

A Quick Employment of Markov Decision Process in Partially Unknown Three-dimensional Discrete Space

Liu Enxu, Zhu Hongyi

liuenxu2020@126.com, hongyi.zhu.20@ucl.ac.uk

Summary—The goal of this project is to develop a planning algorithm under partially unknown 3D discrete space. Value iteration is employed since Markov Decision Process complies with the state transition and decision patterns here, with an adaption for changing transitional probabilities and rewards/costs. Eventually, the robot conducts an optimal policy in a fully known map and reaches the destination in partially unknown environment with less efficiency.

I. INTRODUCTION

3D navigation of robots in complex environments is one of the topical problems as its interdisciplinary with artificial intelligence thrives. However, not all researchers have access to robotics with the best setup, e.g., highly precise sensors and powerful maneuvering ability, to face complicated environment with vague or little information, such as a submarine robot operating underwater. Thus, it is necessary to relax the problem into a discrete space to help develop a solution which quickly deploys a simple robot exploring in the partially unknown operation field, trading-off a feasible policy at a cost of losing a bit resolution. The environment is set up so that an incomplete map is given to the robot with positions of unknown obstacles hidden, and therefore the objective is to urge the underwater robot to avoid both known and unknown obstacles and find a safe path to its destination. Preliminary materials and theories are shown in section II., and section III. shows how a dynamic value iteration is employed to solve the problem. Section IV. introduces the result of implementation and section V will discuss the limitations and future work. Related figures will be appended in the VI part.

II. BACKGROUND AND RELATED WORK

Multiple studies of navigation problems have been taken since last century. J. Forbes, T. Huang, K. Kanazawa, and S. Russell combined an MDP problem with model-free reinforcement learning techniques, use stochastic theory to model complex environments [1]. ‘Probabilistic MDP-Behavior Planning for Cars’ further describes how to cope with uncertain systems with expanding methods of MDP [2]. Other articles recommend a useful package namely ‘mdptoolbox’, which supports a variety of algorithms including value iteration. In addition, the author shows how to implement a simple 2D navigation with the package [3]. A mature preliminary work has been done in 2D space planning, however it’s easy to tell planning in complex 3D environment is yet to be concentrated on. An MDP planning in a partially observable continuous 3D space is developed in [4] to avoid obstacles in operation fields by employing Monte Carlo Value Iteration(MCVI) to obtain a “threat resolution logic”, which reduces the collision probability by 70 times at most comparing with previous 2D discrete models. This essay relaxes the computation by discretizing the space considering the hardware limitations of researchers, and tries to put forward a simpler version of planning algorithms for scenarios in which some a priori is given, such as updating the change of terrains of a water area previously explored.

PRELIMINARY THEORY OF MARKOV DECISION PROCESS

The basic concept [5] of Markov Decision Process (MDP) is to model the problem with agents and environment, where the agent should perform specific actions (a) in the environment at certain state (s), and then we can give a reward (r) to the agent based on the action performed at that state, and therefore the result would give a sequence of actions which direct the agent to optimized states.

The goal of the process is to maximize the sum of rewards (or to minimize the sum of the cost, e.g. in the case the essay discusses), achieved by adding the maximum rewards available in the future with a discount factor to the current rewards, thus influencing the current action with considerations of future potential rewards at the same time, which is the weighted sum of the expected values of rewards for all future actions starting from the current state. The resultant policy $\pi(s)$ thus maximizes discounted cumulative reward according to certain criterion and .

Markov decision process (MDP) is used to describe typical state transition problems. A basic MDP problem can be represented by (S, A, T, R, γ), and the meanings of these symbols are:

S: all possible discrete states in the scenario

A: all possible actions

T: transition probability function

R: reward function

γ : discount factor

$$\text{Total reward} = \sum_{k=0}^{\infty} \gamma^k r(s_k, a_k).$$

Currently there is value-based reinforcement learning and policy-based reinforcement learning, they solve problems in different iterative ways. Policy iteration is directly related to policy. A given policy is evaluated by an iterative method to obtain the value function, which is used to iterate over various policies to determine the best one. While value iteration is related to policy indirectly, the resultant policy is equivalently optimal when the values converge.

In this project we choose the very basic value iteration algorithm, which bases on the Bellman equations, which involve equations of optimal Q-value and optimal value function as follows:

$$Q_{n+1} = \sum_{s'} t(s'|s, a) (R(s, a, s') + \gamma \max_a Q_n(s', a'))$$

$$V_{n+1}^* = \max_a \sum_{s'} T(s'|s, a) (R(s, a, s') + \gamma V_n^*(s'))$$

Or conversely in the case of cost model:

$$Q_{n+1} = \sum_{s'} t(s'|s, a) (C(s, a, s') + \gamma \min_a Q_n(s', a'))$$

$$V_{n+1}^* = \min_a \sum_{s'} T(s'|s, a) (C(s, a, s') + \gamma V_n^*(s'))$$

$$V^*(s)=0 \text{ (if } s \in G) \text{ where } V^* \text{ is optimal} \\ = \min Q^*(s, a) \text{ (if } s \text{ not } \in G)$$

It successively approximates V^* with a V_n function, such that the sequence of $V_n(s)$ converges to V^* in the limit as n tends to infinity.

Initialize array V arbitrarily for each state

$n \leftarrow 0$

Repeat

```

n←n+1
For each s∈S do
    V=V(s)
     $V_n^* = \min_a \sum_{s'} T(s'|s, a)(C(s, a, s') + \gamma V_{n-1}^*(s'))$ 
    n←max(n|V-V(s))
end
until n<epsilon
return greedy policy
 $\pi(s) = \operatorname{argmin}_a \sum_{s'} T(s'|s, a)(C(s, a, s') + \gamma V_n^*(s'))$ 

```

However, in this case the V_n function is slightly different:

$$V_n^* = \min_a \sum_{s'} T(s'|s, a)(C(s, a) + \gamma V_{n-1}^*(s'))$$

In the cost/reward matrix the next state is always unknown, as the agent could possibly bump into hidden obstacles after trying to perform an action, which means the actual transition result is not *a priori*.

III. METHODS

A. Implementation Tool

To help us implement the value iteration process, the `pymdptoolbox` package is imported. It requires the declaration of a transition probability ($A \times S \times S$) array P and a cost ($S \times A$) matrix R [6]. A ($A \times S \times S$) shape is also valid for cost matrix, however only the only ($S \times A$) is used for this model (Figure 1).

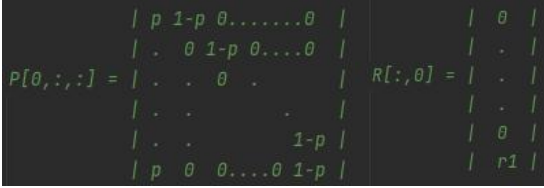


Figure 1. Transition Probability and Cost Array Examples

B. State Space and Action Space

The state space is defined to be discrete positions forming a square canvas. The state number denotation increases from left to right, up to down, then bottom to top. The position coordinates are expressed in list type variables with three elements in each: [layer, row, column]. Additionally, the presence of hidden obstacles would require the total position number to be multiplied with *hidden obstacle number+1* (known before decision to simplify the case) to obtain total state number, rather than multiplying a sum of combinations (i.e., combinations of finding 0,1,2...n unknown obstacles in n total unknown obstacles: $\sum_0^n \binom{n}{i}$) which causes redundancy as only number matters here. States in which the agent knows n hidden obstacles at position S_x have denotations as $S_{x+n \times \text{total position number}}$.

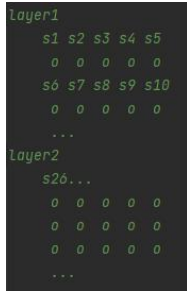


Figure 2. A Brief Graphic Demo of State Denotation with respect to Positions

A set of actions is defined to describe 6-direction movements in a discrete 3D space, including an additional scanning action. Each positional move adds or subtracts 1 in the relative dimension. Up, right and forward would plus 1, others subtract 1. All actions are collected in a dictionary type variable to relate the name of actions with figure change in the coordinates.

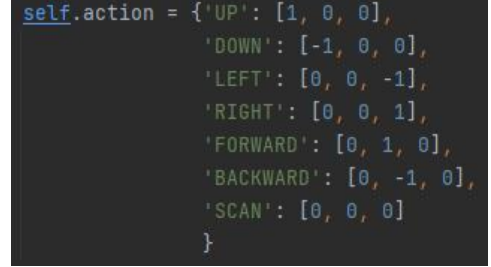


Figure 3. Action Space

C. Transition Matrix and Cost Matrix

The states are aligned in the increasing order of the state number denotation given in section B. The transition model is Stochastic which means the sum of probability of a row should be 1.

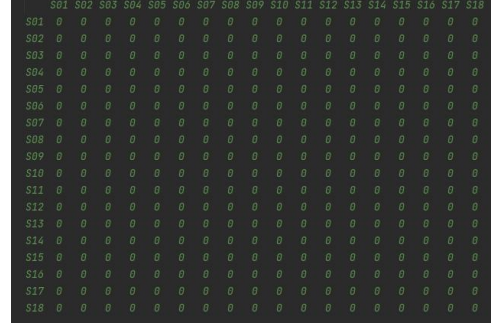


Figure 4. Empty Transition Table of 18 States, 9-Position Map with 1 Unknown Obstacle

The robot was initially given with a general probability of obstacle occurrence in the map when testing scanning function, but this may always have to be randomly higher or lower than the real occurrence probability, thus less convincing. This was replaced by given number of unknown obstacles and an obstacle occurrence probability sheet (three-dimensional list type variable) with known obstacles marked with 1 (definitely has an obstacle) and unknown obstacle occurrence probability marked for the rest of the positions initially. After each move is taken, if the move is *successful*, the resultant position is fill with 0, otherwise filled with 1; each time the robot scans a 3 by 3 by 3 cubic area is fully labelled, free positions are marked as 0, hidden obstacles are marked as 1. With each hidden obstacle marked as 1, the remaining hidden obstacle number minus 1. Thus the unknown obstacle occurrence probability can be updated as:

$$\frac{\text{remaining hidden object num}}{\text{position num} - \text{known obstacle num} - \text{known free position num}}$$

The probability is passed to the state transition matrix so that when the robot takes a move, the bumping happens at the move target position with a general probability; when it scans, the transition probability of knowing n more unknown block(s) is calculated by:

$$\begin{aligned}
& \binom{\text{(undetected position num)}}{n} \\
& \times \\
& \text{(hidden obstacle probability)}^n \\
& \times \\
& (1 - \text{hidden obstacle probability})^{\text{undetected position num}-n}
\end{aligned}$$

The probability of scanning nothing is calculated by subtracting all probabilities calculated for knowing 1 to n more obstacles from 1. Therefore, the probability of hidden obstacle occurrence changes along with the robot progressing, and decision process is dynamically determined.

The cost matrix setting is closely related to the transition matrix. The cost at each position is a negative cost coefficient multiplied with the probability of failed scanning at that position (taken from transition probability table), indicating that the higher probability it is to fail in scan, the less encouraged it is in the cost matrix.

$$k \cdot (\text{Pr}(\text{scans nothing})), k < 0$$

If the robot is bumped accidentally, the cost of bumping will be calculated in the next move to simulate the difficulty in recovering in reality. Additionally, a dynamic cost table is designed for the scanning action. A two-state cost is set to discourage the robot to scan continuously, once a scan takes place the cost for scanning at any position would be negative and the magnitude would suffice to force the next action to be a move.

D. Implementation

The scenario is set up and divided into the following classes: the obstacles, the map, the robot and the environment, and each class has an instance initialized in the class on its right in this sequence.(Figure 38 in section VI) The obstacles are divided into known and unknown ones, the second of which is not directly accessible to the robot. The robot instance should try to find a safe and optimal path with information given by the environment and the available part of map.

Main calculations of value iteration take place in the robot class(i.e., functions of Robot), any transition towards known obstacles is blocked in the state transition table and the end state is set to be absorption state by forbidding moving to adjacent positions once step on it.

In the start-up scenario, the robot generates policies by MDP considering only the known obstacles, and conducts a pure trial and error, which means bumping into unknown obstacles still occurs and would append these positions to the list of the known, then update the optimal policy towards the destination.

In the advance scenario the error-triggered update is preserved, however the scanning function is involved to simulate the sensor functions of the robot.

E. Proof-of-concept

The robot is represented by r, trying to avoid obstacles and reach the end point (shown as a square character). Known obstacles are shown as “=” and unknown ones are shown as “.”. “o” stands for empty positions. The policy index from 0 to 6 stands for: up, down, left, right, forward, backward, scan.

```

Episode: 1          position: [1, 1, 0]
start position:[0, 1, 0] layer 1
step:1----
layer 1             0 0
0 0                 0 0
0 0                 layer 2
                     0 □
layer 2             3 5
3 5                 r -
3 5

```

Figure 5. First Step of Start-up Scenario Example

```

step:2---- position: [1, 1, 0]
layer 1       Bump!
0 0           layer 1
0 0           0 0
              0 0
layer 2       layer 2
3 5           0 □
3 5           x =

```

Figure 6. Bumping and Recognizing the Unknown Obstacle

```

step:3---- position: [1, 0, 0]
layer 1       layer 1
0 0           0 0
0 5           0 0
              layer 2
layer 2       r □
3 5           0 =
5 5

```

Figure 7. Updating Policy and Making a Detour

```

step:4---- position: [1, 0, 1]
layer 1       layer 1
0 0           0 0
0 5           0 0
              layer 2
layer 2       0 r
3 5           0 =
5 5           cost:8

```

Figure 8. Reaching the Destination

The start-up scenario works perfectly to exert an optimal policy to by-pass the known obstacles, even if unexpected blockage is encountered, the robot still updates policy correctly to find its way to the end(Figure3-6). The figures above show the steps from start point (0,1,1) to end point (1,0,1). This means our algorithm is working on the simplest 3D navigation.

```

start position:[0, 0, 0]
step:1----
next move is SCAN
***Policy table***
layer 1
6 3 4
6 3 4
3 3 6
***Policy table***

+++Probability table+++ position: [0, 0, 0]
layer 1
0 1 0.75
0 0 0.75
1 0.75 0.75
+++Probability table+++
r = 0
0 0 0
= - □

```

Figure 9. Scans to Uncovers Hidden Obstacles

```

step:2----
next move is FORWARD
***Policy table***
layer 1
4 3 4
3 3 4
3 3 6
***Policy table***
+++Probability table+++
layer 1
0 1 0.75
0 0 0.75
1 0.75 0.75
+++Probability table+++
position: [0, 1, 0]
layer 1
0 = 0
r 0 0
= - □

```

Figure 10. Policy Updates

```

step:5----
next move is FORWARD
***Policy table***
layer 1
6 3 6
3 3 4
3 3 2
***Policy table***
+++Probability table+++
layer 1
0 1 0.99
0 0 0
1 0.99 0
+++Probability table+++
position: [0, 2, 2]
layer 1
0 = 0
0 0 0
0 0 0
= - r

```

Figure 11. Robot Succeed in Reaching the Goal

In the advance scenario, not only does the robot find the end state, but also decide to scan when it decides it's highly possible to encounter hidden obstacles in a 3x3x3 cube (for display convenience the example has 1 layer only). The scan successfully reveals a hidden obstacle and updates the safe policy (Figure 7-9).

```

step:1----
next move is SCAN
***Policy table***
layer 1
6 3 4
6 3 4
3 3 6
***Policy table***
position: [0, 0, 0]
layer 1
r 0 -
0 0 0
= - □

```

Figure 12. A Wasted Scan

However, the robot may conduct unnecessary scans while fail to stop and inspect when beside the unknown obstacles due to different positions of hidden obstacles (Figure 10), and this is largely related to the setting of initial general probability.

F. Further Improvement

To tackle with the wasted scan problem, the program iterates through multiple additional episodes to find the most appropriate cost coefficient in the advance scenario. This is done by multiplying a bias (float type) variable (ranging from 1 to 2) to the original reward matrix of scanning action, which increase the magnitude of negative numbers in order to discourage scanning at a reasonable level.

The basic logic of code runs as follows. In the first and second episode the robot is forced to make decisions under $\text{bias}=0$ and $\text{bias}=1$. The cost in the second episode is therefore more expensive than the first recorded as scanning would be almost banned from action list. The bias in the first episode (0) is recorded as lastBoundBias . Then the new episode takes half of the sum of the current bias and lastBoundBias to reduce the discouragement as the new bias. If the new episode is still more costly than the minimum cost recorded, repeat the last step again; if not, try to increase the bias by taking the half of the sum of the current bias and last bias that make it more costly than the minimum recorded. After that the program proceeds until the end of the episodes. If the bias is the same with an accuracy of at least four decimal places for 3 successive episodes, the bias is converged.

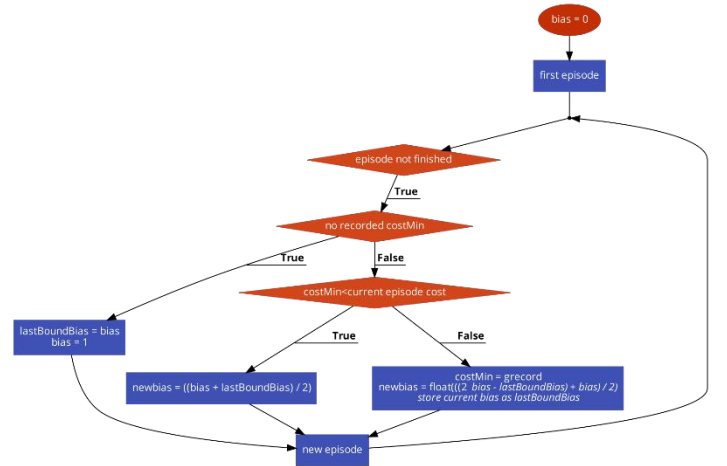


Figure 13. Flowchart of Bias Calculation Logic

Additionally, when resetting each episodes, the initial probability given is no longer the occurrence probability of unknown obstacles, but rather occurrence probability for both unknown and known obstacles.

Three different maps will be designed to compare the steps taken and the cost among start-up scenario, advance scenario and advance scenario with iterations.

IV. RESULTS

The following results show figures of the map design interface and the route render interface. Different colored squares have different meanings which can be checked in the denotation table in section V.

A. Map 1: 1x3x3 (layer, row, column)

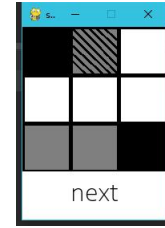


Figure 14. The First Map Tested (Map Design Interface)

The first map tested is 1x3x3 one layer space with start point on the top left and end point on the bottom right. Two known obstacles (grey blocks) and one unknown obstacles (grey block with slashes) are placed to test the performance under basic conditions.

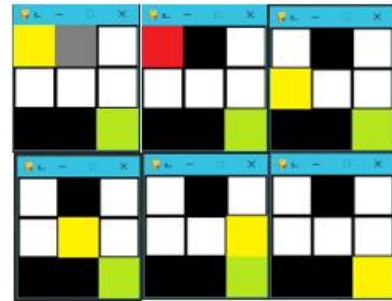


Figure 15. Start-up Scenario Route for Map1


```

51 step:5----
52 next move is FORWARD
53 position: [0, 2, 2]
54 layer 1
55 o = o
56 o o o
57 = = r
58
59 time left:95;done:True
60 cost:-13.0
61 -----

```

Figure 16. Start-up Scenario Console Result for Map1

In the start-up scenario, the agent directly bumps into the hidden obstacle beside it, taking a huge cost before reaching the end. The resultant cost is -13 and the step used is 5.

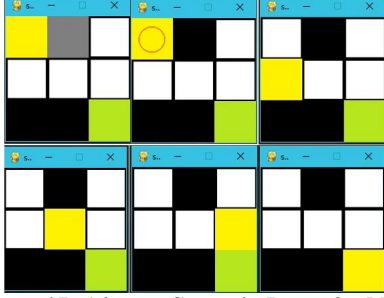


Figure 17. Advance Scenario Route for Map1

```

58 step:5----
59 next move is FORWARD
60 The bias is: 0.0
61 position: [0, 2, 2]
62 layer 1
63 o = o
64 o o o
65 = = r
66
67 time left:95;done:True
68 cost:-3.6527777777777777
69 -----

```

Figure 18. Advance Scenario Console Result for Map1

In the advance scenario, the agent first scans (yellow), then discovers the hidden obstacle beside it. It eventually reaches the end. The resultant cost is -3.65 and the step used is 5. The solution is less expensive, but uses same time step.

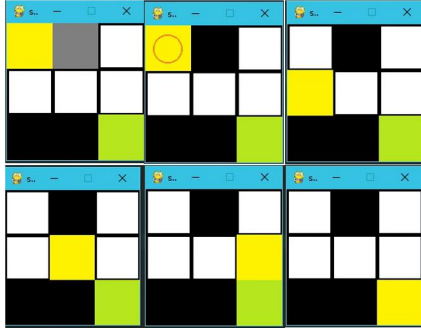


Figure 19. Advance (with Iteration) Scenario Route for Map1

```

788 time left:95;done:True
789 cost:-3.5569809027777777
790 The bias is: 0.0009765625(1.0009765625) in episode 12
791 costMin -3.556;grecoed -3.557
792 reduce bias!
793 -----
854 time left:95;done:True
855 cost:-3.558268229166665
856 The bias is: 0.00048828125(1.00048828125) in episode 13
857 costMin -3.556;grecoed -3.556
858 reduce bias!
859 -----
920 time left:95;done:True
921 cost:-3.55691189236111
922 The bias is: 0.000244140625(1.000244140625) in episode 14
923 costMin -3.556;grecoed -3.556
924 reduce bias!
925 -----
986 time left:95;done:True
987 cost:-3.55623372395833
988 The bias is: 0.0001220703125(1.0001220703125) in episode 15
989 costMin -3.556;grecoed -3.556
990 reduce bias!
991 -----

```

Figure 20. Advance (with Iteration) Scenario Console Result for Map1

In the advance (with iteration) scenario, the convergence in bias gives the same route as in normal advance scenario. The resultant cost is -3.556 and the step used is 5. The cost is the least expensive in all the scenario. Time step used is still the same.

B. Map 2: 1x5x5

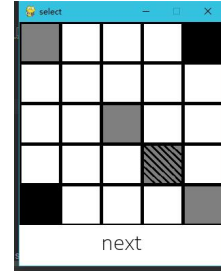


Figure 21. The Second Map Tested (Map Design Interface)

The second map tested is 1x5x5 one layer space with start point on the bottom left and end point on the top right. Three known obstacles (grey blocks) and one unknown obstacles (grey block with slashes) are placed to test the performance of path decision under half-known situation, i.e. one of two path is blocked secretly.

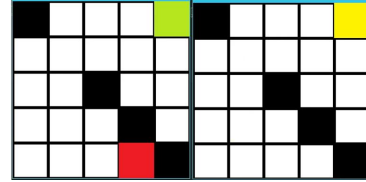


Figure 22. Start-up Scenario Vital Steps for Map2

```

142 step:13----
143 next move is BACKWARD
144 position: [0, 0, 4]
145 layer 1
146 = o o o r
147 o o o o o
148 o o = o o
149 o o o = o
150 o o o o =
151
152 time left:87;done:True
153 cost:-21.0
154 -----

```

Figure 23. Start-up Scenario Console Result for Map2

In the start-up scenario, the agent bumps into the hidden obstacle in one of the paths, tollay ignoring the hidden obstacles, taking a huge cost before it reaches the end. The resultant cost is -21 and the step used is 13.

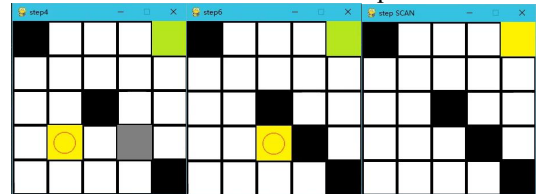


Figure 24. Advance Scenario Vital Steps for Map2

```

148 step:12----
149 next move is BACKWARD
150 The bias is: 0.0
151 position: [0, 0, 4]
152 layer 1
153 = o o o r
154 o o o o o
155 o o = o o
156 o o o = o
157 o o o o =
158
159 time left:88;done:True
160 cost:-10.413121837148843
161 -----

```

Figure 25. Advance Scenario Console Result for Map2

In the advance scenario, the agent first scans multiple times, then discovers the hidden obstacle. It eventually

reaches the end. The resultant cost is -10.413 and the step used is 12. The solution is less expensive and takes fewer time steps.

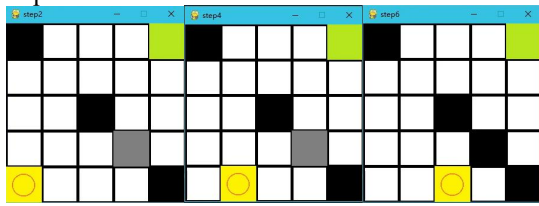


Figure 26. Advance (with Iteration) Scenario Vital Steps for Map2, Episode 1

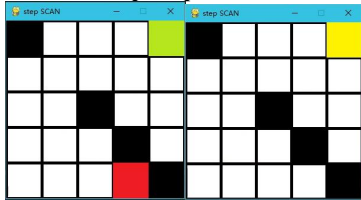


Figure 27. Advance (with Iteration) Scenario Vital Steps for Map2, Episode 2-3

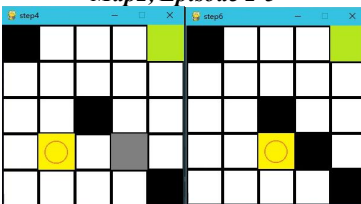


Figure 28. Advance (with Iteration) Scenario Vital Steps for Map2, Episode 4

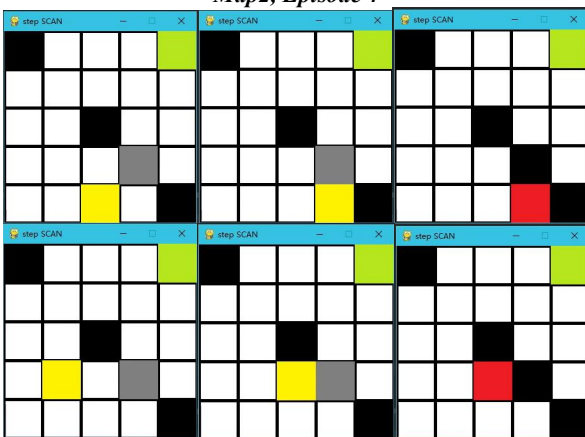


Figure 29. Advance (with Iteration) Scenario Route before Bumping in Map2, Episode 5(upper row) and 6 (lower row)

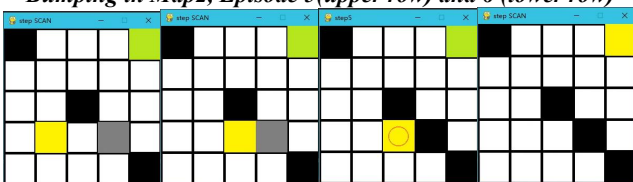


Figure 30. Advance (with Iteration) Scenario Vital Steps for Map2, Episode 7



Figure 31. Bumping Position in Advance (with Iteration) Scenario in Map2, Episode 8 and 9

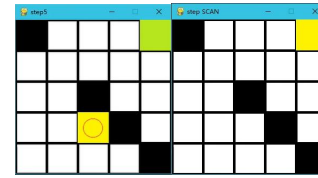


Figure 32. Advance (with Iteration) Scenario in Map2, Last Episodes

```
1822 time left:89;done:True
1823 cost:-9.928352442807954
1824 The bias is: 0.2822265625(1.2822265625) in episode 12
1825 costMin -9.928;grecord -9.929
1826 reduce bias!
1827 -----
1966 time left:89;done:True
1967 cost:-9.927998919410085
1968 The bias is: 0.28173828125(1.28173828125) in episode 13
1969 costMin -9.928;grecord -9.928
1970 reduce bias!
-----
2110 time left:89;done:True
2111 cost:-9.927822157711148
2112 The bias is: 0.281494140625(1.281494140625) in episode 14
2113 costMin -9.928;grecord -9.928
2114 reduce bias!
2115 -----
2254 time left:89;done:True
2255 cost:-9.92773377686168
2256 The bias is: 0.2813720703125(1.2813720703125) in episode 15
2257 costMin -9.928;grecord -9.928
2258 reduce bias!
2259 -----
```

Figure 33. Advance (with Iteration) Scenario Console Result for Map2

In the advance scenario with iterations, the agent takes more scans in the first episode than it does in the normal advance scenario, then starts to calibrate the bias in the later episodes. In figure 23 and 24 it is clear that the calibration brings down the number of time of scans while avoiding bumping at the same time. Then the agent starts to calibrate further to reduce the scanning, which cause the robot to bump again in episodes 5 and 6, but finally scans only once beside the hidden obstacle. It takes several episodes more to find the most accurate bias, bumping still takes place occasionally, but eventually the best solution is obtained. The resultant cost is -9.928 and the step used is 11. This result is the least expensive and takes the least time steps of all.

C. Map 3:2x5x5

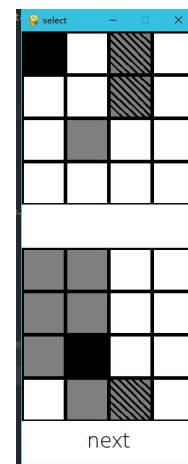


Figure 34. The Third Map Tested (Map Design Interface)

The third map tested is 2x5x5 two-layer space with start point on the first layer, top left and end point on the second layer. Known obstacles (grey blocks) and unknown

obstacles (grey block with slashes) are placed to test the performance of navigation under more complicated 3D environment.

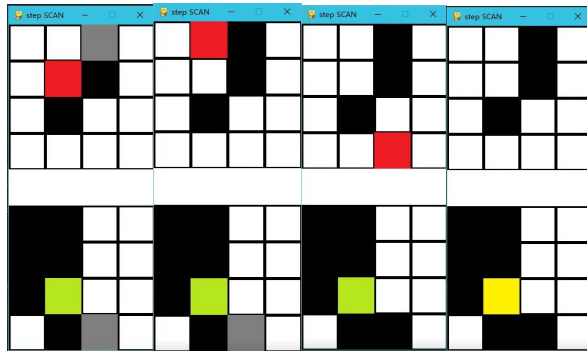


Figure 35. Start-up Scenario Vital Steps for Map3

```

230 step:15---
231 next move is LEFT
232 position: [1, 2, 1]
233 layer 1
234 o o = o
235 o o = o
236 o = o o
237 o o o o
238 layer 2
239 = = o o
240 = = o o
241 = r o o
242 o = = o
243
244 time left:85;done:True
245 cost:-41.0
246 -----

```

Figure 36. Start-up Scenario Console Result for Map3

In the start-up scenario, the agent bumps into every hidden obstacle, taking huge costs before reaching the end. The resultant cost is -41 and the step used is 15.

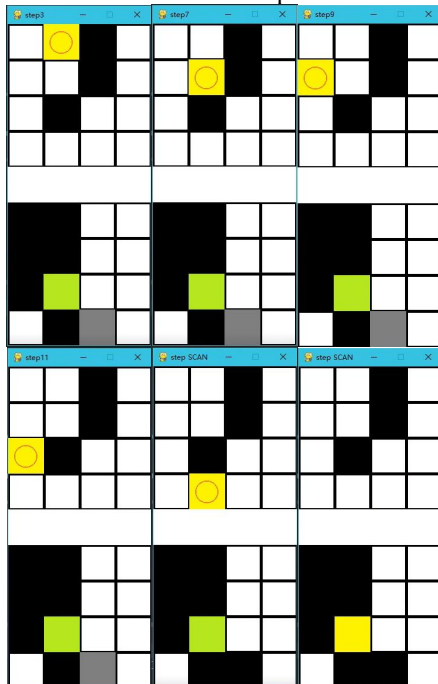


Figure 37. Advance Scenario Vital Steps for Map3

```

287 step:17---
288 next move is LEFT
289 The bias is: 0.0
290 position: [1, 2, 1]
291 layer 1
292 o o = o
293 o o = o
294 o = o o
295 o o o o
296 layer 2
297 = = o o
298 = = o o
299 = r o o
300 o = = o
301
302 time left:83;done:True
303 cost:-13.794479638296261
304 -----

```

Figure 38. Advance Scenario Console Result for Map3

In the advance scenario, the agent first scans multiple times to discover all the hidden obstacle before reaching the end, yet a couple of the scans are wasted scans. The resultant cost is -13.794 and the step used is 17. The cost is much fewer than it is in the start-up scenario but takes too much time.

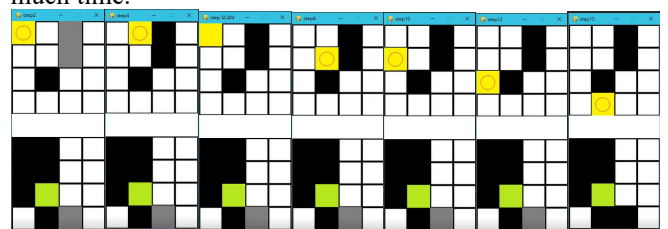


Figure 39. Advance (with Iteration) Scenario Vital Steps for Map3, Episode 1

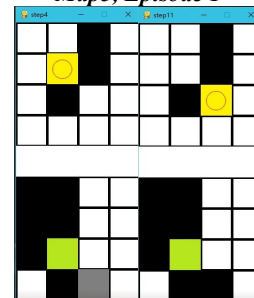


Figure 40. Advance (with Iteration) Scenario Vital Steps for Map3, Episode 2

```

2668 step:12----
2669 next move is LEFT
2670 The bias is: 1.0009765625(2.0009765625)
2671 position: [1, 2, 1]
2672 layer 1
2673 o o = o
2674 o o = o
2675 o = o o
2676 o o o o
2677 layer 2
2678 = = o o
2679 = = o o
2680 = r o o
2681 o = = o
2682
2683 time left:88;done:True
2684 cost:-10.768330718950399
2685 The bias is: 1.0009765625(2.0009765625) in episode 12
2686 costMin -10.768;grecord -10.769
2687 reduce bias!
2688 -----
2883 step:12----
2884 next move is LEFT
2885 The bias is: 1.00048828125(2.00048828125)

2886 position: [1, 2, 1]
2887 layer 1
2888 o o = o
2889 o o = o
2890 o = o o
2891 o o o o
2892 layer 2
2893 = = o o
2894 = = o o
2895 = r o o
2896 o = = o
2897
2898 time left:88;done:True
2899 cost:-10.767899208282035
2900 The bias is: 1.00048828125(2.00048828125) in episode 13
2901 costMin -10.768;grecord -10.768
2902 reduce bias!
2903 -----
3098 step:12----
3099 next move is LEFT
3100 The bias is: 1.000244140625(2.000244140625)
3101 position: [1, 2, 1]
3102 layer 1
3103 o o = o
3104 o o = o
3105 o = o o
3106 o o o o
3107 layer 2
3108 = = o o
3109 = = o o
3110 = r o o
3111 o = = o

3113 time left:88;done:True
3114 cost:-10.767683452947855
3115 The bias is: 1.000244140625(2.000244140625) in episode 14
3116 costMin -10.768;grecord -10.768
3117 reduce bias!
3118 -----
3313 step:12----
3314 next move is LEFT
3315 The bias is: 1.0001220703125(2.0001220703125)
3316 position: [1, 2, 1]
3317 layer 1
3318 o o = o
3319 o o = o
3320 o = o o
3321 o o o o
3322 layer 2
3323 = = o o
3324 = = o o
3325 = r o o
3326 o = = o
3327
3328 time left:88;done:True
3329 cost:-10.767575575280764
3330 The bias is: 1.0001220703125(2.0001220703125) in episode
15
3331 costMin -10.768;grecord -10.768
3332 reduce bias!
3333 -----

```

Figure 41. Advance (with Iteration) Scenario Console Result for Map3

In the advance scenario with iterations, the agent takes more scans in the first episode than it does in other scenario, and even takes redundant routes. Then immediately in the second episode it finds the best route and takes only 2 scans. In figure 37 it shows that the calibration brings down the cost to -10.768, and the time step taken is 12, both are the least in among all the scenario.

V. CONCLUSIONS AND DISCUSSION OF FUTURE WORK

Generally, the algorithm succeeds in guiding the robot to reach destination(if it's reachable). It finds optimal policy towards the destination dynamically, and the scanning function helps in planning a safer route than it does without any scanning, but when the map starts to get larger or complicated it takes longer time. After iterations, the solution routes are generally cheaper and faster. This is probably because a larger initial occurrence probability in combination with more discouragement in scanning makes the agent take scanning more seriously and careful when deciding the path, trying to make it more valueable. The algorithm is ready to be applied in simple real-world environment for further tests. However, there are still limitations and relative research prospects that need to be discussed.

A. A priori?

The first one regards the probability mentioned above. In the example given the robot is given with the actual number of hidden obstacles, which cannot happen in reality. Also, most researchers have little idea about the operation spaces they are facing as well as its probability of hidden obstacle occurrence in the scenario we mentioned, which means a more reasonable way of determination of occurrence probability is needed. Take underwater area for example, the researchers may have to obtain a rough probability through inspecting marine geology and plate movement in advance. Besides from having to reach out to other means to have a hypothesis of these figures, it's possible the robot could be damaged after testing for multiple times if we want the robot to learn the best bias to avoid random guess and inefficiency. This calls for assigning different probabilities distributions to different positions at the start and a better reward matrix for scanning actions, which may improve the efficiency of iterations.

B. Uncertainty in the Number of States

The second issue regards the state space determination. An obvious contradiction is that the state space is determined by knowing the number of hidden obstacles previously while theoretically this is not accessible. In future work one urgent need is to allow appending new states and find the proper conditions to append extra states before encountering a bump.

C. Other Further Improvements

The third includes a set of future advancement and addressing solutions to detailed difficulties: in a more realistic sense the obstacles would move periodically or randomly, giving possibilities of occurrence within a certain area rather than static hidden obstacles, the robot should learn these patterns of occurrence; noises will be introduced to the scanning function to better simulate the realistic equipment limitations; a tracking mode of dynamic destination can be briefly designed to comply with more applications and needs.

VI. FIGURES

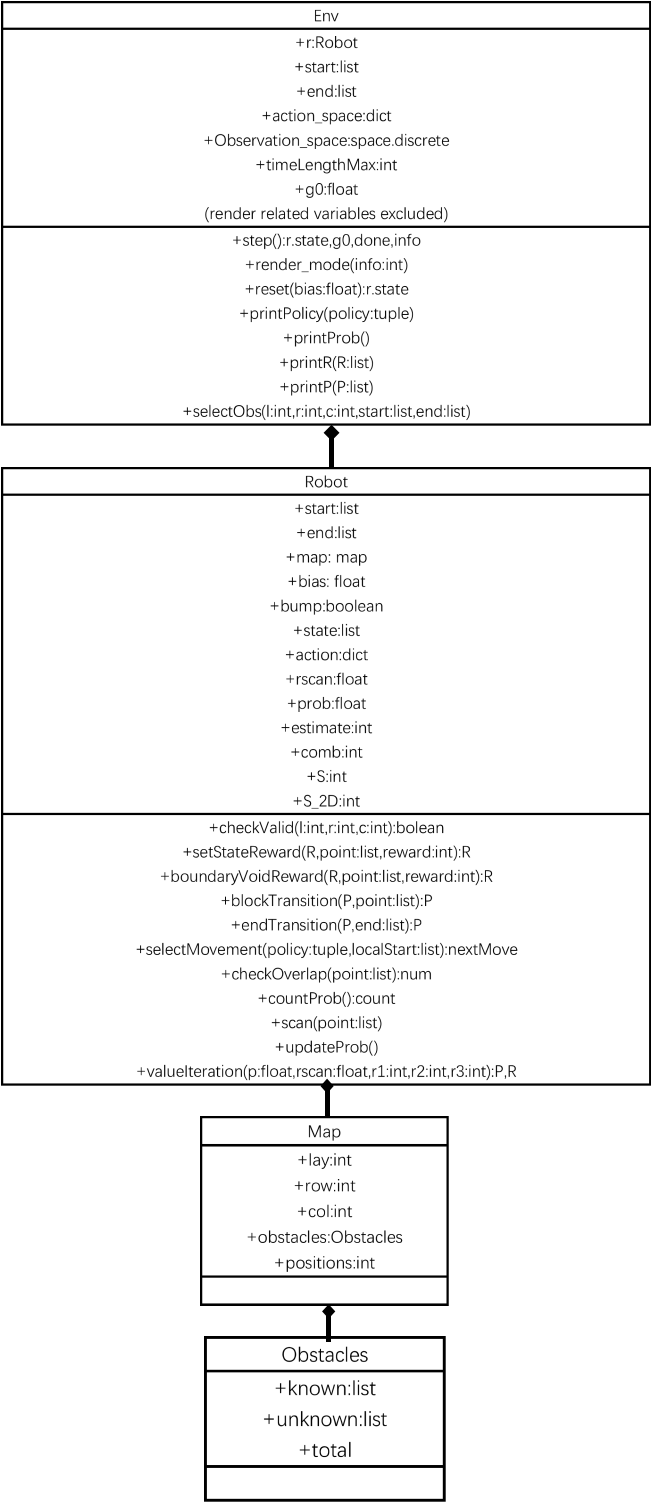


Figure 42. UML Diagram of the Scenario

Known obstacles	
Unknown obstacles	
Vacant	
Start and end	

Figure 43. Denotation Table of Different Patterns in Map Design Interface

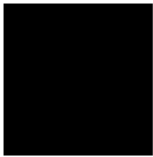

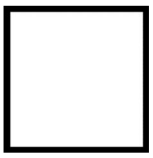
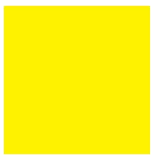

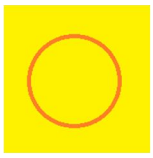

Known obstacles	
Unknown obstacles	
Vacant	
Robot (normal)	
Robot (bumped)	
Robot (scan)	
End	

Figure 44. Denotation Table of Different Patterns in Route Render Interface

APPENDIX: Contribution

Algorithm design and programming: Liu Enxu

Report writing: Liu Enxu, Zhu Hongyi

REFERENCES

- [1] J. Forbes, T. Huang, K. Kanazawa, and S. Russell, "The batmobile: Towards a bayesian automated taxi," in Int. Joint Conf. on Artificial Intelligence, Montreal, Quebec, Canada, 1995, pp. 1878–1885.
- [2] Sebastian Brechtel, Tobias Gindele, and Rüdiger Dillmann, "Probabilistic MDP-Behavior Planning for Cars", IEEE, pp.1537-1542, 2011

- [3] Mark A. Mueller, "REINFORCEMENT LEARNING: MDP APPLIED TO AUTONOMOUS NAVIGATION", Machine Learning and Applications, Vol.4, 2017
- [4] Bai H, Hsu D, Kochenderfer MJ, Lee WS. Unmanned aircraft collision avoidance using continuous-state POMDPs. Robotics: Science and Systems VII. 2012 Jun 29;1:1-8
- [5] Kolobov A. Planning with Markov decision processes: An AI perspective. Synthesis Lectures on Artificial Intelligence and Machine Learning, 38-41, 2012 Jun 30
- [6]" Markov Decision Process (MDP) Toolbox for Python"
<https://pymdptoolbox.readthedocs.io/en/latest/api/mdptoolbox.html>
(accessed Aug 14, 2022).