2019 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)
December 11–12, 2019, Amity University Dubai, UAE

Reinforcement Learning Based Approach For Mobile Robot Navigation

Mohammed Jaseem M
Department of Mechanical Engineering
Indian Institute of Technology Madras
Chennai, India
jaseem.ktk@gmail.com

Robins Mathew
Department of Mechanical Engineering
Indian Institute of Technology Madras
Chennai, India
connectrobins@gmail.com

Somashekhar S Hiremath
Department of Mechanical Engineering
Indian Institute of Technology Madras
Chennai, India
somashekhar@iitm.ac.in

***Abstract* — Autonomous navigation of mobile robots has been a subject of research for many decades. Many algorithms, both classical and heuristic have been developed for the navigation of the wheeled mobile robots. Classical approaches can get tedious and can get stuck at local optima as the environment gets more complex. Heuristic approaches are gaining prominence nowadays because of its closeness to human way of behavioral learning. Reinforcement learning (RL) is the idea of having a robot learn how to accomplish a task. It is one of the heuristic approaches which gains knowledge by trial and error, so there is no need for any expert knowledge. In the current work, an attempt has been made to implement reinforcement learning for autonomous navigation of wheeled mobile robots. A RL agent is created for controlling the obstacle avoidance and goal-seeking behavior of a single robot. Then, a kinematic controller is developed to help the robot to move through the waypoints generated by the RL algorithm. Subsequently, the performance analysis of the controller is carried out and the benefits and limitations of the RL algorithm is studied.**

Keywords—Mobile robot, Reinforcement learning, Kinematic control, Autonomous robot navigation

I. INTRODUCTION

Autonomous navigation is the ability of a robot to plan its motion through an environment. Autonomous mobile robots have applications in the areas of agriculture, mining, healthcare, exploration, etc. Development of algorithms for this purpose has been a fundamental need in robotics. Path planning is the determination of an optimized path by the robot from start position to goal position without hitting any obstacles. Optimization can be based on several criterions such as time, distance or energy where distance is the most commonly used one. Different path planning approaches have been proposed and tested in various environments with static and dynamic obstacles. Classification of path planning algorithms can be done in different ways. Based on the availability of information about the environment, it can be divided into local path planning (on-line) and global path planning (off-line). Global path planners have complete information about its environment in advance, so it can make a low-resolution high-level path by looking globally. Local path planners can generate a high-resolution low-level path based on its immediate surroundings. Path planning algorithms can also be classified into classical approaches and heuristic-based approaches as shown in Fig.1. In Classical approaches, either a solution will be found, or it would be proven that the solution does not exist. These approaches are generally easier to implement when the environment is simple but the algorithm, is computationally more expensive. Bug algorithms are one of the popular classical approaches.

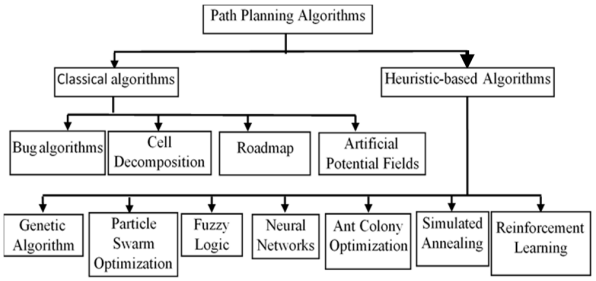

Fig. 1. Classification of path planning algorithms

It is one of the most straightforward path planning algorithms which assume that the robot is in a 2D environment equipped with a short-range sensor and the obstacles are unknown and convex. The Bug1 and Bug2 algorithms [1] are simple but guarantee a solution when possible. Even though classical approaches are useful, it can be tedious and has a chance to get stuck in local optima and may not work in a complex environment. Heuristic approaches are being increasingly implemented because of their closeness to the human way of behavior learning. Moreover, path planning with multiple obstacles is NP-hard problem [2].

In the recent decades, machine learning has been revolutionizing each and every field of engineering by adding intelligence to the system. Autonomous navigation is one such field that requires intelligence. Reinforcement learning (RL) is an area of machine learning where an agent learns how to accomplish a task in an environment by performing actions. RL is one of the heuristic approaches that does not require any expert knowledge and are unsupervised. Also, RL approach is scalable, knowledge transferable and learns while working with the environment. In RL, an agent tries to interact with the environment by taking actions based on the state of the environment. The action causes change of state of environment and produces a reward. Thus, the robot learns about the environment from the rewards that it gets. The fundamental goal of the agent is to try and maximize the reward. The framework of a RL scenario is shown in Fig. 2. There are different learning algorithms which can be used for RL. However, Q learning is one of the prominent method for RL.

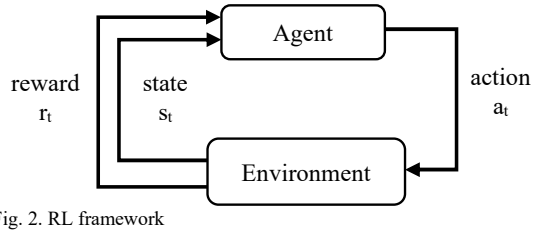

Fig. 2. RL framework

978-1-7281-3778-0/19/$31.00 ©2019 IEEE

Q learning is a model-free RL algorithm which finds the best action given the current state. During Q learning a table is created called the Q table where each row of the table corresponds to a state and each column of the table corresponds to an action and the entries in the table are called Q values. The table is initialized into a zero matrix and is updates based on the rewards the system obtains from the actions it takes. The agent can interact with the environment in two ways, i.e., exploitation and exploration. If the agent takes the Q table as a reference and takes action based on which it can get maximum reward, then it is called exploiting but if the agent takes actions randomly to explore new states and provide new information to Q table, it is called exploring. The Q values in Q table is updated using the equation given in (1). Here s_t is state at time t, a_t is the action taken at time t, α is the learning rate, r_{t+1} is the reward for obtained for change of state from s_t to s_{t+1} ,γ is the discounting factor.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (1)$$

The agent in this case is the robot itself and the environment is the place where the robot moves. The state can be defined by the position and orientation of the robot and actions will be the movements carried out by the robot. The robot is trained until the Q values converge to optimum values. Therefore, a trained robot will be able to decide optimal action at any given state. The training time increases with the no. of states and actions. Hence, the states and the actions are discretized to arrive at a reasonable time for training.

Several researches have implemented autonomous navigation using Q learning as well as other RL algorithms. [3] implemented Q learning on a single agent system to achieve successful obstacle avoidance and goal-seeking behavior but without a kinematic controller. [4] used a combination of fuzzy logic and RL for navigation. The input for the fuzzy system is the state space provided by range sensors and output is the action space. [5] speeded up the learning process by splitting the learning phase into two stages. In the initial phase, a supplied control policy is given and the RL agent learns passively. After that, the agent takes control and learns actively. [6] proposed improvements for RL agent working in a dynamic environment. Implementation of a forgetting mechanism which degrades the Q value of unvisited states increases the exploration rate and makes it better suitable for dynamic environment. Feature-based input, when combined with hierarchical structuring of RL agent, showed promising results. [7] presented a reactive on-line motion planner, Robust autonomous waypoint generation (RAW), for mobile robots navigating in unknown and unstructured environments. RAW generates a locally maximal ellipsoid around the robot, using semi-definite programming, such that the surrounding obstacles lie outside the ellipsoid. One shot RL is used to navigate in a known environment with fixed goal in [8] which reduces the training time in real world. Continuous control of mobile robot is established in [9] with the help of sparse 10-dimensional range findings. A novel method for path planning in dynamic environment using Simultaneous localization and mapping (SLAM) is proposed in [10].

A reinforcement learning agent then generates a local waypoint in the robot's field of view, inside the ellipsoid. The robot navigates to the waypoint and the process iterates until it reaches the goal. However, Navigation of a mobile robot from one waypoint to another is also a challenging task. Thus, an integration of a kinematic controller and RL algorithm can end up giving some good results.

II. Kinematic Controller

A waypoint tracking controller for differential drive mobile robot is being used in this project. Waypoints are the points through which the robot has to move in order to reach its final position. The controller is a method proposed by [11] is used here. The posture p_i of a robot at an instant i in a physical space is described using x, y coordinates and the orientation of the robot as given in (2).

$$p_i = \begin{bmatrix} x_i \\ y_i \\ \theta_i \end{bmatrix} \quad (2)$$

If the robot is assumed to be under pure rolling,

$$\dot{p}_i = \begin{bmatrix} \dot{x}_i \\ \dot{y}_i \\ \dot{\theta}_i \end{bmatrix} = \begin{bmatrix} \cos\theta_i & 0 \\ \sin\theta_i & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_i \\ \omega_i \end{bmatrix} \quad (3)$$

The posture error p_{ie} with respect to the k[th] waypoint posture transformed into local coordinates will be,

$$p_{ie} = \begin{bmatrix} x_{ie} \\ y_{ie} \\ \theta_{ie} \end{bmatrix} = \begin{bmatrix} \cos\theta_i & \sin\theta_i & 0 \\ -\sin\theta_i & \cos\theta_i & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{wk} - x_i \\ y_{wk} - y_i \\ \theta_{wk} - \theta_i \end{bmatrix} \quad (4)$$

where θ_{wk} is calculated as

$$\theta_{wk} = \tan^{-1}\left(\frac{y_{wk} - y_i}{x_{wk} - x_i}\right) \quad (5)$$

The linear and angular velocities of a differential drive robot can be written in terms of wheel velocities as shown in (6), (7). Here d_w is the wheel diameter, w_T is the wheel track width, ω_{1i} and ω_{2i} are velocities of left and right velocities.

$$\begin{bmatrix} v_i \\ \omega_i \end{bmatrix} = \frac{1}{4} \begin{bmatrix} d_w & d_w \\ \frac{2d_w}{w_T} & \frac{-2d_w}{w_T} \end{bmatrix} \begin{bmatrix} \omega_{1i} \\ \omega_{2i} \end{bmatrix} \quad (6)$$

$$\begin{bmatrix} \omega_{1i} \\ \omega_{2i} \end{bmatrix} = \frac{1}{d_w} \begin{bmatrix} 2 & w_T \\ 2 & -w_T \end{bmatrix} \begin{bmatrix} v_i \\ \omega_i \end{bmatrix} \quad (7)$$

The control equation proposed by [8] is shown in (8). Here v_{ci},ω_{ci} are the controlled velocities, v_{ri},ω_{ri} are the reference velocities supplied by the user, k_1, k_2, k_3 are the controller gains that needs to be optimized. The controller is able to track trajectory efficiently.

$$\begin{bmatrix} v_{ci} \\ \omega_{ci} \end{bmatrix} = \begin{bmatrix} v_{ri}\cos\theta_{ie} + k_1 x_{ie} \\ \omega_{ri} + k_2 v_{ri} y_{ie} + k_3 v_{ri} \sin\theta_{ie} \end{bmatrix} \quad (8)$$

III. Simulation Results

The task of navigation can be divided into two subproblems namely obstacle avoidance and goal seeking. A virtual robot is represented by a circle of radius 35 units. It is assumed to be a wheeled mobile robot with two independently actuated wheels on the side and three range sensors on left,

right and front side of the robot and is placed on a bench of fixed length and width. Obstacles generated are also represented by circles at random positions. Distance and angle to objects can be sensed virtually by differencing the distance between the centers of the robot and obstacle from the sum of radius of robot and obstacle. A GUI is created in MATLAB for giving user inputs as shown in Fig. 3.

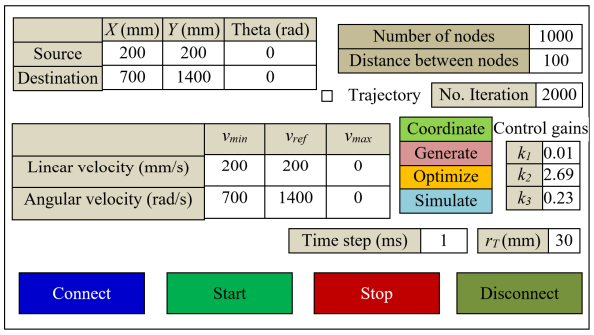

Fig. 3. MATLAB GUI

The user inputs like position of robot, obstacles and goal is fed to the RL agent through this GUI. The simulations are divided into three stages. First of all, only goal- seeking behavior is simulated to ensure the proper verification of basic framework of the system. Then obstacle avoidance and goal-seeking behavior is combined in an environment where obstacles are generated at random points to result in the autonomous navigation of a single robot. The agent generates the waypoints based on the state of the system. Waypoints are abstract points through which the robot has to be controlled to reach the goal. In the final stage, the kinematic controller is implemented. The waypoints generated are given to the kinematic controller which outputs the required the velocities to reach the desired point.

A. Simulation of goal-seeking behavior

For training a mobile robot to move from one position to another the agent needs to be supplied with a control policy that helps it to reach the goal. If there are no obstacles, it's relatively simple and easy to implement a traditional algorithm. Q learning is implemented as the control policy for our study. The three important parts of Q learning are states, actions and rewards. The whole state is divided into segments based on two parameters namely, distance from goal and orientation of the robot with respect to the goal position. Each variable is divided into three classes creating a total of nine classes. The goal position and the position that are irrelevant are considered as two states. Thus, a total of eleven states are created. The robot is able to do three kinds of actions:

(a) move forward by a distance 'd'

(b) rotate right by an angle 'θ'

(c) rotate left by an angle 'θ'

Finally, a reward system is established for rewarding the actions taken by the robot. The robot is given a high positive reward for reaching the goal. To ensure that the robot reaches the goal position in a minimum number of steps the robot is given a small negative reward for each step taken. If we give a positive reward, the robot will try to maximize the number of steps so it will follow a spiral path. Negative reward is given for reducing distance towards the goal position. Also, to restrict the domain in which the robot moves it is given a high negative reward if the distance from the goal position to robot position crosses 2500 mm. The reward values for different actions are shown in Table I.

TABLE I. REWARD SYSTEM

Actions	Reward value
Reaching Goal	+100
Reduction distance with goal	-1
Increase in distance with goal	-3
Facing the goal	-1
Facing other direction	-3
Moving out of domain space	-100

Reward policy of the robot can be further improved by giving it a slight negative reward for facing the goal position and higher negative reward for facing other directions. For goal-seeking behavior, the robot was trained for around 30 iterations to give good results. The results for the simulation of a robot moving from the point (200,200) to (700,700) is shown in Fig. 4.

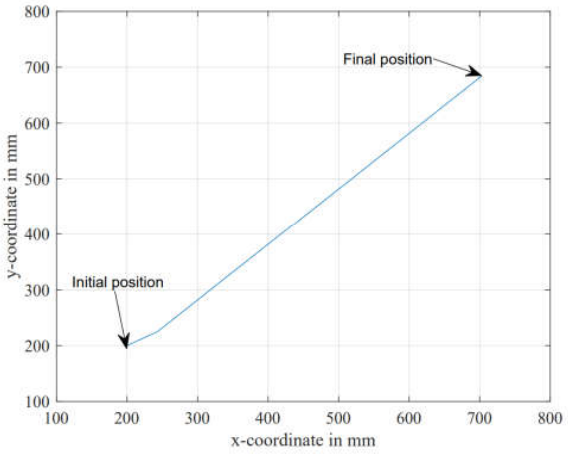

Fig. 4. Simulation of goal-seeking behavior

B. Simulation of obstacle avoidance behavior

Obstacle avoidance behavior should be combined with goal seeking behavior for autonomous navigation. This can be done in two ways. Separate RL agents can be created to control each of the behavior or single RL agent can be used by updating the reward policy accordingly. The latter mentioned method is used here. Here, the state space is defined by four parameters namely distance from goal position, distance from the obstacle, the orientation of robots towards goal position and the orientation of robots towards obstacle position. Each variable is divided into three classes generating 81 states. One state for goal position and one for any irrelevant cases gives a total of 83 states. If there is more than one obstacle, only the nearest obstacle facing the robot is considered. The possible set of actions remains three same as that used earlier. The reward system is modified by keeping all the previous reward rules and adding some additional rules. The robot is given a high negative reward (-100) if the distance between the robot and obstacle is lesser than a specified distance. Additional reward rules can be defined based on the orientation of robot with respect to the obstacle i.e., a negative reward can be given for facing the obstacle. The coordinates of obstacles can be

added using the GUI created earlier or it can be randomly generated. The robot is said to be reached goal state when it is sufficiently close to the goal position. Here the robot is trained for about 180 iterations. The results for the simulation of a robot moving from the point (200,200) to (200,1200) is shown in Fig. 5. The robots were able to reach the goal position for most of the cases but the path followed was not always optimum

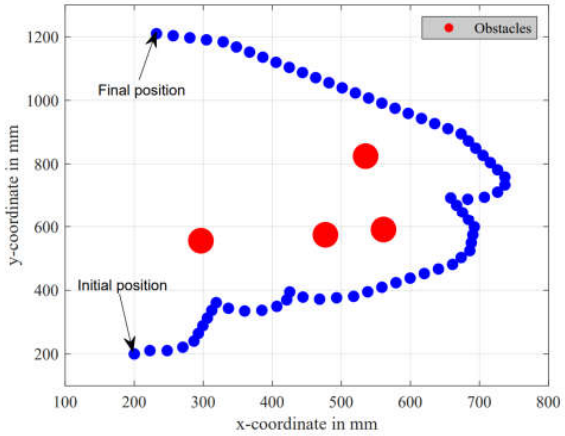

Fig. 5. Simulation of obstacle avoidance behavior

C. Simulation of kinematic controller

The simulations so far have generated only the waypoints. But for a practical application, knowing the next point is not enough, the system needs to also find out how much translational and rotational velocity needs to be applied at each instant of time to move from one waypoint to another. For this, the system needs to have a kinematic controller. So, the controller developed by [8] is integrated to the project. The values of k_1, k_2, k_3 are optimized. The user needs to supply the range of angular and linear velocities in which robot moves to the kinematic controller. Each time when the RL algorithm outputs the next waypoint, it is applied as the input to the kinematic controller to generate the velocities needed. The result of the controller implemented in single robot is shown in Fig. 6. Here the blue curve represents the curve generated by connecting waypoints and the red curve represents the output from the controller and the red circle are obstacles. There are some deviations of the controller trajectory from the RL algorithm trajectory during tight turns due to physical limitations of the robot.

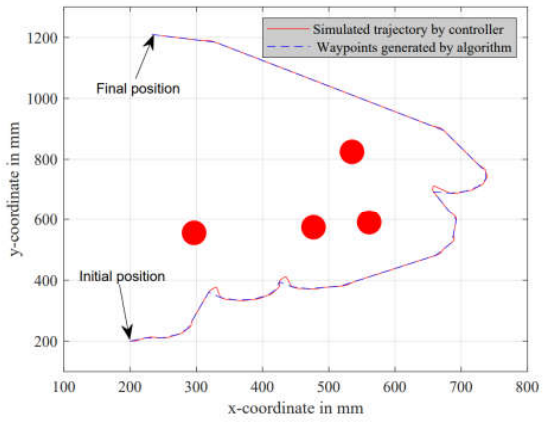

Fig. 6. Simulation of controller in a single robot

IV. Conclusion

A wide variety of algorithms have been developed for autonomous navigation of mobile robots. Recent trends suggest that there has been an increase usage of heuristic approaches over classical approaches. This is due to the fact that heuristic approaches are closer to the human way of behavior-based learning. In the current work RL method using Q learning is used for navigation of a mobile robot. Q learning proved to be an effective algorithm for navigation of a single robot as well as formation of robots. The algorithm once trained can function without any hiccups. The performance of the algorithm can also be increased while the system is on run. Results of the simulation of kinematic controller suggests that the combination of the kinematic waypoint tracking controller along with the Q learning based strategy is an effective way for obstacle avoidance and navigation of a mobile robot. If the number of states and actions is reduced by eliminating the redundant cases, the training time can be further reduced.

References

[1] V. Lumelsky and A. Stepanov, "Path planning strategies for point mobile automaton moving amidst unknown obstacles of arbitary shape," *Algorithmica,* pp. 403-430, 1987.

[2] J. Canny and J. Reif, "New lower bound techniques for robot motion planning problems," in *Proceedings of IEEE Symposium on the Foundations of Computer Science*, Los Angeles, 1987.

[3] N. Altuntas, E. Imal, N. Emanet and C. N. Ozturk, "Reinforcement learning-based mobile robot navigation," *Turkish Journal of Electrical Engineering & Computer Sciences,* vol. 24, no. 3, pp. 1747 -1767, 2016.

[4] H. R. Beom and H. S. Cho, "A Sensor-Based Navigation for a Mobile Robot Using Fuzzy Logic and Reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 25, no. 3, pp. 464-477, 1995.

[5] W.D. Smart and L. P. Kaelbling, "Effective Reinforcement Learning for Mobile Robots," in *Proceedings 2002 IEEE International Conference on Robotics and Automation*, Washington, DC, USA, USA, 2002.

[6] G. G. Yen and T. W. Hickey, "Reinforcement learning algorithms for robotic navigation in dynamic environments," *ISA Transactions,* no. 43, pp. 217-230, 2004.

[7] S. Sharma, "Autonomous Waypoint Generation with Safety Guarantees: On-Line Motion Planning in Unknown Environments," *arXiv preprint arXiv:1709.00546* (2017)

[8] J. Bruce, N. Suenderhauf, P. Mirowski, R. Hadsell and M. Milford, "One-Shot Reinforcement Learning for Robot Navigation with Interactive Repaly," in *Proceedings of the NIPS Workshop on Acting and Interacting in the Real World: Challenges in Robot Learning*, 2017.

[9] L. Tai, G. Paolo and M. Liu, "Virtual-to-real Deep Reinforcement Learning: Continuous Control of Mobile Robots for Mapless Navigation," in *Proceedings 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vancouver, BC, Canada, 2017.

[10] C. C. E. Chewu, and V. M. Kumar, "Autonomous navigation of a mobile robot in dynamic indoor environments using SLAM and reinforcement learning," in *Proceedings 2nd International Conference on Advances in Mechanical Engineering*, 2018.

[11]R. Mathew and S. S. Hiremath, "Control of Velocity-Constrained Stepper Motor-Driven Hilare Robot for Waypoint Navigation," *Engineering*, Vol. 4, pp. 1121-1126, 2008.