

Implementation of Q Learning and Deep Q Network For Controlling a Self Balancing Robot Model

MD Muhaimin Rahman^{1,a}, SM Hasanur Rashid^{2,a} and M.M Hossain^b

^a Department of Mechanical Engineering, ^bDepartment of Electrical and Electronic Engineering
Bangladesh University of Engineering & Technology
Email:¹sezan92@gmail.com
²hrshovon@gmail.com
³monir.eee.buet@gmail.com

Abstract— In this paper, the implementation of two Reinforcement learnings namely, Q Learning and Deep Q Network(DQN) on a Self Balancing Robot Gazebo model has been discussed. The goal of the experiments is to make the robot model learn the best actions for staying balanced in an environment. The more time it can stay within a specified limit, the more reward it accumulates and hence more balanced it is. Different experiments with different learning parameters on Q Learning and DQN are conducted and the plots of the experiments are shown.

I. INTRODUCTION

Control systems is one of the most important aspects of Robotics Research. Gazebo is one of the most robust multi robot simulators at present. The ability to use Robot Operating System (ROS) with Gazebo makes it more powerful. But there are very few documentations on how to use ROS and Gazebo for Controllers development. In our previous paper, [1], we attempted to demonstrate and document the use of PID, Fuzzy logic and LQR controllers using ROS and Gazebo on a self balancing robot model. Later on, we have worked on Reinforcement learning. In this paper, implementation of Q Learning and Deep Q Network on the same model is discussed. The paper is structured as follows. Section shows the related works on the subject. Section discusses the Robot Model. Section shows the Implementation of Q Learning and DQN as controllers. Finally, section is the conclusion.

II. RELATED WORKS

Lei Tai and Ming Liu [2], had worked on Mobile Robots Exploration using CNN based reinforcement learning. They trained and developed turtlebot Gazebo simulation to develop an exploration strategy based on raw sensor value from RGB-D sensor. The company *ErleRobotics* have extended OpenAI environment to Gazebo [3]. They have deployed Q-learning and Sarsa algorithms for various exploratory environments. Loc Tran et al [4] developed training model for an Unmanned aerial vehicle to explore with static obstacles in both Gazebo and real world. Their proposed Reinforcement learning is unclear from the paper. Volodymyr Sereda [5] used Q-learning on a custom Gazebo model using ROS in for exploration strategy. Rowan Border [6] used Q-learning with

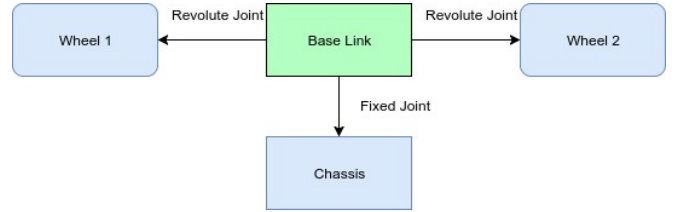


Fig. 1: Simple Block Diagram of the Model

neural network presentation for robo search and rescue using ROS and Turtlebot.

III. ROBOT MODEL

The Robot Model is described in paper [1]. It has one chassis and two wheels. The task of the model is to keep the robot balanced i.e. keeping its pitch angle in between $\pm 5^\circ$. The more it remains in between the limits, the more it gets the reward. The Fig. 1 shows the block diagram and the Fig.2 shows the Gazebo model.

A. Controller

The robot's IMU sensor measures the roll, pitch and yaw angles of the chassis every second and sends it to the controller. The controller calculates optimum action value to make the chassis tilt according to set point. Fig. 3 shows the control system of the robot.

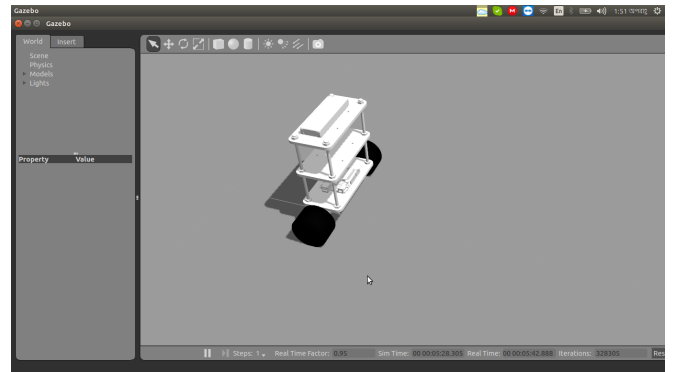


Fig. 2: Gazebo Model

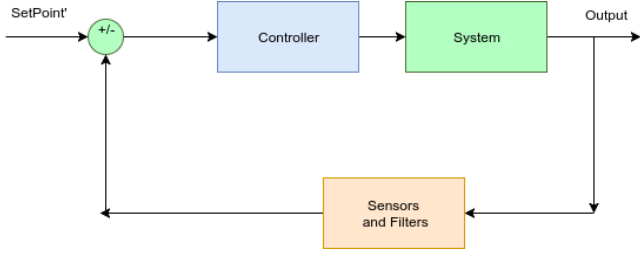


Fig. 3: Controller Block Diagram

IV. REINFORCEMENT LEARNING METHODS AS CONTROLLERS

In [1], we worked on traditional Controllers like PID, Fuzzy PD, PD+I & LQR. The biggest problem with those methods is that, they need to be tuned manually. So, reaching optimal values of Controllers depends on many trials and errors. Many a times optimum values aren't reached at all. The biggest benefit of Reinforcement learning algorithms as Controllers is that, the model tunes itself to reach the Optimum values. The following two sections discuss Q Learning and Deep Q Network.

A. Q Learning

Q-learning was developed by Christopher John Cornish Hellaby Watkins [7]. According to Watkins, "it provides agents with the capability of learning to act optimally in Markovian domains by experiencing the consequences of actions, without requiring them to build maps of the domains." [8]. In a Markovian domain, Q function- the model to be generated using the algorithm- calculates the expected utility for a given finite state s and every possible finite action a . The agent - which is the robot in this case- selects the optimum action a having the highest value of $Q(s, a)$, this action choosing rule is also called Policy. [8]. Initially, the $Q(s, a)$ function values are assumed to be zero. After every training step, the values are updated according to the following equation

$$Q(s, a_t) \leftarrow Q(s, a_t) + \alpha(r + \gamma \max_a Q(s_{t+1}, a)) \quad (1)$$

The objective for the Model in our project is to keep it within limits i.e. $\pm 5^\circ$. At first, the robot model, Q matrix, Policy π are initialized. There are some interesting points to make. The states are not finite. Within limit range, hundreds and thousands of pitch angles are possible. Having thousands of columns is not possible. So, the state values were discretized. We discretized the values to 20 discrete state angles from -10° to 10° . For action value, we chose 10 different velocities. They are $[-200, -100, -50, -25, -10, 10, 25, 50, 100, 200] m s^{-1}$. The Q matrix had 20 columns, each column representing a state and 10 rows each representing every action. Initially, the Q -values were assumed to be 0 and random actions were specified for every state in the policy π . The training was done for 1500 episodes and in each episode, the training

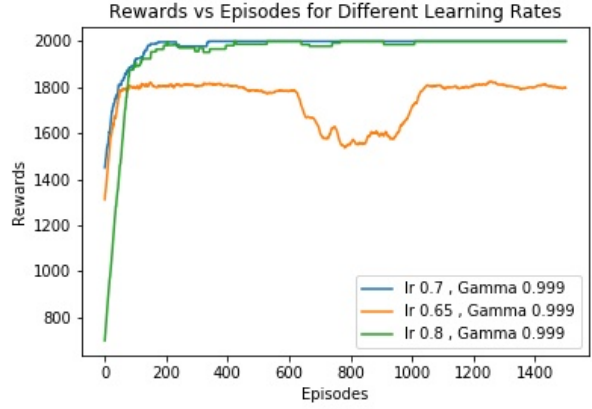


Fig. 4: Rewards for different α

was iterated 2000 times. At the beginning of each episode, the simulation was refreshed. Whenever the robot's state exceeded the limit it was penalized by assigning reward to -100 . The Q Table is updated at each step according to equation 1. The Algorithm 1 shows the full algorithm.

The simulation was run for three different α values (0.7, 0.65, 0.8), with γ value of (0.999). The Fig. 4 shows the Rewards vs Episodes for those α s. It is evident that, the robot couldn't reach the target rewards within the training period for those learning rates. We see that, for the α values 0.7 and 0.8, the robot reaches maximum possible accumulated rewards, 200, within 400 episodes. The curve with α value 0.7 is less stable compared to that of 0.8. But The curve with α value 0.65 never reaches the maximum accumulated reward.

B. Deep Q Network (DQN)

V Mnih et al [9] first used Deep Learning as a variant of Q Learning algorithm to play six games of Atari 2600, which outperformed all other previous algorithms. In their paper, two unique approaches were used.

- Experience Replay
- Derivation of Q Values in one forward pass

The technique of Experience Replay, experiences of an agent, i.e. $(state, reward, action, state_{new})$ are stored over many episodes. In the learning period, after each episode random batches of data from experience are used to update the model. [9]. There are several benefits of such approach. According to the paper,

- It allows greater data efficiency as each step of experience can be used in many weight updates
- Randomizing batches breaks correlations between samples
- Behaviour distribution is averaged over many of its previous states

In the classical Q learning approach, one has to give state and action as an input resulting in Q value for that state and action. Replicating this approach in Neural Network is problematic. Because in that case one has to give state and

Algorithm 1: Q Learning Algorithm as applied in the system

```
Initialize Robot;
Initialize Q Matrix  $Q$ ;
Initialize Policy  $\pi$ ;
Initialize Penalty Reward  $pen$ ;
for number of episodes do
  Reset simulation ;
  Wait for 1 second ;
  Pause simulation ;
  Read the pitch angle  $\phi$  of the robot ;
   $state \leftarrow \phi$  ;
  Unpause simulation ;
  for number of iterations do
    Generate a random number  $rand$ ;
    if  $rand \leq \epsilon$  then
      take random action ;
    end
    else
      take action based on  $\pi$  ;
    end
     $state_{new} \leftarrow \phi$ ;
    Pause simulation;
    if absolute value of  $state_{new} \geq limit$  then
      if  $reward_{total} \leq Target$  then
         $reward \leftarrow pen$ ;
        Update  $Q$  ;
        Update  $\pi$ ;
      end
      Break ;
    else
      Print Passed ;
      Break ;
    end
  end
  else
     $reward \leftarrow 1$ ;
    Update  $Q$ ;
    Update  $\pi$   $state \leftarrow state_{new}$ 
  end
end
```

action for each possible action of the agent to the Model. It will lead to many forward passes in the same model. Instead, they designed the model in such a way that it will predict Q values for each action for a given state. As a result, only one forward pass is required. Figure 5 shows a sample architecture for one state with two actions

The implementation of the DQN on our Robot model is similar to Q Learning Method. However, there are some exceptions. At first, a model was initialized instead of Initializing Q matrix. In the ϵ greedy policy, instead of choosing action based on policy π , Q values were calculated

Algorithm 2: DQN Algorithm as applied in the system

```
Initialize Robot;
Initialize model  $M$ ;
Initialize Penalty Reward  $pen$ ;
for number of episodes do
  Reset simulation ;
  Wait for 1 second ;
  Pause simulation ;
  Read the pitch angle  $\phi$  of the robot ;
   $state \leftarrow \phi$  ;
  Unpause simulation ;
  for number of iterations do
    Generate a random number  $rand$ ;
    if  $rand \leq \epsilon$  then
      take random action ;
    end
    else
       $Q \leftarrow M(state)$ ;
       $action \leftarrow action_{formax}(Q)$ ;
    end
     $state_{new} \leftarrow \phi$ ;
    Pause simulation;
    if absolute value of  $state_{new} \geq limit$  then
      if  $reward_{total} \leq Target$  then
         $reward \leftarrow pen$ ;
         $experience \leftarrow (state, reward, action, state_{new})$ ;
        Add Experience to Memory;
      end
      Break ;
    else
      Print Passed ;
      Break ;
    end
  end
  else
     $reward \leftarrow 1$ ;
     $experience \leftarrow (state, reward, action, state_{new})$ ;
    Add Experience to Memory;
     $state \leftarrow state_{new}$ 
  end
end
  Take random minibatch of Experience;
  if  $reward == pen$  then
     $Q_{pred} \leftarrow reward$ ;
  end
  else
     $Q_{pred} \leftarrow reward + \gamma max(Q(state_{new}, action))$ 
  end
  ;
  Train the model according to loss
   $abs(Q_{pred}(state, action) - Q_{pred}(state, action))$ ;
end
```

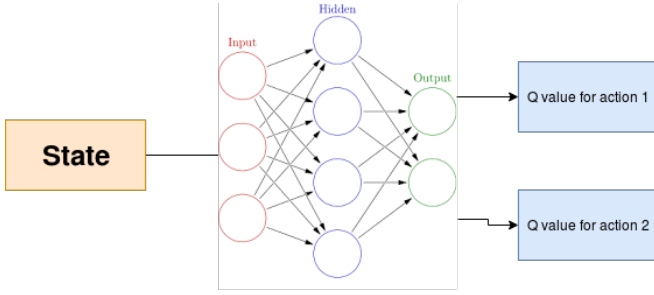


Fig. 5: Sample Deep Q Network Architecture

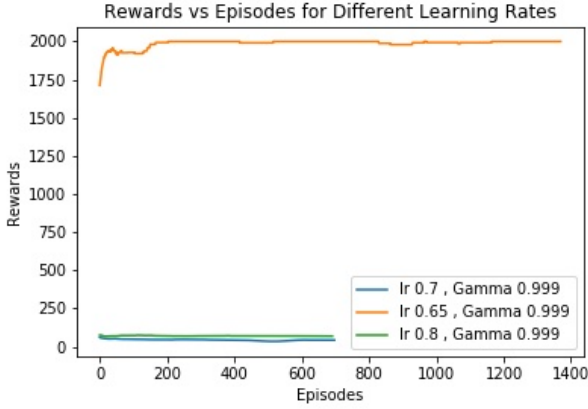


Fig. 6: "Rewards for three different α s with γ 0.999"

according to the model. At the end of every episode, the model was trained using random minibatches of Experience. At first, an architecture with 2 hidden Relu layers of 20 units was selected. The last layer was a Linear Dense layer with 10 units. With the γ 0.999 and α of (0.65, 0.7, 0.8) . Algorithm 2 shows the DQN algorithm as implemented on the robot model.

From figure 6, we see that the total rewards for α 0.65 is significantly higher. It starts approximately from 1750 and reaches the maximum total rewards, 2000 within 200th episode. But the accumulated rewards with α values of 0.7 and 0.8 are very low. They have accumulated rewards approximately 50-60 for the whole time. Later, the architecture was changed to 2 hidden layers of 40 Relu Units. The γ was selected to be 0.9. Figure 7 shows that the both curves reached highest accumulated rewards within 200 episodes in the new configuration.

V. COMPARISON TO TRADITIONAL METHODS

In our previous paper, [1], we compared PID, Fuzzy Logic and LQR. The Fig. 8 shows the performance curves for different controllers. It shows that LQR and Fuzzy controllers were not so stable like PID. Although we had to tune all of them manually. But the Fig. 7 shows that DQN learning is more stable after some iterations.

VI. CONCLUSION AND FUTURE WORK

The implementation of Q Learning and Deep Q Network

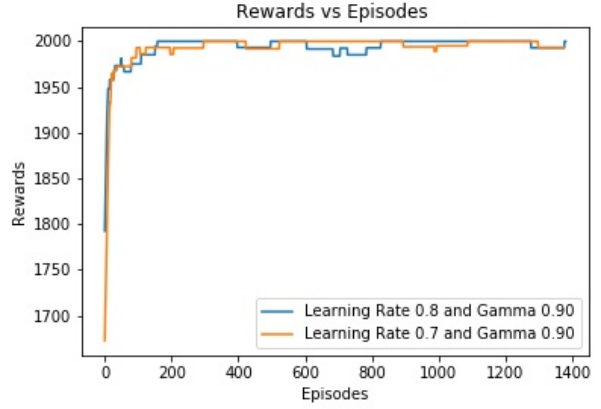


Fig. 7: Rewards vs Episodes for New Architecture

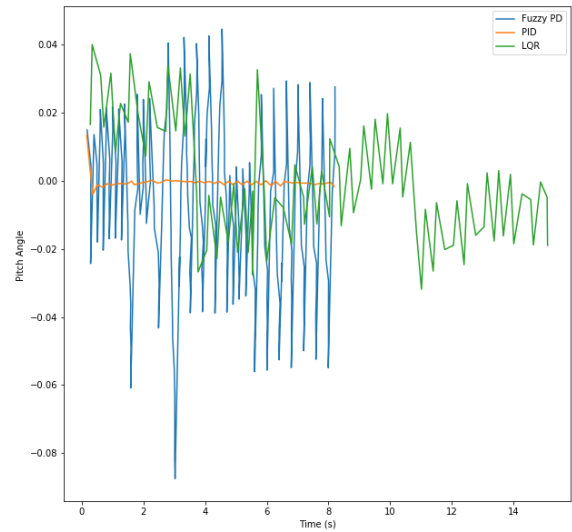


Fig. 8: Performance curve for PID, Fuzzy logic and LQR

as a controller in the Gazebo Robot Model was shown in this paper. It showed the details of the algorithms. However, some further improvements can be done. Like, It was assumed that the robot will work on Markovian State space, which generally not the case. In general Inverted pendulum models are Non-markovian models. So there must exist some kind of dependancies among the states. So In future, Recurrent Neural Network have a great possibility. Moreover, 10 predefined values of velocities for action were used. In the real world application, action values have continuous range. So for more complex models, this method may not work. In that case, deep reinforcement learning algorithms with continuous action space like Actro Critic Reinforcement Learning algorithm [10] can be used. Finally, this work should be improved for real world scenarios.

VII. DECLARATIONS

A. Competing Interests

The Authors declare that they have no competing interests

B. Author's contribution

The original project is this paper and [1]. The Contributions of MD Muhaimin Rahman is the simulations and writing of this paper. The contributions of SM Hasanur Rashid and M.M. Hossain is reviewing both papers.

C. Funding

The paper has no external source of funding.

REFERENCES

- [1] M. D. M. Rahman, S. M. H. Rashid, K. M. R. Hassan, and M. M. Hossain, "Comparison of different control theories on a two wheeled self balancing robot," *AIP Conference Proceedings*, vol. 1980, no. 1, p. 060005, 2018. [Online]. Available: <https://aip.scitation.org/doi/abs/10.1063/1.5044373>
- [2] L. Tai and M. Liu, "Mobile robots exploration through cnn-based reinforcement learning," *Robotics and Biomimetics*, vol. 3, no. 1, p. 24, Dec 2016. [Online]. Available: <https://doi.org/10.1186/s40638-016-0055-x>
- [3] I. Zamora, N. G. Lopez, V. M. Vilches, and A. H. Cordero, "Extending the openai gym for robotics: a toolkit for reinforcement learning using ROS and gazebo," *CoRR*, vol. abs/1608.05742, 2016. [Online]. Available: <http://arxiv.org/abs/1608.05742>
- [4] L. D. Tran, C. D. Cross, M. A. Motter, J. H. Neilan, G. Qualls, P. M. Rothhaar, A. Trujillo, and B. D. Allen, "Reinforcement learning with autonomous small unmanned aerial vehicles in cluttered environments," *15th AIAA Aviation Technology, Integration, and Operations Conference*, Jun 2015. [Online]. Available: <https://doi.org/10.2514/6.2015-2899>
- [5] V. Sereda, "Machine learning for robots with ros," Undergraduate thesis, Maynooth University, Maynooth, Co. Kidare, 2017.
- [6] R. Border, "Learning to save lives: Using reinforcement learning with environment features for efficient robot search," White Paper, University of Oxford, 2015.
- [7] C. J. Watkins, "Learning from delayed rewards," Ph.D. dissertation, Kings's College, London, May 1989.
- [8] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992. [Online]. Available: <https://doi.org/10.1007/BF00992698>
- [9] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: <http://arxiv.org/abs/1312.5602>
- [10] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1928–1937. [Online]. Available: <http://proceedings.mlr.press/v48/mniha16.html>