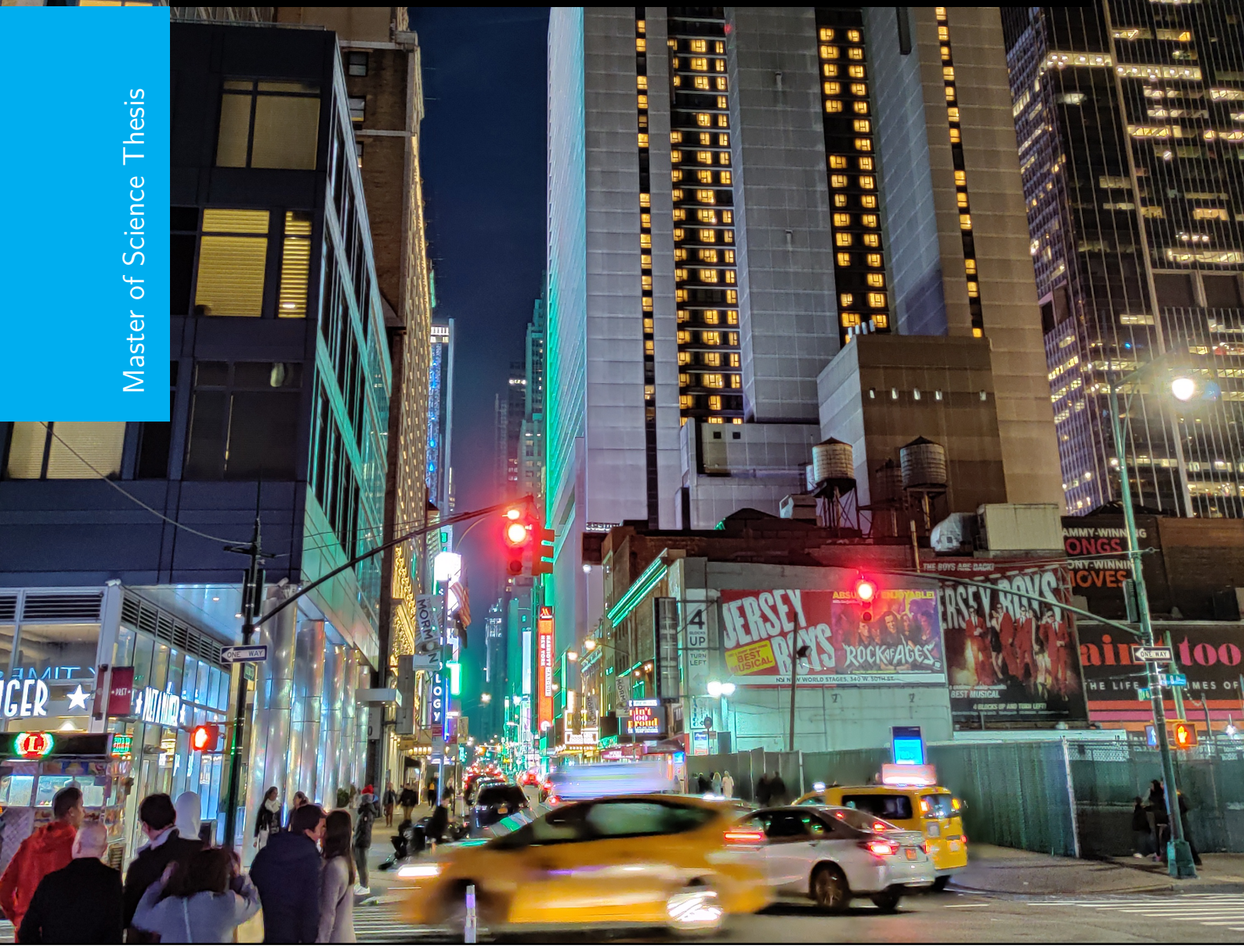
Combining MPC and reinforcement learning in a model-reference framework for urban traffic signal control

W.J. Remmerswaal

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

Combining MPC and reinforcement learning in a model-reference framework for urban traffic signal control

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

W.J. Remmerswaal

January 20, 2022

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Table of Contents

Acknowledgements

With the delivery of this thesis, my student life comes to an end. It simultaneously preludes the start of my professional career. Writing a thesis during a pandemic is not the most glamorous way to end this journey and it has certainly has not been the easiest. I am happy and proud that I can present the final results. Of course, this had not been possible without the support of many and I would like to thank these people.

First of all, I would like to thank my daily supervisor Dingshan, for his everlasting trust and encouragement during the whole process. We had fruitful discussions about the thesis, but it was also interesting and fun getting to know you and sharing our cultural experiences.

Thanks to my supervisor Bart De Schutter for his support, critical attitude, and helpful insights during this process.

Thanks go to Midas for always being there for me, my parents for their unconditional support during my whole studies, and all other friends and family who made this year a lot more fun. Especially, I would like to thank my roommates Anneli, Hannah, Hanneke, Lisette, and Vera for spending all nights with me during the lockdown(s) and providing some much-appreciated distraction.

Delft, University of Technology W.J. Remmerswaal
January 20, 2022

Chapter 1

Introduction

When the first Dutch traffic jam occurred in May 1955, it was an interesting sight for anybody passing. It was so interesting in fact that some oncoming traffic stopped at the other side of the road to have a look at this fascinating event. In this day and age, however, a society without traffic congestion is unimaginable. The last day without any traffic jams in the Netherlands was on 30 December 2007. Since that day congestion has only aggravated due to the increasing number of vehicles, and it is still worsening in that the Dutch government predicts an increase in loss of travel time on the Dutch roads of 20% in 2025 with respect to 2019 [24].

Traffic congestion has a negative impact on the world in terms of environmental, economic and social effects. Through congestion, the emission of greenhouse gasses increases, as cars will spend more time on the road and often have nonsmooth deceleration and acceleration patterns [3]. This last pattern is especially appearing in urban areas where cars are regularly slowed down by traffic lights. In 2019 19% of the emission of greenhouse gasses in the Netherlands was due to road traffic [8]. In the EU 40% of the CO_2 emission and 70% of other pollutants caused by road traffic are due to traffic in urban areas [13], which could all be majorly reduced by the efficient traffic management strategies [3]. In 2018 the economic loss through the congestion of freight traffic in the Netherlands alone is estimated at around 1.4 billion euros [22]. This is mostly due to freeway congestion. Social externalities such as discomfort and stress could be a direct result of travel time loss or noise pollution. Busy roads can also cause feelings of insecurity for other road users such as pedestrians.

In order to minimize traffic congestion and its effects, proper control strategies are to be designed and utilized for optimal use of available infrastructure. The most used control strategy for urban traffic control is the efficient use of traffic signal control systems [40]. The inappropriate utilization of traffic signals is the main cause of urban traffic congestion [11]. The first controllers for urban traffic signal control are fixed-time, i.e. their control strategy is determined offline based on historical data. More efficient are traffic-responsive controllers, which are able to react to traffic situations as they use real-time data. Traffic-responsive controllers also include model-based controllers, one of which is model predictive control (MPC). MPC controllers have shown to be very effective and popular controllers

for urban traffic control [55], as for their flexible and robust nature. However, the use of an MPC controller has some disadvantages as well. Centralized control of large scale systems can become computationally infeasible. Besides, when a system (model) suffers from disturbances and uncertainties the control performance becomes suboptimal.

In recent years the use of data-driven reinforcement learning (RL) for the control of urban traffic systems has gained a lot of interest. Model-free RL algorithms do not need a model to obtain a well-performing control law and they are of adaptive nature, where they can change their control solution according to changing traffic situations. This makes them very capable of dealing with disturbances in a system. Although controllers based on RL techniques have to learn a control law before online application, their online computation time is very low.

The characteristics of MPC and RL controllers complement each other. In this thesis, the aim is to exploit both their advantages by combining MPC and RL in a model-reference framework such that it can be applied to an urban traffic signal control problem.

1-1 Research objective

In the context presented, the main goal of this thesis is:

> *To design a model-reference control framework that combines MPC and RL for centralized control of an urban traffic control network and compare its performance with a conventional fixed-time controller, MPC controller and RL-based controller.*

Two sub-questions are composed to indicate whether the model-reference MPC-RL framework is able to outperform the other controllers.

1. *Can the framework outperform a conventional MPC controller when presented with disturbances and/or uncertainties in terms of system performance?*
2. *Can the framework outperform a conventional RL-based controller in terms of sample efficiency and its performance during the learning process?*

It is hypothesized that these questions will be true for the designed framework. The hypotheses will be assessed by means of a small case study.

1-2 Thesis outline

Here we will discuss the outline of the remainder of this thesis. Chapter 2 discusses all background en preliminaries that are needed for the reader for the rest of the thesis. It includes an introduction of the concept of reinforcement learning and model-reference control. Besides, it discusses some background of urban traffic control including the use of model predictive control. It also includes a description of the BLX model which is the prediction model that is used in the rest of the thesis. In Chapter 3 the design of the model-reference RL framework combined with MPC is defined. Chapter 4 includes the results of two case studies that are performed. Finally, in Chapter 5 the thesis is concluding and future research recommendations are presented.

Chapter 2

Background and Preliminaries

This chapter contains the background information relevant to the research. In Section 2-1 an introduction on reinforcement learning is given. This is followed by an explanation of different kinds of urban traffic signal control in Section 2-2, including model predictive control. In Section 2-3 the prediction model that will be used for the model predictive controller is presented. Lastly, the concept of model-reference adaptive control is introduced in Section 2-4, this control scheme is used a guideline for the control framework designed in this thesis.

2-1 Reinforcement learning

In this section the concept of reinforcement learning (RL) is introduced. RL is an area of machine learning that trains an agent to interact with its environment by rewarding or punishing its behaviour. By trial and error the agent learns how to behave in an optimal manner. The most well-known example of RL must be the victory of an RL agent against the world champion Go player, a very complex board game [45].

RL algorithms aim to train these agents to optimize their behaviour, and can be used to construct a well-performing control law. They are capable of finding solutions for complex problems that are not easily solved by humans. For a large class of RL algorithms, called model-free RL, no model is needed to find the desired control input. This is a big advantage compared to other conventional control methods, such as MPC. A sufficient model is not always present. Besides, models can become very complex when trying to adequately describe the system's dynamics. Furthermore, not requiring a model reduces the online complexity of an algorithms significantly. Another advantage of using RL algorithms for control is that these algorithms can create an adaptive control strategy, this is due to the fact that these agents can keep learning through interaction during online application.

In this chapter the mathematical framework used in formulating an RL problem, the Markov decision process, is explained. We will discuss how an optimal control policy can be found using this framework and some basic model-free RL algorithms are introduced. The main drawback of these basic algorithms is the *curse of dimensionality*, and a way to tackle this

problem is function approximation. In the last section, we discuss the deep Q-network (DQN) algorithm that uses a neural network as a function approximator and other innovative techniques to tackle some major challenges of RL.

2-1-1 Markov decision process

The finite Markov decision process (MDP) is the idealized formal mathematical framework used to model RL problems and will be elaborated on below [47, 54]. The formal definition of an MDP is the tuple $< S, A, R, T >$. Here, S is defined as the finite set of environment states $\{s_1, ..., s_N\}$, the state space. Moreover, A is the finite set of agent actions $\{a_1, ..., a_N\}$, the action space. $T(s_{t+1}|s_t, a_t)$ is the value of the transition function $T(.)$ defined as the probability of transitioning from state s_t to s_{t+1} after performing action a_t. It is essentially a description of the environment. $R(s_t, a_t, s_{t+1})$ is the value of the reward function $R(.)$ which is the expected scalar reward for performing an action in a state and transitioning to the next state. For a system to be modelled as an MDP, the system needs to be fully observable and must comply to the *Markov property*. This means that the future states in a stochastic system solely depend on the the current state and not on previous ones. In other words, the current state embeds the information of all previous ones, which is mathematically defined as:

$$T(s_{t+1}|s_t, a_t) = P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, ..., s_{t-N}, a_{t-N}) = P(s_{t+1}|s_t, a_t). \tag{2-1}$$

An agent (controller) interacts with its environment (plant) by means of a sequence of actions (control inputs), states and rewards. The agent can observe the environment state ($s_t \in S$) at every discrete step, and then perform an action ($a_t \in A$). As a consequence the environment returns the subsequent state (s_{t+1}) according to the transition function and the immediate scalar reward ($r_t = R(s_t, a_t, s_{t+1})$) according to the reward function. This scheme is depict in Figure 2-1. The interaction between the agent and the environment takes place during an episode. The agent starts in an initial states and performs actions until the episode is over, which can be because a terminal state is reached or a certain time has elapsed.

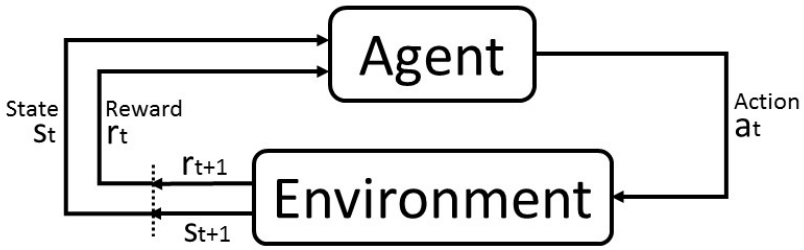


Figure 2-1: The agent-environment interaction in a MDP, adapted from [47]: the agent receives the current state s_t and corresponding reward r_t, where after it performs action $a_t = \pi(s_t)$, the environment then returns the successive state s_{t+1} and reward r_{t+1}.

An agent's goal is to perform actions that are rewarding over a longer period. The long term reward, also called the return, is defined as:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{i=0}^{N} (\gamma^i \cdot r_{t+i+1}), \tag{2-2}$$

where $\gamma \in [0, 1)$ is the discount factor, which indicates the decreasing importance of future reward and makes the sum finite. N is the time step at which the episode ends. It is also possible that an episode is infinitely long, which means $N = \infty$. Given the MDP tuple, a policy $\pi(a_t|s_t)$ can be determined. A policy describes the agent's behavioural strategy in its environment and represents the mapping from the state space to the preferred action that is to be taken in all these states. The goal of an agent is to find the optimal policy π^* that maximizes the return over each initial state s_0.

To be able to learn policies, value functions are introduced. The value of a state is the expected return that can be achieved when starting in state s while following the current policy π. The combined value of all states is the state-value function $V_\pi(s) = E[G_t|s_t = s]$, which is an indication of how well an agent behaves in the environment given the current state and following the current policy π. The action-value function, or Q-function, $Q_\pi(s, a) = E[G_t|s_t = s, a_t = a]$ is the combined value of each action that can be performed in each state following the current policy. Or in other words: the expected value of the return when starting in state s and taking action a at the first time step while following policy π.

The aim of RL algorithms is to find the value function and the corresponding policy that maximize the obtained reward. These are the optimal value function, $V^*(s)$ or $Q^*(s, a)$ and optimal policy, π^*. It is possible that multiple optimal policies exist, but all of them will share the same optimal value function, which is unique. Both optimal value functions must satisfy the Bellman optimality equation; this equation for the Q-function is:

$$\begin{aligned} Q^*(s, a) &= \max_\pi Q_\pi(s, a) && \forall s \in S, a \in A, \\ &= E\big[r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})\big]. \end{aligned} \tag{2-3}$$

The optimal policy π^* is the policy that selects an action with the largest Q-value possible for each state and is also called the greedy policy. When the optimal action-value function is known the optimal policy can be extracted as:

$$\pi(s) = \arg\max_a Q^*(s, a). \tag{2-4}$$

The Bellman optimality equation can theoretically be solved explicitly, but in practice this is rarely possible [47]. The dynamics of the system, i.e. the transition function (T) and the reward function (R) of a system must be accurately known. It is however often difficult and could be impossible, to determine T and R exactly [35]. When the dynamics is known exactly, every combination of initial states and action sequences must be examined on their properties. In the case of large state and action spaces, finding a solution might take several to an infinite amount of years. Another way of finding the optimal policy is by means of RL algorithms, which will be discussed in the next section.

2-1-2 Determining the policy

RL algorithms can be divided into model-based and model-free RL algorithms [47]. Model-based RL is also called dynamic programming. These algorithms thus require knowledge of the system dynamics. When it is very difficult or expensive to obtain a sufficiently accurate model, model-free RL algorithms are useful. These algorithms use only experience samples to determine a policy, which is obtained by interaction with the environment. The well-known model-free RL algorithms Q-learning is discussed in Section 2-1-3.

There are two main approaches for solving both dynamic programming and model-free RL problems: one strategy is based on finding the value function, and the other is based on direct policy search [2, 6]. In order to find the optimal value function, the value function is updated recursively through experience. There are two iterative methods for doing this, either by *policy iteration* or *value iteration*.

Policy iteration consists of two iterations steps, the *policy evaluation* and the *policy improvement* step. The policy evaluation step investigates the performance of the current policy by computing the value of the state-value function V_π or the Q-function Q_π. In the policy improvement step, an improved policy is computed based on the value of V_π. This is done by assessing all actions for all states with the aim of finding actions in certain states that perform better than the action that the current policy proposes.

In value iteration, the RL algorithm finds the optimal value function by iteratively updating the value function until convergence. In dynamic programming, this can be done by iterative use of the Bellman equation. With this optimal value function, the optimal policy can be derived. Usually, policy-iteration needs fewer steps before convergence, while value-iteration is computationally less complex [28].

When using a direct policy search method, one does not need to maintain a value function. The optimal policy is directly found by using optimization methods such as (among others) gradient descent or genetic algorithms. The optimization problem is, however, often non-differentiable and could be non-convex. When this is the case, global and gradient-free methods are preferred [6]. Policy search optimizes over each possible initial state, resulting in high computational complexity, usually much higher than value and policy iteration.

There are also RL algorithms that use both a value function and a policy search approach; these are called actor-critic methods [2]. These methods are able to profit from progress made in both research fields and can therefore combine the benefits of both methods.

We will focus on model-free RL algorithms that use the Q-function for finding the optimal policy. Two common methods for solving model-free RL problems are Monte Carlo algorithms and temporal-difference (TD) learning [47]. The main difference between the two is the way that the Q-value is updated. In Monte Carlo methods the Q-value of each visited state-action pair is updated at the end of an episode. In TD learning the Q-value of a state-action pair is updated directly after it is visited by the agent, thus after one time step. There are also techniques that are somewhere in between these two extremes, where the Q-value is updated after n times steps. These are called TD(λ) learning methods. $\lambda \in [0, 1]$ represents the number of steps before the Q-value is updated, where TD(0) is normal TD learning and TD(1) is a Monte Carlo method.

2-1-3 Q-learning

In this section the most well-known RL algorithm is discussed, namely Q-learning [52], which is a method that is based on TD learning. Q-learning is an off-policy algorithm, which means that it uses a different policy for the estimation of the Q-value, i.e. the target policy, then for the action-selection, i.e. the behaviour policy. On-policy methods also exist, where the target and the behaviour policy are the same. Q-learning is an off-policy and value-based algorithm. In the Q-learning algorithm, the Q-value is iteratively updated after every time step by:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t[r_{t+1} + \gamma \max_{a_{t+1}} Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)], \tag{2-5}$$

where $\alpha \in [0, 1]$ is the learning rate of the agent, that indicates how much the new Q-value is updated in accordance to the old Q-value and toward the Q-value update target. The learning rate is often time varying, decreasing with time. The term between the brackets of the equation is the temporal-difference (TD), the difference between the current estimate of the optimal Q-value ($Q_t(s_t, a_t)$) and the estimated target Q-value ($r_{t+1} + \gamma \max_{a_{t+1}} Q_t(s_{t+1}, a')$). Since Q-learning uses an estimate of the target Q-value to update its Q-value, it performs bootstrapping: updating an estimate, based on an estimate. Bootstrapping can cause the updated Q-value to be biased towards this estimate [47].

Exploration vs. exploitation

Q-learning is proven to converge with probability 1 [51, 47] when (1) all states, actions, and the Q-value are represented discretely, (2) all actions in all states are executed, and (3) a learning rate (α_t) is chosen that satisfies the conditions, $\sum_{t=1}^{\infty} \alpha_t = \infty$ and $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$.

The second condition can be satisfied by always allowing the agent to select a random action with non-zero probability during the learning process. This is the trade-off between extitexploration and *exploitation*, in other words between the random choice of an action and the greedy choice of an action, i.e. the action with the highest return according to the current policy. Two well-known strategies for this trade-off are the *ϵ-greedy* and the *Boltzmann exploration* strategy. The ϵ-greedy method works as follows: the agent chooses a random action with a probability of $\epsilon \in [0, 1]$ and a greedy action with probability $(1 - \epsilon)$. Conventionally ϵ decreases over time such that increasingly more greedy actions are chosen when learning progresses. Boltzmann exploration uses a probability distribution to not completely randomize the exploration phase. The probability of the agent choosing action a in state s_t is:

$$\pi(s_t, a) = \frac{e^{(Q_t(s_t,a)/T)}}{\sum_{i=1}^{N} e^{(Q_t(s_t,a^i)/T)}}, \tag{2-6}$$

where $T \geq 0$ is used as temperature parameter to denote the degrees of exploration. When $T = 0$ the agent only exploits, while when $T \to \infty$ the agent always chooses a random action. For all values between these two extremes the actions are ranked upon their probability of being chosen. Higher valued actions will have a higher probability of being chosen then lower valued actions.

Q-table

In basic Q-learning the Q-value estimate for each state-action pair is explicitly stored in a so-called Q-table. When a system becomes larger, the size of this table grows exponentially. A system with a lot of states and actions must therefore have an enormous Q-table. Even when storing such a table is possible, visiting all state-action pairs to fill the table becomes infeasible. Also, propagating the Q-values from a particular state to similar states is not possible. Q-learning in combination with a Q-table is therefore not a suitable technique for large scale systems such as in urban traffic control. To solve this problem, function approximation of the Q-function is needed, which will be discussed in the next section.

2-1-4 Function approximation

A big challenge that arises in RL problems is the so-called *curse of dimensionality*. In a larger environment the number of state-action pairs can become very large as its number increases exponentially with the number of states. The memory needed for the large Q-tables becomes very big, but more importantly the time it takes to learn the Q-value for all state-action pairs will become too large. It becomes computationally infeasible to visit all state-action pairs. Urban traffic systems often become quite big, especially when the number of intersections that ought to be studied, grows. It is therefore often inevitable to use a large state-space for control purposes. A solution for this limitation can be function approximation of the Q-function. The approximation is used to generalise the Q-function across state and action space. The approximated Q-function is a parameterized version of the real Q-function:

$$Q(s,a) \rightarrow \hat{Q}(s,a,\theta), \tag{2-7}$$

where θ indicates some weight vector, used to parameterize the Q-function. Instead of updating the Q-value for each state-action pair the (limited amount of) parameters can be updated. This significantly reduced the number of parameters to be updated. When the parameterized Q-value function is updated after receiving the reward of a certain state-action pair, this will also result in a better approximation for comparable state-action pairs. There is a variety of function approximators, of which two will be discussed: a general class of linear approximators, to introduce the concept of function approximation, and the non-linear artificial neural networks (ANNs), as this is the approximator that will be used within the research. Both can be combined with RL and when RL and an ANN are combined this is called deep RL.

Linear function approximators

The first and simplest class of function approximators is linear function approximators. Their simple characteristic makes them intuitive to understand and easy to implement [54]. Besides, they are often less computationally expensive than other classes of function approximators. A linear approximated Q-function is a parameterized function and could, in general, be described as:

$$\hat{Q}(s,a,\theta) = \sum_{j=0}^{N} \theta_j \phi_j(s,a), \tag{2-8}$$

where θ denotes the linear parameters and $\phi(\cdot)$ is some (non)linear function embedding the characteristics of the real function. This approximation is linear as the function is linear in the parameters. There are different methods for updating the weight vector θ, roughly classified by gradient-based and gradient-free methods. Gradient-based techniques are used most frequently, as they are simple and converge fast. Often stochastic gradient descent is used, where the vector is iteratively updated into the direction of the gradient of the error. The error could be measured as the mean squared error, resulting in an update of the weights as:

$$\begin{aligned}\theta_{t+1} &= \theta_t - \frac{1}{2}\alpha_t \nabla_\theta (Q(s_t, a_t) - \hat{Q}(s_t, a_t, \theta))^2, \\ &= \theta_t + \alpha_t (Q(s_t, a_t) - \hat{Q}(s_t, a_t, \theta)) \cdot \nabla_\theta \hat{Q}(s_t, a_t, \theta),\end{aligned} \tag{2-9}$$

where $\nabla_\theta f(\theta)$ is the partial derivative with respect to the weight vector θ, and α is the step size, potentially varying over time. A function can however be non-convex and/or non-differentiable. This is when gradient-free methods are often preferred. Examples of gradient-free optimization methods are genetic algorithms, simulated annealing, and cross-entropy optimization. These methods are often computationally heavy, but are capable of finding global minima in non-convex functions without the use of a gradient.

It would be naive to directly apply a linear approximator for the approximation of all value functions, as naturally not all functions can be described by a combination of linear functions. Besides, it is not always immediately clear which class of functions would perform best. To resolve this issue, we will discuss another class of function approximators below, namely non-linear function approximators.

Non-linear function approximation: Neural networks

Artificial neural networks (ANNs) [39, 49] are networks that are inspired on the neural networks present in the human or animal brain. They can be used as non-linear function approximators. They are often used, because of their flexible, scalable, and universal nature [19] (i.e their ability to approximate functions of any class). This makes them capable of approximating very complex and highly non-linear functions. An ANN is a network consisting of artificial neurons, or perceptrons, embedded in an input layer, a number of hidden layers and an output layer. There exist different types of ANN, which are all suitable for different purposes. The first is a feedforward ANN. In this network the layers are fully connected, meaning that each neuron in a layer is connected to all neurons in the next layer. It also means that no feedback loops are present. When there are feedback loops in the network, such a network is called a recurrent ANN, which are more similar to how the human brain works. A third kind of ANN is the convolutional neural network, which is a specific kind of feedforward neural network where the layers are not fully connected. These kind of CNN are designed to receive images as their input data, they are easier to train than conventional feedforward ANNs [27]. In this section we will study a normal feedforward ANN.

Perceptrons As said, an ANN is built up from perceptrons, which are linear classifiers. They are developed by Rosenblatt [42]. By itself, a perceptron can thus be used as a linear

classifier. A perceptron receives an input or multiple input values (x) and then calculates a temporary output value (z) by summing a bias b and the multiplication of the inputs by a weight vector w, as:

$$z = \sum_{j=0}^{N} w_j x_j + b. \tag{2-10}$$

The bias represents the threshold value separating certain classes. The calculated value z is fed as an input for a certain non-linear activation function ($f(\cdot)$) for computing the output (y). A schematic representation can be found in Figure 2-2.

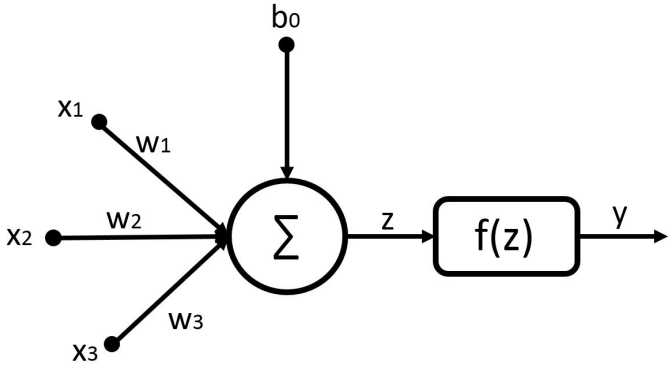


Figure 2-2: A single perception, adapted from [49], the perceptron receives input values x, these are multiplied by their related weights w and summed with bias b. This will result in temporary output z, which is the input value of activation function $f(.)$, resulting in output y.

Conventionally the perceptron was designed such that it receives binary inputs and returns a binary or signum (-1 if $z < 0$, 0 if $z = 0$ and 1 if $z > 0$) output indicating if the output z was above or below a certain threshold [39]. In this way the perceptron can classify the input vector, indicating to which class the vector belongs. A great disadvantage of this linear perceptron is that a small change in the inputs, weights or bias could flip the output of the perceptron completely, e.g. from 1 to 0. This might change the behaviour of a network of perceptrons in an undesired way. Another limitation is that these kinds of functions do not have useful derivatives, as its derivative is non-existing for $z = 0$ and zero for all other values. Proper differentiable functions are preferred to be able to use gradient-based methods for updating the weights.

To overcome this problem many different activation functions are conceivable, all mapping the input z to an output $y \in [0, 1]$. Common non-linear function used in neural networks are the rectified linear unit (ReLU) function ($f(z) = \max(0, z)$) or sigmoid functions such as the hyperbolic tangent ($f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$) or the logistic function ($f(z) = \frac{1}{1+e^{-z}}$) [27]. When using a sigmoid function as an activation function, the perceptron is often called a sigmoid neuron. Sigmoid functions are differentiable, which means that gradient-based methods can be used for updating the weight vectors. Even though ReLU functions are not differentiable in the origin, in practice gradient-based methods are often used in combination with this activation function. It is shown that hidden layers composed of ReLU typically train faster than the

ones composed of a sigmoid function [14]. In Figure 2-3 the signum, a sigmoid, and the ReLU function are illustrated.

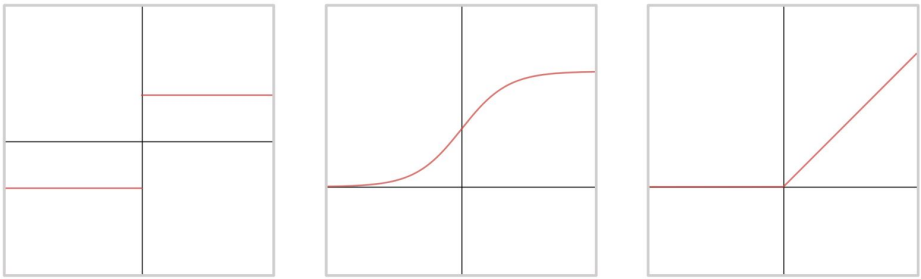


Figure 2-3: Three activation functions, from left to right: (1) the signum function (2) the logistic function, a sigmoid function, and (3) the rectified linear unit function.

In an ANN the connected perceptrons are often just called neurons. The feedforward ANN that is discussed, is also called a multilayer perceptron. The output that is determined by the neuron in an ANN is fed to the next layer. The schematic overview of the layout of feedforward ANN can be found in Figure 2-4. Just like in other function approximators, the parameters (i.e. the weights of the connections between neurons) need to be updated to obtain a good approximation of the Q-function. The goal is to find the optimal value for the weights and the bias terms, which is done during training. In feedforward ANNs, backpropagation is often used for updating the parameters.

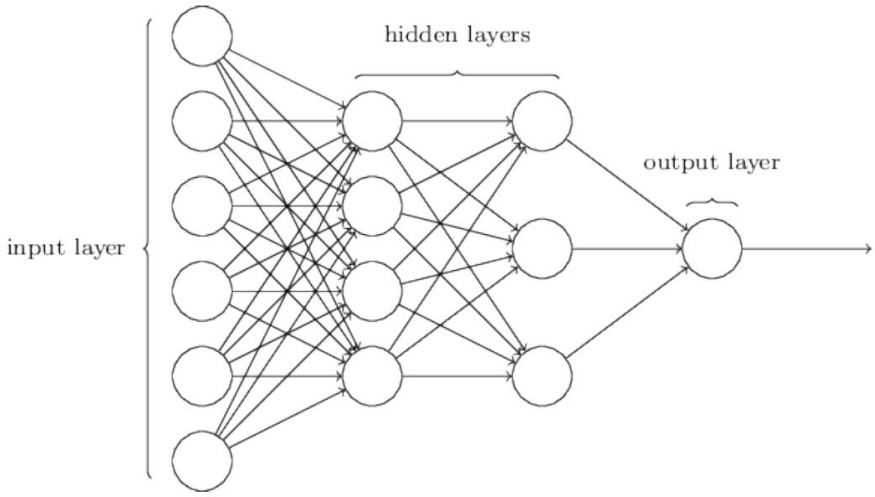


Figure 2-4: An example of a feedforward artificial neural network with two hidden layers [39], the neurons are fully connected.

Backpropagation Here we will describe the process of training with backpropagation [27]. First, all weights in the network are initialized to random values. Then a forward pass is performed where the output of all neurons is computed, in order to generate output data. The output data of the output layer is then compared to the expected output of the system. A

loss based on the error between the two is determined, e.g. by computing the mean squared error. Lastly, a backwards pass is performed. The partial derivative of each weight with respect to the loss function is calculated, in order to do so the chain rule of derivation needs to be applied, such as:

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial y_j}\frac{\partial y_j}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}} \tag{2-11}$$

With this gradient the weights of the neuron connection can be adjusted by simple gradient descent methods, such as in Equation 2-9. Choosing sensible initial values for the weight in the network could potentially speed up the learning process of the network significantly.

Limitations of ANN We have seen that ANN can be very good function approximators for their flexibility, scalability and their ability to approximate all function classes. There are however also some drawbacks of using an ANN as a function approximator. Their black-box nature makes it difficult or even impossible to understand why weights are updated in certain ways. On the contrary, extracting this kind of information from a linear approximator is feasible and can be very useful.

Also overfitting can be a result of applying ANNs. A neural network has a very large number of free variables to tune. A model with lots of free variables is able to fit most data set 'perfectly', with 'perfect' implicating that every data point is fitted precisely including noisy points or outliers. Overfitting on a training set can cause bad performance when the ANN is applied to a different test data set. There are many different strategies to prevent overfitting [39]; these are called regularization techniques. The first is early stopping, where the accuracy of a validation data set will be determined after each epoch, i.e. each update of the network weights. If this accuracy saturates, the training will be stopped. Secondly, increasing the size of the training set will reduce overfitting significantly. Unfortunately, a sufficient amount of data is not always available, as it could be expensive to acquire. Furthermore, it is possible to penalize the magnitude of the weight of the neural network in the cost function in a so-called regularization term. Small weights are preferred, and large weights are only used when they improve the cost function considerably. Other ways are artificially expanding the training data and dropouts, where some neurons in the neural network are temporally dropped out during training.

When ANN and RL are combined, instability can occur. Specifically, when function approximation, bootstrapping (e.g. in TD methods and dynamic programming), and off-policy training are combined, there is a great danger of instability [47]. This is exactly the case when combining Q-learning and an ANN, which can cause the Q-value to diverge from the true value. In Q-learning both the behaviour and the estimated target policy are used for updating the Q-value. When the Q-value is updated, the Q-target is updated likewise, changing the target at every update. The network is in this manner always updated towards a moving target. This can cause instability during the learning process and overestimation of the Q-value. Correlation between consecutive data points can also be a cause of instability. Two techniques to prevent instability are target network and experience replay.

2-1-5 Deep Q-network

In the previous section we have seen that ANNs are very powerful function approximators, but also that instability can occur when combining an ANN with Q-learning. To overcome these challenges Mnih et al. [36, 37] introduce the novel Deep Q-Network (DQN), in the study the agent is successfully trained to play Atari games. An ANN is combined with a variant of Q-learning and two key techniques: experience replay and a target network that periodically updates. DQN uses a discrete action, but can use a continuous state space.

Target network

As seen in Section 2-1-3, off-policy methods like Q-learning use a different target policy and behaviour policy. When the target policy updates at every update of the behaviour policy this results in a network that is constantly updated towards a moving target, which can cause instability. To deal with this moving target a second neural network is introduced [36]. The Q-network is duplicated and the second network is called the target network, which is used for generating the Q-function targets. During training the Q-value is iteratively updated towards the target network after each action. The target network only updates every few iterations by copying the weights of the most recent network, which ensures a stationary target at least for several updates. The Q-value update becomes as follows:

$$Q_{t+1}(s_t, a_t; \theta_t) = Q_t(s_t, a_t; \theta_t) + \alpha_t[\underbrace{r_{t+1} + \gamma \max_{a'} \hat{Q}_t(s_{t+1}, a'; \theta'_t)}_{\text{target network}} - \underbrace{Q_t(s_t, a_t; \theta_t)}_{\text{online network}}], \tag{2-12}$$

where the target network is thus not updated at every Q-value update of the online network. This technique will remove the moving target and reduce correlation between the Q-network and the target network, which will result in stability of the network.

Experience replay

Experience replay, first introduced in [30], is the act of reusing past experience for training an agent. Each experience, consisting of (s_t,a_t,r_t,s_{t+1}), is saved in a data set. For training, the agent can randomly extract these samples as if it is experiencing again what it has experienced before. These samples are thus used for achieving the Q-function updates. The use of experience replay has several advantages [37]. As experience can be reused, it can likely be used in different weight updates. The method is thus data efficient. Secondly, consecutive experience samples are strongly correlated which can cause an overestimation of the Q-value towards this correlation. Random selection of the data ensures that data always tends to be independently and identically distributed (i.i.d.), in contrast to consecutive experience. Experience replay thus breaks correlation and overestimation. Lastly, in on-policy methods the weight of the network influences the subsequent experience. This might lead the network to oscillate between local minima. Experience replay smooths out learning and avoids oscillation of parameters. Experience replay can be extended to prioritized experience replay [43] where better performing or rare samples are prioritized, and thus have a higher probability of being chosen from the memory set. This extension results in even more efficient learning of the agent.

2-1-6 Conclusions

In this section we have been acquainted with reinforcement learning (RL). An RL problem can be modelled as a Markov Decision Process consisting of states, actions, rewards and a transition function describing transition probability from one state to the next after performing an action. A policy describes an agent's behaviour in this environment, and a value function describes the value of this particular policy. The optimal policy and value function can be iteratively determined through different strategies. The transition function of a system is often unknown, this is why model-free RL algorithms are introduced. A well-known model-free RL algorithm is Q-learning. Conventional Q-learning combined with a Q-table has shown to be computationally infeasible for large state and action spaces. This problem is called the curse of dimensionality. To still allow the use of RL, function approximation is needed to compute a generalized approximate Q-value function. Two classes of function approximators are discussed: the most straightforward one, linear function approximators, and the non-linear function approximator, artificial neural networks (ANN). ANNs have shown to be very good function approximators for their flexibility, scalability and their ability to approximate all function classes. However, instability can occur when combining an ANN with an RL algorithm. To overcome this limitation a target network and experience replay can be added to a deep RL algorithm. This combination is called deep Q-network (DQN), which is a very effective and popular deep RL algorithm.

2-2 Urban traffic signal control

Traffic congestion in urban areas is a big problem in modern society, making it an interesting and essential topic for research. There are several ways of improving the throughput of vehicles in a traffic system; three of them are: (1) the existing infrastructure could be expanded or improved, (2) the use of alternative modes such as public transportation could be encouraged to reduce the current load on the system, or (3) more efficient control strategies can be employed on existing traffic infrastructures. Expending traffic infrastructure in an urban setting is often not a solution, as there is limited space available for new infrastructure. The improvement of the existing infrastructure is an option, but a costly one. Improving control strategies is economically the most favourable, and is, therefore, most commonly employed. In the setting of urban traffic control the use of signal control, i.e. controlling the traffic light at an intersection, is most common. At present, the traffic lights are mainly controlled by a fixed-time controller or by traffic-responsive controllers that use real-time data to obtain their control solution, through e.g. the use of sensors in the road actuated by vehicles [10]. In the following section we will introduce both fixed-time controllers and traffic-responsive controllers. Thereafter we will focus on the use of MPC for traffic signal control.

2-2-1 Traditional urban traffic signal control

Fixed-time control

The first and most basic urban traffic signal controllers are fixed-time controllers. The control strategies are derived offline based on the characteristics of the specific intersections, such

as time of the day and historical data of the demand of that intersection. Webster [53] developed one of the first fixed-time traffic signal controllers. This controller aims to minimize the average delay per vehicle for a single intersection. However, the control of an isolated intersection is not an efficient control solution for urban traffic control, as it does not allow for an optimal control strategy of a larger network. An optimal control sequence in one intersection could potentially lead to congestion somewhere else in the network. When two signals at successive intersections are not aligned with each other's phase sequence, this can cause cross-blocking. This is where vehicles have to wait in front of a green light because the downstream street is completely occupied.

To achieve a global optimum for a whole network, control strategies for multiple intersections are proposed. Two well-known and often used fixed-time controllers, which can be used in a system with multiple intersections, are MAXBAND [34] and TRANSYT [29]. MAXBAND aims to optimize the so-called green wave on a main road (or arterial). The strategy consists of establishing the optimal offset of signals at several consecutive intersections to maximize the bandwidth of the cars, i.e. the time frame in which the first and last vehicle can pass without stopping. TRANSYT minimizes the sum of the average queues in an area and also allows for offset coordination. The fixed-time controllers' dependence on historical data, in comparison to real-time data, is their biggest disadvantage [40]. The historical data presents a very generalized representation of the real system. For example, the expected (average) demand and turn ratio are never identical to the encountered values on a specific day. Furthermore, these controllers cannot account for sudden changes in the traffic situation, e.g. an accident on the road. These drawbacks reduce the performance of the fixed-time controller. It would therefore be more efficient to use controllers that can respond to traffic situations in real-time. It must be noted that these methods are often more expensive, as systems need to be installed that are able to real-time observe traffic behaviour and control the signals at each intersection.

Traffic-responsive control

Traffic-responsive controllers use online control techniques based on real-time data. The controllers are able to adapt their control strategy in accordance with changes measured in the traffic system. SCOOT [41] is a widely used control strategy aiming for network-wide coordination control. The approach is known as the traffic-responsive version of TRANSYT and has the same control objective. Another well-known control algorithm is SCAT [46]. SCAT differs from SCOOT, because it has a distributed and hierarchical control approach. Data is collected locally, and control is performed on subsystems, with the size of one to ten intersections.

Other responsive methods are model-based methods such as OPAC [9], PRODYN [17], RHODES [44], and CRONOS [5]. These methods solve an online optimization problem based on a prediction model. They all use a rolling horizon scheme; the optimization problem is solved for a certain time horizon N_p, but the input is only applied to a shorter period. Hereafter new measurements of the traffic state are performed and the optimization problem is solved again for time horizon N_p. The biggest challenge for these methods is the application on larger systems (i.e. more than one intersection), where the methods are not real-time applicable anymore [40]. This is due to the complex solving algorithms that are used by these

methods, which are all based on dynamic programming. In dynamic programming, the problem is split into multiple (suboptimal) problems, which are solved recursively. Furthermore, the prediction models used in all these methods are limited by their prediction horizon, which is very short [32]. This reduces the performance of the methods, as the controllers can not anticipate enough to ensure an optimal control sequence over a longer period, i.e. the controllers are not myopic. To overcome this limitation, other models must be used. Model predictive control (MPC) is also a model-based traffic-responsive control method. It uses improved models compared to the traditional model-based methods talked about above. Besides, the optimization problem is solved without the use of dynamic programming algorithms.

Besides fixed-time and traffic-responsive controllers, adaptive traffic controllers exist for urban traffic signal control. Reinforcement learning (RL) algorithms can be used to obtain adaptive controllers for urban traffic control [1]. These controllers are self-learning and data-driven, solely basing their control input on empirical data.

2-2-2 Model predictive control

Model predictive control (MPC) [7] is an online control technique that uses a dynamic model that predicts system behaviour to determine the optimal control input for a system. Just like the other model-based control approaches, MPC consist of three steps: a prediction model, an optimization problem, and a rolling horizon scheme.

The rolling horizon means that at each control step t the predicted optimal control input is redetermined. In each control step the MPC controller solves an optimization problem based on the model of system over a certain prediction horizon N_{p}. The first control input is implemented. The controller receives new information about the updated states of the system and external disturbances, after which the horizon is rolled forward by one time step, such that $t+1$ is the new current control step and the optimization problem is solved again over the shifted horizon N_{p}. Through the rolling horizon the control loop of the MPC controller is closed. The feedback makes the MPC controller robust to uncertainties and disturbances in the system. A representation of the control scheme is shown below in Figure 2-5.

MPC is already widely studied as a control technique for urban traffic signal control [55] and has shown great results. MPC is a very promising control technique for traffic signal control for different reasons. The objective for controlling an urban traffic system may vary in different situations. With MPC it is possible to define different objectives and to optimize and change them according to different situations. Examples of objectives are the minimization of traffic delay, emissions or unsafe situations. Moreover, it is possible to control multiple or combined control objectives simultaneously. Furthermore, it is easy to change the objective of the problem and the model can be easily interchanged with an updated one, must this be necessary. Additionally, a main characteristic of MPC is that it is able to handle constraints directly. A big advantage of an MPC scheme is thus its ability to apply not only input constraints, but also stat constraints. Furthermore, an MPC controller is not only capably of predicting a long term control strategy, but the feedback scheme (i.e. the rolling horizon) also assures that the MPC is able to respond robustly to external disturbances or uncertainties, e.g. in the demand. The presence of disturbances and uncertainties, however, results in a suboptimal performance of an MPC controller. The controller is not intrinsically adaptive,

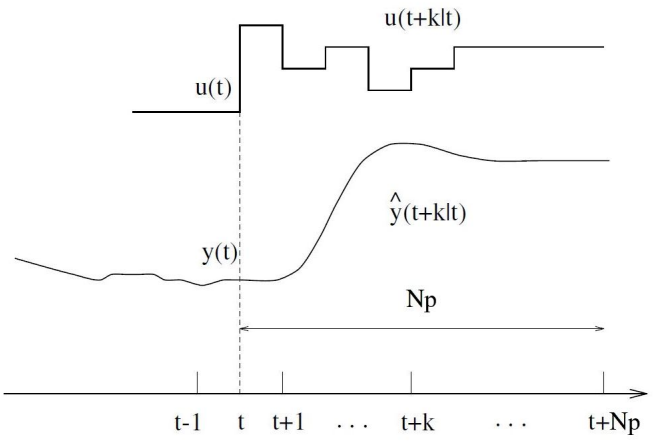


Figure 2-5: The model predictive control scheme [7], at control step t the optimal control input u is determined for control horizon N_p by optimizing the predicted output $\hat{y}$. The first input $u(t)$ is implemented and the horizon is shifted by one time step towards control step $t+1$.

and thus not able to change its control law according to changing circumstances. An MPC controller is thus robust, but not adaptive.

Nonetheless, with the application of an MPC controller the issue of real-time implementation arises. When an MPC controller is applied to a real world system, the optimization problem has to be solved online within the sampling time, before the next control input has to be applied. As systems get larger and more complex the computation time of the the controller also increases. This makes it often impossible to solve the problem within the sampling time. In literature, several approaches are already taken in the aim of making the MPC controller real-time implementable, such as changing the control structure and choosing an appropriate model. These approaches can be found in general articles or in the context of urban traffic signal control. In the next section we will briefly touch upon the topic of choosing a model and then the prediction model that will be used in this thesis, the BLX model, will be explained.

2-3 Traffic modelling

The model used in the control strategy plays an important role in the computational complexity of the problem. For the choice of a model there is always a trade-off between accuracy of a model and the computational complexity. The choice for a traffic model can be roughly divided into two: microscopic and macroscopic models [25]. Microscopic models describe the individual behaviour of traffic users, and thus being the most accurate of the two. Macroscopic models on the other hand describe the collective flow of vehicles while individual behaviour is not explicitly represented. Examples of the traffic flow characteristics are the vehicle density of the average speed. Macroscopic models have the advantage to microscopic models that the computation time does not increase with an increase of traffic demand, as the amount of variables does not depend on the number of vehicles on the road. For this reason macroscopic models, instead of microscopic models, are often chosen as prediction model of an MPC controller. Within the macroscopic models different choices can be made as well to find a good

balance between accuracy and efficiency. A model with a close resemblance to real world behaviour results in better performance. These kind of models, however, contain more features or non-linearities, leading to higher computational complexity.

2-3-1 BLX model

In this thesis the BLX model is used as a prediction model for MPC. This macroscopic link model was first introduced by van der Berg et al. [50] and later extended by Lin and Xi [33], thus the BLX-model. Besides, the work of Lin [31] is used as a reference for the model in this thesis.

In the model the set of junctions is defined by J and the set of links as L. We consider link $(u, d) \in L$ as in Figure 2-6, where $u \in J$ is the upstream junction and $d \in J$ the downstream junction. $I_{u,d} \subset J$ is the set of incoming nodes for link (u, d) and $O_{u,d} \subset J$ the set of outgoing nodes. The simulation time step (T_s) of the model is set to 1 s. Each equation is updated for each simulation period $[k \cdot T_s, (k+1) \cdot T_s]$.

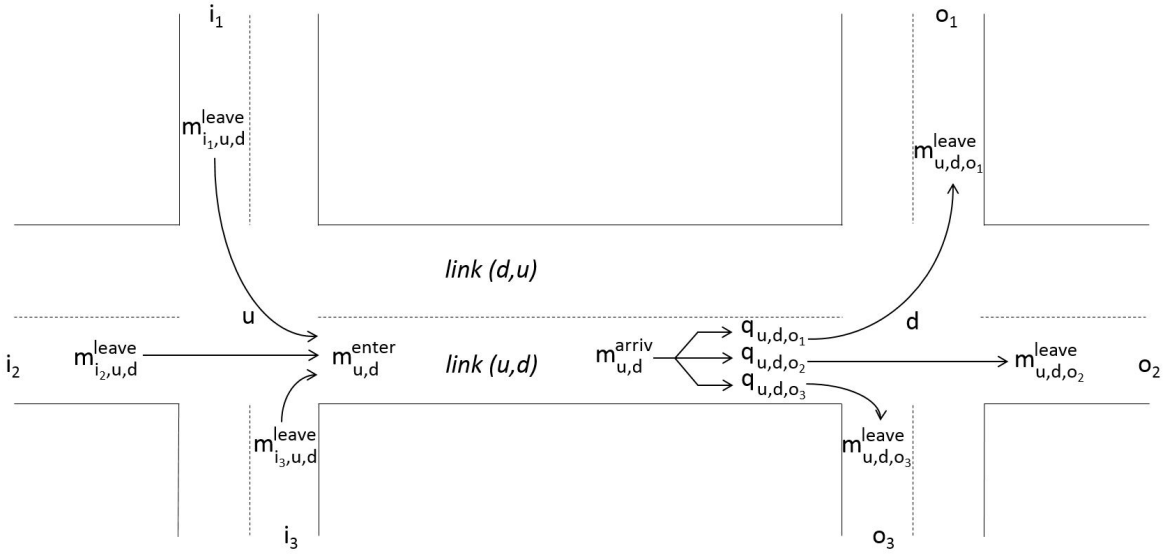


Figure 2-6: The graphical representation of two intersections connected by a link, with all variables relevant to the BLX model.

Whenever a traffic light for link (u, d) is green in direction $o \in O_{u,d}$, the number of vehicles that leave the lane within one time step is the minimum of three values: (1) the saturated flow rate leaving the lane $\mu_{u,d,o}$, (2) the available storage at the destination link $S_{d,o}(k)$, and (3) the number of vehicles in the queue on the origin lane $(q_{u,d,o}(k) + m^{\text{arriv}}_{u,d,o}(k))$. This last value consists of the arriving vehicles at the queue within one time step $m^{\text{arriv}}_{u,d,o}(k)$ and the already existing queue length $q_{u,d,o}(k)$. The number of vehicles leaving link (u, d) towards each destination $o \in O_{u,d}$ is therefore given by:

$$m^{\text{leave}}_{u,d,o}(k) = \begin{cases} 0 & \text{if } g_{u,d,o}(k) = 0, \\ \max(0, \min(\mu_{u,d,o} \cdot T_s, S_{d,o}(k), q_{u,d,o}(k) + m^{\text{arriv}}_{u,d,o}(k)) & \text{if } g_{u,d,o}(k) = 1, \end{cases} \tag{2-13}$$

where $g_{u,d,o}(k)$ is the controllable input of the model. $g_{u,d,o}(k) = 1$ indicates that the traffic light is green for the vehicles on link (u, d) in direction $o \in O_{u,d}$. When $g_{u,d,o}(k) = 0$, this would indicate a red traffic light in this direction.

The traffic that arrives at the queue on link (u, d) is determined by the leaving traffic from origin destinations $i \in I_{u,d}$ with a small delay. This can be mathematically described by:

$$\begin{aligned} m_{u,d}^{\text{arriv}}(k) = &\frac{T_s - \gamma(k)}{T_s} \cdot \sum_{(i,u)\in I_{u,d}} m_{i,u,d}^{\text{leave}}(k - \tau(k)) + \\ &\frac{\gamma(k)}{T_s} \cdot \sum_{(i,u)\in I_{u,d}} m_{i,u,d}^{\text{leave}}(k - \tau(k) - 1). \end{aligned} \tag{2-14}$$

The parameters to describe the delay (τ and γ) are:

$$\begin{aligned} \tau(k) &= \text{floor}\left\{ \frac{S_{u,d}(k) \cdot l_{\text{veh}}}{N_{u,d}^{\text{lane}} \cdot v_{u,d}^{\text{free}} \cdot T_s} \right\}, \\ \gamma(k) &= \text{rem}\left\{ \frac{S_{u,d}(k) \cdot l_{\text{veh}}}{N_{u,d}^{\text{lane}} \cdot v_{u,d}^{\text{free}}}, T_s \right\}, \end{aligned} \tag{2-15}$$

where l_{veh} is the average length of a vehicle, $N_{u,d}^{\text{lane}}$ is the number of lanes on the link, $v_{u,d}^{\text{free}}$ the free-flow vehicle speed on the link, and $S_{u,d}(k)$ the available storage on link (u, d).

The number of vehicles that arrive at the end of the queue on the destination link is divided into three separate arrival rates based on the direction each vehicle is going. The total number of arriving vehicles on the edge is multiplied by a turn ratio describing the fraction of vehicles going a certain direction. Thus three separate lanes are formed on three separate lanes for each subsequent destination link. The section of vehicles arriving at the queue on link (u, d), turning to $o \in O_{u,d}$ is:

$$m_{u,d,o}^{\text{arriv}}(k) = \beta_{u,d,o}(k) \cdot m_{u,d}^{\text{arriv}}(k), \tag{2-16}$$

where $\beta_{u,d,o}(k)$ is the turn ratio of the specific lane. The turn ratio does not have to be time dependent as described here. The length of the queue on link (u,d) going to destination $o \in O_{u,d}$ is given by the old queue length plus the arriving traffic minus the leaving traffic:

$$q_{u,d,o}(k + 1) = q_{u,d,o}(k) + m_{u,d,o}^{\text{arriv}}(k) - m_{u,d,o}^{\text{leave}}(k). \tag{2-17}$$

The separate queues on each lane can be added to obtain the total queue length on link (u, d):

$$q_{u,d}(k) = \sum_{o\in O_{u,d}} q_{u,d,o}(k). \tag{2-18}$$

The number of vehicles present on link (u, d) is also dependent on the number of vehicles leaving and entering the link, and is given by the old number of vehicles on the link plus the

vehicles leaving the origin $i \in I_{u,d}$ from link (u, d) and minus the leaving vehicles to source $o \in O_{u,d}$:

$$n_{u,d}(k+1) = n_{u,d}(k) + \sum_{i \in I_{u,d}} m^{\text{leave}}_{i,u,d}(k) - \sum_{o \in O_{u,d}} m^{\text{leave}}_{u,d,o}(k). \tag{2-19}$$

The available storage on link (u, d) is essentially the initial capacity $C_{u,d}$ minus the number of vehicles on the lane, but is also given by the old capacity plus the leaving vehicles minus the vehicles leaving the origin $i \in I_{u,d}$, and thus entering link (u, d):

$$\begin{aligned} S_{u,d}(k+1) &= S_{u,d}(k) - \sum_{i \in I_{u,d}} m^{\text{leave}}_{i,u,d}(k) + \sum_{o \in O_{u,d}} m^{\text{leave}}_{u,d,o}(k), \\ &= C_{u,d} - n_{u,d}(k+1). \end{aligned} \tag{2-20}$$

Based on these equations a model for a larger urban traffic network can be established by defining the different characteristics of each link and the topology of the network.

2-4 Model-reference adaptive control

System uncertainties can affect the performance of a controller on a system. Systems can have highly uncertain characteristics, and when systems are very complex this can lead to unmodeled system behaviour. There are also systems with characteristics that change over time, e.g. an aircraft that burn fuel while flying and hence becomes lighter. In these kind of circumstances adaptive controllers can be valuable to counteract the negative effects of the uncertainties on the controller performance. An adaptive controller is able to estimate system uncertainties online and adjust the control law accordingly. An adaptive controller and a robust controller are different, as a robust controller determines its control law such that it can overcome the worst assumed uncertainty, making the control law often conservative [26]. The robust controller needs information about the uncertainties bound before design, while an adaptive one does not.

One adaptive control method is model-reference adaptive control (MRAC) [38]. In short an MRAC system is an uncertain plant controlled by a controller that is tuned by an adaptive law which receives the error between the states of the plant and the states of the reference model, which portraits the desired behaviour of the plant. The MRAC scheme is schematically shown below in Figure 2-7.

The MRAC system thus consist of four part: an uncertain plant, a controller, a reference model and an adaptive law. The uncertain plant can have different types of uncertainties, such as structured and unstructured uncertainties, and unmodeled dynamics. The reference model specifies the desired output behaviour of the uncertain plant. It receives a reference signal as its input and its output is a model-reference signal, which represent the desired output response of the uncertain plant.

The controller is used to control the uncertain plant to match the output of the plant with the output of the reference model. When the plant is not uncertain, the plant is minimum phase, and the plants parameters are known exactly, the controller can be designed such that the transfer function of closed-loop plant matches the one of the reference model [20]. As the

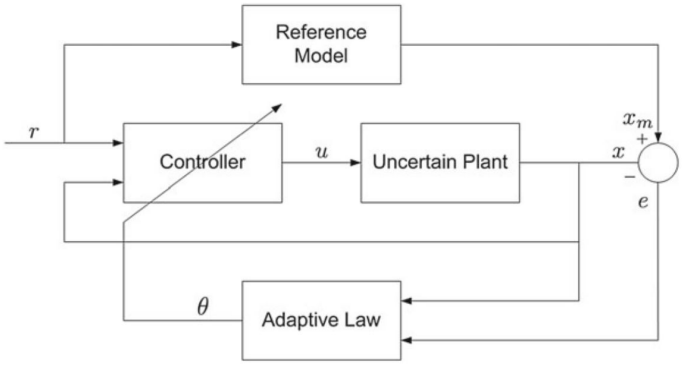


Figure 2-7: The model-reference adaptive control scheme [38], the uncertain plant is controlled by a controller that is tuned by an adaptive law which receives the error between the states of the plant and the states of the reference model.

plant is uncertain and its parameters are often unknown the MRAC scheme is used where the controller is adapted by the adaptive law. The adaptive law is essentially an online parameter estimator; it estimates parameters in each control loop that can tune the control law of the controller. The way that the adaptive law influences the controller can typically be done in two ways [20, 38]: by direct or indirect adaptive control, although it is possible to combine the two methods to get a hybrid direct/indirect adaptive control scheme. In the direct, or implicit, approach, the adaptive law directly estimates parameters present in the control law. Such a control law can be expressed as:

$$u = K_x \cdot x + K_r \cdot r, \tag{2-21}$$

where K_x and K_r are gains that can be directly adjusted by the adaptive law. In indirect, or explicit, adaptive control the adaptive law first estimates the parameters of the plant, which are then used to determine the controller parameters indirectly. The method is explicit as it directly estimates the plant parameters, while the direct method implicitly estimates the plant model by directly estimating the controller parameters. The control law for an indirect adaptive approach can be generalized as:

$$u = K_x(\theta) \cdot x + K_r(\theta) \cdot r, \tag{2-22}$$

where K_x and K_r are still the controller gains or parameters and θ are the plant parameters estimated online by the adaptive law. As an input the adaptive law receives the error between the reference model and the uncertain plant ($e = x_m - x$).

The largest advantage of an MRAC scheme is the application of the reference model. In [48] it is already argued that finding the reference trajectory when solving a tracking problem is the most important part of the problem. It is not always trivial what the exact trajectory should be. Therefore the determination of the reference trajectory is a problem in itself and can be a challenging one. The reference model solves this problem as it specifies the desired output behaviour and thus creating the desired output trajectory of the plant.

2-5 Conclusions

In this chapter we have explained the concept of reinforcement learning, including the use of artificial neural networks as a function approximator. Furthermore, multiple control strategies for urban signal control are discussed, where we have seen that traffic-responsive controllers outperform fixed-time controllers in terms of performance, especially in areas with more traffic. The earliest model-based controllers are able to perform well through their predictive nature, but are unable to control systems that are larger than a single intersection, because they are dependent on dynamic programming approaches for determining the control input.

MPC is a popular control strategy for urban traffic signal control. It has the advantage that it is easy to change the prediction model or the objective function to obtain a different result. Its rolling horizon scheme makes the controller robust to disturbances and uncertainties, although the controller performance will decrease by these uncertainties. Besides, input and output constraints can be included in the design of an MPC controller. However, there is one big disadvantage when using an MPC controller for urban traffic control. Finding a solution to the optimization problem can be computational very complex, making real-world implementation a challenge. The macroscopic BLX model is introduced which will be used as the prediction model for MPC control in the rest of the thesis.

Model-reference adaptive control is an effective adaptive control strategy. It will be used as a guideline for the design of the novel model-reference RL controller that will be introduced in the next chapter.

Chapter 3

Model-reference reinforcement learning framework

This chapter presents the model-reference control framework that combines MPC and RL for the control of signals in an urban traffic network. Section 3-1 describes the main motivation and inspiration of the model-reference RL framework. Then, Section 3-2 formalises the design of the control framework, where the MPC and RL components are discussed separately. Finally, Section 3-3 concludes the chapter.

3-1 Motivation

In the previous chapter, we have discussed model predictive control (MPC) and reinforcement learning (RL) as well as their limitations and advantages. We will now summarize these findings and discuss the differences between the two control methods and how they could (potentially) benefit from each other.

MPC needs a prediction model and thus is a model-based control method. The prediction ability and the performance of the MPC controller could be limited when an accurate prediction model is not available. More accurate models increase the computational complexity of the controller, and real-time feasibility is often a challenge during the application of MPC controllers. Nevertheless, MPC controllers are able to handle input and output constraints. They have advanced stability and feasibility theory and are proven to be robust through their rolling horizon scheme. However, these controllers are not adaptive and therefore perform suboptimally in the presence of uncertainties.

RL algorithms on the other hand can be used to obtain an adaptive controller with a low online computational complexity. A large group of RL algorithms is model-free. For these model-free methods, no convex optimization problem is needed for efficiently finding an optimal solution. RL algorithms are therefore able to obtain effective control policies for very complex, nonlinear and nontrivial systems. However, RL techniques do have a high offline computational

effort as the agent needs to pursue a training process. Besides, most RL techniques do not have any stability and feasibility guarantees. While in MPC these guarantees are established theoretically, these statements for reinforcement learning controllers are made after control design, analytically or by simulation [18]. This especially becomes an issue when dealing with safety-critical systems. RL controller can also only meet input constraints and no output constraints. Although output constraints can be met indirectly by accurately composing a reward function that contains penalties for constraint violation [15].

In Table 3-1 the advantages and limitations of MPC and model-free RL are summarized. It is clear to see that the two methods exceptionally complement each other: where the one lacks on certain performance criteria, the other method excels. In the rest of the chapter we will further elaborate on a framework to combine these two methods and is potentially able to take advantage of these findings.

Table 3-1: Comparison of RL and MPC characteristics

	MPC	**RL**
Model needed	Yes	No
Stability guarantee	Yes	No
Feasibility guarantee	Yes	No
Robustness guarantee	Yes	No
Constraints handling	Yes	No
Adaptive	No	Yes
Online computational complexity	High	Low
Offline computation time	Low	High

3-1-1 Model-reference reinforcement learning control

Zhang, Pan and Reppa [56] have designed a novel control framework that combines model-reference control, as explained in Section 2-4, with model-free deep RL for the control of autonomous surface vehicles. In the controller, a baseline control law of a conventional controller is employed to both a nominal model and the real system. The output of the nominal system defines the optimal performance of the vehicle. The output of the nominal model and the output of the real vehicle are compared, and its error is the input of an RL-based controller. The RL controller is able to compensate this error through its learned control law. The block diagram for the control framework is shown in Figure 3-1.

There are several advantages to the proposed control design. Conventional model-free RL-based controllers are able to adapt to great uncertainties and disturbances. Through its adaptive and model-free nature, it is able to react to changes in the system. However, closed-loop stability is not guaranteed when applying an RL control approach. For the model-based conventional baseline controller used in the research, it is, on the other hand, achievable to determine a control law that does guarantee closed-loop stability. This control law may however be very conservative and is not able to handle uncertainties and disturbances very well. This is why combining the two controllers is very valuable. The baseline controller stabilizes the system, while the RL controller compensates for all uncertainties and disturbances. The application of a baseline control law before the RL control law also increases the sample

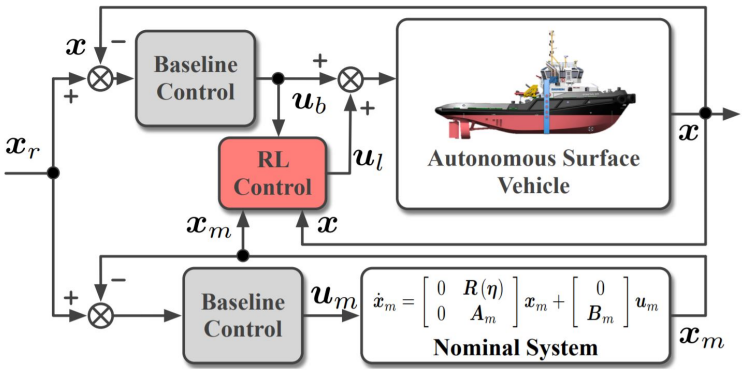


Figure 3-1: Model-reference RL control block diagram for the control of autonomous surface vehicles [56].

efficiency of the training of the deep RL network. The research shows in simulation that the individual RL controller is unstable and the individual baseline controller has a very low performance. However, the combined model-reference RL control law is able to follow the reference trajectory precisely.

The control scheme described above can be classified as a model-reference adaptive control (MRAC) scheme as discussed in section 2-4. A clear difference between conventional MRAC and the framework from [56] is the adaptive law used to adapt the controller. There are in fact two big differences in this adaptive law, namely: (1) The RL control used as an adaptive law in the framework is a model-free method. In conventional MRAC the adaptive law used is model-based as it is composed based on the controller design and the plant model. (2) In conventional MRAC the adaptive law is used to directly or indirectly tune parameters in the control law of the nominal controller. It could be said that the adaptive law is an online parameter estimator. In the framework from Figure 3-1 the RL controller is not used to estimate parameters and tune the baseline controller. The control law of the RL controller is merely added to the control law of the baseline controller to compensate for uncertainties that arise in the system. The RL controller tunes the input of the baseline controller, but not the controller itself.

In the paper, a soft actor-critic RL algorithm is used with a neural network as its function approximator. Note that the baseline controller used in the control scheme can be chosen arbitrarily. As long as the baseline controller is able to stabilize the plant it is a suitable controller.

3-1-2 Urban traffic signal control as a regulation problem

When considering control of traffic signals in an urban traffic system, one does not automatically consider a tracking problem. In this section, we will elaborate on different circumstances in which a regulation problem would be very useful.

In urban traffic there are different kinds of road users and corresponding modes to travel with. When a multi-modal traffic system is considered, multiple objectives arise. One very important and necessary mode is public transportation. Most car users would benefit from

reducing their travel time. However, this is not the most important objective of public transportation vehicles. Public vehicles need to obey their schedule. Their objective is to be exactly in time, as being early can also be harmful if a vehicle can not wait at a stop. Some vehicles (e.g. busses) participate in the same traffic system as regular cars, which makes their travel time unpredictable. Some research focuses on regulating the arrival time of busses by controlling traffic signals. Bhouri et al. [4] do this by designing a multi-agent control approach where a traffic signal agent takes priority of passing busses into consideration. Kachroudi and Mammar [23] have designed a multi-objective MPC approach that minimized delay while optimizing the punctuality of the public transportation in the network.

The management of public transportation through the regulation of traffic lights can be combined with all sorts of objectives: e.g. the maximization of throughput, the minimization of traffic emission, the regulation of noise pollution in a neighbourhood, or the priority of emergency vehicles. These combined objectives can be used to form a reference signal in the model-reference RL control framework that is introduced in the previous section. In Figure 3-2 the general framework is shown. The reference signal is obtained by the nominal traffic model based on one or multiple objectives and a suitable baseline controller.

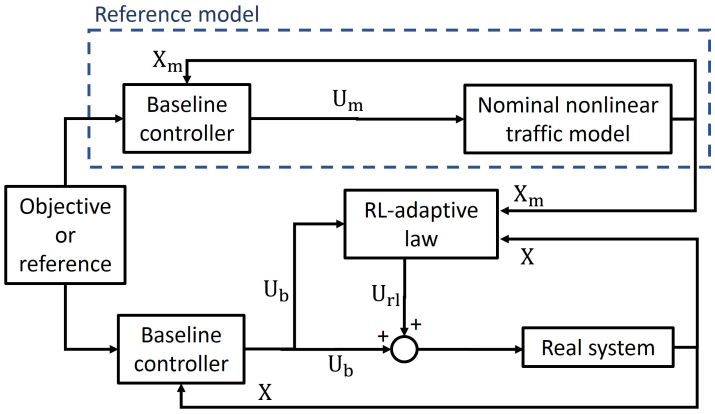


Figure 3-2: The general model-reference RL adaptive control framework for urban traffic signal control.

3-1-3 Combining MPC and RL in the model-reference adaptive control scheme

In the previous section, the general model-reference RL adaptive control framework for urban traffic signal control is described. Besides, we have seen that MPC and RL are two control methods that complement each other's limitations remarkably. These two findings give rise to the idea of using MPC as the baseline control input within the framework.

In Figure 3-3 the combined MPC and RL for the MRAC scheme is illustrated. This framework can potentially mitigate the drawbacks of both MPC and RL control. The adaptive framework may be able to reduce the negative effect that uncertainties have on the performance of an MPC controller. The addition of an adaptive law in the form of an RL controller may also reduce the computational complexity in comparison to a conventional MPC controller because an RL algorithm can operate with a very low online computational effort. One could accept a reduction in performance of the MPC model, e.g. by premature termination of the

optimization process or the choice of a less accurate model. The adaptive law is potentially able to compensate for the performance loss. Note that the reduction of computation time of the framework compared to conventional MPC control is not investigated in this thesis. The improvement of performance under the presence of uncertainties will potentially be the largest advantage of the application of the framework in traffic signal control compared to a normal MPC controller in a traffic signal control setting.

As regards the possible improvements of the framework compared to a conventional RL controller, the MPC control law offers a baseline control law which might results in improved system performance during training of the RL agent compared to the conventional RL strategy. This might also make the framework be more sample efficient. Besides, the framework will have a guaranteed stable performance as long as the MPC controller is stable. Additionally to the model-reference RL control scheme found in Subsection 3-1-2, this framework will also be robust through the rolling horizon scheme of the MPC controller because the MPC controller presents a stable and robust control law. Stability is not the biggest concern for RL in traffic signal control, but especially the improvement of sample efficiency and performance during learning is of big interest.

It must be noted that the two MPC controllers that can be observed in Figure 3-3 essentially are the same. They provide the same baseline control input to both the nominal system and the real system at every control time step because the MPC controller is updated with the state of the real system. this is different to the control scheme presented in Figure 3-2 where the baseline control input is not necessarily the same.

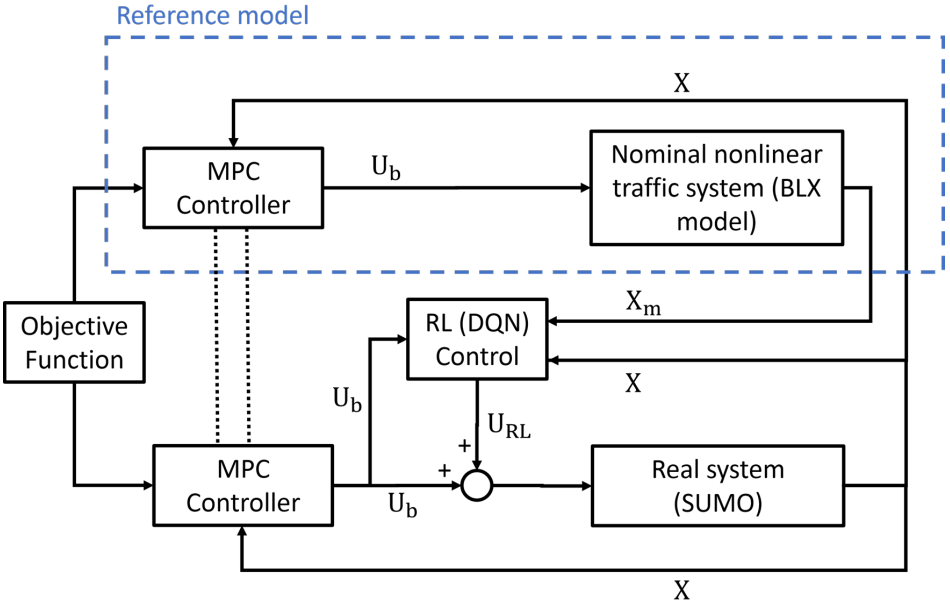


Figure 3-3: The combined model predictive control and reinforcement learning model-reference framework.

3-2 Framework design

This section will focus on the design of the proposed control framework. The objective is to obtain a control law that lets the state of the real system track the state of the nominal prediction model. As could be seen in Figure 3-3 the control law for the urban traffic system is described as:

$$u = u_{\mathrm{b}} + u_{\mathrm{rl}}, \tag{3-1}$$

where $u_{\mathbf{b}}$ is the baseline control input that is generated by the MPC controller based on the nominal prediction model, and u_{rl} is the control input generated by the deep RL adaptive control law. Both the MPC and the RL components have the same controller time step T_{c}. First, the design of the MPC controller for the computation of the baseline control law is discussed, then we will discuss how to obtain the adaptive RL control law.

3-2-1 Model predictive control

The prediction model used in the MPC is the BLX model which is formally described in Section 2-3. An optimization problem is solved at each controller time step T_c. The current time is defined as $t = k_{\mathrm{c}}T_{\mathrm{c}} = kT_{\mathrm{s}}$, where T_{s} is the simulation sampling time as in Section 2-3. The MPC solves an optimization problem over prediction horizon N_{p}.

The control input for the MPC controller consists of two parts per intersection $d \in J$, the cycle time $T_{\mathrm{cyc},d}$ and the green time percentage $\pi_{\mathrm{green},d}$, expressed as a percentage of the cycle time, corresponding to each traffic cycle. As the network considered in this thesis uses a two-phased cycle, the green time percentage is consists one variable per cycle per intersection. The input is computed at each controller time step k_{c}. The current cycle is defined as $l \in \{1, 2, 3, ..., N_{\mathrm{cyc}}\}$ and N_{cyc} is the total number of cycles of whole simulation period.

Dummy variables

The cycle time $T_{\mathrm{cyc},d}$ and green time percentage $\pi_{\mathrm{green},d}$ are determined independently for each intersection. As the cycle time is variable and the prediction horizon is static, it is not predefined how many decision variables must be considered per optimization. Therefore we must determine the maximum number of decision variables that can be present per optimization. The maximum number of cycles N_{cyc,N_p} per prediction horizon N_{p} is:

$$N_{\mathrm{cyc},N_{\mathrm{p}}} = \mathrm{ceil}(\frac{N_{\mathrm{p}}}{T_{\mathrm{cyc,min}}}) + 1, \tag{3-2}$$

where $T_{\mathrm{cyc,min}}$ is the lower bound of the cycle time. In the equation a value of 1 is added to make sure that the sum of the cycle times within N_{cyc,N_p} is never lower than prediction horizon N_{p}. It can happen that a cycle is already occurring when an optimization problem is solved. When subsequently the found solution for each cycle time of an intersection is equal to the lower bound, it is possible that an extra cycle starts within the prediction horizon. The maximum number of decision variables is for the MPC controller then becomes:

$$\mathrm{dv} = N_{\mathrm{cyc},N_\mathrm{p}} \cdot 2 \cdot \sum_{i=1}^{d \in J} i, \tag{3-3}$$

where the number of decision variables per cycle is twice the number of intersections, as each intersection has two decision variables per cycle. All decision variables that do not fall within the prediction horizon are dummy variables and will not be optimized.

Optimization problem

The MPC controller solves the following optimization problem at each controller time step T_c to determine the predicted optimal control inputs:

$$\min_{\boldsymbol{u}_{N_\mathrm{p}}(k_\mathrm{c})} \sum_{(u,d)\in L} \sum_{k=1}^{N_\mathrm{p}\cdot T_\mathrm{c}} T_\mathrm{s} \cdot \hat{n}_{u,d}(k), \tag{3-4}$$

$$\text{s.t. } x(k+j+1) = f(x(k+j), u(k_\mathrm{c}+j_\mathrm{c})) \quad \forall j = \{0,...,N_\mathrm{p}\cdot T_\mathrm{c}\}, \forall j_\mathrm{c} = \{0,...,N_\mathrm{p}\}, \tag{3-5}$$

$$u_{\mathrm{min,d}}(k_c) \leq \boldsymbol{u}_{N_\mathrm{p}}(k_c) \leq u_{\max}, \tag{3-6}$$

$$\mathcal{U}(\boldsymbol{u}_{N_\mathrm{p}}(k_c)) \leq 0, \tag{3-7}$$

here $\boldsymbol{u}_{N_\mathrm{p}}(k_c) = [\boldsymbol{u}(l), ...\boldsymbol{u}(l + N_{\mathrm{cyc},N_\mathrm{p}} - 1)]$, where $\boldsymbol{u}(l)$ contains the cycle time ($T_{\mathrm{cyc},d}$), and the green time percentage ($\pi_{\mathrm{green},d}$) for each intersection $d \in J$ at controller time step k_c. The function $f(\cdot)$ is the prediction model of the MPC controller, which is the BLX model. The objective function represents the total time spent (TTS) of all vehicles in the network within the prediction horizon, where $\hat{n}_{u,d}(k)$ represents the predicted number of vehicles on link $(u,d) \in L$ at time step k. $\mathcal{U}(\boldsymbol{u}_{N_\mathrm{p}}(k_\mathrm{c}))$ represents the nonlinear inequality constraints on the control input, which will be discussed later in this section. The decision variables of the optimization problem have an upper and a lower bound, where $u_{\max}$ is a vector with the maximum cycle time and green time percentage of appropriate size. The lower bound of the cycle time is adjusted after each optimization such that the end time of a cycle can never be moved to a time before the current time.

$$u_{\mathrm{min,d}}(l) = [\max(T_{\mathrm{cyc,min}}, t - \sum_{i=1}^{l-1} T_{\mathrm{cyc},d}(i)), \pi_{\mathrm{green,min}}]. \tag{3-8}$$

The control inputs, consisting of the cycle time and the green time percentage can be converted into the binary input that is accepted by the BLX model. A distinction is made between traffic arriving from the east and west direction for which $\mathrm{g}_{\mathrm{EW},d}(t)$ is used, and the north and south direction for which $\mathrm{g}_{\mathrm{NS},d}(t)$ is used. They are defined by:

$$\mathrm{g}_{\mathrm{EW},d}(t) = \begin{cases} 1 & \text{if } t \in \bigcup_{l=1}^{N_\mathrm{cyc}-1} [t^0 + \sum_{i=1}^{l} T_{\mathrm{cyc},d}(i) + y_d \\ & \qquad t^0 + \sum_{i=1}^{l} T_{\mathrm{cyc},d}(i) + \pi_{\mathrm{green},d}(l+1) \cdot T_{\mathrm{cyc},d}(l+1)], \\ 0 & \text{otherwise,} \end{cases} \tag{3-9}$$

and

$$g_{\mathrm{NS},d}(t) = \begin{cases} 1 & \text{if } t \in \bigcup_{l=1}^{N_{\mathrm{cyc}}-1} [t^0 + \sum_{i=1}^{l} T_{\mathrm{cyc},d}(i) + \pi_{\mathrm{green},d}(l+1) \cdot T_{\mathrm{cyc},d}(l+1) + y_d, \\ & \qquad t^0 + \sum_{i=1}^{l+1} T_{\mathrm{cyc},d}(i)], \\ 0 & \text{otherwise,} \end{cases} \tag{3-10}$$

where y_d is the length of the yellow phase, t^0 is the start time of the simulation, usually zero, and t is the current time. So essentially the starting and end time of each phase is determined and from these values the binary input needed for the BLX model is extracted.

The baseline control input (u_b) at time step t, described in Equation 3-1, is formulated per intersection as the total duration of the current phase and can be mathematically be described as:

$$u_b = \begin{cases} \pi_{\mathrm{green},d}(l) \cdot T_{\mathrm{cyc},d}(l) & \text{if } t < \sum_{i=1}^{l-1} T_{\mathrm{cyc},d}(i) + \pi_{\mathrm{green},d}(l) \cdot T_{\mathrm{cyc},d}(l), \\ T_{\mathrm{cyc},d}(l) - \pi_{\mathrm{green},d}(l) \cdot T_{\mathrm{cyc},d}(l) & \text{if } t \geq \sum_{i=1}^{l-1} T_{\mathrm{cyc},d}(i) + \pi_{\mathrm{green},d}(l) \cdot T_{\mathrm{cyc},d}(l). \end{cases} \tag{3-11}$$

Non-linear constraints

When an optimization problem is solved, the predicted optimal control input is implemented for control sampling time T_{c}. The challenge with using variable cycle time for each intersection is that the next optimization problem is often solved when a cycle is not exactly finished. This means the same values of the control inputs are regularly recomputed. The value of the decision variables, therefore, have to be constraint whenever part of a cycle has already occurred.

At the end of the implementation of the computed control input for control sampling time T_{c} the current time is t. When at time $t$the first phase of the cycle has not yet passed, the following non-linear inequality constraint must apply in the next optimization problem:

$$\mathcal{U}(\boldsymbol{u}_{N_{\mathrm{p}}}(k_{\mathrm{c}})) = t - (\sum_{i=1}^{l-1} T_{\mathrm{cyc},d}(i) + \pi_{\mathrm{green},d}(l) \cdot T_{\mathrm{cyc},d}(l)) \leq 0. \tag{3-12}$$

The constraint is such that the moment of the green time switch can not occur before t. Figure 3-4 clarifies the situation.

It could also happen that at the end of the implementation of the predicted optimal control input for control sampling time T_{c}, the current phase is the second phase of the cycle. This is illustrated in Figure 3-5, and is described as:

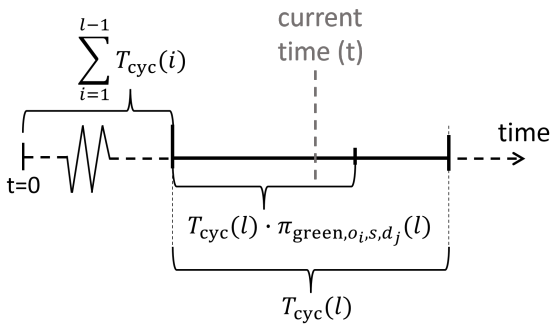


Figure 3-4: Graphical representation of the situation when the non-linear inequality constraint is active.

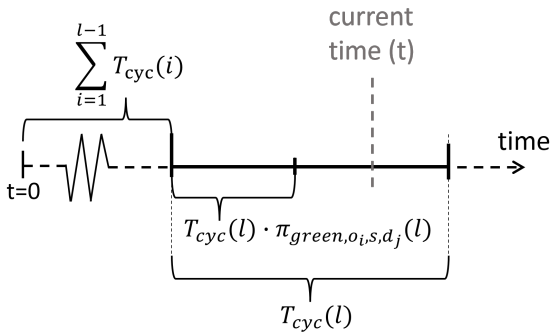


Figure 3-5: Graphical representation of the situation when the non-linear equality constraint is active.

$$t > \sum_{i=1}^{l-1} T_{\text{cyc},d}(i) + \pi_{\text{green},d}(l) \cdot T_{\text{cyc},d}(l). \tag{3-13}$$

For the next optimization problem, the moment of the green time switch must stay at the exact same time instants. The cycle time $T_{\text{cyc},d}$ is still allowed to be extended by the controller, but the product of cycle time and green time percentage must stay the same. It is decided to not include this requirement in the optimization problem as an equality constraint, but instead omit this constraint when this situation occurs. This is done by reducing the number of decision variables of the foremost cycle for that specific intersection. First, the length of the first phase of the cycle (t_{phase_1}) is saved as

$$t_{\text{phase}_1} = \pi_{\text{green},d}(l) \cdot T_{\text{cyc},d}(l). \tag{3-14}$$

Then the green time percentage $\pi_{\text{green},d}(l)$ of the first cycle of the intersection is removed from the decision variables. Only the cycle time $T_{\text{cyc},d}(l)$ of the first cycle is optimized for that intersection. After the new optimization, where a new cycle time is determined, the $\pi_{\text{green},d}$ can be calculated as:

$$\pi_{\text{green},d}(l) = \frac{t_{\text{phase}_1}}{T_{\text{cyc},d}(l)}, \tag{3-15}$$

where $\pi_{\text{green},d}(l)$ is the re-optimized value compared to Equation (3-14). When the cycle completely finished, the control inputs of that intersection for the finished cycle (consisting of the cycle time $T_{\text{cyc},d}(l)$ and the green time percentage $\pi_{\text{green},d}(l)$) are saved and the next cycle is considered to be the current cycle ($l+1 \longrightarrow l$ and $l+2 \longrightarrow l+1$, etc.).

Optimization approach

As the prediction model of the MPC controller is highly nonlinear, the optimization problem is a non-convex problem by definition. To try to approach a global solution, the problem will be solved using the *fmincon* solver with the SQP algorithm of the Optimization Toolbox of

Matlab in combination with a multi-start algorithm. With multi-start the same optimization problem will be solved multiple times using a different set of initial starting points for the decision variables. In this way, multiple local optima are found. The lowest value is considered the best estimate for the global optimal solution. Using more initial conditions will increase the reliability of a good approximation for the global solution. The solver uses ten initial points:

1 & 2: the best solutions of the last optimization problem and the previous solution shifted by one cycle,

3, 4 & 5: the values at a quarter, half, and three quarters on the range between the lower and the upper bound,

6 to 10: random values between the lower and upper bound.

The default settings of *fmincon* with SQP are used with one alteration:

- *FiniteDifferenceStepSize* is set to $1 \cdot 10^{-3}$.

This setting is changed as the default step size is too small for the optimization problem. As the BLX model is a discrete model with simulation sample time $T_\mathrm{s} = 1$ s, changes in the phase length that are smaller than 1 s will not change the behaviour of the system. The gradient of the cost function will therefore often be zero, and the algorithm settles for a local minimum near the starting point. It is therefore important that the step size of the algorithm is large enough. The precise value of the setting is determined by experimentation.

3-2-2 Reinforcement learning as an adaptive law

To obtain the RL-based adaptive law, the urban traffic system is represented as a Markov decision process (MDP) denoted by tuple $< S, A, R, T >$ as discussed in subsection 2-1-1. The state $s(k_\mathrm{c}) \in S$, the action $u_\mathrm{rl}(k_\mathrm{c}) \in A$ and the immediate reward $R(k_\mathrm{c}) = R(s(k_\mathrm{c}), u_\mathrm{rl}(k_\mathrm{c}))$ are defined in this section.

State

The state-space of the RL agent contains a vector with the difference between the relevant states of the reference model and (x_m) of the real system (x) at the current time step. This is the difference in the number of vehicles on each link of the network. The vector containing all these values is described as:

$$\delta_n(k_\mathrm{c}) = \hat{n}_{u,d}(k_\mathrm{c}) - n_{u,d}(k_\mathrm{c}) \quad \forall (u,d) \in L. \tag{3-16}$$

Furthermore, information is provided of the control input of the baseline controller. This is translated in threefold, namely (1) the phase ($\mathrm{ph}_d(k_\mathrm{c}) \in \{1, 0\}$) of the traffic light at junction $d \in J$ at the current time step as binary input, where 1 represents that the north-south

phase is green and 0 that the east-west phase is green. (2) The total duration of the current phase ($T_{\text{ph,total}}(k_c)$) at each intersection in seconds, and (3) the time that has already elapsed ($T_{\text{ph,done}}(k_c)$) at each intersection within the current phase in seconds. The state $s(k_c) \in S$ is computed as:

$$s(k_c) = [\delta_n(k_c), \text{ph}_d(k_c), T_{\text{ph,tot,d}}(k_c), T_{\text{ph,done,d}}(k_c)] \qquad \forall d \in J. \tag{3-17}$$

The state information consists of one value per link, and the baseline control input information consists of three values per intersection. The size of the action space per control time step is 20 for the case study considered in this thesis.

Action

The RL-based adaptive law must be able to adapt the control input that is computed by the MPC controller. This input consists the length of current phase. The RL agent is able to change the length of the current phase of the cycle by a certain amount of seconds. Deep Q-network (DQN) is used as RL algorithm, it is therefore needed that the action space of the RL agent is discretely represented such that the size of the action space is finite an within reasonable bounds. The action for one intersection $a_d(k_c)$ will be an integer value between some chosen bound, where the set of action is not necessarily evenly distributed over the range. The action space A consists of all possible combinations of action for all intersections in the network. In general the action $u_{\text{rl}}(k_c)$ can be defined as:

$$\begin{aligned} u_{\text{rl}}(k_c) &= [a_1(k_c), ...a_d(k_c)], a_d(k_c) \in \mathbb{Z} \qquad \forall d \in J, \\ a_{\min} &\leq a_d(k_c) \leq a_{\max}, \end{aligned} \tag{3-18}$$

where $a_{\min}$ and $a_{\max}$ are the minimum and maximum value for the action. In this thesis the set of actions for one intersection is:

$$a_d(k_c) \in \{-10, -5, -3, -1, 0, 1, 3, 8, 10\} \quad \forall d \in J. \tag{3-19}$$

There are nine actions per intersection, which means that the agent can choose from a total of $9^{(\sum_{i=1}^{d \in J} i)}$ action combination. In the case study in this thesis, where two intersections are considered, this means an action space of size 81.

Reward

Within the model-reference RL framework, it is the goal of the RL agent to follow the relevant reference state determined by the MPC controller exactly. The reward is based on tracking the predicted number of vehicles $\hat{n}_{u,d}$ in the network. As it is not possible to model constraints for the RL agents explicitly, input constraints can be added implicitly in the reward function. When applicable, a reward of $R_c = -10$ is added to the immediate reward. This happens when an infeasible action is chosen by the RL agent. Which is the case when the RL agent wants to shorten the phase more than there is time left within this phase. During training,

it has appeared that the agent stops choosing infeasible actions very quickly. The immediate reward $R(k_c)$is:

$$\begin{aligned} R(k_c) &= -(x(k_c) - x_m(k_c))H(x(k_c) - x_m(k_c)) + R_c, \\ &= -(n_{u,d}(k_c) - \hat{n}_{u,d}(k_c))H(n_{u,d}(k_c) - \hat{n}_{u,d}(k_c)) + R_c \quad \forall (u,d) \in L, \end{aligned} \tag{3-20}$$

where $H > 0$ is a positive definite scaling matrix. No extra reward is considered when the terminal state is reached. The return is determined by Equation (2-2).

The reward is specifically designed for the situation and network that is considered in this thesis. However, when other objectives are of interest as well, as discussed in 3-1-2, the reward can be changed accordingly. In this thesis it would have also been a logical choice to set the reference state to zero with the aim to achieve the best performance. However, it is more important to examine the achievement of the RL adaptive law when following a reference signal than to maximize performance.

Training

During training the parameters θ of the Q-network are trained using the deep Q-network (DQN) algorithm. The DQN algorithm for the training of the neural network is illustrated in Figure 3-6. It uses a target network and experience replay as described in Section 2-1-5. The training algorithm including the MPC component is presented in Appendix A-1.

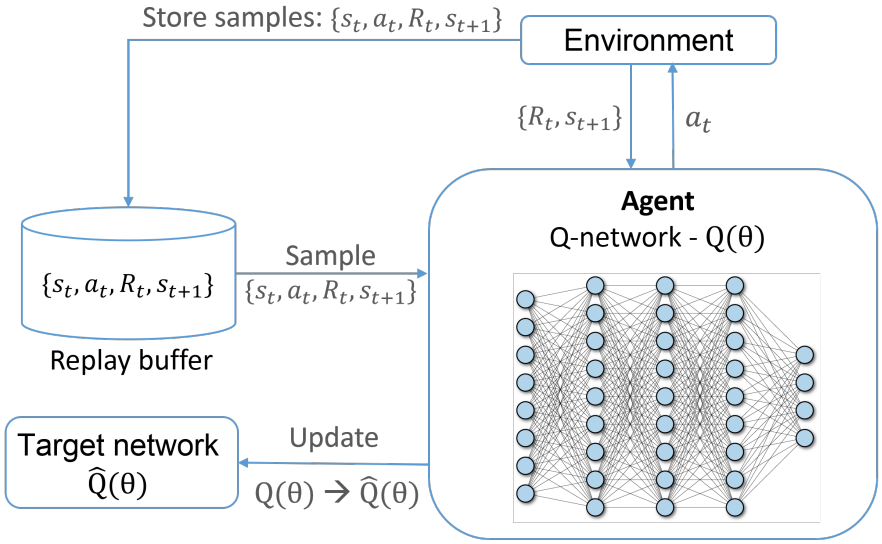


Figure 3-6: The offline training process of the deep reinforcement learning using DQN, at each time step the sample $\{s(k_c), u_{rl}(k_c), R(k_c), s(k_c+1)\}$ is obtained and stored in the replay buffer. These samples are randomly sampled in batches with as goal to update the Q-network. The target network ($\hat{Q}(\theta)$) is updated after T_{targ} controller time steps.

The critic network is chosen to be a fully connected multiple layer perceptron with rectified linear unit (ReLU) non-linearities as activation functions. The chosen network consists of five fully connected layers: the input layer, three inner layers and an output layer. The number

of neurons in each inner layer is 256, 128 and 64 respectively. The input layer is as big as the number of states and the output layer as the number of actions. As traffic signal control is essentially an infinite process, an episode is therefore terminated after a set amount of time. The parameters that are used by the DQN algorithm in the framework are presented in Table 3-2. Here α is the learning rate of the critic network, γ is the discount factor used for the computation of the return, ϵ_0 is the starting value of ϵ in the ϵ-greedy method for balancing exploration and exploitation, δ_ϵ is the ϵ decay rate and $\epsilon_{\min}$ is the minimum value of ϵ during training. The target network is updated every T_{targ} time step. The other parameters are the experience replay batch size, the optimization method, the activation function of all the neurons, the maximum size of the experience replay buffer and all the dimensions of the network layers.

Table 3-2: Parameters for the DQN algorithm used for the model-reference framework.

Episodes	α [-]	γ [-]	ϵ_o [-]	δ_ϵ [-]	$\epsilon_{\min}$ [-]	T_{targ}
1000	0.01	0.95	1	0.001	0.01	10

Batch size	Optimizer	Act. func.	Buffer size	Input dims.	Output dims.	No. layers
256	Adam	ReLU	$1 \cdot 10^6$	14	81	5

3-3 Conclusions

Model predictive control and reinforcement learning have the potential to perfectly complement each other. Besides, the significance of describing signal control in an urban traffic network as a regulation problem is recognized. In this chapter, we have defined the model-reference RL framework that combined MPC and RL. In the model-reference RL framework an MPC controller provides a baseline control input and a RL agent is used as an adaptive law to compensate for disturbance or unmodeled dynamics. The framework is designed specifically for the control of traffic signals in an urban traffic network. The model predictive control problem is designed as the baseline control input defined in the framework. The BLX model is used as a prediction model and corresponding control input and constraints are defined. The RL adaptive law is defined as a Markov decision process, where the agent can adjust the control input of the baseline MPC controller by changing the phase length of each intersection. A neural network with three inner layers shall be used to allow for function approximation of the system and DQN is used as the training algorithm.

Chapter 4

Case study

In this chapter, simulations are performed on a small grid-based traffic network. The set-up of the network for the case study is discussed first. This consists of the topology of the network, the settings for the model parameter, performance indicators, the demand patterns and disturbance for the simulation. The three benchmark controllers that are used for comparison are discussed as well.

Next, two case studies are considered. Both are based on the set-up in Section 4-1. The first case study considers a fixed cycle time as a control input for the MPC controller and uses a disturbed version of the BLX model to represent the real traffic states. The case study has been successful and shows promising results. The system performance and training results are analyzed and compared to a conventional MPC controller, a conventional RL-based controller and a fixed-time controller, all of which will be discussed and described in this chapter as well. In the second case study SUMO is introduced: a traffic simulations tool. The training results are discussed together with the efforts taken to improve these training results.

4-1 Network set-up

In order to analyse the performance of the combined MPC and RL model-reference framework, the conventional MPC and RL-based controller, a small urban traffic network is designed. The network is mathematically described using the BLX model.

The network considered is the grid-based traffic network shown in Figure 4-1. The network consists of two intersections with controllable traffic signals, each with four connected two-way roads. There are seven two-way roads in total, each road in each direction has three lanes dedicated to right, straight and left going traffic. The network has six source and sink nodes. Most roads have a length of 500 m with the exception of road numbers 7 and 8, these have a length of 550 m. We consider a small network as the goals is to gain insight on the performance of the framework and to proof its potential. The network is small enough to use a discrete action for the RL adaptive law and allowing for the use of DQN. Furthermore, the network is small enough for efficient centralized control of the baseline MPC controller.

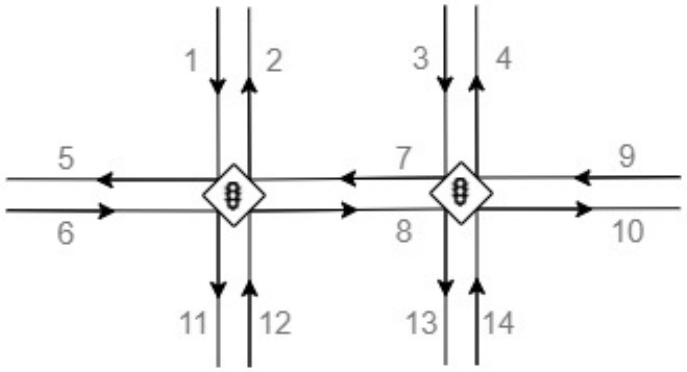


Figure 4-1: The studied grid-based traffic network. The numbers present in the network are the link IDs, the arrows in each line represent the driving direction.

The free-flow speed of the vehicles in the network is 50 km/h and the average length of each vehicle is 7 m. The vehicles enter the network at the source links 1, 3, 6, 9, 12 and 14. The prediction horizon of the MPC controller in the framework and the conventional MPC controller is $N_{\mathrm{p}} = 3$ (i.e. 180 s), which is enough time for a vehicle to leave the network, whichever route is chosen. The control horizon is equal to the prediction horizon.

Each intersection has a controllable traffic signal with a fixed traffic cycle. Each traffic cycle consists of two phases as shown in Figure 4-2. The first is a green light for all traffic arriving from the north or south direction, the other is a green light for all traffic arriving from the east and west direction. The yellow time at the beginning of each phase is 5 s for all situations and controller types.

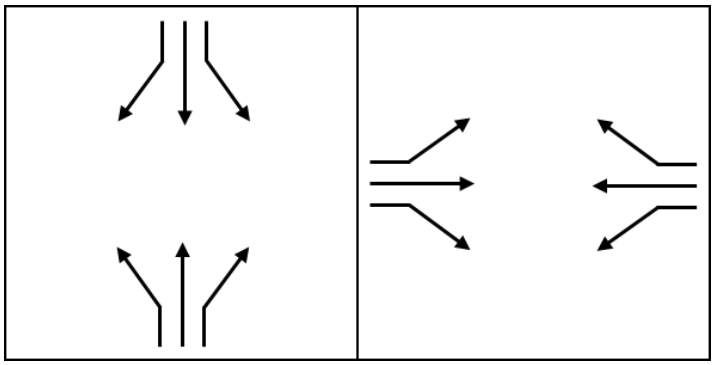


Figure 4-2: The two traffic phases for the controller. In phase 1 all traffic arriving from the east and the west direction have right of way, in phase 2 all traffic arriving from north and south direction have right of way.

The amount of traffic on each road and link in the network is determined by two components: (1) the number of vehicles entering the network on the six entering roads (namely, link 1, 3, 6, 9, 12 and 14), and (2) the turn ratio of each link. For each link, the turn ratios of its three roads sum up to one. The value of the turn ratio for each road can be found in Table 4-1.

Table 4-1: The turn ratio of all origin links of the network

Direction	β_1	β_3	β_6	β_7	β_8	β_9	β_{12}	β_{14}
Right	0.12	0.44	0.44	0.35	0.35	0.12	0.35	0.12
Straight	0.50	0.44	0.44	0.35	0.35	0.48	0.35	0.44
Left	0.38	0.12	0.12	0.30	0.30	0.40	0.30	0.44

4-1-1 Performance indicator

All controllers are compared in terms of system performance. This is measured as cumulative delay, or in other words the total time spent (TTS) of all vehicles in the network. The TTs is determined as:

$$\text{TTS} = \sum_{(u,d)\in L} \sum_{t=0}^{T_{\text{end}}} T_s \cdot n_{u,d}(t), \tag{4-1}$$

where $n_{u,d}(t)$ is the number of vehicles on lane (u, d) on time step t, T_s is the simulation sampling time, which is 1 s, and T_{end} is the end time of the simulation. Besides the absolute TTS, the controllers are also compared relative to a benchmark, which is the fixed-time controller. The relative TTS is:

$$\text{TTS}_{\text{rel}} = \frac{\text{TTS}_{\text{x}} - \text{TTS}_{\text{ft}}}{\text{TTS}_{\text{ft}}} \cdot 100\%, \tag{4-2}$$

where TTS_{ft} stands for the TTS of the fixed-time controller and subscript x represent any other controller.

Next, the training performance of the model-reference RL framework can be compared to the conventional RL-based controller. The performance indicators used to assess training performance are:

1. The convergence speed of the training process,
2. The system performance during training.

The convergence speed, or sample efficiency of the agent, can be tested by observing the number of episodes or time steps needed until a steady-state of the episode reward is reached. As for the system performance during training: both agents use a different reward function for training. The TTS for each training episode is extracted such that a fair comparison can be made between the system performance of the two agents during training.

4-1-2 Demand generation

To be able to compare the performance of the different controllers, three demand patterns are designed. One with constant demand high demand and two with fluctuating demand. In the first fluctuating demand, the demand arriving from the north and south directions (lane 1, 3, 12 and 14) is equal, as well as the demand on the remaining lanes (lanes 6 and 9). In the second fluctuating demand data set, all demand from the west direction (lane 1, 6 and

12) is the same, as well as the traffic demand from the east direction (lanes 3, 9 and 14). From here on, the constant reward will be referred to as demand A, and the two fluctuating demand patterns are demand B and C respectively. The three demand sets are depicted in Figure 4-3.

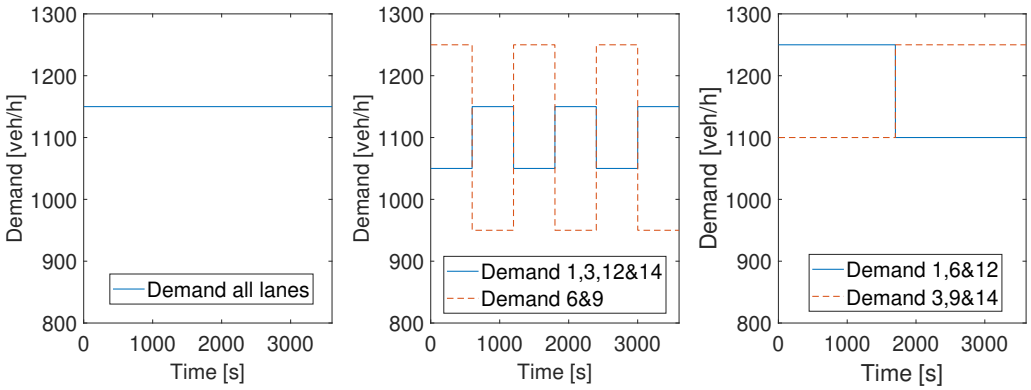


Figure 4-3: The demand input of the different scenarios. The first plot is demand A: the high demand case, the second plot shows demand B: fluctuating demand 1 for north and south lanes, and the third demand C: fluctuating demand 2 for east and west lanes.

Before each simulation or training episode, the network is initialized with some traffic entering the network. Low traffic demand is inserted into the network for 300 seconds with the signal sequence of the fixed-time controller, such that there is some traffic present in the network that will be used as an initial state.

4-1-3 Disturbed demand

The model-reference RL framework is designed for improving performance in an uncertain system such as an urban traffic network. Therefore, disturbance needs to be added to the mathematical model representing the real traffic system to test the performance of the framework. Disturbance is applied on the traffic demand that enters on the different source links $(1, 3, 6, 9, 12$ and $14)$, numbered as in Figure 4-1. The demand is disturbed with the sinusoidal wave as shown in Figure 4-4 such that it has the following characteristics: (1) the disturbance is varying over time, but not random and (2) the sum of all disturbed demand is zero at each point in time.

4-2 Benchmark

The framework will be compared to a fixed-time controller, a conventional MPC controller and a conventional RL-based controller. In this section, the design of the benchmark controllers is discussed.

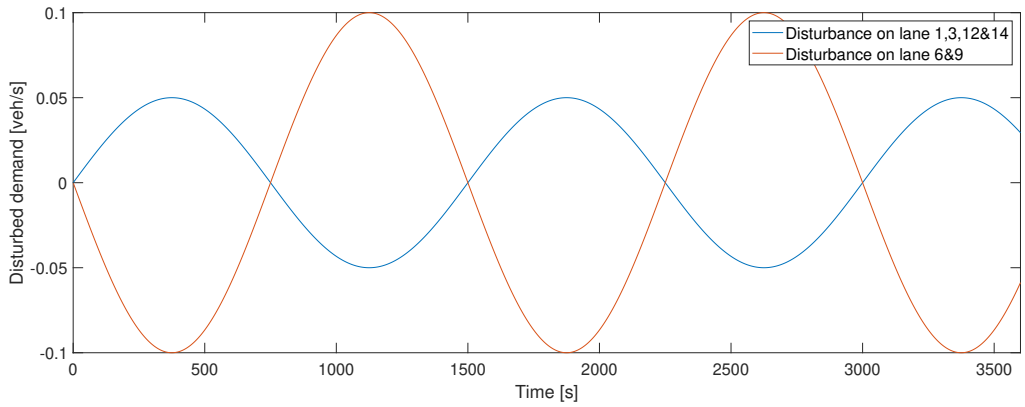


Figure 4-4: The disturbance on the traffic demand for the different lanes in the network. The blue line represents the disturbance on lanes 1, 3, 12 and 14 of the network, the red lane the disturbance on lanes 6 and 9.

4-2-1 Fixed-time control

The fixed time controller is essentially the no control case. The cycle time is fixed to 60 s and the green time percentage is set to the mean green time percentage of the intersection (i.e. 50% for the two-phased network). With a yellow time length of 5 s for each phase, this means that each phase has 25 s of green time. By comparing the framework and the other benchmark controllers to the fixed-time control case, conclusions can be drawn to the performance increase related to each controller.

4-2-2 Conventional MPC controller

The conventional MPC controller that is used to compare the performance of the framework is the same MPC controller that is used within the framework. The design of the controller can be found in Section 3-2-1.

4-2-3 Conventional RL-based controller

To be able to test the training performance of the model-reference RL framework, the framework is compared to a normal RL-based controller. The RL-based controller is designed by representing the problem as a Markov decision problem, with the suggestions of problem design of El-tantawy et al. [12] as a reference. The action, state and reward are described below.

Action

In the system a fixed phase sequence is considered, this is an immediate result of the two-phased traffic light. The action at each control time step T_{r}, which is set to 1 s, is chosen to be binary: $\{1, 0\}$ for each traffic light. Here, 0 is the choice to be in the first phase of the cycle and 1 to be in the second phase of the cycle. The action a_t per intersection is:

$$\begin{aligned} a_t &= [a_1, ..., a_d] \quad \forall d \in J, \\ a_d &= \{0, 1\}. \end{aligned} \tag{4-3}$$

The total number of differentiable actions for the whole network depends on the number of junctions in the network and is equal to $2^{(\sum_{i=1}^{d \in J} i)}$. For the two-junction network considered in this case study, the number of actions is four.

State

The state space is represented vector-based and consists of three parts. The first is the index of the green phase currently employed for each traffic light, which is equal to the previous action. The second is the time elapsed in the current phase, this is represented in two-fold by the length of the yellow time (Y_t), and the length of the green time of the current phase (G_t). These cannot be nonzero simultaneously. Furthermore, the state includes the number of vehicles in the queue on each lane $(u, d) \in L$ and the number of vehicles present on each link (u, d). The state s_t at time step t is formalized as:

$$s_t = [u_{t-1}, G_{t,d}, Y_{t,d}, \boldsymbol{x}_{u,d}(t-1)] \quad \forall d/in J, \tag{4-4}$$

where $\boldsymbol{x}_{u,d}(t)$ is a vector with the number of vehicles and the queue length $\boldsymbol{x}_{u,d}(t) = [n_{u,d}(t), q_{u,d,o}(t)]$. The total number of states N_s is equal to $N_s = 3 \cdot (\sum_{i=1}^{d \in J} i) + 4 \cdot (\sum_{j=1}^{(u,d) \in L} j)$. For the case study considered this will become $3 \cdot 2 + 4 \cdot 14 = 58$ states.

Reward

The reward is represented by a scalar function and is computed by the sum of the difference between the queue length of the current time instance and the queue length of the previous time instance for each lane on each link. The reward can be both positive and negative. When the reward is positive this means the total queue length in the network is reduced after the action implementation. A negative reward means an increase in queue length. The immediate reward is defined as:

$$r_t = \sum_{(u,d) \in L} \sum_{o \in O_{u,d}} (q_{u,d,o}(t-1) - q_{u,d,o}(t)). \tag{4-5}$$

Algorithm design

DQN is used as the RL algorithm. In Table 4-2 the parameters used by the algorithm are presented. These are the learning rate α, the discount factor γ, the initial epsilon ϵ_0, the epsilon decay rate δ_ϵ, the minimum value of epsilon during the learning process $\epsilon_{\min}$ and the target update frequency of the target network T_{targ}.

The neural network used to represent the system consists of one input layer with a dimension of 62, representing the states, two inner layers of size 128 and 64 consecutively and an output layer representing all action options, thus with a dimension of 4. The ReLU function is used as an activation function by the neurons in the inner layers.

Table 4-2: Parameters for the DQN algorithm used for the agent of the conventional RL-based controller.

Episodes	α [-]	γ [-]	ϵ_o [-]	δ_ϵ [-]	$\epsilon_{\min}$ [-]	T_{targ}
1000	$1 \cdot 10^{-4}$	0.95	1	$1 \cdot 10^{-3}$	$5 \cdot 10^{-2}$	10

Batch size	Optimizer	Act. func.	Buffer size	Input dims.	Output dims.	No. layers
256	Adam	ReLU	$1 \cdot 10^6$	14	81	5

4-3 Case study I: Simplified set-up

To do a performance comparison of the model-reference RL controller, a simplified version of the framework is considered. The flexibility of the MPC controller is reduced by fixing the cycle time of the system to 60s, meaning the controller can only change the green time percentage. The action space of the RL agent that is used in the framework as the adaptive law is also slightly adjusted. The agent can still change the time of the current phase, but in the original set-up, this would also mean a change in the total cycle time. This is not enforced in this set-up. The RL agent can adjust the green time percentage once every cycle at the beginning of the cycle, but the cycle time doesn't change. The explicit action is still a change in phase time of the first phase, but as the first phase is extended the second phase is shortened such that the cycle time stays 60 s. To represent the real system a disturbed version of the BLX model is used. As discussed in Section 4-1-3, the disturbance is implemented by changing the traffic demand pattern.

A case study is performed, such that the performance of the model-reference RL framework can be compared to other benchmark controllers, namely a fixed-time controller a conventional MPC controller and a conventional RL-based controller. The design of these controllers is discussed in section 4-2. Note that also for the conventional MPC controller a fixed cycle time is used such that a fair comparison can be made between the framework and the MPC controller. It is assumed that the conventional MPC controller and the MPC controller in the network receive an accurate state update after each control time step.

Note that the non-linear constraints mentioned in Section 3-2-1 will never be active when a fixed cycle time is used, this is the case for both the framework and the MPC controller. Also, the penalty for an infeasible action that is added to the reward function of the agent of the RL adaptive law is not considered anymore as no infeasible action can be chosen with the setting of the fixed cycle time.

4-3-1 Training

The model-reference RL framework will be compared to the conventional RL-based controller on their performance during the learning process. As both controllers have a different reward function, it is not possible to directly compare the values of their reward function during training. To accomplish a fair comparison, the TTS of the vehicles in the system is documented during the training process and these values are compared. A total of 11 training demand sets are created, all with the length of one hour (3600 time steps). The episode terminates when t = 3600 s for all training episodes. Similar to the demand during simulation, the system is

initialized with some traffic by inserting a low demand into the network for 300 s with the fixed-time control signal input. Both controllers are trained using these same demand sets. The training lasts for 1000 episodes.

In Figure 4-5 the training process of the conventional RL-based controller is presented. Figure 4-6 shows the training process of the model-reference RL controller. The red line in both figures represents the moving average of the last ten episodes. Note that due to the use of different demand sets during training the reward value of both learning curves does not converge to a specific value, but will always fluctuate in all stages of the learning process. Despite this fluctuation, the plots show that the episode reward of the framework converges to an optimal policy.

The RL algorithm of the RL-based controller learns a well-performing policy after approximately 200 episodes. After approximately 220 episodes the reward function deteriorates again, which is likely to be caused by a combination of catastrophic forgetting and overfitting of the network. Both the agent that is retrieved at the end of the 1000 episode learning process as the agent at episode 199 are saved, in Section 4-3-2 we will further analyse which agent is preferred for simulation purposes.

Figure 4-7 shows the TTS of both the agents for RL-based controller and the framework during training over all episodes. The TTS represents the system performance the controllers have achieved. The first thing that can be concluded is that the framework performs very well in all training episodes, already from the first episode the TTS is low. This happens because the performance of the framework is mainly dependent on the performance of the baseline controller. The baseline control input of the MPC controller provides stable and well-performing input from the start of the learning process. The RL agent for the RL-based controller, however, does not perform well up until around the 160^{th} episode. The RL-based controller has to learn his entire policy from scratch. The flexibility of being able to change the phase every second causes bad performance in the starting phase of the learning process.

In Table 4-3 the training results of both RL algorithms are summarized. The start TTS is the average TTS after the first ten training episodes. The avg. TTS is the average TTS of the last ten episodes. For the RL-based controller, the avg. TTS consists of two values, the first is the average of the last ten values before episode 199, the other the last ten values before episode 1000. Besides, the convergence speed of each algorithm is approximated both in the number of episodes as the number of time steps.

It can be concluded that the framework is superior in terms of system performance during training, as it performs well in every training episode. The algorithm for the RL-based control is superior in terms of convergence speed. Although the algorithm for the framework needs fewer time steps to converge to an well-performing policy, the one for the RL-based controller needs way fewer episodes. In another system where the control sampling time for both control structures is the same (i.e. not urban signal control), the framework has a great potential to be more sample efficient than the conventional RL agent.

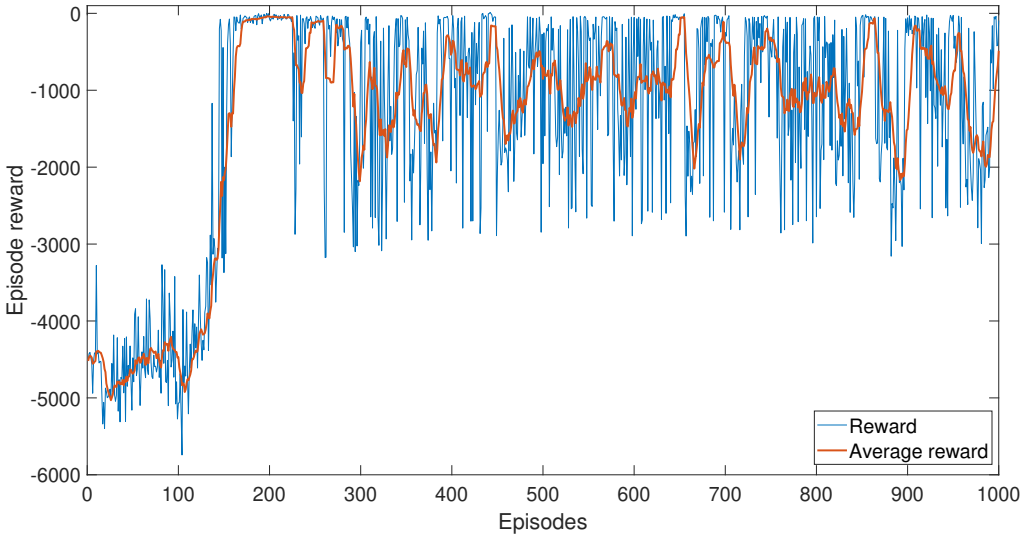


Figure 4-5: The learning curve of agent used in the conventional RL-based controller.

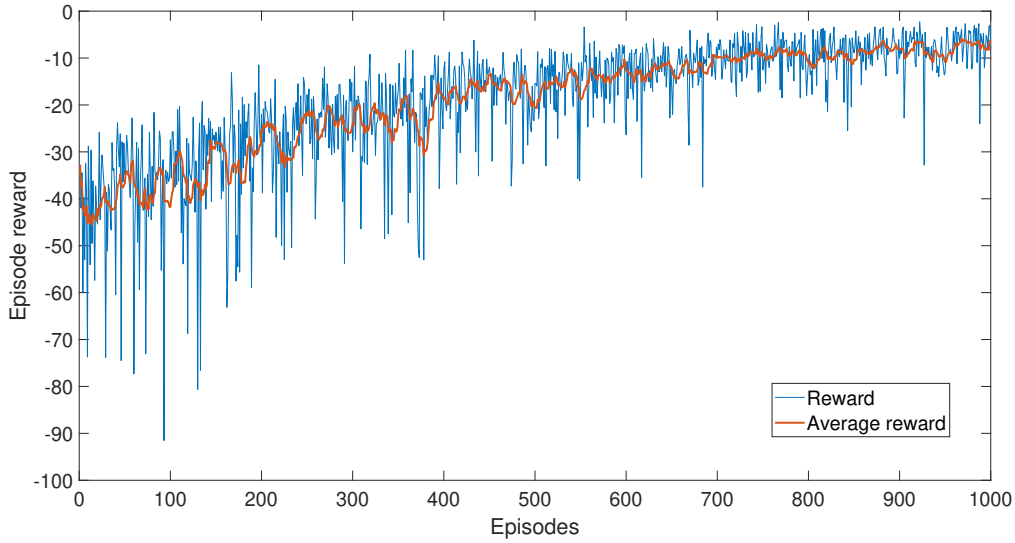


Figure 4-6: The learning curve of the agent of the adaptive RL law of the model-reference RL framework.

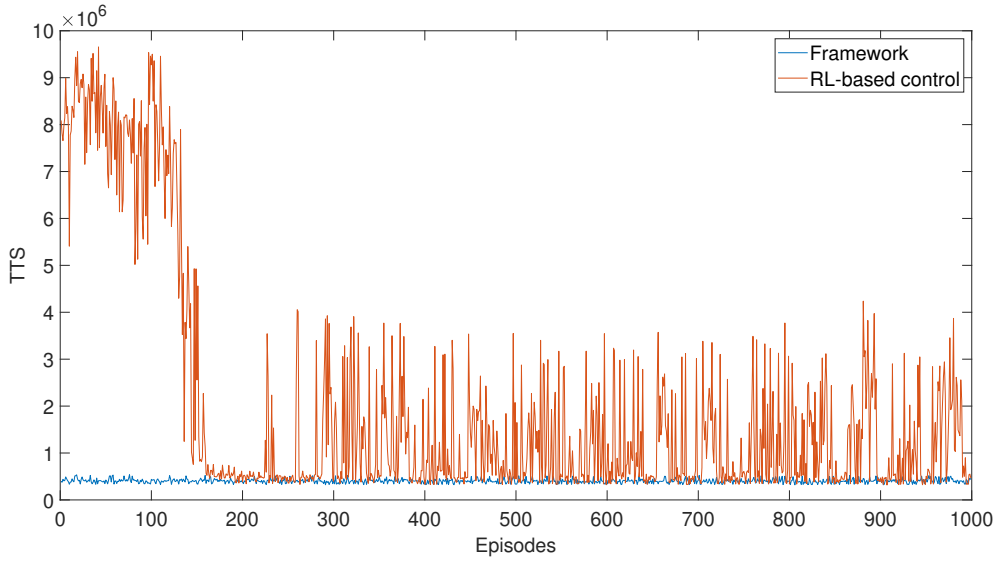


Figure 4-7: The system performance is expressed in TTS during training for the conventional RL-based controller and the framework.

Table 4-3: The training performance of the RL-based controller and the model-reference RL framework, in terms of the TTS at the start, the average TTS at the final training episode and the convergence speed.

Controller	start TTS [veh/h]	Avg. TTS [veh/h]	Approx. ep. till conv.	Approx. time steps till conv.
RL-based control	21,837	135/144	200	720,000
Model-reference RL framework	114	110	900	54,000

4-3-2 RL agent selection

As for the results of the training (i.e. the learning curve), it it not directly apparent which agent is the logical choice for the conventional RL-based controller. The agent has been saved at different instances throughout the training process. Two agents are selected for analysis, namely the agent at the end of the learning process (agent 1000) and the agent obtained after episode 199 (agent 199), where the agent seemed to be converged.

Simulations are conducted with the RL-based controller using both agents for all three demand types. In Figure 4-8 the results are shown for demand A, B and C respectively from top to bottom and Table 4-4 contains the TTS for each simulation. Agent 1000 performs better in two of the three simulations in terms of TTS. In the last simulation, it can be observed that after a certain time the states of the system increase linearly, thus becoming unstable. Agent 1000 has been over-fitted on the training data and has shown the be very susceptible to generating unstable actions. Agent 199 has a more general adaptive policy that is able to perform well in a wider range of situations, because it can react better to new conditions.

Agent 199 is selected as the best performing agent, as a reliable agent in multiple situations is preferred over a good performing agent with changes of instability. Agent 199 will therefore be used for further comparison.

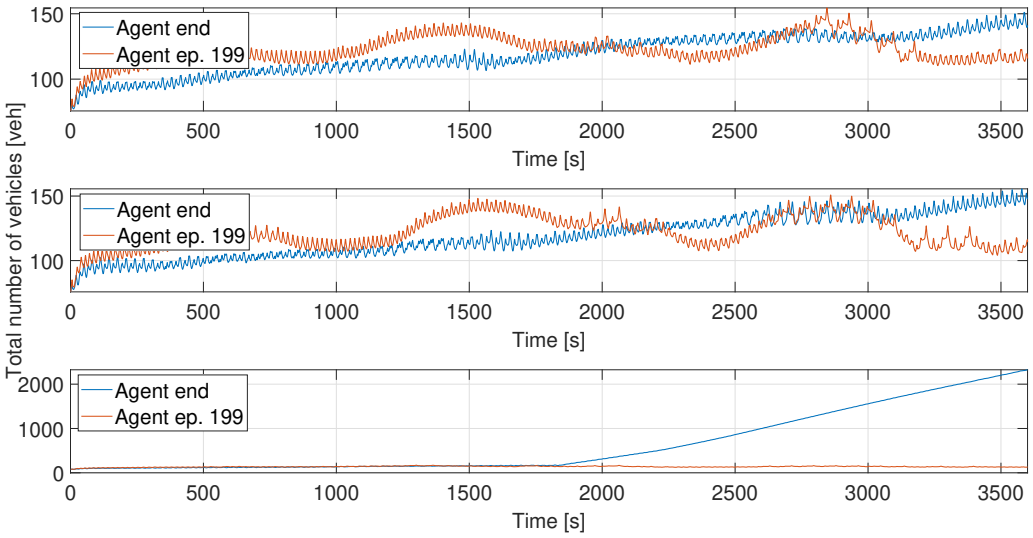


Figure 4-8: The TTS of the two candidate RL agents for the RL-based controller for all three demand patterns. From top to bottom these are demand A, demand B and demand C.

Table 4-4: The system performance of the two candidate RL agents for the RL-based controller in terms of TTS for all three demand patterns.

Agent	Demand A TTS [veh/h]	Demand B TTS [veh/h]	Demand C TTS [veh/h]
Agent 199	122.5	121.6	138.2
Agent 1000	118.7	118.8	650.6

4-3-3 Simulation results

After training the agent of the framework and the RL-based controller, the performance of all controllers is analyzed by simulating them. The simulations are done with the demand pattern described in Section 4-1-2. Note that the value of ϵ in the exploration method is set to zero for each RL agent to achieve a deterministic policy.

First, we analyze the behaviour of the framework after training. Figure 4-9 shows results of a simulation using the trained framework with demand B: the fluctuating demand set 1. It depicts the total number of vehicles in the network set by the reference model, compared to the real states. Remember that the reward function of the RL adaptive law of the framework is based on adjusting the real state of the system to copy the reference state that is set by the nominal model. The figures show that the RL adaptive law is very capable to compensate for the imposed disturbance such that the real state follows the reference state well. Note that the baseline controller (i.e. the MPC controller) receives a state update after each control time step, which means that every 60s the nominal state and the real state are equalized.

All four controllers are simulated with all three demand data sets. In Table 4-5, 4-6 and 4-7 the system performance in terms of TTS and relative TTS of the different controllers for all three demand scenarios (i.e. demand A: high demand, demand B: fluctuating demand 1 and demand C fluctuating demand 2 respectively) are shown and compared. The relative TTS is taken

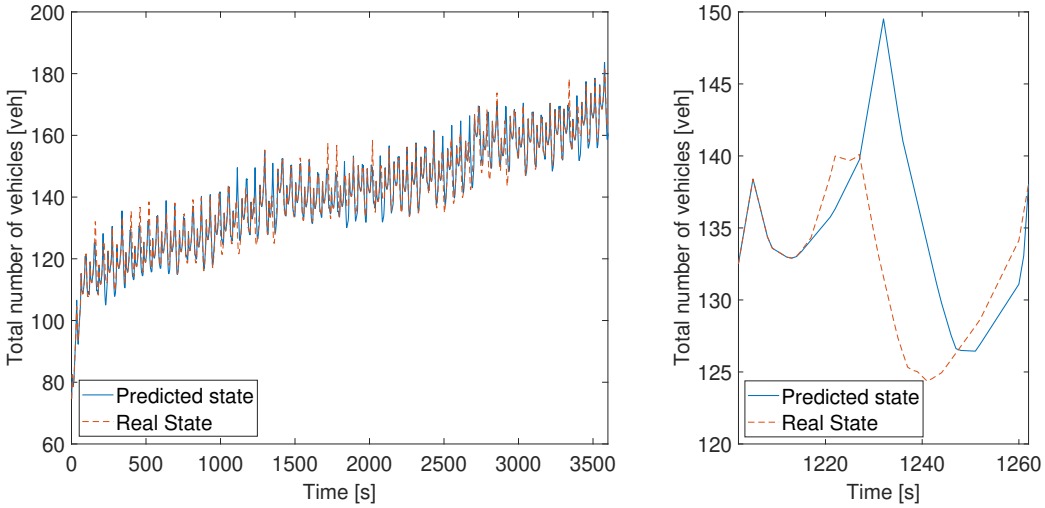


Figure 4-9: Simulation results of the framework with the second demand set (demand B: first fluctuating demand). The figure shows the reference state of the nominal system and the state of the real system with the influence of the RL adaptive law. On the left the states for the complete simulations are shown, on the right, the graph is zoomed in on one controller time step $T_c = 60$.

with respect to the fixed-time controller, computed using Equation (4-2). Corresponding results are shown in Figure 4-10, 4-11 and 4-12. The figures show the total number of vehicles present in the network at any given time.

In all three demand cases, the MPC controller, the RL-based controller and the framework reduce the waiting time with respect to the fixed-time controller. In the first two demand cases, the framework has the lowest TTS, in the last demand case the RL-based controller performs the best. It can be concluded that the framework performs better than the MPC controller in all cases. The TTS is reduced by $7.0\%, 4.1\%$ and 3.3% compared to the conventional MPC controller for the three demand sets respectively. This performance improvement compared to the MPC controller is caused by the addition of the RL adaptive law, which is designed to reject the disturbance in the network.

The conventional RL-based controller has better performance than the MPC controller in all the demand cases and than the framework in the third demand case. The fact that the RL-based controller performs better can be explained by the flexibility of the control input. For the MPC controller and the framework the cycle time is set to 60 s. The agent of RL-based controller, however, can adjust the phase length freely. By doing this, the controller has the potential to create a cycle offset between the two intersections which can benefit the system performance. With the current settings, the RL-based controller has the capability to perform better than all other controllers. When one would consider a variable cycle time for the MPC using the BLX model, the same result could theoretically also be achievable.

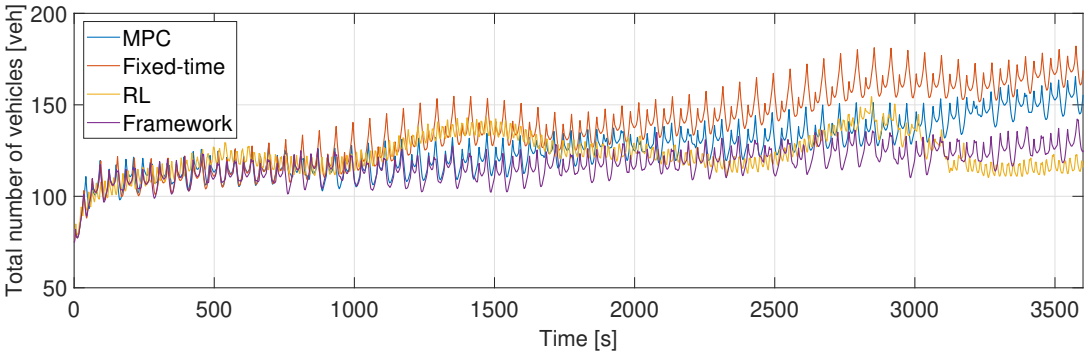


Figure 4-10: Number of vehicles in the network during demand A: the constant high demand scenario for all different controllers.

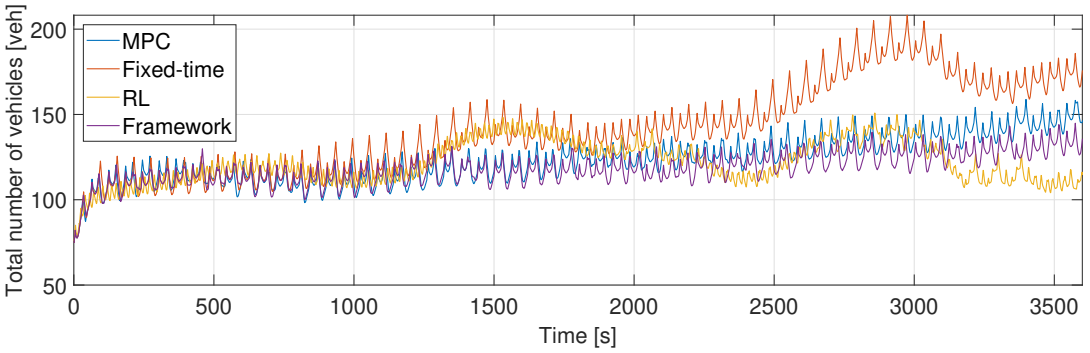


Figure 4-11: Number of vehicles in the network during demand B: the fluctuating demand 1 scenario for all different controllers.

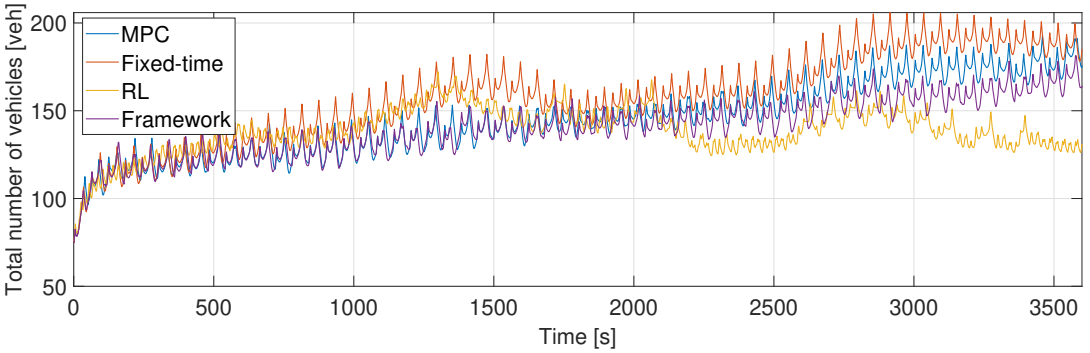


Figure 4-12: Number of vehicles in the network during demand C: the fluctuating demand 2 scenario for all different controllers.

Table 4-5: The system performance in terms of TTS and relative TTS of the different controllers for demand A: high demand.

Controller	TTS [veh/h]	Relative TTS [%]
Fixed-time control	137.6	-
MPC	125.2	-8.5
RL-based control	122.5	-10.9
Model-reference RL framework	117.1	-14.9

Table 4-6: The system performance in terms of TTS and relative TTS of the different controllers for demand B: fluctuating demand 1.

Controller	TTS [veh/h]	Relative TTS [%]
Fixed-time control	141.3	-
MPC	123.2	-12.8
RL-based control	121.6	-13.9
Model-reference RL framework	118.1	-16.4

Table 4-7: The system performance in terms of TTS and relative TTS of the different controllers for demand C: fluctuating demand 2.

Controller	TTS [veh/h]	Relative TTS [%]
Fixed-time control	157.0	-
MPC	145.1	-7.6
RL-based control	138.2	-12.0
Model-reference RL framework	140.3	-10.7

4-4 Case study II: SUMO implementation

In the previous section, we have discussed the results that have been achieved by the framework using the disturbed BLX model as a placeholder for the real system. We believe that more significant claims on the possible application of the framework in a real-life scenario can be made when deploying the framework on a system that represent real-life better than the mathematical model. This it thus the logical next step. To do so the software Simulation of Urban MObiliy (SUMO) [33] is used for simulating the traffic network.

4-4-1 Set-up adjustments

Some adjustments are made to the set-up of Section 4-1 to be able to run the SUMO simulation. The free-flow speed and the length of the vehicles can be set in SUMO. Besides, we define the entering speed at each source link (i.e. 1, 3, 6, 9, 11 and 14) to be 36 km/h in SUMO. This value is lower than the free-flow speed because the entering speed on the middle links (numbers 7 and 8), and the sink links (numbers 2, 4, 5, 10, 11 and 13) is also not equal to the free-flow speed.

Within SUMO the two traffic lights are set to be unregulated, which means that conflicting traffic of the opposite roads can cross each other without stopping and without a collision

happening. This specific setting is chosen such that a better match can be achieved between the BLX model and the traffic simulation of SUMO.

4-4-2 Parameter estimation

To match the BLX model that is used as a prediction model for the MPC controller with the specific behaviour of the traffic simulation in SUMO, some parameters of the model need to be estimated. For estimation two data sets are created from the SUMO simulation. The data sets contain data of the queue length of each lane and the number of vehicles on each road at every time step (1 s). They are obtained by simulating the system that is described in Section 4-1 for one hour, meaning that there are 3600 data points per stored variable. It is assumed that all turning rates are known and fixed. The demand patterns of both data sets are designed to capture different traffic behaviour. In both simulations, the cycle time of each intersection is constant and set to 60 s without any offset between the traffic signals. The cycle has an equal phase split, and with a yellow time of 5 s, this means each phase is 25 s long. One data set is used for parameter estimation, the other one is for validating purposes.

The parameters that are to be estimated are the length of each road l_{road}, the length of the vehicles in the network l_{veh}, the free-flow velocity v^{free} and the saturated flow rate of the road μ. These variables can be found in the equations of Section 2-3-1. The length of all source/sink links are considered the same, just like the length of link 7 and 8. The saturated flow rate for each lane in the network is considered to be equal to reduce the number of parameters to be estimated.

The model and the simulation is matched for a prediction horizon of 5 min, or 300 s, and a receding horizon with time steps of 50 s. In Matlab, the solver *lsqnonlin* is used with the levenberg-marquardt algorithm, which solves a nonlinear least-squares problem. The solver is run with 1000 initial conditions and the group of parameters with the lowest objective value are analysed to find the parameters that best describe the simulation. The performance indicator used for this analysis is the variance accounted for (VAF), which can be calculated by:

$$\text{VAF}_{(u,d)} = \left(1 - \frac{\text{var}(x_{(u,d)} - \hat{x}_{m,(u,d)})}{\text{var}(x_{(u,d)})}\right) \cdot 100\% \quad \forall (u,d) \in L \tag{4-6}$$

The VAF is only calculated for the number of vehicles in the network so $x_{(u,d)} = n_{(u,d)}$ as in Equation (2-19). The parameters in Table 4-8 are the best-performing ones. A VAF value higher than 80% has been achieved for all links in the network when simulated with the validation data set.

Table 4-8: The estimated parameters of the BLX model for the 2x1 grid based network, with $i \in \{7, 8\}$ and $j \in \{1, 2, 3, 4, 5, 6, 9, 10, 11, 12, 13, 14\}$ which are corresponding to the number of the links as in Figure 4-1.

Parameter	l_{veh} [m]	l^i_{road} [m]	l^j_{road} [m]	μ [veh/s]	v^{free} [m/s]
Value	17.9	454	485	1485	34

4-4-3 Training

The goal is to train the model-reference RL framework with the same settings as in case study I, with as an addition a variable cycle time of the MPC controller. Some training experiments are performed with the complete set-up. In the first episodes of each training an improvement in the reward function is observed, which is the result of the agent learning not to take infeasible actions. In this way the penalty for performing infeasible action (R_c as in Equation (3-20)) is omitted. However, no improvements are observed in the episodes after, meaning that the training with the complete set-up was not immediately successful.

To find the cause of the training dysfunction, a step-by-step approach is followed making the training incrementally more complex. In the most basic set-up we simplify the training in two ways. (1) A fixed control input is used as a placeholder for the MPC controller. This will accelerate the training process immensely in terms of computation time as no optimization must be performed. The fixed control input is used is to generate the reference states by means of the nominal model and is also used as baseline control input for the real system. Consequentially the controller time step of the RL adaptive law is equal to the cycle time of this fixed control input such that the states will always be retrieved at the same instance within the cycle. (2) Only one demand set is used for training. This will also accelerate the learning process but in terms of sample efficiency (and consequentially computation time), as the agent is encouraged to find a policy that fits the circumstances precisely. Furthermore, improvements between episodes will be more apparent as each episode is based on the same demand pattern.

The simplified set-up did not improve the learning process. In all instances, no significant improvements in the episode reward function are observed. To further investigate the obtained training results, a simulation is run where the RL adaptive law does not interfere with the control input of the baseline controller. Meaning that the action of the adaptive law is $u_{\text{rl}} = [0, 0]\ \forall t$. The episode reward of the system without action is obtained and used as a reference performance for training performance. The moving average value of the reward function over five episodes did never match or surpass the results of the no action value of the reward function during any training experiments.

Improvement efforts

Multiple changes to original design are implemented with the aim of achieving good training results using the SUMO software. A selection of the efforts for improving the dysfunctional framework is mentioned below. All these statements have been tested to a certain extent, but these attempts have not improved the training results. However, it can not be said with certainty that implementing one, or a combination, of these statements can not improve the training results of the designed framework. As already mentioned it is not trivial to exactly pinpoint the issue within a RL training process. The following improvement efforts are tested:

Changing the action space while considering two things: reducing the action space and at the same time including impactful actions within the action space.

Initially the action space of one intersection was: $-10 \leq a \leq 10, a \in \mathbb{Z}$. The size of the action space of a two intersection network is l_a^2, where l_a is the size of the actions space of one intersection. When one intersection allows for 21 different actions, this would result in a

size of the total action space for two intersections of $l_a^2 = 21^2 = 441$ actions. This number of actions is way too big. The goal is to reduce the size of the action space to below a hundred, while still including some impactful actions. An example of such a chosen action space where $a \in \{0, -10, -5, -3, -1, 1, 3, 5, 10\}$, which will have a size of $l_a^2 = 9^2 = 81$. This is the action space that has been used in all simulations. Although no direct improvement of the training process is observed in the SUMO implementation, the described action space is used because it has proven to work well for the fist case study.

Varying the algorithm settings, e.g. the hyperparameters concerning the learning process and the size of the neural network and experimenting with different RL algorithms.

From experience with the design of the algorithms for the conventional RL-based controller, and from various literature, it became apparent that the right choice of parameters can make or break the success of the learning process. Various combinations of hyperparameters have been tried, with the focus on varying the learning rate, the parameters concerning the exploration, the discount factor, and the size of the neural network. However, no extensive study on the choice of these parameters could be conducted due to the time-consuming learning process of the framework combined with (or without) MPC.

Furthermore, at the start of the research, the framework was designed such that the agent could supply a continuous action. DQN was not suitable for this purpose. Both the soft-actor critic (SAC) and twin-delayed deep deterministic policy gradient (TD3) algorithm were implemented. However, both had vital problems with the exploration method. The agent did not explore at all. DQN has shown to most potential to be the best performing RL algorithm for this purpose.

Changing the reward function such that not only negative rewards can be achieved, with as inspiration the reward function of the conventional RL-based controller.

A well-chosen reward function can be vital to the learning process of an RL algorithm. It is often recommended that a reward function does not only achieve negative immediate reward and that the learning process might be more successful when a positive reward can be achieved. This mostly has to do with an agent attempting to reach the terminal state faster to avoid an accumulation of the negative reward. In the case of the designed episodes for the urban traffic networks, this is not relevant as the episodes are designed to end after a certain time, not when a terminal state is reached. However, positive reward could still accelerate learning towards the state-action pairs that generate positive reward. The reward was changed such that error between the reference state and the real state between two controller steps was positive if the error increase and negative if the error decreased. It did not result in acceptable training, so this idea was not applied for other experiments.

4-4-4 Improvement suggestions

It cannot be said with certainty what is the solution that would solve the learning issues of the framework combined with SUMO. A few potential issues have been identified. They will be elaborated on below.

1. The nominal model differs too much from the system simulated in SUMO, which causes problems in following a reference signal designed by this model.

A theory why the implementation is not successful is that the nominal system differs too much from the SUMO simulation in order to sufficiently follow the desired reference state. The SUMO simulation contains a lot of uncertainties in the form of unmodeled dynamics compared to the nominal BLX model. These uncertainties are complex and differ over time. It is assumed that the RL adaptive law is not able to learn the specific behaviour of the SUMO simulation.

A possible solution is to redo the parameter estimation to achieve a better match between SUMO and the model. More parameters can be estimated. Also, the number of vehicles in the queue (a state in the BLX model) can be used as a performance indicator during the estimation process. Another option is to change the prediction model used in the framework with the goal of a better match between the SUMO simulation and the prediction model.

2. The designed task of following the states computed by the MPC controller exactly is too complex to complete.

A challenge that has arisen with using a model-reference approach in the context of urban traffic control has to do with the flexible property of the BLX model. The BLX model can set the cycle time of each cycle. This can increase the performance of the controller but is not optimal for the designed framework. The controller sampling time is fixed, while the cycle time is not. This has a big drawback: the phase switch always occurs at a different controller implementation period (e.g. it can happen at the beginning of the period, but also after 10 s). The difference in the number of vehicles per link between the model and the real system (i.e. the value on which the reward function is based) is very dependent on the time of the phase switch, as this is the tipping point of an increase vs. a decrease of vehicles on a lane. The difference between the states the prediction model and the real are measured either at the controller sampling time k_c, or at each time step k for the whole control implementation time T_c. In both cases, the obtained reward is very dependent on the time of the phase switch. Although the cycle time information is available for the RL agent, it could potentially be a problem for the RL agent that the value of the reward is not only linked to implemented RL action but is also very dependent on the current place within the cycle. There it would be more logical to use a fixed cycle-time as done in the case study in Section 4-3, such that the difference between the model and the real system (i.e. the reward) is always obtained in the same point in the cycle, namely the end of the cycle. However, using the BLX model, in this case, would be rather cumbersome. It would have been a better option to use the S-model as a prediction and nominal model in the framework with this setting, as the S-model is computationally faster and already has a fixed cycle time. Note that training efforts with fixed cycle time and SUMO have been made as well, unfortunately, this was not yet successful.

Another solution to deduce the complexity of the regulation problem would be to consider an easier reference to track. In the paper that inspired this thesis [56], there are two different baseline controllers, the first that generates a control input for the nominal prediction model and the second for the real system. The first controller is only updated with the states of the nominal model, this means that the reference path is already established at the start of the simulation. In the model-reference RL control framework that is combined with MPC, the MPC controller is updated with the states of the real system, as is usual in a rolling horizon scheme. This leads to a moving reference state, which could be a problem for the training performance of the framework. One could fix the reference at the beginning of the

simulation or, for examination purposes, set the reference state to be zero. This last solution would essentially make the problem an optimal control problem, and we believe that this could also be a good control framework approach in a situation where reference tracking is not important. Section 5-2 contains an elaboration on the suggestion of implementing this change.

3. The settings of the hyperparameters that have been tried were not optimal for achieving good training results.

As discussed, the hyperparameters of the RL algorithm play a big role in the training progress. For an effective learning process, it is important to select the hyperparameters of the RL algorithm for learning and exploration carefully. Although some tuning has been done to achieve sufficient learning, it is recommended to perform a systematic tuning procedure. The learning process is to be repeated systematically for a selected range of different hyperparameters. To fairly compare the behaviour of the parameters, each experiment must be repeated multiple times, or the ANN must be initialized likewise for each set of hyperparameters.

4-5 Conclusions

In case study I, the model-reference RL framework was compared to a fixed-time controller, a conventional MPC controller and a conventional RL-based controller on their system performance through the total time, spend. We have seen that the model-reference RL framework can be successfully implemented on a small and simplified urban traffic network and that the framework is able to successfully reject a sinusoidal disturbance in the demand. The framework has been shown to outperform the fixed-time controller and the MPC controller in all simulations.

Besides, the training progress of the model-reference RL framework was compared to the training of the agent of the conventional RL-based controller. We have seen that the framework has a better performance than the conventional RL controller during the entire learning process. However, the sample efficiency in terms of episodes needed for convergence of algorithm used for the RL-based controller is better than the sample efficiency of the framework. This is due to different the control time step of the RL-based controller and the framework, which are 1 s and 60 s respectively.

In case study II the framework is implemented and simulated using the traffic simulation software SUMO. It has appeared that the RL agent of the adaptive law is not able to find a policy that improves the tracking ability of the framework. In this chapter we have discusses the measures that have been taken to improve the training ability of the RL agent. Besides, some potential vital problems are established as well as suggestions for solutions to these problems.

Chapter 5

Conclusions

The effective control of signals in urban traffic networks is the most efficient solution in the process of solving the problems that accompany a delayed throughput of traffic in an urban traffic system. Reinforcement learning (RL) and model predictive control (MPC) are both proven to achieve good results in urban traffic control studies. However, real-world implementation is still a challenge, as for the large computation time of an MPC controller and the bad learning performance of an RL controller. In this thesis, the novel model-reference RL controller combined with MPC is developed for an urban traffic problem. In this chapter we will conclude the assessed results.

5-1 Concluding remarks

The research focuses on the combination of reinforcement learning and model predictive control in the context of urban traffic signal control. The goal of this research is defined as:

> *To design a model-reference control framework that combines MPC and RL for centralized control of an urban traffic control network and compare its performance with a conventional fixed-time controller, MPC controller and RL-based controller.*

To see if the design is successful, two main sub-questions were asked that have been tested by two case studies. We will now assess these questions.

1. *Can the framework outperform a conventional MPC controller when presented with disturbances and/or uncertainties in terms of system performance?*

   In case study I, we have seen that by introducing the RL adaptive law the framework is able to follow the desired reference state that is defined by the baseline control input of the MPC controller very well. The results show that the model-reference RL framework could outperform the conventional MPC controller and the fixed-time controller in terms of system performance expressed in the total time spent (TTS). Besides, the

performance of framework the framework is very similar to the performance of the RL-based controller. The system performance results of the framework are very promising because the RL-based controller has a lot more freedom in the choices of control input. It has the potential to perform better than the three other assessed controllers.

As regards the comparison to the MPC controller, the framework outperformed the MPC controller with $7.0\%, 4.1\%$ and 3.3% respectively for the three demand cases. It can be concluded that the framework is capable of outperforming the MPC controller in terms of system performance when presented with demand disturbance. This can only be concluded for repetitive disturbance patterns for which the RL adaptive law of the framework have been trained.

2. *Can the framework outperform a conventional RL-based controller in terms of sample efficiency and its performance during the learning process?*

   In the first case study, we have seen that the framework has a very good performance during training already from the first episode. The RL-based controller needs a lot of training before it will reach performance that is acceptable, let alone good. The framework is, therefore, easier implementable in a real-world setting, because training or the RL adaptive law will not compromise the system performance. The main reason for this result is that the baseline control input of the MPC controller already provides a well-performing stable control input. The RL adaptive law that adjusts this input can not worsen the performance of the system significantly.

   With regard to sample efficiency: with the settings that are used in this case study, no improvements have been made with accelerating the convergence of the framework in comparison to the RL-based controller. This can be mainly explained by the sampling time of each controller. The framework supplies a new action every 60 s, while the RL-based algorithm provides an action every 1 s. In one episode of 3600 s, the RL-based controller has gained a lot more experience than the framework as it had performed sixty times more actions. The framework, therefore, needs fewer samples to obtain convergence, but a lot more episodes are needed to train on the same amount of data samples. In the context of urban traffic control, these settings for the sampling time of both agents were needed for the design of well-performing controllers. Although the agent has the potential of being more sample efficient in a system where the sampling time of the controller is equal, this is not the case in this research.

We have seen that in case study I the model-reference RL framework has performed successfully and multiple hypotheses have been confirmed: The framework outperforms the conventional MPC controller and the fixed-time controller in terms of systems performance (i.e. in terms of TTS). The framework also outperforms the conventional RL-based controller in terms of performance during the learning process because the framework performs well in every stage of learning. The model-reference RL frameworks is, however, not able to outperform the conventional RL-based controller in terms of sample efficiency in the specific case study of this thesis.

From the complete case study set-up with SUMO the hypothesis cannot be confirmed. In the second case study we have not yet achieved good training results. This does not necessarily mean that it is not possible to achieve good results in a more complex simulation environment using this framework. The results are inconclusive. Some ineffective measures have already

been taken in trying to improve the performance of the RL agent. Besides, some improvement suggestions are provided. The main concern is the mismatch between the prediction model and the simulated system, which can potentially be solved by better parameter estimation or using a more advanced model.

5-2 Recommendations for future work

Some suggestions for future work on the topic of this thesis are:

Improvements to the framework The main recommendation would be to improve the framework in the context of urban traffic signal control to be able to implement it using a simulation program such as SUMO. Most recommendations are given in Section 4-4-4. Here we summarize these and some other suggestions:

- Generate a better match between simulation and mathematical model,
- Tuning the hyperparameters,
- Tuning of state and/or reward function,
- Proof on concept on an even simpler network (i.e. single intersection).

MPC with RL tuning In this thesis we have combined MPC and RL in a model-reference adaptive control framework. There are also other ways to combine MPC with RL, as this has the potential to be a very powerful tool for complex control problems. Sometimes the use of a reference state is not the most logical choice in urban traffic signal control. It is therefore proposed for further research to investigate a similar framework, but without using a reference model. The main difference will be the definition of the reward function of the RL part of the framework, which will become some kind of performance indicator, e.g. similar to the objective of the MPC controller. We have seen that the definition of the reward function is vital to the success of an RL controller. It is therefore essential to choose and tune the correct reward function. The control scheme for MPC with RL tuning is depicted in Figure 5-1.

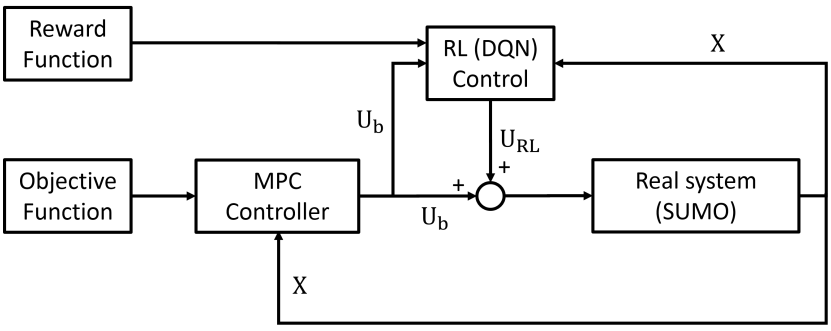


Figure 5-1: The proposed control scheme for MPC with RL tuning of the control law.

Larger network The network considered in the case study in this thesis consists of two intersections. The possibility of studying a larger network has also been considered, namely a network consisting of four intersections. When enlarging the network the size of the action and state-space grow exponentially. It is not possible to keep using DQN as an RL algorithm as this algorithm can only handle a discrete action space. An alternative algorithm that is able to handle a continuous action space has to be used, e.g. a soft actor-critic (SAC) agent [16] or twin-delayed deterministic policy gradient (TD3) agent. Some experiments have been run with a network with four intersections and both of the mentioned agents without any success due to vital problems of the exploration process during the training of both agents. It is essential to do some further investigation in RL agents that can be combined with continuous action.

Using another baseline controller The MPC controller within the framework can in principle be replaced by any other baseline control input. With the aim of performance conservation, it is recommended to replace the normal MPC controller with a parametric control law, such as designed by Jeschke and De Schutter [21]. The parametric control law in this paper is designed for the S-model, but can be adapted to fit the BLX model as well. The choice for the S-model as a prediction model and/or nominal model in the framework can also be made, making it easier to implement the already designed parametric control law.

A study on computation time of the control system A potential advantage of the model-reference RL combined with MPC compared to a conventional MPC controller is a reduction in computation time. It is recommended to investigate ways to reduce the computation time of the framework compared to a conventional MPC controller. This can for example be achieved by adjusting the prediction model of the MPC controller for reduced computational effort or considering a parameterized control law.

Disturbance When further investigating the potential of the model-reference RL framework, it would be really interesting to include disturbance patterns that better resemble a real-world setting. This can for example be done by using real data for an existing intersection and the corresponding prediction data.

Other applications We have already seen that the model-reference RL framework works really well in the context of surface vehicles [56] and has shown some promising results in an urban traffic network. It is recommended to implement a similar framework for other applications, preferably applications that can be logically described as a regulation problem.

Appendix A

Reinforcement learning algorithms

A-1 DQN for conventional RL-based control

Algorithm 1 RL algorithm using DQN

procedure TRAIN
  Initialize experience buffer
  Initialize network Q with random parameters ϕ
  Initialize target network $\hat{Q}$ with $\hat{\phi} = \phi$
  Initialize $\epsilon = 1$
  loop
    $s \Leftarrow s_0$
    $t = 0$
    while $t < t_{\text{end}}$ **do**
      Chose action a using ϵ-greedy
      Take action a, observe next state s' and reward r
      Store (s, a, r, s') in experience buffer
      Sample random mini-batch M of (s, a, r, s') from experience buffer
      Update network Q across all samples experience
      if Target update frequency **then**
        Update target network $\hat{Q} = Q$
      end if
      Update ϵ based on δ_ϵ
      $s \Leftarrow s'$
      $t = t + 1$
    end while
  end loop
end procedure

A-2 DQN for the model-reference RL framework

Algorithm 2 Model-reference RL algorithm using DQN

procedure TRAIN
  Initialize experience buffer
  Initialize network Q with random parameters ϕ
  Initialize target network $\hat{Q}$ with $\hat{\phi} = \phi$
  Initialize $\epsilon = 1$
  loop
    $s \Leftarrow s_0$
    $t = 0$
    while $t < t_{\text{end}}$ **do**
      MPC: Solve optimization to find $T_{\text{cyc},d}$ and $\pi_{\text{green},d}$
      Chose action a using ϵ-greedy
      if Action is infeasible **then**
        Add penalty to reward
        Change action accordingly
      end if
      Take action a, observe next state s' and reward r
      Store (s, a, r, s') in experience buffer
      Sample random mini-batch M of (s, a, r, s') from experience buffer
      Update network Q across all samples experience
      if Target update frequency **then**
        Update target network $\hat{Q} = Q$
      end if
      Update ϵ based on δ_ϵ
      $s \Leftarrow s'$
      $t = t + 1$
      MPC: update active constraints and environment states
    end while
  end loop
end procedure

Bibliography

[1] B. Abdulhai, R. Pringle, and G. J. Karakoulas. Reinforcement Learning for True Adaptive Traffic Signal Control. *Journal of Transportation Engineering*, 129(June):278–285, 2003.

[2] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Processing Magazine*, 34(6):26–38, nov 2017.

[3] M. Barth and K. Boriboonsomsin. Traffic congestion and greenhouse gases. *Access*, (35):2–9, 2009.

[4] N. Bhouri, F. Balbo, and S. Pinson. An agent-based computational approach for urban traffic regulation. *Progress in Artificial Intelligence*, 1(2):139–147, 2012.

[5] F. Boillot, J. M. Blosseville, J. B. Lesort, V. Motyka, M. Papageorgiou, and S. Sellam. Optimal Signal Control of Urban Traffic Networks. In *Proc. 6th IEE Int. Conf. Road Traffic Monitoring and Control*, pages 75–79, 1992.

[6] L. Buşoniu, R. Babuška, B. D. Schutter, and D. Ernst. *Reinforcement learning and dynamic programming using function approximators*. CRC Press, 2010.

[7] E. F. Camacho and C. Bordons. *Model Predictive control*. Advanced Textbooks in Control and Signal Processing. Springer London, London, second edition, 2007.

[8] CBS. Welke sectoren stoten broeikasgassen uit? https://www.cbs.nl/nl-nl/dossier/dossier-broeikasgassen, 2019.

[9] H. Chen, S. L. Cohen, N. H. Gartner, and C. C. Liu. Simulation Study of OPAC: A Demand-Responsive Strategy for Traffic Signal Control. In N. H. Gartner and N. H. M. Wildon, editors, *Transportation and Traffic Theory*, pages 233–249. Elsevier Science Publishing Co., Inc., 1987.

[10] B. De Schutter, H. Hellendoorn, A. Hegyi, M. van den Berg, and S. K. Zegeye. Model-based Control of Intelligent Traffic Networks. In *Chapter 11 in Intelligent In- frastructures (R.R. Negenborn, Z. Lukszo, and H. Hellendoorn, eds.)*, volume vol. 42 of, pages 277–310. Springer Netherlands, Dordrecht, 2010.

[11] Y. Du, W. Shangguan, D. Rong, and L. Chai. RA-TSC: Learning Adaptive Traffic Signal Control Strategy via Deep Reinforcement Learning. *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019*, pages 3275–3280, 2019.

[12] S. El-Tantawy, B. Abdulhai, and H. Abdelgawad. Design of reinforcement learning parameters for seamless application of adaptive traffic signal control. *Journal of Intelligent Transportation Systems: Technology, Planning, and Operations*, 18(3):227–245, 2014.

[13] European Commission. Urban Mobility. https://ec.europa.eu/transport/themes/urban/urban_mobility_en, 2019.

[14] X. Glorot, A. Bordes, and Y. Bengio. Deep Sparse Rectifier Neural Networks. In *14th international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[15] D. Görges. Relations between Model Predictive Control and Reinforcement Learning. *IFAC-PapersOnLine*, 50(1):4920–4928, jul 2017.

[16] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *arXiv:1801.01290v2*, 2016.

[17] J. Henry, J. Farges, and J. Tuffal. The PRODYN Real Time Traffic Algorithm. *IFAC Proceedings Volumes*, 16(4):305–310, 1983.

[18] L. Hewing, K. P. Wabersich, M. Menner, and M. N. Zeilinger. Learning-Based Model Predictive Control : Toward Safe Learning in Control. *Annu. Rev. Control Robot. Auton. Syst.*, 3:269–296, 2020.

[19] K. Hornik, M. Stinchcombe, and H. White. Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*, 2:359–366, 1989.

[20] P. A. Ioannu and J. Sun. *Robust Adaptive Control*. Upper Saddle River: Prentice-Hall Inc., 1996.

[21] J. Jeschke and B. De Schutter. Parametrized Model Predictive Control Approaches for Traffic Networks. *IFAC PapersOnLine*, 54(2):284–291, 2021.

[22] E. Jonkers and B. van Berne. Economische Wegwijzer 2019. Technical report, Panteia, 2019.

[23] S. Kachroudi and S. Mammar. The effects of the control and prediction horizons on the urban traffic regulation. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, pages 267–272, 2010.

[24] Kennisinstituut voor Mobiliteitsbeleid. Kerncijfers Mobiliteit 2020. Technical report, Ministerie van Infrastructuur en Waterstaat, 2020.

[25] F. Kessels. *Traffic Flow Modelling - Introduction to Traffic Flow Theory Through a Genealogy of Models*. Springer, 2019.

[26] E. Lavretsky and K. A. Wise. *Robust and Adaptive Control - With Aerospace Applications*. Springer, 2013.

[27] Y. Lecun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–444, 2015.

[28] F. L. Lewis, D. Vrabie, and K. G. Vamvoudakis. Reinforcement Learning and Feedback Control: Using Natural Decision Methods to Design Optimal Adaptive Controllers. *IEEE Control Systems*, 32(6):76–105, dec 2012.

[29] M.-T. Li and A. C. Gan. Signal Timing Optimization for Oversaturated Networks Using TRANSYT-7F. *Transportation Research Record: Journal of the Transportation Research Board*, 1683(1):118–126, jan 1999.

[30] L. J. Lin. Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching. *Machine Learning*, 8(3):293–321, 1992.

[31] S. Lin. *Efficient model predictive control for large-scale urban traffic networks*. PhD thesis, 2011.

[32] S. Lin, B. De Schutter, Y. Xi, and H. Hellendoorn. Study on fast model predictive controllers for large urban traffic networks. In *12th International IEEE Conference on Intelligent Transportation Systems*, volume 19, pages 961–696, 2009.

[33] S. Lin and Y. Xi. *An Efficient Model for Urban Traffic Network Control*, volume 41. IFAC, 2008.

[34] J. D. Little, M. D. Kelson, and N. H. Gartner. Maxband: a Program for Setting Signals on Arteries and Triangular Networks. *Transportation Research Record*, (795):40–46, 1981.

[35] P. Mannion, J. Duggan, and E. Howley. An Experimental Review of Reinforcement Learning Algorithms for Adaptive Traffic Signal Control. In *Autonomic Road Transport Support Systems*, pages 47–66. Springer International Publishing, Cham, 2016.

[36] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv:1312.5602v1*, pages 1–9, dec 2013.

[37] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[38] N. T. Nguyen. *Model-Reference Adaptive Control*. Advanced Textbooks in Control and Signal Processing. Springer International Publishing, Cham, 2018.

[39] M. Nielsen. *Neural Networks and Deep Learning*. http://neuralnetworksanddeeplearning.com/, 2019.

[40] M. Papageorgiou, C. Kiakaki, V. Dinopoulou, A. Kotsialos, and Yibing Wang. Review of road traffic control strategies. *Proceedings of the IEEE*, 91(12):2043–2067, dec 2003.

[41] D. I. Robertson and R. D. Bretherton. Optimizing Networks of Traffic Signals in Real Time-The SCOOT Method. *IEEE Transactions on Vehicular Technology*, 40(1):11–15, 1991.

[42] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[43] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, (November), 2016.

[44] S. Sen and L. Head. Controlled Optimization of Phases at an Intersection. *Transportation Science*, 31:5–17, 1997.

[45] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. V. D. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7585):484–489, 2016.

[46] A. G. Sims and K. W. Dobinson. The Sydney Coordinated Adaptive Traffic (SCAT) System Philosophy and Benefits. *IEEE Transactions on Vehicular Technology*, 29(2):130–137, 1980.

[47] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An introduction*. Cambridge, MA : The MIT Press, 2018.

[48] R. S. Sutton, A. G. Barto, and R. J. Williams. Reinforcement learning is direct adaptive optimal control. *IEEE Control Systems*, 12(2):19–22, apr 1992.

[49] S. Theodoridis, K. Koutroumbas, and C. D. E. Spyropoulos. Pattern Recognition and Neural Networks. In *Machine Learning and its Applications*, pages 169–195. Springer, 2001.

[50] M. van den Berg, A. Hegyi, B. De Schutter, and J. Hellendoorn. A macroscopic traffic flow model for integrated control of freeway and urban traffic networks. In *Proceedings of the 42nd IEEE Conference on Decision and Control*, pages 2774–2779, Maui, Hawaii, 2003.

[51] C. Watkins and P. Dayan. Technical Note Q-Learning. *Machine Learning*, 8(3-4):279–292, 1992.

[52] C. J. C. H. Watkins. Learning from delayed rewards, 1989.

[53] Webster. Traffic Signals Webster. *Road Research Technical Paper*, (39):1–44, 1958.

[54] M. Wiering and M. van Otterlo. Reinforcement Learning State-of-the-Art. In *Adaptation, Learning, and Optimization Volume 12*, pages 613–630. Springer, 2012.

[55] B. L. Ye, W. Wu, K. Ruan, L. Li, T. Chen, H. Gao, and Y. Chen. A survey of model predictive control methods for traffic signal control. *IEEE/CAA Journal of Automatica Sinica*, 6(3):623–640, 2019.

[56] Q. Zhang, W. Pan, and V. Reppa. Model-Reference Reinforcement Learning Control of Autonomous Surface Vehicles with Uncertainties. *arXiv preprint arXiv:2003.13839*, mar 2020.

List of abbreviations

ANN	Artificial neural network
DQN	Deep Q-network
MDP	Markov decision process
MPC	Model predictive control
MRAC	Model-reference adaptive control
RL	Reinforcement learning
SUMO	Simulation of Urban MObility
TD	Temporal-difference
TTS	Total time spent