Markov Decision Processes for Path Planning in Unpredictable Environment

Timothy Boger and Mike Korostelev

May 7, 2012

Abstract

In the context of autonomous aerial and land vehicles, path planning is a challenging problem in unknown and semi-known environments. The problem we propose to address with this work is how to make decisions about what paths to take when the environment is known but unknown modifications have been made. This work will present the problem, the application of Markov Decision Process to approach solving it as well as a modification to MDP to formulate a damaged, inconsistent environment.

1 Introduction and Motivation

The motivation of this project is to construct a platform and develop software and automatons navigation techniques for a fully autonomous quadcopter to compete in the Drexel IARC (Indoor Aerial Robotics Competition) in May 2012. The quadcopter will be capable of lifting off from a designated spot, flying through a maze like set of corridors and landing at the designated destination. In order to do this efficiently path planning and maze solving algorithms will be combined with computer vision in an on-board Android device.

To create a scope for this problem we will consider a 5x5 maze with a single origin and a single goal. Multiple paths can be taken from the origin to goal. The agent traversing the maze has previous knowledge of the maze's configuration however after some shock to the environment, some of the maze spaces will present a hardship for a robot to traverse.

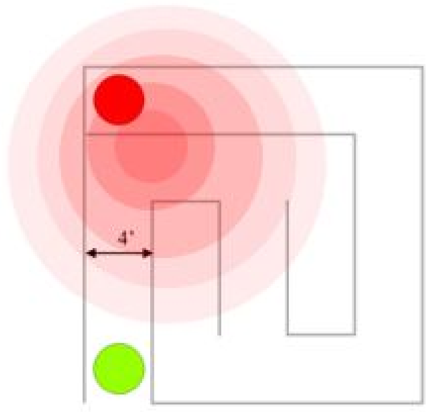
Figure 1: The type of UAV use as the application platform

Even with general knowledge of the shock event location, the agent can only discover these spaces by exploration. The challenge here is, sometimes because of a newly altered environment, an agent may have to reconsider its path to the goal. However as the agent is traveling through the maze the first shortcut may not necessarily be the best one and it may be more beneficial to not take this path and keep on course until a better one shows up.

This becomes similar to the secretary problem in the optimal stopping theory described by choosing a time to take a particular action, in order to maximize an expected reward or minimize an expected cost. To quantify the quality of the paths a reward system is considered, and since this problem involves a maze, we also plan to consider maze traversal algorithms like A* with a Manhattan heuristic to plan the paths. In our preliminary research, we came across papers that deal with similar exploration problems.

We plan to borrow some concepts in model based reinforcement learning and Markov Decision Processes for implementation in our scenario. Model free approaches attempt to learn optimal policies on without explicitly estimating the dynamics of the surrounding space. While mode based approaches attempt to estimate the model of the environments dynamics and use it to determine the value of actions that are possible.

Figure 2: A floorplan of a maze that has been affected by a damaging event that may change our movement patterns in the affected area.



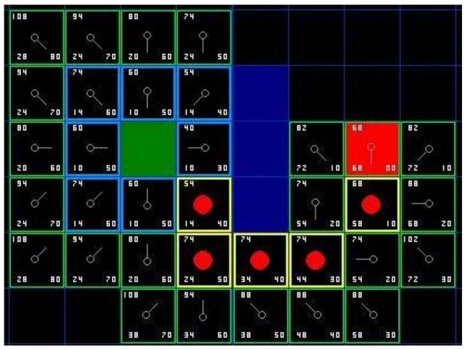
As a result of this project we want to decide weather it is beneficial to take a shorter path through a damage affected area or will the price outweigh the benefit and a shorter path is better.

2 Related Work

2.1 A* Algorithm

The A* Search Algorithm, Manhattan Heuristic is a very popular method for maze solving. Figure 1. shows an optimal route determined by A* Search, Manhattan Heuristic.

Figure 3: The progress of an agent through a Manhattan grid space using the A* algorithm with the Manhanttan heuristic



It is important to understand how this method works to understand how our proposed method differs. It begins by acknowledging the starting location, the green square, and adds it to the "open list." It then looks for the lowest cost square described by the following equations:

$$F = G + H$$

where, G is the movement cost to move from the starting point A to a given square on the grid, following the path generated to get there. H is the estimated movement cost to move from that given square on the grid to the final destination, point B. This lowest costing square is referred to as the "current square" and is switched to the "closed list". Each of the adjacent 8 squares from the current square are analyzed by determining which path to each of the squares is better. The current square becomes the parent and the path each square is checked. If the square is not walkable or if a is already on the "closed list" it is ignored. If a square is walkable and not on the "open list", it is added. The F, G, and H costs of the square is determined and if is already on the open list, its check, using the G cost, to see if the path to that square is better. A better path is determined with a lower G cost. If it is, the parent of the square is changed to the current square and recalculates the G and F scores of the square. The process stops when the target square is added to the closed list. Now to find the optimal path, the method works backwards from the target square going from each square to its parent square, shown as red dots, until the starting square is reached. Though A* Search does find the optimal path, it does not account for environmental statistics including potential obstacles, shortcuts, or errors that the hardware solving the maze may experience in certain environments. Markov Decision Process takes into account some of these aspects.

2.2 Markov Decision Process for Maze Solving

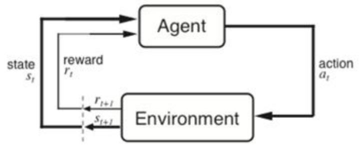
The Markov property is memoryless and when a reinforcement learning task has this property it is a Markov Decision Process or MDP. MDP is defined as by the following

$$S \text{ set of possible states where } s_t \text{ subset of } S$$
$$A(s_t) \text{ set of possible actions in state at time } t \text{ where } a_t \text{ subset of } A(s_t)$$

The MDP policy is a mapping from states to actions,

$$\pi^* : S \longrightarrow A$$

Figure 4: The reinforced learning process



Given the scenario of a 5X5 maze, with the current state S_t, the best action to take is evaluated.

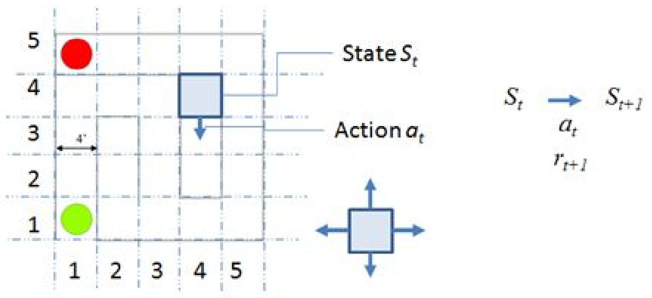
MDP essentially works on "One-step dynamics" where there is a transition probability associated with each direction. It is denoted by the following equation,

$$P^a_{ss^{'}} = Pr\{s_{s+1} = s^{'}|s_t = s, a_t = a\} \text{ for all } s, s_i subset of S, a subset of A(s)$$

The hardware that is performing the maze solving has a probability of transitioning to the next state. Example transition probabilities are shown below.

Though the hardware is told to move forward, there is a probability that an error could occur and the hardware would turn left or right instead. When weighing in this probability, MDP may avoid dangerous areas in fear that the hardware will maneuver into a restricted area in error. For the MDP, transition probabilities are static. For our project, we plan to dynamically change these transition probabilities to account for obstacles and in-turn determine an optimal path in the presence of potential path damage. In theory, our method would weigh the cost of taking the optimally shortest path, that may contain several obstacles, with that of a safer but longer route.

Figure 5: Maze scenario with MDP approach



Agent and environment interact at discrete time steps t = 0, 1, 2, ...

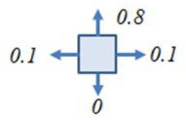
Agent observes state at step t: $s_t \in S$

produces action at step t: at $\in A(s_t)$

gets resulting reward: $r_{t+1} \in R$

gets to resulting state: s_{t+1}

Figure 6: State transition probability



3 Methodology

In order to evaluate the benefits of implementing Markov Decision Process with a dynamic state transition probability in our maze traversal routine, we need to first implement standard MDP.

The idea behind dynamic state transition is that depending on wether we are in the damage affected area, our probability $P^a_{ss'}$ of making an error movement becomes higher as we enter the area. The figure 6 probabilities now become a function of the states themselves.

With this in mind we need to still maximize our expected rewards:

$$E\{R\} \text{ where } R_t = r_{t+1} + r_{t+2} + ... + r_T$$

The T is the time until the terminal state is reached. When a task requires a large number of state transitions a discount factor γ is considered.

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^3 r_{t+3}... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$
$$where \gamma, 0 \leq \gamma \leq 1$$

A γ close to 1 corresponds to a farsighted method, and a γ close to zero corresponds to a nearsighted method. The best state to move to is determined by the Value of State which is the expected return starting from the state. The equations that will be affected by our dynamic $P^{a}_{ss'}$ are the Bellman Equations are a set of equations, one for each state and express a relationship between the value of a state and the value of its successor state. respectively,

$$Q^{\pi} = \sum_{S} P^{a}_{ss'} * [R^{a}_{ss'} + \gamma * \sum_{A(s')} \pi(s^{'}, a^{'} * Q^{\pi}(s^{'}, a^{'})]$$

$$V^{\pi} = \sum_{A(s)} \pi(s,a) \sum_{S} \pi(s,a) P^{a}_{ss'} * [R^{a}_{ss'} + \gamma * V^{\pi}(s^{'})]$$

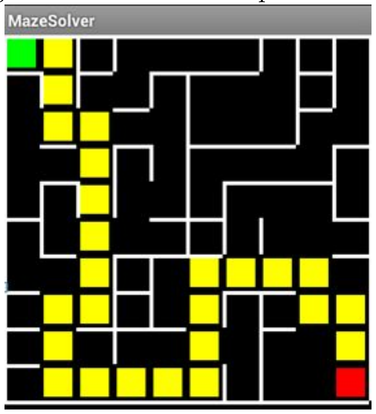
And with our extension, $P^{a}_{ss'}$ becomes:

$$P^{a}_{ss'}(s \subset S^{affected})$$

4 Experiments

We implemented, A* as well as MDP, A* was implemented in Java Android with successful results. The android implemenation was straightforward. Random mazes were generated and solved using the algorithm.

Figure 7: Android A* implementation



We have implemented MDP in MATLAB and tried placing a damaging event centered at various locations of the maze. This damaging event affected the state transition probabilities in the locations surrounding it. As a result, the state values changed accordingly to reflect the damaged area. In the following figure, a heat map of the values shows where the damage has affected the area. In the lower left is the goal of agent. In green, is the focus of the damage event.

In the following, we can also see that when we increase the size of the map and the damage is significantly far away from the goal, the value is affected less.

Figure 8: Damaged Environment Heat Maps

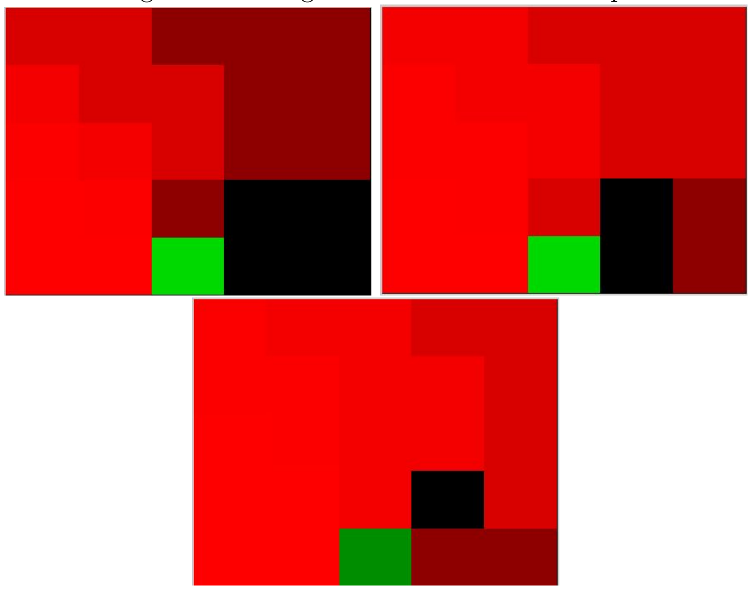
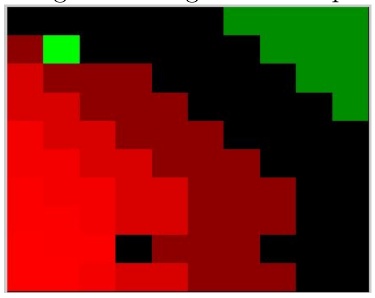


Figure 9: Larger Heat Map



5 References

1. `http://publications.asl.ethz.ch/files/bouabdallah07design.pdf`

2. `http://www.nada.kth.se/utbildning/grukth/exjobb/rapportlistor/2008/rapporter08/sikiric_vedran_08027.pdf`

3. Hongshe Dang; Jinguo Song; Qin Guo; , "An Efficient Algorithm for Robot Maze-Solving," Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2010 2nd International Conference on , vol.2, no., pp.79-82, 26-28 Aug. 2010

4. Sutherland, I.E.; , "A Method for Solving Arbitrary-Wall Mazes by Com-

puter," Computers, IEEE Transactions on , vol.C-18, no.12, pp. 1092-1097, Dec. 1969

5. `http://robotics.ece.drexel.edu/events/iarc/wp-content/uploads/2011/09/IARC-2012-v2.pdf`

6. Heckerman, D. (1998) , A tutorial on learning with Baseian networks. In M. I. Jordan ed., 'Learning in Graphical Models' Kluwer, Dordrecht, Netherlands.

7. Implementation of Q — Learning algorithm for solving maze problem, Osmankovic, D.; Konjicija, S.; Dept. of Autom. Control & Electron., Univ. in Sarajevo, Sarajevo, Bosnia-Herzegovina 2011

8. Reinforcement learning using chaotic exploration in maze world Morihiro, K.; Matsui, N.; Nishimura, H.; Hyogo Univ. of Teacher Educ., Japan 2004