iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 9 **Issue:** V **Month of publication:** May 2021

DOI: https://doi.org/10.22214/ijraset.2021.34296

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 9 Issue V May 2021- Available at www.ijraset.com

Driving Simulation Game based on Reinforcement Learning

Saurabh Pahwa[1], Dhruv Patel[2], Harish Nair[3], Yash Sheth[4], Prof. Payel Thakur[5]
[1, 2, 3, 4, 5]*Computer Department, Pillai College of Engineering, Maharashtra, India*

Abstract: ***The Multi-Agent Driving Game Simulator will consist of a real-time driving environment with accurate physics simulation. A population of AI drivers is trained using reinforcement learning. At each step the progress and live simulation is demonstrated. The environment can be made more complex as the AI gets better at traversing it. After sufficient training the efficiency and proficiency of the AI drivers can be demonstrated whereby they compete for driveable space and co- operate with other agents to minimize crashes. Later human decision makers can be introduced to the environment to further test the proficiency of AI in the presence of unpredictable scenarios. This type of dynamic and adaptive environment is a powerful test-bed for demonstration of reinforcement learning algorithms and their proficiency in solving hard problems by exploration in tandem with exploitation. This project is aimed at aiding the Autonomous Vehicle industry and provides a safe and quick assessment of the self-learning algorithms.***

I. INTRODUCTION

There are numerous techniques currently deployed in the self-driving space to reach autonomy. The two most prevalent techniques are Localization by LIDAR and end-to-end deep learning by computer vision. The former lacks in the task of accounting for the unpredictability that occurs in the driving task whereas the latter is limited by the amount and the quality of data available for training an end-to-end deep learning model. In this project, reinforcement learning techniques are used in a dynamic environment which makes the learning rate much more representative as the agent has more degree of freedoms in contrast to the aforementioned techniques

II. LITERATURE SURVEY

Driver Modeling through Deep Reinforcement Learning and Behavioral Game Theory. Berat Mert Albaba,Yildiray Yildiz. Advantages: Deep Q-Learning Algorithm gives excellent predictive power. Disadvantages: Complexity increases with the size of the environment.

Emergent Tool Use From Multi-Agent Autocurricula Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glen Powell, Bob McGrew, Igor Mordatch. Advantages: Demonstration of multi-agent Co-operation. Disadvantages: environment can be exploited by the model.

Deep Reinforcement Learning using genetic algorithms for Parameter Optimization Adarsh Sehgal, Hung Manh La, Sushil J. Louis, Hai Nguyen. Advantages: Combination of DDPG and Genetic algorithm is a powerful repertoire for a complete self-driving algorithm. Disadvantages: need for an optimizing tool for Reinforcement Learning Performance.

Exploration by Random Network Distillation.Yuri Burda ,Harrison Edwards ,Amos Storkey ,Oleg Klimov. Advantages: Parameter optimization and no bias. Disadvantages: Cannot be applied to high level exploration problems.

PPO DASH Improving generalization in deep reinforcement learning: Improving Generalization in Deep Reinforcement Learning Joe Booth. Advantages: Can be efficiently applied to a gamified environment with a large population of agents. Disadvantages: Large Number of iterations on a large population increases complexity.

III.ALGORITHMS AND TECHNIQUES

A. Existing System

Autonomous driving is not one single technology but rather a complex system integrating many technologies.The autonomous driving technology stack consists of three major subsystems: algorithms, including sensing, perception, and decision; client, including the operating system and hardware platform; and the cloud platform, including data storage, simulation, high-definition (HD) mapping, and deep learning model training. The algorithm subsystem extracts meaningful information from sensor raw data to understand its environment and make decisions about its actions. The client subsystem integrates these algorithms to meet real-time and reliability requirements. For instance, if the sensor camera generates data at 60 Hz, the client subsystem needs to make sure that the longest stage of the processing pipeline takes less than 16 milliseconds (ms) to complete. The cloud platform provides offline computing and storage capabilities for autonomous cars. Using the cloud platform, we are able to test new algorithms and update the HD map—plus, train better recognition, tracking, and decision models.

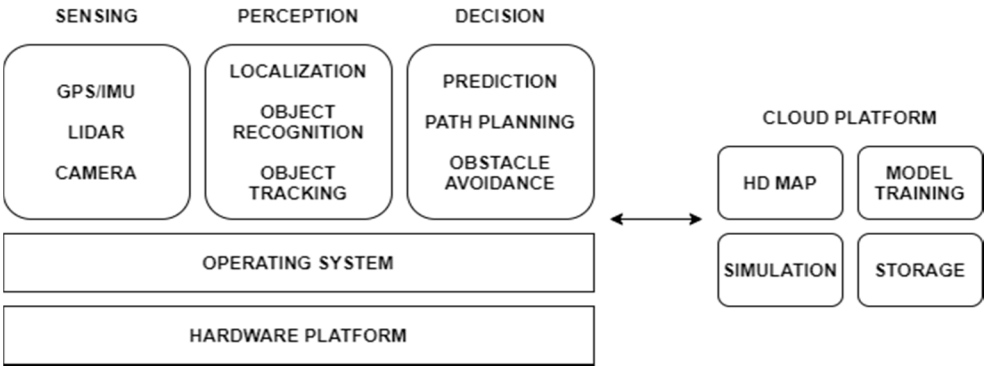


Fig. 3.2 Existing system architecture used for autonomous driving technologies [6]

The high level architecture of a content- based recommender system is depicted in Figure 3.2. The recommendation process is performed in three steps, each of which is handled by a separate component:

1) *Sensing*: A variety of sensors are used in autonomous driving like GNSS receivers, LiDAR and Cameras. GNSS receivers help autonomous vehicles localize themselves by updating global positions with at least meter-level accuracy. LiDAR is commonly used for creating High-Definition maps, real-time localization, as well as obstacle avoidance Cameras are mostly used for object recognition and tracking tasks, such as lane detection, traffic light detection, pedestrian detection.
2) *Perception:* It is about understanding the environment, including the current location, object recognition, and tracking etc. Several can be utilized to acquire accurate vehicle position updates in real-time. The natural choice for localization is to use GNSS directly.
3) *Decision Making:* In the decision making stage, action prediction and path planning mechanisms are combined to generate an effective action plan in real time. The main challenge of autonomous driving planning is to make sure that autonomous vehicles travel safely in complex traffic environments. The decision making unit generates predictions of nearby vehicles before deciding on an action plan based on these predictions. Planning the path of an autonomous vehicle in a dynamic environment is a complex problem, especially when the vehicle is required to use its full maneuvering capabilities.

B. Proposed System

Proposed System Architecture

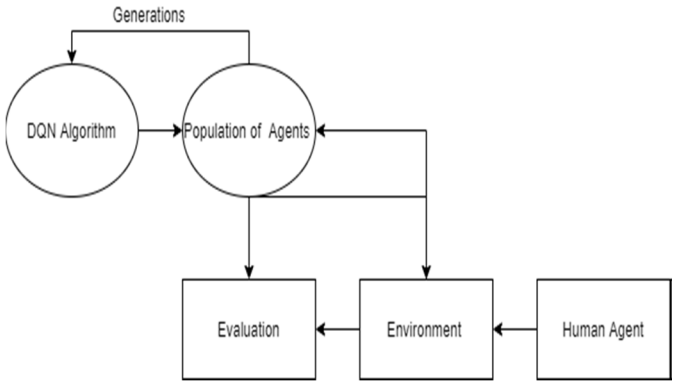


Fig. 3.3 Proposed system architecture

C. DDPG Algorithm

Deep Deterministic Policy Gradient or commonly known as DDPG is basically an off-policy method that learns a Q-function and a policy to iterate over actions. It employs the use of off-policy data and the Bellman equation to learn the Q function which is in turn used to derive and learn the policy.

The learning process is very closely related to Q-learning where if you know the optimal-action-value function Q*(s, a), the best and optimal action to taken in that state can be found out using a*(s) which is [3].

$$a^*(s) = \arg max_{\alpha} \; Q^*(s,a)$$

Q-learning based algorithms, specifically DDPG employs the use of the following to deal with a continuous action space: Make use of the Bellman equation to obtain the optimal action for a given state using its state-action/Q-value[3]

$$Q^*(s,a)= E_{s'\sim P}[r(s,a)+\gamma max_{a'}Q^*(s',a')]$$

DDPG employs the use of mean-squared Bellman error (MSBE) function which estimates how close Q* comes close to satisfying the Bellman equation as shown in the equation[3]:

$$L(\phi,D)=E_{(s,a,r,s',d)\sim D}[(Q\phi(s,a)-(r+\gamma(1-d)max_{a;}Q\phi(s',a')))^2]$$

D. Deep Q Learning Algorithm:

1) Initialize the memory D to capacity N
2) Initialize the main network and the target network. QN and QN^T. with weights sampled from a uniform distribution of range $[-\sqrt{\frac{6}{n_{input}+n_{output}}},\sqrt{\frac{6}{n_{input}+n_{output}}}]$ where n_{input} and n_{output} are the number of input and output neurons, respectively.
3) Set T = 50
4) for episode = 1 to M do
5) for t=1 to K do
6) Sample action at , using the probability values Pt(ai) = eQNt (ai)/Tj=0n-1eQNt (aj)/T, i= 1.2.....size(ActionSpace)
7) Execute action at , and observe the reward rt , and the transitioned state st+1
8) Store the experience (st , rt , at , st+1) in D
9) if size(D) ≥ n, then
10) Sample a random batch of experiences. consisting of P four-tuples (sj , rj, aj, sj+1)
11) for j = 1 to P do
12) Set yj =rj + γmax, QNT (sj+1 , a; W)
13) if sj+1 , is terminal, i.e. ego driver crashes. Then
14) Set yj = rj
15) end if
16) Perform a gradient descent step using the cost function yj-NN(sj+1, aj;W))2 with respect to weight matrix W
17) end for
18) end if
19) if sj+1 , is terminal, i.e. ego driver crashes, then
20) break
21) end if
22) end for
23) if T>1 then
24) Update Boltzmann Temperature T = T*c, c < 1
25) end if
26) end for

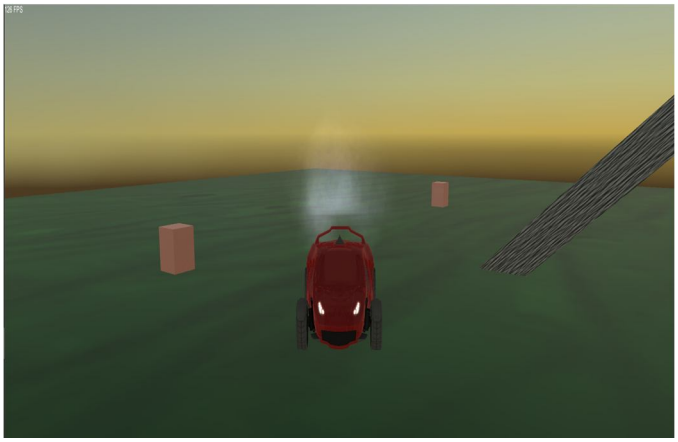


Fig. 3.4 Car Agent Asset

E. Proximal Policy Optimization

Policy gradient methods work by computing an estimator of the policy gradient and plugging it into a stochastic gradient ascent algorithm.

$$g\hat{} = E\hat{} t h \nabla\theta \log \pi\theta(at \mid st)A\hat{} t i$$

where $\pi\theta$ is a stochastic policy and $A\hat{} t$ is an estimator of the advantage function at timestep t. Here, the expectation $E\hat{} t [. . .]$ indicates the empirical average over a finite batch of samples, in an algorithm that alternates between sampling and optimization. Implementations that use automatic differentiation software work by constructing an objective function whose gradient is the policy gradient estimator; the estimator $\hat{}g$ is obtained by differentiating the objective

$$L P G(\theta) = E\hat{} t h \log \pi\theta(at \mid st)A\hat{} t i$$

While it is appealing to perform multiple steps of optimization on this loss L P G using the same trajectory, doing so is not well-justified, and empirically it often leads to destructively large policy updates.

F. PPO Algorithm, Actor-Critic Style

for iteration=1, 2, . . . do
    for actor=1, 2, . . . , N do
        Run policy $\pi\theta$old in environment for T timesteps
        Compute advantage estimates $A\hat{}1, . . . , A\hat{}T$
    end for
    Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
    θold $\leftarrow \theta$
end for

IV. IMPLEMENTATION DETAILS

The Reinforcement learning algorithms with optimizations can be inferred from the open-ai baselines paper. all the dependencies to train the agent are installed via the open source github repo openai/baselines.

The environment is made using unity and unreal engines to provide an in depth and high quality textures. The assets can be loaded in to the environment and their inputs and outputs can be controlled via C#.

The agent has a ray-perception sensor which collects observations from the environment in a spatial vector space. The size of this vector space depends on the number of spatial observations collected by the sensor. In the Driver agent, it receives a three dimensional vector space observation to detect the distance to the checkpoints and the wall. The actions taken by the agent are output in the form of float variables ranging from -1 to 1. This action space is called continuous actions and there are two continuous action vectors that the agent gives as an output.

The heuristic provides the information to the agent for mapping the output of the continuous action vector to the movement of the car. The turning of the wheel of the car is mapped to a continuous action that ranges from -1 to 1 and the other action is mapped to the acceleration/braking and reverse axis.

The performance of the agents on the environment is consistently evaluated and the data generated is stored in a database to further analyze the behavior and learning patterns in order to optimize the agents.

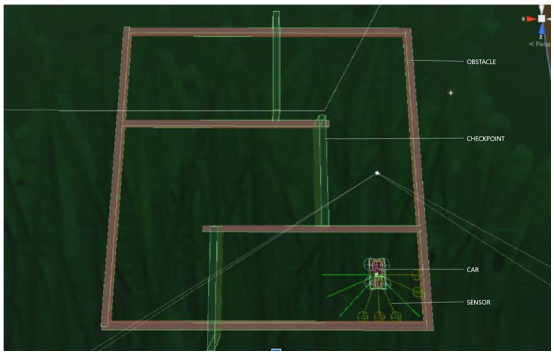


Fig. 3.5 Proposed system simulated environment

A. Evaluation Metrics

Reinforcement Learning Algorithms are evaluated at each step of the training process by pre-defined measurement parameters. In the case of PPO the fitness of the model is evaluated by using a Surrogate loss function.

Loss Function: The surrogate loss function used to evaluate the ppo model is robust in that hyperparameter initialization and search can lead to improved results.

A policy is a set of actions an RL agent can take. In policy gradient methods generally the agent starts with an initial policy, interacts with the environment, gets a reward, and the policy is improved using that reward. This results in a new policy. Policy gradient algorithms typically have two steps. In the first step, transitions are gathered. In the second step the policy is improved.

No clipping or penalty: $L_t(\theta) = r_t(\theta)\hat{A}_t$

Clipping:

$$L_t(\theta) = m \quad (r_t(\theta)\hat{A}_t, clip(r_t(\theta)), 1-\epsilon, 1+\epsilon)\hat{A}_t$$

KLpenalty (fixed or adaptive): $L_t(\theta) = r_t(\theta)\hat{A}_t - \beta KL[\pi_{\theta_{old}}, \pi_\theta]$

In Addition to this loss function the ppo algorithm also contains an entropy hyperparameter that modifies the surrogate function using the entropy coefficient. Entropy is a categorical distribution of the actions taken by the agent using the proximal policy and the loss function minimizes this entropy to ensure the actions taken by the agent are consistent and accurate. The entropy coefficient is represented mathematically below:

$$L_t^{CLIP+VF+S}(\theta) = \hat{E}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)]$$

c1 is the value function coefficient and c2 is the entropy coefficient.

PPO uses the Adam optimizer and it contains the hyperparameter for Adam Epsilon which is calculated at each step of training.

As the Agent Takes actions which consist of acceleration, braking and turning at each step. the hyperparameters are calculated and the loss function is updated.

B. Result Analysis

1) Cumulative Reward

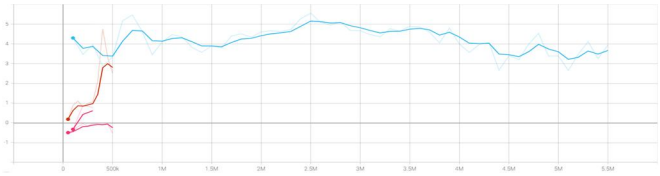


Fig 4.1 Environment/Cumulative Reward

This graph shows the appropriate reward distribution for 4 different training models trained at regular intervals. The more the car crashed the lesser the reward it gained. As it crossed each of the checkpoints the greater reward it generated resulting in reaching the desired goal state. The lowest run had just 500 thousand iterations with a reward score of -0.1 because the reward distribution was lower for the car to learn and it preferred to stay in a halted position and avoid negative reinforcement rather than move towards the right direction. The longest run according to the cumulative tensor board had 5 million iterations with a reward score of 3.8 consistently each of which helped the car learn more about the environment and the obstacles it had to avoid.

2) Value Loss

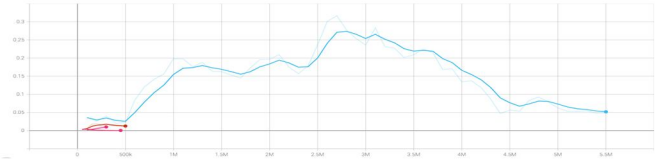


Fig 4.2 Losses/Value Loss

Value loss is a direct representation of how well our model fits within the given parameters set in the environment. Gradually the loss keeps increasing upto a certain peak value of 0.3 where the car keeps hitting the walls and learns that it gives a negative reward. Thereafter, it keeps reducing which ensures that the loss is minimized and the optimal loss is achieved at 5.5 million iterations in our longest run.

3) Entropy

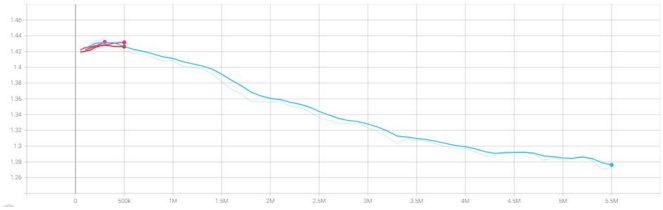


Fig 4.3 Policy/Entropy

In the PPO algorithm, the actions are defined as a probability distribution, condition on the state of the environment: p(a | s). When an agent takes a discrete action, picking one of many possible actions, a categorical distribution is used. In the case of a continuous control agent, a gaussian distribution with a mean and standard deviation may be used. With these kinds of policies, the randomness of the actions an agent takes can be quantified by the entropy of that probability distribution.
In the Driving agent implementation, The continuous actions taken by the agent constructs a probability distribution for each action space, In this case there are two action spaces for the agent so the entropy is calculated for each action and the total entropy is the accumulation of the probability distributions of both the actions. The 5 million Step Agent run demonstrates consistent decrease in entropy.

V. APPLICATIONS

A. Automated Cars

A fully automated vehicle's commercial availability appears to be adjusted around the corner. The final obstacles to commercial realization may be the necessary changes to road safety regulations and perhaps more difficult to define, the overcoming of individual and community fears of relying on the wheels of an automated machine on our roads. Tests on current automated vehicles, however are proving their reliability, with test results better than human judgments made on the road.

B. Vehicle Automation in Different Fields

We can use it in farming vehicles like tractors, irrigators and many more, it can also be used in mining vehicles like drilling rigs and it can also be implemented on industrial vehicles like forklift, robots and car crash testing vehicles.

C. GPS Acquisition and Processing

GPS is used to obtain the absolute position of an object in an open space environment. In our simulation, our proposed design is able to navigate a route based on GPS rather than pre saved maps that are not frequently updated and do not include all roads of all countries. In addition to gps there are gyroscope and accelerometers which help to contain the xyz axis coordinates of the ai based agent.

D. Landmark Assistance in local Positioning Systems

The car detects forms that are meant to be there as natural, such as trees, barriers, posts, et cetera. Usually, it may be difficult to locate natural landmarks, reducing system effectiveness. A tree, for instance, may change form, or obstacles emerge from various angles.
Artificial landmarks, in contrast, provide the autonomous vehicle with a simple, reliable way to define its location.

E. Control of the Automated Vehicle

The information acquired by the automated vehicle is used to drive. The vehicle controls the speed and direction of the vehicle. The first and foremost task of the vehicle is to control the motion along a specific line. So, it uses Proportional Integral Derivative to get the orientation difference and the distance. Mainly Proportional Integral Derivative checks where the automated vehicle should be and where it is. Reason behind this is to reduce the difference between it. For this automated vehicle applies a theoretical model to get the final result. This can help to minimize the human input in driving vehicles thus reducing the errors while controlling the agent.

F. Automated Vehicle path Planning

When an in line motion is achieved during autonomous vehicle control, the results can be extrapolated from one line to another. The vehicle agent can create paths of any desired shape which is achieved after determining its own path using decision rules. This can be helpful to normal, disabled and elderly people. It can be used on roads, facilities like factories, airports and campuses to transport passengers or loads while aiming to reduce overhead cost and energy for traditional path planning

.

G. Obstacle Avoidance

Obstacle avoidance For successful independent driving, automated vehicles are required to navigate a number of path modifications. If an obstacle is on the way to the initial route, an autonomous vehicle also needs to decide one of the path modifications by itself. An obstacle, if it is in motion, could be stationary or in a collision path. The autonomous vehicle has to take into account the vehicle model and the environmental map to decide the shortest route to regain the original path, taking into account the minimum clearance of the nearby barriers, in order to eliminate the barrier. Decision algorithms are used to segment the map into a grid to do so.

VI. FUTURE SCOPE

Provide a test bed for reinforcement learning algorithms and different driving scenarios. The environment and the agent can be deployed in any environment and result analysis and comparison can be done between different scenarios of driving.

The project can be improved by using various emergency scenarios, and also by including more complex environments. Opportunities for analyzing the safety demographics of high diversion or accident-prone areas like construction sites, curbs, hairpin turns and steep slopes.

VII. SUMMARY

Reinforcement learning in real world autonomous driving applications is still an evolving field. Although a few commercial applications are successful, very little literature or large-scale public datasets are available. Therefore, we were inspired to formalize and coordinate RL autonomous driving applications. Interacting agents are involved in autonomous driving scenarios which include negotiation and complex decision making that fits RL. However in order to provide mature solutions that we address in depth, there are several problems to be solved.

VIII. ACKNOWLEDGEMENT

We are also grateful to Dr. Sandeep Murlidhar Joshi, principal of Pillai College of Engineering for giving us the opportunity to work with them and providing us the necessary resources for the project.

We would like to express our deep gratitude to Dr. Sharvari S. Govilkar, our Head of Department for Computer Engineering, for their patient guidance, enthusiastic encouragement and useful critiques of this research work.

We would like to convey our sincere gratitude to Prof Payel Thakur for her useful and positive feedback during the preparation and production of this Project. Her willingness to give her time so generously was very much appreciated.

REFERENCES

[1] B. Albaba and Y. Yildiz, "Driver Modeling through Deep Reinforcement Learning and Behavioral Game Theory", Arxiv.org, 2020. Available: https://arxiv.org/pdf/2003.11071.pdf. [Submitted on 24 Mar 2020]

[2] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever and I. Mordatch, "Emergent Complexity via Multi-Agent Competition", arXiv.org, 2020. Available: https://arxiv.org/abs/1710.03748. [Submitted on 10 Oct 2017 (v1), last revised 14 Mar 2018 (this version, v3)]

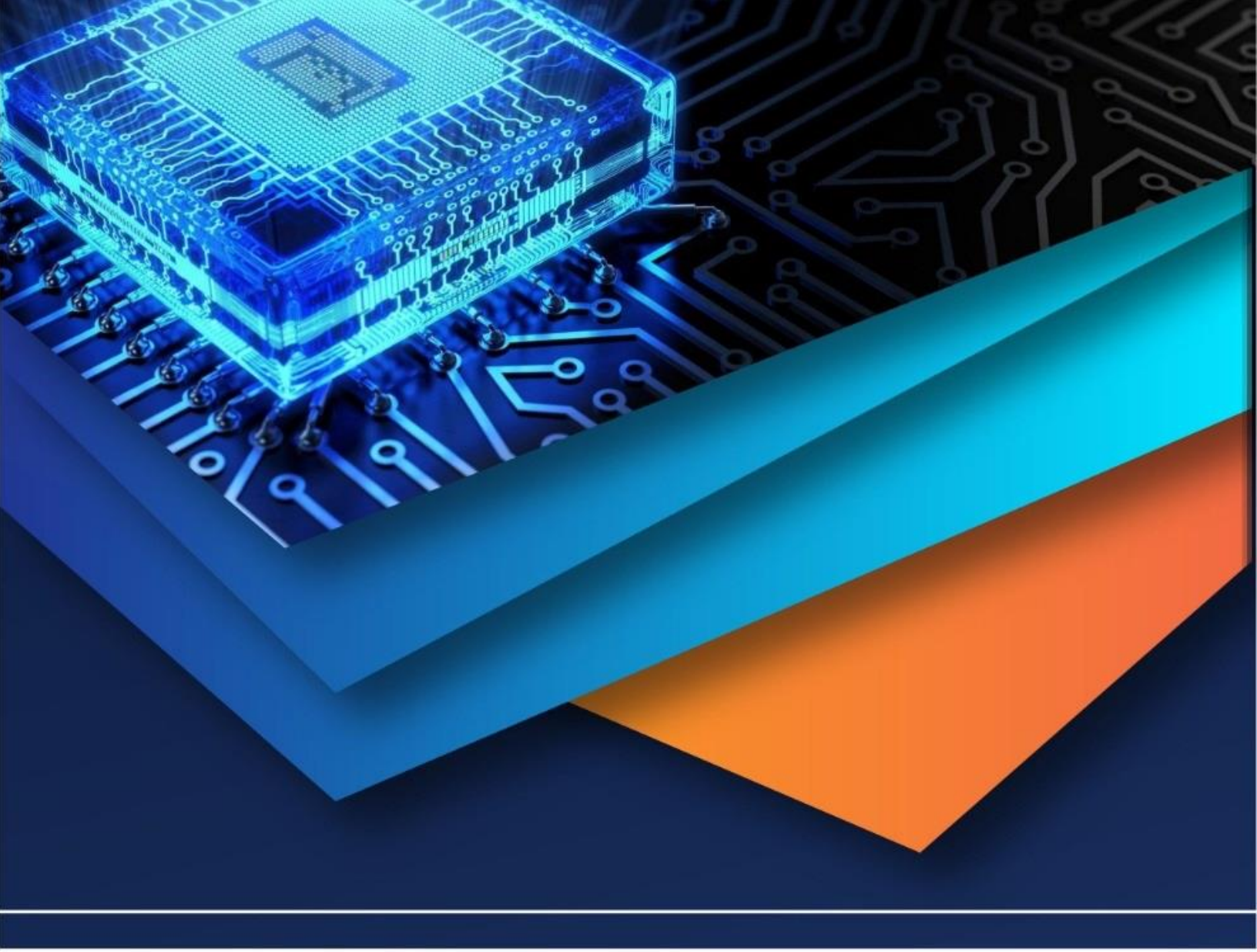
[3] T. Salimans, J. Ho, X. Chen, S. Sidor and I. Sutskever,"Evolution Strategies as a Scalable Alternative to Reinforcement Learning" Arxiv.org, 2020. Available: https://arxiv.org/pdf/1703.03864. [Submitted on 10 Mar 2017 (v1), last revised 7 Sep 2017 (this version, v2)]

[4] Y. Burda, H. Edwards, A. Storkey and O. Klimov, "Exploration by Random Network Distillation", arXiv.org, 2020. Available: https://arxiv.org/abs/1810.12894. [Submitted on 30 Oct 2018]

[5] Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell and A. A. Efros, Pathak22.github.io, 2020. Available: https://pathak22.github.io/large-scale-curiosity/resources/largeScaleCuriosity 2018. [Submitted on 13 Aug 2018]

[6] https://www.oreilly.com/radar/creating-autonomous-vehicle-systems

[7] Kaelbling, Leslie P.; Littman, Michael L. Moore, Andrew W. (1996). "Reinforcement Learning: A Survey". Journal of Artificial Intelligence Research. 4: 237–285. arXiv:cs/9605103. doi:10.1613/jair.301. S2CID 1708582. Archived from the original on 2001-11-20.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY