A

MINI PROJECT

**Self driving car simulation using simple neural network ...**

Submitted

By

Ayush\_chamoli\_K1\_23021179

(Bachelor of Technology)

3rd SEM

Under the Guidance of

DR. Amit Gupta

Department of Computer Science



Department of Computer Science

Faculty of technology

Dehradun ,Uttarakhand 248002

January 2025

**CANDIDATE’S DECLARATION**

I hereby declare that the mini project work being presented in this report entitled “**Self driving car simulation using simple neural network**” submitted in the department of computer science , **FACULTY OF TECHONOLOGY**, Graphic Era Hill University , Dehradun is the authentic work out by me under the guidance of **DR. Amit Gupta,** Department of computer Science, Graphic Era Hill University , Dehradun.

Date \_ \_/\_ \_/\_ \_ \_ \_

**AYUSH CHAMOLI**

B.Tech 3rd sem

department of computer science

Graphic Era Hill University

Dehradun 248002 (U.K)

**CERTIFICATE**

This is to certify that this report represent the original work done by **AYUSH CHAMOLI** during this project submission as a partial fulfilment of the requirement for the system Design Project of Bachelor of Technology, IIIrd semester ,of the Graphic Era Hill University , Dehradun (UK-248002).

Date \_ \_/\_ \_/\_ \_ \_ \_

**AYUSH CHAMOL**

B.Tech 3rd sem

department of computer science

Graphic Era Hill University

Dehradun 248002 (U.K)

**ACKNOWLEDGEMENT**

I would like to express my special thanks of gratitude to my teacher **DR. Amit Gupta** who me the golden opportunity to do this wonderful project on the topic **Self driving car simulation using simple neural network**, which also helped me in doing a lot of Research and I come to know about so many new think I am really thankful to them.

Date \_ \_/\_ \_/\_ \_ \_ \_

**AYUSH CHAMOL**

B.Tech 3rd sem

department of computer science

Graphic Era Hill University

Dehradun 248002 (U.K)

Synopsis

**1. TITLE OF THE PROJECT:**

Self driving car simulation using simple neural network

**2. OBJECTIVE OF THE PROJECT**

Autonomous Navigation: Develop a simulation where the car can navigate a virtual environment without human intervention by making real-time driving decisions.

Obstacle Detection and Avoidance: Enable the car to detect objects like walls or other vehicles and avoid collisions using inputs from simulated sensors or environment data.

Neural Network Learning: Implement a simple neural network to process sensor data, predict driving actions, and improve performance based on training data or pre-defined rules.

Path Following: Teach the car to follow specific routes or lanes accurately, using visual inputs or waypoints to mimic real-world lane-keeping systems.

Performance Evaluation: Test and evaluate the simulation for efficiency, accuracy, and decision-making to improve neural network design and fine-tune parameters for optimal performance.

**3. CODE REQUIRMENT**

* **index.html**

<!DOCTYPE html>

<head>

<title>Self-driving car - No libraries</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<canvas id="carCanvas"></canvas>

<div id="verticalButtons">

<button onclick="save()">💾</button>

<button onclick="discard()">🗑</button>

</div>

<canvas id="networkCanvas"></canvas>

<script src="visualizer.js"></script>

<script src="network.js"></script>

<script src="sensor.js"></script>

<script src="utils.js"></script>

<script src="road.js"></script>

<script src="controls.js"></script>

<script src="car.js"></script>

<script src="main.js"></script>

</body>

</html>

* **car.js**

class Car{

constructor(x,y,width,height,controlType,maxSpeed=3){

this.x=x;

this.y=y;

this.width=width;

this.height=height;

this.speed=0;

this.acceleration=0.2;

this.maxSpeed=maxSpeed;

this.friction=0.05;

this.angle=0;

this.damaged=false;

this.useBrain=controlType=="AI";

if(controlType!="DUMMY"){

this.sensor=new Sensor(this);

this.brain=new NeuralNetwork(

[this.sensor.rayCount,6,4]

);

}

this.controls=new Controls(controlType);

}

update(roadBorders,traffic){

if(!this.damaged){

this.#move();

this.polygon=this.#createPolygon();

this.damaged=this.#assessDamage(roadBorders,traffic);

}

if(this.sensor){

this.sensor.update(roadBorders,traffic);

const offsets=this.sensor.readings.map(

s=>s==null?0:1-s.offset

);

const outputs=NeuralNetwork.feedForward(offsets,this.brain);

if(this.useBrain){

this.controls.forward=outputs[0];

this.controls.left=outputs[1];

this.controls.right=outputs[2];

this.controls.reverse=outputs[3];

}

}

}

#assessDamage(roadBorders,traffic){

for(let i=0;i<roadBorders.length;i++){

if(polysIntersect(this.polygon,roadBorders[i])){

return true;

}

}

for(let i=0;i<traffic.length;i++){

if(polysIntersect(this.polygon,traffic[i].polygon)){

return true;

}

}

return false;

}

#createPolygon(){

const points=[];

const rad=Math.hypot(this.width,this.height)/2;

const alpha=Math.atan2(this.width,this.height);

points.push({

x:this.x-Math.sin(this.angle-alpha)\*rad,

y:this.y-Math.cos(this.angle-alpha)\*rad

});

points.push({

x:this.x-Math.sin(this.angle+alpha)\*rad,

y:this.y-Math.cos(this.angle+alpha)\*rad

});

points.push({

x:this.x-Math.sin(Math.PI+this.angle-alpha)\*rad,

y:this.y-Math.cos(Math.PI+this.angle-alpha)\*rad

});

points.push({

x:this.x-Math.sin(Math.PI+this.angle+alpha)\*rad,

y:this.y-Math.cos(Math.PI+this.angle+alpha)\*rad

});

return points;

}

#move(){

if(this.controls.forward){

this.speed+=this.acceleration;

}

if(this.controls.reverse){

this.speed-=this.acceleration;

}

if(this.speed>this.maxSpeed){

this.speed=this.maxSpeed;

}

if(this.speed<-this.maxSpeed/2){

this.speed=-this.maxSpeed/2;

}

if(this.speed>0){

this.speed-=this.friction;

}

if(this.speed<0){

this.speed+=this.friction;

}

if(Math.abs(this.speed)<this.friction){

this.speed=0;

}

if(this.speed!=0){

const flip=this.speed>0?1:-1;

if(this.controls.left){

this.angle+=0.03\*flip;

}

if(this.controls.right){

this.angle-=0.03\*flip;

}

}

this.x-=Math.sin(this.angle)\*this.speed;

this.y-=Math.cos(this.angle)\*this.speed;

}

draw(ctx, color = "blue", drawSensor = false) {

// Body of the car

ctx.fillStyle = this.damaged ? "gray" : color;

ctx.beginPath();

ctx.moveTo(this.polygon[0].x, this.polygon[0].y);

for (let i = 1; i < this.polygon.length; i++) {

ctx.lineTo(this.polygon[i].x, this.polygon[i].y);

}

ctx.closePath();

ctx.fill();

// Add car details

if (!this.damaged) {

// Windows

ctx.fillStyle = "lightblue";

const windowOffset = this.width / 5;

ctx.fillRect(

this.x - this.width / 4,

this.y - this.height / 2 + windowOffset,

this.width / 2,

this.height / 4

);

// Headlights

ctx.fillStyle = "yellow";

const headlightSize = this.width / 6;

ctx.fillRect(

this.x - this.width / 2 + headlightSize / 2,

this.y - this.height / 2,

headlightSize,

headlightSize / 2

);

ctx.fillRect(

this.x + this.width / 2 - headlightSize \* 1.5,

this.y - this.height / 2,

headlightSize,

headlightSize / 2

);

// Tail lights

ctx.fillStyle = "red";

ctx.fillRect(

this.x - this.width / 2 + headlightSize / 2,

this.y + this.height / 2 - headlightSize / 2,

headlightSize,

headlightSize / 2

);

ctx.fillRect(

this.x + this.width / 2 - headlightSize \* 1.5,

this.y + this.height / 2 - headlightSize / 2,

headlightSize,

headlightSize / 2

);

}

// Sensors (optional)

if (this.sensor && drawSensor) {

this.sensor.draw(ctx);

}

}

}

* **controls.js**

class Controls{

constructor(type){

this.forward=false;

this.left=false;

this.right=false;

this.reverse=false;

switch(type){

case "KEYS":

this.#addKeyboardListeners();

break;

case "DUMMY":

this.forward=true;

break;

}

}

#addKeyboardListeners(){

document.onkeydown=(event)=>{

switch(event.key){

case "ArrowLeft":

this.left=true;

break;

case "ArrowRight":

this.right=true;

break;

case "ArrowUp":

this.forward=true;

break;

case "ArrowDown":

this.reverse=true;

break;

}

}

document.onkeyup=(event)=>{

switch(event.key){

case "ArrowLeft":

this.left=false;

break;

case "ArrowRight":

this.right=false;

break;

case "ArrowUp":

this.forward=false;

break;

case "ArrowDown":

this.reverse=false;

break;

}

}

}

}

* **main.js**

carCanvas.height=window.innerHeight;

carCanvas.width=200;

networkCanvas.height=window.innerHeight;

networkCanvas.width=298;

const carCtx=carCanvas.getContext("2d");

const networkCtx=networkCanvas.getContext("2d");

const road=new Road(carCanvas.width/2,carCanvas.width\*0.9);

const N=1;

const cars=generateCars(N);

const traffic = [];

const trafficCount = 50;

const laneWidth = road.width / road.laneCount;

for (let i = 0; i < trafficCount; i++) {

const lane = Math.floor(Math.random() \* 3);

const y = -200 \* i;

traffic.push(

new Car(road.getLaneCenter(lane), y, 30, 50, "DUMMY", 2)

);

}

let bestCar=cars[0];

if(localStorage.getItem("bestBrain")){

for(let i=0;i<cars.length;i++){

cars[i].brain=JSON.parse(

localStorage.getItem("bestBrain"));

if(i>0){

NeuralNetwork.mutate(cars[i].brain,0.1);

}

}

}

animate();

function animate(){

for(let i=0;i<traffic.length;i++){

traffic[i].update([],[]);

}

for(let i=0;i<cars.length;i++){

cars[i].update(road.borders,traffic);

}

bestCar=cars.find(

c=>c.y==Math.min(

...cars.map(c=>c.y)

));

carCanvas.height=window.innerHeight;

networkCanvas.height=window.innerHeight;

carCtx.translate(0,-bestCar.y+carCanvas.height\*0.7);

road.draw(carCtx);

for(let i=0;i<traffic.length;i++){

traffic[i].draw(carCtx);

}

carCtx.globalAlpha=0.2;

for(let i=0;i<cars.length;i++){

cars[i].draw(carCtx);

}

carCtx.globalAlpha=1;

bestCar.draw(carCtx,true);

Visualizer.drawNetwork(networkCtx,bestCar.brain);

requestAnimationFrame(animate);

}

function generateCars(N){

const cars=[];

for(let i=1;i<=N;i++){

cars.push(new Car(100,100,30,50,"AI"));

}

return cars;

}

function save(){

localStorage.setItem("bestBrain",

JSON.stringify(bestCar.brain));

}

function discard(){

localStorage.removeItem("bestBrain");

}

* **networks.js**

class NeuralNetwork{

constructor(neuronCounts){

this.levels=[];

for(let i=0;i<neuronCounts.length-1;i++){

this.levels.push(new Level(

neuronCounts[i],neuronCounts[i+1]

));

}

}

static feedForward(givenInputs,network){

let outputs=Level.feedForward(

givenInputs,network.levels[0]);

for(let i=1;i<network.levels.length;i++){

outputs=Level.feedForward(

outputs,network.levels[i]);

}

return outputs;

}

static mutate(network,amount=1){

network.levels.forEach(level => {

for(let i=0;i<level.biases.length;i++){

level.biases[i]=lerp(

level.biases[i],

Math.random()\*2-1,

amount

)

}

for(let i=0;i<level.weights.length;i++){

for(let j=0;j<level.weights[i].length;j++){

level.weights[i][j]=lerp(

level.weights[i][j],

Math.random()\*2-1,

amount

)

}

}

});

}

}

class Level{

constructor(inputCount,outputCount){

this.inputs=new Array(inputCount);

this.outputs=new Array(outputCount);

this.biases=new Array(outputCount);

this.weights=[];

for(let i=0;i<inputCount;i++){

this.weights[i]=new Array(outputCount);

}

Level.#randomize(this);

}

static #randomize(level){

for(let i=0;i<level.inputs.length;i++){

for(let j=0;j<level.outputs.length;j++){

level.weights[i][j]=Math.random()\*2-1;

}

}

for(let i=0;i<level.biases.length;i++){

level.biases[i]=Math.random()\*2-1;

}

}

static feedForward(givenInputs,level){

for(let i=0;i<level.inputs.length;i++){

level.inputs[i]=givenInputs[i];

}

for(let i=0;i<level.outputs.length;i++){

let sum=0

for(let j=0;j<level.inputs.length;j++){

sum+=level.inputs[j]\*level.weights[j][i];

}

if(sum>level.biases[i]){

level.outputs[i]=1;

}else{

level.outputs[i]=0;

}

}

return level.outputs;

}

}

* **roads.js**

class Road{

constructor(x,width,laneCount=3){

this.x=x;

this.width=width;

this.laneCount=laneCount;

this.left=x-width/2;

this.right=x+width/2;

const infinity=1000000;

this.top=-infinity;

this.bottom=infinity;

const topLeft={x:this.left,y:this.top};

const topRight={x:this.right,y:this.top};

const bottomLeft={x:this.left,y:this.bottom};

const bottomRight={x:this.right,y:this.bottom};

this.borders=[

[topLeft,bottomLeft],

[topRight,bottomRight]

];

}

getLaneCenter(laneIndex){

const laneWidth=this.width/this.laneCount;

return this.left+laneWidth/2+

Math.min(laneIndex,this.laneCount-1)\*laneWidth;

}

draw(ctx) {

const gradient = ctx.createLinearGradient(this.left, 0, this.right, 0);

gradient.addColorStop(0, "#2c3e50");

gradient.addColorStop(0.5, "#34495e");

gradient.addColorStop(1, "#2c3e50");

ctx.fillStyle = gradient;

ctx.fillRect(this.left, this.top, this.width, this.bottom - this.top);

ctx.lineWidth = 2;

ctx.strokeStyle = "white";

ctx.setLineDash([20, 20]);

for (let i = 1; i <= this.laneCount - 1; i++) {

const x = lerp(this.left, this.right, i / this.laneCount);

ctx.beginPath();

ctx.moveTo(x, this.top);

ctx.lineTo(x, this.bottom);

ctx.stroke();

}

ctx.setLineDash([]);

ctx.lineWidth = 5;

ctx.strokeStyle = "#e74c3c";

ctx.beginPath();

ctx.moveTo(this.left, this.top);

ctx.lineTo(this.left, this.bottom);

ctx.stroke();

ctx.beginPath();

ctx.moveTo(this.right, this.top);

ctx.lineTo(this.right, this.bottom);

ctx.stroke();

}

}

* **sensor.js**

class Sensor{

constructor(car){

this.car=car;

this.rayCount=5;

this.rayLength=150;

this.raySpread=Math.PI/2;

this.rays=[];

this.readings=[];

}

update(roadBorders,traffic){

this.#castRays();

this.readings=[];

for(let i=0;i<this.rays.length;i++){

this.readings.push(

this.#getReading(

this.rays[i],

roadBorders,

traffic

)

);

}

}

#getReading(ray,roadBorders,traffic){

let touches=[];

for(let i=0;i<roadBorders.length;i++){

const touch=getIntersection(

ray[0],

ray[1],

roadBorders[i][0],

roadBorders[i][1]

);

if(touch){

touches.push(touch);

}

}

for(let i=0;i<traffic.length;i++){

const poly=traffic[i].polygon;

for(let j=0;j<poly.length;j++){

const value=getIntersection(

ray[0],

ray[1],

poly[j],

poly[(j+1)%poly.length]

);

if(value){

touches.push(value);

}

}

}

if(touches.length==0){

return null;

}else{

const offsets=touches.map(e=>e.offset);

const minOffset=Math.min(...offsets);

return touches.find(e=>e.offset==minOffset);

}

}

#castRays(){

this.rays=[];

for(let i=0;i<this.rayCount;i++){

const rayAngle=lerp(

this.raySpread/2,

-this.raySpread/2,

this.rayCount==1?0.5:i/(this.rayCount-1)

)+this.car.angle;

const start={x:this.car.x, y:this.car.y};

const end={

x:this.car.x-

Math.sin(rayAngle)\*this.rayLength,

y:this.car.y-

Math.cos(rayAngle)\*this.rayLength

};

this.rays.push([start,end]);

}

}

draw(ctx){

for(let i=0;i<this.rayCount;i++){

let end=this.rays[i][1];

if(this.readings[i]){

end=this.readings[i];

}

ctx.beginPath();

ctx.lineWidth=2;

ctx.strokeStyle="yellow";

ctx.moveTo(

this.rays[i][0].x,

this.rays[i][0].y

);

ctx.lineTo(

end.x,

end.y

);

ctx.stroke();

ctx.beginPath();

ctx.lineWidth=2;

ctx.strokeStyle="black";

ctx.moveTo(

this.rays[i][1].x,

this.rays[i][1].y

);

ctx.lineTo(

end.x,

end.y

);

ctx.stroke();

}

}

}

* **style.css**

/\* General Body Styling \*/

body {

margin: 0;

background: linear-gradient(to bottom, #2c3e50, #4ca1af); /\* Gradient background \*/

overflow: hidden;

display: flex;

justify-content: center;

align-items: center;

font-family: 'Arial', sans-serif; /\* Modern font \*/

color: #ffffff;

}

#verticalButtons {

display: flex;

flex-direction: column;

gap: 10px;

}

button {

border: none;

border-radius: 8px;

padding: 10px 15px;

margin: 5px;

font-size: 16px;

font-weight: bold;

cursor: pointer;

background: linear-gradient(135deg, #3498db, #2980b9);

color: #ffffff;

box-shadow: 0px 4px 6px rgba(0, 0, 0, 0.3);

transition: all 0.3s ease;

}

button:hover {

background: linear-gradient(135deg, #27ae60, #2ecc71); /\* Change color on hover \*/

transform: scale(1.1); /\* Slightly enlarge \*/

box-shadow: 0px 6px 8px rgba(0, 0, 0, 0.5); /\* Deepen shadow \*/

}

/\* Canvas Styling \*/

#networkCanvas {

background: #1c1c1c; /\* Darker background for better contrast \*/

border: 2px solid #3498db; /\* Add border for distinction \*/

border-radius: 10px; /\* Rounded corners \*/

box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.6); /\* Shadow for depth \*/

}

#carCanvas {

background: #ecf0f1; /\* Light background for visibility \*/

border: 2px solid #34495e; /\* Subtle border \*/

border-radius: 10px; /\* Rounded corners \*/

box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.6); /\* Shadow for depth \*/

}

/\* Centered Layout with Better Spacing \*/

#container {

display: flex;

flex-direction: column;

gap: 20px; /\* Space between elements \*/

align-items: center;

padding: 20px;

}

/\* Headings and Text Styling \*/

h1, h2, h3 {

color: #ffffff;

text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.7); /\* Glow effect \*/

text-align: center;

}

p {

color: #ffffff;

font-size: 16px;

line-height: 1.6;

max-width: 600px; /\* Limit width for readability \*/

text-align: center;

}

* **utils.js**

function lerp(A,B,t){

return A+(B-A)\*t;

}

function getIntersection(A,B,C,D){

const tTop=(D.x-C.x)(A.y-C.y)-(D.y-C.y)(A.x-C.x);

const uTop=(C.y-A.y)(A.x-B.x)-(C.x-A.x)(A.y-B.y);

const bottom=(D.y-C.y)(B.x-A.x)-(D.x-C.x)(B.y-A.y);

if(bottom!=0){

const t=tTop/bottom;

const u=uTop/bottom;

if(t>=0 && t<=1 && u>=0 && u<=1){

return {

x:lerp(A.x,B.x,t),

y:lerp(A.y,B.y,t),

offset:t

}

}

}

return null;

}

function polysIntersect(poly1, poly2){

for(let i=0;i<poly1.length;i++){

for(let j=0;j<poly2.length;j++){

const touch=getIntersection(

poly1[i],

poly1[(i+1)%poly1.length],

poly2[j],

poly2[(j+1)%poly2.length]

);

if(touch){

return true;

}

}

}

return false;

}

function getRGBA(value){

const alpha=Math.abs(value);

const R=value<0?0:255;

const G=R;

const B=value>0?0:255;

return "rgba("+R+","+G+","+B+","+alpha+")";

}

* **visualizer.js**

class Visualizer{

static drawNetwork(ctx,network){

const margin=50;

const left=margin;

const top=margin;

const width=ctx.canvas.width-margin\*2;

const height=ctx.canvas.height-margin\*2;

const levelHeight=height/network.levels.length;

for(let i=network.levels.length-1;i>=0;i--){

const levelTop=top+

lerp(

height-levelHeight,

0,

network.levels.length==1

?0.5

:i/(network.levels.length-1)

);

ctx.setLineDash([7,3]);

Visualizer.drawLevel(ctx,network.levels[i],

left,levelTop,

width,levelHeight,

i==network.levels.length-1

?['🠉','🠈','🠊','🠋']

:[]

);

}

}

static drawLevel(ctx,level,left,top,width,height,outputLabels){

const right=left+width;

const bottom=top+height;

const {inputs,outputs,weights,biases}=level;

for(let i=0;i<inputs.length;i++){

for(let j=0;j<outputs.length;j++){

ctx.beginPath();

ctx.moveTo(

Visualizer.#getNodeX(inputs,i,left,right),

bottom

);

ctx.lineTo(

Visualizer.#getNodeX(outputs,j,left,right),

top

);

ctx.lineWidth=2;

ctx.strokeStyle=getRGBA(weights[i][j]);

ctx.stroke();

}

}

const nodeRadius=18;

for(let i=0;i<inputs.length;i++){

const x=Visualizer.#getNodeX(inputs,i,left,right);

ctx.beginPath();

ctx.arc(x,bottom,nodeRadius,0,Math.PI\*2);

ctx.fillStyle="black";

ctx.fill();

ctx.beginPath();

ctx.arc(x,bottom,nodeRadius\*0.6,0,Math.PI\*2);

ctx.fillStyle=getRGBA(inputs[i]);

ctx.fill();

}

for(let i=0;i<outputs.length;i++){

const x=Visualizer.#getNodeX(outputs,i,left,right);

ctx.beginPath();

ctx.arc(x,top,nodeRadius,0,Math.PI\*2);

ctx.fillStyle="black";

ctx.fill();

ctx.beginPath();

ctx.arc(x,top,nodeRadius\*0.6,0,Math.PI\*2);

ctx.fillStyle=getRGBA(outputs[i]);

ctx.fill();

ctx.beginPath();

ctx.lineWidth=2;

ctx.arc(x,top,nodeRadius\*0.8,0,Math.PI\*2);

ctx.strokeStyle=getRGBA(biases[i]);

ctx.setLineDash([3,3]);

ctx.stroke();

ctx.setLineDash([]);

if(outputLabels[i]){

ctx.beginPath();

ctx.textAlign="center";

ctx.textBaseline="middle";

ctx.fillStyle="black";

ctx.strokeStyle="white";

ctx.font=(nodeRadius\*1.5)+"px Arial";

ctx.fillText(outputLabels[i],x,top+nodeRadius\*0.1);

ctx.lineWidth=0.5;

ctx.strokeText(outputLabels[i],x,top+nodeRadius\*0.1);

}

}

}

static #getNodeX(nodes,index,left,right){

return lerp(

left,

right,

nodes.length==1

?0.5

:index/(nodes.length-1)

);

}

}