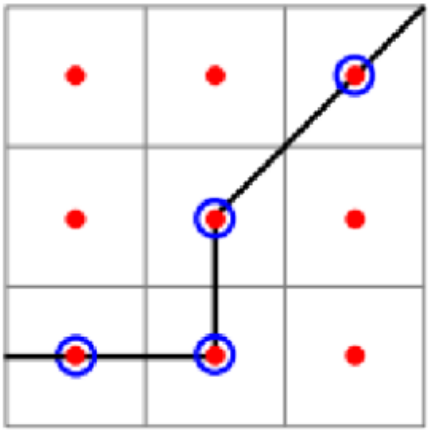
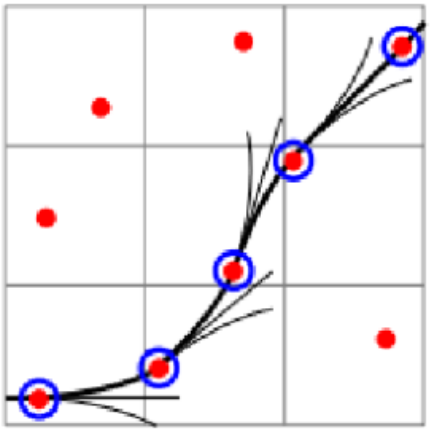
Hybrid A Star

https://zhuanlan.zhihu.com/p/593406203

将Sample in control space的lattice search应用于栅格上，并且**规定每个栅格内只能有一个扩展的节点**，如果在一个栅格内已经有了一个节点，此时又有一个节点扩展过来了，需要比较二者的代价函数，选择代价少的那个。



Grid Based



KinoDynamic

扩展节点的过程

两个节点之间并不是通过一次动力学积分得到的，而是经过若干次积分，当达到指定长度的时候才将最后一个节点作为扩展的节点。其中积分的次数和积分停止由motion_resolution和arc_l决定，其中前者表示在地图的分辨率尺度上一次走的路径长度，arc_l表示最多走的路径长度，它一般比地图分辨率大一点，防止两次扩展都在一个格中。

```
# 各个list存储的是两个节点之间的模拟路径点
for _ in np.arange(0, arc_l, MOTION_RESOLUTION):
        x, y, yaw = move(x, y, yaw, MOTION_RESOLUTION * direction, steer)
        x_list.append(x)
        y_list.append(y)
        yaw_list.append(yaw)

def move(x, y, yaw, distance, steer, L):
    x += distance * cos(yaw)
    y += distance * sin(yaw)
    yaw += pi_2_pi(distance * tan(steer) / L)  # distance/2

    return x, y, yaw
```

节点的cost

个人理解cost由以下几个部分组成：traveled_cost + motion_cost + map_cost + heuristic_cost

traveled_cost是父节点已经走过的路程

motion_cost是父节点到当前节点的cost，由方向盘方向变化惩罚、车辆前进后退切换惩罚等组成

map_cost是和障碍物的距离

heuristic_cost是到终点的启发性函数，分为考虑动力学但不考虑障碍物和考虑障碍物但不考虑动力学。后者可以在做路径规划前在整个地图上以终点作为起点，用Dijkstra算法计算终点到地图上每个点的最短距离，**前者暂时还没搞明白具体咋做**

```
SB_COST = 100.0          # switch back penalty cost
BACK_COST = 50.0         # backward penalty cost
STEER_CHANGE_COST = 2.0  # steer angle change penalty cost
NON_STRAIGHT_COST = 0.0  # steer angle not zero cost
H_COST = 3.0             # Heuristic cost
M_COST = 3.0             # Cost map coeffient


def calc_next_node(current, steer, direction, config, ox, oy, kd_tree):
    x, y, yaw = current.x_list[-1], current.y_list[-1], current.yaw_list[-1]

    arc_l = XY_GRID_RESOLUTION * 1.5
    x_list, y_list, yaw_list = [], [], []

    # Simulate about one grid in length.
    # put all the intermediate points into the list.
    for _ in np.arange(0, arc_l, MOTION_RESOLUTION):
        x, y, yaw = move(x, y, yaw, MOTION_RESOLUTION * direction, steer)
        x_list.append(x)
        y_list.append(y)
        yaw_list.append(yaw)

    # Make sure there is no collision along the way.
    if not check_car_collision(x_list, y_list, yaw_list, ox, oy, kd_tree):
        return None

    d = direction == 1
    # Use the last point as the next node.
    x_ind = round(x / XY_GRID_RESOLUTION)
    y_ind = round(y / XY_GRID_RESOLUTION)
    yaw_ind = round(yaw / YAW_GRID_RESOLUTION)

    # Calculate the cost base on the actions.
    added_cost = 0.0

    if d != current.direction:
        added_cost += SB_COST

    # steer penalty
    added_cost += NON_STRAIGHT_COST * abs(steer)

    # steer change penalty
    added_cost += STEER_CHANGE_COST * abs(current.steer - steer)

    # cost = huristic cost + motion cost + traveled cost
    cost = current.cost + added_cost + arc_l

    node = Node(x_ind, y_ind, yaw_ind, d, x_list,
```

```
                y_list, yaw_list, [d],
                parent_index=calc_index(current, config),
                cost=cost, steer=steer)

    return node
```

one shot机制

每隔一段时间，会选取一个最新的节点进行one shot计算，即直接利用Reeds Shepp曲线计算该节点到终点的路径，然后检查该路径是否遇到障碍物，如果没有，则直接搜索结束。

对于间隔时间，作者给了一个初始值N，即每20个节点做一次，然后N随着heuristic代价的减小而减小（离终点越近越有可能one_shot成功）