Continuous & Mapless Navigation for robots using Policy Gradient Algorithm

Avnish Gupta, Shreedhar Kodate, Anamika Kumari, Ashwij Kumbla
Worcester Polytechnic Institute
{agupta8,sskodate,akumari,akumbla}@wpi.edu

1 INTRODUCTION

Autonomous Mobile Robots are increasing their presence in our day-to-day lives. It is very common to find robots finding and moving packages in large retail warehouses. They need to traverse accurately in the warehouse without colliding with shelves, walls, pillars, and other static or moving obstacles. They need to carefully move alongside other robots or humans. This requires an a intricate clockwork which is not rigidly based on a map, so that robots can dynamically handle different situations while set out to complete their tasks. Thus, it is important to make the mobile robots to sense their surroundings using laser sensors and move around without colliding.

However, using a real-world robot to learn an intelligent behavioral model to navigate the mapless environment is very expensive. Hence, it is important to simulate the real-world situation in software like ROS/Gazebo and train a Reinforcement Learning based agent to navigate through obstacles and reach the desired goal.

We seek to build a reinforcement learning based mapless motion planner that takes 90-dimensional range finding and target position with respect to the robot as input and generates continuous steering (linear and angular) command as output. Traditional motion planners for robots with a laser range sensor mainly depend on a predefined obstacle map of a navigation environment where both the highly precise laser sensor and obstacle map building work of the surroundings are indispensable. We used the Actor-Critic based policy gradient learning method to train a mapless motion planner without any predefined features and demonstrations. The proposed mapless motion planner aims to navigate the non-holonomic robot to desired targets without colliding with any obstacles.

Earlier deep reinforcement learning methods have achieved great success in tasks like simulation control agents and video games. But the application of deep reinforcement learning in robotics is primarily limited to manipulation where the environment is fully observable and stable. Traditional methods like SLAM handle this problem through the prior obstacle map of the navigation environment based on dense laser range findings. But there are two significant issues in this method: 1) building and updating the obstacle map is time-consuming, and 2) the high dependency on precise dense layer sensors for the mapping work and local cost-map prediction. It is challenging for a motion planner to generate global navigation behaviors with the local observation and the target position information directly without a global obstacle map. Thus, we present a training-based mapless motion planner through deep reinforcement learning.

Further, we give a brief literature review in the Related Work section 2. The Task Overview section 3 explains the use case of this project and

Supervised by Prof. Yanhua Li.

,
2022.

descriptions about the environment and the robot and the actions it can take. In section 4, we explain the DDPG algorithm using pseudo code and some tricks and tips that have been employed to train the RL agents. In section 5, we explain the complete environment setup, network architectures, design decisions in the reward function, training the models and experimental setup. After that we explain our Results in section 6, conclude in section 7 and explain ideas for future work in 8.

2 RELATED WORK

Mapless Navigation Task has been approached earlier using both Deep Learning and Deep Reinforcement Learning.

A. Deep Learning-based Navigation: Deep Neural networks have shown an excellent possibility for solving complex estimation problems in the past. Deep Neural Networks have been successfully applied to monocular images[2] and depth images[7] for learning obstacle avoidance. Shabbir et al.[6] provides a meticulous research survey associated to deep learning techniques for mobile robot applications, with a specific focus on the advantages and obstacles, in comparison to traditional robotics. They show that the focus of current deep learning research is more vision-based, whereas current real mobile robotic applications are more based on laser scanners.

The deep neural network was used to extract semantic information from the images, build a robot with a 3D laser scanner, and train the behaviour of the autonomous vehicle to avoid obstacles by Chen et al.[1] . However, they have fixed their control commands to be very strict, like turning left or turning right, which may lead to rough navigation behaviours.

In 2014, J. Long et al.[4] extended the deep neural network from perception to decision making and proposed a fully connected convolutional neural network that dramatically reduces the computation redundancy and could be adapted to inputs of random size.

These various proposed architectures have been validated on a typical dataset, but it is still unclear how well these methods are in the real world.

B. Deep Reinforcement Learning: Reinforcement Learning has been widely applied in the field of various navigation tasks. Robot Navigation based on depth image was trained on DQN by Zhang et al[9] They used successor features to transfer the strategy to an unknown environment. The shortfall of DQN is that it can only be used in discrete action space.

Deep Q Network was implemented by Minh et al.[5]; that utilizes deep neural networks to estimate the function of value-based reinforcement learning. Lillicrap et al.[3] proposed a deep deterministic policy gradient DDPG model-free algorithm that utilizes a neural network on the actor-critic methodology that can operate on continuous action space. Tai et al.[8] proposed a learning-based mapless motion planner by taking sparse 10-dimensional range findings and target position with respect to the mobile robot coordinate frame as input and continuous steering command as output. They have deployed their network to a physical robot platform. Since the absence of a 3D sensor, they have to prepare the environment to avoid obstacles outside of the 2D plane sensed by the laser scanner. We extend their work on 90-Dimensional Range findings and modify their DDPG actor-critic network architecture by adding more layers and updating the parameters. Furthermore, we have trained the agent for both backward and forward motion, whereas the previous implementation was only for forward motion.

3 TASK OVERVIEW

We are trying to implement a mapless motion planner which takes current information of the environment using laser range finder and goal location as shown in 1. The planner generates control outputs for the robot to follow and reach the goal. We also aim to develop intelligence for avoiding dynamic and static obstacles while reaching the

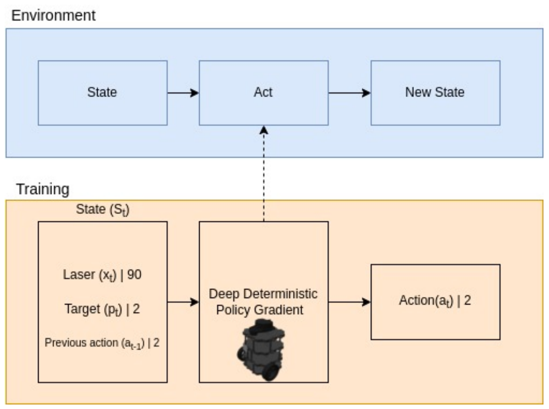

Figure 1: RL based mapless motion planner

goal. We will train a policy gradient based reinforcement learning model which takes in the state vector and directly outputs the desirable control linear and angular velocity for the robot.

Action Space: The 2-dimensional action of every time step includes the angular and the linear velocities of the differential mobile robot. To constrain the range of angular velocity in $(-0.5, 0.5)$, a hyperbolic tangent function (tanh) is used as the activation function. Moreover, the range of the linear velocity is constrained in $(0, 0.5)$ through a sigmoid function for forward motion and the range of the linear velocity is constrained in $(-0.5, 0.5)$ through a tanh function for backward motion. Considering the real dynamics of the robot we clip the angular velocity at 1 radian per second and linear velocity at 0.5 meter per second.

State Space: The state vector is abstracted from 90-dimensional laser range findings, the previous action, the relative target position (relative distance and heading) are merged together as a 94-dimensional input vector. The laser range findings are sampled from the raw laser findings between 0 and 360 degrees in a trivial and fixed angle distribution of 4 degrees. The range information is normalized to $(0, 1)$. The 2-dimensional target position is represented in polar coordinates (distance and angle) with respect to the mobile robot coordinate frame.

4 DEEP DETERMINISTIC POLICY GRADIENT

Deep Deterministic Policy Gradient (DDPG)[3] is an algorithm which concurrently learns a Q-function and a policy. It uses off-policy data and the Bellman equation to learn the Q-function, and uses the Q-function to learn the policy. This approach is closely connected to Q-learning, and is motivated the same way: if you know the optimal action-value function Q^*(s,a), then in any given state, the optimal action a^*(s) can be found by finding the argmax of Q^*(s,a) for all possible actions. Because the action space is continuous, the function Q^*(s,a) is presumed to be differentiable with respect to the action argument. This allows us to set up an efficient, gradient-based learning rule for a policy μ(s).

DDPG interleaves learning an approximator to Q^*(s,a) with learning an approximator to a^*(s), and it does so in a way which is specifically adapted for environments with continuous action spaces. DDPG, are largely based on minimizing this Mean Square Error(MSE) loss function. There are two main tricks employed in DDPG algorithm which make the learning stable.

1. Replay Buffers - All standard algorithms for training a deep neural network to approximate $Q^*(s, a)$ make use of an experience replay buffer. This is the set $\mathcal{D}$ of previous experiences. In order for the algorithm to have stable behavior, the replay buffer should be large enough to contain a wide range of experiences, but it may not always be good to keep everything. If you only use the very-most recent data, you will overfit to that and things will break; if you use too much experience, you may slow down your learning.

2. Target Networks - Q-learning algorithms make use of target networks. The term is called the target, because when we minimize the loss, we are trying to make the Q-function be more like this target. Problematically, the target depends on the same parameters we are trying to train: ϕ. This makes minimization unstable. The solution is to use a set of parameters which comes close to ϕ, but with a time delay. We use a second network, called the target network, which lags the first. The parameters of the target network are denoted ϕ_{targ}. We update the target networks after some fixed number of steps with the weighted averaging.

DDPG trains a deterministic policy in an off-policy way. Because the policy is deterministic, if the agent were to explore on-policy, in the beginning it would probably not try a wide enough variety of actions to find useful learning signals. To make DDPG policies explore better, we add noise to their actions at training time. The authors of the original DDPG paper recommended time-correlated Ornstein-Uhlenbeck Process Action.

The pseudo code to explain the DDPG model training is given in Algorithm 1.

5 METHODOLOGY

We try to find such a velocity control function $F(x_t, p_t, v_{t-1})$ where x_t is the observation from the raw sensor information, p_t is the relative position (Euclidean distance) of the target respect to robot position, and v_{t-1} is the velocity (linear and angular) of the mobile robot in the last time step. They can be regarded as the instant state s_t of the mobile robot. The model directly maps the state to the action, which is the next time velocity v_t. As an effective motion planner, the control frequency must be guaranteed so that the robot can react to new observations immediately.

This section introduces the implementation details of the problem. It is divided into five subparts: environment setup, network architecture, reward function, training, experimental setup.

5.1 Environment Setup

The training environment is implemented virtually and is simulated in Robot Operating System (ROS) and Gazebo. ROS is not an operating system (OS) but a set of software frameworks for robot software development. Gazebo is an open-source 3D robotics simulator. It integrated the ODE physics engine, OpenGL rendering, and support code for sensor simulation and actuator control.

```
Data: Init model parameters policy $\theta$,
      Q-value $\phi$, empty experience replay
      buffer $\mathscr{D}$.;
Set target model's parameters: $\theta_{target} \leftarrow \theta$,
$\phi_{target} \leftarrow \phi$.;
Result: Converged Autonomous Mobile
        Robot (AMR) agent
while not converged do
    Observe state $s$, select action
      $a = clip(\mu_\theta(s) + \epsilon, a_{low}, a_{high}$ where $\epsilon$
      in $\mathscr{N}$.;
    Take action $a$ in the environment.;
    Observe next state $s'$, reward $r$, and
      boolean done signal $d$ to indicate
      whether terminal state.;
    Store $(s, a, r, s', d)$ in replay buffer $\mathscr{D}$.;
    if s' is terminal then
        Reset environment state.;
    end
    if it's time to update then
        for however many updates do
            Randomly sample a batch of
              experience from the buffer.;
            Compute targets $y(r, s', d) = r +$
              $\gamma(1-d)Q_{\phi_{target}}(s', \mu_{\theta_{target}}(s'))$.;
            Update Q-function by one step
              gradient descent.;
            Update policy by one step of
              gradient.;
            Update target networks with
              $\theta_{target} \leftarrow \rho\theta_{target} + (1-\rho)\theta$,
              $\phi_{target} \leftarrow \rho\phi_{target} + (1-\rho)\phi$;
        end
    end
end
```

Algorithm 1: Deep Deterministic Policy Gradient (DDPG)

Our agent is a standard platform robot simulated in ROS and Gazebo called as Turtlebot3. We use burger model of Turtlebot3 as out agent for this project. The simulation of Turtlebot3 in Gazebo is a non-holonomic differential drive robot equipped with sensor like laser range finder, camera etc. It can be directly controlled in Gazebo simulated world by publishing velocity message. We use the laser finding sensor for perceiving the surrounding of the robot. The sensor is configured with a minimum range of 0.1 meter and maximum range of 3.5 meter. Further we have added a Gaussian process noise to the sensor data to make is more like a real LIDAR sensor. The data from the laser range finder is sampled in all 360 degrees at a regular interval of 4 degrees. This gives a 90 dimensional laser scan data for the environment around the robot.

We train the agent in this Gazebo simulated world where it interacts with it to reach the randomly positioned 0.4 square meter red goal box within the tolerance of 0.15 meter. The location of the goal is changed randomly within some bounds at the start of each episode. This helps our agent to generalize for world it is interacting. We train our model in two environments separately. First is a simple environment with no obstacles shown in 2 and second is a complex environment with lots of obstacles as shown in 3.

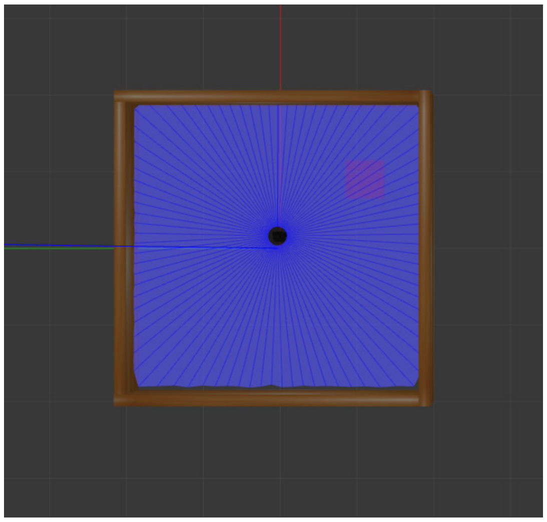

Figure 2: Environment Stage 1 - Simple without obstacles

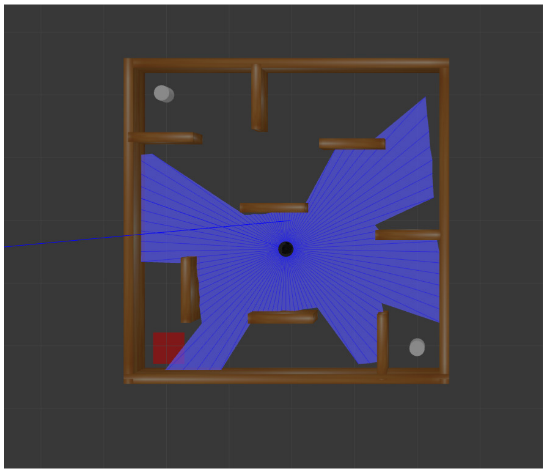

Figure 3: Environment Stage 4 - Complex with static obstacles

5.2 Network Architecture

For training the robot we used the Actor-Critic deep deterministic policy gradient based reinforcement learning. We have trained two different actor models with similar architecture for forward only and forward plus backward motion. We call the actor model for forward motion as modelF shown in Figure 4 and for the backward motion as modelFB shown in Figure 5.

The architecture of both the models have been shown in figure. The actor network takes 94 dimensional state input followed by five fully connected layers and produces an embedding of size two as output. Except for the output layer ReLU activation is used for all the hidden layers. In the case of modelF, the last layer has a sigmoid for producing positive linear velocity output between zero to one and a tanh activation function for producing angular velocity outputs between -1 to 1. For modelFB, tanh activation function is used for both linear and angular velocity outputs. The final output of actor network is scaled with the respective max values of linear and angular velocities.

The critic network architecture is the same for both backward and forward motion to predict the Q-value of the state and action pair. We merge the action and state input after a single 128-dense layer, followed by 4 fully connected layers. The

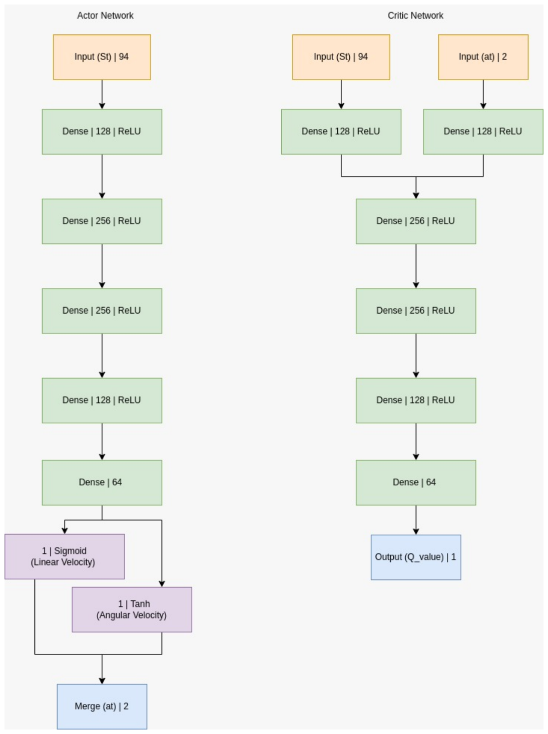

Figure 4: ModelF - Actor-Critic for forward motion

ReLU activation function is used in all the hidden layers, except the last output layer which has no activation function.

The input and output of the actor and critic network are summarized as:

(1) Actor
  (a) Input
    (i) State: 94D vector
      (A) The laser range findings
      (B) The previous action (2D)
      (C) The relative target position in polar coordinates(2D)
  (b) Output
    (i) Action: 2D
      (A) Linear velocity (0, 0.5) m/s for forward motion and (-0.5, 0.5) for backward motion
      (B) Angular velocity (-0.5, 0.5) rad/s
(2) Critic
  (a) Input
    (i) State: 94D vector
      (A) The laser range findings
      (B) The previous action (2D)
      (C) The relative target position in polar coordinates(2D)
  (b) Output
    (i) Action: 2D
      (A) Linear velocity (0, 0.5) m/s for forward motion and (-0.5, 0.5) for backward motion
      (B) Angular velocity (-0.5, 0.5) rad/s

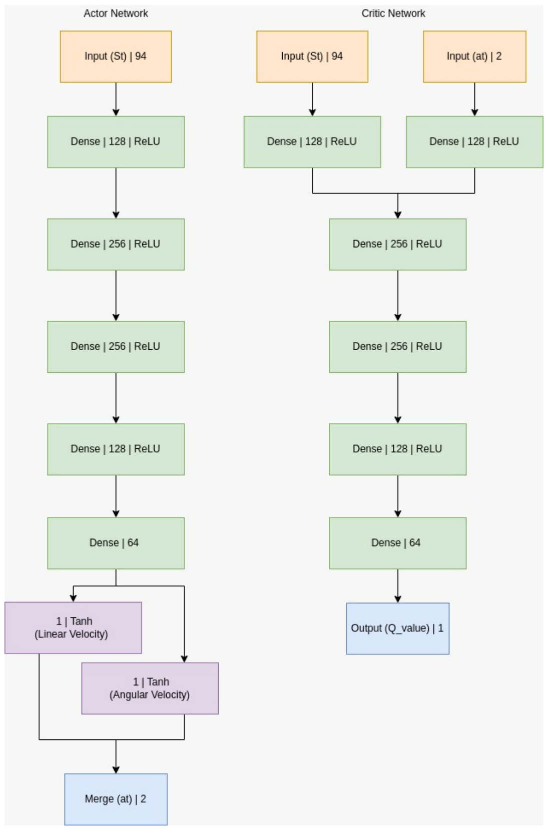

Figure 5: ModelFb - Actor-Critic for Forward and Backward motion

5.3 Reward Function

Relative distance of the robot and the target location is calculated at every time step t of the robot movement simulation and is named as $(dist\_goal_t)$. The robot can be in any one of the following states:

(1) Moving closer to the goal $(move\_closer)$

(2) Moving away from the goal ($move\_away$)
(3) Collided with the obstacles ($collided$)
(4) Reached goal state ($reached\_goal$)

Let the difference of current distance from the goal and past distance be:

$$dist\_rate = dist\_goal_{t-1} - dist\_goal_t$$

Based on the $sign(dist\_rate)$, the robot goes either into

(1) **Move closer**: The reward is then set to be $200 * dist\_rate$. The idea is to motivate the robot to seek more positive reward proportional to it's velocity towards the goal.
(2) **Move away**: However, if the robot moves away from the goal, then it is given a static negative reward of -8.

Thus, the robot is discouraged from roaming around the map and quickly moves towards the goal location. There are special rewards for the two terminal states:

(1) **Collided**: While moving, the robot might get stuck in the same position for a long time and that could be interpreted as the sign of a collision. Thus, if the robot does not change it's position for 20 timesteps then it enters the collided state and is given a negative reward of -10.
(2) **Reached goal**: The goal state is defined as a region of $(0.4x0.4m^2)$ in the environment, and whenever the robot successfully reaches the goal within predefined steps, it gets a large positive reward of $+100$.

Thus, the robot successfully learns to reach the goal state with a shortest path possible and avoiding all the obstacles.

5.4 Training

For training the agent we used Actor Critic based DDPG algorithm along with replay buffer and fixed targets. The minimum replay buffer size before the training starts was set to be 1500 and maximum replay buffer size was set to be 50000. The training sequence does not starts until the replay buffer is filled with the minimum number of (state, action, reward, next state) pairs.

Once the minimum buffer size exceeds the minimum, we start to train the actor and critic networks at each 20th steps. At each 20th step we train the network for 20 times to speed up the training of our actor and critic. We update the target network with a soft update function at each two hundredth step during the training. We train our network for approximately six thousand episodes before it converges. For Environment stage 1 the maximum steps allowed in the episode is set to be 150 as it is rather a simple environment. For Environment stage 2 the maximum steps allowed in the episode is set to be 250 as it is more complex environment to train on.

5.5 Experimental Setup

We have trained our model on a machine with 16GB of RAM and Nvidia RTX 3060 6BG GPU. The learning rate for actor and critic were set to be same as 0.0001. The batch size for training was set to be 256. Discount factor was set to be 0.99 and soft update constant for updating the target network was set to be 0.05. The linear velocity for forward motion was clipped within [0, 0.5]m/s and linear velocity for backward motion was clipped within [-0.5, 0.5]m/s. The angular velocity was clipped within [-0.5, 0.5]rad/s.

Further, we trained the agent on environment stage 1 and environment stage 2 separately for forward motion for approximately six thousand episodes which took almost 20 hours to train. We trained agent on environment stage 2 for forward and backward motion for approximately six thousand episodes which took almost 25 hours to train.

6 RESULTS

We have successfully trained RL agents using DDPG algorithm to handle mapless navigation in multiple different environments.

Figure 6 shows the training loss while training the Critic network. The loss is increasing initially due to the exploratory episodes. As the agent

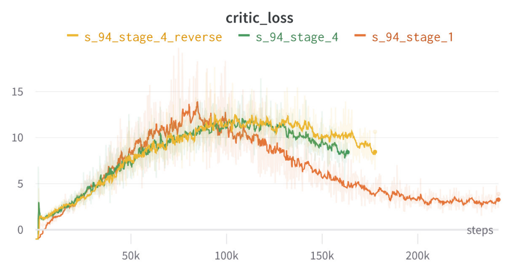

Figure 6: Critic Loss

learns better actions, the loss decreases and starts to converge after 200k steps.

Figure 7 shows the average rewards obtained by the agent during training. Initially the rewards are negative or close to zero because of exploratory actions in the beginning. Also, there are more collisions during the first 2000 episodes, and hence, there is no positive reward. Overtime, the agent learns to avoid collisions. Based on the specific reward function designed, the agent eventually learns to reach the goals faster and hence the converged positive reward after 6000 episodes. Also, we should note that the stage 4 agent has to travel more distance to avoid obstacles before reaching goal, hence it's average reward is slightly higher than stage 1.

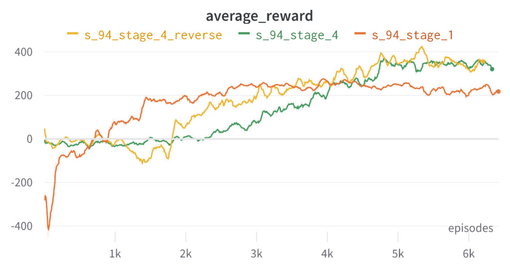

Figure 7: Average train Loss

Figure 8 shows the average rewards obtained by the agent during testing. We can understand from here that the rewards are high in general for the stage 4 environment because the agent gets more positive rewards due to more steps to reach due to presence of the obstacles.

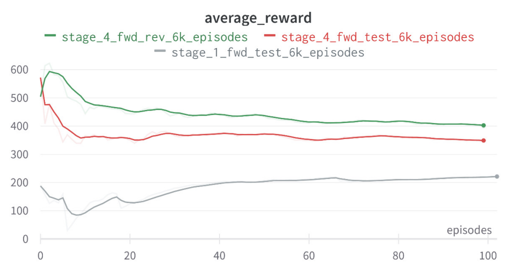

Figure 8: Average test Loss

7 CONCLUSION

The DDPG algorithm shows promising results for the Autonomous Mobile Robot control in different mapless environments. DDPG worked well for this continuous space of state and actions. Because, the state space is of 94 dimensions, agent needs a large number of exploration steps in the episodes and hence, it starts performing better after 1000 episodes. By taking the 90-dimensional sparse range findings and the target position relative to the mobile robot coordinate frame as input, the proposed motion planner can be directly applied in unseen real environments without fine-tuning, even though it is only trained in a virtual environment. When compared to the low-dimensional map-based motion planner, our approach proved to be more robust to extremely complicated environments.

8 FUTURE WORK

The DDPG performs really well on the given turtlebot environments. However to see if other algorithms can perform better and converge faster, they can be compared to the following Policy Gradient algorithms for modelling the Robot movements:

(1) ADDPG - Asynchronous Deep Deterministic Policy Gradient: The Asynchronous training can help the model converge faster than normal training. It'll update the network parameters frequently and gain by exploring different scenarios in parallel.
(2) MADDPG - Multi-agent Deep Deterministic Policy Gradient: In the real world, there

will be multiple robots wanting to achieve their goals as soon as possible. Hence, as mentioned in the article by Lillian Weng [?], we can train a multi-agent system where the DDPG agents can learn to collaborate and synchronize their motions to achieve individual goals. And, also to maintain system safety and efficiency by not colliding with each other.

Reward shaping: We have used positive intermediate rewards to motivate the agent to move towards the goal. But, it might create a loophole for the agent to go slower when reached towards goal and gain more positive rewards just by roaming near goal but not reach it sooner. Hence, we can also experiment with rewards of different magnitude, no intermediate reward and long-term rewards.

9 FURTHER READING

- https://arxiv.org/pdf/1703.00420.pdf
- https://arxiv.org/pdf/2005.13857.pdf

REFERENCES

[1] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2722–2730, 2015.

[2] Y. Lecun, U. Muller, E. Cosatto, and B. Flepp. In *Advances in Neural Information Processing Systems (NIPS 2005).*

[3] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2015.

[4] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[6] Jahanzaib Shabbir and Tarique Anwer. A survey of deep learning techniques for mobile robot applications. *CoRR*, abs/1803.07608, 2018.

[7] Lei Tai, Shaohua Li, and Ming Liu. A deep-network solution towards model-less obstacle avoidance. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2759–2764, 2016.

[8] Lei Tai, Giuseppe Paolo, and Ming Liu. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. *CoRR*, abs/1703.00420, 2017.

[9] Jingwei Zhang, Jost Tobias Springenberg, Joschka Boedecker, and Wolfram Burgard. Deep reinforcement learning with successor features for navigation across similar environments. *CoRR*, abs/1612.05533, 2016.