TErSLA: The Evolutionary and Reinforcement based Self-Learning Vehicle Approach

R10922147 楊敦捷[1], R11944004 李勝維[2] and R11944021 廖金億[2]

[1]Department of Computer Science and Information Engineering, National Taiwan University
[2]Graduate Institute of Networking and Multimedia, National Taiwan University

Abstract

In this final project, we present a deep learning neural networks implementation for training an intelligent car to navigate complex urban environments. All of our group members confirm on a clear statement that this final project does not have any double assignment or any completed work before this semester. We approach the problem by two methods: reinforcement learning(Deep Q Network) and genetic algorithm, which could both learn an optimal policy for the car to follow. We evaluate our approach on a simulated urban driving environment and demonstrate the ability that it is able to learn a policy in terms of safety and efficiency. Our results suggest that the choice of loss function(MSE or Huber) while training a Deep Q Networks could lead to an efficient and promising result depends on the complexity of the environment and also find that larger population sizes and suitable mutation rates in genetic algorithms could promote the performance of the model but meanwhile sacrifice the training time.

Contributions

楊敦捷	Reinforcement Learning, Report, and Presentation
李勝維	Topic proposal, Map design, Report, and Presentation
廖金億	Genetic Algorithm, Report, and Presentation

Keywords Reinforcement learning · Evolutionary learning · Self-Learning vehicle · Self-Driving vehicle

1 Introduction

Self-driving vehicles have garnered significant attention in recent years as a potential solution to the issue of human error in transportation. One contributing factor to this potential is the recent advancements in neural network technology, which enable self-driving vehicles to more effectively process and interpret data received from their sensors. Additionally, self-driving vehicles are equipped with obstacle avoidance capabilities, which further enhance their safety on the roads. In this work, we propose two self-learning paradigms utilizing reinforcement learning and genetic algorithms. Our model has been tested on custom designed maps using self-driving vehicles equipped with five lidar sensors, which provide the necessary information for the vehicle to navigate and make decisions. The aim of this research is to investigate the potential of self-driving vehicles to improve transportation safety.

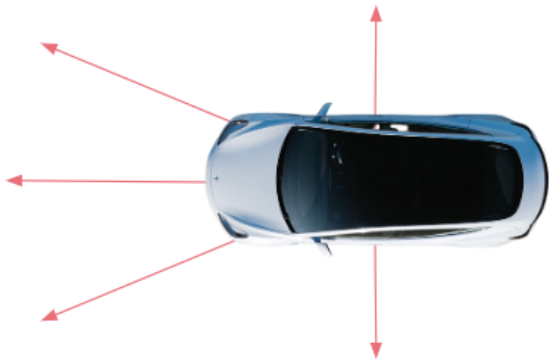


Figure 1: Five lidars located and the five angles(actions) the car can choose while driving

2 Method

2.1 Problem Definition

We consider the task in which our intelligent self-learning car interacts with an environment E, in a sequence of actions, observations and rewards. At each time step the intelligent car selects an action a_t from the set of legal driving actions, which is $A = A_1, A_2, A_3, A_4, A_5$. The action is one of the five angles, which is also the direction our five lidars will detect, showing at Figure 1. Since the car only observes images of the current map, we therefore consider sequences of actions and observations, $s_t = x_1, a_1, x_2, ..., a_{t-1}, x_t$, where x_t stores the position of our car and the information from our five lidars, further learning driving strategies that depend upon these sequences by the networks proposed in the Section 2.2, 2.3. The goal of our self-learning car is to interact with the environment by selecting driving actions in a way that maximises future rewards. We make the standard assumption that future rewards are discounted by a factor of γ per time-step, and define the future discounted return R_t at time t as $R_t = \sum_{t=1}^{N} \gamma^t r_t$, where t is the time-step.

2.2 Q Learning and Deep Q Networks(DQN)

Q learning[1] is a reinforcement learning algorithm that seeks to learn an optimal policy for an agent interacting with an environment through trial and error. It does this by estimating the expected future reward for each possible action the agent can take, referred to as the action-value function or Q function. The agent then selects the action with the highest estimated Q value at each time step. The Q function is updated using the Bellman equation [2], which takes into account the reward received for the previous action and the estimated future reward for the current action. This process continues until the Q function converges to the optimal policy, which maximizes the expected reward for the agent. Deep Q Network (DQN)[3] can be consider a type of neural network that is used to learn the optimal action to take in a given state, so as to maximize the expected reward, which is really similar to Q-learning. DQN uses a neural network to approximate the action-value function(Q function) and updates the network parameters through experience replay, which involves storing and sampling from a replay buffer of past experiences. This allows the agent to learn from past experiences without the need to continually interact with the environment, which can be computationally expensive. One key aspect of DQN is the use of experience replay, which allows the agent to learn from past experiences without the need to continually interact with the environment. This is useful because it can be computationally expensive to interact with the environment, and it also helps to stabilize the learning process by breaking the correlations between consecutive experiences. Another important aspect of DQN is the use of a target network, which is a separate copy of the action-value function that is used to compute the TD error. The target network is updated less frequently than the primary network, which helps to stabilize the learning process and improve the performance of the agent. The illustration of training a DQN will be shown at figure 2 and belowed algorithm 1

Algorithm 1 Deep Q Network (DQN) algorithm

while $t \neq EndingTime$ do

1. The car begins by interacting with the environment and observing the current state s_t.

2.The agent selects an action with the maximum reward, which is determined by the Q-function(DQN). The Q function is a function approached by neural networks that estimates the expected reward for taking a given action in a given state. $Action = \Phi^*(state) = argmax_a ctionQ(state, action)$

3. The agent takes the action and receives a reward R_t from the environment. It also observes the next state. $R_t = \sum_{t=1}^{N} \gamma^t r_t$

4. The agent stores the experience (state, action, reward, next state) in a replay buffer.

5. The agent samples a batch of experiences from the replay buffer and uses them to update the action-value function by minimizing the temporal difference (TD) error, which measures the difference between the estimated and actual reward for a given action in a given state. The TD error is used to compute the gradients that are used to update the network parameters.

$TD = \delta = Q(s_t, action) - R_t + \gamma\, Q(s_{t+1}, \Phi^*(s_{t+1}))$

6. The agent repeats the process until it has learned an optimal policy

end while

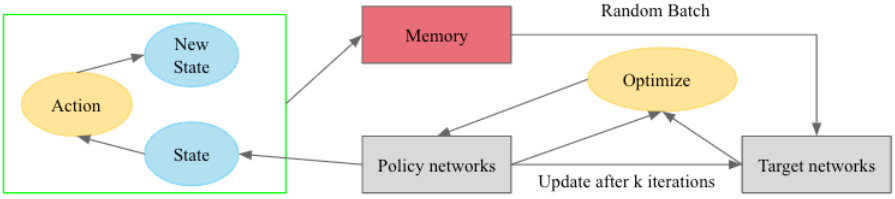


Figure 2: Deep Q learning (DQN) training loop

2.3 Genetic Algorithm

Genetic Algorithms[4][5] are a type of search algorithm inspired by the theory of Darwin' s natural evolution. They are used to find solutions to problems that are difficult to solve using traditional algorithms. It finds the solution by the process shown in Figure 3.

First, a population of candidate solutions is represented as a set of chromosomes. The chromosomes in the population are evolved over a series of generations, by applying genetic operators such as crossover, mutation, and selection [6].

During each generation, the genetic algorithm evaluates the fitness of each chromosome in the population, based on the pre-defined objective function of the chromosome. (In our problem we defined as driving distance) The chromosomes with the highest fitness values are more likely to be selected for reproduction, while those with lower fitness values are less likely to be selected.

Crossover involves combining the genetic material of two parent chromosomes to produce a child chromosome. Mutation is a random alteration of the genetic material in a chromosome, which can introduce new genetic diversity into the population.

The genetic algorithm continues to evolve the population until it reaches a satisfactory level of fitness, or until a pre-defined stopping condition is reached. The final population of chromosomes represents a set of candidate solutions that are likely to be near-optimal for the given optimization problem.

3 Experiment

We have performed experiments on both Q-learning algorithm and Genetic Algorithm across three different level of Maps, which are *Easy*, *Medium*, and *Hard*. We then discussed their corresponding training time, training results and the effects of using different loss function in Q-learning. We use the same network architecture, learning algorithm and hyperparameters settings across all experiments.

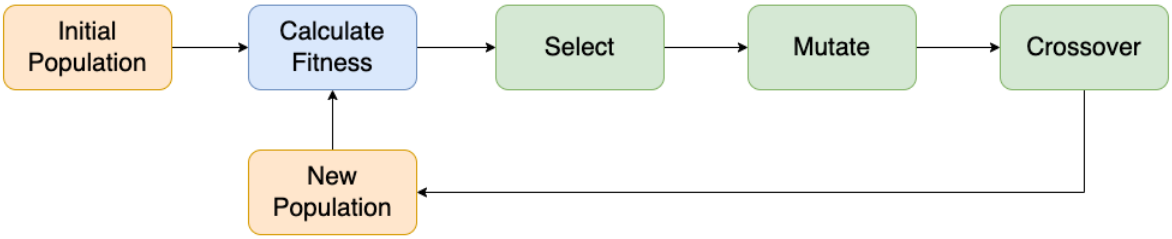


Figure 3: Genetic Algorithm training loop.

3.1 Map Configurations

Figure 4 illustrates the three different configurations of the map used in this study. The easy and medium maps are designed to be relatively simple, with the easy map only requiring right turns and the medium map requiring equal amounts of left and right turns. These two maps serve to test the basic capabilities of the model. The hard map, on the other hand, is designed to be as close to real-life scenarios as possible. To achieve this, we extracted a map of the surrounding area of Lane 118 (Lane 118, Section 2, Heping E Rd, Da' an District, Taipei, Taiwan) from Google Maps and selected the most significant roads for use in our map. This map aims to test the model's ability to handle more complex and realistic environments.

3.2 Q learning

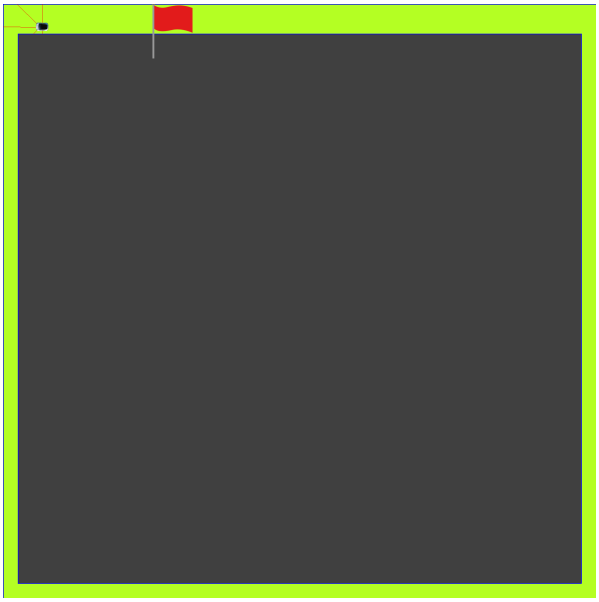
In these experiments, we will update the *target network* in every five iterations and our reward function will always remain the same. We fixed all positive rewards to be 1 and all negative rewards to be -1, where the negative rewards indicates that this action will collide with the wall or obstacles in the future. The behavior of choosing policy will also remain the same during training which was greedily choosing the action with the maximum future discounted return R_t at that time step t. We trained for a maximum epochs of 2000 and used a replay memory of 200 most recent frames.

3.2.1 Training time, stability and results

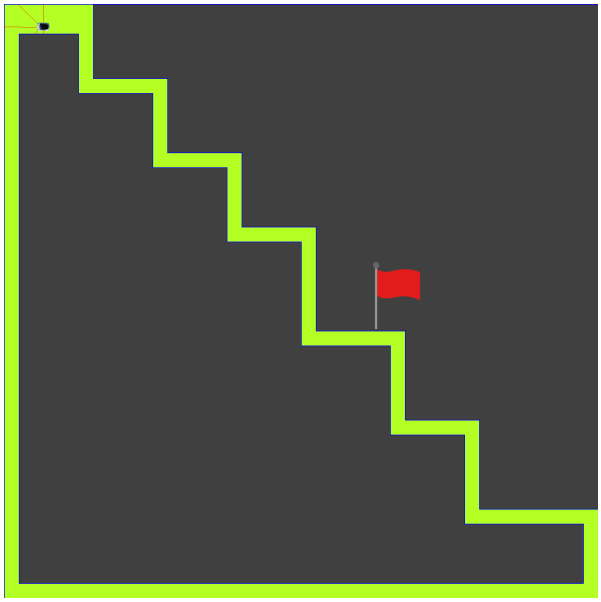
In reinforcement learning, precisely evaluating the progress of our intelligent driving car during training will be really challenging. We can see either in Figure 5, Figure 6 or Figure 7, the reward per training episode tends to be noisy because small changes to the weights of a choosing a policy can lead to large changes in the distribution of states the policy visits. Therefore, loss function and reward function will play a big role in reinforcement learning. Compared to mean squared error(MSE) loss, The Huber loss[7] acts like the mean squared error when the error is small, but like the mean absolute error when the error is large; therefore, this makes it more robust to outliers when the estimates of Q are very noisy. As a result, figure 5 indicates the fact that training with Huber loss can reach a higher result in a much complex environment, which is the *hard* map in our case. As see in figure 8, using Huber loss can successfully choose the right direction and finally reach the target point on the *hard* map. However, Huber loss didn't outperform MSE loss in the other two cases. From Figure 6 and Figure 7, we can deduce that the reason why training with MSE loss is better on *medium* and *easy* map is simply because both of the maps are less complex than the *hard* one, thus the model of Q-learning can converge faster. The training results from figure 9 and figure 10 shows that there is no need to change the directions several times, thus the information from lidars would be more useful for training Q model since it imply there will not occur collision in the future if the car is driving along a straight line. Furthermore, table 1 indicates that training time is longer while training with Huber loss than MSE and the training time is also proportional to the level of the map. In conclude, Huber loss is better in training with harder and more complex map but will be much more time-consuming than MSE; on the contrary, training with MSE will converge faster in the easier environment.

3.3 Gentic Algorithm

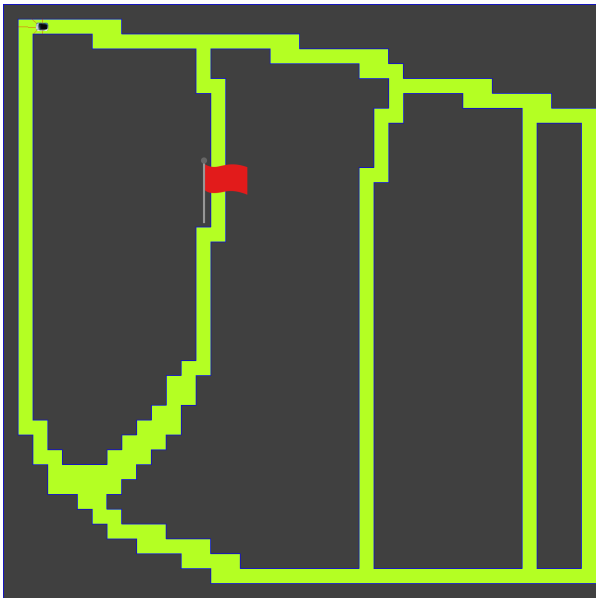
It is noteworthy to mention that various methodologies exist for the implementation of a genetic algorithm, and the specific details will be dependent on the complexity of the problem being addressed. The experiment was conducted on each level of the designed map mentioned in section 3.1, and a stopping point was estab-



(a) Easy Map



(b) Medium Map



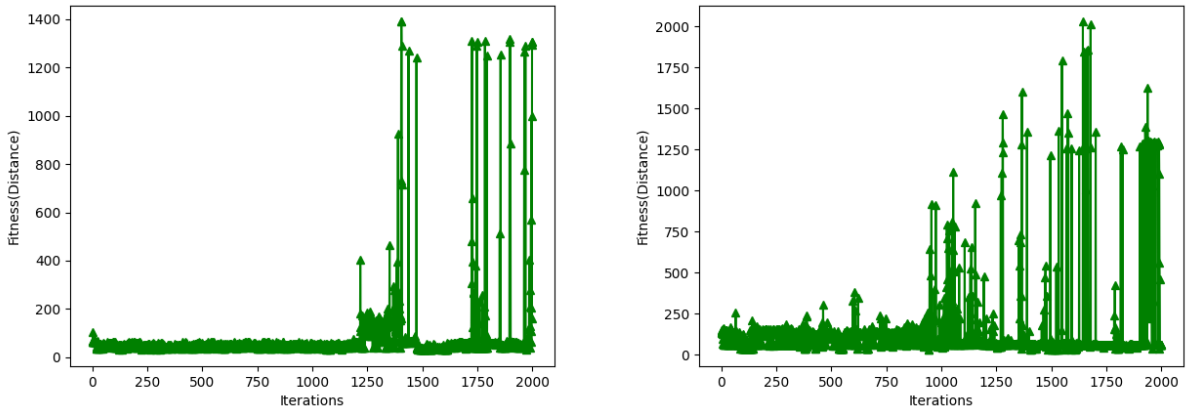
(c) Hard Map

(d) Extracted Google Map Region of Lane 118

Figure 4: Three different configurations of the map and their corresponding goals

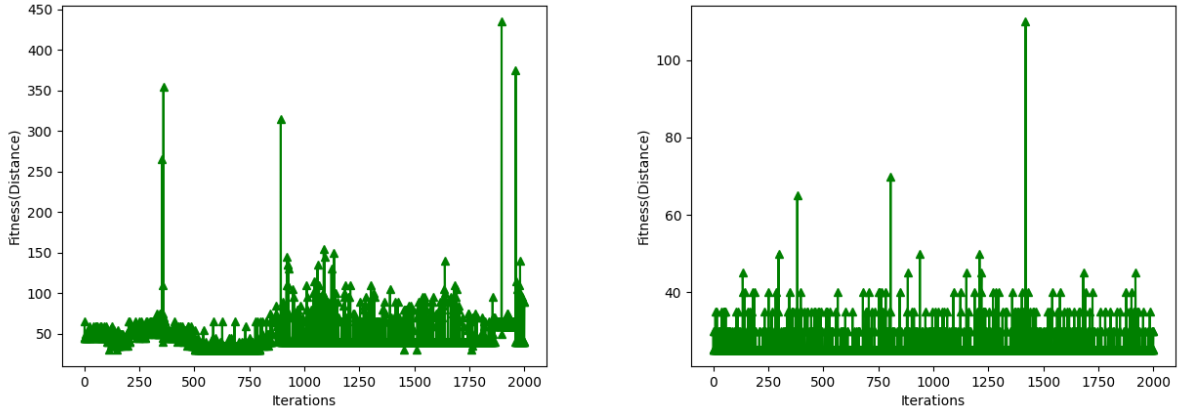
Table 1: The training time of different maps with different loss in Q-learning(*secs*)

Map	Huber loss	MSE loss
Hard	10597	9606
Medium	8807	7522
Easy	5763	298



(a) MSE Loss (b) HUBER Loss

Figure 5: The average reward per episode during training *hard* map on Huber loss and MSE loss



(a) MSE Loss (b) HUBER Loss

Figure 6: The average reward per episode during training *medium* map on Huber loss and MSE loss

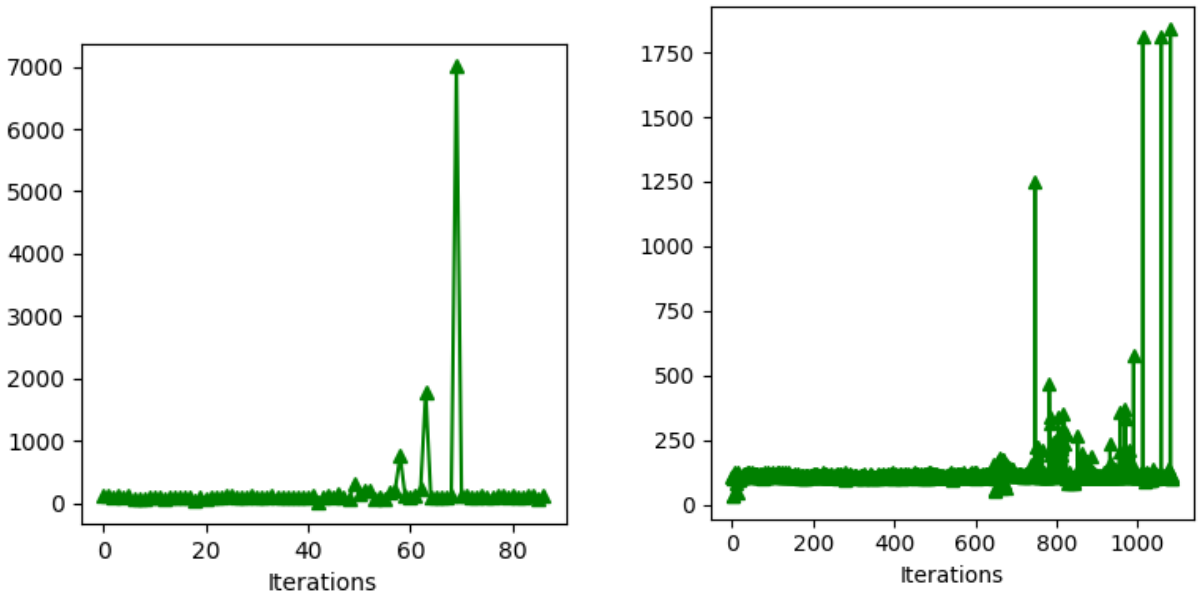
lished at either a fitness level of 10000 or after 200 generations, whichever occurred first. In this section, all other hyperparameters were kept constant while population size and mutation rate were varied in response to the various levels of difficulty present in the designed map.

3.3.1 Population size

The population size in a genetic algorithm refers to the number of individuals within the population. Our experiment was conducted with different population sizes of 10,35,50,100,200. As shown in Figure 11, it appears that the population size has a lower impact on simple problems such as the medium and easy maps.

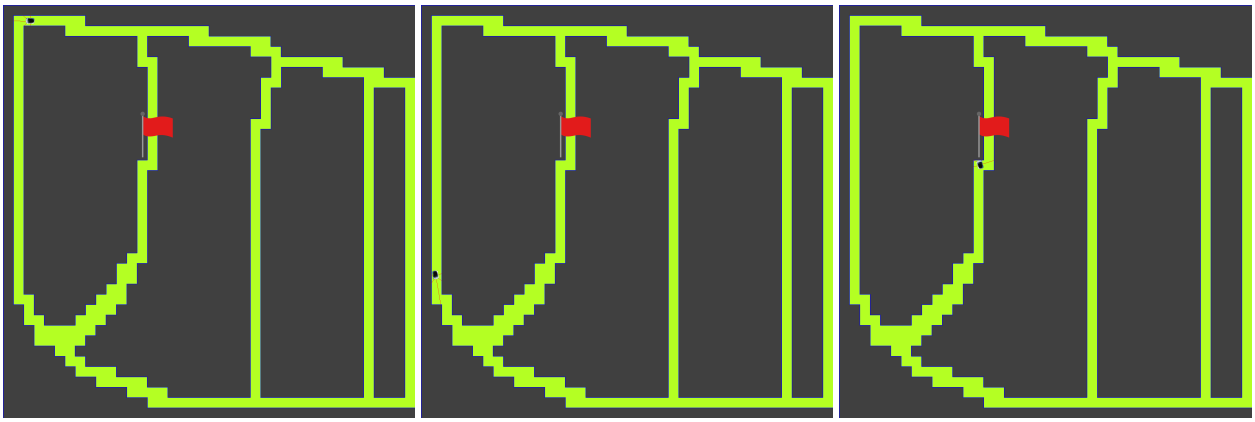
However, in the hard map, the problem is more complex, and a smaller population size may be unable to find the optimal solution within the limited number of generations. On the other hand, a larger population size can result in a more diverse set of solutions and increase the likelihood of finding a solution, which can be advantageous if the problem contains multiple local optima.

Nevertheless, a larger population size may also have disadvantages as it may take longer to converge on a satisfactory solution due to the increased number of individuals that need to be evaluated, as shown in Table 2. The generation time refers to the final generation in which one of the populations achieved the threshold of fitness. It can be observed that the time increases as the population size increases. Therefore,



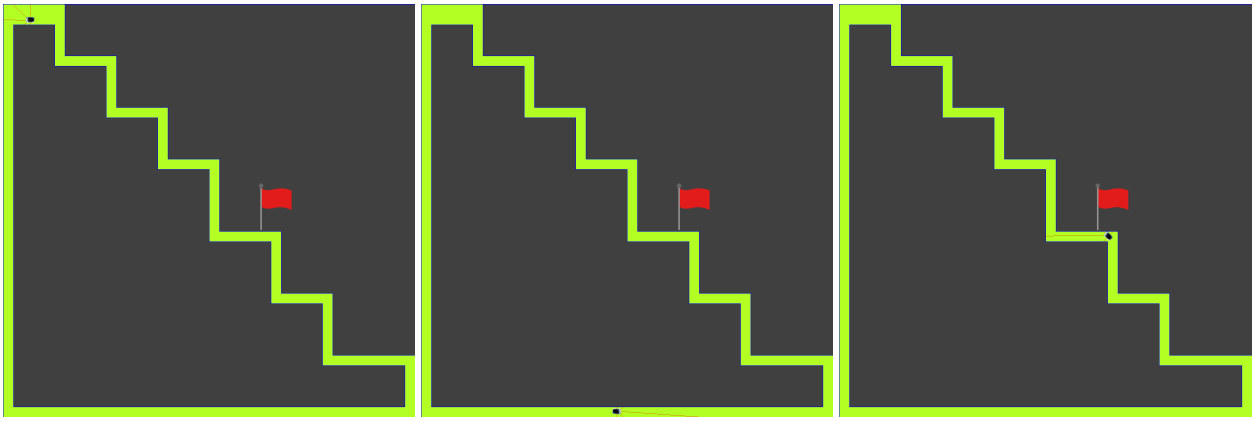
(a) MSE Loss (b) HUBER Loss

Figure 7: The average reward per episode during training *easy* map on Huber loss and MSE loss



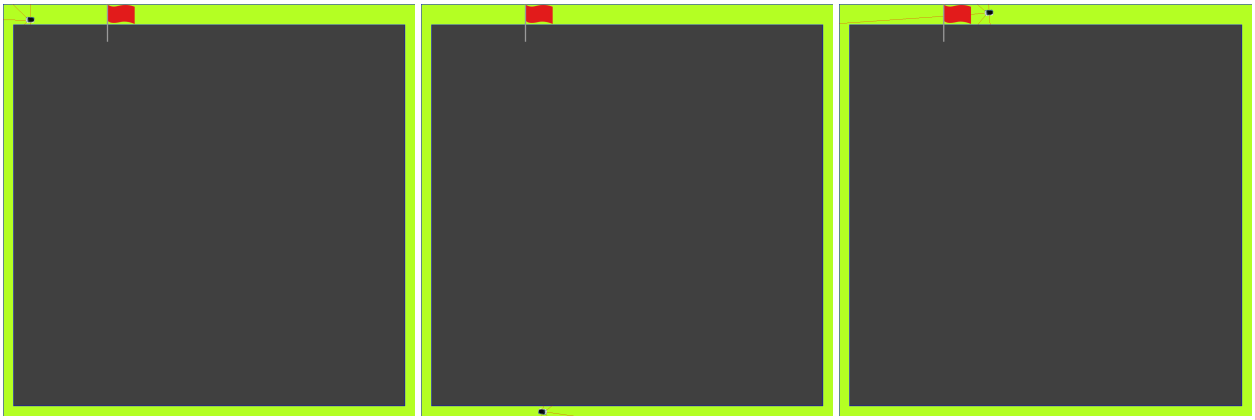
(a) Start (b) Middle (c) End

Figure 8: The training result of *hard* map with Huber loss



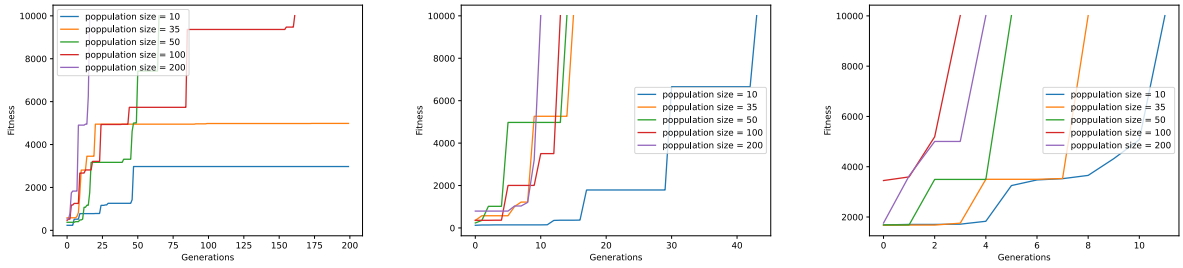
(a) Start (b) Middle (c) End

Figure 9: The training result of *medium* map with MSE loss



(a) Start (b) Middle (c) End

Figure 10: The training result of *easy* map with MSE loss



(a) Hard Map (b) Medium Map (c) Easy Map

Figure 11: Training the self-driving vehicles with different population size in genetic algorithm.

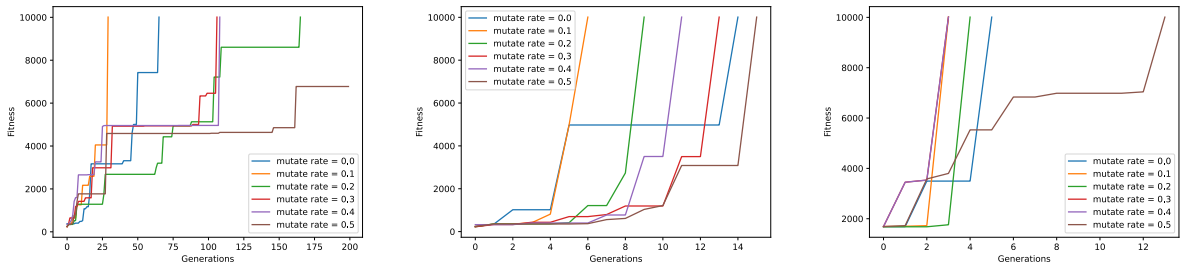
the optimal population size will depend on the specific characteristics of the problem we are trying to solve and the constraints of your computation resources. On medium and easy maps, a population size of 50 was sufficient to find a solution, while a population size of 100 or 200 was required for the hard map.

Table 2: The generation time of different maps with different population size. (*secs*)Where "-" denote that fitness can't achieve the threshold within 200 epochs.

Hard Map					
Population size	10	35	50	100	200
Generation time	-	-	-	148	248
Medium Map					
Population size	10	35	50	100	200
Generation time	8	20	25	42	95
Easy Map					
Population size	10	35	50	100	200
Generation time	9	13	16	24	43

3.3.2 Mutation rate

As shown in Figure 12, the mutation rate in a genetic algorithm determines the probability that mutation will occur during breeding. Mutation introduces random changes to the genetic material of an individual, which can help to introduce diversity into the population. This can be beneficial if the problem has multiple local optima, as it can allow the algorithm to escape from local optima and explore other regions of the search space.



(a) Hard Map (b) Medium Map (c) Easy Map

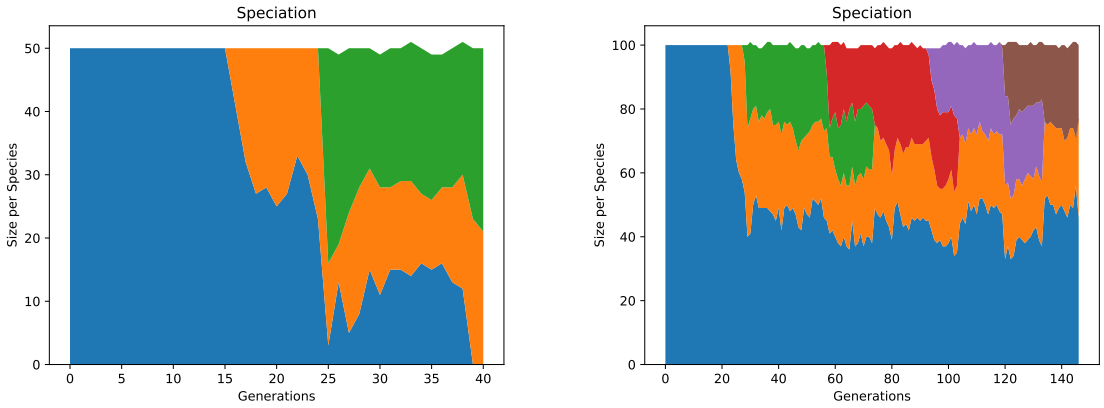
Figure 12: Training the self-driving vehicles with different mutation rate in genetic algorithm.

However, a high mutation rate can also make it more difficult to converge on a satisfactory solution because it introduces more randomness into the population, as shown in Figure 13. This can cause the algorithm to get stuck in suboptimal solutions or prevent it from finding the global optimum.

On the other hand, a low mutation rate can help to speed up convergence, but it may also result in a less diverse population. This can make it more difficult to escape from local optima and find the global optimum solution. In general, it's a trade-off between diversity and convergence. A higher mutation rate may be more likely to find the global optimum solution, but it may take longer to do so. A lower mutation rate may converge faster, but it may not be as likely to find the global optimum. Therefore, the optimal mutation rate will also depend on the specific characteristics of the problem you are trying to solve.

4 Conclusion

Based on the results of our experiments, we have found that both Q learning and genetic algorithms have the potential to improve transportation safety through the use of self-driving vehicles. We proposed two self-learning paradigms utilizing Q learning and genetic algorithms, which were tested on custom designed maps using self-driving vehicles equipped with five lidar sensors.



(a) Mutation rate = 0.4

(b) Mutation rate = 0.1

Figure 13: Species diversity changed over generations.

Our results show that the Q-learning algorithm can be used to effectively learn optimal driving policies, with the choice of loss function (MSE or Huber) being dependent on the complexity of the environment. We also found that the population size and mutation rate in genetic algorithms can affect the performance of the model, with larger population sizes and suitable mutation rates potentially leading to better solutions but taking longer to converge.

Overall, our results demonstrate the potential of self-driving vehicles to improve transportation safety through the use of advanced machine learning techniques. As a future direction, it would be interesting to investigate the incorporation of pedestrian avoidance capabilities into our self-driving vehicle model. This would involve designing a suitable reward function and training the model to recognize and respond appropriately to the presence of pedestrians on the road.

References

[1] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. Machine Learning, 8(3–4):279–292, May 1992.

[2] Kjetil Haugen. Stochastic Dynamic Programming. 04 2016.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[4] John H. Holland. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. 1992.

[5] J. Stender. Introduction to genetic algorithms. In IEE Colloquium on Applications of Genetic Algorithms, pages 1/1–1/4, 1994.

[6] Chen Lin. An adaptive genetic algorithm based on population diversity strategy. In 2009 Third International Conference on Genetic and Evolutionary Computing, pages 93–96, 2009.

[7] Peter J. Huber. Robust Estimation of a Location Parameter. The Annals of Mathematical Statistics, 35(1):73 – 101, 1964.