Reinforcement Learning based Autonomous Vehicle for Exploration and Exploitation of Undiscovered Track

Razin Bin Issa
Dept. of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
raz.issa00@gmail.com

Md. Saferi Rahman
Dept. of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
saferi2310@gmail.com

Modhumonty Das
Dept. of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
modhumanty.sreya@gmail.com

Monika Barua
Dept. of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
monikabdctg@gmail.com

Md. Golam Rabiul Alam
Dept. of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
rabiul.alam@bracu.ac.bd

***Abstract*— This research focuses on autonomous traversal of land vehicles through exploring undiscovered tracks and overcoming environmental barriers. Most of the existing systems can only operate and traverse in a distinctive mapped model especially in a known area. However, the proposed system which is trained by Deep Reinforcement learning can learn by itself to operate autonomously in extreme conditions. The dynamic double deep Q-learning (DDQN) model enables the proposed system not be confined to only in known environments. The ambient environmental obstacles are identified through Faster R-CNN for smooth movement of the autonomous vehicle. The exploration and exploitation strategies of DDQN enables the autonomous agent to learn proper decisions for various dynamic environments and tracks. The proposed model is tested in a gaming environment. It shows the overall effectiveness in traversing of autonomous land vehicles. The goal is to integrate Deep Reinforcement learning and Faster R-CNN to make the system effective to traverse through undiscovered paths by detecting obstacles.**

Keywords*— *Reinforcement Learning; Faster R-CNN; Double Deep Q Learning; Markov Decision Process; Autonomous Vehicle; Object Classifier;

I. INTRODUCTION

Reinforcement learning [1] involves taking decision to perform suitable action by maximizing reward in a specific situation. Employed by various software and machines, it finds the best possible behavior or path it should take in a particular situation. A reinforcement learning based agent learns from interacting with the environment. It can decide how to perform a given task on its own from a training dataset. However, it must learn from its experience in the absence of a dataset. In this modern age, autonomous vehicles are considered one of the most important parts of intelligent transportation system. An autonomous vehicle first takes perception from the environment, makes a decision, plans it and then controls the vehicle. Thus, it is one of the most emerging technologies that exists today. The brain of an autonomous vehicle is the decision-making module. Hence, integrating reinforcement learning in an autonomous vehicle will help the vehicle perform in any environment through exploitation and exploration. Thus, the vehicle or autonomous agent will be able to take its own decision to traverse and drive in undiscovered tracks. The conventional ways of autonomous vehicles are limited within certain maps. However, integrating autonomous vehicle and reinforcement learning will make an agent take decisions more accurately and control it in that manner. The proposed research suggests, traversing of an autonomous vehicle which tries to find its own path by detecting and identifying obstacles along the way. It takes data and information of rough and rocky surface and obstacles through sensor data. The agent feeds the data in our proposed algorithm for taking decisions in future based on the fed conditions. The sensor that is mainly used to implement the mentioned proposal is Camera. The agent explores the environment using Double Deep Q-Learning. It estimates more accurate values [2]. The process of the agent always picking the highest q-value for exploration is called the epsilon greedy strategy. The three terms that are used in this strategy are state, reward and action. In order to perform the mentioned method, the algorithm uses Bellman Equation. Furthermore, Faster R-CNN [3] will be applied at first for the prototype as the agent and vehicle tries to track and detect objects while traversing. Faster R-CNN, at this time, is one of the most prominent algorithms for object detection. In Faster-RCNN, region proposal and object detections are done using convolutional network which make this algorithm faster in terms of objection detection. By merging the mentioned processes of path finding and object tracking, we are aiming to develop an agent so that it can find its own path by avoiding obstacles on its way. We test our algorithm on a gaming environment. Additionally, to test our algorithm, we plan to use BRACU Mongol-Tori and Autonomous Car as our prototyping agents.

This research contributes to the advancement of technology and uncovers areas which have not been explored and considered before. The contributions of this study are as follows:

- Implementing Double Deep Q-Learning (DDQN) for autonomous traversal
- Integrating Faster R-CNN for image classification

The rest of the paper is organized as follows. Section II describes the related works that have already been recognized. In Section III, the procedure of making the object classifier through Faster R-CNN has been discussed. Section IV describes the application of reinforcement learning in the autonomous vehicle. Result and simulation have been discussed in Section V. Finally, section VI concludes the paper.

II. Related Works

In recent years, neural networks have turned into the main technique for reliable object identification. In [4] most neural networks images are classified, clustered by their similarity using R-CNN, Fast R-CNN, Faster –RCNN algorithm. Thus, in object detection these algorithms are explicitly used.

CNN [5] is the abbreviation for Convolutional Neural Network. It consists of convolution, RELU, Pooling and Fully connected layer. Initial approach towards object detection is classifying some interested regions and use CNN to it. The CNN [6] in RCNN focuses on a single region at a time to minimize the interface. Here, regions are specified. So, it is also called region proposal.

However, to make object detection more effective and fast Faster R-CNN [5] is used. Other algorithms use selective search for regions but a separate network is used in Faster R-CNN to predict region proposals. By reshaping the proposed regions, the value of bounding boxes is predicted.

A work uses [7] region-based convolutional neural network for the detection of road obstacles using deep learning system. However, our proposal does not only detect obstacles, it takes decisions upon them using double deep q-learning.

Additionally, psychology includes study of learning and reinforcement which has had a well-built impact on Artificial Intelligence related work. In fact, all algorithms of reinforcement learning can be considered as reverse engineering of some psychological learning processes [8].

Although, work related to reinforcement learning has been done in the field of web-spidering for optimal sequential decision making [9], our proposal uses reinforcement learning for efficient path finding.

Another paper [10] suggests eight extensions of reinforcement learning which includes adaptive heuristic critic (AHC) learning, Q-learning, and three further extensions to both basic methods to speed up learning. This proposal mainly focuses on DDQN based learning to train the agent.

One related work [11] determines driving policy on highways using reinforcement learning. A different driving simulator from the mentioned work has been used in our implementation which is discussed later in this paper.

Longitudinal control of autonomous land vehicles has been proposed [12] by using parameterized reinforcement learning in another related work. It mainly uses PBAC algorithm which differs from the DDQN algorithm that we have used.

Thus, Reinforcement learning is said to be an interesting learning technique which only requires a performance feedback from the environment. Till date, it has been used to solve simple learning problems. Our proposal investigates reinforcement learning to be used as a decision maker for path finding.

Hence, we propose to implement a reinforcement learning based autonomous vehicle which traverse through exploration and exploitation of environments. It finds its own path by detecting and identifying objects and obstacles along the way using Faster RCNN.

III. Object Classification through Faster R-CNN

A. Faster R-CNN

The R-CNN technique adopts the clear strategy of trimming remotely computed box proposition out of an input image and applying neural system classifier on it. Nonetheless, this methodology can be costly because many crops are required, which leads to large overlap calculation from overlapping crops. Fast R-CNN moderated this issue by driving the entire picture through feature extractor. Crop from a middle layer allow crops to share the load of highlight extraction [17]. Although R-CNN and Fast R-CNN depends on an external proposal generator, lately it has been proven that generating box proposals using neural nets is possible. Here, it is normal that there can be few boxes on beat of each other on the picture at distinctive outline, scales and perspective proportions, which is called "anchors". Now, a model is prepared to anticipate for each anchor: (a) a discrete class expectation for each anchor, and (b) a cumulative forecast of the counterbalance, according to which the anchor has to be relocated to fit in the ground truth bounding box. In the accompanying para, minimization of a combined classification and regression loss is discussed. [15, 16, 18]

The best matching ground-truth box b is first to be found for each anchor a. If we can identify a match, we consider it as a "positive anchor", and assign it (a) a class label $y_a \in \{1 \ldots k\}$ and (b) a vector encoding of b with respect to anchor a (called the box encoding $\phi(b_a; a)$). On the off chance that no such match could be found, we consider it as a "negative anchor", and set the class name to be $y_a = 0$. Considering the anchor as a, if we predict box encoding $f_{loc}(I;\, a;\, \theta)$ and corresponding class $f_{cls}(I;\, a;\, \theta)$, where I is the image and θ the model parameters, then the loss for I is measured as a weighted sum of a location-based loss and a classification loss:

$$\mathcal{L}(a;\, I;\, \theta) = \alpha \cdot 1[a \text{ is positive}] \cdot \ell_{loc}\big(\phi(b_a; a) - f_{loc}(I;\, a;\, \theta)\big) + \beta \cdot \ell_{cls}\big(y_a, f_{cls}(I;\, a;\, \theta)\big) \quad (1)$$

Here α, β are weights balancing localization and classification losses. Equation 1 is averaged over anchors, in order to train the model and reduced with respect to parameters θ. [15]

The determination of anchors has noteworthy outcomes for exactness and calculation as well. Previously these anchors were computed from clustering ground-truth boxes within the dataset. Nowadays, the process is handled by tiling a collection of boxes at various scales and aspect ratios, routinely over the image. The good side of a customary network of stays is that the forecasts can be composed as tiled indicators on the picture with shared parameters. This process resembles traditional sliding window method [16, 18].

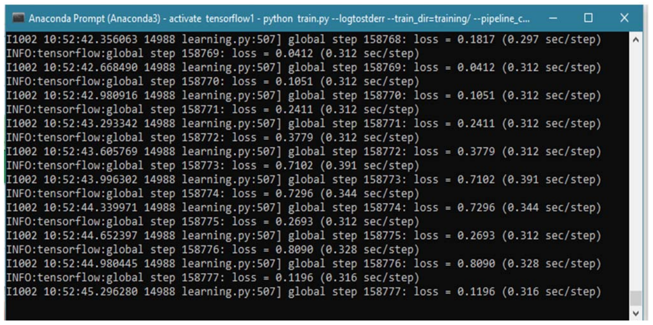
B. Acquiring Dataset

Experiments were conducted on the huge dataset named Open Image V5. It's an open-source database made by Google. It consists of annotated images of 600 boxable object classes. The total number of training images in this database are 1,743,042. These images include annotated bounding boxes, object segmentations, and visual relationships, as well as the full validation (41,620 images) and test (125,436 images) sets. But for our Object Classifier we don't need all those 600 classes.

Using 'OIDv4_ToolKit' 7 classes from 600 were separated. Those are of Bicycle, Bus, Person, Motorcycle, Truck, Van, Car. For training criteria, 8,355 images were separated from the whole dataset, where minimum of 1000 images were of each class. And 2,360 of them were taken for testing, having a minimum of 300 images of each class. The labels of those images were then combined together in .xml format.

C. Training Object Classifier

For training our dataset and to prepare our object classifier we chose Faster R-CNN Inception V2 feature extractor on TensorFlow framework. For training and testing, a computer having NVIDIA GTX 1050 GPU, with CUDA core support has been used. The training process was consistently run for 13hours, until the total loss became persistent under 0.8 for a long time. A total of 1,58,777 steps were conducted to prepare our desired image classifier.



```
Anaconda Prompt (Anaconda3) - activate  tensorflow1 - python  train.py --logtostderr --train_dir=training/ --pipeline_c...
I1002 10:52:42.356063 14988 learning.py:507] global step 158768: loss = 0.1817 (0.297 sec/step)
INFO:tensorflow:global step 158769: loss = 0.0412 (0.312 sec/step)
I1002 10:52:42.668490 14988 learning.py:507] global step 158769: loss = 0.0412 (0.312 sec/step)
INFO:tensorflow:global step 158770: loss = 0.1051 (0.312 sec/step)
I1002 10:52:42.980916 14988 learning.py:507] global step 158770: loss = 0.1051 (0.312 sec/step)
INFO:tensorflow:global step 158771: loss = 0.2411 (0.312 sec/step)
I1002 10:52:43.293342 14988 learning.py:507] global step 158771: loss = 0.2411 (0.312 sec/step)
INFO:tensorflow:global step 158772: loss = 0.3779 (0.312 sec/step)
I1002 10:52:43.605769 14988 learning.py:507] global step 158772: loss = 0.3779 (0.312 sec/step)
INFO:tensorflow:global step 158773: loss = 0.7102 (0.391 sec/step)
I1002 10:52:43.996302 14988 learning.py:507] global step 158773: loss = 0.7102 (0.391 sec/step)
INFO:tensorflow:global step 158774: loss = 0.7296 (0.344 sec/step)
I1002 10:52:44.339971 14988 learning.py:507] global step 158774: loss = 0.7296 (0.344 sec/step)
INFO:tensorflow:global step 158775: loss = 0.2693 (0.312 sec/step)
I1002 10:52:44.652397 14988 learning.py:507] global step 158775: loss = 0.2693 (0.312 sec/step)
INFO:tensorflow:global step 158776: loss = 0.8090 (0.328 sec/step)
I1002 10:52:44.980445 14988 learning.py:507] global step 158776: loss = 0.8090 (0.328 sec/step)
INFO:tensorflow:global step 158777: loss = 0.1196 (0.316 sec/step)
I1002 10:52:45.296280 14988 learning.py:507] global step 158777: loss = 0.1196 (0.316 sec/step)
```

Fig. 1. Training data model object classification

There are two stages is the Faster R-CNN detection process. The primary stage is called region proposal network (RPN). Here images are processed by a feature extractor, for which we have used Faster R-CNN Inception V2 feature extractor, and features are used to anticipate class diagnostic box proposals, at some selected intermediate level. The loss function for this first stage appears as Equation of Loss using a grid of anchors tiled in space, scale and aspect ratio.

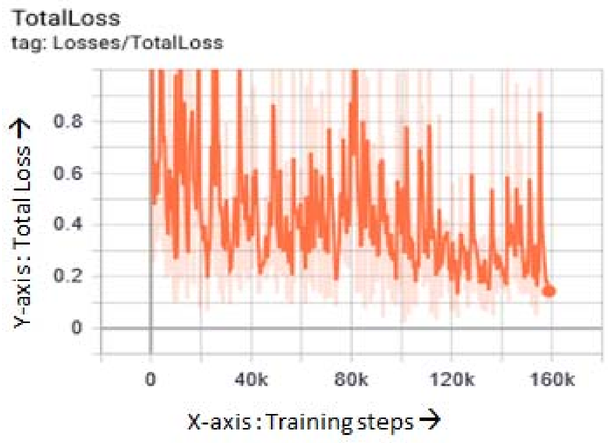


Fig. 2. Total loss graph of the trained data model

In the subsequent stage, the box proposals which were predicted in the first stage, are utilized at cropping features from the same common feature map which are subsequently used to the rest of the feature extractor so as to predict a class and class-specific box refinement for each proposal. The loss function for this second stage box classifier likewise appears as the Equation of Loss using the proposals generated from the RPN as anchors. Here, Fig. 2 shows the normalized form of total loss graph.

IV. Distributional Agent for Autonomous Driving

A. Double Deep Q Network (DDQN)

Double Deep Q Network (DDQN) algorithm was proposed by H. V. Hasselt [13]. This calculation utilizes the concept of Double Q-learning and is an extension of H. V. Hasselt's previous proposal [14] and it is applied to DQN. DQN is one of the few Q-learning based algorithms. These Q-learning based algorithms have overestimation problems which are caused by estimation errors. Overoptimistic fee estimation and performance dilapidation happen as a result of overestimation. However, the strategy for DDQN does not just diminish the overoptimistic esteem estimation, yet additionally gives better execution than DQN on a few game conditions. Selection and evaluation process are isolated by DDQN while it gets to target an incentive with two Q-functions. The target esteem conditions of DQN and DDQN are appeared underneath:

$$y^{DQN} = R_t + \gamma \max_{A_{t+1}} Q\ (S_{t+1}, A_{t+1}; \theta^-) \tag{2}$$

$$y^{DDQN} = R_t + \gamma Q(S_{t+1}, \operatorname*{argmax}_{A_{t+1}} Q\ (S_{t+1}, A_{t+1}; \theta);\ \theta^-) \tag{3}$$

B. Markov Decision Process for Path Distribution

Markov Decision Process articulates the acquiring path direction for autonomous driving in this research. The agent decides his own action in every step and immediately a reward is received for that action. Markov Decision Process is narrated by the tuple {S, P, A, R, γ} which has already been stated before. For our problem, a brief summary of MDP is stated below:

- s ε S is the finite state space which contains a gray scale image from camera of the agent.
- P is the transition function where P(s'|s a): S×A×S → [0,1]
- a ε A is finite action space which works for an agent.
- R(s,a): S×A → R where R is reward function
- γ defines the discount factor where γ → [0,1] for delayed reward

MDP states' s ε S can be used for the high dimensional observations by using deep neural networks. Fig. 3 represents the perception of the surrounding coverage by using 3 cameras mounted at the front.

The autonomous driving agent has 5 distinct actions. The finite action space A consists of forward, left, right, stop and deceleration. 5 kph is added or subtracted from the current agent speed for forward and deceleration. The agent speed is confined in the range of 30 kph to 80 kph. The agent automatically adjusts the speed for vehicles in a certain distance so that it maintains a safe distance from the front vehicle. The 'stop' activity happens instantly when the vehicle in front all of a sudden brakes or any other vehicle cuts in abruptly before our agent vehicle.

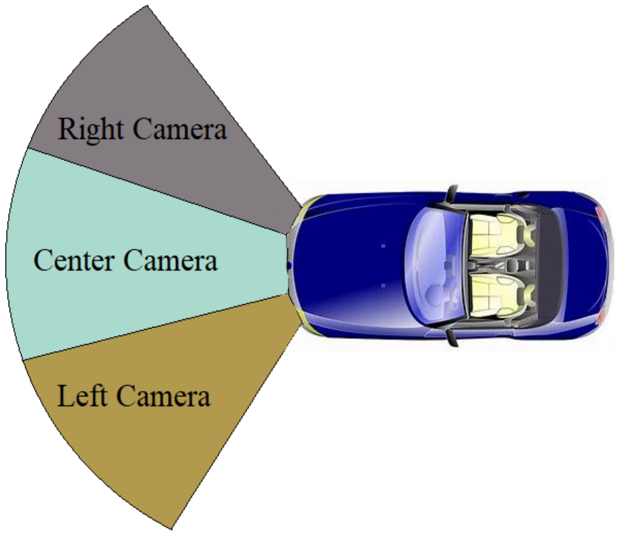


Fig. 3. Camera Coverage of perception.

C. *Data Preprocessing*

The images are cropped so that the model will not be trained with the sky and the car's front parts. Those are resized to 160x320 (3 YUV channels) as per NVIDIA model. Those images are normalized (image data divided by 127.5 and subtracted 1.0). As stated in the Model Architecture section, this is to avoid saturation and make gradients work better.

D. *Model Architecture Design*

The main target of the agent π(a|s) is mapping the perception state, S and making the following move on action space, A. The entirety of the activity will be led in a stochastic driving condition. However, to accomplish this mapping the model needs to satisfy to distinct conditions: (1) extract and capture significant highlights from 3 camera images,(2) it should take account of the inherent randomness of the environment for choosing any particular action.

Here, to satisfy the first condition; spatio-rational information retrieving from camera sensor need to be sensed by the network. This process is conducted using Convolutional Neural Network (CNN). It is popular for extracting spatial features from images. Moreover, high dimensional camera images are refined into visual feature vector using three two-dimensional convolutional layers.

Furthermore, the second condition can be fulfilled by using DQN framework. Driving environments that are stochastic use this framework. For each action there is a return distribution which is created by the completely associated layer with the help of θ. Here, the $Q(s,a)$ result can be estimated as the desire of quantiles, $\sum_i q_i\theta_i\,(s,a)$. Additionally, the maximum Q value can be retrieved from the best action, a^* which also can be picked from accessible limited Q values of action space, A.

$$a^* = argmax_a Q(s,a)$$

Proposed DDQN architecture for Reinforcement Learning based Autonomous vehicle is shown in Fig. 4. Keras has been used to train this network.

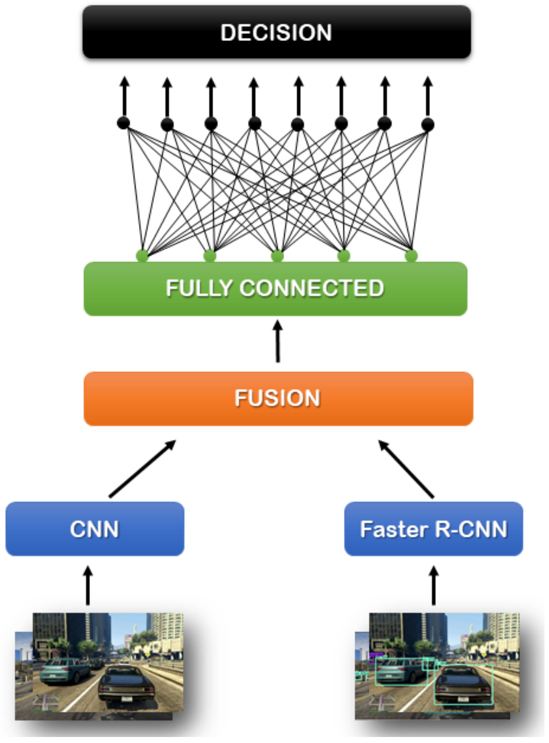


Fig. 4. Proposed DDQN architecture for Reinforcement Learning based Autonomous vehicle

E. *Hyperparameters*

The network is designed following NVIDIA model. To perform end-to-end self-driving test by NVIDIA, it has been used. Supervised image classification or regression problems can be solved in seamless procedure using deep convolutional network. NVIDIA model itself is well documented. Thus, our main focus lies on adjusting the training images for delivering the best result. However, to acquire the best result we need to make necessary adjustments for avoiding over-fitting nature and adding non linearity to make the prediction accurate. Additionally, we have added the following adjustment to the model.

- Lambda layer is introduced to normalize input images to make gradients work more smoothly and to avoid saturation.
- Additional dropout layer is added after the convolution layers to avoid the over-fitting nature.
- Then we have implemented ReLU for activation function to ensure linearity.

Adam Optimizer with epsilon 0.0001 at a learning rate 1e^{-5} and mini-batches of size 32 is used to train the network for optimum accuracy. Here, Xavier initializer has been used to initialize network weights and all inputs are normalized into [-1,1]. For achieving the accuracy of the prediction of steering angle for each image we have used mean squared errors for estimating the loss function. The table shows Hyperparameters of this driving policy network. We have set the value of support Q as 200. The size of the replay memory is 5000000 and γ which is the discount factor has been fixed to 0.99. ε-greedy policy has been used where ε was gradually diminished from 1.0 to 0.1 in each step and then fixed to 0.1. All of these policies have been implemented during 3000000 steps trainings.

TABLE I. AUTONOMOUS DRIVING POLICY NETWORK: HYPERPARAMETERS

Data	Layer Type	Actuation	Hyperparameters of Policy Network
Camera Data	Convolution 2D	ReLU	Patch size = (5×5) Stride = 4 No. of filters = 24
	Convolution 2D	ReLU	Patch size = (5×5) Stride = 4 No. of filters = 36
	Convolution 2D	ReLU	Patch size = (5×5) Stride = 4 No. of filters = 48
	Convolution 2D	ReLU	Patch size = (3×3) Stride = 1 No. of filters = 64
	Convolution 2D	ReLU	Patch size = (3×3) Stride = 1 No. of filters = 64
Concatenated Data	Fully Connected Layer	ReLU	No. of Units = 512

F. *Model Training*

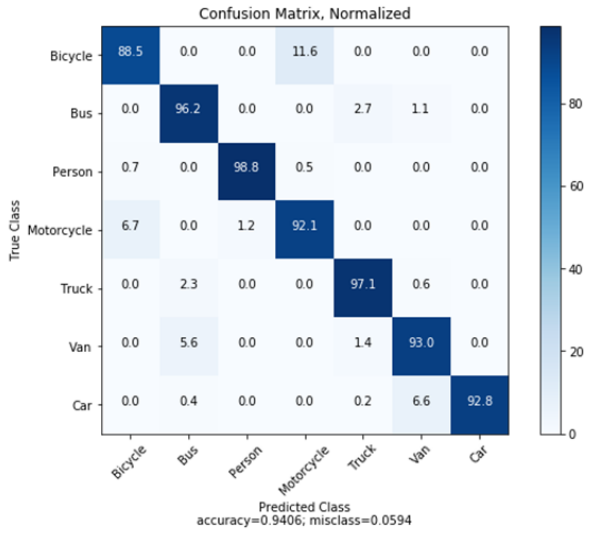
For training, we used the following augmentation technique along with Python generator to generate an unlimited number of images:

- Randomly choose right, left or center images.
- For left image, steering angle is adjusted by +0.2
- For right image, steering angle is adjusted by -0.2
- Randomly flip image left/right
- Randomly translate image horizontally with steering angle adjustment (0.002 per pixel shift)
- Randomly translate image vertically
- Randomly added shadows
- Randomly altering image brightness (lighter or darker)

Using the left/right images is useful to train the recovery driving scenario. The horizontal translation is useful for difficult curve handling.

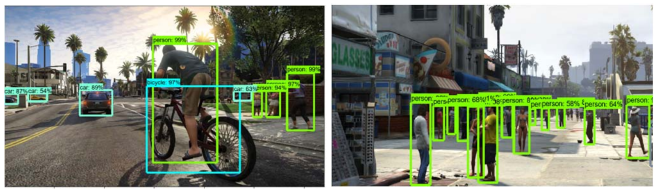
V. SIMULATION AND RESULT

In the trained image classifier, the images are passed which are retrieved from the camera of the autonomous vehicle. Then those images are classified and they show accuracy on identifying objects. To understand the accuracy in terms of False positive and True negative for each class, we have constructed a confusion matrix, which is shown in Fig. 5. Moreover, to test the accuracy level of this system and to evaluate its performance the whole procedure is implemented on the game environment. The result that is received from this test is quite optimum and have high accuracy level. Fig. 6 shows the accuracy level of the classifier.



True Class	Bicycle	Bus	Person	Motorcycle	Truck	Van	Car
Bicycle	88.5	0.0	0.0	11.6	0.0	0.0	0.0
Bus	0.0	96.2	0.0	0.0	2.7	1.1	0.0
Person	0.7	0.0	98.8	0.5	0.0	0.0	0.0
Motorcycle	6.7	0.0	1.2	92.1	0.0	0.0	0.0
Truck	0.0	2.3	0.0	0.0	97.1	0.6	0.0
Van	0.0	5.6	0.0	0.0	1.4	93.0	0.0
Car	0.0	0.4	0.0	0.0	0.2	6.6	92.8

Fig. 5. Confusion matrix of the image classification model.



(a) (b)

Fig. 6. Image classification through R-CNN (a) Object detection, and (b) Pedestrian detection.

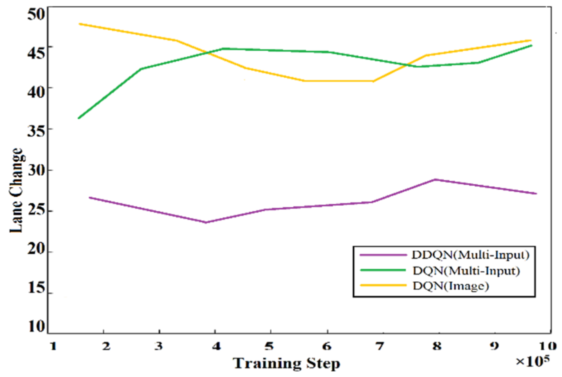
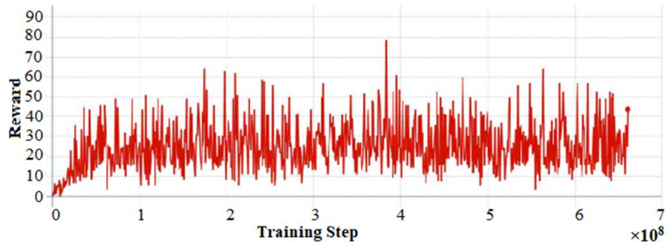


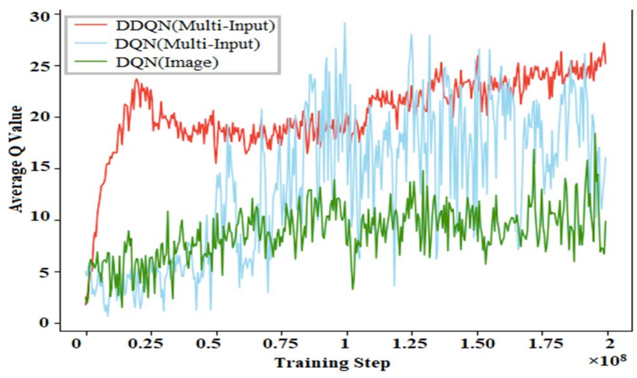
Fig. 7. Lane Changing comparison of different algorithms of Reinforcement Learning.

The graph above justifies the algorithm we used, which is tested in GTA V game environment.. The agent has been tested in every possible environment which is similar to real life scenario (e.g. day, night, rainy, foggy, crowded etc.). Figure 7 represents the lane changing comparison of different algorithms of Reinforcement Learning.

Finally, the graphs in Fig. 8 are generated from the output section of the algorithm that we have implemented for the uninterrupted traversing of our agent. This traversing algorithm involves reward method which is considered as an inevitable part of deep Q learning to make this decision making process more accurate. Moreover, we are delivering the reward to our agent based on the q values of every raining data. Here, Fig 8(a) shows the average reward of the training dataset. On the other hand, Fig 8(b) defines the average Q values.



(a)



(b)

Fig. 8. Characteristics of DDQN hyper-parameters (a) Average Reward (b) Average Q Value.

VI. Conclusion

In this research, DDQN (Double Deep Q Learning) algorithm has been implemented for autonomous exploration of the vehicle. This algorithm is one of the foremost productive distributional reinforcement learning which gives better efficiency than any other non-distributional algorithm. Further, multi-input architecture has been integrated for better autonomous exploration. DDQN is executed through reward system policy which helps the vehicle to take decision in any environment. It also enables the vehicle to take rational action according to the perceived decision. Alongside, Faster R-CNN plays an integral in our system to make decisions from any unexpected or uneven situations. Acquiring values from Faster R-CNN; reward function can be manipulated which accelerates the efficiency in decision making process and smoothen autonomous exploration of vehicle. This system is not confined to any particular environment like other existing autonomous vehicle systems which are implemented for only mapped model that limits their exploration range and accuracy. Though our work shows ultimate accuracy, our future plan is to use more adaptive algorithm which can be more appropriate for optimizing autonomous traversing.

References

[1] Greenwald, A., Hall, K., & Serrano, R. (2003, August). Correlated Q-learning. In ICML (Vol. 3, pp. 242-249).

[2] Van Hasselt, H., Guez, A., & Silver, D. (2016). . In AAAI Conference on Artificial Intelligence. Retrieved from https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12389/11847

[3] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov. Scalable object detection using deep neural networks.InCVPR, 2014.

[4] M. Roh and J. Lee, "Refining faster-RCNN for accurate object detection," 2017 Fifteenth IAPR International Conference on Machine Vision Applications (MVA), Nagoya, 2017, pp. 514-517.

[5] Takayuki Ujiie, Masayuki Hiromoto, Takashi Sato. "Approximated Prediction Strategy for Reducing Power Consumption of Convolutional Neural Network Processor", 2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2016

[6] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov. Scalable object detection using deep neural networks.InCVPR, 2014.

[7] G. Prabhakar, B. Kailath, S. Natarajan and R. Kumar, "Obstacle detection and classification using deep learning for tracking in high-speed autonomous driving," 2017 IEEE Region 10 Symposium (TENSYMP), Cochin, 2017, pp. 1-6.

[8] Sutton R.S. (1992) Introduction: The Challenge of Reinforcement Learning. In: Sutton R.S. (eds) Reinforcement Learning. The Springer International Series in Engineering and Computer Science (Knowledge Representation, Learning and Expert Systems), vol 173. Springer, Boston, MA

[9] Rennie, J., & McCallum, A. (1999, June). Using reinforcement learning to spider the web efficiently. In ICML (Vol. 99, pp. 335-343)

[10] Lin LJ. (1992) Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching. In: Sutton R.S. (eds) Reinforcement Learning. The Springer International Series in Engineering and Computer Science (Knowledge Representation, Learning and Expert Systems), vol 173. Springer, Boston, MA

[11] K. Min, H. Kim and K. Huh, "Deep Distributional Reinforcement Learning Based High-Level Driving Policy Determination," in IEEE Transactions on Intelligent Vehicles, vol. 4, no. 3, pp. 416-424, Sept. 2019.

[12] Z. Huang, X. Xu, H. He, J. Tan and Z. Sun, "Parameterized Batch Reinforcement Learning for Longitudinal Control of Autonomous Land Vehicles," in IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 49, no. 4, pp. 730-741, April 2019.

[13] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning.” in AAAI, vol. 2. Phoenix, AZ, 2016, p. 5.

[14] H. V. Hasselt, “Double q-learning,” in Advances in Neural Information Processing Systems, 2010, pp. 2613–2621.

[15] Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Murphy, K.: Speed/accuracy trade-offs for modern convolutional object detectors. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 7310–7311 (2017)

[16] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems, pages 91–99, 2015.

[17] R. Girshick. Fast r-cnn. In Proceedings of the IEEE International Conference on Computer Vision, pages 1440–1448, 2015.

[18] C. Szegedy, S. Reed, D. Erhan, and D. Anguelov. Scalable, high-quality object detection. arXiv preprint arXiv:1412.1441, 2014.