A Reinforcement Learning Intelligent Agent for Electric Bicycle Navigation

Konstantinos Roumeliotis

DIPLOMA THESIS

Department of Computer Science and Engineering

School of Engineering

University of Ioannina

March 2023

TABLE OF CONTENTS

ABSTRACT

Konstantinos Roumeliotis, Diploma, Department of Computer Science and Engineering, School of Engineering, University of Ioannina, Greece, March 2023.

A Reinforcement Learning Intelligent Agent for Electric Bicycle navigation.

Advisor: Konstantinos Blekas, Professor.

In this work we present a reinforcement learning intelligent agent for autonomous moped driving that acquires knowledge from multiple mopeds loaded in the scenario and performing different routes. The designed reward function involved in training the artificial neural network aims to complete the vehicle trips, quickly and safely without collisions. The reinforcement learning algorithm that used to train the intelligent agent is Greedy Epsilon DQN, which enables the system to consider many random actions and learn from them. To evaluate the efficiency of the scheme, several experiments were implemented on different maps in the S.U.MO environment. The challenge we face is to achieve according to a no traditional method to train efficiently the intelligent agent.

Στην παρούσα εργασία παρουσιάζουμε έναν ευφυή πράκτορα ενισχυτικής μάθησης για αυτόνομη οδήγηση μοτοποδηλάτων, που αποκτά γνώσεις από πολλαπλά μοτοποδήλατα που συμμετέχουν στο σενάριο και εκτελούν διαφορετικές διαδρομές. Η σχεδιασμένη συνάρτηση ανταμοιβής που εμπλέκεται στην εκπαίδευση του τεχνητού νευρωνικού δικτύου, στοχεύει στην γρήγορη και ασφαλή ολοκλήρωση των διαδρομών του οχήματος χωρίς συγκρούσεις. Ο αλγόριθμος ενισχυτικής μάθησης που χρησιμοποιείται για την εκπαίδευση του ευφυούς πράκτορα είναι ο Greedy Epsilon DQN, ο οποίος επιτρέπει στο σύστημα να εξετάσει πολλές τυχαίες ενέργειες και να μάθει από αυτές. Για την αξιολόγηση της αποτελεσματικότητας του συστήματος, εφαρμόστηκαν διάφορα πειράματα σε διαφορετικούς χάρτες, στο περιβάλλον S.U.MO. Η πρόκληση που αντιμετωπίζουμε είναι να εκπαιδεύσουμε αποδοτικά τον πράκτορα.

LIST OF FIGURES

CHAPTER 1

Introduction

Artificial Intelligence is divided into two levels, the sub-symbolic and the symbolic. The first studies the processing of signals and numerical information that a system receives from sensors and converts them into symbols. Machine learning falls into this broader field and its purpose is for a system to learn from the data it receives as input without being explicitly programmed.

1.1 Machine Learning

Learning is the process of improving the performance of a system in a specific task after observing many examples. For learning to occur, 3 basic elements are required:

- An environment that provides data in the form of examples to the system.
- A criterion for assessing the performance of the system.
- A specific task that the system is asked to perform.

Machine learning considers the development of learning algorithms, i.e. algorithms that improve the performance of a system. To achieve the improvement, the algorithm is given several of examples to enable it to be trained. The improvement is usually incremental because the algorithm in most cases is iterative, i.e. it examines examples at multiple learning epochs.[1]

1.2 Types of Machine Learning

There are 4 basic types of machine learning algorithms: supervised, semi-supervised, unsupervised and reinforcement.

Supervised Learning: supervised learning algorithms are learning algorithms that learn to associate some input with some output, given a training set of examples of inputs x and outputs y. In many cases the outputs y may be difficult to collect automatically and must be provided by a human “supervisor,” but the term still applies even when the training set targets were collected automatically.

Semi-Supervised Learning: is a branch of machine learning that combines a small amount of labeled data with a large amount of unlabeled data during training. Semi-supervised learning falls between unsupervised (with no labeled training data) and supervised (with only labeled training data). Semi-supervised learning aims to alleviate the issue of having limited amounts of labeled data available for training. Semi-supervised learning is motivated by problem settings where unlabeled data is abundant and obtaining labeled data is expensive.

Unsupervised Learning: unsupervised algorithms are those that experience only “features” but not a supervision signal. The distinction between supervised and unsupervised algorithms is not formally and rigidly defined because there is no objective test for distinguishing whether a value is a feature, or a target provided by a supervisor. Informally, unsupervised learning refers to most attempts to extract information from a distribution that do not require human labor to annotate examples. The term is usually associated with density estimation, learning to draw samples from a distribution, learning to denoise data from some distribution, finding a manifold that the data lies near, or clustering the data into groups of related examples.

Reinforcement Learning: reinforcement learning algorithms interact with an environment, so there is a feedback loop between the learning system and its experiences. In the context of reinforcement learning, an autonomous agent must learn to perform a task by trial and error, without the supervision from the human operator. The Agent takes actions in an environment, to maximize the notion of cumulative reward. [2]

1.3 Applications of Machine Learning

In machine learning there is a large number of applications. Image recognition is one of the most common applications of machine learning. It is used to identify and detect objects, persons, vehicles, etc. One of the most popular use cases of computer vision is the image recognition and face detection. For example, facebook provides us a feature of auto friend tagging suggestion. Whenever we upload a photo with our Facebook friends, then an algorithm suggests us to tag the profile of the person who is with us in the photo. The technology behind this, is machine learning's face detection and recognition algorithm. While using Alexa, Cortana or Siri we can talk, and she can answer us like a real person. This occurs under speech recognition, and it's a popular application of machine learning. Speech recognition is a process of converting voice instructions into tensors, and it is known as “Computer speech recognition”. Machine learning is making our online transaction safe and secure by detecting fraud transaction. Whenever we perform some online transaction, there may be various ways that a fraudulent transaction can take place such as fake accounts, fake ids, and steal money in the middle of a transaction. So, to detect this, Feed Forward Neural network helps us by checking whether it is a genuine transaction or a fraud transaction. The technology behind this is machine learning. The other way that can detect a system a fraud transaction is by using data mining algorithms. The field of data mining is close to the scientific field of machine learning. One of the last, most exciting applications of machine learning is self-driving cars and autonomous robots. The algorithms that are used for the training of self-driving cars or for the navigation of an autonomous robot are Machine learning algorithms. These methods are highly efficient and that is why more and more people in recent years are choosing vehicles from companies involved in the production of this type of car. Popular companies which are engaged in the manufacturing of autonomous cars are Tesla, Waymo and General Motors - Cruise. Tesla, the most popular car manufacturing company is working on self-driving car. It is using reinforcement learning method to train the car models and other super/unsupervised methods to detect people and objects while driving.

1.4 Artificial Neural Networks

Artificial neural networks were inspired by biological neural networks. A component of a biological neural network is the neuron, which consists of the dendrites, the cell body, the neuroaxis and the synapses. In fact, an artificial neuron simulates the functions of a biological neuron. The first and most important attempt to build an artificial neuron and thus an artificial neural network was made in 1957 by Frank Rosenblatt at the Cornell Aeronautical Laboratory with the creation of the Perceptron. This model is the simplest neural network that can be designed and is a network consisting of a single neuron. Perceptron is a binary classifier, i.e. a function that maps the input x to an output value y. Input x is a vector with real values and y is a single binary value. W is a scalar number that represents the weights of the neuron. Also, b is a special term called bias and aims to improve the model's performance. The perceptron model implements the transfer function $\sum wi \times xi + b$. At the output of the neuron the step activation function is applied. Because the function is linear the perceptron has less performance in solving most machine learning problems. For this reason, artificial neural networks with multiple neurons and layers of neurons are practically used, which are nonlinear classifiers as opposed to the perceptron which is a linear binary classifier.

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{else} \end{cases}$$

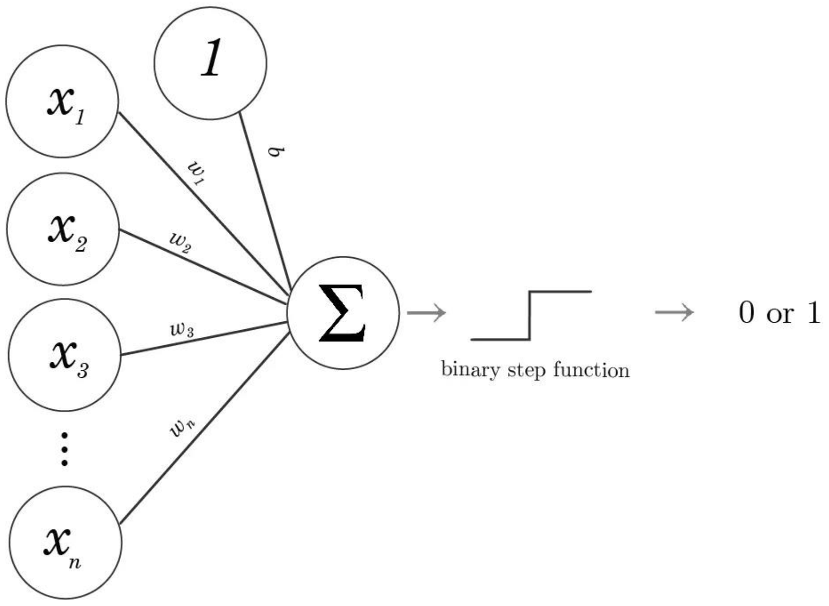


Figure 1.1 Perceptron Model

The most common type of Artificial Neural Network is the Multilayer Perceptron. A Multilayer Perceptron has input and output layers, and one or more hidden layers with many neurons stacked together. Multilayer Perceptron can use any arbitrary activation function. Multilayer Perceptron falls under the category of feedforward algorithms, and that, because inputs are multiplied with the initial weights in a weighted sum and driven to the activation function, just like in the Perceptron. Each linear combination is propagated to the next layer in a MLP model, instead of the perceptron that consist by one neuron. Each layer is feeding the next one with the result of their computation, their internal representation of the data. This goes all the way through the hidden layers to the output layer. The goal of a MLP feedforward network is to approximate some function f*. For example, for a classifier, y = f(x) maps an input x to a category y. A feedforward network defines a mapping y = f*(x; θ) and learns the value of the parameters θ that result in the best function approximation. These models are called feedforward because information flows through the function being evaluated from x, through the intermediate computations used to define f, and finally to the output y. There are no feedback connections in which outputs of the model are fed back into itself. When feedforward neural networks are extended to include feedback connections, they are called recurrent neural networks. Different activation functions are used in artificial neural networks. The most significant of these are Sigmoid, Rectified Linear Unit (ReLu) and Hyperbolic Tangent.

Activation functions:

- Sigmoid:

$$f(x) = \frac{1}{1+e^{-x}}$$

- ReLU:

$f(x) = \max(0,z)$

- Hyperbolic Tangent:

$$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

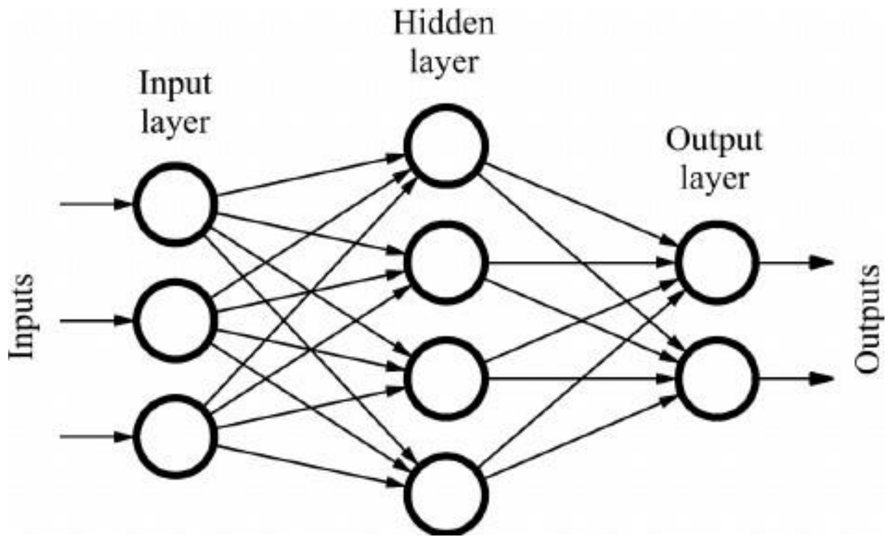


Figure 1.2 Multilayer Perceptron Model

1.5 Back Propagation

Forward pass function in artificial neural networks flows forward the data x in tensor term and leads to an output y. The scalar input x provides the initial information to our network and propagates up to the hidden units at each layer and finally produces y. This is called forward propagation. The back-propagation algorithm (Rumelhart et al., 1986a), often simply called backprop, allows the information from the cost to then flow backwards through the network, in order to compute the gradient. Gradient is a very important term in Stochastic Optimization field. Computing an analytical expression for the gradient can be quite computational expensive. The back - propagation to achieve low computational cost, using a simple and inexpensive method. The term back-propagation is often misunderstood as meaning the whole learning algorithm for multi-layer neural networks. Actually, back-propagation computes only the gradient, while another algorithm, such as stochastic gradient descent or Adam, is used to perform learning using the gradient. Furthermore, back-propagation has to deal with the computation of derivatives for a lot of cases of optimization problems. Specifically, Back Propagation computes the gradient $\nabla xf(x, y)$ for an arbitrary function f , where x is a set of variables whose derivatives are desired, and y is an additional set of variables that are inputs to the function but whose derivatives are not required. In learning algorithms, the gradient we most often require is the gradient of the cost function with respect to the parameters, $\nabla\theta J(\theta)$. Many machine learning tasks involve computing other derivatives, either as part of the learning process, or to analyze the learned model. The back - propagation algorithm can be applied to these tasks as well and is not restricted to computing the gradient of the cost function with respect to the parameters. A very popular function that gradient descent is used to search the best performance is the Least Square function which is used in several machine learning problems for estimation. In conclusion gradient descent is a very significant method which reduce the error of the neurons iteratively according to the Learning Rate factor and leads in better performance a machine learning system. [2]

CHAPTER 2

Reinforcement Learning

2.1 Introduction

Reinforcement learning is a computational approach to understanding and automating goal-directed learning and decision-making. It is an area of machine learning where a system learns through interaction with its environment and is the result of a collaboration between the fields of automatic control and comparative psychology. One of the most important equations used in automatic control problems is the Bellman equation, which is equally important for solving reinforcement learning problems. Of fundamental importance for any reinforcement learning problem are Markov decision processes because all problems can be formalized by this method and each environment can be described independently of its characteristics. In order to solve a reinforcement learning problem, we first need to identify the components of the problem. In this category of machine learning the most important elements are the intelligent agent, the environment it interacts, the states, the actions, and the numerical reward signal. An agent to achieve the solution of such a problem must record the states after each action and evaluate them according to the designed reward function designed for the problem at hand. His aim is to maximize the reward, thus achieving the best possible result. Moreover, the learner is not told which actions to take, as in many forms of machine learning, but instead must discover which actions yield the most reward by trying them out. One of the challenges that arise in reinforcement learning, and not in other kinds of learning, is the trade-off between exploration and exploitation. Exploration finds more information about the environment and Exploitation exploits known information to maximize reward. Each intelligent agent based on its specific characteristics can be divided into 3 categories. It can be a Value Based, Policy Based or Actor Critic agent, which is a category that combines the characteristics of the other two. Reinforcement learning algorithms according to their characteristics are divided into two categories. The model free algorithms and the model-based algorithms.

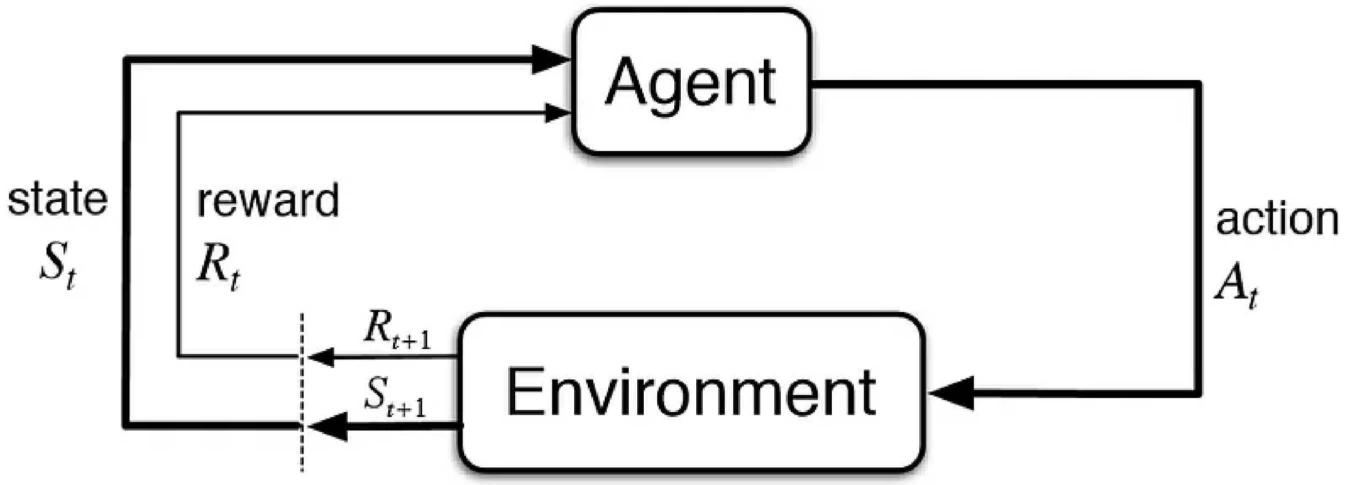


Figure 2.1 Reinforcement Learning Procedure

2.2 Exploration vs Exploitation

One of the toughest challenges, as described at the beginning of the chapter, is the trade-off between exploration and exploitation. To achieve large rewards, a reinforcement learning agent must prefer actions that have been previously tested and found to be effective in generating rewards. However, to maximize rewards it is necessary and very important to also explore new situations, choosing actions that have not been used before. The dilemma is that neither exploration nor exploitation can be pursued exclusively without the project failing. Exploitation refers to taking action from the flow, the best version of the policy we have learned - actions that we know will achieve high reward. An example from real life is choosing the best restaurant we've ever been to. Exploration refers to taking actions specifically to obtain more educational data. Exploration can be implemented in many ways, ranging from occasionally taking random actions aimed at covering the entire space of possible actions, to model-based approaches that compute an action choice based on the expected reward and the magnitude of the model's uncertainty about that reward. An example of exploration in the real world is to choose to go to a restaurant we have never visited before. Since this problem has been studied by mathematicians for many decades, some strategies have been developed to solve it as best as possible. One of these strategies is the e-greedy algorithm, an algorithm which is used in this work. Another Bayesian technique which is used for the trade-off between exploration and exploitation is the Upper Confidence Bound (UCB). Finally, an equally important method which was developed is Thompson Sampling. In conclusion, the aforementioned strategies are applied to reinforcement learning problems for better exploration and not to other forms of machine learning such as supervised learning or unsupervised learning, as in these types of learning there is no dilemma. [3]

2.3 Markov Decision Process (MDP)

In mathematics, a Markov decision process (MDP) is a discrete-time control process. It provides a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. MDPs are useful for studying optimization problems solved via dynamic programming. Markov decision processes formally describe an environment for a reinforcement learning problem.

A Markov Decision Process (MDP) is a tuple of 5 elements $(S, A, P_\alpha, R_\alpha, \gamma)$

- S is a finite set of states called the state space. A state S_t is Markov if and only if $P [S_t+1 | S_t] = P [S_t+1 | S1, ..., S_t]$ (Markov Property)
- A is a finite set of actions called the action space.
- $P_\alpha(s,s') = P_r(s_{t+1} = s'|s_t = s, a_t = a)$ is a state transition probability matrix. Describes the probability that action α in state S_t will lead to state S'_t , due to action a'.
- R_α is a reward function, $R_s = E [R_t+1 | S_t = s, A_t = a]$. Is the immediate reward received after transitioning from state S_t to state S'_t , due to action.
- γ is a discount factor $\gamma \in [0, 1]$. With this discount factor avoids infinite returns in cyclic Markov processes. [4]

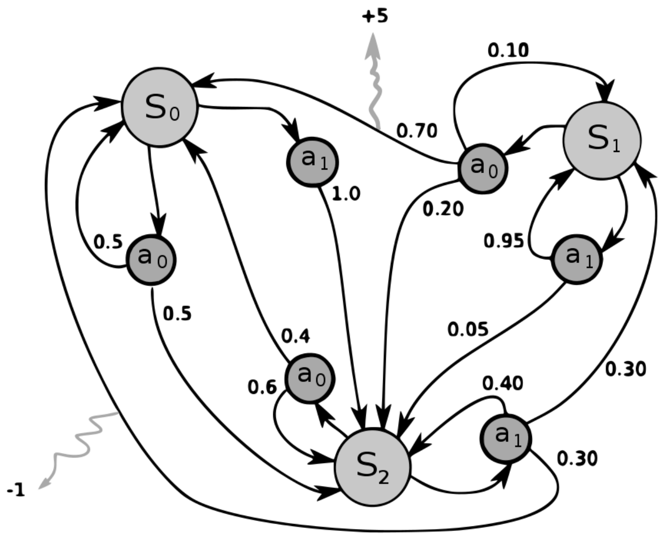


Figure 2.2 Markov Decision Process (MDP)

2.4 Policy

The mapping from context to action is called a policy. A policy π is a distribution over actions given states and fully defines the behavior of the intelligent agent. Also, a policy is a mapping from perceived states of the environment to actions to be taken when in those states. It corresponds to what in psychology would be called a set of stimulus–response rules or associations (provided that stimuli include those that can come from within the animal). In some cases, the policy may be a simple function, while in others the policy may be a simple function. in others it may involve extensive calculations, such as a search procedure. MDP policies depend only on the current state and not on history. At this point it is important to stress that policies are stationary, i.e. completely independent of time. Our goal in a reinforcement learning problem is to find the optimal policy to maximize the reward. Policy is the core of reinforcement learning. [5]

$\pi(a|s) = P\ [At = a \mid St = s]$

2.5 Return

The return G_t is the total discounted reward from time-step t and is defined as some specific function of the reward sequence. The agent tries to select actions so that the sum of the discounted rewards it receives over the future is maximized. The discount factor γ is a real number and is the present value of future rewards. If γ is close to 0 then leads to myopic evaluation. If is close to 1 then leads to far-sighted evaluation. Most Markov reward and decision processes are discounted because are mathematically convenient to discount rewards and avoid infinite returns in cyclic Markov processes. Sometimes it is possible to put $\gamma = 1$ when all sequences terminate. [4]

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

2.6 Value Function

One of the most important issues in reinforcement learning algorithms is the evaluation of how good a given situation is for the agent, or to further specify, how good an action chosen for a given situation is. To clarify that we need a value function that strictly follows the policy we want to implement. Recall that a policy, π, is a representation of each state, $s \in S$, and action, $a \in A(s)$, to the probability $\pi(a|s)$ of taking action a when we are in state s. Informally, the value of a state s under a policy π, denoted by $v\pi(s)$, is the expected reward when starting from state s and following π thereafter. The value of the terminal state, if any, is always zero. Function $v\pi$ is called the state-value function for policy π. For MDPs, we can define $v\pi(s)$ formally as:

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right]$$

Similarly, the definition of an action value function where the intelligent agent chooses an action a in state s under a policy π, denoted qπ(s, a) and describing the expected return, is written as follows:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right]$$

We call qπ the action-value function for policy π. [4]

2.7 Bellman Equations

In the late 1950s the term optimal control was coined in connection with the problem of designing a controller to minimize a measure of a dynamical system's behavior over time. One of the approaches to solving the problem was that of Bellman. This way introduces the concepts of the state of a dynamic system, a value function or optimal return function to define a functional equation which is called Bellman. That class of methods solving optimal control problems and is very important for the reinforcement learning problems. The Bellman equations formulate the problem of maximizing the expected sum of rewards. Also, can be formulated in terms of a recursive formula and succeed in dividing the optimization problem into smaller subproblems.

The state-value function can be decomposed into immediate reward plus discounted value of successor state as:

$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right]$$

The action-value function can similarly be decomposed as:

$$q_\pi(s, a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a\right]$$

The optimal state-value function v∗(s) is the maximum value function over all policies. If policy π is such that in each state s it selects an action that maximize value, then π is an optimal policy. If we want to find the optimal policy we need to use greedily the v∗(s). We can formulate the optimal state-value function as:

$v_{*}(s) = \max_\pi v_\pi(s)$

The optimal action-value function q∗(s, a) is the maximum action-value function over all policies. We can formulate the optimal action-value function as:

$q_*(s, a) = \max_\pi q_\pi(s, a)$ [4]

2.8 Model-free Methods

Model free methods doesn't need any environment model at all. In Model-free Algorithms an intelligent agent never learns task *T* and environment *E* explicitly. A strategy of this kind relies on stored values for state-action pairs. When the environment of a free model agent changes the way in which it reacts to the agent's actions, the agent needs to obtain new experience in the changed environment, during which it can update its policy and/or value function. An agent to change the action its policy specifies for a state, or to change an action value associated with a state, it must move to that state, act from it, possibly many times, and experience the consequences of its actions. At the end of the learning, agent knows how to act, but doesn’t know anything about the environment. These methods can be divided into 2 categories, depending on whether they change their policy to produce new actions. The first category is On-policy methods, which use their current policy to update their policy and produce new actions. The policy that is used for updating and the policy used for acting is the same. An algorithm of this category is the SARSA (state-action-reward-state-action). The other category is the off-policy methods which use search policies and compare them with their current policy to improve their policy. An important algorithm in this case is the Q-Learning. Q-learning is called off-policy because the updated policy is different from the behavior policy. Deep learning algorithms are model-free methods.

SARSA (state-action-reward-state-action). On-policy algorithm

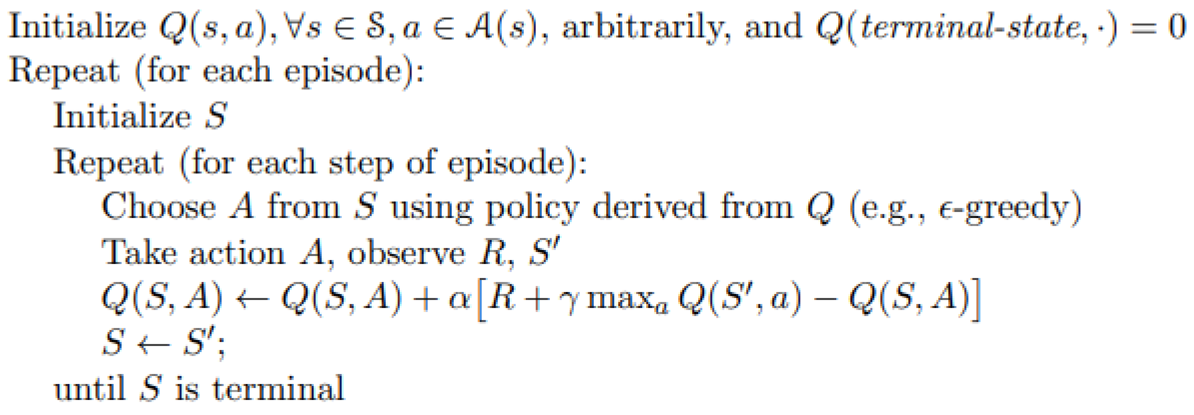
The Sarsa algorithm is an algorithm for TD-Learning. Unlike Q learning, the maximum reward for the next state is not necessarily used to update Q values. A new action, and hence the reward, is chosen using the same policy that determined the initial action. The name Sarsa actually comes from the fact that updates are made using the quintuple Q(s, a, r, s', a'). Where: s, a are the initial state and action, r is the reward observed in the next state, and s', a' are the new state-action pair. The pseudocode form of the algorithm is:

```
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
    Repeat (for each step of episode):
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until $S$ is terminal
```

[3]

Q-Learning. Off-policy algorithm

Q-Learning is an algorithm for Temporal Difference learning. It can be proven that given sufficient training under any e-soft policy, the algorithm converges with probability 1 to a close approximation of the action-value function for an arbitrary target policy. Q-Learning learns the optimal policy even when actions are selected according to a more exploratory or even random policy. The pseudocode form of the algorithm is:



```
Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Repeat (for each step of episode):
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$;
    until $S$ is terminal
```

[3]

Its simplest form, one-step Q-learning, is defined by $Q(S_t , A_t) \leftarrow Q(S_t , A_t) + \alpha \ [R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t , A_t)]$.

2.9 Model-based Methods

The basic idea behind model-based methods is that a problem becomes simpler if it is modelled. In fact, a model is the transformation of a set of assumptions into a precise mathematical form. This set of assumptions creates a description of the world, which can be used to learn the system. In model-based machine learning, the model is then used to create a bespoke algorithm to answer a specific question about the problem domain, such as making a prediction or performing some reasoning. This type of machine learning can be used mainly for either classification or clustering or control. This is why model-based methods are popular in robotics applications. Deep Neural Networks is an example of model (model-based machine learning). So unlike model-free methods it is clean that these methods try to acquire information from the environment they interact with. Model-based algorithms are grouped into four categories to highlight the range of uses of predictive models. [7]

Analytic gradient computation:

Assumptions about the form of the dynamics and cost function maybe aren't the best solution but are convenient because they can yield closed-form solutions for locally optimal control, as in the LQR framework. Even when these assumptions are not valid, receding-horizon control can account for small errors introduced by approximated dynamics. Similarly, dynamics models parametrized as Gaussian processes have analytic gradients that can be used for policy improvement. Controllers derived via these simple parametrizations can also be used to provide guiding samples for training more complex nonlinear policies.

Sampling-based planning:

We no longer have assurances of local optimality in the completely general case of nonlinear dynamics models and are forced to use action sequence sampling. In the most basic form of this strategy, called random shooting, prospective actions are randomly selected from a fixed distribution, evaluated using a model, and the action that is judged to be most promising is selected. More complex variations, such as the path integral optimal control (PIOC) or cross-entropy method (CEM) used in PlaNet, PETS, and visual foresight, iteratively change the sampling distribution (used in recent model-based dexterous manipulation work). Yet, in discrete-action scenarios, searching over tree structures is more frequent than repeatedly honing a single itinerary of waypoints. Iterated and MCTS are two common tree-based search algorithms that have recently produced outstanding results in game playing. Sampling-based planning, in both continuous and discrete domains, can also be combined with structures physics-based, object-centric priors.

Model-based data generation:

A method for artificially expanding a training set's size is a crucial component of many machine learning success stories. Although it is challenging to describe a manual data augmentation process for policy optimization, we may think of a predictive model as a learnt technique for producing synthetic data. The Dyna Algorithm by Sutton, which alternates between model learning, data production under a model, and policy learning using the model data, is the source of the initial idea for such a combination. This approach has been scaled to image observations, integrated with iLQG, model ensembles, and meta-learning, and is open to theoretical study. The application of a model to enhance target value estimates through temporal difference learning is a close relative of model-based data production.

Value-equivalence prediction:

A final technique that does not neatly fit into the model-based versus model-free categorization is to include computation that resembles model-based planning without supervising the model's predictions to match actual states. Instead, the model constrains plans to match real-world trajectories only in terms of their predicted cumulative reward. These value-equivalent models have been demonstrated to be effective in high-dimensional observation spaces where traditional model-based planning has proven difficult.

2.10 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) is a subfield of machine learning that combines Reinforcement Learning and Deep Learning. As previously described, RL considers the problem of an intelligent agent that learns to make decisions by trial and error. DRL allows an agent to make decisions from unstructured input data. This can be done because deep neural networks can extract information from unstructured data due to the hidden layers of neurons and their nonlinear activation functions. A deep neural network (DNN) is an artificial neural netw work (ANN) with multiple layers between the input and output layers and it consists of the same components as the simple neural networks. DRL algorithms can take as input large amount of data and decide what actions to perform, to optimize the reward. Also, DRL can solve problems that reinforcement learning cannot do. This is why Deep Reinforcement Learning is very popular nowadays, as it solves real-life problems. DRL applications cover the fields of autonomous driving, robotics, gaming, healthcare, and economy.

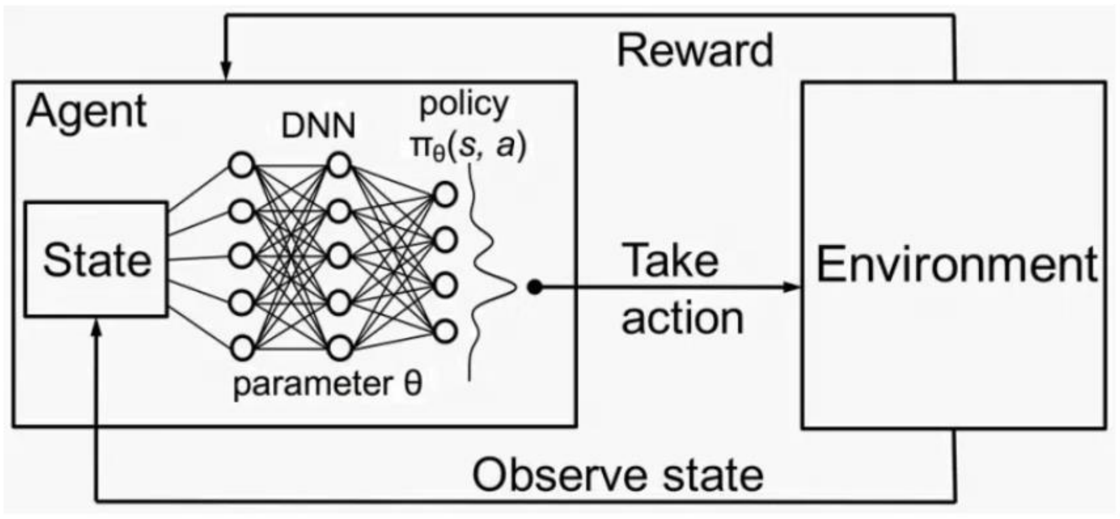


Figure 2.3 Deep Reinforcement Learning Procedure

2.11 Deep Q-Learning

The core of DRL is the Deep Q-Learning algorithm.

Deep Q-Learning with Experience Replay

```
1: Initialize replay memory D to capacity N
2: Initialize action value function Q with random weights
3: **repeat**
4:     Observe initial state $s_1$
5:     **for** t = 1, T **do**
6:         Select an action $a_t$ using policy from Q
7:         Implement action $a_t$
8:         Observe the reward $r_t$ and the new state $s_{t+1}$
9:         Store transition $(s_t,a_t,r_t,s_{t+1})$ in replay memory buffer D
10:        Sample random minibatch of transitions $(s_j,a_j,r_j,s_{j+1})$ from replay memory buffer D
11:        Calculate target for each transition minibatch
12:        **if** $s_j$ is terminated **then**
13:            $y_j = r_j$
14:        **else**
15:            $y_j = r_j + \gamma \max_{a'} Q(s_{j+1},a';\theta)$
16:        **end if**
17:        Train the Q network on $(y_j - Q(s_j,a_j;\theta))$ with the formula $\nabla\theta iQL(\theta_i^Q) = Es\sim\rho_{\pi(\cdot)},a\sim\pi_{(\cdot)},_{s'}\sim\epsilon\ [(r + maxa_{'}Q(s',a';\theta_i^Q-1) - Q(s,a;\theta_i^Q))\nabla\theta iQQ(s,a;\theta_i^Q)]$
18:    **end for**
19: **until** terminated
```

CHAPTER 3

DQN For Autonomous Driving

3.1 Autonomous driving

One of the most popular and exciting applications of deep reinforcement learning is self-driving cars. In recent years, as technology has advanced, the need for safe, fast, and environmentally friendly transportation has increased. One such form of transportation that meets these demands is autonomous vehicles. A self-driving vehicle today is furnished with a amount of sensors such as cameras, RADAR, LiDAR and Ultrasonic actuators in order to understand the environment in which it is located at any given time. The data received via these sensors is useful because through the appropriate algorithms it can understand the density of traffic, identify objects to avoid them, if necessary, choose the best lane and avoid collisions. The main techniques that an autonomous vehicle uses to make the right decisions while driving can be categorized. The first category is Environment Perception. This category of techniques includes visual navigation, laser navigation and radar navigation. All these methods provide valuable data to a vehicle to enable it to perceive other vehicles, understand the distance from them and act accordingly. Another important category of methods used by self-driving vehicles is pedestrian detection. Pedestrian detection is a hard challenge for researchers in the field, as so far there are significant difficulties in this type of problem. Pedestrian detection methods are primarily computer vision algorithms and either 3D Lidar or camera is used to acquire the data. Another class of methods used for vehicle navigation on road networks is Path Planning. A good autonomous car must be able to analyze the information it acquires from the environment and accurately predict the best path to follow. A traditional method that is also applied in the field of robotics for path planning is the cubic polynomial algorithm. However, in a modern car mixed methods are used to calculate it as best as possible, which take consideration data from 3D SLAM space mapping, odometer system data, GPS data and use deep convolutional neural networks. Finally in the real world we live in, it would be impossible not to consider the risk of hacking. The concern lies in the fact that a system such as an autonomous car to perform its operations and to understand its environment, uses many sensors that interact with telecommunication systems. Thus, such a vehicle is vulnerable to cyber-attacks, which is very important, as it is not about a loss of money, but can lead to an injury at best. So, the last category of methods, which is used in self-driving vehicles is vehicle cyber security. All control operations of the vehicle must be secure and that is why the more autonomous car models are used, the more algorithms are used to counter such attacks. For all these characteristics described above, more and more people trust self-driving cars and as a result, more and more automakers are turning to their manufacturing.[6]

Figure 3.1 Google self-driving car

3.2 SUMO

Eclipse SUMO (Simulation of Urban MObility) is an open source, portable, microscopic and continuous multi-modal traffic simulation package[8] designed to handle large networks. SUMO is developed by the German Aerospace Center. It has been freely available as open-source since 2001, and since 2017 it is an Eclipse Foundation project. Is used worldwide and is downloaded over 35.000 times every year. Traffic simulation within SUMO uses software tools for simulation and analysis of road traffic and traffic management systems. New traffic strategies can be implemented via a simulation for analysis before they are used in real-world situations. SUMO is used for research purposes like traffic forecasting, evaluation of traffic lights, route selection and the users of that package are able to make changes to the program source code through the open-source license to experiment with new approaches.[9] That package has implemented in C++, Java, and Python programming languages.

Figure 3.2 SUMO Logo

Applications of SUMO

SUMO has been used within several projects for answering a large variety of research questions:

- Evaluate the performance of traffic lights, including the evaluation of modern algorithms up to the evaluation of weekly timing plans.

- Vehicle route choice has been investigated, including the development of new methods, the evaluation of eco-aware routing based on pollutant emission, and investigations on network-wide influences of autonomous route choice.
- SUMO was used to provide traffic forecasts for authorities of the City of Cologne during the Pope’s visit in 2005 and during the Soccer World Cup 2006.
- SUMO was used to support simulated in-vehicle telephony behavior for evaluating the performance of GSM-based traffic surveillance.
- SUMO is widely used by the V2X community for both, providing realistic vehicle traces, and for evaluating applications in an on-line loop with a network simulator.
- AI training of traffic light plans.
- Simulation of the traffic effects of autonomous vehicles and platoons.
- Simulation and validation of autonomous driving function in cooperation with other simulators.
- Simulation of parking traffic.
- Simulation of railway traffic for AI-based dispatching of vehicles.
- Traffic safety and risk analysis.
- Calculation of emissions (noise and pollutants). [10]

Components

The SUMO package contains the following important components and not only:

- **SUMO**:

  command line simulation

- **SUMO-GUI**:

  simulation with a graphical user interface

- **NETCONVERT**:

  network importer

- **NETGENERATE**:

  abstract networks generator

- **OD2TRIPS**:

  converter from O/D matrices to trips

- **JTRROUTER**:

  routes generator based on turning ratios at intersections

- **DUAROUTER**:

  routes generator based on a dynamic user assignment

- **DFROUTER**:

  route generator with use of detector data

- **MAROUTER**:

  macroscopic user assignment based on capacity functions

- **NETEDIT**:

  Visual editor for street networks, traffic lights, detectors, and further network elements

- **EMISSIONSMAP**:

  This tool generates matrices of emissions for a given range of velocities, accelerations, and slopes, given the vehicle's emission class mainly

- **POLYCONVERT**:

Imports geometrical shapes (polygons or points of interest) from different sources, converts them to a representation that may be visualized using sumo-gui

Features

The SUMO package provides these Features:

- **Microscopic Simulation**

  Simulate the movement of every individual object by modeling all vehicles, pedestrians and public transport explicitly

- **Multimodal Traffic**

  Combine different modes of transportation and simulate cars, buses, trains, bicycles, pedestrians, public transport and more

- **Online Interaction**

  Control the behavior of all simulation objects during a live simulation with the Traffic Control Interface (TraCI)

- **Network Import**

  Import road networks from common network formats such as OpenStreetMap, VISUM, VISSIM, NavTeq, MATsim and OpenDRIVE

- **Demand Generation**

  Use traffic counts on streets and junctions, origin-destination-matrices or virtual population models to generate realistic demand profiles

- **Traffic Lights**

  Modify traffic light schedules visually with netedit, import schedules from external data sources or generate schedules automatically

- **Performance**

Boost your simulation with an unlimited network size, an unlimited amount of simulated vehicles and an unlimited simulation time*

- **Portability**

Use SUMO on a variety of platforms (Windows, Linux or macOS) as it is implemented in C++ and Python and uses portable libraries

- **Open Source**

Use and modify SUMO according to your needs as it is available under Eclipse Public License v2.0 and GNU General Public License v2.0 [10]

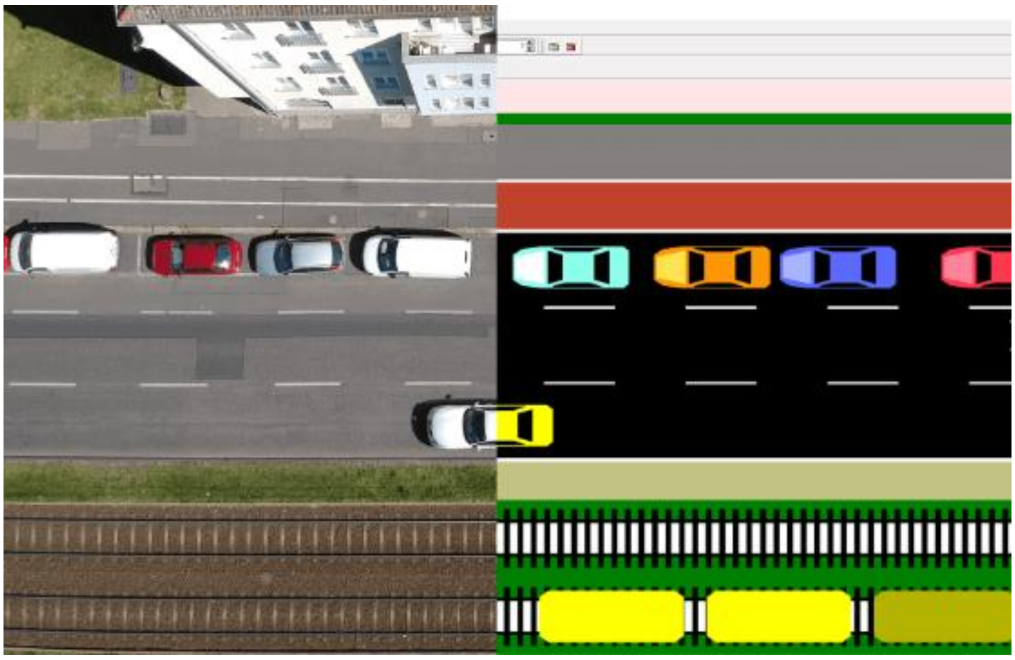


Figure 3.3 Real world in SUMO environment

In Simulation of Urban MObility there are two ways to build a road network. The first is to set up a custom road network via netedit. The second way and more realistic is to construct the road network from a real map. This map can be imported from OSM(Open Street Map) and with the appropriate commands in the terminal and with the netedit tool it can be converted to the appropriate format to be simulated in sumo. This is the hard way, as sumo provides the option of the OSM Web Wizard, where through a graphical interface we can select the road network we want and build a realistic simulation in minutes. It provides the option to set up the parameters of the simulation. We can select the type of the vehicle that we want to load , like as cars, motorcycles, buses, trucks, taxi, etc. The other parameters are about the traffic flow of the network. We can determine the number of the vehicles that will enter the network and also the generation rate of vehicles.

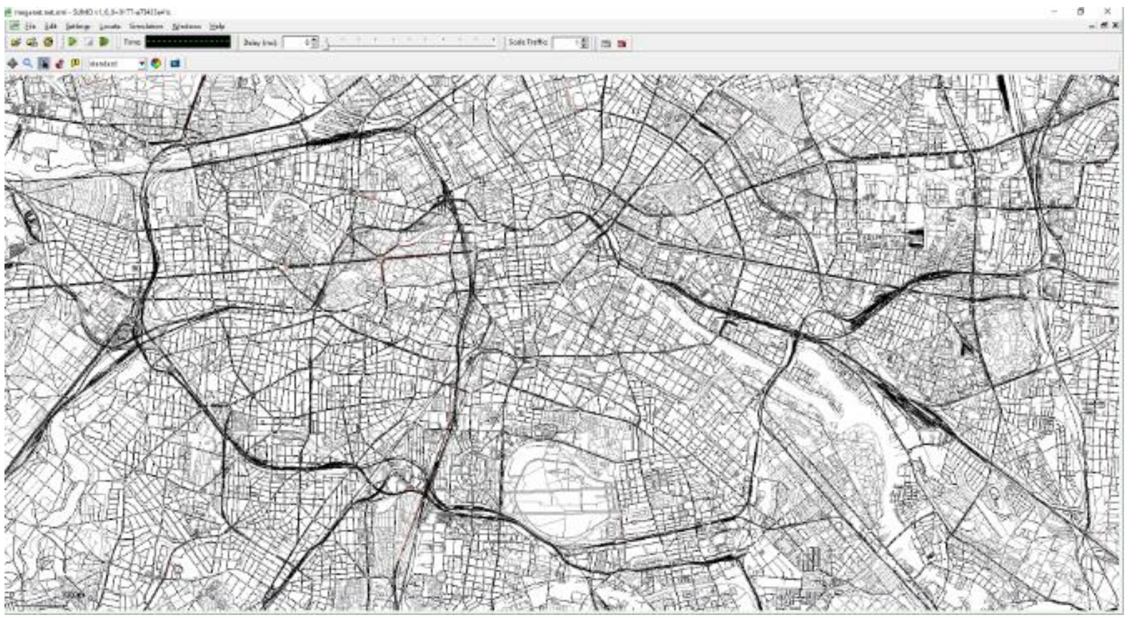


Figure 3.4 Real map in SUMO environment

3.3 Proposed Method

In our training scenario 71 mopeds enter the road network sequentially. Their goal is to arrive at their destination quickly and with safety. On the road network there are hundreds of other vehicles which are not under the agent's control. The intelligent agent we use receives the information from the 71 mopeds and trains itself to increase the reward. The maximum steps of the scenario are 900. At each timestep the agent does 71 new observations. After the observations he is trained with the e-greedy DQN algorithm and chooses an action for all the mopeds. An observation includes:

- The current velocity.
- Speed of the leader vehicle.
- Distance from the leader vehicle.
- Speed of the follower vehicle.
- Distance from the follower vehicle.

When the simulation starts, we wait until the first moped enters the road network. In the next step, we start receiving the data from all the mopeds that have entered the road network and store it in the replay buffer. Then, after the vehicles controlled by the neural network have done their observations, we continue the simulation to the next step. Before performing the actions, we apply a control method to check which mopeds are still on the road network and remove from the list of vehicles those that have reached their destination or collided. After the checking process is completed for each moped, the neural network selects an action for each one. At this point we emphasize that the actions are made either randomly or based on the decision of the neural network. This is determined by the epsilon factor. After the action is executed, we do the observations for all mopeds, evaluate their actions based on the reward policy, and store the observations of the last two steps in the replay buffer. Next, we apply the DQN learning algorithm to update the network weights. This process is repeated at each step until the episode's end.

3.4 States

As described above the agent collects states from 71 mopeds at each step. A state in a step of a moped consists of its speed, distance from the leader vehicle, speed of the leader vehicle, speed of the follower vehicle and distance from the follower vehicle. All this information is obtained with the interface provided by Simulation of Urban Mobility. However, the data is not always without noise. When a vehicle reaches the end of an edge or enters a junction, neither the distance from it nor the speed is provided to the following vehicle. Thus, the following vehicle is not aware that there is a car in front of it and there is a high probability of a collision. To reduce this risk, we use our own technique where each moped stores the leader that was last in front of it. So, when the leader car enters an intersection and we do not receive any information about its status, we use the id of the last leader and get its speed if it is still in the simulation. Next step before we collect the distance is to use the corresponding function provided by sumo to find the coordinates of the last leader vehicle. After finding the coordinates of the last leader vehicle we compare them with those of the moped, applying the Euclidean distance formula. This way the intelligent agent has cleaner data at its disposal and can store more accurate states in the replay memory buffer. This has the effect that after sufficient training it can take better decisions.

A state s can be written as a list of 5 elements:

s = [moped_speed,leader_veh_speed,leader_veh_dist,follower_veh_speed,follower_veh_dist]

3.5 Actions

Because the algorithm we use is e-greedy DQN, initially the actions chosen by our agent are random. However, since we perform one learning per simulation step the epsilon decreases quite fast and reaches 0.1 which is the threshold we have set. So, from this point onwards the intelligent agent is able, to perform 5 acceleration actions. The acceleration options range from -2 to +2 (m/s^2). To be more specific the actions that agent can choose are A= [-2, -1, 0, 1, 2]

3.6 Reward

To best approach the solution of this reinforcement learning problem, in order to have as few collisions between vehicles as possible, we designed an efficient reward function, which serves our policy. We have set as safe distance threshold the value 20 meters. The reward policy fully determines the behaviour of the agent and therefore of the 71 mopeds during the simulation. The reward function is simple as we can point below.

Table 1 reward function

If a moped arrives at its destination	Reward = 1000
If a moped involved in a collision	Reward = -1000
If the distance from the front vehicle is more than 20m	Reward = $0.5 \cdot speed\_moped$
If the distance from the front vehicle is less than 20m	Reward= $-\frac{dist\_thres}{factor} \cdot \left(\frac{speed_{\_moped} + 200}{20}\right)$

speed_moped: is the speed of a moped.

factor: Because we want to avoid infinite negative values that can convert our system to biased, we use the factor. If the distance from the front vehicle < 1 then factor = 1, else factor is min(front_veh_dist, dist_thres).

In fact, with this reward policy, we give a fairly negative signal to the agent whenever a moped is involved in a collision. Instead, if it manages to arrive at its destination, it receives the opposite positive reward. This has the effect of trying to avoid collisions, in order to gain the best possible reward. In the intermediate non-terminal states our policy states that if the distance from the vehicle in front is more than 20m i.e., the safe threshold we have set, then it will be rewarded with half of its speed. So, the agent understands that depending on how much more speed it develops, the more reward it will earn. Finally, the last reward formula is for cases where the distance from the vehicle in front is less than 20m. Since we want the speed to decrease and the distance to increase, we used the speed of the moped and the distance from the vehicle in front, because these quantities are inversely proportional. This way if a moped decreases the speed and increases the distance, then it will receive as little negative reward signal as possible.

3.7 Algorithm Implementation

Q-Learning for autonomous moped

```
1: Initialize the replay memory buffer
2: Initialize the neural network with random weights
3: while E < max_episodes do
4:     while $t$ < episode_max_steps do
5:         for moped m in simulation do
6:             Observe state $s^m_t$ .
7:             Choose an action $a^m_t$, according to e-greedy algorithm.
8:         end for
9:         t = t + 1
10:        for moped m in simulation do
11:            Observe new state
12:            Receive reward $R(s^m_t, a^m_t)$
13:            Store previous_state, current_state, action, terminal_factor, and reward in memory buffer.
14:        end for
15:        Train the neural network according to DQN algorithm.
```

```
16:     **end while**
17:     Store the weights of the Network if the agent achieves the best score in E episode.
18:     E = E + 1
19: **end while**
```

CHAPTER 4

Results

4.1 Training Scenario

Before we present the results of the training, we will describe the parameters that used for the training of neural network:

Table 4.1 : e-Greedy DQN parameters

Gamma	0.99
Epsilon	1
Episode end	0.01
Episode decay	5e-4

In the next table we present the most important components of the artificial neural network's structure.

Table 4.2 : Artificial Neural Network and Training parameters

Number of episodes	3000
Learning rate	0.003
Batch size	128
Replay buffer	100.000
Number of neurons input layer	5
Number of neurons 1st hidden layer	256
Number of neurons 2nd hidden layer	128
Number of neurons output layer	5
Activation function 1st hidden layer	ReLU
Activation function 2nd hidden layer	ReLU
Activation function input/output layer	Linear
Optimization Algorithm	Adam

The area in which the training scenario was implemented is presented in the photo below. More specifically, it is a road network in the center of Berlin. This road network according to the netedit tool contains 169 nodes and 280 edges. 238 cars and 71 autonomous mopeds entered the network for the experiment. A new car enters the road network every 3 seconds and one new moped every 5 seconds.

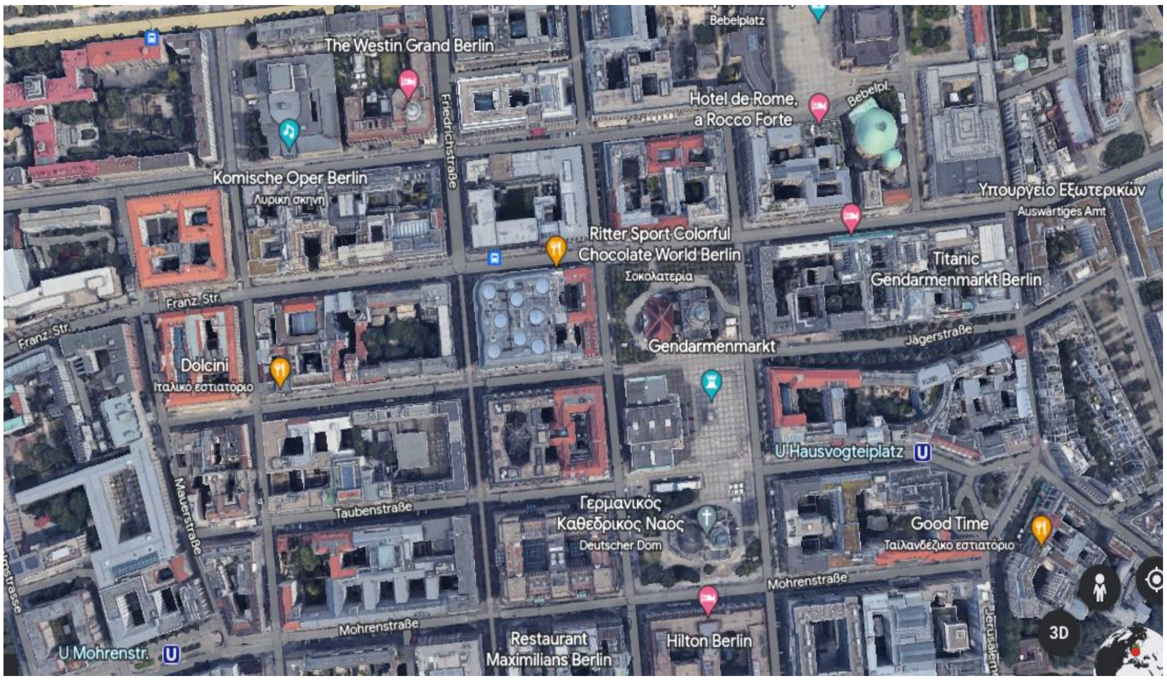


Figure 4.1 Map of Training

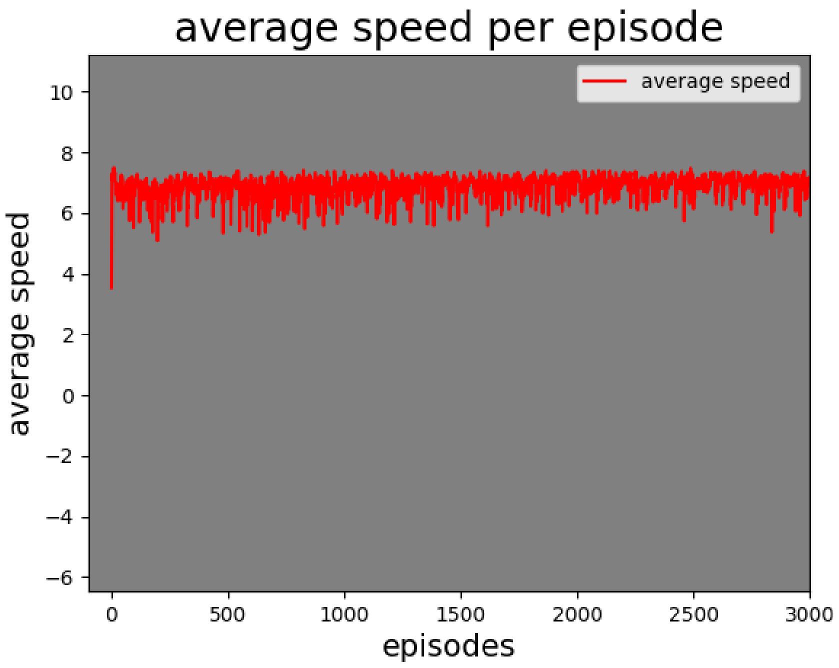


Figure 4.2

Average speed of 71 mopeds/episode

Considering that this area has a high traffic volume and according to netedit (SUMO component) the maximum allowed speed for vehicles on almost all edges of the scenario is about 9km/h(simulation velocity), mopeds quickly learn the policy and adapt to the environment, depending on the speed of other cars. The mean speed approaches 7.3 km/h because mopeds in many cases slow down to avoid collisions. The speed in real word according to SUMO is 26.259 km/h. This kind of speed is desired for a vehicle as moped in a road network like this.

The next plots present the average score/episode and collisions/episode. As we can observe the score/reward graph has a similar performance to the speed graph. This is because the reward is proportional to the speed. Also, the second graph, on the next page describes the relationship between collisions and episodes. We can note that when the simulation starts at the first episode there are many collisions. Quite quickly the curve is reduced because for each step we implement and one training of the Neural Network. The intelligent agent obtains significant much experience and adapts its behavior.

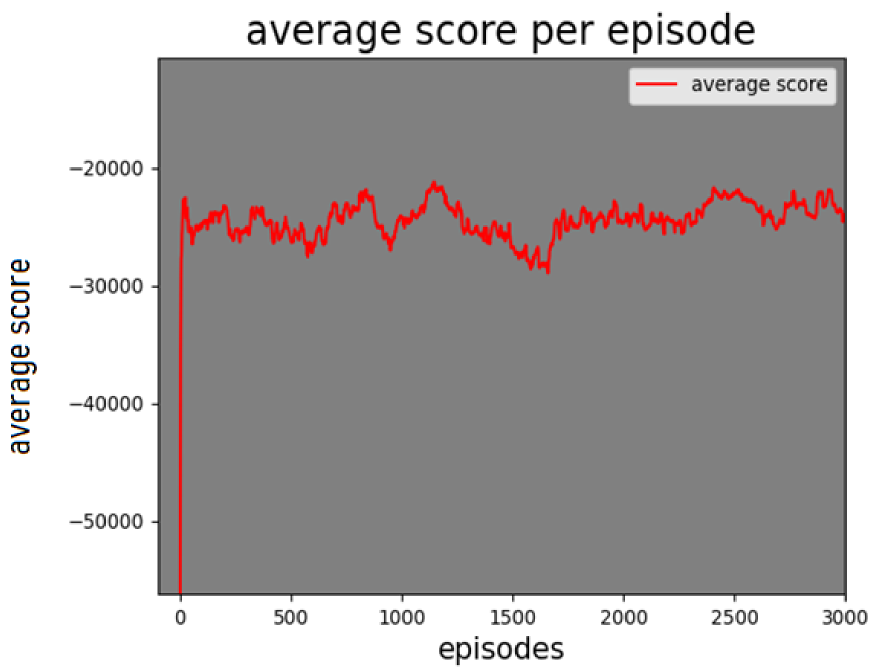


Figure 4.3

Average score of 71 mopeds/episode

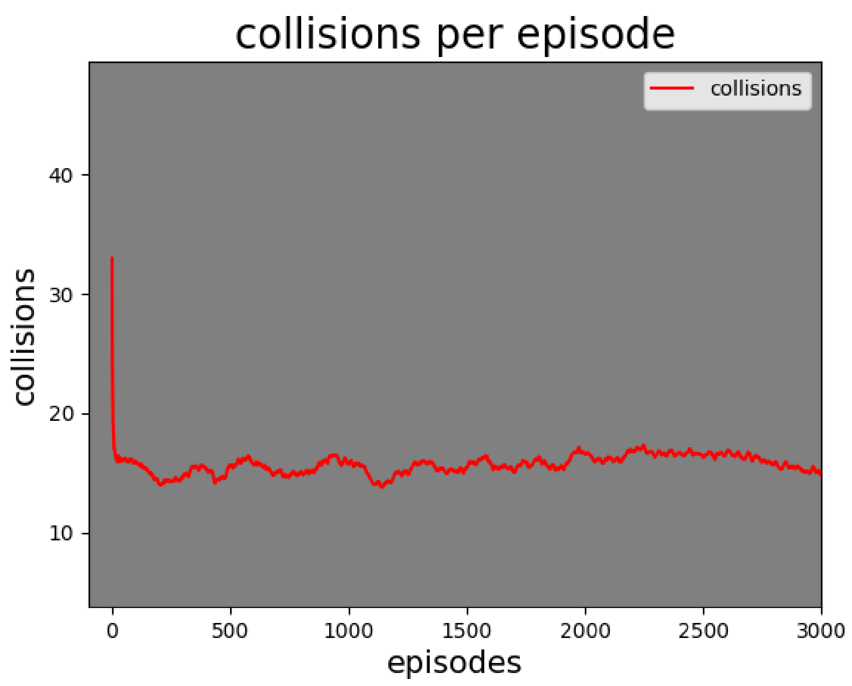


Figure 4.4

Average collisions of 71 mopeds/episode

4.2 Test Scenarios

To test the method that we followed, we used 2 different road networks. One map is a known environment for the agent, while the other is known only in part of the. To make it harder for the intelligent agent, to react efficiently to extreme situations during learning, we had set the conflict factor to 1.2 (for the default option-physical collision the factor should be 0). This means that our vehicles were more likely to collide with other cars. In the Test procedure we will reduce the factor to 1.

The first scenario which was implemented on the same map as the road network we used to train the neural network, consists of the following components:

- Duration = 25 minutes
- Self-driving mopeds = 100
- Cars = 300

The first experiment had as result 0 collisions. The mean velocity of self-driving mopeds in simulation is 8.0. This means that in real world according to SUMO is 28,77km/h. The plot below describes the mean speed of mopeds per simulation step (SUMO speed).

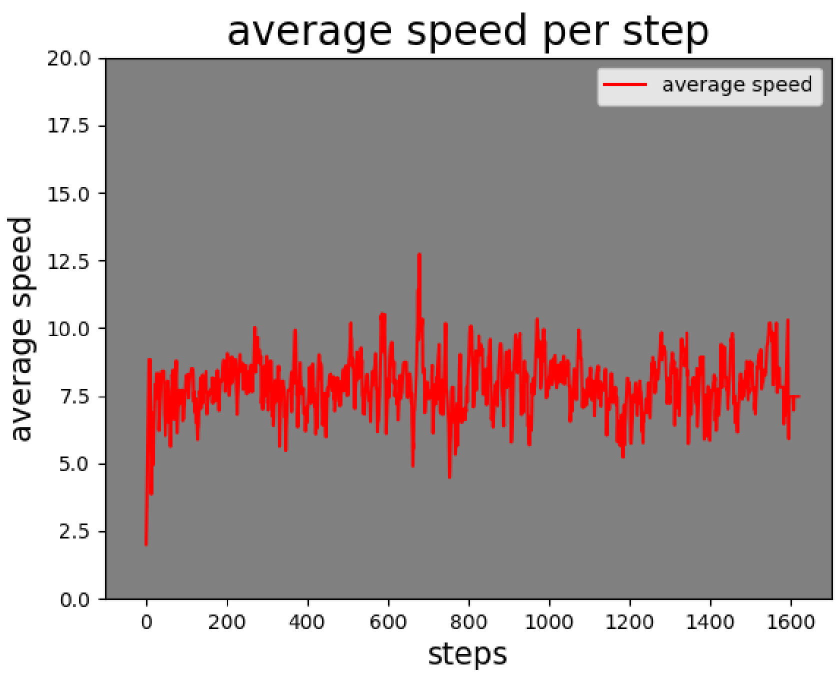


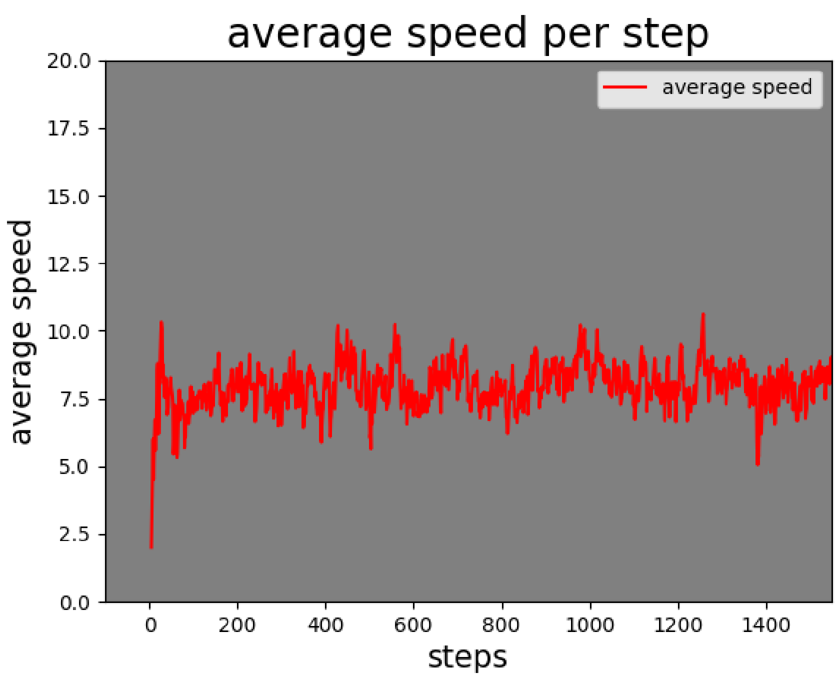
Figure 4.5

Average speed per step - 1st scenario

The second scenario implemented on different road network and consists of the following components:

- Duration = 25 minutes
- Self-driving mopeds = 150
- Cars = 300

This experiment had as result 0 collisions. The mean velocity in simulations for the self-driving vehicles is 8.01. This means that in real world according to SUMO is 28,92km/h. The plot below describes the mean speed of mopeds per simulation step (SUMO speed).



4.6 Average speed per step - 2nd scenario

In conclusion we can support that the test scenarios prove the efficiency of the method according to the results of the experiment. The vehicles approximate 30 km/h (Real life) without colliding. Considering the maximum speeds set by SUMO for the specific type of vehicle we used, as well as the maximum speed limit of the roads where the simulation was performed, we can say that we achieved the goal which is described in the abstract of this work. The self-driving mopeds fast and safely reach their destination. In the image below is the new map where the scenario was implemented.

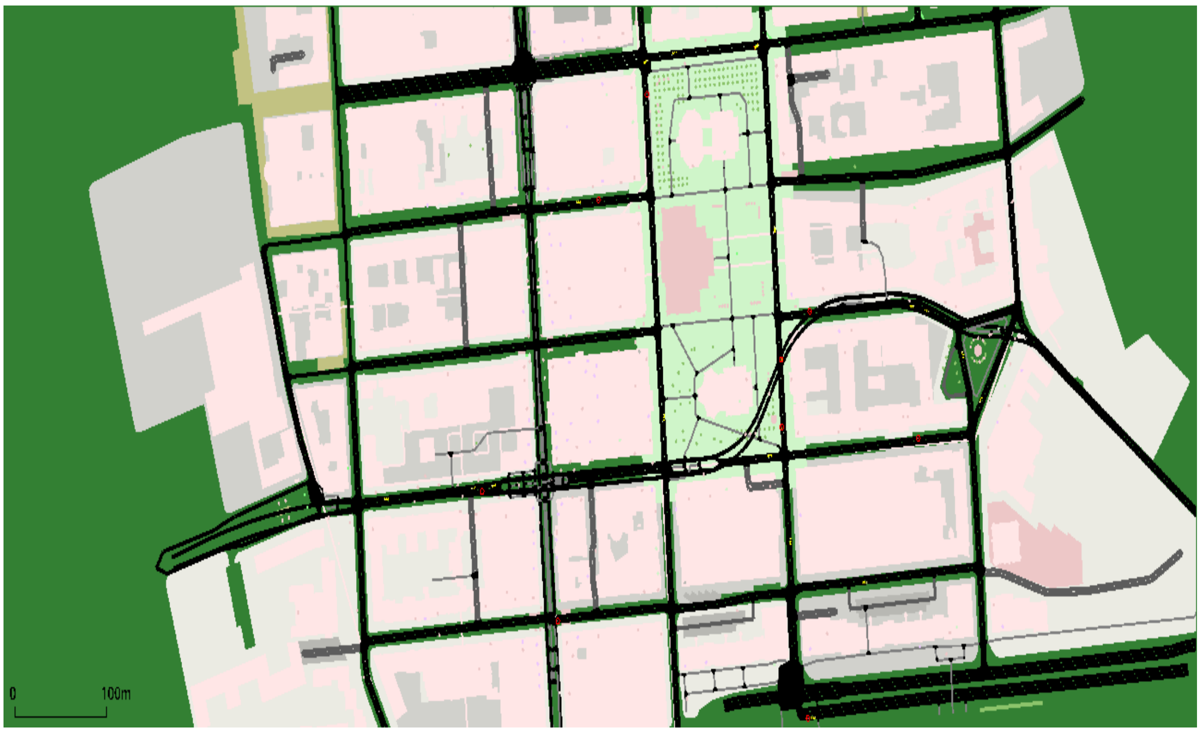


Figure 4.7

Road network – 2nd scenario

CHAPTER 5

Conclusion And Future Research

In this thesis we tried to address autonomous driving from a unique perspective. The traditional approach off reinforcement learning problems is to use one intelligent agent for one vehicle or a multiagent system for many vehicles. For training we used multiple mopeds which provided valuable information to the neural network, using only one neural network, without multiagent's learning contracts. Thus, by feeding it with many different states we were largely able to deal with the exploration problem. Although in one episode the epsilon parameter was equal to 0.01, which means that the neural network does not choose random actions, the many different states provided by all the mopeds helped it to continue to train and receive new data. The test results in unknown environment, showed better performance than the training results, which is satisfactory. Future research could focus on how we can collect more and cleaner data without high computational cost. It would still be important to research new more efficient reward functions to achieve better and faster results. Finally, reinforcement learning, in addition to the problem of autonomous driving, could in the future also solve the problem of vehicle stability, like the ones we used in this work, so one day they will be able to get on the road.

BIBLIOGRAPHY

[1] K. Diamantaras and D. Botsis, “Machine Learning”, Klidarithmos Publications, 2019

[2] I. Goodfellow, Y. Bengio and A. Courville, “Deep Learning”, MIT Press, 2015

[3] R. S. Sutton and A. G. Barto, “Reinforcement learning: An introduction”, MIT Press, 2014

[4] D. Silver, “Lecture notes on reinforcement learning”, 2015.

[5] R. Divyam, “Deep reinforcement learning for bipedal motors” Delft University of Technology, August 2017.

[6] D. Parekh, N. Poddar, A. Rajpurkar, M. Chahal , N. Kumar, G. P. Joshi and W. Cho, Article “A Review on Autonomous Vehicles: Progress, Methods and Challenges”, electronics, MDPI, Available Electronics | Free Full-Text | A Review on Autonomous Vehicles: Progress, Methods and Challenges (mdpi.com), 2022

[7] M Janner, “Model-Based Reinforcement Learning: Theory and Practice”, BERKELEY ARTIFICIAL INTELLIGENCE RESEARCH, 2019

[8] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, “Microscopic traffic simulation using sumo” 21st IEEE International Conference on Intelligent Transportation Systems. IEEE, 2018. [Online]. Available: https: //elib.dlr.de/124092/

[9] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker-Walz, “Recent development and applications of sumo - simulation of urban mobility” International Journal On Advances in Systems and Measurements, vol. 34, 12 2012.

[10] SUMO site, https://www.eclipse.org/sumo/