Introduction

Background

- Traffic accidents cause an average of 3290 deaths each day around the world. An estimated 90% of these are due to human error.
- Autonomous cars solve this problem, and are currently being researched and developed by many leading technology companies like Google, Tesla, and Uber.
- Artificial intelligence (AI) is often used for image recognition and sensor processing, yet my research found no evidence of AI being used as a method for controlling an autonomous car. So I set out to see if an artificial neural network (ANN) can be used for this purpose.
- This is a novel approach compared to the traditional software solution, as it is very flexible and adaptive. The ANN would be trained to work with any available sensors, controls, and driving conditions.
- I believe that my project may contribute to saving human lives and making independent transportation available to elderly and people with disabilities.

Purpose

- To create a self-learning control system capable of autonomously driving a car in the simulated environment using an artificial neural network trained by my neuroevolution algorithm.

Hypothesis

- My neuroevolution algorithm will train an artificial neural network that will control an autonomous car.
- The car will reach to the destination without collisions within the time it takes to drive an equivalent distance in a city at 30km/h.

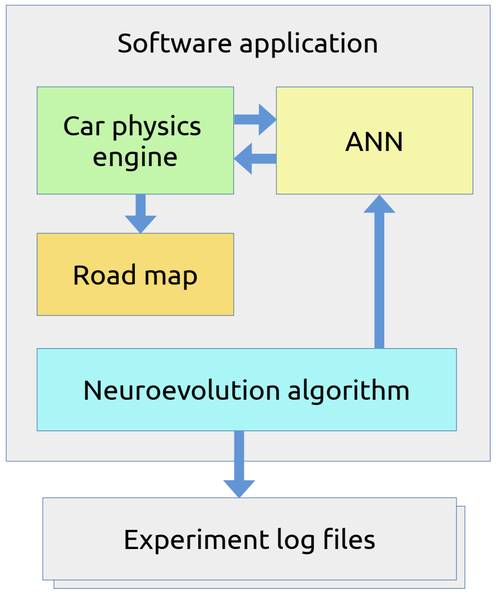


Figure 1. Project components

Artificial Neural Network

The sensor inputs are taken from the environment at regular intervals, and a decision is made using that information. The ANN used in the experiment has three layers of nodes: inputs, hidden nodes, and outputs.

A node (neuron) receives impulses from the previous layer and multiplies it by the connection's weight, then sums the result and passes it through limitation function to decide whether the neuron will "fire" the signal to the next neurons.

Adjusting the weights (importance) of the connections between neurons modifies the behaviour of the ANN.

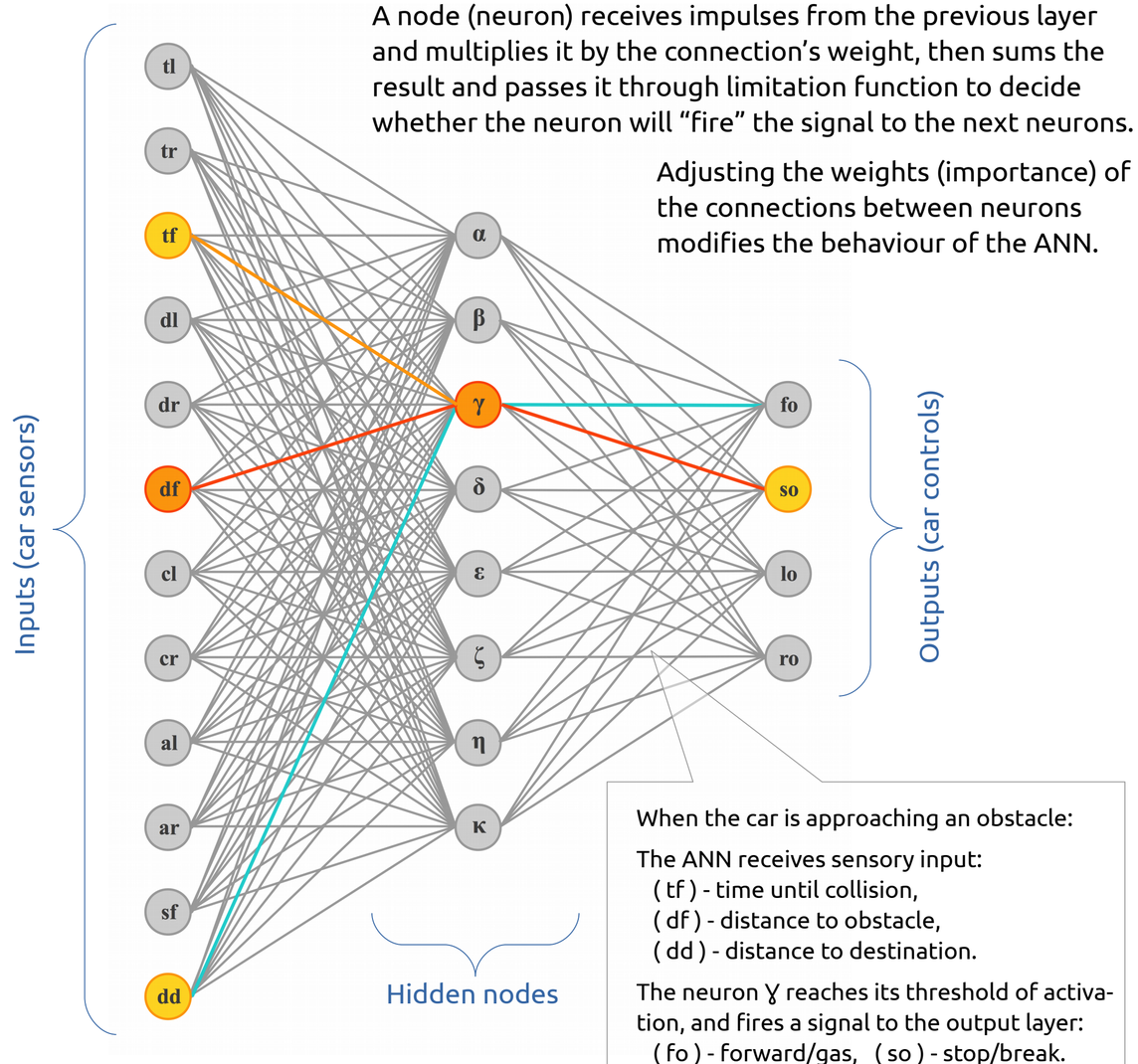


Figure 2. Structure of the ANN

When the car is approaching an obstacle:

The ANN receives sensory input:
(tf) - time until collision,
(df) - distance to obstacle,
(dd) - distance to destination.

The neuron γ reaches its threshold of activation, and fires a signal to the output layer:
(fo) - forward/gas, (so) - stop/break.

As a result, the car would ease on gas and press the break.

Design

Road Map with Obstacles

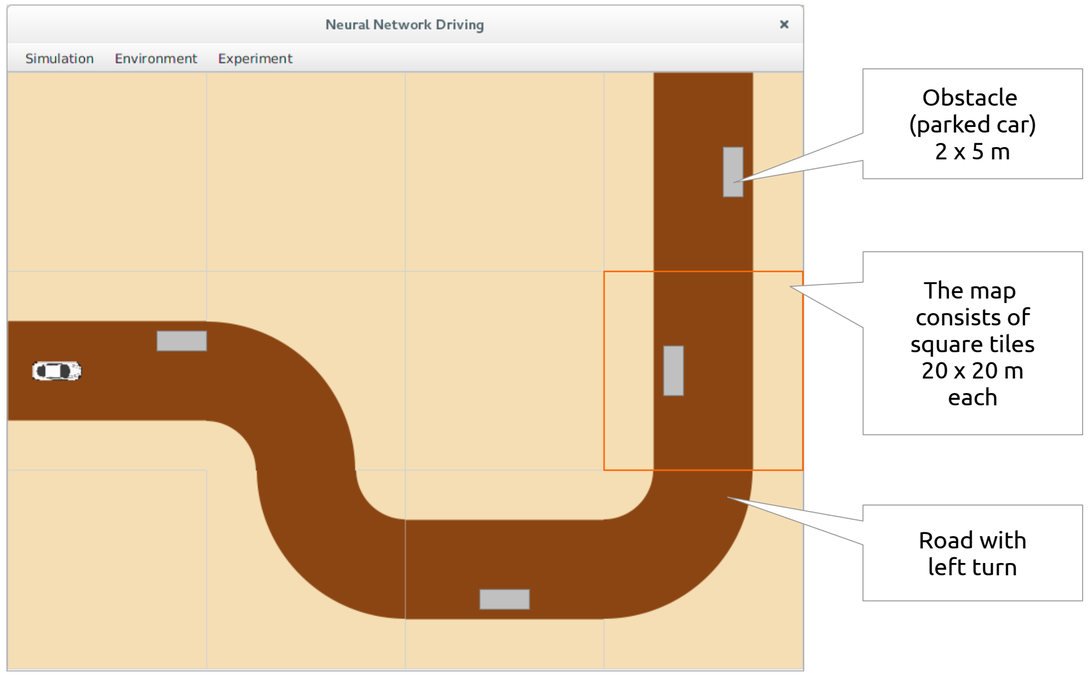


Figure 3. Application screenshot with road map

Car Physics Engine

- For a driving car, forces and acceleration change very rapidly. To accurately calculate car movement I implemented an incremental model which performs calculations every 25 ms.
- The physics engine takes in car controls (gas, break, steering) as real number inputs. It then calculates forces, acceleration, velocity, and position.

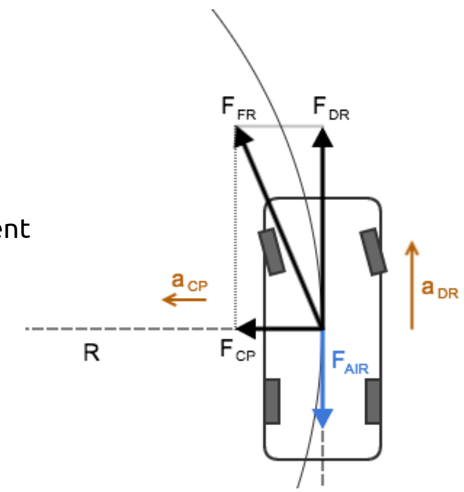
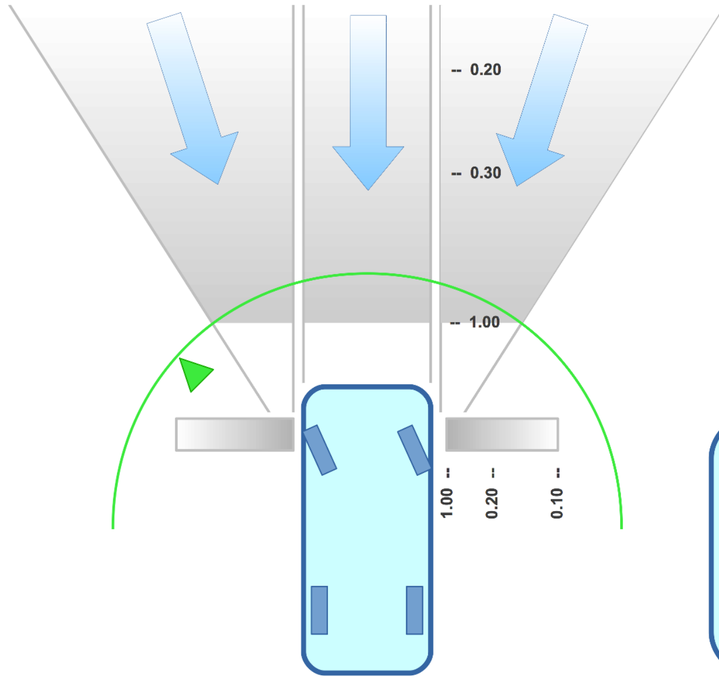


Figure 4. Car force diagram during left turn

Car Sensors and Controls



- Distance to obstacle, m (x3)
- Time until impact, s (x3)
- Road side clearance. m (x2)
- Azimuth to path, ° (x2)
- Current speed, m/s
- Distance to destination, m

For distance, time, and clearance I used inversely proportional value (1/x)

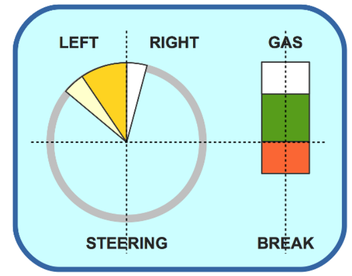


Figure 5. Car sensors and controls

Variables

Independent	Dependent	Controlled
Environment configuration	Best fitness in generation	Physics engine
Population size (1k - 100k)	ANN weights	Car characteristics
Survival Rate (1% - 5%)	Percentage of cars reaching goal	Fitness calculation
Mutation rate (0.1 - 0.5)	Percentage of cars off-road	ANN structure
Number of generations	Percentage of crashes	Car sensors and controls
		Simulation time limit

Self-Learning Process

Neuroevolution

- Assess fitness of each ANN in the current generation.
- Select the best, adjust ANN weights by mating and mutation.
- Create new ANN generation.

Simulation

- Read sensor values, provide them to ANN.
- Compute ANN state, ANN behaviour is defined by its weights.
- Read outputs from ANN, apply car physics, calculate new environment state.

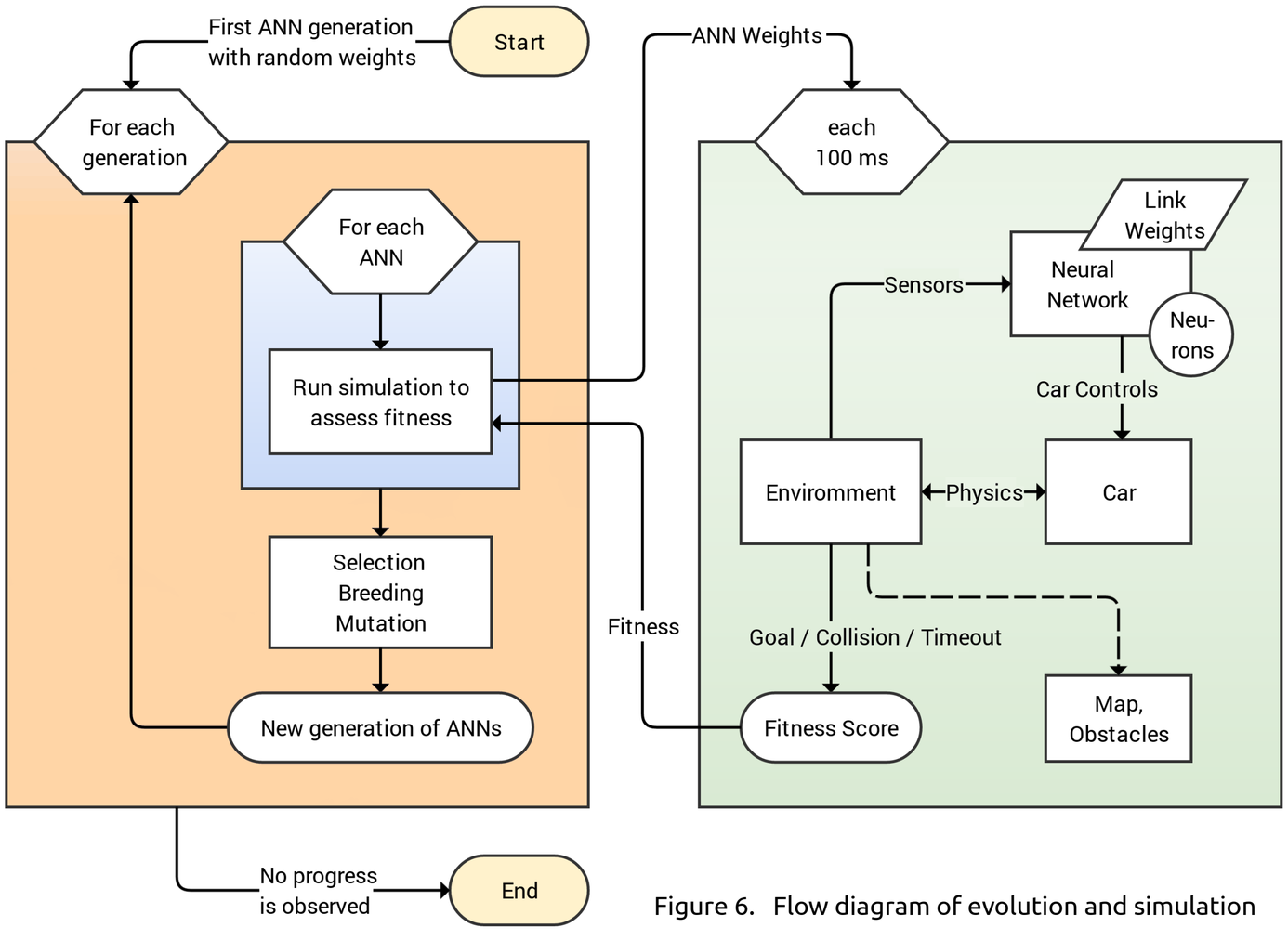


Figure 6. Flow diagram of evolution and simulation

Fitness Function

```
if ( car.reachedDestination() ) {
     fitness = 100.0 + 100.0 * (timeLimit-time) / timeLimit;
 }
 else {
     fitness = 100.0 * car.pathCompletion;
     if ( car.hadCollision() ) {  fitness = fitness * 0.8; }
     if ( car.wasOffRoad() ) {  fitness = fitness * 0.9; }
 }
```

Selection Rules

- Selection of survivors is performed based on their rank in the population, using Gaussian probability.
- Survival rate is defined by standard deviation of the function.

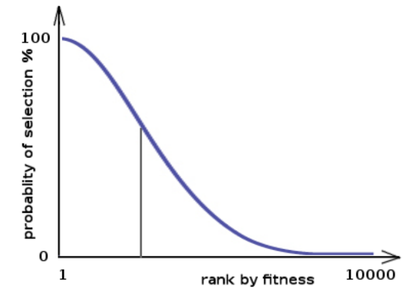


Figure 7. Selection probability

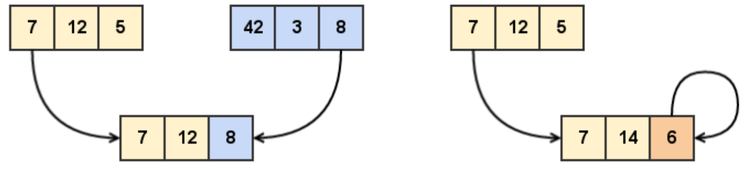


Figure 8. Illustration of mating (left) and mutation (right)

- I use mating to make new organisms, when a neuron from one organism replaces a neuron from another.
- Mutation is also used, when input weights of one neuron in an organism are slightly modified.
- Degree of change is random and is controlled by mutation rate.

Results

Experiment Data

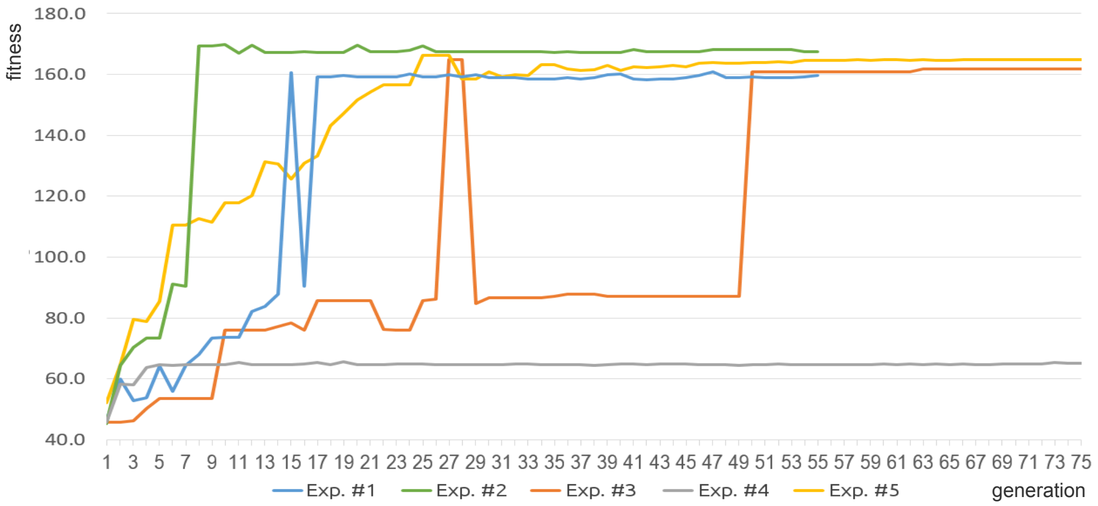


Figure 9. Best fitness per generation (map 2)

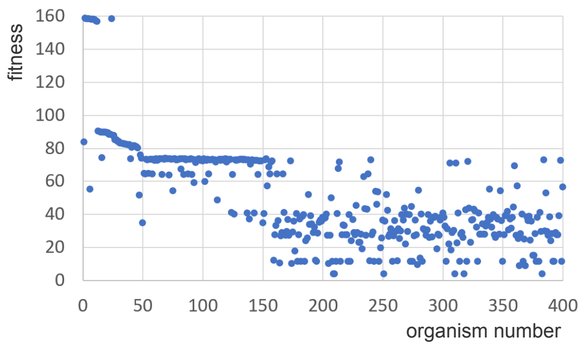


Figure 10. Fitness distribution in a generation (map 2, experiment #5, generation 50)

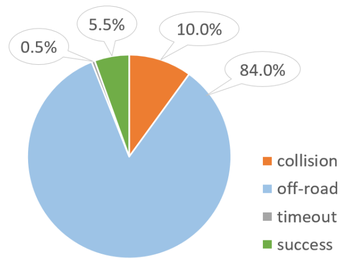


Figure 11. Outcomes for survivors (map 2, experiment #5, generation 50)

Future Work

- I want to experiment with different ANN structures, improving the physics engine, creating new ways the neuroevolution can evolve the ANNs.
- I am planning to test my artificial neural network by integrating it with a remote control car using an Arduino controller.

Observations

- Neuroevolution is a non-deterministic process. In 4 out of my 5 experiments, the neuroevolution process succeeded in training the car to reach to the destination. It took approximately 50 generations to reach the plateau in fitness.
- In every experiment the algorithm ends up with different behaviours that achieve the same goal.
- The algorithms tend to over optimize to best suit the specific training map, not taking into account risks and safety.
- As the improvement reaches a plateau, the variation between the organisms in the population start to disappear as they shift towards the best performing ones.

Conclusions

- My simplistic structure of my ANN proved to be sufficient to drive an autonomous car in my theoretical physics engine environment. The neuroevolution algorithm succeeded in teaching the ANN in controlling the car.
- In the most successful experiment, the car reached the destination in an equivalent 54 km/h (15 m/s).
- For training, a larger population size provides a higher chance for a good result while requiring more processing power. A population of 10000 organisms has proven to be a good balance of results and processing requirements for my home system.
- A higher mutation rate speeds up the improvement, but if it’s too high, it may mutate too much and lose the achieved progress. If it’s too low, it requires much more time to achieve the result .
- The fitness function was vital for successful ANN training. I had to change it four times to properly define the incentives. The fitness function was ultimately set to value distance to destination over time without collisions. However, for real life usage it must be more complex to take into account safety and rules of the road.

Acknowledgements

I would like to thank Andrew Alferov and Larisa Alferov for their guidance.