

Online Policy Training vs Heuristic Search

Using Reinforcement Learning to Avoid Dynamic Obstacles

Collin Crowell*, Tianyi Gu*, Mostafa Hussein*, Yishan Luo*, Sangeeta Patnaik*, Yuncong Zhou*,

Abstract—This paper focus on finding better methods to solve the problem of dynamic obstacles avoidance for mobile robots. We studied two deterministic approaches which use heuristic search techniques, and four stochastic approaches which use reinforcement learning techniques. We proved these two approaches are mathematically different. The experiment results show that deterministic approaches work better for this problem. They are not only faster but also robuster than stochastic approaches. But stochastic approaches still applicable for certain problem scenarios.

I. INTRODUCTION

The ability to avoid dynamic obstacle is very crucial for mobile robots to operate in real world. Once the robot gathers the sensor information, it needs a planner to come up with an executable plan, say a sequence of actions, that navigate the robot to the goal position without intersecting with any obstacles. Figure 1 gives a brief description of the problem. There are two ways to solve this problem. One is to predict the movement of those dynamic obstacles so that the problem becomes a stationary problem, then apply deterministic algorithms to solve it. We call this framework deterministic approach. The second approach is to solve the stochastic problem directly by applying reinforcement learning algorithms. We call it stochastic approach.

In this paper, we focus on studying the planning algorithms and assume fully observability of the robot. We compared deterministic approaches and stochastic approaches. More specifically, we implement two search algorithms for the deterministic approach and four reinforcement algorithms for the stochastic approach. We developed a simulation framework to mimic the real-time planning scenario to support the experiments. The experiment results show that the deterministic approach is not only faster but also robuster than the stochastic approach. We also proved that the deterministic approach is mathematically different from the stochastic approach.

In the following, Section 2 reviews the widely used deterministic approaches and stochastic approaches. Section 3 presents the six implemented algorithms. Section 4 presents the simulation framework and experiments that compare all those algorithms. Section 5 discuss the similarity and difference between the deterministic approach and the stochastic approach. We conclude with a summary and bring up some future work in Section 6.

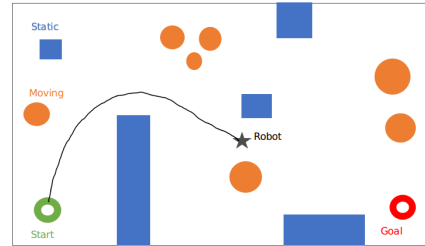


Fig. 1: We focus on the problem of navigating the robot (star in the middle) from the start position (green ring) toward the goal position (red ring) without intersect any static obstacles (blue squares) and dynamic obstacles (orange circles).

II. RELATED WORK

A. Deterministic Approach

Several deterministic approaches have been implemented to solve similar obstacle avoidance problems. Work has been done in the planning space to safely navigate around dynamic obstacles that move with uncertain motion patterns. Planning a safe trajectory with dynamic obstacles requires accurate prediction of the obstacles location and future behavior. An algorithm was created to solve this problem by building a learned motion pattern model by combining a sample based reachability computation and Gaussian processes. Simulations demonstrated that this planner could safely navigate an autonomous vehicle around a complex environment in real-time while mitigating the chance of a collision with the dynamic obstacles [1].

Several other planning algorithms have been created that can efficiently plan and re-plan trajectories dynamically by modeling the dynamic obstacle uncertainty [16][21]. A different planning approach tries to tackle the challenge of cluttered and highly dynamic environmental surroundings. There are two major challenges under these conditions: trying to predict the trajectories of the dynamic obstacles is noisy and planning is computationally expensive because of the large state space. Also, re-planning needs to occur frequently since the trajectories of the dynamic obstacles change. These problems are addressed with a path planning algorithm that models the dynamic obstacles predicted trajectories and the uncertainty of the predictions. A time-bounded lattice is used which combines short-term planning with time with long term planning without time[12].

Another approach in planning with dynamic obstacles is using a sampling-based motion planning algorithm like *RRT* [15] specifically developed for large robotic vehicles

*Equal contribution

with complex dynamics and operating in uncertain, dynamic environments like URBAN challenge [20] [13].

Finally, we can use any real-time algorithm that uses incremental planning if we could solve the problem of dynamic obstacles prediction like [9][11].

B. Stochastic Approach

One widely used model-free reinforcement learning approach is *Sarsa* [19]. The idea is to update the Q value by the rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(S', a') - Q(S, a)]$$

Algorithm 1 Sarsa

```

1: initialize  $Q(s, a), \forall_s \in S, a \in A(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$ 
2: for each episode do
3:   Initialize  $S$ 
4:   Choose  $A$  from  $S$  using policy derived from  $Q$ 
5:   for each step of episode do
6:     Take action  $A$ , observe  $R, S'$ 
7:     Choose  $A'$  from  $S'$  using policy derived from  $Q$ 
8:      $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma * Q(S', A') - Q(S, A)]$ 
9:      $S \leftarrow S'; A \leftarrow A'$ 
10:  until  $S$  terminal

```

Algorithm 10 is the pseudo code of *SARSA*, it always updates the Q value by the $Q(s, a)$, which is derived by the current learned policy. Another model-free reinforcement learning approach is Q -learning [22]. The idea is to update the Q value by the rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{a'} Q(S', a'))$$

Algorithm 2 Q-Learning

```

1: initialize  $Q(s, a), \forall_s \in S, a \in A(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$ 
2: for each episode do
3:   Initialize  $S$ 
4:   for each step of episode do
5:     Choose  $A$  from  $S$  using policy derived from  $Q$ 
6:     Take action  $A$ , observe  $R, S'$ 
7:      $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_{a'} Q(S', a) - Q(S, A)]$ 
8:      $S \leftarrow S'$ 
9:  until  $S$  terminal

```

Algorithm 9 is the pseudo code of Q -learning, it always update the Q value by the $\max_a Q(s, a)$.

Model-based approaches such as LSTD [5] and LSPI [14] are also widely used as reinforcement learning approaches. These two approaches efficiently use sample experiences to learn state value function and state-action value function

respectively. Both of them implicitly learn the embed MDP model. There are also other approaches that learn the MDP model explicitly. iPOMDP[7] tries to learn a POMDP model which is much harder than learn a MDP. The way they achieve this problem is to assume some prior distribution over the model and do inference based on observation. Once a MDP model is learned, the problem of solving the reinforcement learning problem transfers into solving the MDP. So we can apply any MDP solver such as dynamic programming approaches: value iteration (VI)[3], policy iteration (PI)[18], modified policy iteration (MPI)[17] or sampling-based approaches: real-time dynamic programming (RTDP)[2] and its improved version labeled-RTDP (LRTDP)[4], Monte Carlo Tree Search (MCTS)[6] and its variation Upper Confidence Bound for Trees (UCT)[10]. In our work, we solved MDP with UCT algorithm considering its remarkable ability of balancing between exploitation and exploration, which is naturally adapt to real-time on-line search scenario.

III. METHOD

A. Deterministic Approach

Our problem here is how to plan under uncertainty of the movement of the dynamic obstacles we have two different approaches based on A^* algorithm [8].

1) A^* algorithm: A^* makes use of two values $g(n)$ as the distance from the start node to the intermediate node we try to reach and $h(n)$ as the heuristic part that represent the distance from the intermediate node to the final goal then the total cost $f(n)$ of the final path we trying to find is simply the sum of the two distances.

$$F(n) = g(n) + h(n) \quad (1)$$

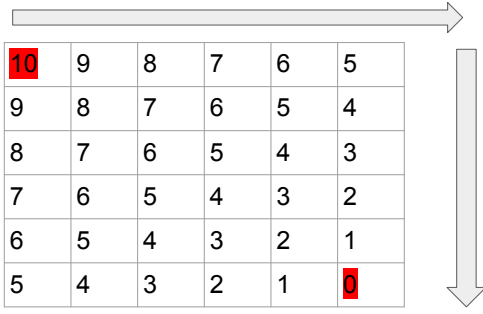
Algorithm 3 A^* Algorithm

```

1: initialize the open list
2: initialize the closed list
3: put the starting node on the open list
4: while open list not empty do
5:   node  $X$  = the node with the least  $f$  on the open list
6:   pop  $X$  off the open list
7:   generate  $X$  children and set their parents to  $x$   $\triangleright$  we don't diagonal movement
8:   for each child do
9:     if child is the goal then
10:      top the search
11:      child.g =  $X.g + 1$   $\triangleright$  for simple grid world
12:      child.h = Manhattan distance from goal to child
13:      child.f = child.g + child.h
14:      add the node to the open list
15:   push  $X$  on the closed list
16: return the path

```

A^* algorithm is one of the oldest and powerful algorithms that work with a guaranty to find the optimal path to the



10	9	8	7	6	5
9	8	7	6	5	4
8	7	6	5	4	3
7	6	5	4	3	2
6	5	4	3	2	1
5	4	3	2	1	0

(a) heuristic Matrix without obstacles

10	9	8	7	6	5
9	8	7	6	5	4
8	7	1000	5	1000	3
7	6	5	4	3	2
6	1000	4	1000	2	1
5	4	3	2	1	0

(b) heuristic Matrix with static obstacles

Fig. 2: Heuristic Matrix

goal but, if we want to use it as real-time motion planner we need to have some small modifications on it.

The first approach that we used is simply run A^* at each time step and select the first action to commit to the agent this way will guaranty that we have the optimal path but it will take a lot of time to compute and then use the first action simply it's waste of time but it will be useful if you must have the optimal path which usually is not required.

The second Method is a simplified approach from $LSS - LRTA^*$ [11] which is using the original A algorithm and run it only for three iterations and select the best action to commit to the agent, this approach is going to be much faster because we only compute a small part of the path but at the final we will not have the optimal path.

2) *Dealing with static obstacles*: The simplest and effective way to deal with the static obstacles is to make it unreachable by assigning the $H(n)$ for the static obstacles position with 1000. For example, if we have 6*6 map with the start at the (0,0) position at the upper left corner and the goal at position (5,5) at the bottom right the original heuristic will look like Fig 2a if we used Manhattan distance. If we have 4 static obstacles at positions (2,2), (2,4), (4,1), (4,3) the final heuristic matrix will look like Fig 2b.

3) *Dealing with Dynamic obstacles*: That is the most important part for us if we manage with some approaches to have a potability matrix that states the probability of a dynamic obstacle D are going to be in position (x,y) , we can use this matrix with the help of Equation 2 for each dynamic obstacle to update the heuristic matrix then we are going to explain how we can get this probability matrix.

10	9	8	7	6	5
9	8	7	6	5	4
8	7	6	5	4	3
7	6	5	4	3	2
6	5	4	3	2	1
5	4	3	2	1	0

(a) heuristic Matrix without obstacles

0	.2	0	0	0	0
0	.2	0	0	.2	.2
0	0	0	0	0	0
0	0	0	0	0	0
0	0	.2	0	0	0
0	0	0	0	0	0

(b) probability matrix of dynamic obstacles position in the next movement

10	19	8	7	6	5
9	18	7	6	15	14
8	7	6	5	4	3
7	6	5	4	3	2
6	5	14	3	2	1
5	4	3	2	1	0

(c) heuristic Matrix with dynamic obstacles and $\alpha = 50$

Fig. 3: Heuristic Matrix updating with dynamic obstacles probability

$$h(x,y) = originalh(x,y) + \sum_{i=1}^N prob(D_i) * \alpha \quad (2)$$

where N is the number of dynamic obstacles and α are the number that used as a trade of time of computation and the chance of collision with an obstacle. In Fig 3 we can see how to update the heuristic matrix based on the dynamic obstacles probability.

4) *Dynamic obstacles movement prediction*: To simplify the complex environment, the movements of the dynamic obstacles were constrained to a known model. This section will describe our model of the dynamic obstacles and explain the movement predictions. The dynamic obstacles are limited to five actions; move up, move down, move left, move right and stay put. At each time step the obstacles can take a

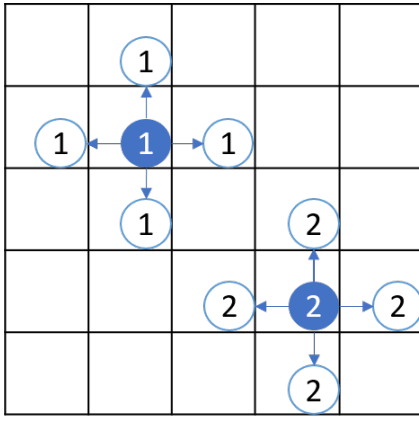


Fig. 4: Dynamic Obstacles movement.

single action. Actions are chosen randomly with a normal distribution and dynamic obstacles move independently from each other. If an action is chosen that would cause a wall collision, the obstacle does not move as if the stay put action were taken. Figure 4 shows a grid with two dynamic obstacles and their possible movements after one-time step.

Since the actions and the distribution are known, we can calculate and predict the obstacles movement by creating a probability matrix. The probability matrix is used in the deterministic approach to avoid the dynamic obstacles in the grid world. The predicted movement of the obstacles can be computed for a variable number of time steps. As the number of time steps increase, the dynamic obstacle can move further and further away from its starting position, increasing the number of possible locations. The probability that the dynamic obstacle is in or near its starting position is much higher than the probability of the obstacle having moved farther away. We calculate the probability of an obstacle being in a grid cell by summing the opportunities that the obstacle had to move to that cell after each time step. The counts in each cell are divided by the total number of possible moves to compute a probability.

B. Stochastic Approach

For the stochastic approach, we first model the problem in Markov decision process (MDP). A MDP model is an agent acting in a stochastic environment. The MDP model can be specified as a tuple $\langle S, A, T, R \rangle$, where S is a set of states, A is a set of agent actions. When agent take action $a \in A$ in state $s \in S$, it moves to a new state $s' \in S$ with probability $T(s, a, s')$ and receives a reward $R(s, a)$. We discretized the continuous state space into a grid-world representation, then each state here represent the agent position and obstacles' position. The actions in our work are "up", "down", "left", "right", and "stop". We provide a goal-reaching reward 1000 upon task completion and a collision penalty -1000. To encourage shorter trajectories, we add a small time penalty -1 as immediate reward. Because we don't have the model for the uncertainty movements of dynamic obstacles, the transition model is unknown initially.



Fig. 5: Our off-line framework separate the training phrase and the running phrase. In the training phrase (left), we run the RL algorithm for a large number of episodes (20,000 in our experiment) in the hope that it can learn a decent policy (middle look-up table). Then in the real-time running phrase (right), we commit the action that follow the policy.

$$f_1 = x + 10y$$

$$f_2 = \frac{x_{d_1} + 10x_{d_1} + 100x_{d_2} + 1000x_{d_3} + \dots + 10^n x_{d_n}}{y_{d_1} + 10y_{d_1} + 100y_{d_2} + 1000y_{d_3} + \dots + 10^n y_{d_n}}$$

$$f_3 = 10^a$$

Fig. 6: Value Implementation function of Sarsa-learning.

To address this problem, we study the model-free approaches and model-based approaches.

1) *Model-free Approach*: We design an off-line reinforcement learning framework to apply two most widely used RL algorithms: *Q-learning* and *SARSA*. Figure 5 shows the off-line framework. To apply *Q-learning*, we follow the pseudopod in 9. To apply *SARSA*, we follow the pseudopod in 10. *Q-learning* is off-policy algorithm and *SARSA* is on-policy algorithm. They perform the updates of action-value function estimate at the end of each step without waiting for the terminal. The challenge of bring these algorithm to on-line scenario is that there were infinite number of states and the visiting each state and calculate Q-value in that limited time is difficult.

To address this problem, we implemented value approximation which is similar to LSPI which is a reinforcement learning algorithm designed to solve control problems. It is uses value function approximation to cope up with large state spaces and batch processing for efficient use of training data. LSPI has been used successfully to solve several large-scale problems using relatively few training. In this implementation value approximation is the method of using the Q-value for set of seen state-action pair and manipulating it to get an appropriate equation which can generate Q-value for unseen data.

The value approximation for Q-value was implemented with the help of Features and Weights as described in Figure 6. Features is structured information about the input parameters of the environment. For the stochastic approach we have considered basically three broad category of input parameter. The first feature is about the information about the agents position. The second feature is about the position of all dynamic obstacles in the grid respectively. The third feature is about the action taken.

One very important factor that was taken care while

designing the feature were that every feature should be unique with respect to the position in the grid. For example, if say

$$f_1 = x + y$$

For coordinates (3,4) and (4,3) the value of the feature would be same. This would be a fault in the feature design. So, all the features were designed in such a way that for every position in the grid it generated unique value.

Weights are the value that are generated with the help of the seen dataset. Initially, we came up with some random weights, these weights were substituted in the value approximation equation. The Q-value was calculated. The calculated Q-value was compared with the actual Q-value and the Error and Arrow was calculated.

Once the Error or Arrow was calculated, the prior was used to adjust the weights. The weights were adjusted with the help of the below mentioned mathematical equation.

$$\begin{aligned}\theta_0 &\leftarrow \theta_0 - \alpha(y' - y) \\ \theta_1 &\leftarrow \theta_1 - \alpha(y' - y)x_1 \\ \theta_2 &\leftarrow \theta_2 - \alpha(y' - y)x_2\end{aligned}$$

This was experimented for a set of all the seen Q-values and correct weights were calculated. The value approximation equation implemented to get Q-value of unseen data is

$$Q = w_1 * f_1 + w_2 * f_2 + w_3 * f_3 + w_4$$

where w_1, w_2, w_3, w_4 are the weights and f_1, f_2, f_3 are the features. The pseudocode of value approximation is in Algorithm 22

Algorithm 4 Value Approximation Approach Algorithm

```

 $f_1 = x + 10y$ 
 $f_2 = \frac{xd_1 + 10xd_1 + 100xd_2 + 1000xd_3 + \dots + 10^nx_d_n}{yd_1 + 10yd_1 + 100yd_2 + 1000yd_3 + \dots + 10^ny_d_n}$ 
 $f_3 = 10^a$ 

```

procedure GET WEIGHT (s, a)

```

Weights[]  $W = w_1, w_2, w_3, w_4$ 
 $q = w_1 * f_1 + w_2 * f_2 + w_3 * f_3 + w_4$ 
for  $q^* \in Set(q)$  do
     $\delta q = q^* - q$ 
     $w_1^* = w_1 - \delta q f_1$ 
     $w_2^* = w_2 - \delta q f_2$ 
     $w_3^* = w_3 - \delta q f_3$ 
     $w_4^* = w_4 - \delta q f_4$ 
     $w_1 = (w_1 + w_1^*)/2$ 
     $w_2 = (w_2 + w_2^*)/2$ 
     $w_3 = (w_3 + w_3^*)/2$ 
     $w_4 = (w_4 + w_4^*)/2$ 

```

return W

procedure GET QVALUE UNSEEN (s, a)

```

Weights[]  $W = NULL$ 
 $W = \text{GET WEIGHT}(s, a)$ 
 $Q = w_1 * f_1 + w_2 * f_2 + w_3 * f_3 + w_4$ 
return  $Q$ 

```

2) *Model-based Approach*: Another way to deal with the uncertainty is directly estimate the transition model, then apply MDP solver to update the policy until time limit. To estimate the model uncertainty, we derive the transition probability from the movement prediction of obstacles. The probability matrices of each dynamic obstacle in the environment can be used to compute the transition probability matrix. The transition probability matrix is used in the model-based stochastic approach to avoid the dynamic obstacles based on the probabilities that the environment transitioned to each possible state. The transition probabilities are computed recursively based on the number of dynamic obstacles in the grid world. The probabilities are computed by multiplying all the probabilities of each dynamic obstacle at each state.

Algorithm 5 Monte Carlo tree search Algorithm

procedure SELECT ACTION (s, d)

loop

 Simulate (s, d, π_0)

return $\text{argmax}_a Q(s, a)$

procedure SELECT ACTION (s, d, π_0)

if $d = 0$ **then**

return 0

if $s \notin T$ **then**

for $a \in A(s)$ **do**

$(V(s, a), Q(s, a)) \leftarrow (N_0(s, a), Q_0(s, a))$

$T = T \cup s$

return Rollout (s, d, π_0)

$a \leftarrow \text{argmax}_a Q(s, a) + c\sqrt{\frac{\log N(s)}{N(s, a)}}$

$(s', a) \sim G(s, a)$

$q \leftarrow r + \gamma \text{SIMULATE}(s', d - 1, \pi_0)$

$N(s, a) \leftarrow N(s, a) + 1$

$Q(s, a) \leftarrow Q(s, a) + \frac{q - Q(s, a)}{N(s, a)}$

return q

Algorithm 6 Rollout evaluation Algorithm

procedure ROLLOUT(s, d, π_0)

if $d = 0$ **then**

return 0

$a \sim \pi_0(s)$

$(s', a) \sim G(s, a)$

return $r + \gamma \text{Rollout}(s', d - 1, \pi_0)$

Once we have the transition model, we can any of the MDP solver to compute the policy. In our work, we use UCT algorithm to solve the problem on-line because of its remarkable ability of balancing between exploitation and exploration, which is naturally adapt to real-time on-line search scenario. The pseudocode of UCT is in Algorithm 18 and 6.

IV. EXPERIMENTS

A. Environment setting

We developed a grid world simulator to exam our path planning algorithms. The world has one agent which we can control, one static goal, and multiple random generated static and dynamic obstacles. Typically, we set the total number of obstacles to 10% of grid size. Every object has a unique location in (x,y). Every moving object has 5 actions [stop, up, down, right, left] and we encode them into integer [0-4] for the ease of communication. All the dynamic obstacles perform 2-dimensional random walk in each iteration.

1) *Structure*: We decouple the path planning algorithm and the simulator with a communication API. Simulator act as a server and planning algorithms act as client in our experiment. Simulator send two messages:

- i. Locations of objects. At initialization stage the simulator send locations of all objects. At update stage the simulator only send the locations of agent and all dynamic obstacles. For example in a world with 2 dynamic obstacles, a typical space-separated message is "0 2 3 3 2 4". This string is also used as state identifier in our stochastic algorithms.
- ii. Ending message. "win" when the agent reaches the goal / "lose" when agent crashes into an obstacle.

Planner send two messages:

- i. Starting message. "start" signals the simulator and get initialization message.
- ii. Actions: one action or a sequence of actions, such as "2 3 0".

2) *Simulator*: The real-time simulator we use for testing all our path planning algorithms has the following setting:

- i. The simulation starts with receiving a start message, ends with sending a win/lose message.
- ii. We assume that actions require time to perform in the real world. Whenever the simulator receives an action from planning algorithms, it will send back the current location of all the moving objects.
- iii. Simulator will wait for no more than 0.5 seconds for the next action, once timeout it will send lose.

3) *Trainer*: We also develop a training module for the stochastic algorithms. It is slightly different from the simulator.

- i. The training process will run indefinitely as long as the planner wants to continue. Once it finished a start-end cycle, it will immediately start a new one with the same initialization message.
- ii. The reward of an action is sent back immediately in the training process.

B. Comparison

We run multiple experiments in our simulation framework. Different map size vary from 6 by 6 to 10 by 10 are compared. We compared two deterministic approaches: A* and LSS-LRTA* as well as three stochastic approaches: Q-learning, SARSA and UCT. For each map size

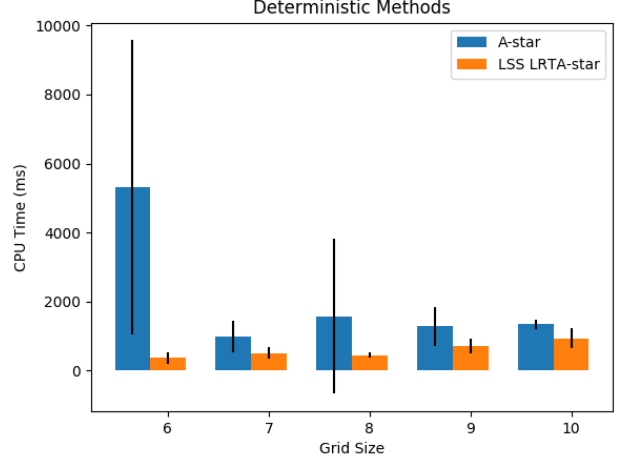


Fig. 7: We show the total computational time for each map size. The bar here is the average over 10 problem with different random seeds. Black line is the standard deviation. LSS-LRTA* is faster than A*. The average total computational time is less than 1000 milliseconds.

and each algorithm, 10 different random seeds are used to generate problems. For the deterministic approach we have one parameter $\alpha = 15$ that's work as a trade off between the speed and the probability of collision with a obstacle. For Q-learning and SARSA, we run 60,000 episode. For UCT, we run 10,000 simulation with 100 step depth, and constant in UCB is set to 1,000. Figure 7 shows the average computational time for deterministic approaches to achieve the goal. The CPU time is accumulate all search iterations. Figure 8 shows the CPU time for stochastic approaches. As we can see, deterministic approaches are much faster than stochastic approaches. This is because we run off-line learning for a very large number of episodes in order to get a decent policy with Q-learning and SARSA. For UCT, although it is on-line method, we still do a large number of simulation in each step in the hope of get better Q value for the state-action.

For the two deterministic approaches, we can see from the figures that LSS-LRTA* is faster than A* that's because we don't compute the complete path we only compute the first three actions, which means the search frontier is always three steps away from the root.

We also show successful rate of all those algorithms in figure 9. As we can see, deterministic approaches also perform better than stochastic approaches. When the map size is larger than 8, there is no success instance for stochastic approaches within 60,000 episodes. So we do extra compute the average steps for different episode numbers in each map size. We observe that as it train longer, the agent survive longer.

For the two deterministic approaches, successes rate for A* are higher than LSS-LRTA*. This is because of the local look ahead for LSS-LRTA* is shorter than A* which compute the complete path to guaranty optimality.

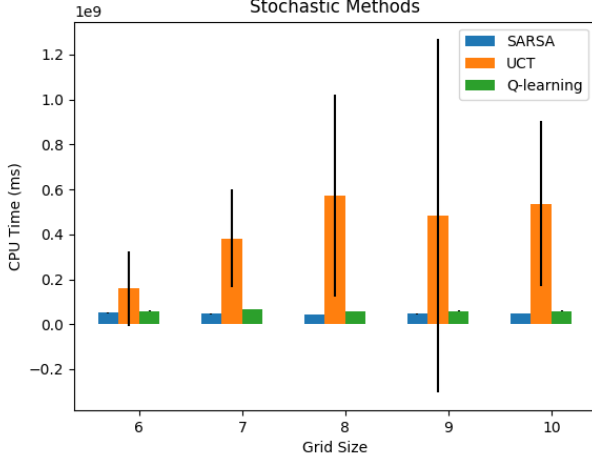


Fig. 8: We show the total computational time for each map size. The bar here is the average over 10 problem with different random seeds. Black line is the standard deviation. SARSA and Q-learning takes less total cpu time than UCT because they only involve off-line training while UCT train a large number of simulation(10,000 here) for every step. Q-learning and SARSA are look close because we use the same training episode number for them (60,000 in this plot).

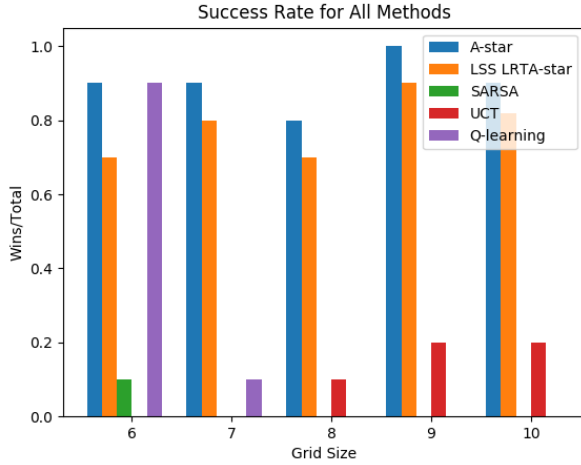


Fig. 9: We show the success rate for each map size. The bar here is the number of win instance over total which is 10. A* performs best. For all map size, its success rate are larger than 0.8. LSS-LRTA* is also not bad, it can achieve high success rate even in map size 9 and 10. For Q-learning and SARSA, they only work in small map size, and SARSA performs even worse, it only win 1 instance in map size 6 by 6. The result of UCT is interesting, it lose all instance in small map size 6 and 7, but start to win in larger map size. We reason this is because as the map size increase, the density of dynamic obstacle is decrease. Thus it increase the chance for on-line approach like UCT to success.

V. DISCUSSION

In this section, we do the theoretically comparison between the deterministic approach and the stochastic approach.

In the deterministic model, the output of the model is fully determined by the parameter values and the initial conditions. But for the stochastic model, the same set of parameter values and initial conditions will lead to an ensemble of different outputs. Thus, in our project, the deterministic method only has one plan while there is a fully policy in the stochastic method. As a result, we found that these two approaches are same in the first expansion, but they are different in the further actions. The example that follows will illustrate this notion.

First of all, the cost of deterministic can be defined as follows

$$TotalCost = \sum_i cost_i \dot{P}_i$$

The reward function of stochastic can be computed with the formula below

$$TotalReward = \sum_{s'} P(s'|s, a) \dot{r}(s')$$

App.A Fig. 10 shows the initial condition and the moving direction of the agent and obstacles. The star is the goal of agent and we have two dynamic obstacles (circle) now. The probability of obstacle moving right is 0.8, and the probability of staying in the same place is 0.2. We are talking about the situation of the first expansion: the agent going up. If there is a collision while moving, the cost or reward is -1000; otherwise it is 0.

As we mentioned above, the deterministic method only has one plan in the first expansion. Both obstacles move to right. The collision happens, and the cost is $0.8 * 0 + 0.8 * -1000 = -800$. The moving direction is shown as App.A Fig. 11.

However, in terms of the stochastic method there are four different transitions. The transition I is obstacles do not move. The reward is $0.2 * 0.2 * 0 = 0$, which means there is no collision. In the transition II, the first obstacle moves to right and the second obstacle stays in the same place. The reward is still 0 calculated by $0.8 * 0.2 * 0$. For the third transition, the second obstacle goes to right and the first obstacle does not move. In this scenario, collision happens and the reward is $0.2 * 0.8 * -1000 = -160$. The last condition is both obstacles move right at the same time. There is a collision as well; the reward is $0.8 * 0.8 * -1000 = -640$. The total reward is $-800 (0 + 0 + -160 + -640)$. App.A Fig. 12 indicates these four transitions respectively.

According to the above analysis results, when the agent is tested in the first expansion the cost of deterministic is equal to the reward of stochastic. Therefore, these two methods are same in the first action. Nonetheless, it does not mean they are always equal. When the agent take the second and more actions, the deterministic approach will use the

Manhattan distance as the heuristic. But in the stochastic approach the reward is accumulated by the subtree through the Bellman equation. App.A Fig. 13 presents the difference. In conclusion, these two methods are different in the second expansion and subsequent actions.

VI. CONCLUSIONS

This paper study deterministic and stochastic approaches to solve the dynamic obstacles avoidance problem for mobile robots. Experiment results show that deterministic approaches work better for this problem. They are not only faster but also robuster than stochastic approaches, and they also scales up much better. However, we argue that off-line stochastic approach still worth to apply in those problem with relatively small state space, and we have considerable off-line training time budget. Then once we finished off-line training, the on-line running would be very efficient because it will involves no computation effort.

In this project, We didn't finish the value approximation. We will keep doing that. There are also several improvement can be done for the simulator such as support on-line object mode update and better support on-line training. The theoretical proof is not sound. We basically just give a general idea in this paper. We could also do more experiments that enable tuning algorithm parameters.

For model the uncertainty of dynamic obstacles movement, in a more realistic environment, the model of the dynamic obstacles would not be known. Dynamic obstacles could move in different directions with unknown probabilities and unknown velocities. Inertia is the resistance for an object to change its state of motion. Using the concept of inertia, we know that obstacles in motion at high velocities with high masses have a higher probability of not changing their direction of movement. This concept could be used to develop a method that can learn from an obstacles history and estimate the obstacles next position. Given a history of the dynamic obstacles positions and the timestamp of each position record, we would like to estimate the probability of the dynamic obstacles future locations.

The Kalman filter is a tried and true method used to estimate a state using a series of observations recorded over time. This method can estimate unknown variables using a history of measurements containing noise and other inaccuracies. The Kalman filter was introduced in 1960 by Rudolf E. Klmn. The algorithm can predict an estimate of the current state variables (position and velocity of the dynamic obstacles) and the uncertainties of the estimates. The subsequent observation can be used to update the current estimation using a weighted average. Estimates with a higher certainty are given more weight. The algorithm works recursively and can be run in real time by using the current observation, calculated state and uncertainty matrix. Using a Kalman filter, we hope that we can accurately predict the motion of the dynamic obstacles with unknown models. This would allow the techniques presented in this paper to be applied to a more realistic environment.

REFERENCES

- [1] Georges S Aoude, Brandon D Luders, Joshua M Joseph, Nicholas Roy, and Jonathan P How. Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns. *Autonomous Robots*, 35(1):51–76, 2013.
- [2] Andrew G Barto, Steven J Bradtke, and Satinder P Singh. Learning to act using real-time dynamic programming. *Artificial intelligence*, 72(1-2):81–138, 1995.
- [3] R Bellman. Dynamic programming. *Princeton University Press Princeton*, 1957.
- [4] Blai Bonet and Hector Geffner. Labeled rtdp: Improving the convergence of real-time dynamic programming. In *ICAPS*, volume 3, pages 12–21, 2003.
- [5] Steven J Bradtke and Andrew G Barto. Linear least-squares algorithms for temporal difference learning. In *Recent Advances in Reinforcement Learning*, pages 33–57. Springer, 1996.
- [6] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.
- [7] Finale Doshi-Velez, David Pfau, Frank Wood, and Nicholas Roy. Bayesian nonparametric methods for partially-observable reinforcement learning. *IEEE transactions on pattern analysis and machine intelligence*, 37(2):394–407, 2015.
- [8] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.
- [9] Sertac Karaman, Matthew R Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime motion planning using the rrt. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1478–1483. IEEE, 2011.
- [10] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *ECML*, volume 6, pages 282–293. Springer, 2006.
- [11] Sven Koenig and Xiaoxun Sun. Comparing real-time and incremental heuristic search for real-time situated agents. *Autonomous Agents and Multi-Agent Systems*, 18(3):313–341, 2009.
- [12] Aleksandr Kushleyev and Maxim Likhachev. Time-bounded lattice for efficient planning in dynamic environments. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 1662–1668. IEEE, 2009.
- [13] Yoshiaki Kuwata, Justin Teo, Gaston Fiore, Sertac Karaman, Emilio Frazzoli, and Jonathan P How. Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on Control Systems Technology*, 17(5):1105–1118, 2009.
- [14] Michail G Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of machine learning research*, 4(Dec):1107–1149, 2003.
- [15] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.
- [16] Jun Miura, Hiroshi Uozumi, and Yoshiaki Shirai. Mobile robot motion planning considering the motion uncertainty of moving obstacles. In *Systems, Man, and Cybernetics, 1999. IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on*, volume 4, pages 692–697. IEEE, 1999.
- [17] M.L Puterman. Markov decision processes: discrete stochastic dynamic programming. *New York: John Wiley & Sons*, 1994.
- [18] Howard R.A. Dynamic programming and markov processes. *MIT Press, Cambridge, Massachusetts*, 1960.
- [19] Gavin A Rummery and Mahesan Niranjana. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering, 1994.
- [20] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.
- [21] Jur P van den Berg and Mark H Overmars. Planning the shortest safe path amidst unpredictably moving obstacles. In *WAFR*, pages 103–118, 2006.
- [22] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

VII. APPENDIX A

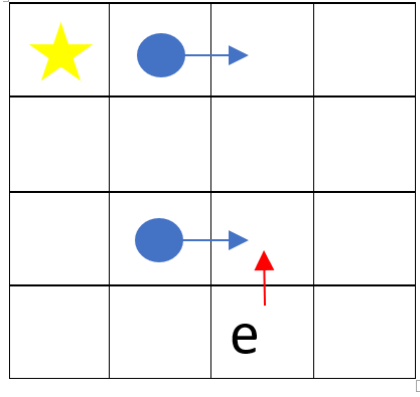


Fig. 10: Initial Condition.

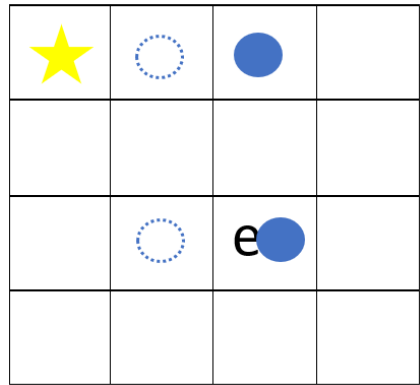


Fig. 11: Moving of Deterministic.

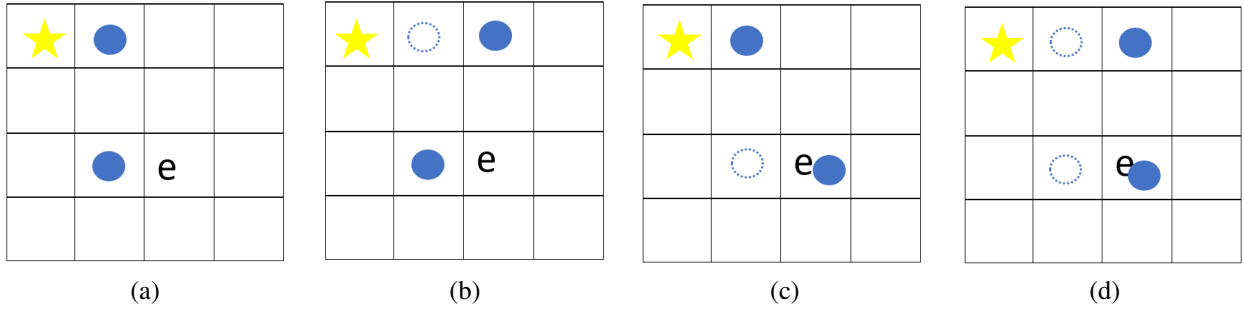


Fig. 12: Four state transitions of action moving up with (a) $0.2*0.2=0.04$, (b) $0.2*0.8=0.16$, (c) $0.8*0.2=0.16$, (d) $0.8*0.8=0.64$

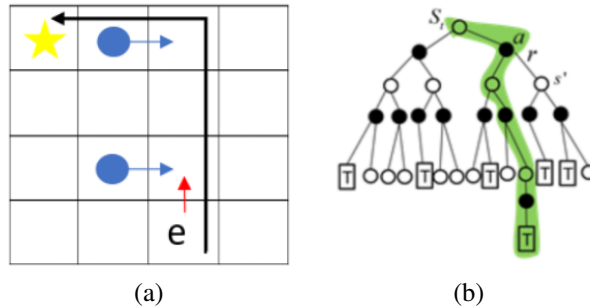


Fig. 13: (a) Manhattan Distance, (b) Subtree