Tactical Decision-Making for Heavy Vehicles using Deep Reinforcement Learning

Computer Science – Algorithms, Languages and Logic
Systems, Control and Mechatronics

MEHMET ATAKAN SERIN
TÕNIS SOODLA

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020:EENX30

Tactical Decision-Making for Heavy Vehicles using Deep Reinforcement Learning

MEHMET ATAKAN SERIN

TÕNIS SOODLA

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2020

Tactical Decision-Making for Heavy Vehicles using Deep Reinforcement Learning
MEHMET ATAKAN SERIN
TÕNIS SOODLA

Supervisors: Sébastien Gros, Department of Electrical Engineering
Carl-Johan Hoel, Volvo Group
Oliver Sundell, Volvo Group
Examiner: Sébastien Gros, Department of Electrical Engineering

Master's Thesis 2020:NN
Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A truck on a road near the sea.
(Adobe Stock Photos, code: 97479896, Standard Licence)

Typeset in LaTeX

Gothenburg, Sweden 2020

Tactical Decision-Making for Heavy Vehicles using Deep Reinforcement Learning
Mehmet Atakan Serin
Tõnis Soodla
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Tactical decision-making for autonomous vehicles is a hard task to solve using classical methods while retaining a general approach with minimal changes to the algorithm when presenting a novel scenario or a situation. Deep reinforcement learning methods can be more general than classical methods which have the potential to relieve this issue. Therefore, deep reinforcement learning methods are investigated in this thesis for autonomous heavy vehicles in urban scenarios such as T-junctions and roundabouts. Furthermore, in the T-junction scenario, a custom truck path that compensates for the inward off-tracking phenomenon of the trailer and occlusions is implemented. These scenarios are modelled as a Markov Decision Process (MDP) in a simulation framework that consists of environments and softwares named Simulation of Urban Mobility, OpenAI Gym, Keras-rl and TensorFlow. Deep Q-Network and Dueling Double Deep Q-Network algorithms with varying network architectures are trained and different MDP formulations are examined. The Deep Reinforcement Learning methods are compared with a classical baseline model, called Time-To-Collision, and the Deep Reinforcement learning algorithms solved approximately the same number of scenarios. However, in situations where the best Deep Reinforcement learning algorithms were on par with or better than the baseline model, they had on average 1.61 times faster completion times.

Keywords: Decision-Making, Deep Reinforcement Learning, POMDP, MDP, autonomous driving, Deep Q-Network, machine learning, SUMO, urban scenarios.

Acknowledgements

We are grateful for the opportunity given to us by Volvo Group having a chance to work on this very exciting topic. Secondly, we want to thank our Volvo supervisors Carl-Johan Hoel and Oliver Sundell who were always ready to discuss the finer details of this thesis. Furthermore, we want to thank our Chalmers supervisor Sébastien Gros with whom we consulted quite often through Skype. In addition, we want to thank the Alfred Ots Scholarship Fund which greatly supported the studies of one author. Last but not least, we want to thank our families and friends for support and encouragement.

Mehmet Atakan Serin and Tõnis Soodla, Gothenburg, January 2020

Abbreviations

DDDQN	**D**ueling **D**ouble **D**eep **Q**-**N**etwork
DQN	**D**eep **Q**- **N**etwork
IDM	**I**ntelligent **D**river **M**odel
MDP	**M**arkov **D**ecision **P**rocess
MOBIL	**M**inimize **O**verall **B**raking **I**nduced by **L**ane changes
POMDP	**P**artially **O**bservable **M**arkov **D**ecision **P**rocess
ReLU	**Re**ctified **L**inear **U**nit
RL	**R**einforcement **L**earning
SUMO	**S**imulation of **U**rban **Mo**bility
TraCI	**Tra**ffic **C**ontrol **I**nterface
TTC	**T**ime **T**o **C**ollision

Contents

List of Figures **xi**

List of Tables **xiii**

1 Introduction **1**
1.1 Background . . . 1
1.2 Related Work . . . 3
1.3 Objective . . . 4
1.4 Limitations . . . 5
1.5 Thesis Outline . . . 5

2 Theory **7**
2.1 Decision-Making under Uncertainty . . . 7
2.2 Markov Decision Process . . . 8
2.3 Partially Observable Markov Decision Process . . . 8
2.4 Reinforcement Learning . . . 9
2.5 Reinforcement Learning on a Markov Decision Process . . . 9
2.6 Dynamic Programming . . . 11
2.7 Q-learning . . . 12
2.8 Exploration and Exploitation . . . 12
2.9 Deep Q-Network . . . 13
2.10 Double Deep Q-Network . . . 14
2.11 Deep Q-Learning with Dueling Architectures . . . 15

3 Methodology **17**
3.1 Tactical Decision-Making in Urban Scenarios . . . 17
3.1.1 Tactical Decision-Making in a T-junction . . . 17
3.1.2 Tactical Decision Making in a Roundabout . . . 18
3.2 Formulation of the Markov Decision Process . . . 19
3.2.1 State Space . . . 19
3.2.2 Action Space . . . 23
3.2.3 Reward Function . . . 24
3.3 Design choices for solving the Markov Decision Process with Deep Reinforcement Learning . . . 25
3.4 Comparing Deep Q-Networks algorithms with the baseline model . . . 27
3.5 Implementation . . . 27
3.5.1 Simulation of Urban Mobility . . . 28

3.5.2 Trailer Modelling and a Custom Path Model 31
3.5.3 Occlusion Modelling 31
3.5.4 Implementation of the Training Scenarios 34
3.5.5 Implementation of a Baseline Model 34
3.5.6 Deep Reinforcement Learning Framework - Keras-RL and OpenAI Gym 37

4 Results **39**
4.1 Non-occluded T-Junction Scenario 39
4.2 Occluded T-Junction Scenario 42
4.3 Roundabout Scenario 45
4.4 Model trained on the T-junction and the Roundabout scenario 45
4.5 Comparison of Action Spaces 46
4.6 Comparison of Different Neural Network Architectures 47

5 Discussion **49**
5.1 Deep Q-Network compared to Dueling Double Q-Network 49
5.2 Neural Network Size 49
5.3 Action Spaces 50
5.4 Behavioral Analysis Between Different Methods 51
5.5 Deep Reinforcement Learning Algorithm in the Roundabout Scenario 51
5.6 Deep Reinforcement Learning Algorithm in Both T-junction and Roundabout Scenarios 55
5.7 Safety Issues 55

6 Conclusion and Future Work **57**
6.1 Strengths and Weaknesses of Different Models 57
6.2 Formulating Urban Scenarios for Tactical Decision-making as a Partially Observable Markov Decision Process 57
6.3 Future Work 58

A Appendix 1 **I**
A.1 Full Training Info I

List of Figures

1.1 A T-junction scenario 2

2.1 The interaction mechanism between the agent and the environment . 7
2.2 Reinforcement learning diagram showing the interaction between the agent and the environment 9

3.1 A T-junction scenario with an autonomous agent in front of an unregulated crossing and other vehicles on the main road 17
3.2 A T-junction scenario with occlusions 18
3.3 A roundabout scenario with an autonomous agent in front of an unregulated roundabout and other vehicles either entering, passing or exiting the roundabout 19
3.4 Visualization of the state space consisting of 49 elements, the upper-right block with diagonal violet stripes is not used 20
3.5 A schematic showing the violet ghost vehicles 22
3.6 Different neural network architectures used while training the DQN models (k is the number of actions in the action space) 26
3.7 The simulation setup showing the flow of information through SUMO, OpenAI Gym, Keras-rl and Tensorflow 28
3.8 The dimensions of the SUMO map and the routes of the vehicles in the T-junction scenario 28
3.9 A T-junction Scenario in SUMO showing the starting and the ending position of the ego vehicle 29
3.10 The dimensions, flows and the route for the Ego vehicle in the roundabout scenario 30
3.11 The implemented custom path for the truck imitating a truck making a right turn on a narrow intersection 31
3.12 The placement of the 12x36 occlusion grid in the simulation shown by the pink rectangle 32
3.13 An example of the occlusion modelling in the simulation highlighting the ghost vehicles shown as violet rectangles 33
3.14 The Time-To-Collision Method for the T-junction scenario emphasizing the red line which marks the line of collision for the Ego and other vehicles 36
3.15 The Time-To-Collision Method for the roundabout scenario emphasizing the red line which marks the line of collision for the Ego and other vehicles 37

4.1 Results for six deep learning models in the non-occluded T-junction scenario . . . 40
4.2 Combined results for six deep learning models in the non-occluded T-junction scenario . . . 41
4.3 Results for six deep learning models in the occluded T-junction scenario 43
4.4 Combined results for six deep learning models in the occluded T-junction scenario . . . 44
4.5 A comparison between the Setspeeds and the Accelerations action spaces using the DDDQN algorithm . . . 46
4.6 A comparison between different networks using the DDDQN algorithm 47

5.1 TTC vs. the best reinforcement learning model (5) in the T-Junction scenario, time between frames is 2 seconds, TTC corresponds to the green truck and the reinforcement learning model corresponds to the yellow truck . . . 53
5.2 A reinforcement learning model in the roundabout scenario compared to TTC, time between frames is 5 seconds, TTC corresponds to the green truck, the reinforcement learning model corresponds to the yellow truck . . . 54

A.1 Training data for algorithm 1 . . . I
A.2 Training data for algorithm 2 . . . II
A.3 Training data for algorithm 3 . . . II
A.4 Training data for algorithm 4 . . . III
A.5 Training data for algorithm 5 . . . III
A.6 Training data for algorithm 6 . . . IV
A.7 SUMO route files used for the T-Junction scenario . . . V
A.7 SUMO route files used for the T-Junction scenario (cont.) . . . VI
A.7 SUMO route files used for the T-Junction scenario (cont.) . . . VII
A.7 SUMO route files used for the T-Junction scenario (cont.) . . . VIII
A.8 SUMO route files used for the roundabout scenario . . . IX
A.8 SUMO route files used for the roundabout scenario (cont.) . . . X

List of Tables

3.1 Description of the state space concerning the ego vehicle 20
3.2 Description of the state space concerning the other vehicles 21
3.3 Description of the state space concerning the ghost vehicles 21
3.4 A speed-based action space 23
3.5 An acceleration based action space 24
3.6 The parameters used for training the DQN models 25

4.1 Comparison between different models in the T-Junction scenario without occlusions on the validation set 41
4.2 Comparison between different models in the T-Junction scenario without occlusions on the test set 42
4.3 Comparison between different models in the occluded T-Junction scenario on the validation set 44
4.4 Comparison between different models in the occluded T-Junction scenario on the test set 45
4.5 Comparison between TTC and the DDDQN algorithm in the roundabout scenario 45
4.6 Comparison between TTC and the DDDQN algorithm in the T-junction and the roundabout scenario 46

1
Introduction

This chapter introduces the concept of tactical decision-making in the context of autonomous vehicles. Secondly, it provides an introduction to the relevant research in form of literature and papers used in this thesis. To conclude, it declares the objectives, states the limitations, and shows the outline of the work.

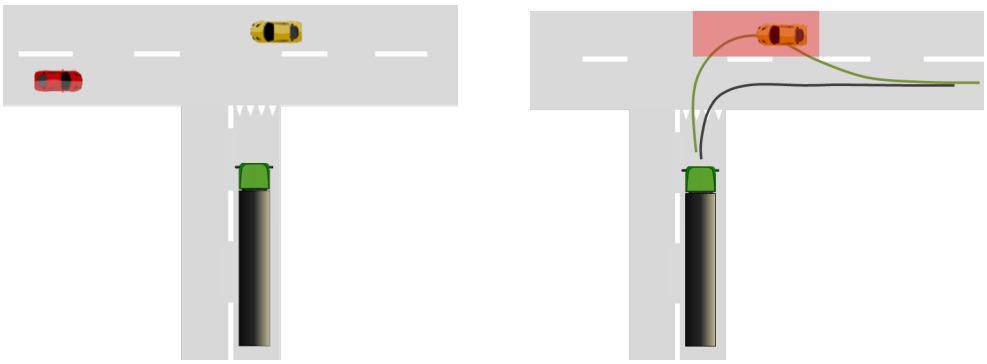
1.1 Background

In autonomous driving, decision-making can be divided into three basic layers which consist of strategical, tactical and trajectory planning decisions. These three layers are distinguished according to their frequency of making with a decision. Strategic decision making is the one with lowest frequency and is responsible of high level decisions such as determining the route when travelling from point A to point B. Trajectory planning makes decisions in a very high frequency and controls the vehicle in a very low level, comprising decisions as choosing the steering angle. Tactical decision-making lies in the middle of strategical decision-making and trajectory planning. It operates in middle level frequency when compared to other layers of decision-making [1]. Tactical decision-making is a challenge for autonomous vehicles on public roads. This problem refers to selecting the best possible vehicle action that satisfies tactical and strategical driving goals. These actions are high-level characterisations of vehicle motion, such as accelerating or stopping. In urban environments, it can decide on complex abstractions, such as when to enter a roundabout or which velocity profile to keep. For human drivers, this task is not time-consuming and is executed in small time intervals. If decisions are made correctly, traffic can flow, otherwise vital damages can occur. For this reason, implementing a good tactical decision-maker can play a significant role in the autonomous driving field.

To illustrate our task in an urban environment for an articulated heavy combination vehicle, consider the scenario given in Figure 1.1. Here the automatically controlled subject vehicle (the long vehicle) is entering an uncontrolled T-junction to turn right, giving way and avoiding collisions. Due to the length of the vehicle, it must follow an unorthodox trajectory to stay on the road. This might result in entering neighboring lanes possibly with an accident. Since the subject of this thesis is a truck with a trailer as a trainable agent, it can not behave as mobile as a regular passenger vehicle. Compulsory intervention to neighboring lanes must be handled successfully. Furthermore, the vehicle must decide on actions that are related to the acceleration profile, also following the planned path. Otherwise stopping in the

middle of an active intersection or accelerating towards another vehicle could cause collisions.

A major challenge for the manoeuvre planning problem is the incomplete and noisy perception of other vehicles causing uncertainty about how the traffic will evolve. Intentions of third-party drivers are unclear thus their future behaviors are hard to predict, but could still be included in the decision-making. One way to solve this task is to use the methods from the subdomain of machine learning called reinforcement learning to find an optimal policy.



(a) An uncontrolled intersection.

(b) A right turn scenario with two possible paths: lower (black) path for the short vehicles, and the above (green) path for the truck.

Figure 1.1: A T-junction scenario

This thesis has possible positive outcomes for the society in the following United Nations sustainability goals [2]:

- 3. Good health and well-being
- 8. Decent work and economic growth
- 9. Industry, innovation and infrastructure
- 11. Sustainable cities and communities

As over 90% of accidents in traffic happen due to human error [3] and according to the World Health Organization - there are more than 1.2 million road traffic deaths annually [4]. Making vehicles autonomous can decrease the amount of human error in traffic and thus save a substantial number of lives every year. It is estimated that for trucks in the logistics sector innovation can be created through decreasing labor costs by lowering the number of drivers. Eliminating or decreasing these kinds of labor costs can create huge amounts of economic growth for society. Making vehicles such as trucks autonomous can have a remarkable effect on innovation. When transporting goods, these kinds of vehicles would not be bound to sleeping schedules but could move all the time excluding maintenance, loading and fueling times. This can make the logistics sector much more effective which will have a positive effect on all other sectors of industry.

With the trend of urbanisation, the problem of excess traffic in cities is relevant as ever. As for sustainability, there are two striking statistics with this. Firstly, a civilian car is driven most often just by one person, even though it could fit more and be more energy-efficient. Secondly, cars in private ownership are not in use about 9/10 of the time [5]. The arrival of autonomous vehicles can optimise these problems by giving people the incentive not to own a car at all and use them more efficiently. The possible negative consequences are that this thesis work can cause people currently driving trucks and other vehicles as and occupation to lose their jobs with the arrival of autonomous vehicles. Secondly, with the mass testing of these vehicles there can be fatalities, but this will be potentially overweighted with the number of lives saved minimising the human error in traffic.

1.2 Related Work

There exist methods that can successfully execute tactical decision-making by using rule-based heuristics such as the Intelligent Driver Model (IDM) for car following, Minimizing Overall Braking Induced by Lane Change (MOBIL) for lane changing, and time-to-collision (TTC) for urban, and highway, scenarios [6, 7, 8]. There has been a need for an approach which would be more general and could take corner cases into account. One approach could be using Deep Reinforcement Learning methods which for example have shown promise with the Deep Q-Network (DQN) algorithm when reaching and surpassing human performance in several Atari 2600 games [9]. In addition, more complex Deep Reinforcement Learning algorithms that combine neural networks and Monte Carlo Tree Search (MCTS) have defeated the European and the World Champion in the Game of Go. The algorithm managed to develop strategies that were unknown to expert human players [10]. With this in mind, many researchers have attempted to integrate MCTS or DQN based methods into the automated driving field. These methods are frequently compared to baseline methods such as TTC, IDM and MOBIL. MCTS methods have yielded promising results as an online planning method on urban scenarios such as T-junctions [11], [12]. Compared to MCTS, DQN methods makes most of the computations in an offline manner. After the offline training phase, the DQN method decides on the action rather quickly making it eligible for real-time purposes [13].

Various modifications to the architectures of the neural networks used within the DQN algorithm have been made to compare the results with the basic DQN. Previous work has suggested using a novel convolutional neural network architecture in a highway lane changing scenario. It has shown better performance than a classic fully connected neural network [14]. Another neural network comparison has been made in a scenario of vehicles negotiating at a four way intersection. Recurrent neural networks were shown to be better learners than a feed-forward neural network [15]. A roundabout merging scenario was studied with drivers having 'aggressive' or 'defensive' intents [16]. In a similar study, a T-junction scenario with two more additional intentions of 'turning' or 'going straight' was investigated [17]. The intents of the vehicles in traffic were not revealed and kept partially observable. In both studies, the agent learned to infer the hidden driver intentions and acted suitably.

As tactical decision-making can encompass various actions, the effect of the choice of base actions on the results of learning a reasonable policy has been investigated. Several action spaces were compared in an occluded T-junction scenario where the action space called 'Creep N Go' stood out with remarkable results [18]. The DQN agent utilised the 'Creep' action where it slowly drove into the intersection by eliminating occlusions and gaining more vision in a controlled manner. Results were better than the TTC method in the terms of the time of execution but worse in the collision percentage metric. A compact semantic space was proposed to adapt the same state space to various scenarios. The same state space was both used in a highway merging and a highway scenario [19]. Li and Czarnecki [20], experimented with a state space that was both used in several T-junctions and a roundabout. It also included higher-level observations such as the TTC value, brake-turn signals, and the right of way. Vehicles were labeled relative to the ego vehicle as 'merge', 'ahead', 'crossing', 'irrelevant'. In a study by Isele, Rahimi, Cosgun, Subramanian, Fujimura [18], occlusions were put into the state variable as a grid world to give the agent a notion of observing uncertainty. Nearly all of the spaces investigated include the position and velocity information.
One of the aims of this thesis is to focus on the DQN algorithm and its modifications and learn how they perform in the field of autonomous driving in urban scenarios with a novelty of using a heavy vehicle while implementing a custom path at least in one scenario.

1.3 Objective

This section will highlight the purpose and the research questions of the thesis. The objective of this thesis is to evaluate strengths and weaknesses of applying reinforcement learning for the tactical decision-making for heavy vehicles in urban situations. Secondly, a specific goal is to experiment in a simulation with two urban environment scenarios: a T-junction and a Roundabout scenario. Furthermore, the proposed research questions for this thesis are:

1. What are the strengths and weaknesses of using a state-of-the-art deep reinforcement learning algorithm such as the Dueling Double Deep Q-Network (DDDQN) in simulation for tactical decision-making in heavy vehicles in a few urban scenarios compared to using a classical model such as the time-to-collision (TTC) approach? This should be compared taking into account performance measures such as the minimisation of collisions and the execution time of decisions.

2. How should the tactical decision-making in an urban scenario be formulated as a Partially Observble Markov Decision Process (POMDP)? For example, what should be the state variables in simulation to achieve as realistic observations as possible, but at the same time maintaining a reasonable training efficiency? Likewise, how should the action-space and the reward function be designed to maximise the performance?

Another important parameter for our task is to determine the agent's perception of the environment. This is referred to as the 'state' in the decision-making field. Determining the state variables has two aspects for our task. One is to make it realistic enough so that the ego-vehicle does not have access to unrealistic information that can never be gathered in real life. The other aspect is to pick those observations in an efficient manner that will shorten the time for training the agent. Since not every piece of information in the simulation will be given to our agent, the environment is considered partially observable.

1.4 Limitations

The known limitations for the thesis are:

- The algorithms are applied or trained on a single simulated truck. This truck is referred to as the ego vehicle.
- Urban scenarios such as T-junctions and roundabouts are considered for this thesis. Traffic lights are not taken into account.
- The roads are assumed to be flat and incremental declines are discarded.
- Even though the number of collisions is a criterion of success, safety is not the primary focus of this thesis. It is assumed that safety standards are satisfied by an external system; see [21].
- Sensor fusion is out of this thesis' scope. Observations consist of high-level state descriptions with "perfect" sensors.
- Other vehicles are designed not to respond to the ego vehicle in order to create a worst-case scenario for the ego vehicle.
- The ego vehicle is assumed to be following a pre-planned path and will not be moving out of this path.

1.5 Thesis Outline

The thesis will be outlined the following way:

- **Chapter 2: Theory** explains the building blocks of reinforcement learning method alongside with Deep Reinforcement Learning methods. The second part shows the principles of the time-to-collision model which will be used as a baseline method.

- **Chapter 3: Methodology** shows how different urban scenarios can be formulated as an MDP and how they can be implemented within a framework consisting of Simulation of Urban Mobility (SUMO), OpenAI Gym, Keras-RL, and Tensorflow softwares.

- **Chapter 4: Results** presents how different reinforcement learning algorithms solve the scenarios compared to a base model.

- **Chapter 5: Discussion and Analysis** demonstrates the analysis of the results from the Results chapter.

- **Chapter 6: Conclusion and Future Work** emphasises how the objectives of the thesis go together with the results and discusses the possible future work which could improve the approach.

2
Theory

This chapter explains the idea of solving the issue of making tactical decisions giving the theoretical background for modelling the problem as a Partially Observable Markov Decision Process and solving it with reinforcement learning methods leading up to to the motivation of the used methodology.

2.1 Decision-Making under Uncertainty

Uncertainty exists in many areas of real-life problems from wildfire management to collision avoidance systems. For uncertain systems, it is not possible to gather a complete picture of how the environment behaves. Not knowing the environment response while taking into regard multiple objectives of the problem can be very challenging. *The agent* is the main subject of decision-making problem and it tries to make the best decision under this uncertain environment. The agent and the environment communicate with each other with *observations* and *actions*. More specifically the environment provides an observation to the agent and the agent takes an action on the environment. The agent's task here is to decide which action to take given the observations accomplishing the success criteria of the problem. There are many approaches and algorithms developed to solve such problems [22]. In this thesis, the reinforcement learning approach is explored further in the following sections.

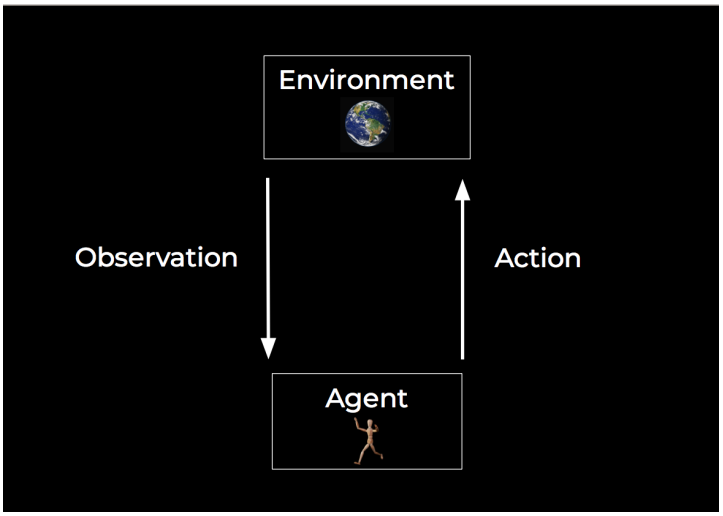


Figure 2.1: The interaction mechanism between the agent and the environment

2.2 Markov Decision Process

A Markov decision process is a mathematical framework for applying reinforcement learning approaches. In MDP, the *state*, s_t refers to the information of the environment. The agent receives a reward, r_t from the environment and it takes an action, a_t in the environment. Formally, a Markov Decision Process (MDP) can be formulated as a tuple, $< \mathcal{S}, \mathcal{A}, R, T, \gamma >$, consisting of the following [23]:

- $\mathcal{S}$ is the set of states
- $\mathcal{A}$ is the set of actions
- R is the reward function
- T is the transition model providing: $P(s_{t+1} \mid s_t, a_t)$ which is the probability of transitioning from s_t to s_{t+1} via action a_t at time t.
- γ is the discount factor defined in $[0, 1]$.

Markov Decision Processes assumes the 'Markov Property'. According to this property future state, s_{t+1} is solely dependent on present action, a_t and present state, s_t regardless of the previous states and previous actions [13]:

$$P(s_{t+1}|a_t, s_t, a_{t-1}, s_{t-1}, ..., a_0, s_0) = P(s_{t+1}|a_t, s_t). \tag{2.1}$$

According to the equation above, T is managed with the Markov property. In other words, the stochastic transition model behaves as memoryless and the whole history is compactly represented in the present time step.

2.3 Partially Observable Markov Decision Process

Partially Observable Markov Decision Process (POMDP), shown with the tuple $< \mathcal{S}, \mathcal{A}, R, T, O, Z, \gamma >$, is a version of MDP, but the states are not fully provided to the agent by the environment. In addition to MDP's stochastic transition model, POMDP also has a stochastic observation model O and observation space Z. O is a distribution over states given the observation and Z is the set defining all the possible observations. In POMDP, the state of the environment is not provided to the agent directly, but rather it can be inferred from observation. This adds an extra layer of uncertainty to the MDP [24].

A driver who makes a tactical decision in a T-Junction might not be able to observe all the features that exist in the reality of the environment. For example, in real life, most of the sensors include noise and have limitations for sensing over certain distances. Furthermore, driver intentions are not given. These ambiguities make the problem partially observable. However, by discarding the observation model, the POMDP can be expressed as an MDP. Rather than trying to infer the environments' state from these observations, every observation is treated as an actual state. Such an approach simplifies the framework and keeps it free from an observation model.

For this reason, the main framework for this thesis is an MDP, even though the problem being partially observable.

2.4 Reinforcement Learning

Reinforcement learning (RL) is a machine learning approach that has its roots in the psychology of learning. By giving a reward signal to an animal or a human, it is possible to condition a desired behaviour. The computational conjugate of this idea is widely applied for shaping a controller's behaviour without explicitly programming it.

Reinforcement learning is an approach for decision-making under uncertainty. Yet there is an additional *reward* signal playing a key role. That is because the main objective is to maximize the total reward in the long run. To achieve this task, the agent has a certain perception of the environment and an interaction mechanism with the environment. Firstly, an observation and a reward signal are provided to the agent by the environment. The agent interacts with the environment by applying actions on the environment. Each action leads to another observation and a reward signal. By introducing a notion of observations and an associated reward, the agent learns to choose an optimized *action sequence* that will yield in collecting the maximum amount of the cumulative reward [13].

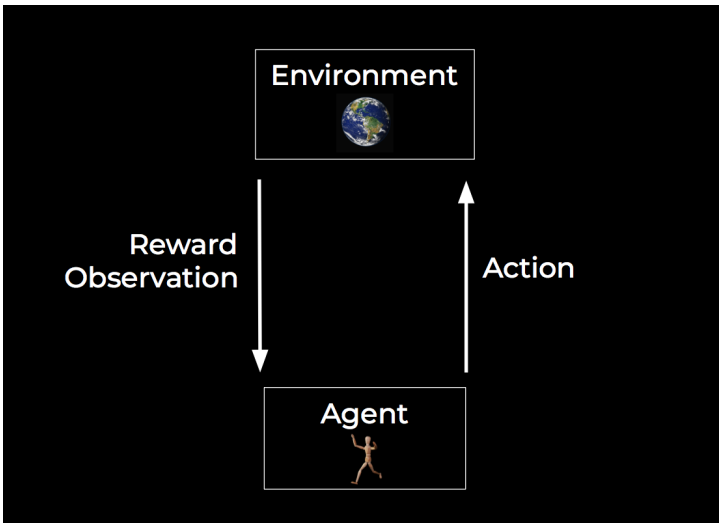


Figure 2.2: Reinforcement learning diagram showing the interaction between the agent and the environment

2.5 Reinforcement Learning on a Markov Decision Process

In this thesis, the environment is set by a Markov Decision Process and the agent is the subject interacting with the MDP. As the agent traverses the MDP environment,

it makes an observation and obtains a state, s_t from $\mathcal{S}$. At the same time step, it takes an action, a_t from $\mathcal{A}$. The transition model T recognizes a_t and directs the agent to the next state, s_{t+1} with a probability of $P(s_{t+1}|a_t, s_t)$. In this transition, the agent is granted a reward r that is calculated via a reward function R. This process can be shown as a *sequence* of states, actions and rewards until the agent reaches terminal state s_T:

$$(s_0 a_0 r_0), (s_1 a_1 r_1), (s_2 a_2 r_2) ... (s_T a_T r_T). \tag{2.2}$$

Each sequence is characterized by a policy telling the agent how to act in each state. Formally, a *policy* π is a stochastic mapping from states to actions. The agent's objective is to follow a sequence which will maximize the expected *return* $\mathbb{E}(G_t)$. Expected return is expressed as expected sum of cumulative rewards with a discount factor γ:

$$\mathbb{E}(G_t) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k}\right]. \tag{2.3}$$

Return can be a simple measurement of how valuable it is to be in a state or how beneficial to perform a specific action in a state. *Value functions* are responsible for these kind of measurements. There are two value functions: the state-value and the action-value function. The state-value function is denoted as $V_\pi(s)$ calculating the expected return of a given state under a certain policy:

$$V_\pi(s_t) = \mathbb{E}_\pi\left[G_t|s_t\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t\right]. \tag{2.4}$$

The state-value function is mostly used for dynamic programming approaches and various reinforcement learning algorithms. There is another function called the action value-function which also considers actions for calculations. The action-value function is denoted as $Q_\pi(s, a)$ and it results in the expected return given a state and an action under a certain policy:

$$Q_\pi(s_t, a_t) = \mathbb{E}_\pi\left[G_t|s_t, a_t\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t, a_t\right]. \tag{2.5}$$

In the end of the optimization process, the agent converges to a *policy* telling the agent how to act in each state. If the converged policy is better than the other possible policies, it said to be an optimal policy π^*. Thus, if the agent follows the sequence dictated by π^*, the expected return is greater than the other possible policies' returns. Optimal policies also have optimal value functions $V_{\pi^*}(s)$ and $Q_{\pi^*}(s, a)$, which have higher values than other policies' value functions for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$:

$$V_*(s) = \max_\pi V_\pi(s), \quad \forall s \in \mathcal{S}, \tag{2.6}$$

$$Q_*(s,a) = \max_{\pi} Q_\pi(s,a), \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}. \tag{2.7}$$

The current state's return value is also included in the next state's return value. For this reason the result of the optimal value function can be expressed as recursive equations for a specific s_t input or a (s_t, a_t) pair input. These equations are called the Bellman optimality equations and play a key role in most of the reinforcement learning algorithms:

$$V_*(s) = \max_{a} \sum_{s'} P\left(s', |s, a\right) \left[r + \gamma V_*\left(s'\right)\right], \quad \forall s, s' \in \mathcal{S}, \tag{2.8}$$

$$Q_*(s,a) = \sum_{s'} P\left(s'|s, a\right) \left[r + \gamma \max_{a'} Q_*\left(s', a'\right)\right], \quad \forall s, s' \in \mathcal{S}, \forall a, a' \in \mathcal{A}. \tag{2.9}$$

where s' are states that can be transitioned from s via using a' and r is the obtained reward during the transition.

2.6 Dynamic Programming

Dynamic Programming is a collection of methods consisting of algorithms for determining an optimal policy for an MDP. The term itself was created by Richard Bellman [25], but here it is looked upon in the context of reinforcement learning. Thus, it is a model-based algorithm inspired by Dynamic Programming and it needs a transition model for the execution. Dynamic programming starts with the *prediction* phase called the *Policy Evaluation*. In this phase return values of each state are calculated by using the state-value under a certain policy. In each iteration i, the state-value function is updated by using transition probabilities. This process, called the *Bellman Update* is as follows:

$$V_{i+1}(s) \rightarrow \sum_{a} \pi(a|s) \sum_{s'} P\left(s'|s, a\right) \left[r + \gamma V_i\left(s'\right)\right], \quad \forall s, s' \in \mathcal{S}, \forall a \in \mathcal{A}. \tag{2.10}$$

Policy evaluation measures how well an agent performs under a certain policy. In order to come up with optimal policy, *Policy Iteration* is used. It utilizes the Bellman Optimality Equation for finding the actions maximizing the state-value function:

$$\pi(s) \rightarrow \arg\max_{a} \sum_{s'} P\left(s'|s, a\right) \left[r + \gamma V\left(s'\right)\right] \quad \forall s, s' \in \mathcal{S}, \forall a \in \mathcal{A}. \tag{2.11}$$

By using policy evaluation and policy iteration collaboratively the agent obtains the optimal policy and the state-value function.

An algorithm called *Value Iteration* eliminates the need of policy evaluation and works directly on Bellman Optimality Equation (2.8) as an update:

$$V_{i+1}(s) \rightarrow \max_a \sum_{s'} P\left(s', |s, a\right) \left[r + \gamma V_i\left(s'\right)\right] \quad \forall s, s' \in \mathcal{S}, \forall a \in \mathcal{A}. \tag{2.12}$$

By this way, the optimal state-value function is obtained. In the end, one policy iteration step is applied to obtain actions that are maximizing the optimal value function. Even though the value iteration algorithm excludes the policy evaluation step, the transition model is obligatory for both algorithms to work. Additionally traversing through each state calculating the actions yielding a maximized return is guaranteed but not computationally feasible. Since the complexity is exponential with respect to the sets $\mathcal{S}$ and $\mathcal{A}$, these algorithms might take too much time for convergence. In the next section, the Q-learning method is investigated which overcomes some of the negative aspects of Dynamic Programming.

2.7 Q-learning

Q-Learning is an off-policy and model-free reinforcement learning technique for controlling the agent. Unlike dynamic programming, Q-learning does not need a transition model. It rather estimates the optimal action-value function from raw experiences making the convergence faster than dynamic programming [13]. The agent picks the best action dictated by the policy derived from state-action function by using an exploration policy (the exploration topic is further described in Chapter 2.8). Then, the agent transitions to a new state and observes the reward, r_{t+1} and next state, s_{t+1}. This transition is called *experience* and is represented by the tuple $(s_t,a_t,r_{t+1},s_{t+1})$:

$$Q\left(s_t, a_t\right) \rightarrow Q\left(s_t, a_t\right) + \alpha \left[r_t + \gamma \max_a Q\left(s_{t+1}, a\right) - Q\left(s_t, a_t\right)\right], \quad \alpha \in [0, 1), \gamma \in [0, 1], \tag{2.13}$$

where α is the learning rate and γ is the discount factor.

The Q-learning utilizes the *target* $[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})]$, which is another expression of return calculated by using the next state and action. The algorithm iterates through the difference $[r_t + \gamma \max_a Q\left(s_{t+1}, a\right) - Q\left(s_t, a_t\right)]$ with a learning rate of α to obtain a good approximation of q^*. Therefore the optimal policy, π^*, can be approximated.

2.8 Exploration and Exploitation

In the previous section, it is mentioned that most of the time the agent takes the action dictated by the policy of an action-value function. This phenomenon is referred to as *exploitation* and these exploiting actions are referred to as *greedy* actions. By doing this the agent mostly works with the same actions and states that are tried out in the past which led to rewarding sequences. Exploitation helps the agent to converge and focus polishing the policy to its best version. Furthermore, it can

prevent the agent from discovering new paths and have a restricted vision of the environment. The opposite practice, *exploration*, is to try out new actions that are not necessarily shown to be successful. It provokes the agent to discover more about the environment and find sequences which may lead to even more return. Since picking the best actions and not necessarily picking the best actions are contradicting commands for the agent, a dilemma named the exploration and exploitation dilemma arises. There exist *exploration strategies* proposing to find a relevant medium to resolve this dilemma [13].

Epsilon greedy strategy is one of the most known ways to deal with the exploration and exploitation topic. It is an algorithmic modification where the agent picks a random action with ϵ probability and with $(1 - \epsilon)$ probability the agent picks a greedy action.

$$a_t = \begin{cases} \text{random action}, & \text{with probability } \epsilon \\ \text{argmax}_a \, Q(s_t, a), & \text{otherwise} \end{cases}$$

By this way, the agent gains an opportunity to explore by trying new actions which are not set by the action-value function. A very common exploration strategy that is used by DeepMind [9, 26] is a modification to the epsilon greedy strategy. The algorithm starts with an ϵ value equal to 1. In each step of the algorithm the ϵ value is linearly annealed by subtracting a small number until it reaches a final value. This approach leads the agent to behave randomly in the beginning. As the time passes actions are taken less randomly and more from the policy. Therefore the search space is expanded in the beginning. After a while the agent focuses on a specific partition of the search space and exploits it.

The *Boltzmann exploration strategy* (also known as the *Softmax exploration stategy*) is yet another technique of action selection for exploration by using the Boltzmann distribution [13]. It builds a probability law for actions and their corresponding Q values [27, 28]:

$$P(a_t) = \frac{e^{Q(s_t,a_t)}}{\sum_a e^{Q(s_t,a)}}, \tag{2.14}$$

where a is an action that can be taken from state, s_t.

Given the state, all the actions have probabilities defined by the Boltzmann distribution. Unlike the epsilon greedy policy, the Boltzmann strategy bases the random action picking related to action's return. It selects actions with higher Q values exponentially more than the other actions.

2.9 Deep Q-Network

Deep neural networks can be seen as universal function approximators [29]. As the cost for searching the whole state space grows to unfeasible limits, it has been

proposed that the state space could be approximated with neural networks. As neural networks are not the direct focus of this thesis, for a more in-depth explanation one can obtain more information from the following source [30]. Deep Q-Networks (DQN) use the neural networks to approximate functions. For example, a DQN agent utilizes neural networks for approximating the action-value function in a model-free manner, surpassing human performance in many Atari games made this method well known and successful among other reinforcement learning algorithms [9].

DQN is a Q-learning algorithm with additional modifications to the conventional one. The main addition is that the DQN algorithm uses weights, θ to approximate the action-value function. An update is applied to these weights by applying the backpropagation algorithm on the neural network. However, the idea of using neural networks for Q-learning was not enough for solving Atari games. There are a few techniques introduced by [9] to fix the problems with this approach.

Experience replay technique was used by Mnih et al. [31] storing experience tuples (s_t,a_t,r_{t+1},s_{t+1}) in a queue called *replay memory* removing very old experiences and keeping the new ones. Unlike plain Q-learning, a minibatch of experiences are sampled from replay memory to apply a gradient descent step. This novel method improves stability and performance by decorrelating experiences.

Another novelty introduced by DQN is to use two neural networks: a main network Q and a *target network* $\hat{Q}$. The main network enforces experience replay and trains the algorithm by applying gradient descent. Target network does not train but calculate the target value for calculations:

$$target \rightarrow r_j + \gamma \max_a \hat{Q}\left(s_{j+1}, a; \hat{\theta}\right). \tag{2.15}$$

The main network is cloned into the target network periodically. Calculating target values from a fixed network increases the performance and the stability of the algorithm.

2.10 Double Deep Q-Network

Double DQN is an extension to the classic DQN approach. The method is similar to the DQN except for calculating the target values. Even though the target values are calculated with a target network just like DQN, the action maximizing the action-value function is calculated by the main network. This change in the algorithm is as follows:

$$target \rightarrow r_j + \gamma Q\left(s_{j+1}, \underset{a}{\text{argmax}} Q\left(s_{j+1}, a; \theta\right); \hat{\theta}\right). \tag{2.16}$$

Network weights θ and $\hat{\theta}$ can be updated symmetrically, interchanging their roles while the algorithm or main network can be copied into the target network. Double DQN reduces overoptimistic predictions of DQN yielding a better performance on the Atari test suite [26].

2.11 Deep Q-Learning with Dueling Architectures

Using a dueling architecture approach for DQN agent introduces a new concept called the advantage function A(s,a). The *advantage* is defined as the difference between the action-value function and state-value function [32]:

$$A(s, a) = Q(s, a) - V(s) \tag{2.17}$$

It is a basic measurement of how much value does a specific action contribute when the agent is on a specific state. Since the main objective is to come up with an action-value function, the action-value function can be expressed as the sum of the state-value function and the advantage [32]:

$$Q(s, a) = V(s) + A(s, a). \tag{2.18}$$

The equation above plays a key role because DQN with dueling architectures estimates the state-value function and the advantage function separately. This process is made by a specially designed neural network having an input layer and some number of layers top of them. Then, it branches into two streams for approximating the state-value function and the advantage term. The state-value stream has fully connected layers on top but the last layer has one output standing for the result of the state-value function. The advantage stream also has fully connected layers added on top but the last layer has a number of outputs equal to the number of actions. These two streams are summed up in order to reach the action-value function [32].

$$Q(s, a; \theta, \beta, \alpha) = V(s; \beta) + A(s, a; \alpha). \tag{2.19}$$

Classic DQN estimates the values of action-state pairs that are used in the experience replay phase. There is a probability that it has observed a state but has not taken the action on that state. On the other hand, DQN with dueling architectures approximates the values of the states [32]. This phenomenon has resulted in better scores in Atari test suite.

It has to be noted that the dueling architecture is not an algorithmic step, but rather a neural network design choice. The same backpropagation step is applied as in classic DQN. Therefore the DQN algorithm remains the same and the Double DQN modification can easily be applied by using dueling architectures. Double DQN with dueling architectures is our main tool for approaching the problem mentioned further in the methodology chapter.

Algorithm 1 Double Deep Q Network Algorithm, [9, 26, 32]

procedure DQN
  Initialize the main network's weights θ randomly.
  Initialize the target network $\hat{\theta}$ randomly
  Pick an exploration strategy Δ
  while convergence is not satisfied **do**
    while s_t is not terminal **do**
      In the state s_t take an action a_t by using Δ
      Observe r_t and s_{t+1}
      Store experience tuple (s_t, a_t, r_t, s_{t+1}) in D
      Randomly sample a minibatch (s_j, a_j, r_j, s_{j+1}) of experiences from D
      if s_{j+1} is a terminal state **then**
        $target \rightarrow r_j$
      else
        $target \rightarrow r_j + \gamma \hat{Q}\left(s_{j+1}, \underset{a}{\text{argmax}} Q\left(s_{j+1}, a; \theta\right); \hat{\theta}\right)$
      end if
      Apply one step of gradient descent on (target - $Q(s_j, a_j; \theta))^2$
      Periodically update target network parameters $\hat{\theta} \rightarrow \theta$
    end while
  end while
end procedure

3
Methodology

This chapter explains and motivates the methodology of formulating the urban scenarios as an MDP and how the framework of training the agent with reinforcement learning algorithms was set up. Furthermore, the implementation of the base model Time-to-Collision is featured.

3.1 Tactical Decision-Making in Urban Scenarios

This section explains the main traffic scenarios that are to be considered by the reinforcement learning agent, which in this case was a two-piece truck. Both of these scenarios are unregulated, therefore they do not introduce traffic lights. Furthermore, they are in urban settings which implies that the maximum allowed vehicle velocity is 50 km/h.

3.1.1 Tactical Decision-Making in a T-junction

T-junctions are a common sight in traffic. In this work an unregulated case where the autonomous agent needs to cross the intersection and take a right turn without colliding with other vehicles as seen in Figure 3.1 was investigated.

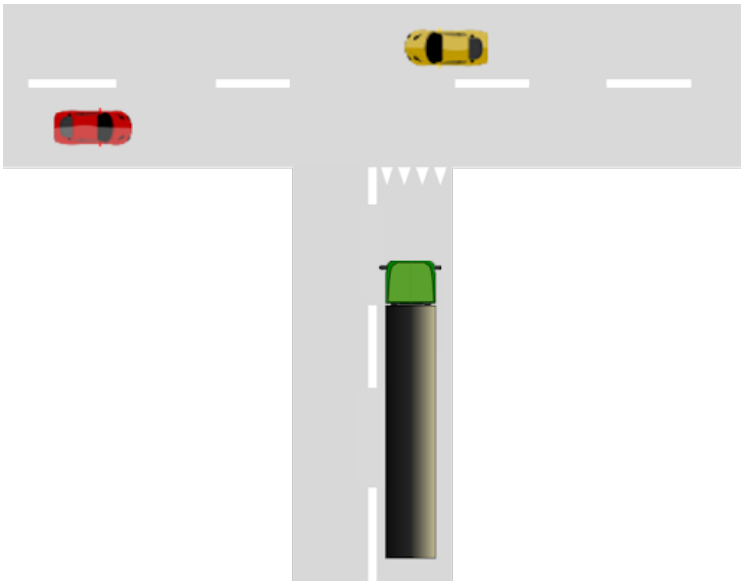


Figure 3.1: A T-junction scenario with an autonomous agent in front of an unregulated crossing and other vehicles on the main road

Secondly, there was an aim to investigate the effects of having occlusions close to the intersection - how much does that change the difficulty of the scenario? This scenario (as seen in Figure 3.2) introduces more realism than the previous scenario because the agent cannot be assumed to perceive the road perfectly.

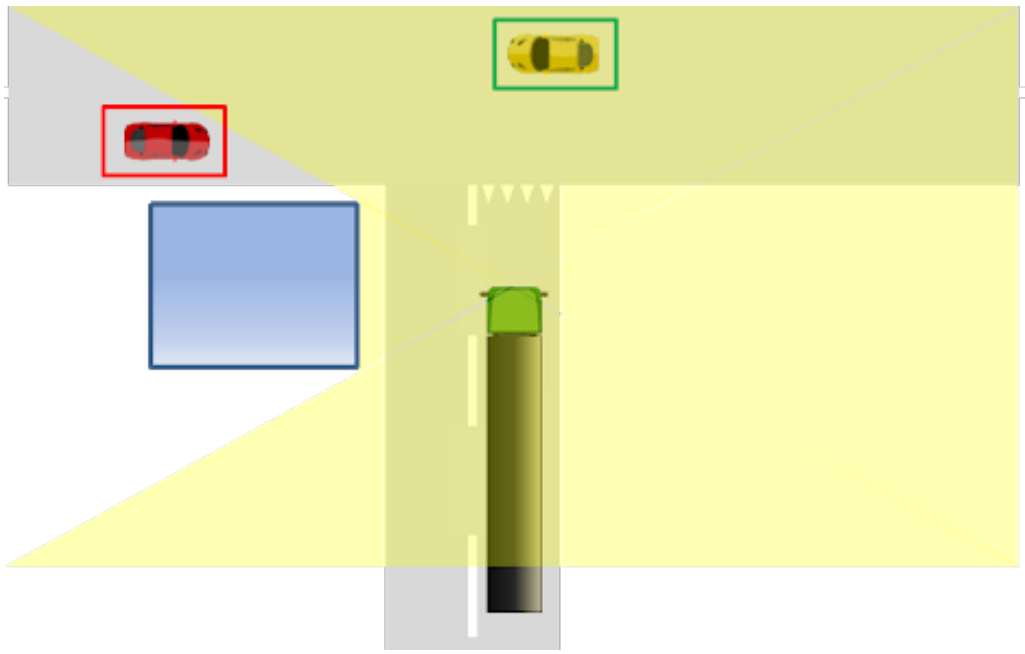


Figure 3.2: A T-junction scenario with occlusions

3.1.2 Tactical Decision Making in a Roundabout

In addition to T-junctions, roundabouts are also quite common in urban situations. Usually, the challenge of making decisions in a roundabout is about having to yield to the vehicles already on the roundabout when crossing and deciding when to make the entry without creating dangerous situations. It can be argued that it is a simpler scenario than the T-junction scenario because there are vehicles only moving in one direction where the ego vehicle has to make a crossing, as opposed to two in the T-junction scenario. On the other hand, it can be argued that for neural networks the constant changing of the angle and thus the direction of the vehicles in the roundabout scenario can pose some challenges. Similarly to the T-junction scenario, the truck needs to enter the intersection and then take a right turn (take the first exit on the roundabout). The main reason for considering this second scenario is to see how well the method generalises in different cases.

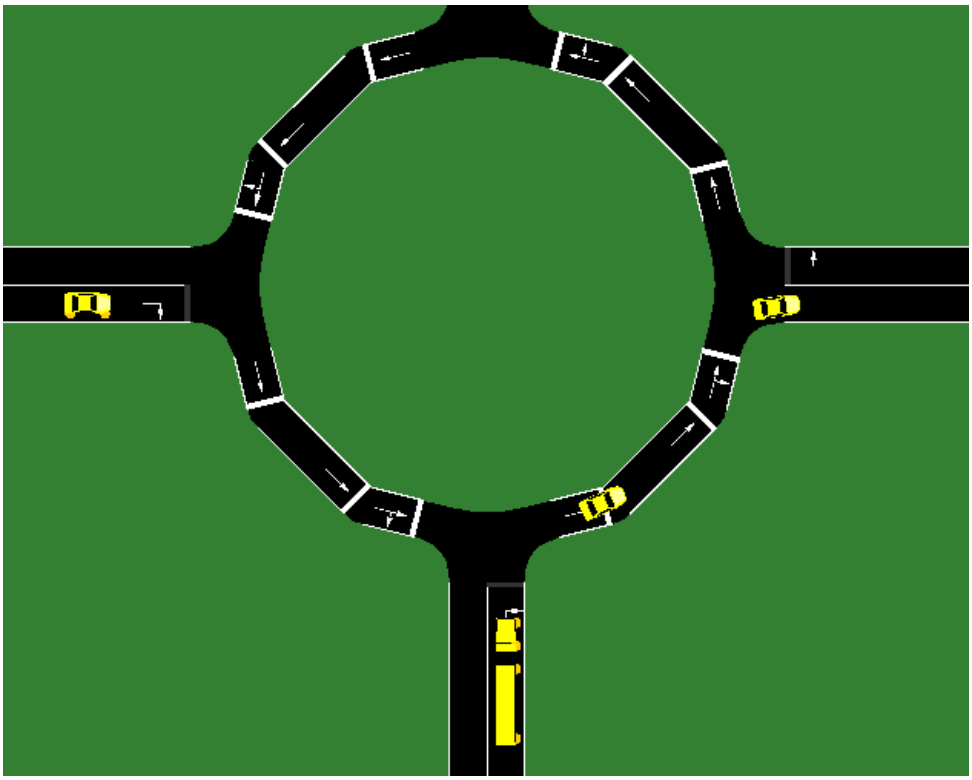


Figure 3.3: A roundabout scenario with an autonomous agent in front of an unregulated roundabout and other vehicles either entering, passing or exiting the roundabout

3.2 Formulation of the Markov Decision Process

To make the previously mentioned scenarios understandable and solvable to the deep learning algorithms, they had to be formulated as a Markov Decision Process. This section explains the reasoning behind creating the MDP. The MDP's for the T-junction Scenario and the roundabout scenario are almost identical except for a few minor details in the implementation of the reward function brought out in subsection 3.2.3.

3.2.1 State Space

The state space for the Agent can be formulated as:

$$s = (s_i, s_k, s_0, act) \text{ where } \; i \in 1, ..., 6 \text{ and } \; k \in 1, 2$$

$$s_i = (x_i, y_i, v_i, \phi_i, w_i),$$

$$s_k = (x_k, y_k, v_k, \phi_k, w_k)$$

$$s_0 = (x_0, y_0, \phi_0, v_0, a_0, x_{tr}, y_{tr}, \phi_{tr})$$

where s_i is the tuple for the nearest vehicles to the ego vehicle, s_k for the *ghost vehicles* (see subsection 3.5.3) and s_0 for the Ego vehicle and trailer parameters. The state space consists of the 49 elements as seen in Figure 3.4.

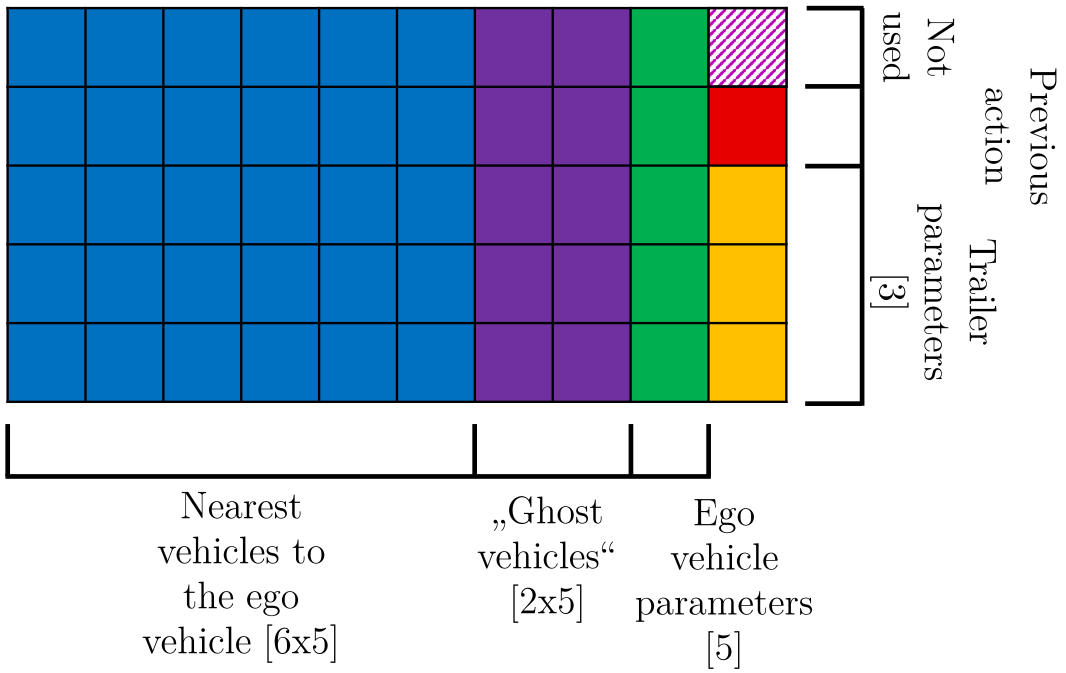


Figure 3.4: Visualization of the state space consisting of 49 elements, the upper-right block with diagonal violet stripes is not used

The descriptions of the variables in the state space are shown in tables 3.1, 3.2 and 3.3. In table 3.2 index i denotes the number of the other vehicles (maximum of 6) and in table 3.3 index k denotes the number of the ghost vehicle (maximum of 2).

Table 3.1: Description of the state space concerning the ego vehicle

Variable	*Name*	*Unit*
x_0	Ego vehicle position (x-coordinate, absolute position)	[m]
y_0	Ego vehicle position (y-coordinate, absolute position)	[m]
ϕ_0	Ego vehicle angle	[degrees]
v_0	Ego vehicle velocity	[m/s]
a_0	Ego vehicle acceleration	[m/s^2]
x_{tr}	Trailer position (x-coordinate, absolute position)	[m]
y_{tr}	Trailer position (y-coordinate, absolute position)	[m]
ϕ_{tr}	Trailer angle	[degrees]
act_{t-1}	Previous action	

Table 3.2: Description of the state space concerning the other vehicles

Variable	*Name*	*Unit*
x_i	Relative position to the Ego vehicle (x-coordinate)	[m]
y_i	Relative position to the Ego vehicle (y-coordinate)	[m]
v_i	Velocity	[m/s]
ϕ_i	Angle	[degrees]
w_i	Waysignal (0 - no signal, 1 - left signal, 2 - right signal)	
N	Number of vehicles the Ego vehicles can see (set as 6)	

Table 3.3: Description of the state space concerning the ghost vehicles

Variable	*Name*	*Unit*
x_k	Position of the "Ghost vehicle" (x-coordinate)	[m]
y_k	Position of the "Ghost vehicle" (y-coordinate)	[m]
v_k	Velocity (always -1)	[m/s]
ϕ_k	Angle of the "Ghost vehicle"	[degrees]
w_k	Waysignal (always -1)	
k	Number of "Ghost vehicles"	

The trailer parameters do not include velocity and acceleration as the Ego vehicle does, because in the implementation the Ego vehicle and the trailer are connected, thus, their velocity and acceleration are the same. The state space for the Agent concerning other vehicles, (s_i), describes the other vehicles in the simulation and the implementation, only the nearest six vehicles are considered because the input size to the neural network needs to be fixed. Therefore, the state space will be filled with empty states when $N < 6$ which sets all the values to -1.

The state space for the Agent concerning the indirect observability of occlusions and the Ghost vehicles is constructed as the following:

- The position of the theoretical vehicle which marks the field of vision on the edge of the occlusion positioned on the left or right (x-coordinate)
- The position of the theoretical vehicle which marks the field of vision on the edge of the occlusion positioned on the left or right (y-coordinate)
- Velocity (always -1)(m/s)
- Angle (degrees)
- Waysignal (always -1)

This was constructed similarly as a regular tuple describing other vehicles so that the neural network could consider them as “worst-case” vehicles coming behind the corner. Secondly, this was done for simplicity as the same representation was used for both vehicles and occlusions. An example can be seen in Figure 3.5 where the dashed red lines mark the line of sight for the truck and the violet cars are the theoretical ghost vehicles and the yellow car is a real vehicle on the road. Ghost vehicles are explained more thoroughly in subsection 3.5.3. Finally, the previous action, a_{t-1}, is added to the state space to make sure the Markovian property stands, because one

part of the reward function (see subsection 3.2.2) was constructed to penalize the changing of actions to prefer more smooth behaviour.

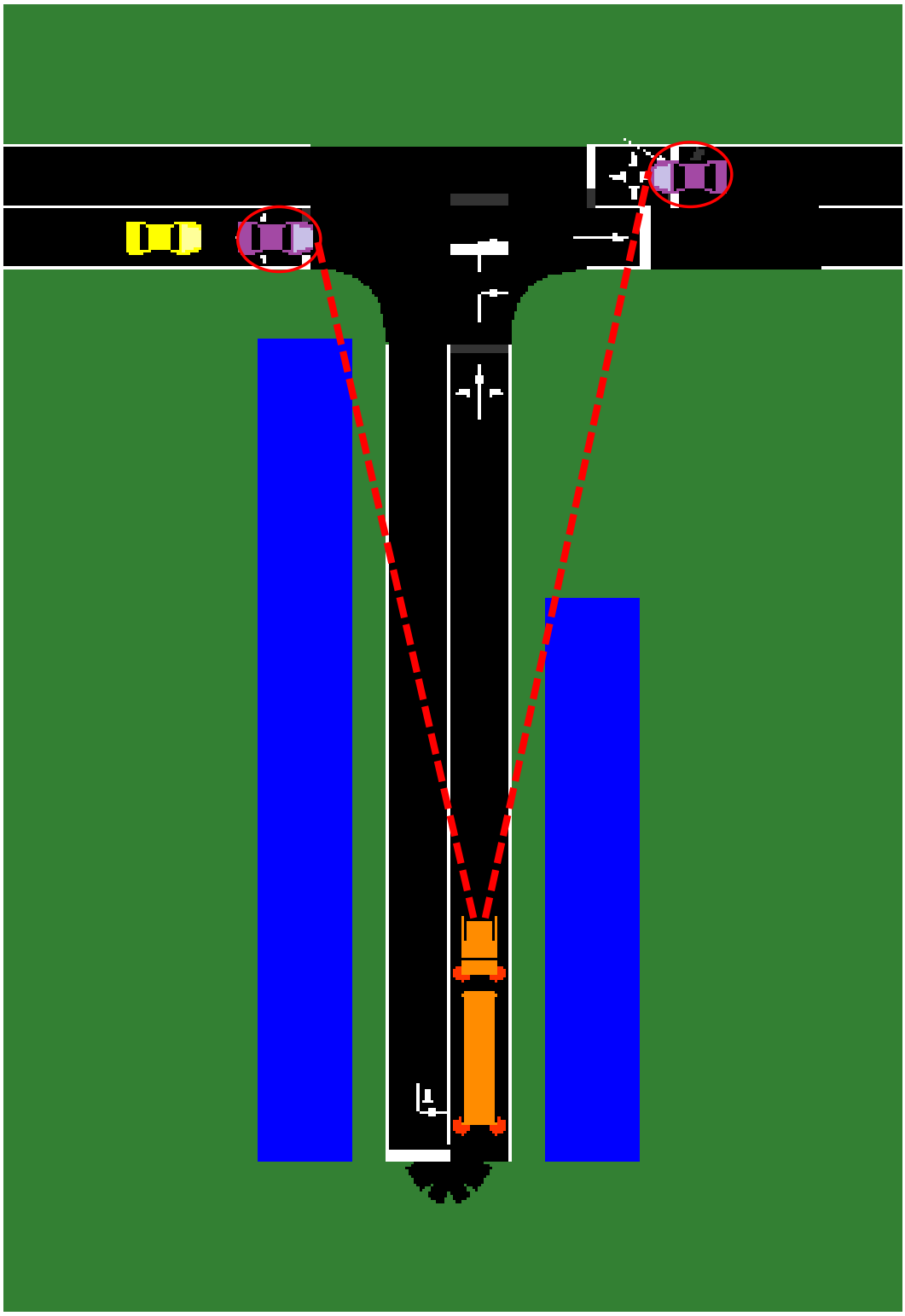


Figure 3.5: A schematic showing the violet ghost vehicles

3.2.2 Action Space

Two action spaces were used for comparison reasons. The first one is a speed-based action space that for each time step agent aims for a target speed and accelerates or decelerates towards that target speed as seen in Table 3.4:

Table 3.4: A speed-based action space

Action	*Description*
Wait	The vehicle stops completely with an acceleration of -4$[m/s^2]$.
Creep	The vehicle moves with a very slow speed of 1$[m/s]$ for having an option to increase vision behind an occlusion while minimizing the risk of a collision. The name of this action is inspired by the paper by Isele, Rahimi, Cosgun, Subramanian, Fujimura [18]. If the current speed is less than 1$[m/s]$, the acceleration is 2$[m/s^2]$, otherwise $-4[m/s^2]$.
Cruise	The vehicle moves with a speed that is acceptable for turning in an intersection and suitable for moving in an uncertain area. The speed was set to 8$[m/s]$. If the current speed is less than 8$[m/s]$, the acceleration is 2$[m/s^2]$, otherwise $-4[m/s^2]$.
Go	This action is meant to reach the regular urban speed after completing a manoeuvre therefore the speed is set to 14$[m/s]$ with an acceleration of 2$[m/s^2]$.

The motivation behind these actions is that these imitate certain behaviour seen in traffic. "Go" can be seen as speed limit in many urban areas in Europe ($14m/s$ roughly equals 50 km/h), "Cruise" is a speed that is acceptable for turning on an intersection, "Wait" describes the action on an intersection when one has to give way and "Creep" describes the movement when wants to move slowly to enhance the view behind an occlusion. It can be argued that this action space contains tactical decisions imitating real-life decisions and thus is "high-level" compared to the second one which resembles actuations.

The second action space is a lower level one, controlling accelerations in each timestep with three different options: accelerate, decelerate and set the acceleration to zero (keep the speed) as seen in Table 3.5:

Table 3.5: An acceleration based action space

Action	*Description*
Accelerate	The vehicle sets the acceleration to 2$[m/s^2]$
Keep the speed	The vehicle sets the acceleration to 0$[m/s^2]$
Decelerate	The vehicle sets the acceleration to -4$[m/s^2]$

When accelerating the vehicle does not exceed the speed limit and when decelerating the vehicle cannot have a speed lower than 0. These action spaces were created to be similar in terms of accelerations so they could be compared.

3.2.3 Reward Function

The reward function consists of the following elements:

1. Reaching the goal (150 points)
2. Collision between the Ego vehicle and another vehicle (-100 points)
3. Timestep penalty (-0.5 per time step)
4. Penalty for changing the action for having less "jerky" motion (-2 per change of action)
5. Penalty for stopping at the intersection (-5 per time step)
6. A smaller penalty (-0.1 per time step) for waiting in front of an intersection

The motivation for choosing the parameter values was the following: the first parameters chosen were "Reaching the goal" and "Collision...". They were chosen as arbitrary values with one being positive (+150) and the other negative (-100), but with the positive one being a little bigger as being successful was the main goal of the training. The simulation was also given a timeout period of 160 seconds (with two time steps per second) which seemed like a reasonable time that the truck should solve the situation and it would create a situation where not moving and thus having zero probability would be worse in the eyes of the agent than crashing. Therefore, the "Timestep penalty" was chosen as -0.5 per time step. Furthermore, a penalty for changing the action was introduced to decrease "jerky" behaviour of the truck which is not desirable for the potential humans, cargo being transported nor the fuel economy. In addition, a relatively high penalty for stopping at intersection was introduced, because there is not an objective reason why a vehicle which has decided to cross an intersection would have to stop in a normal condition. It has to be noted that the previous penalty was not applied in the roundabout scenario, because as the intersection is being passed quite slowly by other vehicles, as it is a roundabout, stopping can be considered reasonable by the truck. Finally, about a 1.5 meter area just before the intersections was marked as a place where the truck would receive less time penalty as usual which would give it a reasonable place to wait to take a tactical decision at the right time. For any timestep only one penalty can apply at once.

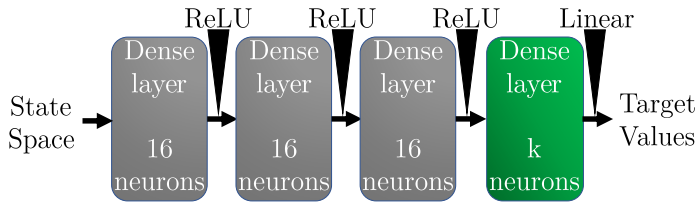
3.3 Design choices for solving the Markov Decision Process with Deep Reinforcement Learning

Different neural network architectures with varying depths and sizes were used for training. The training parameters itself can be seen in Table 3.6. The learning rate was 0.0001 and it was chosen as quite low to not risk converging to a suboptimal solution while potentially sacrificing speed. Training one model was usually done for 500 000 simulation steps, which was chosen because the training time for one model usually fell into the domain of 5-10 hours which was suitable for seeing results and convergence in some terms. For filling the memory before actually starting the training, 5000 steps were used for warmup. As this is 1/100 of the whole training, it was found suitable. For optimization, Adam was used as this is quite a standard choice in the field of machine learning [33]. For computing the error, mean average error was used which makes the results quite easy to interpret. For memory, 200000 steps were saved thus during the training process the older experience with suboptimal policies could be forgotten. For the exploration policy, Boltzmann Q policy was used as this often gets better results than the often-used ϵ-greedy policy [34]. The discount factor was chosen as 0.99 which is relatively close to one because we wanted the truck make its decisions based on a long run perspective.

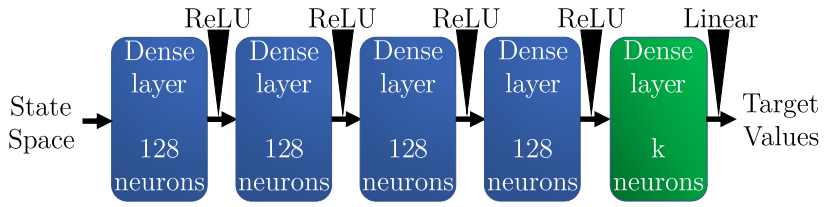
Table 3.6: The parameters used for training the DQN models

Parameter	*Value*
Learning rate	0.0001
Number of steps for the warm-up phase	5000
Number of steps for one training	500 000
Length of the experience replay while training	200 000
Discount factor (γ)	0.99
Tensorflow version	1.13.1

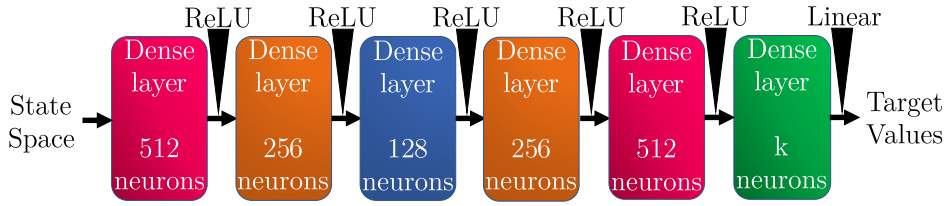
Parameter	*Method*
Exploration policy	Boltzmann Q policy
Optimizer of the objective function	Adam
The error computation method	Mean Square Error



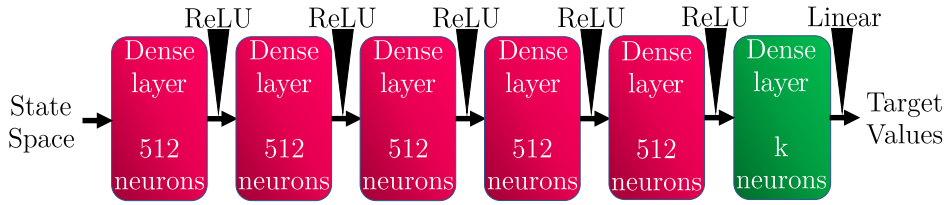
(a) The "small" network architecture (1)



(b) The "baseline" network architecture (2)



(c) The "decoder" network architecture (3)



(d) The "big" network architecture (4)

ReLU – the activation function named „Rectified Linear Unit“
k – number of neurons in the action space
Linear – the linear activation funtion (identity)

Figure 3.6: Different neural network architectures used while training the DQN models (k is the number of actions in the action space)

3.4 Comparing Deep Q-Networks algorithms with the baseline model

The DDDQN algorithm is known to be a reasonably well-performing algorithm as tested by the DeepMind's Atari test suite, inspired by [35]. After the Rainbow algorithm, DDDQN seems to be on par or slightly worse performing than the Distributional DQN agent.
The obvious choice would have been choosing the Rainbow algorithm which consists of multiple different DQN modifications [35], but that would be hard for us to analyse and report the results because of its very high level of complexity and it would be outside of the scope of this thesis. The Distributional DQN agent also requires changes in the Q-learning at the update step. Furthermore, most importantly we wanted to use a tested algorithm by using a reinforcement learning library named Keras-rl. DDDQN was provided us by Keras-rl, however, Rainbow and Distributional DQN were not. Even though the algorithm was not the best one, it would still be sufficient to understand if the DQN approach is appropriate for the goals of this thesis. Taking this into account, it was decided to carry our experiments with DDDQN due to its high potential for performance. However, there are examples where DDDQN performs worse on some games in the Atari test suite than the regular DQN. Therefore we also wanted to see DQN's performance with a Setspeed action space and fixed neural network. Besides, even though most of the models use the DDDQN algorithm, DQN can still be compared with them.
The final models trained were:

1. DDDQN - Acceleration action space with a baseline neural network, shortened as DDDQN.Acc.Base
2. DDDQN - Setspeeds action space with a baseline neural network, shortened as DDDQN.Spd.Base
3. DQN - Setspeeds action space with a baseline neural network, shortened as DQN.Spd.Base
4. DDDQN - Setspeeds action space with a small neural network, shortened as DDDQN.Spd.Sml
5. DDDQN - Setspeeds action space with a decode neural network, shortened as DDDQN.Spd.Dcd
6. DDDQN - Setspeeds action space with a big neural network, shortened as DDDQN.Spd.Big

3.5 Implementation

The simulation is set up in a framework named SUMO (**S**imulation of **U**rban **Mo**bility) which is an open-source traffic simulator [36]. The simulation has been set up as an OpenAI Gym environment for it could be interfaced with a reinforcement learning library named Keras-RL [27, 37]. For the deep reinforcement training, Keras-RL uses an open-source software library named TensorFlow. The flow of information inside the framework is shown in Figure 3.7.

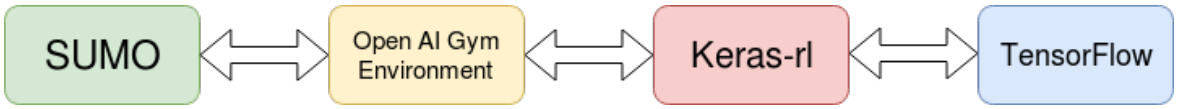


Figure 3.7: The simulation setup showing the flow of information through SUMO, OpenAI Gym, Keras-rl and Tensorflow

3.5.1 Simulation of Urban Mobility

SUMO is a tool which allows creating traffic scenarios with relative ease. SUMO features an interface called TraCI (Traffic Control Interface) which allows for communication with the simulation and controlling the different vehicles and objects. The simulation was chosen to run at 10 Hz for smooth visualization purposes, but the timestep for making decisions was chosen as 0.5 seconds. As for making a decision, a forward pass must be done through the neural network, thus the 0.5 seconds was chosen as a compromise between taking decisions at a reasonable rate which we want to maximise and the training speed which we want to minimise. Two scenario templates were created: a T-junction and a roundabout.

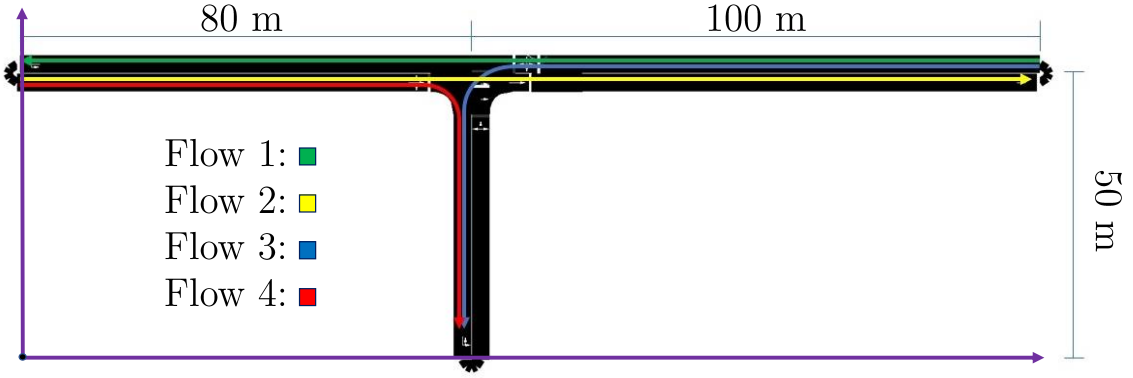


Figure 3.8: The dimensions of the SUMO map and the routes of the vehicles in the T-junction scenario

A T-junction was constructed with SUMO as seen in Figure 3.9. For the scenario, a map of an intersection was constructed. The origin of the coordinate systems in SUMO is usually in the lower left corner as seen in Figure 3.8 corresponding to the violet arrows. Four flows of vehicles are used in this scenario not taking into account the ego vehicle. The flows are the following:

1. Vehicles moving from east to west
2. Vehicles moving from west to east
3. Vehicles moving from east to south
4. Vehicles moving from west to south

For training, validating and testing various sub-scenarios were created. A sub-scenario in this context notes a certain case where the scenario, map and, the goal stay the same for the Ego vehicle, but the flows of the other vehicles differ. For training the agent, six sub-scenarios were constructed. Secondly, five sub-scenarios were

constructed for validating the models with random entry times for the Ego vehicle which in total came up to 100 validation cases. These validation cases were the same for the non-occluded and occluded case with the occluded case also having random occlusions on top of these validation cases. Finally, additional five sub-scenarios were created for testing which, similarly to the validation set, also had 100 cases in total. The specifics of these sub-scenarios are brought out in subsection 3.5.4.

These vehicle flows were set to have variable speeds from 25 to 50 km/h to imitate realistic urban driving behaviour. Secondly, they were modelled as "aggressive" which means that they react minimally to other vehicles. There were two reasons for this choice: one being that it would always assure that the truck gives way as required by law and it will make the scenario look as the worst case, which is reasonable when trying to obtain a "safe" policy. On the other hand, the drawback of this approach can be that it is not quite realistic, as in realistic situations other vehicles rather stop when a truck is being seen making an extensive manoeuvre.

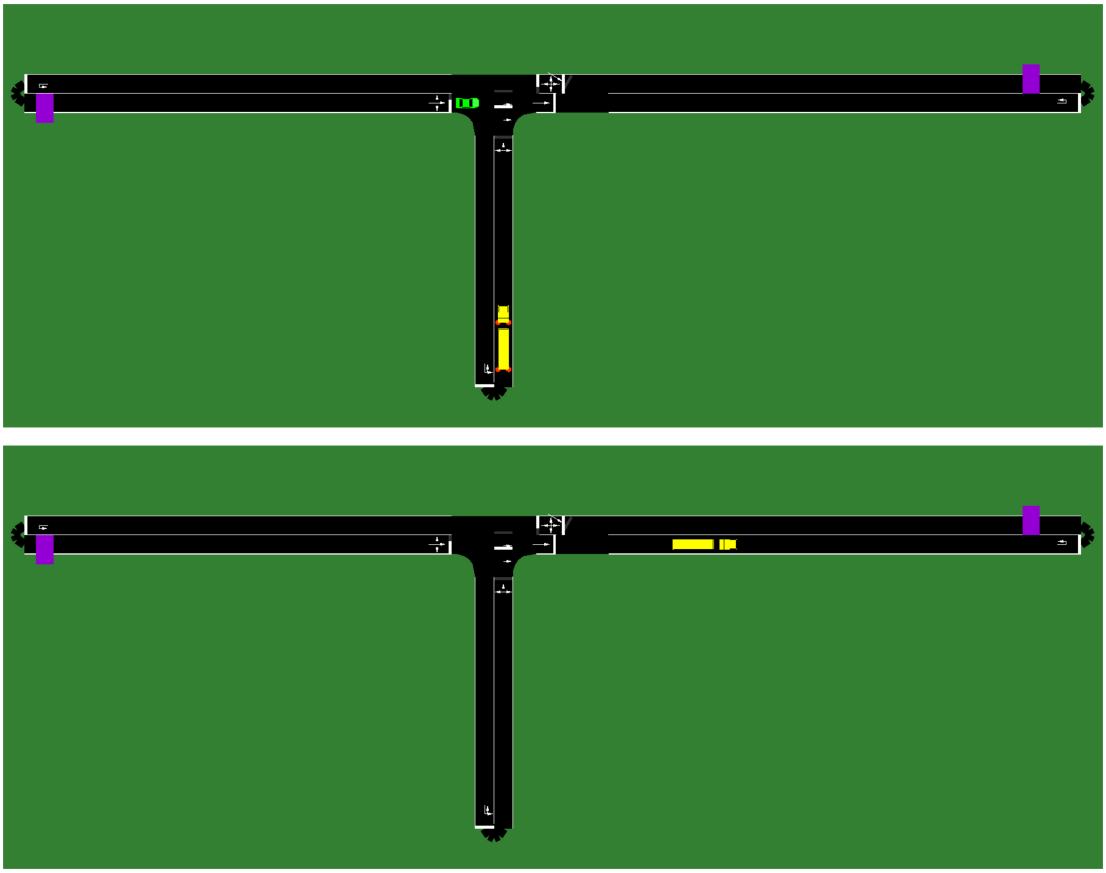


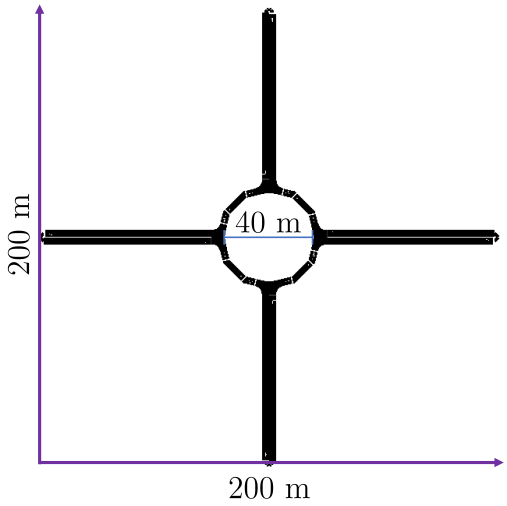
Figure 3.9: A T-junction Scenario in SUMO showing the starting and the ending position of the ego vehicle

Secondly, a roundabout scenario was created as seen in Figure 3.10. In this scenario, a map of a roundabout with roads leading to it was constructed. The origin of the coordinate systems in SUMO is usually in the lower left corner as seen in Figure

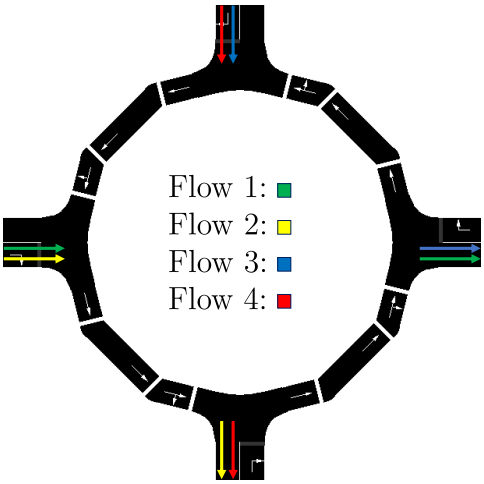
3.10, Subfigure (a) corresponding to the violet arrows. Four flows of vehicles are used in this not taking into account the ego vehicle which are brought out in Figure 3.10, Subfigure (b). The flows are the following:

1. Vehicles moving from west to east
2. Vehicles moving from west to south
3. Vehicles moving from north to east
4. Vehicles moving from north to south

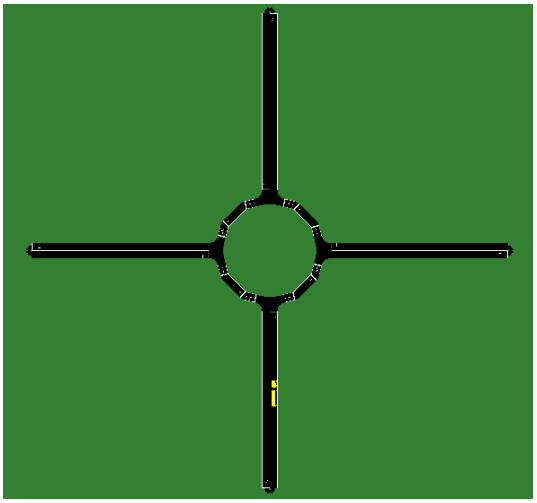
For training the agent, four sub-scenarios were constructed. Furthermore, for testing additional four sub-scenarios were created. The specifics of these sub-scenarios are brought out in subsection 3.5.4.



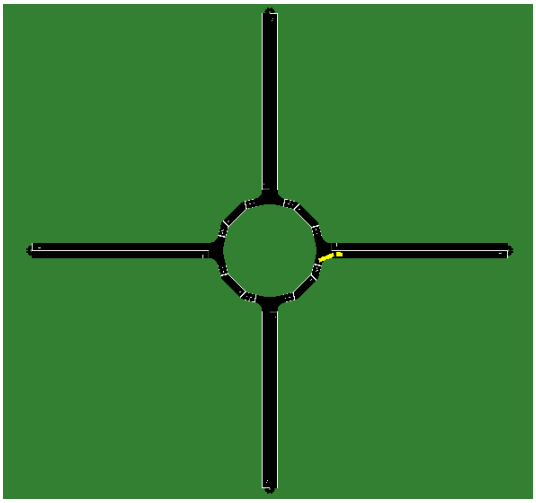
(a) The dimensions of the SUMO map in the roundabout scenario



(b) The vehicle flows in the roundabout scenario (zoomed in)



(c) The starting position of the ego vehicle in the roundabout scenario



(d) The final position of the Ego vehicle in the roundabout scenario

Figure 3.10: The dimensions, flows and the route for the Ego vehicle in the roundabout scenario

3.5.2 Trailer Modelling and a Custom Path Model

As there was no default trailer model in the T-junctions scenario that served the needs of the implementation in SUMO, one had to be created. The reason is that there was a need to model the truck and the trailer separately to simulate the turning and collision dynamics. This was achieved by summoning two vehicles into the simulation together and making the "trailer" vehicle move with the same speed and acceleration as they were physically connected. This approximation has been sufficient enough to serve our simulation needs.

When making a right turn in the simulation, the heavy vehicle is modelled to complete the manoeuvre through the opposite lane as seen in Figure 3.11. The reason is that when the heavy vehicle would only use the correct lane, the trailer behind would cut into the area outside the road which is potentially dangerous to pedestrians and the trailer itself. Our implementation imitates this behaviour but does not completely replicate it as usually in realistic situations the truck enters the opposite lane and the trailer does not fully, which is not the case in the simulation. Nevertheless, the implementation does not make the scenario easier because of this approximation - the opposite lane will be occupied longer than it would be in realistic situations.

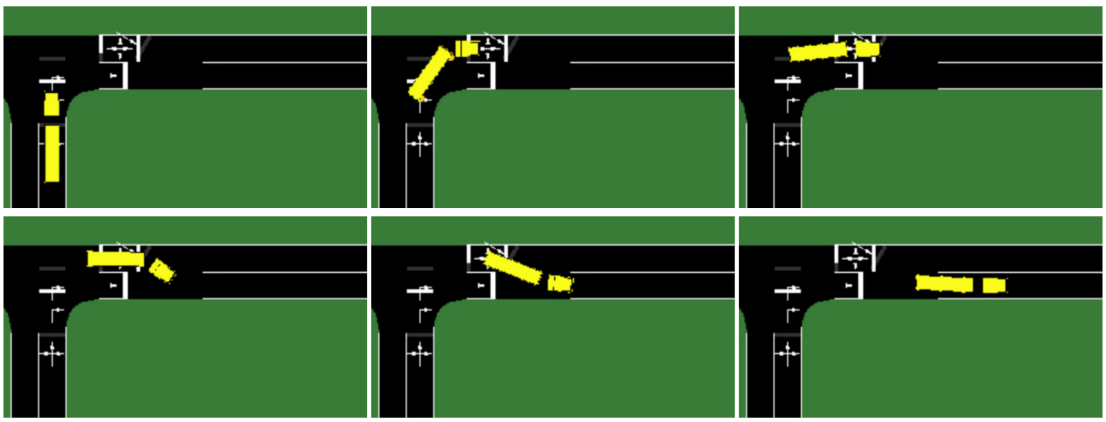


Figure 3.11: The implemented custom path for the truck imitating a truck making a right turn on a narrow intersection

3.5.3 Occlusion Modelling

During a real traffic situation, one cannot presume that drivers have all the relevant visual information available. For example, there can be visual obstructions such as houses next to an intersection. One goal of this thesis is to take these kinds of issues into account when training a reinforcement learning model as described in Chapter 2.3.

For modeling the occlusions in the T-junction scenario, a python implementation of a recursive field of view algorithm was used and modified to fit the needs of this work [38]. The T-junction scenario is fitted with a 12x36 grid world as shown in Figure

3.12 with the pink rectangle. The size of the grid is chosen as small as possible for keeping the computational expenses for the reinforcement learning training low while still having enough resolution to have the occlusions serve its purpose.

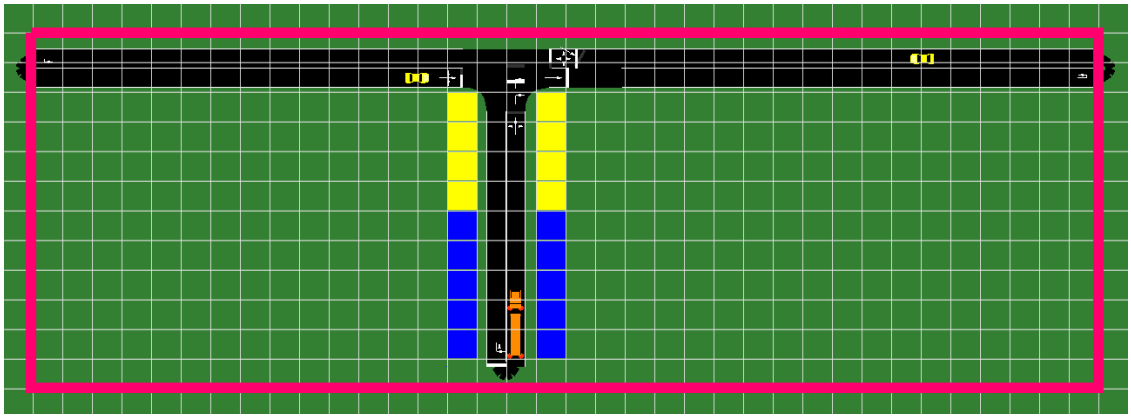


Figure 3.12: The placement of the 12x36 occlusion grid in the simulation shown by the pink rectangle

The sizes of the grid blocks correspond to 5x5 meters in the simulation and the corners of the grid are situated at (0,-5);(0,55);(180,-5);(180,55). When a decision is made to imitate an occlusion on either or both sides of the roads, the area marked by blue boxes is always occupied which also mark the minimal length of these occlusions. These occlusions can be extended by the length marked by the yellow boxes up until the intersection. Furthermore, if there are occlusions on both sides of the road, they can have different lengths. The base length of the occlusions was chosen such that the ego vehicle would not have a complete view of the horizontal road when being summoned. Secondly, the maximum length imitates the situation where the ego vehicle should get as close to the intersection as possible while not crashing and it should be possible to have a complete view of the horizontal road. Examples of these occlusions can be seen in Figure 3.13.

For the agent could have information about the existence of occlusions, certain tuples named *ghost vehicles* are created consisting of x and y coordinates, an angle, velocity and a turning signal which mark the lines of sights on the horizontal road (see Figure 3.5). For these variables would be similar to the rest of the state space for the sake of simplicity, they are modelled as vehicles on the edge of the lines of sight. This can be seen as a heuristic choice: if one cannot see behind an occluded area, it should be assumed that a vehicle can appear at any time. These variables are visually represented in the simulation as purple boxes as seen in Figure 3.13. It must be noted that these colored boxes are approximations as the road does not exactly align with the grid, but are good enough for this thesis. While training the scenario with occlusions, the variables are fed into the state space.

An alternative to this approach was considered which would have been feeding a fairly large grid in conjunction with the rotation of the ego vehicle describing the visible and non-visible parts of the scenario to the ego vehicle. This was not chosen

because of the potential computational expenses this would have brought to the training process. As seen in Figure 3.13, visible vehicles are marked with a green color and the vehicles which are occluded from the ego vehicle are marked with a red color. If the vehicles are occluded, they are not fed into the state space while training.

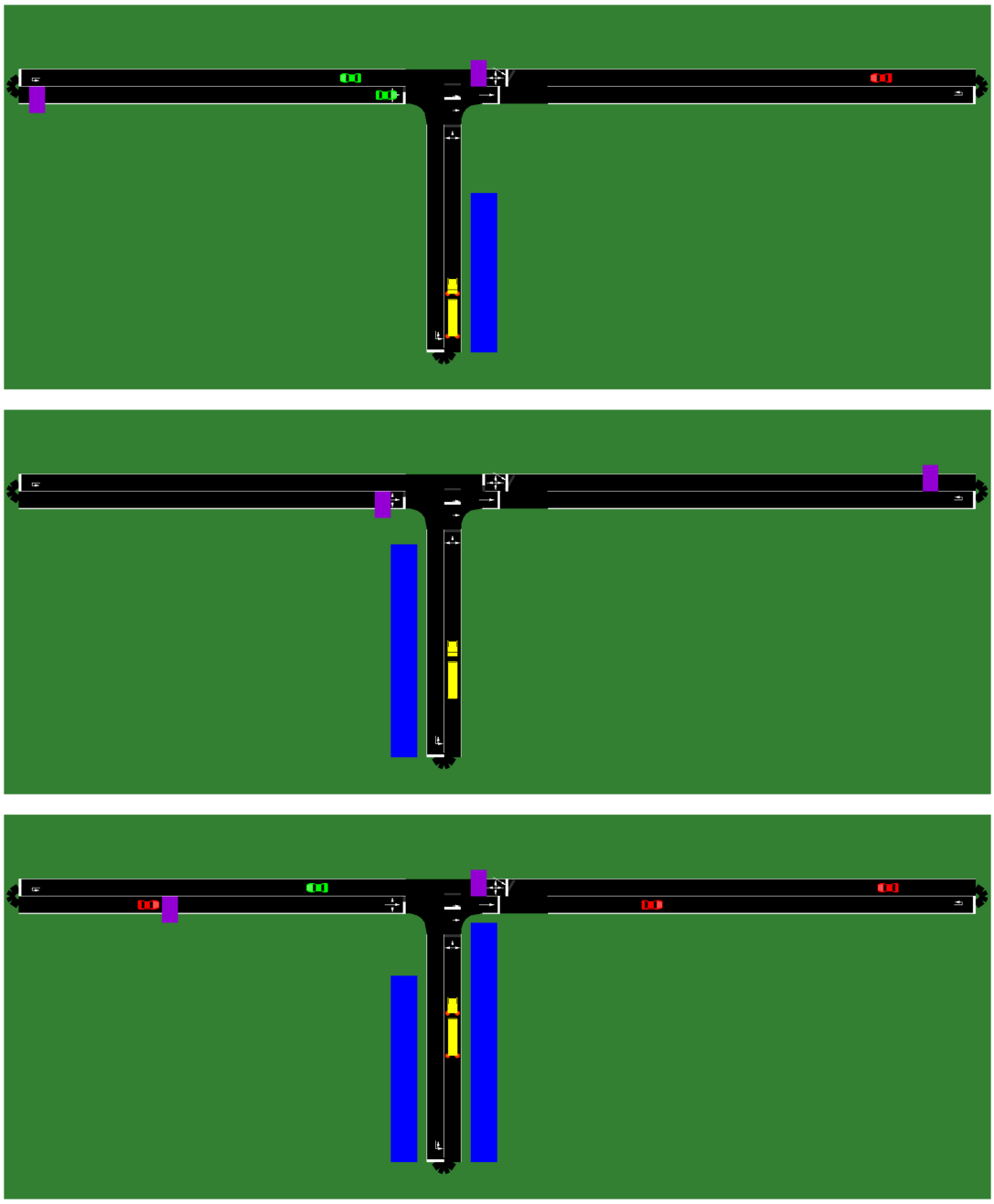


Figure 3.13: An example of the occlusion modelling in the simulation highlighting the ghost vehicles shown as violet rectangles

3.5.4 Implementation of the Training Scenarios

For training the agent in the T-junction scenario, six sub-scenarios were created ranging from easy to hardly solvable. One of the fundamental tools of SUMO is the availability of defining vehicle flows. One of the ways to implement these flows is to set a probability to a vehicle being summoned to the simulation in any given second. This means that the average summoning time for vehicles can be defined, but the exact times for appearing in the simulation are random when using the random option offered by SUMO. If this option is not used, the random number generator offered by SUMO defaults to the seed 23423 [39].

Secondly, four different possibilities for occlusions appearing were created:

- No occlusions
- Occlusion on the right
- Occlusion on the left
- Occlusions on both sides

For any given training episode, one of the six sub-scenarios is loaded and then one of the four occlusion placements are applied. Furthermore, to introduce more randomness to the training to achieve generality, the time for the ego vehicle to be summoned is not fixed, but rather time is chosen randomly between 5 to 30 seconds from starting one episode. The configurations of these different scenarios can be seen in Figure A.7 in the Appendix. Aside from this, 100 random test scenarios were created which are based on 5 baseline scenarios and the occlusions are modelled randomly as seen in Figure 3.12. Furthermore, the time for summoning and starting the ego vehicle was set as a random time between 5 and 30 seconds.

For the roundabout scenario, four sub-scenarios were created. The main difference between the roundabout and T-junction scenarios is that the roundabout scenarios do not use any occlusion modelling at all and are only used to test the generality of the DQN methods. The time for summoning and starting the ego vehicle was set as a random time between 20 and 50 seconds. The reasoning why this is higher than in the T-junctions scenario is that when the other vehicles are summoned in the simulation, it takes more time for them to reach the position where collisions mostly happen (For the T-junction - the middle of the intersection, for the roundabout - the position where the Ego vehicle enters the roundabout).

3.5.5 Implementation of a Baseline Model

Time to Collision (TTC) is a rule-based heuristic method that can be used to avoid collisions taking into account the speeds and directions of other vehicles. It is often used as a baseline method to compare reinforcement learning based methods [11]. The method is based on the idea of choosing a possible position where vehicles can crash into each other (the position can change in time) and estimating their time of arrival to the position. This idea can be implemented differently in various scenarios, for example, highway situations or intersections, but the fundamental idea is similar.

$$TTX_n = \frac{||\vec{r}_+ - \vec{r}_n||}{|\vec{v}_n|} \operatorname{sign}\left((\vec{r}_+ - \vec{r}_n) \cdot \vec{v}_n\right)$$

Where:

- TTX is the time to collision for one vehicle
- $\vec{r}_+$ is the vector representation of the collision coordinate
- n denotes the index of the vehicle
- the sign() is used to understand whether the vehicle has passed the collision coordinate (if passed, TTX is negative)

As vehicles are not points, but objects with a certain length, a buffer variable is used which can be tuned in accordance with a scenario. To avoid collisions, the following condition must be satisfied:

$$|TTX_n - TTX_{n+1}| < \alpha$$

Where:

- α is the buffer variable
- TTX_n is the time to collision metric for different vehicles

The bigger the buffer variable is, the less chance there is for a crash, but the average time for solving scenarios [40].

The inspiration to implement a Time-To-Collision base model came from a paper by Bouton, Cosgun, Kochenderfer [11]. The central idea is to first set the most probable location where the collisions between the ego vehicle and other vehicles would take place. Secondly, then the time it takes for all the vehicles to reach that line in the simulation is estimated. The key here is to set an optimal buffer time marking acceptable time difference between the times so that the ego vehicle could cross the intersection.
The implemented equations for TTC are the following:

$$y = \frac{(y_c - y_e)}{v_e}$$

$$y_n = \frac{|(x_c - x_v)|}{v_v}$$

Where:

- y is the TTC metric for the ego vehicle
- y_n the TTC metric for the other vehicles
- n is the index of the vehicle
- y_c is the y coordinate of the position where the ego vehicle touches the lower part of the red line in Figure 3.14
- y_e is the current y-coordinate for the ego vehicle
- v_e is the current velocity for the ego vehicle
- x_c is the y coordinate where the vehicle's centre point crosses the red line in Figure 3.14
- x_v is the current x-coordinate for the other vehicles

- v_v is the current velocity for the other vehicles

For the vehicles coming into the simulation from the east:

$$|y - y_n| > 13$$

For the vehicles coming into the simulation from the west:

$$|y - y_n| > 8$$

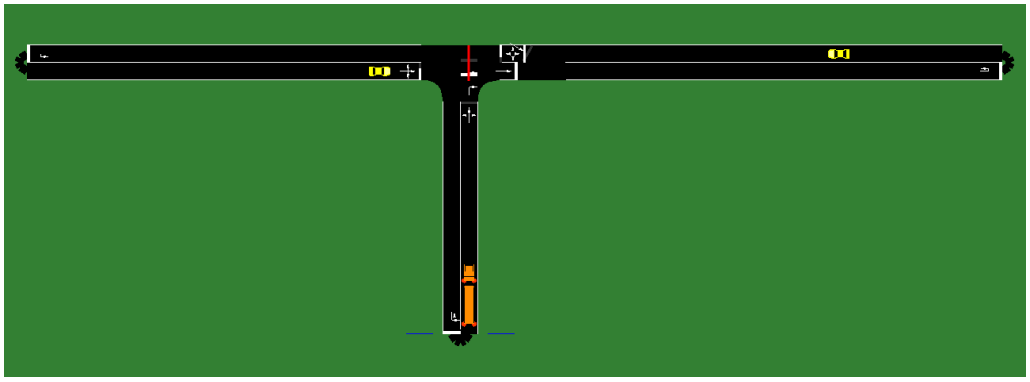


Figure 3.14: The Time-To-Collision Method for the T-junction scenario emphasizing the red line which marks the line of collision for the Ego and other vehicles

The line was chosen aligned with the axis of the ego vehicle when it is close to the intersection. If TTC is used in the occluded T-junction scenario, the model will not cross the intersection until it can see the roads leading to the intersection. It does this by having the same input as the DQN models have - Ghost vehicles (see Figure 3.4). In the roundabout scenario, TTC works quite similarly to the T-junction one, but the difference is that there is only one parameter for tuning as the vehicles are only coming from one side:

$$|y_{ra} - y_n| > 16$$

where y_{ra} is the TTC metric for the ego vehicle in the roundabout scenario. Figure 3.15 shows the point where the TTC algorithm is applied.

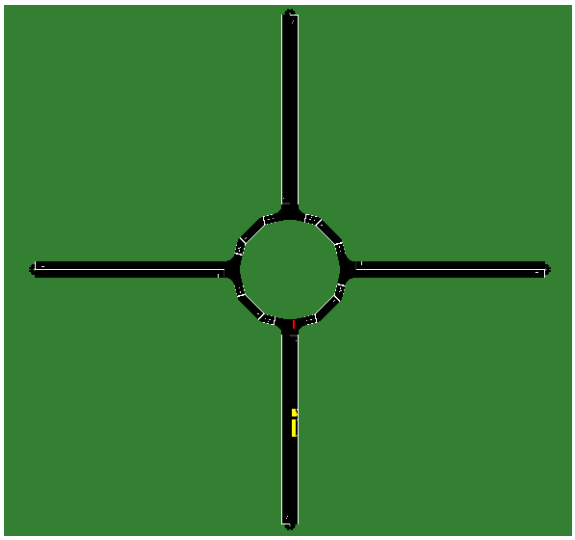


Figure 3.15: The Time-To-Collision Method for the roundabout scenario emphasizing the red line which marks the line of collision for the Ego and other vehicles

Different buffer times (such as 13, 8 and 16) were chosen empirically to avoid collisions by running a lot of simulations and seeing that the probability of collisions would be taken down to a minimum in both scenarios.

3.5.6 Deep Reinforcement Learning Framework - Keras-RL and OpenAI Gym

For the implementation of the Deep Reinforcement Learning Framework, a package called Keras-RL was used [27]. This library offers the implementation of different reinforcement learning algorithms such as DQN (Deep Q-Network), Double DQN, Dueling DQN and Deep Deterministic Policy Gradient. OpenAI Gym is a toolkit for using and developing different reinforcement learning algorithms [37]. The reason for using for using this particular environment is that it seamlessly integrates with Keras-RL giving us a convenient way to use different algorithms.

4
Results

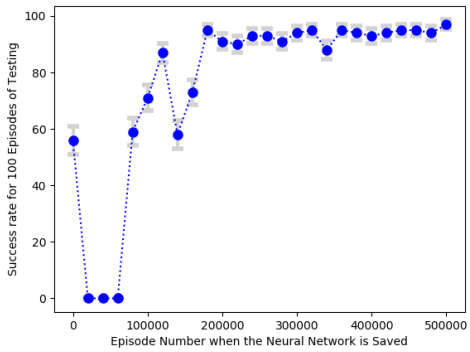
This chapter demonstrates the results of comparing different reinforcement learning algorithms while training taking into account their performance on the urban scenarios such as the T-Junction and the roundabout created for testing. All of these models were compared with the baseline model Time-To-Collision.

4.1 Non-occluded T-Junction Scenario

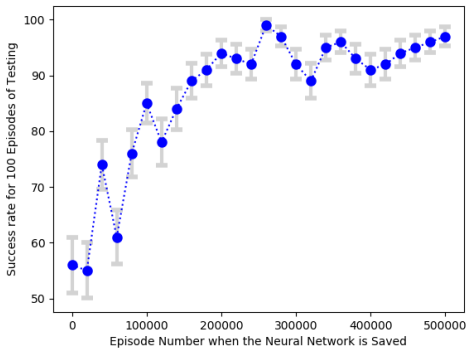
There are six different deep reinforcement learning models which were evaluated:

1. DDDQN - Acceleration action space with a baseline neural network, shortened as DDDQN.Acc.Base
2. DDDQN - Setspeeds action space with a baseline neural network, shortened as DDDQN.Spd.Base
3. DQN - Setspeeds action space with a baseline neural network, shortened as DQN.Spd.Base
4. DDDQN - Setspeeds action space with a small neural network, shortened as DDDQN.Spd.Sml
5. DDDQN - Setspeeds action space with a decode neural network, shortened as DDDQN.Spd.Dcd
6. DDDQN - Setspeeds action space with a big neural network, shortened as DDDQN.Spd.Big

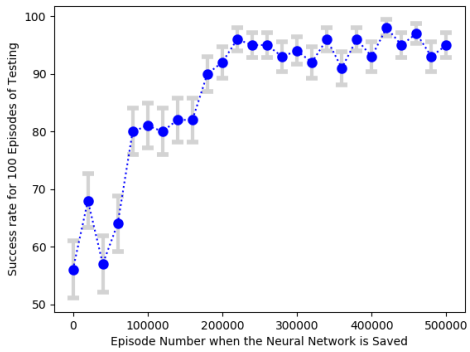
These algorithms were compared with the baseline model TTC in the non-occluded scenario as shown in Figure 3.14 taking into account different metrics. Algorithms were all trained for 500 000 steps. The number of episodes differed because of the randomness which is apparent in the training process varying the lengths of the episodes. For all of the models, after 20 000 steps the models were saved while training. These saved models were all tested on the 100 random validation cases and then the mean success rate with the standard deviation of the mean (marked as gray error bars in Figures 4.1 and 4.3, and as coloured error bars in Figures 4.2 and 4.4) was evaluated as shown in Figure 4.1. The cases were randomly created, but kept fixed when evaluating the performance of different cases. Then the best-performing save-points were evaluated for each model and put together into Table 4.1. The exploration was removed in the validation and test episodes which means no random actions were taken. The same models were taken and evaluated on 100 random test cases and the results were put into Table 4.2. A similar approach was used in section 4.2.



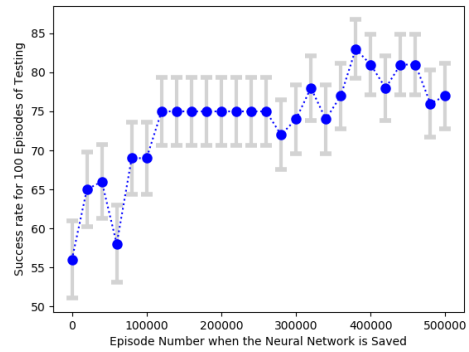
(a) DDDQN - Acceleration action space with a baseline neural network (1)



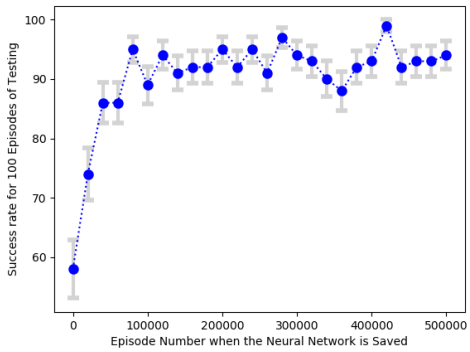
(b) DDDQN - Setspeeds action space with a baseline neural network (2)



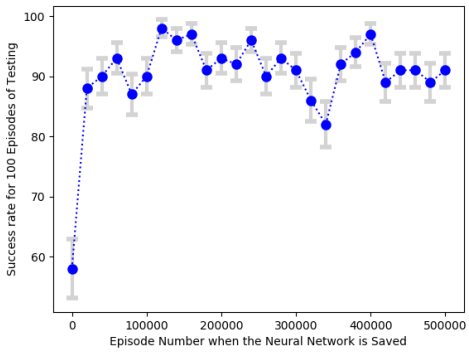
(c) DQN - Setspeeds action space with a baseline neural network (3)



(d) DDDQN - Setspeeds action space with a small neural network (4)



(e) DDDQN - Setspeeds action space with a decoder based neural network (5)



(f) DDDQN - Setspeeds action space with a big neural network (6)

Figure 4.1: Results for six deep learning models in the non-occluded T-junction scenario

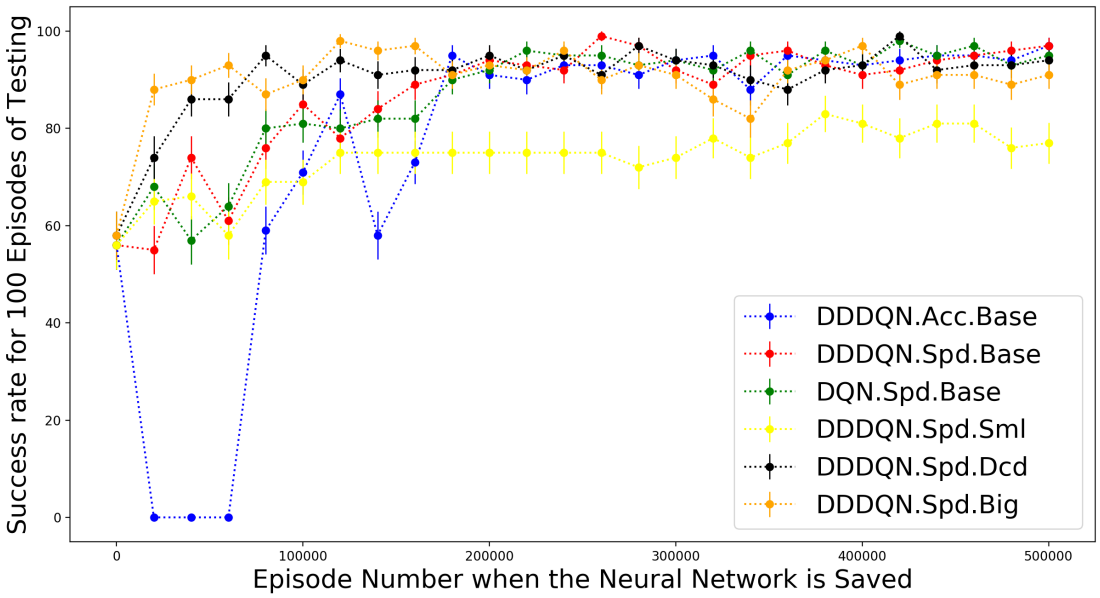


Figure 4.2: Combined results for six deep learning models in the non-occluded T-junction scenario

In Tables 4.1, 4.2, 4.3, 4.4 and 4.5 the agents with random actions with both the Setspeed and Acceleration actions spaces (8 - Setspeed and 9 - Acc) were added to the results to see how they compare to trained agents being a baseline metric. Furthermore, the strategy to start moving right away when possible was implemented as the strategy "Go" which is also a viable example. In the T-junction scenario without occlusions on the test set the most successful Deep Reinforcement learning model in terms of the success percentage (DDDQN.Acc.Base) was 1.66 times faster than the baseline TTC model (see Table 4.2). The deep learning model fails in one case on the test set where the truck has a frontal collision with a passenger vehicle making a left turn of the main road.

Table 4.1: Comparison between different models in the T-Junction scenario without occlusions on the validation set

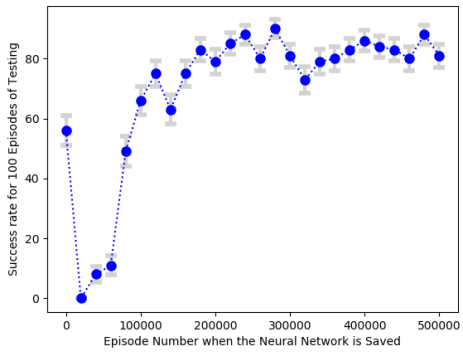
	Success %	Timeout %	Crash %	Average time (s)
TTC	100	0	0	74.875
DDDQN.Acc.Base (1)	100	0	0	33.960
DDDQN.Spd.Base (2)	100	0	0	65.405
DQN.Spd.Base (3)	100	0	0	46.105
DDDQN.Spd.Sml (4)	87	0	13	27.33
DDDQN.Spd.Dcd (5)	100	0	0	60.885
DDDQN.Spd.Big (6)	100	0	0	32.41
Go	61	0	39	27.595
Random (8 - Setspeed)	53	0	47	48.29
Random (9 - Acc)	59	0	41	53.615

Table 4.2: Comparison between different models in the T-Junction scenario without occlusions on the test set

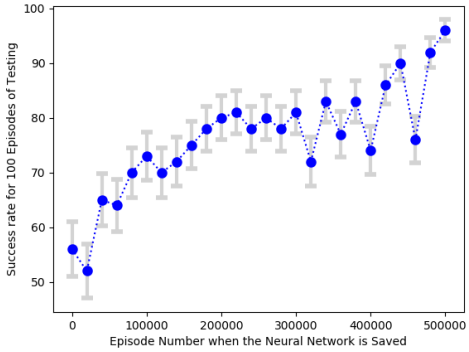
	Success %	Timeout %	Crash %	Average time (s)
TTC	93	7	0	56.03
DDDQN.Acc.Base (1)	99	0	1	33.595
DDDQN.Spd.Base (2)	93	4	3	51.1
DQN.Spd.Base (3)	93	0	7	35.305
DDDQN.Spd.Sml (4)	82	0	18	27.39
DDDQN.Spd.Dcd (5)	97	0	3	31.94
DDDQN.Spd.Big (6)	93	0	7	34.635
Go	70	0	30	27.885
Random (8 - Setspeed)	64	0	36	50.495
Random (9 - Acc)	60	0	40	53.68

4.2 Occluded T-Junction Scenario

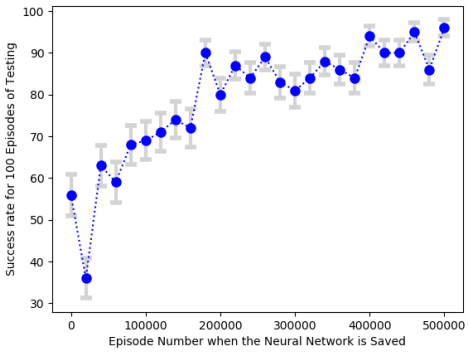
Similarly as in section 4.1, different Deep Reinforcement learning algorithms were compared with the baseline model TTC with the difference of the T-junction scenario having occlusions featured such as shown in Figure 3.13. The results for the 100 test cases with check-points from different models are featured on figure 4.3. As with the non-occluded scenario, test cases were randomly created, but kept fixed when evaluation the performance of different cases and the exploration was removed in the validation and test episodes. The models with the highest success rates were chosen and those were put together in table 4.3. Furthermore, these models were evaluated on a test set consisting of 100 random cases and the results from that testing were compiled into table 4.4.



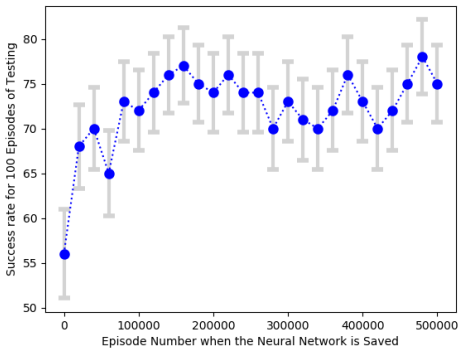
(a) DDDQN - Acceleration action space with a baseline neural network (1)



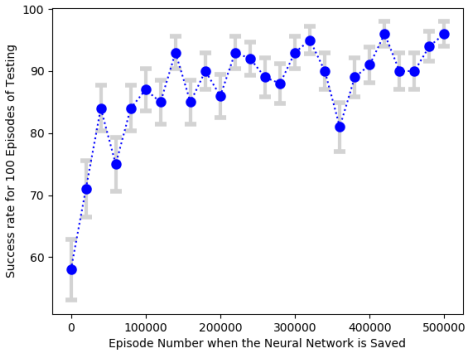
(b) DDDQN - Setspeeds action space with a baseline neural network (2)



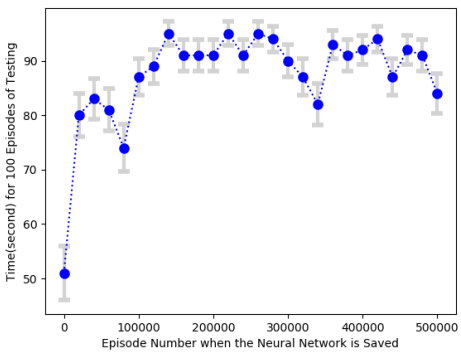
(c) DQN - Setspeeds action space with a baseline neural network (3)



(d) DDDQN - Acceleration action space with a small neural network (4)



(e) DDDQN - Setspeeds action space with a decoder based neural network (5)



(f) DDDQN - Setspeeds action space with a big neural network (6)

Figure 4.3: Results for six deep learning models in the occluded T-junction scenario

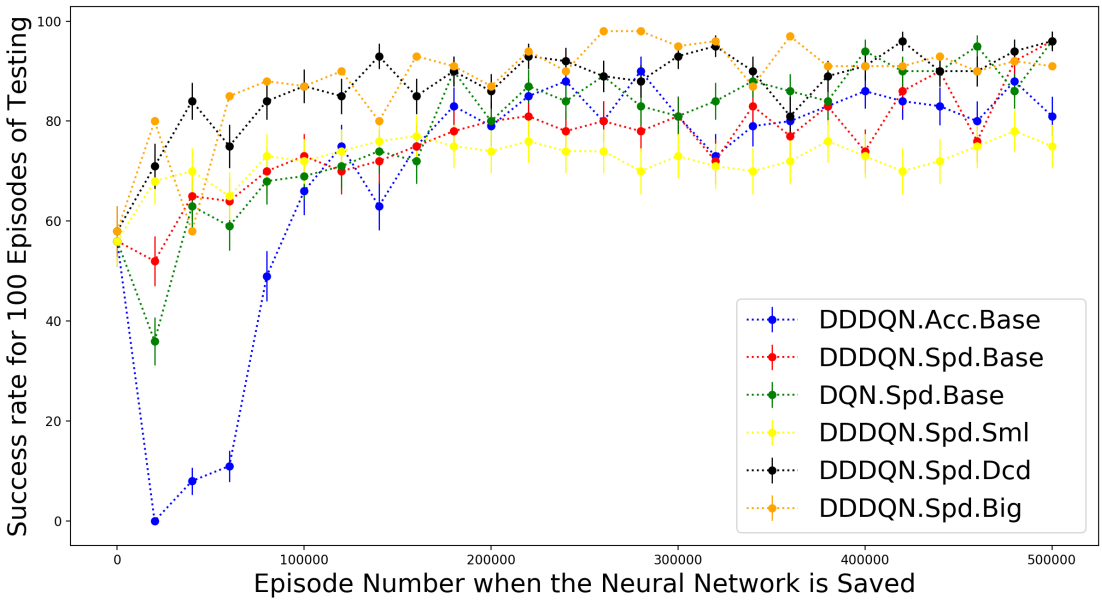


Figure 4.4: Combined results for six deep learning models in the occluded T-junction scenario

In the T-junction scenario with occlusions on the test set the most successful Deep Reinforcement learning model in terms of the success percentage (DDDQN.Spd.Dcd) was 1.75 times faster than the baseline TTC model (see Table 4.4). The deep learning model fails once on the test set, when the truck approaches the intersection and begins to accelerate to cross it. The truck brakes, but as stopping is not instant, it drives into the vehicle in the opposite lane.

Table 4.3: Comparison between different models in the occluded T-Junction scenario on the validation set

	Success %	Timeout %	Crash %	Average time (s)
TTC	100	0	0	75.695
DDDQN.Acc.Base (1)	93	3	4	46.93
DDDQN.Spd.Base (2)	96	0	4	56.045
DQN.Spd.Base (3)	96	0	4	40.665
DDDQN.Spd.Sml (4)	82	0	18	30.55
DDDQN.Spd.Dcd (5)	98	0	2	33.165
DDDQN.Spd.Big (6)	100	0	0	59.195
Go	61	0	39	27.595
Random (8 - Setspeed)	53	0	47	48.29
Random (9 - Acc)	59	0	41	53.615

Table 4.4: Comparison between different models in the occluded T-Junction scenario on the test set

	Success %	Timeout %	Crash %	Average time (s)
TTC	93	7	0	57.0
DDDQN.Acc.Base (1)	88	4	8	42.68
DDDQN.Spd.Base (2)	96	2	2	47.975
DQN.Spd.Base (3)	96	0	4	34.3
DDDQN.Spd.Sml (4)	77	0	23	33.93
DDDQN.Spd.Dcd (5)	99	0	1	32.41
DDDQN.Spd.Big (6)	96	0	4	36.9
Go	70	0	30	27.885
Random (8 - Setspeed)	64	0	36	50.495
Random (9 - Acc)	60	0	40	53.68

4.3 Roundabout Scenario

To show the generality of DQN methods, one DDDQN model was trained and compared with the baseline TTC model and tested on 100 random test cases. The results are brought out in table 4.5. The best DQN model was 1.44 times faster than TTC in the roundabout scenario on the test set. The deep learning model failed 9 times on the test set - most often by frontal collisions, but collisions from the back were also apparent.

Table 4.5: Comparison between TTC and the DDDQN algorithm in the roundabout scenario

	Success %	Timeout %	Crash %	Average time (s)
TTC	91	9	0	115.955
DDDQN.Spd.Base (2)	91	0	9	80.19
Go	61	0	39	45.24
Random (8 - Setspeed)	33	0	67	62.165
Random (9 - Acc)	31	0	69	64.44

4.4 Model trained on the T-junction and the Roundabout scenario

As a second way to show the generality of DQN methods, one DDDQN model was trained on both the T-junction and roundabout scenarios and compared with the baseline TTC model and tested on 100 random test cases. The results are brought out in table 4.6.

Table 4.6: Comparison between TTC and the DDDQN algorithm in the T-junction and the roundabout scenario

	Success %	Timeout %	Crash %	Average time (s)
TTC	93	7	0	85.57
DDDQN.Spd.Big (6) Best	84	0	16	53.77

4.5 Comparison of Action Spaces

This work featured two different action spaces as shown in subsection 3.2.2. These action spaces are compared on training the T-Junction scenario with the baseline neural network architecture for 500 000 steps. Even though the episode reward comparison can be seen in Figure 4.5, the actual performance can be compared on Tables 4.2 and 4.4.

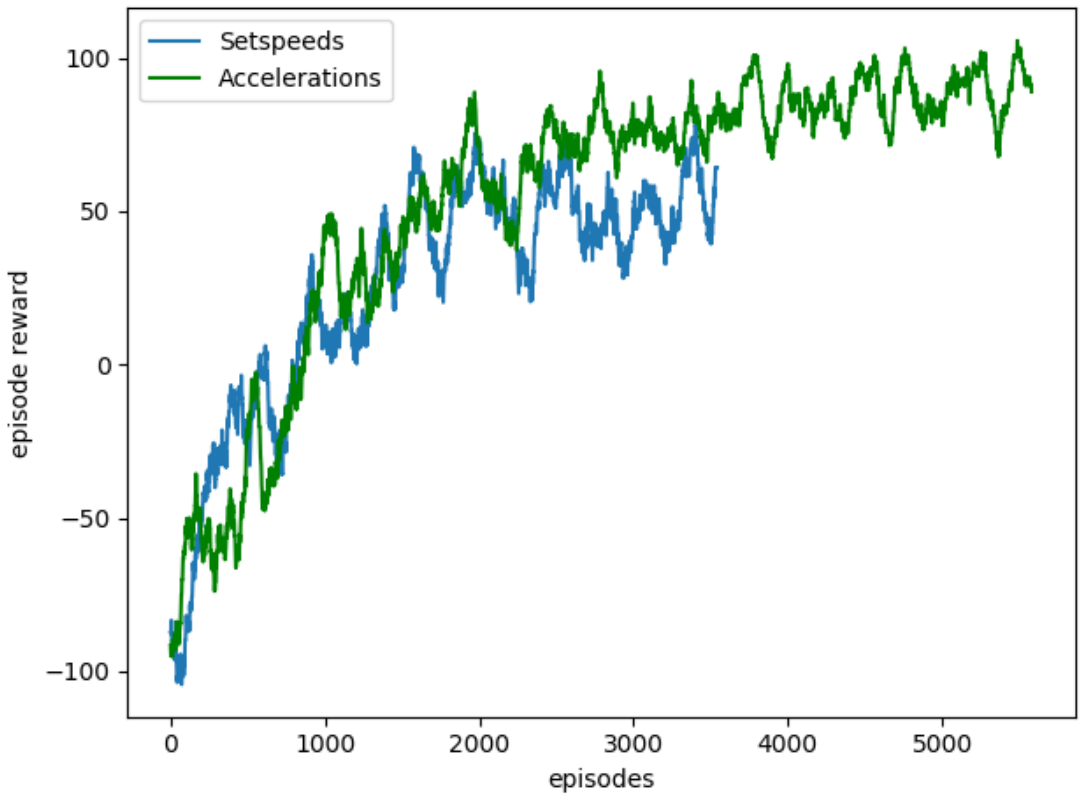


Figure 4.5: A comparison between the Setspeeds and the Accelerations action spaces using the DDDQN algorithm

4.6 Comparison of Different Neural Network Architectures

The episode reward for different networks as seen in figure 3.6 was compared using a DDDQN algorithm on the occluded T-junction scenario for 500 000 steps. The results can be seen on figure 4.6. The results correspond to algorithms 2 ("Baseline"), 4 ("Small"), 5 ("Decoder") and 6 ("Big") from section 4.1.

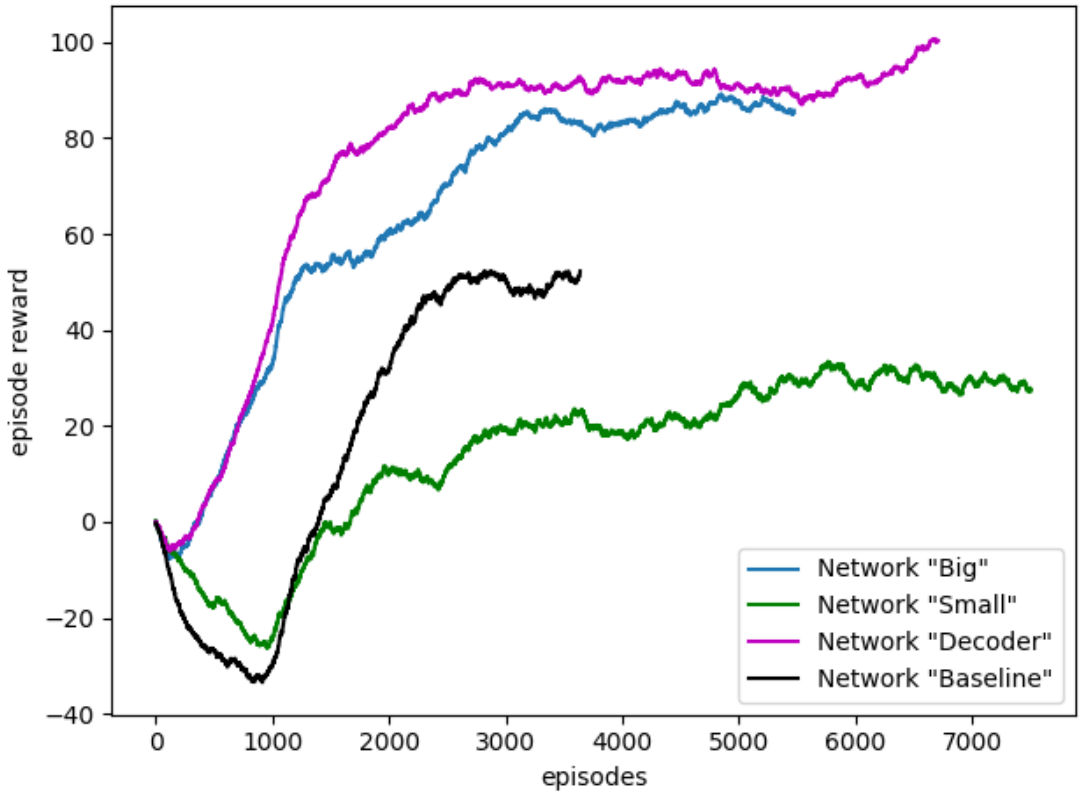


Figure 4.6: A comparison between different networks using the DDDQN algorithm

5

Discussion

This chapter analyses the previous results chapter by discussing the performance and behaviour of algorithms, the effects of the neural network choices and the different action spaces used to train the deep reinforcement learning models.

5.1 Deep Q-Network compared to Dueling Double Q-Network

Comparing DQN and DDDQN is done by keeping the main architecture, action and state spaces, the reward function and the training parameters fixed. However, DDDQN has additional two streams to the main network architecture - one estimating the state-value function and the other one called "advantage" which is summed up to reach an action-value function as mentioned in section 2.11. For the rest of the comparison of DDDQN.Spd.Base(2) and DQN.Spd.Base(3) are used.

Both DDDQN.Spd.Base(2) and DQN.Spd.Base(3) have the same success rate for occluded and non-occluded scenarios in validation and test sets. In the time metric DQN.Spd.Base(3) seems to complete faster than the DDDQN.Spd.Base(2) in all cases. For example, in the occluded scenario test set DQN.Spd.Base(3) has an average time of 34.3 while DDDQN.Spd.Base(2) has 47.9 seconds. If we investigate crashes and timeouts on occluded and non-occluded test cases it can be seen that DQN.Spd.Base(3) has more crashes than DDDQN.Spd.Base(2) while DQN.Spd.Base(3) has 0 timeouts and DDDQN.Spd.Base(2) has timeouts occasionally. Considering the average time and crash-timeout distribution DQN.Spd.Base(3) seems to have a faster and a more aggressive policy than DDDQN.Spd.Base(2). On the other hand, DDDQN.Spd.Base(2) avoids some of the collisions by not interfering the traffic and reduces its crash rate by making a timeout.

Considering both methods having the same success rate and DDDQN.Spd.Base(2) having less crashes - this makes it a safer choice. DDDQN.Spd.Base(2) sacrifices time while keeping safer results in comparison with DQN.Spd.Base(3).

5.2 Neural Network Size

A comparison of the the small network architecture DDDQN.Spd.Sml(4) with a baseline architecture DDDQN.Spd.Base(2) which has a higher number of neurons

and layers, but they have the same Setspeed action space, reward function, state space, and the training parameters, is presented. On testing, the small network can perform with a 82% success rate in the non-occluded and with a 77% success rate in the occluded case while the bigger network has percentages of 93% in the non-occluded and 96% in the occluded accordingly. This shows that the bigger network performed better than the smaller network in terms of successes in the scenarios. Furthermore, by investigating the validation results, the smaller network has the highest success rate of 87% for the non-occluded case and 82% for the occluded case, while the bigger network has a perfect 100% success rate for the non-occluded case and a 96% percent success rate for the occluded case. When compared to a bigger network, the small network has a success percentage gap around 11% for non-occluded and 19% for occluded test case. Episodic reward graphs shown in figure 4.6 indicates that the small network (Network "small") seems to plot a worse profile during obtaining rewards when compared with the bigger (Network "baseline") network. Additionally by checking the loss function figures A.1 and A.4 the small network has a bigger and a shakier loss profile during training. Considering the success rate gap, the reward and loss graph comparisons, the smaller network might have underfitting issues about estimating the Q-values. On the other hand, the bigger network does not seem to suffer from overfitting or underfitting since the test and validation results are close and the test results are satisfactory.

5.3 Action Spaces

The comparison of action spaces is made by using the DDDQN algorithm.
DDDQN.Acc.Base(1) uses the acceleration-based action space and
DDDQN.Spd.Base(2) uses the Setspeed action space but all the other parameters of training are kept the same for comparison reasons. During the training, the Acceleration and the Setspeed action space both show quite stable success rates in the validation test cases and display a converging trend after 200 000 episodes as it can be seen in Figure 4.3.

It is interesting to note that under the same amount of training steps (500 000), the Acceleration action space trained around 5000 episodes while action space trained around 3500 episodes. This means that the average time spent per episode is bigger for the Setspeed than the Acceleration action space. From tables 4.1, 4.2, 4.4 and 4.3 it can be seen that the Acceleration action space is faster than the Setspeed actions in all cases and for both testing and validating. Furthermore, for the non-occluded scenario, the Acceleration action space is much faster. It can be seen from the table 4.1 that DDDQN.Acc.Base(1) has an average time of 33.595 seconds while DDDQN.Spd.Base(2) has an average time of 51.1 seconds. Likewise, the Acceleration action space has a better success percentage when compared with Setspeed action space in non-occluded case: 99% to 93%. It can be concluded that using the Acceleration action space had better time and success results for the non-occluded case. Nevertheless, the Setspeed action space shows that it is more advantageous when training it with on the occluded scenario. Even though the test time performance is slightly worse than the Acceleration action space with times of 47.9 and

42.7 seconds, the success rate is better. The Setspeed action space solves 96% of the test cases successfully while the Acceleration action space can only solve 88% at best making the speed based action space more eligible for the occluded scenario.

5.4 Behavioral Analysis Between Different Methods

The results show that the main difference between the trained reinforcement learning models and TTC is that the reinforcement learning models solve the scenarios faster. In the non-occluded T-junction, the best model is 1.66 times faster, in the occluded T-junction, 1.75 times faster and in the roundabout, 1.44 times faster on the test sets - which on an average is 1.61 times faster. On the other hand, they are at the same time more prone to crashes. In the T-Junction scenario, Figure 5.1 is an example of how reinforcement learning models solve the scenarios faster. The TTC model moves to the intersection with a cruising speed and then slowly starts to creep that the field of view would cover all of the roads whereas the reinforcement learning model slows down close to the intersection, but does it in a very hasty manner. Nevertheless, out of 100 test scenarios it only fails once and that surprisingly happens in a non-occluded setting. When the truck approaches the intersection and begins to accelerate to cross it, the truck brakes. As stopping is not instant, it drives into the vehicle in the opposite lane.

In the roundabout scenario as seen on figure 5.2, the TTC model moves to the intersection more quickly than the reinforcement learning model but waits there until there are no other vehicles on the roundabout. On the other hand, the reinforcement learning model seems very cautious when entering the simulation - it constantly breaks and moves with small increments. It can be said that this is far from optimal behaviour. Despite this, it eventually reaches the roundabout and stops to wait for the right time to enter the roundabout. When the distance to the next vehicle coming from the left on the roundabout is big enough, the reinforcement learning model enters the roundabout and subsequently exits the roundabout as originally planned. When the reinforcement learning model finishes the scenario, the TTC model is still waiting to enter the roundabout.

5.5 Deep Reinforcement Learning Algorithm in the Roundabout Scenario

The idea to train a DDDQN algorithm on a different scenario than the T-Junction scenario and then to compare it with TTC was based on the reasoning to show that the reinforcement learning method is general. As the reinforcement learning method reaches a 91% success rate as does the TTC, it can be said that they are most certainly comparable. Nevertheless, TTC does not get into any crashes as the reinforcement learning method does - it rather reaches a timeout state. On the other hand, the reinforcement learning method solves the scenarios over 30 seconds faster

per scenario than the TTC method on average, which is also similar to the results concerning the T-Junction scenario.

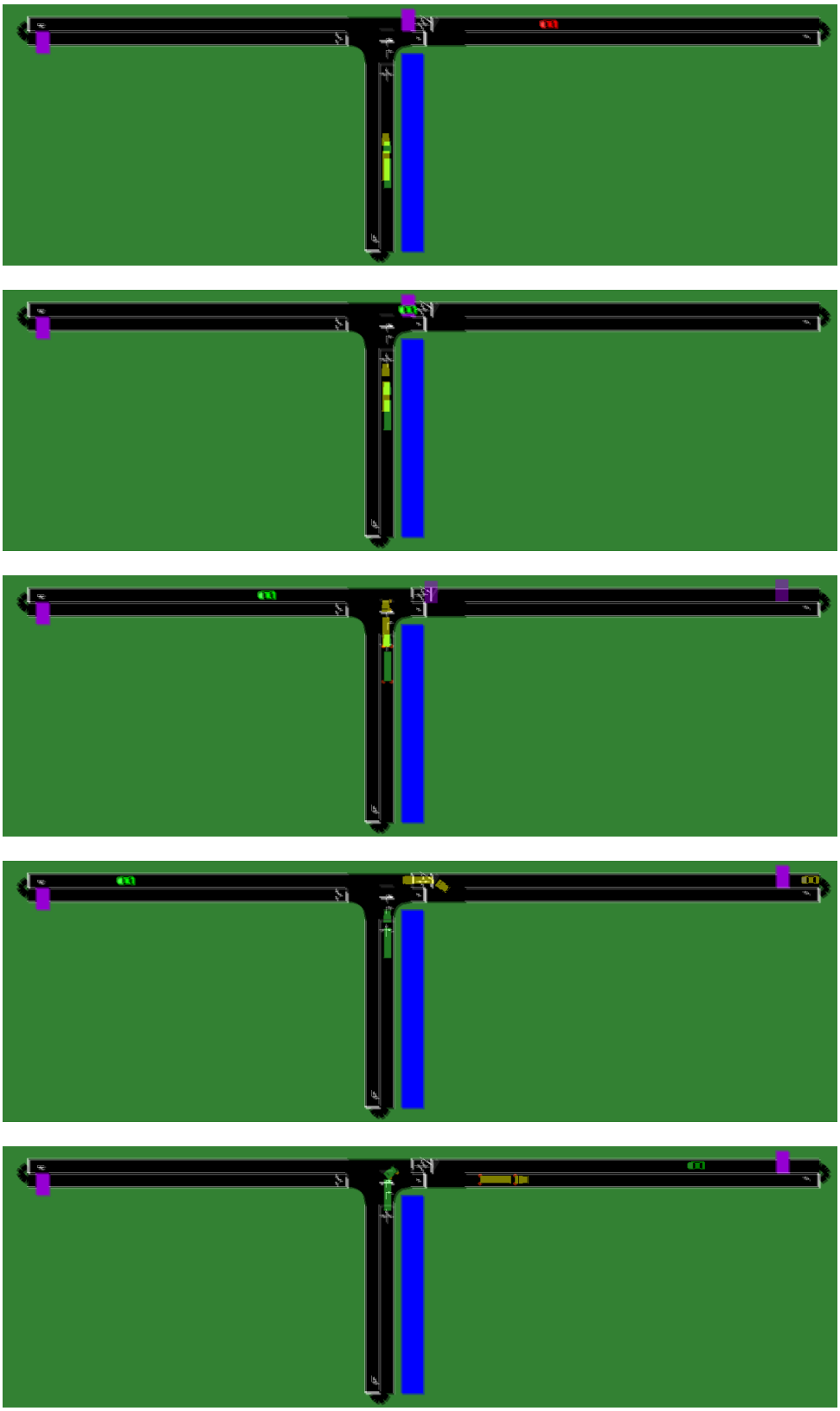


Figure 5.1: TTC vs. the best reinforcement learning model (5) in the T-Junction scenario, time between frames is 2 seconds, TTC corresponds to the green truck and the reinforcement learning model corresponds to the yellow truck

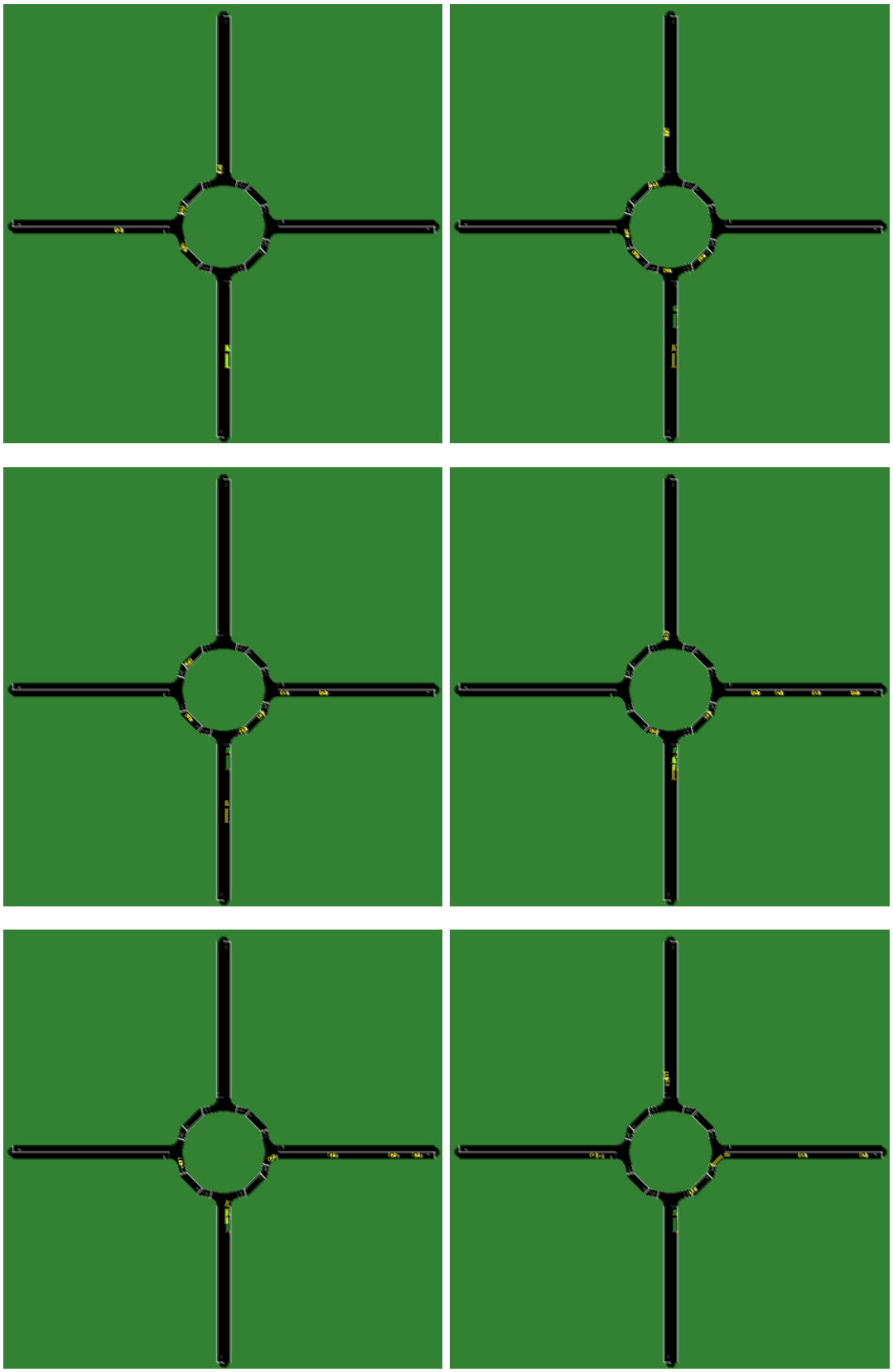


Figure 5.2: A reinforcement learning model in the roundabout scenario compared to TTC, time between frames is 5 seconds, TTC corresponds to the green truck, the reinforcement learning model corresponds to the yellow truck

5.6 Deep Reinforcement Learning Algorithm in Both T-junction and Roundabout Scenarios

As a final way to show possible generality of the DQN methods, a reinforcement learning algorithm was trained on both the T-junction and roundabout scenarios. The results were not as successful as the TTC method - 93% compared to a 84% success rate. Furthermore, TTC did not crash even once, while the DDDQN.Spd.Big(6) algorithm crashed 16% of the times. Despite this, a 84% success rate with the DDDQN.Spd.Big(6) algorithm can be seen as a big step forward as it was trained for the same amount of steps as all the other algorithms, but for two scenarios at the same time. For achieving better results, algorithm optimisation and prolonging the training can be considered for future work.

5.7 Safety Issues

In case these algorithms were to be modified and transferred into real life, it is crucial that there would further safety measures implemented. Since vehicles are often manned by people and valuable assets a crash should be avoided at any cost. In our experiments, most of the test cases have a non-zero crash ratio. This problem could be overcome by introducing an architecture which combines Deep reinforcement learning and hard constraints for eliminating crash cases. This means that a separate safety layer should be established which double-checks the decisions made by the reinforcement learning agent.

For example, Shalew-Schwartz, Ben-Zrihem, Cohen and Shashua have proposed a complementary method for ensuring safety criteria to eliminate collisions [41]. A DQN policy was used with hard-coded safety constraints in a two-way merging scenario. In another study [42], Q-masking technique was proposed, which combines TTC and Deep reinforcement learning methods by mirroring hard safety constraints of the former paper. This method used the actions of DQN policy generally, but in dangerous cases, TTC based actions are used. This yielded a result with no collisions and a better time metric than the TTC method in a highway scenario.

The previously mentioned hybrid methods of Deep reinforcement learning and hard constraints could be more successful in eliminating crashes than TTC alone in urban scenario and implementing such a method for urban scenarios is promising for future work.

6
Conclusion and Future Work

This chapter summarizes the results from the previous chapter and answers the research questions proposed at the beginning of this thesis. Finally, possible future improvements are brought out.

6.1 Strengths and Weaknesses of Different Models

On the examples of the urban scenarios used in this thesis, it can be concluded that the strengths of deep reinforcement learning methods lie in solving tasks faster on average than TTC. In cases where the reinforcement learning models have the same or a better success rate, they are on average 1.61 times faster. Secondly, they are more general and do not need as much modification as classical methods. The greater amount of optimisation which has to be done with the classical methods to achieve desirable results in terms of performance for different cases in traffic compared to the reinforcement learning approach favors the latter as it may not be feasible to do this optimisation in every situation or scenario. Nevertheless, at least in our test cases, these methods do not surpass the success rate offered by the classical baseline model, but they are certainly comparable.

6.2 Formulating Urban Scenarios for Tactical Decision-making as a Partially Observable Markov Decision Process

In this work, multiple urban scenarios were formulated as a (PO)MDP composed of a state space with 49 elements, two different action spaces and a reward space composed of 6 elements. Results show that the final proposed (PO)MDP is sufficient enough to produce good enough results in all of the three urban scenarios used in this work - the best reinforcement learning models except in the multi-scenario case all surpassed a 90% success rate and had a lower average execution time than the base model. From the two action spaces, it was shown that the action space which mimicked a more high level decision making achieved better success rates in test scenarios than the action space which dealt directly with accelerations on the heavy vehicle.

6.3 Future Work

For future work of this thesis there are several aspects which were not featured, but are quite relevant for future purposes:

- Taking into account the friction factor of the road when training the reinforcement learning agent.
- Having more scenarios to test on, for the state space could be extended and be more general.
- Using Prioritized Experience Replay while training deep reinforcement learning models to see how it affects the performance [43].
- Testing the convolutional layer based weight sharing architecture often used in the literature on the scenarios provided by this work, which can the training faster and more accurate [44].
- Bayesian networks can be investigated which could give a performance boost to the success rate of solving the scenarios proposed in this work.
- Other cars' behavior could be extracted from real data and used with this work which would make the scenarios more closer to real life.
- DQN with having different lengths for the trailer can be trained which would make the model more universal.
- TTC and DQN hybrid methods can be investigated - POMDP formulation could carry TTC values. One option is inputting the TTC values into the state. The second value would enable the reward function to utilize TTC values [41].
- Training the agent with noisy sensor data - this would make the scenarios resemble more real life, because there is always sensor noise in measurements.
- The models can be trained with another algorithm such as Rainbow which has shown promise.
- Filling the experience replay memory with TTC actions and letting the DQN polish them can be a technique for hybrid methods which has the potential for training these kinds of models faster [45].
- The information that is inputted to the agent can be preprocessed further. The data processing can be in favor of data efficiency or helping the algorithm converge more easily. For example in the work of Isele et. al. a compact state space was proposed with semantic values rather than numeric values. [19]

Bibliography

[1] Peter Nilsson. *Traffic situation management for driving automation of articulated heavy road transports*. PhD thesis, Thesis for the degree of doctor of philosophy in machine and vehicle systems, 2017.

[2] United Nations. Transforming our world: the 2030 Agenda for Sustainable Development. Technical Report October, 2015.

[3] National Highway Traffic Safety Adminstriation. 2016 Fatal Motor Vehicle Crashes: Overview. *Traffic Safety Facts*, (October):1–9, 2017.

[4] World Health Organization. Road traffic injuries. {https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries}, 2018.

[5] RAC Foundation. Keeping the Nation Moving: Facts on Parking. (October), 2012.

[6] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical review E*, 62(2):1805, 2000.

[7] Arne Kesting, Martin Treiber, and Dirk Helbing. General lane-changing model mobil for car-following models. *Transportation Research Record*, 1999(1):86–94, 2007.

[8] Richard Van Der Horst and Jeroen Hogema. Time-to-collision and collision avoidance systems. 1993.

[9] Ioannis Antonoglou, Andreas K. Fidjeland, Daan Wierstra, Helen King, Marc G. Bellemare, Shane Legg, Stig Petersen, Martin Riedmiller, Charles Beattie, Alex Graves, Amir Sadik, Koray Kavukcuoglu, Georg Ostrovski, Joel Veness, Andrei A. Rusu, David Silver, Demis Hassabis, Dharshan Kumaran, and Volodymyr Mnih. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[10] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[11] Maxime Bouton, Akansel Cosgun, and Mykel J Kochenderfer. Belief State

Planning for Autonomously Navigating Urban Intersections. *CoRR*, abs/1704.0, 2017.

[12] Daniel Althoff, Constantin Hubmann, Christoph Stiller, Marvin Becker, and Jens Schulz. Automated Driving in Uncertain Environments: Planning With Interaction and Uncertain Maneuver Prediction. *IEEE Transactions on Intelligent Vehicles*, 3(1):5–17, 2018.

[13] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction (second edition)*. The MIT press, Cambridge, Massachusetts, 2018.

[14] Carl-Johan Hoel, Krister Wolff, and Leo Laine. Automated Speed and Lane Change Decision Making using Deep Reinforcement Learning. *CoRR*, abs/1803.1, 2018.

[15] Tommy Tram, Anton Jansson, Robin Grönberg, Mohammad Ali, and Jonas Sjöberg. Learning Negotiating Behavior between Cars in Intersections using Deep Q-Learning. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 2018-Novem:3169–3174, 2018.

[16] Shai Shalev-Shwartz, Nir Ben-Zrihem, Aviad Cohen, and Amnon Shashua. Long-term Planning by Short-term Prediction. pages 1–9, 2016.

[17] V. Sezer, T. Bandyopadhyay, D. Rus, E. Frazzoli, and D. Hsu. Towards autonomous navigation of unsignalized intersections under uncertainty of human driver intent. pages 3578–3585, Sep. 2015.

[18] David Isele, Reza Rahimi, Akansel Cosgun, Kaushik Subramanian, and Kikuo Fujimura. Navigating Occluded Intersections with Autonomous Vehicles using Deep Reinforcement Learning. pages 2–7, 2017.

[19] Peter Wolf, Karl Kurzer, Tobias Wingert, Florian Kuhnt, and J Marius Zöllner. Adaptive Behavior Generation for Autonomous Driving using Deep Reinforcement Learning with Compact Semantic States. In *2018 {IEEE} Intelligent Vehicles Symposium, {IV} 2018, Changshu, Suzhou, China, June 26-30, 2018*, pages 993–1000, 2018.

[20] Changjian Li and Krzysztof Czarnecki. Urban Driving with Multi-Objective Deep Reinforcement Learning. (1), 2018.

[21] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. On a formal model of safe and scalable self-driving cars. *CoRR*, abs/1708.06374, 2017.

[22] M.J. Kochenderfer. *Decision Making Under Uncertainty: Theory and Application*. MIT Lincoln Laboratory series. MIT Press, 2015.

[23] Philip S. Thomas. A notation for markov decision processes. *CoRR*, abs/1512.09075, 2015.

[24] Michael L. Littman. A tutorial on partially observable Markov decision processes. *Journal of Mathematical Psychology*, 53(3):119–125, 2009.

[25] Richard Bellman et al. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515, 1954.

[26] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015.

[27] Matthias Plappert. keras-rl. `https://github.com/keras-rl/keras-rl`, 2016.

[28] Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bulletin*, 2(4):160–163, 1991.

[29] D. A. Vaccari and E. Wojciechowski. Neural networks as function approximators: teaching a neural network to multiply. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 4, pages 2217–2222 vol.4, June 1994.

[30] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[31] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. pages 1–9, dec 2013.

[32] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling Network Architectures for Deep Reinforcement Learning. (9), 2015.

[33] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[34] Nicolò Cesa-Bianchi, Claudio Gentile, Gábor Lugosi, and Gergely Neu. Boltzmann exploration done right. *CoRR*, abs/1705.10257, 2017.

[35] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining Improvements in Deep Reinforcement Learning. 2017.

[36] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker. SUMO - Recent Development and Applications of {SUMO - Simulation of Urban MObility}. *International Journal On Advances in Systems and Measurements*, 5(3):128–138, 2012.

[37] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[38] Python shadowcasting implementation - RogueBasin. http://www.roguebasin.com/index.php?title=PythonShadowcastingImplementation.

[39] Simulation of Urban Mobility. Simulation/randomness. https://sumo.dlr.de/wiki/Simulation/Randomness, 2018.

[40] Ronald Miller and Qingfeng Huang. An adaptive peer-to-peer collision warning system. *IEEE Vehicular Technology Conference*, 1:317–321, 2002.

[41] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving. 2016.

[42] Mustafa Mukadam. Tactical Decision Making for Lane Changing with Deep Reinforcement Learning. 2017.

[43] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2015.

[44] Carl-Johan Hoel, Krister Wolff, and Leo Laine. Automated speed and lane change decision making using deep reinforcement learning. *CoRR*, abs/1803.10056, 2018.

[45] Tim Brys, Anna Harutyunyan, Halit Bener Suay, Sonia Chernova, Matthew E. Taylor, and Ann Nowé. Reinforcement learning from demonstration through shaping. In *IJCAI*, 2015.

A

Appendix 1

A.1 Full Training Info

The training data for the following algorithms is brought out on figures A.1, A.2, A.3, A.4, A.5 and A.6 respectively:

1. DDDQN - Acceleration action space with a baseline neural network, shortened as DDDQN.Acc.Base (Yellow)
2. DDDQN - Setspeeds action space with a baseline neural network, shortened as DDDQN.Spd.Base (Black)
3. DQN - Setspeeds action space with a baseline neural network, shortened as DQN.Spd.Base (Red)
4. DDDQN - Setspeeds action space with a small neural network, shortened as DDDQN.Spd.Sml (Green)
5. DDDQN - Setspeeds action space with a decode neural network, shortened as DDDQN.Spd.Dcd (Violet)
6. DDDQN - Setspeeds action space with a big neural network, shortened as DDDQN.Spd.Big (Blue)

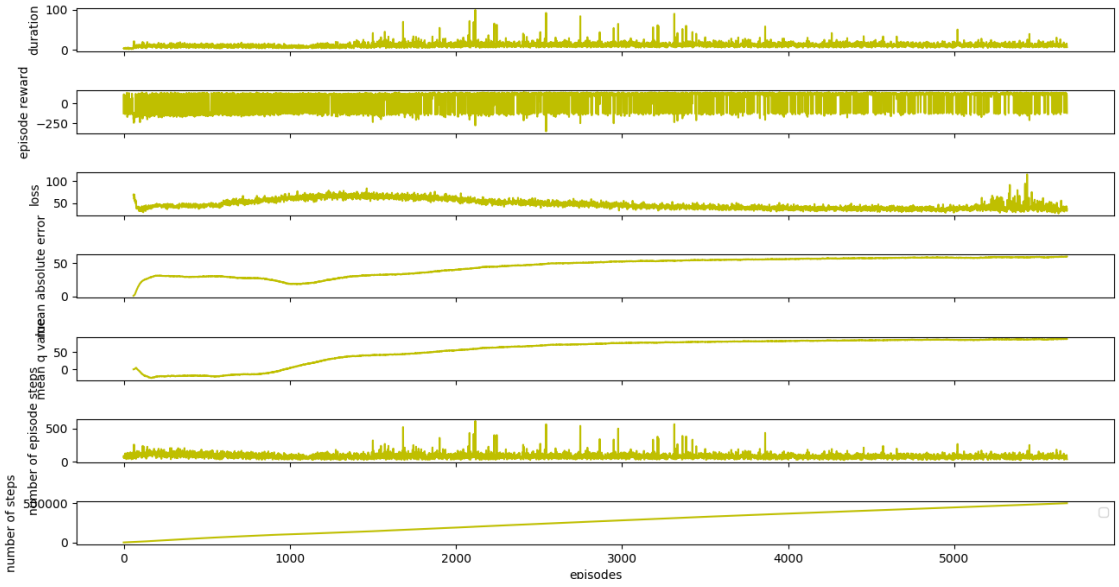


Figure A.1: Training data for algorithm 1

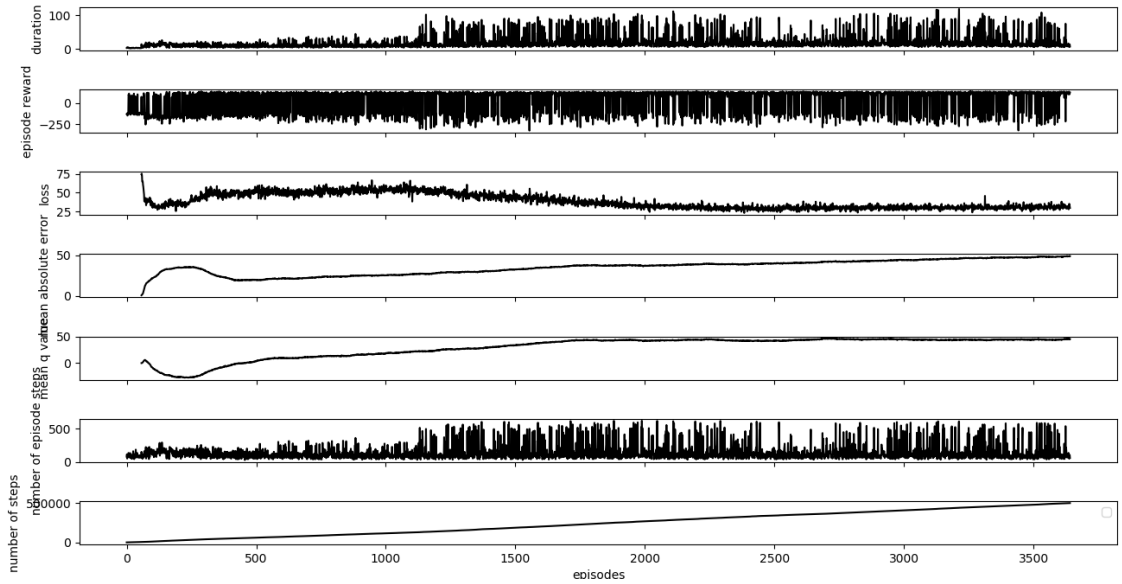


Figure A.2: Training data for algorithm 2

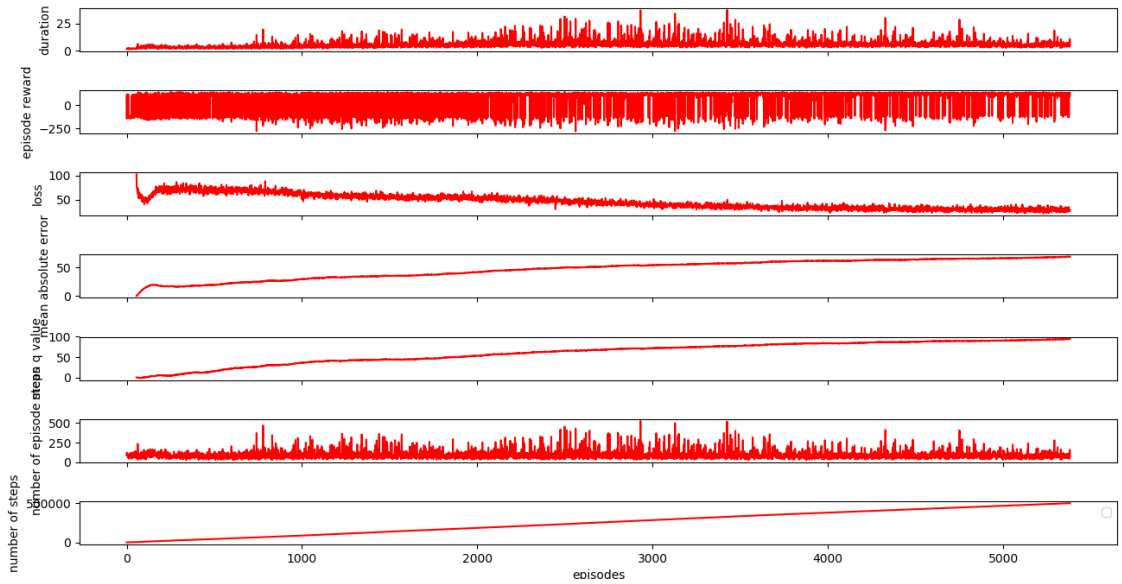


Figure A.3: Training data for algorithm 3

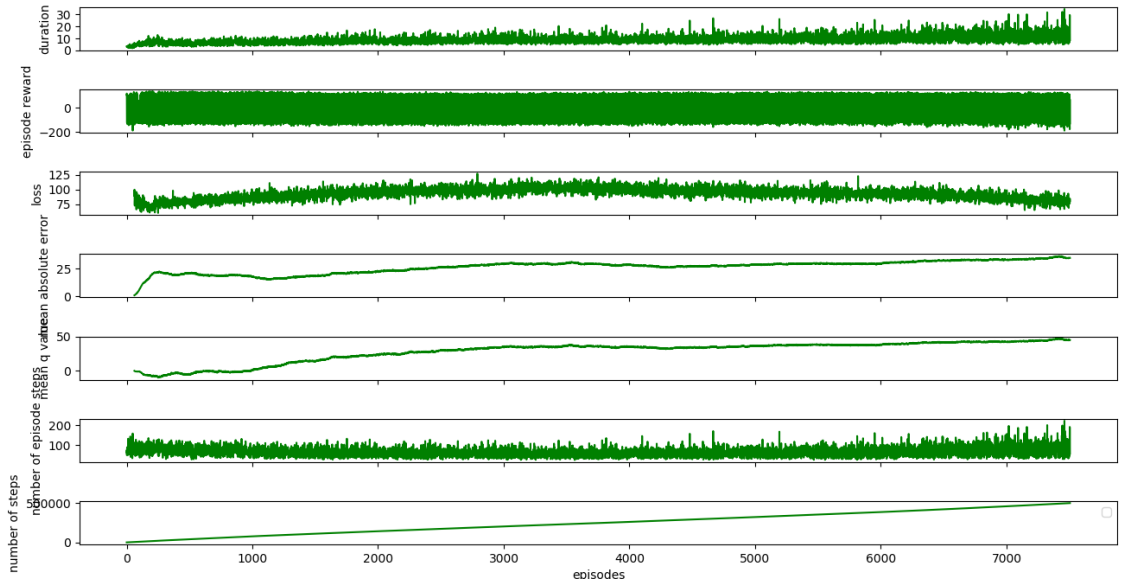


Figure A.4: Training data for algorithm 4

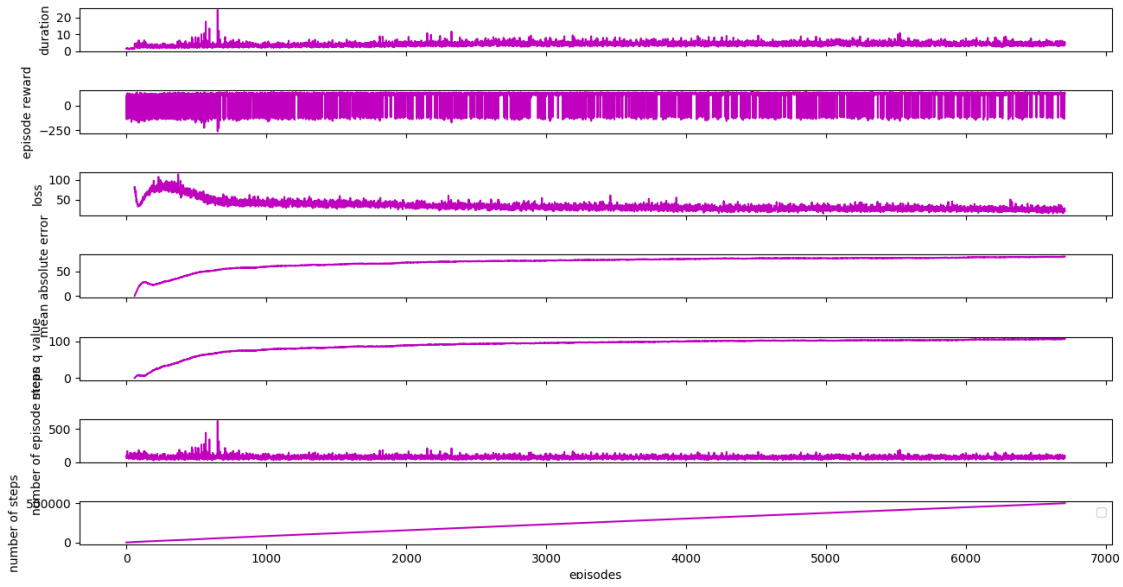


Figure A.5: Training data for algorithm 5

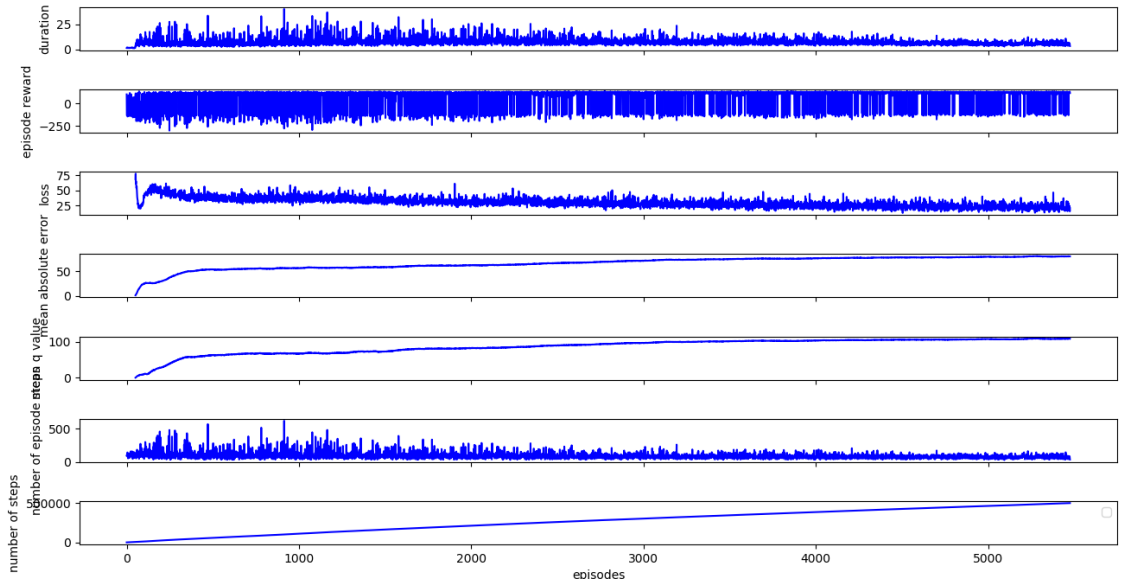


Figure A.6: Training data for algorithm 6

For figure A.7, a,b,c,d,e and f are the descriptions for the training set, g,h,i,j and k are the descriptions for the validation set and l,m,n,o and p are the descriptions for the test set for the T-Junction scenario. For figure A.8, q,r,s and t are the descriptions for the training set and u,v,w and x are the descriptions for the test set for the roundabout scenario.

```
<routes>
    <vType accel="5.0" decel="0.0" emergencyDecel="0.0" id="Car" length="4.0" maxSpeed="16" impatience="0.99" speedFactor="normc
(1,0.7,0.1,1.9)" speedDev="0.7"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="typeSN" length="3"  minGap="0.0" maxSpeed="14" vClass="passenger" guiShape="truck"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="trailerType" length="7" minGap="0.0"  maxSpeed="14" guiShape="bus"/>

    <route id="route_ew" edges="e_0 0_w"/>
    <route id="route_es" edges="e_0 0_s"/>
    <route id="route_we" edges="w_0 0_0_3 0_3_e"/>
    <route id="route_ws" edges="w_0 0_s"/>
    <route id="route_sw" edges="s_0 0_w"/>
    <route id="route_see2" edges="s_0 0_0_2 0_2_e_2 e_2_0_3 0_3_e"/>

    <flow id="flow_e_w" route="route_ew" type="Car" begin="0" end="120" probability="0.01" departSpeed="12" />
    <flow id="flow_w_e" route="route_we" type="Car" begin="2" end="130" probability="0.01" departSpeed="12" />
    <flow id="flow_w_s" route="route_ws" type="Car" begin="4" end="140" probability="0.01" departSpeed="12" />
    <flow id="flow_e_s" route="route_es" type="Car" begin="5" end="150" probability="0.01" departSpeed="12" />
    <flow id="flow_ew_" route="route_ew" type="Car" begin="60" end="160" probability="0.01" departSpeed="12" />
    <flow id="flow_we_" route="route_we" type="Car" begin="70" end="170" probability="0.01" departSpeed="12" />

</routes>
```

(a)

```
<routes>
    <vType accel="5.0" decel="0.0" emergencyDecel="0.0" id="Car" length="4.0" maxSpeed="16" impatience="0.99" speedFactor="normc
(1,0.7,0.1,1.9)" speedDev="0.7"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="typeSN" length="3"  minGap="0.0" maxSpeed="14" vClass="passenger" guiShape="truck"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="trailerType" length="7" minGap="0.0"  maxSpeed="14" guiShape="bus"/>

    <route id="route_ew" edges="e_0 0_w"/>
    <route id="route_es" edges="e_0 0_s"/>
    <route id="route_we" edges="w_0 0_0_3 0_3_e"/>
    <route id="route_ws" edges="w_0 0_s"/>
    <route id="route_sw" edges="s_0 0_w"/>
    <route id="route_see2" edges="s_0 0_0_2 0_2_e_2 e_2_0_3 0_3_e"/>

    <flow id="flow_e_w" route="route_ew" type="Car" begin="0" end="120" probability="0.10" departSpeed="10" />
    <flow id="flow_w_e" route="route_we" type="Car" begin="2" end="130" probability="0.10" departSpeed="10" />
    <flow id="flow_w_s" route="route_ws" type="Car" begin="4" end="140" probability="0.10" departSpeed="10" />
    <flow id="flow_e_s" route="route_es" type="Car" begin="5" end="150" probability="0.10" departSpeed="10" />
    <flow id="flow_ew_" route="route_ew" type="Car" begin="60" end="160" probability="0.10" departSpeed="10" />
    <flow id="flow_we_" route="route_we" type="Car" begin="70" end="170" probability="0.10" departSpeed="10" />

</routes>
```

(b)

```
<routes>
    <vType accel="5.0" decel="0.0" emergencyDecel="0.0" id="Car" length="4.0" maxSpeed="16" impatience="0.99" speedFactor="normc
(1,0.7,0.1,1.9)" speedDev="0.7"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="typeSN" length="3"  minGap="0.0" maxSpeed="14" vClass="passenger" guiShape="truck"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="trailerType" length="7" minGap="0.0"  maxSpeed="14" guiShape="bus"/>

    <route id="route_ew" edges="e_0 0_w"/>
    <route id="route_es" edges="e_0 0_s"/>
    <route id="route_we" edges="w_0 0_0_3 0_3_e"/>
    <route id="route_ws" edges="w_0 0_s"/>
    <route id="route_sw" edges="s_0 0_w"/>
    <route id="route_see2" edges="s_0 0_0_2 0_2_e_2 e_2_0_3 0_3_e"/>

    <flow id="flow_e_w" route="route_ew" type="Car" begin="0" end="120" probability="0.15" departSpeed="9" />
    <flow id="flow_w_e" route="route_we" type="Car" begin="2" end="130" probability="0.15" departSpeed="9" />
    <flow id="flow_w_s" route="route_ws" type="Car" begin="4" end="140" probability="0.15" departSpeed="9" />
    <flow id="flow_e_s" route="route_es" type="Car" begin="5" end="150" probability="0.15" departSpeed="9" />
    <flow id="flow_ew_" route="route_ew" type="Car" begin="60" end="160" probability="0.15" departSpeed="9" />
    <flow id="flow_we_" route="route_we" type="Car" begin="70" end="170" probability="0.15" departSpeed="9" />

</routes>
```

(c)

```
<routes>
    <vType accel="5.0" decel="0.0" emergencyDecel="0.0" id="Car" length="4.0" maxSpeed="16" impatience="0.99" speedFactor="normc
(1,0.7,0.1,1.9)" speedDev="0.7"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="typeSN" length="3"  minGap="0.0" maxSpeed="14" vClass="passenger" guiShape="truck"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="trailerType" length="7" minGap="0.0"  maxSpeed="14" guiShape="bus"/>

    <route id="route_ew" edges="e_0 0_w"/>
    <route id="route_es" edges="e_0 0_s"/>
    <route id="route_we" edges="w_0 0_0_3 0_3_e"/>
    <route id="route_ws" edges="w_0 0_s"/>
    <route id="route_sw" edges="s_0 0_w"/>
    <route id="route_see2" edges="s_0 0_0_2 0_2_e_2 e_2_0_3 0_3_e"/>

    <flow id="flow_e_w" route="route_ew" type="Car" begin="0" end="1200" probability="0.20" departSpeed="8" />
    <flow id="flow_w_e" route="route_we" type="Car" begin="2" end="1300" probability="0.20" departSpeed="8" />
    <flow id="flow_w_s" route="route_ws" type="Car" begin="4" end="1400" probability="0.20" departSpeed="8" />
    <flow id="flow_e_s" route="route_es" type="Car" begin="5" end="1500" probability="0.20" departSpeed="8" />
    <flow id="flow_ew_" route="route_ew" type="Car" begin="60" end="1600" probability="0.20" departSpeed="8" />
    <flow id="flow_we_" route="route_we" type="Car" begin="70" end="1700" probability="0.20" departSpeed="8" />

</routes>
```

(d)

Figure A.7: SUMO route files used for the T-Junction scenario

```
<routes>
    <vType accel="5.0" decel="0.0" emergencyDecel="0.0" id="Car" length="4.0" maxSpeed="16" impatience="0.99" speedFactor="normc
(1,0.5,0.1,1.9)" speedDev="0.5"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="typeSN" length="3"  minGap="0.0" maxSpeed="14" vClass="passenger" guiShape="truck"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="trailerType" length="7" minGap="0.0"  maxSpeed="14" guiShape="bus"/>

    <route id="route_ew" edges="e_0 0_w"/>
    <route id="route_es" edges="e_0 0_s"/>
    <route id="route_we" edges="w_0 0_0_3 0_3_e"/>
    <route id="route_ws" edges="w_0 0_s"/>
    <route id="route_sw" edges="s_0 0_w"/>
    <route id="route_see2" edges="s_0 0_0_2 0_2_e_2 e_2_0_3 0_3_e"/>

    <flow id="flow_e_w" route="route_ew" type="Car" begin="0" end="1200" probability="0.20" departSpeed="8" />
    <flow id="flow_w_e" route="route_we" type="Car" begin="2" end="1300" probability="0.20" departSpeed="8" />

</routes>
```

(e)

```
<routes>
    <vType accel="5.0" decel="0.0" emergencyDecel="0.0" id="Car" length="4.0" maxSpeed="16" impatience="0.99" speedFactor="normc
(1,0.7,0.1,1.9)" speedDev="0.7"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="typeSN" length="3"  minGap="0.0" maxSpeed="14" vClass="passenger" guiShape="truck"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="trailerType" length="7" minGap="0.0"  maxSpeed="14" guiShape="bus"/>

    <route id="route_ew" edges="e_0 0_w"/>
    <route id="route_es" edges="e_0 0_s"/>
    <route id="route_we" edges="w_0 0_0_3 0_3_e"/>
    <route id="route_ws" edges="w_0 0_s"/>
    <route id="route_sw" edges="s_0 0_w"/>
    <route id="route_see2" edges="s_0 0_0_2 0_2_e_2 e_2_0_3 0_3_e"/>

    <flow id="flow_e_w" route="route_ew" type="Car" begin="0" end="1200" probability="0.20" departSpeed="8" />
    <flow id="flow_w_e" route="route_we" type="Car" begin="2" end="1300" probability="0.20" departSpeed="8" />
    <flow id="flow_w_s" route="route_ws" type="Car" begin="4" end="1400" probability="0.20" departSpeed="8" />
    <flow id="flow_e_s" route="route_es" type="Car" begin="5" end="1500" probability="0.20" departSpeed="8" />
    <flow id="flow_ew_" route="route_ew" type="Car" begin="60" end="1600" probability="0.20" departSpeed="8" />
    <flow id="flow_we_" route="route_we" type="Car" begin="70" end="1700" probability="0.20" departSpeed="8" />

</routes>
```

(f)

```
<routes>
    <vType accel="5.0" decel="0.0" emergencyDecel="0.0" id="Car" length="4.0" maxSpeed="16" impatience="0.99" speedFactor="normc
(1,0.7,0.1,1.9)" speedDev="0.7"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="typeSN" length="3"  minGap="0.0" maxSpeed="14" vClass="passenger" guiShape="truck"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="trailerType" length="7" minGap="0.0"  maxSpeed="14" guiShape="bus"/>

    <route id="route_ew" edges="e_0 0_w"/>
    <route id="route_es" edges="e_0 0_s"/>
    <route id="route_we" edges="w_0 0_0_3 0_3_e"/>
    <route id="route_ws" edges="w_0 0_s"/>
    <route id="route_sw" edges="s_0 0_w"/>
    <route id="route_see2" edges="s_0 0_0_2 0_2_e_2 e_2_0_3 0_3_e"/>

    <flow id="flow_e_w" route="route_ew" type="Car" begin="0" end="120" probability="0.01" departSpeed="12" />
    <flow id="flow_w_e" route="route_we" type="Car" begin="2" end="130" probability="0.01" departSpeed="12" />
    <flow id="flow_w_s" route="route_ws" type="Car" begin="4" end="140" probability="0.01" departSpeed="12" />
    <flow id="flow_e_s" route="route_es" type="Car" begin="5" end="150" probability="0.01" departSpeed="12" />
    <flow id="flow_ew_" route="route_ew" type="Car" begin="60" end="160" probability="0.01" departSpeed="12" />
    <flow id="flow_we_" route="route_we" type="Car" begin="70" end="170" probability="0.01" departSpeed="12" />

</routes>
```

(g)

```
<routes>
    <vType accel="5.0" decel="0.0" emergencyDecel="0.0" id="Car" length="4.0" maxSpeed="16" impatience="0.99" speedFactor="normc
(1,0.7,0.1,1.9)" speedDev="0.7"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="typeSN" length="3"  minGap="0.0" maxSpeed="14" vClass="passenger" guiShape="truck"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="trailerType" length="7" minGap="0.0"  maxSpeed="14" guiShape="bus"/>

    <route id="route_ew" edges="e_0 0_w"/>
    <route id="route_es" edges="e_0 0_s"/>
    <route id="route_we" edges="w_0 0_0_3 0_3_e"/>
    <route id="route_ws" edges="w_0 0_s"/>
    <route id="route_sw" edges="s_0 0_w"/>
    <route id="route_see2" edges="s_0 0_0_2 0_2_e_2 e_2_0_3 0_3_e"/>

    <flow id="flow_e_w" route="route_ew" type="Car" begin="0" end="120" probability="0.05" departSpeed="10" />
    <flow id="flow_w_e" route="route_we" type="Car" begin="2" end="130" probability="0.05" departSpeed="10" />
    <flow id="flow_w_s" route="route_ws" type="Car" begin="4" end="140" probability="0.05" departSpeed="10" />
    <flow id="flow_e_s" route="route_es" type="Car" begin="5" end="150" probability="0.05" departSpeed="10" />
    <flow id="flow_ew_" route="route_ew" type="Car" begin="60" end="160" probability="0.05" departSpeed="10" />
    <flow id="flow_we_" route="route_we" type="Car" begin="70" end="170" probability="0.05" departSpeed="10" />

</routes>
```

(h)

Figure A.7: SUMO route files used for the T-Junction scenario (cont.)

```
<routes>
    <vType accel="5.0" decel="0.0" emergencyDecel="0.0" id="Car" length="4.0" maxSpeed="16" impatience="0.99" speedFactor="normc
(1,0.7,0.1,1.9)" speedDev="0.7"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="typeSN" length="3"  minGap="0.0" maxSpeed="14" vClass="passenger" guiShape="truck"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="trailerType" length="7" minGap="0.0"  maxSpeed="14" guiShape="bus"/>

    <route id="route_ew" edges="e_0 0_w"/>
    <route id="route_es" edges="e_0 0_s"/>
    <route id="route_we" edges="w_0 0_0_3 0_3_e"/>
    <route id="route_ws" edges="w_0 0_s"/>
    <route id="route_sw" edges="s_0 0_w"/>
    <route id="route_see2" edges="s_0 0_0_2 0_2_e_2 e_2_0_3 0_3_e"/>

    <flow id="flow_e_w" route="route_ew" type="Car" begin="0" end="120" probability="0.07" departSpeed="9" />
    <flow id="flow_w_e" route="route_we" type="Car" begin="2" end="130" probability="0.07" departSpeed="9" />
    <flow id="flow_w_s" route="route_ws" type="Car" begin="4" end="140" probability="0.07" departSpeed="9" />
    <flow id="flow_e_s" route="route_es" type="Car" begin="5" end="150" probability="0.07" departSpeed="9" />
    <flow id="flow_ew_" route="route_ew" type="Car" begin="60" end="160" probability="0.07" departSpeed="9" />
    <flow id="flow_we_" route="route_we" type="Car" begin="70" end="170" probability="0.07" departSpeed="9" />

</routes>
```

(i)

```
<routes>
    <vType accel="5.0" decel="0.0" emergencyDecel="0.0" id="Car" length="4.0" maxSpeed="16" impatience="0.99" speedFactor="normc
(1,0.7,0.1,1.9)" speedDev="0.7"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="typeSN" length="3"  minGap="0.0" maxSpeed="14" vClass="passenger" guiShape="truck"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="trailerType" length="7" minGap="0.0"  maxSpeed="14" guiShape="bus"/>

    <route id="route_ew" edges="e_0 0_w"/>
    <route id="route_es" edges="e_0 0_s"/>
    <route id="route_we" edges="w_0 0_0_3 0_3_e"/>
    <route id="route_ws" edges="w_0 0_s"/>
    <route id="route_sw" edges="s_0 0_w"/>
    <route id="route_see2" edges="s_0 0_0_2 0_2_e_2 e_2_0_3 0_3_e"/>

    <flow id="flow_e_w" route="route_ew" type="Car" begin="0" end="100" probability="0.09" departSpeed="8" />
    <flow id="flow_w_e" route="route_we" type="Car" begin="2" end="110" probability="0.09" departSpeed="8" />
    <flow id="flow_w_s" route="route_ws" type="Car" begin="4" end="120" probability="0.09" departSpeed="8" />
    <flow id="flow_e_s" route="route_es" type="Car" begin="5" end="130" probability="0.09" departSpeed="8" />
    <flow id="flow_ew_" route="route_ew" type="Car" begin="60" end="140" probability="0.09" departSpeed="8" />
    <flow id="flow_we_" route="route_we" type="Car" begin="70" end="150" probability="0.09" departSpeed="8" />

</routes>
```

(j)

```
<routes>
    <vType accel="5.0" decel="0.0" emergencyDecel="0.0" id="Car" length="4.0" maxSpeed="16" impatience="0.99" speedFactor="normc
(1,0.5,0.1,1.9)" speedDev="0.5"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="typeSN" length="3"  minGap="0.0" maxSpeed="14" vClass="passenger" guiShape="truck"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="trailerType" length="7" minGap="0.0"  maxSpeed="14" guiShape="bus"/>

    <route id="route_ew" edges="e_0 0_w"/>
    <route id="route_es" edges="e_0 0_s"/>
    <route id="route_we" edges="w_0 0_0_3 0_3_e"/>
    <route id="route_ws" edges="w_0 0_s"/>
    <route id="route_sw" edges="s_0 0_w"/>
    <route id="route_see2" edges="s_0 0_0_2 0_2_e_2 e_2_0_3 0_3_e"/>

    <flow id="flow_e_w" route="route_ew" type="Car" begin="0" end="120" probability="0.15" departSpeed="8" />
    <flow id="flow_w_e" route="route_we" type="Car" begin="2" end="130" probability="0.15" departSpeed="8" />

</routes>
```

(k)

```
<routes>
    <vType accel="5.0" decel="0.0" emergencyDecel="0.0" id="Car" length="4.0" maxSpeed="16" impatience="0.99" speedFactor="normc
(1,0.7,0.1,1.9)" speedDev="0.7"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="typeSN" length="3"  minGap="0.0" maxSpeed="14" vClass="passenger" guiShape="truck"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="trailerType" length="7" minGap="0.0"  maxSpeed="14" guiShape="bus"/>

    <route id="route_ew" edges="e_0 0_w"/>
    <route id="route_es" edges="e_0 0_s"/>
    <route id="route_we" edges="w_0 0_0_3 0_3_e"/>
    <route id="route_ws" edges="w_0 0_s"/>
    <route id="route_sw" edges="s_0 0_w"/>
    <route id="route_see2" edges="s_0 0_0_2 0_2_e_2 e_2_0_3 0_3_e"/>

    <flow id="flow_e_w" route="route_ew" type="Car" begin="0" end="120" probability="0.02" departSpeed="12" />
    <flow id="flow_w_e" route="route_we" type="Car" begin="2" end="130" probability="0.02" departSpeed="12" />
    <flow id="flow_w_s" route="route_ws" type="Car" begin="4" end="140" probability="0.02" departSpeed="12" />
    <flow id="flow_e_s" route="route_es" type="Car" begin="5" end="150" probability="0.02" departSpeed="12" />
    <flow id="flow_ew_" route="route_ew" type="Car" begin="6" end="160" probability="0.02" departSpeed="12" />
    <flow id="flow_we_" route="route_we" type="Car" begin="7" end="170" probability="0.02" departSpeed="12" />

</routes>
```

(l)

Figure A.7: SUMO route files used for the T-Junction scenario (cont.)

```
<routes>
    <vType accel="5.0" decel="0.0" emergencyDecel="0.0" id="Car" length="4.0" maxSpeed="16" impatience="0.99" speedFactor="normc
(1,0.7,0.1,1.9)" speedDev="0.7"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="typeSN" length="3"  minGap="0.0" maxSpeed="14" vClass="passenger" guiShape="truck"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="trailerType" length="7" minGap="0.0"  maxSpeed="14" guiShape="bus"/>

    <route id="route_ew" edges="e_0 0_w"/>
    <route id="route_es" edges="e_0 0_s"/>
    <route id="route_we" edges="w_0 0_0_3 0_3_e"/>
    <route id="route_ws" edges="w_0 0_s"/>
    <route id="route_sw" edges="s_0 0_w"/>
    <route id="route_see2" edges="s_0 0_0_2 0_2_e_2 e_2_0_3 0_3_e"/>

    <flow id="flow_e_w" route="route_ew" type="Car" begin="0" end="150" probability="0.12" departSpeed="10" />
    <flow id="flow_w_e" route="route_we" type="Car" begin="2" end="160" probability="0.12" departSpeed="10" />
    <flow id="flow_w_s" route="route_ws" type="Car" begin="30" end="170" probability="0.05" departSpeed="10" />
    <flow id="flow_e_s" route="route_es" type="Car" begin="50" end="180" probability="0.05" departSpeed="10" />

</routes>
```

(m)

```
<routes>
    <vType accel="5.0" decel="0.0" emergencyDecel="0.0" id="Car" length="4.0" maxSpeed="16" impatience="0.99" speedFactor="normc
(1,0.7,0.1,1.9)" speedDev="0.7"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="typeSN" length="3"  minGap="0.0" maxSpeed="14" vClass="passenger" guiShape="truck"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="trailerType" length="7" minGap="0.0"  maxSpeed="14" guiShape="bus"/>

    <route id="route_ew" edges="e_0 0_w"/>
    <route id="route_es" edges="e_0 0_s"/>
    <route id="route_we" edges="w_0 0_0_3 0_3_e"/>
    <route id="route_ws" edges="w_0 0_s"/>
    <route id="route_sw" edges="s_0 0_w"/>
    <route id="route_see2" edges="s_0 0_0_2 0_2_e_2 e_2_0_3 0_3_e"/>

    <flow id="flow_e_w" route="route_ew" type="Car" begin="0" end="160" probability="0.18" departSpeed="9" />
    <flow id="flow_w_e" route="route_we" type="Car" begin="60" end="170" probability="0.18" departSpeed="9" />

</routes>
```

(n)

```
<routes>
    <vType accel="5.0" decel="0.0" emergencyDecel="0.0" id="Car" length="4.0" maxSpeed="16" impatience="0.99" speedFactor="normc
(1,0.7,0.1,1.9)" speedDev="0.7"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="typeSN" length="3"  minGap="0.0" maxSpeed="14" vClass="passenger" guiShape="truck"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="trailerType" length="7" minGap="0.0"  maxSpeed="14" guiShape="bus"/>

    <route id="route_ew" edges="e_0 0_w"/>
    <route id="route_es" edges="e_0 0_s"/>
    <route id="route_we" edges="w_0 0_0_3 0_3_e"/>
    <route id="route_ws" edges="w_0 0_s"/>
    <route id="route_sw" edges="s_0 0_w"/>
    <route id="route_see2" edges="s_0 0_0_2 0_2_e_2 e_2_0_3 0_3_e"/>

    <flow id="flow_e_w" route="route_ew" type="Car" begin="0" end="100" probability="0.17" departSpeed="8" />
    <flow id="flow_w_e" route="route_we" type="Car" begin="2" end="100" probability="0.17" departSpeed="8" />
    <flow id="flow_w_s" route="route_ws" type="Car" begin="4" end="100" probability="0.06" departSpeed="8" />
    <flow id="flow_e_s" route="route_es" type="Car" begin="5" end="100" probability="0.06" departSpeed="8" />

</routes>
```

(o)

```
<routes>
    <vType accel="5.0" decel="0.0" emergencyDecel="0.0" id="Car" length="4.0" maxSpeed="16" impatience="0.99" speedFactor="normc
(1,0.5,0.1,1.9)" speedDev="0.5"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="typeSN" length="3"  minGap="0.0" maxSpeed="14" vClass="passenger" guiShape="truck"/>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="trailerType" length="7" minGap="0.0"  maxSpeed="14" guiShape="bus"/>

    <route id="route_ew" edges="e_0 0_w"/>
    <route id="route_es" edges="e_0 0_s"/>
    <route id="route_we" edges="w_0 0_0_3 0_3_e"/>
    <route id="route_ws" edges="w_0 0_s"/>
    <route id="route_sw" edges="s_0 0_w"/>
    <route id="route_see2" edges="s_0 0_0_2 0_2_e_2 e_2_0_3 0_3_e"/>

    <flow id="flow_w_s" route="route_ws" type="Car" begin="3" end="170" probability="0.20" departSpeed="8" />
    <flow id="flow_e_s" route="route_es" type="Car" begin="4" end="160" probability="0.20" departSpeed="8" />

</routes>
```

(p)

Figure A.7: SUMO route files used for the T-Junction scenario (cont.)

```
<routes>
    <vType accel="3.0" decel="0.0" emergencyDecel="0.0"  id="Car" length="4.0" maxSpeed="8.0" tau="0.0" impatience="1" sigma="0.0" mingap="0.0"
speedFactor="normc(1,0.7,0.9,1.1)" speedDev="0.1" />
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="typeSN" length="3"  minGap="0.0" maxSpeed="8.0" vClass="passenger" guiShape="truck"/
>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="trailerType" length="7" minGap="0.0"  maxSpeed="8.0" guiShape="bus"/>
    <route id="route_ne" edges="n_n1 n1_nw1 nw1_nw2 nw2_w1 w1_ws1 ws1_ws2 ws2_s1 s1_se1 se1_se2 se2_e1 e1_e"/>
    <route id="route_ns" edges="n_n1 n1_nw1 nw1_nw2 nw2_w1 w1_ws1 ws1_ws2 ws2_s1 s1_s"/>
    <route id="route_we" edges="w_w1 w1_ws1 ws1_ws2 ws2_s1 s1_se1 se1_se2 se2_e1 e1_e"/>
    <route id="route_ws" edges="w_w1 w1_ws1 ws1_ws2 ws2_s1 s1_s"/>
    <route id="route_se" edges="s_s1 s1_se1 se1_se2 se2_e1 e1_e"/>

    <flow depart="1" departPos="50" id="flow_n_e" route="route_ne" type="Car" begin="0" end="130" probability="0.25" departSpeed="8.0"/>
    <flow depart="1" departPos="50" id="flow_n_s" route="route_ns" type="Car" begin="1" end="130" probability="0.03" departSpeed="8.0"/>
    <flow depart="1"  departPos="50" id="flow_w_e" route="route_we" type="Car" begin="2" end="130" probability="0.2" departSpeed="8.0"/>
    <flow depart="1"  departPos="50" id="flow_w_s" route="route_ws" type="Car" begin="3" end="130" probability="0.03" departSpeed="8.0"/>

</routes>
```

(q)

```
<routes>
    <vType accel="3.0" decel="0.0" emergencyDecel="0.0"  id="Car" length="4.0" maxSpeed="8.0" tau="0.0" impatience="1" sigma="0.0" mingap="0.0"
speedFactor="normc(1,0.7,0.9,1.1)" speedDev="0.1" />
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="typeSN" length="3"  minGap="0.0" maxSpeed="8.0" vClass="passenger" guiShape="truck"/
>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="trailerType" length="7" minGap="0.0"  maxSpeed="8.0" guiShape="bus"/>
    <route id="route_ne" edges="n_n1 n1_nw1 nw1_nw2 nw2_w1 w1_ws1 ws1_ws2 ws2_s1 s1_se1 se1_se2 se2_e1 e1_e"/>
    <route id="route_ns" edges="n_n1 n1_nw1 nw1_nw2 nw2_w1 w1_ws1 ws1_ws2 ws2_s1 s1_s"/>
    <route id="route_we" edges="w_w1 w1_ws1 ws1_ws2 ws2_s1 s1_se1 se1_se2 se2_e1 e1_e"/>
    <route id="route_ws" edges="w_w1 w1_ws1 ws1_ws2 ws2_s1 s1_s"/>
    <route id="route_se" edges="s_s1 s1_se1 se1_se2 se2_e1 e1_e"/>

    <flow depart="1" departPos="50" id="flow_n_e" route="route_ne" type="Car" begin="0" end="150" probability="0.13" departSpeed="8.0"/>
    <flow depart="1"  departPos="50" id="flow_w_e" route="route_we" type="Car" begin="1" end="150" probability="0.16" departSpeed="8.0"/>

</routes>
```

(r)

```
<routes>
    <vType accel="3.0" decel="0.0" emergencyDecel="0.0"  id="Car" length="4.0" maxSpeed="8.0" tau="0.0" impatience="1" sigma="0.0" mingap="0.0"
speedFactor="normc(1,0.7,0.9,1.1)" speedDev="0.1" />
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="typeSN" length="3"  minGap="0.0" maxSpeed="8.0" vClass="passenger" guiShape="truck"/
>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="trailerType" length="7" minGap="0.0"  maxSpeed="8.0" guiShape="bus"/>
    <route id="route_ne" edges="n_n1 n1_nw1 nw1_nw2 nw2_w1 w1_ws1 ws1_ws2 ws2_s1 s1_se1 se1_se2 se2_e1 e1_e"/>
    <route id="route_ns" edges="n_n1 n1_nw1 nw1_nw2 nw2_w1 w1_ws1 ws1_ws2 ws2_s1 s1_s"/>
    <route id="route_we" edges="w_w1 w1_ws1 ws1_ws2 ws2_s1 s1_se1 se1_se2 se2_e1 e1_e"/>
    <route id="route_ws" edges="w_w1 w1_ws1 ws1_ws2 ws2_s1 s1_s"/>
    <route id="route_se" edges="s_s1 s1_se1 se1_se2 se2_e1 e1_e"/>

    <flow depart="1" departPos="10" id="flow_n_e" route="route_ne" type="Car" begin="0" end="140" probability="0.15" departSpeed="8.0"/>
    <flow depart="1" departPos="10" id="flow_n_s" route="route_ns" type="Car" begin="2" end="140" probability="0.05" departSpeed="8.0"/>
    <flow depart="1"  departPos="10" id="flow_w_e" route="route_we" type="Car" begin="4" end="140" probability="0.14" departSpeed="8.0"/>
    <flow depart="1"  departPos="10" id="flow_w_s" route="route_ws" type="Car" begin="6" end="140" probability="0.06" departSpeed="8.0"/>

</routes>
```

(s)

```
<routes>
    <vType accel="3.0" decel="0.0" emergencyDecel="0.0"  id="Car" length="4.0" maxSpeed="8.0" tau="0.0" impatience="1" sigma="0.0" mingap="0.0"
speedFactor="normc(1,0.7,0.9,1.1)" speedDev="0.1" />
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="typeSN" length="3"  minGap="0.0" maxSpeed="8.0" vClass="passenger" guiShape="truck"/
>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="trailerType" length="7" minGap="0.0"  maxSpeed="8.0" guiShape="bus"/>
    <route id="route_ne" edges="n_n1 n1_nw1 nw1_nw2 nw2_w1 w1_ws1 ws1_ws2 ws2_s1 s1_se1 se1_se2 se2_e1 e1_e"/>
    <route id="route_ns" edges="n_n1 n1_nw1 nw1_nw2 nw2_w1 w1_ws1 ws1_ws2 ws2_s1 s1_s"/>
    <route id="route_we" edges="w_w1 w1_ws1 ws1_ws2 ws2_s1 s1_se1 se1_se2 se2_e1 e1_e"/>
    <route id="route_ws" edges="w_w1 w1_ws1 ws1_ws2 ws2_s1 s1_s"/>
    <route id="route_se" edges="s_s1 s1_se1 se1_se2 se2_e1 e1_e"/>

    <flow depart="1" departPos="10" id="flow_n_e" route="route_ne" type="Car" begin="0" end="140" probability="0.15" departSpeed="8.0"/>
    <flow depart="1" departPos="10" id="flow_n_s" route="route_ns" type="Car" begin="2" end="140" probability="0.05" departSpeed="8.0"/>
    <flow depart="1"  departPos="10" id="flow_w_e" route="route_we" type="Car" begin="4" end="140" probability="0.14" departSpeed="8.0"/>
    <flow depart="1"  departPos="10" id="flow_w_s" route="route_ws" type="Car" begin="6" end="140" probability="0.06" departSpeed="8.0"/>

</routes>
```

(t)

Figure A.8: SUMO route files used for the roundabout scenario

```
<routes>
    <vType accel="3.0" decel="0.0" emergencyDecel="0.0"  id="Car" length="4.0" maxSpeed="8.0" tau="0.0" impatience="1" sigma="0.0" mingap="0.0"
speedFactor="normc(1,0.7,0.9,1.1)" speedDev="0.1" />
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="typeSN" length="3"  minGap="0.0" maxSpeed="8.0" vClass="passenger" guiShape="truck"/
>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="trailerType" length="7" minGap="0.0"  maxSpeed="8.0" guiShape="bus"/>
    <route id="route_ne" edges="n_n1 n1_nw1 nw1_nw2 nw2_w1 w1_ws1 ws1_ws2 ws2_s1 s1_se1 se1_se2 se2_e1 e1_e"/>
    <route id="route_ns" edges="n_n1 n1_nw1 nw1_nw2 nw2_w1 w1_ws1 ws1_ws2 ws2_s1 s1_s"/>
    <route id="route_we" edges="w_w1 w1_ws1 ws1_ws2 ws2_s1 s1_se1 se1_se2 se2_e1 e1_e"/>
    <route id="route_ws" edges="w_w1 w1_ws1 ws1_ws2 ws2_s1 s1_s"/>
    <route id="route_se" edges="s_s1 s1_se1 se1_se2 se2_e1 e1_e"/>

    <flow depart="1" departPos="50" id="flow_n_e" route="route_ne" type="Car" begin="0" end="125" probability="0.13" departSpeed="8.0"/>
    <flow depart="1" departPos="50" id="flow_n_s" route="route_ns" type="Car" begin="1" end="125" probability="0.07" departSpeed="8.0"/>
    <flow depart="1"  departPos="50" id="flow_w_e" route="route_we" type="Car" begin="2" end="125" probability="0.15" departSpeed="8.0"/>
    <flow depart="1"  departPos="50" id="flow_w_s" route="route_ws" type="Car" begin="3" end="125" probability="0.08" departSpeed="8.0"/>

  </routes>
```

(u)

```
<routes>
    <vType accel="3.0" decel="0.0" emergencyDecel="0.0"  id="Car" length="4.0" maxSpeed="8.0" tau="0.0" impatience="1" sigma="0.0" mingap="0.0"
speedFactor="normc(1,0.7,0.9,1.1)" speedDev="0.1" />
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="typeSN" length="3"  minGap="0.0" maxSpeed="8.0" vClass="passenger" guiShape="truck"/
>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="trailerType" length="7" minGap="0.0"  maxSpeed="8.0" guiShape="bus"/>
    <route id="route_ne" edges="n_n1 n1_nw1 nw1_nw2 nw2_w1 w1_ws1 ws1_ws2 ws2_s1 s1_se1 se1_se2 se2_e1 e1_e"/>
    <route id="route_ns" edges="n_n1 n1_nw1 nw1_nw2 nw2_w1 w1_ws1 ws1_ws2 ws2_s1 s1_s"/>
    <route id="route_we" edges="w_w1 w1_ws1 ws1_ws2 ws2_s1 s1_se1 se1_se2 se2_e1 e1_e"/>
    <route id="route_ws" edges="w_w1 w1_ws1 ws1_ws2 ws2_s1 s1_s"/>
    <route id="route_se" edges="s_s1 s1_se1 se1_se2 se2_e1 e1_e"/>

    <flow depart="1" departPos="50" id="flow_n_e" route="route_ne" type="Car" begin="0" end="135" probability="0.17" departSpeed="8.0"/>
    <flow depart="1"  departPos="50" id="flow_w_e" route="route_we" type="Car" begin="1" end="135" probability="0.21" departSpeed="8.0"/>

  </routes>
```

(v)

```
<routes>
    <vType accel="3.0" decel="0.0" emergencyDecel="0.0"  id="Car" length="4.0" maxSpeed="8.0" tau="0.0" impatience="1" sigma="0.0" mingap="0.0"
speedFactor="normc(1,0.7,0.9,1.1)" speedDev="0.1" />
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="typeSN" length="3"  minGap="0.0" maxSpeed="8.0" vClass="passenger" guiShape="truck"/
>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="trailerType" length="7" minGap="0.0"  maxSpeed="8.0" guiShape="bus"/>
    <route id="route_ne" edges="n_n1 n1_nw1 nw1_nw2 nw2_w1 w1_ws1 ws1_ws2 ws2_s1 s1_se1 se1_se2 se2_e1 e1_e"/>
    <route id="route_ns" edges="n_n1 n1_nw1 nw1_nw2 nw2_w1 w1_ws1 ws1_ws2 ws2_s1 s1_s"/>
    <route id="route_we" edges="w_w1 w1_ws1 ws1_ws2 ws2_s1 s1_se1 se1_se2 se2_e1 e1_e"/>
    <route id="route_ws" edges="w_w1 w1_ws1 ws1_ws2 ws2_s1 s1_s"/>
    <route id="route_se" edges="s_s1 s1_se1 se1_se2 se2_e1 e1_e"/>

    <flow depart="1" departPos="10" id="flow_n_s" route="route_ns" type="Car" begin="2" end="140" probability="0.11" departSpeed="8.0"/>
    <flow depart="1"  departPos="10" id="flow_w_s" route="route_ws" type="Car" begin="6" end="140" probability="0.09" departSpeed="8.0"/>

  </routes>
```

(w)

```
<routes>
    <vType accel="3.0" decel="0.0" emergencyDecel="0.0"  id="Car" length="4.0" maxSpeed="8.0" tau="0.0" impatience="1" sigma="0.0" mingap="0.0"
speedFactor="normc(1,0.7,0.9,1.1)" speedDev="0.1" />
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="typeSN" length="3"  minGap="0.0" maxSpeed="8.0" vClass="passenger" guiShape="truck"/
>
    <vType accel="0.0" decel="0.0" emergencyDecel="0.0" id="trailerType" length="7" minGap="0.0"  maxSpeed="8.0" guiShape="bus"/>
    <route id="route_ne" edges="n_n1 n1_nw1 nw1_nw2 nw2_w1 w1_ws1 ws1_ws2 ws2_s1 s1_se1 se1_se2 se2_e1 e1_e"/>
    <route id="route_ns" edges="n_n1 n1_nw1 nw1_nw2 nw2_w1 w1_ws1 ws1_ws2 ws2_s1 s1_s"/>
    <route id="route_we" edges="w_w1 w1_ws1 ws1_ws2 ws2_s1 s1_se1 se1_se2 se2_e1 e1_e"/>
    <route id="route_ws" edges="w_w1 w1_ws1 ws1_ws2 ws2_s1 s1_s"/>
    <route id="route_se" edges="s_s1 s1_se1 se1_se2 se2_e1 e1_e"/>

    <flow depart="1" departPos="10" id="flow_n_e" route="route_ne" type="Car" begin="0" end="130" probability="0.09" departSpeed="8.0"/>
    <flow depart="1" departPos="10" id="flow_n_s" route="route_ns" type="Car" begin="2" end="130" probability="0.11" departSpeed="8.0"/>
    <flow depart="1"  departPos="10" id="flow_w_e" route="route_we" type="Car" begin="4" end="130" probability="0.08" departSpeed="8.0"/>
    <flow depart="1"  departPos="10" id="flow_w_s" route="route_ws" type="Car" begin="6" end="130" probability="0.10" departSpeed="8.0"/>

  </routes>
```

(x)

Figure A.8: SUMO route files used for the roundabout scenario (cont.)