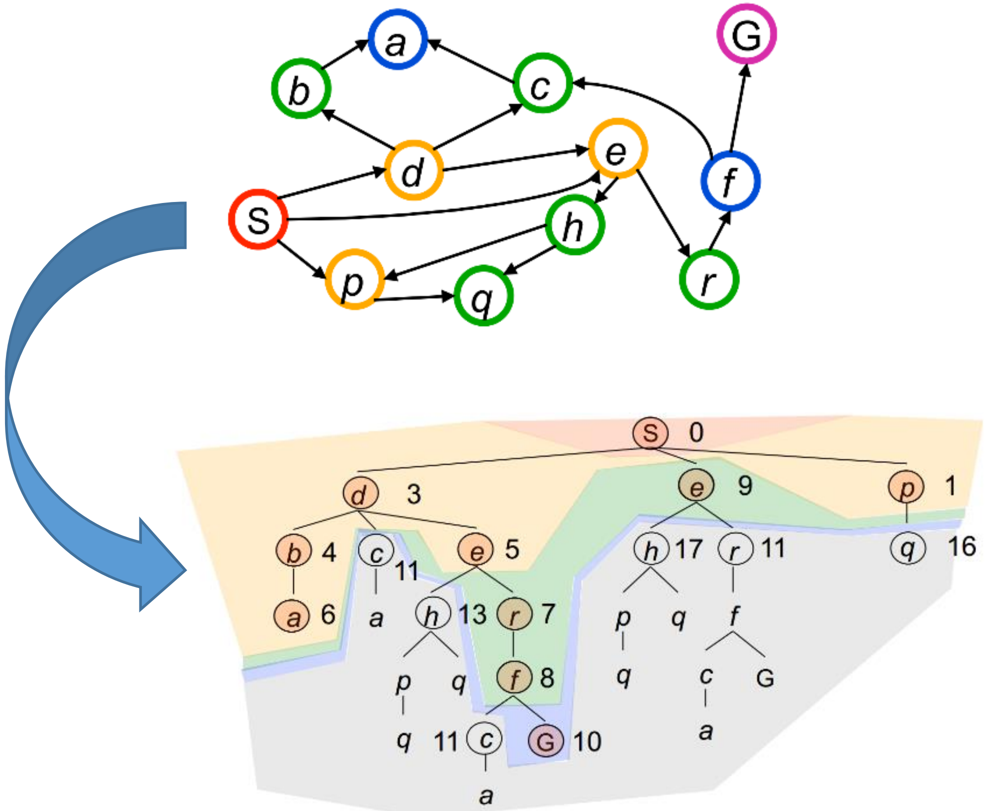
状态栅格规划
State Lattice Planning

基础思路

- 基于搜索的路径生成
- 对于规划问题，如何构建一个图？
- 这个图是否适用于我们真实的机器人？

搜索图

搜索树



- 维护一个优先队列来存储所有要扩展的节点
- 对所有节点的启发式函数 h(n) 都是提前定义好的
- 优先队列被起始状态 X_S 初始化
- 在图中对所有节点赋值 $g(X_S)=0$, g(n)=infinite
- Loop
    - 如果队列为空, return FALSE; break;
    - 从队列中移除f(n)=g(n)+h(n)值最低的节点"n"
    - 标识节点"n"为被拓展过的
    - 若节点"n"是目标状态, return TRUE; break;
    - 对于节点"n"所有的未被拓展过的邻居节点"m"
        - 如果 g(m) = infinite
            - g(m)= g(n) + Cnm
            - 将节点 "m" 加入队列
        - 如果 $g(m) > g(n) + C_{nm}$
            - g(m)= g(n) + Cnm
- End Loop

基础思路

- 我们有很多方法去解决图搜索问题
- 假设机器人是一个质点已经不再适用
- 我们现在需要一个具有可行运动连接（feasible motion connections）的图

- 我们手动构建一个所有边都可由机器人执行的图
  - 正向：在控制空间采样
  - 反向：在状态空间采样

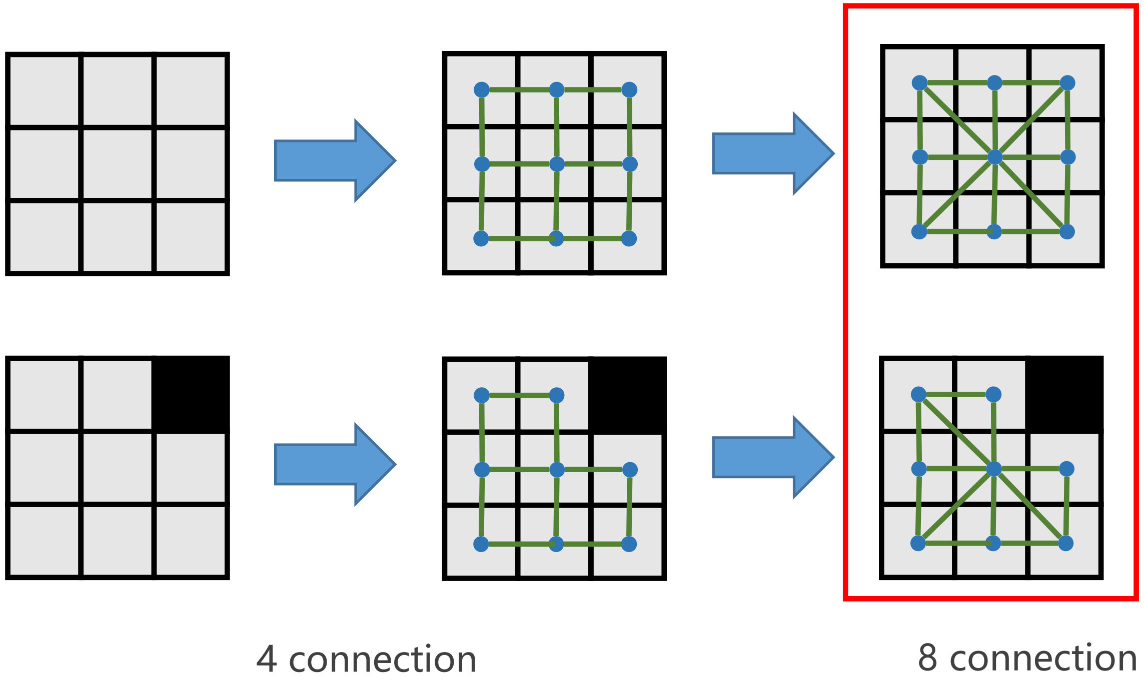
- 这是所有运动动力学规划的基本动机
- State lattice planning 是最直接的一种

基础思路

- 我们手动构建一个所有边都可由机器人执行的图

正向：在控制空间采样

反向：在状态空间采样



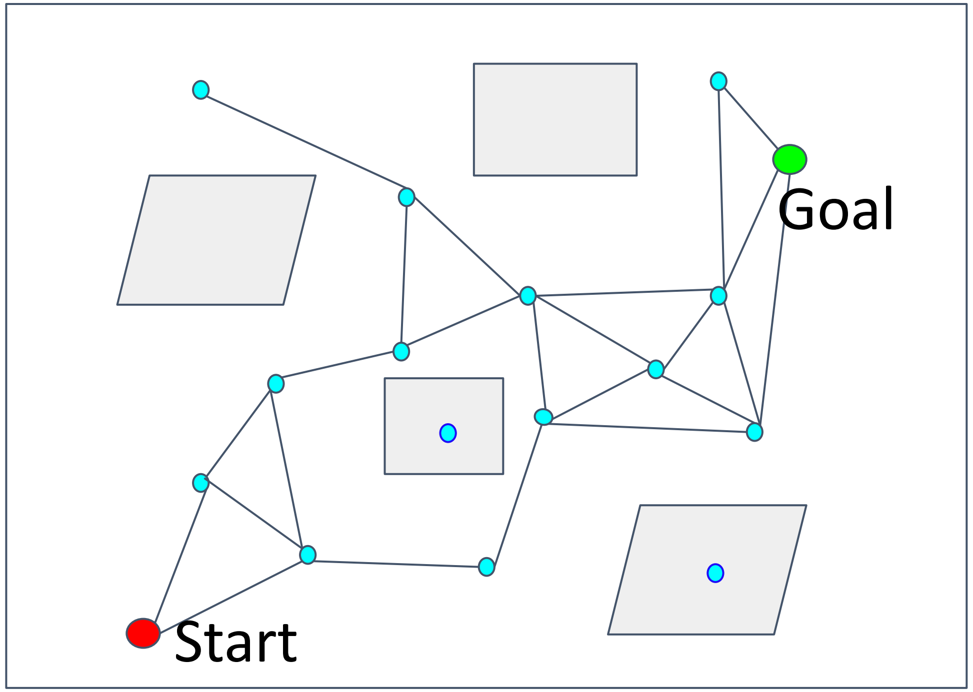
- 实际上，这是在控制空间进行离散化采样
- 我们假设机器人可以向4/8个方向运动

基础思路

- 我们手动构建一个所有边都可由机器人执行的图

正向：在控制空间采样

反向：在状态空间采样



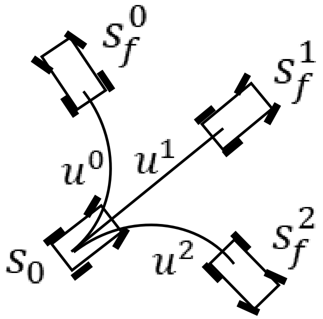
- 实际上，这是在状态空间进行离散化采样
- 这里的状态在R^2上，只有位置(x, y)被考虑

控制空间采样 vs 状态空间采样

对于一个机器人模型：$\dot{s}=f(s,u)$

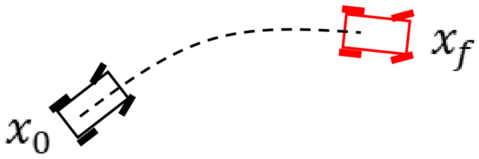
- 机器人可以被微分驱动
- 我们有一个机器人的初始状态s_0
- 通过下面的方法生成可行的局部运动：

□ 选择一个输入u，固定一个持续时间T，前向模拟系统（数值积分）



- 前向模拟
- 固定u, T
- 没有任务引导
- 易于实现
- 低规划效率

□ 选择一个s_f，找到两个状态s_0和s_f之间的连接（一段轨迹）



- 反向计算
- 需要计算u, T
- 较好的任务引导
- **难以实现**
- 高规划效率

控制空间采样

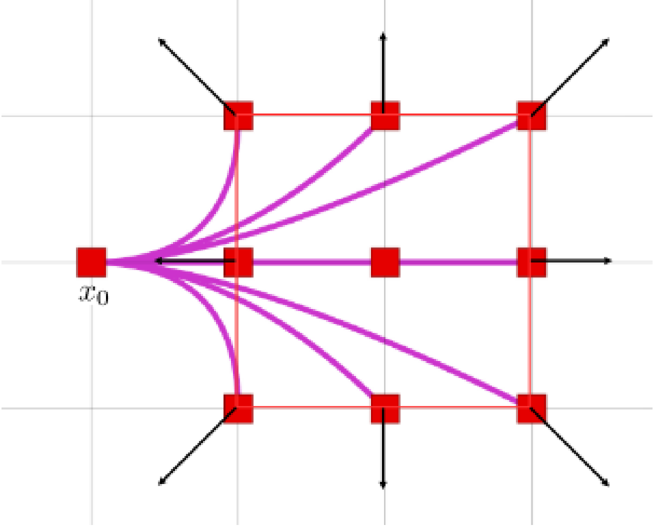
State: $s = \begin{pmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix}$ Input: $u = \begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix}$

System equation: $\dot{s} = A \cdot s + B \cdot u$

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow$$

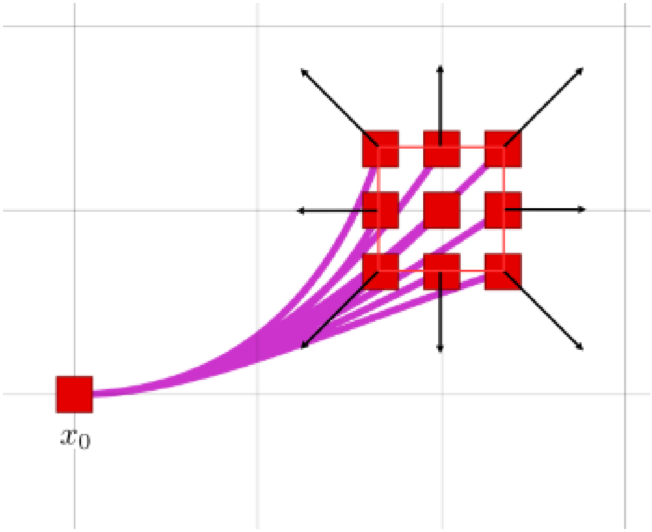
$$\dot{s} = A \cdot s + B \cdot u$$

$$A = \begin{bmatrix} 0 & I_3 & 0 & \cdots & 0 \\ 0 & 0 & I_3 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & I_3 \\ 0 & \cdots & \cdots & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ I_3 \end{bmatrix}$$



Discretize acceleration

$$v_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$



Discretize jerk

$$v_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, a_0 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

- 高阶积分器
- 幂零（nilpotent）

Search-based Motion Planning for Quadrotors using Linear Quadratic Minimum Time Control, Sikang Liu, Nikolay Atanasov, Kartik Mohta, and Vijay Kumar

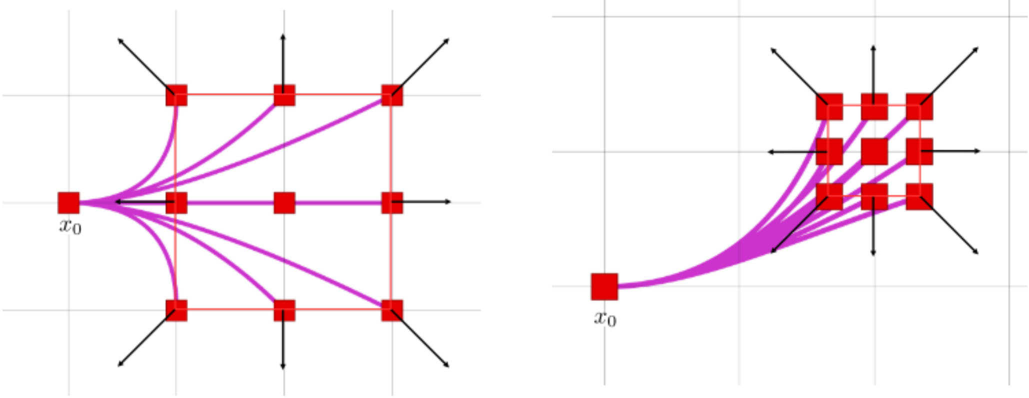
控制空间采样

State: $s = \begin{pmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix}$ **Input**: $u = \begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix}$

System equation: $\dot{s} = A \cdot s + B \cdot u$

$$\dot{s} = A \cdot s + B \cdot u$$

$$A = \begin{bmatrix} 0 & I_3 & 0 & \cdots & 0 \\ 0 & 0 & I_3 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & I_3 \\ 0 & \cdots & \cdots & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ I_3 \end{bmatrix}$$



$$s(t) = \underbrace{e^{At}}_{F(t)} s_0 + \underbrace{\left[\int_0^t e^{A(t-\sigma)} B d\sigma \right]}_{G(t)} u_m$$

e^{At}：状态转移矩阵，对积分至关重要

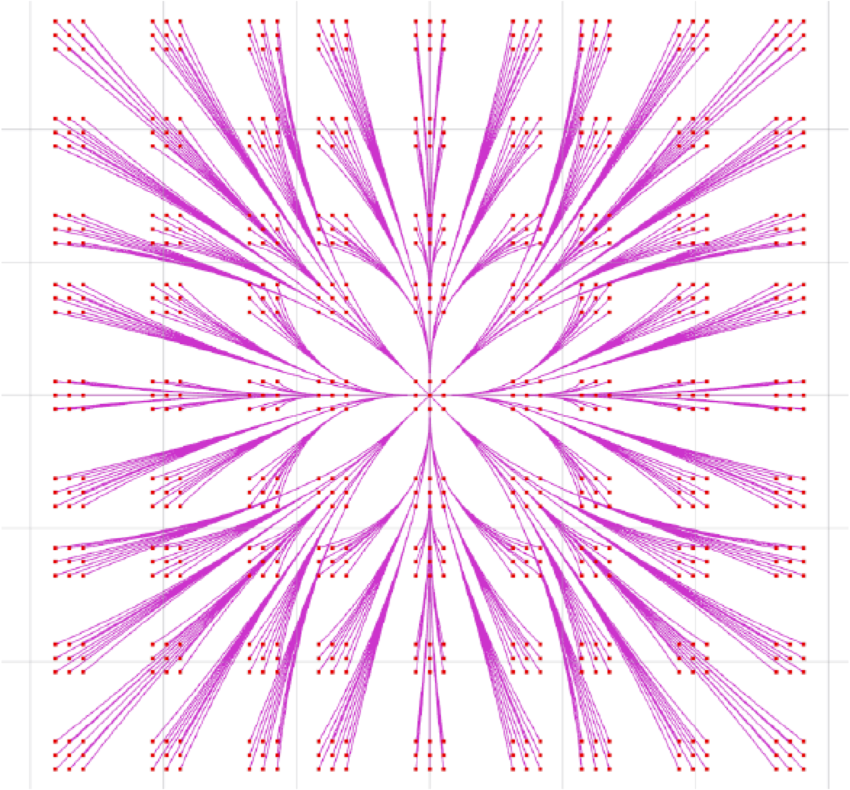
$$\boxed{e^{At} = I + \frac{At}{1!} + \frac{(At)^2}{2!} + \frac{(At)^3}{3!} + \cdots + \frac{(At)^k}{k!} + \cdots}$$

如果矩阵$A \in R^{n \times n}$是幂零的（nilpotent），如$A^n = 0$，那么e^{At}就有一个以t为变量的$(n-1)$次矩阵多项式形式的闭式表达

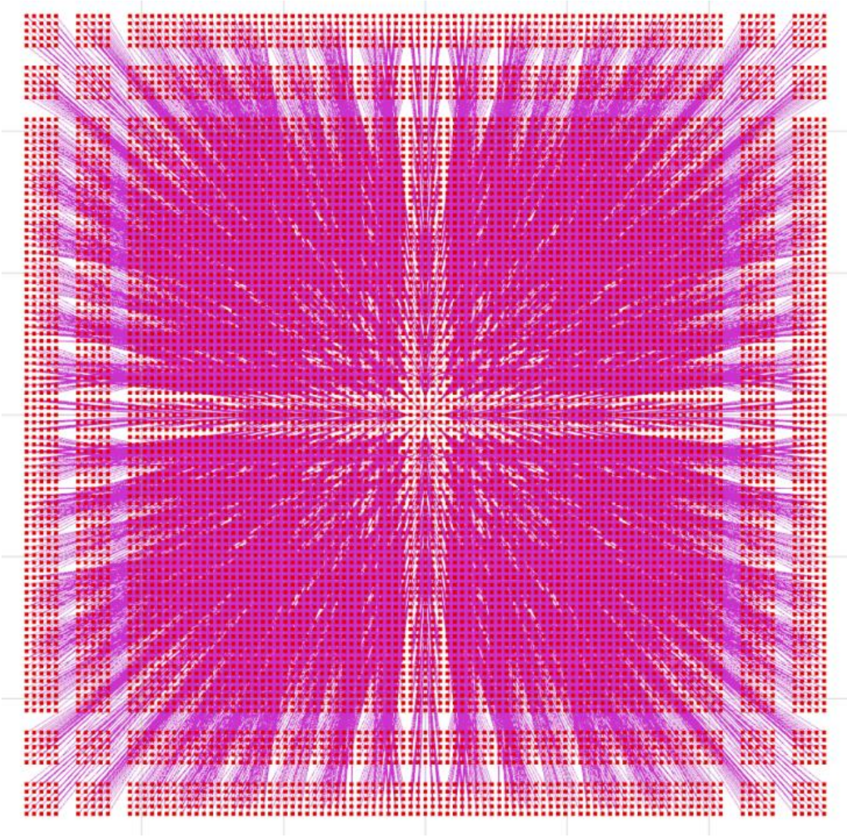
Search-based Motion Planning for Quadrotors using Linear Quadratic Minimum Time Control, Sikang Liu, Nikolay Atanasov, Kartik Mohta, and Vijay Kumar

控制空间采样

通过搜索得到的图(lattice graph)



9 discretization



25 discretization

Note

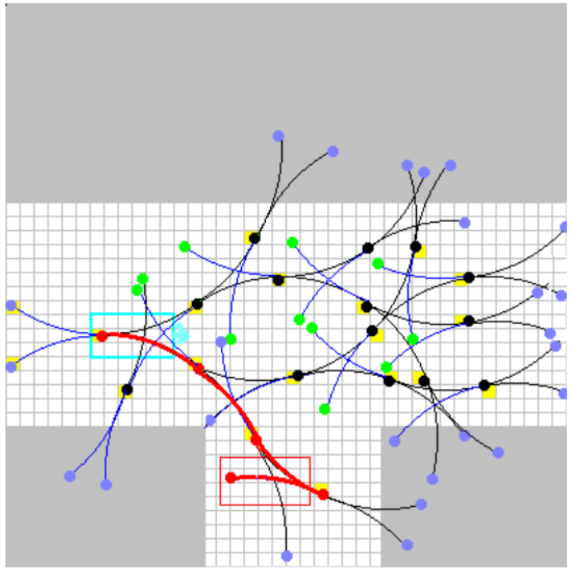
- 在搜索的过程中可以根据需要构建图
- 当新发现节点时，创建节点(状态 state)和边(运动基元 motion primitive)
- 节省计算时间/空间

控制空间采样

- 从搜索树中找到 $s \in T$
- 选择控制输入 u
- 固定时间，积分
- 将无碰撞运动添加到搜索树

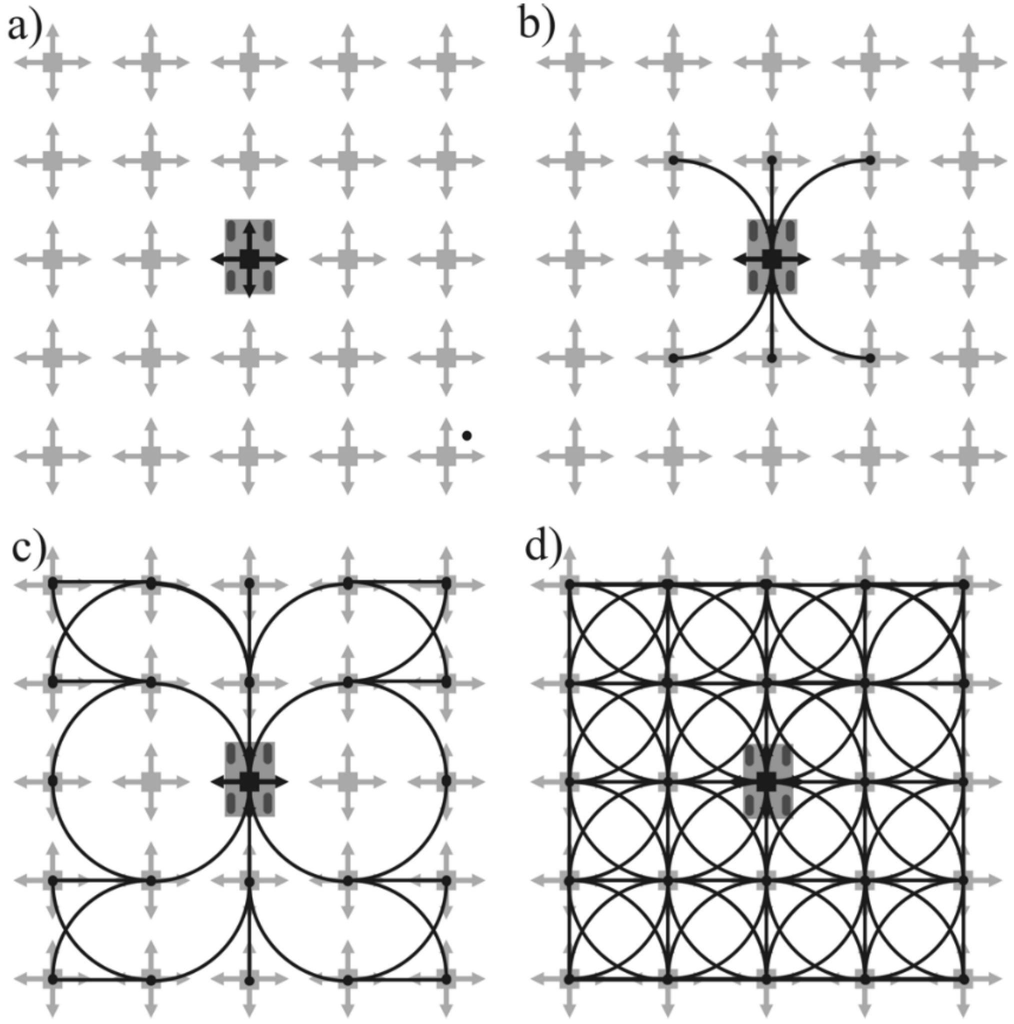
$$\text{状态}: s = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \qquad \text{输入}: u = \begin{pmatrix} v \\ \emptyset \end{pmatrix}$$

$$\text{系统模型}: \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} v \cdot cos\theta \\ v \cdot sin\theta \\ \frac{r}{L} \cdot tan\emptyset \end{pmatrix}$$



1) Select a $s \in T$
2) Pick $v, \emptyset$ and τ
3) Integrate motion from s
4) Add result if collision-free

状态空间采样



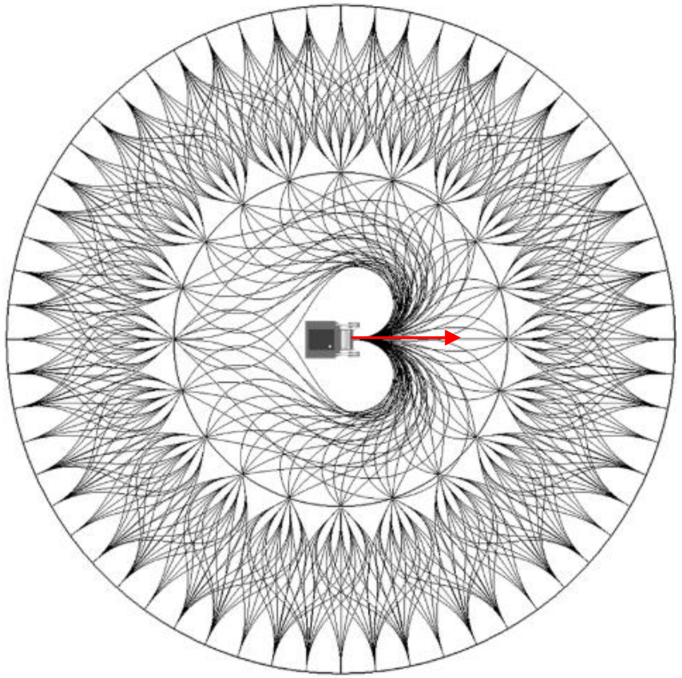
建立 lattice graph:

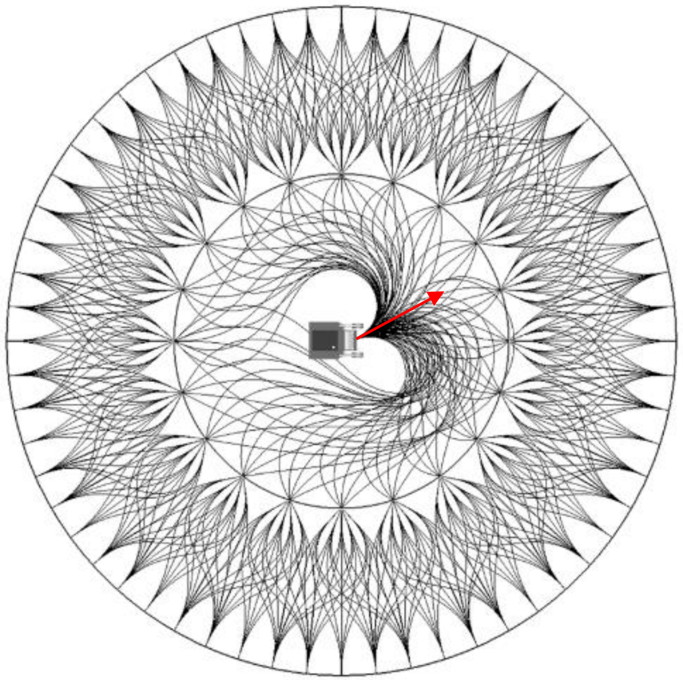
- 8个相邻节点，找到了可行的路
- 向外延伸到24个邻居.

Reeds-Shepp Car Model

Generating Near Minimal Spanning Control Sets for Constrained Motion Planning in Discrete State Spaces, Mihail Pivtoraiko and Alonzo Kelly

控制空间采样

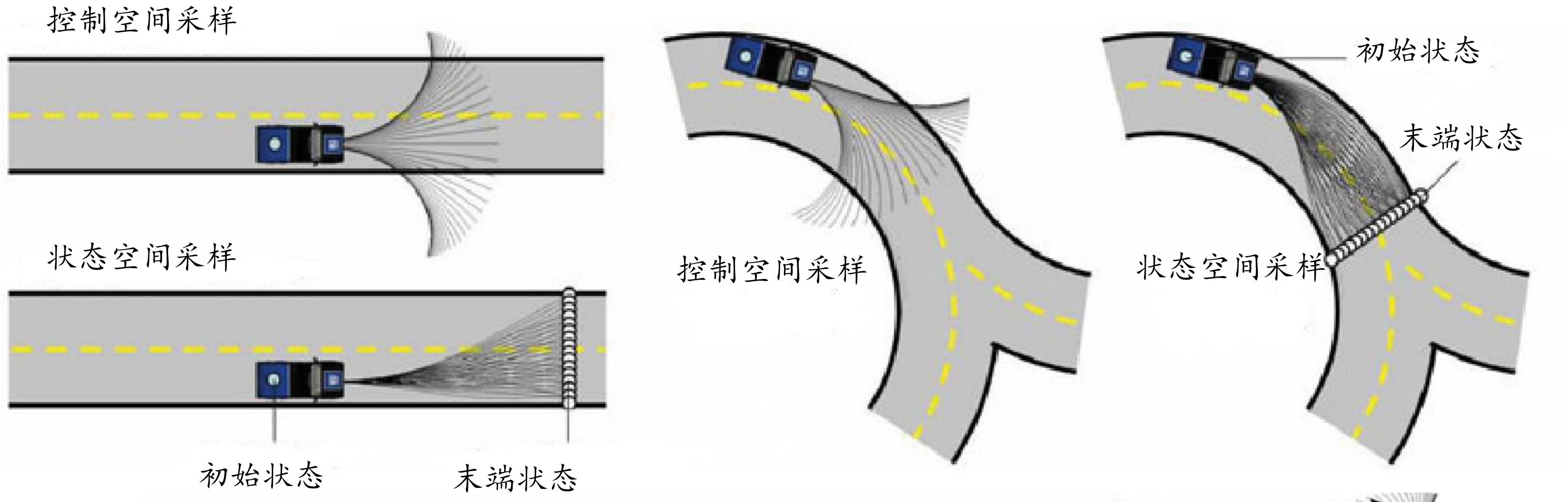


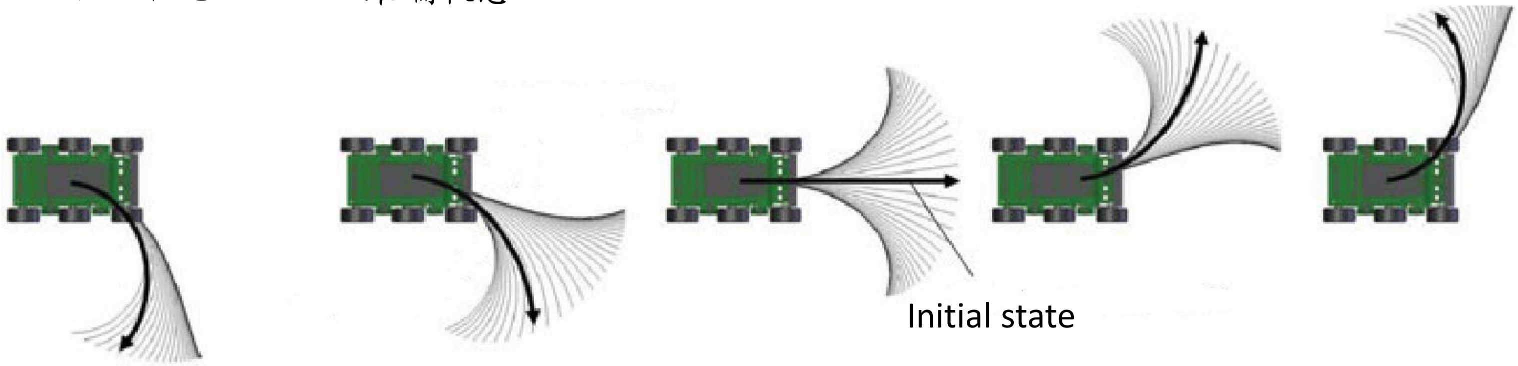


- 2层 lattice graph
- 只有第一层不一样
- 初始状态不同

Optimal Rough Terrain Trajectory Generation for Wheeled Mobile Robots, Thomas M. Howard Alonzo Kelly

控制空间采样 vs 状态空间采样





- 轨迹在初始角速度的方向上密度更大。
- 几个不同输入的输出非常相似。

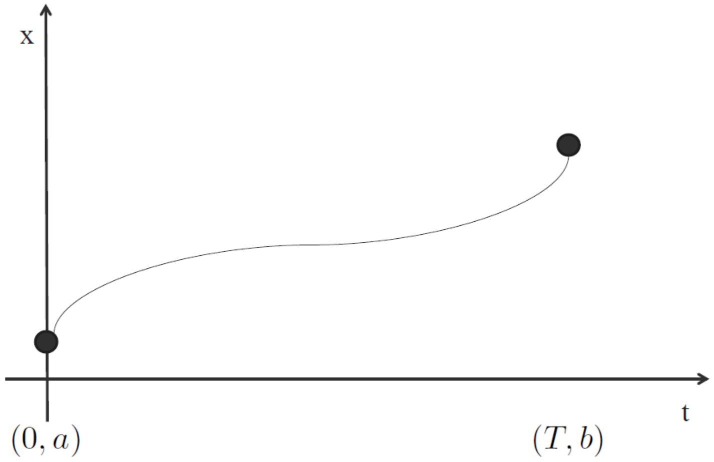
State Space Sampling of Feasible Motions for High-Performance Mobile Robot Navigation in Complex Environments, Thomas M. Howard, Colin J. Green, and Alonzo Kelly

Boundary Value Problem

边值问题 Boundary Value Problem (BVP)

- BVP是状态空间采样规划的基础
- 没有通用的解决方案
- 复杂的数值优化

- 设计一条轨迹$x(t)$ 使得:
  - $x(0) = a$
  - $x(T) = b$



Pontryagin's 最小值原理

优化目标:

$$J = \boxed{h(s(T))} + \boxed{\int_0^T g(s(t),u(t)) \cdot dt}$$

最终状态　　转移代价

写出哈密顿量和状态

$$H(s,u,\lambda) = g(s,u) + \lambda^T f(s,u)$$

$$\lambda = (\lambda_1, \lambda_2, \lambda_3)$$

定义:

s^*: 最优状态

u^*: 最优输入

最小值原理

$$\dot{s}^*(t) = f(s^*(t), u^*(t)), \quad given: s^*(0) = s(0)$$

$\lambda(t)$ 是如下公式的解:

$$\dot{\lambda}(t) = -\nabla_s H(s^*(t), u^*(t), \lambda(t))$$

有着边值条件为

$$\lambda(T) = \nabla h(s^*(T))$$

最优控制输入为

$$u^*(t) = \arg\min_{u(t)} H(s^*(t), u(t), \lambda(t))$$

A Computationally Efficient Motion Primitive for Quadrocopter Trajectory Generation, **Mark W. Mueller, Markus Hehn, and Raffaello D'Andrea**

Dynamic Programming and Optimal Control, **D. P. Bertsekas**

最优边值问题(OBVP)

Optimal Boundary Value Problem (OBVP)

建模

目标：最小化jerk平方的积分

$$J_\Sigma = \sum_{k=1}^{3} J_k,\quad J_k = \frac{1}{T}\int_0^T j_k(t)^2 dt.$$

状态: $s_k = (p_k, v_k, a_k)$ 输入: $u_k = j_k$

系统模型: $\dot{s}_k = f_s(s_k, u_k) = (v_k, a_k, j_k)$

求解

通过 Pontryain’s 最小值原理，我们引入协态:

$$\lambda = (\lambda_1, \lambda_2, \lambda_3)$$

定义哈密顿方程

$$H(s,u,\lambda) = \frac{1}{T} j^2 + \lambda^T f_s(s,u)$$

$$= \frac{1}{T} j^2 + \lambda_1 v + \lambda_2 a + \lambda_3 j$$

最小值原理

$$\dot{s}^*(t) = f(s^*(t), u^*(t)),\quad given: s^*(0) = s(0)$$

$\lambda(t)$ 是如下公式的解:

$$\dot{\lambda}(t) = -\nabla_s H(s^*(t), u^*(t), \lambda(t))$$

有着边值条件为

$$\lambda(T) = \nabla h(s^*(T))$$

最优控制输入为

$$u^*(t) = \arg\min_{u(t)} H(s^*(t), u(t), \lambda(t))$$

A Computationally Efficient Motion Primitive for Quadrocopter Trajectory Generation, **Mark W. Mueller, Markus Hehn, and Raffaello D’Andrea**

Dynamic Programming and Optimal Control, **D. P. Bertsekas**

最优边值问题(OBVP)

建模

目标：最小化jerk平方的积分

$$J_\Sigma = \sum_{k=1}^{3} J_k,\quad J_k = \frac{1}{T}\int_0^T j_k(t)^2 dt.$$

状态: $s_k = (p_k, v_k, a_k)$ 输入: $u_k = j_k$

系统模型: $\dot{s}_k = f_s(s_k, u_k) = (v_k, a_k, j_k)$

求解

通过 $\boxed{\text{Pontryain's 最小值原理,}}$ 我们引入协态:

$$\lambda = (\lambda_1, \lambda_2, \lambda_3)$$

定义哈密顿方程

$$H(s,u,\lambda) = \frac{1}{T} j^2 + \lambda^T f_s(s,u)$$

$$= \frac{1}{T} j^2 + \lambda_1 v + \lambda_2 a + \lambda_3 j$$

$$\dot{\lambda} = -\nabla_s H(\boxed{s^*}, \boxed{u^*}, \lambda) = (0, -\lambda_1, -\lambda_2)$$

最优状态　最优输入

协态可以被如下公式求解

$$\lambda(t) = \frac{1}{T}\begin{bmatrix} -2\alpha \\ 2\alpha t + 2\beta \\ -\alpha t^2 - 2\beta t - 2\gamma \end{bmatrix}$$

最优控制输入可被如下公式求解

$$u^*(t) = j^*(t) = \arg\min_{j(t)} H(s^*(t), j(t), \lambda(t))$$

$$= \frac{1}{2}\alpha t^2 + \beta t + \gamma$$

最优状态轨迹可被如下公式求解

$$s^*(t) = \begin{bmatrix} \frac{\alpha}{120} t^5 + \frac{\beta}{24} t^4 + \frac{\gamma}{6} t^3 + \frac{a_0}{2} t^2 + v_0 t + p_0 \\ \frac{\alpha}{24} t^4 + \frac{\beta}{6} t^3 + \frac{\gamma}{2} t^2 + a_0 t + v_0 \\ \frac{\alpha}{6} t^3 + \frac{\beta}{2} t^2 + \gamma t + a_0 \end{bmatrix}$$

初始状态: $s(0) = (p_0, v_0, a_0)$

A Computationally Efficient Motion Primitive for Quadrocopter Trajectory Generation, **Mark W. Mueller, Markus Hehn, and Raffaello D'Andrea**

Dynamic Programming and Optimal Control, **D. P. Bertsekas**

最优边值问题(OBVP)

代价函数

$$J=\gamma^2+\beta\gamma T+\frac{1}{3}\beta^2T^2+\frac{1}{3}\alpha\gamma T^2+\frac{1}{4}\alpha\beta T^3+\frac{1}{20}\alpha^2T^4$$

J 仅由T和已知的边界状态决定, 因此我们甚至可以获得最优的 T!

How?

多项式函数求根问题

α,β,γ 被如下公式求解:

$$\begin{bmatrix}\frac{1}{120}T^5 & \frac{1}{24}T^4 & \frac{1}{6}T^3\\ \frac{1}{24}T^4 & \frac{1}{6}T^3 & \frac{1}{2}T^2\\ \frac{1}{6}T^3 & \frac{1}{2}T^2 & T\end{bmatrix}\begin{bmatrix}\alpha\\ \beta\\ \gamma\end{bmatrix}=\begin{bmatrix}\Delta p\\ \Delta v\\ \Delta a\end{bmatrix}$$

$$\begin{bmatrix}\Delta p\\ \Delta v\\ \Delta a\end{bmatrix}=\begin{bmatrix}p_f-p_0-v_0T-\frac{1}{2}a_0T^2\\ v_f-v_0-a_0T\\ a_f-a_0\end{bmatrix}$$

$$\Rightarrow$$

$$\begin{bmatrix}\alpha\\ \beta\\ \gamma\end{bmatrix}=\frac{1}{T^5}\begin{bmatrix}720 & -360T & 60T^2\\ -360T & 168T^2 & -24T^3\\ 60T^2 & -24T^3 & 3T^4\end{bmatrix}\begin{bmatrix}\Delta p\\ \Delta v\\ \Delta a\end{bmatrix}$$

这个推导适用于固定的最终状态: $s(T)=(p_f,v_f,a_f)$

最优边值问题(OBVP)

- 前文是关于最终状态固定的问题.
- 那么当最终状态是（部分）自由的时候如何处理?
- 来关注一下边界条件在哪里?

$$\lambda(\mathrm{T})=\nabla h(s^*(T))$$

对于最终状态固定的问题

$$h(s(T))=\begin{cases}0, & if \quad s=s(T)\\ \infty, & otherwise\end{cases}$$ 不可微

因此我们放弃这个条件，
用已知的$x(T)$ 来直接求解未知变量

$$s^*(t)=\begin{bmatrix}\frac{\alpha}{120}t^5+\frac{\beta}{24}t^4+\frac{\gamma}{6}t^3+\frac{a_0}{2}t^2+v_0t+p_0\\ \frac{\alpha}{24}t^4+\frac{\beta}{6}t^3+\frac{\gamma}{2}t^2+a_0t+v_0\\ \frac{\alpha}{6}t^3+\frac{\beta}{2}t^2+\gamma t+a_0\end{bmatrix}$$

对于（部分）自由最终条件的问题

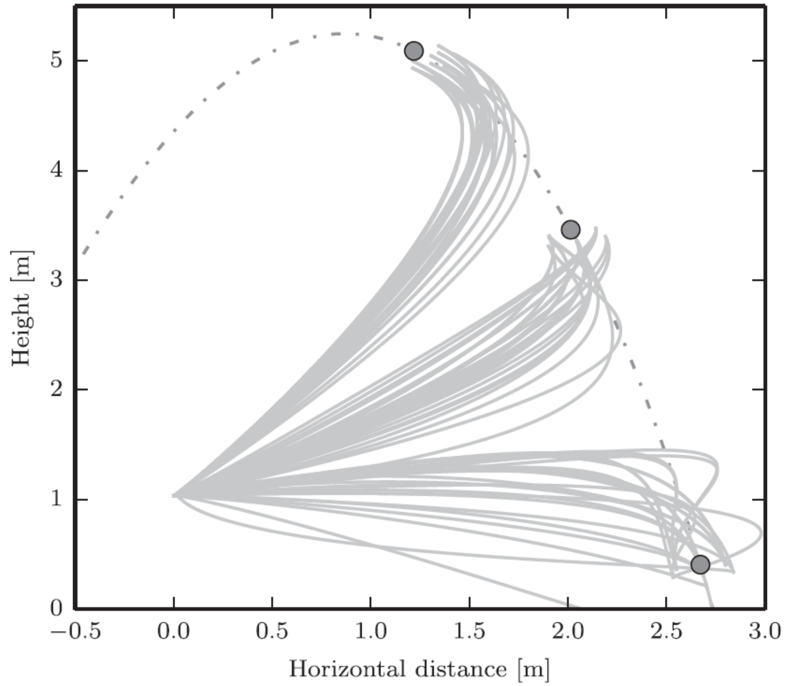
$$given \quad s_i(T),\ i\in I$$

我们对其他协态有边界条件

$$\lambda_j(T)=\frac{\partial h(s^*(T))}{\partial s_j},\quad for\ j\neq i$$

然后求解即可

边值问题 Boundary Value Problem (BVP)



------------------- 边值条件

这里只考虑边值条件

BVP经常被应用于基于运动基元的轨迹规划方法

我们的最优条件显示，解为

一个 5 阶多项式 当$s = 3$ (minimum jerk)

一个 7阶多项式 当$s = 4$ (minimum snap)

Wang et al., Geometrically Constrained Trajectory Optimization for Multicopters, TRO 2022

Boundary-intermediate Value Problem (BIVP)

Position

Velocity

Acceleration

Jerk

Snap

Crackle

Pop

$\frac{\mathrm{d}}{\mathrm{d}t}$

边值条件: --

中值条件: --

BIVP经常被应用于基于运动基元的轨迹规划方法

我们的最优条件表明，只有中间路径点条件情况下

分段5阶多项式轨迹有着连续的snap当 s=3 (minimum jerk trajectory)

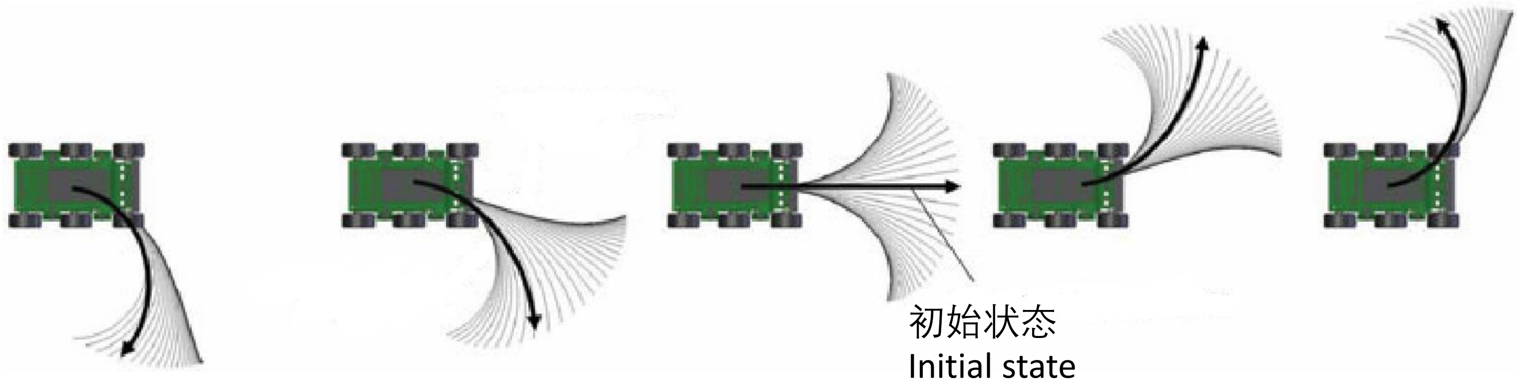
分段7阶多项式轨迹有着连续的pop 当 s=4 (minimum snap trajectory)

Wang et al., Geometrically Constrained Trajectory Optimization for Multicopters, TRO 2022

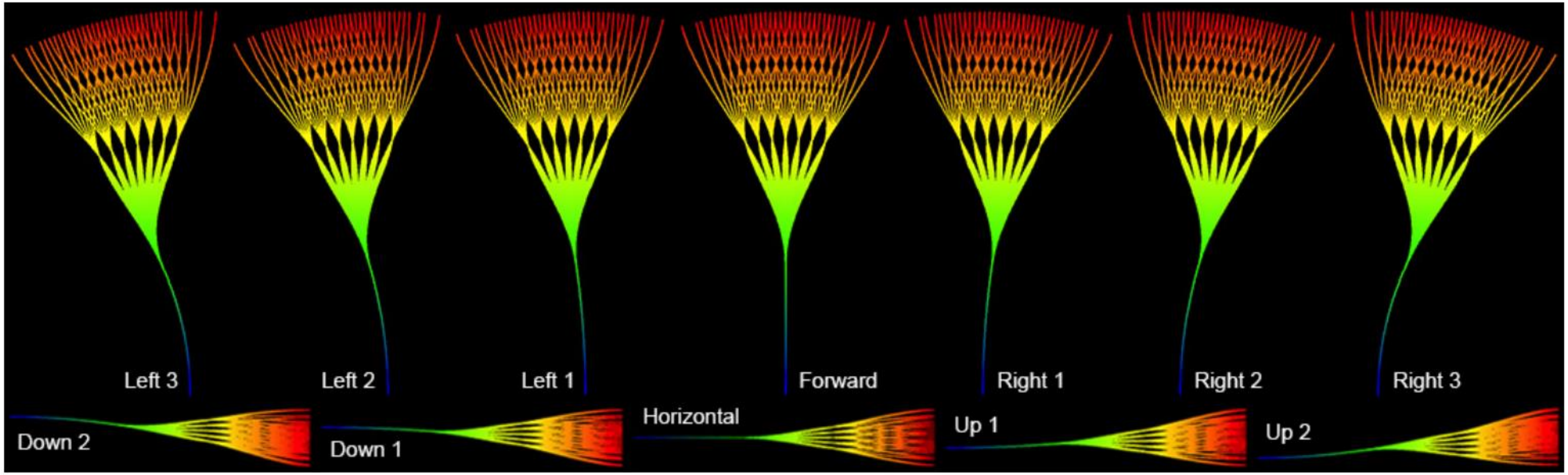
轨迹库（Trajectory library）

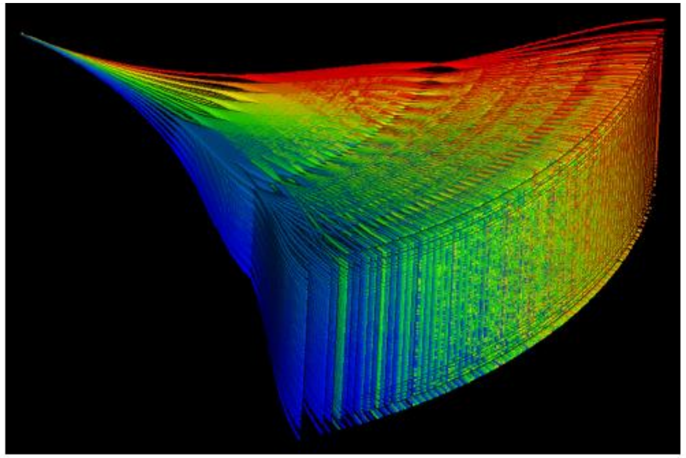
- 单层lattice planning是一种常见的用于局部避障的选择
- 不需要图搜索，只需要轨迹选择
- 基于多项成本函数对每个轨迹进行评级

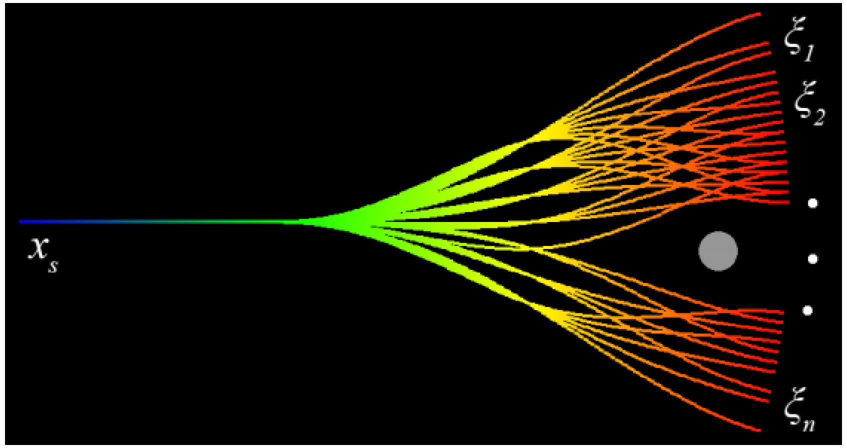
碰撞风险、信息获取、舒适度、能量等



轨迹库（Trajectory library）







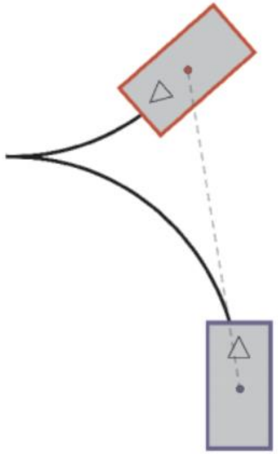
Maximum Likelihood Path Planning for Fast Aerial Maneuvers and Collision Avoidance Ji Zhang, Chen Hu, Rushat Gupta Chadha, and Sanjiv Singh

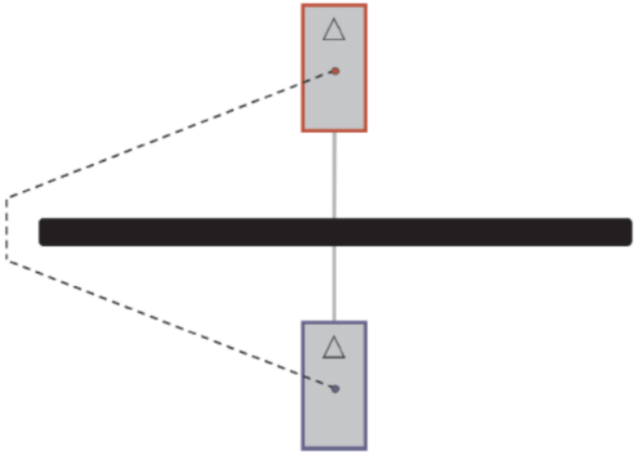
启发式函数

启发式函数设计

问题简化

- 不考虑动力学
- 不考虑障碍物

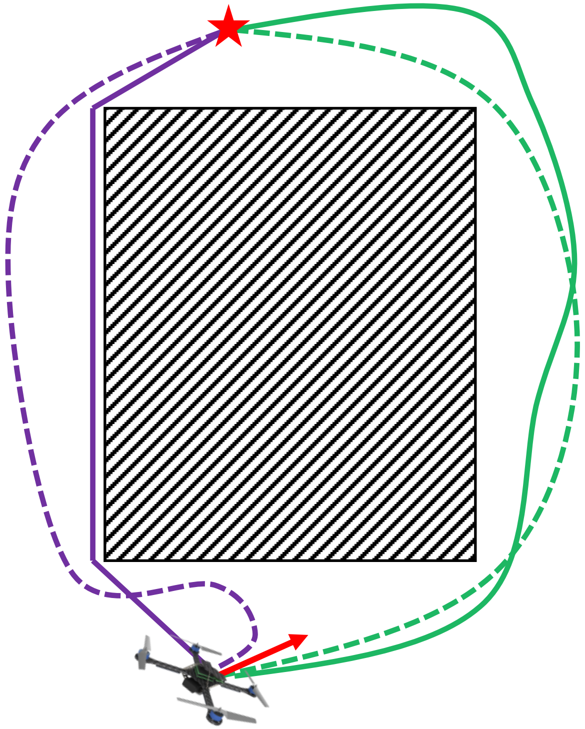




Planning Long Dynamically Feasible Maneuvers for Autonomous Vehicles, Maxim Likhachev, Dave Ferguson

启发式函数设计

对于每个节点（状态），**不考虑动力学**模型，搜索其最短路径



启发式函数设计

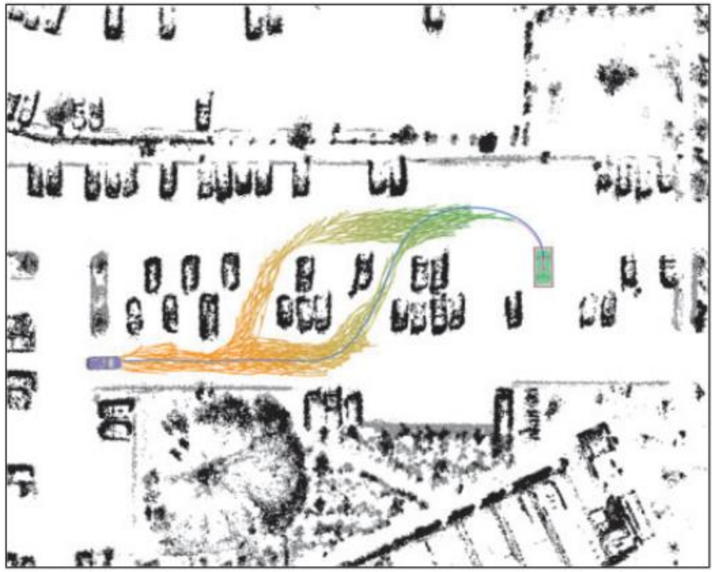
对于每个节点（状态），将OBVP作为启发式函数求解到规划目标状态h

- Maintain a priority queue to store all the nodes to be expanded
- The heuristic function h(n) for all nodes are pre-defined
- The priority queue is initialized with the start state X_S
- Assign $g(X_S)=0$, and g(n)=infinite for all other nodes in the graph
- Loop
  - If the queue is empty, return FALSE; break;
  - Remove the node "n" with the lowest $f(n)=\boxed{g(n)}+\boxed{h(n)}$ from the priority queue
  - Mark node "n" as expanded
  - If the node "n" is the goal state, return TRUE; break;
  - For all unexpanded neighbors "m" of node "n"
    - If g(m) = infinite
      - g(m)= g(n) + Cnm
      - Push node "m" into the queue
    - If $g(m) > g(n) + C_{nm}$
      - g(m)= g(n) + Cnm
  - end
- End Loop

Accumulate cost

对比

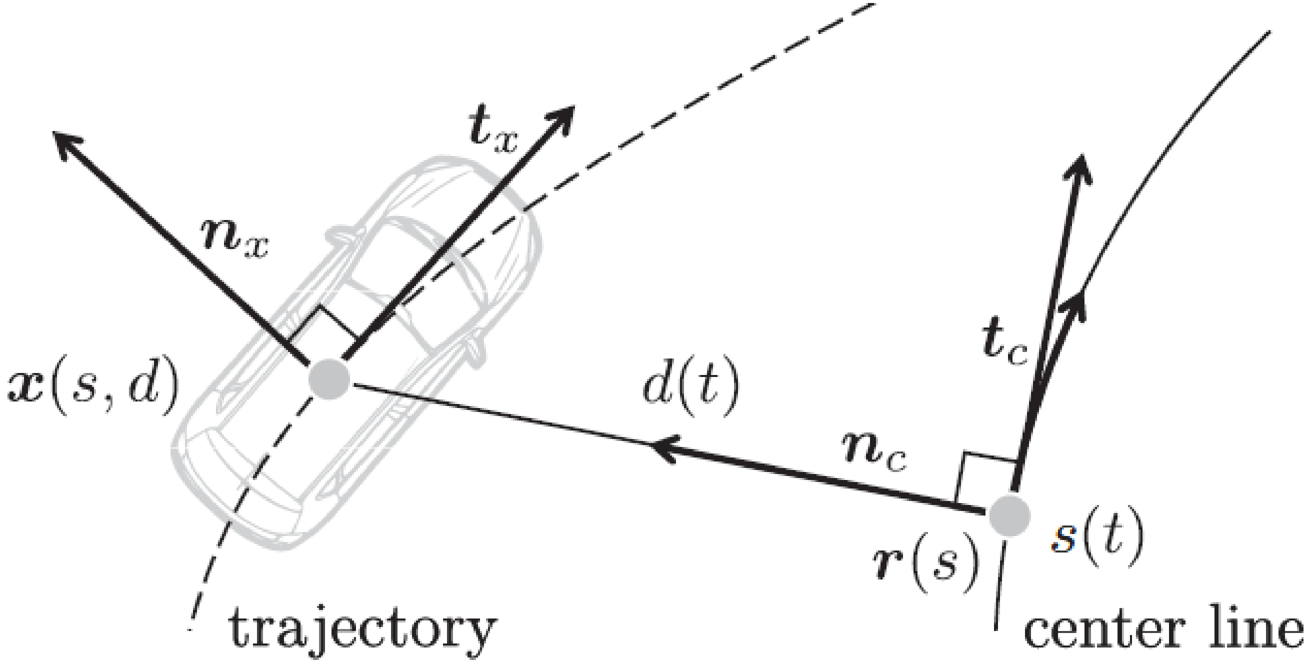
欧式二维距离



non-holonomic-without-obstacles

Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments, Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, James Diebel

基于Frenet坐标系的规划



- 动态坐标系
- 切向和径向独立
- 对于车道跟随问题
- 是解耦的

✓ 运动/控制参数化

✓ ：五次多项式

$$d(t) = a_{d0} + a_{d1}t + a_{d2}t^2 + a_{d3}t^3 + a_{d4}t^4 + a_{d5}t^5$$

$$s(t) = a_{s0} + a_{s1}t + a_{s2}t^2 + a_{s3}t^3 + a_{s4}t^4 + a_{s5}t^5$$

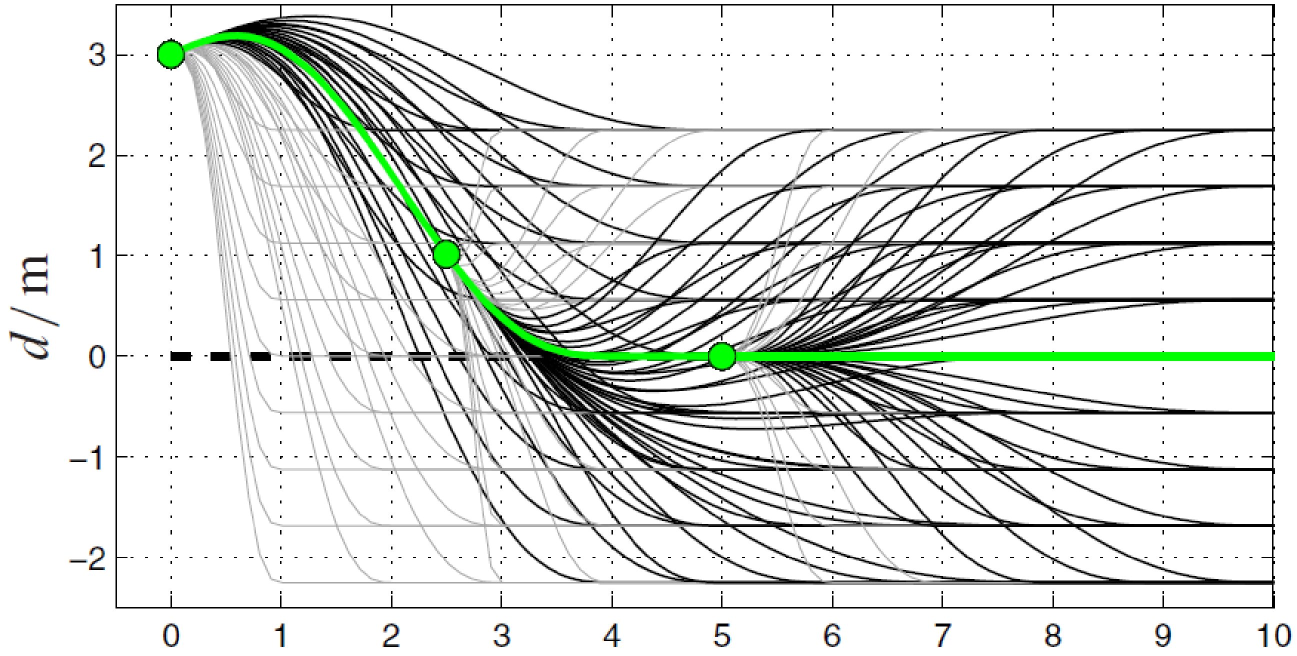
✓ 解最优控制问题

这里只讨论横向规划，
径向规划请参考：

Optimal Trajectory Generation for Dynamic Street Scenarios in a Frenet Frame, Moritz Werling, Julius Ziegler, Sören Kammel, and Sebastian Thrun

Optimal trajectories for time-critical street scenarios using discretized terminal manifolds, Moritz Werling, Sören Kammel, Julius Ziegler and Lutz Gröll

基于Frenet坐标系的规划



Lateral trajectory

$$d(t) = a_{d0} + a_{d1}t + a_{d2}t^2 + a_{d3}t^3 + a_{d4}t^4 + a_{d5}t^5$$

初始条件:

$$D(0) = \begin{pmatrix} d_0 & \dot{d}_0 & \ddot{d}_0 \end{pmatrix}$$

终止条件:

$$D(T) = \begin{pmatrix} d_f & \dot{d}_f & \ddot{d}_f \end{pmatrix}$$

车道跟随:

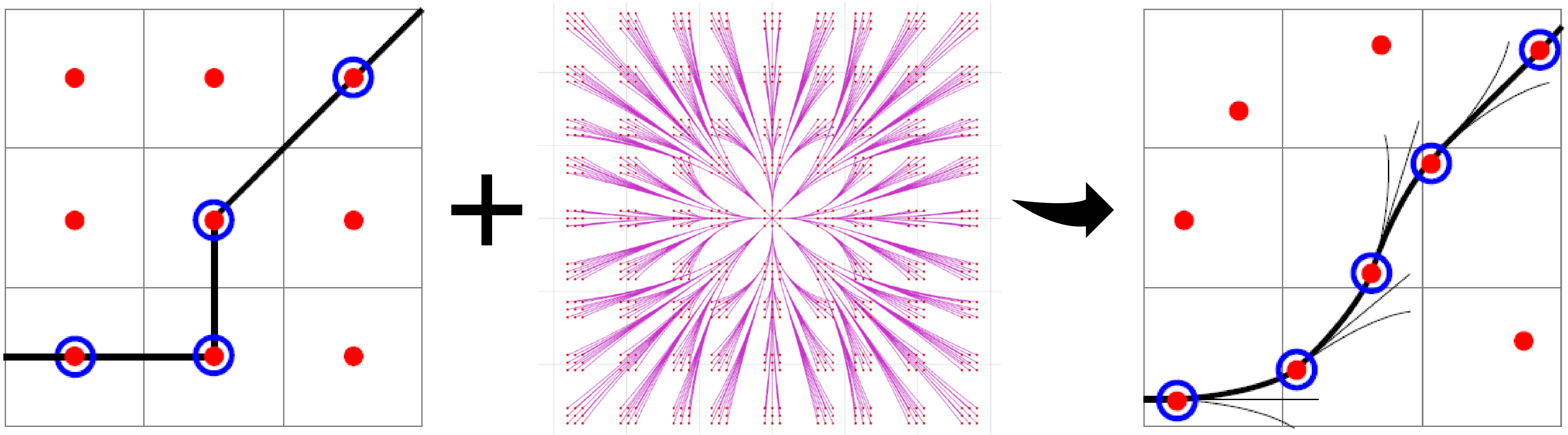
$$D(T) = \begin{pmatrix} d_f & 0 & 0 \end{pmatrix}$$

$$\begin{bmatrix} T^3 & T^4 & T^5 \\ 3T^2 & 4T^3 & 5T^4 \\ 6T & 12T^2 & 20T^3 \end{bmatrix} \begin{bmatrix} a_{d3} \\ a_{d4} \\ a_{d5} \end{bmatrix} = \begin{bmatrix} \Delta p \\ \Delta v \\ \Delta a \end{bmatrix} \qquad \begin{bmatrix} \Delta p \\ \Delta v \\ \Delta a \end{bmatrix} = \begin{bmatrix} d_f - (d_0 + \dot{d}_0 T + \frac{1}{2}\ddot{d}_0 T^2) \\ \dot{d}_f - (\dot{d}_0 + \ddot{d}_0 T) \\ \ddot{d}_f - \ddot{d}_0 \end{bmatrix}$$

Hybrid A* Workflow

基本思路

- 在线生成密集栅格需要花费太多时间
- 聚集，修剪一些怎么样？
- 定义剪枝的规则，利用栅格地图



Practical Search Techniques in Path Planning for Autonomous Driving, Dmitri Dolgov, Sebastian Thrun
Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments, Dmitri Dolgov, Sebastian Thrun

Hybrid A*

- Maintain a priority queue to store all the nodes to be expanded
- The heuristic function h(n) for all nodes are pre-defined
- The priority queue is initialized with the start state X_S
- Assign $g(X_S)=0$, and g(n)=infinite for all other nodes in the graph
- Loop
  - If the queue is empty, return FALSE; break;
  - Remove the node “n” with the lowest f(n)=g(n)+h(n) from the priority queue
  - Mark node “n” as expanded
  - If the node “n” is the goal state, return TRUE; break;
  - For all unexpanded neighbors “m” of node “n”
    - If g(m) = infinite
      - $g(m)= g(n) + C_{nm}$
      - Push node “m” into the queue

      +
    - If $g(m) > g(n) + C_{nm}$
      - $g(m)= g(n) + C_{nm}$

      +
  - end
- End Loop

选择合适的启发式方法

通过对节点中的状态进行前向积分来查找邻居

记录节点“m”内部的状态

更新节点“m”内部的状态

Hybrid A*启发式函数设计


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

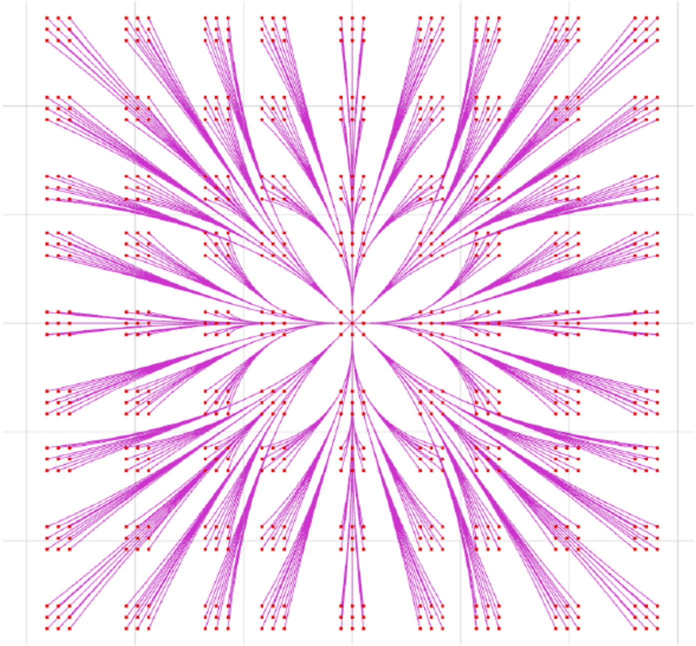
(a) 2D欧式距离

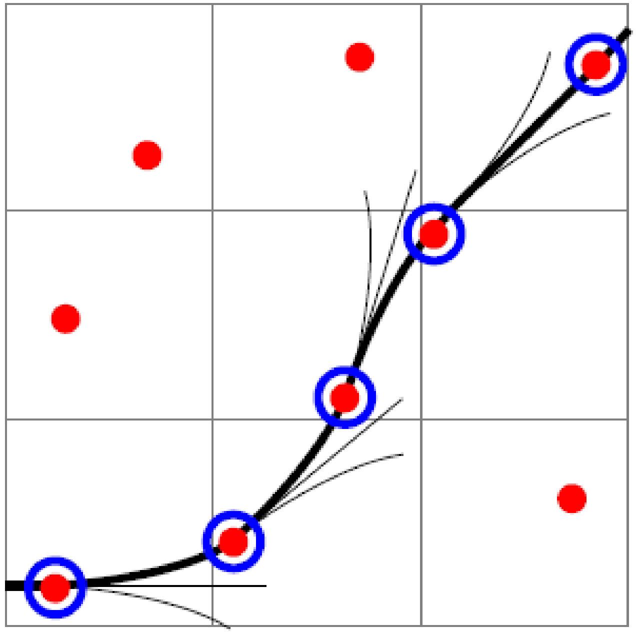
(b) 不考虑障碍物

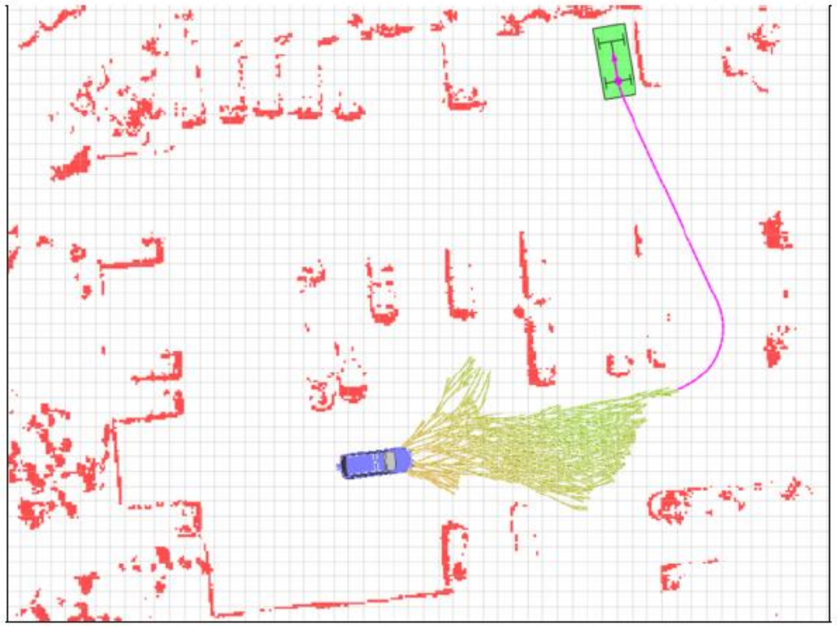
(c) 不考虑障碍的非完整性，在死胡同中表现不佳

(d) 不考虑障碍＋（2D最短路径）

Hybrid A*启发式函数设计







一次性启发 **One shot**

Kinodynamic RRT*

Workflow

Similar to RRT* but different in details

Algorithm 2: RRT Algorithm

Input: $\mathcal{M}$,x_{init},x_{goal}
Result: A path Γ from x_{init} to x_{goal}
$\mathcal{T}$.init();
for $i = 1$ *to* n **do**
  $x_{rand} \leftarrow Sample(\mathcal{M})$;
  $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
  $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
  if *CollisionFree*(x_{new}) **then**
    $X_{near} \leftarrow NearC(\mathcal{T}, x_{new})$;
    $x_{min} \leftarrow ChooseParent(X_{near}, x_{near}, x_{new})$;
    $\mathcal{T}.addNodEdge(x_{min}, x_{new})$;
    $\mathcal{T}.rewire()$;

Kinodynamic RRT*

Input: E, x_init, x_goal

Output: A trajectory T from x_init to x_goal

T.init();
for i = 1 to n **do**
  x_rand ← Sample(E);
  X_near ← Near(T, x_rand);
  x_min ← ChooseParent(X_near, x_rand);
  T.addNode(x_rand);
  T.rewire();

1. How to “Sample”

Kinodynamic RRT*

Input: E, x_init, x_goal

Output: A trajectory T from x_init to x_goal

```
T.init();
for i = 1 to n do
    x_rand ← Sample(E);
    X_near ← Near(T, x_rand);
    x_min ← ChooseParent(X_near, x_rand);
    T.addNode(x_rand);
    T.rewire();
```

LTI 系统状态空间方程：

$$\dot{x}(t) = A\mathrm{x}(t) + Bu(t) + c$$

如双积分系统：

$$x = \begin{bmatrix} p \\ v \end{bmatrix}, A = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ I \end{bmatrix}$$

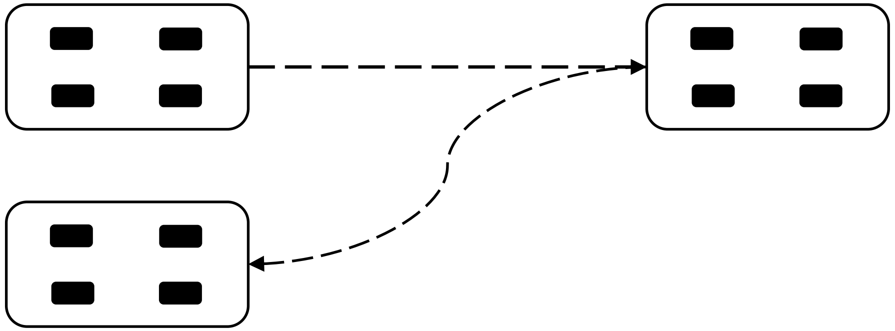
不像RRT 只在位置空间进行采样，还需要采样速度

sample in full state space.

2. How to define “Near”

Kinodynamic RRT*
Input: E, x_init, x_goal
Output: A trajectory T from x_init to x_goal

```
T.init();
for i = 1 to n do
    x_rand ← Sample(E);
    X_near ← Near(T, x_rand);
    x_min ← ChooseParent(X_near, x_rand);
    T.addNode(x_rand);
    T.rewire();
```



小车不能侧向移动

- 如果没有运动约束，可以使用欧氏距离或曼哈顿距离。
- 在具有运动约束的状态空间中，引入**最优控制**。

如何要实现最优控制：可以定义状态转移的 cost functions

$$c[\pi] = \int_0^{\tau} \left(1 + u(t)^T R u(t)\right) dt$$

通常，采用时间-能量最优的二次形式。

如果从一个状态转移到另一个状态的 cost 很低，则两个状态接近。
（注意：如果状态反向转换， cost 可能会有所不同）

2. How to define “Near”

$$c[\pi]=\int_0^{\tau}\left(1+u(t)^T R u(t)\right)dt$$

如果知道到达时间 **τ** 和控制输入 **u(t)** ，就可以计算cost 。

经典的最优控制解决方案。(OBVP)

2. How to define “Near”

- 固定终点状态 x_1，固定到达时间 τ

最优控制输入 $u^*(t)$

$$u^*(t) = R^{-1}B^T exp[A^T(\tau - t)]G(\tau)^{-1}[x_1 - \bar{x}(\tau)].$$

其中$G(t)$）权重:

$$G(t) = \int_0^t exp[A(t-t')]BR^{-1}B^T exp[A^T(t-t')]dt'.$$

李雅普诺夫方程方程的解:

$$\dot{G}(t) = AG(t) + G(t)A^T + BR^{-1}B^T, G(0) = 0.$$

F. Lewis, V. Syrmos. Optimal Control. John Wiley & Sons

2. How to define “Near”

- 固定终点状态 x_1，固定到达时间 τ

$$u^*(t) = R^{-1}B^T exp[A^T(\tau - t)]G(\tau)^{-1}[x_1 - \bar{x}(\tau)].$$

$x\bar{}(t)$）为如果没有应用控制输入，x在t时的状态:

$$\bar{x}(t) = exp(At)x_0 + \int_0^t exp[A(t - t')]cdt'.$$

微分方程的解:

$$\dot{\bar{x}}(t) = A\bar{x}(t) + c, \bar{x}(0) = x_0$$

F. Lewis, V. Syrmos. Optimal Control. John Wiley & Sons

2. How to define “Near”

- 固定终点状态 x_1 ，改变到达时间 τ

如果要找到最佳到达时间τ,
可以通过改变控制输入$u^*(t)$，评估cost function $c[\pi]$ 来实现:

$$c[\tau] = \tau + [x_1 - \bar{x}(\tau)]^T G(t)^{-1} [x_1 - \bar{x}(\tau)].$$

最佳到达时间τ可以通过$c[\tau]$对τ求导得到:

$$\dot{c}[\tau] = 1 - 2(Ax_1 + c)^T d(\tau) - d(\tau)^T B R^{-1} B^T d(\tau).$$

其中：

$$d(\tau) = G(t)^{-1}[x_1 - \bar{x}(\tau)].$$

2. How to define "Near"

- 固定终点状态 x_1，改变到达时间 τ

求解 τ^*

$$\dot{c}[\tau] = 0.$$

$c[\tau]$可能有多个局部最小值

对于双积分系统，$c[\tau]$为4阶多项式。

给定最佳到达时间τ^*，

问题转变为固定终点状态 x_1，固定终点时间τ 问题。

3. How to "ChooseParent"

```
Kinodynamic RRT*
Input: E, x_init, x_goal
Output: A trajectory T from x_init to x_goal
T.init();
for i = 1 to n do
    x_rand ← Sample(E);
    X_near ← Near(T, x_rand);
    x_min ← ChooseParent(X_near, x_rand);
    T.addNode(x);
    T.rewire();
```

在高纬空间对一个随机状态进行采样，计算从树中当前节点到采样状态的控制输入和 cost 。

选择一个cost最低的，并**检查$\boldsymbol{x}(\boldsymbol{t})$ 和$\boldsymbol{u}(\boldsymbol{t})$ 在边界内**。

如果找不到合格的父节点，对另一个状态进行采样。

4. How to find near nodes efficiently

```
Kinodynamic RRT*
Input: E, x_init, x_goal
Output: A trajectory T from x_init to x_goal
T.init();
for i = 1 to n do
    x_rand ← Sample(E);
    X_near ← Near(T, x_rand);
    x_min ← ChooseParent(X_near, x_rand);
    T.addNode(x);
    T.rewire();
```

每次采样一个随机状态***x_rand***,

需要检查树中的每个节点以找到其父节点,
即为每个节点求解OBVP,

这是不有效的。

4. How to find near nodes efficiently

Kinodynamic RRT*

Input: E, x_init, x_goal

Output: A trajectory T from x_init to x_goal

```
T.init();
for i = 1 to n do
    x_rand ← Sample(E);
    X_near ← Near(T, x_rand);
    x_min ← ChooseParent(X_near, x_rand);
    T.addNode(x);
    T.rewire();
```

如果设置**cost tolerance** r,

可以计算当前状态的**前向可达集**，终止状态的**向后可达集**

如果以kd树的形式存储节点，就可以在树中进行范围查询。

4. How to find near nodes efficiently

$$c[\tau] = \tau + [x_1 - \bar{x}(\tau)]^T G(t)^{-1}[x_1 - \bar{x}(\tau)].$$

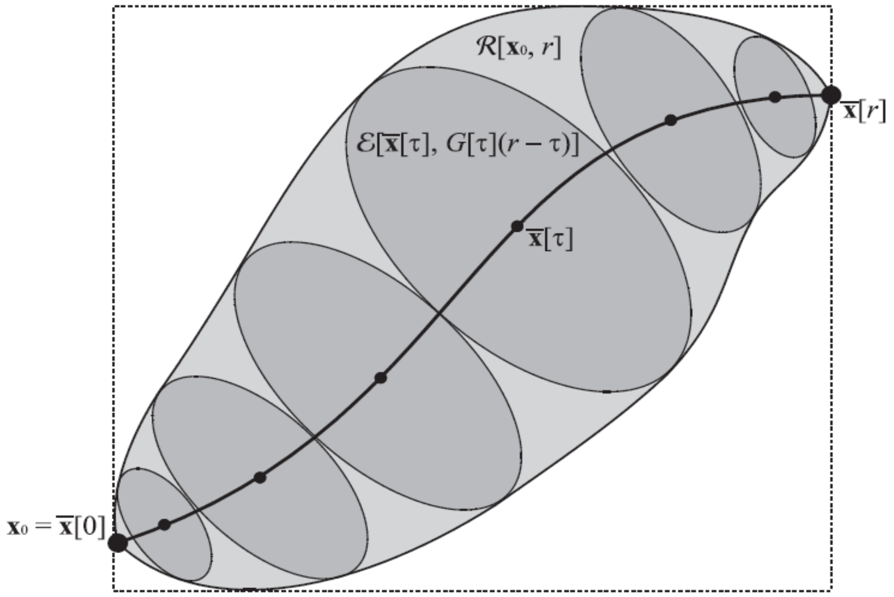
此公式描述了从状态$\boldsymbol{x_0}$到状态$\boldsymbol{x_1}$的cost，如何随着到达时间 $\boldsymbol{\tau}$ 的变化而变化。

给定初始状态$\boldsymbol{x_0}$ 、 cost tolerance $\boldsymbol{r}$ 和到达时间$\boldsymbol{\tau}$ ， $\boldsymbol{x_0}$ 的前向可达集合为:

$$\begin{aligned} & \{x_1 \,|\tau + [x_1 - \bar{x}(\tau)]^T G(t)^{-1}[x_1 - \bar{x}(\tau)] < r\} \\ = & \left\{x_1 \mid [x_1 - \bar{x}(\tau)]^T \frac{G(t)^{-1}}{r-\tau}[x_1 - \bar{x}(\tau)] < 1\right\}. \\ = & \mathcal{E}[\bar{x}(\tau), G(t)(r-\tau)]. \end{aligned}$$

4. How to find near nodes efficiently

$$
\begin{aligned}
&\{x_1 \mid \tau + [x_1 - \bar{x}(\tau)]^T G(t)^{-1} [x_1 - \bar{x}(\tau)] < r\} \\
&= \left\{x_1 \mid [x_1 - \bar{x}(\tau)]^T \frac{G(t)^{-1}}{r-\tau} [x_1 - \bar{x}(\tau)] < 1\right\}. \\
&= \mathcal{E}[\bar{x}(\tau), G(t)(r-\tau)].
\end{aligned}
$$



其中$\mathcal{E}[x, M]$是一个具有中心x和正定权重矩阵M的椭球体，可以定义为:

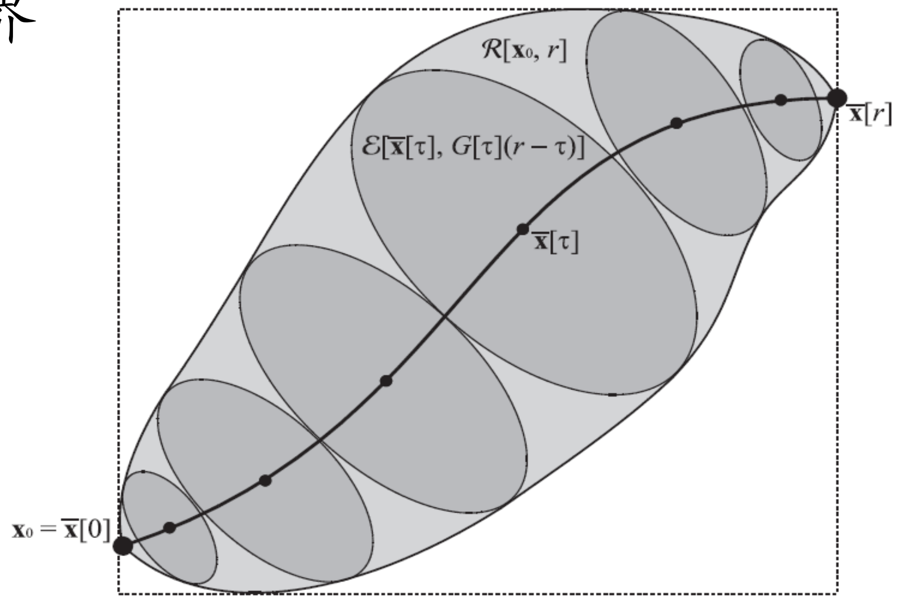
$$\mathcal{E}[x, M] = \{x' \mid (x' - x)^T M^{-1} (x' - x) < 1\}.$$

因此，前向可达集是所有可能到达时间τ的高维椭球体的并集

4. How to find near nodes efficiently

问题简化：，
对几个 τ 进行采样，为每个 τ 计算椭球的轴对齐边界框，更新每个维度中的最大值和最小值：

$$\prod_{k=1}^{n}\begin{bmatrix} min\{0<\tau<r\}\left(\bar{x}(\tau)_k-\sqrt{G[\tau]_{(k,k)}(r-\tau)}\right), \\ max\{0<\tau<r\}\left(\bar{x}(\tau)_k+\sqrt{G[\tau]_{(k,k)}(r-\tau)}\right) \end{bmatrix}.$$



向后可达集的计算类似。

4. How to find near nodes efficiently

Kinodynamic RRT*

Input: E, x_init, x_goal

Output: A trajectory T from x_init to x_goal

```
T.init();
for i = 1 to n do
    x_rand ← Sample(E);
    X_near ← Near(T, x_rand);
    x_min ← ChooseParent(X_near, x_rand);
    T.addNode(x);
    T.rewire();
```

当进行邻居节点查询和选择父节点时，

可以从x_rand的后向可达集中找到 X_near。

5. How to “Rewire”

Kinodynamic RRT*

Input: E, x_init, x_goal

Output: A trajectory T from x_init to x_goal

```
T.init();
for i = 1 to n do
    x_rand ← Sample(E);
    X_near ← Near(T, x_rand);
    x_min ← ChooseParent(X_near, x_rand);
    T.addNode(x);
    T.rewire();
```

当“Rewire”时，

可以计算*x_rand*的前向可达集，并求解OBVPs。