轮式机器人5-8周实验报告

3200102888 米博宇

第五周 路径规划算法

Rapidly-exploring Random Trees(RRT)算法

算法描述

给定地图、起点、终点和规划步长$step$，维护一个决策列表L，列表中每个元素均为地图中的一个点。开始时将起点加入决策列表，之后开始循环，每次在地图中随机选取一个点d_0，寻找列表中与其距离最近的点d_1，在d_0, d_1两点连线上选择点d_n，使得d_n与d_1距离为s。若d_n处于障碍物区域内，则重新选择d_0，继续循环，否则将d_n加入L，并将其前驱设为d_1。

如此循环，直至d_n与终点距离小于所设阈值。此时路径规划完毕，对空列表$path$执行以下操作：将d_n加入$path$，同时每当一个点加入$path$，将其前驱也加入$path$，直至起点加入$path$。

$path$即为所规划的路径。

伪代码

```
def RRTPlanner(obs_list, start_node, end_node, step):
    path = []
    node_list = [start_point]
    while 1:
        new_node = random_node()#随机生成一个点
        nearest_node = find_nearest_node()#找到最近点
        nearest_node_copy = nearest_node
        nearest_node.x += step * cos(theta) #theta为new_node，nearest_node连线与水平方向的夹角
        nearest_node.y += step * sin(theta)
        if nearest_node in any of obstacle:#在障碍物内
            continue
        elif distance(nearest_node, end_node) < threshold:#到达终点
            node = nearest_node
            while node is not None:
                path.append(node)
                node = node.parent
            return path
        else:
            node_list.append(nearest_node)
            nearest_node.parent = nearest_node_copy
```

Astar算法

算法描述

首先对地图进行栅格化：将地图按照给定分辨率划分为若干小栅格，并根据障碍物位置和大小将某些方格栅格为“不可通过”。

地图上每个可以通行的栅格均具有$cost$属性，$cost$由两部分相加得到：

- 与前驱的距离

- 与目标点的距离

维护两个集合：$closed\_set$ and $open\_set$。将开始点加入$close_set$，然后开始循环：

- 从$open_set$找出$cost$最小的点c_node
- 如果c_node所在栅格与终点所在栅格相同，则跳出循环。
- 将c_node加入$closed_set$，遍历其上、右上、右、右下、下、左下、左、左上八个栅格。
  - 如果当前栅格为"不可通过"或已在$close_set$内，不进行操作。
  - 计算当前栅格的$cost$值，若所得$cost$比原先$cost$小，则将其更新为较小$cost$； 若当前栅格不在$open_set$中， 则使其加入$open_set$。

最后，$closed_set$中的栅格即为路径上的所有栅格，根据所记录的前驱关系即可计算路径。

伪代码

```
def AStarPlanner(map, start_node, end_node):
    open_set, closed_set = dict(), dict()
    open_set[grid_index(start_node)] = start_node

    while 1:
        if len(open_set) == 0:
            break
        min_idx = the index of nodes of mininum cost in open_set
        current = open_set[c_id]
        if current is the end_node:
            goal_node.parent_index = current.parent_index
            goal_node.cost = current.cost
            break
        remove open_set[c_id]
        closed_set[c_id] = current
        for nodes around current:
            n_id = grid_index(node)
            node_parent_index = grid_index(current)
            node.cost = cal_cost(node) # 计算cost
            if n_id is obstacle:
                continue
            if n_id in closed_set:
                continue
            if n_id not in open_set:
                open_set[n_id] = node  # 发现新点
            else:
                if open_set[n_id].cost > node.cost:
                    open_set[n_id] = node
    return calc_final_path(goal_node, closed_set)
```

第六周 自动避障算法

Dynamic Window Approach(DWA)算法

算法描述

机器人运动过程中也需要对不在地图信息中的障碍物做出反应。为了实现实时避障效果，需要使用DWA算法。

DWA算法根据每一时刻机器人位置、速度和障碍物、目标的位置进行实时规划，选择出最佳的线速度和角速度。

规划过程为：根据最大速度、最大加速度约束获得该时刻下机器人线速度和角速度范围，遍历所有线速度和角速度组合，计算出在给定$predict_time$内机器人以该速度组合运动的轨迹并给出评价分数。在所有预测轨迹中选择分数最高的轨迹，其对应的线速度和角速度即为规划速度。

对速度的评价函数由三部分组成：

- 轨迹末端与目标点的距离
- 轨迹末端是否在障碍物范围中/轨迹末端与最近障碍物的距离
- 规划线速度与最大线速度差值

将三个评价值分别与对应系数的乘积相加，即为对路径的评分。

实际规划过程中，需要先使用路径规划算法生成一条路径，在路径上时刻选取与机器人距离相同的点$midpos$作为局部规划的目标。

伪代码

```
def DWAPlanner(v, vw, x, y, theta, obs, midpos):
    min_cost = float("inf")
    best_v, best_vw = 0.0, 0.0
    for v in np.arange(v_min, v_max, v_reso):
        for w in np.arange(vw_min, vw_max, yawrate_reso):
            tmp_path = cal_path(v, w)#计算路径
            goal_cost = calc_goal_cost(tmp_path, midpos)
            speed_cost = speed_cost_gain * (max_speed - v)
            obs_cost = calc_obs_cost(tmp_path, obs)
            cost = goal_cost + speed_cost + obs_cost
            if min_cost > cost:
                min_cost = cost
                best_v, best_vw = v, w
    return best_v, best_vw
def calc_goal_cost(path, goal):
    return distance(the end of path, goal)
def calc_obs_cost(path, obs):
    if the end of path is in any obstacle:
        return 1 / inf
    else:
        return 1 / the minimum distance between a point of path and a
obstacle
```

第七、八周 机器人路径规划、自主避障仿真

实验内容

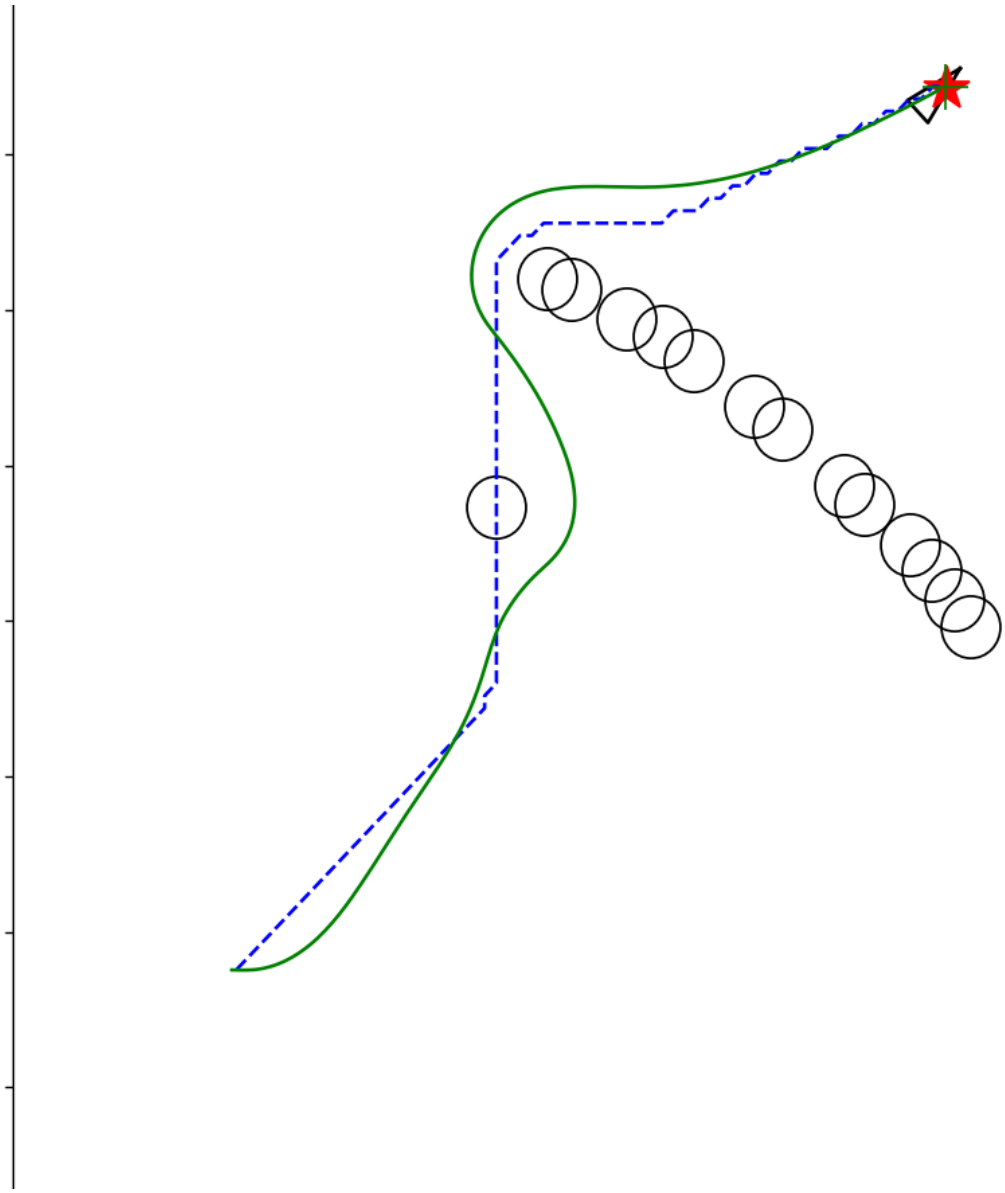
将工作区中的localplanner与globalplanner分别替换为DWA算法和任意路径规划算法，并调整其接口以与其他控制函数相适应即可。

在仿真过程中需要根据环境调节相关参数，尤其是半径膨胀的相关参数：若膨胀太多，则小车无法规划出通过较狭窄通道的路线，导致规划距离太长；若膨胀不足，则小车转向时容易与障碍物相撞，失去控制。

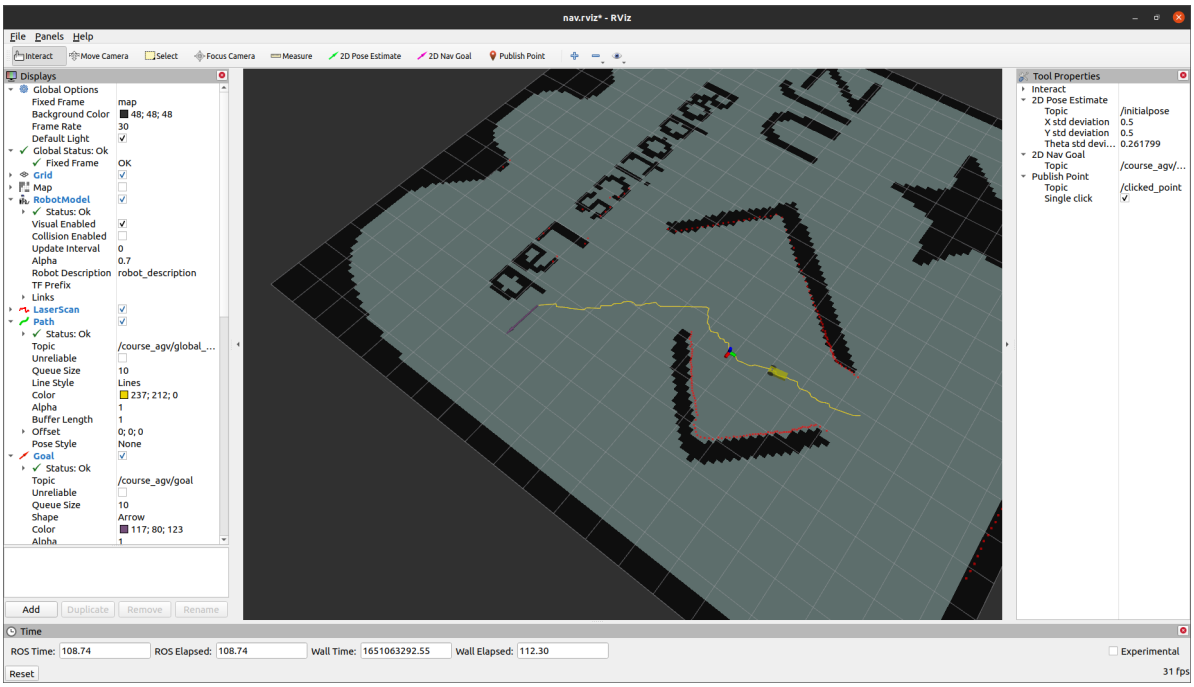
实验感想

通过四周的学习，我了解并实践了三种路径、速度规划的算法，并将前八周所学知识综合运用，在仿真界面对机器人进行了初步调试，为实物验证奠定了基础。在实验中，最困难的地方在于参数的调节和算法性能的优化。参数调节方面，可以改变一个参数，固定其他参数观察该参数对结果的影响。算法性能优化方面，可以使用numpy, scipy的相关函数提高运算效率。

实验结果截图



DWA算法 + AStar算法测试结果



仿真结果