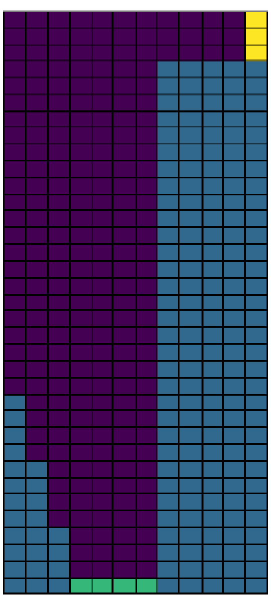
深蓝学院
shenlanxueyuan.com

第7章作业解析

刘 博

深蓝学院
shenlanxueyuan.com

Homework



Grid Map

1. $X = \{(x, y) \mid 0 \le x \le 11, 0 \le y \le 34\}$
2. $X_I = \{\text{green grids}\}$
   $X_F = \{\text{yellow grids}\}$
3. $U = \{(\ddot{x}, \ddot{y}) | \ddot{x} \in \{0, 1\}, \ddot{y} \in \{0, 1\}\}$
4. $\Theta = \{\theta_1, \theta_2\}$
   - θ_1: $f(\mathbf{x}_{k+1}, \mathbf{x}_k, \mathbf{u}_k) = \mathbf{x}_k \quad p_1 = 0.1$
   - θ_2: $f(\mathbf{x}_{k+1}, \mathbf{x}_k, \mathbf{u}_k) = \mathbf{x}_{k+1} \quad p_1 = 0.9$
5. $l(\mathbf{x}_k, \mathbf{x}_k, \theta_k) = -1$
6. Find an optimal plan from X_I to X_F

DP & RTDP

Initialize G values of all states to finite values;
while *not converge* **do**
  for *all the states* x **do**
    $G(x_F) = 0;$
    $G_k\left(x_k\right) =$
     $\min_{u_k \in U(x_k)} \left\{E_{\theta_k}\left[l\left(x_k, u_k, \theta_k\right) + G_{k+1}\left(x_{k+1}\right)\right]\right\},$
     if $x_k \neq x_F$;
  end
end

1. Initialize G values of all states to admissible values;
2. Follow greedy policy picking outcomes at random until goal is reached;
3. Backup all states visited on the way;
4. Reset to x_s and repeat 2-4 until all states on the current greedy policy have Bellman errors $< \Delta$, where $\Delta(x_k) = \|G(x_k) - G(x_{k+1})\|$;

DP RTDP

Initialize G values

```
def Init_G_value(node):
    min_dis = 100000000.0
    for pnt in FINISH_LINE:
        (target_px, target_py) = (pnt[0], pnt[1])
        dis = ((target_px - node.px)**2 + (target_py - node.py)**2)**0.5
        if dis < min_dis:
            min_dis = dis
    return min_dis
```

```
for key in graph.keys():
    state = graph[key]
    if state.is_goal:
        state.g_value = 0
    else:
        if state.px == 32 or state.px == 33 or state.px == 34:
            state.g_value = 11 - state.py
        else:
            state.g_value = (32 - state.px) + (11 - state.py)
```

greedy_policy

```
rand_start = np.random.randint(low=0, high=3, size=1)[0]
greedy_plan = greedy_policy(idx=rand_start)
```

```python
def greedy_policy(idx=0, explore=True):
    start_node = Node(START_LINE[idx][0], START_LINE[idx][1], 0, 0)
    start_key = start_node.key
    state = graph[start_key]
    trajectory = [state.key]
    while not state.is_goal:
        value_uk = []
        for child_idx in range(len(ACTION_SPACE)):
            child_key_9 = state.next_prob_9[child_idx]
            child_9 = graph[child_key_9]
            value_uk.append(child_9.g_value)

        action_idx = np.argmin(value_uk)
        if explore:
            action_idx = explore_action(action_idx)
        child_key = state.next_prob_9[action_idx]
        trajectory.append(child_key)
        state = graph[child_key]
        # if [state.px, state.py] in START_LINE:
        #     trajectory = [state.key]
        print('finding feasible path: {}, {}'.format(state.px, state.py))
    # print('found trajectory: {}'.format(trajectory))
    return trajectory
```

```python
def explore_action(u_idx, epsilon=0.2):
    if np.random.uniform(0, 1) < epsilon:
        return np.random.randint(0, len(ACTION_SPACE))
    else:
        return u_idx
```

运动规划第七章作业

1、用 Python 实现 Real time dynamic programming

下面简要介绍程序所实现的算法：

（1）初始化 State 的 g_value，在这里我们用 State 到所有终点的最短距离作为初始值。

（2）然后选择某个 Start State 作为搜寻路径的开始点。

（3）随机进行 Exploitation 和 Greedy Policy Exploration 操作得到新的状态（这里我们设置进行 Exploitation 的概率是 0.2，Exploration 的概率是 0.8）。

（4）当搜寻到完整的一条路径的时候，我们进行 BackUp 相关操作，在更新路径上 State 的 g_value 的同时，计算整条路径的 Bellman Error，直到 Bellman Error 收敛，从而得到 RTDP 结果。

算法流程图如下所示：

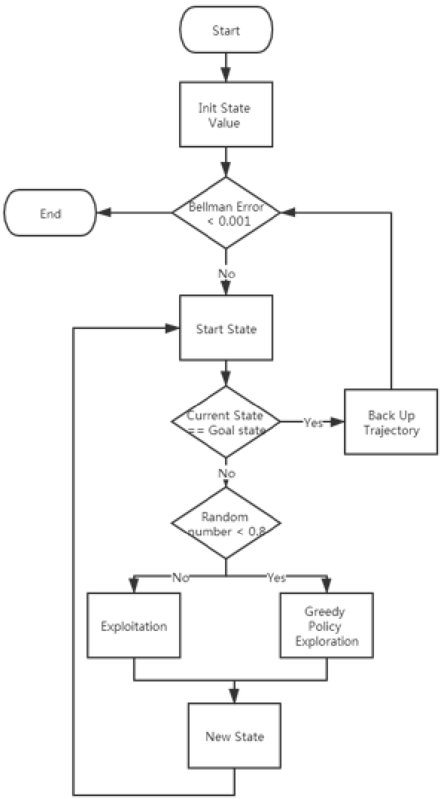


图 1 RTDP 算法流程图

以下对比 RTDP 与 DP 程序结果：

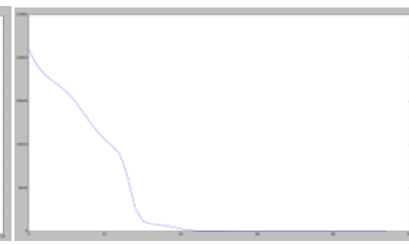
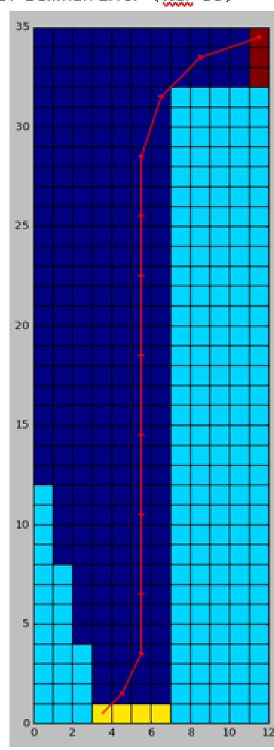
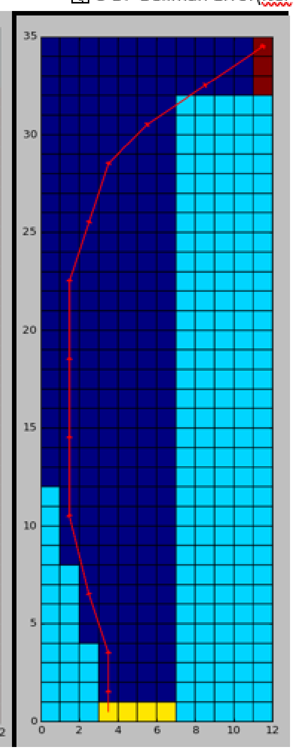


图 2 RTDP Bellman Error（iter=30）　　图 3 DP Bellman Error(iter = 44)





Motion Planning for Mobile Robots

第七次作业

本次作业完成了 RTDP(real time dynamic programming)，下面比较了 RTDP 和 MDP 两种方法产生的结果差异。

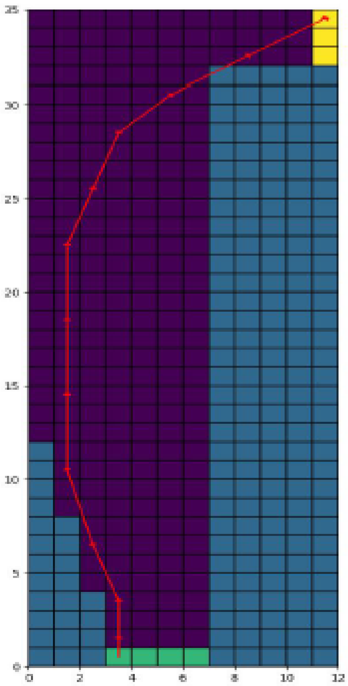


FIG 1: MDP

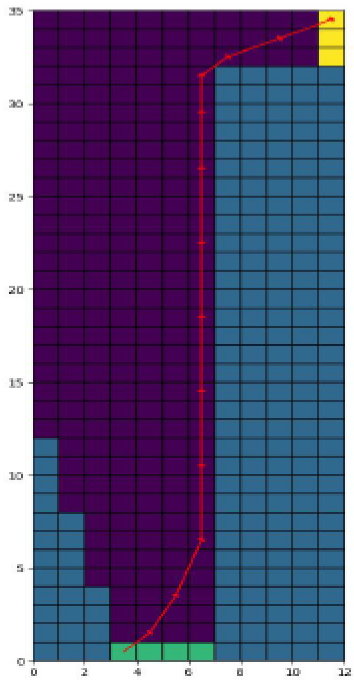


FIG 2: RTDP

	ITR_NUM	TIME	POINT_NUM
MDP	52	143	1020580
RTDP	172	37	2320

FIG 3: MDP 和 RTDP 比较

从实验结果看，虽然 RTDP 的迭代总数大于 MDP，但是在总时间上 RTDP 是小于 MDP 的，从节点数量上看，MDP 是远大于 RTDP，所以从而增大了 MDP 的时间。原因是 MDP 需要计算所有 state 的值，而 RTDP 只是计算与其 state 相关的值。