Université Gustave Eiffel

Mathematical Modeling for Intelligent Systems

FENNI Hassen FREDJ Najeh

M2 SIA

Supervised by : Farhi Nadir

2020/2021

Contents

Introduction 3

1 Project framework **4**

1.1 Objectives 4

1.2 Tools and technologies 4

1. Eclipse SUMO 4
2. Netedit 4
3. TraCI 4
4. PyCharm 5

2 Realisation **5**

2.1 User guide 5

2.2 Variables & functions 5

2.3 Q learning algorithm 6

1. Q-table 6
2. Implementation of the algorithm 7

3 Results **8**

3.1 High density 8

3.2 Low density 12

INTRODUCTION

Traffic simulation or the simulation of transportation systems is the mathematical modeling of transportation systems through the application of computer software help plan, design, and operate transportation systems. It started over forty years ago, and it is an important area of discipline in traffic engineering and transportation planning today.

Simulation in transportation is significant because it can study models too complicated for analytical or numerical treatment, can be used for experimental studies, can study detailed relations that might be lost in analytical or numerical treatment and can produce attractive visual demonstrations of present and future scenarios.

1 Project framework

1.1 Objectives

The aim of this project is to apply the algorithm of Q-learning to learn by numerical simulation the process of driving (car-following) on a ring road of one lane. We consider a circular road of one lane, where a given number of cars run without passing.

We will be simulating the dynamics of all cars, except one, which we control with the Q-learning algorithm.

1.2 Tools and technologies

1.2.1 Eclipse SUMO:

Eclipse SUMO (Simulation of Urban Mobility), an open source, highly portable, microscopic and continuous multi-modal traffic simulation package designed to handle large networks.

SUMO allows modelling of intermodal traffic systems including road vehicles, public transport and pedestrians. Included with SUMO is a wealth of supporting tools which handle tasks such as route finding, visualization, network import and emission calculation. SUMO can be enhanced with custom models and provides various APIs to remotely control the simulation.

1.2.2 Netedit:

Netedit is a Road network editor for the road traffic simulation in SUMO. Using this utility users can quickly design road networks and obtain Network xml file which is part of SUMO configuration.

1.2.3 TraCI

TraCI is the short term for "**Tra**ffic **C**ontrol **I**nterface". Giving access to a running road traffic simulation, it allows to retrieve values of simulated objects and to manipulate their behavior "on-line".

1.2.4 PyCharm

PyCharm is an integrated development environment used for programming in Python. It allows code analysis and contains a graphical debugger. It also allows the management of unit tests, the integration of version management software, and supports web development with Django.

2 Realization

2.1 User guide

The project should be ran through the command line from the project's folder, we give the ability to specify 2 parameters:

1. -nc = Number of cars, this parameter ranges from 2 to 50.
2. -e = Number of episodes, this parameter ranges from 1 to 100, with each episode corresponding to a thousand simulation steps.

Exemple : main.py -nc 50 –e 100.

Note :
The car controlled by the Q-learning algorithm is represented in **Red** while other cars controlled by SUMO are represented in **Yellow**.

2.2 Variables & functions

We have three main state variables:

- The speed of the controlled car, we fetch this value through traci at each simulation step using the following command: **traci.vehicle.setSpeed(controlled_car_id, speed)**
- The relative speed discretized of the leader car
- The space-headway to the leader car.

We have one controlled variable:

- Acceleration of the controlled vehicle

To take full control of our simulation we used Traci functions:

- traci.simulationStep(controlled_car_id) to perform one step in the simulation.
- traci.vehicle.setSpeedMode(controlled_car_id, mode)to set the speed.
- traci.vehicle.getLeader(controlled_car_id, dist=0.0) returns the leader car id on the edge.
- traci.vehicle.getPosition(controlled_car_id) to return position coordinates of the car .
- vehicle.getRoadID(controlled_car_id) to get the current edge of the car.

2.3 Q learning algorithm

Q-learning is a model-free reinforcement learning algorithm to learn quality of actions telling an agent what action to take under what circumstances. "Q" names the function that the algorithm computes with the maximum expected rewards for an action taken in a given state.

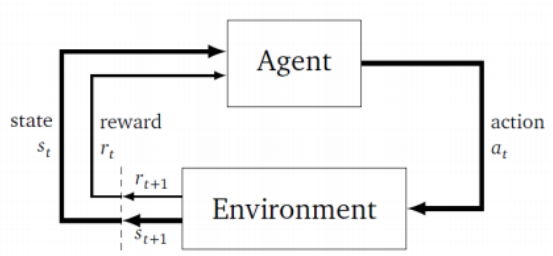


Figure: Schematic figure for reinforcement learning

2.3.1 Q-table:

Q-Table is a lookup table where we calculate the maximum expected future rewards for an action at each state. Basically, this table will guide us to the best action at each state.

Scenarios:

In our case we can accelerate by increasing the car's speed by 1 m/s, and decelerate by decreasing it by 1 m/s.

At every step of the simulation, the car has constant accelerations and decelerations of +/- 1 m/s².

Discretization:

The state variables are discretized into 6 ranges as follows:

- Space headway: 0 to 150 m.
- Relative speed: -30 to 30 km/h.
- Speed: 0 to 50 km/h.

Control variable:

- Action: accelerate, decelerate, none

The Q-table is thus a 4-dimensional array of size: 6 * 6 * 6 * 3 = 648

2.3.2 Implementation of the algorithm:

Hyperparameter optimization:

- The discount factor γ = 0.99,
- The learning rate α = 0.1,
- The epsilon greedy probability ϵ = 0.1.

The discount factor determines the importance of future rewards. A factor of 0 will make the agent "myopic" by only considering current rewards, while a factor approaching 1 will make it strive for a long-term high reward. In our case, we are looking to maximize the car's speed without causing accidents, to achieve this, the car must continuously strive for long term rewards. Since the car cannot perform an emergency break but instead decelerates discrete steps, it must anticipate accidents multiple simulation steps ahead. Therefore, we will set the discount 0.99.

The learning rate or step size determines to what extent newly acquired information overrides old information. Since we are in a continuous learning process without a terminal condition (as opposed to the mountain car seen in class where reaching the flag corresponds to achieving our goal and the end of an episode), for this reason and the same reasons stated above, we will set the learning to 0.1 to maintain old information.

The greedy policy hyperparameter will be set to 0.1 in order to maximize the exploitation rate of our algorithm from the get-go, this choice has proven to give us the best results, as setting it to a higher value would cause the algorithm to require much more simulation steps to achieve convergence. In addition, each 1000 simulation steps, epsilon is decayed by 10%, giving up the need for exploration in favor of the need for exploitation.

.

Reward mechanism

Coming up with a good solution for the reward mechanism that should be implemented has not been an easy task. As a matter of fact, we had to undergo an iterative thought process of trial and error. The reward mechanism we have decided to implement has the following rules:

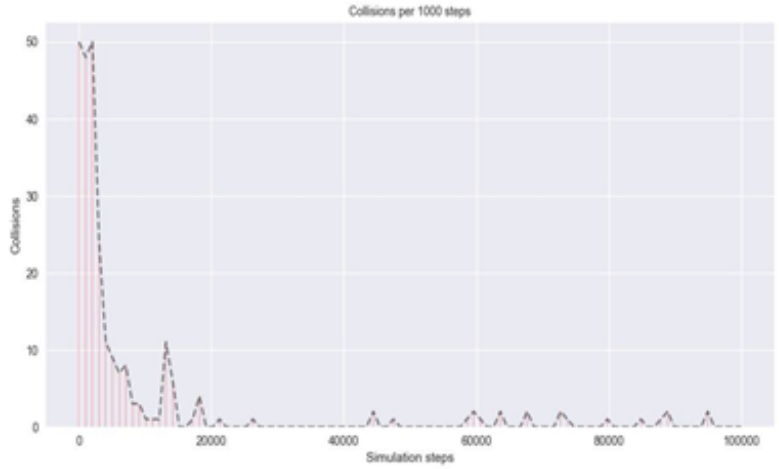
- The reward is deduced by 10 if the car causes a collision.
- The reward is increased/decreased by “scoring” value ranging -1 and 1 at every iteration

Note: The scoring value is calculated based on the space headway and the distance between the controlled car and the one following it, its job is to encourage the car to keep a safe distance between itself and both cars surrounding it and penalize if it does not do so. This forces the car to adapt its speed to the speeds of other vehicles in SUMO and further decreases the number collisions because the car is now forced to plan its decelerations multiple steps ahead of a collision.

3 Results

3.1 High density

For a density of 50 cars and one hundred thousand simulation steps, we get the following results:

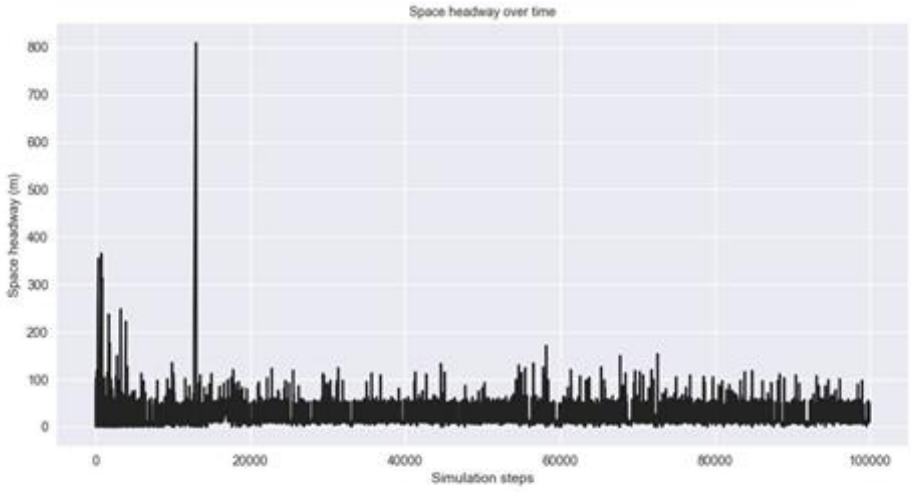


Collisions

Steps	Collisions	Total
1000	50	50
2000	48	98
3000	50	148
4000	24	172
5000	11	183
6000	9	192
7000	7	199
8000	8	207
9000	3	210
10000	3	213
15000	6	233
20000	0	238
25000	0	239
30000	0	240
35000	0	240
40000	0	240
45000	2	242
50000	0	243
60000	2	246
70000	0	251
80000	1	255
90000	0	259
100000	0	261

Figure: Collisions

The number of collisions converges after around 15000 simulation steps, collisions nearly never happen for the rest of the simulation.

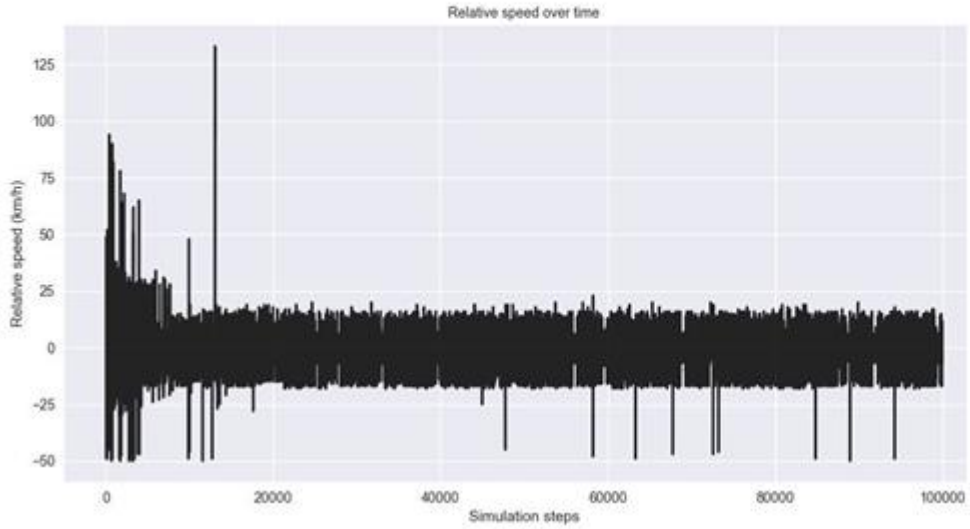


Space headway (m)

Steps	Mean	Std
1000	107.28	115.17
2000	46.19	51.63
3000	20.37	21.84
4000	47.62	61.69
5000	18.76	15.06
6000	13.69	13.88
7000	11.96	11.47
8000	14.48	12.92
9000	14.13	13.12
10000	24.14	23.11
15000	26.05	14.99
20000	17.65	13.66
25000	20.27	13.87
30000	22.57	15.25
35000	19.56	13.1
40000	15.29	10.94
45000	20.67	15.54
50000	16.17	12.24
60000	15.77	12.23
70000	19.43	14.16
80000	19.7	14.11
90000	15.56	11.71
100000	16.12	12.1

Figure: Space headway(m) over time

The space headway quickly converges to values between 0 and 50 meters with a low standard deviation. As the simulation evolves, values oscillate as they tend towards a lower mean and standard deviation overall in a relatively slow manner.

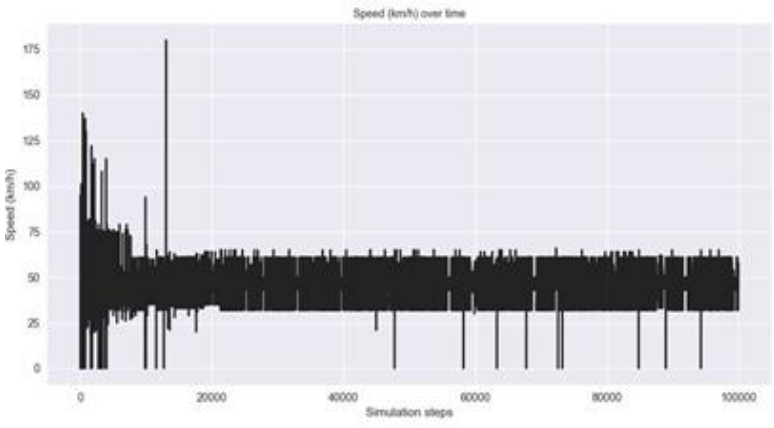


Relative speed (km/h)

Steps	Mean	Std
1000	0.87	21.44
2000	0.28	14.97
3000	0.82	11.75
4000	0.13	14.53
5000	0.16	8.58
6000	0.15	6.5
7000	0.04	5.43
8000	-0.04	6.7
9000	0.08	5.39
10000	-0.21	8.94
15000	0.1	7.82
20000	0.04	6.38
25000	-0.04	6.27
30000	-0.07	6.8
35000	0.04	6.17
40000	-0.04	5.02
45000	0.05	6.9
50000	0.1	5.28
60000	0.03	5.53
70000	-0.08	6.16
80000	-0.06	6.26
90000	0.02	5.26
100000	-0.0	5.04

Figure: Relative speed over time

The relative speed converges to values ranging from -20 to 20 km/h. The mean at the end of the simulation is 0 with a standard deviation of 5.04 km/h. This shows that the controlled car has successfully adapted its speed to the speeds of the other cars within the simulation.



Speed (km/h)

Steps	Mean	Std
1000	48.21	21.47
2000	47.43	14.97
3000	47.98	11.78
4000	47.26	14.75
5000	47.45	8.36
6000	47.49	6.23
7000	47.32	5.22
8000	47.2	6.56
9000	47.27	5.15
10000	47.0	8.75
15000	47.33	7.71
20000	47.42	6.14
25000	47.44	6.07
30000	47.36	6.56
35000	47.37	6.0
40000	47.31	4.83
45000	47.38	6.74
50000	47.42	5.16
60000	47.44	5.25
70000	47.21	5.94
80000	47.29	6.16
90000	47.3	5.14
100000	47.29	4.72

Figure: Speed over time

As the car learns, its speed quickly converges to lower values, this is explained by the fact that it is adapting itself to the other cars within our simulation, which have lower maximum speeds than the controlled car. The speed converges to values ranging from 40 to 60 km/h.

We can notice that the mean speed across the simulation is almost constant at around 50km/h, however, the standard deviation is getting better over time and slowly tends to zero.

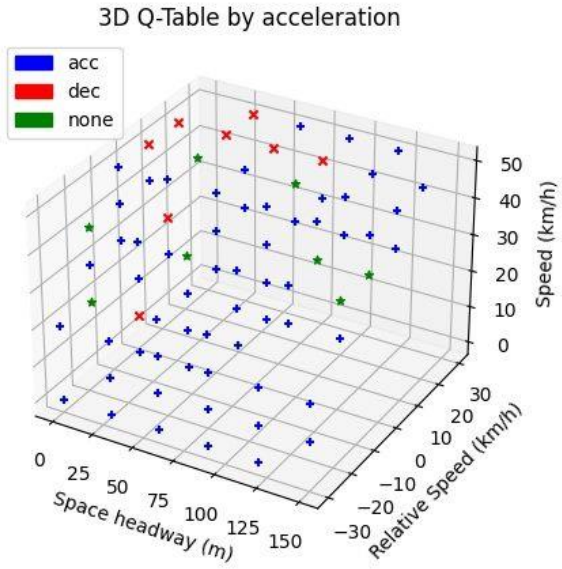
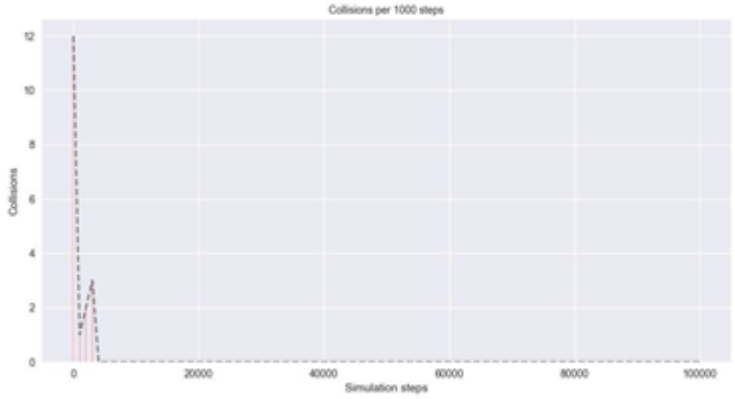


Figure: Q-Table group by acceleration

The Q table shows that the car's strategy is to:
accelerate whenever possible
decelerate at low space headways, negative relative speeds and speeds close to 50km/h
keep constant speed around 50km/h, around relative speeds of 0km/h, and high space headways

3.2 Low density

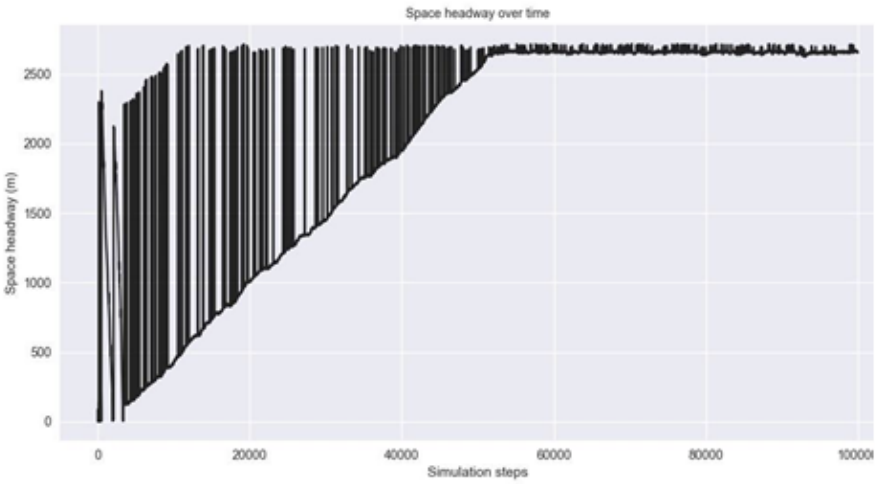


Collisions

Steps	Collisions	Total
1000	12	12
2000	1	13
3000	2	15
4000	3	18
5000	0	18
6000	0	18
7000	0	18
8000	0	18
9000	0	18
10000	0	18
15000	0	18
20000	0	18
25000	0	18
30000	0	18
35000	0	18
40000	0	18
45000	0	18
50000	0	18
60000	0	18
70000	0	18
80000	0	18
90000	0	18
100000	0	18

Figure: Collisions

The number of collisions converges after around 4000 simulation steps, the convergence happen much faster than in the case of high density.

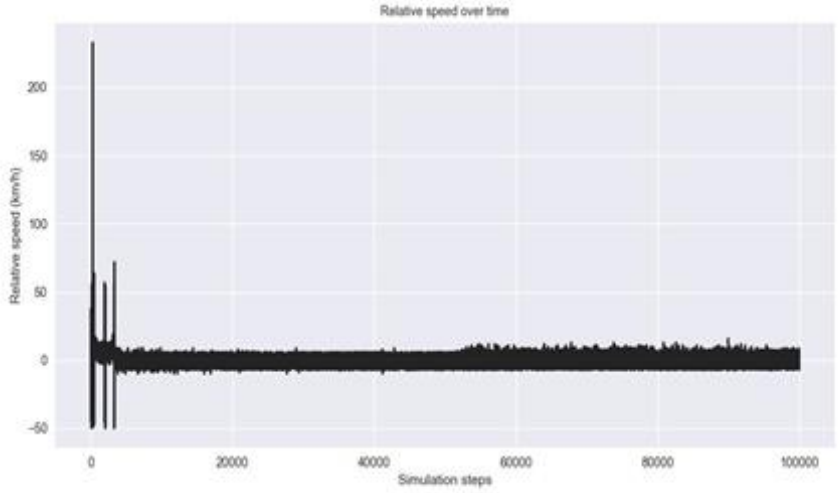


Space headway (m)

Steps	Mean	Std
1000	1369.13	900.19
2000	760.4	388.82
3000	1409.86	528.09
4000	223.58	209.28
5000	164.63	136.61
6000	215.4	137.26
7000	267.09	138.7
8000	310.45	138.33
9000	361.31	169.22
10000	404.76	119.25
15000	713.97	89.1
20000	997.35	133.54
25000	1223.61	82.23
30000	1435.86	57.36
35000	1739.64	34.11
40000	1924.88	56.21
45000	2276.51	33.66
50000	2508.28	14.77
60000	2657.87	6.95
70000	2656.53	7.0
80000	2647.35	9.14
90000	2658.07	6.13
100000	2661.0	4.86

Figure: Space headway (m) over time

The space headway increases in a linear fashion and converge to value of 2650 m, after around 60000 time steps.

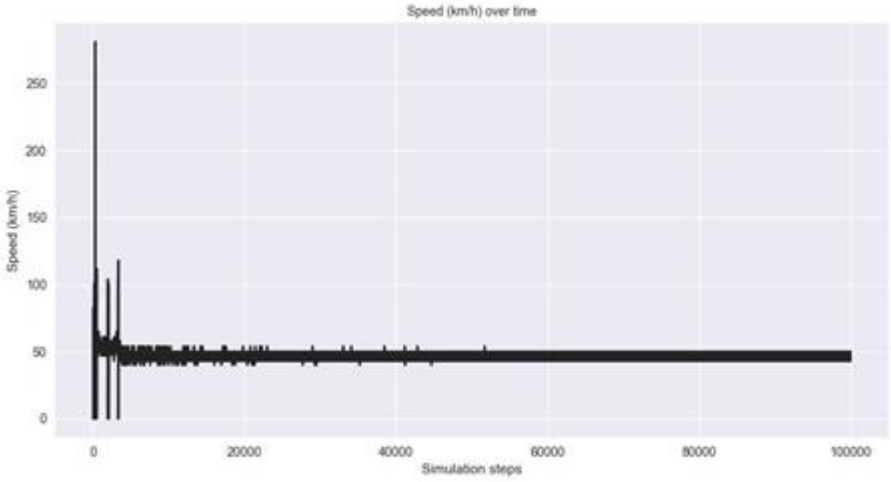


Relative speed (km/h)

Steps	Mean	Std
1000	11.63	36.33
2000	4.93	6.05
3000	5.21	7.88
4000	2.85	10.42
5000	-0.19	2.98
6000	-0.18	2.99
7000	-0.16	3.0
8000	-0.15	3.0
9000	-0.23	3.0
10000	-0.17	2.93
15000	-0.2	2.92
20000	-0.24	2.94
25000	-0.2	2.98
30000	-0.12	3.0
35000	-0.13	2.93
40000	-0.17	2.98
45000	-0.25	2.95
50000	-0.12	2.98
60000	-0.06	3.17
70000	-0.01	3.17
80000	0.05	3.19
90000	0.01	3.24
100000	0.01	3.22

Figure: Relative speed over time

The relative speed converges to values ranging from -5 to 5 km/h with low standard deviation of around 3km/h.



Speed (km/h)

Steps	Mean	Std
1000	58.79	36.45
2000	52.05	6.05
3000	52.43	7.75
4000	50.21	10.29
5000	47.15	2.52
6000	47.11	2.51
7000	47.08	2.53
8000	47.17	2.48
9000	47.14	2.49
10000	47.19	2.47
15000	47.17	2.42
20000	47.08	2.43
25000	47.12	2.42
30000	47.17	2.43
35000	47.18	2.42
40000	47.13	2.42
45000	47.04	2.45
50000	47.25	2.38
60000	46.76	2.49
70000	46.86	2.47
80000	47.02	2.45
90000	46.78	2.48
100000	46.65	2.5

Figure: Speed over time

The speed quickly converges to a value of around 50km/h, with little standard deviation.

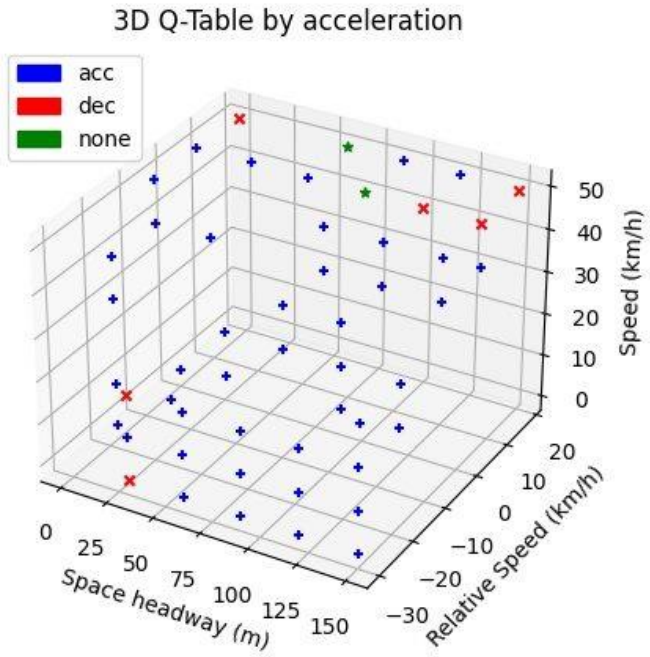


Figure: Q-Table group by acceleration

The Q table shows that the car's strategy is to:
Accelerate whenever possible
Decelerate at low space headways, negative relative speeds and speeds close to 50km/h
Keep constant speed around 50km/h, around relative speeds of 0km/h, and high space headways