iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 10 **Issue:** VI **Month of publication:** June 2022

DOI: https://doi.org/10.22214/ijraset.2022.44784

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: 7.538
Volume 10 Issue VI June 2022- Available at www.ijraset.com

Rapid Autonomous Vehicle Drifting with Deep Reinforcement Learning

Surekha Samsani[1], Phani Sasidhar Bollapragada[2]
[1]*Assistant Professor,* [2]*MCA. Student, Department of Computer Science & Engineering, University College of Engineering, Kakinada (Autonomous), Jawaharlal Nehru Technological University Kakinada, Kakinada, India*

Abstract: ***Now days all types of vehicles run on road are upgrading to automated self-driving techniques. Drifting is a complicated procedure for autonomous vehicle control which is used for high-speed sideslip cornering. It is difficult to drift a vehicle at a high speed of 80-120 km/hr by using traditional techniques like two-state single-track model and three-state single-track models, which depend on the knowledge of tire or road forces that are not to be known accurately due to the real-world environmental complexity. Because of these inaccuracies in these parameters there will be poor control performance and may lead to road accidents. In this scenario, this project presents a strong drift control algorithm, which is based on the most recent model-free deep reinforcement learning algorithm called soft actor-critic (SAC). SAC is used to control front-wheel drive (FWD) vehicles to drive at high speed (80– 120 km/h), and to drift through sharp corners quickly and stability which in turn helps to prevent road accidents at the time of drifting. The efficiency of this algorithm is evaluated by conducting experiments on Trajectories dataset taken from Git-Hub repository, which consists of some road maps with reference drift trajectories. The SAC algorithm can also deal with vehicle types with various actual properties, like mass, tyre friction, and so on to show its notable generalization ability.***

Keywords: Reinforcement Learning, Soft Actor Critic, Front Wheel Drive, Autonomous Drifting.

I. INTRODUCTION

The high-speed sideslip cornering, known as drifting, represents a difficult vehicle control. In order to make a fast turn through sharp corners, experienced drivers execute drifts by consciously inducing deep saturation of the rear tires by oversteering or using the throttle, thereby drifting the vehicle.

They then stabilise the vehicle as it begins to spin by controlling it under a high sideslip operation. Vehicle instability and corresponding control difficulty both increase as the sideslip angle increases. Therefore, drifting is a risky control technique to operate the vehicle efficiently and safely beyond its limits. Compared with the normal cornering in which drifting is usually avoided by lowering the speed and making slower turns but there is a chance of getting overspeed at the corners. high-speed drifting techniques can help decrease the accidents during vehicle driving at high speed.

During drifting, a list of high-frequency decisions like steering and throttle should be controlled correctly and safely. Therefore, by studying drift behaviours, we can design drift control algorithm which fully cover vehicle dynamics to high-speed sideslip cornering.

A. Introduction to Reinforcement Learning

Reinforcement learning is an area of Machine Learning. It is about taking related action to increase reward in a particular condition. It is utilized by various software and machines to find the best possible path it should take in a particular situation. Reinforcement learning is different from supervised learning in a way that in supervised learning the training data has the answer with it so the model is trained with the correct way itself but in reinforcement learning, there is no answer but the reinforcement agent decides to perform the given task. In the absence of a training dataset, it is bound to learn from its experience.

This type of machine learning wants the use of a reward/penalty system. The aim is to reward the machine when it understands correctly and to warn or stop the machine when it learns incorrectly.

Reinforcement Machine Learning is a part of Artificial Intelligence. With the broad range of possible answers from the data, the process of this type of learning is an iterative step.

©IJRASET: All Rights are Reserved

B. Types of Reinforcement Learning

There are mainly three ways to implement reinforcement-learning in ML as shown in figure 1.1, which are:

1) *Value-based:* The value-based approach is about to find the best value function, which is the higher value at a state under any policy. Therefore, the agent expects the return in long term at any state(s) under policy π.
2) *Policy-based:* Policy-based approach is to find the better policy for the increasing the future rewards without using the value function. In this approach, the agent applies such a policy that the action done in each step helps to increase the future reward.

The policy-based approach has mainly two types of policy:

- *Deterministic:* The same action is produced as output by the policy (π) at any state.
- *Stochastic:* In this policy, probability gives a produced action.

3) *Model-based:* In the model-based approach, a virtual model is created for the environment, and the agent travel entirely in that environment to learn it. There is no correct solution or algorithm for this approach because the model representation is varies for each environment.

II. RELATED WORK

This project is about whether a vehicle can drift accurately at turnings at a speed of 80 to 120 kmph automatically without any interference of driver by using reinforcement learning algorithms. For that an important model free reinforcement learning algorithm is used which is called as Soft Actor Critic (SAC).

The SAC algorithm is implemented on the trajectories dataset which consists of different road map's slip angles, turning ratios, throttle, steering angle and reference drift trajectories that make the vehicle to drift automatically by using a vehicle simulator called as CARLA and observe whether the vehicle is drifting or not.

If the vehicle is drifting then the result of the different drifts performed by it in the map using SAC are calculated and compared with different other reinforcement learning algorithms to measure the accuracy and prove SAC is better algorithm for autonomous vehicle drifting.

A. Basic Structure and algorithm of SAC:

SAC controller training algorithm

Data: Buffer D, total number of transitions N, update threshold η, number of updates λ

Result: Optimal control policy π_φ

1) Initialize parameters of all networks
2) D <- θ, N <- 0, s_t <- s_0;
3) For each round do
4) While $s_t \neq s_T$ do
5) at ~ $\pi\varphi(a_t|s_t)$, s_{t+1}~$p(s_{t+1}|s_t,a_t)$;
6) D <- D U $\{s_t,a_t,r(s_t,a_t),s_{t+1})\}$;
7) N <- N+1, s_t <- s_{t+1};
8) End
9) If N ≥ η then
10) Update all networks λ times;
11) End
12) End

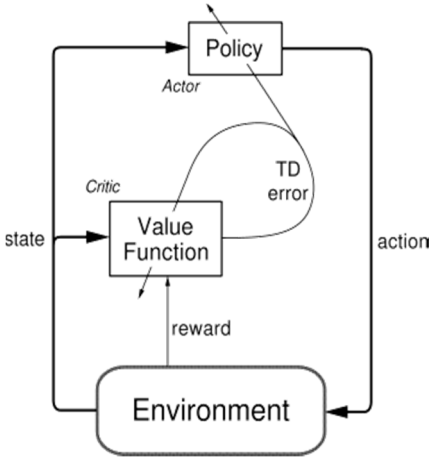
B. SAC Algorithm

Soft Actor-Critic (SAC) is one of the states of the art reinforcement learning algorithm developed jointly by UC Berkely and Google. It is considered as one of the most efficient RL algorithms to apply in real-world robotics.

The algorithm is based on a maximum entropy RL where the objective is to find the optimal policy that maximises the expected long-term reward and long-term entropy.

$$J(\pi_\theta) = E_{\pi_\theta}[\sum_{t=0}^{T-1} \gamma^t R(s_t, a_t) + \alpha H(\pi(.|s_t))]$$

The objective function of Maximum entropy RL.



Actor-Critic architecture.

As the name suggests SAC is an actor-critic algorithm as shown in the above figure. In a simple description, actor-critic is a combination of policy-based and value-based approach.

Learning of the actor is based policy gradient approach and critic is learned in value-based fashion. In SAC, there are three networks: the first network represents state-value(V) parameterised by ψ, the second one is a policy function that parameterised by ϕ, and the last one represents soft q function parameterised by θ.

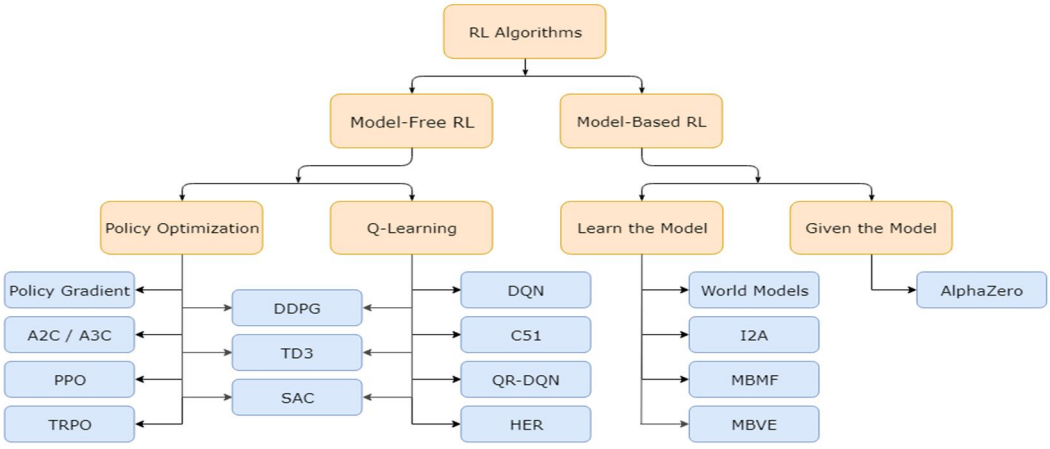
III. MATERIALS AND METHODS

A. *Dataset*

1) *Dataset Description:* There are six different Trajectories Dataset provided by the Git Hub Repository contains the vehicle world location, heading angles, body-frame velocities and slip angles, to provide reference states for training and evaluation.
2) *Description of the Features :* Each data set contain more than 3000 rows and 11 columns. Outcome is the column, which is going to predict, which says if the drifting properly or not It also shows how much time a vehicle is taking to drift and how much time it is taking to complete one round. The below two figure shows the datasets used for drifting with its outputs and its way points as follows.

B. *Reinforcement Learning Techniques*

Various reinforcement learning algorithms are used for drifting vehicle. There are many other methods for drifting vehicles but following Reinforcement learning Techniques are used in this work. All the work was done on Spider in anaconda and Carla simulator.



Types of Reinforcement learning Algorithms

We now concentrate on the value-based method, which is in "model-free" approaches, and more specifically, we will know the DQN method, which belongs to Q-learning. For that, we review some necessary mathematical background. Let's define some mathematical quantities:

1) Expected Return

An RL agent objective is to find a policy which optimizes the expected return (V-value function):

$$(1)\ (s) = E \quad \Upsilon k r t+k \mid st = s, \pi$$

k=0 ∞

where E is expected value operator, ϒ is the discount factor, and pi is a policy. The optimal expected return is defined as:

$$(2)\ V*(s)=max\pi\ V\pi(s)$$

The optimal V-value function is the desired discounted reward when in a given state s the agent continue with the policy pi* thereafter.

2) Q value

There are some other functions of interest. One of them is the Quality Value function:

$$(3)\ (s, a) = E\ \Upsilon^k r_{t+k} \mid s_t = s, a_t = a, \pi$$

k=0 ∞

Similarly, to the V-function, the optimal Q value is given by:

$$(4)\ Q*(s, a) = max\pi Q\pi(s, a)$$

The optimal Q-value is the expected discounted return when in a given state s and for a given action, a, the agent follows the policy pi* thereafter. The optimal policy can be obtained directly from this optimal value:

$$Q^{\pi}(s) = \arg max_a\ Q^{\pi}(s, a) \qquad (5)$$

3) Advantage Function

We can relate between the last two functions:

$$A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s) \qquad (6)$$

It describes "how good" the action a is, compared to the expected return when following direct policy pi.

4) Bellman equation

To learn the Q value, the Bellman equation is used. It promises a unique solution Q*:

$$Q^*(s, a) = (BQ^*)(s, a) \qquad (7)$$

where B is the Bellman operator.

$$BQ^*(s,a) = \sum_{s' \in S} T(s, a, s')(R(s, a, s') + \gamma max_{a'} Q^*(s', a')) \qquad (8)$$

To promise optimal value: state-action pairs are represented discretely, and all actions are repeatedly sampled in all states.

C. Q-Learning

Q learning in an off-policy method takes the value of action in a state and learning Q value and choosing how to run in the actual environment. We define state-action value function: an expected return when starting in s, performing a, and following pi. Represented in a tabulated form. According to Q learning, the agent uses any policy to estimate Q that maximizes the future reward. Q directly approximates Q*, when the agent keeps updating each state-action pair.

$$Q^{\pi}_{t+1}(s_t, a_t) = (1-\alpha)Q^{\pi}_{t}(s_t, a_t) + \alpha(R_t + \gamma \max_a Q^{\pi}_{t}(s_{t+1}, a)) \quad (9)$$

For non-deep learning approaches, this Q function is just a table:

	a 1	a 2	a 3	a 4
s 1	10	52	15	-2
s 2	14	30	8	7
s 3	42	0	-5	-10
s 4	-3	-1	-7	-20

states ↕ actions ↔

Table representing Q functions

In the above table, each one of the elements is a reward value, which is updated every time during the training so it will be in steady-state, it should reach the required/desired value of the reward with the discount factor that is equivalent to the Q* value. In real-world scenarios, value looping is impractical. Hence, the number of rows in the Q-table is:

$256^{84\times84\times4} \approx 10^{69,970}$

D. DQN: Deep Q-Networks

We use a neural network to approximate the Q function:

$$Q(s, a; \theta) \approx Q^*(s, a) \quad (10)$$

The neural network is same as a function approximator. DQN was used in the Atari games. The loss function has two Qs functions:

$$L = E[(r + \gamma \max_{a'} Q(s', a'; \theta_k) - Q(s, a; \theta_k))^2] \quad (11)$$

Target: the predicted Q value of taking action in a particular state. Prediction: the value you get when actually taking that action (calculating the value on the next step and choosing the one that minimizes the total loss).

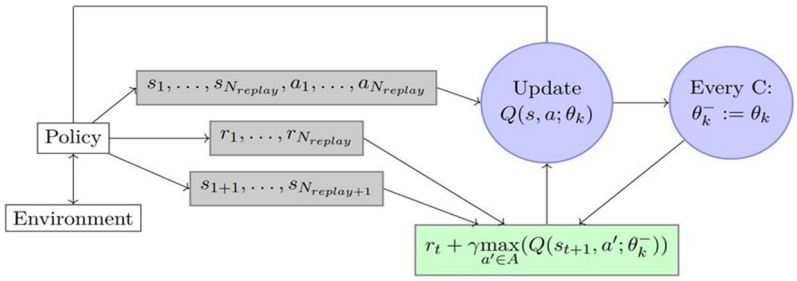
E. Parameter Updating

$$\theta_{k+1} = \theta_k + \alpha(r + \gamma \max_{a'} Q(s', a', \theta_k))\nabla_{\theta_k} Q(s, a; \theta_k) \quad (12)$$

When updating the weights, 1 also changes the target. Due to the generalization of neural networks, large or big errors are occurred or created in the state-action space. Hence, the Bellman equation is not converged. Errors may propagate with this update rule (slow / unstable/ etc.).

DQN algorithm can produce strong performance in an online setting for a different ATARI game and directly learns from pixels.

Two heuristics to limit the instabilities: 1. The parameters of the required or main Q-network are updated only every N loops. This stops the instabilities to propagate quickly and decreases the risk of divergence.



DQN Architecture (MDPI: Comprehensive Review of Deep Reinforcement Learning Methods and Applications in Economics)

Interpolating Between Policy Optimization and Q-Learning. Serendipitously, policy optimization and Q-learning are not incompatible, and there is a range of algorithms that are present in between the two extremes. Algorithms that are in this spectrum are able to carefully trade-off between the strengths and weaknesses of both sides. Examples include

- DDPG, an algorithm which continuously learns a deterministic policy and a Q-function by using each to improve the other,
- and SAC, a same type with different features which uses stochastic policies, entropy regularization, and a few different tricks to do a constant learning and score more than DDPG on standard benchmarks.

What to Learn in Model-Based RL?

Unlike model-free RL, there are no small number of easy-to-define clusters for model-based RL. There are many different orthogonal ways of using models. We'll give a few examples, but the list is far from exhaustive. In each case, the model may either be given or learned.

Background: Pure Planning. The most basic approach is never explicitly representing the policy, and instead of that we use pure planning techniques like model-predictive control (MPC) to select actions. In MPC, every time the agent carefully observes the environment and computes a plan which is optimal with respect to the model, where the plan shows all actions to take some fixed window of time after the present. The agent then executes the first action of the plan, and remove the rest of it. It analyse a new plan each time it prepares to communicate with the environment, to avoid using an action from a plan with a shorter-than-desired planning horizon.

- The MBMF work explores MPC with learned environment models on some standard benchmark tasks for deep RL.

Expert Iteration. A correct follow-on to pure planning have to use and learn an explicit representation of the policy. The agent utilizes a planning algorithm (like Monte Carlo Tree Search) in the model, producing candidate actions for the plan by sampling from its latest policy. The planning algorithm creates an action which is more better than what the policy only produce, so it is an "expert" relative to the policy. The policy is then updated to produce an action more like the planning algorithm's output. The ExIt algorithm uses this approach to train deep neural networks. Alpha Zero is another example of this approach.

Data Augmentation for Model-Free Methods. Use a model-free RL algorithm to train a policy or Q-function, but either 1) augment real experiences with fictitious ones in updating the agent, or 2) use only fictitous experience for updating the agent.

- See MBVE for an example of augmenting real experiences with fictitious ones.
- See World Models for an example of using purely fictitious experience to train the agent, which they call "training in the dream."

Embedding Planning Loops into Policies. Another approach is planning procedure directly to a policy as a subroutine so that total plans become side information for the policy while training the output of that policy with any of the standard model-free algorithm. The key concept is that in this framework, the policy can learn to choose how and when to use the plans. This makes model bias less of a problem, because if the model is bad for planning in some states, the policy can simply learn to ignore it.

IV. RESULTS

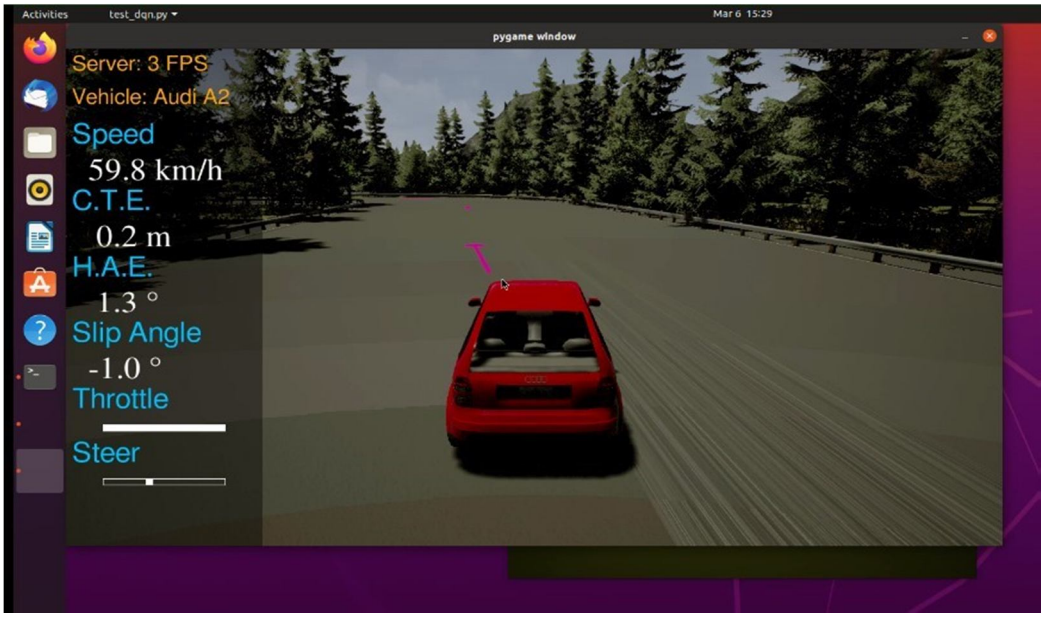
By using the above code related to different reinforcement learning algorithms in the Carla simulator the autonomous vehicle drifting is done which can be observed from figure 3.1 and 3.2 , the below two tables are created by observing the C.T.E, H.A.E, Speed, and Slip angle of the vehicle drifting using different RL algorithms in Carla simulator as in figure 3.1and 3.2 which shows the results of the autonomous vehicle drifting done by implementing three different reinforcement learning algorithms like DQN DDPG and SAC.

Vehicle	Methods	Performance at Corners			
		C.T.E.↓	H.A.E.↓	AVG-VEL↑	SLIP↑
		(m)	(o)	(km/h)	(o)
Audi A2	DQN	Audi A2	3.121	77.89	19.43
	DDPG	2.665	5.532	52.43	29.01
	SAC	1.143	3.012	79.76	29.23

3.1 Table representing performance over whole track

Vehicle	Methods	Performance over the whole track				
		C.T. E↓	H.A.E.↓	MAX-VEL↑	L.T↓	SMOS↓
		(m)	(o)	(km/h)	(s)	
Audi A2	DQN	1.288	7.941	98.57	150.89	0.132
	DDPG	2.045	14.678	90.59	215.45	0.401
	SAC	0.899	5.699	103.71	145.9	0.13

3.2 Table representing performance at corners



Vehicle drifting is shown with different metrics

We adopt seven metrics to measure the performance of different methods.
C.T.E. and H.A.E. is the cross-track error and heading angle error, respectively. The down arrow in these columns should be minimum as smaller value will be considered,
MAX-VEL and AVG-VEL is the maximum and average velocity of a driving test, respectively. Here the up arrow means both the velocities maximum value is considered.
L.T. is the time to reach the destinations (the lap time). Minimum will be considered
SMOS measures the smoothness of driving, calculated by the rolling standard deviation of steering angles during a driving test.
SLIP is the maximum slip angle during a driving test. Since larger slip angles mean larger usable state spaces beyond the handling limits, it can indicate a more powerful drift controller.

Comparing results of Proposed system with Existing Systems
The above table is mainly divide the results of drifting into two parts i.e. performance through whole track and performance through corners.
Where the CTE and HAE are two error identifiers present in both parts in table which get lesser values while using SAC when compared with existing systems like DQN and DDPG i.e., 0.899 and 5.699.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 10 Issue VI June 2022- Available at www.ijraset.com

Then coming to the max velocity and Average velocity where there are present in the performance over the whole track and performance through corners. These two velocities are showing maximum values while using SAC compared to DDPG and DQN i.e., 103.71 and 79.76

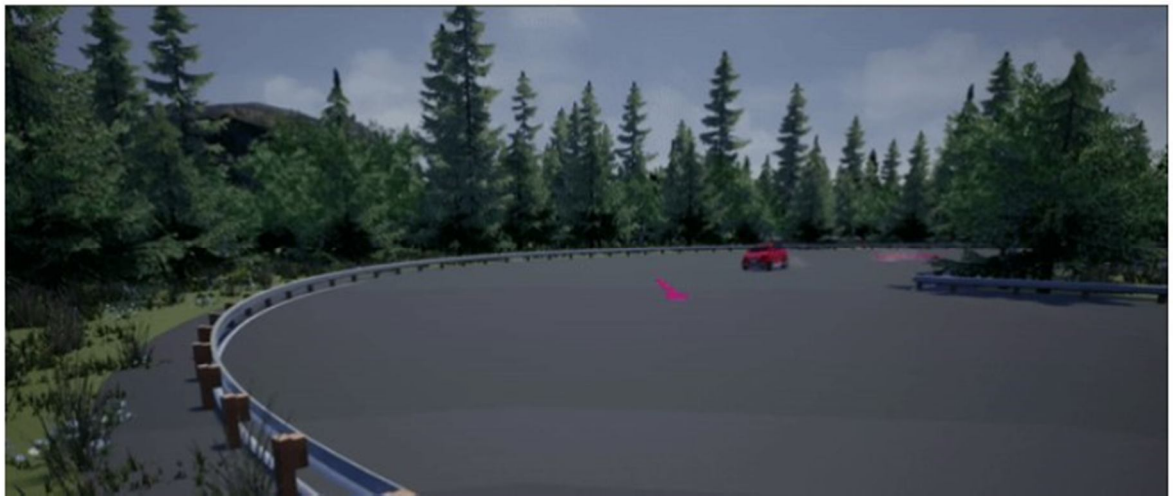
The remaining LT, SMOS an SLIP are representing the lap time, smoothness in drift and slip angle where these three should be minimum for accurate or perfect drifting.

While comparing SAC with DDPG and DQN these metrics show less values than existing system algorithms values. So, from the above table we can prove than SAC is the better reinforcement learning algorithm for autonomous vehicle drifting.

In the simulator we can see the vehicle drifting in the following figures from three different angles.







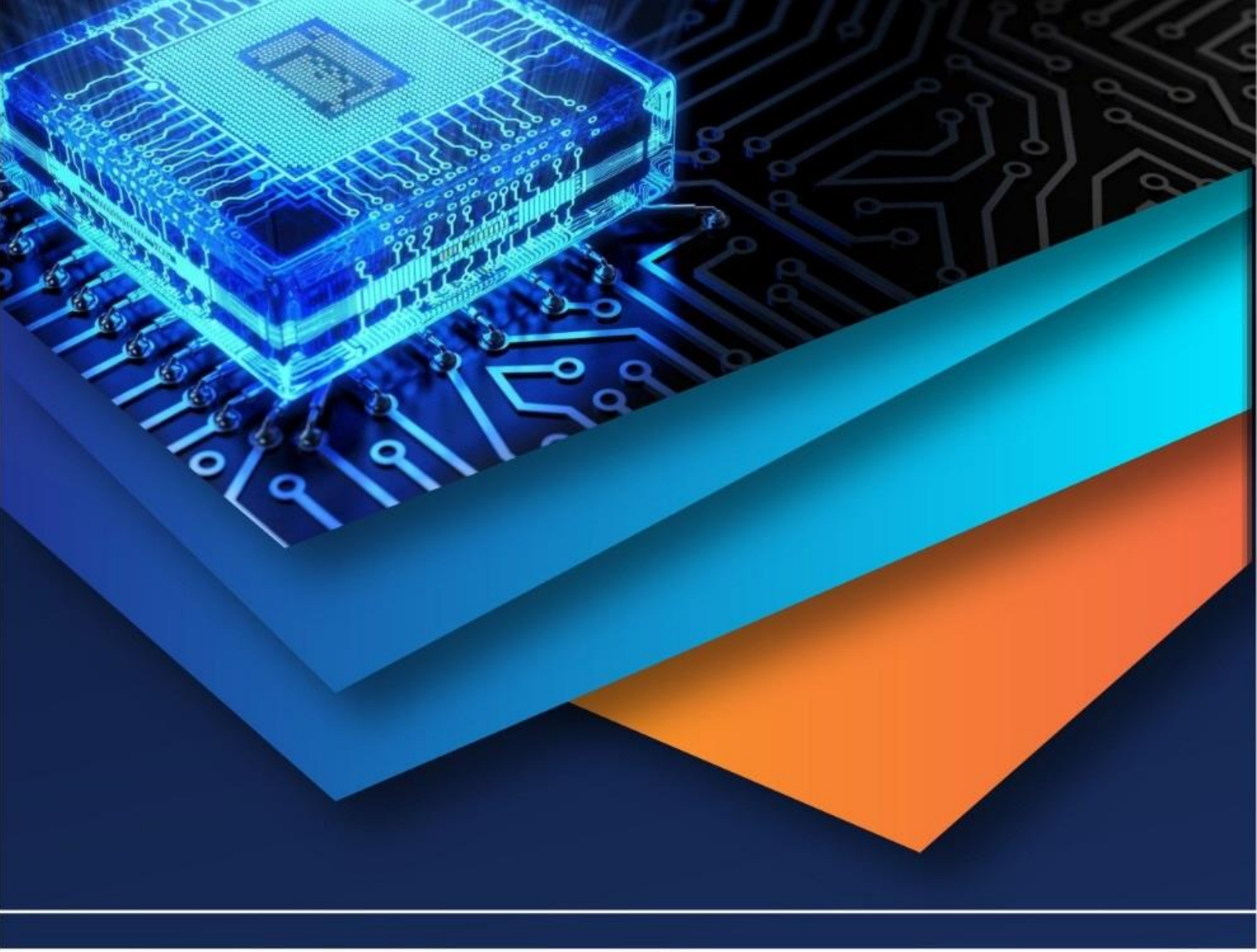
©IJRASET: All Rights are Reserved

V. CONCLUSION

In this project, to realize high-speed drift control through manifold corners for autonomous vehicles, we propose a closedloop controller based on the model-free deep RL algorithm soft actor critic (SAC) to control the steering angle and throttle of simulated vehicles. The error-based state and reward are carefully designed and an action smoothing strategy is adopted for stable control outputs. Maps with different levels of driving difficulty are also designed to provide training and testing environments. After the twostage training on six different maps, our SAC controller is sufficiently robust against varied vehicle mass and tire friction to drift through complex curved tracks quickly and smoothly. Moreover, we have discussed the necessity of slip angle information, and the nondegraded performance with a rough and easy-to-access reference trajectory during testing, which is valuable for applications. So, by this project it is concluded that the use of SAC algorithm for vehicle drifting will prevent accidents at the time of drifting.
This project can be further developed to drift two or more vehicles at a time in single lane in same direction or in opposite direction by giving different way points using different datasets.

REFERENCES

[1] I. Zubov, I. Afanasyev, A. Gabdullin, R. Mustafin, and I. Shimchik, "Autonomous drifting control in 3d car racing simulator," in 2018 International Conference on Intelligent Systems (IS). IEEE, 2018, pp 235–241.
[2] C. Voser, R. Y. Hindiyeh, and J. C. Gerdes, "Analysis and control of high sideslip manoeuvres," Vehicle System Dynamics, vol. 48, no. S1, pp. 317–336, 2010.
[3] M. Acosta and S. Kanarachos, "Teaching a vehicle to autonomously drift: A data-based approach using neural networks," Knowledge-Based Systems, vol. 153, pp. 12–28, 2018.
[4] F. Zhang, J. Gonzales, K. Li, and F. Borrelli, "Autonomous drift cornering with mixed open-loop and closed-loop control," IFAC-PapersOnLine, vol. 50, no. 1, pp. 1916–1922, 2017.
[5] F. Zhang, J. Gonzales, S. E. Li, F. Borrelli, and K. Li, "Drift control for cornering maneuver of autonomous vehicles," Mechatronics, vol. 54, pp. 167– 174, 2018.
[6] S. Bhattacharjee, K. D. Kabara, R. Jain, and K. Kabara, "Autonomous drifting rc car with reinforcement learning," 2018.
[7] P. Frère, Sports Car and Competition Driving. Pickle Partners Publishing, 2016.
[8] J. Y. Goh, T. Goel, and J. C. Gerdes, "A controller for automated drifting along complex trajectories," in 14th International Symposium on Advanced Vehicle Control (AVEC 2018), 2018.
[9] J. Y. Goh and J. C. Gerdes, "Simultaneous stabilization and tracking of basic automobile drifting trajectories," in 2016 IEEE Intelligent Vehicles Symposium (IV). IEEE, 2016, pp. 597–602.
[10] E. Velenis, D. Katzourakis, E. Frazzoli, P. Tsiotras, and R. Happee, "Steady-state drifting stabilization of rwd vehicles," Control Engineering Practice, vol. 19, no. 11, pp. 1363–1376, 2011.
[11] E. Velenis and P. Tsiotras, "Minimum time vs maximum exit velocity path optimization during cornering," in 2005 IEEE international symposium on industrial electronics. Citeseer, 2005, pp. 355–360.
[12] R. Y. Hindiyeh and J. C. Gerdes, "A controller framework for autonomous drifting: Design, stability, and experimental validation," Journal of Dynamic Systems, Measurement, and Control, vol. 136, no. 5, p. 051015, 2014.
[13] M. Cutler and J. P. How, "Autonomous drifting using simulation-aided reinforcement learning," in 2016 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2016, pp. 5442–5448.
[14] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in Proceedings of the 28thInternational Conference on machine learning (ICML-11), 2011, pp.465–472.
[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G.Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovskiet al., "Humanlevel control through deep reinforcement learning, "Nature, vol. 518, no. 7540, p. 529, 2015.
[16] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa,D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," arXiv preprint arXiv:1509.02971, 2015.
[17] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Offpolicy maximum entropy deep reinforcement learning with a stochastic actor," arXiv preprint arXiv:1801.01290, 2018.
[18] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," arXiv preprint arXiv:1711.03938, 2017.
[19] Nexon, "Popkart," 2019. [Online]. Available: https://popkart.tiancity.com/homepage/v2/guide/runway.html
[20] VectorZero, "Roadrunner," 2019. [Online]. Available: https://www.vectorzero.io/roadrunner



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)