# *Visualization of Autonomous Car: Evolutionary Machine Learning Model*

**By**

**Rutva Patel**
**16bit129**

**Nisarg Vaghela**
**16bit085**


**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**
**Ahmedabad 382481**

# *Visualization of Autonomous Car: Evolutionary Machine Learning Model*

**Minor Project**

Submitted in fulfillment of the requirements

For the degree of

**Bachelor of Technology in Information Technology**


By

**Rutva Patel**
**16bit129**

**Nisarg Vaghela**
**16bit085**

Guided By
**Prof. Preksha Pareek**
**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**


# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
# Ahmedabad 382481

## CERTIFICATE

This is to certify that the Minor Project entitled "Visualization of Autonomous Car: Evolutionary Machine Learning Model" submitted by Rutva Patel (16BIT129) and Nisarg Vaghela (16BIT085), towards the partial fulfillment of the requirements for the degree of Bachelor of Technology in Information Technology of Nirma University is the record of work carried out by them under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination.


Preksha Pareek
Assistant Professor
Department of Computer Science & Engg.,
Institute of Technology,
Nirma University,
Ahmedabad

Dr. Madhuri Bhavsar
Dept. of Computer Science & Engineering,
Institute of Technology,
Nirma University,
Ahmedabad

# ACKNOWLEDGEMENT

We would like to express our deepest appreciation to all those who provided me 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.

# ABSTRACT

Reinforcement learning is considered to be one of the strongest paradigms in AI domain, which can be applied to teach machines how to behave through environment interaction. In this paper we have used the implementation of Differential Evolutionary Model after comparisons with several other models highlighting the current achievements for autonomous driving vehicles.

# CONTENTS

# Chapter-1 Introduction

## 1.1 *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 ("this action is suitable, the other one would is not suitable") 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).

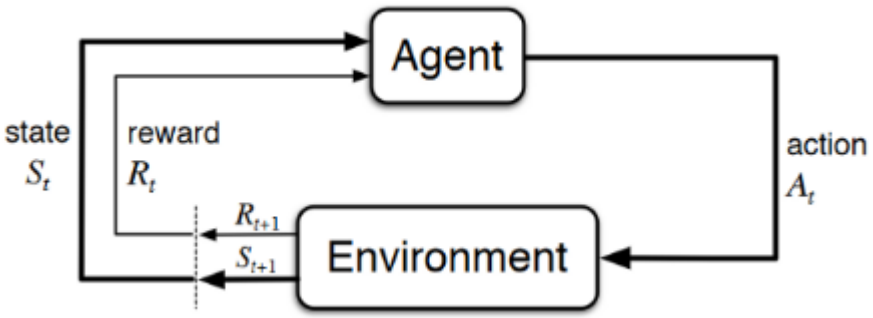


Fig 1: The agent-environment interaction in a MDP

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 $S_0$,

Discount factor Y,

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.

Further implementation:

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.

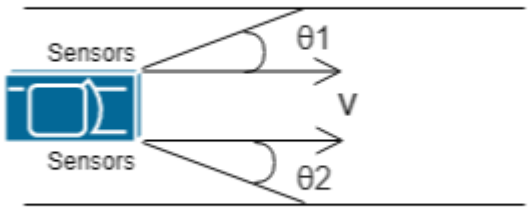


Fig 2: An illustration of hardware implementation

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, we'll look at the algorithm details that we have implemented i.e. Differential Evolutionary algorithm.

## 1.2 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.

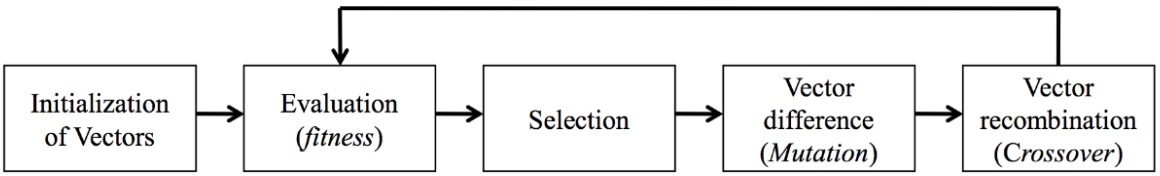


Fig 3: Differential Evolutionary model steps

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 DE's 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).

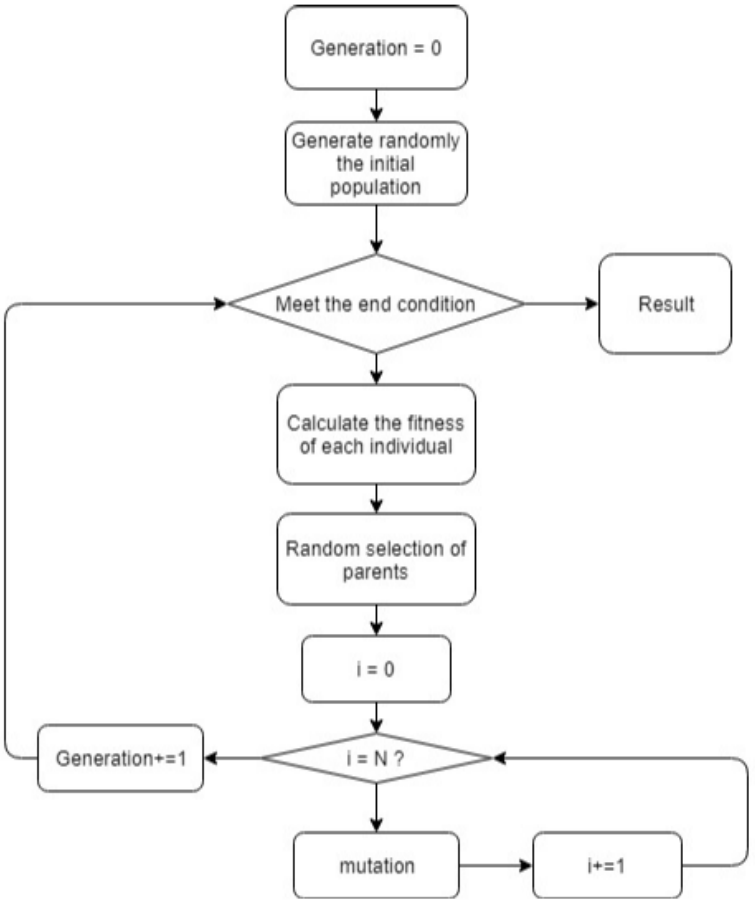


Fig 4: Differential Evolutionary Model Flowchart

APPROACH 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.

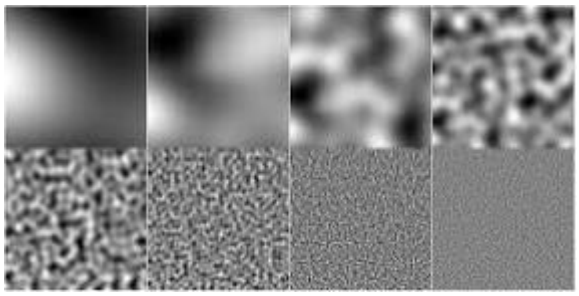

Fig 5: Perlin Noise

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 by using a random function.

### ***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.

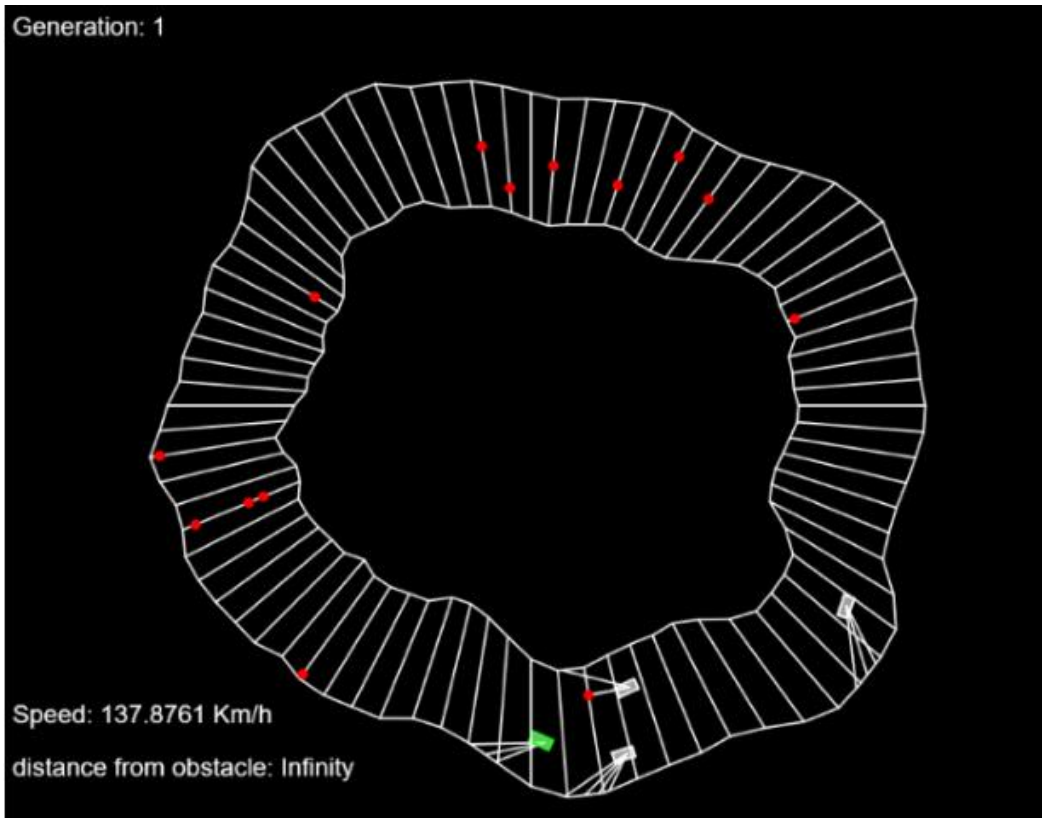


Fig 6: Frames (short term goals) for each agent

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.

The fitness is calculated by the following algorithm:

```
check(checkpoints) {
  if (!this.finished) {
    this.goal = checkpoints[this.index];
    const d = pldistance(this.goal.a, this.goal.b, this.pos.x, this.pos.y);
    if (d < 5) {
      this.index = (this.index + 1) % checkpoints.length;
      this.fitness++;
      this.counter = 0;
    }
  }
}

calculateFitness() {
  this.fitness = pow(2, this.fitness);
}
```

### ***Step 4*: Parent Selection**

This aims at selecting the agent that would perform best in a similar environment and has achieved maximum fitness value.

We cannot choose the parent with best fitness function because otherwise it would overfit the tracks. That is one of the reason we chose Differential Evolution over GA.

The algorithm that we have implemented for the parent selection is as follows:

```
function pickOne() {
  let index = 0;
  let r = random(1);
  while (r > 0) {
    r = r - savedagents[index].fitness;
    index++;
  }
  index--;
  let particle = savedagents[index];
  let child = new Particle(particle.brain);
  child.mutate();
  return child;
}
```

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 5: Mutation of the best agent & re-assignment of weights/parameters***

The mutation makes use of random Guassian Generation for generating a smooth error function

The algorithm used in the model is pictured below:

```
mutate(rate) {
  tf.tidy(() => {
    const weights = this.model.getWeights();
    const mutatedWeights = [];
    for (let i = 0; i < weights.length; i++) {
      let tensor = weights[i];
      let shape = weights[i].shape;
      let values = tensor.dataSync().slice();
      for (let j = 0; j < values.length; j++) {
        if (random(1) < rate) {
          let w = values[j];
          values[j] = w + randomGaussian();
        }
      }
      let newTensor = tf.tensor(values, shape);
      mutatedWeights[i] = newTensor;
    }
    this.model.setWeights(mutatedWeights);
  });
}
```

For the re-assignment of the parameters, we have used feedforward model using ANN sequential model. We used sigmoid activation function owing to it’s promising results.

```
  createModel() {
    const model = tf.sequential();
    const hidden = tf.layers.dense({
      units: this.hidden_nodes,
      inputShape: [this.input_nodes],
      activation: 'sigmoid'
    });
    model.add(hidden);
    const output = tf.layers.dense({
      units: this.output_nodes,
      activation: 'sigmoid'
    });
    model.add(output);
    return model;
  }
}
```

***Step 5: Create Next Generation***

The below image is an snapshot from the implementation

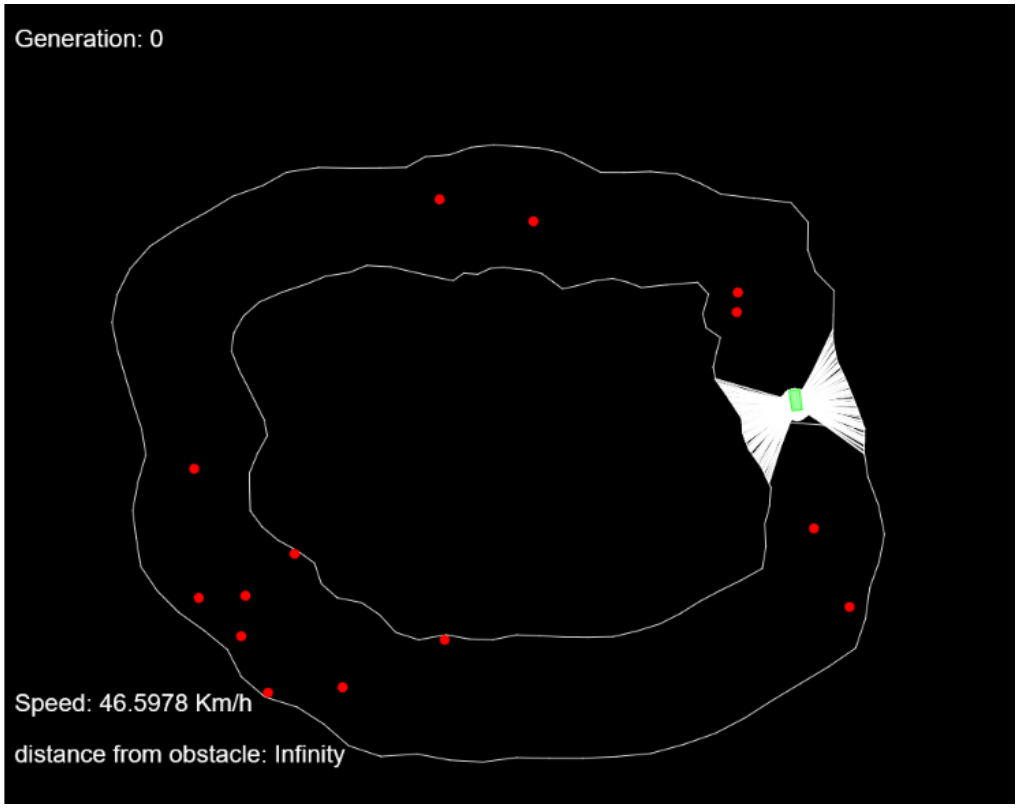


Fig 7: Autonomous cars in simulator

## Conclusion and Observation:

From this project, we observed various reasons after experimenting as such why Differential Evolutionary algorithm is more suitable for Self Driving Cars compared to Genetic Algorithm with the above mentioned reasons. We also observed that by changing track every generation by adding noise we can train the agents with better results.

We conclude that after more rigorous training of the cars, we can implement the sensors around the car to demonstrate the real world application.

## REFERENCES


1. Deep Reinforcement Learning For Autonomous Vehicles-State Of The Art Liviu Marina, A sandu
2. CARMA: A Deep Reinforcement Learning Approach to Autonomous Driving Matt Vitelli, Aran Nayebi
3. kdnuggets.com/
4. A Reinforcement Learning Based Approach for Automated Lane Change Maneuvers
5. Medium.com