Worcester Polytechnic Institute

Reinforcement Learning CS525
Final Project Report

MOTION PLANNER

MOTION PLANNING OF TURTLEBOT USING DEEP REINFORCEMENT LEARNING

Author:
Adhitya Athyur Ganesh

Course Instructor:
Prof. Yanhua Li

December 11, 2019

Table of Contents

1. Introduction

a. Background

With the advancements in autonomous agents, motion planning i.e. planning the agent's motion so that they can move from one position to another in environments, involving static or dynamic obstacles, without any human intervention is still under debate. The objective of this tool is to make turtle robot capable to move in diverse environments by using reinforcement learning. The robot will be placed at random position and it has to reach the goal by analyzing the surrounding environment, inferring an action and executing it to maximize the total reward.

b. Robot

We are planning to use a turtlebot3 burger, a mobile robot with a Raspberry pi3 computer to carry out the tasks using a Realsense Kinect to detect the environment and return the states.

TurtleBot is a ROS standard platform robot. Turtle is derived from the Turtle robot, which was driven by the educational computer programming language Logo in 1967. urtleBot3 is a small, affordable, programmable, ROS-based mobile robot for use in education, research, hobby, and product prototyping. The goal of TurtleBot3 is to dramatically reduce the size of the platform and lower the price without having to sacrifice its functionality and quality, while at the same time offering expandability. The TurtleBot3 can be customized into various ways depending on how you reconstruct the mechanical parts and use optional parts such as the computer and sensor. In addition, TurtleBot3 is evolved with cost-effective and small-sized SBC that is suitable for robust embedded system, 360 degree distance sensor and 3D printing technology.

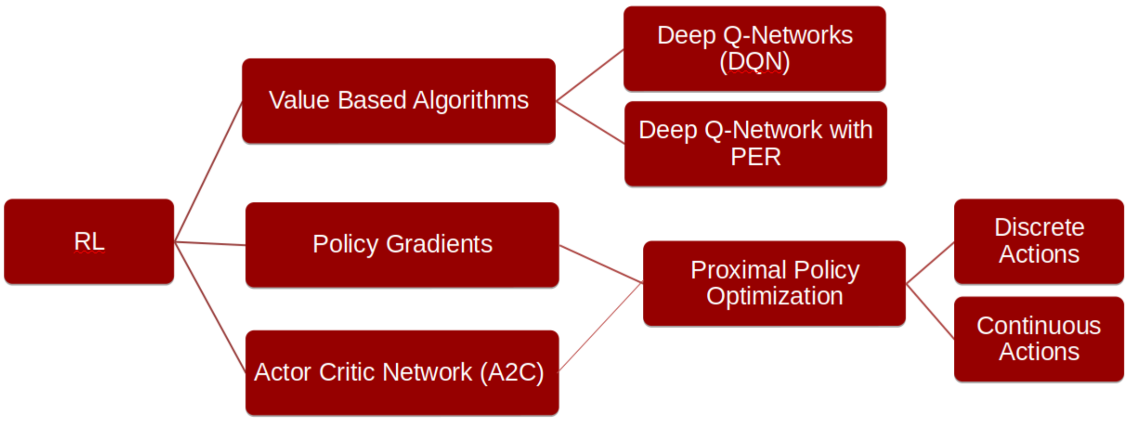
To detect the landmarks we will use fiducial markers . We are controlling the robot through ROS environment and we might also use the LIDAR for mapping in case it is needed. Initially we will feed a map to it to solve a motion planning problem. Later we will only use the feedback from the robot to get information about the environment.

c. Challenges

There are several challenges in this project. The main challenge is interacting hardware (i.e., RL programmed robot) with the surrounding environment. How to update its state and take action. 1. How to train? Because the robot should interact with the environment during the training 2. How define obstacles, 3. How to define the start and stop location for the robot, 4. How to convert the sparse reward problem to dense one 5. How to get information from the surrounding environment, e.g., using LIDAR, camera, map, etc.

2. Approaches

We implement different deep reinforcement algorithms to see which model learns quickly in motion planning domain. Deep Q-Learning along with prioritized experience was selected from the bin of Value Based algorithms considering their best performance for discrete actions. We combined the Proximal policy optimization, algorithm from policy gradient bin with Actor critic in the way that policy optimization of actor is done using PPO. We not only tried PPO-A2C with discrete actions but also with continuous action space.



a. DQN

Deep Q-network (DQN) is able to combine reinforcement learning with a class of artificial neural network known as deep neural networks. Notably, recent advances in deep neural networks, in which several layers of nodes are used to build up progressively more abstract representations of the data, have made it possible for artificial neural networks to learn concepts such as object categories directly from raw sensory data.Reinforcement learning is known to be unstable or even to diverge when a nonlinear function approximator such as a neural network is used to represent the action-value (also known as Q) function . This instability has several causes: the correlations present in the sequence of observations,the fact that small updates to Q may significantly change the policy and therefore change the data distribution, and the correlations between the action-values(Q) and the target values. DQN address these instabilities with a novel variant of Q-learning,which uses two key ideas. First, DQN uses a biologically inspired mechanism termed experience replay that randomizes over the data, thereby removing correlations in the observation sequence and smoothing over changes in the data distribution. Second, it uses an iterative update that adjusts the action-values (Q) towards target values that are only periodically updated,thereby reducing correlations with the target.

We used DQN with following configurations:

- Number of episodes: 2000
- Discount factor: 0.99
- Learning rate: 0.00025
- Epsilon decap: 0.99
- Batch size: 64
- Replay memory: 1e+6

Network Architecture for our policy and target network is as follows:

Layer Name	Layer Type	Number of Neurons	Activation
Input	Dense	26	-
FC1	Dense	64	ReLU
FC2	Dense	64	ReLU
Output	Dense	5	-

b. DQN with PER

Prioritized Experience Replay (PER) was introduced in 2015. The idea is that some experiences may be more important than others for our training, but might occur less frequently.
Because we sample the batch uniformly (selecting the experiences randomly) these rich experiences that occur rarely have practically no chance to be selected. That's why, with PER, we try to change the sampling distribution by using a criterion to define the priority of each tuple of experience. We want to take in priority experience where there is a big difference between our prediction and the TD target, since it means that we have a lot to learn about it. We use the absolute value of the magnitude of our TD error: $p_t = |delta_t| + e$. And we put that priority in the experience of each replay buffer. We implemented Prioritized experience replay with dqn in the same configurations as above.

c. Proxy Policy Optimization with Actor Critic

Policy gradient methods are fundamental to recent breakthroughs in using deep neural networks for control, from video games, to 3D locomotion, to Go. But getting good results via policy gradient methods is challenging because they are sensitive to the choice of the step size — too small, and progress is hopelessly slow; too large and the signal is overwhelmed by the noise, or one might see catastrophic drops in performance.
With supervised learning, we can easily implement the cost function, run gradient descent on it, and be very confident that we'll get excellent results with relatively little hyperparameter tuning. The route to success in reinforcement learning isn't as obvious — the algorithms have many moving parts that are hard to debug, and they require substantial effort in tuning in order to get good results. PPO strikes a balance between ease of implementation, sample complexity, and ease of tuning, trying to compute an update at each step that minimizes the cost function while ensuring the deviation from the previous policy is relatively small. We implemented PPO by incorporating it in A2C where actor part is optimized using PPO. We trained our model for 2000 episodes and actor/critic network is updated after 100 steps for 4 epochs. We have used Q-value minus value of state as advantage function. However, the network architecture differs for discrete and continuous action space.

I. Discrete Action Space

For discrete actions, the network output logits which are then mapped to categorical distribution and action is then sampled from this distribution

Layer Name	Layer Type	Number of Neurons	Activation
Input	Dense	26	-
FC1	Dense	64	ReLU
FC2	Dense	64	ReLU
Output (Value)	Dense	1	-
Output (Logits)	Dense	5	-

II. Continuous Action Space:

For continuous action space, the network output mean in range of (-1,+1). This mean is used to sample action from normal distribution with standard deviation of 0.6.

Layer Name	Layer Type	Number of Neurons	Activation
Input	Dense	26	-
FC1	Dense	64	ReLU
FC2	Dense	64	ReLU
Output (Value)	Dense	1	-
Output (Mean)	Dense	1	-

3. Experiments:

a. Simulations

We used 3D robot simulator Gazebo for planning the motion of turtlebot3 burger. Environment consists of four static obstacles and walls and model is trained for 2000 episodes. Each episode terminates either robot roams for more than 500 steps or collides with an obstacle.

I. State space

State is described by 28 features in which we had 24 Laser Distance Sensor (LDS) values, distance and angle to goal and minimum range and angle to obstacle

II. Action Space

- Discrete: The actions range from 0 - 4. and corresponds to either increase/decrease in angular velocity.

Action	Angular Velocity
0	-1.5
1	-0.75
2	0
3	0.75
4	1.5

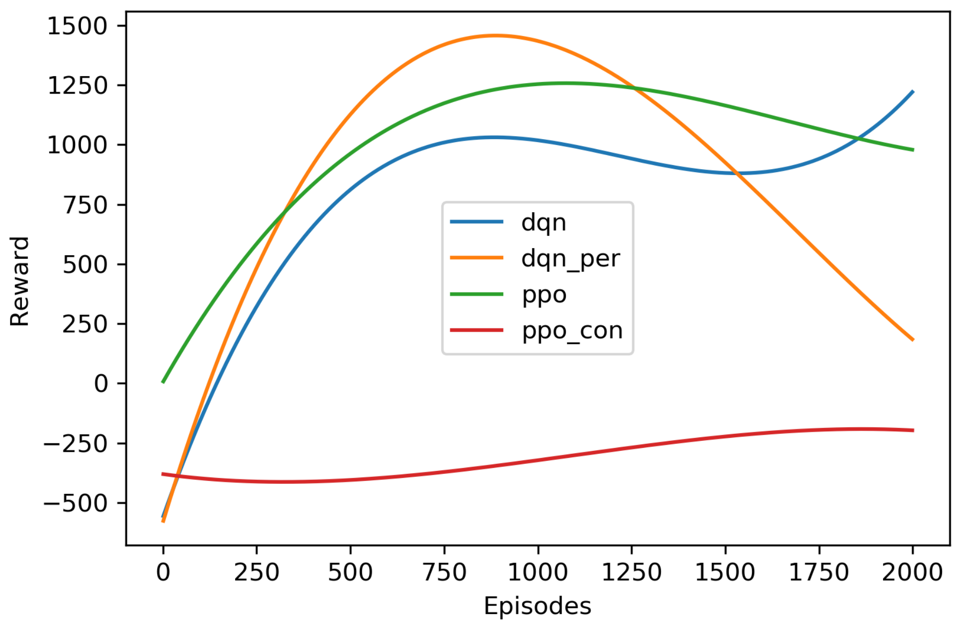
- Continuous: Actions are angular velocity ranging from -2 to +2.

III. Reward

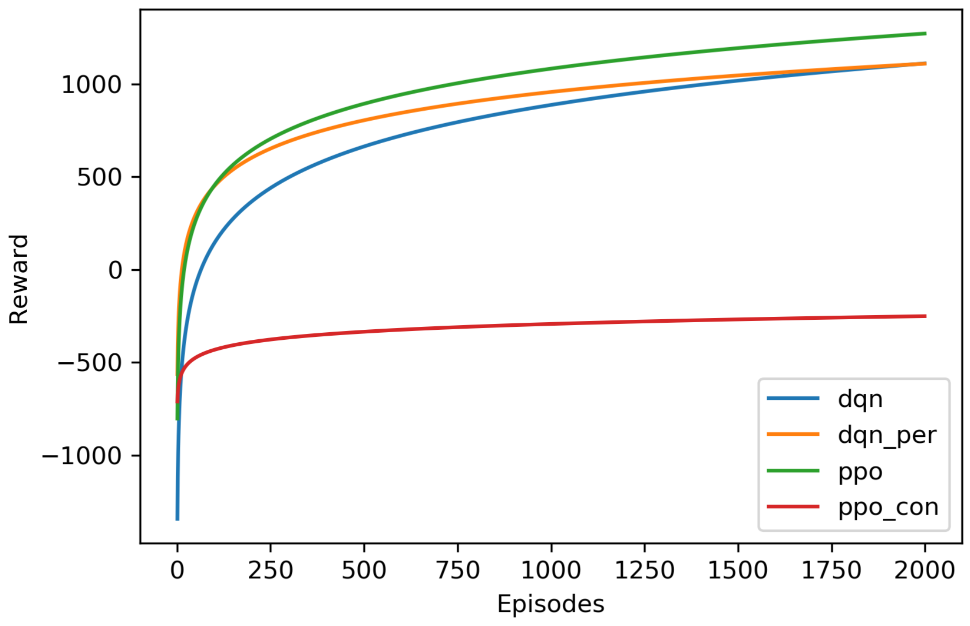
This motion planning is a sparse reward problem in which one can reward the agent on reaching the goal while penalize it on colliding with any obstacle. The sparse reward problems always take time to converge. Considering this, we changed this sparse reward to dense one that gives negative reward on colliding with the obstacle and positive on reaching the goal. Apart from this, agent receives positive reward on heading towards the goal and negative on moving away from it. This reward depends on the distance of the robot from goal position.

$$\text{Reward} = \begin{cases} 200 & \textit{On reaching Goal} \\ -150 & \textit{On colliding with Obstacle} \\ \textit{Goal\_dist + dist\_angle\_to\_goal} & \textit{If robot roams freely in environment} \end{cases}$$

b. Results



The reward over 2000 episodes by cubic polynomial fit



The reward over 2000 episodes by log fit

4. Analysis and Conclusion

In this project, as a first step, the behavior of different reinforcement learning algorithms in discrete space, i.e., DQN, DQN with PER, A2C with Proximal Policy Optimization on robot path planning are investigated. The results are as follows:

- ★ A2C with Proximal Policy Optimization (PPO) and discrete actions outperforms rest of the algorithms. PPO allows the policy update without much deviation from the previous one, thus it emerges as one of the most stable algorithms.
- ★ DQN with PER had a better performance than DQN but unexpectedly both converged to the same reward after 2000 episodes which can be due to this fact that if samples are independent or at least less correlated, algorithm will learn faster, but it seems that its impact on the final result is not too high and negligible.
- ★ A2C with Proxy policy optimization was also implemented with continuous action space. Since, the action space is much bigger than the discrete space which results in the learning time for this algorithm be much longer. Therefore, even though the discrete algorithm can do the job perfectly after 2000 episodes, agent in this algorithm only learned how to avoid negative reward.

5. Future Work

- ★ The continuous PPO-A2C with continuous action space algorithm did not give satisfactory results in the beginning. There are various reasons which might have caused this. We will try and rectify the setbacks and generate a better algorithm

- ★ We can change the static obstacles to dynamic ones in order to make the motion planner more robust.

- ★ We may try to implement model on other algorithms like Deep Deterministic Policy Gradients(DDPG) as it also deals with continuous spaces well.

- ★ We can deploy this trained model into a physical robot and evaluate its efficiency.

References

1. http://emanual.robotis.com/docs/en/platform/turtlebot3/ros2_machine_learning/
2. Zeng, Taiping. (2018). Learning Continuous Control through Proximal Policy Optimization for Mobile Robot Navigation.
3. Nihal Altuntas¸, Erkan Imal, Nahit Emanet, and Ceyda Nur Ozt¨urk, “Reinforcement learning-based mobile robot navigation”, Turkish Journal of Electrical Engineering & Computer Sciences 24 (2016), no. 3, 1747–1767.
4. https://blog.varunajayasiri.com/ml/ppo_pytorch.html
5. Tom Schaul, John Quan, Ioannis Antonoglou, David Silver, “*Prioritized Experience Replay*,” arXiv:1511.05952v4 [cs.LG] 25 Feb 2016.
6. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al., Human-level control through deep reinforcement learning, Nature 518 (2015), no. 7540, 529
7. https://openai.com/blog/openai-baselines-ppo/