Report of Evolutionary Algorithm Course

Project title: Auto driving Car

Name (student number): Md Tauhidul Islam (2016521460553)
Other group members:
Sk Asiful Huq (2016521460554)
Sirajus Salehin(2016521460551)

Team Leader: Md Tauhidul Islam

Project Supervisor: Professor Haixian, Zhang

Grading Results: _____

Comments: _____

_____

_____

Time for Submitting the Report: 2019/12/21

Signature:

AUTO DRIVING CAR

Abstract:

On a daily basis, people worldwide spend a big portion of their day moving from one place to another. Shouldn't it be easier and safer to go wherever we want to go? Auto driving car can be a possible solution for this.

This report shows how evolutionary computation techniques can help in this field. where we will demonstrate the training method of multilayer neural network using genetic algorithm with its respective crossover and mutation functions. As the Darwin's law says the entity (car) with the best genes will survive and it will propagate the genes to the next generation. They will evolve in a better way and survive more time.

We also tried to show a possible training method with back-propagation for our project, as we have it as our course topic.

This is possible primarily because human driver has vision with sense to make a decision on which way to turn the steering wheel and when to step on the brake. In our project, we used the sensors only to navigate and detect the obstacles in certain distance in its environment. So, it can't detect the traffic signals and road navigation yet. But it can solve one major aspect of the problem.

Key-words:

Neural network, Genetic Algorithm (initialization, selection, crossover, mutation), terrain (car track), lasers, camera and controller.

Introduction:

We designed an auto driving car simulation using neural network and genetic algorithm. The project is developed in Unity 3D using C# as programming language. we will explain how we approach towards our solution using neural network and respective algorithm in our environment.

We used unity 3d game engine and unity free assets to create our experiment environment.

The project can be divided in different aspects. Here, I will explain the following aspects.

- Neural Network: we will be using a multilayer neural network.
- Genetic Algorithm with its respective crossover and mutation functions.
- Visual simulator in 3D with Unity: create car track.
- Lasers: input data of the neural network.
- Car movement: output actions of the network.
- Camera: viewing the simulation in an intelligent way.
- Controller: handles all the executions and the algorithms.

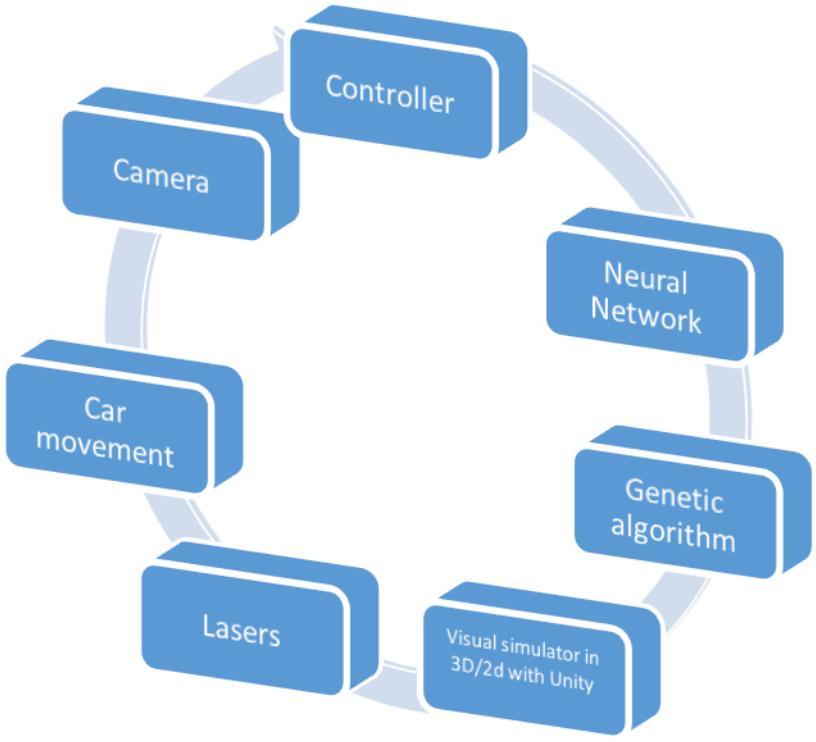


Figure: 1

Project Background and rough idea for understanding:

Project environment:

1. Unity 3d game engine

   - Creating game track in 3d with obstacles like real life race track.
   - Test environment for simulation
   - Creating Game agent: cars

2. Visual studio

   - Writing C# code for controlling game agent.
   - Creating neural network architecture

Game track in unity 3d:

The track of the simulator is made in Blender with Bezier curves. In Unity 3D is a mesh that has a collider component.

Our game agent cars should learn to drive through the track without touching the obstacles. Our primary track is simple as it is our learning project in this field. Here I will show how we are going to implement our vision with neural network and genetic algorithm for our auto driving car in the track.

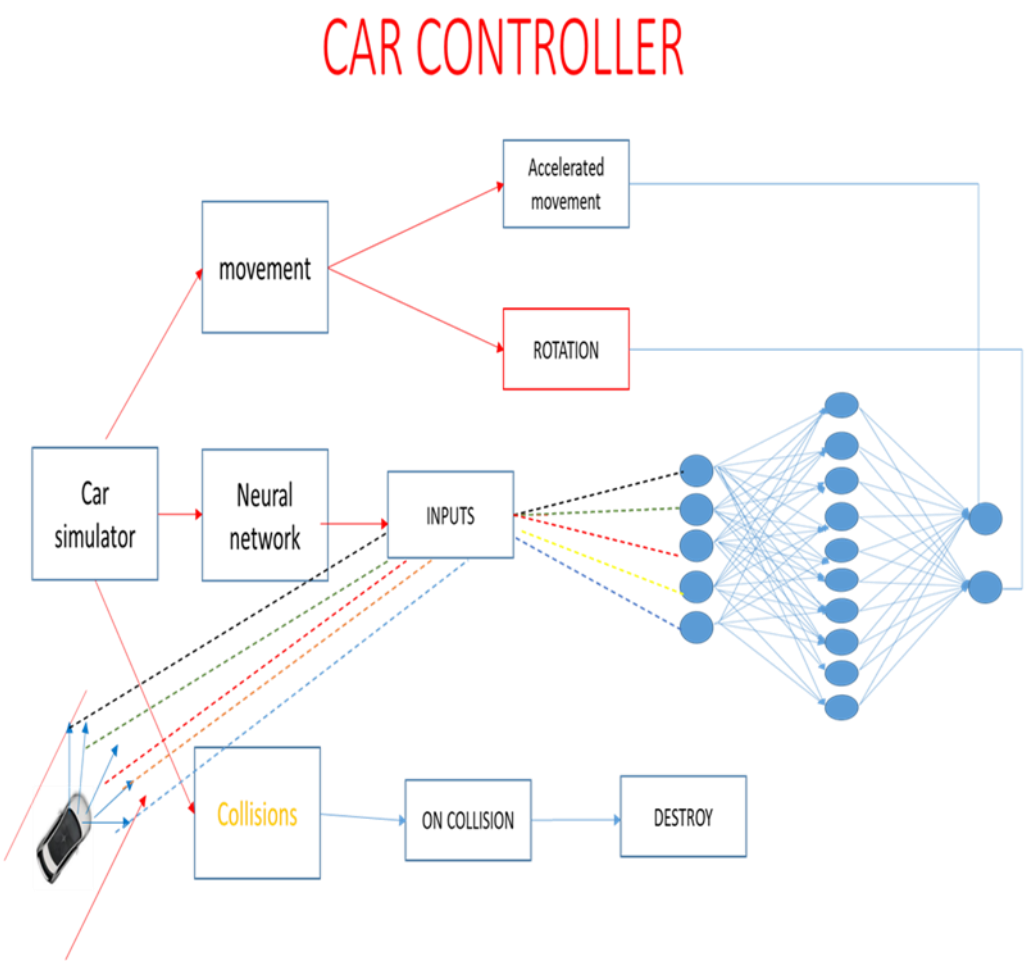
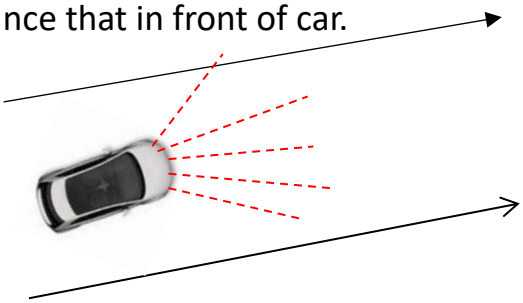


Figure: 3

simple car controlling system architecture for our project entity car.

Cars property:

As we showed in figure 3 each car will have 5 sensors that we defined as lasers in our implementation. That will be the equivalent of the real LiDAR sensors. We assigned the lasers in the front side of the car to get the obstacles distance that in front of car.



The output of the sensor will be the max distance of it divided by the collision point distance from the car. The output will be a value from [0-1]. If the sensor hasn't detected any object it will return 1.

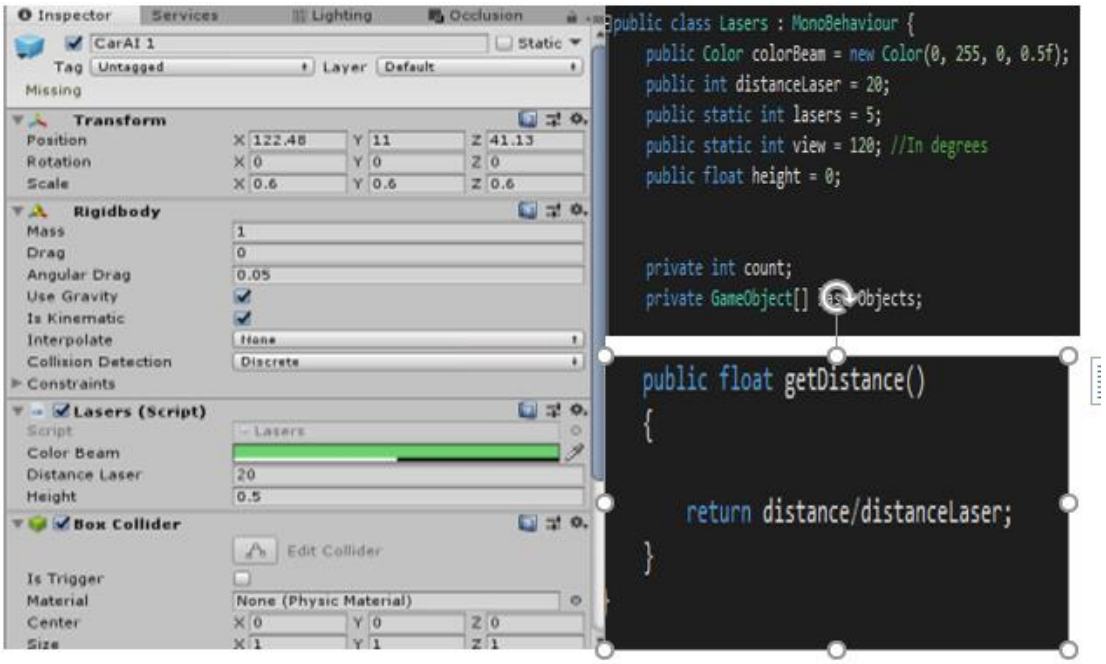


Figure: 4

Neural network:

Neural network is a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs.

There are several types of neural network. Such as feedforward neural network, convolutional network, recurrent neural network, modular neural network etc.

As we will train the car by function evaluation through different layer, we chose multilayer neural network called feedforward neural network for our project.

Here, we will explain how we used feedforward in our project.

Before we start to explain the implementation of feedforward in our project, we would show the basic structure of feedforward neural network. In a general architecture of neural network has following component:

1. Input layer
2. Hidden layer
3. Output layer

Those layers component is designed with biological components called neurons and weights.

In our project each car will have its own neural network.

The neural network has one input layer with five neurons, one hidden layer with 10 neurons and one output layer with 2 neurons.

In total there are 17 neurons and 70 synapsis among those neuros with 70 weights value.

Architecture of Neural Network:

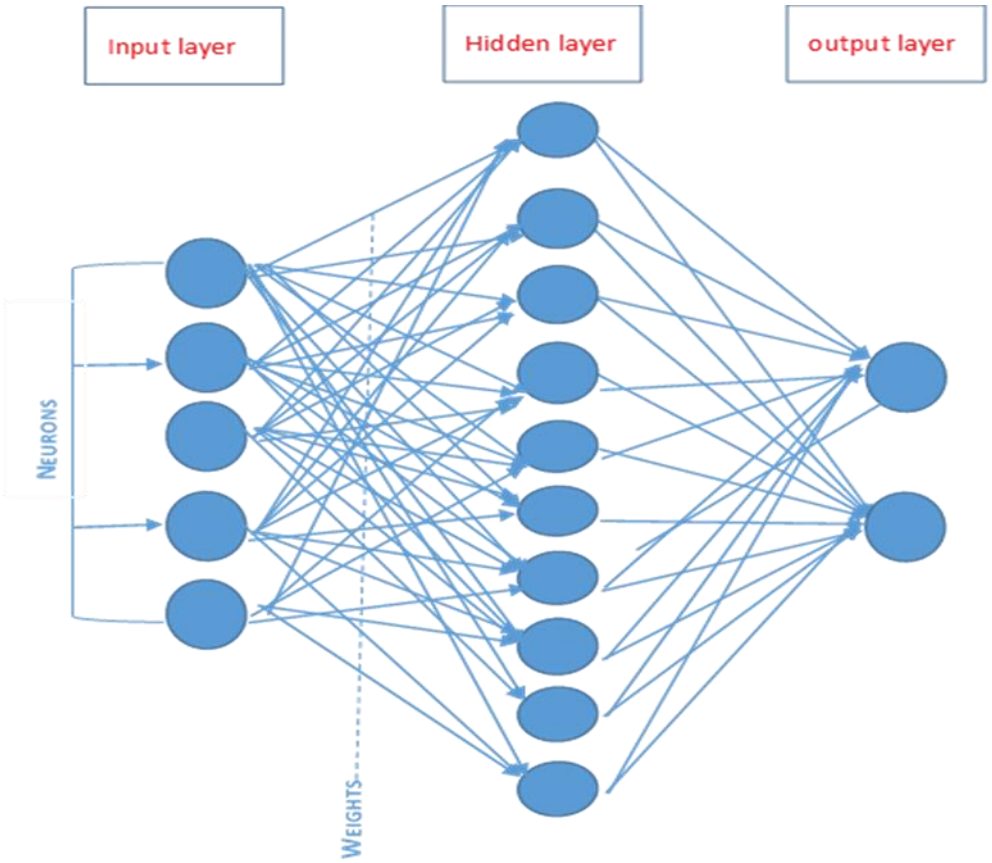
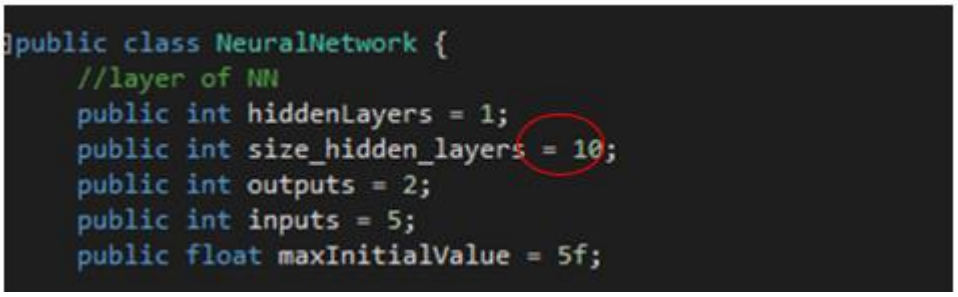


Figure: 5

There is no bias node here because the system doesn't use back propagation to train the model. Instead of using back propagation, we used genetic algorithm with respective crossover and mutation function to train the model.

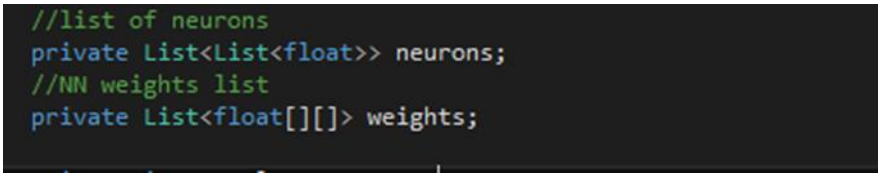
Each car will have this architecture as neural network. In input layer we have five neurons for five lasers distance as input. Each sensor will have one neuron value. The output will be always 2 because of the acceleration and rotation of each car. The hidden layers will vary and will be able to be customized by the user.

The architecture of hidden layer can be edited in this segment:



```
public class NeuralNetwork {
    //layer of NN
    public int hiddenLayers = 1;
    public int size_hidden_layers = 10;
    public int outputs = 2;
    public int inputs = 5;
    public float maxInitialValue = 5f;
```

The values of the weights and neurons will be stored in different array:



```
//list of neurons
private List<List<float>> neurons;
//NN weights list
private List<float[][]> weights;
```

The Neural Network class will have the feed-forward update method that will be charge of calculating the outputs with the input data and the weights values.

This is the feed-forward algorithm implemented in code:

```
//feedforward update function
    public void feedForward(float[]inputs)
    {

        //Set inputs in input layer
        List<float> inputLayer = neurons[0];
        for(int i = 0; i < inputs.Length; i++)
        {
            inputLayer[i] = inputs[i];
```

```
            }
            //Update neurons from the input Layer to the output Layer
            for(int layer =0;layer< neurons.Count-1; layer++) {

                float[][] weightsLayer = weights[layer];//current layer
                int nextLayer = layer + 1;//next layer
                List<float> neuronsLayer = neurons[layer];//neurons in current layer
                List<float> neuronsNextLayer = neurons[nextLayer];//neurons in next layer
                for(int i = 0; i < neuronsNextLayer.Count; i++) //Next layer
                {
                    float sum = 0;
                    for(int j = 0; j < neuronsLayer.Count; j++)
                    {
                        //this is to calculate the multiplication of neuron & weight
                        sum += weightsLayer[j][i] * neuronsLayer[j];/*Feed-forward
multiplication*/
                    }
                    //assign neighors next layer in position i with activation function
                    neuronsNextLayer[i] = sigmoid(sum);

                }
            }

}
```

To do the feedforward, we need to initialize the weights to the each layer synapsis. In our project the car will learn to drive geneation by generation using Darwin's law of genetic. In first generation it will initialize random weights between [-maxInitialValue,maxInitialValue]. In our implementation we defined a neural network constructor in c# to assign weights to each layer.

```
//NN constructor with randome initialization
    public NeuralNetwork()
    {
        totalLayers = hiddenLayers + 2;// hidden layers + inputslayer+outputlayer
        //Initialize weights and the neurons array
        weights = new List<float[][]>();
        neurons = new List<List<float>>();
```

```
        //Fill neurons and weights totallayers=3
        for(int i = 0; i < totalLayers; i++)
        {
            float[][] layerWeights;
            List<float> layer = new List<float>();
            int sizeLayer = getSizeLayer(i);//0
            if (i != 1 + hiddenLayers)
            {
                layerWeights = new float[sizeLayer][];//layer weight=[0][];
                int nextSizeLayer = getSizeLayer(i + 1);//set synapsis connection between
previous layer output and next layer neuron 0+1=1
                for(int j = 0; j < sizeLayer; j++)//current layer sizelayer=1
                {
                    layerWeights[j] = new float[nextSizeLayer];
                    for(int k = 0; k < nextSizeLayer; k++)// next layer
                    {
                        layerWeights[j][k] = genRandomValue();
                    }
                }
                weights.Add(layerWeights);//for adding a new layer weight
            }
            for(int j = 0; j < sizeLayer; j++)
            {
                layer.Add(0);
            }
            neurons.Add(layer);//add neuron to layer
        }

    }
```

Here, we initialized all the weights for first generation. Now, we can do the feedforward for the first generation.

In our system obstacle distance is the input that we are going to feedforward. In figure 4 section car property we explained it.

Here, we used ray cast of lasers to get obstacles distance and return the normalize distance as input.

Code segment for laser input:

Code segment:

```
// Update is called once per frame
void Update () {
        Vector3 finalPoint = transform.position + transform.forward * distanceLaser;
        RaycastHit collisionPoint;
        if(Physics.Raycast(transform.position,transform.forward,out collisionPoint,
distanceLaser))
        {
            lineRenderer.SetPosition(0, transform.position);
            lineRenderer.SetPosition(1, collisionPoint.point);
            distance = collisionPoint.distance;

        }
        else
        {
            lineRenderer.SetPosition(0, transform.position);
            lineRenderer.SetPosition(1, finalPoint);
            distance = distanceLaser;
        }
}
    public float getDistance()
    {
        //distance will be between [0,1]
        return distance/distanceLaser;
    }
}
```

Now we can calculate the feedforward value for our system using the distance inputs, weights and sigmoid activation function as we applied sigmoid activation in our system.

As we have a complicated neural network architecture, here I will explain the feedforward for a simple architecture with three input neurons, two hidden neurons and two output neurons to make it more understandable and visualized.

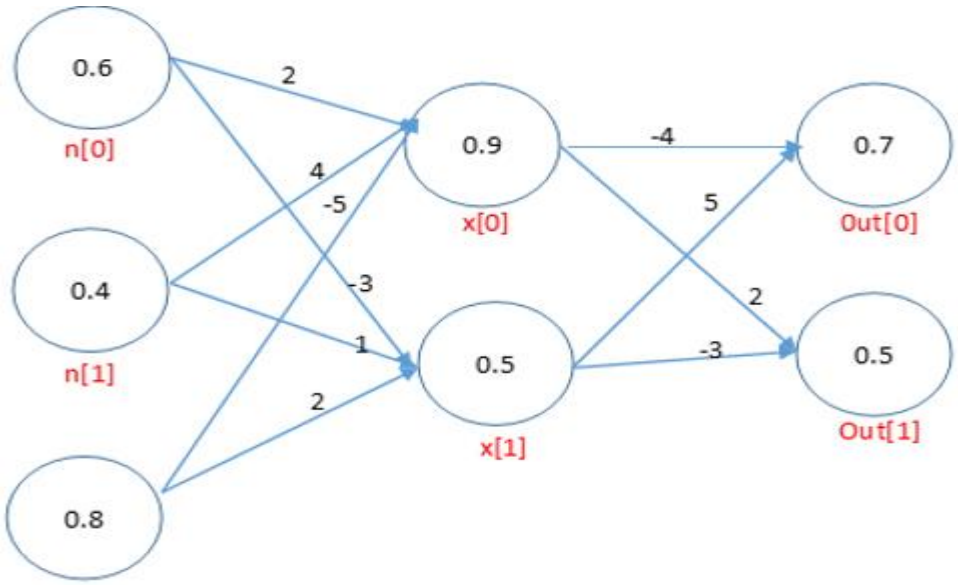


Figure:6

Here,

n [0], n [1] and n [2] are the lasers input distance.

The weights are as follows:

n[o] to x [0] =2;

n [1] to x [0] =4;

n [2] to x [0] =-5;

```
//activation function
public float sigmoid(float x)
{
    //return the net value of neural network
    return 1 / (float)(1 + Mathf.Pow(EULER_NUMBER, -x));
}
```

so according to feed-forward we can calculate,

sum=(0.6*2)+(0.4*4)+(0.8*-5)=-1.2

now we will do the activation with sigmoid activation function.

So,

X [0] =1/(1+e^(-sum))=0.9

We used this technique in our whole architecture to get output for our neural network.

Through feedforward we will get two output as output [0] and

output [1]. After optimizing these outputs, we can update the car movement with rotation and acceleration in delta time. We will get output between 0 and 1. we take 1 as input of obstacle and 0 as obstacle free output. In our implementation we defined updateMovement constructor in CarMov.cs class to update the car movement with neural network output.

```
//update the rotation and accelaration
    public void updateMovement(List<float> outputs)
    {
        //if output is higher than 0.5 car rotation will be increased
        if (outputs[0] * 2 > 1f)
        {
            vyRot = (outputs[0] * 2 - 1) * speedRotation * Time.deltaTime;
        }
        else
            //decressed
        {
            vyRot = -(outputs[0] * 2) * speedRotation * Time.deltaTime;
        }

       //if output is higher than .5 acceleration will be increased
        if (outputs[1] * 2 > 1f)
        {
            acceleration = (outputs[1] * 2 - 1) * increaseAcc;
        }
        else
            //decressed
        {
            acceleration = -outputs[1] * 2 * increaseAcc;
        }
    }
```

Here, we completed our first generation of auto driving car with random weight (DNA) initialization for each car, respective feedforward and sigmoidal activation.

But still we didn't get a good result in our experiment simulation. Because

the cars will learn through reinforcement learning (unsupervised learning).

LEARNING ALGORITHMS:

Genetic algorithm:

In our system we used the reinforcement learning to train our model with the genetic algorithm applied to every car. As the Darwin's law says the entity (car) with the best genes will survive and it will propagate the genes to the next generation. They will evolve in a better way and survive more time.

If we want to express this mathematically for cars in our experiment environment, we should then start by creating DNA for each car with its own list of genes. **There will be as much genes as the quantity of weights in the neural network of each car.** When we create the array of the weights, we will start by creating it in a random way in the first generation as we showed above.

We can divide the Genetic Algorithm in different sections:

- Random Initialization of DNAs
- Selection of best cars depending on the fitness function
- Crossover of the genes of the parents
- Mutation the new DNA

Initialization:

The initialization of the DNAs in a random way will be executed only in the first generation when the Neural Networks of the cars are created. We created the DNA as a List of weights for each car in Neural Network. In section of neural network, we explained the random weights initialization and its output for first generation.

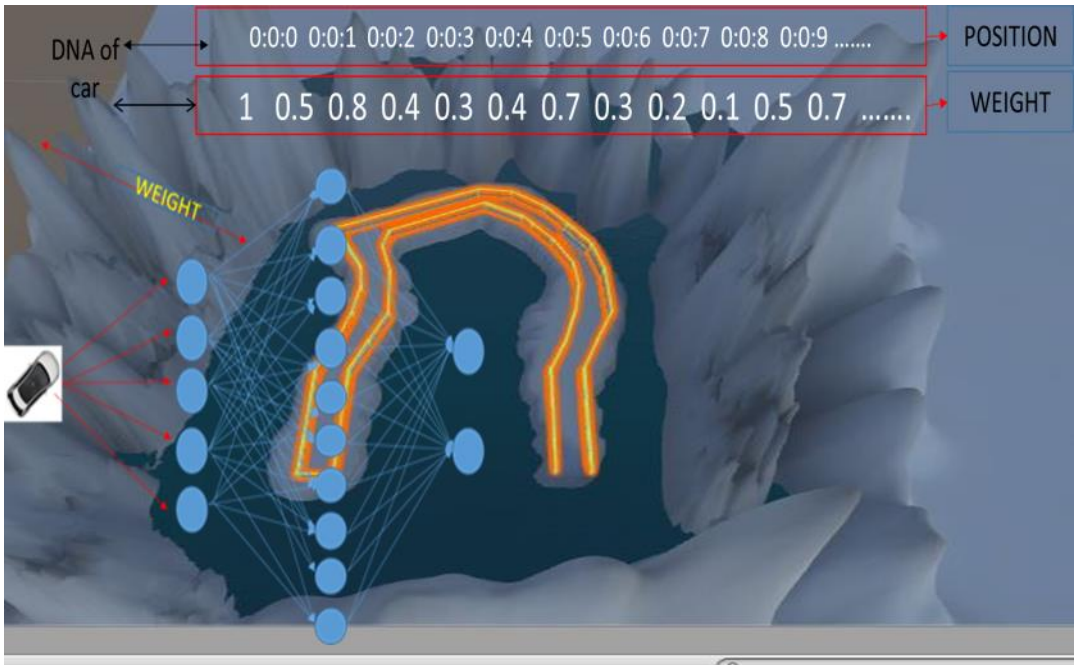


Figure:7

Selection:

When the simulator will finish the first generation of learning through colliding with environment wall, we should choose which car have the best genes. we can do this in a lot of different ways. Depending on the distance traveled, the average speed or the time driving. we chose for this project the time of the car driving, so we will get the genes for the first and second car that survived more time in the track.

All this factor could be joined to obtain a better selection of parents. It can depend also on the diversity we want to achieve.

We could create a function that depends on the factors mentioned before in order to get a specific score for each car called fitness.

Here, is the selection model that we used in our project:

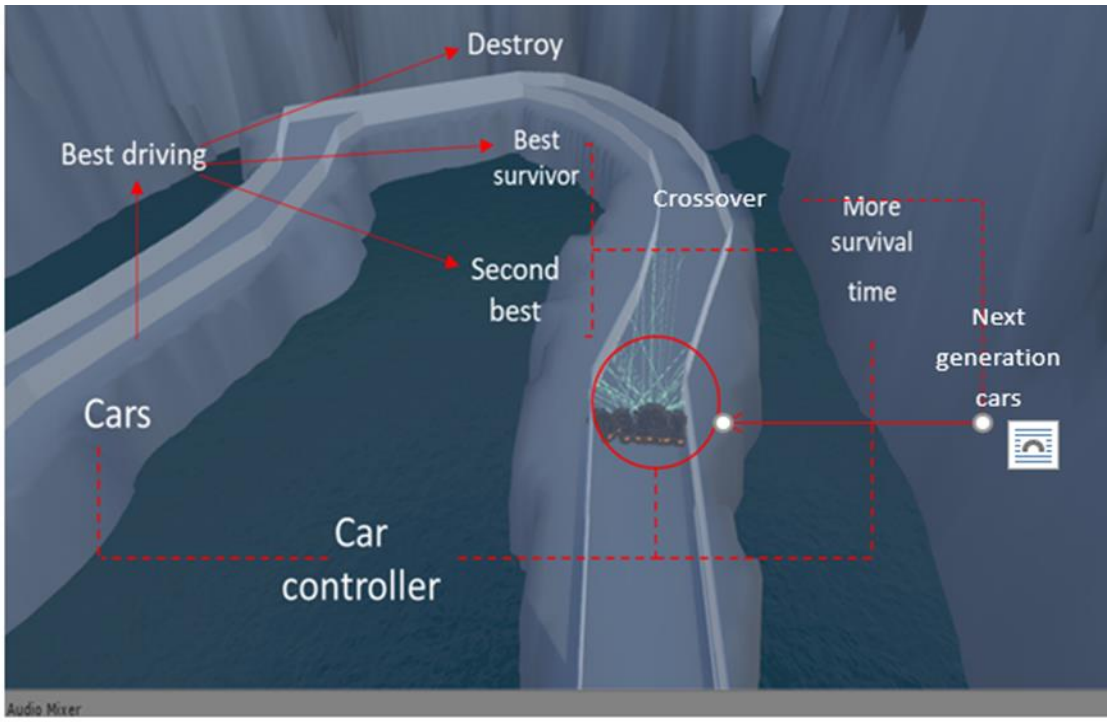


Figure:8

Here, we will find the best two cars from each generation and destroy the car object of current generation. Then we did the crossover to generate the next generation of population.

Code segment to select best two cars:

```
    public void changeCamera()
    {
        CarControllerAI controller = GameObject.Find("CarController").GetComponent<CarControllerAI>();
        List<GameObject> cars = controller.getCars();
```

```
if (cars.Count == 2)
{
    controller.winner = cars[0].GetComponent<Car>().getDNA();
    controller.secWinner = cars[1].GetComponent<Car>().getDNA();
}
if (cars.Count==1)
{
    if (!controller.winner.Equals(cars[0].GetComponent<Car>().getDNA())){
        DNA inter = controller.secWinner;
        controller.secWinner = controller.winner;
        controller.winner = inter;
    }
    cars.Remove(gameObject);
    controller.newPopulation(true);
    Destroy(gameObject);
}
```

Crossover:

With the parents selected we will swap the genes to create a new DNA with a mix of both parents preferring random fragments from one parent and the other. This random value could be modified depending if the parents have very different fitness or very similar. As they have similar fitness score, they will have more similar probability to be chosen the genes.

Here, we will show in figure 10 how we applied crossover function to propagate the next generation. In our explanation we showed with 3 input nodes, 2 hidden nodes and 2 output nodes to make it more visualized. In our project we have more nodes but we implemented with same techniques.

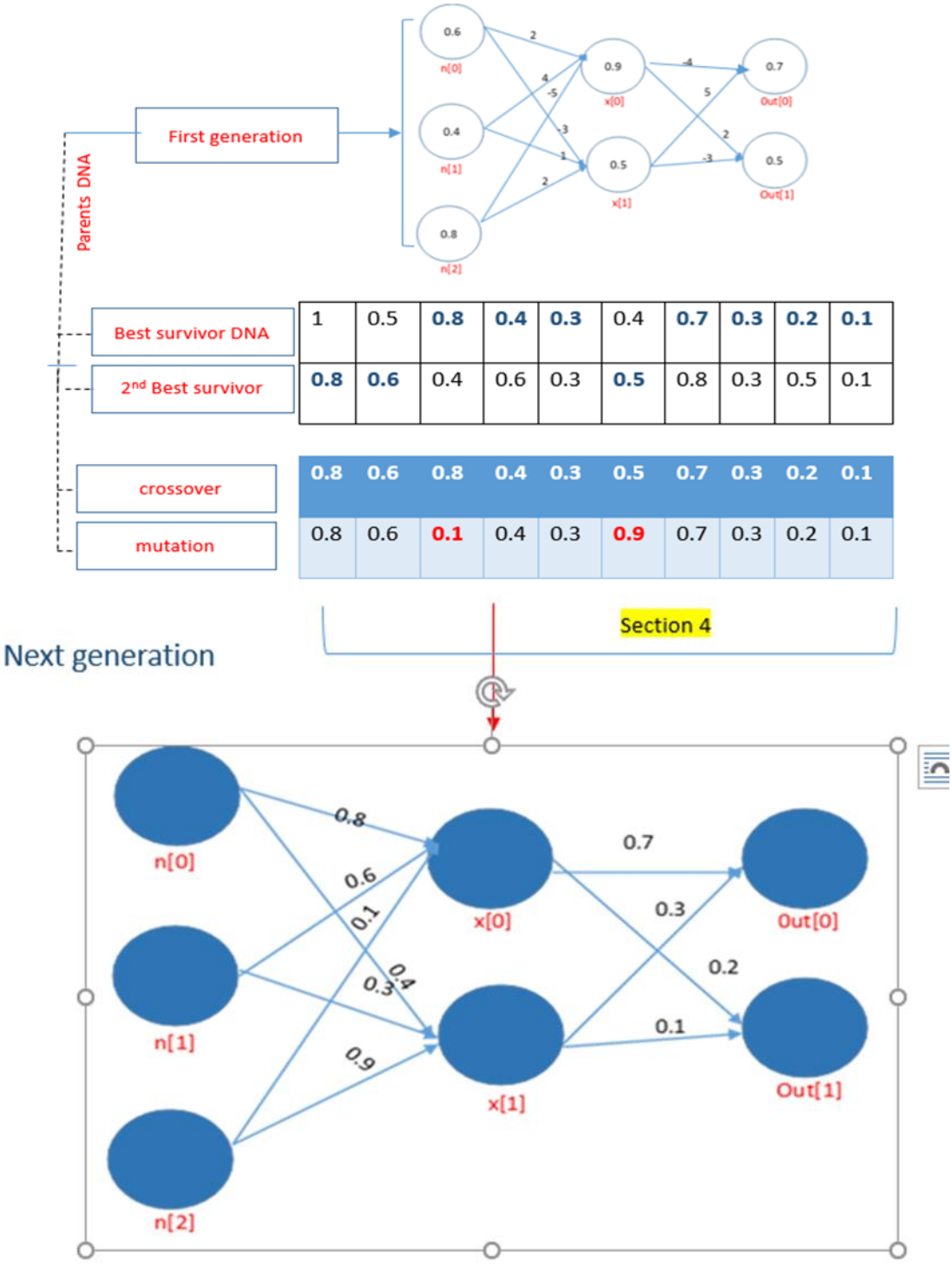

First generation
Parents DNA
0.6
n[0]
0.4
n[1]
0.8
n[2]
0.9
x[0]
0.5
x[1]
0.7
Out[0]
0.5
Out[1]
Best survivor DNA
2nd Best survivor
1 0.5 0.8 0.4 0.3 0.4 0.7 0.3 0.2 0.1
0.8 0.6 0.4 0.6 0.3 0.5 0.8 0.3 0.5 0.1
crossover
mutation
0.8 0.6 0.8 0.4 0.3 0.5 0.7 0.3 0.2 0.1
0.8 0.6 0.1 0.4 0.3 0.9 0.7 0.3 0.2 0.1
Section 4
Next generation
n[0]
n[1]
n[2]
x[0]
x[1]
Out[0]
Out[1]
0.8
0.6
0.1
0.4
0.3
0.9
0.7
0.3
0.2
0.1

Figure:9

Mutation:

When we create for each new car for the new generation the mixture between both parents, we will create different mutations (section 4: figure 9). This is used because the parents are the best from the generation but not the best driving cars. We must create small mutations in the DNA of each new car to get better cars with some diversity from last generation best to new generation.

Code segment to create new population with genetic function:

```
public void newPopulation(bool geneticManipulation)
    {
        if (geneticManipulation)
        {
            Debug.Log("aa");
            cars = new List<GameObject>();
            for(int i = 0; i < population; i++)
            {
                DNA dna = winner.crossover(secWinner);
                DNA mutated = dna.mutate();
                GameObject carObj = Instantiate(car);
                cars.Add(carObj);
                carObj.GetComponent<Car>().Initialize(mutated);
            }
        }
        generation++;
        carsCreated = 0;
        GameObject.Find("Camera").GetComponent<CameraMovement>().Follow(cars[0]);
    }
    public void restartGeneration()
    {
        cars.Clear();
        newPopulation();
    }
}
```

So basically, we implemented a neural network for each car. Then we tried to train our neural network with genetic algorithm. For that purpose, we defined our car as an individual DNA and find out best car from each generation. Then we applied the crossover and mutation to propagate next generation car and the cars learn generation by generation. We defined a CarController script to perform all the process we applied here.

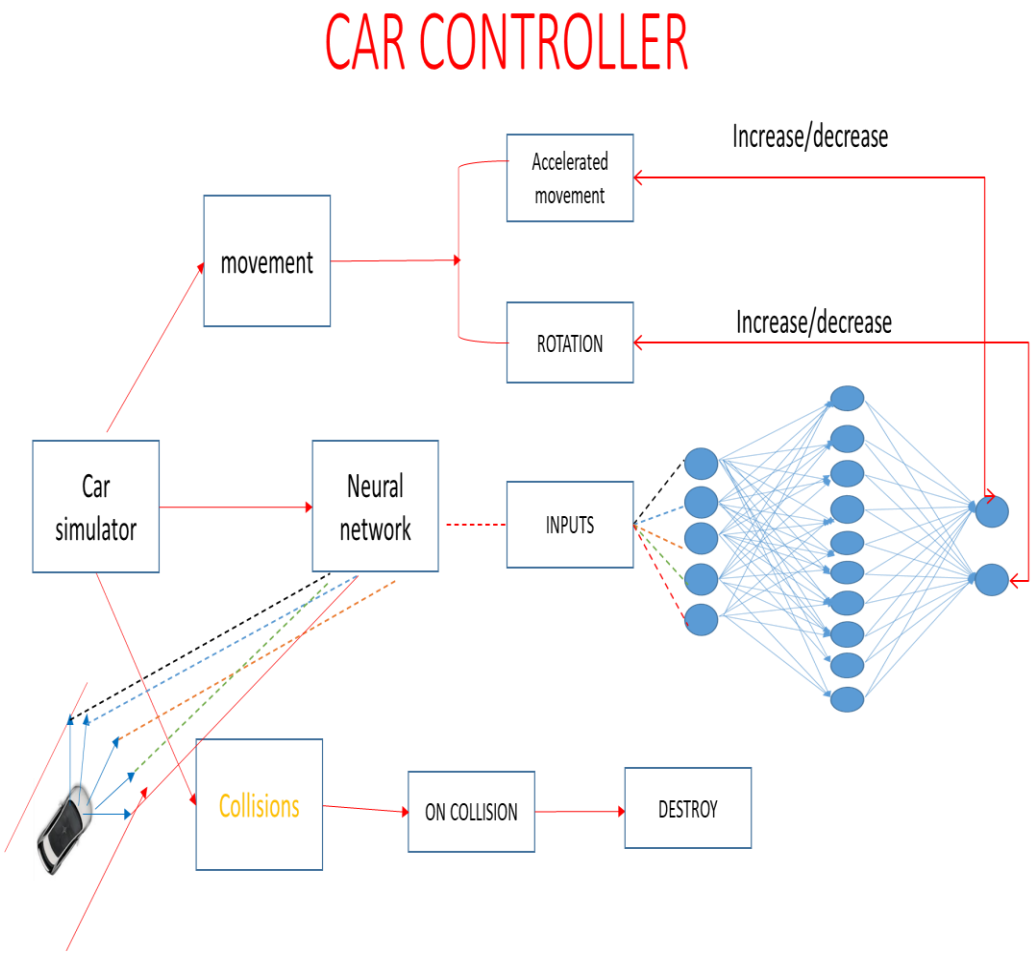


Figure:10

Experiment Result:

In our environment we have successfully test our project. In our system we have both positive and negative test result. We tested our system on several basis. We tried different parameter to get a good solution. We tried with different neural network architecture. Such as we changed neuron size in hidden layer and also, we tried different laser size. Here we tried to show our simulator result in graphical representation. We defined result graph with two basic parameters here. We defined generation in x coordinate and learning rate with respect to travel distance in y coordinate. As our auto driving car will learn trough reinforcement learning. It will improve its learning generation by generation.

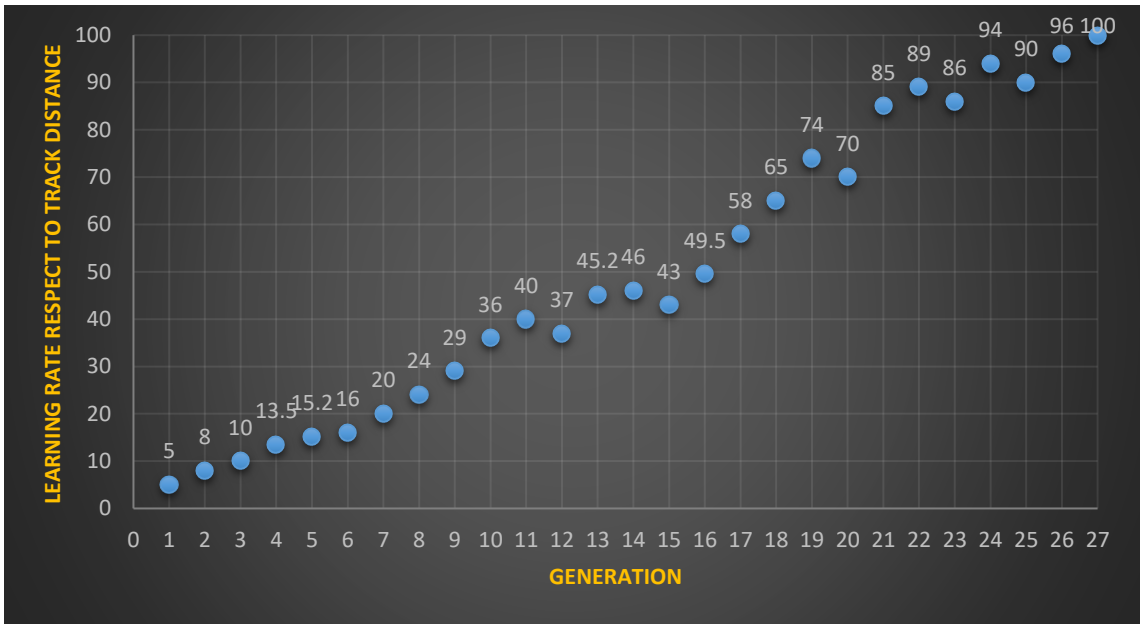


Figure: Test result 1

Graph Explanation:

We defined cars learning rate on basis of race track length and distance travelled by each generation car in the track. We defined 100% learning rate for completing the finish line in track.

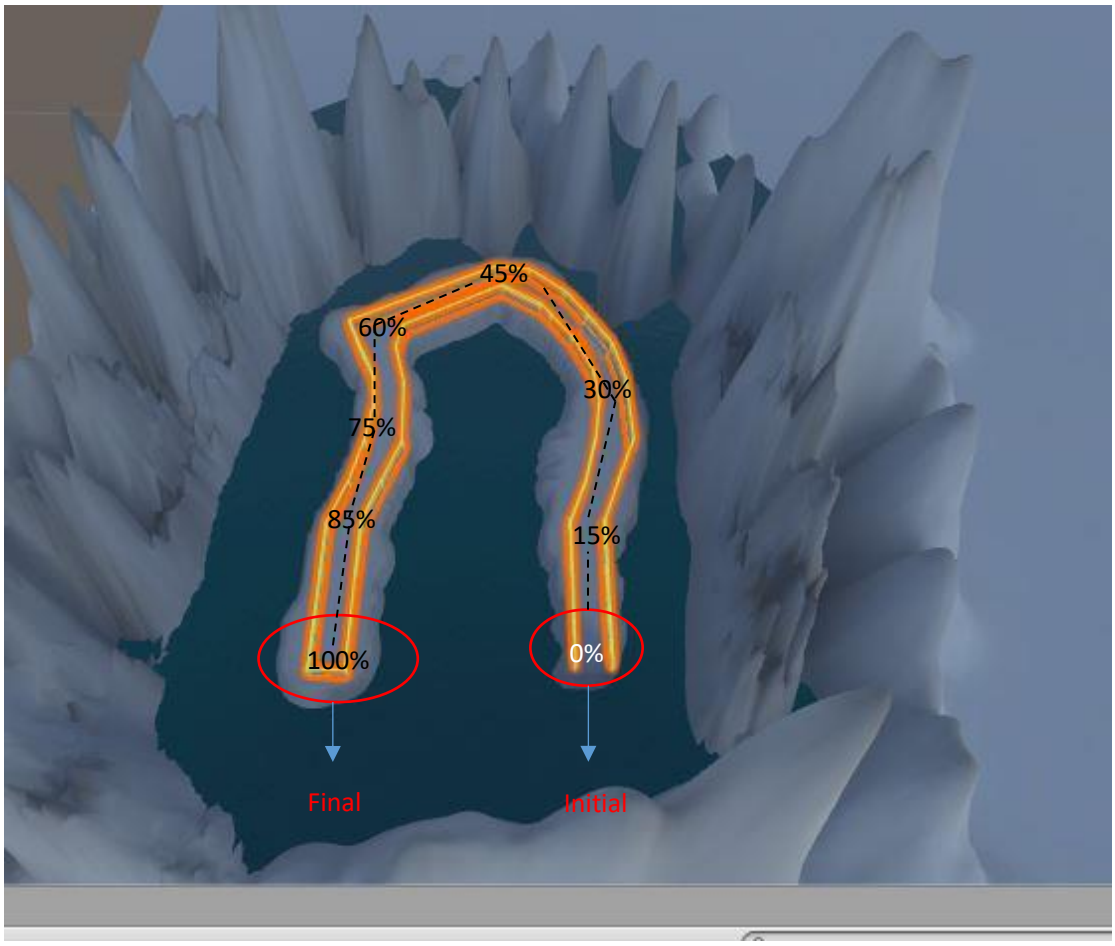


Figure:11

As you can see in figure 12, we defined learning rate in different curve of race track and run our simulation to get the graph data from each generation. In our environment the cars reached at the final track after 27 successive generation.

Experiment analysis:

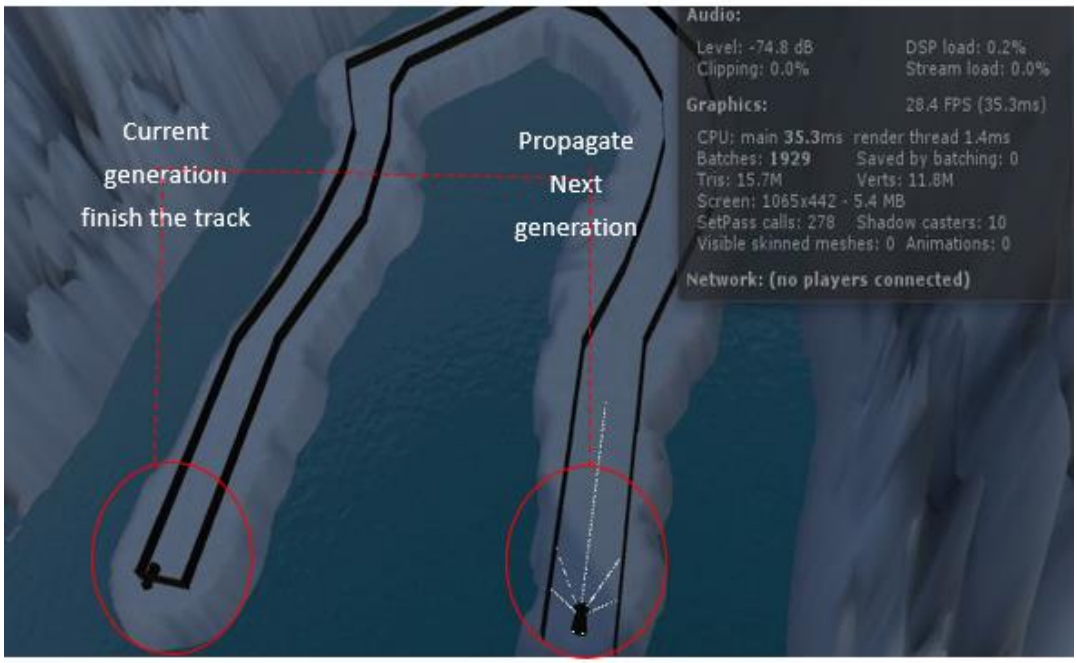
As we can see in test result 1 the learning rate is not in increasing order. Sometimes it's in increasing order and sometimes it's in decreasing order. Here, it reaches the final destination without touching the obstacles after 27 successive generation. Through all those generation sometimes its learning rate was high and sometimes it falls the previous generation. It's because in each generation we did mutation with crossover to bring the diversity in new generation as we can get some different DNA in new generation. Sometimes it gives good diversity and sometimes bad. As you can see in generation 20 and 21.

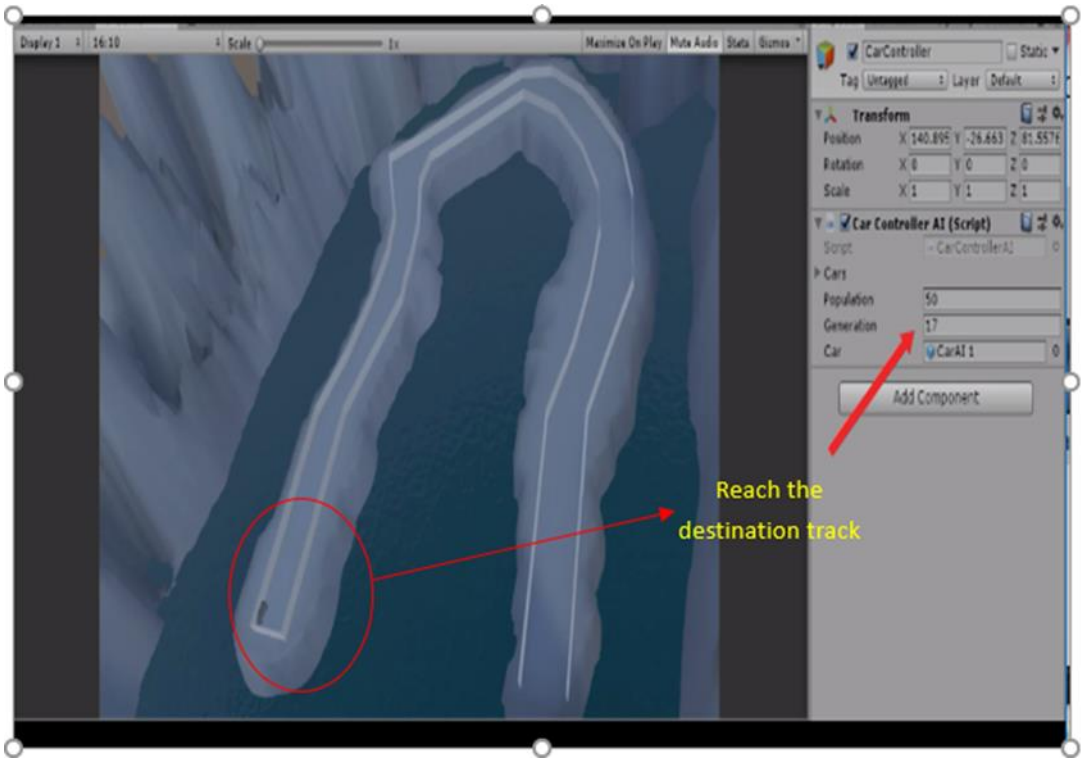
Here we attached one video simulator of our auto driving car what we got in our test result:

This is Baidu storage link with password. As we can't attach video in Microsoft word file, we uploaded this file in online storage to share for good review.

链接: https://pan.baidu.com/s/12zx470pg7NP3MeYcLIlOWA 提取码: vrtw.

Here, we added some screenshot from our simulator:





Figures: Screenshots from our simulator

Future work plan:

In our project, we just used the laser input for our system while real driver has vision for traffic signals and road navigation. In our future project we would like add vision for our car. We want add camera model with computer vision technology as our car can see and analyze the traffic signals and road navigation with convolution neural network techniques. Where we can train our model with real life environment image.

In our current project we still like to test it with different neural network architecture and laser size.

Conclusion:

In the era of artificial intelligence auto driving is the best solution for safe transportation. Genetic algorithm can be very effective learning algorithm for auto driving car.

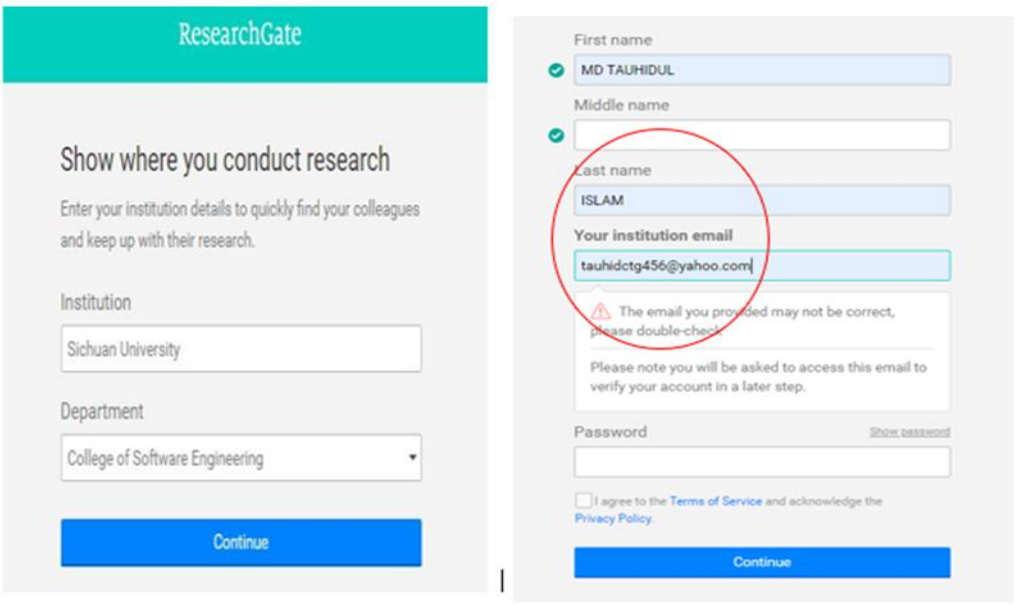
Throughout the project we learned many new things in this field as it is our course project. We are very grateful to professor Haixian, Zhang who made this topic very easy and clear throughout her class presentation and vast experience in this field. We are very grateful to her for encouraging us with different class activities like team work, presentation and project work etc.

Especially, I would like to thank her personally because biology was the scariest subject for me before attending her class. She made it very easy and clear in her class. As a result, I could do something with genetic algorithm.

Difficulties throughout the project:

Here, I want to point out one thing that is the limitation of resources. As a student we need to read many published papers to enrich our knowledge. Such as, when I was doing this project, I was trying to read some papers on this topic but I couldn't sign up in so many publication sites because I didn't have university email address what I should have as a student of Sichuan university.

Here, I attached a screenshot as proof. That might help you to understand my problems.



Hope, you can discuss about this problem with our department for us.

Thank you.

References:

1. https://selfdrivingcars.mit.edu/
2. https://awesomeopensource.com/project/Carmezim/MIT-6.S094
3. https://work.caltech.edu/telecourse.html
4. https://docs.unity3d.com/Manual/
5. http://videolectures.net/DLRLsummerschool2018_toronto/
6. https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html#introduction
7. Goldberg, D. E. 1989. Genetic algorithms in search, optimization and machine learning. Addison-Wesley.
8. Holland, J. H. 1975. Adaption in natural and artificial systems. University of Michigan Press.
9. Bäck T (1996) Evolutionary algorithms in theory and practice. Oxford University Press, OxfordzbMATHGoogle Scholar
10. Goldberg D (1989b) Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Reading, MA

Acknowledgement:

if you have any suggestion for us, we would like to take it very seriously as a beginner.

Email: tauhidctg456@yahoo.com

WeChat: tauhid456