RL based Differential Drive Primitive Policy for Transfer Learning

Author

Mahrukh Shahid

Registration Number

00000277078

Supervisor

Prof. Dr. Yasar Ayaz (Pride of Performance)

DEPARTMENT OF ROBOTICS AND INTELLIGENT MACHINE ENGINEERING

SCHOOL OF MECHANICAL & MANUFACTURING ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

AUGUST 2022

RL based Differential Drive Primitive Polices for Transfer Learning

Author

MAHRUKH SHAHID

Registration Number

00000277078

A thesis submitted in partial fulfillment of the requirements for the degree of

MS Robotics and Intelligent Machine Engineering

Thesis Supervisor:

Prof. Dr. Yasar Ayaz (Pride of Performance)

Thesis Supervisor's Signature: _____

DEPARTMENT OF ROBOTICS AND INTELLIGENT MACHINE ENGINEERING

SCHOOL OF MECHANICAL & MANUFACTURING ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY,

ISLAMABAD

AUGUST 2022

Declaration

I certify that this research work titled "*RL based Differential Drive Primitive Policies for Transfer Learning"* is my own work. The work has not been presented elsewhere for assessment. The material that has been used from other sources it has been properly acknowledged / referred.

(Signature of Student)

Mahrukh Shahid

00000277078

Plagiarism Certificate (Turnitin Report)

This thesis has been checked for Plagiarism. Turnitin report endorsed by Supervisor is attached.

(Signature of Student)

Mahrukh Shahid

00000277078

(Signature of Supervisor)

Copyright Statement

- Copyright in text of this thesis rests with the student author. Copies (by any process) either in full, or of extracts, may be made only in accordance with instructions given by the author and lodged in the Library of NUST School of Mechanical & Manufacturing Engineering (SMME). Details may be obtained by the Librarian. This page must form part of any such copies made. Further copies (by any process) may not be made without the permission (in writing) of the author.
- The ownership of any intellectual property rights which may be described in this thesis is vested in NUST School of Mechanical & Manufacturing Engineering, subject to any prior agreement to the contrary, and may not be made available for use by third parties without the written permission of the SMME, which will prescribe the terms and conditions of any such agreement.
- Further information on the conditions under which disclosures and exploitation may take place is available from the Library of NUST School of Mechanical & Manufacturing Engineering, Islamabad.

Acknowledgements

I am thankful to my Creator Allah Subhana-Watala to have guided me throughout this work at every step and for every new thought which You setup in my mind to improve it. Indeed, I could have done nothing without Your priceless help and guidance. Whosoever helped me throughout the course of my thesis, whether my parents or any other individual was Your will, so indeed none be worthy of praise but You.

I am profusely thankful to my beloved parents who raised me when I was not capable of walking and continued to support me throughout in every phase of my life.

I would also like to express special thanks to my supervisor Prof. Dr. Yasar Ayaz for his help throughout my thesis.

I would also like to pay special thanks to Ahmed Hasnain Johar a dear friend for his tremendous support and cooperation. Each time I got stuck in something; he came up with the solution. Without his help I wouldn't have been able to complete my thesis. I truly appreciate his patience and guidance throughout the whole thesis.

I would also like to thank Dr Sara Ali and Dr Khawaja Fahad Iqbal for being on my thesis guidance and evaluation committee and express my special thanks to Seemab Buzdar for his unconditional help.

Finally, I would like to express my gratitude to all the individuals who have rendered valuable assistance to my study.

Dedicated to the never-ending courage, hope & unconditional support

Abstract

To ensure the steady navigation for robot stable controls are the basic unit and control values selection is highly environment dependent. **A**dding Generalization to system is the key to reusability of control parameters to ensure adaptability in robots to perform with sophistication, in the environments about which they have no prior knowledge, for this Reinforcement Leaning (RL) based control systems are promising. However, tuning appropriate parameters to train RL algorithm is a challenge. Therefore, we designed a continuous reward function to minimizing the sparsity and stabilizes the policy convergence, to attain control generalization for differential drive robot. We Implemented Twin Delayed Deep Deterministic Policy Gradient-TD3 on Open-AI Gym Race Car. System was trained to achieve smart primitive control policy, moving forward in the direction of goal by maintaining an appropriate distance from walls to avoid collisions. Resulting policy was tested on unseen environments and observed precisely performing results. Upon comparative analysis of TD3 with DDPG, TD3 policy outperformed the DDPG policy in both training and testing phase, proving TD3 to be resource efficient and stable.

Table of Contents

List of Figures

List of Tables

CHAPTER 1: INTRODUCTION

Artificial intelligence and **robotics** collectively aim to achieve human like artificial entities to assist humans in the society. Learning and perceiving the environment in way as human do along with planning and acting to attain the required goal are the fundamental requirements. Numerous techniques and learning frameworks have been introduced in literature to strive in the direction of designated aim. However, domain of **reinforcement learning (RL)** is proving its worth by adding learning instincts over path planning techniques, and specifically over control layer, to ensure promising level of autonomy in the social assistive robot system.

Although RL is one competent learning technique but there exist complications that limits its incorporation in the real word. This opens-up a research gap in the domain of RL. The foremost concern of RL is its requirement of tremendous amount of **training time** and **resource consumption**. Resolving this would make it a step closer to become the part of the physical world. The aim of this research is to provide with an RL based training strategy which is **less time consuming** and provides a **reusable skill**.

1.1 Background, Scope, and Motivation

1.1.1 Controls

Typically, robot systems are designed and developed in such a way that they are empowered to successfully carry out the specified task. Control systems make it possible for constituents of the robot to move and operate. As well as it enables the robot to carry out a predetermined series of motions and forces, even in the event of an unexpected error occurrence [1].

As suggested by a typical task operation architecture figure 1, for any robotic system to accomplish the assigned task, planning is essential. However, to execute the planed task controls of robots comes into action and directly interacts with the environment. Therefore, a steady task performance for robot cannot be accomplished without stability in controls of robot. However, control commands are typically environment dependent [2]. As, actuator dynamics, joint flexibility

and other numerous system uncertainties needs to be considered while dealing with the control problem of robot manipulators [1].

Typically, robots are required to performing in dynamic environments specifically socially assistive robots and autonomous cars. They need to generalize their controls over the changing environment problem to perform autonomous operations. To attain generalization in controls, various learning-based controls architectures came into existence e.g. intelligent computational techniques such as Fuzzy Logic theory (FL), Artificial Neural Networks (ANN) and other evolutionary computational methods such as Genetic Algorithm (GA) have provided us with a new array of solutions to problems in various control systems [1].

However, reinforcement learning based controls supersedes the other learning techniques. As, it excludes the need to explicitly model the system, or compute the kinematic equations, unlike other learning techniques. Moreover, the dynamics of the robot are highly non-linear and difficult to be modeled [1] RL can effectively abstract out these hardware level dependencies.

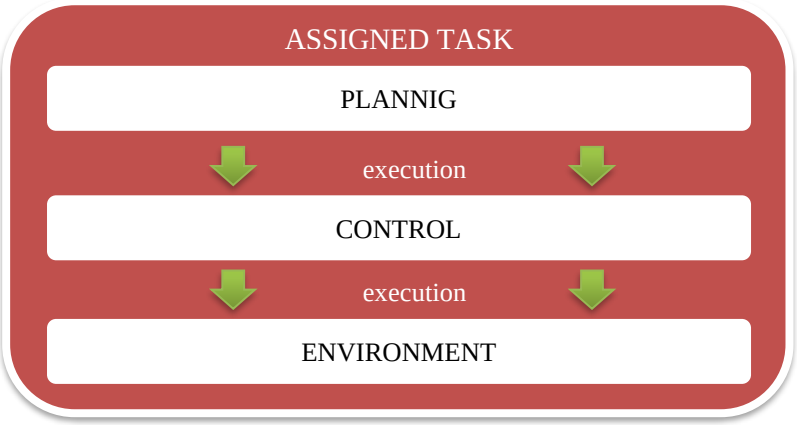

Figure 1: Task operation architecture

1.1.2 Reinforcement Learning

Reinforcement learning is a hit and trial reward signal-based learning paradigm [3]. In which an agent explores the environment and its own behavior, analyze it on the bases of received reward signal, by the goal defining reward function, in the quest of achieve maximum cumulative reward. Figure 2 shows the typical framework of reinforcement learning.

Unlike other machine learning approaches, RL can be a promising framework for new skill set development in robots. It can make agent learning newfangled tasks which cannot be demonstrated physically by humans, and tasks which are complex to be modeled. It can even effectively incorporate non-linear complexities e.g., handling objects with delicacy or following a specified walk. Moreover, it ensures problem generalization instincts in robots i.e., learned skill is robust enough to be deployed in a previously unobserved environment.

Furthermore, by simply utilizing a known cost function, RL can achieve the optimization goals of complex problems with no analytic formulation, and issues with unknown closed form solutions. It can efficiently handle the issues even when the human instructors are not sure about the optimal solution. In Addition to that, RL can guarantee the capacity to dynamically adapt to changes in the agent itself, such as a robot adjusting to hardware modifications [4]. These properties prove RL framework to be a strong problem-solving mechanism.

However, RL is based on reward maximization in the given environment over timesteps therefore, it requires extensive trial and error in environment to achieve the desired goal i.e., policy. Policy is the selection of action given state in a way to maximize the reward. This originates the sample inefficiency problem. Moreover, numerous hyperparameter needs to be tuning to converge the RL algorithm [10].

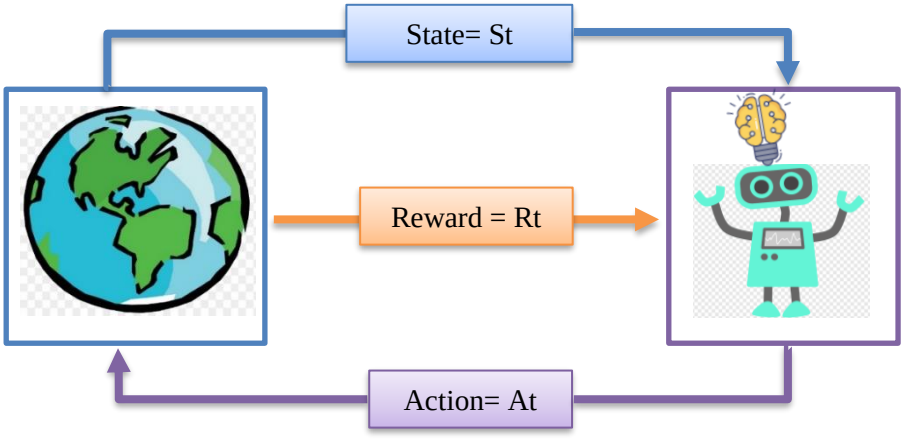

Figure 2: Reinforcement learning framework

1.1.3 Markov Decision Process (MDP)

RL problems setting can be effectively modeled using Markov decision process (MDP), figure 3. MDP models decision making on discrete, stochastic, and sequential environments. Each state of MDP observes Markov property i.e., probability of occurrence of state s_{t+1} is equal to the sum of probabilities of states from s_0 to s_t. this excludes the need to preserve the entire sequence of states, as single state holds the transition information from past to future.

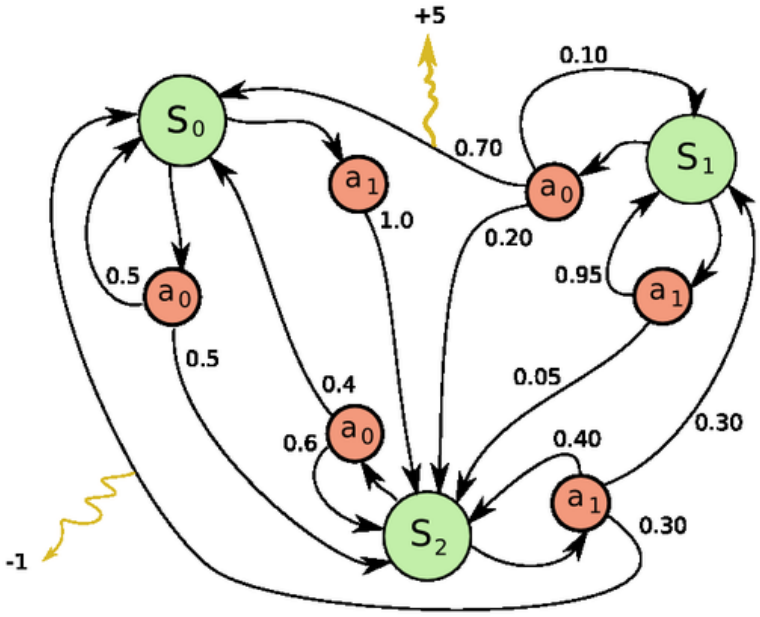

Figure 3: Typical Markov decision process (MDP)

Tuple of MDP consists of **states space S** representing the states of environment and **action space A** feasible action from the given state. **P is the transition probability** to state s_{t+1} given state s_t and action a_t.

$$P_{ss'}^{a} = P[S_{t+1} = s' \mid S = s, \quad A_t = a]$$

Reward signal R is estimated reward of transition r_{t+1} given state s_t and action at.

$$R_s^a = E[R_{t+1} \mid S_t = s, \quad A_t = a]$$

Discount factor γ, to compute **return Gt** estimated discounted reward from future states. Return Gt adds perspective of possible impact of future states based on reward, to the current state. Where, γ is bounded between [0, 1] ranging myopic to farsighted evaluation.

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Adding perspective of MDP to RL agent is required, on bases of which it evaluates the states and select action to solve MDP. Therefore, **policy π** distribution of action A given state S is used by agent to draw action given state. Selection of maximum reward generating action will solve the MDP.

$$\pi(a|s) = P[A_t = a \mid S_t = s]$$

However, **Value function** is used to evaluate the goodness of state, or state-action pair. **Q-value function** is the expected return given state s and action a. Therefore, maximization of Q value over all policies leads to optimal solution.

$$Q_\pi(s, a) = E[R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

Bellman equation for Q:

$$Q(s, a) = R_s^a + \gamma \sum_{s' \epsilon S} P_{ss'}^a Q(s', a)$$

Bellman equation for optimal Q-value function:

$$Q^*(s, a) = R_s^a + \gamma \sum_{s' \in S} P_{s's}^a \, max_{a'} \, Q^*(s', a')$$

1.2 Gap Analysis

Robots interacting with human-inhabited, unstructured, and highly uncertain environment, traditional control system in such environment is most likely to be doomed to failure as they are based on manually preprogramming and traditional physics-based modelling tools and hand-crafted models**.** To overcome this limitation model learning effective incorporate environment's nonlinearities and can generalize over system, as model can be estimated directly from the real data of the system [2]. However, to deploy model-predictive controls, state-of-the-art approaches often adopt a sequential approach with components as, starting with state estimation, managing contact with environment followed by trajectory prediction and optimization, model-based control prediction and finally operational space command [5][6]. Designing and development of such approach is dependent on availability of accurate dynamic models of robot which is not a trivial task and requires technical expertise of the system. In contrast, end-to-end deep reinforcement learning can effectively abstract the low-level system dependent specifications with relatively similar reward function equation. Therefore, excluding the need for any prior knowledge about the robot's and environment's dynamics. This can ensure the performance of robot without explicit system identification or manual engineering. Therefore, if deep reinforcement learning is effectively implemented, it may automate controller design, eliminating the requirement for system identification and producing controls that are directly tuned for a specific robot and environment [6].

As RL is based on reward maximization in the given environment over timesteps therefore, it requires extensive trial and error in environment to achieve the desired goal i.e., policy selecting of action given state in a way to maximize the reward, this originates the sample inefficiency problem, along with that numerous hyperparameter needs to be tuning. So, to design and train RL policy for continuous action spaces, Twin delayed deep deterministic policy gradient (TD3)[7] a successor of DDPG [8], ensure a stable and robust actor update along with the minimization of overestimation bias in critic network and eventually stabilizes the learning for continuous action spaces. However, TD-3 is not implemented for differential drive car.

1.3 Problem Statement

Training TD-3 to convergence is a complicate and time-consuming process. However, it is considered as stable algorithm for continuous action spaces. No implementation of TD-3 for differential drive is available in literature. Therefore, implementing TD-3 to analyze the convergence in comparison with DDPG and to ensure the trained policy reusability, design a primitive policy for transferable learning.

CHAPTER 2: LITERATURE REVIEW

As the identified problem hovers over two different and prevalent domains of knowledge, both have attained sophisticated level of maturity in terms of development and implementation, control system and reinforming learning. Therefore, a reasonable amount of literature assessment is required to find an apt and efficient solution, in terms of stability, scalability and robustness to ensure a reliable merger to best address the specified problem.

Therefore, this chapter is consisting of comparison between traditional controls architecture and learning based control techniques in robotic. It covers the different learning techniques along with multiple RL algorithms in search of an effective solution to the specified problem.

2.1 Traditional v/s Learning Based Control Approaches

Traditional control system functions on the bases of **accurate analytical model**. However, due to the complexity of contemporary robot systems, obtaining precise analytical models, is a challenging task [2].

Moreover, the interaction of robots with **human-inhabited environment** which are typically **unstructured** and **highly uncertain**, traditional control system in such environment is most likely to diminution in terms of performance or even lead to a total failure [2].

Contrary to traditional controls, learning a model instead of **manually preprogramming,** it proves to be an effective option since the model can be **estimated directly from the real data** of the environment which ensures **robustness** and **environment generalization** to the system. Furthermore, even **unknown nonlinearities** can be immediately taken into consideration by learning models which will boost the efficiency of system, whereas traditional physics-based modelling tools and hand-crafted models both ignore them entirely [2].

However, online learning of such models learning approaches is required in order to **generalize the learnt models over a broader state space** and adjust the models to **time-dependent changes**. RL fill this gap effectively [2].

Moreover, RL can effectively replace the requirement of supervised learning dataset along with the requirement of kinematic equation's parameters by **reward function** [6].

	Requires accurate analytical models	Kinematic Equation Dependent	Manual Pre-programming	Supervised Dataset Dependent	Reusable, On Unseen Environment	Adaptive To Time Dependent Changes	Environment Generalization	Robust, Unknown Nonlinearities	Robust To Uncertainty	Training Time Efficiency	Training Resource Efficiency
Classical Control Methods	✓	✓	✓	✗	✗	✗	✗	✗	✗	✓	✓
Model Learning Control	✗	✓	✗	✓	✗	✗	✓	✓	✓	✗	✗
RL Control	✗	✗	✗	✗	✓	✓	✓	✓	✓	✗	✗

Table 1: Comparison between Classical controls, Model learning controls and Reinforcement learning controls

2.2 RL Policy Gradient Algorithms

To develop a sophisticated policy, policy learning based RL algorithms were extensively reviewed, in search of appropriate a stable algorithm.

Starting off with Deep-Q Networks (DQN) which led the bases for further development. DQNs are Q value function-based learning, aims to converge Bellman optimality equation, to attain maximum discounted Q value-based reward [9].

Although DQN is a promising technique but due to maximization over state and action pare value, its scope is limited to discrete action space. This limitation is effectively resolved by Deep Deterministic Policy Gradient (DDPG) models, by introducing a separate policy approximator and instead of directly maximizing over Q-value function it considers the output from policy approximator and effectively handles the continuous action space [8].

Despite of resolving the continuous action space limitation, the algorithm lacked in stable convergence of policy approximator(actor). Twin Delayed Deep Deterministic Policy Gradient (TD-3) [7] came up with improvement solution, ensuring the stable actor convergence and reduce overestimation bias propagation by critic along robustness to noisy states. Figure 4 briefly describes the significance of TD-3.

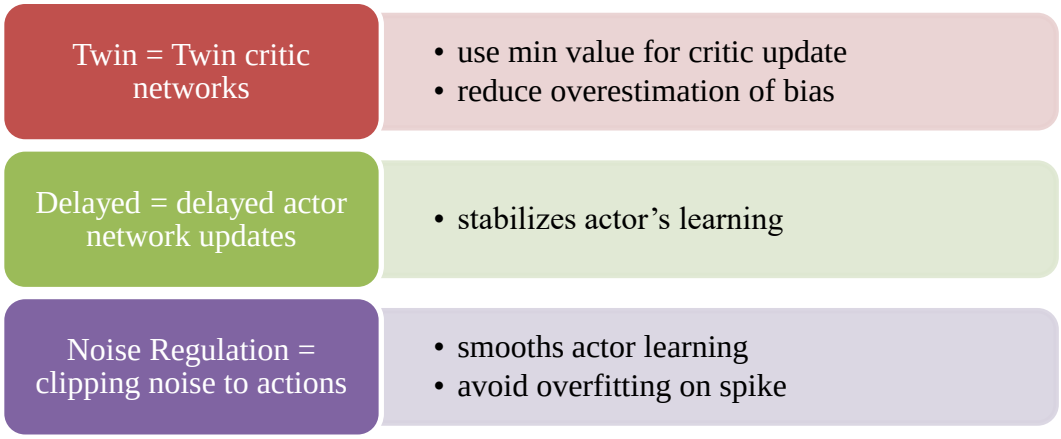

Figure 4: Properties of TD-3

Algorithm	Continuous Action Space	Stable Actor Update	Bias Propagation	Noise Regulation	No. of Approx. Networks
DQN	X	X	✓	X	2
DDPG	✓	X	✓	X	4
TD-3	✓	✓	X	✓	6

Table 2: Comparison between DQN, DDPG & TD-3 algorithm

Table 2 specifies the comparison between state of the are RL policy gradient algorithm, in terms of stability and performance.

2.3 TD-3 Implementation in Literature

TD-3 is implemented on numerous environments in literature including, Half-cheetah environment, Inverted pendulum environment, Reacher environment, Hopper environment and Walker-2D environment [7] primarily on 2-D environments. However, literature do not provide any implementation of TD-3 for differential drive race car environment.

CHAPTER 3: PROPOSED METHODOLOGY

3.1 Suggested Policy Design Flow

The suggested flow of policy design and training comprises of following features:

- **Environment:** where robot/agent explores the problem in hand by *experiencing* ***states space*** *of environment along with implementing action from* ***action space*** *of robot,* gain reward to evaluate its performance.
- **Reward function:** Criteria on which agent compares its performance, *it precisely binds environment state space with the problem in hand,* reward function is entirely problem dependent. Reward function proves to be the key to the solution.
- **Training RL Algorithm:** Technique under which robot/agent process the environment by making use of reward function in order to approximate the policy function which maps state to action in way to resolve the problem.
- **RL Agent-Model:** sophisticated learned policy approximator, that can be tested in unseen environments.

All these components are inter-connected and dependent on each other, in terms of performance to accomplish the anticipated goal i.e., a well-trained policy. Figure 5 blow effectively explains the flow of policy design.

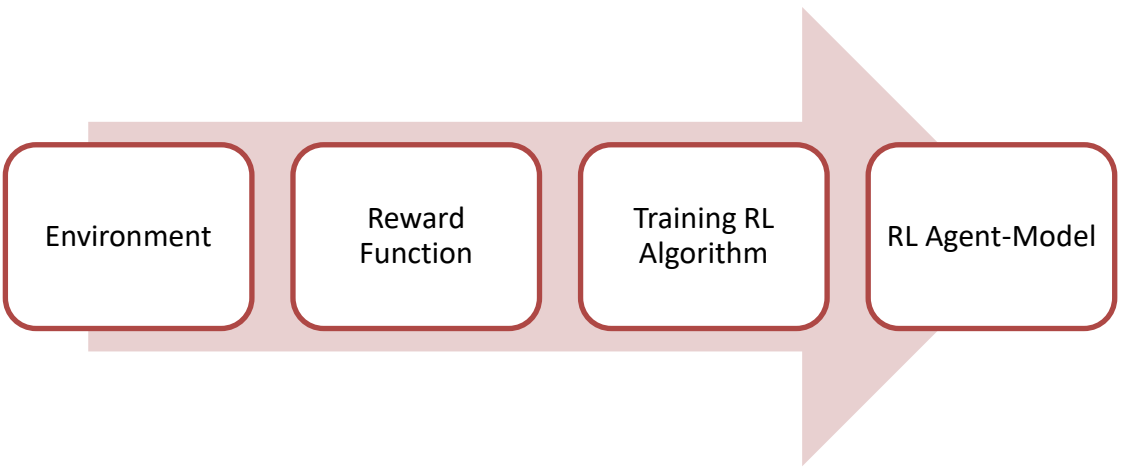

Figure 5: Policy design flow chart

3.2 Architectural Design

Proposed system architecture is shown in figure 6. RL agent aims to interact with the environment and perceive the state. Perceived state is analyzed on the bases of reward function to estimate a Value function, which in return contributes to policy estimation. Estimated policy is required to compute actions. Estimated action is executed in environment and next state is perceived on bases of which value function is updated. That way policy and value function estimator will contribute to improving one another.

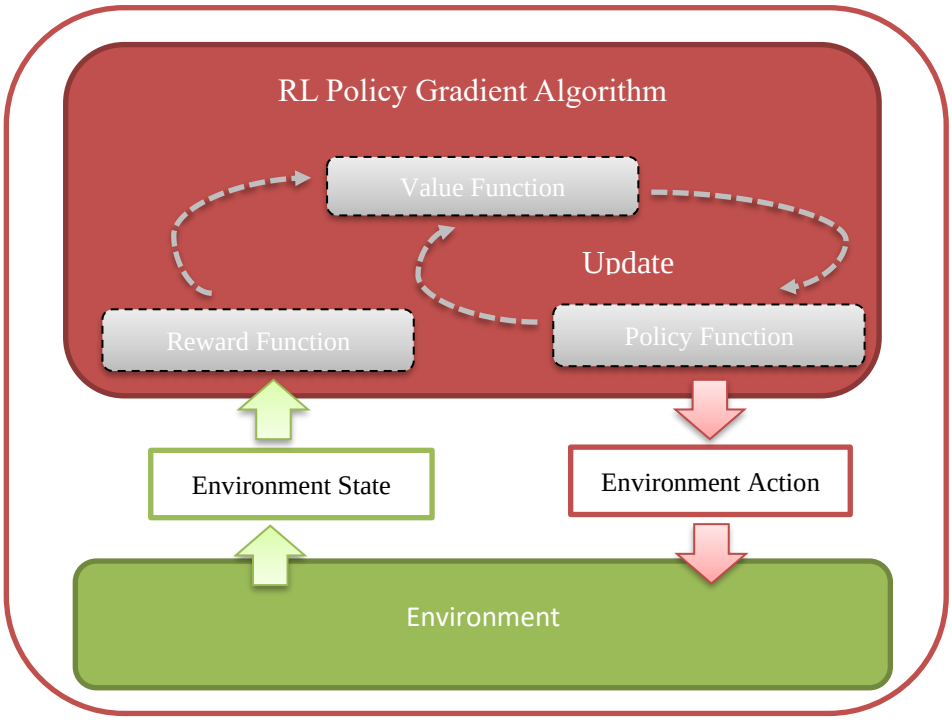

Figure 6: Designed architecture of system

CHAPTER 4: DESIGN & IMPLIMENTATION

4.1 System Architecture

A typical robotics system consists of **physical layer** dealing with hardware of robot and its surroundings i.e., world, however to analysis the perceived data and draw an appropriate conclusion in order to perform any action an **analytical layer** is required. Therefore, the proposed system figure 7, consists of multiple components synchronizing with each other to accomplish a unanimous goal.

Physical Layer:

- **Environment:** consisting of **robot model** and the **model of world.**
- **State Space:** describes the **robot's perception** of the world
- **Action Space:** describes the **robot's actuators state** to manipulate the world

Analytical Layer:

- **Reward function:** criteria on which perceived state is analyzed on the bases of defined goal.
- **Twin delayed deep deterministic policy gradient-TD3:** develop a generic policy on the bases of reward and state space to accomplish the defined goal

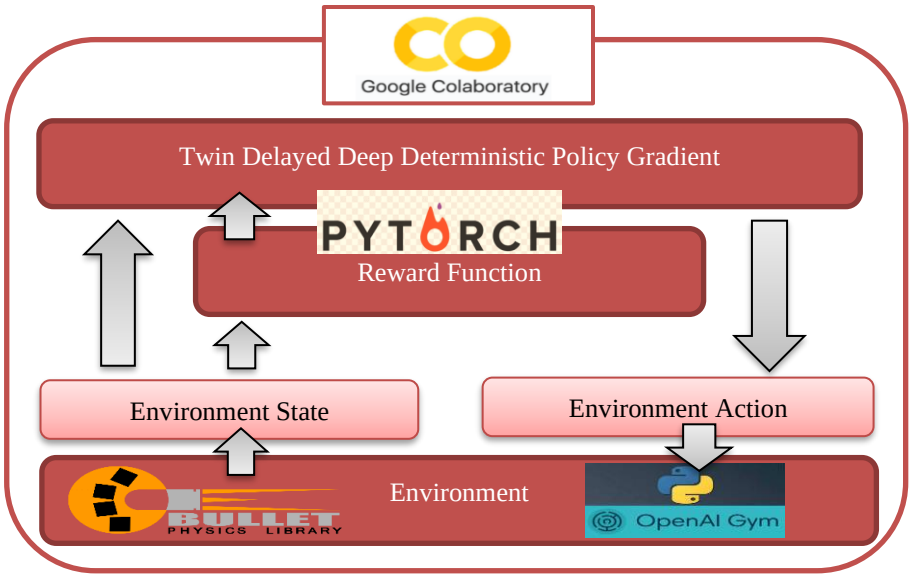

Figure 7: Implemented architecture of system

As described in the figure above tools used to develop the systems are, environment: Properties of physics are implemented by, PyBullet- opensource physics engine-based library. Model of world is implemented by OpenAI Gym environmnet, specifically designed and extensively used to simulate RL problems. However, Reward function and TD-3 is implement using Pytorch, a well-recognized tool for Machine learning problem domain. Whereas the whole system collaborated under Google Collaboratory.

4.2 Environment Specifications

To train and evaluate the performance of system on differential drive, race car model in Open AI Gym environment is utilized. Race car is equipped with LIDAR sensor, to incorporate obstacle detection. LIDAR of 100 rays is fixed to Hokuyo joint covering the front portion of the car, visualized in figure 8. A forward moving path environment was designed, bounded by wall from all four sides, following all the constrains of typical world model of Gazebo to train the RL agent, as shown in figure 9.

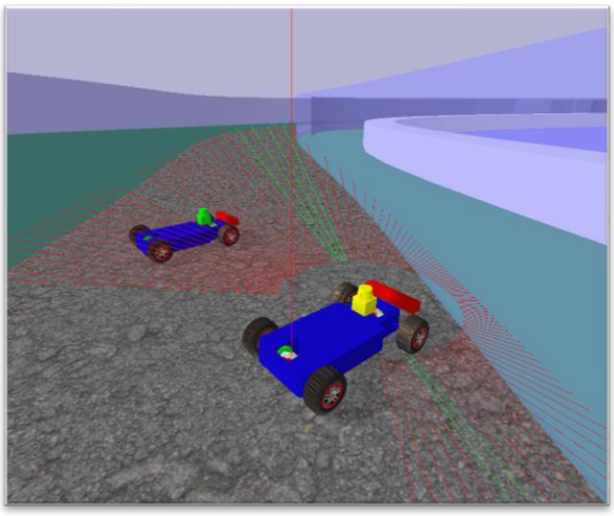

Figure 8: Differential drive car model

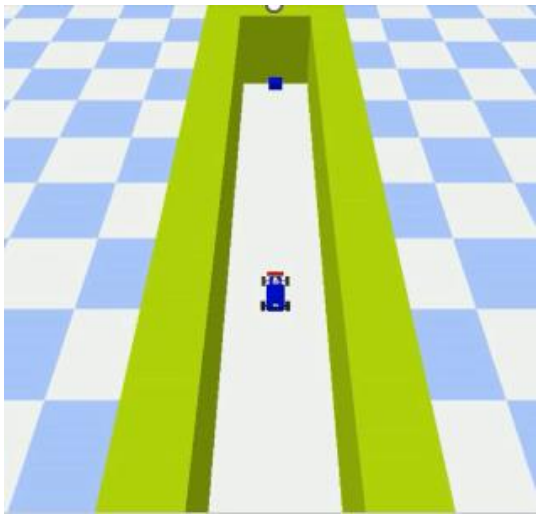

Figure 9: Designed training environment

4.3 State Space Design

State space is the only visible interpretation of environment for the agent. Therefore, it must be comprised of factor, effective enough for agent makes sense of its environment, consisting of enough information to achieve the desired policy. Thus, a continuous state space is designed to ensure **localization and mobility** towards the goal along with **obstacle detection,** state space consist of following component: **robot's base position and orientation** along with **distance to goal** this adds *localization.* To ensure *mobility in the direction of goal* **robot's linear velocity** and **robot's angular velocity** is added. However, **active lidar ray** along with the **lidar hit point** are for *obstacle detection*. Figure 10 visually describes the state space, dimension of entire state space is (1 × 414).

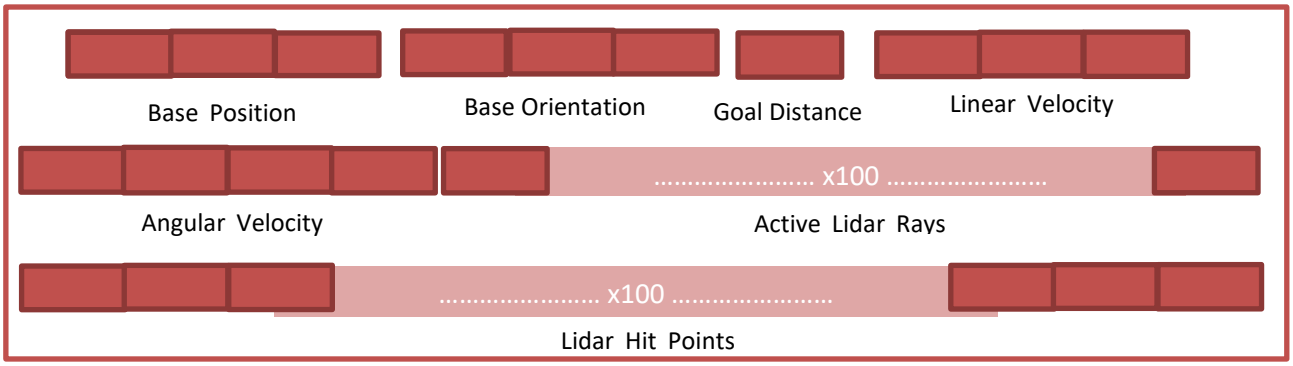

Figure 10: Designed state space

4.4 Action Space

Action space of robot consist of 2 components, shown in figure 11. Both the components are in continuous in nature bounded by [-1, 1], dimension of action space is (1 × 2).

Figure 11: Action space

4.5 Reward Equation Design

Reward function is the criteria on which agent compares its performance, *it precisely binds environment state space with the problem*, to attain the desired goal.

Correlation between designed state space and reward function guarantees the solution. Therefore, for continuous state space a continuous reward function is required to be mapped over the continuous action space. Reward function is entirely problem dependent and proves to be the key to the solution.

To attain a compatibility between state space and reward function, reward function comprises of 4 components, each component adds a distinct feature to achieve a step toward the goal. All the components are normalized to get a continuous composite reward equation (5) bounded between -1 and 1, with -1 being the worst 1 being the best as it approaches goal. Reward function consists of following components:

To avoid collision, **collision reward** is one of the components. It generates a sharp -1 signal for colliding with the wall and 1 otherwise.

$$collision\ reward = \begin{cases} -1, & distance\ to\ wall < 0 \\ 1, & Else \end{cases} \quad (1)$$

To add perception about the goal, and encourage the displacement in the direction of goal, **closeness to goal** is incorporated (2). Where *current distance* is the Perpendicular distance from the car base and goal and *total distance* is perpendicular distance between car base and goal at initial position i.e., start of episode. *Closeness to goal* is bounded between -1 and 1, approaches -1 if car's displacement is away from goal and 1 if it approaches goal.

$$Clossness\ goal\ reward = 1 - \frac{current\ distance}{total\ distance},\ [-1, 1] \quad (2)$$

To boost the mobility straight in the direction of goal i.e., forward in y-direction, **linear velocity reward (3)** suppresses the velocity in x-direction and encourages high velocity in y-direction. *Resultant velocity* of system is bounded between 1 and -1 therefore, max attainable velocity in either direction (x or y) is 1 and minimum is -1 in case the other component is suppressed to 0. So, the difference between y component and x component of linear velocity will be bounded between 1 and -1 representing movement in positive and negative direction of axis respectively. However, to double the weightage of linear velocity, factor of 2 is multiplied, to avoid local maxima.

$$linear\ velocity\ reward = 2 * \left(Y_{velocity} - \left|X_{velocity}\right|\right), \quad [-2, 2] \qquad (3)$$

To avoid deflection from the straight path by suppressing angular velocities of car, **angular velocity reward component (4)** is added, it ensures sharp -1 reward if angular velocity exceeds from the empirically set threshold value however, reward approaches to 1 if angular velocities in all the directions are suppressed to zero.

$$angular\ velocity\ reward = \begin{cases} -1, & \left|Y_{angular}\right| + \left|X_{angular}\right| + \left|Z_{angular}\right| > 0.09 \\ 1 - \left|Y_{angular}\right| + \left|X_{angular}\right| + \left|Z_{angular}\right|, & Else \end{cases} \qquad (4)$$

The composite, normalized, continuous reward equation, combining the components of reward function is as follows:

$$Reward = \frac{\begin{array}{c} collision\ reward + \\ closseness\ to\ goal\ reward + \\ linear\ velocity\ reward + angular\ velocity\ reward \end{array}}{5}, [-1,1] \qquad (5)$$

4.6 Implementation of TD3

TD3 is implemented to enable RL-agent to develop an *action approximator know as policy* which maps give state of environment to action space of the robot, maximizing cumulative reward, to act in the quest of solving the given task as shown in figure 12.

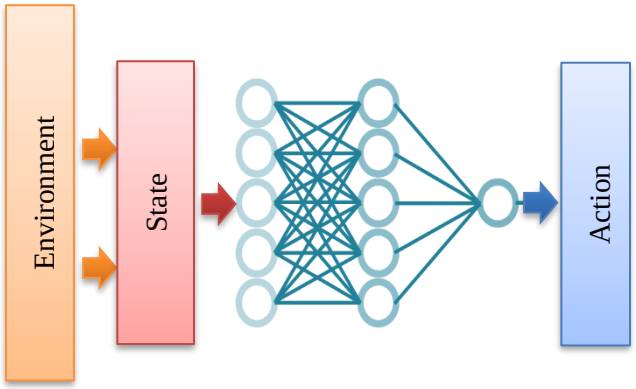

Figure 12: Policy based RL-agent

To train an RL-agent to solve a problem, it requires an environment representation for agent to work on. *Tuple of Markov decision process representing environment* is used. **Environment tuples <S, A, R, S'>** consist of:

- ***current state S***
- ***current action A***
- ***current reward R****, received by following action A from state S*
- ***next state S'*** *which is led by action A from state S*

However, to *add perspective of environment tuple to agent*, in order to *analyze the state and action set* and *take an appropriate action* in the direction of solution i.e., attain maximum cumulative reward, value function and policy approximators are required.

Therefore, TD-3 consists of 6 neural networks in total, which converge simultaneously to attain a stable system. 4 neural networks are Q-value approximators known as critic networks and 2 are policy approximators known as actor networks. Both **Q-value network** and **policy networks** works hand in hand to learn an appropriate policy out of given **environment tuple**.

Critic networks or Q-value networks, 2 out of 4 are considered as ***target Q-networks*** represented as gray block of neural networks in figures below, with frozen weights and are not updated on regular bases i.e., on each iteration to *avoid update of critic in an inappropriate direction and getting overfit on current limited experience.* However, the other pair is ***current Q-network*** represented as red block of neural networks, is used to assign Q-value to the current state

and action set and are step ahead of target Q-networks. This pair of networks is updated on regular bases on each iteration and on every environment experience.

Actor network or policy network, 1 of the two networks is ***target policy*** network, represented as gray block in figures below, with frozen weights and are not updated on regular bases i.e., on each iteration to *avoid actor getting overfit on an unstable critic update*. However, other policy network is known as ***current policy network*** represented as blue block in figures below, likewise they do not experience regular weights update to *avoid unstable actor update on unstable critic update*.

4.6.1. Critic Network Update

Critic network as evident from the name adds criticism to the performance of the agent in the environment. Its takes state and action pair and assign it the value in terms of received reward over time steps. Formally, critic network estimates the **expected return** i.e., cumulative reward over the discounted steps given the state and action pair.

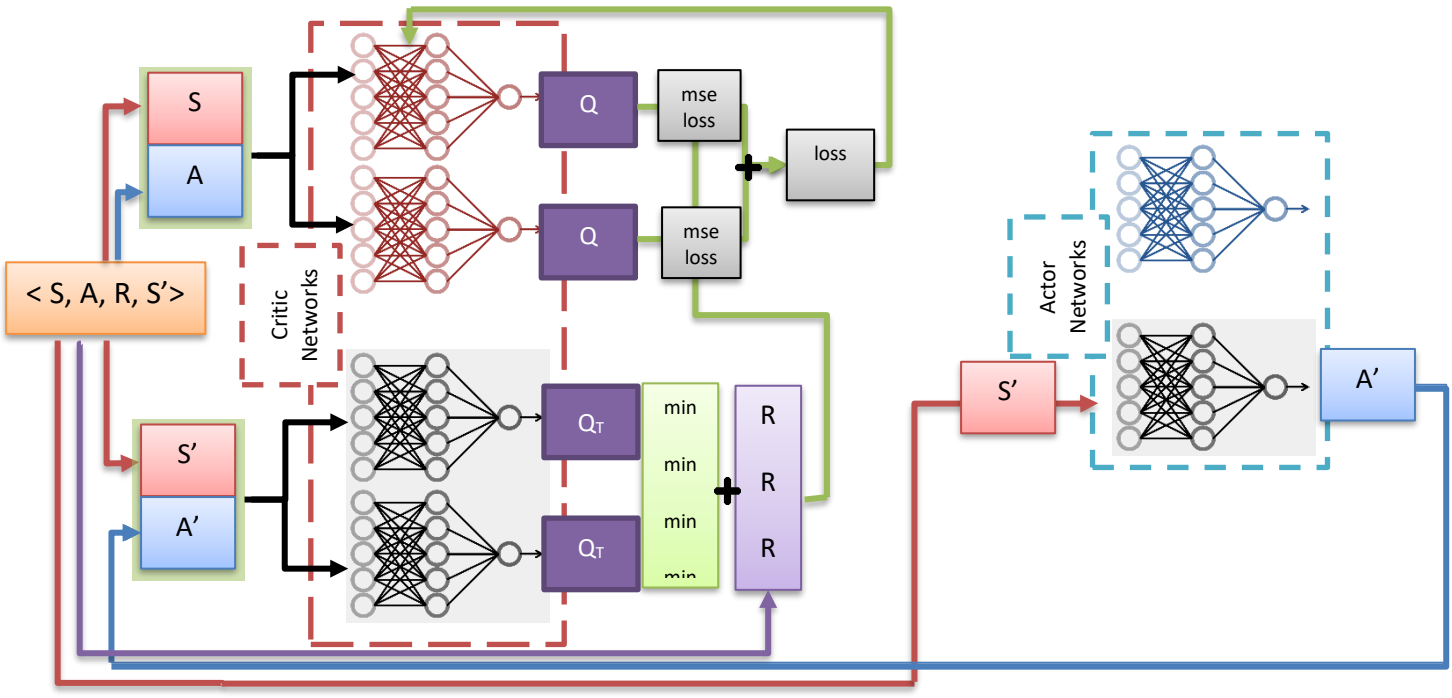

Figure 13: Critic network update

Therefore, to train an estimator it requires a target to be approached and minimize the difference between ***estimated return*** and ***estimated target return*** by backpropagating the error. This process in referred as critic network update, the architectural flow of critic network update is briefed in figure 13.

Starting with the target Q-value $\mathbf{Q_T}$ generation, received environment tuple is utilized, next state **S'** is feed to frozen actor network (gray actor network) to estimate the next action **A'**. S' and A' are feed to frozen critic networks (gray critic network) to get Q_T value (estimated discounted reward from given S' and estimated A'), min of Q_T is summed with **R** actual reward received from the transition from S to S' following A. *Minimum of Q_T is used to suppress the propagation of overestimation bias in Q network and stabilize the training.*

To estimate the current Q value **Q**, current critic networks (red critic networks) are exposed to **S** and **A** from environment tuple. ***Sum of mean square error loss*** *is computed for* ***each Q value*** *with the* ***estimated Q target (Q_T + R)*** and propagated backward to update the current critic network in the direction of loss minimization.

$$Loss(Q) = E\left[\left(Q_1(s,a) - \left(r + \gamma\, min\left(Q_T\left(s', \pi_{target}(s')\right)\right)\right)\right)^2\right] + E\left[\left(Q_2(s,a) - \left(r + \gamma\, min\left(Q_T\left(s', \pi_{target}(s')\right)\right)\right)\right)^2\right]$$

This update of critic network is conducted *once on every time step*. However, Target critic networks are updated with **soft copy** of current critic networks *once after every 2-time steps*, to keep appropriate difference between target and current critic network, to avoid convergence in an inappropriate direction.

4.6.2 Actor Network Update

Actor network or policy network are the estimated distribution of action given state to maximize the expected reward. Therefore, actor network needs to ensure an appropriate action selection to enhance the performance of agent in the given problem setting. Action is analyzed by the given critic network and updated in the direction to maximize the return.

A stable actor is ensured by a stable critic therefore, *actor is update once in every two timesteps* giving critic the time to get mature over the experiences of environment as critic is updated on every timestep. Figure 14 defines the actor update steps.

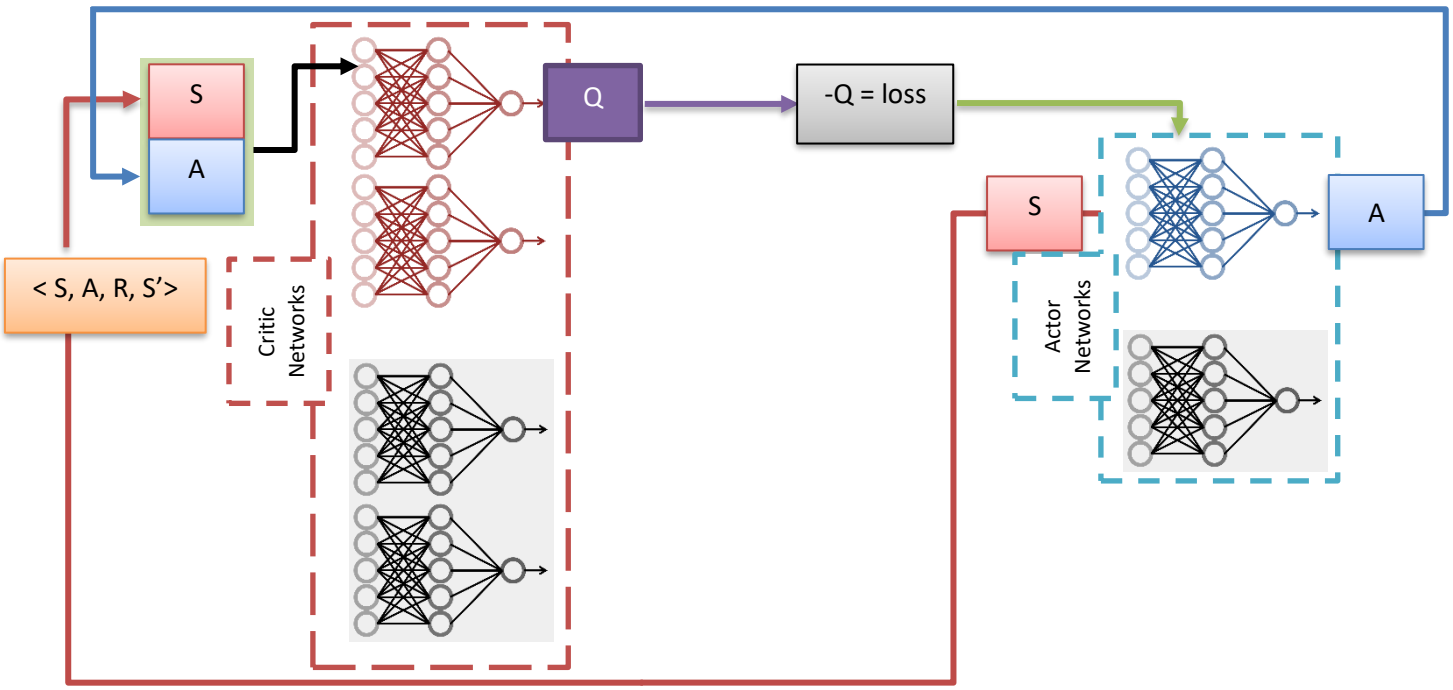

Figure 14: Actor network update

Current state **S** from the tuple is utilized it is feed to the *current actor network* $\boldsymbol{\pi}$ to get ***estimated current action*** **A**. To draw the analysis on the estimated action it is exposed to current critic network along with the current state, as a result $\boldsymbol{\pi}$ *is updated on the bases of generated Q-value in a way that selected* ***action A maximizes the Q-value***.

Therefore, loss to be backpropagated to actor network is **-Q**, negative sign is appended to solve maximization problem using gradient decent.

$$Loss(\pi) = -Q\big(s, \pi(s)\big)$$

However, **soft copy updates** i.e., probabilistically updating weights of frozen actor networks $\boldsymbol{\pi_{target}}$ by current actor network are performed once in every 2 timesteps, to ensure appropriate difference between the two networks and stabilize the training by restraining the target network from divergence.

4.6.3 Critic Network Architecture

All critic networks are identical in terms of architecture, consisting of one input layer and 2 hidden layers following the output layer. Input layer consisting of 416 neurons ensuring compatibility with 414 components for state space and 2 components of action space, nonlinearity is ensured by incorporating Relu activation function. First hidden layer comprises of 400 neurons along with Relu nonlinearity following 300 component based linear hidden layer 2 converging the flow to signal neuron comprising output layer. The finalized architecture of critic neural networks figure 15 is designed to attain the best performance of system.

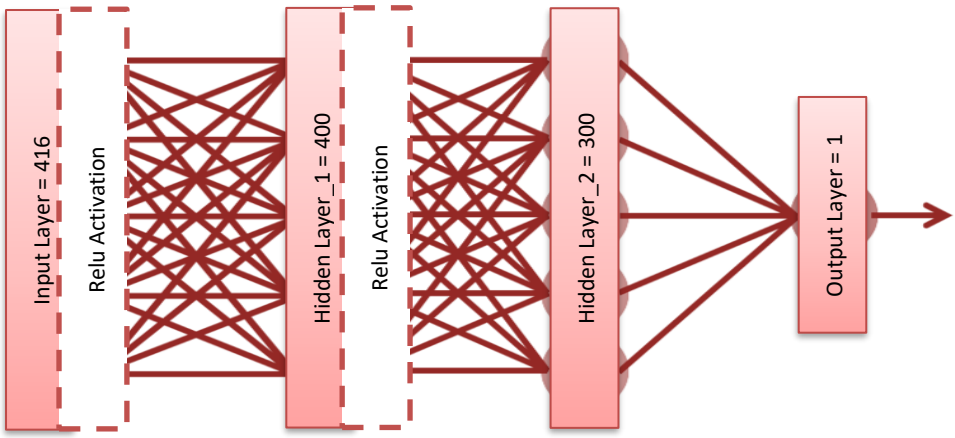

Figure 15: Critic network architecture

4.6.4 Actor Network Architecture

Actor networks consisting of an input layer consisting of 414 components to accommodate the state space, along with 2 hidden layers, comprising of 400 neurons and Relu nonlinearity and 300 neurons with tanh nonlinearity respectively. Finally converging to 2 neurons based linear output layer, to accommodate 2-dimensional action space. The architectural design of actor neural network figure 16 is based on overall system performance enhancement.

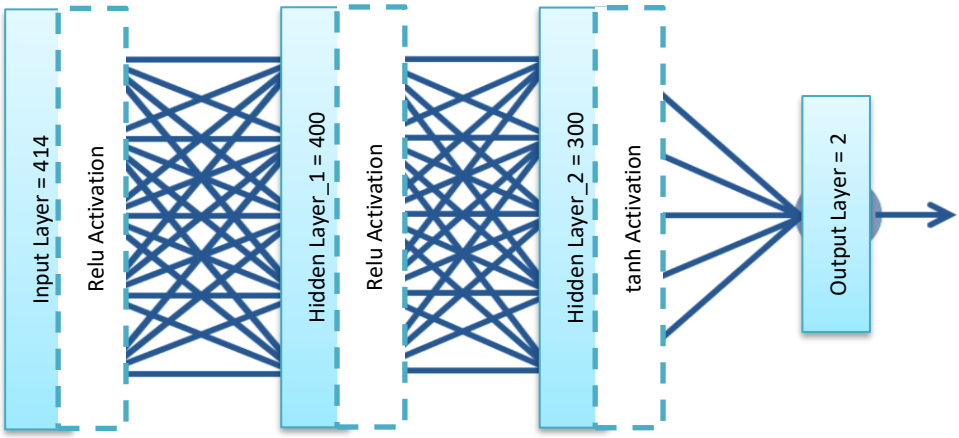

Figure 16: Actor network architecture

4.6.5 Hyperparameters of System

All hyperparameters are tuned to attain the best possible performance of the system. Hyperparameters are listed in the table.

Key	Value
BATCH_SIZE	100
EVAL_FREQUENCY	5000
EXPLORATION	5000000
EXPLORE_NOISE	0.1
GAMMA	0.99
NOISE	0.2
NOISE_CLIP	0.5
OBSERVATION	10000
POLICY_FREQUENCY	2
SEED	0
TAU	0.005

Table 3: List of TD-3 hyper-parameters

CHAPTER 5: TRAINING & EVALUATION

5.1 Training Results

The training of implemented system has acquired precisely converging graphical results. Losses of both actor and critic network are converging effectively justifying the theoretical explanation and maximizing the received reward. Best trained policy was attained at 26k steps of training, generated average reward value of 235 when evaluated over 10 episodes.

Actor network's loss plot figure 17 shows loss decreasing against training time step. Stared from zero and settled at around -30 after 150k training step. As actor's loss is negative Q-value of estimated action against the given state, therefore, loss minimization is proportional to Q-value maximization. Lower the actor's loss higher the Q-value for the estimated action.

Critic network's loss defines the closeness between estimated target Q-value and estimated current Q-value, minimum difference i.e., closer to zero ensures convergence. Critic's loss plot figure 18 shows, starting from zero due to identical target and critic network, progressed to 4 after being exposed to 150k steps of training. Proving to be acceptable results as gained reward is increasing with time steps.

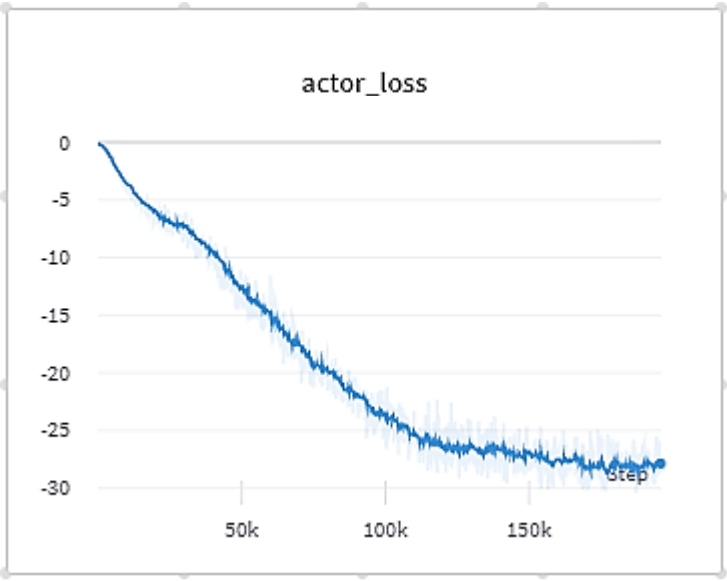

Figure 17: Actor's convergence per training step

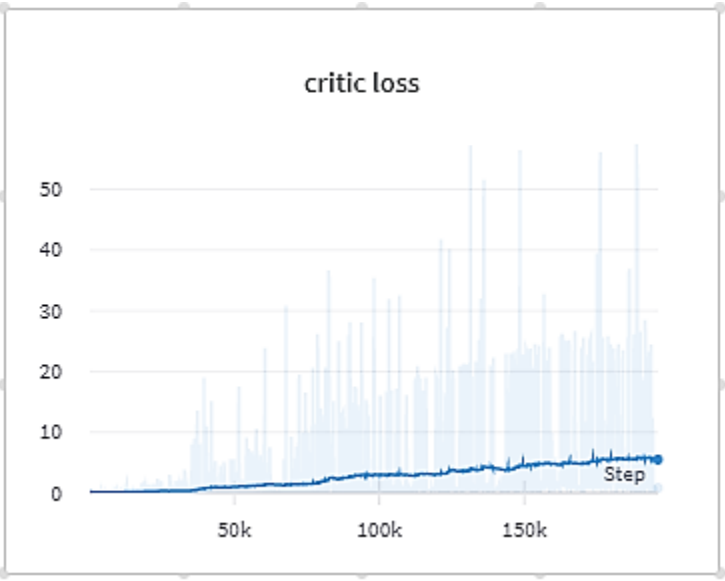

Figure 18: Critic's convergence per training step

To estimate the performance of training models, models are saved on regular intervals to be evaluated in terms of gained reward, as shown in figure 19. High reward generating peaks and stable plateau of the saved models plot are evaluated. Best performing and highest reward producing model, peak at 26k steps generating reward value of 235 was selected for the testing on unseen environment.

However, overall performance of training can be evaluated by average reward plot against training steps figure 20. It started increasing after a sharp decline on the initial training steps, this ensures agent exploring environment and eventually converging in the anticipated direction.

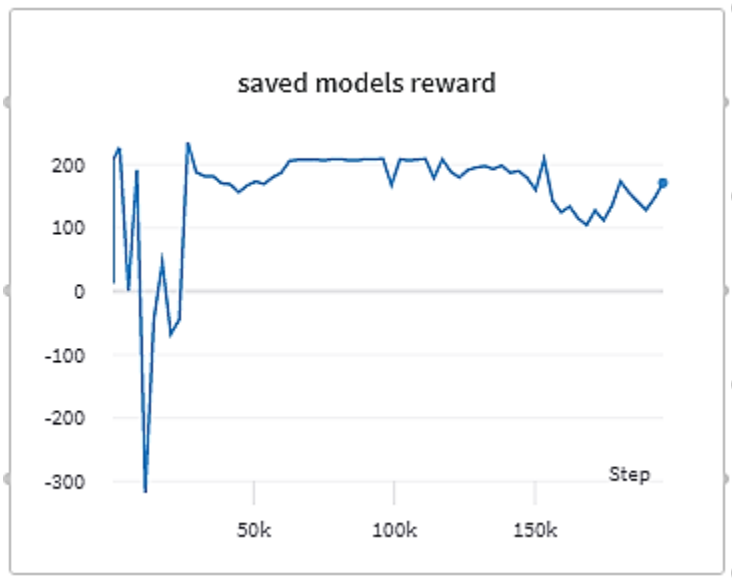

Figure 20: Saved models reward on regular training intervals plot

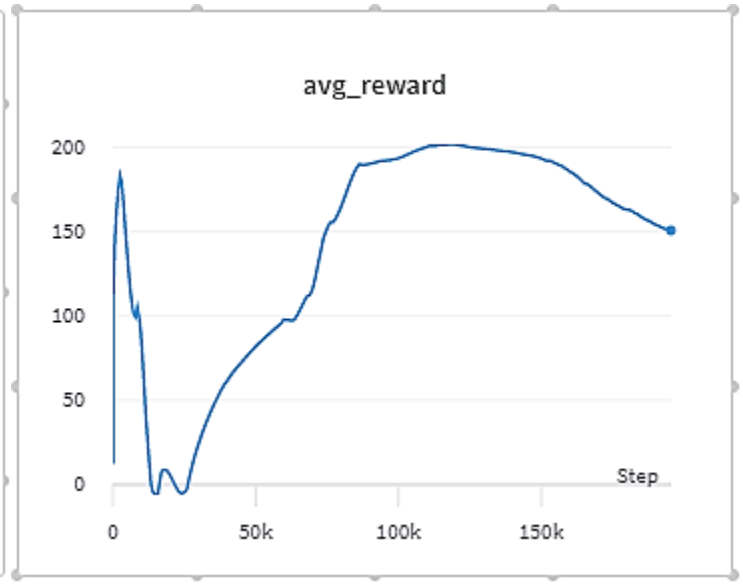

Figure 19: Average reward of training per training steps

Reward per episode and reward per step againts traning steps are ploted in figure 21 and figure 22 respectively, to analyze per step traninig performance and overall episode performance.

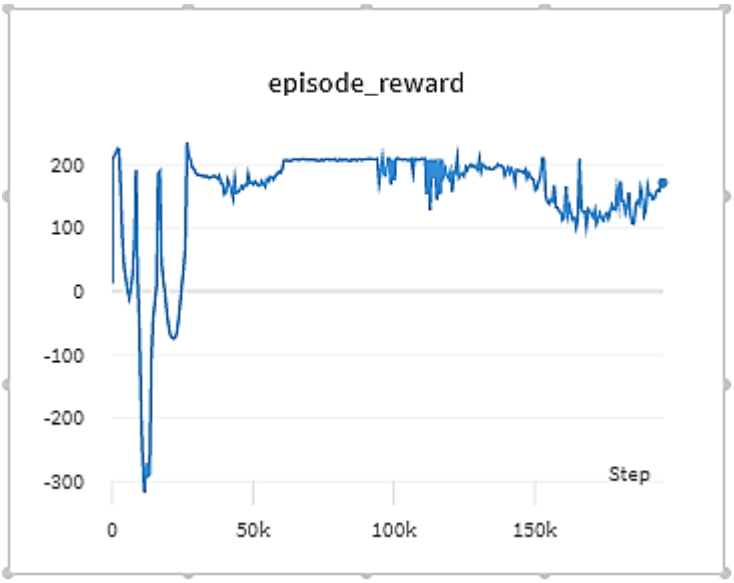

Figure 21: Episode reward per training step plot

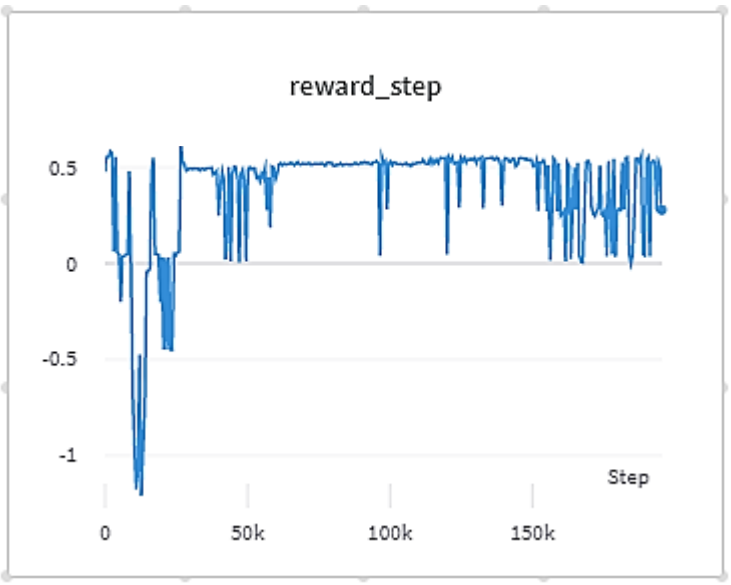

Figure 22: Step reward per training step plot

5.2 Testing Results of TD3

Best reward gaining policy on the training environment was selected to be tested on the unseen environment. During testing the policy performed precisely in new environment, gaining reward. Policy was tested of 3 different environment, which agent have not experienced before:

- Dynamic goal environment
- Boundary free environment
- Continuous path environment

5.2.1 Dynamic Goal Environment

Agent's performance was evaluated on a continuous path environment in which goal is shifted forward, as the robot approaches the defined vicinity of goal figure 23. The performance of agent can be evaluated by reward per step graph figure 24. Starting from less than 0.5, increasing on every step approaching 1. However, it is experiencing a sharp decline when the goal is shifted but avidly recovers the gaining reward.

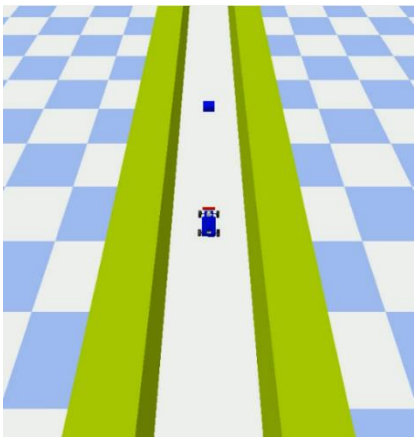

Figure 23: Dynamic goal environment

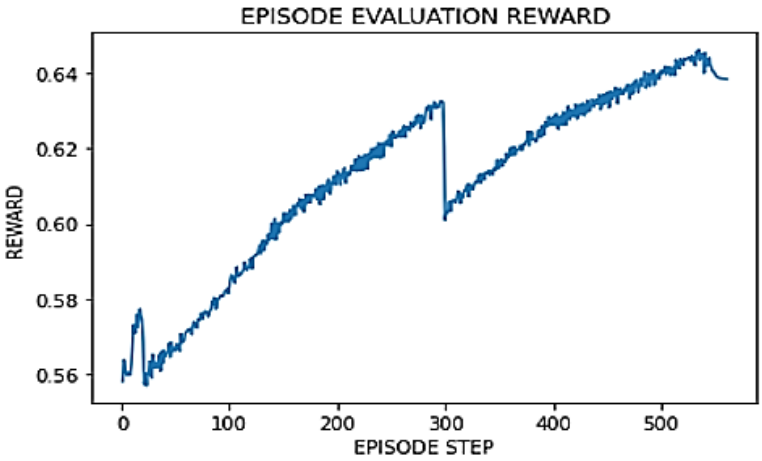

Figure 24: Reward gain plot for dynamic goal environment

5.2.2 Boundary Free Environment

Boundary free environment neglects the notion of obstacles by removing the boundary wall of the environment, visualized in figure 25. This can effectively change the knowledgeable state space values, as agent will experience no active lidar values unlike training environment. Evaluation results clearly states that change in environment is not affecting the agent's performance, evident in gained reward plot figure 26.

Reward graph states the increasing reward gaining policy performance, starting from lesser than 0.44 reward and progressing to 0.54 reward in just 400 steps episode, as agent approaches in the direction of goal, without deviating from the straight path.

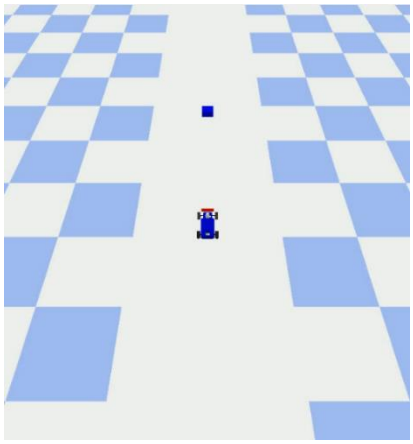

Figure 25: Boundary free environment

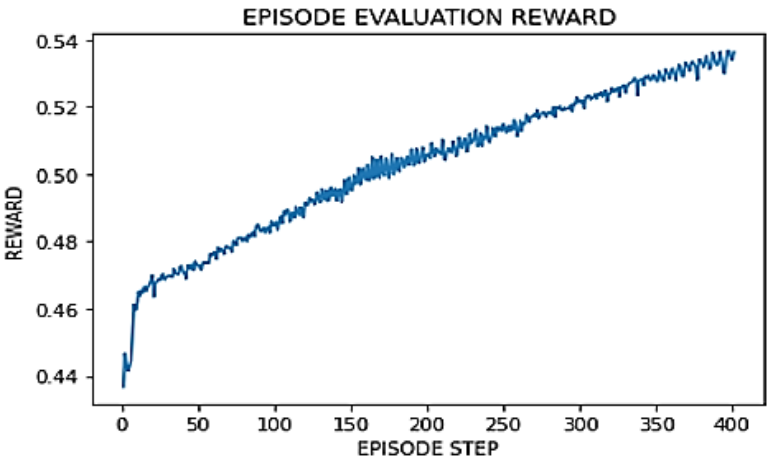

Figure 26: Reward gain plot for boundary free environment

5.2.3 Continuous Path Environment

To evaluate the agent on continuous path visualized in figure 27, front wall of the training environment is removed which was the closest obstacle to goal in the training environment. Subsequently this changes the experienced states of environment for agent.

Performs of agent can be evaluated by gained reward plot figure 28. Reward graph states the increasing reward gaining policy performance, starting from lesser than 0.44 reward and progressing to 0.56 reward in just 500 steps episode, as agent approaches in the direction of goal.

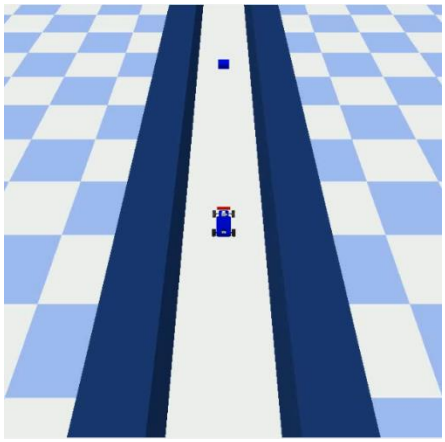

Figure 27: Continuous path environment

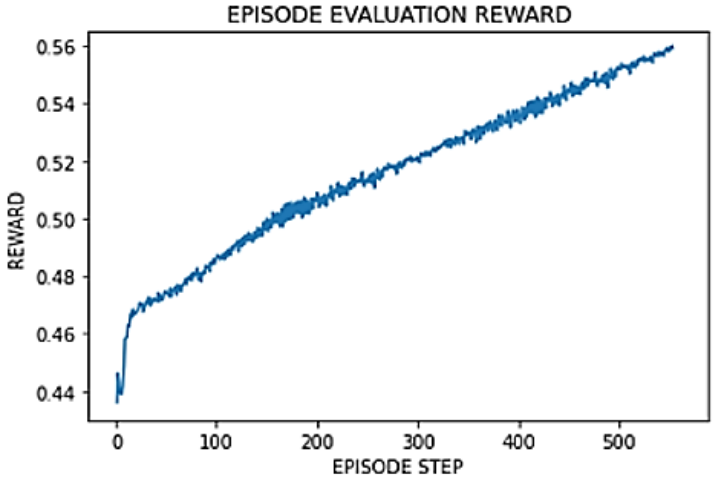

Figure 28: Reward gain plot for continuous path environment

5.3 TD3 & DDPG Comparison

To conduct an appropriate comparison between TD3 and DDPG both the agents are trained and tested under similar conditions.

5.3.1 DDPG Training

To validate the selection of RL algorithm comparison on the bases of training performance is conducted, between the designed TD3 agent with DDPG agent. For that reason, a DDPG agent was trained under similar conditions. Performance of training was evaluated on the bases of training stability and convergence time required for agent. After evaluation of several peaks of saved models during training of DDPG, best policy selected is the peak after 100k steps of training generating reward value of 210.

Convergence of actor is evident from figure 29 actor's loss over training steps. It started from zero decreasing over training step approached -20 over 250k training steps. However, is not fully converged even after 250k of training. Critic's loss is converging well figure 30, after 250k training steps loss value is around 1.5 which is close to zero.

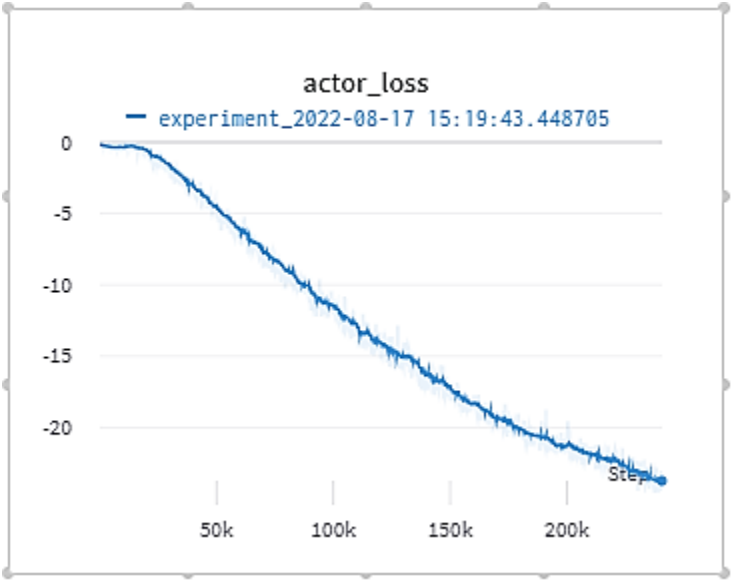

Figure 29: DDPG actor convergence against time step plot

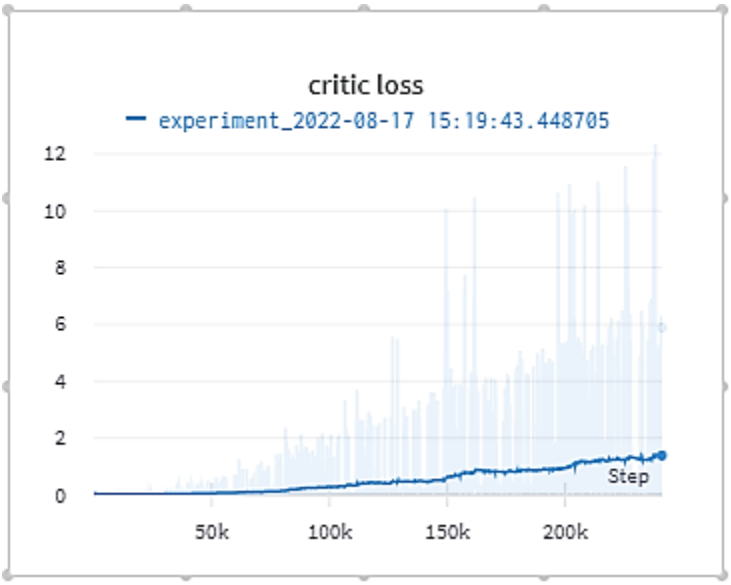

Figure 30: DDPG critic convergence against time step plot

Although average reward against training is increasing figure 31 but Rewards against training steps plots figure 32, figure 33, figure 34 shows instability in training. As its reward plots are extremely noisy, comprising of numerous peaks with barely any smooth plateaus. These plots are highlighting the instability in the training system of DDPG.

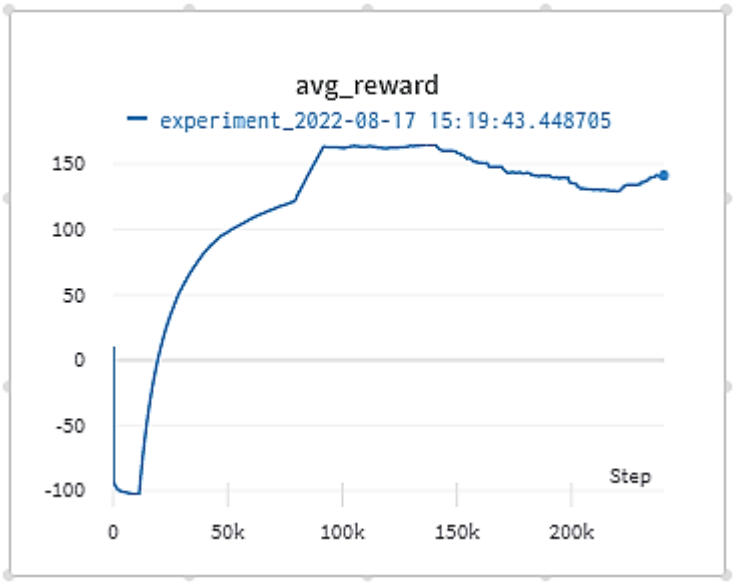

Figure 31: Average reward of training per training steps plot

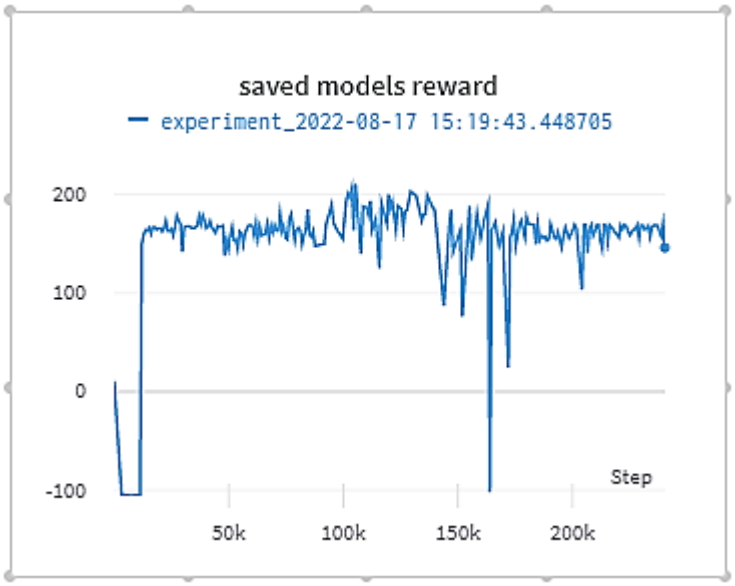

Figure 32: Saved models reward on regular training intervals plot

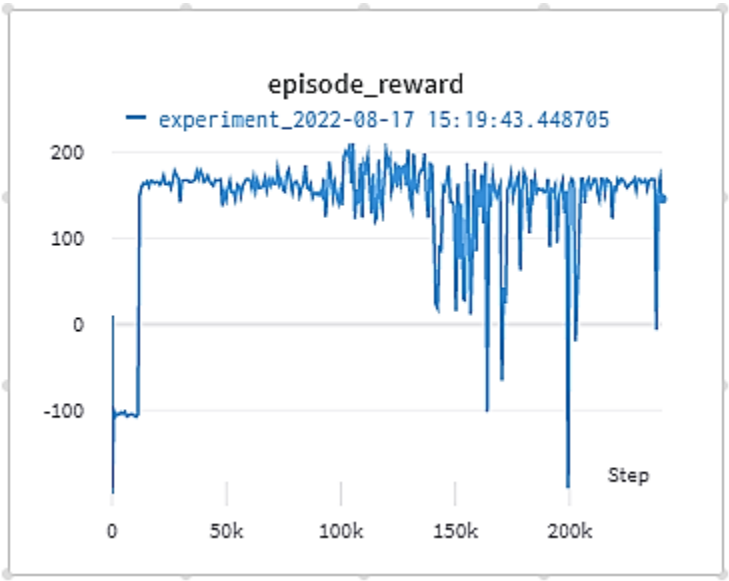

Figure 34: Episode reward per training step plot

Figure 33: Step reward per training step plot

5.3.2 TD3 v/s DDPG Training

To conduct justifiable training performance comparison between TD-3 and DDPG, both the algorithms were trained under similar conditions. Values of corresponding hyper parameters were kept equal while training both TD-3 and DDPG. However, performance on the bases of convergence and reward maximization over training steps were observed, results are summarized in table 4.

Algorithm	Batch Size	Actor's Update Frequency	Discount Factor	Actor Learning Rate	Critic Learning Rate	Training Steps	Actor Loss Convergence	Critic Loss Convergence	Max Avg Reward	Max Episode Reward
TD-3	100	2	0.99	10^{-6}	10^{-6}	200k	-30	5	201.86	235.27
DDPG	100	1	0.99	10^{-4}	10^{-2}	250k	-20	1	165.26	210.69

Table 4: Training Performance Comparison between TD3 and DDPG

Training updates of TD-3 and DDPG were based on batch size of 100 steps, discount factor γ was set to 0.99 to maximize the reward foresight. However, algorithm dependent hyper parameters were tuned accordingly, e.g., actor update frequency was set 2 for TD3 and 1 for DDPG. Although, architecture of both actor and critic networks were kept similar for both the algorithms but learning rate differed to aid the convergence of different setups. For TD3 both actor and critic observed same learning rates of 10^{-6}. Whereas actor and critic learning rate for DDPG were 10^{-4} and 10^{-2} respectively.

Despite of keeping the similar training architecture convergence performance of the two algorithms visibly differ from each other, as summarized in Table 1. Actor network loss of DDPG has converged to -20 and critic network loss has converged to 1 with maximum average reward of 165.26 even after 250k steps of training. In contrary to that, actor network loss of TD3 has converged to -30 and critic network loss has converged to 5 with maximum average reward rising to 201.86 in just 200k steps of training. However, maximum reward generating policy for DDPG

is with episode reward value of 210.69 on 105k training steps and for TD3, best policy is generating 235.27-episode reward value on 26k training policy.

Moreover, instability in training of DDPG is evident by noisy reward graphs. Overall training performance of two algorithms suggest that TD3 outperformed DDPG in the designed setting. Training TD3 is resource efficient and stable as compared to DDPG.

5.3.2 TD3 v/s DDPG Testing

To compare the performance of TD3 and DDPG agents in unobserved environment, both the agents were tested in all three designed testing environments, namely:

1. Dynamic goal environment
2. Boundary-free environment
3. Continuous path environment

In Dynamic goal environment, performance of trained agent was evaluated on a continuous path in which goal is shifted forward, as the robot approaches the defined vicinity of goal, environment shown in figure 35a. Figure 35b states the performance of TD3 agent. Starting from less than 0.5, increasing on every step approaching 1 experiencing max reward of 0.63. It is experiencing a sharp decline when the goal is shifted but avidly recovers by gaining reward. Episode ended with total reward 241.38 over 400 steps. However, performance of DDPG policy is relatively noisy, shown in figure 35c. Starting from 0.25 and approaching 1 with max reward experienced is 0.58. Episode ended with total reward of 217.58 over 400 steps.

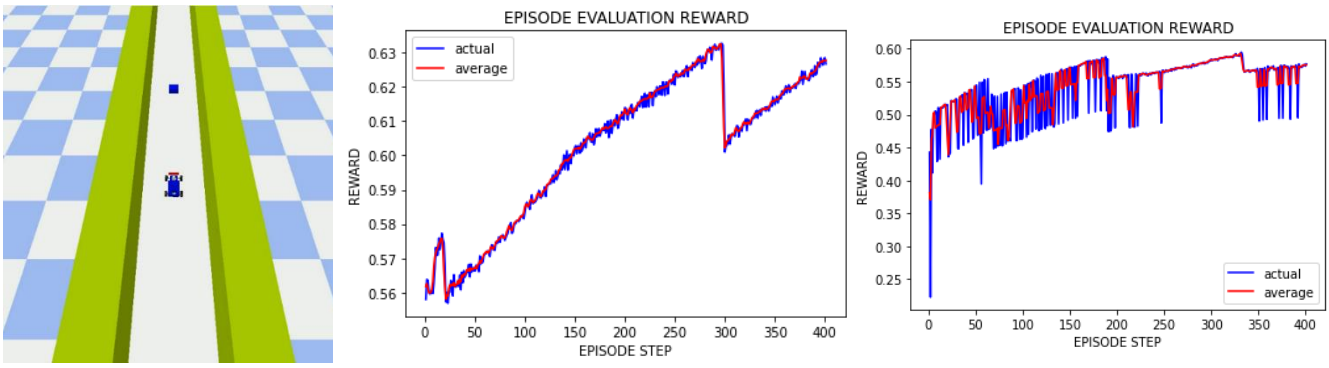

Figure 35: Testing results of TD3 and DDPG on dynamic goal environment

Boundary free environment neglects the notion of obstacles by removing the boundary wall of the environment, visualized in Figure 36a. This can effectively change the knowledgeable state space values, as agent will experience no active lidar values unlike training environment. Evaluation results of TD3 figure 36b, clearly states that change in environment is not affecting the agent's performance, evident by gained reward plot. Trend of reward graph shows the increasing reward gaining policy performance, starting from lesser than 0.44 reward and progressing to 0.54 reward, as agent approaches in the direction of goal, without deviating from the straight path. Total reward of 400 steps episode is 238.73. However according to reward plot of DDPG agent figure 36c, starting reward is 0.52 and it is increasing maximum up to 0.625. sharp spike on initial steps shows the noisy performance of policy. Total reward gained is 213.78 over the episode.

To evaluate the agent on continuous path visualized in figure 37a, front wall of the training environment is removed which was the closest obstacle to goal in the training environment. Subsequently this changes the experienced states of environment for agent. Performs of TD3 agent on this environment can be evaluated by gained reward plot figure 37b.

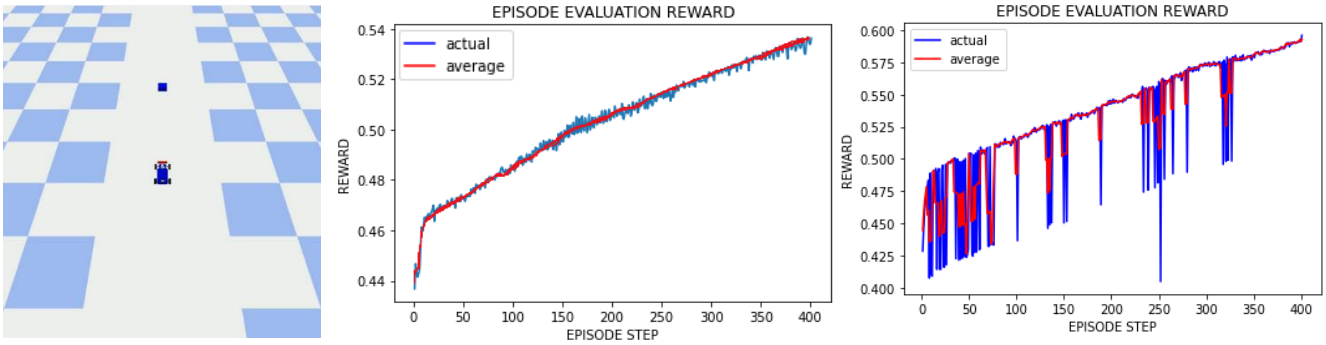

Figure 36: Testing results of TD3 and DDPG on boundary free environment

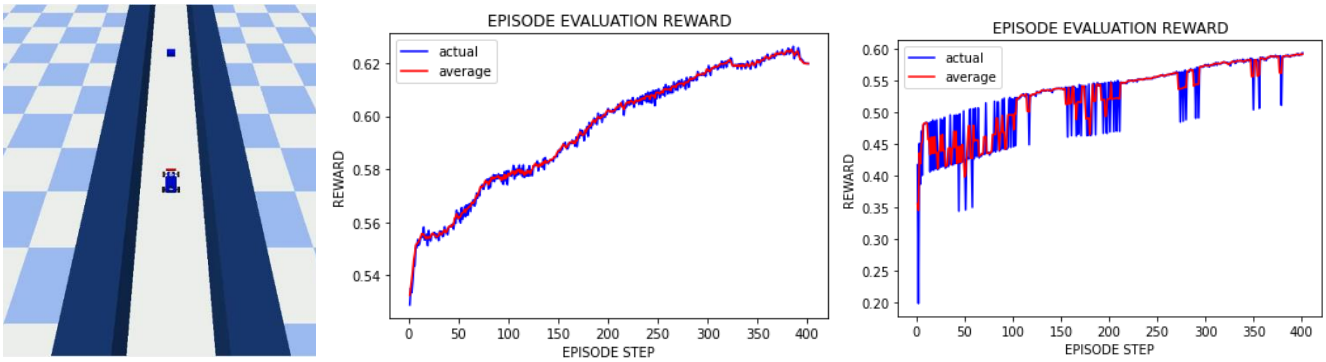

Figure 37: Testing results of TD3 and DDPG on continuous path environment

Graph states the increasing reward gaining policy performance, starting from lesser than 0.53 reward and progressing to 0.625 reward in just 400 steps episode, as agent approaches in the direction of goal. Total reward of episode is 238.60. Performance of DDPG agent on continuous path environment is shown in figure 37c, starting from 0.20 and reaching 0.60. However, experienced reward plot is considerably noisy. Total reward of episode is 213.79.

Testing Results of DDPG and TD3 on unseen environment clarified the performance and stability of TD3 agent over DDPG agent. Testing results stated in table 5, shows the performance of both the agents in previously unobserved environment, tested over 50 episodes, where each episode is of 400 steps. In which TD3 out-performed DDPG in all the environments, generating average rewards of 241.38, 238.72 and 238.60 in dynamic goal environment, boundary free environment and continuous path environment respectively. In contrary to that, DDPG generated 217.58, 213.79 and 210.69 in dynamic goal environment, boundary free environment and continuous path environment respectively.

Environments	RL Agent	Episode Steps	Total Evaluated Episodes	Average Reward
Dynamic Goal Environment	TD3	400	50	241.38
	DDPG	400	50	217.58
Boundary Free Environment	TD3	400	50	238.72
	DDPG	400	50	213.79
Continuous Path Environment	TD3	400	50	238.72
	DDPG	400	50	213.78

Table 5: Testing Results on DDPG and TD3 on unseen environments

CHAPTER 6: CONCLUSION AND FUTURE WORK

This thesis concludes by effectively training TD-3 for Differential Drive Race Car with continuous action space, to move forward in the direction of goal. This will exclude the need for differential drive dependent analytical module, by designing the reward function. The reward function correlates with the state pace of the environment to train a policy. Such policy effectively maps state space over the action space by maximizing the cumulative reward. The trained policy was effectively tested on unseen environments including the curved path providing sophisticated results.

Moreover, an effective comparative result between DDPG and TD3 are included in the study. The comparison reflets that TD3 ensures a stable and rapidly converging training system, which is resource efficient. This comparison validates the selection of algorithm for policy free, model free, differential drive system with continuous action space.

Future of this work can be directed to the designing and training more primitive policies and combine them is an effective manner to construct a compound policy, performing in a complex task environment. Furthermore, this policy can be implemented on real robot to be tested in the real world.

REFERENCES

[1] A. Ibrahim, R. R. Alexander, M. Shahid, U. Sanghar, R. Donate, and D. " Souza, “Control Systems in Robotics: A Review,” *Int. J. Eng. Invent.*, vol. 5, no. 5, pp. 29–38, 2016, [Online]. Available: www.ijeijournal.com

[2] D. N. J. Peters, “Model learning for robot control : a survey,” pp. 319–340, 2011, doi: 10.1007/s10339-011-0404-1.

[3] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, vol. 1999, no. December. 2006.

[4] P. Kormushev, S. Calinon, and D. G. Caldwell, “Reinforcement learning in robotics: Applications and real-world challenges,” *Robotics*, vol. 2, no. 3, pp. 122–148, 2013, doi: 10.3390/robotics2030122.

[5] G. Bledt, M. J. Powell, B. Katz, J. Di Carlo, P. M. Wensing, and S. Kim, “MIT Cheetah 3: Design and Control of a Robust, Dynamic Quadruped Robot,” *IEEE Int. Conf. Intell. Robot. Syst.*, pp. 2245–2252, 2018, doi: 10.1109/IROS.2018.8593885.

[6] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, “Learning to Walk via Deep Reinforcement Learning,” *Robot. Sci. Syst.*, 2019, doi: 10.15607/RSS.2019.XV.011.

[7] S. Fujimoto, H. Van Hoof, and D. Meger, “Addressing Function Approximation Error in Actor-Critic Methods,” *35th Int. Conf. Mach. Learn. ICML 2018*, vol. 4, pp. 2587–2601, 2018.

[8] T. P. Lillicrap *et al.*, “Continuous control with deep reinforcement learning,” *4th Int. Conf. Learn. Represent. ICLR 2016 - Conf. Track Proc.*, 2016.

[9] P. Dayan, “Q-Learning,” vol. 292, pp. 279–292, 1992.

[10] Ding, Zihan, and Hao Dong. "Challenges of reinforcement learning." *Deep Reinforcement Learning*. Springer, Singapore, 2020. 249-272.