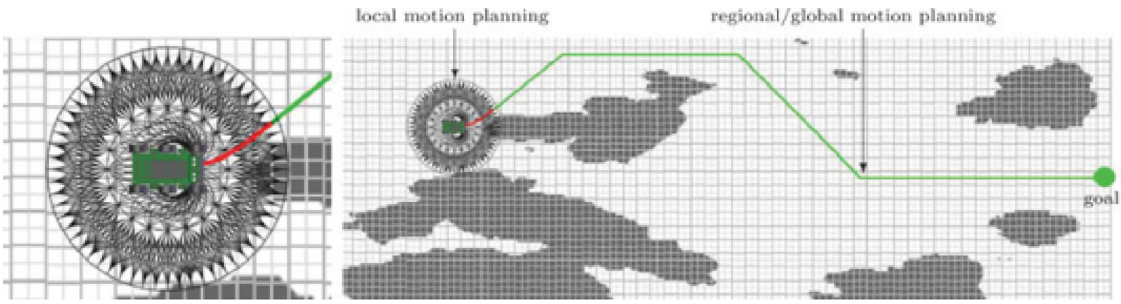
状态栅格搜索算法（State Lattice Search）

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

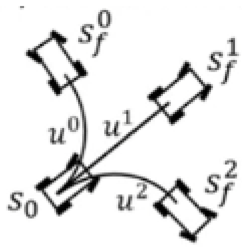


机器人运动微分方程

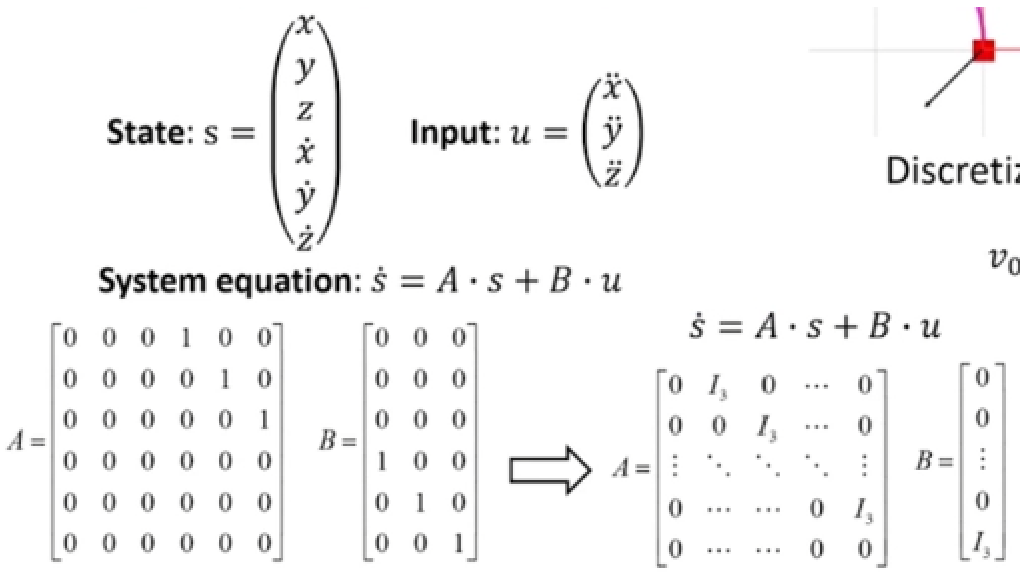
For a robot model：

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

Sample in control space



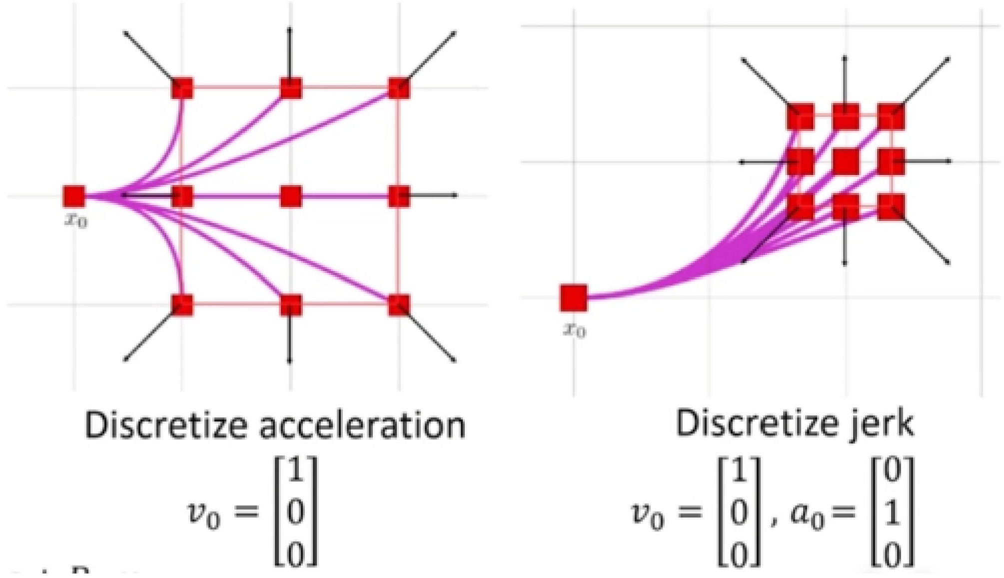
根据机器人运动模型，通过对当前状态施加不同的u，代入微分方程计算机器人在T时间后的位置，以此构建graph。比如u的范围为[-u_{max},u_{max}]，那么就将这个区间10等分或者20等分取到不同的u



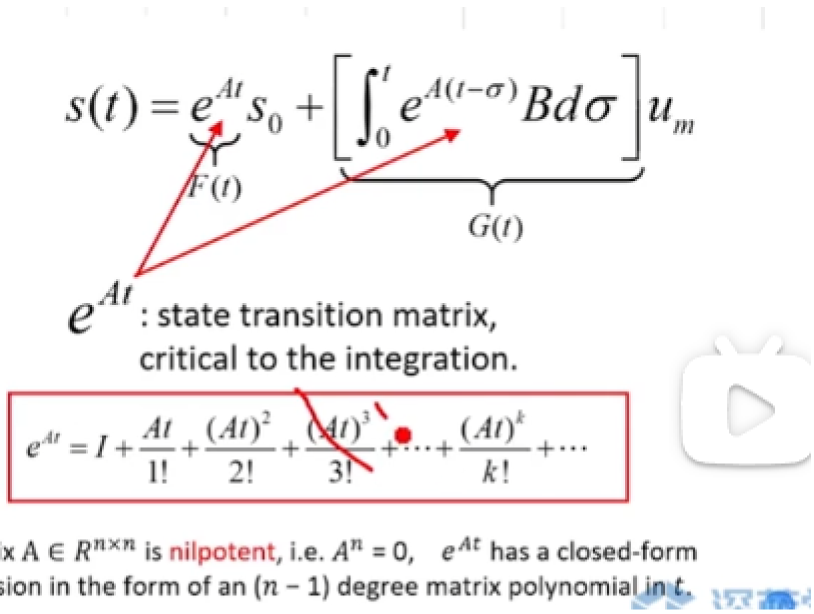
对于一个线性模型（图中是无人机的例子，所以有z轴），有状态方程：

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

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。

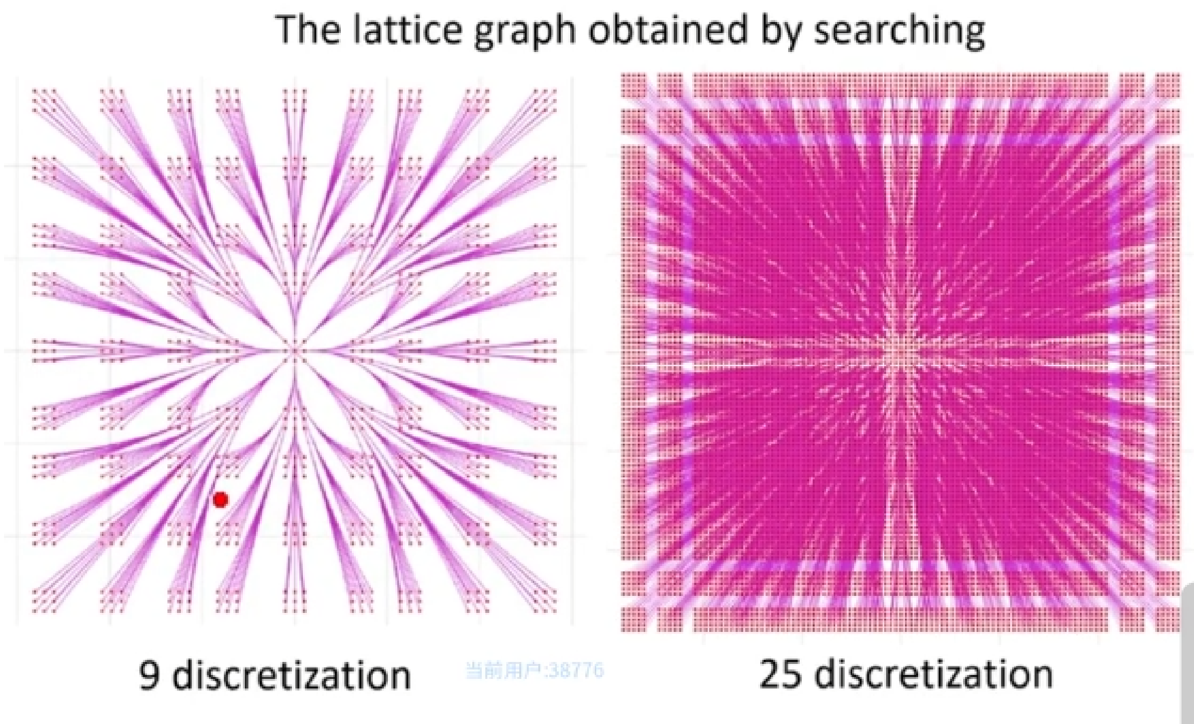


左图输入为加速度，右图为jerk，可以看到右图的曲线明显更加平滑。

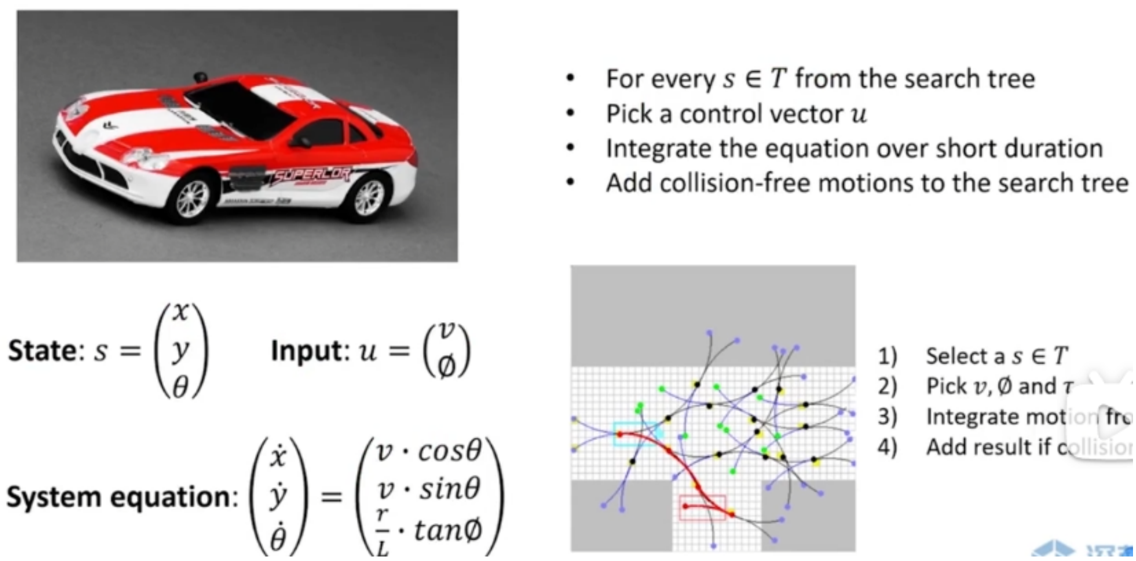


对于这样一个线性系统，它的状态方程为图中所示（将微分方程积分后的结果），其中指数项可以泰勒展开，并且由于A的幂等性质，后面的项都可以去掉。

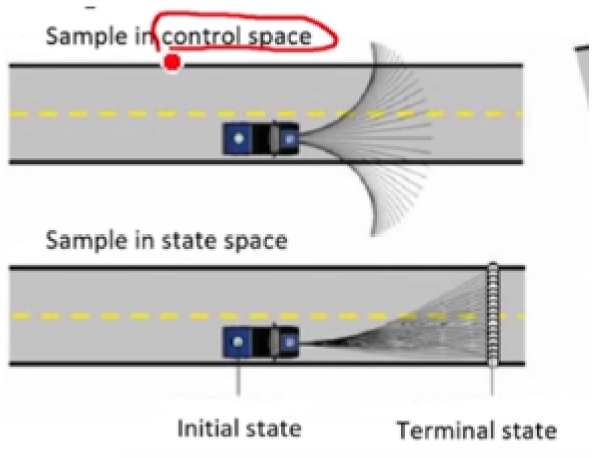
这样就可以计算出在T时间后，施加不同输入量的车辆的状态了，这些状态加入search graph中。



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。



对于小车模型也是同理，只是状态模型不同。



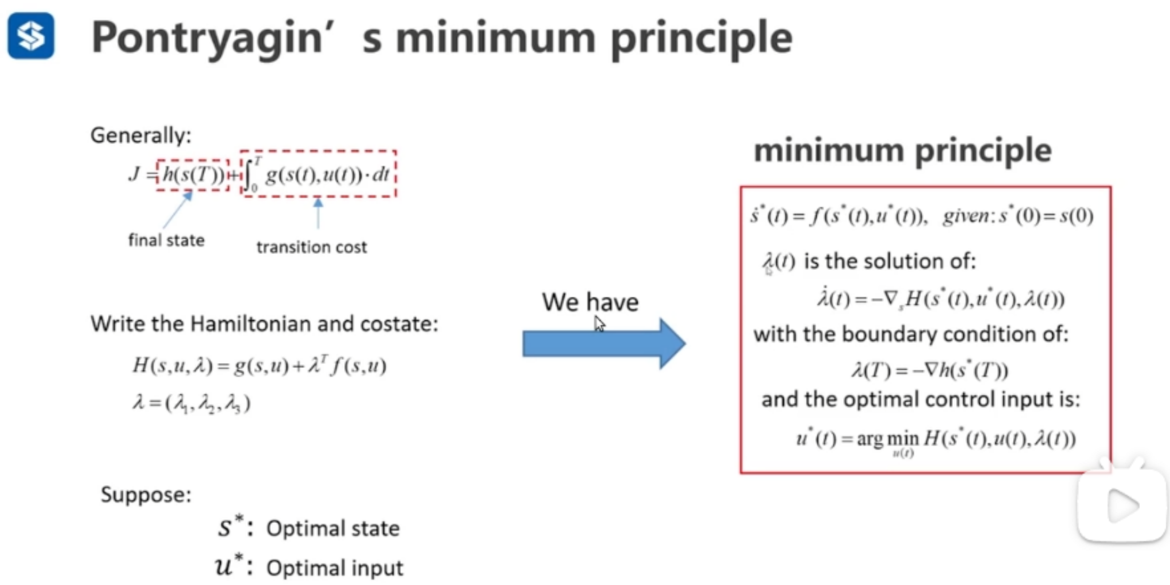
该方法的缺点在于:

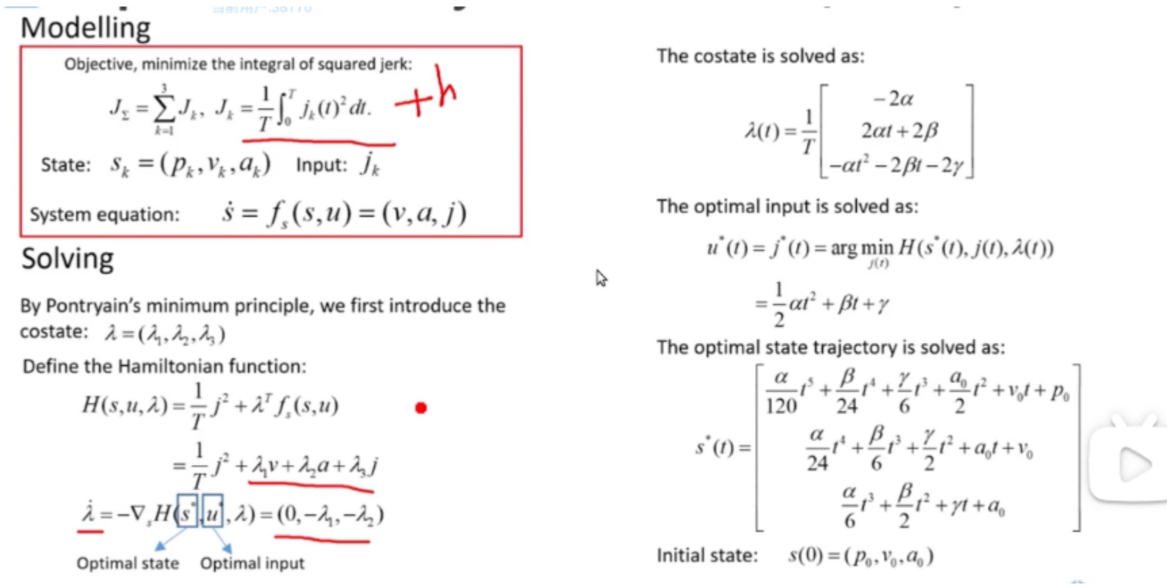
- 在搜索lattice graph的时候，他是没有目的性的。比如在车道上就很容易搜索到车道外面。
- 可能对一个节点进行扩展时，所有connections都撞到障碍物了，过于稠密

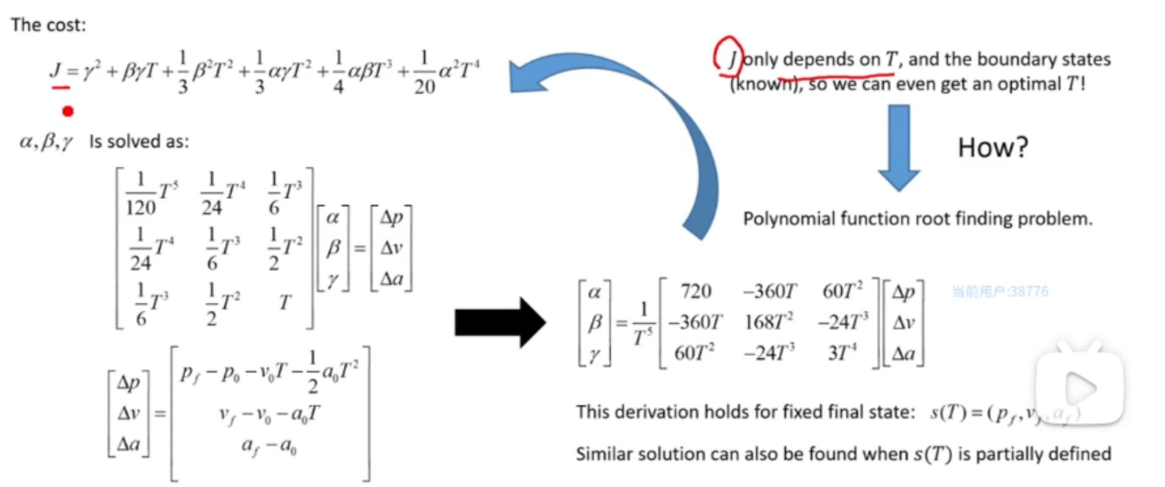
Sample in state space

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。







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，λ(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)