

状态栅格搜索算法 (State Lattice Search)

用于解决搜索树构建问题中路径的不平滑问题，算法中将两点间这样平滑的路径成为feasible motion connections。其中分为两种方式：Sample in control space 和 Sample in state space。

TODO:这俩理解的还不是很透彻，等再读读论文，看看源码

两个算法最核心的还是他们的想法吧，一个根据运动学模型前向搜索，一个根据采样的状态倒推路径。感觉没必要纠结具体怎么实现了，文章说的都比较理论性

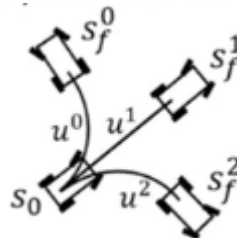


机器人运动微分方程

For a robot model:

$$\dot{s} = f(s, u)$$

Sample in control space



根据机器人运动模型，通过对当前状态施加不同的 u ，代入微分方程计算机器人在 T 时间后的位置，以此构建graph。比如 u 的范围为 $[-u_{\max}, u_{\max}]$ ，那么就将这个区间10等分或者20等分取到不同的 u

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



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

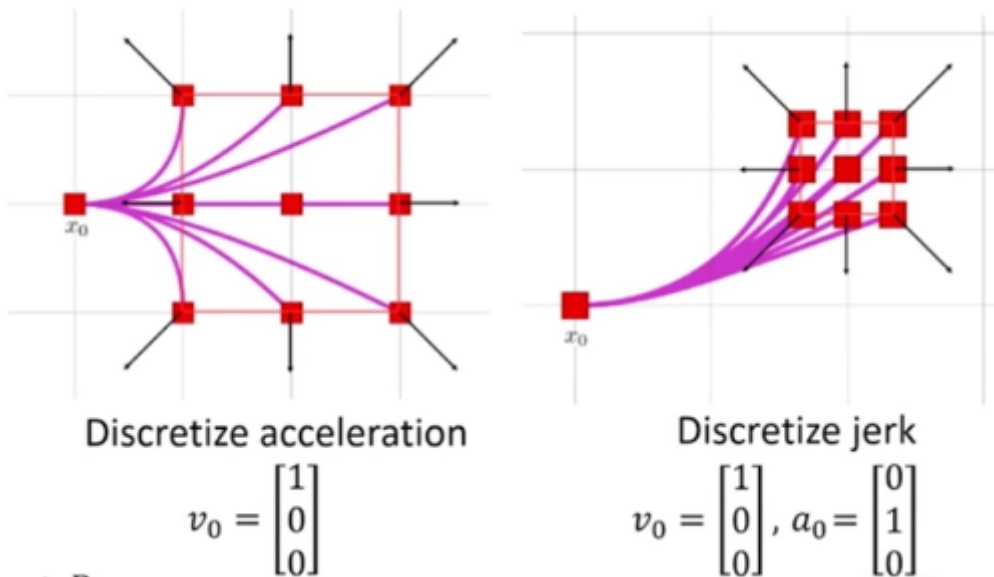
$$v_0$$

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow A = \begin{bmatrix} 0 & I_3 & 0 & \dots & 0 \\ 0 & 0 & I_3 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & 0 & I_3 \\ 0 & \dots & \dots & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ I_3 \end{bmatrix}$$

对于一个线性模型（图中是无人机的例子，所以有z轴），有状态方程：

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

其中A是一个幂等矩阵。在该例子中，输入控制量为加速度，但也可以是jerk（加加速度），将jerk作为输入量的好处是可以控制无人机或汽车状态变化的幅度，后续例子的输入控制量也均为jerk。



左图输入为加速度，右图为jerk，可以看到右图的曲线明显更加平滑。

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

e^{At} : state transition matrix, critical to the integration.

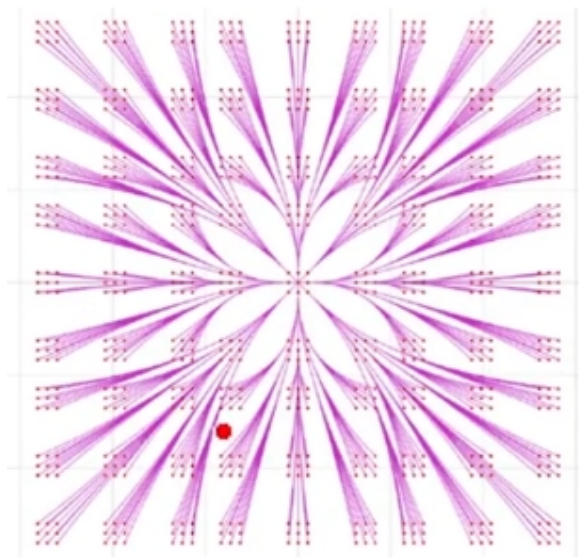
$$e^{At} = I + \frac{At}{1!} + \frac{(At)^2}{2!} + \frac{(At)^3}{3!} + \dots + \frac{(At)^k}{k!} + \dots$$

If matrix $A \in R^{n \times n}$ is **nilpotent**, i.e. $A^n = 0$, e^{At} has a closed-form expression in the form of an $(n - 1)$ degree matrix polynomial in t .

对于这样一个线性系统，它的状态方程为图中所示（将微分方程积分后的结果），其中指数项可以泰勒展开，并且由于A的幂等性质，后面的项都可以去掉。

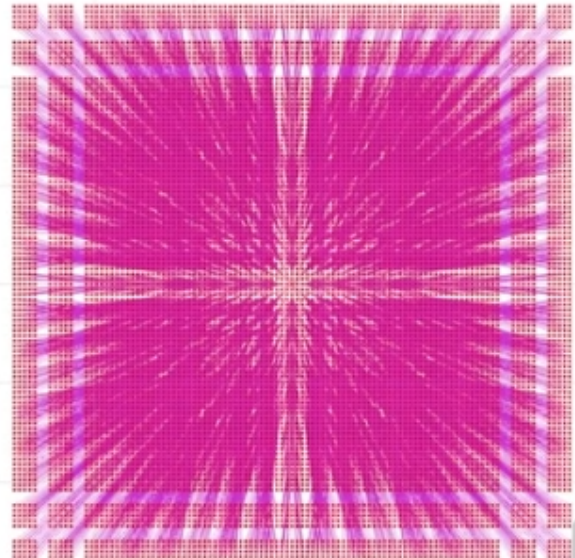
这样就可以计算出在T时间后，施加不同输入量的车辆的状态了，这些状态加入search graph中。

The lattice graph obtained by searching



9 discretization

当前用户:38776



25 discretization

对于lattice graph来说，他不像栅格地图一样是一个从一开始就开辟一部分内存用于存放，lattice graph的“栅格”是在搜索过程中不断载入内存的，因此在实时性的场景中，可以根据一些启发式函数去指引Forward的栅格方向，将带有目的性的栅格载入内存而非将整张图都载入。

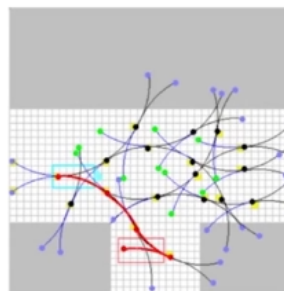


$$\text{State: } s = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \quad \text{Input: } u = \begin{pmatrix} v \\ \phi \end{pmatrix}$$

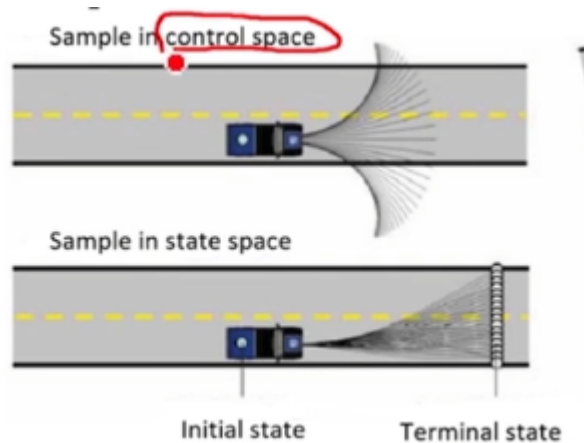
$$\text{System equation: } \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} v \cdot \cos\theta \\ v \cdot \sin\theta \\ \frac{r}{L} \cdot \tan\phi \end{pmatrix}$$

对于小车模型也是同理，只是状态模型不同。

- For every $s \in T$ from the search tree
- Pick a control vector u
- Integrate the equation over short duration
- Add collision-free motions to the search tree



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



该方法的缺点在于：

- 在搜索lattice graph的时候，他是没有目的性的。比如在车道上就很容易搜索到车道外面。
- 可能对一个节点进行扩展时，所有connections都撞到障碍物了，过于稠密

Sample in state space

先给出机器人的目标状态，然后当前状态到目标状态的路径。它的好处在于有明确的目的性，但计算苦难。

该算法旨在解决一个问题：在已知初始状态和终止状态时，如何规划最优路径？该问题被称为 Optimal Boundary Value Problem(OBVP)，可以利用Pontryain极小值原理求解。

Pontryagin' s minimum principle

Generally:

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

final state transition cost

Write the Hamiltonian and costate:

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

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

Suppose:

s^* : Optimal state
 u^* : Optimal input

We have



minimum principle

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

$\lambda(t)$ is the solution of:

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

with the boundary condition of:

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

and the optimal control input is:

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



Modelling

Objective, minimize the integral of squared jerk:

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

State: $s_k = (p_k, v_k, a_k)$ Input: j_k

System equation: $\dot{s} = f_s(s, u) = (v, a, j)$

Solving

By Pontryagin's minimum principle, we first introduce the costate: $\lambda = (\lambda_1, \lambda_2, \lambda_3)$

Define the Hamiltonian function:

$$\begin{aligned} H(s, u, \lambda) &= \frac{1}{T} j^2 + \lambda^T f_s(s, u) \\ &= \frac{1}{T} j^2 + \lambda_1 v + \lambda_2 a + \lambda_3 j \\ \dot{\lambda} &= -\nabla_s H(s, u, \lambda) = (0, -\lambda_1, -\lambda_2) \end{aligned}$$

Optimal state Optimal input

The costate is solved as:

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

The optimal input is solved as:

$$\begin{aligned} u^*(t) &= j^*(t) = \arg \min_{j(t)} H(s^*(t), j(t), \lambda(t)) \\ &= \frac{1}{2} \alpha t^2 + \beta t + \gamma \end{aligned}$$

The optimal state trajectory is solved as:

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

Initial state: $s(0) = (p_0, v_0, a_0)$

The cost:

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

α, β, γ is solved as:

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

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

J only depends on T , and the boundary states (known), so we can even get an optimal T !

How?

Polynomial function root finding problem.

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

This derivation holds for fixed final state: $s(T) = (p_f, v_f, a_f)$

Similar solution can also be found when $s(T)$ is partially defined

算法流程如上，值得注意的有以下几点：

- 算到最后可以发现 α, β, γ 都是关于 T 的函数，因此当 T 未知时，将 j^* 代入到 J 的表达式中，可以得到一个关于 T 的函数，然后对 T 求导等于0，可以解出最优的 T ，进而解出输入量 j^* 。
- 如果模型的终态是一个定死的常数，那么在解出 j^* 后，通过积分求解 s^* ，进而算出 $s^*(T)$ ，即最优最终状态，然后就可以求解出 α, β, γ 。此时可以看成对终态的惩罚项是无穷大或0的两点函数，不可导，但可以直接通过这种方式求出。
- 如果模型的终态有某几个量不是定死的，而是自由量，则 J 的形式中还要加上一个对终态的惩罚项 h ，这个 h 其实就是解出的 $s^*(T)$ ，此时应用定理， $\lambda(T) = -h$ 对 T 的导数得到方程，进而求解出 α, β, γ
- 对jerk积分是为了让整段轨迹的能量消耗最小

在Lattice Graph上搜索

还是用A*搜索，但是 h 有两种情况：

- 不考虑障碍物
- 不考虑动力学(不能直穿障碍物，但可以用直线连接两点)

