LSTM Based Dynamic Obstacle Avoidance

Srihari Subramanian, Vivek Mallampati, Manan Patel

***Abstract*—This research investigates the problem of dynamic obstacle avoidance on a robot with car-like dynamics using an LSTM neural network model. The inputs to the network are the LIDAR readings, the robots current location and goal location. The network outputs the probability of turning left or right which is then used to weight the action being taken given a maximum limit. Reward shaping was used as a reinforcement signal while training the network in a simulation. The weights were then updated using Adam optimizer. The network was able to reach the goal in the presence of no obstacles, however, the results were unsatisfactory in case of static and dynamic obstacles. The robot although avoided obstacles, was prone to being stuck in a local minima. One of the potential causes for this issue can be the hand-crafted dense reward function which might not capture complete information. Another cause might have been the rollout length since in most of the prior work, the roll out length was three orders of magnitude higher than the one considered here, which allows for more exploration and thus, would avoid converging to a local minima.**

***Index Terms*—Deep Reinforcement learning, Obstacle avoidance, and LSTM**

I. Introduction

With robots being deployed in people environment, it becomes pertinent to address the challenge of it safely maneuvering without collision with obstacles. The problem can be roughly divided into tow categories: global and local planners. The global techniques, such as road-map and cell decomposition require a complete model of the robot's environment is available. Since in most the cases the robot will encounter varying environment dynamics, these methods cannot be extended to a more general solution. However, one advantage of these methods is that the plan can be generated offline.

On the other hand, local planners such as dynamic window approach [1] and potential field based approach output a feasible trajectory given input measurements by optimizing over a cost function. However, when one considers a fairly cluttered environment with a plethora of moving pieces, one can imagine that it is not possible to come up with a generalised objective function over which to optimize which can be deployed in different environments. This is where a Neural Network based approach might prove to be effective given enough time and input data.

II. Prior Work

Deep Reinforcement Learning (DRL) was successful in providing solutions for the instability of function approximation techniques in reinforcement learning. DRLs have demonstrated that RL agents could be trained on raw, high-dimensional observations, solely based on reward functions or signals. [2] The idea of DRL is essentially using the neural networks for approximation of the value function and better updating of parameters in Q-learning. The research conducted in [2] uses a similar approach however, uses a feed forward network (in soft actor critic) along with a path planner to reach the goal. The path planner is deployed in this paper so as to avoid the robot from getting stuck in a local minima. Q-Learning is used by the author in [1] however, it assumes the velocity of any obstacle can be acquired through a camera.

LSTM is a recurrent neural network architecture, originally designed for supervised time-series learning, but the work in [3] showed that it can be used for reinforcement learning. The RL problems using a time series data can exploit the benefits of the LSTM and help the algorithm perform better. The "remembering" nature of LSTM, through the hidden states [4], helps long run RL agents to remember the state from a few time frames ago and improve its current performance.

There has been prior work to tackle the problem of dynamic obstacle avoidance. Seen from the research conducted in [5] and [6], RL algorithms are a stronger alternative to path planners. However, most of these approaches consider feed-forward networks and pass in a time series reading to implicitly encode the velocity information of the robot and the dynamic obstacles in the environment. This project on the other hand, will investigate an LSTM based

neural network to encode this information. Also, the dynamics considered in this project are those that mimic motion of an Ackermann-steering car. Therefore, a goal of this project is also to investigate the intricacies of physics with respect to the policies generated by the neural network.

III. Methodology

A. Background

Recurrent neural networks are useful in problems which require time series as a feature vector input. The key difference between a feed forward and RNN is the implementation of a hidden state. When a state is fed into an RNN, it generates the output vector and a hidden state vector. This hidden state vector is then used jointly with the next input state to predict the next output vector. The hidden state is thus useful in encoding information from past inputs to reason about the current input. However, one major issue faced with RNNs is the vanishing gradient problem. The hidden state is a representation of the information from all the previous states. Thus, with longer time series, it has less and less information about the previous states and thus the weight update with respect to information from the past is low.
An LSTM solves this problem by implementing a set of gates, which in themselves are neural networks, as mentioned in [3]. The input to an

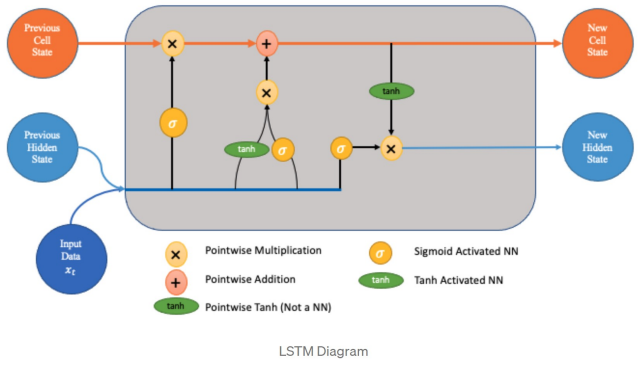

Fig. 1. LSTM Architecture and overview

LSTM is the current state, previous hidden state and the previous cell state as can be seen in the above fig 1. The first gate decides what information to retain by multiplying the sigmoid output (with previous hidden state and current state as input) and multiplying with the previous cell state. The next input gate and memory gate are responsible for determining what new information to add to the cell state. And finally, the new hidden state is created, which when passed through a linear activation function outputs the desired action to be taken.

B. Problem Description

1) ***Environment****:* Reinforcement learning is employed when transition dynamics are unknown. That is, the dynamics of the state space cannot be easily determined or simulated. Instead, the learner takes as input a policy rollout, which is an aggregation of decisions made by the agent in an iteration of the network. In this case, the rollout simulates the movement of the car as it explores the environment. The car employs Ackerman steering; the outer and inner wheels turn different angles to reduce wheel slippage as the car turns. The car's position relative to the world is defined as an x-position x, y-position y, and world heading θ, and turning angle ϕ. The kinematic update of such a vehicle is given by:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} \cos\theta \\ \sin\theta \\ \frac{1}{L}\tan\phi \\ 0 \end{bmatrix} u_1 + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u_2 \qquad (1)$$

Where $\dot{x}$ is the instantaneous velocity in the x-direction, $\dot{y}$ is the instantaneous velocity in the y-direction, $\dot{\theta}$ is the instantaneous angular velocity in the world frame, and $\dot{\phi}$ is the instantaneous angular velocity in the body frame of the car. Additionally, u_1 is the instantaneous acceleration $\dot{v}$ and u_2 is the instantaneous angular velocity $\dot{\phi}$.

The state vector consists of:
$< d_{LIDAR}, d_{goal}, d_{angle}, r_{pos} >$ where,
d_{LIDAR} = distance measurements from LIDAR
d_{goal} = distance to goal
d_{angle} = angle to goal
r_{pos} = co-ordinates of robot

The LIDAR measurements are used to implicitly encode the information about the dynamic objects in the environment. Robot position, distance to goal and angle to goal are inputs to further provide the network with detailed information about the direction in which to rotate.

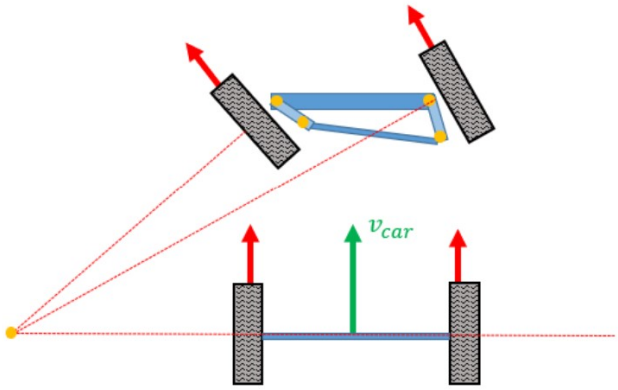

Fig. 2. Physics of a front-wheel drive car

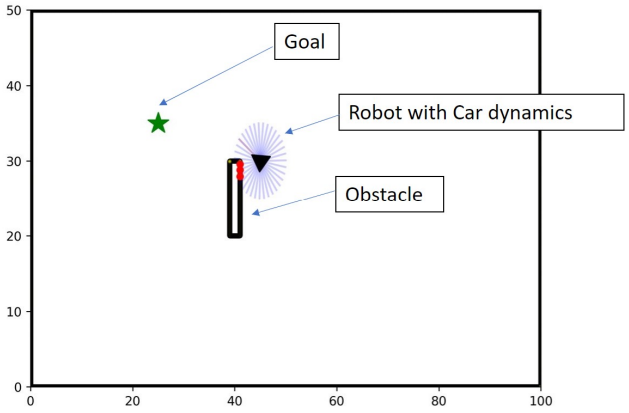

Fig. 3. Problem setup and environment

To make the problem simpler, the robot is provided with a constant velocity and thus only takes the decision of turning left and right using the neural network. However, to retain enough complexity, we decided to go with a hybrid approach in which we get the probabilities of turning left and right and the action is then a value which is a scaled difference in these probabilities. Thus, we can output a continuous action even when the decision made by the network is that of classification. The equation for the velocity given the probabilities is

$$v = P(L) * \dot{\phi}_{max} - P(R) * \dot{\phi}_{max}$$

where $\dot{\phi}_{max}$ is the maximum possible angular velocity of the car.

Fig 3 helps understand the setup better and also visualizes the different components of the system we are working on.

*2) **Reward function**:* There are two primary types of reward functions: dense rewards and sparse rewards. Sparse rewards are the primary rewards which are given to the agent when it completes or fails to do a task. This promotes high exploration as the agent is not focused on explicitly reaching the goal. Sparse rewards although being simple and easy to implement, have their drawback of requiring a high number of iterations. Dense rewards, on the other hand, are hand-crafted which allow the agent to be more focused towards achieving the goal. The agent in this case receives small positive and negative rewards with each action it takes while also receiving large primary rewards for completing or failing the task. Since most of the current work has been done using sparse reward functions, we wanted to investigate reward shaping and how it affects the policy of the robot. The reward function was developed by considering the following aspects:

- Primary rewards: The robot was given a high positive value of 10 when it reached the goal and a high negative value of -10 when it hit an obstacle or did not reach the goal.
- Avoiding obstacles: LIDAR readings in the front 180 degree field of view were considered. The change in LIDAR reading was used to give a positive or negative reward depending on whether the robot is moving towards or away from an obstacle. Each LIDAR reading (0-90 degree and 359-269 degree) was weighted with a Gaussian centered around 0 degree. Thus, obstacles directly in front of the robot would generate high negative reward (if motion is in their direction).
- Moving towards the goal: Distance from goal was used for this purpose. The robot was rewarded with a scaled value of change in distance from goal. This would encourage the robot to seek paths which lead towards the goal.

*3) **Neural Network**:* Since an LSTM can be used to encode time series information, we wanted to learn a model which would implicitly encode the motion of the robot and the obstacles to avoid collision and reach the goal. A single LSTM cell was used with Cross-entropy loss in PyTorch for our model. While generating the rollouts, the past 5 states were considered to generate the output of the current state. The output dimension of the network was set to two since the robot has to only decide whether to turn left or right.

IV. Results

A. Goal Reaching

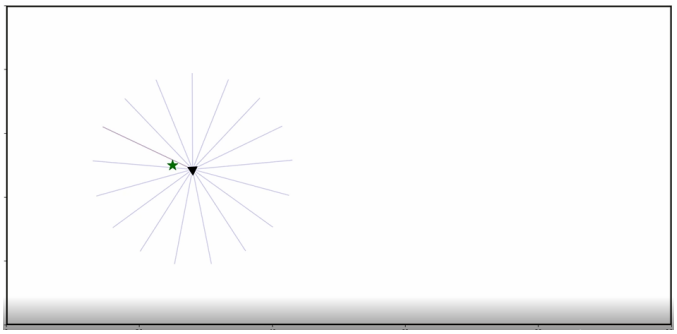

Fig. 4. Agent reaching the goal without an obstacle, given starting position and goal position

The agent was able to consistently reach the target after 8000 epochs (individual rollouts) when no obstacles were present in the environment. After training, the agent was also relatively successful in generalizing from various, but not most, starting configurations. The agent also seemed to overcompensate when aligning to the targer, and was unsuccessful in reaching the goal when oriented directly perpendicular to the target. Hyper-parameters significantly impacted the result of training. Specifically, the agent was very sensitive to changes to the learning rate and reward function scaling. These values could have been adjusted for better results. Another improvement would have been to use more varied training data.

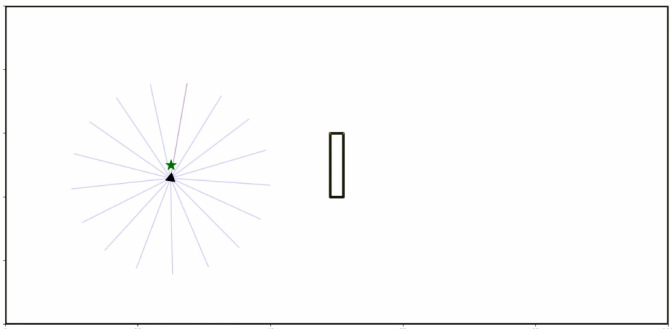

Fig. 5. Agent successfully reaches the goal while avoiding a static obstacle

In case of static obstacles, for some orientations, the agent was able to avoid the obstacle and reach the goal; however, often, after avoiding the obstacle, the agent experienced a local minima which manifested in the car constantly rotating in a circle. In this case the agent was unable to generalize to different environment configurations; the model was most likely over-fitted to the training data. It may also be an indication of a buggy reward function; spinning in a circle may result in a higher cumulative reward than reaching the goal, and thus the model prefers to fall into this state. Another observation is that sometimes, instead of accumulating negative rewards as the agent turns away from the target, it instead prefers to hit the obstacle. This results in a high negative reward, though lower than that of avoiding the obstacle entirely.

B. Obstacle avoidance

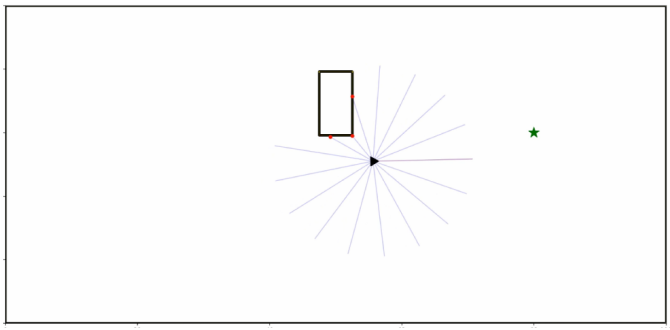

Fig. 6. Agent avoiding a dynamic obstacle

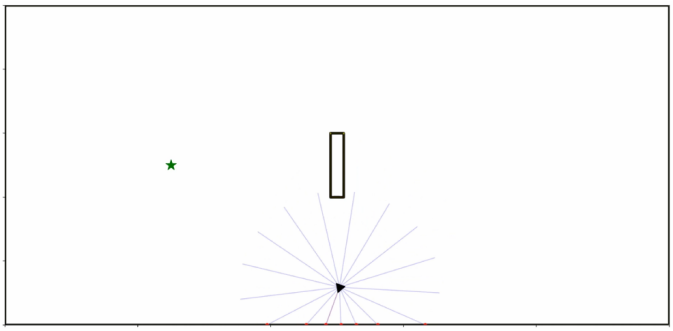

Fig. 7. Agent avoiding a static obstacle, but fails to reach goal

In specific environment configurations, the agent is able to avoid dynamic obstacles. It is once again able to reach the goal in training, but changing the configuration in testing results in failures. It seems to learn to avoid the obstacle at all costs; i.e., the model prioritizes actions that move the robot away from obstacles over those that move it towards the target. The most likely cause of this is that the cumulative reward for avoiding an obstacle is significantly greater than that of reaching the goal.

V. Discussion

A. Limitations and Future Work

A potential improvement would be to use a separate neural network for regulating the velocity of the robot. Since, currently, the velocity is constant, the physics of the robot might be difficult to train to avoid an obstacle while at the same time reach a goal. Also, having to approximate the utility function can be improved upon by having a separate Q network. Using the neural network to directly output the steering angle might have been a better approach than to generate a continuous action space by weighing the action with the difference in probability outputs specific algorithm and environment. Creating the reward function was a challenge and having a complicated function to start with might not have been the best way to go. Rather, starting with a simple reward function and then adding on to it would have helped understand the intricacies of dense reward functions. Finally, combining the LSTM architecture with AC or A2C algorithms where the critic helps the policy to converge to a stable optimal policy can be an improvement.

1) Parameter Tuning: The simplest yet the hardest part of implementing RL algorithms is hyperparameter tuning. Initial improvement strategy was to increase the number of epochs, which helped the model to converge in the simple test cases. Once no further improvement was detected, the number of epochs was held constant while changing the learning rate. After a few trials, the learning rate was set to an order of 10^{-6}. A few simulations showed that the robot was getting stuck in a local minima and moving in a circular path after avoiding an obstacle.

2) Algorithm: This paper implements a Reinforce algorithm using LSTM architecture for the neural network which apporximates the policy. The action taken by the agent in the rollout is sampled from the current policy so as to ensure exploration. The predicted label then becomes this sampled action and the actual label is the action which output the maximum probability. The Cross-entropy loss is then calculated and used to backpropogate and update the weights along the action which was taken.

VI. Conclusion

This paper shows the implementation of an LSTM neural network for obstacle avoidance in dynamic environments while trying to reach a goal. The simulation results show that the robot is able to reach the goal in the absence of obstacles but only sporadically in case of static and dynamic obstacles. In case of static obstacles, often after managing to avoid the obstacle, the agent fails to reorient itself towards the goal and converges to a local minima. The same issue occurs in case of dynamic obstacles. A higher rollout length with a simpler reward function might be able to solve this issues as it would promote more exploration and avoid the local minima.

References

[1] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[2] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[3] B. Bakker, "Reinforcement learning with long short-term memory," *Advances in neural information processing systems*, vol. 14, 2001.

[4] R. Dolphin, "Lstm networks: A detailed explanation." 2020.

[5] J. Choi, G. Lee, and C. Lee, "Reinforcement learning-based dynamic obstacle avoidance and integration of path planning," *Intelligent Service Robotics*, vol. 14, no. 5, pp. 663–677, 2021.

[6] J. Qiao, Z. Hou, and X. Ruan, "Application of reinforcement learning based on neural network to dynamic obstacle avoidance," in *2008 International Conference on Information and Automation*, 2008, pp. 784–788.