# Visualization of Autonomous Car: Evolutionary Machine Learning Model

Nisarg Vaghela
*Nirma University*
*name of organization (of Aff.)*
Ahmedabad, India
16bit085@nirmauni.ac.in

Rutva Patel
*Nirma University*
*name of organization (of Aff.)*
Ahmedabad, India
16bit129@nirmauni.ac.in

***Abstract*—The automotive industry as well as academia are currently conducting a lot of research related to autonomous driving. Autonomous driving is an interesting topic that holds the potential to benefit society in many ways, such as reduce the number of fatalities and reducing the environmental footprint of modern day traffic. In this paper we intend to show the visualization of a Self Driving car by implementing the Differential Evolution Algorithm.**

***Index Terms*—Differential Evolution Algorithm, Reinforcement Learning, Fitness function**

## I. Introduction

Self Driving cars has been a large research area in the last few years, and is most likely to remain in the research framework until they can be implemented into all kinds of environments without any safety failures. A popular approach to deal with high data input conundrums in the field of Machine Learning is Deep Learning. Deep Learning strategies are fit for extracting important features from high-dimensional data, by passing the high-dimensional information through a function that uses a multi-layer computational diagram called a Neural Network. However, in recent years instead of training the data on labelled datasets which most of the times do not work appropriately in real life data, the research is shifted more towards Reinforcement learning models which are trained by interacting with the environment. The RL models explore the environment in an iterative manner and find an optimal solution or path. Primarily these deep reinforcement learning models have been deployed in games, such as old Atari games [2] and the ancient game of Go [3]. There have also been some breakthroughs for continuous control Reinforcement Learning methods. These algorithms have primarily been evaluated with good results on the MujoCo physics engine, which acts as a benchmark for reinforcement learning algorithms. One of the drawbacks is that deep learning networks requires plethora of varied data inspite of the existence of data augmentation methods and it takes immense time to train such large networks. In the case of self driving cars, it is difficult to obtain training dataset because it is very likely to assume the real world scenarios as the ideal driving conditions. As a solution to this problem, we can easily stimulate the non ideal scenarios into the synthetic environment. It is impossible for an RL model to learn in a real environment as the chances of accidents or the car being crashed is very high.

## II. Related work

The topic of this thesis is to investigate recent methods in the field of Artificial Intelligence (AI), for autonomous driving. There are numerous implementations in both simulation environments and the physical world. In 2005 the DAVE project developed an end to end solution to control a RC-car to navigate in outdoor environments. They trained a shallow network on data collected by human interaction and achieved results where the agent was capable of driving roughly 20 meters without bumping into obstacles. The DAVE project has expanded in recent years and upgraded their hardware for their robots. The new hardware has been used in research of learning long-range vision for autonomous driving. A group of developers at NVIDIA were inspired by the work in DAVE and developed DAVE2. The project used modern hardware and created an autonomous agent that is capable of driving a full-sized car in urban environments. What is most impressive is that the Neural Network in DAVE-2 only takes inputs from camera images captured by the camera of the car and the current steering wheel angle. Similar to DAVE-2 Brody Huval et al. , have applied Neural Networks to images captured in front of a car. The projects focus was to implement a real-time system evaluating Neural Networks performance in highway environment detecting lanes and cars. Similar to DAVE and DAVE-2 they used supervised learning where they trained on data recorded from human drivers. The training was conducted during a 14 day period where they registered data for a few hours per day. It is evident that data acquisition is a tedious process where many hours of human resources are needed. When developing in the framework of Machine Learning, the bene2 1. Introduction fits from simulation environments are tremendous since the data can be generated instantly, instead of recording data in real life. Recently the project (CAD)2RL showed promising results, where they trained collision avoidance using a discrete RL-algorithm called Q-learning to a quadrotor. Remarkably they only used synthetic data for training and deployed the agent in the real world without any fine tuning on the parameters of the network with successful results . One of the most popular simulation plat-

forms for racing is Torcs [16], which has been used to deploy numerous autonomous agents. Examples of algorithms utilized in the environment are Monte Carlo tree search, evolutionary algorithms and Q-learning. The Monte Carlo project used a Tree Search algorithm and forward motion model to explore randomly in the action space to maximize an objective function designed for racing. To use their forward motion model, they transformed the sensor inputs to Euclidean space. Similar to the Monte Carlo project Loiacono (et al.), handcrafted the feature-vectors from the state-space and controlled the car with high-level navigation inputs. Nevertheless, they train a neural network to outperform the currently best AI-methods for overtaking in Torcs. An evolutionary algorithm has expanded the state-space with images and utilized the Fourier transform to refine the images and train a Neural Network to drive autonomously. Apart from Torcs, the CARMA project was inspired by DeepMinds [2] progress with the Deep Q Network (DQN) algorithm in the ATARI environment. They scaled up the project in the Vdrift [20] environment and discretized the action space to fit the DQN algorithm. Using a handcrafted and simplistic reward function together with both sensory inputs and images they could obtain results where they outperformed the handcrafted controller in three criteria, namely, average reward, average speed, and top speed.

## III. INTRODUCTION

### A. What is Reinforcement Learning?

Reinforcement learning is a type of machine learning where an agent learns how to conduct actions and see results in an environment. The concept is to measure the fitness function by either earning rewards or receiving negative reward (penalty). There is no response key in reinforcement learning (RL), but the reinforcement learning agent still has to determine how to behave to accomplish his mission. The agent is learning from experience in the absence of current training data. It collects the examples of learning (quot;this action is suitable, the other one would is not suitablequot;) by trial-and-error as it attempts its mission, with the goal of optimizing long-term reward. Henceforth, Reinforcement Learning is called semi-supervised Learning Model. It is usually called as Markov Decision Process (MDP).

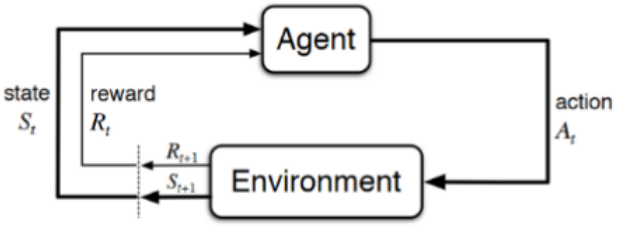



Fig 1: The agent-environment interaction in a MDP

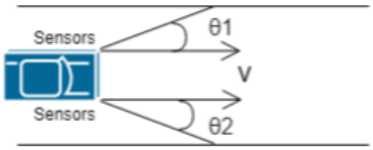
Markov decision process (MDP) composes of:
(S, A, r, P 0, Y)

An MDP is defined by: Set of states S, Set of Actions A, Transition function P(S — S, a),

Reward function R(S, a, S), Start state S 0,

Action (A): Actions taken by agent State (S): current state the environment. Reward (R): An indication that the last activity by the agent is suitable for the environment in that particular track. This reward determines the next step of the action indirectly.

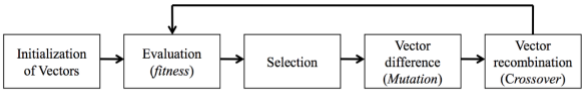
For further implementation of this project as hardware, in robotic controls we use sensors to measure the join-angles, velocity and the end-effector pose which is demonstrated in fig 2.




However, for this minor project we have implemented only the software edition i.e model in simulation. An action can be a move of the car in a particular direction at a particular direction. For a Self Driving cars, the reward is very sparse: 1 if we complete the nearest goal (short term goal or frame that is assigned randomly to the randomly generated paths. This shall be explained in the later sections.) or -10000 if we collide. Now, well look at the algorithm details that we have implemented i.e. Differential Evolutionary algorithm.

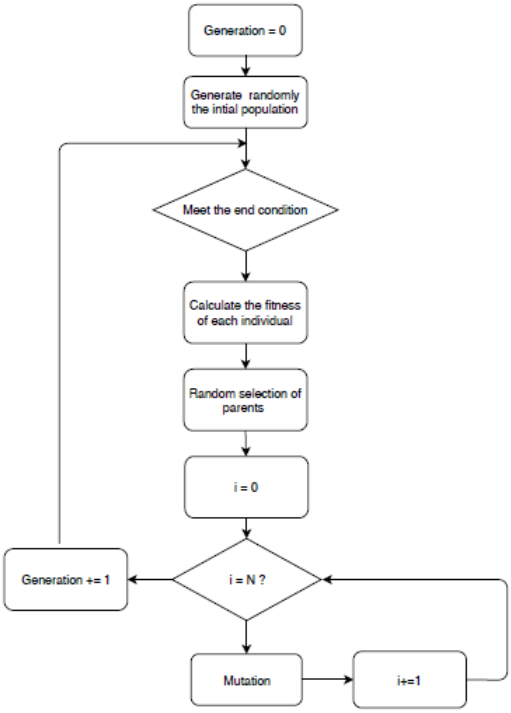
### B. Differential Evolutionary Model

Differential Evolution (DE) is a meta-heuristic algorithm based on vectors that is somewhat similar to pattern search and genetic algorithms because of its convergence and mutation application. In addition, with explicit updating equations, DE can be regarded as a further advancement of genetic algorithms, allowing for some theoretical analysis. DE is a stochastic, self-organizing search algorithm and does not use derivative data. It is therefore a population-based approach that is derivative-free. DE also uses real numbers as solution strings, so there is no need for encoding and decoding.




Similar to genetic algorithms, layout parameters are represented as vectors in a d-dimensional search space, and different genetic operators work over their string bits. Unlike genetic algorithms, however, differential evolution performs operations across each element (or solution dimension). In terms of vectors, almost everything is finished. For example, mutation is performed at one site or several sites of a chromosome in genetic algorithms, while a differential vector of two randomly selected population vectors is used to interrupt an existing vector in differential evolution. These vectorized mutation can be seen from the point of view of implementation as a more effective solution. Such form of disturbance occurs over each vector of the population and can therefore be expected to be more effective. Similarly, the crossover is also an exchange of chromosomes or vector parts dependent on vectors,

component-wise. DE has explicit updating formulas in addition to using mutation and crossover as differential operators. This also makes the implementation and design of new variants straightforward. DE optimizes a problem by maintaining a population of candidate solutions and creating new candidate solutions by combining existing ones according to its simple formulae, and then keeping whichever candidate solution has the best score or fitness on the optimization problem at hand. It is a stochastic, population-based optimization algorithm for solving a nonlinear optimization problem There are three DEs control parameters: (1) the population size NP (2) the mutation factor F (a real-value factor that controls amplification of differential variations) and (3) the crossover factor CR (also a real value, controlling the crossover operation).




Consider an optimization problem Minimize problem f(x) Where X = [x1, x2, , xD], D is the number of variables. This is a population-based algorithm. Consider a population size of The population matrix can be shown as Tn, i = [ Tn,1, Tn,2, Tn,3, ., Tn,x] Where, g is generation and n = 1,2,3.x

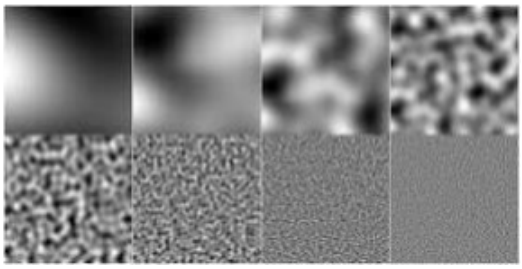
## IV. PRESENTED APPROACH

### A. Computer Architechture

Experiments were run on ASUS ROG Strix G G731GT FHD Gaming Laptop GTX 1650 with 4 GB Graphics, Core i7-9750H 9th Generation, 16 GB RAM with windows 10. The main reason of using tensorflow.js is that it can utilize WebGL to run on an underlying GPU whenever its available.

### B. Steps followed

**Step 0**: Create an environment Road tracks are created by using p5.js library. The tracks are modified every generation by using Perlin Noise in the tracks.



Hence, the agent with maximum fitness function value (green car) does not get acquainted with one type of track considering the real environment.

**Step 1**: Randomly initialize the population The initial population is generated randomly between the upper lower and upper bound.

**Step 2**: Calculate fitness function of each agent Calculating fitness through Euclidian distance would be difficult in conditions where the starting points and the ending points are same as the the fitness function f(x) = 1/(Euclidian distance) would lead to infinity score. Hence, to overcome this we introduced the concept of short term goals. After the random generation of path, the path would be divided into n frames and the goal of the agent would be to reach the nearest goal in order to earn rewards. The path shown in Fig 7 is one example of the frames (short term goal of agents) for a randomly generated path. Other than this, if the car collides with the wall then, a negative reward will be awarded. The agent with current maximum fitness will be highlighted in green colour for better visualization.

**Step 3**: Parent Selection This is the first step which aims at selecting individuals for individuals for reproduction. In order to choose the best and the fittest individuals, selection is done on the basis of the fitness of each individual participating in selection.

If we choose the best fitness parent in every case then it would overfit the model and might underestimate the potential of the second best agent in a few cases. This algorithm chooses parent randomly from the top agents to perform better than above-mentioned.

**Step 4** Mutation of the best agent re-assignment of weights/parameters The mutation makes use of random Guassian Generation for generating a smooth error function For the re-assignment of the parameters, we have used feedforward model using ANN sequential model. We used sigmoid activation function owing to its promising results.

**Step 5**: Create Next Generation The below image is an snapshot from the implementation

Why DE over GA for this model?

The limiting factor of a GA search run is in most cases the numbers of fitness evaluation. Fitness evaluations consume a lot of time in real-world application and might even involve testing by an expert. Thus, the goal of every GA should be to get the best results with regard to a limited number of fitness evaluations. The key for an efficient search is the Mating Pool New Population Two best individuals balance between exploration and exploitation. In the beginning of the search premature convergence should be avoided before

having covered as much of the search space as possible. In this phase exploration must be enforced while at the end of the search process it is favorably to make the most out of the already found best solutions. So exploitation is the better choice. In other words, the degree of exploitation should be monotonically decreasing and the degree of exploration should be monotonically increasing during the search run, respectively. Moreover, in our case crossover is not possible which in total leaded us to Differential Evolution.

Other than this, if the car collides with the wall then, a negative reward of - 1000 will be awarded. The agent with current maximum fitness will be highlighted in green colour for better visualization.

## ACKNOWLEDGMENT

We would like to express our deepest appreciation to all those who provided us the possibility to complete this report. We acknowledge with thanks, the support rendered by Prof. Preksha Pareek under whose aegis we were able to complete the task in a given period. We also appreciate the constructive suggestions given Dr Ankit Thakkar for enhancement of content of the report. At the home front, we are extremely grateful to our family members for the support and encouragement we got from them in successfully completing the report.

## REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, Playing atari with deep reinforcement learning, CoRR, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, Mastering the game of go with deep neural networks and tree search, Nature, vol. 529, no. 7587, pp. 484489, 2016. [Online]. Available: http://dx.doi.org/10.1038/nature16961

[3] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, Trust region policy optimization, CoRR, vol. abs/1502.05477, 2015. [Online]. Available: http://arxiv.org/abs/1502.05477

[4] 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, CoRR, vol. abs/1509.02971, 2015. [Online]. Available: http://arxiv.org/abs/1509.02971

[5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.