Iterative SARSA: The Modified SARSA Algorithm for Finding the Optimal Path

Prajval Mohan, Pranav Narayan, Lakshya Sharma, Tejas Jambhale, Simran Koul

Abstract*: This paper presents a thorough comparative analysis of various reinforcement learning algorithms used by autonomous mobile robots for optimal path finding and, we propose a new algorithm called Iterative SARSA for the same. The main objective of the paper is to differentiate between the Q-learning and SARSA, and modify the latter. These algorithms use either the on-policy or off-policy methods of reinforcement learning. For the on-policy method, we have used the SARSA algorithm and for the off-policy method, the Q-learning algorithm has been used. These algorithms also have an impacting effect on finding the shortest path possible for the robot. Based on the results obtained, we have concluded how our algorithm is better than the current standard reinforcement learning algorithms.

***Keywords*: Iterative SARSA, Off-policy, On-policy, Optimal path, Q-learning, Reinforcement learning, SARSA**

I. INTRODUCTION

The optimal path is the shortest path there is, after applying all the parameters and restrictions. The optimal path can be either safe or unsafe to execute. The safe paths do not pass close to obstacles or cliffs, unless absolutely necessary. On the other hand, the unsafe paths disregard their closeness to cliffs and obstacles in order to find the most optimal path (or the shortest path). The main need of path finding algorithms heightened when the concept of self-driving cars started to float. In the present scenario, most of the path calculations are done via reinforcement learning algorithms. Some of the reinforcement learning algorithms that have been highlighted in the recent times are SARSA and Q-learning. These algorithms trace the path of robot, helping it to avoid obstacles as well as reach the target location via the shortest path. The main concept used in these algorithms is the concept of rewards. Rewards are calculated based on two key policies, i.e., on-policy and off-policy. The on-policy method calculates the coordinate of the next step using the coordinate of the current step as well as the action of the current policy, by the process of exploration. This removes the question of experimental risk, which might lead to a sudden collapse of the program. SARSA uses this risk-free approach. On the other hand, the off-policy method predicts the next coordinate of the robot using the current coordinate of the robot along with a greedy action. The reward is calculated using this greedy policy, although the robot follows the path as though the greedy action wasn't used. The off-policy has a better outcome factor in terms of shortest path execution speed, but there is a heightened risk wherein the program might end up crashing. Developing on these algorithms, we have come up with a new algorithm, called Iterative SARSA, that provides us with the safest as well as the most optimal path available.

Manuscript received on March 15, 2020.
Revised Manuscript received on March 24, 2020.
Manuscript published on March 30, 2020.
* Correspondence Author

Prajval Mohan*, B. Tech in Computer Science and Technology from Vellore Institute of Technology, Vellore, India
Pranav Narayan, B. Tech in Computer Science and Technology from Vellore Institute of Technology, Vellore, India
Lakshya Sharma, B. Tech in Computer Science and Technology from Vellore Institute of Technology, Vellore, India
Tejas Jambhale, B. Tech in Computer Science and Technology from Vellore Institute of Technology, Vellore, India
Simran Koul B. Tech in Computer Science and Technology from Vellore Institute of Technology, Vellore, India

II. RELATED WORK

A. Reinforcement Learning

A striking assortment of issues in mechanical autonomy might be normally expressed as issues of reinforcement learning.[7] Reinforcement learning empowers a robot to self-sufficiently find an ideal conduct through experimentation associations with the given condition. Rather than expressly specifying the answer to the issue, in order to calculate each step of the robot, the reinforcement methodology utilizes the feedback received from the agent in the form of a scalar function.[12],[18]
The main aim of reinforcement learning to figure out an optimal policy $\pi*$ that maps states to actions, hence, maximizing the expected return J, which is analogous to the aggregated expected reward.[17]

$$J = E\left\{ \sum_{h=0}^{H} R_h \right\}$$

The rewards can also be discounted to accommodate errors. This is done using the discount factor γ (where, $0 \le \gamma < 1$)

$$J = E\left\{ \sum_{h=0}^{\infty} \gamma^h R_h \right\}$$

where,
R_h is the overall reward w.r.t. agent h

B. Optimal Path

Traditionally, paths were calculated manually by entering the coordinates of the path into the program.

Since this method was tedious and posed various problems, the need to create a new self-learning approach to autonomously calculate the path was needed.[8],[9] To calculate the path, the work environment with all the cliffs and obstacles is created and provided. The program runs the algorithm on this environment. This training is done with reinforcement learning and neural networks.[10],[13]

C. On-policy

The on-policy is a method of reinforcement learning in which the next state is calculated using the current state along with the data received by the exploration steps. The SARSA algorithm uses this policy. Although, this policy may be a bit more expensive, when compared to the off-policy method, it is safer and has lower risks.[11]

D. Off-policy

The off-policy is a method of reinforcement learning in which the next state is calculated using the current state along with a greedy step. This policy predicts the next coordinate/state instead of calculating it, as seen in the on-policy method. The coordinate with the greatest reward is chosen as the next state. Even though the next state is chosen based on the reward system, the program processes it as though the greedy algorithm wasn't used at all. On using this algorithm, even though we get the most optimal path possible, it might be risky in some situations, in which we end up with a high negative reward. The Q-learning algorithm uses this policy.[6]

E. SARSA

SARSA (State-Action-Reward-State-Action) is called so, because it experiences to update the Q-values. It uses the on-policy method. SARSA is represented with the attributes {S, A, γ, α}, where, S is the set of states, A is a set of actions, $\gamma(0\leq\alpha\leq1)$ is the discount and $\alpha(0<\alpha\leq1)$ is the step size. These are used to calculate the Q-values (Q [S, A]).[5],[15],[16]

$$Q_{new}(s_t,a_t) \leftarrow Q_{old}(s_t,a_t) + \alpha[r_t + \gamma.Q_{exploration\ value}(s_{t+1},a_{t+1}) - Q_{old}(s_t,a_t)]$$

where,
s_t represents the initial state,
s_{t+1} represents the final state,
r_t represents the reward

F. Q-learning

Q-learning is the reinforcement algorithm in which, the agent takes the most optimal action under different circumstances. This algorithm uses the off-policy model to obtain its Q-values, and hence, reward is calculated based on the greedy action. Q-learning is also represented with the attributes {S, A, γ, α}, where, S is the set of states, A is a set of actions, $\gamma(0\leq\alpha\leq1)$ is the discount and $\alpha(0<\alpha\leq1)$ is the step size. These are used to calculate the Q-values (Q [S, A]). [1],[2],[3],[14]

$$Q_{new}(s_t,a_t) \leftarrow Q_{old}(s_t,a_t) + \alpha[r_t + \gamma.\max_a.Q_{estimated\ future\ value}(s_{t+1},a) - Q_{old}(s_t,a_t)]$$

where,
s_t represents the initial state,
s_{t+1} represents the final state,
r_t represents the reward

III. PROPOSED METHOD

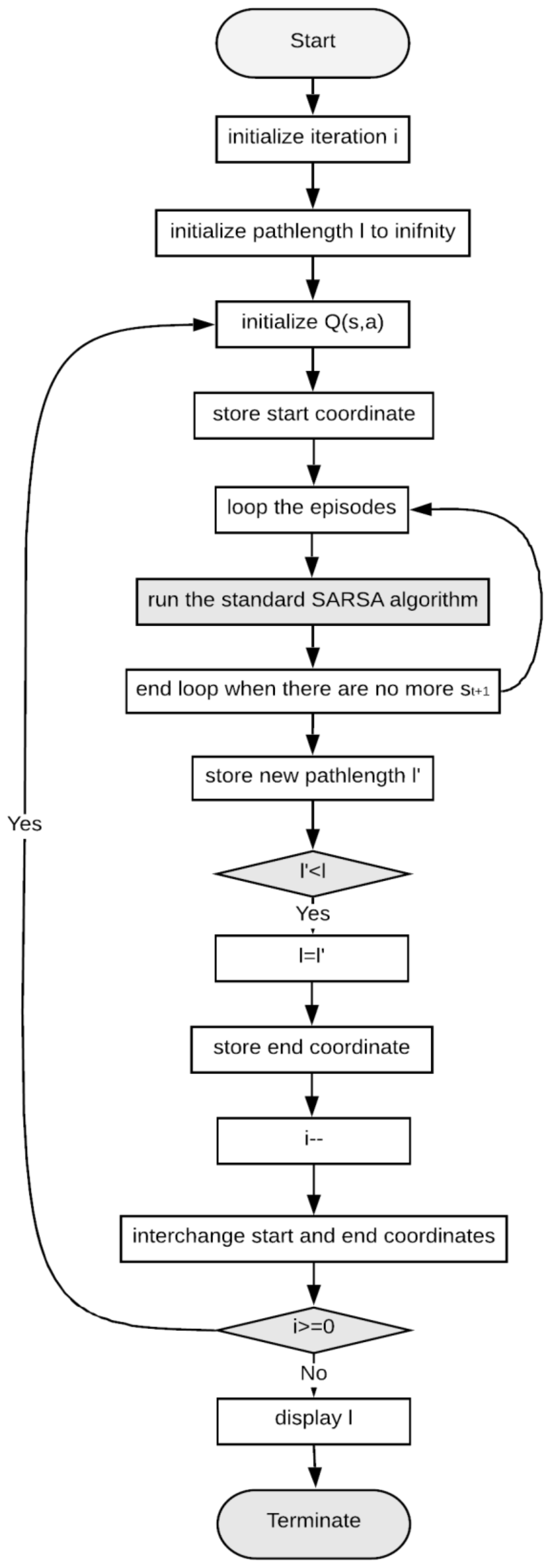
Iterative SARSA.

Elaborating on the concept of the SARSA algorithm, we have proposed our new iterative algorithm. Like in the SARSA algorithm, the actor moves from the start to the end location. In the iterative SARSA algorithm, once the actor reaches the final destination, the start and end point are interchanged and the algorithm is rerun. According to the concept of SARSA, the path found is always a safe one, unlike in the Q-learning algorithm. Hence, any path that's calculated is going to be safe. Once the iteration is completed, the shortest path is selected out of all the iterations and the path is highlighted. Although, iterative SARSA has a high time complexity, it provides us with the best safe path.[4]

Algorithm of Iterative SARSA.

```
1: initialize number of iterations 'i'
2: initialize pathlength l → ∞ (based on grid environment)
3: initialize Q (s, a) for all s, a
4: store the coordinate value of the start location
5: loop over episodes
6: initialize s
7: repeat this for each step of the episode
8: choose a from s using the π* policy extracted from Q
9: $Q_{new}(s_t,a_t) \leftarrow Q_{old}(s_t,a_t)$
        $+\alpha[r_t+ \gamma.Q_{exploration\ value}(s_{t+1},a_{t+1}) - Q_{old}(s_t,a_t)]$
10: $s_t \leftarrow s_{t+1}$
11: end loop
12: store the length of the path calculated in l'
13: if (l'< l)
    {
        l = l';
    }
14: store the coordinate value of the end location
15: i--;
16: interchange the start and end coordinate values
17: if (i ≥ 0)
    {
        jump to step 3
    }
    else
    {
        display l and terminate
    }
```

Flowchart of Iterative SARSA.



IV. EXPERIMENTAL SETUP

A. Software Specifications

We have executed the algorithm on Python 3.7.0 and its dependent modules. The modules used in the program are:

Table- I: Software Specifications

Pandas	Used to read and manipulate the data.
Matplotlib	It is a python library used to depict the data in the form of various interactive graphs.
Pillow	It's image library in Python to manage, manipulate and save.
Timeit	It's a python library used to time blocks of code.

V. IMPLEMENTATION

The analysis for the program was done by creating a fixed environment wherein, we ran the Q-learning, SARSA and Iterative SARSA algorithm. Based on the time of execution, optimal path and safeness of the algorithm, we have concluded the appropriate scenarios for each algorithm. For the shortest path, we have used the Q-table to calculate and count the path length. As for the time calculation, we have used the timeit() function. The episode via steps and episode via cost graphs have also been calculated. An, episode is a sequence of steps the actor takes in each transversal. The episode via steps graph shows us the numbers of episodes vs. the number of steps in each episode and the episode via cost graph shows us the number of episodes vs. the cost of each episode.

A. Environment

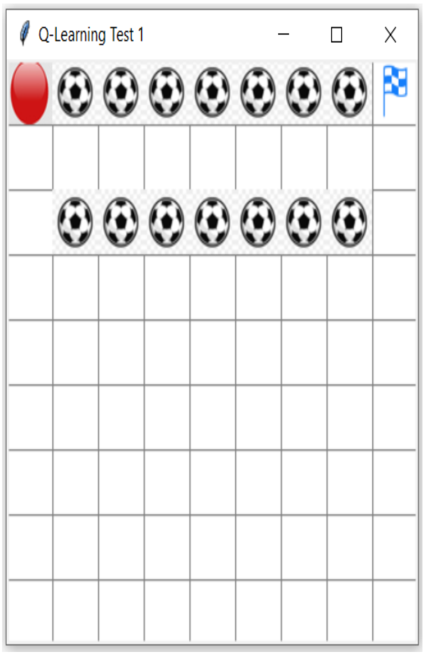


Fig. 1. The red dot is the actor and also represent the start location. The line of footballs represent the cliff and the Flag represents the end location.

B. Q-learning

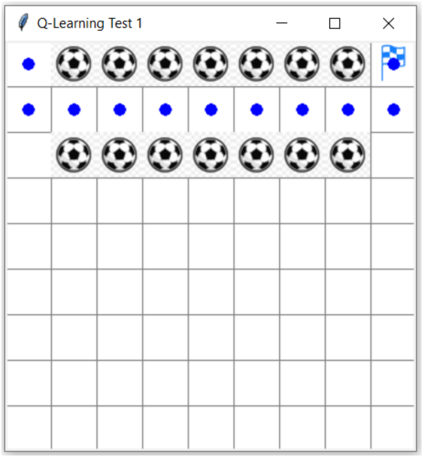


Fig. 2. Shortest path calculated using Q-learning with path length of 10

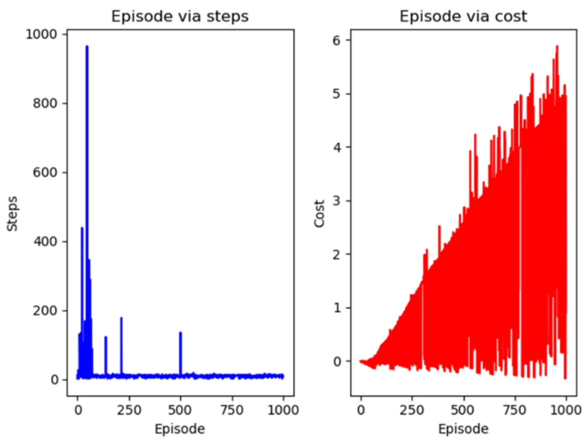


Fig. 3. Episode via steps and Episode via cost graph

C. SARSA

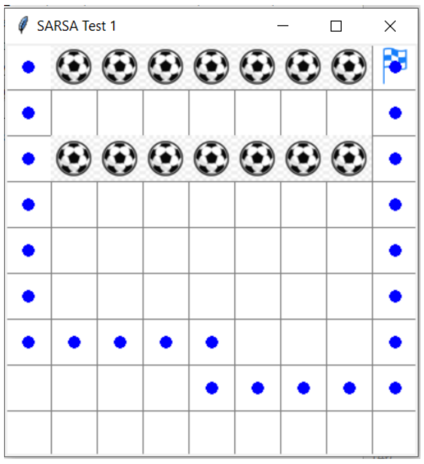


Fig. 4. One of the safe paths calculated using the SARSA algorithm with path length of 22

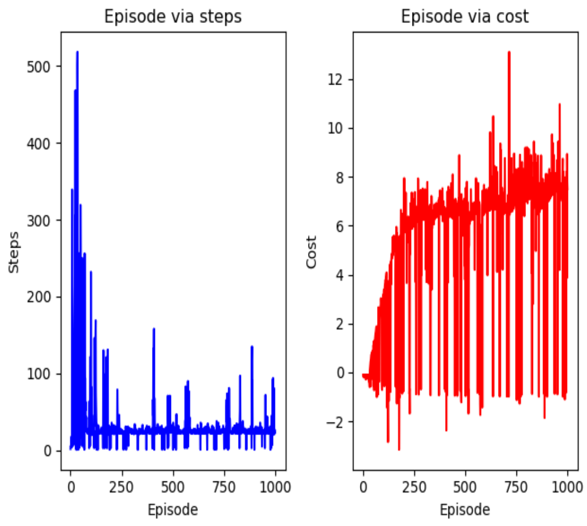
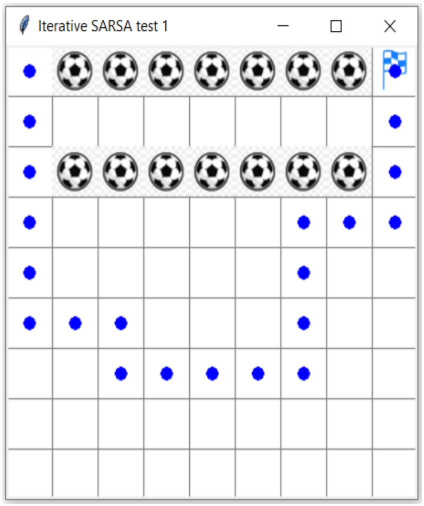


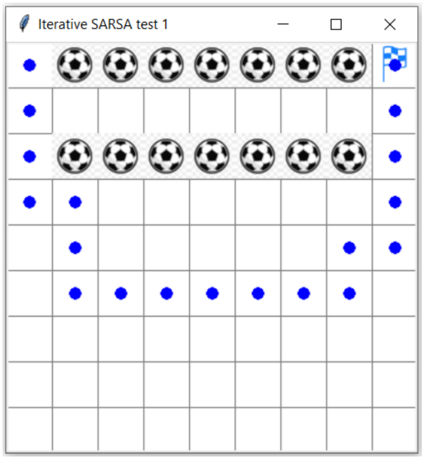
Fig. 5. Episode via steps and Episode via cost Graphs

D. Iterative SARSA

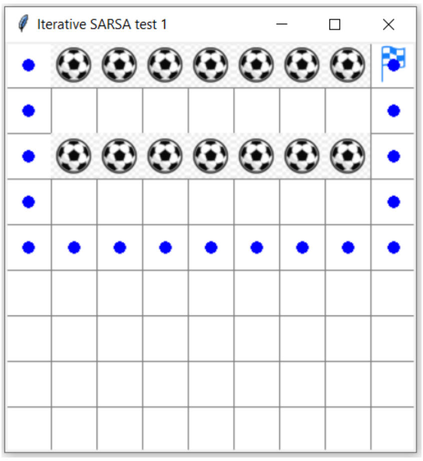
Iterations considered while Running the Code are 3.



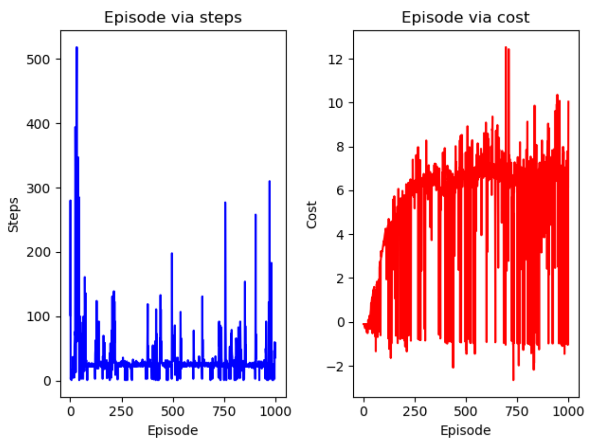
(a) Iteration 1 (from start to end) with path length 20



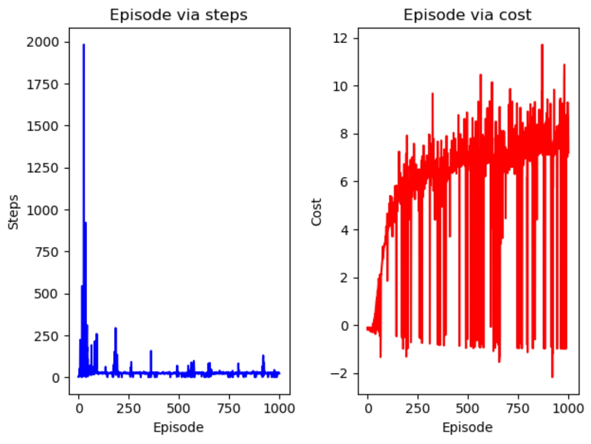
(b) Iteration 2 (from end to start) with path length 18



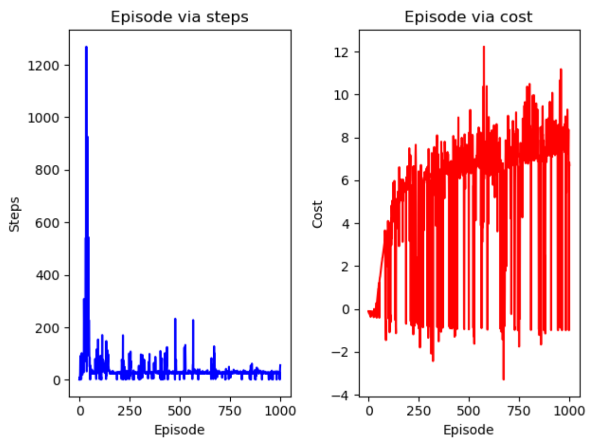
(c) Iteration 3 (from start to end) with path length 16
Fig. 6(a)(b)(c). Safe path calculated for each iteration using the Iterative SARSA algorithm



(a) Iteration number 1



(b) Iteration number 2



(c) Iteration number 3
Fig. 7(a)(b)(c). Episode via steps and Episode via cost graphs for each iteration using the Iterative SARAS algorithm

Inference Table.

Table- II. Analysis of all the algorithms

Algorithm		**Path Length**	**Execution Time (sec)**	**Path Description**
Q-learning		10	11.3482	Optimal Path with Executional Risks
SARSA		22	16.2297	Safe but not Optimal
Iterative SARSA	Iteration 1	20	17.6531	Safest Optimal Path Possible
	Iteration 2	18	14.5246	
	Iteration 3	16	16.3137	

VI. CONCLUSION

According to our inference, the Q-learning algorithm executes the fastest and gives the most optimal path. But, as we can see in Fig2., the path in chosen by Q-learning passes between the 2 cliffs. This possesses a high executional risk in which the program might crash. Coming to SARSA, this algorithm takes comparatively more time to execute. Although this algorithm is safe, it has a longer path length (Fig4.). As we come to the Iterative SARSA algorithm, although it has a much higher time complexity, it finds the safest optimal path (Fig6(c).).

Hence, if one wants to calculate the path without considering the executional risks, the Q-learning algorithm can be considered. If a safe path, disregarding the path length, and with low execution time has to be calculated, the SARSA algorithm can be used. The Iterative SARSA algorithm can be used when time complexity is not an issue and we want the safest most optimal path.

REFERENCES

1. Park, Kui-Hong, Yong-Jae Kim, and Jong-Hwan Kim. "Modular Q-learning based multi-agent cooperation for robot soccer." Robotics and Autonomous Systems 35.2 (2001): 109-122.
2. Chen, Chunlin, Han-Xiong Li, and Daoyi Dong. "Hybrid control for robot navigation-a hierarchical Q-learning algorithm." IEEE Robotics & Automation Magazine 15.2 (2008): 37-47.
3. Konar, Amit, et al. "A deterministic improved Q-learning for path planning of a mobile robot." IEEE Transactions on Systems, Man, and Cybernetics: Systems 43.5 (2013): 1141-1153.
4. Wang, Yin-Hao, Tzuu-Hseng S. Li, and Chih-Jui Lin. "Backward Q-learning: The combination of Sarsa algorithm and Q-learning." Engineering Applications of Artificial Intelligence 26.9 (2013): 2184-2193.
5. H. van Seijen, H. van Hasselt, S. Whiteson and M. Wiering, "A theoretical and empirical analysis of Expected Sarsa," 2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning, Nashville, TN, 2009, pp. 177-184.
6. Gu, Shixiang, et al. "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates." 2017 IEEE international conference on robotics and automation (ICRA). IEEE, 2017.
7. Lin, Lian-ming, Hao Wang, and Yi-xiong Wang. "Sarsa Reinforcement Learning Algorithm based on Neural Networks." Computer Technology and Development 1 (2006): 30-32.
8. Sariff, N., and Norlida Buniyamin. "An overview of autonomous mobile robot path planning algorithms." 2006 4th Student Conference on Research and Development. IEEE, 2006.
9. Jan, Gene Eu, Ki Yin Chang, and Ian Parberry. "Optimal path planning for mobile robot navigation." IEEE/ASME transactions on mechatronics 13.4 (2008): 451-460.
10. Fan, Xiaoping, et al. "Optimal path planning for mobile robots based on intensified ant colony optimization algorithm." IEEE International Conference on Robotics, Intelligent Systems and Signal Processing, 2003. Proceedings. 2003. Vol. 1. IEEE, 2003.
11. Banerjee, Bikramjit & Sen, Sandip & Peng, Jing. (2004). On-policy concurrent reinforcement learning. J. Exp. Theor. Artif. Intell.. 16. 245-260. 10.1080/09528130412331297956.
12. Nair, Ashvin, et al. "Overcoming exploration in reinforcement learning with demonstrations." 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018.
13. Raja, Purushothaman, and Sivagurunathan Pugazhenthi. "Optimal path planning of mobile robots: A review." International journal of physical sciences 7.9 (2012): 1314-1320.
14. Tai, Lei, and Ming Liu. "A robot exploration strategy based on q-learning network." 2016 IEEE International Conference on Real-time Computing and Robotics (RCAR). IEEE, 2016.
15. Ganger, Michael, Ethan Duryea, and Wei Hu. "Double Sarsa and double expected Sarsa with shallow and deep learning." Journal of Data Analysis and Information Processing 4.4 (2016): 159-176.
16. Yu, Shanqing, et al. "Q value-based Dynamic Programming with SARSA Learning for real time route guidance in large scale road networks." The 2012 International Joint Conference on Neural Networks (IJCNN). IEEE, 2012.
17. Gosavi, Abhijit. "A reinforcement learning algorithm based on policy iteration for average reward: Empirical results with yield management and convergence analysis." Machine Learning 55.1 (2004): 5-29.
18. Singh, Satinder, et al. "Convergence results for single-step on-policy reinforcement-learning algorithms." Machine learning 38.3 (2000): 287-308.

AUTHORS PROFILE

Prajval Mohan, was born in Hyderabad, India on 23rd October, 1998. He completed his Senior High School from FIITJEE Junior College, Hyderabad, graduated with 96.1 percent and received the Honorary Certificate of Merit in the year 2016. Prajval is currently pursuing his B. Tech in Computer Science and Engineering from Vellore Institute of Technology, Vellore, India. His areas of interest include Robotics, Machine Learning, Artificial Intelligence and Cloud Computing. He has advanced working knowledge of Robotics and Database handling which were strengthened by completing various projects and internships in the respective fields. He also has ongoing research in the field of Operating Systems and Parallel Distributed Computing.

Pranav Narayan was born in Thane, India on July 29, 1999. He completed his senior high school in Indian School Muscat, Oman. He is currently pursuing his Bachelor's in Computer Science and Engineering from Vellore Institute of Technology, Vellore. He is currently working on projects which involve computer science concepts like Operating Systems, Robotics and Software Engineering.

Lakshya Sharma was born in Jaipur, India on 25th January, 1999. He was raised in Delhi, India and completed his Senior high school from D.A.V Public School. He is currently pursuing B. Tech in Computer Science Engineering from Vellore Institute of Technology, Vellore. His research interests include Deep learning, artificial intelligence, autonomous object avoiding and path planning robots. He has worked on several machine learning, deep learning projects and has an ongoing research in offline signature recognition using Siamese networks.

Tejas Jambhale was born in Maharashtra, India on 17th November, 1998. He is a student of Vellore Institute Technology, Vellore currently pursuing Computer Science Engineering. He has skills in domains including machine learning, web development and deep learning. He has completed research in cyber security and deep learning and likes building different projects to explore his skills.

Simran Koul was born in Jammu, India on 10th November, 1998. She completed her senior high school in Indian School Ahmadi, Kuwait. She is currently pursuing her Bachelor's Degree in Computer Science Engineering in VIT, Vellore, India. She is currently working on projects which involve concepts of Robotics and Artificial intelligence and Natural Language Processing.