Path Planning and re-planning using Model-Based Reinforcement Learning

Shubham Jain
Department of Electrical
and Computer Engineering
The University of Arizona
Tucson, AZ 85721
shubhamjain@email.arizona.edu

Abstract

Limited movement in constrained environments, like in medical surgeries, where free space for the instrument movement is limited, efficient and quick motion planning and re-planning for a specific task performance is necessary. Unfortunately, this space is defined with uncertainty as the obstacles might be moving, like, sudden blood flow or rhythmic heart-beat. At present, motion planners look for approaches that can make motion planning and re-planning, in dynamic environments, a quick task. This paper explores Finite Horizon problem from Model-Based Reinforcement Learning(MBRL) to suggest a novel approach, which, to best of my knowledge, is overlooked. Reinforcement Learning (RL) takes benefit of modeling the system as Finite Markov Decision Process and uses the concepts of Dynamic Programming to solve the problem at hand. Moreover, obstacle avoidance and collision checking are essential components in some situations, especially in medical surgeries, as these can have fatal consequences. The idea here is to reduce the time for re-planning and my approach takes advantage of random policy convergence to optimality through Contraction Mapping. Moreover, the fact that policy converges well before the values of the states, helps in finding the policy in short time. The simulation is done on a 30x30 grid, with the introduction of a 10x7 obstacle in the free space and test cases involve horizontal and diagonal movement, at every iteration. The policy and values are applied in a closed loop fashion, which helps considerably quick re-planning, dropped to few milliseconds from few seconds, on an average giving 140 times quick re-planning, than if it would have been initialized with random policy and values. Moreover, it is shown that initial *set-up time* can be reduced by carefully choosing theta and there exit a tradeoff between the time of initial iteration and subsequent iterations depending on the threshold, *theta*. Additionally, since movement effect, only a part of the environment and not the whole space, re-planning verifies every stage for policy convergence. Adaptive Dimensionality, which avoids checking all the states and just those which have the highest possibility of collision, can be further used for better results.

1 Introduction

Motion planning is a fundamental research area in robotics. Reinforcement Learning can be proved crucial for this application. RL involves an agent whose goal is to maximize the reward it gains for every step it chooses to take in the environment [6]. The success or failure is defined based on the rewards accumulated either until it reaches the goal state if one exists or in the amount of time that is allocated for the task. The RL agent is deployed in a finite horizon model and by using Policy Evaluation and Policy Improvement [7] over the iterations, it suggests the

optimal policy at every stage. The iterations for convergence is the planning stage which involves devising a collision-free strategy from the current location, or configuration, to the desired goal location or configuration. Sampling-based methods offer an efficient solution for what is otherwise a rather challenging dilemma of path planning. Also, the problem RL tackles cannot, in general, be solved by A* [8]. Additionally, RL problems are specified by Markov Decision Process, which in some cases can be shown as a shortest-path problem. Fortunately, as proved through Contraction Mapping theorem, Figure 1, the policy is certain to converge, hence providing the optimal path for motion.

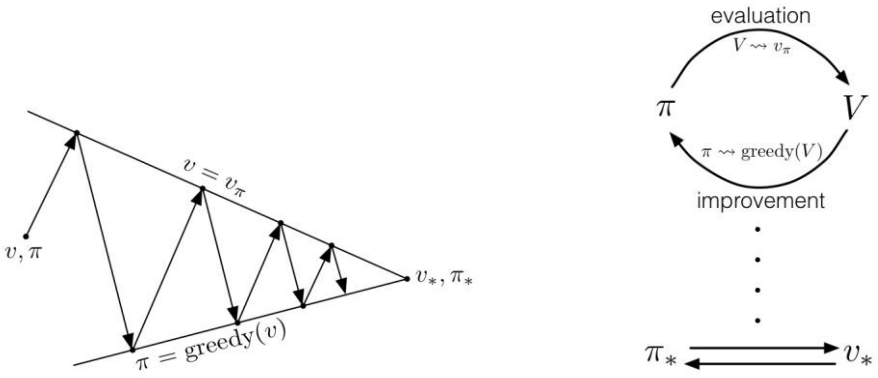


Figure 1: Illustration of convergence of Policy through Contraction Mapping

From Bandit Problem to full reinforcement Learning Problem

The most important feature of Reinforcement learning is that it uses training information that evaluates the action rather than providing instructs. Bandit problem is the *non-associative* setting, which avoids the complexity of full reinforcement learning problem, which is *associative*, that is, actions are taken in more than one situation. In other words, full RL is a multi-arm-multi-bandit problem, where optimal policy suggests pulling a specific arm, leading to jump to a specific bandit and pulling the suggested arm again. We repeat the process until we reach the bandit with has the arm to be pulled, winning us a jackpot, that is, the goal state.

Path Planning and collision checking

Collision checking is one of the major concerns, let it be in autonomous vehicles or robotics in that sense. Search algorithms such as Dijkstra and A* find an optimal solution in a connectivity graph, whereas D* and AD* are tailored to dynamic graphs [9]. The use of graph search methods involves discretization of the workspace and their performance degrades in high dimensions.

This paper introduces a novel approach, by using MBRL for optimal path planning in Dynamic Environments for the implementation of the same. The idea of Adaptive Dimensionality helps in reducing the search space by increasing the dimension of time in the areas where a collision is highly probable to occur.

2 Related Work

Although RL is not much explored for path planning, there are many attempts for motion planning in constraint environments with dynamic obstacles. Traditionally there are two approaches for path planning: Off-line planning, which assumes perfectly known and stable environment, and on-line planning, which focuses on dealing with uncertainties when the robot traverses the environment. On-line planning is also referred to by many researchers as the navigation problem. Additional difficulties in approaching navigation problem is that some environments are dynamic, i.e., the obstacles which are present there, need not be static [10] The Open Motion Planning Library [1] developed by researchers at Rice University, was initially implemented for motion planning and extensions are used for collision checking using standard search algorithms.

Moreover, there is often a problem of increase in the dimension of time in online motion planning when the environment is dynamic. Researchers focused on avoiding the dimension of time but others show that is it unavoidable [3] and rather introduces the concept of Adaptive Dimensionality [2], which focus on a state space that is high dimensional only where the higher dimensionality is absolutely necessary, that is, where collision may occur. PILCO [5] is a data efficient approach which uses limited data to reduce model bias, one of the key problems of model-based reinforcement learning, in a principled way. By learning a probabilistic dynamics model and explicitly incorporating model uncertainty into long-term planning. In this approach, policy gradients are computed analytically for policy improvement.

3 Motivation and Methodology

3.1 Motivation

Re-planning in dynamic environments is a major problem for motion planning engineers. If there is a split-second delay in re-planning, in some situations, results can be fatal. For instance, in medical instruments [4], motion planning for assisting surgeons, or for robotic surgery needs re-planning in case of sudden blood flow. Additionally, collision checking is another major concern. This paper introduces a novel approach, by using MBRL for optimal path planning in Dynamic Environments.

3.2 Problem Description

In a basic implementation, the agent is inside a cell, defined as 'states' of a grid environment, called Grid-world. It needs to reach one of the two-goal states situated diagonally opposite on the upper left and lower right corner, by accumulating maximum reward it can by its travel. The steps of transitions are one of these four: *UP, RIGHT, LEFT* and *DOWN*, Figure 2. Moreover, it depends on the cell, the agent is in that to which goal state it will reach, both being equally rewarding. The agent gets a reward of -1 for every step it makes and ends up getting 0 rewards at goal. The purpose of the agent is to maximize the reward. If it bumps into the wall, then it gets -1 reward and ends up in the same state. My implementation introduces an obstacle space in the environment, where if the agent bangs in, it will get a -100 reward. The agent finds a path avoiding the obstacle creating the policy to traverse the grid and creating a value for each state. The solution exploits concepts of Markov Decision Process using Dynamic Programming.

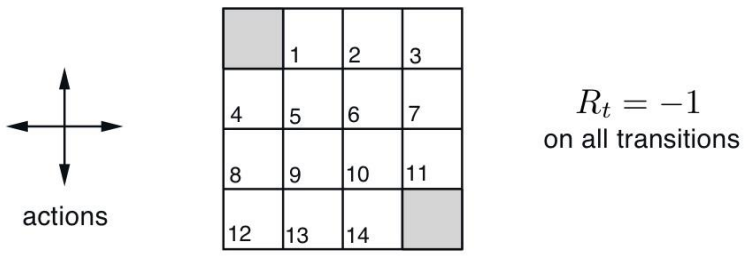
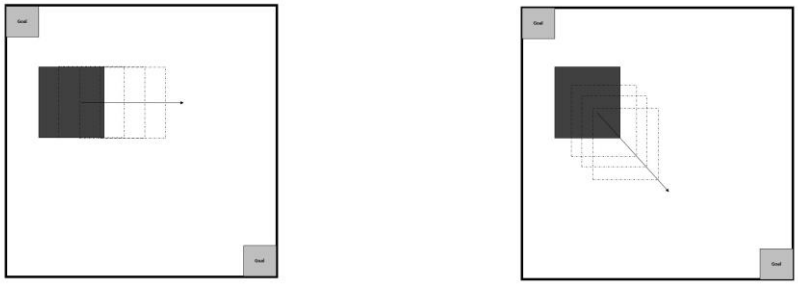


Figure 2: Possible actions, 4x4 Grid representation with goal states and reward for all transitions

3.3 Methodology

This project exploits finite horizon problems in Model-Based Reinforcement Learning to model the problem of path planning. Standard motion planning algorithms like A*, PRM, and others, use Sample Based methods to convert the continuous space into discrete space to find the solution, which is otherwise a challenging task. The sampled states are combined and converted to a grid, called a grid world. With the implementation of the described environment, the test cases involve moving the obstacle space horizontally and diagonally, Figure 3. In traditional problems, policy and values are randomized and due to the certainty of convergence, proved via Contraction Mapping, policy converges to optimality. Therefore, instead of calculating the policy and value at

every step, those calculated in the first step can be fed back as the initial condition for the next iteration. The convergence is reached much quickly in the later case. Starting with the random policy it takes a lot of iterations to converge but due to the less change in policy and value, as only some part of the environment is dynamic, the solution converges fast.



(a) (b)

Figure 3: Movement of 10x7 obstacle space (a) Horizontally, and (b) Diagonally

4 Experimentation and Results

4.1 Experimental Approach

The experiment is done by taking obstacle of size 10x7 in the obstacle space of 30x30 and moving it first horizontally and then diagonally, 12 steps each. The main aim is to check the re-planning time when the obstacle in one frame overlaps with other frame's obstacle space. The approach uses iteration convergence over all the states and getting an update every time until all the states stabilize. This is done by implementing Policy Evaluation of every state, and then improving the policy by Policy Improvement algorithm, the whole process is called policy iteration, Figure 4 [6].(The parameters have their usual meaning as in [6])

Policy iteration (using iterative policy evaluation)

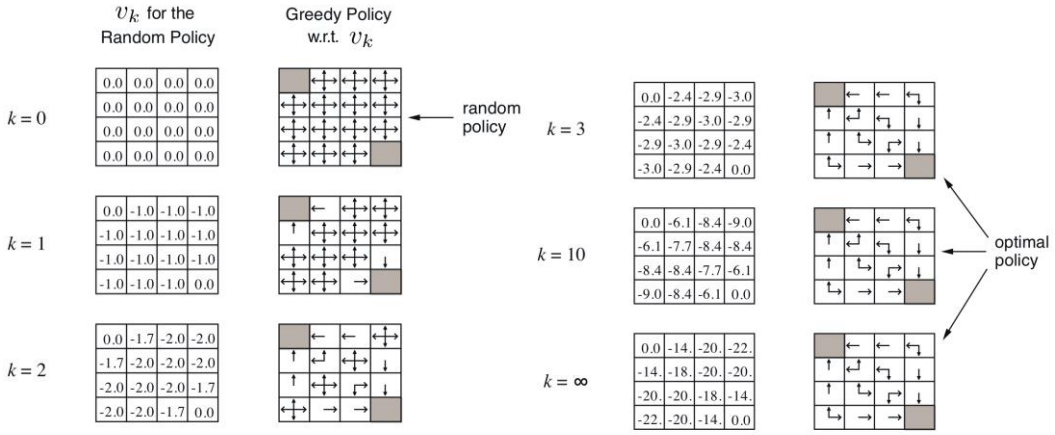
1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\Delta \leftarrow 0$
   For each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\left[r + \gamma V(s')\right]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

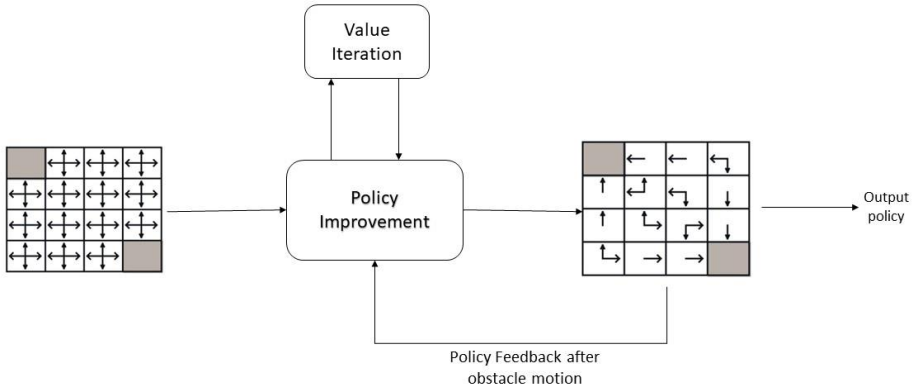
3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Figure 4: Policy Iteration Algorithm for Optimal Policy

Usually, approach start with the random policy as it is meant to converge, but the fact that more policy is close to the optimal policy, faster it converges, is explored here. Moreover, the aim is to get the optimal policy, which is the solution to the RL problem, which in turn is found by using policy iteration, value stabilization is not of much importance. Policy converges much faster than the value. As shown in Figure 5(a) [12], the optimal policy is achieved in the third iteration, but it can take an infinite number of steps to stabilize the value. By analysis, I found that reducing the threshold 'θ' can cause to exit the process of Policy Evaluation much faster, and hence reducing the time of convergence without changing the result, that is Policy. This had a real impact on setting up the system in the initial state. Moreover, it depends on the problem, but the results proved that taking 'θ' carefully can be extremely useful. The approach is illustrated in Figure 5(b).



(a)



(b)

Figure 5: 4x4 Grid showing (a) Policy convergence well before Value, (b) Approach used to exploit the fact in (a).

4.2 Results

The implementation is done on 30x30 grid and the policy from the previous configuration is fed back to the newly changed environment. The results for the same are shown in the table with every iteration corresponding to movement of one step horizontally and diagonally, Figure 6. Moreover, the graphs presented linear scale do not show much variance, so the logarithmic scale is used to show the difference. Moreover, as the implementation is done on 'Spyder' interface by Anaconda using OpenAI gym [11]-[12], taking the average over 100 models are taken to remove noise in

time due to system latency.

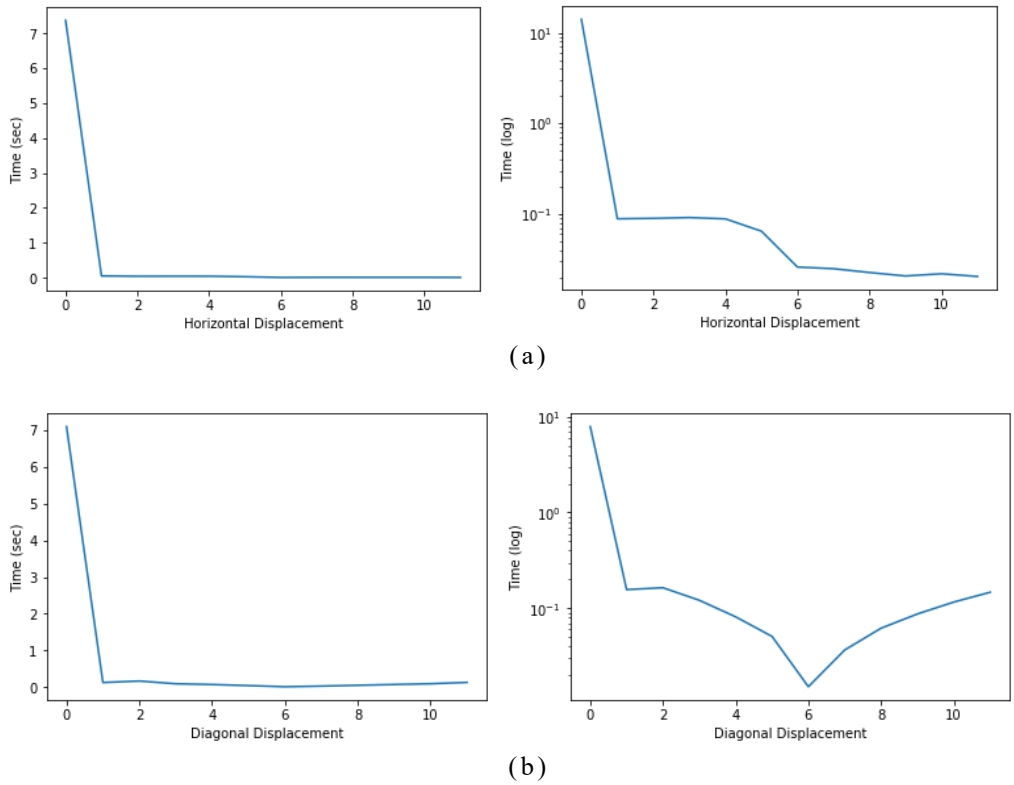


Figure 6: Simulation results in sec and log scale (for a detailed description), respectively averaged over 100 iterations for the obstacle movement (a) Horizontally (b) Diagonally

As it is apparent from the results that the time for re-planning is reduced from few seconds to few milliseconds. This is a good improvement over traditional methods. Moreover, threshold value 'theta' can also be changed without the optimal policy to differ. I used different values for the proof. The results indicate that theta is logarithmically related to the time it takes for the first iteration. Moreover, there is a trade-off between the time of planning for successive iteration and first iteration as seen in Table 1 by the values of theta as 0.01 and 0.001. This suggests that finding optimal theta for a configuration can lead to reducing the planning time. Additionally, there is a comparative increase in re-planning time after the step when there is no overlap during diagonal movement, but not in horizontal movement. This is because the obstacle is moving towards the goal in diagonal movement so re-planning involves modification of values in the main stream than it is otherwise in the horizontal case.

Table 1: Effect of Theta on Initial setup and subsequent iterations for same(optimal) Policy

Theta	*First Iteration*	*Second Iteration*	*Third Iteration*
0.1	7.3694	0.1459	0.1819
0.01	11.8244	0.1399	0.1521
0.001	16.5536	0.1393	0.1510
0.0001	20.9135	0.1621	0.1831
0.00001	25.4749	0.1438	0.1407

The following, Figure 7 illustrates the policy for an optimal path in 18x18 grid, in states with two or more possible best steps, one is chosen at random.

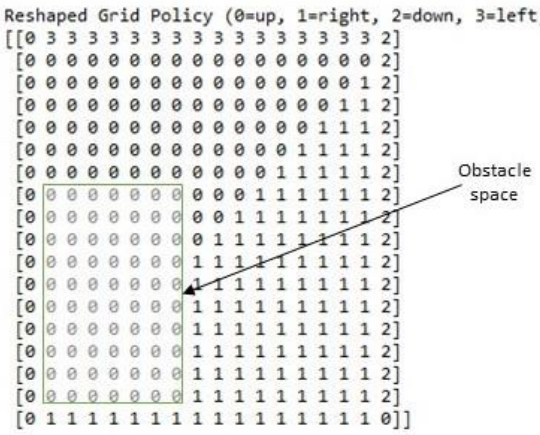


Figure 7: Policy representation using a simulation of an obstacle of 10x7 units in 18x18 grid

5 Conclusion and future Directions

The new approach to path planning is devised in this paper using Reinforcement Learning, which is supposed to be a slow learning process. A grid world of considerable size is implemented, and an obstacle space is introduced. The obstacle space moves Horizontally and Diagonally, and results show that re-planning can be highly optimized in time by using close loop on Policy. As a result, the time for re-planning is reduced from a few seconds to a few milliseconds. Moreover, selecting parameters optimally further reduces the time for re-planning but may increase the initial set-up time.

Although the planning is quick the fact that all states are traversed to get the optimal policy can cause issues in high dimensional space. Therefore, for future implementation, the approach of Adaptive Dimensionality can be helpful. Moreover, as in most of the cases obstacle area itself is fixed, though it moves, the area of the obstacle body can be avoided from policy verification by jumping the states corresponding to obstacle size. Alternatively, machine learning can be used to define a confidence factor around the obstacle, in which collision may occur, and therefore the area with high confidence can be explored leaving the remaining by some risk factor. Another approach may involve converting the whole grid into chunks, checking the variance of every chunk in terms of changing states and using eigen vectors to define the most probable chunks those changes state the most and prioritizing the chunks for policy optimality by using already implemented approach. Moreover, using the concepts of Feature Selection, a confidence can be introduced, which can help to avoid updating certain chunks again introducing some risk.

References

[1] Sucan, Ioan A., Mark Moll, and Lydia E. Kavraki. "The open motion planning library." *IEEE Robotics & Automation Magazine* 19, no. 4 (2012): 72-82.

[2] Gochev, Kalin, Benjamin Cohen, Jonathan Butzke, Alla Safonova, and Maxim Likhachev. "Path planning with adaptive dimensionality." In *Fourth annual symposium on combinatorial search*. 2011.

[3] Vemula, Anirudh, Katharina Muelling, and Jean Oh. "Path planning in dynamic environments with adaptive dimensionality." In *Ninth Annual Symposium on Combinatorial Search*. 2016.

[4] Napalkova, Liana, Jerzy W. Rozenblit, George Hwang, Allan J. Hamilton, and Liana Suantak. "An optimal motion planning method for computer-assisted surgical training." *Applied Soft Computing* 24 (2014): 889-899.

[5] Deisenroth, Marc, and Carl E. Rasmussen. "PILCO: A model-based and data-efficient approach to policy search." In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pp. 465-472.

2011.

[6] Sutton, Richard S., and Andrew G. Barto. *Reinforcement learning: An introduction*. Vol. 1, no. 1. Cambridge: MIT press, 1998.

[7] Szepesvári, Csaba. "Algorithms for reinforcement learning." *Synthesis lectures on artificial intelligence and machine learning* 4, no. 1 (2010): 1-103.

[8] Nilsson, Nils J. *Principles of artificial intelligence*. Morgan Kaufmann, 2014.

[9] Kavraki, Lydia. "Sampling-Based Robot Motion Planning."

[10] Smierzchalski, Roman, and Zbigniew Michalewicz. "Path planning in dynamic environments." In *Innovations in Robot Mobility and Control*, pp. 135-153. Springer, Berlin, Heidelberg, 2005.

[11] Brockman, Greg, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. "Openai gym." *arXiv preprint arXiv:1606.01540* (2016).

[12] Analytics, Continuum. "Download Anaconda Python Distribution." (2015).

[13] Udacity course on Reinforcement Learning – (https://classroom.udacity.com/courses/ud600)

[14] David Silver. "COMPM050/COMPGI13 - *Reinforcement Learning"* lecture series, (http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html)