Optimal Cooperative Intersection Control

F.M.G. Creemers

DC 2017.002

Master's thesis

Coach(es): dr.ir. A.A.J. Lefeber
ir. A.I. Morales Medina

Supervisor: prof.dr.ir. N. Van de Wouw

Eindhoven University of Technology
Department of Mechanical Engineering
Dynamics and Control Group

Eindhoven, January, 2017

Summary

Road intersections play a significant role in traffic congestion and traffic accidents, especially in urban environments. Both these problems can be solved by automating vehicles and road intersections. Autonomous vehicles have already been introduced to a certain extent, and research on automated intersections which coordinate the vehicles through the intersection is ongoing. However, current research on automated intersections often either does not optimise the sequence in which vehicles cross the intersection, or does so in a way that is rather computationally expensive. This report aims to design a high-level controller which optimises the crossing sequence for an existing low-level vehicle controller called Cooperative Intersection Control.

First, a high-level, abstracted model of the intersection is developed, which bears resemblance to queueing systems. The model is formulated as a hybrid system, which evolves both in continuous-time flow and in discrete-time events. This hybrid system relates the entry of individual vehicles into the intersection to a measure of the delay that they experience, and is used in the design of a high-level controller using model predictive control. The objective of the controller is to minimise the number of vehicles in the queues, thereby minimising their delay. Constraints are placed on the optimisation problem to ensure that a safe distance is maintained between vehicles, and also to make the decisions of the controller fair from the driver's perspective. Both the model and the controller are implemented in Simulink and simulations are performed. It is shown that the model predictive controller achieves a steady-state cycle that is similar to what is expected from results in queueing theory. A comparison between the model predictive controller, a first-come-first-serve policy and a vehicle actuated traffic light show that the MPC controller outperforms the other methods during the transient trajectory. For a low utilisation, the steady-state results are similar for all three control methods. In the case of a high utilisation, the MPC controller outperforms both the FCFS policy and the traffic light control scenario.

Nomenclature

c	Weighting coefficient
J	Cost
j	Event counter
k	Time-step
m	Queue activity
N_p	Prediction horizon
N_w	Moving average window length
n_l	Number of queues
q	Queue number
T^s	Service time
t	Time
t_c	Inter-departure time
t_w	Inter-arrival time
u	Control input
V	Velocity
w^a	Vehicle arrivals
x	Queue length
y	Output
z	State
δ_{des}	Desired virtual inter-vehicle distance
λ	Average arrival rate
μ	Service rate
ρ	Server utilisation
σ	Set-up time
ϕ	Solution of the system
$\mathcal{C}$	Flow set
$\mathcal{D}$	Jump set
$\mathcal{Q}$	Index set of all queues
$\mathcal{U}$	Input set
$\mathbb{N}_0$	Set of natural numbers including zero
$\mathbb{R}_{\geq 0}$	Set of non-negative real numbers

Contents

Chapter 1

Introduction

As a consequence of population growth and the rapid increase in motorisation and urbanisation, traffic congestion occurs more often. This results in increased delays in the transportation of people and goods. Moreover, since vehicles frequently need to accelerate and slow down in dense traffic, traffic congestion also leads to increased fuel consumption and environmental pollution. In an urban environment, traffic congestion typically occurs at road intersections, where vehicles with crossing trajectories compete for access to the intersection. This is currently regulated through stop signs, roundabouts and traffic lights, all of which require vehicles to wait for their turn to cross the intersection. This obstructs the flow of vehicles and limits the throughput of such intersection, leading to larger delays for users of the intersection. Meanwhile, traffic intersections also account for a significant part of traffic accidents, since drivers are easily distracted or can make mistakes which lead to a collision. In the previous decade, just over 20% of traffic fatalities in the EU occurred at intersections [1]. The safety and efficiency of intersections can be improved by automating both individual vehicles and road intersections as a whole.

1.1 Background

Technical advances have already made it possible, to a certain extent, to construct fully autonomous vehicles capable of navigating through traffic. Through the use of sensory technology, wireless communication and positioning systems, these vehicles are able to perceive their surroundings and communicate with each other and with traffic infrastructure. It is anticipated that in the future many, if not most, vehicles will be autonomous. Current research efforts are directed to automating intersections in order to increase their efficiency, in terms of throughput, and safety. Existing methods to automate intersections can be classified as being centralised or distributed. Centralised methods rely on a coordination unit which communicates with the vehicles in the intersection and makes decisions centrally [2]. Distributed methods involve only communication between the vehicles. In the latter case, each vehicle makes its decisions locally, and communication between the vehicles allows them to negotiate with each other to determine a crossing sequence. In some cases, a mixed approach is used, where a temporary unit is selected to temporarily coordinate the intersection traffic [2].

Two of the current major methods for automated intersections are cooperative resource reservation methods and trajectory planning methods [2]. Cooperative resource reservation methods, such as in [3], split the intersection into tiles. Vehicles then need to reserve the tiles on their trajectory for certain time slots. In general, reservations are taken by a central unit, which accepts reservations on a first-come-first-serve basis. If some of the requested tiles are taken, the request is rejected and the vehicle needs to make a new request. The crossing sequence is not optimised, though [4] proposes an auction-based method. A distributed version of the reservation protocol is proposed in [5], where vehicles claim a reservation and cancel it when they have passed the intersection.

Trajectory planning methods use the fact that vehicles follow a given trajectory, and aim to plan and control the motion of a vehicle along that trajectory. In [6], a distributed scheduling algorithm

is proposed which determines all possible crossing sequences and executes the sequence which takes the shortest time. A centralised solution is proposed in [7], where traffic flows on different lanes are grouped together based on safe patterns. At each scheduling step, one group is chosen to pass the intersection. A different method is proposed in [8, 9], where fixed paths are assumed and the acceleration of the vehicles is controlled to achieve a safe crossing based on conflict zones. An unfortunate downside of these trajectory planning methods is that they tend to rely on simulating the motion of the vehicles through the intersection in some form, which results in high computational costs.

Unfortunately, most of the methods created so far are lacking when it comes to optimising the crossing sequence and thus do not achieve an optimal traffic flow, or are very computationally expensive. This report aims to improve an existing intersection controller to achieve an optimal crossing sequence. The intersection controller used in this report is Cooperative Intersection Control by [10], which uses virtual platooning to control the dynamical behaviour of the vehicles in a simple and computationally cheap way. The system architecture is decoupled into a vehicle controller, and a controller which determines the crossing sequence. The latter controller currently uses a first-come-first-serve policy, and improving on this is the focus of this report.

1.1.1 Cooperative intersection control

Cooperative Intersection Control (CIC) is a method of automating road intersections for autonomous vehicles proposed by Morales *et al.* in [10]. CIC is applied in a region with radius R around the centre of the intersection, which is called the Cooperation Zone (CZ), an example of which is shown in Figure 1.1. A safe crossing through the intersection is achieved by means of virtual platooning of the vehicles. Every vehicle that enters the CZ is assigned a target vehicle which it must follow, and a desired virtual inter-vehicle distance is defined between them. This allows vehicles approaching the intersection from different directions to form a virtual platoon, where each vehicle closely follows the vehicle in front of it. Each vehicle maintains its virtual inter-vehicle distance through Cooperative Adaptive Cruise Control, which relies on inter-vehicle communication. The combination of CACC and virtual inter-vehicle distance is referred to as Virtual CACC (VCACC). The system architecture of CIC is shown in Figure 1.2.

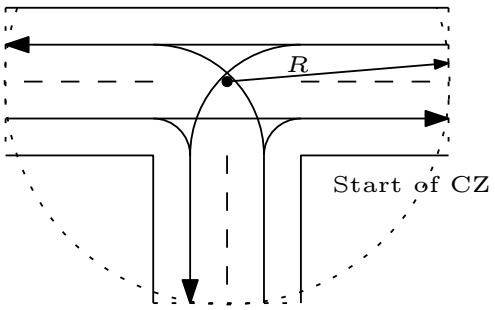

Figure 1.1: *An example of an intersection with different trajectories, controlled using CIC.*

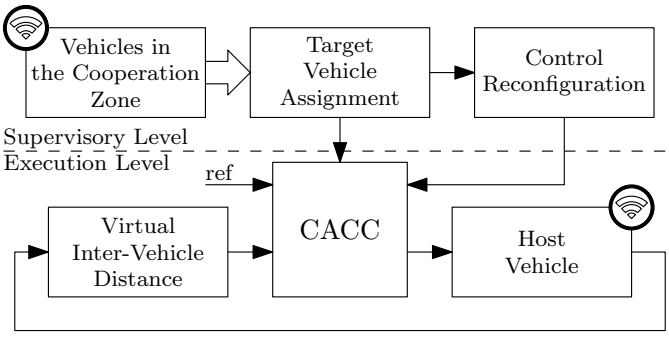

Figure 1.2: *The system architecture of cooperative intersection control in [10].*

As shown in Figure 1.2, Cooperative Intersection Control consists of a supervisory-level controller and an execution-level controller. When an incoming vehicle enters the CZ, the supervisory-level controller determines that vehicle's priority and assigns it a target vehicle. In this way, the supervisory controller creates one or more virtual platoons of vehicles, where multiple platoons are formed as long as they do not cross paths. In [10], the supervisory controller assigns priority on a first-come-first-serve (FCFS) basis. After a vehicle has received its target vehicle, the execution-level controller determines its virtual inter-vehicle distance by performing certain coordinate transformations. Since vehicles can have different trajectories, the virtual inter-vehicle distance is calculated by comparing the two vehicles' travelled distance along their scaled trajectories. The vehicles' dynamics are then controlled through VCACC in order to reach a reference virtual inter-vehicle distance.

1.2 Research objective and approach

In its current form, Cooperative Intersection Control lacks any optimisation when it comes to assigning vehicles their priority, and instead uses a simple first-come-first-serve policy. A supervisory controller which assigns the priority of vehicles in some optimal way can further increase the performance of CIC. In this case, the performance of the intersection is measured through the average delay of the vehicles, which must be minimised. Thus, the main objective of this report is *to design and implement a high-level supervisory control algorithm which, in conjunction with the execution-level controller of CIC in [10], minimises the delay of autonomous vehicles travelling through a single generalised, isolated intersection, while also being fair from the driver's perspective.*

For the purposes of this report, the supervisory controller is viewed in isolation, i.e., without the execution-level controller and with simplified vehicle dynamics. It is assumed that the execution-level controller is implemented and functions correctly. The physical movement of the vehicles does not come into play in this report. The whole of the Cooperation Zone and the vehicles in it are viewed simply as a black box. Vehicles wait outside the CZ, and the supervisory controller manages the access of those vehicles to the CZ. An abstracted model of the intersection and the vehicles in the CZ is designed in Chapter 2, which relates the inflow of vehicles into the intersection to some measure of the average delay. Vehicle arrivals are seen as an external disturbance to this model. The supervisory controller then performs its optimisation based on the current state of the abstract intersection model, as described in Chapter 3. The system architecture used in this report is depicted schematically in Figure 1.3.

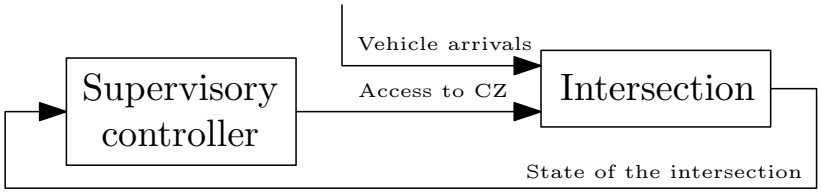

Figure 1.3: *The simplified system architecture used in this report.*

The entire execution-level of Figure 1.2 is contained in the intersection block in Figure 1.3. The supervisory controller in Figure 1.3 manages when arriving vehicles can access the CZ, which is akin to the target vehicle assignment in Figure 1.2.

1.3 Outline of the report

This report consists of three parts. In Chapters 2 and 3, the intersection is modelled and the supervisory-level controller is designed, respectively. Chapter 4 presents their numerical implementation, after which simulation results are presented in Chapter 5. A concise description of each chapter is given below.

Chapter 2 presents the abstracted, high-level model of the intersection and the vehicles in the CZ. The vehicle dynamics of the execution-level controller are simplified. The formulation of the intersection model necessitates certain constraints on the supervisory controller, which are also presented.

Chapter 3 formulates the supervisory controller using model predictive control. A prediction model is created from the abstracted intersection model of the previous chapter. The arrival of vehicles at the intersection is included as a disturbance model, which increases the accuracy of the predictions.

Chapter 4 gives the implementation of the abstracted intersection model and the supervisory controller in Simulink. The nature of the intersection model and the controller require some considerations in their numerical implementation, which are elaborated upon.

Chapter 5 presents simulation results of the Simulink model, in which the behaviour and performance of the supervisory controller is examined. The simulation results are compared with existing theory on queueing systems, and the performance of the supervisory controller is compared with two other control methods.

Finally, **Chapter 6** concludes the report with a summary and the general conclusions of all chapters and gives suggestions for future research.

Chapter 2

Intersection model

The first step in designing a supervisory controller which optimises the flow of vehicles through the intersection is to model that intersection. In an intersection automated through Cooperative Intersection Control, different phenomena can be observed. Every vehicle which enters the Cooperation Zone is assigned a target vehicle and adjusts its own velocity in order to follow that vehicle at a desired virtual inter-vehicle distance, thus forming a virtual platoon. While the individual vehicles' behaviours are fairly simple in this way, when combined, the behaviour of all the vehicles in the intersection becomes quite complex. Based on their trajectories, the vehicles can form multiple different platoons. If vehicles arrive at a fast rate, they may need to slow down to a crawl in order to maintain their virtual inter-vehicle distance, even if there are no vehicles directly in front of them, creating phantom traffic jams.

Rather than taking all these phenomena into account, using an abstraction of the intersection and the behaviour of the vehicles in it is more desirable. Therefore, the supervisory controller considers a dynamical model where everything in the Cooperation Zone is seen as a black box which vehicles enter and later exit. In this high-level intersection model, the supervisory controller manages the access to the Cooperation Zone. Arriving vehicles wait in queues just outside the CZ until the supervisory controller allows them to depart from their queues. The vehicle dynamics are simplified; once a vehicle departs from its queue, it instantly moves forward with a constant velocity V. One or more queues exist per direction of approach to the intersection, depending on the physical layout of the intersection, and vehicles in individual queues depart from their queues on a First-In-First-Out (FIFO) basis. Figure 2.1 shows a schematic view of this high-level intersection.

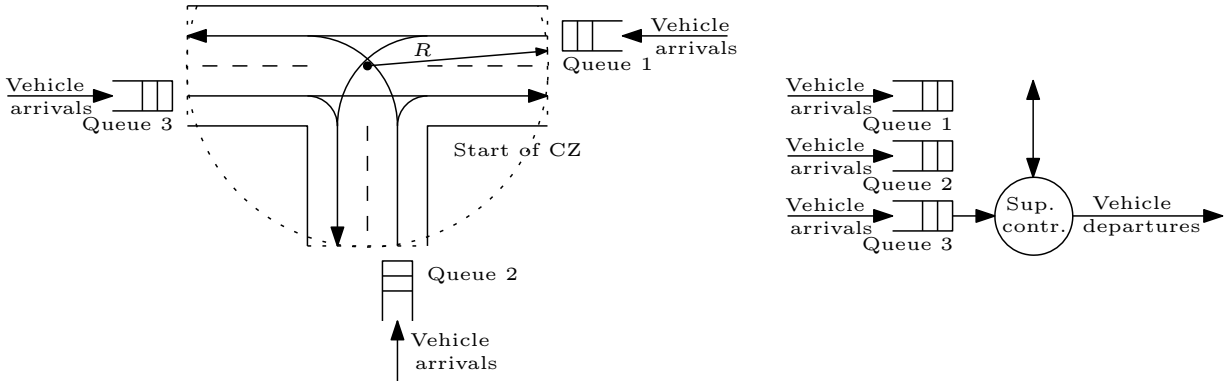

Figure 2.1: *A schematic depiction of the intersection on the left, where vehicles wait in queues outside the Cooperation Zone. The right shows the same intersection modelled as a queueing system, where the server must choose which queue to serve.*

The high-level model of the intersection closely resembles polling systems, which are systems in queueing theory where one server serves multiple queues and must choose between them [11]. The supervisory controller fulfils the role of the server and chooses when a vehicle may depart from some queue, as depicted in Figure 2.1. Since the CIC method acts on individual vehicles, the supervisory

controller considers vehicles in the queues as individual entities. To ensure that vehicles can cross the intersection safely, some minimum time interval must pass between any two served vehicles. This time interval is called the service time T^s.

First, the service times are defined in Section 2.1. These are based on a simplified version of the vehicle dynamics, where vehicles only move with constant velocity V. Then the intersection as a whole and the vehicles in the CZ are modelled as a hybrid system in Section 2.2. This hybrid system is able to describe the arrival and departure of individual vehicles at the intersection, while also keeping track of the time between arrivals and departures. In order to ensure that the service time T^s is respected between departures and that vehicles pass through the intersection safely, it is necessary to place certain constraints on the system. These constraints are formulated in Section 2.3. The chapter ends with a short summary in Section 2.4.

2.1 Service times

As mentioned in the introduction of this chapter, some minimum time interval must pass between any two vehicles which depart from their queues, which is called the service time T^s. This ensures that the virtual inter-vehicle distance is sufficient to guarantee a safe passage through the intersection when these vehicles form a virtual platoon. The service time is defined through a simple dynamical model of the vehicles. It is assumed that vehicles move only with constant velocity V, and wait in their queues until they are allowed to depart. Then, the service time can be calculated from the desired virtual inter-vehicle distance δ_{des}. This distance is the minimum safe virtual distance between two vehicles. In general, δ_{des} is smaller for subsequent vehicles departing from the same queue, since these vehicles can rely on their own sensors to determine their inter-vehicle distance, instead of receiving information on the other vehicle's position solely through wireless communication. In the case of departures from different queues, the service time varies depending on their respective trajectories. A trajectory is defined by the queue from which a vehicle departs and the direction through which the vehicle wishes to exit the intersection. An example to illustrate the difference in service time between two trajectories is shown in figures 2.2 and 2.3.

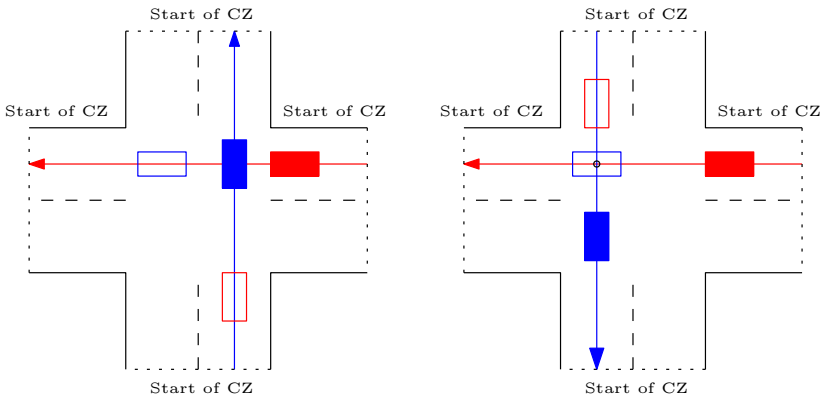

Figure 2.2: *A schematic view of two situations where a red vehicle follows a blue vehicle. The outlined vehicles show each vehicle's virtual positions with respect to the other vehicle. In both cases the virtual inter-vehicle distance is equal, but the physical situation is vastly different.*

Figure 2.2 shows two situations in which the red vehicle follows the blue vehicle and is about to enter the intersection. The virtual inter-vehicle distance is the same in both cases, but is too small for the situation on the left, and too large in the situation on the right. This is due to the key role that the layout of the intersection plays in the desired virtual inter-vehicle distance. The desired virtual inter-vehicle distance δ_{des} differs between the two situations because the vehicle's trajectories do not cross in the centre of the intersection. The lanes have a physical width which changes the physical distance between them. If one direction of approach is longer, this also affects the distance between them, and δ_{des} must be changed to compensate. Figure 2.3 shows the same situations but with more appropriate inter-vehicle distances. In the left situation, δ_{des} is larger so that the vehicles can cross safely. On the right, δ_{des} is smaller so that the throughput of the intersection is higher.

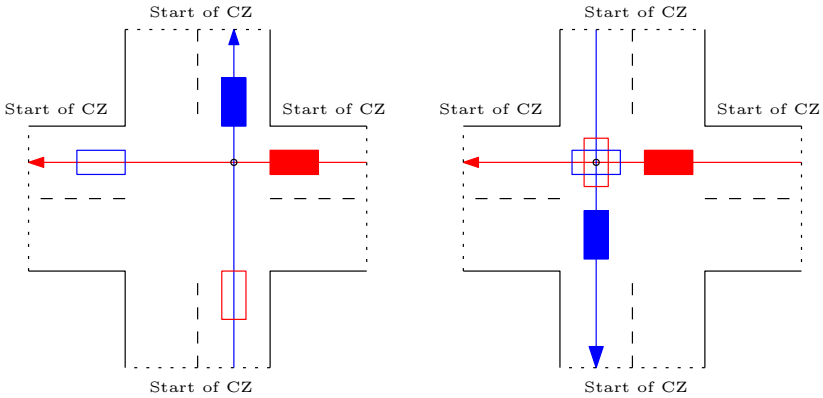

Figure 2.3: *The same situations as depicted in Figure 2.2, but with more appropriate virtual inter-vehicle distances.*

Let $T^s_{q_1,q_2}$ be the service time between a departure from queue q_1 and a subsequent departure from queue q_2, and let $\mathcal{Q} = \{1, \ldots, n_l\}$ be the index set of all queues with $q_1, q_2 \in \mathcal{Q}$, where n_l is the number of queues. The directions through which these vehicles wish to exit the intersection are denoted by q_{out,q_1} and q_{out,q_2}, respectively, as illustrated in Figure 2.4. This information is assumed to be provided to the controller by the vehicles in the queues, and the front vehicle in the queue determines the value of $q_{out,q}$ of that queue. Then a trajectory is defined by a pair $(q, q_{out,q})$ for $q, q_{out,q} \in \mathcal{Q}$, and the desired inter-vehicle distance between two vehicles from queues q_1 and q_2 is defined as a function of the vehicles' trajectories $\delta_{des}(q_1, q_{out,q_1}, q_2, q_{out,q_2})$. Then the service time is given by

$$T^s_{q_1,q_2} = \frac{\delta_{des}(q_1, q_{out,q_1}, q_2, q_{out,q_2})}{V}, \tag{2.1}$$

where V is the constant velocity of the vehicles. In the case that two vehicles have trajectories which do not intersect, they will not form a virtual platoon. In that case, the virtual inter-vehicle distance does not matter, and both δ_{des} and $T^s_{q_1,q_2}$ are defined to be equal to 0. This allows those vehicles to depart from their queues at the same time. Note that the service times between two queues is not necessarily symmetrical, i.e., $T^s_{q_1,q_2} \neq T^s_{q_2,q_1}$. This is easily seen by switching the positions of the blue and red vehicles in Figure 2.2 along their own trajectories. If the blue vehicle follows the red vehicle, the service times are different than the service times if the red vehicle follows the blue vehicle.

The different service times can form an asymmetrical matrix $T^s \in \mathbb{R}^{n_l \times n_l}$, which can either be time-invariant or time-varying due to the layout of the intersection. If vehicles with different trajectories $(q, q_{out,q})$ depart from the same queue q, then the service times associated with that queue vary in time as $q_{out,q}$ changes. On the other hand, in many intersections vehicles which have different intentions will be on different lanes. An example of this is shown in Figure 2.4.

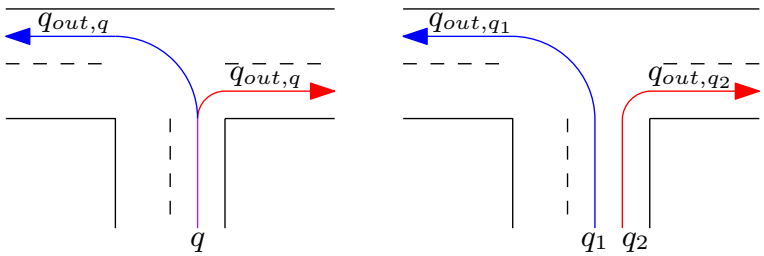

Figure 2.4: *A schematic view two similar intersections showing different trajectories. On the left, the red and blue trajectories share the same incoming queue, while on the right, each trajectory has its own queue.*

In the intersection shown in the right figure, there exists a different queue for every trajectory that vehicles can have. This results in there being more queues, but also means that T^s is time-invariant. The intersection model which is designed in Section 2.2 is able to deal with both situations.

2.2 High-level hybrid intersection model

In the high-level description of the intersection as a queueing system, arriving vehicles wait in queues outside the CZ until the supervisory controller allows them to depart from their queue into the CZ, as shown in Figure 2.1. This is modelled by describing the evolution of the number of vehicles in the queues, i.e., the queue lengths, which is given by the arrival of individual vehicles to a queue, and the departure of vehicles from that queue. To facilitate a description using individual vehicles while also keeping track of the exact time of a vehicle's arrival or departure, the intersection is modelled as a hybrid system. This is a system which can evolve both in continuous-time t and discrete-time j. A modelling and analysis framework for hybrid systems has been developed by Goebel, Sanfelice and Teel in [12, 13], and is used in this report. In the hybrid system, arrivals or departures occurring at any queue are considered instantaneous events which cause the system to evolve in discrete-time and increase the event counter j. Before the system dynamics can be modelled, the arrival and departure signals must be defined, since these govern whether the system jumps in discrete-time or not. Arrivals to a queue $q \in \mathcal{Q}$ are given by a measured disturbance $w_q^a(t)$ and departures from a queue are controlled by the input $u_q(t)$. These signals indicate the arrival or departure of a single vehicle at a time t, at which time $w_q^a(t) = 1$ or $u_q(t) = 1$. Because events are instantaneous, the input and departure signals are discontinuous and are modelled as

$$\begin{aligned} w_q^a(t) &= \begin{cases} 1 & \text{if a vehicles arrives at queue } q \text{ at time } t, \\ 0 & \text{otherwise,} \end{cases} \\ u_q(t) &= \begin{cases} 1 & \text{if a vehicles departs from queue } q \text{ at time } t, \\ 0 & \text{otherwise,} \end{cases} \end{aligned} \tag{2.2}$$

with $w^a(t) = \begin{bmatrix} w_1^a(t) & \dots & w_{n_l}^a(t) \end{bmatrix}^T \in \{0,1\}^{n_l}$ and $u(t) = \begin{bmatrix} u_1(t) & \dots & u_{n_l}(t) \end{bmatrix}^T \in \{0,1\}^{n_l}$ the vectors for arrivals and departures, respectively. These signals are inputs to the hybrid system and exist only in continuous-time t because the event counter j at which an event occurs is not known a priori. Therefore, $w^a(t)$ and $u(t)$ must first be defined in the hybrid time domain. A subset E of $\mathbb{R}_{\geq 0} \times \mathbb{N}$ is a hybrid time domain if it is the union of a number intervals of the form $[t_j, t_{j+1}] \times \{j\}$ with $t_0 = 0$ and $t_{j+1} \geq t_j$. Furthermore, the solution of the system ϕ is the trajectory that the system follows along the hybrid time domain E. Using these two definitions, arrivals and departures can be described in hybrid time as

$$\begin{aligned} w^a(t,j) = w^a(t,j) \quad & \forall (t,j) \in \text{dom } \phi, \\ u(t,j) = u(t) \quad & \forall (t,j) \in \text{dom } \phi. \end{aligned} \tag{2.3}$$

In the notation by Goebel *et al.*, the system flows in continuous time if there are no events, at which time the signals $w^a(t,j)$ and $u(t,j)$ are in the flow set $\mathcal{C}$. When $w^a(t,j)$ or $u(t,j)$ enter the jump set $\mathcal{D}$, an event occurs and the dynamics jump in discrete-time j. However, the flow and jump set cannot be defined solely through arrivals and departures. It is necessary to ensure a minimum passage of time between two events. Otherwise, a situation could occur in which an infinite number of events occur in finite time t, which is called Zeno behaviour. Therefore, the following assumption is required:

Assumption 1. *For all $t \geq 0$ such that $u_q(t) = 1$ for some queue q, there exists an $\varepsilon_u > 0$ such that $u_q(t', j') = 0 \ \forall t' \in (t, t + \varepsilon_u]$ and $(t', j') \in$ dom ϕ. Additionally, for all $t \geq 0$ such that $w_q^a(t) = 1$ for some queue q, there exists an $\varepsilon_w > 0$ such that $w_q^a(t', j') = 0 \ \forall t' \in (t, t + \varepsilon_w]$ and $(t', j') \in$ dom ϕ.*

This assumption is justified by the fact that vehicles cannot physically be in the same location at the same time, and therefore there is always some minimum time interval between any two arrivals to or departures from the same queue. A possible value of ε_u and ε_w is $\frac{l}{V}$, where l is the length of a vehicle and V its velocity, though any small, finite value will be sufficient to preclude Zeno behaviour.

Assumption 1 must also be enforced by incorporating ε_w and ε_u in the flow and jump sets through the use of inter-event timers. To this end, an inter-departure timer $t_{c,q}(t,j)$ and an inter-arrival timer $t_{w,q}(t,j)$ are defined for every queue $q \in \mathcal{Q}$. Their continuous-time dynamics are described by

$$\begin{aligned} \dot{t}_{c,q}(t,j) &= 1, \\ \dot{t}_{w,q}(t,j) &= 1, \end{aligned} \tag{2.4}$$

and their discrete-time dynamics are given by

$$\begin{aligned} t_{c,q}(t,j+1) &= \begin{cases} t_{c,q}(t,j) & \text{if } u_q(t,j)=0, \\ 0 & \text{otherwise,} \end{cases} \\ t_{w,q}(t,j+1) &= \begin{cases} t_{w,q}(t,j) & \text{if } w_q^a(t,j)=0, \\ 0 & \text{otherwise,} \end{cases} \end{aligned} \tag{2.5}$$

for each $q \in \mathcal{Q}$, which gives the vectors $t_c(t,j) = \begin{bmatrix} t_{c,1}(t,j) & \dots & t_{c,n_l}(t,j) \end{bmatrix}^T \in \mathbb{R}_{\geq 0}^{n_l}$ and $t_w(t,j) = \begin{bmatrix} t_{w,1}(t,j) & \dots & t_{w,n_l}(t,j) \end{bmatrix}^T \in \mathbb{R}_{\geq 0}^{n_l}$. The flow and jump sets can then be defined using the inter-event timers and the arrival and departure signals. Two flow sets and two jump sets $\mathcal{C}_u$, $\mathcal{C}_w$, $\mathcal{D}_u$ and $\mathcal{D}_w$ are created in order to link the inter-departure timer $t_{c,q}(t,j)$ to $u_q(t,j)$ and to link the inter-arrival timer $t_{w,q}(t,j)$ to $w_q^a(t,j)$. To achieve the desired behaviour, these sets must be defined such that the system only evolves in discrete-time if 1) an event occurs at some queue q and 2) the relevant inter-event timer is greater than the minimum inter-event time. With this in mind, the flow and jump sets are given by

$$\begin{aligned} \mathcal{C}_u &= \{u(t,j) = \underline{0}^{n_l}\} \times \left\{ t_c(t,j) \in \mathbb{R}_{\geq 0}^{n_l} \right\} \\ &\quad \bigcup \{u(t,j) \in \{0,1\}^{n_l}\} \setminus \{\underline{0}^{n_l}\} \times \{t_{c,q}(t,j) < \varepsilon_u \ \exists q \in \mathcal{Q} \text{ s.t. } u_q(t,j)=1\}, \\ \mathcal{C}_w &= \{w^a(t,j) = \underline{0}^{n_l}\} \times \left\{ t_w(t,j) \in \mathbb{R}_{\geq 0}^{n_l} \right\} \\ &\quad \bigcup \{w^a(t,j) \in \{0,1\}^{n_l}\} \setminus \{\underline{0}^{n_l}\} \times \left\{ t_{w,q}(t,j) < \varepsilon_w \ \exists q \in \mathcal{Q} \text{ s.t. } w_q^a(t,j)=1 \right\}, \\ \mathcal{D}_u &= \{u(t,j) \in \{0,1\}^{n_l}\} \setminus \{\underline{0}^{n_l}\} \times \{t_{c,q}(t,j) \geq \varepsilon_u \ \forall q \in \mathcal{Q} \text{ s.t. } u_q(t,j)=1\}, \\ \mathcal{D}_w &= \{w^a(t,j) \in \{0,1\}^{n_l}\} \setminus \{\underline{0}^{n_l}\} \times \left\{ t_{w,q}(t,j) \geq \varepsilon_w \ \forall q \in \mathcal{Q} \text{ s.t. } w_q^a(t,j)=1 \right\}, \end{aligned} \tag{2.6}$$

where $\underline{0}^{n_l}$ is the zero vector of length n_l. Note that each flow set consists of the union of two sets. The first set gives the situation where $u(t,j) = w^a(t,j) = \underline{0}^{n_l}$, in which case the inter-event timers may have any value in $\mathbb{R}_{\geq 0}$. The second set is similar in definition to the jump sets, and contains all possible vectors which can be created from zeros and ones, except for the zero vector. This ensures that if there is an event at some queue and the inter-event timers of that queue do not satisfy Assumption 1, e.g. $t_{c,q}(t,j) < \varepsilon_u$, then the system flows in continuous-time and there is no discrete-time jump. With (2.6), the flowing and jumping of the system in hybrid time is governed by

$$(u(t,j), t_c(t,j)) \in \mathcal{C}_u \wedge (w^a(t,j), t_w(t,j)) \in \mathcal{C}_w, \tag{2.7}$$

$$(u(t,j), t_c(t,j)) \in \mathcal{D}_u \vee (w^a(t,j), t_w(t,j)) \in \mathcal{D}_w, \tag{2.8}$$

where the system flows in continuous-time if (2.7) holds and jumps in discrete-time if (2.8) holds.

Now that the flow and jump sets have been defined, the remaining dynamics are quite simple. The state of the queues in the intersection is characterised only by the queue lengths $x_q(t,j)$ and by the activity of those queues $m_q(t,j)$. The queue length of a queue $q \in \mathcal{Q}$ can be described purely through arrivals and departures by the the following hybrid dynamics:

$$\begin{aligned} \dot{x}_q(t,j) &= 0 && (u(t,j), t_c(t,j)) \in \mathcal{C}_u \wedge (w^a(t,j), t_w(t,j)) \in \mathcal{C}_w, \\ x_q(t,j+1) &= x_q(t,j) + w_q^a(t,j) - u_q(t,j) && (u(t,j), t_c(t,j)) \in \mathcal{D}_u \vee (w^a(t,j), t_w(t,j)) \in \mathcal{D}_w, \end{aligned} \tag{2.9}$$

with $x(t,j) = \begin{bmatrix} x_1(t,j) & \dots & x_{n_l}(t,j) \end{bmatrix}^T \in \mathbb{N}_0^{n_l}$ the vector of the queue lengths.

When a vehicle departs from a queue, that queue then becomes active. A queue being active signifies that a vehicle has recently departed from that queue, and that its service time has not passed yet. This lets the controller determine of which queues the service times need to be taken into account when choosing to let a later vehicle depart from its queue. When the largest service time of an active queue has passed, that queue then becomes inactive, since it no longer conflicts with later departures.

The queue activity $m_q(t,j)$ is binary variable with dynamics given by

$$\begin{aligned}
&\dot{m}_q(t,j) = 0, && (u(t,j),t_c(t,j)) \in \mathcal{C}_u \\
&&& \wedge\, (w^a(t,j),t_w(t,j)) \in \mathcal{C}_w, \\
&m_q(t,j+1) = \begin{cases} 1 & \text{if } u_q(t,j) = 1 \\ 0 & \text{if } t_{c,q}(t,j+1) \geq \max_{i\in\mathcal{Q}|\{q\}} T^s_{q,i} \\ m_q(t,j) & \text{otherwise} \end{cases} && (u(t,j),t_c(t,j)) \in \mathcal{D}_u \\
&&& \vee\, (w^a(t,j),t_w(t,j)) \in \mathcal{D}_w,
\end{aligned} \tag{2.10}$$

with $m(t,j) = \begin{bmatrix} m_1(t,j) & \dots & m_{n_l}(t,j) \end{bmatrix}^T \in \{0,1\}^{n_l}$ the vector of the queue activity, which completes the system dynamics.

Remark. *Note that, since the system only evolves in discrete-time if there is an arrival or departure, it is very likely that a queue remains active for some time even after the maximum service time has passed. While this could be considered an undesirable feature of the current model, in Section 2.3 the constraints on vehicle departures are formulated such that this has no effect. Should it be necessary to ensure that a queue becomes inactive as soon as $t_{c,q}(t,j) = \max_{i\in\mathcal{Q},\{q\}} T^s_{q,i}$, a new flow and jump set need to be defined. This is the case because the system should jump in discrete-time regardless of whether there is a vehicle arrival/departure or not. The third flow and jump set can be defined as*

$$\begin{aligned}
\mathcal{C}_m &= \{m(t,j) \in \{0,1\}^{n_l}\} \times \left\{ t_{c,q}(t,j) < \max_{i\in\mathcal{Q},\{q\}} T^s_{q,i}\ \forall q \in \mathcal{Q}\ s.t.\ m_q(t,j) = 1 \right\}, \\
\mathcal{D}_m &= \{m(t,j) \in \{0,1\}^{n_l}\} \times \left\{ t_{c,q}(t,j) \geq \max_{i\in\mathcal{Q}|\{q\}} T^s_{q,i}\ \exists q \in \mathcal{Q}\ s.t.\ m_q(t,j) = 1 \right\},
\end{aligned} \tag{2.11}$$

so that the hybrid dynamics are then governed by

$$\begin{aligned}
&(u(t,j),t_c(t,j)) \in \mathcal{C}_u \wedge (w^a(t,j),t_w(t,j)) \in \mathcal{C}_w \wedge (m(t,j),t_c(t,j)) \in \mathcal{C}_m, \\
&(u(t,j),t_c(t,j)) \in \mathcal{D}_u \vee (w^a(t,j),t_w(t,j)) \in \mathcal{D}_w \vee (m(t,j),t_c(t,j)) \in \mathcal{D}_m.
\end{aligned} \tag{2.12}$$

This change in the system behaviour would achieve an exact description of the desired dynamics, but also makes the flowing and jumping of the system more complicated. Instead, the above flow and jump set are not used in the rest of this report in favor of (2.6)*.*

All of the system dynamics have now been defined and the full state of the system $z(t,j)$ is given by

$$z(t,j) = \begin{bmatrix} x(t,j)^T & t_c(t,j)^T & t_w(t,j)^T & m(t,j)^T \end{bmatrix}^T, \tag{2.13}$$

where $x(t,j)$ is the vector of queue lengths, $m(t,j)$ is the vector of the queue activities and $t_c(t,j)$ and $t_w(t,j)$ are the vectors of inter-event timers. The full hybrid dynamics of the intersection are described by

$$\begin{aligned}
&\dot{x}_q(t,j) = 0 && \\
&\dot{t}_{c,q}(t,j) = 1 && (u(t,j),t_c(t,j)) \in \mathcal{C}_u \\
&\dot{t}_{w,q}(t,j) = 1 && \wedge\, (w^a(t,j),t_w(t,j)) \in \mathcal{C}_w, \\
&\dot{m}_q(t,j) = 0 && \\
&x_q(t,j+1) = x_q(t,j) + w^a(t,j) - u(t,j) && \\
&t_{c,q}(t,j+1) = \begin{cases} t_{c,q}(t,j) & \text{if } u_q(t,j) = 0 \\ 0 & \text{otherwise} \end{cases} && \\
&t_{w,q}(t,j+1) = \begin{cases} t_{w,q}(t,j) & \text{if } w^a_q(t,j) = 0 \\ 0 & \text{otherwise} \end{cases} && (u(t,j),t_c(t,j)) \in \mathcal{D}_u \\
&&& \vee\, (w^a(t,j),t_w(t,j)) \in \mathcal{D}_w, \\
&m_q(t,j+1) = \begin{cases} 1 & \text{if } u_q(t,j) = 1 \\ 0 & \text{if } t_{c,q}(t,j+1) \geq \max_{i\in\mathcal{Q}|\{q\}} T^s_{q,i} \\ m_q(t,j) & \text{otherwise} \end{cases} &&
\end{aligned} \tag{2.14}$$

for each $q \in \mathcal{Q}$. As described in Section 1.2, the system outputs the state of the intersection to the controller. This defines the output of the hybrid system as

$$y(t,j) = z(t,j), \tag{2.15}$$

which is used by the controller to optimise the crossing sequence.

2.3 Constraints on vehicle departures

Now that the service times have been defined and the high-level intersection model has been formulated, it is possible to define certain necessary constraints on the control input $u(t,j)$ so that the service times are respected. Strictly speaking, these constraints are part of the design of the supervisory controller, and not of the intersection model. Still, they are discussed here because these constraints arise from the description of the intersection as a queueing system with given service times between any two vehicle departures. Thus, they are considered to be a part of the hybrid system itself and a parameter of the intersection, which the controller must take into account.

The constraints are as follows: first, a vehicle may only depart from its queue if the service times of all active queues have passed. Assuming that a departure from queue $q_2 \in \mathcal{Q}$ follows a departure from queue $q_1 \in \mathcal{Q}$, this can be formulated as

$$\begin{aligned} u_{q_2}(t,j) &= 0 \quad \text{if } \exists q_1 \in \mathcal{Q} \text{ s.t. } \left(m_{q_1}(t,j) = 1 \wedge t_{c,q_1}(t,j) < T^s_{q_1,q_2}\right), \\ u_{q_2}(t,j) &\in \{0,1\} \quad \text{if } \nexists q_1 \in \mathcal{Q} \text{ s.t. } \left(m_{q_1}(t,j) = 1 \wedge t_{c,q_1}(t,j) < T^s_{q_1,q_2}\right). \end{aligned} \tag{2.16}$$

These constraints ensure that no vehicle departs from its queue if doing so creates an unsafe situation with a vehicle which has already entered the Cooperation Zone. Note that in the actual intersection considered by Morales in [10], this formulation ensures that a departing vehicle always follows the vehicle whose service time passes last.

By using the queue activity and the inter-departure times, (2.16) constrains vehicle departures based on previous departures. However, this does not take into account vehicles that are still in other queues. It is possible that there exist multiple queues for which all service times have passed and the vehicles in them are allowed to depart. An additional constraint is necessary to ensure that these vehicles do not depart at the same time if their trajectories cross. Let $q, i \in \mathcal{Q}$ be the queues from which these two vehicles wish to depart. Using the fact that the service time between two vehicles whose trajectories do not cross is defined as $T^s_{q,i} = T^s_{i,q} = 0$, the second constraint is given by

$$\begin{bmatrix} u_q(t,j) \\ u_i(t,j) \end{bmatrix} \neq \begin{bmatrix} 1 \\ 1 \end{bmatrix} \text{ if } T^s_{q,i} > 0, \tag{2.17}$$

which must hold for all $q, i \in \mathcal{Q}$ such that $i > q$. The reason that this only needs to be defined for $i > q$ the service times $T^s_{q,i}$ and $T^s_{i,q}$ are equal. Therefore, there is no need to define this constraint for all possible pairs $q, i \in \mathcal{Q}$, since this would create multiple identical constraints.

The final constraint which must be placed on vehicle departures is one to ensure that the controller is fair from the driver's perspective. As mentioned in Section 1.2, the goal of the supervisory controller is to minimise the average delay in a manner which is fair from the driver's perspective. To ensure fairness, it is necessary to ensure that no vehicles are kept waiting in their queue for too long. But if one queue has a far higher arrival rate than another queue, drivers in the queue with the lower arrival rate may be kept waiting indefinitely if the goal is only to minimise the average delay. This is a well-known scheduling problem called starvation [14, 15], which is solved through the use of so-called fairness constraints which ensure that no queues are starved even if traffic flows are unbalanced. Two possible constraints are given by

$$\begin{aligned} t_{c,q}(t,j) &\leq T_{q,\max}, \\ x_q(t,j) &\leq X_{q,\max}, \end{aligned} \tag{2.18}$$

for all $q \in \mathcal{Q}$, which ensure that every queue q is served after at most $T_{q,\max}$ seconds and that no queue exceeds $X_{q,\max}$ vehicles in length. However, this only creates fairness from the perspective of the queues, not for the drivers in those queues. To ensure fairness from the driver's perspective, the time each vehicle spends in its queue should be limited or the server should be forced to serve a queue for a minimum amount of time. The latter is often implemented in traffic literature and is called the minimum green time. Unfortunately, it is not possible to implement either of these methods in the current model formulation, since neither the time each vehicle has been in its queue nor the time spent serving a queue are available in the current model. As such, (2.18) is used to achieve fairness, even though it may not fully achieve the desired goal. Alternative methods to implement fairness are briefly discussed in Section 3.4, and the effect of these fairness constraints is investigated in Section 5.2.3.

The notation of the input constraints can be shortened by defining a state-dependent input set $\mathcal{U}(x(t,j), m(t,j), t_c(t,j))$. This set follows from the constraints given in (2.16)–(2.18), and the exact mathematical definition of this set is not important for the purposes of this report. By using this set the constraints on the input are written as

$$u(t,j) \in \mathcal{U}(x(t,j), m(t,j), t_c(t,j)). \tag{2.19}$$

2.4 Summary

In this chapter, the automated intersection has been modelled as a hybrid system, which can evolve both in continuous-time and in discrete-time. By using a simplified model for the vehicle dynamics, the intersection is characterised as a polling system where the vehicles wait in queues until they are served. The hybrid system evolves in discrete-time only when there is a vehicle arrival or departure. Through the use of inter-arrival and inter-departure timers, the hybrid system description allows for a service time to be enforced between any two vehicle departures. The service times ensure that the desired virtual inter-vehicle distance is respected, which allows vehicles to pass through the intersection safely. Two control constraints are formulated which ensure that the controller respects the service times, and an additional constraint is formulated to ensure that the controller is fair from the driver's perspective.

In Chapter 3, using the model proposed in this chapter, a controller for the hybrid system is designed. It is designed such that it optimises the departures of vehicles from their queues to minimise their average delay, while respecting the constraints given in Section 2.3. The tooling for the numerical implementation of the hybrid system and the controller are discussed in Chapter 4.

Chapter 3

Controller design

The automated intersection has been modelled as a hybrid system, which allows for a service time to be enforced between vehicle departures, in Chapter 2. In this chapter, the controller for the hybrid system is designed. This controller must optimise the input it provides to the system in order to minimise the average delay of the vehicles, while also respecting the constraints formulated in Section 2.3. These constraints challenge the design of a feedback law. Given the combination of an optimisation objective and the constraints, it is decided to opt for a Model Predictive Control (MPC) approach. Model predictive controllers select the control actions which lead to the best predicted outcome over some finite horizon. The controller uses a model of the system to make its predictions, and updates its control actions as it receives new observations of the system [16]. While this makes MPC an attractive control method for most applications, there is currently little research into MPC for hybrid systems formalised in the framework of [12, 13]. Instead, most applications of MPC are aimed at discrete-time systems. In order to apply MPC to the intersection problem, the controller samples the system, and the controller uses a discretised version of the system in its predictions.

The remainder of this chapter is organised as follows. Section 3.1 presents a general description of model predictive control and explains the notions of stability and feasibility. In Section 3.2, the hybrid system is discretised in support of the controller design, and a disturbance model which predicts vehicle arrivals is formulated in Section 3.3. Section 3.4 gives the cost function and the complete formulation of the MPC optimisation problem. The chapter concludes with a summary in Section 3.5.

3.1 Model predictive control

Model predictive control, also called receding horizon control, has become an increasingly popular control method mostly applied on discrete-time systems. This increasing interest is driven by the need for tighter performance specifications and more and more constraints which need to be satisfied [17]. In general, MPC consists of solving an optimal control problem subject to the system dynamics and other constraints. The control problem has a finite horizon N_p and must be solved on-line. Every time-step, the controller uses the dynamical model to predict the behaviour of the system over the prediction horizon and determines the input trajectory such that a given cost function is minimised in open-loop. A feedback mechanism is achieved by implementing only the first step of the optimal open-loop input trajectory and repeating the optimisation every time-step.

While model predictive control is an attractive control method, it also has its disadvantages. In order for it to control a system accurately, an accurate model is needed. An inaccurate model or the presence of a disturbance acting upon the system may lead to poor performance. Secondly, the controller's prediction horizon must be carefully chosen. Ideally, the prediction horizon is infinite. However, it is clear that the longer the horizon, the greater the computational cost. In order to allow a real-time solution, a short horizon is preferable. On the other hand, if the horizon is too short, the closed-loop trajectory of the system will differ from the predicted open-loop trajectory, because

important events remain unobserved within the horizon [16]. This occurs even if the model is perfect and there is no disturbance present [17]. Thus, the prediction horizon must be carefully chosen.

Two other key aspects of MPC are the notions of stability and recursive feasibility. Whether the finite horizon MPC strategy leads to stability of the closed-loop is one of the key questions of MPC [17]. As mentioned earlier, the open-loop prediction and closed-loop trajectory are often different because the prediction horizon is finite. As such, many finite horizon MPC strategies aim to approximate an infinite horizon scheme in order to guarantee stability, which also guarantees recursive feasibility. The optimisation problem is said to be feasible if a solution exists. However, feasibility at the current time-step does not necessarily guarantee feasibility at the next time-step. Naturally, the MPC problem must be feasible at every time-step, which is called recursive feasibility [16]:

Definition 1. The MPC problem is called *recursively feasible* if for all feasible initial states feasibility is guaranteed at every state along the closed-loop trajectory.

A common approach to ensure both recursive feasibility and stability is the dual-mode control approach. This approach makes use of a terminal constraint which implicitly extends the horizon to infinity [18]. By fulfilling this terminal constraint, the states are in a given terminal set. This terminal set is defined to be feasible and positively invariant under some local control law, which means that once the states are in that set, they stay in that set. However, using such a terminal constraint reduces the region of attraction, which is the set of initial states for which the states reach the equilibrium point. A different method often applied in practice is simply to choose a large enough prediction horizon, but determining a sufficiently long horizon can be difficult.

Unfortunately, guaranteeing both recursive feasibility and stability of the intersection controller is difficult. The input set is state dependent, and vehicle arrivals are a disturbance which forever push the queue lengths away from the origin. This makes formulating a control law which would stabilise a terminal set arduous. Instead, it is assumed that the constraints of the to-be-designed model predictive controller are such that the controller is both recursively feasible and stable. Assuming stability and recursive feasibility is quite common in work on MPC, where typically additional constraints are added to guarantee a priori that loss of feasibility cannot occur [19].

3.2 Model discretisation

The MPC controller operates in discrete-time by sampling the output of the plant $y(t, j)$. In order to determine how the sampled system is observed by the controller, the hybrid system is first discretised. The discretised system can then be used as the prediction model that the MPC controller uses in its optimisation. The controller also requires a model of the disturbance, which in this case is a model to predict vehicle arrivals. This disturbance model is discussed in Section 3.3.

Most of the dynamics of the hybrid system in (2.14) already evolve purely in discrete-time, making most of the discretisation straightforward. First of all, the hybrid system is sampled with a constant sampling interval Δt. Then the time instance t_k at time-step $k \geq 0$ is equal to

$$t_k = k\Delta t. \tag{3.1}$$

The next step is to define the discretised states and output, which are sampled at some time in the hybrid time domain. It is assumed that if a jump occurs at the sampling time t_k, the sampled states and output are equal to the states and output after the jump. Then $z(k)$ and $y(k)$ are defined as

$$\begin{aligned} z(k) &= z(t_k, j^*),\ j^* = \max\{j \mid (t_k, j) \in \text{dom } \phi\}, \\ y(k) &= y(t_k, j^*),\ j^* = \max\{j \mid (t_k, j) \in \text{dom } \phi\}. \end{aligned} \tag{3.2}$$

Though the states and output can be easily sampled, incorporating arrivals and departures in the discretised system requires some considerations. These are both instantaneous events and therefore cannot simply be sampled or integrated. Instead, the number of arrivals or departures can be counted. However, if Δt is large it is theoretically possible that there are multiple arrivals or departures between

two sampling instances. In that case $w^a(k)$ or $u(k)$ would no longer lie in the defined flow and jump sets. To preclude this from occurring in the discretised system, the following assumption is made.

Assumption 2. *The sampling time Δt is such that there is at most one arrival to each queue and at most one departure from each queue between two subsequent sampling instances t_{k-1} and t_k. In other words, $u_q(t,j)$ and $w^a(t,j)$ are only non-zero once for $t \in (t_{k-1}, t_k]$, $k \geq 0$ for each $q \in \mathcal{Q}$ and $(t,j) \in dom\ \phi$.*

Under this assumption, $w^a(k)$ and $u(k)$ are always in the flow and jump sets defined in (2.6). Together with Assumption 1, Assumption 2 means that it must hold that $\Delta t \leq \min\{\varepsilon_u, \varepsilon_w\}$. If not, Assumption 2 could be violated. Now, $w_q^a(k)$ and $u_q(k)$ can be defined for each $q \in \mathcal{Q}$ as follows:

$$\begin{aligned} w_q^a(k) &= \begin{cases} 1 & \text{if } \exists t^* \in [t_{k-1}, t_k] \text{ s.t. } w_q^a(t^*, j^*) = 1 \text{ for } (t^*, j^*) \in \text{dom } \phi, \\ 0 & \text{otherwise,} \end{cases} \\ u_q(k) &= \begin{cases} 1 & \text{if } \exists t^* \in [t_{k-1}, t_k] \text{ s.t. } u_q(t^*, j^*) = 1 \text{ for } (t^*, j^*) \in \text{dom } \phi, \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{3.3}$$

With the sampling of the states, inputs and outputs of the hybrid system as defined above, the hybrid dynamics of (2.14) can be discretised. As mentioned earlier, this is fairly straightforward since most of the system's dynamics are already in discrete-time. The only exceptions are the inter-event timers, which do have continuous-time dynamics. These are easily discretised by adding Δt to their value every time-step, and re-setting to Δt when there is an event. The timers are re-set to Δt instead of 0 because Δt seconds pass between the occurrence of the event at time t_k and the new state at time t_{k+1}. Thus, the discretised dynamics of the intersection are given by

$$\begin{aligned} x_q(k+1) &= x_q(k) + w_q^a(k) - u_q(k), \\ t_{c,q}(k+1) &= \begin{cases} t_{c,q}(k) + \Delta t & \text{if } u_q(k) = 0, \\ \Delta t & \text{otherwise,} \end{cases} \\ t_{w,q}(k+1) &= \begin{cases} t_{w,q}(k) + \Delta t & \text{if } w_q^a(k) = 0, \\ \Delta t & \text{otherwise,} \end{cases} \\ m_q(k+1) &= \begin{cases} 1 & \text{if } u_q(k) = 1, \\ 0 & \text{if } t_{c,q}(k+1) \geq \max_{i \in \mathcal{Q} \mid \{q\}} T_{q,i}^s, \\ m_q(k) & \text{otherwise,} \end{cases} \end{aligned} \tag{3.4}$$

for each $q \in \mathcal{Q}$. Note that in constrast to the hybrid dynamics, the discretised dynamics of the active queues $m_q(k)$ are no longer flawed. In the hybrid system an active queue only becomes inactive when an event occurs. Since the discrete-time system jumps regardless of whether there is an event or not, an active queue now becomes inactive immediately after the maximum service time has passed. The discretised dynamics are written in shorthand notation as

$$\begin{aligned} z(k+1) &= F(z(k), w^a(k), u(k)), \\ y(k) &= z(k). \end{aligned} \tag{3.5}$$

The final step in the discretisation is to discretise the constraints on vehicle departures that are placed on the MPC controller, which have been formulated in (2.16)–(2.18) in Section 2.3. The discrete-time formulation of the constraints is functionally identical to their formulation in hybrid time. Constraint (2.16) on departures from a queue $q_2 \in \mathcal{Q}$ which follows a departure from some queue $q_1 \in \mathcal{Q}$ is given by

$$\begin{aligned} u_{q_2}(k) &= 0 \qquad \text{if } \exists q_1 \in \mathcal{Q} \text{ s.t. } \left(m_{q_1}(k) = 1 \wedge t_{c,q_1}(k) < T_{q_1,q_2}^s\right), \\ u_{q_2}(k) &\in \{0,1\} \quad \text{if } \nexists q_1 \in \mathcal{Q} \text{ s.t. } \left(m_{q_1}(k) = 1 \wedge t_{c,q_1}(k) < T_{q_1,q_2}^s\right). \end{aligned} \tag{3.6}$$

Constraint (2.17) on two simultaneous departures from queues $q, i \in \mathcal{Q}$ such that $i > q$ is equal to

$$\begin{bmatrix} u_q(k) \\ u_i(k) \end{bmatrix} \neq \begin{bmatrix} 1 \\ 1 \end{bmatrix} \text{ if } T_{q,i}^s > 0, \tag{3.7}$$

and the fairness constraints of (2.18) are as follows:

$$\begin{aligned} t_{c,q}(k) &\leq T_{q,\max}, \\ x_q(k) &\leq X_{q,\max}. \end{aligned} \tag{3.8}$$

Note that the fairness constraints also place a minimum on the duration of the prediction horizon $N_p\Delta t$ in order for the optimisation problem to be recursively feasible. This minimum is given by $N_p\Delta t \geq \max_{q_1,q_2\in\mathcal{Q}} T^s_{q_1,q_2}$, and is necessary to ensure that the controller can still switch queues if the fairness constraints become active within the prediction horizon. If this condition does not hold and $t_{c,q}(k)$ nears $T_{q,\max}$ for some $q \in \mathcal{Q}$, then the controller does not see this within its prediction horizon until it is already too late to switch queues. At this point, the optimisation problem becomes infeasible.

As in Section 2.3, the notation of the input constraints is shortened through the state-dependent input set $\mathcal{U}(x(k), m(k), t_c(k))$. This can be written as

$$u(k) \in \mathcal{U}(x(k), m(k), t_c(k)). \tag{3.9}$$

3.3 Disturbance model

In addition to the dynamical prediction model, the MPC controller also uses a model of the disturbance. In most applications of MPC, the disturbance model is used to increase the robustness of the controller with respect to model uncertainty and noise. Then the control system is called robust if stability is maintained and the performance specifications are met for a specified range of model variations and a class of noise signals [20]. In the case of the intersection, the input $u(t, j)$ must be designed solely to account for the disturbance $w^a(t, j)$. After all, if there were no vehicle arrivals, there is no need for vehicle departures either. Yet a disturbance model is not strictly necessary since the optimisation can also be based purely on the current queue lengths. Instead, the disturbance model is used to increase the reliability of the controller's predictions, since predicting vehicle arrivals reflects the behaviour of the intersection in the future far more accurately. For example, if one queue has a higher arrival rate, but is currently (near) empty, using a disturbance model can lead to better control actions.

There are several methods by which vehicle arrivals are often modelled. If the intersection is located in a network of intersections, then vehicles can be assumed to arrive in batches. On a highway, the inter-arrival time is often modelled according to a Poisson distribution, which is a realistic model as long as traffic is only lightly congested [21]. This report focuses on an isolated intersection, where vehicles can also be assumed to arrive according to this distribution. However, because stochastic arrivals are impossible to predict exactly, the disturbance model assumes that vehicles arrive in a deterministic, uniform pattern with an average arrival rate λ_q for queue q. Then the disturbance model is given by the predicted arrivals $\hat{w}^a_q(k)$, equal to

$$\hat{w}^a_q(k) = \begin{cases} 1 & \text{if } t_{w,q} \geq \frac{1}{\lambda_q}, \\ 0 & \text{otherwise,} \end{cases} \tag{3.10}$$

for each $q \in \mathcal{Q}$. This disturbance model also requires knowledge of λ_q. In some cases, the average arrival rate λ_q can be a known constant. But in reality the arrival rate is likely to vary in time. The arrival rate is higher during rush hour and lower at night, and may also be different in the weekend than during weekdays. Additionally, the arrival rate is not always known. Though traffic flow is often measured at busy intersections, this data is not always available or up-to-date. Therefore, the controller needs a method to estimate the current average arrival rate based on its own historic data. One method is to use a moving average window, which calculates the average value of a signal up to some horizon in the past. Assuming a horizon of N_w steps, $\lambda_q(k)$ can be calculated by counting the number of past vehicle arrivals in the window as

$$\lambda_q(k) = \frac{1}{\Delta t N_w} \sum_{i=0}^{N_w} w^a_q(k-i), \tag{3.11}$$

with $\lambda_q(0) = w_q^a(0)$ and $w_q^a(k) = 0$ for $k < 0$.

Finally, it may also be necessary to predict the trajectory of the vehicles $(q, q_{out,q})$ for $q, q_{out,q} \in \mathcal{Q}$. As mentioned in Section 2.1, if vehicles with different trajectories share the same queue, the service time $T^s_{q_1,q_2}$ is time-varying. This can greatly affect the predicted system behaviour, and thereby affect the optimal input. One possibility is to assume the worst-case scenario where $q_{out,q}$ is always such as to ensure the highest possible service times. But this is likely too conservative, since $q_{out,q}$ is partially available, because this information is received from the vehicles in the queues. This allows for an accurate prediction until the controller reaches a point in its prediction where those vehicles have already departed. After that, the prediction becomes less accurate and could instead be based on historic data. The simplest method is to always choose the most common exit direction. A slightly more elaborate method is to choose the exit direction in proportion to how often vehicles from each queue exit the intersection there. Either way, these methods can give ambiguous results, and therefore for the remainder of this report it is assumed that there exists a dedicated queue on each entry lane for each trajectory, i.e., for each exit lane as depicted in Figure 2.4. This makes the service times T^s time-invariant.

3.4 MPC controller design

The prediction model, the disturbance model and the constraints which form the input set of the MPC controller have been formulated in the previous sections. What remains is the definition of the objective function. The objective function follows from the goal of the controller, which is to minimise the average delay of the vehicles passing through the intersection. While the average delay cannot be directly measured, it can easily be calculated by integrating the queue lengths. After all, the only delay the vehicles incur is the time spent waiting in the queues, since they always move through the Cooperation Zone with constant velocity. With prediction horizon N_p, the cost function $J(k)$ is given by

$$J(k) = \sum_{i=0}^{N_p-1} c^T x(k+i), \tag{3.12}$$

where $c = \begin{bmatrix} c_1 & \dots & c_{n_l} \end{bmatrix}$ is a vector which can be used to weigh the individual queue lengths. The weights in c can be varied if one queue should be given a higher priority than another, so that the controller will keep that queue shorter. The cost function is chosen as linear because the queue lengths can only be positive, and thus there is no chance of a negative queue length cancelling out a positive one. It also ensures that every vehicle in a queue is weighed the same. The cost function may also be chosen as quadratic if this is more desirable for the automated intersection. Using a quadratic cost function is also a simple way to make the controller more fair from the driver's perspective, and is a way to implement the constraints of (3.8) as soft constraints. This is because longer queues have a far higher cost than short queues. This will make the controller switch to a starved queue as its length increases, even without a hard limit on $t_{c,q}(k)$ or $x_q(k)$. A similar method is to use a penalty function which penalises queues above some value, though this would make the cost function more complicated.

Using the above cost function, the controller must optimise its inputs $u(k+i)$ over a period $i = 0, \dots, N_p - 1$. The full MPC optimisation problem is now given by

$$\begin{aligned}
\min_{u(k),\dots,u(k+N_p-1)} \; & J(k) = \sum_{i=0}^{N_p-1} c^T x(k+i), \\
\text{s.t. } & z(k+i+1) = F(z(k+i), \hat{w}^a(k+i), u(k)) && \forall i = 0, \dots, N_p - 1, \\
& x(k+i) \in \mathbb{N}_0^{n_l} && \forall i = 0, \dots, N_p - 1, \\
& t_c(k+i), t_w(k+i) \in \mathbb{R}^{n_l}_{\geq 0} && \forall i = 0, \dots, N_p - 1, \\
& m(k+i) \in \{0,1\}^{n_l} && \forall i = 0, \dots, N_p - 1, \\
& u(k+i) \in \mathcal{U}(x(k+i), m(k+i), t_c(k+i)) && \forall i = 0, \dots, N_p - 1, \\
& \hat{w}^a(k+i) = F_w(t_w(k+i), \lambda(k)) && \forall i = 0, \dots, N_p - 1,
\end{aligned} \tag{3.13}$$

where $F_w(t_w(k), \lambda(k))$ is given by (3.10). Once the controller has performed this optimisation and provides an input $u(k)$, this discrete-time signal must then be converted to a continuous-time signal. A direct conversion $u(t) = u(k)$ for $t \in [t_k, t_{k+1})$ is not possible. This creates a rectangular signal in continuous-time, which does not match with the previous definition of $u(t)$ which states that departures are instantaneous. In order to ensure that the input provided by the controller is of the form defined in (2.2), $u(t)$ is converted from $u(k)$ as follows:

$$u(t) = \begin{cases} 1 & \text{if } u(k) = 1 \text{ at time } t = t_k, \\ 0 & \text{otherwise.} \end{cases} \tag{3.14}$$

This definition ensures that a vehicle departure always occurs only at the controller's sampling instances, between which $u(t)$ is always zero.

3.5 Summary

This chapter presents a controller design approach to control the hybrid system of Chapter 2. A discrete-time model predictive controller is used, which samples the hybrid system and predicts the behaviour of the system behaviour in order to choose an input which minimises a cost function. The notions of stability and recursive feasibility of an MPC controller are introduced, but unfortunately cannot be guaranteed. Instead, it is assumed that the constraints and the prediction horizon are such that the controller is both stable and recursively feasible.

Next, the hybrid model is discretised in order to determine what the sampled system looks like to the controller. The discretised dynamics are nearly identical to the discrete-time dynamics of the hybrid system. Only vehicle arrivals $w^a(k)$ and departures $u(k)$ cannot be directly sampled from the hybrid system. These are instead counted during the sampling interval. In order to ensure that these signals always lie in the defined flow and jump sets, it must be assumed that both $w_q^a(t, j)$ and $u_q(t, j)$ are only non-zero once between any two sampling instances. The controller uses the discretised model in combination with a model of the disturbance to predict the behaviour of the hybrid system behaviour. The disturbance model predicts vehicle arrivals based on the average vehicle arrival rate λ. If the average arrival rate is unknown, it can be determined from historic data using a moving average window.

The goal of the controller is to minimise the average delay. This is realised in the cost function by integrating the queue lengths over time. Since vehicles only incur delay in the queues, this gives a direct measure of the total delay. Finally, the discrete-time input is converted to continuous-time in such a way as to comply with the initial input definition of (2.2).

In Chapter 4, the hybrid system of Chapter 2 and the controller formulated in this chapter are implemented in Simulink to support numerical simulations.

Chapter 4

Numerical implementation

In Chapter 2, the automated intersection has been modelled as a hybrid system. This system evolves in discrete-time when a vehicle arrives or departs, and otherwise evolves in continuous-time to keep track of the inter-arrival and inter-departure times. The hybrid system is controlled by a discrete-time MPC controller formulated in Chapter 3. This controller acts as the supervisory controller of the intersection, and optimises the vehicle crossing sequence in order to minimise the average delay. The next step is to realise an implementation of both the hybrid system and the controller in order to perform simulations and test the behaviour of the controller behaviour.

Hybrid systems are still a new area of research and as such few tools exist with which they can be simulated, not to mention whether those can model its interaction with a discrete-time controller. In this report, the hybrid system and controller are implemented in Simulink, for which Sanfelice, Copp and Nanez have created the Hybrid Equations (HyEQ) Toolbox [22], and which can cope with differences in sampling times between different parts of a system. Section 4.1 gives the implementation of the hybrid system using the HyEQ toolbox.

The implementation of the controller requires some extra consideration due to the complicated nature of the constraints on vehicle departures. In order to implement the logic which is present in these constraints, the prediction model plus the constraints are formulated as a mixed-logical dynamical (MLD) system. These systems provide a framework by which physical laws, logical rules and operating constraints can be combined. The model predictive controller can then be generated from the MLD system using the Multi-Parametric Toolbox [23]. Section 4.2 presents the full implementation of the controller, including a general explanation of MLD systems.

Other than the implementation of the hybrid system and the controller, it is also necessary to generate vehicle arrivals, which is discussed in Section 4.3. The chapter ends with a summary in Section 4.4.

4.1 Hybrid equations toolbox

The Hybrid Equations (HyEQ) Toolbox is designed by Sanfelice, Copp and Nanez [22] for the simulation of hybrid systems in MATLAB/Simulink. It considers the framework for hybrid systems used in [12, 13], which is the framework used when modelling the intersection in Chapter 2. In this framework, the hybrid system is defined by the flow set $\mathcal{C}$, the jump set $\mathcal{D}$, and the flow and jump map which jointly form the hybrid dynamics. The hybrid simulator of the HyEQ toolbox includes one block for each of these four parts, making it exceptionally easy to create the hybrid system of (2.14) in Simulink. The evolution of the system in hybrid time is computed by an integrator system which computes the continuous-time dynamics when the flow set is active, and triggers a jump to update the state in discrete-time when the jump set is active.

One important quirk of the hybrid simulator is that it keeps track of the simulation time internally. In the integrator system, the integrator receives a constant value of 1 if the flow set is active, and

receives a value of 0 if the jump set is active. This means that the simulation time given by Simulink itself is different from the internal simulation time of the hybrid system. After all, the simulation time of the hybrid system remains constant if the system evolves in discrete-time, but time still passes in Simulink itself. To prevent any mismatch between different subsystems in the simulation, all subsystems must use the simulation time given by the hybrid system. This is especially important when generating vehicle arrivals, which is described in Section 4.3.

4.2 Mixed-logical dynamical systems

In most applications of MPC, the controller can be easily implemented by specifying the state update equations, the state space, and the input set. However, this is not possible with the current formulation of the prediction model and the constraints on vehicle departures, since these are in large part described by logic, which many tools for MPC were not designed for. Furthermore, most of these tools assume a constant input set, and are not designed for an input set which is state-dependent. Therefore, to implement the prediction model and the constraints, these are first re-formulated as a mixed-logical dynamical (MLD) system. The MLD systems framework has been developed by Bemporad and Morari in [24] in order to systematically model and control systems which combine physical laws, logical rules and operating constraints. Any logic in the system is transformed into linear inequality constraints involving integer and continuous variables through the use of propositional calculus described in Chapter 9 of [25]. This creates a system given by linear dynamic equations subject to linear mixed-integer inequalities, i.e. inequalities where the variables can be continuous or binary. This includes continuous and discrete states, inputs and auxiliary variables.

Re-formulating the entire prediction model and the constraints as an MLD system can be a difficult and painstaking task. Thankfully, this task is mostly automated through the HYbrid System DEsctiption Language 3 (HYSDEL3) [26]. HYSDEL3 requires only the state update equations and the auxiliary variables, and constraints can be implemented as conditions which must be fulfilled at all times. Then HYSDEL3 can create a mixed-logical dynamical system of the form

$$\begin{aligned} z(k+1) &= Az(k) + B_u u(k) + B_{\text{aux}} v_{\text{aux},2}(k) + B_{\text{aff}}, \\ y(k) &= Cz(k) + D_u u(k) + D_{\text{aux}} v_{\text{aux},2}(k) + D_{\text{aff}}, \\ E_z z(k) &+ E_u u(k) + E_{\text{aux}} v_{\text{aux},2}(k) \leq E_{\text{aff}}, \end{aligned} \tag{4.1}$$

where $v_{\text{aux},2}(k)$ is an auxiliary vector given by $v_{\text{aux},2}(k) = \begin{bmatrix} v_{\text{aux},1}(k) & \delta(k) \end{bmatrix}^T$, with $\delta(k)$ the auxiliary variables and $v_{\text{aux},1}(k) = \delta(k)z(k)$. All matrices in the system are generated by HYSDEL3 and placed in an m-file, from which the system can be loaded into MATLAB. The matrices $B_{\text{aux}}, D_{\text{aux}}$ and E_{aux} are used to include the effects of the auxiliary variables into the dynamics, while the matrices B_{aff} and D_{aff} allow for affine terms in the dynamics and the output. The inequality constraints that the MLD system is subject to are given through the matrices E_z, E_u, E_{aux} and E_{aff}.

Implementing the discretised dynamics of (3.4) in HYSDEL3 are relatively straightforward. The dynamics of the queue lengths can be implemented directly, and the inter-event timers can be expressed using IF-THEN-ELSE statements. Only the queue activities $m_q(k)$ must be expressed differently in HYSDEL3, because these are binary variables. First an auxiliary variable is defined for each $q \in \mathcal{Q}$ as

$$\left[\delta_{t_{c,q},\max}(k) = 1\right] \iff \left[t_{c,q} \geq \max_{i \in \mathcal{Q}|\{q\}} T^s_{q,i}\right], \tag{4.2}$$

which is used to determine if the maximum service time of a queue has passed. The state update equation of $m_q(k)$ is expressed using notation from boolean algebra. In this notation, the symbol $\neg$ denotes negation, and the variable on the left-hand side is true if the condition on the right-hand side holds. Then the state update equation of $m_q(k)$ for each $q \in \mathcal{Q}$ is given by

$$m_q(k+1) \equiv u_q(k) \vee \left(m_q(k) \wedge \neg\delta_{t_{c,q},\max}(k)\right). \tag{4.3}$$

Following the implementation of the system dynamics, the disturbance model can be implemented as booleans for each $q \in \mathcal{Q}$ given by

$$\left[\hat{w}_q^a(k) = 1\right] \iff \left[t_{w,q}(k) - \lambda_q^{-1}(k) \geq 0\right], \tag{4.4}$$

where $\lambda_q(k)$ must be inverted in Simulink, since HYSDEL3 only accepts affine expressions. Additionally, since all variables in HYSDEL3 must be bounded, $\lambda_q(k)$ must always have some non-zero minimum value, for example $\lambda_{q,min}(k) = \frac{1}{\Delta t N_w}$. Otherwise, the simulation crashes if there are no arrivals in the moving average window.

The implementation of the disturbance model requires an extra constraint on vehicle departures. If a vehicle is predicted to arrive an empty queue, i.e. $\hat{w}_q^a(k) = 1$ and $x_q(k) = 0$, then the controller is able let that vehicle depart from its queue immediately through an input $u_q(k) = 1$ without violating the constraint that the queue lengths must be non-negative. However, in reality, vehicles arrive stochastically, so it is unlikely that there will actually be a vehicle arrival at that time-step. In that case, an input $u_q(k) = 1$ would lead to a negative queue length. To prevent this from occurring, extra auxiliary variables are used to determine if a queue is empty. These are defined for every $q \in \mathcal{Q}$ as

$$\left[\delta_{x_q,0}(k) = 1\right] \iff \left[x_q(k) = 0\right]. \tag{4.5}$$

Then the situation described above is avoided through the constraints

$$\neg(u_q(k) \wedge \hat{w}_q^a(k) \wedge \delta_{x_q,0}(k)), \tag{4.6}$$

for all $q \in \mathcal{Q}$, which ensures that the controller does not let a vehicle depart which, in reality, may not actually exist.

The implementation of the constraints on vehicle departures given by (3.6)–(3.8) require a number of auxiliary binary variables. To implement the constraint given by (3.6), it is first necessary to define one binary variable for each service time. These are given by

$$\left[\delta_{t_c,q_1,q_2}(k) = 1\right] \iff \left[t_{c,q_1}(k) \geq T^s_{q_1,q_2}\right], \tag{4.7}$$

for every $q_1, q_2 \in \mathcal{Q}$. These auxiliary variables can be used in combination with the queue activity $m(k)$ to define another set of variables which indicate that a vehicle may depart from a queue. These variables are given by

$$\delta_{u_{q_2}}(k) \equiv \neg\left(\neg\delta_{t_c,1,q_2}(k) \wedge m_1(k)\right) \wedge \cdots \wedge \neg\left(\neg\delta_{t_c,n_l,q_2}(k) \wedge m_{n_l}(k)\right), \tag{4.8}$$

for all $q_2 \in \mathcal{Q}$. The boolean $\delta_{u_{q_2}}$ is true if either no queues are active, or if the service times of all active queues with respect to queue q_2 have passed. Then the constraint on vehicle departures is implemented for every $q_2 \in \mathcal{Q}$ as

$$\left[\delta_{u_{q_2}}(k) = 1\right] \Longleftarrow \left[u_{q_2}(k) = 1\right]. \tag{4.9}$$

The one-way implication ensures that the input must be zero if $\delta_{u_{q_2}} = 0$, and that the input can be either zero or one if $\delta_{u_{q_2}}(k) = 1$. This perfectly captures the constraint of (3.6).

The implementation of (3.7) requires another set of binary variables, this time to determine whether two trajectories intersect or not. These are given by

$$\left[\delta_{T^s_{q,i}} = 1\right] \iff \left[T^s_{q,i} = 0\right], \tag{4.10}$$

for all $q, i \in \mathcal{Q}$ such that $i > q$. They only need to be defined for $i > q$ since the service times $T^s_{q,i} = T^s_{i,q} = 0$ if two trajectories do not intersect, thus halving the number of constraints. The constraint of 3.7 is then implemented in HYSDEL3 for all $q, i \in \mathcal{Q}$ such that $i > q$ as

$$\neg(u_q(k) \wedge u_i(k) \wedge \neg\delta_{T^s_{q,i}}), \tag{4.11}$$

which is always fulfilled unless both $u_q(k)$ and $u_i(k)$ are true, but $\delta_{T^s_{q,i}}$ is false. Finally, the fairness constraints given in (3.8) can be implemented directly without the use of auxiliary variables.

After HYSDEL3 has converted the prediction model, the disturbance model and the constraint into an MLD system, this system can then be used to design the MPC controller. This task is also automated using the Multi-Parametric Toolbox 3 (MPT) [23]. This is an open source MATLAB-based toolbox which is capable of directly generating an MPC controller for an MLD system created by HYSDEL3. MPT only requires the weights of the linear or quadratic cost function and the length of the prediction horizon N_p to create the intersection controller. Once the MPC controller has been generated, it can be evaluated for any value of the states. The controller can then be implemented in Simulink as an interpreted MATLAB function, and is evaluated for the current state of the intersection.

4.3 Vehicle arrivals

So far, the hybrid system and the controller have been implemented, but the simulation also requires vehicle arrivals to the hybrid system as an external disturbance. As mentioned in Section 3.3, vehicles can be assumed to arrive according to Poisson distributed inter-arrival times in the case of an isolated intersection. However, for the purposes of interpreting the simulation results, deterministic vehicle arrivals are more desirable, even if this is not as realistic as stochastic arrivals. Deterministic inter-arrival times are implemented first, after which implementing stochastic arrivals requires only minor adjustments.

To generate vehicle arrivals with the right inter-arrival times, it necessary to know the current simulation time. As explained in Section 4.1, the hybrid system uses its own internal simulation time, and therefore this simulation time must be used instead of Simulink's simulation time. Vehicle arrivals are most easily generated using a MATLAB function block. To implement deterministic inter-arrival times, the arrival time of the last vehicle is remembered, and a new vehicle is generated when the inter-arrival time has passed. Then the current simulation time becomes the arrival time of the last vehicle, and another vehicle is generated when the inter-arrival times passes again, and so on.

The same scheme which is used to generate deterministic vehicle arrivals can also be used to generate vehicle arrivals stochastically. The only adjustment needed is that a new inter-arrival time must be calculated after every arrival. In a Poisson process the probability P that there is an arrival at queue q in the interval $[0, t]$ is given by

$$P(X_q \leq t) = 1 - \exp^{-\lambda_q t}, \tag{4.12}$$

where X_q is the inter-arrival time and $P(X_q \leq t)$ gives the probability of X_q being less than or equal to the time since the last arrival t. By choosing $P(X_q \leq t)$ as a uniform random number using the `rand()` function in MATLAB, Poisson distributed inter-arrival times can be calculated by inverting the above function. This gives

$$X_q = -\frac{1}{\lambda_q} \ln\left(1 - \texttt{rand}()\right). \tag{4.13}$$

By calculating a new inter-arrival time according to this equation after every arrival, vehicles arrive at the intersection according to a Poisson distribution.

4.4 Summary

This chapter describes the methods by which the hybrid system and the controller are implemented in Simulink, and by which vehicle arrivals are generated. The hybrid system is easily implemented using the Hybrid Equations Toolbox which requires only the hybrid dynamics and the flow and jump sets. An important point of this toolbox is that the hybrid system keeps track of the simulation time internally, and that this time is not equal to Simulink's own simulation time.

The controller is implemented by first converting the discretised model into a mixed-logical dynamical system using HYSDEL3, which is able to combine the system dynamics with the logical constraints

defined by the service times. The constraints are applied to the MLD system through the use of auxiliary binary variables. The model predictive controller is then created with the Multi-Parametric Toolbox, and evaluated in Simulink as an interpreted MATLAB function.

Finally, vehicle arrivals are generated with a MATLAB function block. This block remembers the arrival time of the previous vehicle and generates a new vehicle when the inter-arrival time has passed. Vehicles can arrive stochastically by calculating a new Poisson distributed inter-arrival time whenever a vehicle departs.

Chapter 5

Simulation results

The implementation of the hybrid system, the controller and the method to generate vehicle arrivals in Simulink have been presented in the previous chapter. In this chapter, the Simulink model is used to perform simulations in order to observe the behaviour of the system and the controller. Before presenting these results, some theoretical results in queueing theory on two-queue servers are given in Section 5.1. These results give some indication of what sort of behaviour to expect in simulations of a two-queue intersection, and as such most simulations use such a model. The simulation results are presented and discussed in Section 5.2, where various aspects such as the prediction horizon, cost function and the service times are varied to see what effect these have on the behaviour of the system. This section also presents a comparison between the performance of the designed supervisory controller, a first-come-first-served crossing sequence and a vehicle actuated traffic light. Finally, the chapter is summarised in Section 5.3.

5.1 Results from queueing theory

Before presenting the simulation results, it is helpful to first introduce some relevant results from queueing theory [27]. This gives an indication of what behaviour to expect from the controller and provides a basis for comparison for the simulation results. In general, in queueing theory every queue $q \in \mathcal{Q}$ has a continuous arrival rate λ_q of vehicles and the server serves a queue with a service rate μ_q. The time between the last served vehicle from some queue q_1 and the next served vehicle from a different queue q_2 is given by the set-up time $\sigma_{q_1,q_2} > 0$. Applying this to the description of the service times in (2.1), the service rates and set-up times are given by

$$\begin{aligned} \mu_{q_1} &= \frac{1}{T^s_{q_1,q_1}}, \\ \sigma_{q_1,q_2} &= T^s_{q_1,q_2}, \end{aligned} \tag{5.1}$$

for $q_1, q_2 \in \mathcal{Q}$. Another important concept in queueing theory is the utilisation ρ_q of the server by a queue. The utilisation indicates whether a queue is stable or not and is given by

$$\rho_q = \frac{\lambda_q}{\mu_q}. \tag{5.2}$$

If $\rho_q > 1$, it is impossible to empty a queue, since vehicles arrive faster than they can be served. For a system of n_l conflicting queues to be stable as a whole, it must hold that $\sum_{q\in\mathcal{Q}} \rho_q < 1$, in which case a stable process cycle exists and the queue lengths are bounded. However, the process cycle varies over time in the case that vehicle inter-arrival times are stochastic. Vehicles can arrive sporadically or in quick succession, the latter of which can be a major cause of delay. To give an indication of the increased delay in the case of stochastic arrivals, the average queue length can be calculated. For a single-queue server with deterministic service rates and Poisson distributed arrivals, the average number of vehicles in the queue is given by

$$L = \frac{1}{2}\frac{\rho^2}{1-\rho}. \tag{5.3}$$

This equation holds for a single-queue server, not for a multi-queue server, and therefore does not take into account the set-up times. However, (5.3) is accurate as long as the set-up times are short and the service rates are equal.

In [28], the optimal periodic behaviour of a two-queue switching server is studied. In Section 5.2 of [28], the goal is to achieve a switching policy which minimises the costs related to the queue lengths starting from any initial state. There, a cost function given by

$$J = \limsup_{t\to\infty} \frac{1}{t} \int_0^t \left[g_1(x_1(s)) + g_2(x_2(s))\right] ds, \tag{5.4}$$

is used, with g_i functions that relate costs to queue lengths x_i for $i \in \{1, 2\}$. Later this is relaxed to linear costs on the queue lengths $J = \frac{1}{T}\int_0^T \left[c_1 x_1(s) + c_2 x_2(s)\right] ds$, similar to (3.12). Without loss of generality, it is assumed that $c_1\lambda_1 \geq c_2\lambda_2$, with c_1, c_2 the weighting coefficients, for which [28] shows that the optimal steady-state process cycle is of the form shown in Figure 5.1.

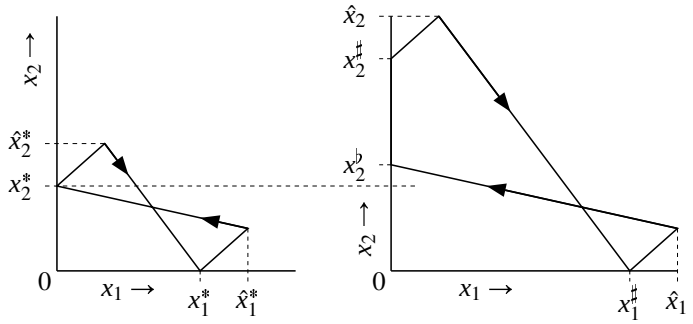

Figure 5.1: *The general form of the optimal process cycle for a switching server with cost function* (5.4). *The left figure shows the pure bow tie, while the right shows the truncated bow tie curve [28].*

In the left graph of Figure 5.1, each queue is emptied in turn before immediately switching, which is referred to as a clearing policy. In the right figure, it can be seen that the server processes vehicles in queue 1 at a rate equal to the arrival rate after queue 1 is emptied, which [28] calls the slow mode. The slow mode occurs when it is advantageous for the server to let queue 2 build up before switching, and is only present if

$$c_1\lambda_1(\rho_1 + \rho_2) - (c_1\lambda_1 - c_2\lambda_2)(1 - \rho_2) < 0. \tag{5.5}$$

The length of the slow mode is equal to $\alpha_1\sigma$, where $\sigma = \sigma_{1,2} + \sigma_{2,1}$ and α_1 is given by

$$\begin{aligned}&\left[c_1\lambda_1\rho_2^2(1-\rho_1) + c_2\lambda_2(1-\rho_1)^2(1-\rho_2)\right]\alpha_1^2 + 2\left[c_1\lambda_1\rho_2^2 + c_2\lambda_2(1-\rho_1)(1-\rho_2)\right]\alpha_1 \\ &\quad + \left[c_1\lambda_1(\rho_1+\rho_2) - (c_1\lambda_1 - c_2\lambda_2)(1-\rho_2)\right] = 0,\end{aligned} \tag{5.6}$$

$$\alpha_1 = \begin{cases} 0 \text{ if } c_1\lambda_1(\rho_1+\rho_2) - (c_1\lambda_1 - c_2\lambda_2)(1-\rho_2) \geq 0 \\ \text{positive real root of (5.6) otherwise.} \end{cases} \tag{5.7}$$

The extrema of the queue lengths $x_1^\sharp, \hat{x}_1, x_2^\flat, x_2^\sharp$ and $\hat{x}_2$ in the optimal process cycle are given in [28]. The slow mode is particularly interesting for the simulations, since it means that the controller has to wait for a vehicle to arrive at a queue even when it can already switch queues.

A different thesis on a similar topic is that of [29], where the control of a switching server is decoupled into an optimal periodic behaviour problem and an optimal transient behaviour problem. When optimising the periodic behaviour, [29] aims to minimise a linear combination of the queue lengths over the length of one or more cycles. Most importantly, [29] also aims to optimise the cycle length, which makes the optimisation horizon variable. The optimisation of the transient behaviour is similar. There, [29] aims to steer the system from any initial condition to the optimal steady-state trajectory with minimal costs during the transient. This is achieved by optimising the deviation of the costs to the steady-state cycle over a period of one or more cycles. Just as in the optimisation

of the periodic behaviour, the cycle length is variable, and therefore the prediction horizon is not fixed.

The optimal transient behaviour resulting from this optimisation is simple in principle. First, the optimal steady-state trajectory must contain a slow mode in order for the transient trajectory to be able to converge to the optimal steady-state cycle in finite time. Otherwise, the system can reach a sub-optimal steady-state cycle and remain unable to reach the optimal cycle since it does not have any free time in which it can process the extra vehicles in the system. If individual vehicle arrivals and departures are considered, this can be seen as a phase shift between the (deterministic) arrivals and the control input. This phase can be shifted by varying the length of the slow mode, but creates irreducible delay if there is no slow mode. It is also possible that the optimal steady-state cycle cannot be reached due to the sampling time of the controller, even if there is a slow mode.

Under the assumption that queue 1 is always emptied when served, there exist two distinct cases for the optimal transient trajectory. In the first case, a clearing policy is optimal, where each queue is emptied followed by immediately switching to empty the other queue. This is repeated until the optimal steady-state cycle is reached. In the second case, if $c_1\lambda_1 > c_2\lambda_2$ and queue 2 is being served, a trade-off exists between letting queue 1 build up, which is expensive, and switching to serve queue 1 before queue 2 is emptied. Thus, the optimal transient trajectory is such that the server switches from queue 2 to queue 1 without emptying queue 2. In the simulations, the initial conditions will be chosen such that not emptying queue 2 is optimal, and see if the controller follows this behaviour.

Note that while the results by [28] and [29] are useful as a tool to explain behaviour in the simulations, the results of the simulations in the next section are not necessarily directly comparable to those of [28] and [29]. The optimal behaviour in Figure 5.1 is derived for a cost function with infinite horizon and where the behaviour is periodic. In contrast, the simulations in Section 5.2 use a finite horizon and are focused on transient behaviour, and the resulting behaviour may not necessarily be periodic. Secondly, [29] looks at transient behaviour for a finite but varying horizon and uses a different cost function, which may also lead to different results.

5.2 Simulation results and discussion

The hybrid system and the controller are defined through multiple aspects, such as the arrival rates, the service times, the cost function and the prediction horizon. Varying these parameters can lead to significantly different behaviour, which is explored in this section. In order to compare the results of the simulations with the theoretical results presented in the previous section, most simulations are performed with only two queues. At the end of this section, the performance of the model predictive controller is compared with the performance of a first-come-first-serve policy and with the performance of a vehicle actuated traffic light. All simulations are performed without the fairness constraints unless mentioned otherwise. Depending on the intersection parameters and the initial conditions, the optimal transient behaviour or the steady-state cycle can violate these constraints, leading to different behaviour. The effect of the fairness constraints is investigated separately in Section 5.2.3.

5.2.1 Simple intersection

First, consider the two-queue system given by

$$\begin{aligned} c &= \begin{bmatrix} 1 & 1 \end{bmatrix}, \\ \lambda &= \begin{bmatrix} \frac{1}{2} & \frac{1}{5} \end{bmatrix}^T, \\ T^s &= \begin{bmatrix} 1.4 & 1.8 \\ 1.8 & 1.4 \end{bmatrix}, \end{aligned} \tag{5.8}$$

for which $\rho_1 + \rho_2 = 0.98 < 1$ and therefore a stable process cycle exists. Figure 5.2 shows a schematic depiction of the physical intersection. The controller samples the system with a sampling time of $\Delta t = 0.1$ seconds and uses a prediction horizon of length $N_p = 20$. Note that this prediction horizon

is very short, and that there are only two vehicle arrivals or departures within the horizon at most. Additionally, the condition in (5.5) does not hold for this system, and therefore there is no slow mode in the steady-state cycle according to the results in [28]. Because a clearing policy is optimal, controlling this intersection through MPC is somewhat unnecessary. However, the goal of these simulations is to see if the controller behaves as expected according to [28] and [29] in a simple setting. The benefit of using MPC is shown in Section 5.2.5.

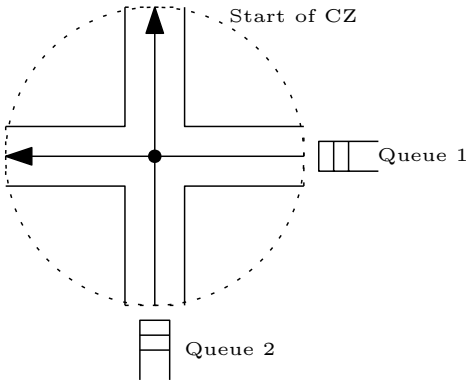

Figure 5.2: *A schematic view of the two-queue intersection.*

According to [28], both queues reach a maximum length of 6 vehicles. Figure 5.3 shows the results of two simulations of this system with deterministic vehicle inter-arrival times. In the left figure, all initial conditions are zero. In the right figure, the initial queue lengths are $x(t_0, j_0) = \begin{bmatrix} 5 & 10 \end{bmatrix}^T$ and queue 1 is initially active. Figure 5.4 shows the steady-state cycle of the same simulations more clearly in a zoomed view.

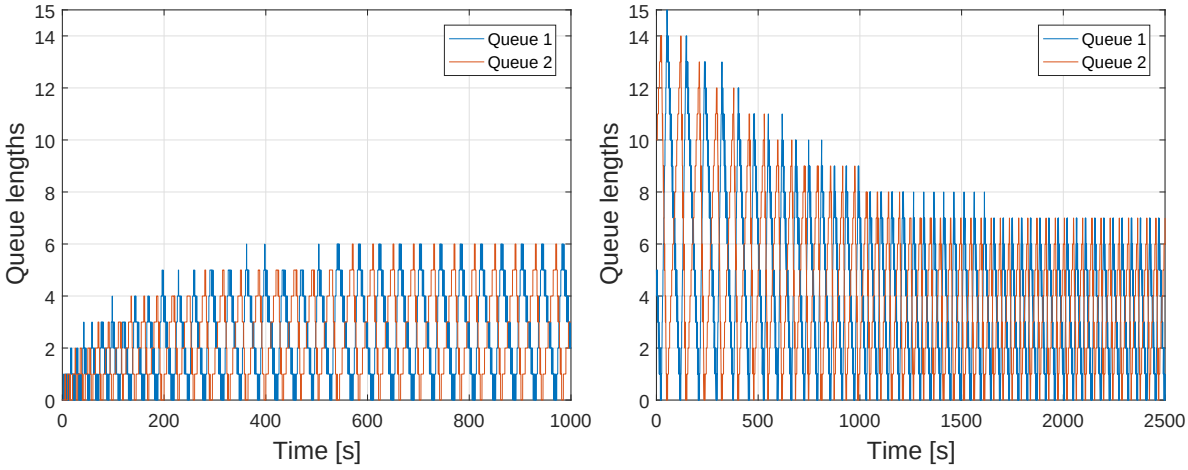

Figure 5.3: *Two simulations of the system given in* (5.8) *for initial conditions zero (left) and initial queue lengths* $x(t_0, j_0) = [5\ 10]^T$ *with queue 1 active (right).*

The figures above show that for both initial conditions the controller always uses a clearing policy. The steady-state cycle is reached after about 600 seconds in the first simulation, for which the maximum queue length is 6 vehicles for both queues. These results are entirely consistent with the results in [28]. In the case of non-zero initial queue lengths in the right figure, the steady-state cycle is reached after more than 1500 seconds, with a maximum queue length of 7 vehicles. It takes the system so long to reach the steady-state cycle because the server utilisation is very high, and the optimal steady-state cycle is not reached in finite time because there is no slow mode. As explained in Section 5.1, there can be a phase shift between the vehicle arrivals and departures in the steady-state cycle which causes delay. In the left figure, a vehicle arrives at queue 1 at $t_{c,1}(t, j) = 1.5$ after queue 1 has been emptied, at which point that vehicle can immediately depart. In the right figure, this occurs at $t_{c,1}(t, j) = 1.7$, which causes the server to waste some extra time. As there is no slow mode, there is no chance for the server to better synchronise its departures with the vehicle arrivals.

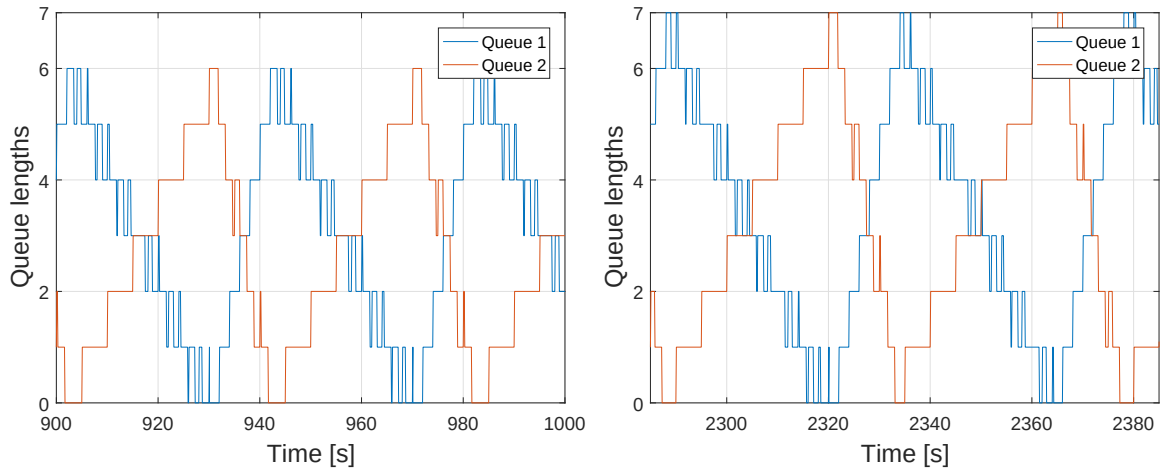

Figure 5.4: *The steady-state cycles of Figure 5.3 over a period of 100 seconds.*

By choosing the initial conditions differently, the phase shift between vehicle arrivals and departures in the steady-state cycle can be varied. This can increase or decrease the average cost in the steady-state cycle compared to the results of Figure 5.3. To show this, Figure 5.5 presents a simulation of system (5.8) with the same initial conditions as in the right figure of Figure 5.3, i.e., initial queue lengths $x(t_0, j_0) = \begin{bmatrix} 5 & 10 \end{bmatrix}^T$ and queue 1 active, with the exception that the initial inter-departure times are now $t_c(t_0, j_0) = \begin{bmatrix} 1.2 & 0 \end{bmatrix}^T$.

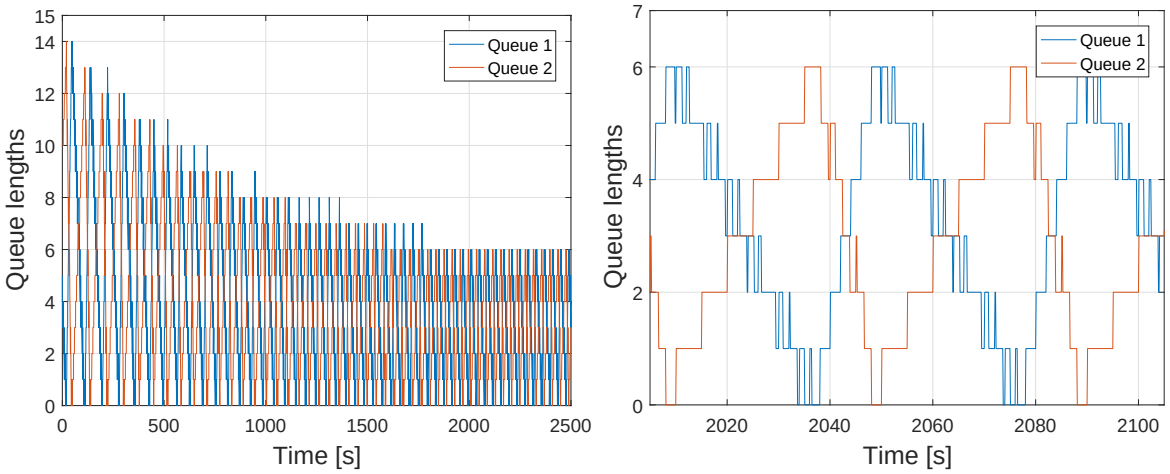

Figure 5.5: *A simulation of the system given in* (5.8) *with initial queue lengths* $x(t_0, j_0) = [5\ 10]^T$*, inter-departure times* $t_c(t_0, j_0) = [1.2\ 0]^T$ *and queue 1 active.*

Figure 5.5 clearly shows that the optimal steady-state cycle can be reached from a non-zero initial state, and that the phase between arrivals and departures can be shifted..

5.2.2 Stochastic vehicle arrivals

All three simulations performed so far have used deterministic vehicle inter-arrival times. In a more realistic setting, the inter-arrival times are stochastic, which leads to different behaviour. To show this behaviour, a simulation of the system given in (5.8) is performed with the same controller parameters as before, i.e. $\Delta t = 0.1$ seconds and $N_p = 20$, but with Poisson distributed vehicle inter-arrival times given in (4.13) with λ_q given by the moving average window. The results are shown in Figure 5.6.

The figure shows that the randomness in the vehicle arrivals leads to far longer queues than in earlier simulations, and thereby longer delays. According to (5.3), the total number of vehicles in the queues

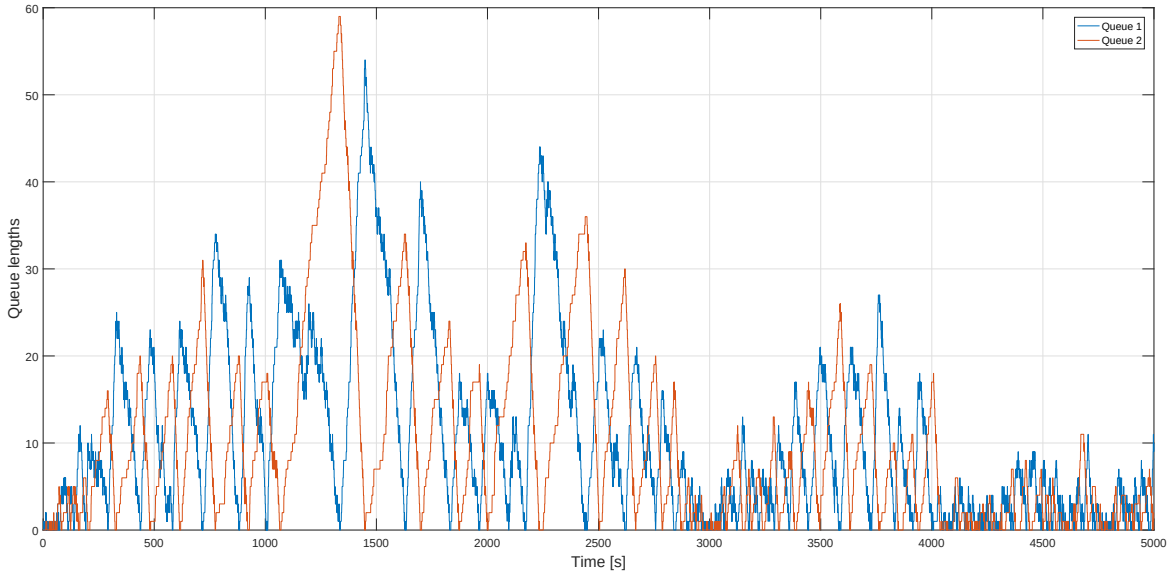

Figure 5.6: *A simulation of the system given in* (5.8) *with stochastic vehicle inter-arrival times.*

is 24 on average. The actual average over the entire simulation is only 21 vehicles. It is likely that the current average is simply due to randomness, and a longer simulation yields a greater average queue length.

Secondly, satisfying the fairness constraints of (3.8) is difficult if vehicle arrivals are stochastic. If there is a period in which vehicles arrive in quick succession, such as around $t = 1500$, then switching queues when $t_{c,q}(t, j)$ reaches a maximum is possible, but limiting the queue lengths at the same time is not, making the optimisation problem infeasible.

It has been mentioned previously that deterministic vehicle arrivals are more desirable for interpreting the simulation results, which is made clear by the above figure. While the server still uses a clearing policy, there is no steady-state cycle present, and the maxima of the queue lengths vary every cycle. In order to better interpret the system behaviour, all other simulation results discussed below use deterministic inter-arrival times.

5.2.3 Cost and fairness

In some intersections, it may be preferable to give a higher priority to one queue. For example, if one queue is on a lane which leads to an important destination, or if the throughput of a certain lane should be higher, then those queues are given priority. This is easily achieved by increasing the cost of that queue. Thus, the system given in (5.8) is modified by doubling c_1 to obtain

$$\begin{aligned} c &= \begin{bmatrix} 2 & 1 \end{bmatrix}, \\ \lambda &= \begin{bmatrix} \frac{1}{2} & \frac{1}{5} \end{bmatrix}^T, \\ T^s &= \begin{bmatrix} 1.4 & 1.8 \\ 1.8 & 1.4 \end{bmatrix}. \end{aligned} \tag{5.9}$$

Figure 5.7 shows the results of three simulations of this system with controller parameters $\Delta t = 0.1$ seconds and $N_p = 20$ in order to test the effectiveness of the fairness constraints. The top-left figure shows the results if none of the fairness constraints are applied. In the top-right figure, only the fairness constraint of (3.8) which restricts $t_{c,q}(t, j)$ to a maximum $T_{q,\max}$ is applied with $T_{q,\max} = 40$ for both queues. This forces the controller to serve each queue every 40 seconds at the least. In the bottom figure, both fairness constraints are applied. The constraint on $t_c(t, j)$ remains, and both queues are restricted to a maximum length of $X_{q,\max} = 20$ vehicles. In all three cases, the initial queue lengths are $x(t_0, j_0) = \begin{bmatrix} 5 & 10 \end{bmatrix}^T$, and the other initial states are zero.

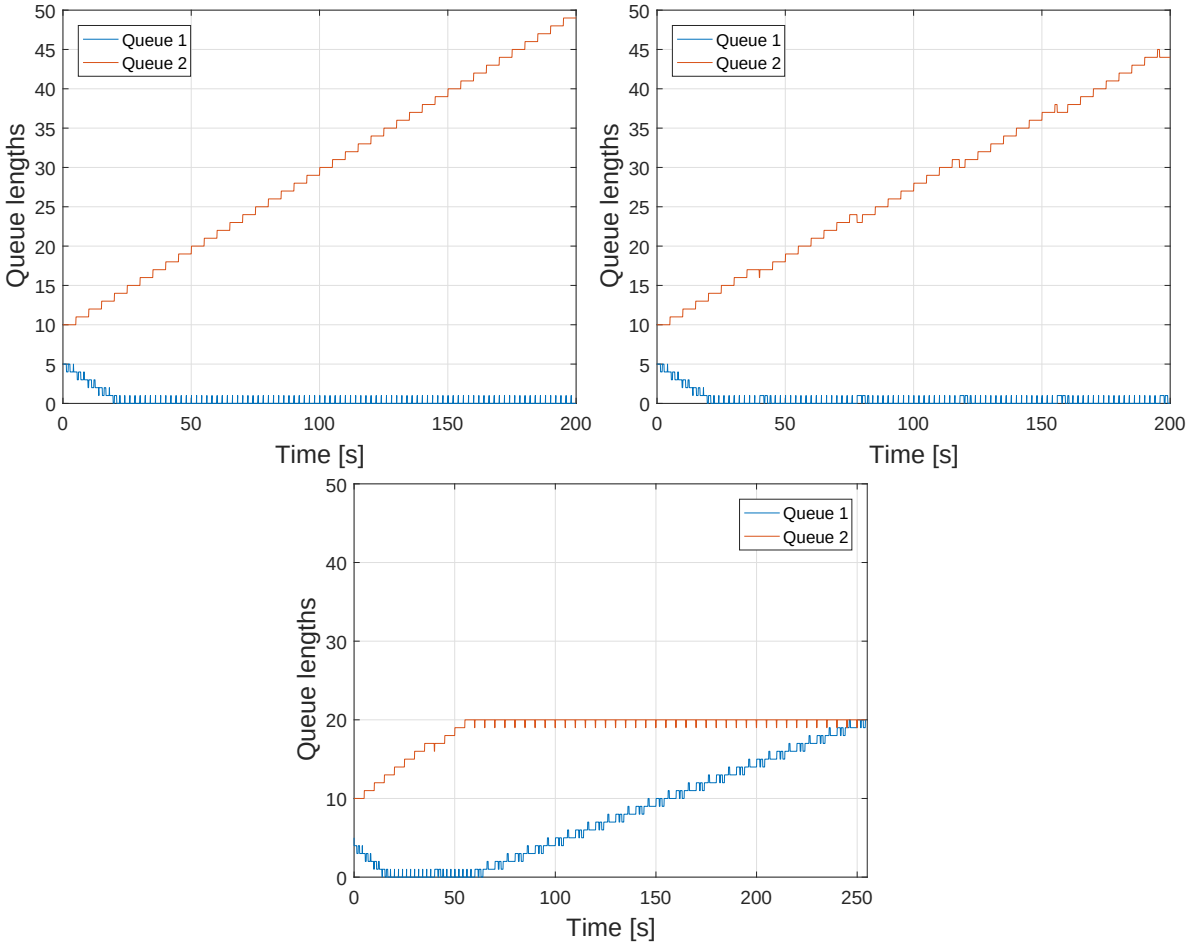

Figure 5.7: *Three simulations of the system given in Equation* (5.9)*, where* $c_1 = 2$*. The top-left figure shows the results without the fairness constraints, whereas one or both fairness constraints are implemented in the other figures. The initial queue lengths are* $x(t_0, j_0) = [5\ 10]^T$

As observed in these figures, increasing the cost of queue 1 leads to unstable behaviour of the MPC-controlled system. In the top-left figure, the server never serves queue 2 at all, and the top-right figure shows that the fairness constraint on $t_{c,q}(t, j)$ is only effective to a limited extent. Whenever $t_{c,2}(t, j)$ reaches $T_{q,\max}$, the server serves a single vehicle from queue 2, then immediately resumes serving queue 1. In the bottom figure, the length of queue 2 is kept below the maximum $X_{2,\max}$, but the server still frequently switches to queue 1, causing queue 1 to become longer. After 250 seconds, the length of queue 1 also reaches $X_{1,\max}$ and the optimisation problem becomes infeasible.

The reason that the server so strongly neglects serving queue 2 is simple; serving the expensive queue 1 is more cost-efficient within the prediction horizon. Solely serving vehicles arriving to queue 1 as they arrive leads to a cost increase of $c_2\lambda_2 = 0.2$ per second. In contrast, serving vehicles in queue 2 leads to a cost increase of $c_1\lambda_1 + c_2(\lambda_2 - \mu_2) = 1 + (\frac{1}{5} - \frac{1}{1.4}) = 0.486$ per second. Hence, serving only queue 1 leads to the slowest increase in cost within the horizon.

Obviously, the behaviour shown in Figure 5.7 does not lead to minimisation of the total cost in the long term, even if serving only queue 1 leads to a lower cost increase per second. The system should reach some steady-state cycle, but the MPC controller is not able to find such a cycle, unlike in the results of [28, 29]. One possibility is that the prediction horizon is too short. The length of the prediction horizon is $N_p\Delta t = 2$ seconds, within which the controller can let at most two vehicles depart. Given the high value of $c_1\lambda_1$ compared to $c_2\lambda_2$, switching to queue 2 is suboptimal within the prediction horizon. Increasing the prediction horizon may make serving both queues more beneficial for the controller. Another possibility is that using a linear cost function is a bad decision in this case. A linear cost function weighs each vehicle in a queue equally, and hence long queues are acceptable. In

order to make the controller more fair for individual vehicles, a quadratic cost function could be used. This could prevent queues from becoming too long, since these weigh far more than short queues. If both methods are insufficient, it may also be possible to change the MPC formulation to one similar to that of [29] which does achieve a steady-state, for example by applying a terminal constraint.

Two simulations have been performed to try to solve the instability seen in Figure 5.7 through other methods than the fairness constraints. In one simulation, the prediction horizon is increased to $N_p = 50$ and the sampling time is increased to $\Delta t = 0.2$. This increases the duration of the prediction horizon to $N_p \Delta t = 10$ seconds, up from 2 seconds in earlier simulations. In the second simulation, the duration of the prediction horizon is still $N_p \Delta t = 2$, but a quadratic cost function given by

$$J(k) = \sum_{i=0}^{N_p - 1} c_1 x_1^2(k+i) + c_2 x_2^2(k+i), \tag{5.10}$$

is used, where $c_1 = 2$ and $c_2 = 1$. None of the fairness constraints are applied. The results of these two simulations are shown in Figure 5.8.

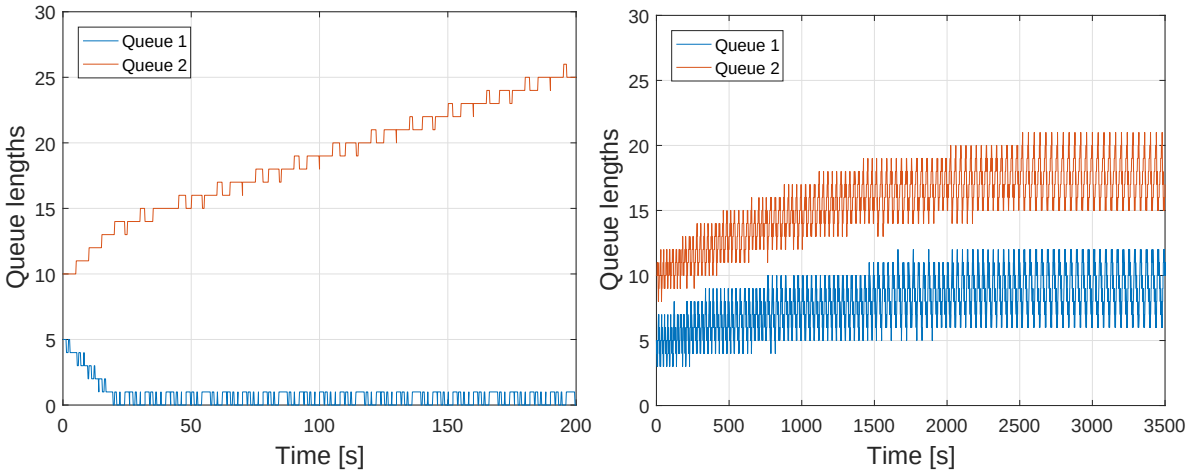

Figure 5.8: *Two simulations of the system given in* (5.9) *to test other methods of implementing fairness. The left figure shows the results if the prediction horizon is increased to* $N_p = 50$ *and* $\Delta t = 0.2$. *The right figure shows the results if the cost function is quadratic in the queue lengths.*

The left-hand figure shows that increasing the duration of the prediction achieves the desired effect, but not to the desired extent since the length of queue 2 still increases to infinity. The server now serves queue 2 more often, but still prefers serving queue 1. It is possible that further increasing the duration of prediction horizon $N_p \Delta t$ could let the controller achieve a cycle as $N_p \Delta t$ approaches the length of the optimal cycle. Unfortunately this becomes far too computationally expensive. The right-hand figure shows that the controller with a quadratic cost function achieves a stable cycle. The length of queue 1 varies between 6 and 12 vehicles, while the length of queue 2 varies between 15 and 21 vehicles. Note that the server no longer uses a clearing policy because the cost of the first five vehicles in queue 1 is very low compared to the cost of the 22nd vehicle in queue 2.

Compared to an increase in the prediction horizon, Figure 5.8 shows that using a quadratic cost function is a simple and effective way to make the controller fair from the driver's perspective if costs between the queues are unbalanced, since all vehicles are guaranteed to be served in finite time. Because of this, it can be argued that it is always better to use quadratic costs instead of linear costs, even if the queues have equal weight. To investigate this, a simulation has been performed on the system given in (5.8), with $c_1 = c_2$, but with a quadratic cost function as given in (5.10). The sampling time and prediction horizon are $\Delta t = 0.1$ seconds and $N_p = 20$. The results are presented in Figure 5.9.

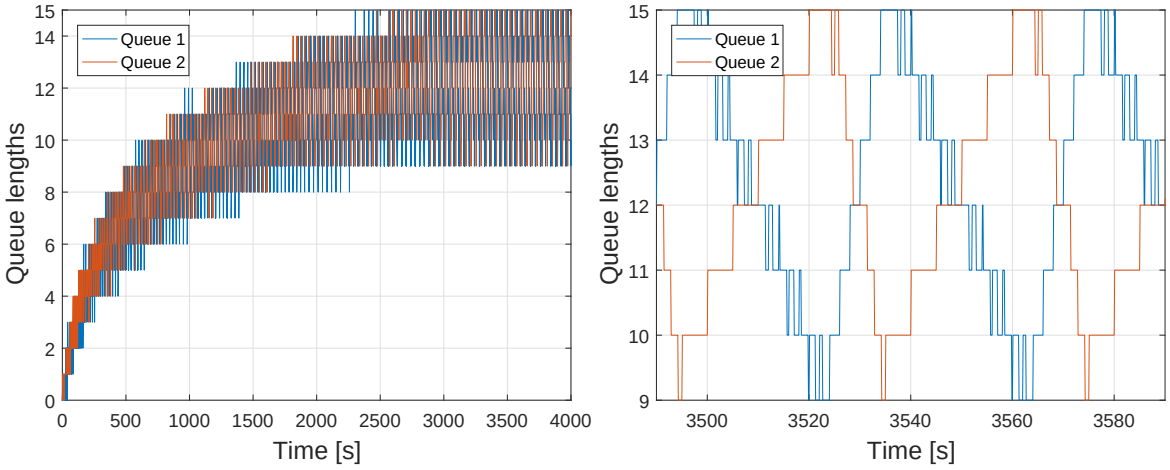

Figure 5.9: *A simulation of the system given in* (5.8) *with quadratic costs.*

The figure shows results similar to those in Figure 5.8 in that the server does not use a clearing policy. Both queues now vary between 9 and 15 vehicles in length. Such results are always observed if the cost function is quadratic, because the relative costs between two queues become smaller as the queues become longer. For example, a queue with six vehicles in it has 36 times the cost of a queue with one vehicle in it. On the other hand, a queue with 15 vehicles in it is only 2.8 times as expensive as a queue with 9 vehicles. Of course, the steady-state cycle varies with the system parameters, but is always of such a form.

Unfortunately, longer queues leads to higher costs and longer delays, and therefore using quadratic costs leads to worse performance compared to the linear costs. Therefore, it is recommended to only implement a quadratic cost function to achieve fairness if using a linear cost function leads to unstable behaviour. It is also possible to achieve fairness by implementing the quadratic costs as a penalty function for long queues. If the queues are short, the costs remain linear.

5.2.4 Slow mode

So far, all of the previous simulations have been performed on systems based on the system given in Equation (5.8). For such a system, a clearing policy is optimal under the assumptions that the cost is linear in the queue lengths and that there are no fairness constraints. But a clearing policy is not optimal for all systems. As mentioned in Section 5.1, the optimal steady-state cycle contains a slow mode according to [28] if (5.5) holds. Now, consider the system given by

$$\begin{aligned} c &= \begin{bmatrix} 1 & 1 \end{bmatrix}, \\ \lambda &= \begin{bmatrix} 2 & \frac{1}{5} \end{bmatrix}^T, \\ T^s &= \begin{bmatrix} 0.2 & 1.2 \\ 1.2 & 0.4 \end{bmatrix}. \end{aligned} \tag{5.11}$$

The physical layout of the intersection is the same as before, and is depicted in Figure 5.2. For this system, (5.5) holds, and according to (5.6) the slow mode is 4.4 seconds long, during which the server lets queue 2 build up to a length of 2 vehicles. According to [28], it is expected that the queue lengths reach maximum values of $\hat{x} = \begin{bmatrix} 7 & 2 \end{bmatrix}^T$. Figure 5.10 shows the results of two simulations for initial conditions zero and for initial queue lengths $x(t_0, j_0) = \begin{bmatrix} 8 & 15 \end{bmatrix}^T$. In both cases, the prediction horizon is $N_p = 40$ and the sampling time is $\Delta t = 0.2$.

In the left-hand figure it is clear that the system quickly reaches a steady-state cycle. As expected, the figure shows that this cycle contains a slow mode, where the server lets queue 2 build. However, contrary to expectations the slow mode is nearly 20 seconds long during which queue 2 builds up to a length of 4 vehicles. Then, while the controller is serving queue 2, queue 1 builds up to a length of 8 vehicles, where the 8th vehicle is allowed to depart as soon as it arrives, and is therefore not shown

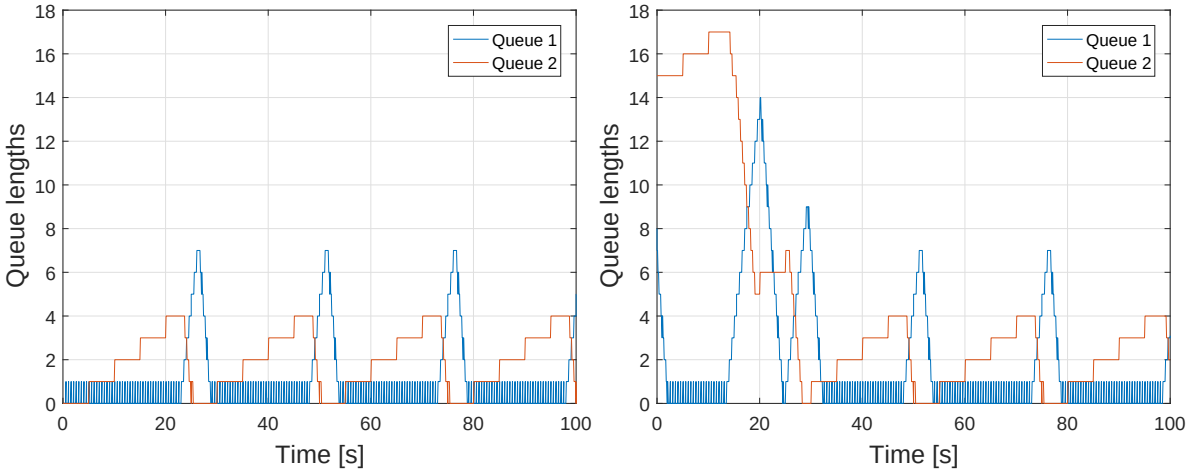

Figure 5.10: *Two simulations of the system given in* (5.11) *for different initial conditions.*

in the figures. Clearly, the system does not follow the optimal steady-state trajectory predicted by [28]. This is likely caused by the finite prediction horizon. In [28], an optimal steady-state cycle is determined over an infinite horizon, which may lead to different results. Additionally, other simulations of the same system with a shorter prediction horizon have shown unstable behaviour similar to that in Figure 5.7. This supports the idea that if the prediction horizon is increased further in those simulations, a steady-state cycle could be reached. It also raises the possibility that the system of (5.11) follows the optimal steady-state cycle predicted by [28] if the prediction horizon is increased further.

The right-hand figure shows that the system reaches the same steady-state cycle as in the left-hand figure, even when starting from different initial queue lengths. This matches with the statement in [29] that the transient trajectory can only reach the optimal steady-state trajectory in finite time if there is a slow mode, which was not the case in Figure 5.3. Moreover, the transient trajectory shows that queue 1 is always emptied, but that queue 2 is emptied over two service periods, which also matches the transient behaviour described by [29].

5.2.5 Comparison with other control strategies

Now that the behaviour of different systems for various controller parameters has been observed, what remains is to judge the performance of the model predictive controller. The controller has been designed to minimise the average delay of the vehicles in the queues, but it is still unknown how it compares to other traffic control strategies. To test the performance of the model predictive controller, it is compared to two other controllers: a first-come-first-serve (FCFS) policy used in Morales' paper [10], and a vehicle actuated traffic light based on data from a real-life intersection. The data used is that of intersection S4 listed in [30], which is a T-junction with five queues (bicycle lanes are ignored). A schematic depiction of this intersection is shown in Figure 5.11.

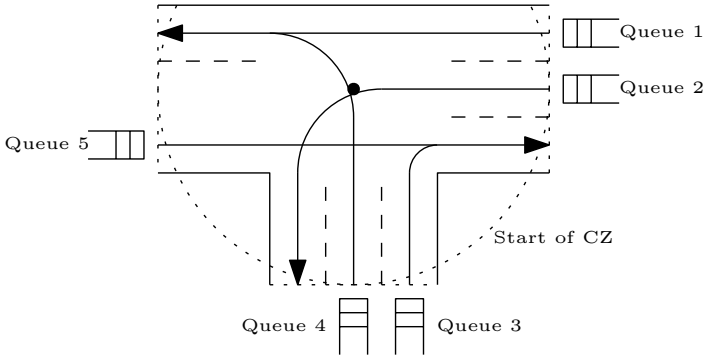

Figure 5.11: *A schematic depiction of intersection S4 of [30].*

This intersection has more than two queues, and therefore it is not possible to predict the optimal behaviour for this intersection using the results presented in Section 5.1. Instead, the focus lies purely on comparing the performance of the three control methods. All controllers are applied to the five-queue intersection and use the same arrival rates as given in [30]. These are equal to

$$\lambda = \begin{bmatrix} \frac{370}{3600} & \frac{164}{3600} & \frac{194}{3600} & \frac{167}{3600} & \frac{705}{3600} \end{bmatrix}^T. \tag{5.12}$$

Two possible service times exist: the real-life data gives the service rates and set-up times for human drivers, while the CIC method in [10] is able to achieve shorter service times by automating the vehicles. In order to make a fair comparison, all three control strategies are simulated using both the service times for the human drivers and those of the automated vehicles.

The service times for human drivers are determined from the service rates and set-up times given in [30]. These are equal to

$$T^s_{\text{human}} = \begin{bmatrix} 1.9 & 0 & 0 & 4 & 0 \\ 0 & 2 & 0 & 5 & 5 \\ 0 & 0 & 2.2 & 0 & 5 \\ 6 & 4 & 0 & 2 & 5 \\ 0 & 5 & 4 & 4 & 2 \end{bmatrix}, \tag{5.13}$$

where queues with $T^s_{q,i} = 0$ have trajectories which do not intersect. The service times for automated vehicles are derived from the desired inter-vehicle distance $\delta_{des} = r + hV$, where r is the stand-still inter-vehicle distance and h is the headway time [10]. Since the physical dimensions of the intersection are unknown, it is assumed that there are only two values for r and h: for vehicles coming from the same queue and for vehicles coming from different queues. These are $r = 8$ and $h = 0.3$ for vehicles coming from the same queue and $r = 10$ and $h = 0.5$ for vehicles coming from different queues, respectively. Then the service times for automated vehicles are given by

$$T^s_{\text{CIC}} = \begin{bmatrix} 1.26 & 0 & 0 & 1.7 & 0 \\ 0 & 1.26 & 0 & 1.7 & 1.7 \\ 0 & 0 & 1.26 & 0 & 1.7 \\ 1.7 & 1.7 & 0 & 1.26 & 1.7 \\ 0 & 1.7 & 1.7 & 1.7 & 1.26 \end{bmatrix}. \tag{5.14}$$

While the FCFS policy and the MPC controller are such that they can determine their own crossing sequence, the vehicle actuated traffic light follows a cycle, within which one or more queues are served until they are empty. The optimal cycle for this intersection is given in Table 5.1. For as long as a given queue receives a green light, vehicles can depart from that queue with the given service time between them. Once the light becomes red, no more vehicles are allowed to depart from that queue.

Table 5.1: *The optimal cycle for the vehicle actuated traffic light.*

Mode	Queues served	Serve until
1	1, 2, 3	Queues 2 and 3 are empty
2	1, 5	Both queues are empty
3	3, 4	Queue 4 is empty

First, simulations using the service times for human drivers are performed with initial queue lengths $x(t_0, j_0) = \begin{bmatrix} 33 & 19 & 27 & 22 & 9 \end{bmatrix}^T$. For the FCFS policy, the order in which these vehicles arrived is set in proportion to the queue lengths. This is achieved by finding the least common multiple of all initial queue lengths, and dividing this by the queue lengths. The resulting values are comparable to inter-arrival times, from which the order of arrival of the vehicles can be determined. Two simulations of the MPC controller are performed for different prediction horizons. A short horizon where $N_p \Delta t < \max_{q,i} T^s_{q,i}$ and a long horizon. The reason for this is that the service times for human drivers of (5.13) are quite long, which means that a long prediction horizon is desirable. Otherwise, if the prediction horizon is shorter than the service times of (5.13), then the controller is unlikely to switch queues. The reason for this is that switching queues requires that the controller remains idle within

the prediction horizon, which is never optimal if there are vehicles in the current queue. On the other hand, a long prediction horizon leads to high computational costs. In order to observe a possible trade-off between optimality and computational costs two horizons are used: in one simulation, the sampling time is $\Delta t = 0.1$ seconds and the prediction horizon is $N_p = 20$ time-steps. In the other, the sampling time is $\Delta t = 0.5$ seconds and the prediction horizon is $N_p = 30$ time-steps. Note that using $\Delta t = 0.5$ seconds means that the controller cannot serve all queues with the service time given in (5.13), since these are not all multiples of 0.5. This is a discretisation error which leads to a loss of performance, but which cannot be avoided.

The results of the simulations with the vehicle actuated traffic light, the FCFS policy and the two MPC controllers when using the service times for human drivers of (5.13) are shown in Figures 5.12, 5.13, 5.14 and 5.15, respectively.

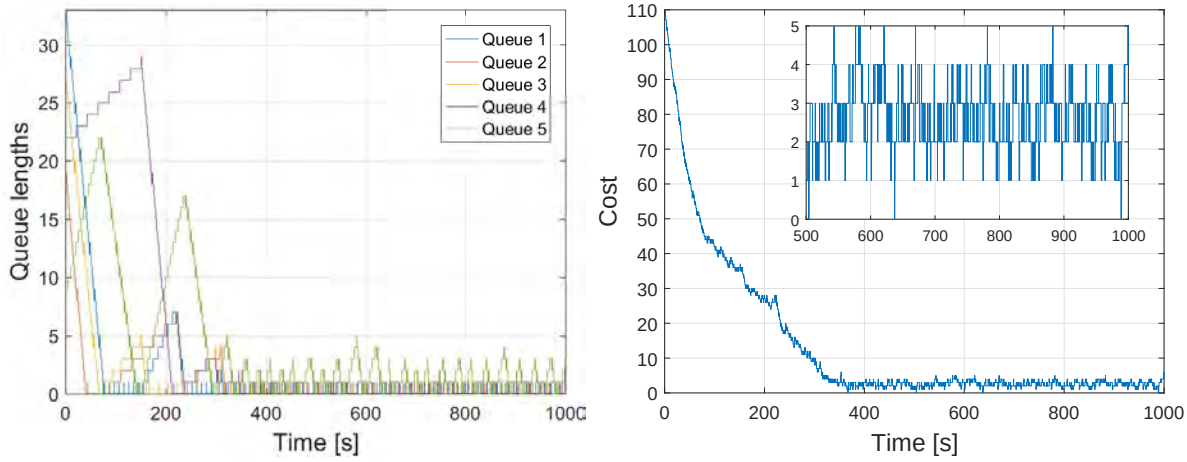

Figure 5.12: *The queue lengths and the total cost for the real-life intersection controlled using a vehicle actuated traffic light with the service times of* (5.13) *for human drivers.*

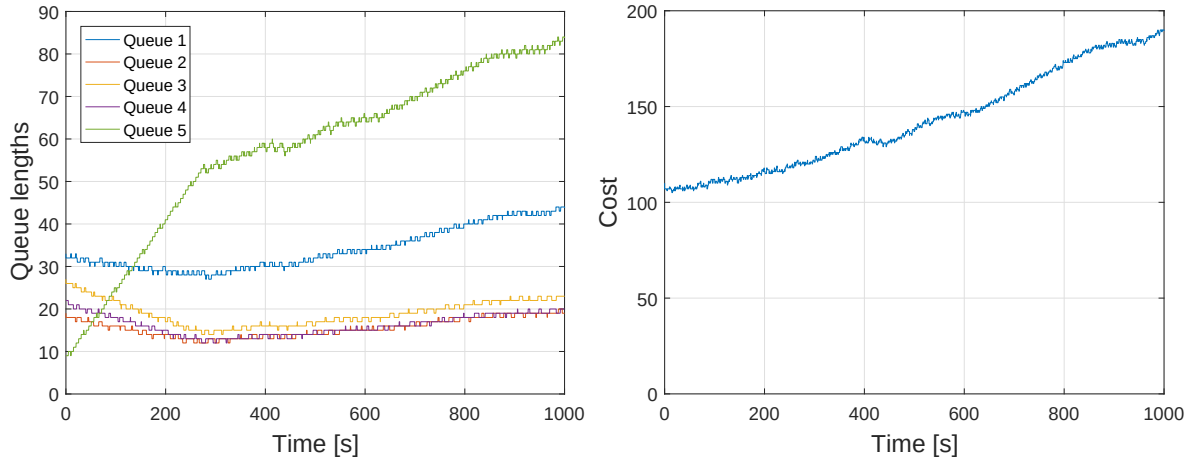

Figure 5.13: *The queue lengths and the total cost for the real-life intersection controlled using a first-come-first-serve policy with the service times of* (5.13) *for human drivers.*

Figure 5.12 shows that the vehicle actuated traffic light goes through its cycle multiple times before bringing the system to its steady-state. After that, the total cost varies between one and five. Figure 5.13 shows that the first-come-first-serve policy is unable to serve all vehicles. The reason for this is that the set-up times of (5.13) are fairly long. Due to the nature of the FCFS policy, the server switches queues often, thereby wasting a lot of time and making it impossible to serve all vehicles. In contrast, in Figures 5.14 and 5.15, both MPC controllers bring the system to a steady-state with lower total costs than the traffic light. Once the steady-state has been reached, all queues rarely exceed two vehicles in length. It is noteworthy that the transient trajectory of the MPC controller with the short horizon in Figure 5.14 is nearly identical to the transient trajectory of the vehicle

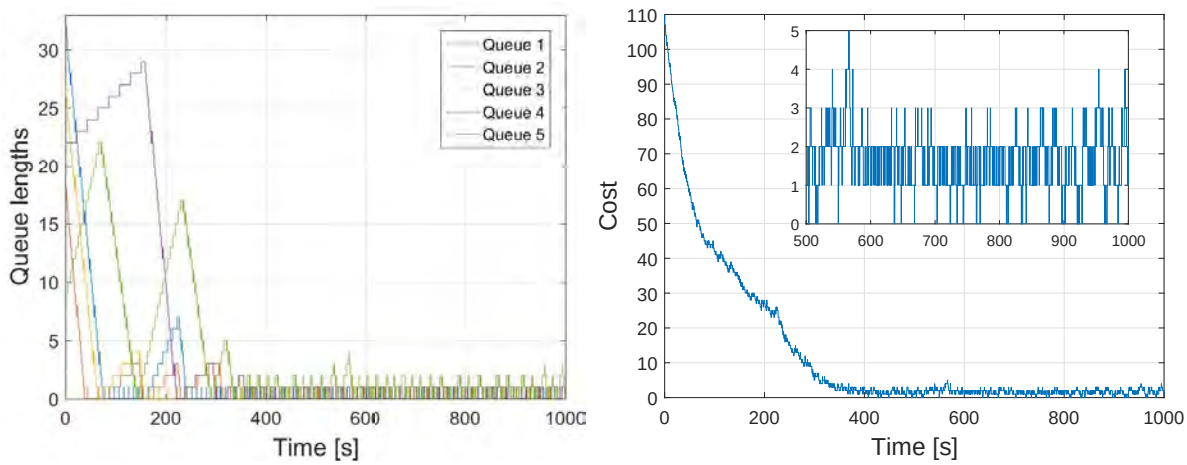

Figure 5.14: *The queue lengths and the total cost for the real-life intersection controlled using MPC with the service times of* (5.13) *for human drivers. The sampling time is* $\Delta t = 0.1$ *seconds and the prediction horizon is* $N_p = 20$ *time-steps.*

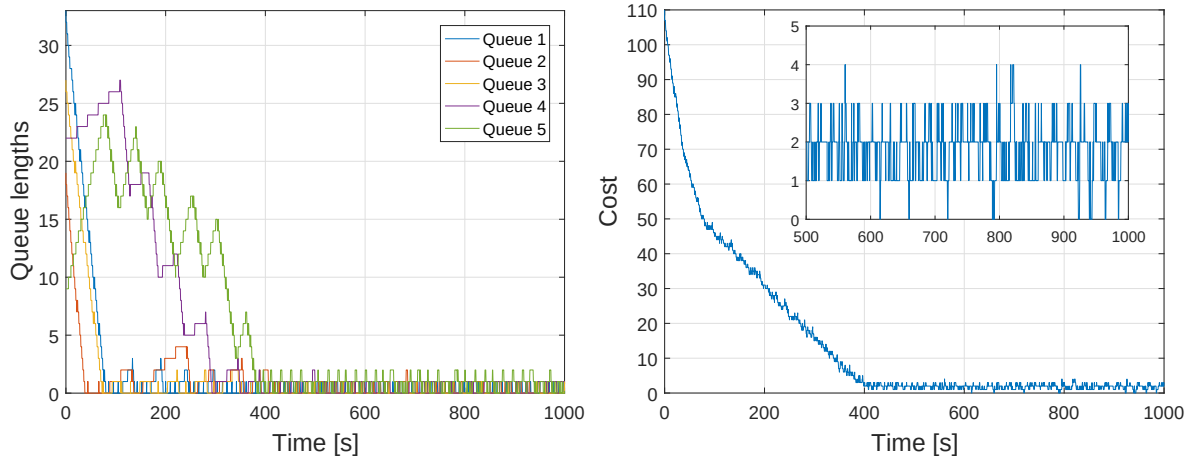

Figure 5.15: *The queue lengths and the total cost for the real-life intersection controlled using MPC with the service times of* (5.13) *for human drivers. The sampling time is* $\Delta t = 0.5$ *seconds and the prediction horizon is* $N_p = 30$ *time-steps.*

actuated traffic light in Figure 5.12. Both controllers achieve behaviour comparable to a clearing policy. The traffic light behaves this way by design, but the MPC controller shows this behaviour because its prediction horizon is only $N_p \Delta t = 2$ seconds. As explained previously, this horizon is shorter than the set-up times, which causes the controller to only switch queues if its current queue remains empty until the set-up time has passed, which leads to behaviour similar to the traffic light. A comparison of Figures 5.14 and 5.15 shows little difference in the steady-state behaviour of the MPC controllers. However, there is a clear difference in the performance of the short horizon MPC controller and the long horizon MPC controller in the transient trajectory. While both controllers reach the steady-state at about 400 seconds, using a long horizon leads to a far smoother trajectory. Compared to the clearing policy-like behaviour in Figure 5.14, Figure 5.15 shows that the controller with a prediction horizon $N_p \Delta t = 15$ seconds switches queues more often. Queues 1, 2 and 3 can be served at the same time, so the controller switches to those queues whenever they build up to some length. This makes the controller more reliable, since the stable behaviour of Figure 5.14 may not be achievable for every intersection, or if vehicle arrivals are stochastic.

Next, simulations are performed using the service times for automated vehicles of (5.14). The initial queue lengths are again equal to $x(t_0, j_0) = \begin{bmatrix} 33 & 19 & 27 & 22 & 9 \end{bmatrix}^T$. The same two prediction horizons are used for the MPC controller, in order to compare the results with those for human drivers. These are $\Delta t = 0.1$ seconds $N_p = 20$ time-steps for the first controller, and are $\Delta t = 0.5$ and $N_p = 30$ for

the second MPC controller. As mentioned earlier, the service times T^s_{CIC} of (5.14) are not a multiple of 0.5, so there is some loss of performance. Figures 5.16, 5.17, 5.18 and 5.19 show the results when using the service times for automated vehicles of (5.14) for intersections controlled by the vehicle actuated traffic light, the FCFS policy and MPC, respectively.

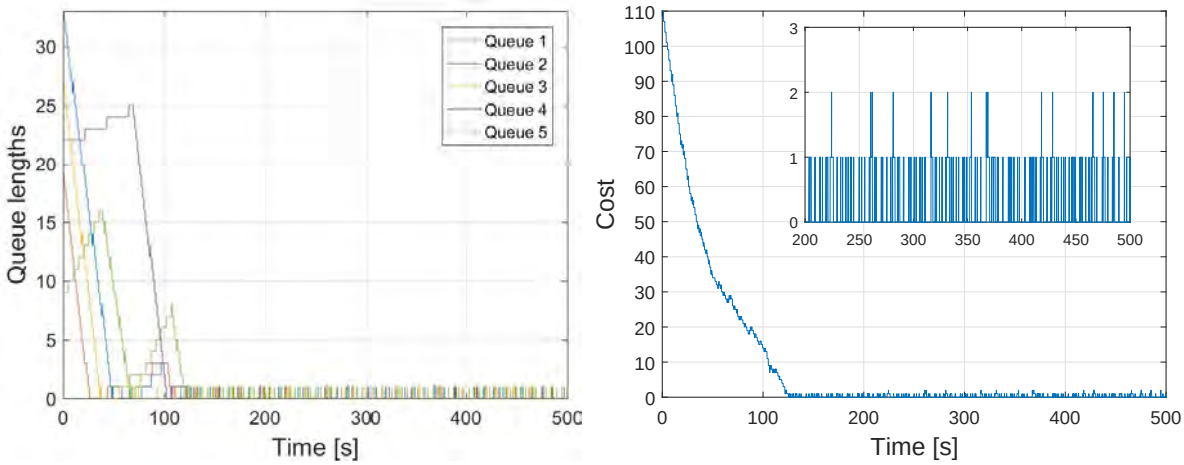

Figure 5.16: *The queue lengths and the total cost for the real-life intersection controlled using a vehicle actuated traffic light with the service times of* (5.14) *for vehicles automated through CIC.*

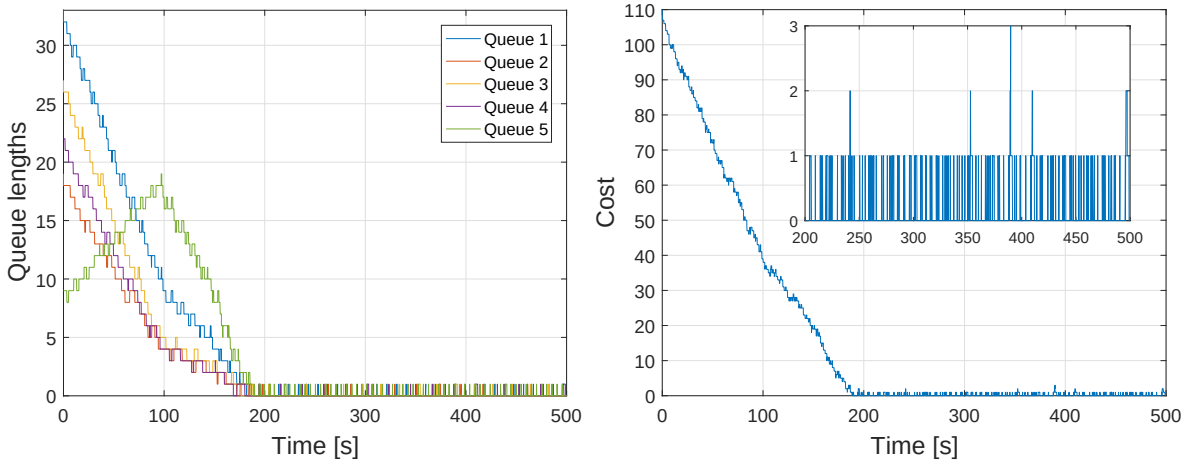

Figure 5.17: *The queue lengths and the total cost for the real-life intersection controlled using a first-come-first-serve policy with the service times of* (5.14) *for vehicles automated through CIC.*

First of all, Figures 5.16–5.19 make it clear that automating vehicles and decreasing the service times lead to a decrease in costs for all three control methods. They all reach the steady-state trajectory in only about half the time as with human drivers, and the queues rarely exceed a single vehicle in length after that. Furthermore, Figure 5.17 shows that, in contrast to Figure 5.13, the first-come-first-serve policy is now able to achieve a steady-state and bring the queues to a minimum. The MPC controllers achieve a similar steady-state in Figures 5.18 and 5.19. The advantage in using MPC over a FCFS policy lies in the transient trajectory, which reaches the steady-state trajectory faster than if a FCFS policy is used. Figure 5.18 shows that the MPC controller with the short horizon again behaves similarly to the traffic light in Figure 5.16, first emptying queues 1, 2 and 3 before moving on to other queues. The FCFS policy continuously switches between the five queues, thereby taking longer to empty them. Comparing the short horizon MPC of Figure 5.18 with the long horizon MPC of Figure 5.19 shows results similar to those in Figures 5.14 and 5.15. If the horizon is short, the MPC controller still mostly uses a clearing policy, even if the set-up times are shorter. With a longer prediction horizon the controller switches queues far more often, which actually decreases its performance compared to Figure 5.18, but is more reliable.

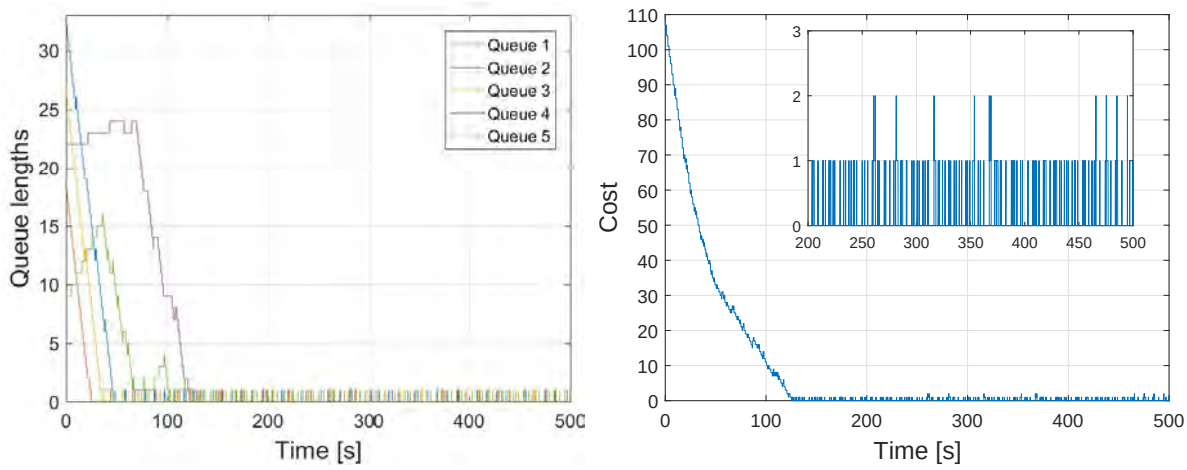

Figure 5.18: *The queue lengths and the total cost for the real-life intersection controlled using MPC with the service times of* (5.14) *for vehicles automated through CIC. The sampling time is* $\Delta t = 0.1$ *seconds and the prediction horizon is* $N_p = 20$ *time-steps.*

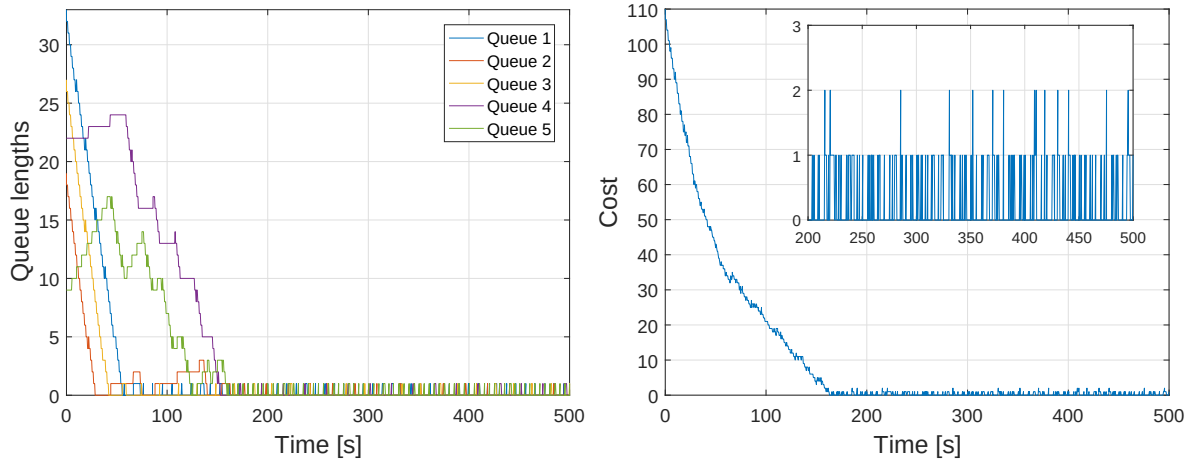

Figure 5.19: *The queue lengths and the total cost for the real-life intersection controlled using MPC with the service times of* (5.14) *for vehicles automated through CIC. The sampling time is* $\Delta t = 0.5$ *seconds and the prediction horizon is* $N_p = 30$ *time-steps.*

The benefit of automating vehicles is clear from Figures 5.16–5.19. Lower service times lead to a lower server utilisation, making it possible to reach a lower steady-state cost. Yet, the steady-state behaviour shown in the figures seems to differ little between the various control methods. In order to see larger differences in the steady-state behaviour, a final set of simulations are performed with the service times of (5.14) for automated vehicles, but with the arrival rates of (5.12) doubled. The new arrival rates are equal to

$$\lambda = \begin{bmatrix} \frac{740}{3600} & \frac{328}{3600} & \frac{388}{3600} & \frac{334}{3600} & \frac{1410}{3600} \end{bmatrix}^T. \tag{5.15}$$

The same four simulations are performed as before, where the intersection is controlled by a vehicle actuated traffic light, using the FCFS policy and the two model predictive controllers, respectively. The prediction horizons of the MPC controllers are $N_p = 20$ with sampling time $\Delta t = 0.1$ seconds and $N_p = 30$ with sampling time $\Delta t = 0.5$ seconds. The initial queue lengths are equal to $x(t_0, j_0) = \begin{bmatrix} 33 & 19 & 27 & 22 & 9 \end{bmatrix}^T$.

Figures 5.20, 5.21, 5.22 and 5.23 show the results when using the service times for automated vehicles of (5.14) and the arrival rates of (5.15) for intersections controlled by the vehicle actuated traffic light, the FCFS policy and MPC, respectively.

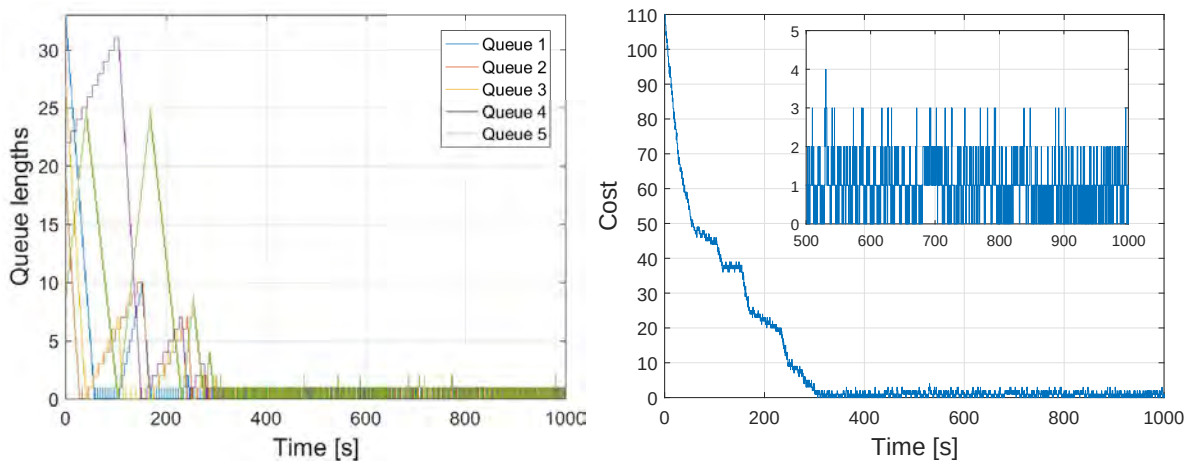

Figure 5.20: *The queue lengths and the total cost for the real-life intersection controlled using a vehicle actuated traffic light with the service times of* (5.14) *for vehicles automated through CIC and the arrival rates of* (5.15).

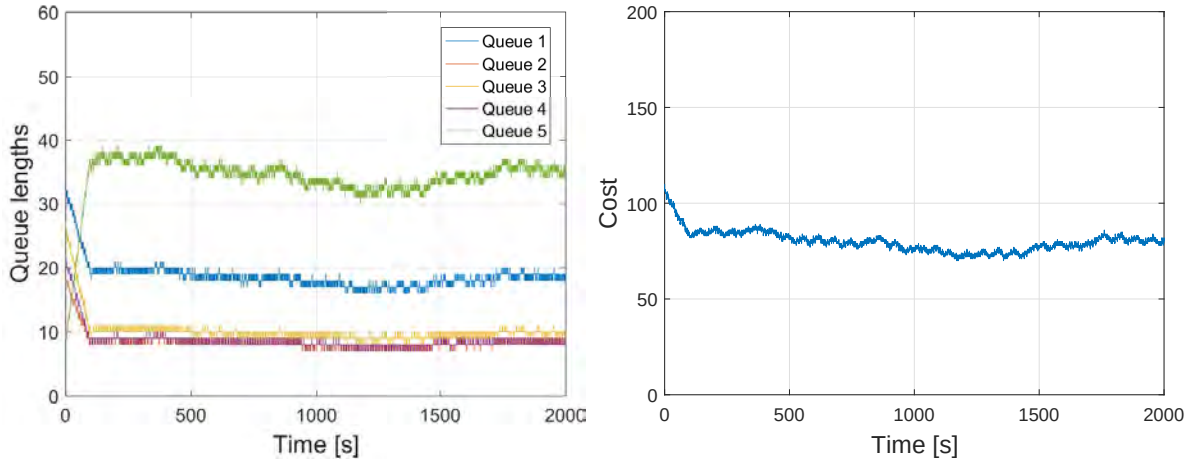

Figure 5.21: *The queue lengths and the total cost for the real-life intersection controlled using a first-come-first-serve policy with the service times of* (5.14) *for vehicles automated through CIC and the arrival rates of* (5.15).

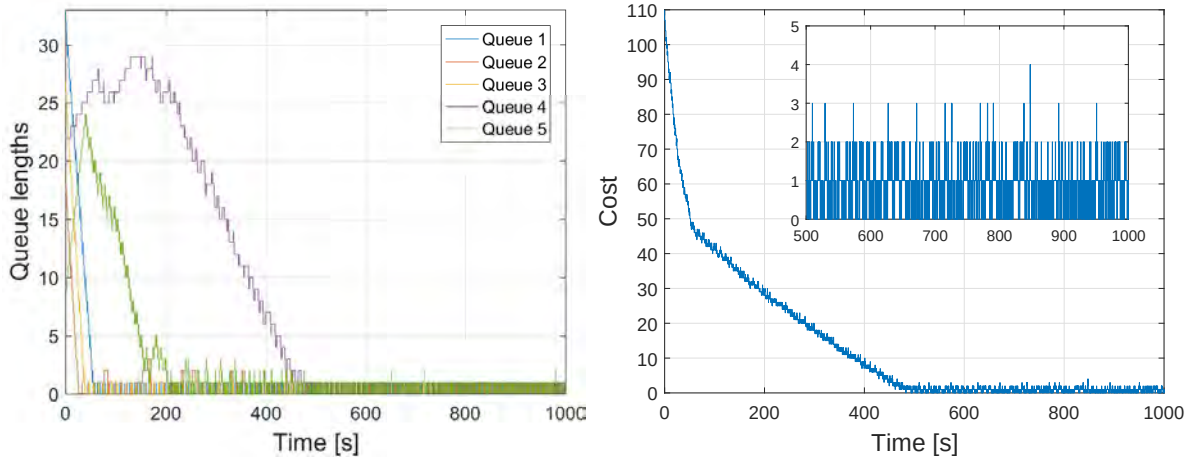

Figure 5.22: *The queue lengths and the total cost for the real-life intersection controlled using MPC with the service times of* (5.14) *for vehicles automated through CIC and the arrival rates of* (5.15). *The sampling time is* $\Delta t = 0.1$ *seconds and the prediction horizon is* $N_p = 20$ *time-steps.*

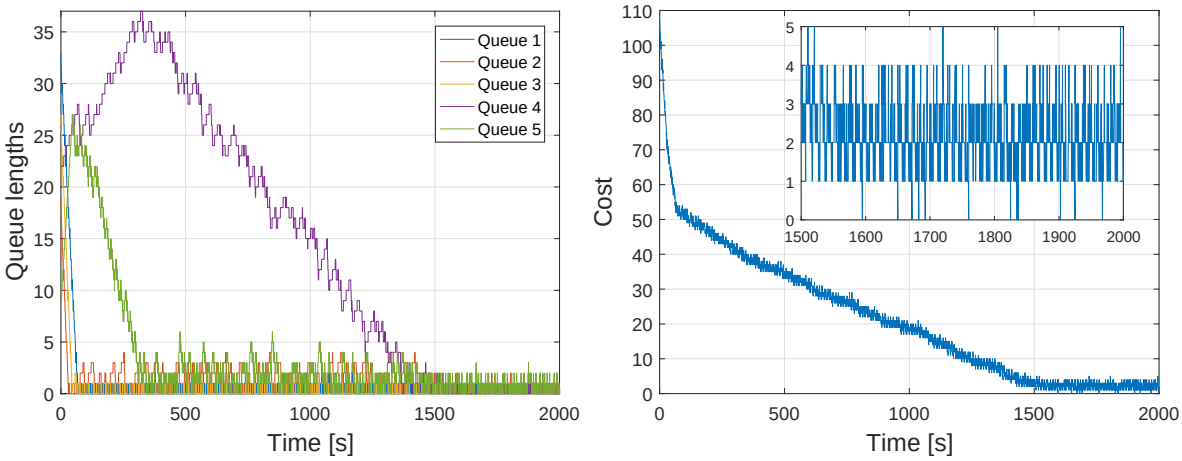

Figure 5.23: *The queue lengths and the total cost for the real-life intersection controlled using MPC with the service times of* (5.14) *for vehicles automated through CIC and the arrival rates of* (5.15). *The sampling time is* $\Delta t = 0.5$ *seconds and the prediction horizon is* $N_p = 30$ *time-steps.*

Compared to Figures 5.16–5.19, Figures 5.20–5.23 show a clear difference in both the steady-state behaviour and the transient trajectory. Figure 5.21 shows that the FCFS policy is unable to empty the queues now that the arrival rates are higher. The vehicle actuated traffic light, shown in Figure 5.20, is able to reach steady-state after 300 seconds, compared to 120 seconds previously. Additionally, the transient trajectory is not as smooth as in Figure 5.12 and 5.16 due to the high arrival rates. The largest difference in behaviour compared to earlier simulation results is seen in the MPC controllers in Figures 5.22 and 5.23. In both figures, queues 4 and 5 seem to be neglected to some extent, and while vehicles in those queues are regularly served, it takes a long time for the system to reach steady-state. This is due to a combination of two factors; 1) the conflicting trajectories between the queues and 2) the differences in arrival rates between the queues. Which queues can be served at the same time is determined from the service times, and is easily seen in the traffic light cycle in Table 5.1. Queues 1, 2 and 3, or queues 1 and 5 can be served at the same time. Both of these modes have a high combined arrival rate, and equal service times. Yet, queue 4 can only be served together with queue 3, both of which have low arrival rates. This leads to queue 4 being neglected by the controller in particular, since serving queues 3 and 4 is relatively inefficient in terms of reducing total cost. Of course, these factors are also present in the previous simulations, where the MPC controller achieves a better transient trajectory. This is likely because the server utilisation ρ of all queues is higher in the results of Figures 5.20–5.23 than in the previous simulations.

The average steady-state costs and delays of all simulations of the five-queue intersection are shown in Tables 5.2 and 5.3. The average delay is calculated by matching every vehicle arrival with its departure. Comparing the resulting average cost and delay between the three sets of simulations clearly shows the benefit of automating vehicles. Compared to human drivers, the average costs for automated vehicles are decreased by a factor 5 to 15, which greatly increases the performance of the intersection. The average delay is similarly decreased, as seen in Table 5.3. In fact, even with double the arrival rates, both the average cost and delay are still lower compared to the traffic-light controlled intersection with human drivers.

Comparing the average costs for the different control methods in Table 5.2 shows that the MPC controllers outperform the traffic light in steady-state when the service times of human drivers are used. However, the steady-state performance of all three control methods is similar when automated vehicles are considered. The differences in performance are likely smaller than for human drivers due to the lower server utilisation. This is also what allows the FCFS policy to reach a steady-state. The traffic light performs slightly worse than the other control methods for automated vehicles because it always has to cycle through its modes in the same order, even if using a different order would be preferable at some point. It is also clear that increasing the horizon of the MPC controller leads to worse performance in all cases, which is expected because the increased sampling time makes the

controller unable to serve vehicles with the minimum service time. The loss of performance due to the discretisation is greater for automated vehicles. This is because none of the service times or set-up times in 5.14 are a multiple of $\Delta t = 0.5$, thus the controller loses performance on all queues. For human drivers, only the service times of queues 1 and 3 in 5.13 are not a multiple of $\Delta t = 0.5$ seconds, and therefore the average cost is lower.

The average delay, shown in Table 5.3, generally follows the same trends as the average costs which have been discussed in the previous paragraph. However, the average delays differ from these trends only in the results of the MPC controller with the long prediction horizon. The average costs of the MPC controller increase with the prediction horizon, but the average delay decreases with the prediction horizon for human drivers. For automated vehicles, the average delay increases with the prediction horizon to a lesser extent than the average cost. These observations may be caused by the fact that the MPC controller with the long prediction horizon tends to switch queues more often, which is easily seen in the transient trajectory. Both in steady-state and in the transient trajectory, this behaviour causes vehicles to spend less time in their queues, even if there are more vehicles in the queues on average.

Table 5.2: *The average steady-state costs of the different control methods for human drivers, automated vehicles and a doubled arrival rate.*

	Average cost		
Control method	Human drivers	Autom. vehicles	Double λ
Traffic light	2.6081	0.1756	0.9555
First-come-first-serve	100+	0.1321	100+
MPC (short horizon)	1.6731	0.1462	0.7810
MPC (long horizon)	1.7758	0.3115	2.1149

Table 5.3: *The average steady-state delay, in seconds, of the different control methods for human drivers, automated vehicles and a doubled arrival rate.*

	Average delay		
Control method	Human drivers	Autom. vehicles	Double λ
Traffic light	5.7901	0.2904	0.9957
First-come-first-serve	100+	0.2858	100+
MPC (short horizon)	3.6758	0.2252	0.7912
MPC (long horizon)	3.4911	0.2526	1.8442

5.3 Summary

This chapter presents the simulation results which have been performed using the Simulink model presented in Chapter 4. First, theoretical results of [28, 29] are summarised in order to have a basis for comparison for both the steady-state and the transient behaviour in the simulation results. The main results are that the steady-state trajectory is in the form of a bow tie or a truncated bow tie, and that a slow mode must be present in order for the transient trajectory to reach the optimal steady-state trajectory in finite time.

In the simulations, both the parameters of the intersection and those of the controller have been varied, and the resulting behaviour has been observed. First, simulations of an intersection without a slow mode have shown that the behaviour of the MPC controller closely matches the optimal behaviour described in [28]. Furthermore, by varying the initial conditions it is shown that a slow mode is indeed necessary for the transient trajectory to reach the optimal steady-state trajectory in finite time. Without a slow mode, it is possible for there to be a phase shift between the arrival and departure of vehicles, leading to a sub-optimal steady-state cycle. A simulation of the same system, but with stochastic vehicle inter-arrival times shows that randomness increases the average queue lengths significantly.

Next, the effect of the cost function and prediction horizon of the controller on fairness is investigated. It is observed that increasing the cost of one queue can lead to unstable behaviour, and that the fairness constraints are unable to solve this instability. Instead, a stable process cycle could be reached either by increasing the prediction horizon or by choosing a quadratic cost function instead of a linear cost function. Since increasing the prediction horizon leads to high computational costs, a quadratic cost function is preferable. Using a quadratic cost function does, however, result in different optimal steady-state behaviour, where no queue is ever emptied, leading to longer delays.

Finally, the performance of the MPC controller has been compared with the performance of a first-come-first-serve policy and traffic light by applying all three control methods to an intersection based on real-life data. Results are compared for human drivers and for automated vehicles. In general, it is seen that automating vehicles decreases the average costs by a factor of 5 to 15. Even when the vehicles arrive twice as often, the average costs for automated vehicles are still lower than for human drivers. When comparing the different control methods, it is observed that the FCFS policy cannot achieve a stable cycle if the set-up times are too long or if the server utilisation is too high. If the service times are short enough, the MPC controller and the FCFS policy achieve similar steady-state results, outperforming the traffic light. The MPC controller realises a better transient trajectory than the FCFS policy, and increasing the prediction horizon improves the transient trajectory and makes the MPC controller more reliable.

Chapter 6

Conclusions and recommendations

This chapter summarises the main conclusions from this report. Some recommendations for further research on the topic of supervisory control of automated intersections are given.

6.1 Conclusions

This report is focused on the design of a supervisory controller of an automated intersection. In conjunction with the execution-level controller of [10], the supervisory controller aims to achieve a crossing sequence which minimises the average delay of the vehicles travelling through the intersection.

In Chapter 2, the intersection and the dynamics of the vehicles in it are abstracted as a hybrid system which evolves both in continuous-time and discrete-time. The abstracted model closely resembles a queueing system; vehicles wait in queues outside the Cooperation Zone, to which the supervisory controller controls access. The hybrid system evolves in discrete-time only when there is an event, e.g., an arrival or departure of a vehicle. Through the use of inter-event timers, the hybrid system is able to enforce a minimum time interval between any two vehicle departures, which is called the service time. The service times are based on the desired virtual inter-vehicle distance, and ensure that vehicles pass through the intersection safely. To ensure that the controller respects the service times, certain constraints are formulated which are placed on the controller.

The supervisory controller is designed using model predictive control in Chapter 3. Model predictive control predicts the behaviour of the system, subject to given constraints, over some prediction horizon, and aims to steer the system along the trajectory which minimises the cost function. The controller operates in discrete-time, and therefore the hybrid system is first discretised. The sampling time of the controller must be small enough that there is at most one departure or arrival per sampling interval in order to match the hybrid system description. In order to improve the accuracy of the controller, vehicle arrivals are predicted by measuring the average vehicle arrival rate through a moving average window. A linear cost function is chosen, which weighs all vehicles in a queue equally. Using a quadratic cost function is also an option, which leads to far higher costs for longer queues.

Chapter 4 presents the implementation of the hybrid system and the controller in Simulink. The hybrid system is implemented by means of the Hybrid Equations Toolbox, where all parts of the hybrid dynamics can be implemented directly. It is found that the hybrid system keeps track of the simulation time internally, which is not equal to the simulation time given by Simulink. In order to implement the controller, it is first converted to a mixed-logical dynamical system, which is able to combine the system dynamics with the logical constraints defined by the service times. This is achieved by defining the constraints through various binary variables in HYSDEL3, after which the controller is generated using the Multi-Parametric Toolbox.

The results of the simulations, performed using the Simulink model, are presented in Chapter 5. It is observed that the controller follows the expected optimal steady-state behaviour, both with and

without a slow mode. It is shown that, if there is no slow mode, then depending on the initial conditions the controller is not always able to reach the optimal steady-state cycle in finite time. Changing the cost function by doubling the weight of one queue creates unstable behaviour. It is proven that the fairness constraints are ineffective when it comes to solving this unstable behaviour or making the controller more fair from the driver's perspective. Instead, the system is stabilised by choosing a quadratic cost function. Using a quadratic cost function results in very different behaviour than a linear cost function. Instead of emptying the queues, both queues vary between two values, because the cost of the first few vehicles in a queue is very low. It is also conjectured that the system can be stabilised by increasing the prediction horizon. However, increasing the prediction horizon can be impractical, since a horizon of at least the cycle length is desired, which leads to high computational costs.

Finally, the results of the MPC controller are compared with a FCFS policy and a vehicle actuated traffic light using real-life data. Simulations are performed based on service times for human drivers and for automated vehicles. The results show that automated vehicles achieve a factor 5 to 15 reduction in the average cost compared to human drivers. Furthermore, the MPC controller is able to reduce the average cost compared to the traffic light by a significant margin. If the service times are long, the FCFS policy is not able to stabilise the system, but if the service times are short, the performance of the FCFS policy and the MPC controller are similar. It is also observed that increasing the prediction horizon of the MPC controller leads to better behaviour, though the average cost is higher due to a discretisation error.

6.2 Recommendations for further research

In this report, a supervisory controller for the Cooperative Intersection Control method has been created by modelling the intersection as a hybrid system and using this model to create a model predictive controller. The supervisory controller could also be developed through different strategies, and due to time constraints the current approach has not been fully explored. Some recommendations for further research are discussed below.

This report has aimed to create a supervisory controller which minimises the delay of vehicles in an intersection automated through Cooperative Intersection Control. While this supervisory controller has been created successfully, it has only been tested in isolation, in a simulation which uses a simplified model of the vehicle dynamics. In order to test if the supervisory controller truly improves the crossing sequence compared to a FCFS policy, it needs to be applied to a real (or, first, a simulated) automated intersection in which vehicle dynamics are controlled through CIC.

Secondly, the current formulation of the hybrid system uses a simplified model of the vehicle dynamics, which the controller then also uses in its predictions. In order to more accurately predict the behaviour of the automated intersection, the vehicle dynamics both inside and outside the Cooperation Zone must be taken into account. Currently, all vehicle dynamics are contained in the service times, which depend only on the desired virtual inter-vehicle distance and a constant velocity V. However, in reality, vehicles do not wait in queues or move with constant velocity. To make the service times more accurate, it is recommended to include a vehicle's position either inside or outside the CZ and take its (predicted) velocity into account. This also involves implementing the service times as time-varying. This requires some additions to the current implementation of the prediction model in HYSDEL3, which in its current form can only model time-invariant service times.

Another aspect of the controller that can be improved upon is the discretisation of the hybrid system. Currently, the discrete-time nature of the MPC controller only allows the controller to serve a vehicle at the sampling instances. If the service times are not a multiple of the sampling times, this creates a discretisation error which leads to a loss in performance. To achieve a more exact discretisation, the controller can be made to provide an additional control input which gives the time within the next sampling interval at which a vehicle can depart from its queue. This means that a control input $u_q(k) = 1$ does not come into affect until the time specified by the additional control input has passed.

This allows the controller to serve vehicles at the highest possible rate, increasing performance.

The main disadvantage of model predictive control is that it is computationally expensive. The simulations have shown that having a long prediction horizon is advantageous, and sometimes even necessary, but unfortunately the computational costs can be prohibitive. One possibility is to reduce the number of times the controller optimises its input through event-triggered control. Currently, the controller samples the system and optimises its input every Δt seconds. This is very wasteful since the input constraints force the input to be zero at most times. Instead, an event-triggered model predictive controller only optimises its input when some event occurs, or when the actual system trajectory deviates too much from the predicted trajectory. In the case of the supervisory, vehicle arrivals or when an inter-departure timer passes a service time can be considered events. Then, the controller only optimises its input when it is able to provide a non-zero input, drastically reducing the number of times the control optimisation problem is solved. At the same time, the prediction horizon can be made to consider N_p events, not time-steps, thus making it possible to increase the duration of the prediction horizon without actually increasing the computational cost.

It has been observed in the simulations that recursive feasibility and stability of the model predictive controller can be affected by the system parameters, certain constraints and the prediction horizon. It is merely assumed in Chapter 3 that the parameters are such that the optimisation problem is recursively feasible and stable, but how exactly these can be guaranteed is not certain. A more rigorous proof of these properties would be helpful in choosing the control parameters and is a possible subject of future research.

Finally, this report has only addressed a single intersection being controlled by a single, centralised controller. Further research on optimising the flow of vehicles through a network of intersections, each controlled by their own controller, is useful for metropolitan areas. Another possibility is to look into distributed control of a single intersection. Many real-life intersections do not have any traffic control infrastructure, and installing a centralised controller at every intersection can be expensive. Distributed control which relies only on communication between vehicles avoids this problem.

Bibliography

[1] Jeremy Broughton, Pete Thomas, Alan Kirk, Laurie Brown, George Yannis, Petros Evgenikos, Panagiotis Papantoniou, Nimmi Candappa, Michiel Christoph, Kirsten van Duijvenvoorde, et al. Traffic safety basic facts 2012: Junctions. 2013.

[2] L. Chen and C. Englund. Cooperative intersection management: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 17(2):570–586, Feb 2016.

[3] Kurt Dresner and Peter Stone. A multiagent approach to autonomous intersection management. *Journal of Artificial Intelligence Research*, 31:591–656, March 2008.

[4] D. Carlino, S. D. Boyles, and P. Stone. Auction-based autonomous intersection management. In *Intelligent Transportation Systems - (ITSC), 2013 16th International IEEE Conference on*, pages 529–534, Oct 2013.

[5] Mark VanMiddlesworth, Kurt Dresner, and Peter Stone. Replacing the stop sign: Unmanaged intersection control for autonomous vehicles. In *AAMAS Workshop on Agents in Traffic and Transportation*, pages 94–101, Estoril, Portugal, May 2008.

[6] L. Li and F. Y. Wang. Cooperative driving at blind crossings using intervehicle communication. *IEEE Transactions on Vehicular Technology*, 55(6):1712–1724, Nov 2006.

[7] F. Yan, M. Dridi, and A. El Moudni. Autonomous vehicle sequencing algorithm at isolated intersections. In *Intelligent Transportation Systems, 2009. ITSC '09. 12th International IEEE Conference on*, pages 1–6, Oct 2009.

[8] Jean Gregoire, Silvere Bonnabel, and Arnaud de La Fortelle. Priority-based coordination of robots. *CoRR*, abs/1306.0785, 2013.

[9] Florent Altché, Xiangjun Qian, and Arnaud de La Fortelle. Time-optimal coordination of mobile robots along specified paths. *CoRR*, abs/1603.04610, 2016. Under peer review.

[10] A. I. Morales Medina, N. van de Wouw, and H. Nijmeijer. Automation of a t-intersection using virtual platoons of cooperative autonomous vehicles. In *Intelligent Transportation Systems (ITSC), 2015 IEEE 18th International Conference on*, pages 1696–1701, Sept 2015.

[11] O. J. Boxma and J. A. Weststrate. *Waiting Times in Polling Systems with Markovian Server Routing*, pages 89–104. Springer Berlin Heidelberg, Berlin, Heidelberg, 1989.

[12] Rafal Goebel, Ricardo G. Sanfelice, and Andrew R. Teel. Hybrid dynamical systems. *IEEE Control Systems*, 29(2):28–93, 2009.

[13] Rafal Goebel, Ricardo G. Sanfelice, and Andrew R. Teel. *Hybrid Dynamical Systems: modeling, stability, and robustness*. Princeton University Press, 2012.

[14] Sébastien Faye, Claude Chaudet, and Isabelle Demeure. A distributed algorithm for multiple intersections adaptive traffic lights control using a wireless sensor networks. In *Proceedings of the First Workshop on Urban Networking*, UrbaNe '12, pages 13–18, New York, NY, USA, 2012. ACM.

[15] L. W. Chen, P. Sharma, and Y. C. Tseng. Dynamic traffic control with fairness and throughput optimization using vehicular communications. *IEEE Journal on Selected Areas in Communications*, 31(9):504–512, September 2013.

[16] J Anthony Rossiter. *Model-based predictive control: a practical approach*. CRC press, 2003.

[17] Rolf Findeisen and Frank Allgöwer. An introduction to nonlinear model predictive control. In *21st Benelux Meeting on Systems and Control*, volume 11, pages 119–141. Technische Universiteit Eindhoven Veldhoven, 2002.

[18] T. Manrique, M. Fiacchini, T. Chambrion, and G. Millerioux. Mpc tracking under time-varying polytopic constraints for real-time applications. In *Control Conference (ECC), 2014 European*, pages 1480–1485, June 2014.

[19] Johan Löfberg. Oops! i cannot do it again: Testing for recursive feasibility in mpc. *Automatica*, 48(3):550 – 555, 2012.

[20] Alberto Bemporad and Manfred Morari. Robust model predictive control: A survey. In *Robustness in identification and control*, pages 207–226. Springer, 1999.

[21] F.L. Mannering and S.S. Washburn. *Principles of Highway Engineering and Traffic Analysis*. Wiley, 2012.

[22] R. G. Sanfelice, D. A. Copp, and P. Nanez. Hybrid Equations (HyEQ) Toolbox. Technical report, 2014.

[23] M. Herceg, M. Kvasnica, C.N. Jones, and M. Morari. Multi-Parametric Toolbox 3.0. In *Proc. of the European Control Conference*, pages 502–510, Zürich, Switzerland, July 17–19 2013. `http://control.ee.ethz.ch/~mpt`.

[24] Alberto Bemporad and Manfred Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407–427, 1999.

[25] H.P. Williams. *Model Building in Mathematical Programming*. Wiley, 1993.

[26] F.D. Torrisi, A. Bemporad, and D. Mignone. HYSDEL — A Tool for Generating Hybrid Models. Technical report, Oct 2000.

[27] Wikibooks. Fundamentals of transportation, 2014. [Online; accessed 26-December-2016].

[28] J.A.W.M. Eekelen. *Modelling and control of discrete event manufacturing flow lines*. PhD thesis, Eindhoven University of Technology, 2007.

[29] D.A.J. van Zwieten. *Fluid flow switching servers: control and observer design*. PhD thesis, Eindhoven University of Technology, 2014.

[30] S.T.G. Fleuren and A.A.J. Lefeber. Data of real-life intersections for fixed-time traffic light control. 2016.