International Research Journal of Computer Science (IRJCS)
Issue 09, Volume 07 (September 2020)
ISSN: 2393-9842
https://www.irjcs.com/archives

CONTROL & NAVIGATION IN ROBOTS USING REINFORCEMENT LEARNING

Palepu Jithin kumar
Dept of ECE, Sharda University, Noida, India
jithink385@gmail.com
Bahurothu Venkata Hemanth
Dept of ECE, Sharda University, Noida, India
hemanthbahuroth@gmail.com
Pallavi Gupta
Dept of ECE, Sharda University, Noida, India
pallavi.gupta2@sharda.ac.in

Publication History
Manuscript Reference No: IRJCS/RS/Vol.07/Issue09/SPCS10087
Received: 02, September 2020
Accepted: 12, September 2020
Published: 22, September 2020
DOI: https://doi.org/10.26562/irjcs.2020.v0709.006
Citation: Palepu,Bahurothu,Pallavi (2020). Control & Navigation in Robots Using Reinforcement Learning. IRJCS:: International Research Journal of Computer Science, Volume VII, 262-268.
https://doi.org/10.26562/irjcs.2020.v0709.006
Peer-review: Double-blind Peer-reviewed
Editor: Dr.A.Arul Lawrence Selvakumar, Chief Editor, IRJCS, AM Publications, India

Abstract: Reinforcement learning (RL) is a subfield of machine learning which is being developed in Artificial Intelligence (AI). This technique is a data independent process. The primary aim of systems this kind is to maximize their reward signal which makes systems do better things trending to goal. Reinforcement Learning alters with techniques like supervised and unsupervised in such a way that in RL the agent gets up with its own insights and maps what action to perform in certain situations. On the other hand, Supervised and unsupervised have answers already embedded in them. In RL, in absence of new data, it can learn from its own experience where others can do. RL is used almost everywhere, the best applications of RL in Robotics specifically in motion control, planning it is also used in finance, gaming etc. Here is this paper demonstrating the navigation and motion control development of a 2 wheeled differential drive robot with the help of reinforcement learning topology. Traditionally, to design the behaviour of controllers in robots, we inevitably need models of how the robot actually behaves in the environment. But here we come up with a RL approach to design the control structure for the robot to navigate in the indoor environment.

I. INTRODUCTION

Reinforcement learning is extremely flexible with problems, especially; it specializes in solving general problems by coming up with general solutions. Thus, making it used in fields ranging from games to agriculture. In industrial applications, RL is seeing use in large scale systems such as data center cooling. One most promising application of RL is in robotics, to be specific, localizing itself, navigation, control without getting hit with obstacles play a critical role in mobile robots. In real-life applications the knowledge about environments is very limited and harder to fetch every detail in it as they have intrinsic stochasticity. Mostly in motion planning, navigating robots from one arbitrary position to another goal position without getting in touch with the obstacles on the way is perused. For mobile robots such as holonomic and nonholonomic, traditional methods such as SLAM (Simultaneous Localization and mapping) are widely used which is a clear and easy approach. In SLAM, a map is drawn by the robot by navigating to all the observations manually, this will take time in exploring the environment the robot is in and it is completely dependent on the sensor in order to get a local and global costmap prediction.

II. METHODOLOGIES

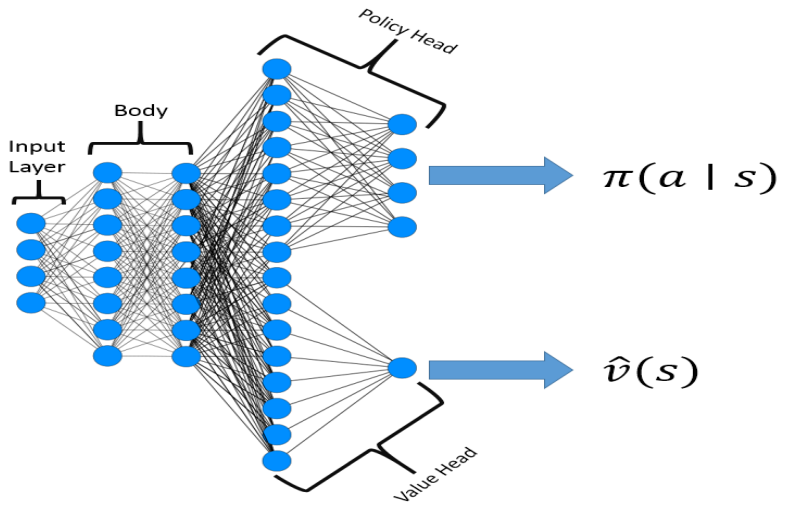
A. Navigation with DL: Recent developments in computational hardware such as GPU, TPUS make it possible to use deep learning in mobile applications as it is at ease to solve a wide range of problems with deep neural networks. However, when it comes to obstacle avoidance, DNN does a great job by learning features, depth from images from which the control commands can be derived easily.

In our case, as the input is from LIDAR and action space is continuous we are using Deep RL based on reading from lidar. To have a well competent agent, we inevitably need a lot of training and a lot of data from the environment.

B. Deep Reinforcement Learning: Although RL is widely used in robotic tasks, however, the project is using DNN for function estimation as we are using Deep Q networks, which is a value-based reinforcement learning approach. As the problem consists of high dimensional state space, we are parameterizing with approximate value function. In general, DQN deals with discrete action space, to make it available for continuous control we are using policy gradients methods (DDPG) where we represent policy and value of the Deep Rl with a neural network. Although DQN has a Normalized advantage function known as NAF, that requires a lot of data in order to optimize, we can use parallel on policy action networks (A3C), but A3C requires multiple simulation environments (Gazebo) to run parallely, as a result, there will be a lot waste in terms of computational power while the agent is being simulated. Thus we choose DDPG as the core algorithm as it needs less training parameters. These subsequent sections will explain how mapless motion planners are being developed using DRL.

III. THEORETICAL BACKGROUND

A. Deep Deterministic Policy Gradient: The core algorithm to this robot is a policy gradient algorithm known as Deep Deterministic Policy Gradient. This framework consists of an actor and a critic, these reinforce algorithms like DQN by extending the action space from discrete to continuous. The whole algorithm uses two separate deep neural networks at its core namely actor and critic network. In actor network, the current state the robot is in is given as input. This network gives us an action to select in that particular state from the available action space. On the other side, critic network's main functionality is to estimate the value of the current action given by the actor network. Learning in this algorithm is done by updating rules done in weights of the actor network, and the critic network learns from the observed TD error of the corresponding state and action. The following picture depicts the representation of the actor-critic networks.



This is the following algorithm we are using in order to make the robot work as in continuous action spaces.

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

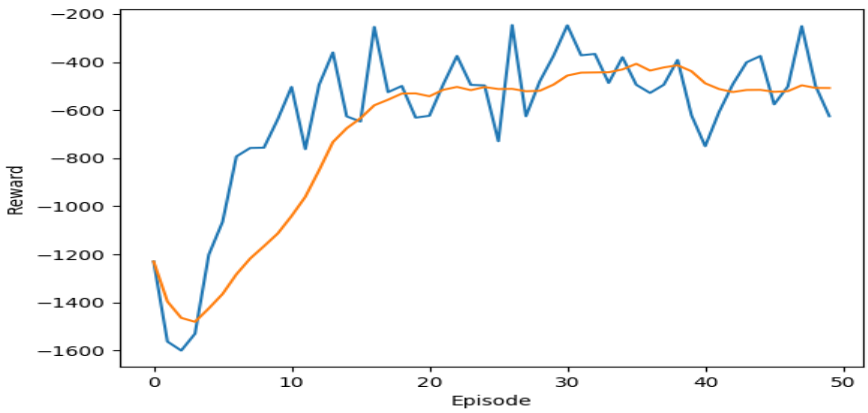
$$\nabla_{\theta^\mu} J(\theta) \approx \frac{1}{N} \sum_i [\nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s=s_i}]$$

J is the policy function that is used to maximize the expected reward the agent receives. Since it is an off policy network and consists of batches of experience, the sum of gradients is calculated for every batch. These are the updates of the target network. These are changed for every batch and often known as soft updates.

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$
$$\text{where} \quad \tau \ll 1$$

IV. IMPLEMENTATION

Before testing the algorithm in the robot itself, we ran it on an experimental environment which in this case is pendulum-v0 by OpenAI to test the effectiveness of DDPG. We used the same network architecture on the robot too. This following graph gives the reward mapping of the DDPG.



The blue line indicates the received rewards and orange indicates the average of it. The increase of the Q-value of DDPG is able to learn the policy to finish this task in different states more efficiently. Here in the proposed, DDPG is collecting one sample for a single back-propagation iteration. The transform function of the robot while in the environment is given below.

$$v_t = f(x_t, p_t, v_{t-1}),$$

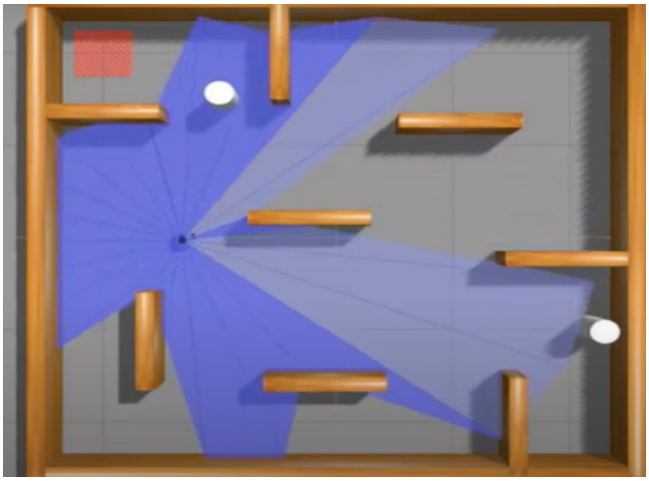
In the equation, Xt is explicit data or the observation made from the lidar which are having high dimensions, Pt is the relative position for the robot to its base and its target position, Vt-1 is the velocity of the robot in the previous state. Scattered laser findings are then taken as a sample from the previous laser readings ranging from -90 to 90 degrees angle distribution. All of this data is normalized into (0,1). The actions of the robot at each state include linear and angular velocities and represented in 2 dimensions, the maximum linear and angular velocity of the robot is 0.7m/s and 1.5rad/s. The target position and angle are mentioned as polar coordinates relative to the robot's base. All this sparse input from the laser is then fed to fc layers, the resultant output is the linear and angular commands of the robot and then they are fed as control signals and are executed by the robot. We used tanh as the activation function in order to narrow the range of the angular velocity which in this case is (-1, 1). The action samples and Q values of the network is using a linear activation function which is:

$$y = kx + b,$$

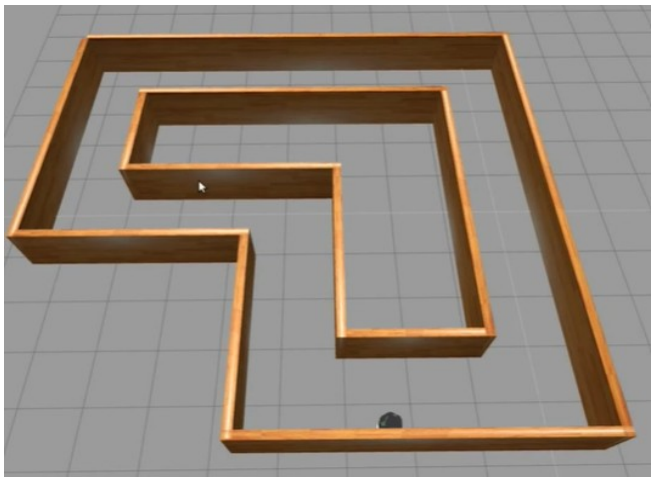
where x is the input of the last layer, y is the predicted Qvalue, and k and b are the trained weights and biases of this layer.

$$r(s_t, a_t) = \begin{cases} r_{arrive} \text{ if } d_t < c_d \\ r_{collision} \text{ if } min_{x_t} < c_o \\ c_r(d_{t-1} - d_t) \end{cases}$$

The robot estimates itself to dt (threshold distance) to reach its goal position. A positive reward which is explicitly mentioned as R_arrive is given for every movement it makes. To consider collision and negative rewards, we have R_collision that awards negative reward after each collision of the robot while reaching its goal position. In order to reach its goal position we are multiplying the difference of last time step to the current time step with Cr (bias) to keep the robot going in the correct direction. Whole training of the model is implemented and simulated in Gazebo in ROS. Here to test the influence of the training environment on robot, we have constructed two environments (Env-1 and Env-2) in Gazebo. One being a dynamic environment and other being a ready environment available in ROS Gazebo environments. In Env-1, there are dynamic obstacles that interact with the robot from its initial to the goal position. In the figures mentioned, the robot is represented as a black dot seen from a bird's view. In each and every episode, the target is randomly initialized within the environment. When it comes to the hyperparameters in the network, the learning rate is set to 0.0001. The whole network model is trained from scratch on GTX 1050ti GPU which took around 36 hours, we used adam optimizer in the network. The mean Q-value of Env-1 turns out to be much higher than expected when compared to other env-2 as it has dynamic obstacles dealing all the time. In the picture below, the goal position is represented by a red square where the robot has to reach with its planner without getting in contact with the obstacles around it. The blue lines represent the laser scans taken while in the environment.



a. Evn- 1, This figure shows the turtle bot during



b. Turtlebot during testing in env-2 training inside env-1.

V. RESULTS AND CONCLUSION

Recapitulating the project, we conclude that RL trained motion planner (mapless) can be revamped to the direct, new unseen environments. From 2 environments, we can also observe that planner is influenced by the navigation trajectories to little extent. However, when we compared the DDPG planned path with other trajectories from the MOVEBASE, it was not much effective. Perhaps, in the future work, reinforcing DDPG with Actor Critic using Kronecker-Factored Trust Region (ACKTR) may generate significantly efficient results which has to be done and observed. The generated path is also twisting and not undulated as the current networks have no recap of previous state. We believe transforming the network to something like RNN (Recurrent Neural Networks) might solve this issue. But when we compare the same approach to low dimensional motion planner it would be effective to some extent and works robust in complex environments.

VI. REFERENCES

1. T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” arXiv preprint arXiv:1509.02971, 2015.
2. S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, “Continuous deep q learning with model-based acceleration,” in Proceedings of The 33rd International Conference on Machine Learning, 2016, pp. 2829–2838.
3. Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, “Target-driven visual navigation in indoor scenes using deep reinforcement learning,” arXiv preprint arXiv:1609.05143, 2016.
4. Virtual-to-real Deep Reinforcement Learning: Continuous Control of Mobile Robots for Mapless Navigation Lei Tai1,2 and Giuseppe Paolo3 and Ming Liu2

5. H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," IEEE robotics & automation magazine, vol. 13, no. 2, pp. 99–110, 2006
6. H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," IEEE robotics & automation magazine, vol. 13, no. 2, pp. 99–110, 2006
7. C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 2722–2730
8. Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates Shixiang Gu∗,1,2,3 and Ethan Holly∗,1 and Timothy Lillicrap4 and Sergey Levine1,5
9. Deep Reinforcement learning for real autonomous mobile robot navigation in indoor environments Hartmut Surmann, Christian Jestel, Robin Marchel, Franziska Musberg, Houssem Elhadj and Mahbube Ardani Computer Science Department, University of Applied Science Gelsenkirchen
10. Yu Fan Chen, Michael Everett, Miao Liu, and Jonathan P. How. Socially aware motion planning with deep reinforcement learning. CoRR, abs/1703.08862, 2017.
11. G. Echeverria, N. Lassabe, A. Degroote, and S. Lemaignan. Modular open robots simulation engine: Morse. In 2011 IEEE International Conference on Robotics and Automation, pages 46–51, May 2011. doi: 10.1109/ICRA.2011.5980252
12. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. CoRR, abs/131
13. Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi ` Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. CoRR, abs/1602.
14. Jahanzaib Shabbir and Tarique Anwer. A survey of deep learning techniques for mobile robot applications. CoRR, abs/1803.07608
15. Lei Tai, Giuseppe Paolo, and Ming Liu. Virtual to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. CoRR, abs
16. Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando ´ de Freitas. Sample efficient actor-critic with experience replay. CoRR, abs.

