Robotics: Manipulation, Decision Making and Learning

Project Work

Implementation of Reinforcement Learning (Q-learning) based robotic navigation system to help a robot to move out of an unknown maze autonomously

Laura Garcia (604697)

Shikha Sharma (605159)

1. Introduction

Navigation task for an autonomous robot in an unknown environment is a challenging task. In this report, reinforcement learning based approach is proposed to help a robot to move out of an unknown maze. The basic idea behind robotic navigation is to find a collision free path from the start state to goal state. For this the robot has to collect information from the environment by using sensors and make navigation decisions based on the sensory data and its knowledge base. Due to its simplicity and good real time performance, Q-learning could be quite effective to make a robot learn the environment quickly and navigate inside it successfully.

Section 2 of the report explains the approach taken to solve the problem in hand and results obtained. Analysis of the results, limitations of the approach and future direction are given in section 3. Section 4 list the references.

2. Approach

The robot has to learn two optimal policies to navigate autonomous out of the maze: (1) Policy-I for obstacles avoidance and (2) Policy-II for traversing the maze. Reinforcement learning is applied for both of the policies. The approach for these two policies is explained in the subsequent section. A small mobile robot was also built to demonstrate the obstacle avoidance.

2.1 Policy-I

A Q-learning based approach is used here to teach the robot obstacle avoidance. As a kind of reinforcement learning algorithm, Q-learning has the following advantages: 1) Low requirements of computational resources, 2) Good real-time performance, 3) it does not need training samples.

In order to determine the world states in the Q-learning algorithm, three IR proximity sensors are used to detect the distances between the robot and the closest obstacles in three specific directions.

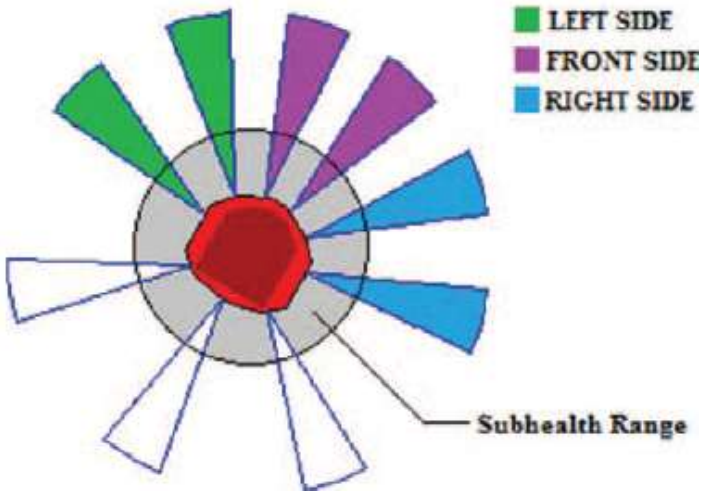


Figure: Schematic Diagram of sensors

The three specific directions have been defined as three different color regions. State vector is defined by relative distance output from the three sensors.

State: s = [left right front]

In particular, the three elements of the vector represent the left, right and front distance and their relative amplitudes. Each element is an integer number ranging from "1" to "3" e.g. [1 2 3] state vector represents largest front distance and smallest left distance. The actions available are:

- Action F: Move forward for 100mm
- Action L: Turn left for 15°
- Action R: Turn right for 15°

In addition, in order to prevent the robot from hitting an obstacle, a protection action is applied in the training process: if one of three distances is smaller than 170mm, the robot will move backward for 100mm.

For each action in a specific state, a reward value is defined. As shown in following figure. The world states are described as health and sub heath state. A health state indicates the obstacles are still very far away from the robot, and they will not threaten its movement. Hence, after an action is executed, if a health state is observed by the robot, a positive reward will be received. On the other hand, a sub-health state indicates that at least one of obstacles is very close to the robot and the robot needs to move away from it immediately. Thus, a robot in a sub-health state will always receive a negative reward.

$R = \begin{array}{c} state \backslash action \\ 123 \\ 321 \\ 132 \\ 312 \\ 213 \\ 231 \end{array} \begin{array}{c} L \\ \begin{bmatrix} 0 \\ 5 \\ 0 \\ 5 \\ 1 \\ 1 \end{bmatrix} \end{array} \begin{array}{c} R \\ \begin{bmatrix} 1 \\ 1 \\ 5 \\ 0 \\ 0 \\ 5 \end{bmatrix} \end{array} \begin{array}{c} F \\ \begin{bmatrix} 5 \\ 0 \\ 1 \\ 1 \\ 5 \\ 0 \end{bmatrix} \end{array}$	$R = \begin{array}{c} state \backslash action \\ 123 \\ 321 \\ 132 \\ 312 \\ 213 \\ 231 \end{array} \begin{array}{c} L \\ \begin{bmatrix} -3 \\ 0 \\ -3 \\ 0 \\ -1 \\ -1 \end{bmatrix} \end{array} \begin{array}{c} R \\ \begin{bmatrix} -1 \\ -1 \\ 0 \\ -3 \\ -3 \\ 0 \end{bmatrix} \end{array} \begin{array}{c} F \\ \begin{bmatrix} 0 \\ -3 \\ -1 \\ -1 \\ 0 \\ -3 \end{bmatrix} \end{array}$
Health state	Sub health state
Reward Table	

2.1.1 Simulation

The obstacle avoidance policy, as introduced above, was first implemented on MatLab and the trajectory of the agent over time was plotted.

The agent (mobile robot) was modeled as a circumference with 3 relevant points: the origin of the circumference and three points on the circumference spaced 90 degrees to each other (representing the 3 sensors: front, left and right). The distance between the obstacles and the robot at any time was the Euclidean distance between the sensor point and the intercept point of a line connecting the sensor point and the obstacle (same line connecting the sensor point and the origin of the circumference). The maze used to simulate the algorithm was a spiral maze.

Algorithm

Estimate the start state
While state != terminal (exit of the maze)
Choose an action based on e-greedy method (exploration and exploitation approach)
Execute the action (forward, right or left)
Estimate the Euclidian distance between each sensors and the obstacle
Update the state vector and its type (health/subhealth)
Check if there is an emergency condition (any of the sensors has a distance shorter than the minimum allowed for subhealth state)
Obtained rewards based on prior state and executed action
Update the action value function Q

The Q-learning parameters were:

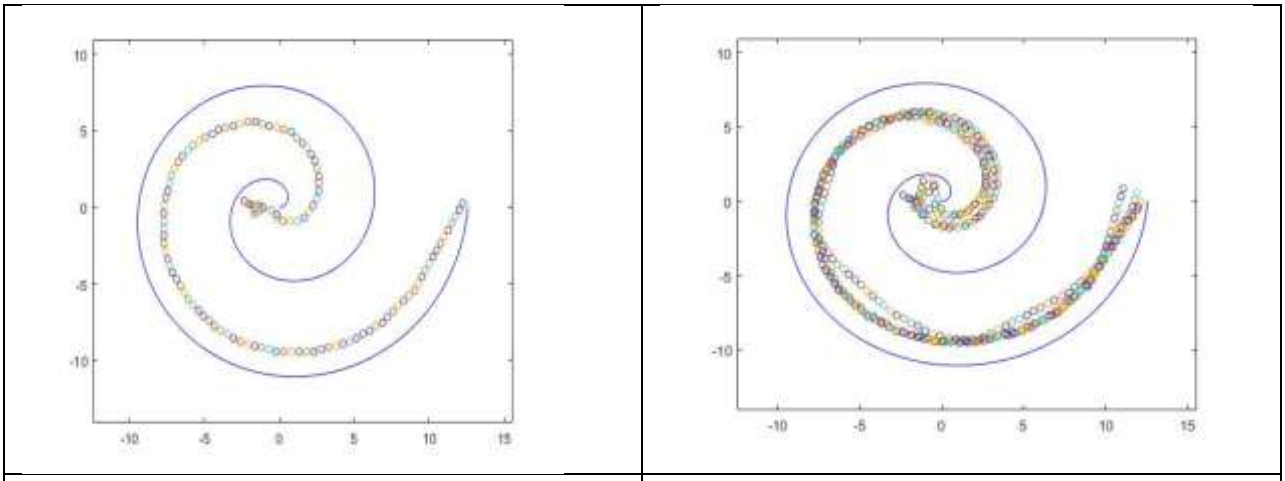
- Discount factor =0.9
- Step size constant=0.1
- Epsilon (e-greedy parameter)=0.1

The results of this implementation are presented on the following pictures, in which it can be seen the trajectory of the origin of the agent from the starting point (randomly selected at the beginning of the spiral maze) and the terminal point (the end of the maze) over time. The picture on the left side shows one episode and the picture on the left shows five episodes.

As it can be concluded, the obstacle avoidance policy based on Q-learning does allow the agent to move out of the maze successfully. The completion of the task takes variable time from iteration to iteration. According to the experience from simulations, the agent can take from just few seconds to minutes (<15mins) to travel from the given starting point to the end of the maze. This, for an action value function Q initialized with value zero and an epsilon of 0.1 (that allows action exploration 10% of the time and exploitation 90% of the time).

The above described approach was simulated in matlab environment with a spiral maze as shown in the following figure. The path becomes smoother as number of episodes is increased. The

distance output of the sensors is simulated by taking the Euclidean distance from the intersection with the maze.



One episode	Five Episodes

Figure: Simulation of the obstacle avoidance with a spiral maze

2.1.2 Demonstration with a robot- Hardware implementation

The obstacle avoidance policy was also implemented and tested on a mobile robot, developed specially for this project, that consists of the following components:

- Microprocessor: a Beaglebone black board running Linux as OS, providing with GPIO and PWM outputs to control the DC motors through the H-bridge. It also provides ADC inputs with a maximum analog voltage of 1.8 to sample the output of the sensors.

- A simple robot chassis consisting of two frames and two wheels, each of them attached to the shaft of a dc motor. There was no need of closed-loop control of the position of the shaft of the dc motors for this application.
- Three proximity sensors (IR sensors) able to detect objects located in a range from 10cms to 80cms away from the sensor
- Chip L293d (double H-bridge to drive the DC motors)
- Linear regulator to 5 volts
- Battery

Algorithm (Implemented on python)

Estimate the start state
While state != terminal (exit of the maze)
Choose an action based on e-greedy method (exploration and exploitation approach)
Execute the action (forward, right or left by enabling both motors, left motor or right motor, respectively)
Measure output voltage of the proximity sensors
Update the state vector and its type (health/subhealth)
Check if there is an emergency condition (any of the sensors has a distance shorter than the minimum allowed for subhealth state)
Obtained rewards based on prior state and executed action
Update the action value function Q

The results of this implementation can be seen on the video of the mobile robot traveling from a random start position in the maze to the end of it, proving that the algorithm was also successful on the hardware implementation.

The hardware implementation has some limitations mainly related to those limitations of the hardware components. For example, the limited sampling of the maze/obstacle provided by the chosen IR sensors (which can only detect obstacles from 10cms to 80 cms away from the sensor) and the number of them (it is recommended to add more sensors to sample more accurately the maze). Also, a closed-loop control of the position of the DC motors will provided a more accurate action execution. .

2.2 Policy-II

To navigate through the maze (as shown in following figure), it requires a policy to traverse from start state to end state. Various options considered for this are: (1) Episodic Q-learning with position in grid as state, (2) Non-episodic Q-learning with position of the obstacles as state and (3) Traversing maze from start to end with decision nodes as state. The policy learned in this section is used in conjunction with the obstacle avoidance policy

Figure: Example of the maze

2.2.1 Episodic Q-learning with position in grid as state

In episodic Q-learning, many iterations/episodes are run with exploring start and Q-table is updated according to the Q-learning algorithm. The states are defined as the position in the grid. It is assumed that the position is known with the help of some localization technique.

State: position in grid

Action: [East North West South]. The actions in each state are taken according to the maximum value of action in the q-table updated according to the q-learning algorithm.

Rewards are given for terminal state and non-terminal states fetch a negative reward

Parameters used for Q-learning: epsilon=0.1(e-greedy, probability of choosing action randomly); gamma=0.9 (discount factor); alpha=0.1 (step size constant)

This method gives the most optimal solution out of the three solutions discussed here but the method assumes to run some episodes and then solve the problem and is not good for the real-time running of the robot.

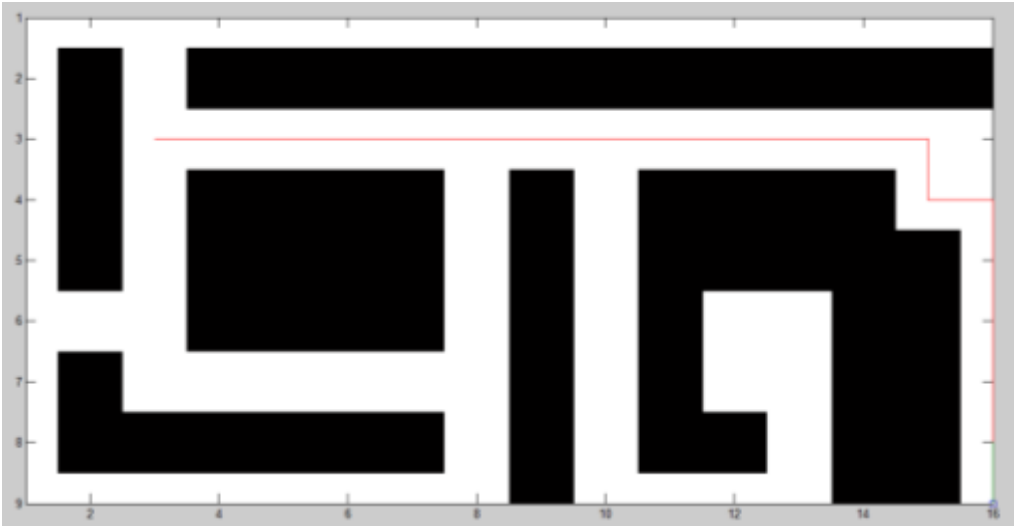


Figure: Episodic Q-learning with position in grid as state

2.2.2 Non-episodic Q-learning with position of the obstacles as state

In Non-episodic Q-learning, the robot starts from the start state are traverse through the maze while updating the Q-table. The states are defined as the position of the obstacles in the grid. At a given location robot senses the obstacles in all four directions (i.e. East, North, West and south) which forms the current state. Hence there are sixteen states. The idea behind this approach is to teach a robot favorable action in a maze depending on the maze. Some actions can be given a priority over others.

State: Obstacle position e.g. [0 0 0 0] – no obstacle; [1 0 0 0]- Obstacle in east; [1 1 1 0]- Obstacles in all directions apart from south. Hence, sixteen combinations of states are possible.

Action: [East North West South]. The actions in each state are taken according to the maximum value of action in the q-table updated according to the q-learning algorithm.

Rewards are given for terminal state and non-terminal states fetch a negative reward

Parameters used for Q-learning: epsilon=0.1(e-greedy, probability of choosing action randomly); gamma=0.9 (discount factor); alpha=0.1 (step size constant)

This method finds a solution but the running time of the algorithm highly depends on the exploration vs exploitation and the paths taken. The policy derived is not optimal.

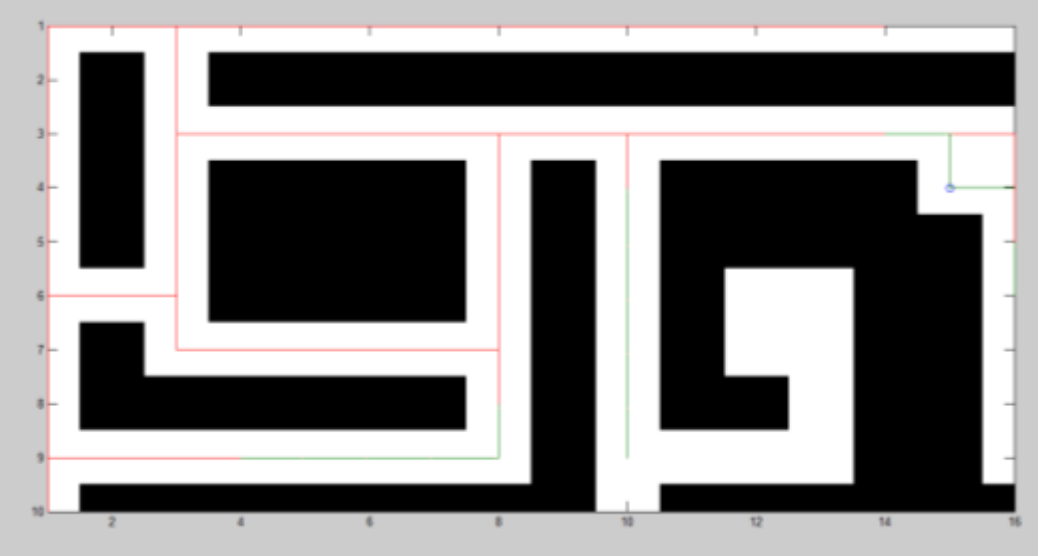


Figure: Non-Episodic Q-learning with position of obstacles as state

2.2.3 Traversing maze from start to end with decision nodes as state

This approach is also a type of non-episodic reinforcement learning, the robot starts from the start state and traverse through the maze while updating rewards for the last state depending on the current state. Negative rewards are given if it ends up in dead end or a loop. If a new state is encountered the positive reward is given to the corresponding action of the last state. The states are defined as the decision nodes where robot has a possibility of choosing an action from more than one action. The accumulated rewards over the time direct the robot to choose actions which have maximum rewards hence which are not dead ends or loop. The approach guarantees a solution. At a given location robot senses the obstacles in all four directions (i.e. East, North, West and south). It is assumed that the robot knows its position in the grid on basis of some localization sensors.

State: All of the decision nodes

Action: [East North West South]. The actions in each state are taken according to the maximum value of action in the updated state table.

Positive rewards are given for reaching meaningful state while dead ends and loops fetch negative reward.

This method finds a solution to the maze but the policy derived is not optimal.

Algorithm

Start traversing from start state
While (current position is not terminal)
{
If(decision_node)

If(state exist in state table)

#loop, update negative reward to the last states corresponding action

Action=Choose action with maximum reward

Else

Add state to state table

Update positive reward to the last states corresponding action

Action=Choose action with maximum reward

Elseif(only forward path available)

Action=Go straight;

Elseif(dead end)

Update negative reward to last states corresponding action;

Action=Goback;

}

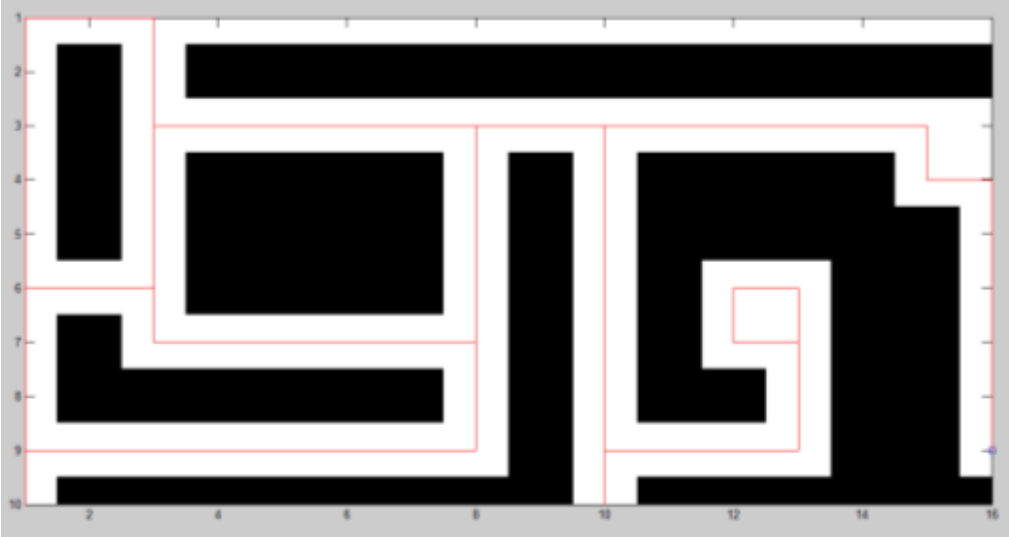


Figure: Traversing maze from start to end with decision nodes as state

3 Conclusions and Future Direction

Policy-I described in this report gives satisfactory results as an obstacle avoidance policy. This has been shown with the help of the simulations and also with hardware implementation. The three options given for policy-II have their positives and negatives. Episodic Q-learning with position in grid as state gives the most optimal policy but it requires a lot of episodes to be executed and is not suitable for running in real time environment. Non-episodic Q-learning with position of the obstacles as state and traversing maze from start to end with decision nodes as state are suitable to be run in an real time environment but both of these do not provide optimal policies. The policy in which maze is traversed from start to end with decision nodes as states is the best among three for the kind of applications we are aiming for.

Inaccuracy estimating states (sensors output being noisy) is the biggest limitation with the approaches described in this report. Hence, POMDP framework can be explored as future work.

4 References

[1] Bashan Zuo1 *Jiaxin Chen1, Larry Wang2 and Ying Wang1,"* A Reinforcement Learning Based Robotic Navigation System*",* 2014 IEEE International Conference on Systems, Man, and Cybernetics, October 5-8, 2014.

[2] Richard S. Sutton and Andrew G. Barto," Reinforcement Learning: An Introduction"