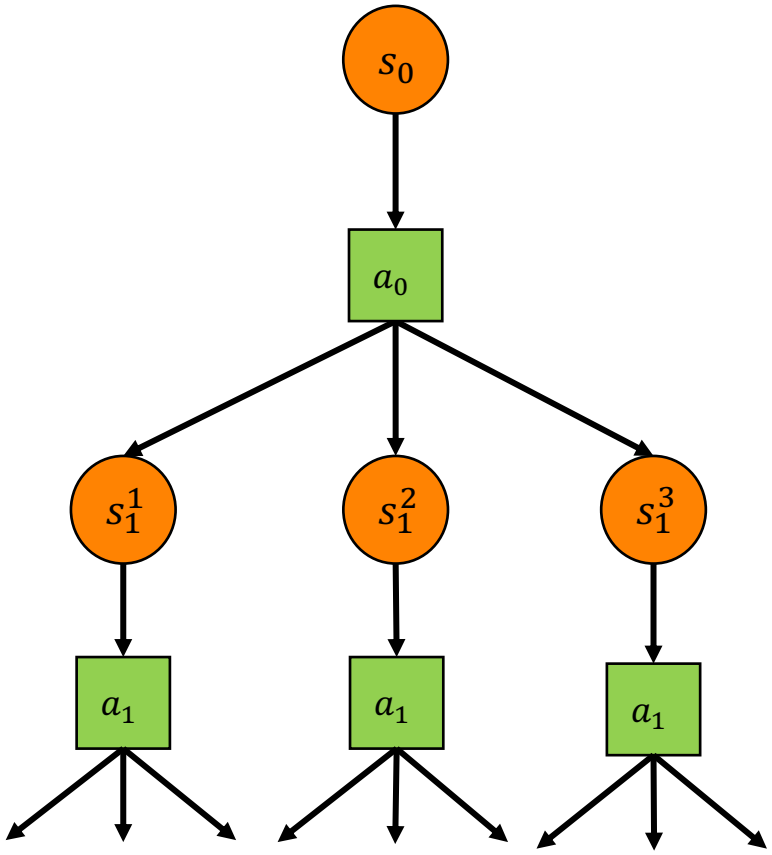
对整个前瞻时域求累积奖励



有限时域 MDP 规划的解

Finite horizon MDP的解可以表达为两种形式：

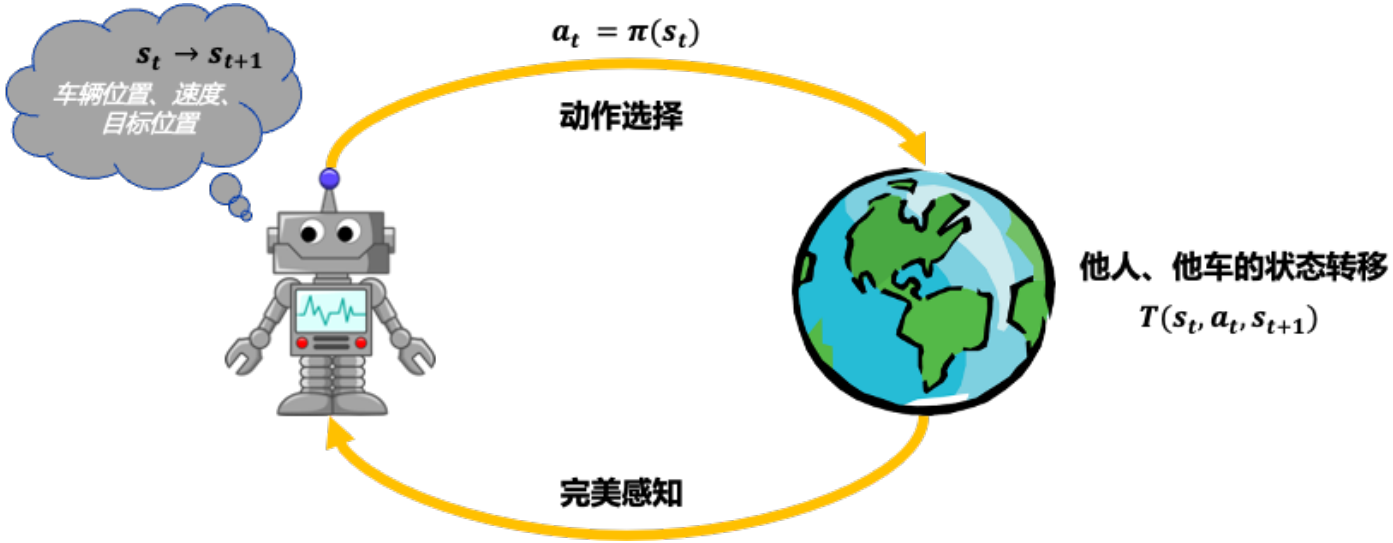
- **开环计划**（Open-loop plan）：$\boldsymbol{a_t = \pi(t)}$
  - 针对任意时间 t ，指定一个相应的机器人动作 a_t
  - 但机器人动作对世界状态的影响结果是不确定的，开环策略无法根据未来可能的情况灵活调整动作

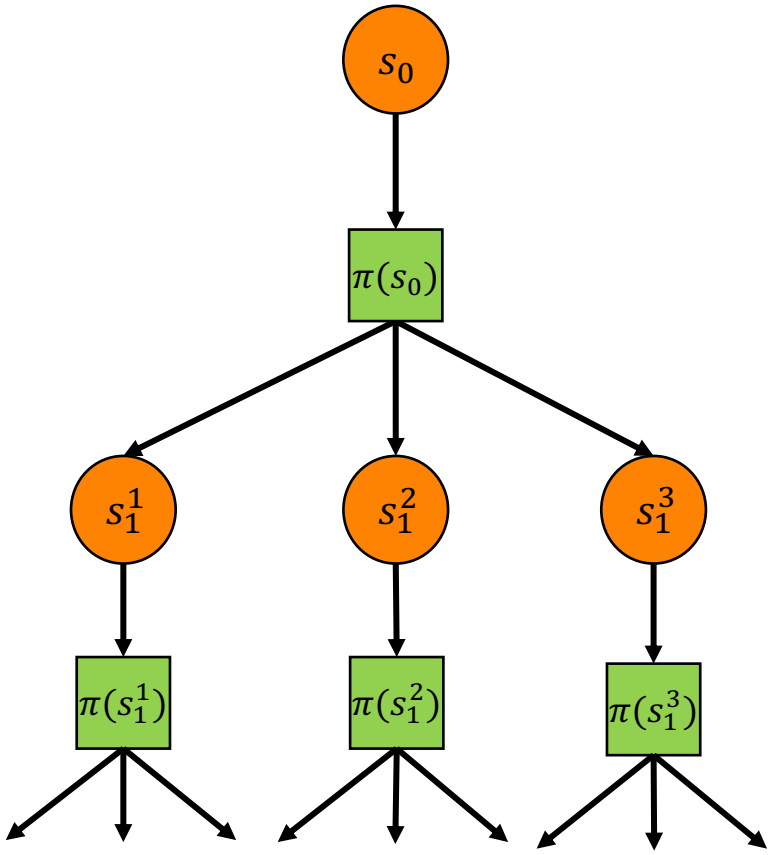


有限时域 MDP 规划的解

Finite horizon MDP的解可以表达为两种形式：

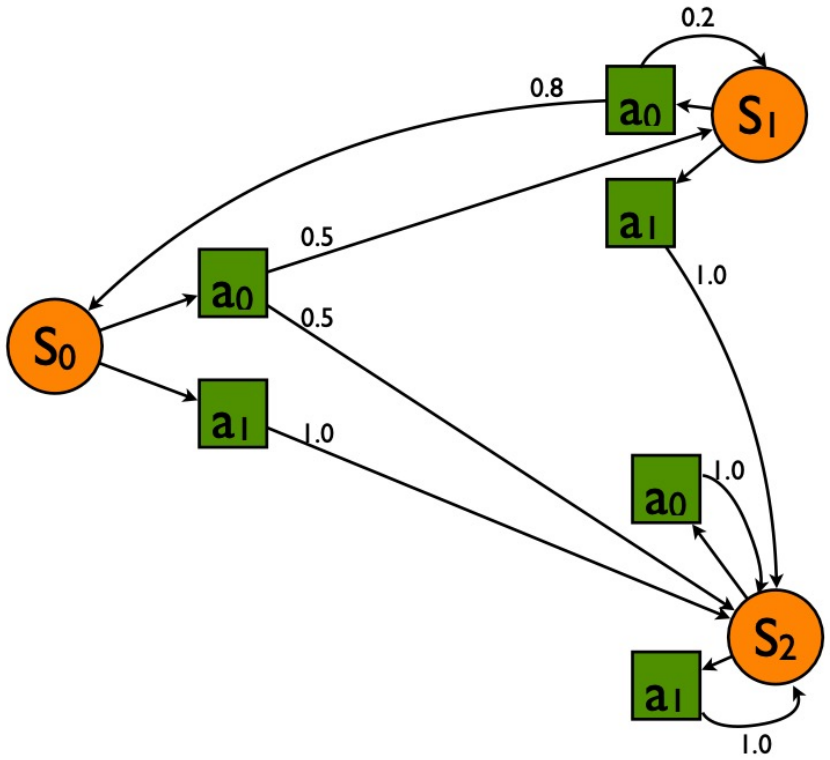
- **闭环计划**（Closed-loop plan）：$a_t = \pi(s_t)$
  - 根据任意的世界状态 s_t，指定一个相应的机器人动作 a_t
  - Closed-loop plan 通常称作**策略（policy）**



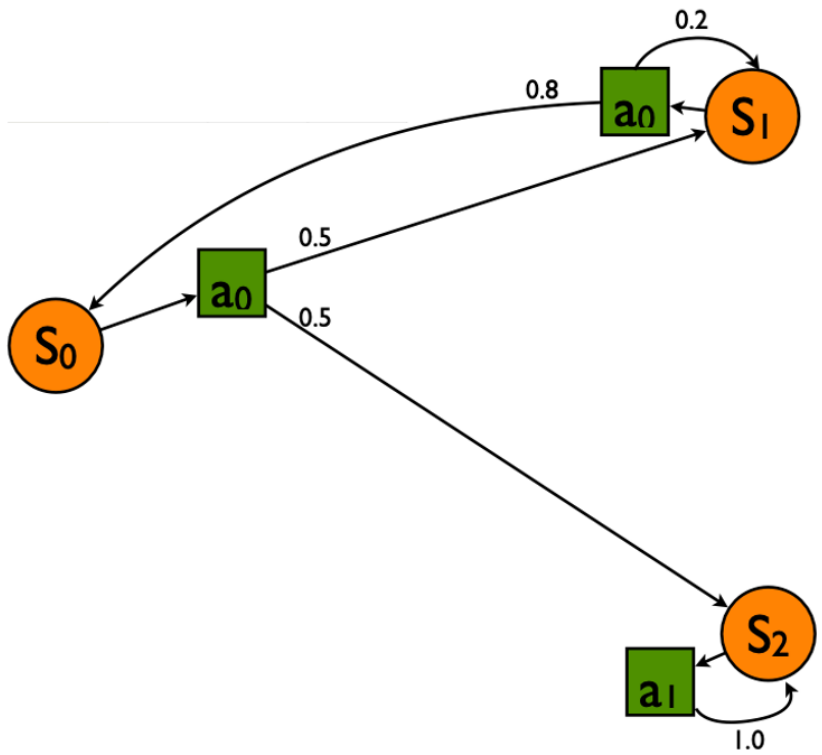


“策略树”（policy tree）

策略图（Policy Graph）



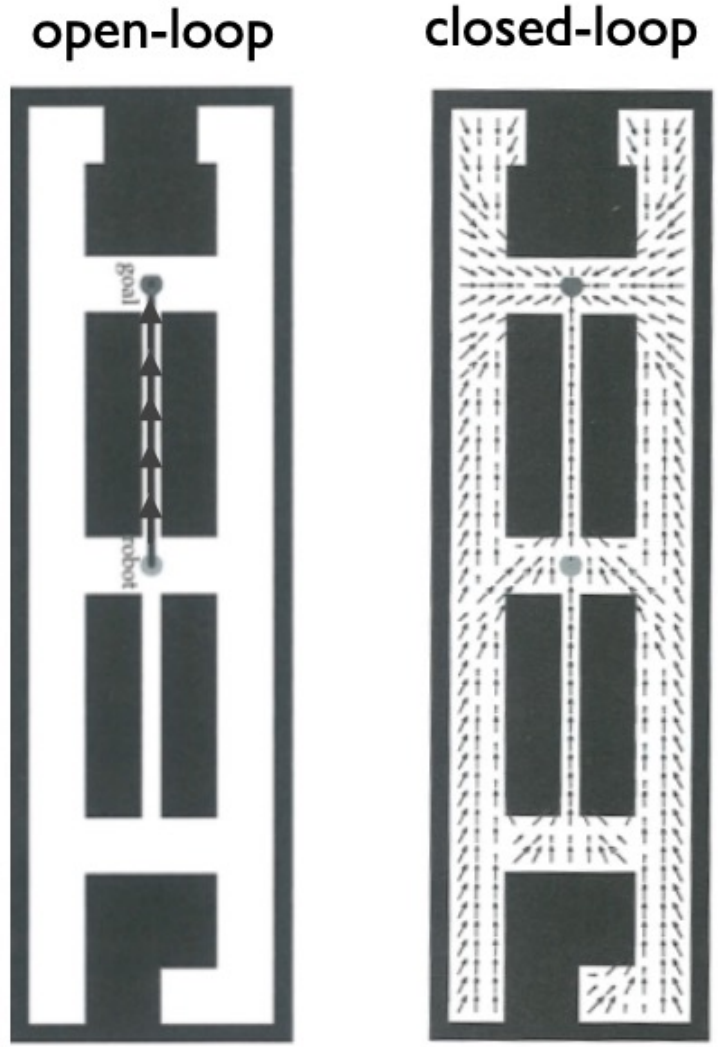
$$\pi : S \rightarrow A$$



有限时域 MDP 规划的解（例）

- **开环计划**（Open-loop plan）：$\boldsymbol{a_t = \pi(t)}$
  - 针对任意时间 t ，指定一个相应的机器人动作 a_t
- **闭环计划/策略**（Closed-loop plan）：$\boldsymbol{a_t = \pi(s_t)}$
  - 针对任意的世界状态 s_t ，指定一个相应的机器人动作 a_t

- RRT 输出的是？
- CL-RRT 输出的是？

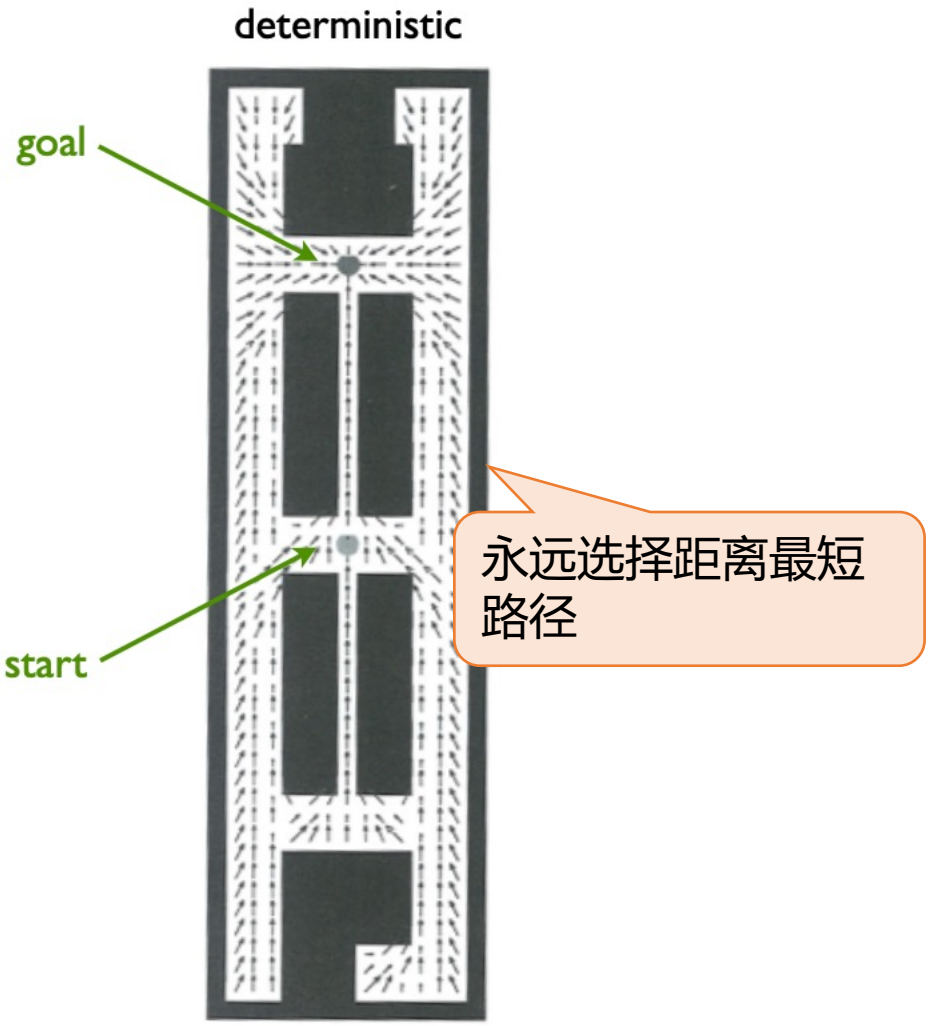


思考题

- 考虑两个相似的环境：
  - 第一个环境具有**确定性**的环境动态（deterministic dynamics），即每次机器人运动都导致同样的结果。
  - 另一个环境具有**随机性**的环境动态（stochastic dynamics），即每次机器人运动都可能导致不同的结果。机器人以 0.9 的概率到达目的位置，以 0.1 的概率偏移到其他位置（可能落入黑色区域，reward 为 -1）。
- 两个环境中的最优策略有何不同？

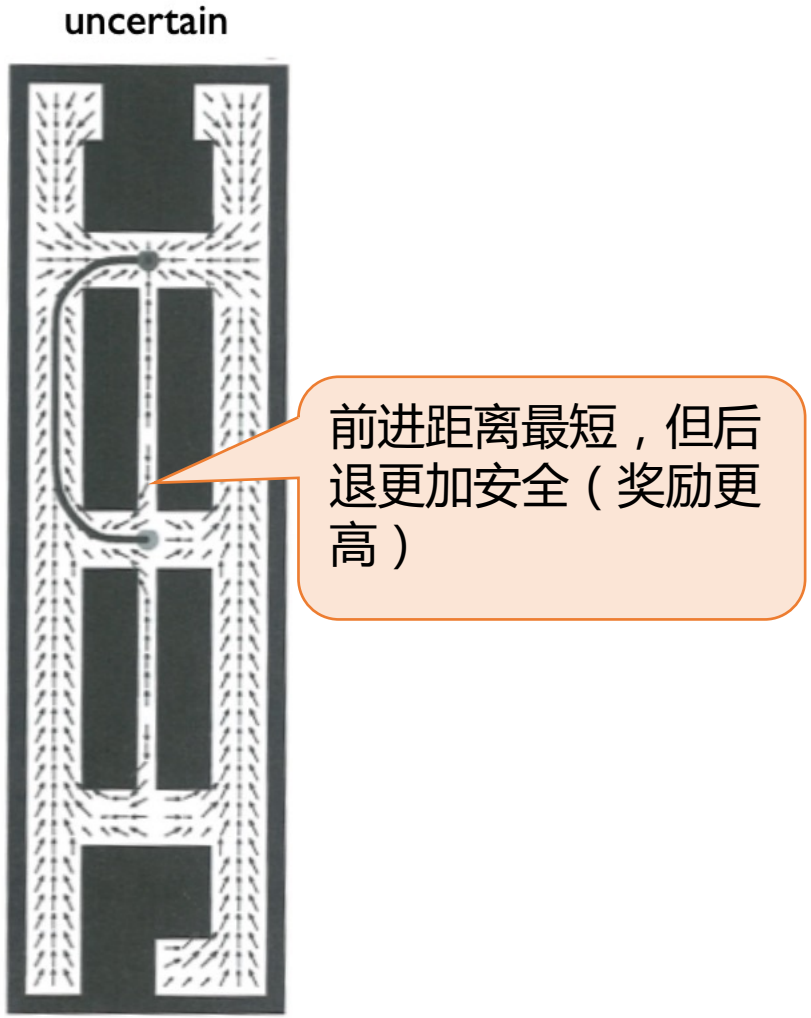
思考题

- 考虑两个相似的环境：
  - 第一个环境具有**确定性**的环境动态（deterministic dynamics），即每次机器人运动都导致同样的结果。
  - 另一个环境具有**随机性**的环境动态（stochastic dynamics），即每次机器人运动都可能导致不同的结果。机器人以 0.9 的概率到达目的位置，以 0.1 的概率偏移到其他位置（可能落入黑色区域，reward 为 -1）。
- 两个环境中的最优策略有何不同？



思考题

- 考虑两个相似的环境：
  - 第一个环境具有**确定性**的环境动态（deterministic dynamics），即每次机器人运动都导致同样的结果。
  - 另一个环境具有**随机性**的环境动态（stochastic dynamics），即每次机器人运动都可能导致不同的结果。机器人以 0.9 的概率到达目的位置，以 0.1 的概率偏移到其他位置（可能落入黑色区域，reward 为 -1）。
- 两个环境中的最优策略有何不同？



有限时域 MDP 规划（闭环策略版本）

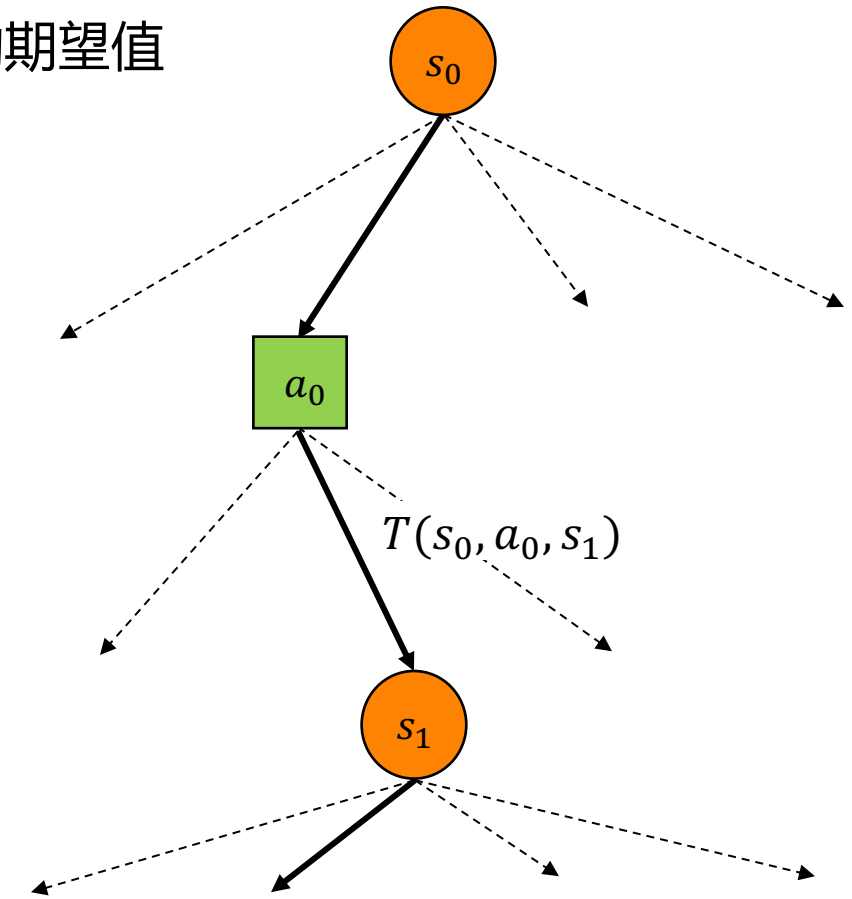
- 找到一个最优策略，使得某个前瞻时域内的价值最大化
  - **策略价值（value）**：执行策略可获得的累积奖励的期望值

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t R(s_t, a_t) \,|s_0 = s, a_t = \pi(s_t)\right]$$

考虑所有可能发生的轨迹

整个前瞻时域内的累积奖励

机器人始终按策略 π 选择动作



  - **最优**?

有限时域 MDP 规划（最优量）

- **最优值**（optimal value）：从状态 s 出发，所能达到的最优策略价值

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

$$= \max_{\pi} \mathbb{E}\left[\sum_{i=0}^{H} \gamma^t R(s_t, a_t) \,|s_0 = s, a_t = \pi(s_t)\right]$$

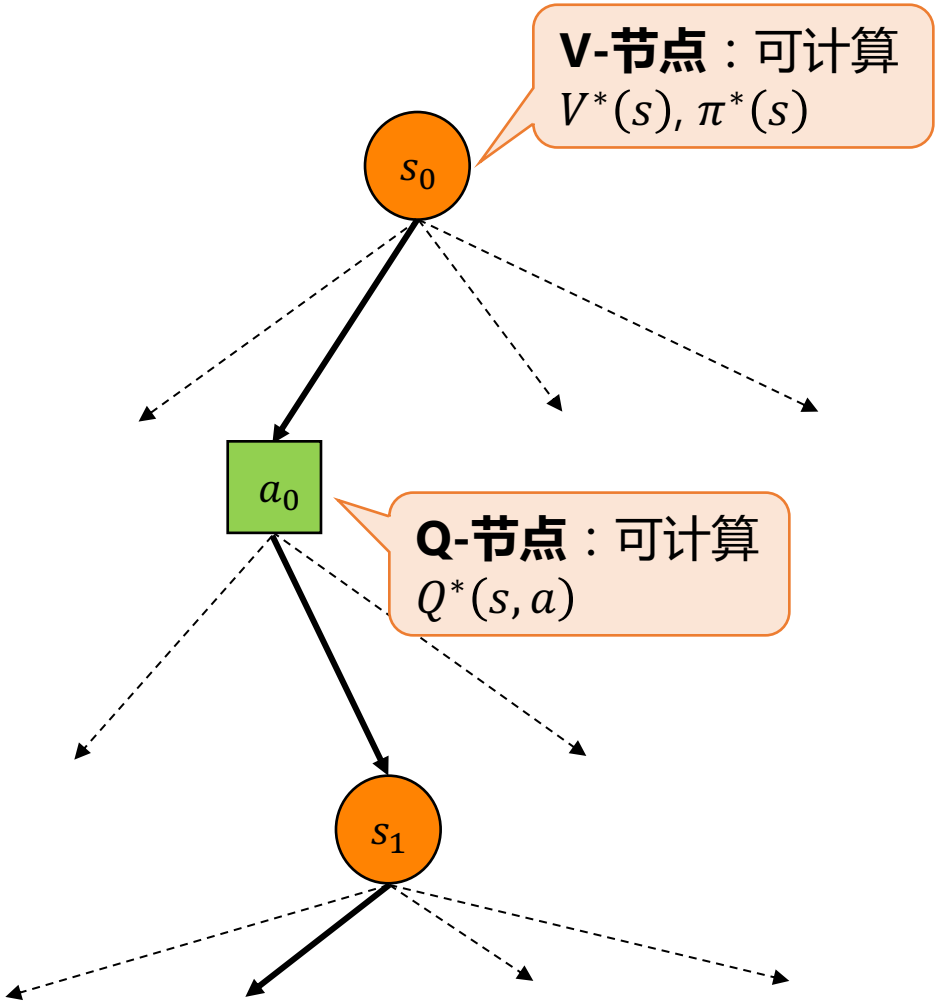
- **最优策略**（optimal policy）：使值最大化的策略

$$\pi^*(s) = \operatorname*{argmax}_{\pi} V^{\pi}(s)$$

- **最优 Q 值**（optimal q-value）：从状态 s 出发，执行动作 a 之后，所能达到的最优值

$$Q^*(s, a) = R(s_0, a) + \max_{\pi} \mathbb{E}[V^{\pi}(s_1)]$$

a 可以是任意动作！



最优量计算：Bellman 最优性条件

MDP的最优值满足以下充分必要条件：

- 最优值（optimal value）版本：

$$V^*(s) = \max_a R(s,a) + \gamma \sum_{s'} T(s,a,s')V^*(s')$$

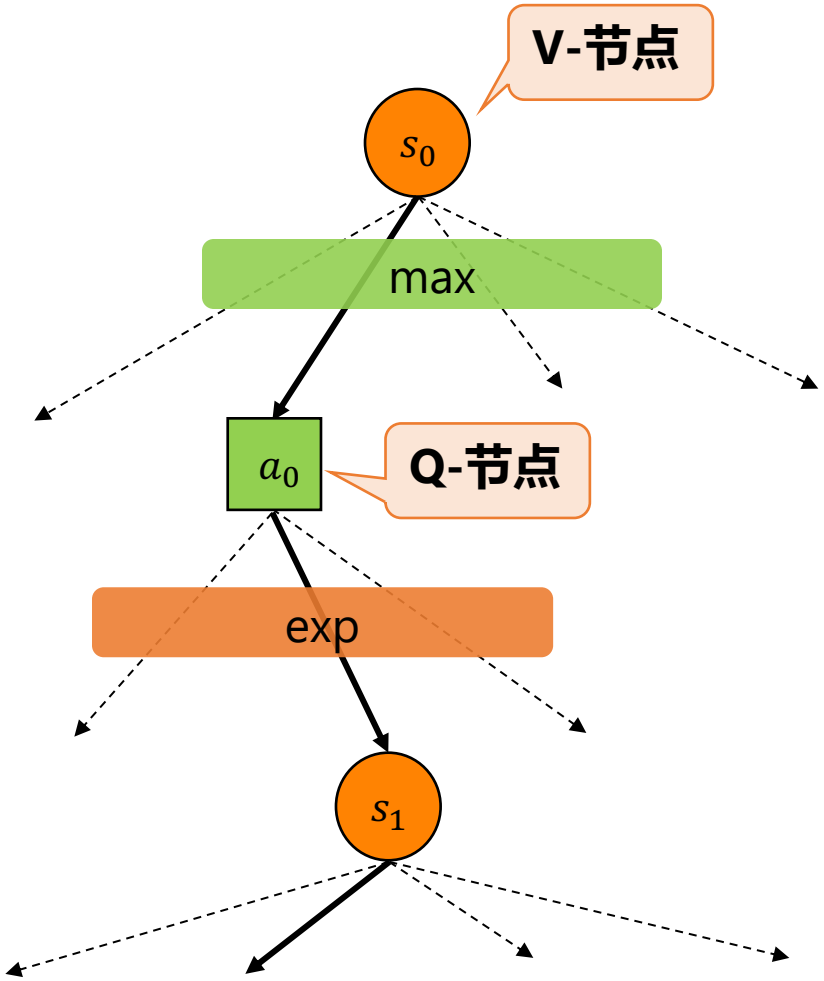
- 最优 Q 值 – V值双循环版本：

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s')\, V^*(s')$$

$$V^*(s) = \max_a Q^*(s,a)$$

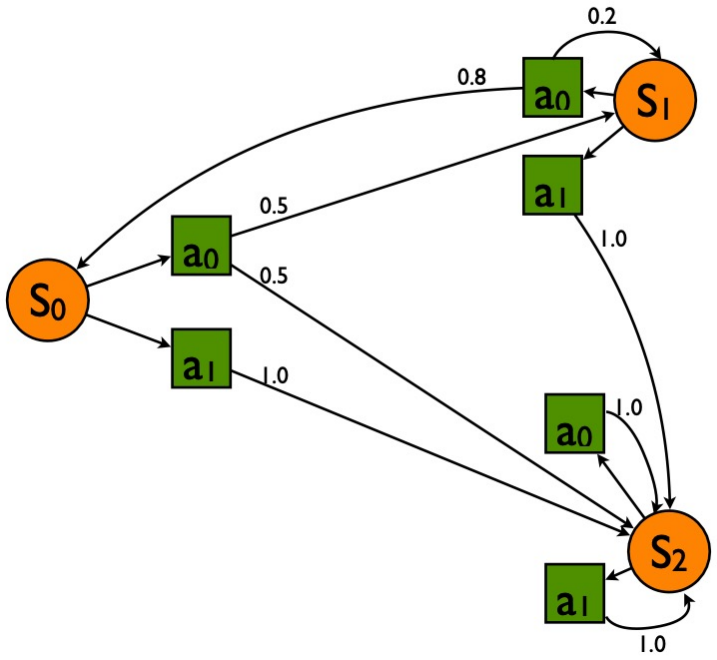
- 最优策略（optimal policy）版本：

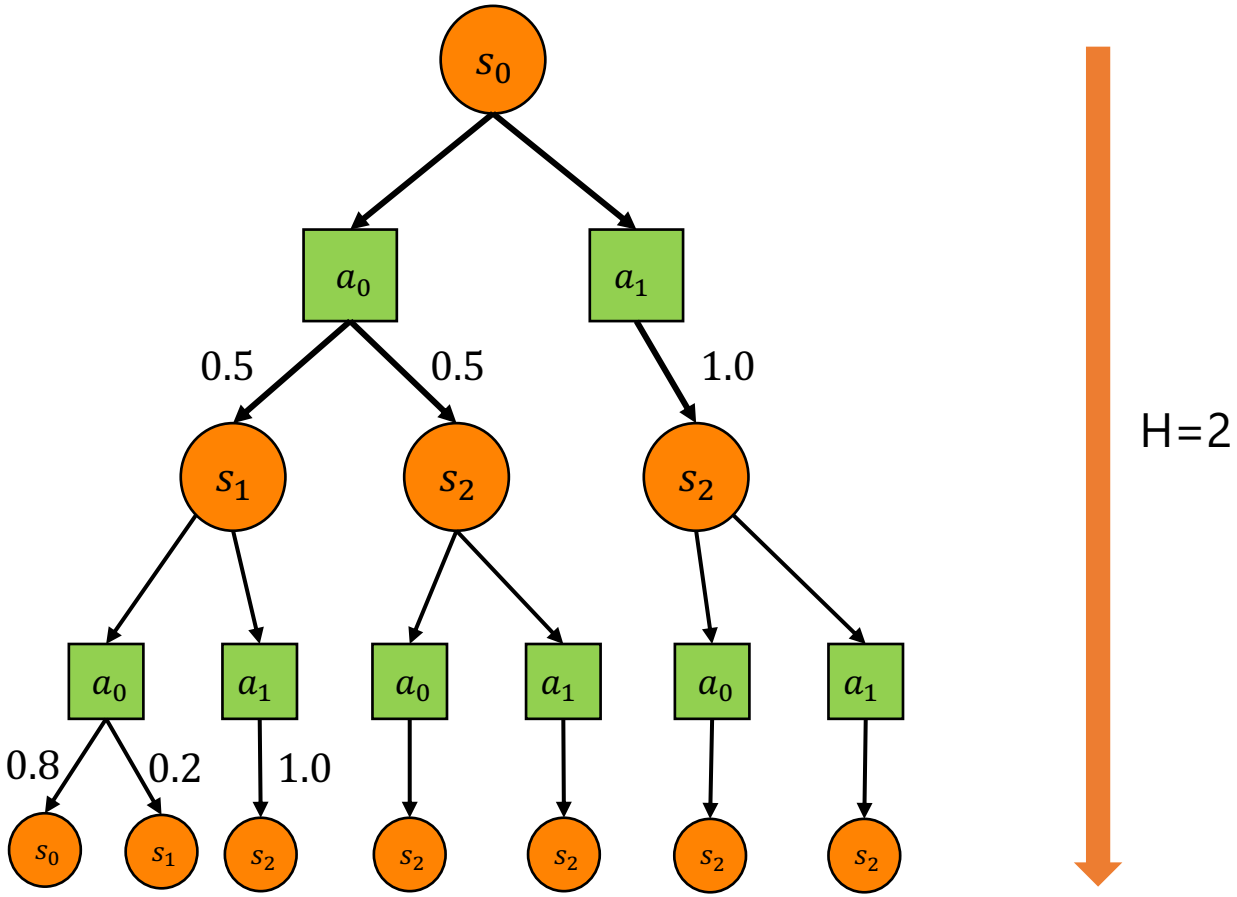
$$\pi^*(s) = \operatorname{argmax}_a Q^*(s,a)$$



MDP 规划与树搜索

- 前瞻时域内，所有可能的轨迹构成一棵树：





MDP规划与树搜索

- 在树中运用 Bellman equation，得到 **动态规划算法**：
  - 在所有叶节点 s 赋值 V(s)=0
  - 上移半层，对所有 Q-节点计算：

$$Q(s,a) = R(s,a) + \sum_{s'} T(s,a,s')V(s')$$

（其中 s′ 是 s 的子节点）

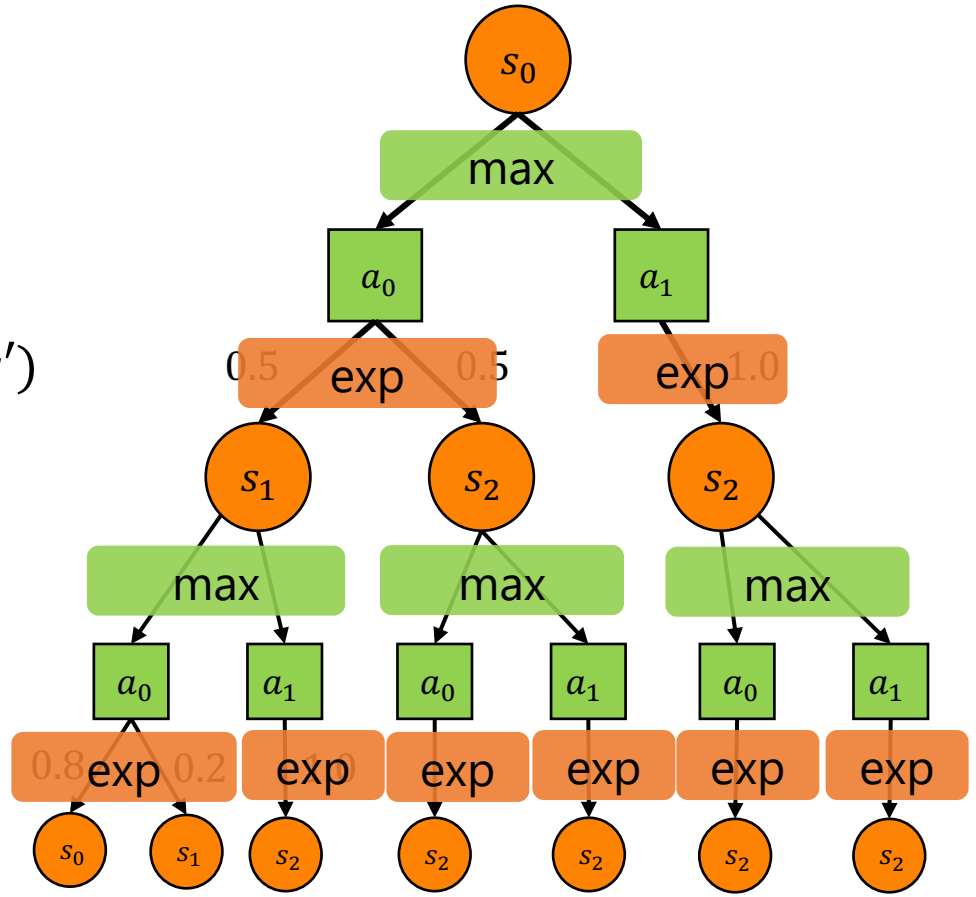
  - 上移半层，对所有 V-节点计算：

$$V(s) = \max_a Q(s,a)$$

（其中 a 是机器人可选择的动作）

  - 循环直至处理完根节点 s_0
  - 对于任意树节点 s，最优策略为:

$$\pi(s) = \operatorname{argmax}_a Q(s,a)$$



H=2

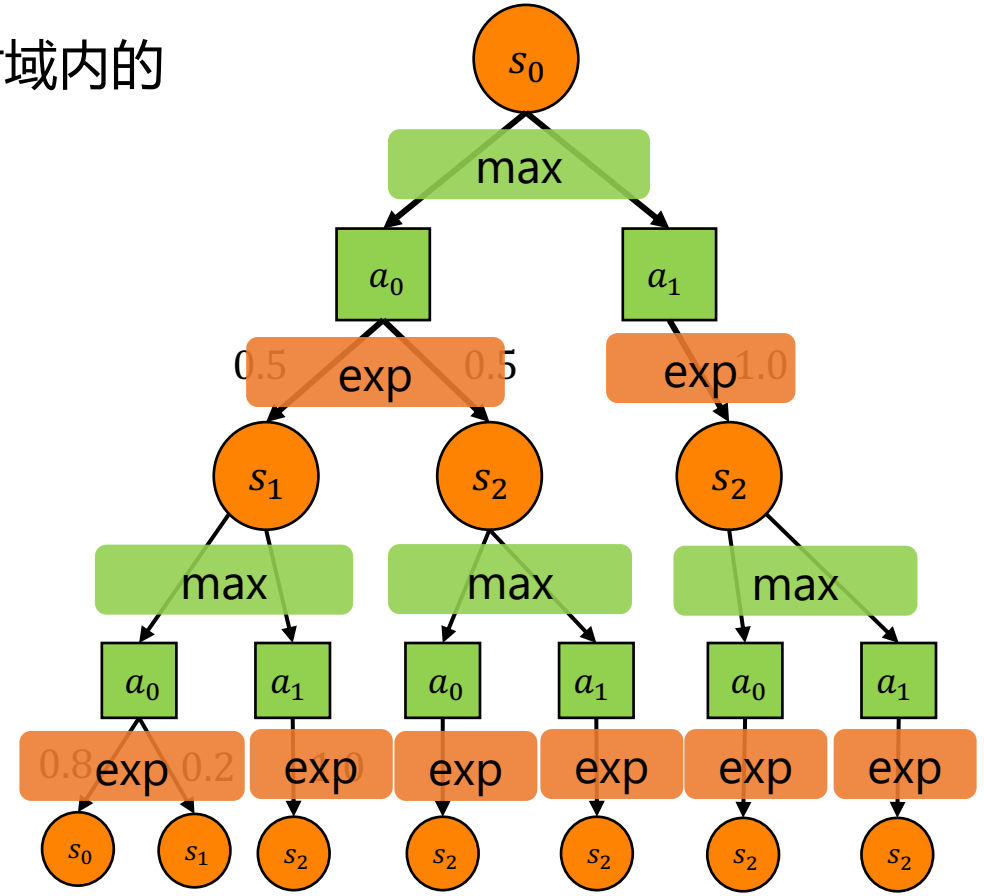
$$V^*(s) = \max_a R(s,a) + \gamma \sum_{s'} T(s,a,s')V^*(s')$$

动态规划算法的优缺点

- 优点：
  - 为所有未来可能遇到的状态计算了在时域内的最优策略，策略树可以运行 H 步
- 缺点：
  - 计算复杂度：

$$O(|A|^H|S|^H)$$

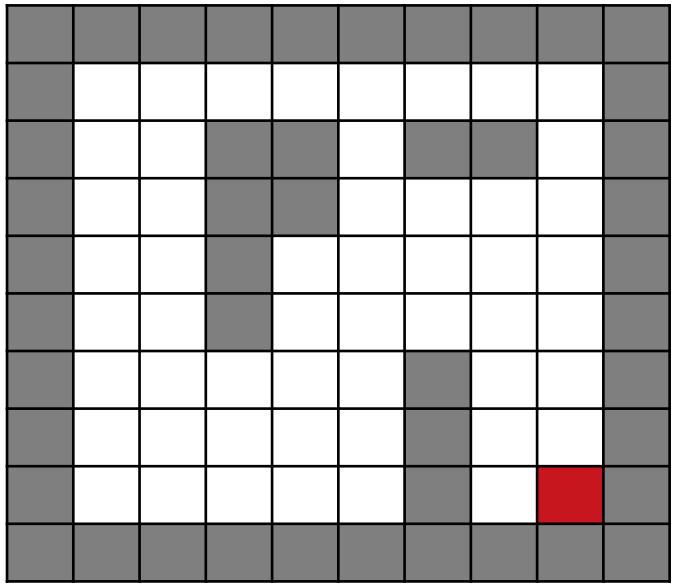
  - 对于示例问题：重复的分支太多



$$V^*(s) = \max_a R(s,a) + \sum_{s'} T(s,a,s')V^*(s')$$

MDP规划问题（例）

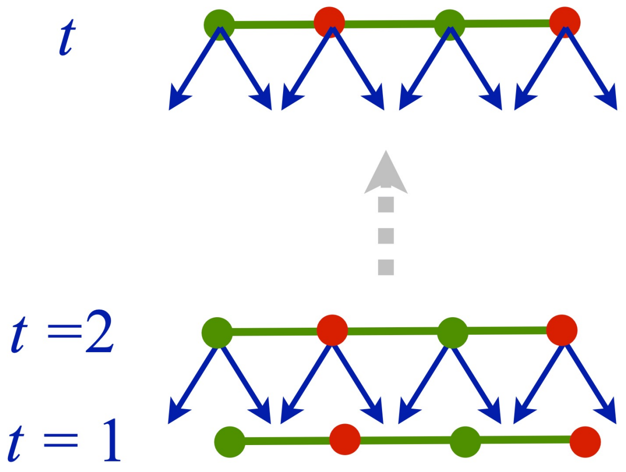
- 机器人的工作空间是一个 10x10 栅格：
  - 灰色代表障碍物
  - 终点在右下角
  - 到达终点取得+1奖励，其余时间没有奖励
- 没有障碍物的情况下，机器人在每一步可以选择移动到相邻4个格子，成功率为3/4。失败时，机器人会以同等的概率滑到其他3个格子。
- 机器人也可以选择停在原地不动
- $\gamma = 0.9$



S？
A？
T？
R？

MDP规划问题（例）

- 值迭代（Value Iteration）
  - 将所有起点状态的动态规划同时进行



  - 初始化时时，对所有状态 s 赋值：

    $V^0(s) = 0$

  - 每次迭代时，对所有状态 s 计算：

$$V^t(s) = \max_{a} R(s,a) + \gamma \sum_{s'} T(s,a,s') V^{t-1}(s')$$

Initialization

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Iteration 1

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0.75	0
0	0	0	0	0	0	0	0.75	1	0
0	0	0	0	0	0	0	0	0	0

Iteration 2

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0.51	0
0	0	0	0	0	0	0	0.56	1.43	0
0	0	0	0	0	0	0	1.43	1.9	0
0	0	0	0	0	0	0	0	0	0

MDP规划问题（例）

- 值迭代（Value Iteration）
  - 初始化时时，对所有状态 s 赋值：

$V^0(s) = 0$

  - 每次迭代时，对所有状态 s 计算：

$$V^t(s) = \max_a R(s,a) + \gamma \sum_{s'} T(s,a,s') V^{t-1}(s')$$

  - 迭代至所有 V(s) 收敛

Iteration 50

0	0	0	0	0	0	0	0	0	0
0	0.44	0.54	0.59	0.82	1.15	0.85	1.09	1.52	0
0	0.59	0.69	0	0	1.52	0	0	2.13	0
0	0.75	0.90	0	0	2.12	2.55	2.98	3.00	0
0	0.95	1.18	0	2.00	2.70	3.22	3.80	3.88	0
0	1.20	1.55	1.87	2.41	2.92	3.51	4.52	5.00	0
0	1.15	1.47	1.74	2.05	2.25	0	5.34	6.47	0
0	0.99	1.26	1.49	1.72	1.74	0	6.69	8.44	0
0	0.74	0.99	1.17	1.34	1.27	0	7.96	9.94	0
0	0	0	0	0	0	0	0	0	0

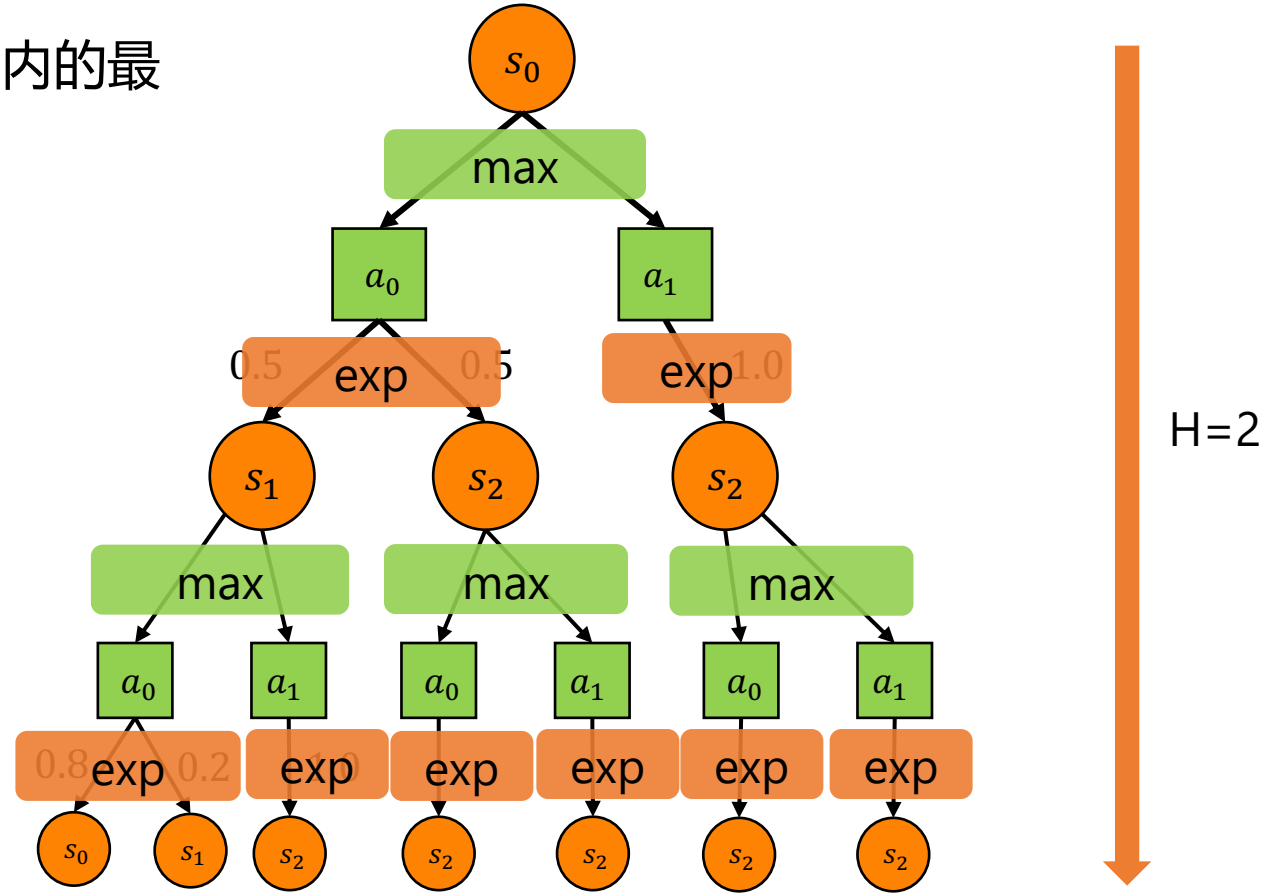
如何提取最优策略？

动态规划算法的优缺点

- 优点：
  - 计算了所有未来可能遇到的状态在时域内的最优策略，策略树可以运行 H 步
- 缺点：
  - 计算复杂度：

    $O(|A|^H|S|^H)$
  - 对于示例问题：重复的分支太多

    -> 值迭代: $O(N|A||S|^2)$
  - 对于自动驾驶问题：
    - |A|=?
    - |S|=?



$$V^*(s) = \max_a R(s,a) + \sum_{s'} T(s,a,s')V^*(s')$$