iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 8 **Issue:** IV **Month of publication:** April 2020

DOI: http://doi.org/10.22214/ijraset.2020.4082

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 8 Issue IV Apr 2020- Available at www.ijraset.com

Self-Driving Car Simulation using Genetic Algorithm

Asst Prof. Teena Varma[1], Varun Singh[2], Rohan Talele[3], Abhishek Solanki[4], Rajan Singh[5]
[1, 2, 3, 4, 5]*Department of Computer Engineering, Xavier Institute of Engineering, Mumbai University, Mumbai, India*

Abstract: ***For the past decade, there has been a surge of interest in self-driving vehicles. This is because of leaps in the field of deep learning especially Artificial Neural Networks which are trained to perform tasks that regularly require human intervention. Genetic algorithms (GA) apply models to identify patterns and features in the environment making them desirous in cases of uncertainty. In this project we have trained an evolving Neural network by using Genetic Algorithms in Unity Game Engine using the ML-agents provided by them. By repeated simulation of the agent in the environment, using this we have created an agent which learns unique features from the environment based on its input and generates a forecast permitting the vehicle to drive without requiring external commands from the player.***

Keywords: Artificial Intelligence (AI), Genetic Algorithm (GA), deep learning, Artificial Neural Network (ANN), Unity Game Engine, ML-agents.

I. INTRODUCTION

As of late, self-sufficient driving calculations utilizing minimal effort vehicle-mounted cameras have pulled in expanding research attempts from both, the scholarly community and industry. Different degrees of robotization have been characterized in independent driving. There's no robotization in level 0. A human driver controls the vehicle. Level 1 and 2 are propelled driver help frameworks where a human driver despite everything controls the framework however a couple of highlights like brake, dependability control, and so forth are mechanized. Level 3 vehicles are self-governing, in any case, a human driver is as yet expected to screen and mediate at whatever point fundamental. Level 4 vehicles are completely self-governing, yet the mechanization is restricted to the operational plan space of the vehicle for example it doesn't cover each driving situation. Level 5 vehicles are relied upon to be completely self-governing and their presentation ought to be equal to that of a human driver.

We are extremely distant from accomplishing level 5 self-governing vehicles sooner rather than later. Notwithstanding, level - 3/4 self-governing vehicles are possibly turning into a reality sooner rather than later. Essential explanations behind extreme specialized accomplishments right now fields are specialized achievements and amazing inquire about being done in the field of PC vision what's more, AI and furthermore the minimal effort vehicle-mounted cameras which can either autonomously give noteworthy data or supplement different sensors. Numerous vision-based drivers help highlights have been generally bolstered in the cutting-edge vehicles. A portion of these highlights incorporate person on foot/bike location, crash shirking by assessing the front vehicle separation, path take-off cautioning, and so forth. Notwithstanding, right now, target independent controlling, which is a generally unexplored errand in the field of PC vision and AI.

Our project aims to create a simulation environment for the agent being used/which will be used for the autonomous vehicle. Before deployment of the agent in real vehicles in real roads, it must have sufficient training with different scenarios. Deploying the trained agent without simulating it could turn out to be disastrous and catastrophic for the user as well as the ones in proximity. This can be achieved with a Neural network without requiring it to evolve and thereby eliminating the need of genetic algorithm in this project, however it has been shown consistently in tests such as the Pole Balancing Test has shown that networks with genetic algorithms such as Neuro Evolution of Augmenting Topologies (NEAT) [1] have performed consistently better than networks without evolution in terms of time to reach the same level of performance.

While we are still quite a way away from complete autonomous vehicles, the required testing and simulation of the Neural Network used in these vehicles would require a simulation environment. This is where our project of creating a simulation environment inside of Unity comes to aid in training and testing of the AI being created. Before deployment of the agent in the real world we need it to perform satisfactorily in terms of the requirements placed on it. In this case we have trained our agent by repeatedly running the training environment inside of Unity after which the trained model can be extracted and then applied to any other model of autonomous vehicles with similar results.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 8 Issue IV Apr 2020- Available at www.ijraset.com

II. RELATED WORK

The DAVE framework was made by DARPA [2] what's more, utilized pictures from two cameras just as left also, right controlling orders to prepare a model to drive. It shows that the system of start to finish learning can be applied to self-sufficient driving. This implies the halfway highlights for example, the stop signs and path markings don't have to be explained or marked for the framework to learn. DAVE is an early task in the field of independent driving. With regards to current innovation, a colossal bit depended on remote information trade because the vehicle couldn't convey the PCs and force hotspots for the framework, which contrasts the lighter gear that exists today. The design of this model was a CNN made up of completely associated layers that originated from organizes recently utilized in object acknowledgment.

The ALVINN framework [3] is a 3-layer back-proliferation organize worked by a gathering at CMU to finish the assignment of path following. It prepares on pictures from a camera and a separation measure from a laser go discoverer to yield the bearing the vehicle should move. ALVINN's model uses a single shrouded layer back-proliferation organize.

We observed an examination by NVIDIA [4]. The

framework utilizes a start to finish approach where the information is first gathered in various distinctive ecological conditions. The information is then enlarged to make the framework powerful to driving askew and to various potential conditions. The subsequent stage is preparing the arrange on this information. The system engineering is an aggregate of 9 layers beginning with convolutional layers what's more, trailed by completely associated layers. These are the various arrangements that we examined and wanted to highlight.

III. SIMULATION ENVIRONMENT

For our simulation environment we have gone with the process of creating a simulation environment in Unity instead of using a proper simulator such as Udacity. For our simulation environment we chose that of a Forest environment with trees and rocks.

Fig. 1 Our Forest Environment

This forest environment was created using the Unity Engine. The engine provides us with tools for creation of the environments, generates lighting, particle effects and most important of all, it provides us the with the ML-agents package.

The Unity Machine Learning Agents Toolkit (ML-Agents) [5] is an open-source Unity plugin that enables games and simulations to serve as environments for training intelligent agents. Agents can be trained using reinforcement learning, imitation learning, neuro evolution, or other machine learning methods through a simple-to-use Python API. It also provides implementations (based on TensorFlow) of state-of-the-art algorithms to enable game developers and hobbyists to easily train intelligent agents for 2D, 3D and VR/AR games. These trained agents can be used for multiple purposes, including controlling NPC behavior (in a variety of settings such as multi-agent and adversarial), automated testing of game builds and evaluating different game design decisions pre-release. The ML-Agents Toolkit is mutually beneficial for both game developers and AI

researchers as it provides a central platform where advances in AI can be evaluated on Unity’s rich environments and then made accessible to the wider research and game developer communities.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 8 Issue IV Apr 2020- Available at www.ijraset.com

Fig. 2 Closeup shot of one area

For our project we have extensively used the ML-agents package. We have imported the Academy class script from MLagents which has been used to modify the behaviour parameters and tweak the step size of our population. Along with this the Generics class script provides us with in built methods to create our populations, set the generation and for configuring the necessary configuration files for the working of the layers.

A path has been created in our environment along with a border boundary along the path. This boundary is used to determine whether our agent in the environment has crashed or not. The boundary has collision attached to it and our cars have Ray casts on them to calculate the distances from them to their nearest objects. There are in total 5 ray casts are only present in the front of the car. One of the limitations of our environment is we haven't provided an option for the agent to slow down, brake or go in reverse direction. However, even with this limitation we can simulate some reasonably positive results.

IV. GENETIC MODEL

In an ordinary Neural network, we aren't given the ability to change the structure of our network depending on the complexity of the problem. The network will arbitrarily pick the number of hidden layers some of which might not have been required. This increases our networks complexity and in turn could affect our efficiency and performance. However, with Neuro Evolution the network will start off with a small number of hidden layer and if the situation, problem demands some additional computation it will automatically change the number of hidden layers present [1]. This flexibility to evolve the network is what gives genetic algorithms a considerable edge when compared to conventional neural network.

Genetic Algorithms work with the principal of Natural Selection of Darwinism philosophy as is the case in Biological Evolution. According to the principle of Natural Selection only the fittest survive of a particular species to pass on their genes to the next generation. This ensures that only the dominant required genes are passed on to the forward generations and any recessive genes are eradicated. This helps in ensuring the survival of a species in nature. A similar approach is used in Genetic Algorithms which create a population of agents (species) which are made to act and behave in a certain manner which will generate some arbitrary reward for them and the genes contributing towards the maximization of this reward are carried forward in the next generation. This evolution (reproduction) of networks is done till a stage arrives when the agent reaches the maximum fitness that it can achieve and can improve no more, essentially becoming unbeatable in the task it was trained to perform.

In our network we are initially starting with number of hidden layers as 1 and the algorithm will only increase it when it requires it to. If required, we are increasing the number of hidden layers by 2 or decreasing it by 2 as well. The max size on the number of hidden layers imposed is 10. The number of inputs is 5 namely our Ray casts which are present in front our agent. These ray casts vectors continuously compute the distances from the front of our agent to the nearest obstacle, wall in the path and accordingly the agent takes an appropriate decision to make left or right turn. The agent will always be accelerating without slowing down. The output layers consist of 2 outputs which are decisions on taking a left or right turn. The reward given to the agent is of 0.1 for every frame it moves without crashing. A penalty of 0.01 is imposed if the agent crashes and a penalty of 0.001 is imposed to encourage the agent to move forward and not stay idle or slow down. We are encouraging the agent to not brake or slow down and proceed with the current speed.

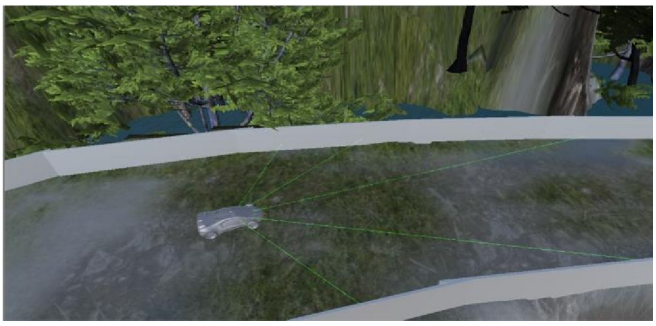


Fig. 3 Ray Casts

We can with a population of any arbitrary size in which each car will be having their own genomes (genes) which are contributing to the neural network in its structuring and decision taking parameters. In each generation the best of all the population species is chosen for mutation i.e. it is taken to be the parent for the next generation. It from this parent that the next generation will derive attributes from. It is necessary to choose a correct parent in the earlier stages for mutation else our model might get trained incorrectly. It is also possible that we may eliminate the correct parent in the beginning mistakenly. NEAT [1] solves this problem by means of [6] explicit fitness sharing where organisms in the same species must share the fitness of their niche, making it difficult for any one species to take over the population. The original finesses are first adjusted by dividing by the number of individuals in the species. Species then grow or shrink depending on whether their average adjusted fitness is above or below the population average:

$$N'_j = \frac{\sum_{i=1}^{N_j} f_{ij}}{\bar{f}}$$

Where N'_j and N_j are the old and the new number of individuals in species j, f_{ij} is the adjusted fitness of individual _
, and f is the mean adjusted fitness in the entire population. The best-performing r% of each species is randomly mated to generate N'_j offspring, replacing the entire population of the species.

NEAT also keeps track of genes through generations by means of Historical Tracking. Essentially it keeps track of all the genes and from which parent they have been derived. If it is observed that a particular attribute of a child is favourable then we are able to know from which gene that attribute was derived. It can help us to find unwanted attributes which have been passed on from certain genes as well.

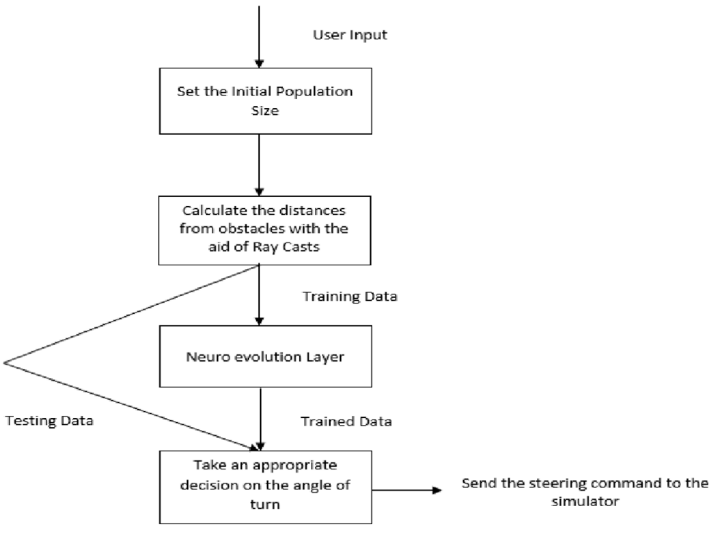


Fig. 4 High Level System Architecture

It has been observed in [7] Reinforcement Learning that training with a small population size can lead to larger training time required, however it also ensures the agents behaviour is optimal. If, however a faster model has to be built then a larger population size is desirable.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 8 Issue IV Apr 2020- Available at www.ijraset.com

V. PERFORMANCE MEASURES

Initially we have started with a population of 50 agents (cars) each of which are contributing to their own genomes which in turn as a whole is contributing towards the evolution of the network. With a population of this size our model quickly reaches the optimal performance of smoothly moving through the environment without any crashes happening. It achieves this in the 3rd generation with a population of 5-7 agents still remaining alive. If the 3rd generation is eliminated and the 4th generation is started then all the agents have achieved complete autonomous capability at least in the simple path that has been created. If the populations size is reduced to 30 then the optimal performance is reached by the 7th generation. If the size is reduced to 1015 then the generations required are 13-15. If the population is kept at 5 then the generations required to optimal performance is around 20th. With each training happening the number of generations can vary a bit but the general result is the same. The optimal performance metric in our case is the ability of the agent to drive completely autonomous.

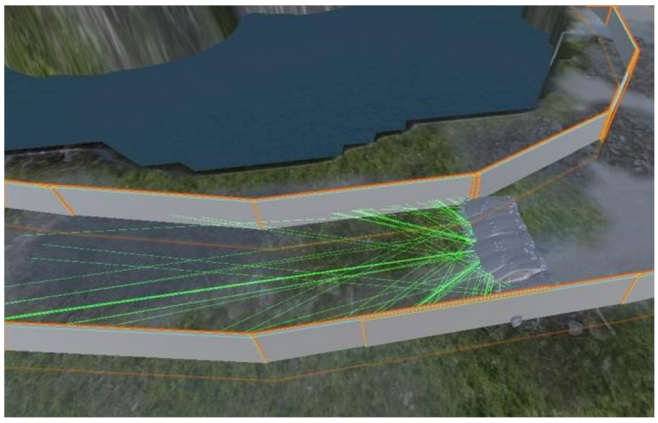


Fig. 5 Population of cars spawning

For performance evaluation during training, mean squared error was used as a loss function to keep a track of the performance of the model.

$$\text{MSE} = (1/n)\sum_{i=1}^{n}(y - y')^2$$

In real life scenarios, while driving on road, the following metric has been proposed in [8]. This metric has been named as autonomy. It is calculated by counting the number of interventions, multiplying by 6 seconds, dividing by the elapsed time of the simulated test, and then subtracting the result from 1.

$$\text{Autonomy} = \frac{1-(number\ of\ interventions)*6\ seconds*100}{elasped\ time\ [seconds]}$$

We ran the above formula on our simulator for 10 seconds with a population of 20 after which the vehicles crashed. By the above formula that would the cars achieved an autonomy of 70%. With a population of more than 40 and more training we were able to get the autonomy to around 90%. However, it was noticed that with a larger population the cars would stick to the right side of the wall for the entire duration of the generation without crashing. This sort of abnormal behaviour is not natural and would likely be removed from more refinement.

VI. FUTURE WORK

We have several ideas to explore and improve upon in the future with a few of these ideas being scrapped due to time constraints in our current project.

Firstly, the implementation of a speed system or a speedometer which would enable the user to change the linear speed of their vehicle. Also, we would like to add a few more tracks and environments with varying difficulties and varieties in the courses along with a proper menu system for the environment so as the user would be able to choose the desired track. On these new tracks there could be a provision for other AI vehicles like traffic to spawn and see how our model then trains and compare it performance measures with our current model.

Secondly, major refinements in the overall presentation, User Interface (UI) and feel of the project could be overhauled to ensure a smoother and much better functionality of the system.

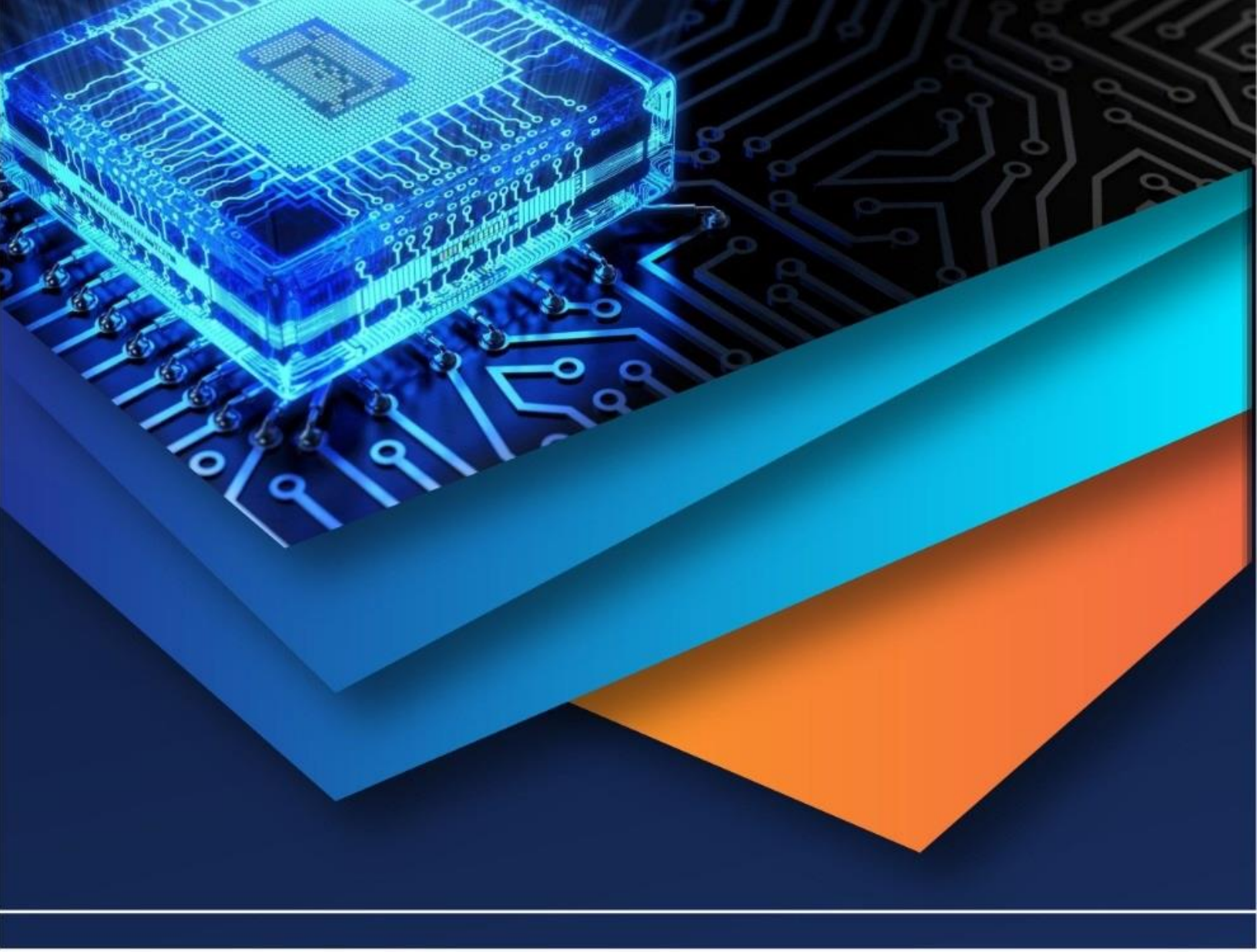
Lastly, we would like to implement the feature of gathering data from the last crash or the most frequent crashes or collisions at a particular location and see that can be incorporated into our model to further increase its efficiency. We would also like to point out the model trained has not been provided as network to a real autonomous vehicle to see out it would fare in the real world. This simulation has only created the network in the simulation environment created in Unity Engine along with all its processing.

VII. CONCLUSION

In this project we were successfully able to implement a simulation of autonomous cars using the tools provided by Unity Engine. We were able to successfully predict the steering angles required by the agents using Neural Networks and evolving them by Neuro evolution of Augmenting Topologies, which also helped us in gaining helpful insight in the ways a neural network works and how it can be tuned. We also demonstrated how the network is able to learn the task of following and completing the track without colliding into obstacles and walls. The training set in case of Genetic Algorithms is generated by the agents themselves acting in the environment. From this only the appropriate and beneficial behaviour generating action in the long run is taken into the training set through which the network is able to evolve. In this project we created our own simulator which could have led to some irregularities and deficiencies in the overall performance such as we weren't able to simulate the model using different traffic lanes, against road signs, on different weather conditions, different locations such as urban, residential, highways. Our system might have lacked in robustness in the above parameters and this could be addressed by improving our own simulator further or by using a simulator such as Udacity. However, the task defined in our paper was successfully accomplished.

REFERENCES

[1] Kenneth O. Stanley, Risto Miikkulainen. Evolving Neural Networks through AugmentingTopologies. *The MIT Press Journals*, Evolutionary Computation 2002 10:2, 99-127
[2] LeCun, Y., et al. DAVE: Autonomous off-road vehicle control using end-to-end learning. Technical Report DARPA-IPTO Final Report, Courant Institute/CBLL, http://www. cs. nyu. edu/yann/research/dave/index. html, 2004.
[3] Pomerleau, Dean A. "Alvinn: An autonomous land vehicle in a neural network." Advances in neural information processing systems. 1989.
[4] Bojarski, Mariusz, et al. "End to end learning for self-driving cars." arXiv preprint arXiv:1604.07316 (2016).
[5] https://github.com/Unity-Technologies/ml-agents
[6] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In J. J. Grefenstette, editor, Proceedings of the Second International Conference on Genetic Algorithms, pages 148–154. San Francisco, CA: Morgan Kaufmann, 1987.
[7] Andrew Senior, Georg Heigold, Marc'Aurelio Ranzato, Ke Yang AN EMPIRICAL STUDY OF LEARNING RATES IN DEEP NEURAL NETWORKS FOR SPEECH RECOGNITION *Google Inc., New York*
[8] Bojarski, Mariusz, et al. "End to end learning for self-driving cars." arXiv preprint arXiv:1604.07316 (2016).



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY