2021 International Conference on Unmanned Aircraft Systems (ICUAS)
Athens, Greece. June 15-18, 2021

Simultaneous Learning and Planning using Rapidly Exploring Random Tree* and Reinforcement Learning

Arup Kumar Sadhu, Shubham Shukla, Sarvesh Sortee, Mohit Ludhiyani and Ranjan Dasgupta

Abstract— **The paper proposes an approach to learn and plan simultaneously in a partially known environment. The proposed framework exploits the Voronoi bias property of Rapidly exploring Random Tree* (RRT*), which balances the exploration-exploitation in Reinforcement Learning (RL). RL is employed to learn policy (sequence of actions), while RRT* is planning simultaneously. Once policy is learned for a fixed start and goal, repeated planing for identical start and goal can be avoided. In case of any environmental uncertainties, RL dynamically adapts the learned policy with the help of RRT*. Apparently, both learning and planning complement each other to handle environmental uncertainties dynamically in real-time and online. Interestingly, more the proposed algorithm runs, its efficiency increases in terms of planning time and uncertainty handling capability over the contender algorithm (i.e., RRT*). Simulation results are shown to demonstrate the efficacy of the proposed approach.**

I. INTRODUCTION

Planning is the sequence of optimal actions aiming to complete a task [1]. Learning refers to the process of acquiring new knowledge through experience aiming to solve a task optimally [2]. A few well-cited planning literature includes graph search algorithm (e.g., Dijkstra's algorithm [3], A* [4], D* [5]), free-space based path planning [6], [7], sampling-based planning algorithm (e.g., Probabilistic roadmap (PRM) [8], Rapidly exploring random trees (RRT) [1], RRT* [9]).

All the graph search algorithms [3], [4] and [5] offer a deterministic path. However, performance of A* and D* mainly depend on the designed heuristic function. The D* algorithm is same as A*, except it can be applied in dynamic environment. But D* has a very high storage requirement compared to A*. On the other hand, Dijkstra's algorithm has a poor time-complexity. Free-space based path planning [6], [7] algorithms are able to provide deterministic path in real-time with very less storage requirement. However, the free-space based path planning is not suitable for planning in a high dimensional configuration space. To plan in the high dimensional configuration space, sampling-based planning algorithms [1], [8] [9] are used. A collision-free road map is created in PRM [1] using sampling approach, which is a popular multiple-query planning algorithm. Now, to obtain a path between a given start and goal, the road-map is used as an input for the graph-search algorithm. However, one-time created road map is unable to handle environmental uncertainties. Unlike PRM, RRT can plan repeatedly amidst environmental uncertainties, which is a single-query planning algorithm. However, in RRT, generated path is random, while RRT* is the optimized version of the RRT in terms of path length. Unfortunately, completeness in all the sampling-based planning algorithms is probabilistic [10], hence, cannot be applied in online, real-time.

On the other hand, few well-known literature on learning-based planning algorithms are [11], [12], [13], [14], and [15]. But all the said learning-based planning algorithms need to learn offline before online planning. A brief literature relevant to the proposed approach are give below.

1) Relevant Literature: Chiang, *et al.* employed reinforcement learning (RL) [2], [16] as a local planner to bias tree-growth towards promising regions in [17]. A hybrid path planner is proposed in [18] for manipulator, where authors employed probabilistic road map to create a road map in the configuration-space. Subsequently, the nodes of the road map are considered as state-action for RL. Finally, a collision-free path is generated amidst dynamic obstacle and environmental uncertainties. Later Aleksandra, *et al.* extended [18] in [19], for long range path planning employing deep RL. However, [17], [18] and [19] involve an offline training (or learning) phase. During this training phase, agent cannot plan. Not only that [18] employed additional methodology to balance exploration-exploitation. In the above circumstances, an approach is proposed to learn and plan simultaneously as given below.

2) Proposed Methodology: Inspired by [18] we propose an approach to learn and plan simultaneously. In the proposed simultaneous leaning and planning algorithm (SLPA), we have employed RL as a learning algorithm and sampling-based planning algorithm as a planning algorithm. Instead of employing traditional Q-learning as RL, here, the backward Q-learning (BQL) [20] is employed for fast learning. On the other hand, RRT* is employed as a sampling-based planning algorithm. RRT* is the optimized version of RRT in terms of path length. The reason of choosing RRT* is that it is Voronoi bias. This property is helpful for the BQL to balance exploration-exploitation. In the beginning of the algorithm, RRT* starts planning for a particular goal. Simultaneously, the BQL starts learning considering vehicle configuration space as state and control signals as actions with respect to the same goal that RRT* is planning for. Once goal is explored by the RRT*, corresponding Q-values are computed employing BQL. These Q-values are now used for planning instead of replanning using RRT* for the same start goal. To handle any environmental uncertainties BQL starts learning using RRT* again, and adapts existing

This work was supported by Tata Consultancy Services (TCS) Research.
[1]The authors are with TCS Research. {arupk.sadhu, shubham.sukla, sarvesh.sortee, mohit.ludhiyani, ranjan.dasgupta}@tcs.com

policy dynamically. These processes continue infinitely. It is apparent that learning and planning are complementing each other for effective real-time online planning. With the best of our knowledge there is no work in the literature, where an agent is learning and planning simultaneously to complement each other. The key benefits of the paper are listed below.

1) Balance exploration-exploitation in BQL with the help of Voronoi bias property of RRT*.
2) Agent can simultaneously learn and plan, vice versa.
3) Environmental uncertainties can be handled by dynamically modifying Q-values.
4) More the algorithm runs, it becomes more robust and can deal with more difficult situations.

Rest of the paper is arranged as follows. Section II provides use case and motivation of the proposed SLPA. A brief preliminaries with some existing limitations are identified in Section III. Proposed formulation for the SLPA is done in Section IV. Algorithm for the proposed SLPA is given in V. Simulation results are provided in Section VI. Finally, the paper is concluded in Section VII with future direction.

II. USE CASE AND MOTIVATION

Apart from [18] the motivation of the proposed SLPA is described here in the perspective of an use case. Suppose, an agent is planning to move from a start s to goal g in a partially known environment (e.g., office, power plat, warehouse, shop floor, etc.) as shown in Fig. 1. Fig. 1 shows a typical warehouse environment, and corresponding floor plan is shown in the top right corner of Fig. 1. It is apparent from Fig. 1 that the floor plan does not include all the objects and another fellow agent as shown in the actual map. It is also unexpected that at the beginning of the planning, agent has the global map information. One best-case scenario is that the planner has a floor plan of the warehouse environment. Hence, based on the available floor plan agent can globally plan a path between start (s) and goal (g) as shown inside floor plan in Fig. 1. However, in reality, the floor plan based global path needs frequent modifications based on the unknown objects and another agent as shown in Fig. 1. In Fig. 1, the black line is showing a path based on the floor plan, while the executed actual path is shown in white. The path marked in white in Fig. 1 cannot be followed always, because the warehouse environment changes time to time. In case of warehouse like scenario, mostly the start (s) and goal (g) are fixed. However, any planning algorithm (e.g., sampling-based, A*, D* etc.) needs to re-plan again and again because of environmental uncertainties, which indeed is time costly. This definitely has a negative impact on the other dependent processes like scheduling, bin packing, assembling etc. Finally, there will be antagonistic effect on the business outcome. Here, warehouse scenario is considered as an example. The proposed SLPA is equally applicable in any industrial, domestic and office like environments. Details of SLPA are discussed below.

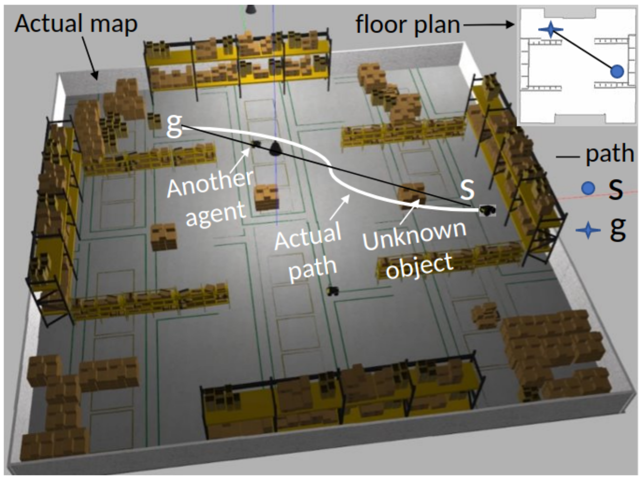
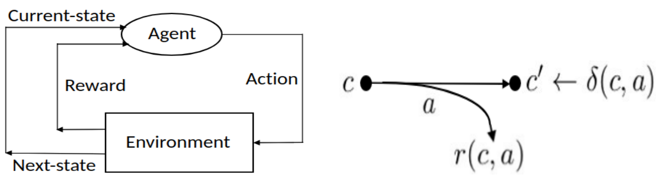


Fig. 1: Actual map of a warehouse with corresponding floor plan (top right corner). s: start, g: goal



(a) Block diagram (b) State-transition

Fig. 2: Reinforcement learning

III. PRELIMINARIES AND LIMITATIONS

This section describes the preliminaries required for the subsequent sections, and identifies the bottleneck to be addressed here.

A. *Reinforcement Learning and Sampling-based Planning*

Reinforcement learning (RL) [2], [16] is one well-known machine learning paradigm. In RL, an agent senses its environment and acts upon it from current-state. In return, the agent receives a feedback (reward) from the environment, and moves to the next state as shown Fig. 2a. Agent stores the state, action, reward combination for future reference. This process continues until all possible state, action, reward combinations are visited.

Q-learning is a RL paradigm coined by Watkins and Dayan in [11], which follows the Markov property [2]. Let $c \in C$ be the current state of an agent, where C is the set of all states. With deterministic assumption, agent moves to the next state $c' \in C$ by executing action $a \in A$ having state-transition probability $P(c'|(c,a)) = 1$. Here, A refers to the set of all possible actions at c. Mathematically the state-transition given in (1).

$$c' \leftarrow \delta(c, a). \tag{1}$$

Agent receives two rewards in Q-learning. One is immediate reward denoted by $r(c, a)$ just after state-transition as shown in Fig. 2b. The other one is the evaluated optimal future reward computed by $\max_{a' \in A} Q(c', a'), c' \in C$ in the next state c', where $Q(c', a')$ is the Q-value at c' because of executing $a' \in A$. The single agent deterministic Q-learning to adapt Q-value ($Q(c,a)$) at (c,a) is given in (2) [11].

$$Q(c,a) \leftarrow (1-\alpha)Q(c,a)+\alpha[r(c,a)+\gamma \max_{a' \in A} Q(c',a')], \quad (2)$$

where, $0 \leq \gamma < 1$ and $0 \leq \alpha \leq 1$ respectively are discounting factor and learning rate [11].

However, in this paper, we have used BQL [20], which is the combination of Q-learning and Sarsa algorithm [2]. BQL is almost similar to the classical Q-learning. Only difference is that in BQL, agent records past $(c, a, r(c,a), c')$ to update Q-values using the same update rule [20], i.e., (2). This offers faster convergence.

On the other hand, there exist several sampling-based motion planning algorithms in the literature [10]. One well-known of them is RRT [1] having Voronoi bias property [1]. Presently we have employed RRT*, which is the optimized variant of RRT in terms of shortest path. Role of RRT* is to assist the BQL algorithm in selecting $a \in A$ at $c \in C$ as and when required based on environmental uncertainties. In spite of the said advantages of BQL and RRT* following limitations are identified.

B. *Limitation*

Action selection in BQL is done randomly [20]. So, exploration-exploitation is unbalanced in BQL [20]. Another bottleneck of any learning algorithm is that, in all learning algorithm, the learning agent cannot learn and plan simultaneously. This is a hindrance for real-time online implication of any learning algorithm.

On the other hand, sampling-based planning algorithms need to plan repetitively for a given start and goal position with probabilistic completeness [1]. This is the bottleneck for quick real-time online planning, which can be avoided by exploiting the previous experiences.

This can be achieved by exploiting the Voronoi bias property of RRT* in BQL, which balances exploration-exploiting. On the other hand, RRT* can exploit the policy learned by the BQL to avoid repeated planning in RRT*. Hence, learning and planning are complementing each other and vice versa.

IV. PROBLEM FORMULATION

This section attempts to formulate an online real-time algorithm to learn and plan simultaneously in a partially known environment amidst environmental uncertainties. Block diagram for the proposed SLPA is shown in Fig. 3. Fig. 3 is twofold. One fold is responsible for learning and other one is for planning. In Fig. 3 ”Q-tree” stores Q-values with state (c) and action (a) information corresponding to each node, where each node contains three information $(c, a, Q(c,a))$.

Considering empty Q-tree, and starting from the ”Start” block in Fig. 3, at the beginning agent does not have any knowledge and needs to learn the policy. Now, if agent goes for learning only, then until the optimal strategy is found agent cannot plan. This will make the algorithm sluggish. Here, RRT* [1] is employed to generate control signals leading to a path. Simultaneously, the BQL algorithm learns considering the RRT* generated control signals as actions. Once, the RRT* obtains a path to reach the goal a policy is evaluated by BQL algorithm and Q-tree is updated. Apparently, instead of exploring the entire environment to learn a strategy, the control signals generated by RRT* assist the learning algorithm to explore goal state quickly. This quick exploration of the goal state saves significant iteration and/or episode.

Once Q-tree is adapted (Q-tree is not empty) by employing BQL for a particular start-goal, agent evaluates the optimal action at current-state as shown in Fig. 3, and evaluates the next-state. Between the current-state and the next-state, agent checks for obstacle. For obstacle free next-state agent moves to the next-state from current-state. In case, obstacle detected agent enters the learning block as shown in Fig. 3 and updates the Q-tree dynamically. It may also happen that agent finds that the detected obstacle between the current and next state has been removed. In such situations, agent relearns the existing policy (or action) from current-state as shown in Fig. 3 and dynamically updates Q-tree. This simultaneous learning and planning process continues infinitely.

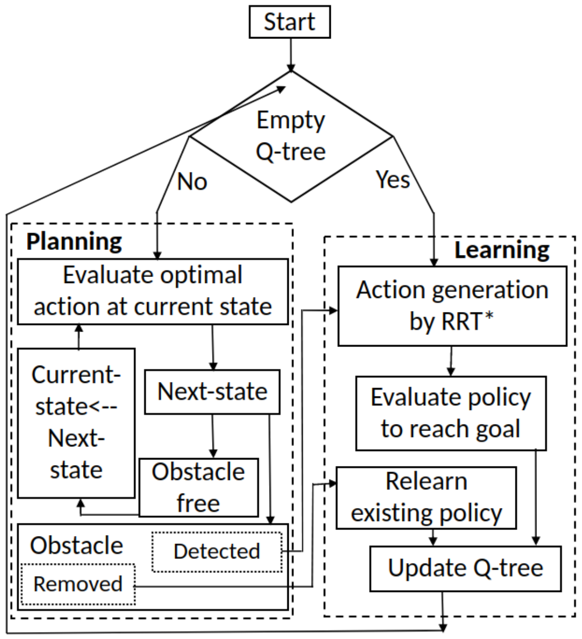

Start
Empty Q-tree
No
Yes
Planning
Evaluate optimal action at current state
Current-state<--Next-state
Next-state
Obstacle free
Obstacle
Detected
Removed
Learning
Action generation by RRT*
Evaluate policy to reach goal
Relearn existing policy
Update Q-tree

Fig. 3: Block diagram for simultaneous learning and planning algorithm

A. *Learning*

In RL, state-action identification and reward shaping are the most important steps. The following sections describe about the state-action identification and reward shaping in RL.

1) State and Action Identification: Here, state is considered as the agent's configuration space, and action is equivalent to the RRT* generated control signals. For example, in a quadcopter, as shown in Fig. 4, state $c = (x, y, z, \phi, \theta, \psi)_I^T$ in inertial frame, where ϕ, θ, ψ respectively are roll, pitch and yaw of the quad (agent). The action $a = (v_x, v_y, v_z, \omega)^T_b$ in body frame; where v_x, v_y, v_z respectively are velocity in x, y, z direction of quad, and ω is the yaw rate.

Fig. 5a shows the expansion of RRT* branches with an aim to explore a path from s to g. The path obtained from the tree is shown in Fig. 5b. There are six intermediate nodes between s and g. Each node is associated with $(c, a, Q(c, a))$, e.g., start node can be written as $(s, a_1, Q(s, a_1))$.

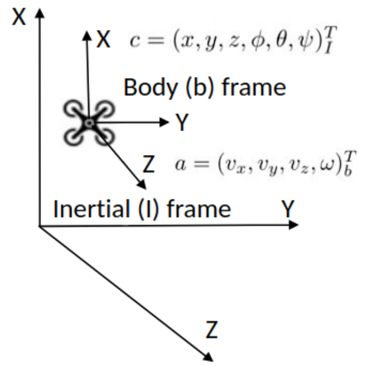

X
X $c = (x, y, z, \phi, \theta, \psi)^T_I$
Body (b) frame
Y
Z $a = (v_x, v_y, v_z, \omega)^T_b$
Inertial (I) frame
Y
Z

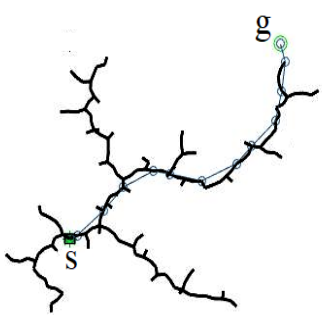
Fig. 4: Quadcopter (ϕ : roll, θ : pitch, ψ : yaw and ω yaw rate)

2) Reward Shaping: The reward shaping is the most crucial in RL, entire learning depends on it. Let $r_{max} \in \mathbb{R}^+$ be the maximum immediate reward agent can have after executing an action a from a state c. We design immediate reward $r(c, a)$ as given in (3) because of executing a at c.

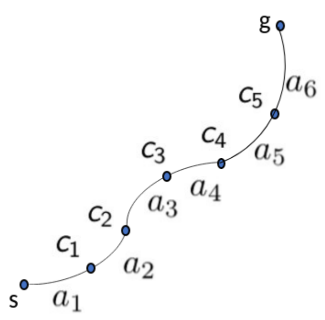
$$\begin{aligned} r(c,a) &= r_{max}, \quad r_{max} \in \mathbb{R}^+, \quad \text{If } c' - g < \epsilon_g, \\ &= 0, \qquad\qquad\qquad\quad \text{If } c' - g \geq \epsilon_g, \\ &= -r_{max}, \qquad\qquad \text{If obstruction detected,} \end{aligned} \tag{3}$$

where $g \in C$ is the goal state, ϵ_g is the goal tolerance and r_{max} is the maximum immediate reward an agent can have because of a state-transition $c' \leftarrow \delta(c, a)$, (Fig. 2b).

In Fig. 5b, $r(c, a)$ is maximum for the goal state-transition $g \leftarrow \delta(c_5, a_6)$, which leads to the goal state g. In case of non-goal state-transition, $r(c, a)$ is set to 0. Any state-transition leading to an obstruction is penalized by setting $r(c, a)$ as $-r_{max}$.



(a) Exploration using RRT*



(b) Learning and planning

Fig. 5: Policy extraction after planning

TABLE I: Q-tree nodes with respect to Fig. 5b and 6

Node	State-transition	Q-value
$(c_5, a_6, Q(c_5, a_6))$	(c_5, a_6)	r_{max}
$(c_4, a_5, Q(c_4, a_5))$	(c_4, a_5)	γr_{max}
$(c_3, a_4, Q(c_3, a_4))$	(c_3, a_4)	$\gamma^2 r_{max}$
$(c_2, a_3, Q(c_2, a_3))$	(c_2, a_3)	$(\gamma^3 - 1)r_{max}$ (with obstacle) $\gamma^3 r_{max}$ (without obstacle)
$(c_1, a_2, Q(c_1, a_2))$	(c_1, a_1)	$\gamma^4 r_{max}$
(Start, a_1, Q(Start, a_1))	(Start, a_1)	$\gamma^5 r_{max}$
$(c_2', a_4', Q(c_2', a_4'))$	(c_2', a_4')	$\gamma^3 r_{max}$
$(c_2, a_3', Q(c_2, a_3'))$	(c_2, a_3')	$\gamma^4 r_{max}$

3) Action Generation And Policy Evaluation: RRT* generates control signals to explore the environment quickly as shown in Fig. 5a. This control signal generation continues until $c' - g < \epsilon_g$, i.e., goal exploration. Once goal explored a path is obtained using RRT* from s to g as shown in Fig. 5b. This path and rewards at each node using (3) are employed as the recorded past data $(c, a, r(c, a), c')$ for BQL to update Q-values at each node.

For example, by BQL using (2) with $\alpha = 1$, $Q(c_5, a_6) = r_{max}, \because Q(g, a') = 0, a' \in A$ (Q-value at goal is undefined). Similarly, by (2) with $\alpha = 1$ using BQL $Q(c_4, a_5) = \gamma r_{max}, \ldots, Q(s, a_1) = \gamma^5 r_{max}$ (Details in Table I). In general, for n intermediate states between s and g one can write,

$$Q(s, a_1) = \gamma^n r_{max}. \tag{4}$$

Now, Q-tree is updated based on the computed Q-values for the path (Fig. 5b) as given in Table I.

4) Policy Adaptation: Based on the detected obstacle, policy learned in Q-tree are dynamically adapted as listed in Table I. Let us consider Fig. 6a, where an obstacle is detected between c_2 and c_3. Now, dynamically update $Q(c_2, a_3)$ by (2) with $\alpha = 1$ as follows,

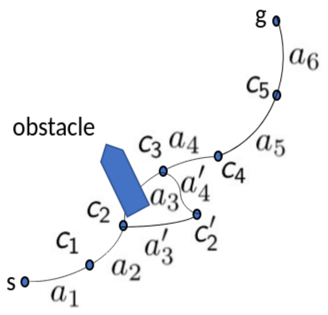
$$\begin{aligned} Q(c_2, a_3) &= r(c_2, a_3) + \gamma Q(c_3, a_4), \\ &= -r_{max} + \gamma * \gamma^2 Q(c_3, a_4), \quad \text{[by (3)]}, \\ &= -r_{max} + \gamma * \gamma^2 r_{max}, [\because Q(c_3, a_4) = \gamma^2 r_{max}], \\ &= (\gamma^3 - 1) r_{max}. \end{aligned} \tag{5}$$

In general, if the obstacle is detected between n^{th} and $(n+1)^{th}$ node, then (5) can be written as,

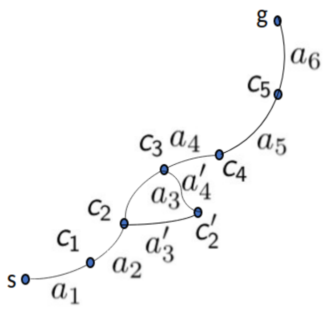
$$Q(c_n, a_{n+1}) = (\gamma^{(n+1)} - 1) r_{max}. \tag{6}$$

In (6), $Q(c_n, a_{n+1}) < 0, \because \gamma < 1$. So, because of the detected obstacle Q-value becomes negative.

It is apparent from Fig. 6a that the path between node c_2 and c_3 is occluded by obstacle. Now, to move from c_2 to g agent needs an alternative path. For that agent simultaneously starts planning by RRT* and learning by BQL at node c_2 in Fig. 6a. In this situation, agent exploits the previously learned policies. For example in Fig. 6a, agent exploits the node c_3 while connecting the alternative path (c_2, c_2', c_3) with the



(a) Adapted policy after obstacle detection



(b) Adapted policy after obstacle removal

Fig. 6: Policy Adaption with environmental uncertainties

existing path. To avoid obstacle the alternative path always connected to a node with higher Q-value, than the node from where alternative path started. For example, in Fig. 6a, both for with and without obstacle $Q(c_3, a_4) > Q(c_2, a_3)$ as tabulated in Table I. Because of the detected obstacle, existing path leading to g in Fig. 6a is amended, hence, the policy too. Therefore, the dynamically appended Q-values in Q-tree are $Q(c_2, a_3') = \gamma^4 r_{max}$ and $Q(c_2', a_3') = \gamma^3 r_{max}$ with respect to Fig. 6a as listed in Table I.

However, in Fig. 6b, the obstacle between node c_2 and c_3 is removed but Q-value at $Q(c_2, a_3)$ is not updated accordingly. Such situation, i.e., removal of detected obstacle, can be easily detected by negative Q-value offered by (6) as shown in Table I. In, such situation, agent needs to relearn the existing strategy by re-initializing the corresponding Q-value to 0. For example, in Fig. 6, $Q(c_2, a_3)$ is relearned using (2), and is shown in Table I.

B. *Q-value Based Planning*

It is quite apparent an agent plans using RRT*, while learning simultaneously using BQL. However, once the agent learned a policy to follow the path, then the agent plans using the learned policy amidst environmental uncertainties.

For Q-value based planning an agent selects an optimal strategy a^*(based on the learned Q-values) at c following

$$a^* = \arg\max_{a\in A}[Q(c,a)], \tag{7}$$

and moves from start to goal. In Fig. 5b, each node has only one action to choose. However, in Fig. 6b, at node c_2 agent has two actions to choose from. By (7) at c_2 in Fig. 6b, agent selects $a^* = a_3$. However, using (7) at c_2 in Fig. 6a, agent selects $a^* = a_3'$. So, agent is dynamically selecting optimal action based on the adapted Q-values because of the environmental uncertainties.

V. Algorithm

The proposed algorithm for SLPA is given in Algorithm 2. Algorithm 2 takes output of Algorithm 1 as input. 2D floor plan is used as an initial map. Based on the floor plan a global path is generated by the agent using Algorithm 1 at the beginning.

Algorithm 1: Learning

```
Input: Agent dynamics $A_d$, $s$, $g$, $\epsilon_g$, $\gamma = 0.9, \alpha = 1$,
  time step size in RRT* $\Delta t$, $\mathcal{M} \leftarrow$ 2D Floor plan;
Output: Q-tree;
Initialize: RRT* Tree, $T \leftarrow \emptyset$; Memory, $M \leftarrow \emptyset$;
$c \leftarrow s$;
begin
    while $||c - g|| \geq \epsilon_g$ do
        $a \leftarrow$ RRT*$(A_d, \mathcal{M}, \Delta t, g, c, \epsilon_g)$;
        Simulate $a$, and observe $c' \leftarrow \delta(c, a)$;
        Compute $r(c, a)$ by (3);
        Update $T \leftarrow T \cup \{c, a, r(c, a), c'\}$;
        Simulate $c \leftarrow c'$;
    end
    $M \leftarrow T(c, a, r(c, a), c')$;
    ▷ Nodes corresponding to shortest path between $s$
      and $g$ are saved in $M$
    while $M \neq \emptyset$ do
        $Q(c, a) \leftarrow$ BQL$(c, a, r(c, a), c', \alpha, \gamma)$ by (2);
        Q-tree $\leftarrow$ Q-tree $\cup \{c, a, Q(c, a)\}$;
        $M \leftarrow M \setminus \{c, a, r(c, a), c'\}$;
    end
    Return Q-tree;
end
```

Algorithm 2: Simultaneous Learning and Planning

```
Input: Q-tree from Algorithm 1, $s$, $g$;
Output: Optimal action $a^*$ at $c \in C$;
$c \leftarrow s$;
while $||c - g|| \geq \epsilon_g$ do
    if $Q(c, a) < 0$, for one or more $a \in A$ then
        Set $Q(c, a) \leftarrow 0$ and relearn $Q(c, a)$;
        Q-tree $\leftarrow$ Q-tree $\cup \{c, a, Q(c, a)\}$;
    end
    Find $a^*$ at $c$ by (7);
    Observe $c' \leftarrow \delta(c, a^*)$;
    if Obstacle between node $c$ and $c'$ then
        Evaluate temporary goal state $c''$ by satisfying
          $Q(c'', a'') > Q(c, a), a'', a \in A$;
        Q-tree $\leftarrow$ Algorithm 1$(s \leftarrow c, g \leftarrow c'', .)$;
        Find $a^*$ at $c$ by (7) and observe $c' \leftarrow \delta(c, a^*)$;
        if No obstacle between node $c$ and $c'$ then
            Execute $a^*$ at $c$ and move to
              $c' \leftarrow \delta(c, a^*)$;
            $c \leftarrow c'$;
        end
    end
    else
        Execute $a^*$ at $c$ and move to $c' \leftarrow \delta(c, a^*)$;
        $c \leftarrow c'$;
    end
end
```

VI. SIMULATION

This section shows simulation results in four parts. Simulation 1 deals with the evolution of the Q-tree (policy) over the iterations. Simulation 2 studies average execution time of the proposed SLPA over that of the RRT*. Simulation 3 shows the efficacy of the proposed SLPA in terms of adapting with the environmental uncertainties. Finally, in Simulation 4, SLPA is tested on Hector platform [21] in Gazebo [22] from Robot operating system (ROS) [23].

A. Setup

Simulation 1-3 are conducted in a system with an Intel Core i7-7500U CPU@ 2.70GHz * 4 processor and 8GB of RAM in MATLAB, and simulation 4 in the same system on ROS. An arena (say floor plan) of 100m $\times$ 100m is considered for all the simulations. An initial global path is created, considering obstacle free arena, using RRT* from start position $(25, 25)$ to goal position $(75, 75)$ as shown in Fig. 7a. The same path is then used by both SLPA and RRT* throughout the simulation. Over the time to introduce uncertainties in the environment, obstacles of dimension 6m $\times$ 6m are created randomly in the said environment. The obstacle configurations are kept identical in corresponding iterations for both SLPA and RRT*. The time step size for RRT* is Δt =2s. Goal tolerance ϵ_g =5m. The learning rate and discounting factor α and γ are respectively set to 1 (for fast learning) and 0.9. The value of r_{max} is fixed at 1. In all the simulations, we have considered that the agent is moving in a plane with quadcopter dynamics [24]. Simulation 4 is conducted on Gazebo [22] based Hector platform [21] in ROS [23]. The said package provides modelling, control and simulation of a quadcopter.

B. Procedure

This section deals with the procedures for four simulations. The first three simulations are conducted for 70 iterations, and the simulations are repeated five times to obtain average values. In simulation 1, the adaption of the Q-tree (policy) over the iterations because of the uncertainties introduced in the environment is demonstrated. In simulation 2, average execution time is the performance metric. Here, average execution time refers to the time taken by an agent to move from a given start to goal position in each iteration calculated from 70 iterations. The said average execution time is analyzed in the following circumstances: (i) for 70 iterations individually; (ii) Mean, standard deviation, min, max of average execution time in the said 70 iterations; (iii) the number of iterations out of 70 in which obstacles encountered.

In Simulation 3, the adaptability towards environmental uncertainties is considered as the performance metric. The adaptability towards environmental uncertainties refers to the situations, where the agent does not require to explore a new path, when an unknown obstacle is detected. Rather agent can exploit the existing learned path (policy) to avoid the detected obstacle. The said performance metric is also analyzed in the following circumstances: (i) frequency of obstacle detection and frequency of exploring a new path; (ii) percentage of frequency of exploring a new path over that of obstacle detection.

In Simulation 4, the Q-tree (policy) learned from simulation 1-3 after 70 iterations is migrated to the Hector platform in ROS, for planning with a different obstacle configuration. Here, the proposed SLPA is tested in Hector platform [21] by employing the quadcopter dynamics. The simulation is conducted five times to verify the environmental uncertainties.

C. Result and Discussion

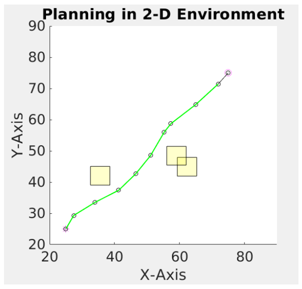
Results for Simulation 1-4 are given below.

1) Simulation 1: Fig. 7 shows the evolution of the Q-tree over 70 iterations. The obstacles are indicated by yellow square, and the executed path in a particular iteration is shown by green continuous line in Fig. 7. Corresponding Q-values of the paths in Fig. 7, are shown in Fig. 8. Fig. 7a shows the initial phase of Q-tree adaption with corresponding Q-values as shown in Fig. 8a. It is apparent from Fig. 7b and 8b that in iteration 4 the Q-tree is adapted by avoiding the obstacles (environmental uncertainties). This Q-tree adaption continues over the iterations as shown in Fig. 7c, 8c and Fig. 7d, 8d.

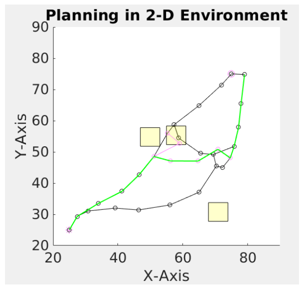
Fig. 9 shows an interesting outcome of the proposed SLPA. In Fig. 9a, the proposed SLPA is employed for three random obstacles, and the policy learned after 73 iterations is shown. A green continuous line is the path in Fig. 9a amidst yellow filled square obstacles. Now, the same learned policy (Fig. 9a) is employed to plan in a different obstacle configuration having ten random obstacles as shown in Fig. 9b. Similar to Fig. 9a in Fig. 9b also a path (green continuous line) obtained in the first iteration amidst yellow filled square obstacles. It is apparent from Fig. 9 that the proposed SLPA can adapt with the environmental uncertainty efficiently.

2) Simulation 2: The three circumstances of simulation 2 are as follows: (i) It is apparent from Fig. 10, that average execution time in SLPA (Fig. 10a) is less in almost every iteration as compared to the same in RRT* (Fig. 10b). (ii) The average minimum (min) and maximum (max) execution time required for SLPA and RRT* among all the 70 iterations are listed in the column 2 and 3 of Table II. The column 4 and 5 of Table II is tabulating the average mean and average standard deviation of the required average execution time for SLPA and RRT* among all the 70 iterations. It is evident from Table II that SLPA is outperforming RRT* in terms of max and mean average execution time. (iii) Fig. 11 shows the average execution time of those iteration where obstacles are detected. It is apparent form Fig. 11 that the average execution time in SLPA (Fig. 11a) is very less in almost every iteration as compared to the same in RRT* (Fig. 11b). This is because SLPA exploits the learned path (policy) if available after obstacle detection in the path.

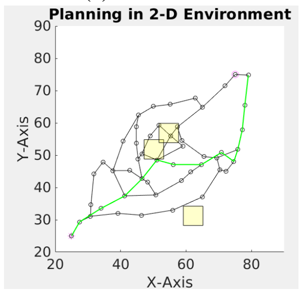
3) Simulation 3: Two circumstances of simulation 3 are given below: (i) Fig. 12b, 13b show the frequency of obstacle detected during the execution. The frequency of exploration is shown in Fig. 12a, 13a. It is apparent that frequency



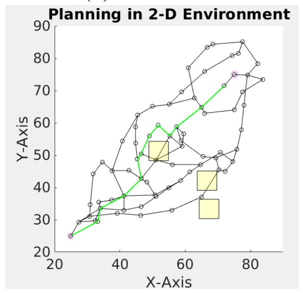
(a) Iteration 1



(b) Iteration 4

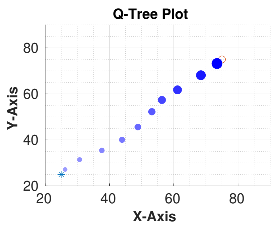


(c) Iteration 20

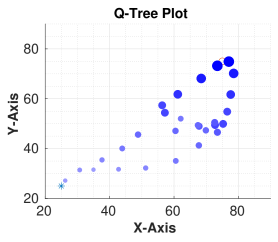


(d) Iteration 52

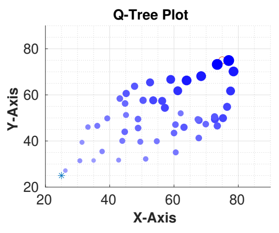
Fig. 7: Q-tree (policy) evolution with respect to environmental uncertainties by SLPA. start: $(25, 25)$ and goal: $(75, 75)$



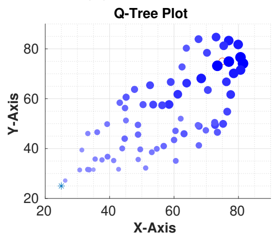
(a) Iteration 1



(b) Iteration 4



(c) Iteration 20

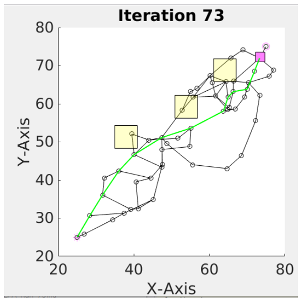


(d) Iteration 52

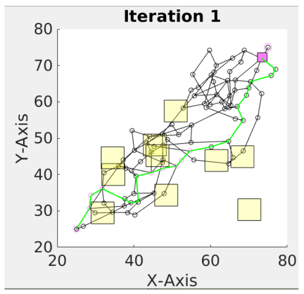
Fig. 8: Q-value (policy) evolution with respect to environmental uncertainties by SLPA. Higher Q-values are represented in the plot with larger circle and more color intensity.

TABLE II: Average execution time analysis (in second)

Algorithm	min	max	mean	standard deviation
RRT*	0.185	53.763	3.947	9.808
SLPA	0.2385	26.214	1.178	3.419

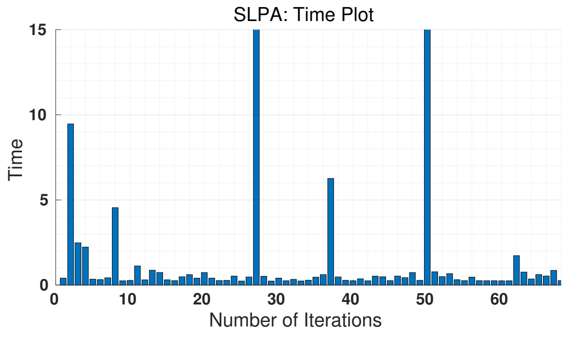


(a) With 3 obstacles

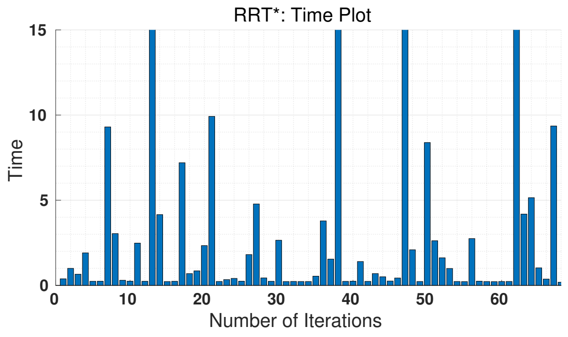


(b) With 10 obstacles

Fig. 9: Learning with three obstacles and planning with ten obstacles. start: $(25, 25)$ and goal: $(75, 75)$



(a) SLPA

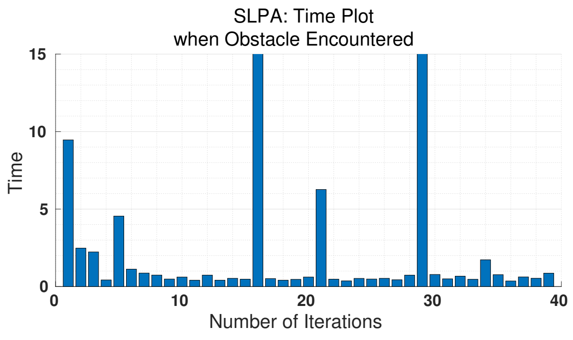


(b) RRT*

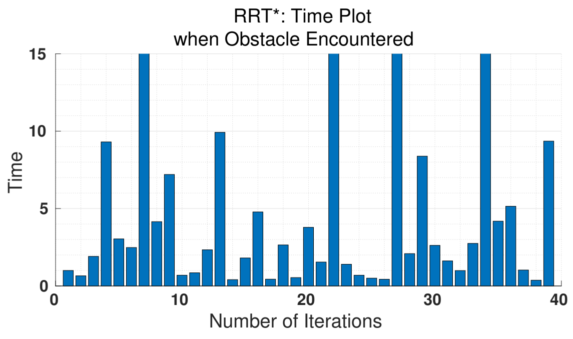
Fig. 10: Average execution time variation (in second)

of exploration in case of RRT* is more as compared to the same in SLPA, even though both the algorithms are facing identical obstacle configurations. (ii) Fig. 14 provides additional supports for SLPA in terms of adaptability towards environmental uncertainty. Here, percentage of frequency of exploration is recorded over the frequency of obstacle detection. The cause of such superiority of SLPA over RRT* is that SLPA exploits the learned paths (policy). As a result of which the frequency of exploration decreases for SLPA as the simulation progresses.

4) Simulation 4: Like Simulation 3 (Fig. 12a and 13a), here, in Fig. 15a and 15b the frequency of exploration is more in case of case of RRT*, which is less in case of SLPA with identical obstacle configuration. It is apparent from Fig. 15 that SLPA is exploiting the learned path (policy) to adapt with the environmental uncertainties, hence, outperforming.

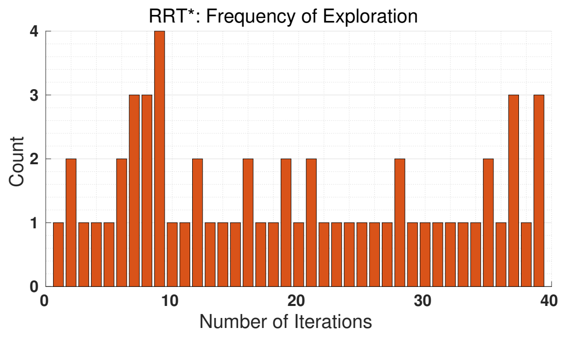


(a) SLPA

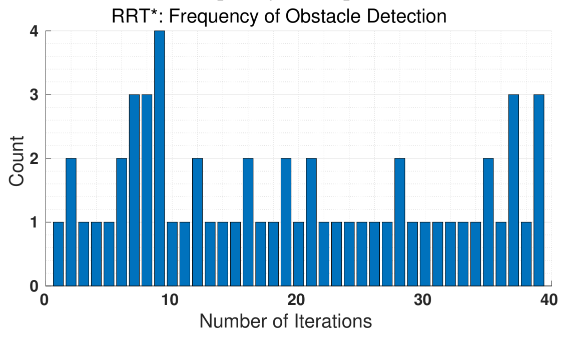


(b) RRT*

Fig. 11: Average execution time variation corresponding to the iterations for which obstacles detected (in second)

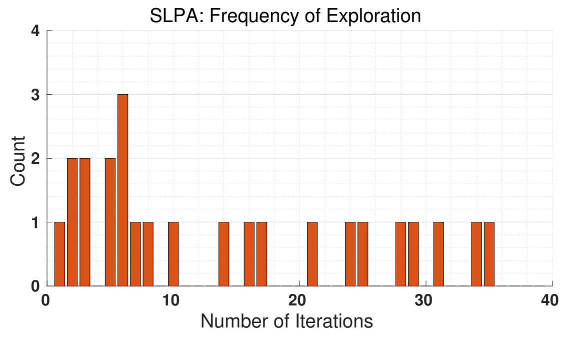


(a) Frequency of Exploration

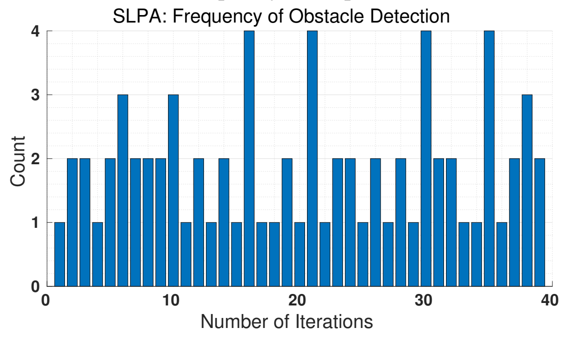


(b) Frequency of Obstacle Detection

Fig. 13: Adaptability to environmental uncertainties of RRT*

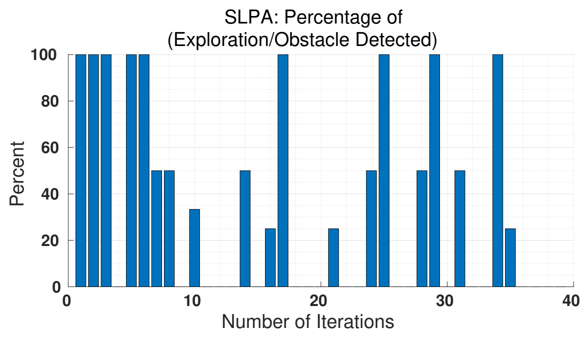


(a) Frequency of Exploration

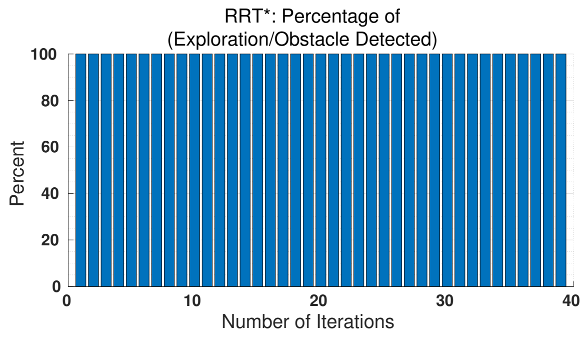


(b) Frequency of Obstacle Detection

Fig. 12: Adaptability to environmental uncertainties of SLPA

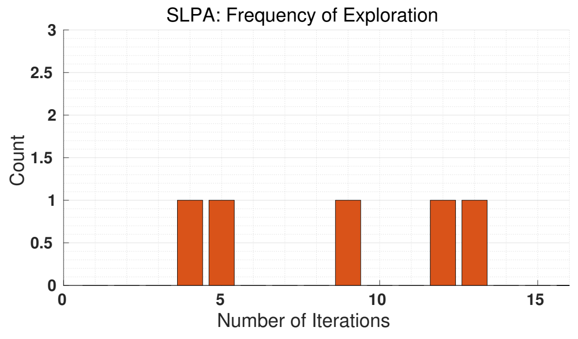


(a) SLPA

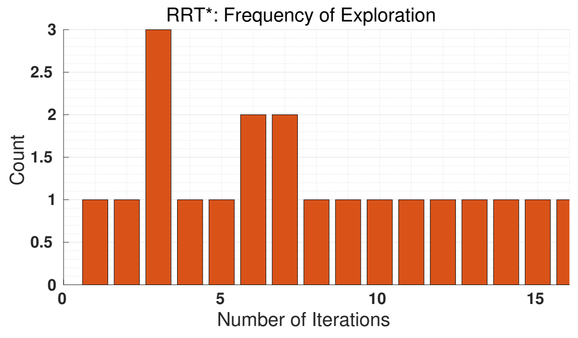


(b) RRT*

Fig. 14: Percentage of frequency of exploration with that of obstacle detected



(a) SLPA



(b) RRT*

Fig. 15: Adaptability to environmental uncertainties in ROS

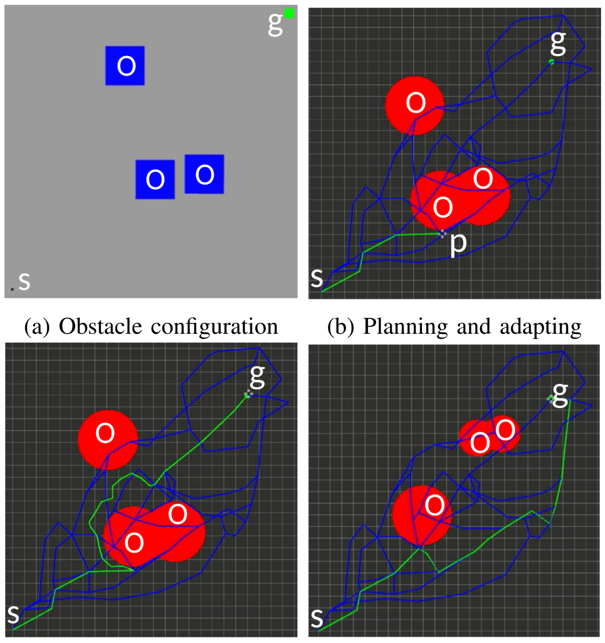
Fig. 16 shows planning outcomes of ROS simulation. The ROS simulations shown in Fig. 16 are conducted after executing SLPA for 70 iterations with different random obstacle configurations in Matlab. Details descriptions of Fig. 16 are provided in figure caption. It is apparent from this simulation that, SLPA can adapt with the environmental uncertainties efficiently.

Apparently, the above simulation results demonstrate the superiority of the proposed SLPA over RRT* mainly in terms of average execution time and handling environmental uncertainties.

VII. CONCLUSIONS AND FUTURE DIRECTION

The paper proposes an approach to learn and plan simultaneously in an uncertain environment. Floor plan of the environment is input to the proposed SLPA, to generate a global path using RRT* from a given start to goal. The backward Q-learning is employed to learn a policy for the said generated path. Backward Q-learning dynamically adapts the policy with uncertainties in the environment. Agent can avoid repeated planning by exploiting the learned policy. Additionally, RRT* balances the exploration-exploitation in backward Q-learning. Simulation results are provided to demonstrate the superiority of the proposed SLPA over the RRT*.

Due to the random nature of RRT*, the path (policy) generated by RRT* may not be the optimal one between start and goal, which can be addressed.



(a) Obstacle configuration (b) Planning and adapting

(c) Planning after adaption (d) Planning

Fig. 16: start: ’s’ (bottom left corner), goal: ’g’ (top right corner). Planning on hector platform in a new obstacle configuration, after policy adaption over 70 iterations in Matlab with different random obstacle configurations. Blue squares marked as ’O’ denotes obstacle in Fig. 16a. Red circles marked as ’O’ are the obstacles inflated by quad dimension in Fig. 16b-16d. Continuous blue lines are the learned policy from Matlab after 70 iterations as shown in Fig. 16b-16d. Continuous green lines are the executed path in Fig. 16b-16d by exploiting the learned policy (path). Fig. 16b shows a situation of exploring new path at point ’p’. After that the explored path is followed by the quad in Fig. 16c. On the other hand, Fig. 16d shows another situation in a different obstacle configuration, where exploitation of the existing policy is done.

REFERENCES

[1] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” 1998.
[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
[3] E. W. Dijkstra *et al.*, “A note on two problems in connexion with graphs,” *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.
[4] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.
[5] A. Stentz, “Optimal and efficient path planning for partially known environments,” in *Intelligent unmanned ground vehicles*. Springer, 1997, pp. 203–220.
[6] R. Deits and R. Tedrake, “Computing large convex regions of obstacle-free space through semidefinite programming,” in *Algorithmic foundations of robotics XI*. Springer, 2015, pp. 109–124.
[7] A. K. Sadhu, S. Shukla, T. Bera, and R. Dasgupta, “Safe and fast path planning in cluttered environment using contiguous free-space partitioning,” in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6972–6978.
[8] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration

spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[9] S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *49th IEEE conference on decision and control (CDC)*. IEEE, 2010, pp. 7681–7687.

[10] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[11] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[12] A. Konar, I. G. Chakraborty, S. J. Singh, L. C. Jain, and A. K. Nagar, "A deterministic improved q-learning for path planning of a mobile robot," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 43, no. 5, pp. 1141–1153, 2013.

[13] A. K. Sadhu and A. Konar, "Improving the speed of convergence of multi-agent q-learning for cooperative task-planning by a robot-team," *Robotics and Autonomous Systems*, vol. 92, pp. 66–80, 2017.

[14] A. K. Sadhu, A. Konar, T. Bhattacharjee, and S. Das, "Synergism of firefly algorithm and q-learning for robot arm path planning," *Swarm and Evolutionary Computation*, vol. 43, pp. 50–68, 2018.

[15] J. Wang, W. Chi, C. Li, C. Wang, and M. Q.-H. Meng, "Neural rrt*: Learning-based optimal path planning," *IEEE Transactions on Automation Science and Engineering*, vol. 17, no. 4, pp. 1748–1758, 2020.

[16] A. K. Sadhu and A. Konar, *Multi-Agent Coordination: A Reinforcement Learning Approach*. John Wiley & Sons, 2020.

[17] H.-T. L. Chiang, J. Hsu, M. Fiser, L. Tapia, and A. Faust, "Rl-rrt: Kinodynamic motion planning via learning reachability estimators from rl policies," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4298–4305, 2019.

[18] J.-J. Park, J.-H. Kim, and J.-B. Song, "Path planning for a robot manipulator based on probabilistic roadmap and reinforcement learning," *International Journal of Control, Automation, and Systems*, vol. 5, no. 6, pp. 674–680, 2007.

[19] A. Faust, K. Oslund, O. Ramirez, A. Francis, L. Tapia, M. Fiser, and J. Davidson, "Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 5113–5120.

[20] Y.-H. Wang, T.-H. S. Li, and C.-J. Lin, "Backward q-learning: The combination of sarsa algorithm and q-learning," *Engineering Applications of Artificial Intelligence*, vol. 26, no. 9, pp. 2184–2193, 2013.

[21] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. Von Stryk, "Comprehensive simulation of quadrotor uavs using ros and gazebo," in *International conference on simulation, modeling, and programming for autonomous robots*. Springer, 2012, pp. 400–411.

[22] *Gazebo: Robot simulation made easy*, (last accessed 10th August, 2020). [Online]. Available: http://gazebosim.org/

[23] *Robot Operating System (ROS)*, (last accessed 10th August, 2020). [Online]. Available: http://www.ros.org/about-ros/

[24] A. Joshi, A. Wala, M. Ludhiyani, S. Singh, M. Gagrani, S. Hazra, D. Chakraborty, D. Manjunath, and H. Chung, "Implementation of distributed consensus with guaranteed real-time communication on an outdoor quadrotor testbed," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, 2017, pp. 2158–2163.