Investigating the optimal trajectories and performance of autonomous vehicles crossing a network of intersections

de Rooij, Janne S.

Award date:
2022

Eindhoven University of Technology
Department of Mathematics and Computer Science

Bachelor Final Project: Investigating the optimal trajectories and performance of autonomous vehicles crossing a network of intersections

---

Janne de Rooij
j.s.d.rooij@student.tue.nl
1333550

Supervisor: Marko Boon
February 7, 2022

Contents

1 Introduction

Everyday, car accidents occur that were caused by human errors. Autonomous vehicles can prevent these accidents, thereby saving lives. Moreover, autonomous vehicles can drive in a more environmental friendly manner [1]. Besides these ethical reasons as to why one would use autonomous vehicles instead of human-driven vehicles, autonomous vehicles can also drive much more efficiently than humans can. Since it is plausible that autonomous vehicles will one day be driving in our society, it is useful to investigate how one can maximize the efficiency of autonomous vehicles. This project will focus on investigating the behavior of autonomous vehicles crossing a network of intersections.

Where human-driven vehicles cross an intersection by guidance of a physical traffic light, autonomous vehicles have no need for those anymore. Instead, there will be a central controller present at the intersection that can communicate with all autonomous vehicles. The controller will ensure that all vehicles will know the time at which they are allowed to cross the intersection, before actually arriving at the intersection. Since the arrival of vehicles is known earlier, cars can be scheduled to cross the intersection in such a way that it can be crossed much more efficiently, resulting in less delay [2].

This project will construct a model for autonomous vehicles crossing a network of intersections. Within this model, the behaviour of a vehicle crossing an intersection is determined by two different algorithms. First, a Platoon Forming Algorithm (PFA) will schedule the times at which each vehicle is allowed to cross an intersection. This algorithm will cause the vehicles to form platoons, which cross the intersection together. Then, a Speed Profiling Algorithm (SPA) will determine the optimal path that each vehicle drives, which will ensure that the intersection is used as efficiently as possible.

The focus of this project will be on two aspects. First of all, this project aims to optimize and then visualize the trajectories corresponding to autonomous vehicles that are crossing a network of intersections. Additionally, this project will investigate the performance of the algorithms used within the given model. One of the performance measures that we will analyze, is the stability of the system. During the project, we will consider the model to be unstable if a vehicle is not able to safely cross the intersection. In particular, this project aims to answer the following questions:

- What is the influence of the PFA and the SPA on the delay of vehicles in a network of intersections?
- What is the influence of the SPA on the stability of vehicles crossing a network of intersections?

In order to answer these questions, section 2 will provide an overview of background information found in literature. In particular, it will give a summary of the paper by Miculescu and Karaman [3], whose model will form the basis of this project. Section 3 will then construct the mathematical model, and provide some related theoretical background knowledge. Subsequently, section 4 will elaborate on a simulation conducted to approximate the mathematical model. The results of the simulation are then presented and explained in section 5. Finally, section 6 draws conclusions from the results.

2 Literature overview

There is not simply one type of autonomous vehicle; for a single feature of the car, such as steering the wheel, cars exist with different ratios between human control and the control of the autonomous system. These correspond to multiple levels of driving automation, created by the Society of Automotive Engineers [4]. The first level corresponds to no automation at all, this represents a vehicle in which the driver has 100% of the control. Then, level 1 describes a car that has driver assistance. Within this setting, the automated systems can take control of the vehicle in specific situations, but they do not completely take over. An example of this given by a vehicle with the cruise control set to a specific speed, the vehicle can then accelerate and decelerate to maintain

this velocity. The next level is partial automation, in which the automated systems can execute some more complex tasks. For example, a vehicle with this level can pair steering and accelerating. Level 3 is called conditional automation. When driving a level 3 vehicle, under certain conditions, the driver does not have to do anything. The system can however request the driver to take over. An example of a feature of a level 3 vehicle is shown when such a vehicle finds itself in a traffic jam; while the car is still stuck in the traffic jam, the automated system can drive the car. However, when the traffic jam is over and the speed of the vehicle is increasing, the system will ask the driver to take over again. We then progress to level 4: high automation. Within this level, a vehicle is able to drive completely by itself. The vehicle will however ask the driver to take over during certain circumstances, for example when driving through heavy snowfall. The last level is number 5, which is called full automation. Vehicles of level 5 automation do not require a driver at all. In this project, vehicles are assumed to be able to control the speed of the vehicle when approaching and crossing an intersection, and will not need a driver in this situation. We will therefore consider autonomous vehicles of level 3 or higher.

One paper that plays a significant role in this research topic, is that of Miculescu and Karaman [3]. In this paper, the concept is studied where autonomous vehicles arrive at an intersection and adjust their speeds to traverse the intersection as rapidly as possible, while avoiding collisions. A coordination algorithm that schedules the crossing of the vehicles was proposed, and it was shown that when using this algorithm, surely no collisions occur and that the expected delay was significantly less compared to the delay experienced by vehicles crossing the intersection with a green-yellow-red traffic light. The intersection is modelled as a polling system with two queues and a single server. A central controller is present at the intersection that determines when which lane is allowed to cross the intersection. The central controller decides this based upon the polling policy used, which enforces that vehicles form platoons which cross the intersection together. In this paper, the exhaustive, gated and k-limited policies were discussed. This policy schedules when a vehicle enters the control region of the intersection, which is the part of the lane that is within a certain distance of the intersection. Additionally, the policy schedules the time at which the vehicle will reach the intersection. Then, a procedure MotionSynthesize is proposed, which generates a trajectory for a vehicle in the control region. This procedure guarantees that the vehicle reaches the intersection at full speed at its time scheduled by the polling policy, and that it does not collide with the vehicle ahead, while minimizing the distance of the vehicle to the intersection. In order to analyze the performance of this procedure, a uniform discretization of MotionSynthesize was created. The paper shows that when using this model, the time by which vehicles are delayed when crossing a single intersection is one to two orders of magnitude smaller than the delays that human-driven vehicles experience with a regular traffic light cycle.

A paper that continues on this concept is that of Timmerman and Boon [5]. They have again analyzed the behaviour of an autonomous vehicle by first using a PFA to schedule the times at which vehicles can cross the intersection, followed by applying the MotionSynthesize procedure to obtain the trajectory of the vehicle. In this paper, a closed form solution is given to the MotionSynthesize procedure that was given by Miculescu and Karaman, and it was shown that they behave similarly. Additionally, an alternative version of the MotionSynthesize procedure was given; instead of having an objective that minimizes the distance to the intersection, the procedure minimizes the absolute value of the acceleration. This has the advantages that the vehicles consume less energy, and that the ride is more pleasant as the car drives more smoothly. This version does however also have disadvantages, including that the physical length of the queue grows and that vehicles cannot enter the control region as close to each other. For this alternate version of the MotionSynthesize procedure, a closed form solution was given as well. After introducing the algorithms and proving the similarities between the closed form solutions and the linear optimization algorithms, the performance of the model was analyzed. A useful property of this model is that it behaves as a polling model, meaning that the results found for polling models can be applied to this model as well. Approximations for the mean delay using three different PFA's were derived, and then compared to the delays of traditional traffic lights. In the paper, it was shown that the average experienced delay when crossing an intersection decreases significantly when using an exhaustive PFA instead of a traffic light cycle.

The purpose of studying the traffic flow of autonomous vehicles, is to enable the implementation of autonomous vehicles in society. In reality, it is of course not a realistic assumption that each intersection is independent of the intersections surrounding them. Therefore, it is also important to study networks of intersections. Xi Lin et al. [6] present an algorithm that autonomous vehicles can follow to travel through a grid network of intersections, without causing collisions. In order to reduce the amount of inter-vehicle conflicts on the traffic throughput, and reduce the computational complexity, the grid networks were considered to be one way. Even though currently this is not a scenario in most real life situations (it was shown that this is still a useful situation), it was still useful because it was still possible to reach each destination from each entrance, and it was shown that for a large enough network size one generally experiences only a minor detour. The proposed concept by Xi Lin et al. consists of two parts; first, all vehicles form virtual platoons based upon a predetermined network rhythm that guarantees that no collisions occur. Then, instead of optimizing the trajectory of each individual vehicle, only the choice of leaving and joining platoons is optimized, leading to a more computationally efficient algorithm. Furthermore, various routing protocols are presented. For these protocols, linear optimizations are provided that approximate the solutions.

3 Mathematical Model

First, a mathematical model for a single intersection will be constructed, after which this model can be extended to a network of intersections. This model will be based on the structure proposed by Miculescu and Karaman [3]. A regular four-way intersection will be considered, where n vehicles will cross this intersection. Each vehicle can either go left, straight or to the right from each lane. Each lane has a control region of fixed length L. Before a vehicle enters the control region of an intersection, it is assumed to drive at maximum speed. From the moment a vehicle enters the control region, the PFA and SPA will control the speed and thus the trajectory of the vehicle. The lanes are ordered from 0 up to and including 3, where we start with lane 0 in the most left lane, and go clockwise to continue numbering the lanes. We will refer to this situation as the **basic model**. Figure 1 illustrates an intersection in the basic model.

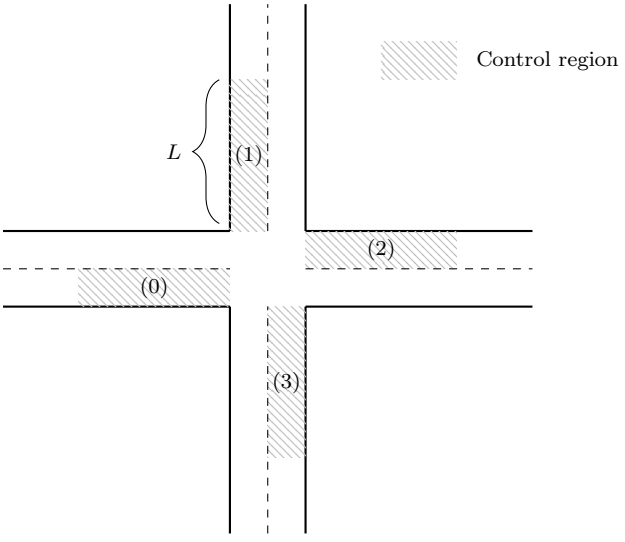


Figure 1: Illustration of an intersection.

Denote by vehicle V_i the i^{th} vehicle that has entered the system (for $i = 1, ..., n$). Each vehicle has the same maximum velocity v_{max}, minimum (negative) acceleration a_{min} and maximum acceleration a_{max}. Also, each vehicle has equal width w and length l.

For this model, the following assumptions will be made:

(i) All vehicles enter the control region at maximum velocity;

(ii) All vehicles exit the control region and thus cross the intersection at maximum velocity;

(iii) All vehicles entering the control region have a minimum distance of one car between them and the car before them;

(iv) Vehicles arrive at each lane according to a Renewal process; the interarrival times are distributed according to $\max(\text{Exponential}(\lambda), c)$ for a certain constant c, denoting the minimum time between two subsequent vehicles;

(v) Only vehicles from one lane can cross the intersection at the same time;

(vi) When the lane that is allowed to cross the intersection switches, this is accompanied by an additional deterministic switch-over time S before vehicles in the next lane can cross.

Observe above that it is assumed that whenever the lane that is allowed to cross the intersection is switched, a switch-over time S is included. This means that if at time $\tilde{t}$, lane l_j is not allowed to cross the intersection anymore, the next lane l_k will be allowed to start crossing the intersection at time $\tilde{t} + S$. This extra time is added to guarantee vehicles do not collide, improving the safety of our model. There will however still potentially be conflicts between vehicles, if too many vehicles arrive too quickly.

We will define the position of vehicle V_i as its distance to the intersection. The end of the control region where the vehicle starts to cross the intersection is defined to have position 0, so when the vehicle enters the control region, it has position $-L$. The time at which vehicle V_i enters the control region is called its arrival time $t_0^{(i)}$. The time at which the vehicle actually crosses the intersection is called its crossing time $t_f^{(i)}$. We can then also define the delay of a vehicle crossing the intersection. The delay of the vehicle will be interpreted as the *extra* time that it takes the vehicle to reach the intersection. If a vehicle would not be delayed, it would constantly drive at full speed. Therefore, it would take the vehicle $\frac{L}{v_{max}}$ seconds to cross the control region. Thus, we can then define the delay D_i of vehicle V_i by:

$$D_i := \left(t_f^{(i)} - t_0^{(i)}\right) - \frac{L}{v_{max}} \geq 0 \tag{1}$$

Observe that this basic model can be seen as a regular queueing model with 4 queues and one roving server. We can interpret the queues as follows: at time t, queue q_j consists of all vehicles in the control region of lane number j that can either cross the intersection at time t, or have a minimum distance to a car that is in the queue. The latter situation corresponds to a vehicle that is in the platoon that can cross the intersection at time t, but is not the first vehicle of the platoon.

3.1 Platoon forming algorithms

When vehicle V_i enters the control region at time $t_0^{(i)}$, a PFA will determine the crossing time $t_f^{(i)}$ of V_i. In this project, two different algorithms will be considered: the exhaustive and k-limited algorithm. As mentioned before, there is a central controller present at the intersection that notifies the vehicles at which time they are allowed to cross the intersection. This controller will not be a traffic light anymore, but simply a computer that can execute an algorithm and then send signals to the vehicles. Therefore, using either the exhaustive or k-limited algorithm, the central controller will compute the crossing time of vehicle V_i and then send it to that vehicle, such that it will cross at the correct time.

Notice that queue q_j represents the vehicles in lane number j that are ready to cross the intersection. This does not include the vehicles that are already in the control region of the lane, but still need some time before they can cross the intersection. Thus, in the queueing model, vehicles suddenly arrive in the queue when they are ready to cross the intersection. This is contrary to the real life situation, in which one can see the vehicles driving in the control region that will later be added to the queue. This is however captured in our model as well; the information of the vehicle is sent to the central controller at the moment it enters the control region, with which the controller will compute its crossing time. During the time that the vehicle is not yet in the queue, it is already

taken into account when scheduling the crossing times of other vehicles arriving in the system. Also, the crossing time of a vehicle can still be altered during the time it is in the control region but not yet in the queue, as a result of vehicles arriving in other lanes of the intersection. The time that a vehicle spends in the control region while not being in the queue is thus valuable for our model. This importance is emphasized by realizing that this information is necessary to compute the delay of the vehicle. The purpose of having a queue containing vehicles that are ready to cross the intersection is to be able to properly execute the PFA, since this schedules the vehicles based on the earliest times at which they can cross the intersection.

When using the exhaustive algorithm, the server keeps serving customers in the queue until the queue is empty [7]. The server then moves to the next queue. In our model, this means that once vehicles in lane l_j are allowed to cross the intersection, this lane will continue to be allowed to cross the intersection as long as at that time, either a vehicle in that lane is currently crossing the intersection or a vehicle in that lane is currently at the end of the control region, ready to cross the intersection. If both scenarios are not the case, meaning that a certain time needs to pass before the next vehicle in lane l_j can cross the intersection, that lane is not allowed to cross anymore. The server moves to the next lane, meaning that after the switch-over time S has passed, the next lane in clockwise direction with a non-empty queue is now allowed to cross the intersection. This process repeats itself until n vehicles have crossed the intersection.

Another algorithm that is often used in a queueing model such as ours, is the k-limited algorithm. With the k-limited algorithm, the server keeps serving customers in the queue until either the queue is empty or it has served k customers since arriving in said queue [7]. Within our model, this resembles the exhaustive algorithm with only one added constraint. When a vehicle in that lane has just finished crossing the intersection at time t, the next car in that lane is only allowed to cross if it is at the end of the control region at time t, *and* less than k vehicles have crossed the intersection in that lane since they were allowed to cross. If either one of these criteria is not satisfied, the next lane in clockwise direction with a non-empty queue after the switch-over time S has passed is allowed to cross. Again, this process continues until n vehicles have crossed the intersection.

In this project, we will consider and compare the performance of both algorithms. They will determine the exact crossing time of each vehicle in the system. After each vehicle is assigned a crossing time, the trajectory of the vehicle in the control region can be computed, which can be found using a SPA.

3.2 Theoretical background

Generally in queueing theory, arrivals are modelled according to a Poisson process. In our basic model however, we need to make sure that each vehicle entering the control region does so with enough distance to its predecessor. This is why we require that our inter-arrival times are the maximum of an exponential random variable and a constant.

3.2.1 $M/D/1$ queueing system

When aiming to analyze the model for a single intersection analytically, one can also see the arrival process from a different perspective. Consider a $M/D/1$ queueing system. The $M/D/1$ queueing system is a queueing model with one server, in which customers arrive according to a Poisson(λ) process and have a deterministic service time of $\frac{1}{\mu}$. The $M/D/1$ model serves customers on a first come first serve basis.

Let L denote the number of customers in a queue of the $M/D/1$ queueing system, S denote the sojourn time and W denote the waiting time of a customer. The $M/D/1$ queueing system satisfies the PASTA property, stating that arriving customers find on average the same situation in the queueing system as an outside observer looking at the system at an arbitrary point in time [8]. As a result, the average number of customers in the system seen by an arriving customer i equals $\mathbb{E}(L)$. All the customers that are still waiting in the queue will have a service time of $\frac{1}{\mu}$, whereas

the one currently in service will have a mean residual service time of $\frac{1}{2\mu}$. Hence, it follows that:

$$\mathbb{E}(W) = \mathbb{E}(L) \cdot \frac{1}{\mu} + \frac{1}{2\mu}$$

Additionally, Little's law is valid for a $M/D/1$ queueing system [8], stating that

$$\mathbb{E}(L) = \lambda \mathbb{E}(W)$$

Furthermore, the relation between the sojourn and waiting time of a customer can be used:

$$\mathbb{E}(S) = \mathbb{E}(W) + \frac{1}{\mu}$$

We can then solve this set of equations to derive the expected sojourn time:

$$\begin{aligned}
&\mathbb{E}(W) = \mathbb{E}(L) \cdot \frac{1}{\mu} + \frac{1}{2\mu} \\
\implies &\mathbb{E}(W) = \lambda \mathbb{E}(W) \cdot \frac{1}{\mu} + \frac{1}{2\mu} \\
\implies &\mathbb{E}(W) \left(1 - \frac{\lambda}{\mu}\right) = \frac{1}{2\mu} \\
\implies &\mathbb{E}(W) = \frac{1}{2(\mu - \lambda)} \\
\implies &\mathbb{E}(S) = \frac{1}{2(\mu - \lambda)} + \frac{1}{\mu}
\end{aligned}$$

The above derivation shows the mean sojourn time of a customer in the $M/D/1$ queueing system. Now, consider the arrival time of a customer in the $M/D/1$ queue with setup times. Then, the arrival time of a customer follows the distribution $\frac{1}{\mu} + X$, where X is a random variable that is 0 with probability ρ, and is an exponential $Exp(\lambda)$ random variable with probability $1 - \rho$. Then, observe that the inter-arrival times of customers in our basic model follow the distribution of the maximum of a constant and an $Exp(\lambda)$ distribution. Thus, we can see that the distribution of the inter-arrival times of our basic model relate to the distribution of the arrival times in an $M/D/1$ queueing system.

Now that it is established that the $M/D/1$ queueing system is relevant to our model, it can be used to analyze our arrival process. In queueing systems, one can distinguish cycles. A cycle consists of two parts; first, a *busy period* BP in which the server is serving customers non-stop, followed by an *idle period* IP during which the server is not serving customers, and is waiting for the next customer to arrive. The expected length of the busy period is therefore related to the expected time during which customers arrive with a minimum inter-arrival time c in our basic model.

Define ρ to be the fraction of time that the server is working. It then follows that [8]:

$$\frac{\mathbb{E}(BP)}{\mathbb{E}(BP) + \mathbb{E}(IP)} = \rho$$

Recall that the inter-arrival times are exponentially distributed. Thus, as a result of its memoryless property, we know that the expected length of the idle period equals the expected inter-arrival time,

which equals $\frac{1}{\lambda}$. Then, an expression for the length of the busy period can be derived:

$$\begin{aligned}
&\frac{\mathbb{E}(BP)}{\mathbb{E}(BP)+\mathbb{E}(IP)} = \rho \\
\implies &\frac{\mathbb{E}(BP)}{\mathbb{E}(BP)+\frac{1}{\lambda}} = \rho \\
\implies &\rho\left(\mathbb{E}(BP)+\frac{1}{\lambda}\right) = \mathbb{E}(BP) \\
\implies &\mathbb{E}(BP)(1-\rho) = \frac{\rho}{\lambda} = \frac{1}{\mu} \\
\implies &\mathbb{E}(BP) = \frac{1}{\mu(1-\rho)}
\end{aligned}$$

3.2.2 Polling systems

Another interesting feature of a model is the conditions under which the system is stable. As mentioned in section 3.1, the exhaustive and k-limited algorithm will be considered to schedule the crossing times of our vehicles. As a result, our basic model can be seen as a polling system. Within polling systems, these stability conditions have already been derived.

A polling system consists of a single server and N queues. The server serves only one queue at a time, and decides which queue is served based on the polling policy. Within our basic model, this policy is either the exhaustive or the k-limited algorithm. Consider a polling model with N queues and a single server. In each queue q_j, customers arrive according to a Poisson process with arrival rate λ_j. Service times of customers within queue q_j are independent and identically distributed with a mean equal to $\frac{1}{\mu_j}$. Additionally, when the server switches queues, it takes a switch-over time S before the server can start serving customers in the next queue. Figure 2 illustrates a typical polling system [9].

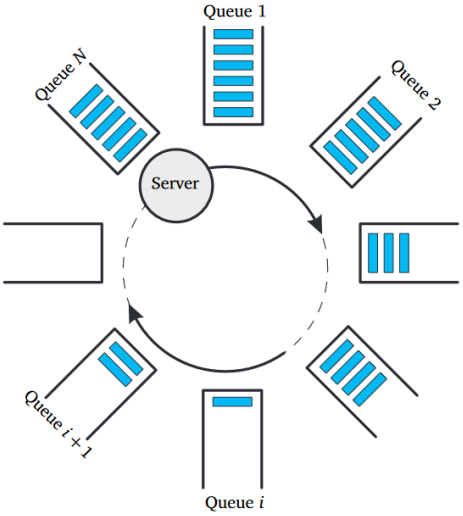


Figure 2: A typical polling system.

The stability condition for the exhaustive discipline is given by [10]

$$\sum_{j=1}^{N} \frac{\lambda_j}{\mu_j} < 1$$

The stability for the k-limited policy is relatively more complicated. Consider the cycle time C, defined as the time between two consecutive times that the server arrives at queue q_j for

$j \in \{1, ..., N\}$. Let ρ_j be the load (of the system) in queue q_j, then the total load of the system is given by $\rho = \sum_{j=1}^{N} \rho_j$. Let S_t be the total switch-over time during cycle C. Then, the expectation of the cycle length is given by [11]

$$\mathbb{E}(C) = \frac{\mathbb{E}(S_t)}{1-\rho}$$

When using the k-limited service discipline, where the server serves at most k_j customers during one visit in queue q_j, we can then derive a condition for the stability of the system. In each queue q_j, at most k_j customers can leave the queue in one cycle. Therefore, at most k_j customers can arrive during one cycle in order for the system to remain stable. Hence, the stability condition $\forall j \in \{1, ..., N\} : \lambda_j \mathbb{E}(C) < k_j$ [11] can be derived. Thus, we can then derive that it should hold that

$$\forall j \in \{1, ..., N\} : \frac{\lambda_j \mathbb{E}(S_t)}{1-\rho} < k_j.$$

Thus, the stability condition of the k-limited discipline is given by

$$\min_{j=1,...,N} \left(\frac{k_j}{\lambda_j} < \frac{\mathbb{E}(S_t)}{1-\rho} \right).$$

3.3 Network of intersections

The goal of this project is to analyze vehicles crossing a network of intersection. Therefore, we wish to extend the basic model to one describing the behavior of vehicles in a network of intersections. A model for a network of intersections will be constructed, containing m rows and m columns of intersections, thus a network containing a total of m^2 intersections. We will henceforth refer to this as a $m \times m$ network of intersections. For this extension, we will need the following *additional* assumptions:

(i) Each intersection has 4 lanes in which vehicles can arrive;

(ii) The inter-arrival times are independent and identically distributed in each external lane. They all follow the same Renewal process, with inter-arrival times given by $\max(\text{Exponential}(\lambda), c)$ for a certain constant c;

(iii) The control region of each intersection is of equal length L;

(iv) After crossing an intersection, if a vehicle moves to intersection I_0, it immediately enters the beginning of the control region of the corresponding lane of intersection I_0;

(v) The service time of vehicles going left and right takes s_{turn} seconds in each intersection, and the service time of a vehicle going straight takes s_{straight} seconds, where $s_{\text{straight}} < s_{\text{turn}}$;

(vi) Every vehicle can leave the network from each lane of each intersection, this corresponds to the vehicle reaching its destination.

For a $m \times m$ network, consider the intersections $I = \{0, ..., m^2 - 1\}$. The intersections are ordered as one would read a book; the i^{th} row is numbered $(i-1) \cdot m$ up to and including $i \cdot m - 1$ from left to right, for $i = 1, ..., m$. Each intersection then has the queues $\{0, 1, 2, 3\}$, which are ordered similarly to the ordering in the basic model. Thus, each vehicle is assigned an intersection number, a lane number and a direction. This direction is sampled from a probability distribution, where with a certain probability each vehicle can go left, straight, right or leave the network. Within our model, when a vehicle's sampled direction is to leave the network, this means that the vehicle first crosses the intersection, after which it reaches its destination. This means that in case a vehicle will leave the network from intersection I_0, it still crosses intersection I_0 after which it is not moved to another intersection anymore. Within each intersection, the trajectory of the car is determined by a SPA, and the ordering in which vehicles can cross the intersection is scheduled by the PFA. Within this model, when vehicles enter the system they always do so in an external lane, which are positioned around the edges of the network; the external lanes are those in the first and last column and row of the network, where vehicles arrive when driving on roads that are connected to the network.

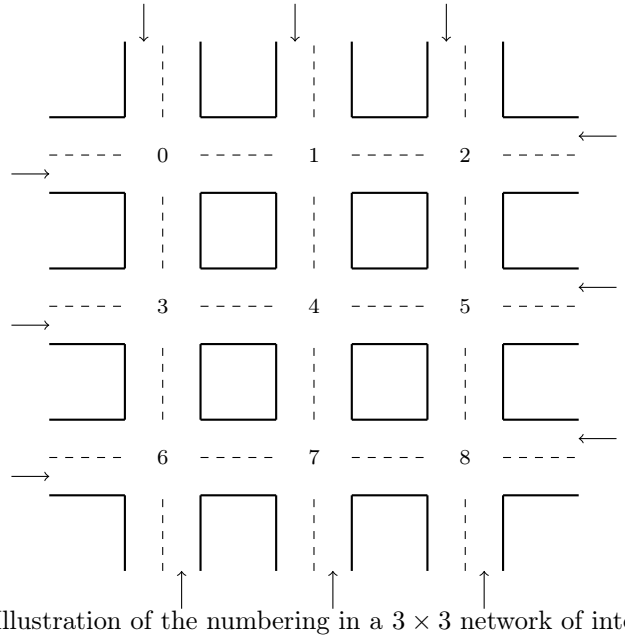


Figure 3: Illustration of the numbering in a 3×3 network of intersections.

Figure 3 provides an illustration of the ordering of the intersections within a 3×3 network. The arrows indicate the external lanes of the network.

3.4 Speed profiling algorithms

An important property of autonomous vehicles is that the speed of the vehicle can be controlled at all times. This is a large difference between autonomous vehicles and human-driven cars: humans can attempt to follow instructions concerning how fast they should drive, but this is very difficult. Autonomous vehicles on the other hand can do this accurately, therefore allowing autonomous vehicles to cross the intersection more efficiently.

A key property of the SPA's that will be considered is that they force the vehicles to arrive at the intersection at maximum velocity. This ensures that the intersection is used as efficiently as possible; if all vehicles drive across it at full speed, each vehicle will use it as short as possible, meaning that the next vehicle can cross as soon as possible.

This project will consider two SPA's; both algorithms solve a linear optimization problem that satisfies the following constraints:

$$\begin{cases} \text{Each vehicle } V_i \text{ arrives at the control region at time } t_0^{(i)} \\ \text{Each vehicle } V_i \text{ crosses the intersection at maximum velocity } v_{\max} \text{ at time } t_f^{(i)} \\ \text{Each vehicle } V_i \text{ maintains at least a fixed distance } l \text{ from its predecessor} \end{cases} \tag{2}$$

Miculescu and Karaman first proposed such an algorithm as a SPA. In addition to constraints (2), their algorithm, called MotionSynthesize, had such an objective that ensures each vehicle is as close to the intersection as possible [3]. This ensures that as many vehicles as possible can enter the control region, therefore allowing the vehicles to pass the intersection as soon as possible given the constraints (2). Suppose that one wants to determine the trajectory of vehicle V_i in lane l_j. An even more general setting is provided, in which the trajectory is determined between the initial position x_0 of vehicle V_i and the intersection, where V_i had a velocity of v_0 when arriving at position x_0. Let vehicle V_k be the vehicle that crossed the intersection from lane l_j before vehicle V_i. Then, all the ingredients one needs to know in order to solve the optimization problem for vehicle V_i, are $x_0, v_0, t_0^{(i)}, t_f^{(i)}, t_f^{(k)}$ and the trajectory of vehicle V_k, denoted by y.

Let x be a function describing the position of a vehicle. Then the MotionSyntesize procedure is given by Algorithm 1.

Algorithm 1 MotionSynthesize procedure minimizing distance to the intersection

1: **Input:** $x_0, v_0, t_0^{(i)}, t_f^{(i)}, t_f^{(k)}, y$
2: **Calculate**

$$\text{MotionSynthesize}(x_0, v_0, t_0^{(i)}, t_f^{(i)}, t_f^{(k)}, y) = \underset{x:[t_0^{(i)}, t_f^{(i)}]\to\mathbb{R}}{\arg\min} \int_{t_0^{(i)}}^{t_f^{(i)}} |x(t)|\, dt$$

subject to:

$$a_{\min} \leq \ddot{x}(t) \leq a_{\max} \text{ for all } t \in [t_0^{(i)}, t_f^{(i)}]$$
$$0 \leq \dot{x}(t) \leq v_{\max} \text{ for all } t \in [t_0^{(i)}, t_f^{(i)}]$$
$$|x(t) - y(t)| \geq l \text{ for all } t \in [t_0^{(i)}, t_f^{(k)}]$$
$$x(t_0^{(i)}) = x_0;\ \dot{x}(t_0^{(i)}) = v_0$$
$$x(t_f^{(i)}) = 0;\ \dot{x}(t_f^{(i)}) = v_{\max}$$

Timmerman and Boon then created a different version of this optimization problem; instead of minimizing the distance to the intersection, they proposed to minimize the absolute acceleration. This ensures that the trajectory of a vehicle is as smooth as possible, making the drive more pleasant [5]. The only aspect that needs to change in the algorithm of Miculescu and Karaman is to alter the objective of the optimization problem to minimize the absolute value of the acceleration of the vehicle. A slightly different implementation of this algorithm was however made. In order to reduce the computational complexity of the algorithm, it was chosen to minimize the square of the acceleration instead of the absolute value of the acceleration. This will result in slightly different trajectories, but still have the property that the realized trajectory is a smooth function. This procedure is given in Algorithm 2.

Algorithm 2 MotionSynthesize procedure minimizing the absolute acceleration

1: **Input:** $x_0, v_0, t_0^{(i)}, t_f^{(i)}, t_f^{(k)}, y$
2: **Calculate**

$$\text{MotionSynthesizeSmooth}(x_0, v_0, t_0^{(i)}, t_f^{(i)}, t_f^{(k)}, y) = \underset{x:[t_0^{(i)}, t_f^{(i)}]\to\mathbb{R}}{\arg\min} \int_{t_0^{(i)}}^{t_f^{(i)}} \ddot{x}^2(t) dt$$

subject to:

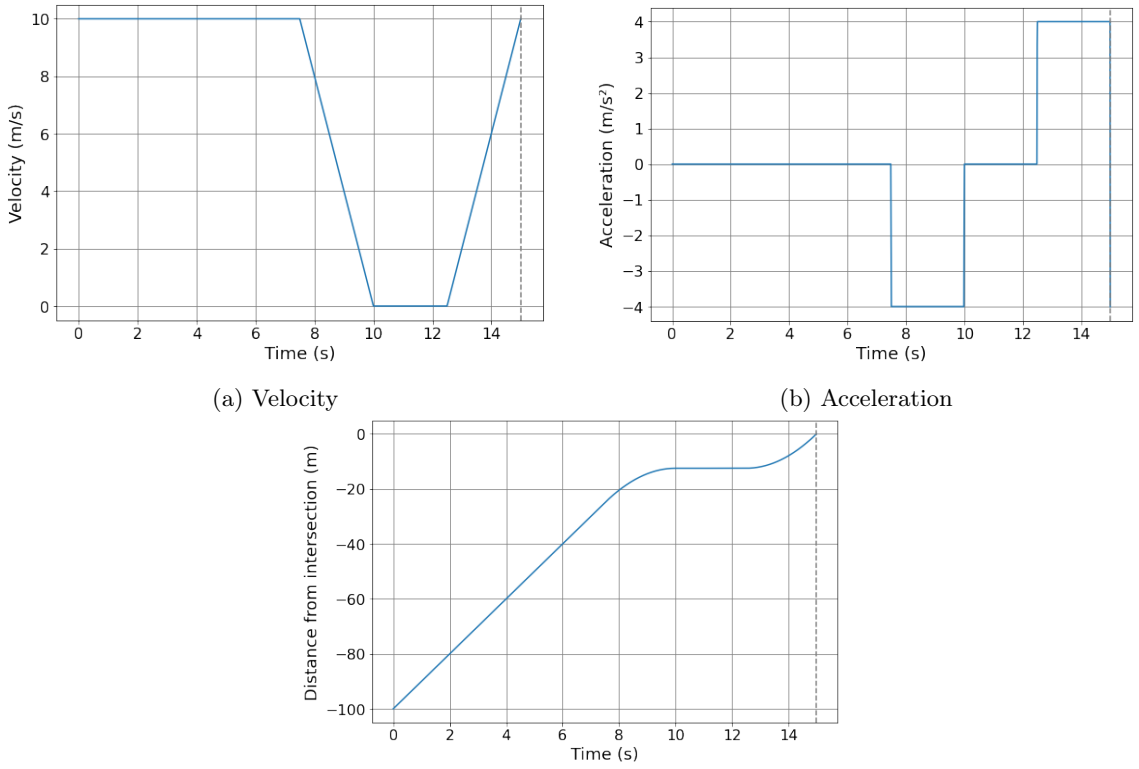
$$a_{\min} \leq \ddot{x}(t) \leq a_{\max} \text{ for all } t \in [t_0^{(i)}, t_f^{(i)}]$$
$$0 \leq \dot{x}(t) \leq v_{\max} \text{ for all } t \in [t_0^{(i)}, t_f^{(i)}]$$
$$|x(t) - y(t)| \geq l \text{ for all } t \in [t_0^{(i)}, t_f^{(k)}]$$
$$x(t_0^{(i)}) = x_0;\ \dot{x}(t_0^{(i)}) = v_0$$
$$x(t_f^{(i)}) = 0;\ \dot{x}(t_f^{(i)}) = v_{\max}$$

3.5 Verification

A uniform discretization of Algorithms 1 and 2 is implemented in Python. This is relatively straightforward, pseudo code of this implementation, Algorithm 5, is provided in the Appendix.

Consider a vehicle in an arbitrary lane that has arrived at the control region at time $t = 0$, and will be scheduled to cross the intersection at time $t = 15$. Additionally, suppose that the predecessor of this vehicle has crossed the intersection before the vehicle has arrived, meaning that this vehicle will not have to take into account maintaining distance to his predecessor. We will compare the driving behavior of this vehicle for the two SPA's. It is assumed that the control region has length $L = 100\ m$, the maximum velocity of the vehicle is 10 m/s, and the minimum and maximum acceleration are -4 and 4 m/s^2 respectively. Figure 4 shows the position, velocity and acceleration of the vehicle when aiming to minimize the distance to the intersection, whereas Figure 5 shows the position, velocity and acceleration of the vehicle when aiming to minimize the absolute acceleration.

Notice that Figure 4 shows the expected behavior. Combining Figures 4a, 4b and 4c, one can see that the vehicle drives at full speed as long as possible, to slow down at the last possible moment such that it will arrive at the end of the control region at its scheduled time $t = 15$, at maximum velocity. This means that indeed, the vehicle is as close to the intersection as possible, given our constraints (2). Finally, observe that all the constraints of the corresponding linear optimization problem, Algorithm 1, are satisfied; the minimum and maximum velocity and acceleration is not exceeded, and the vehicle enters and leaves the control region at the correct times.



(a) Velocity

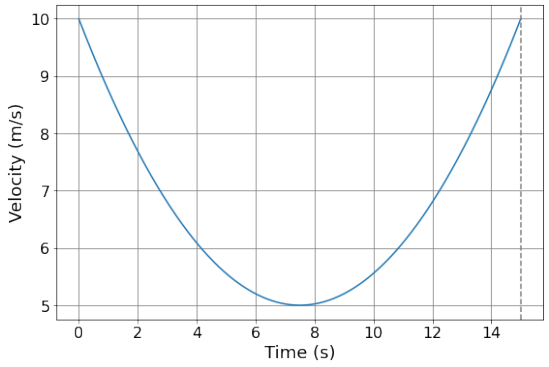
(b) Acceleration

(c) Position

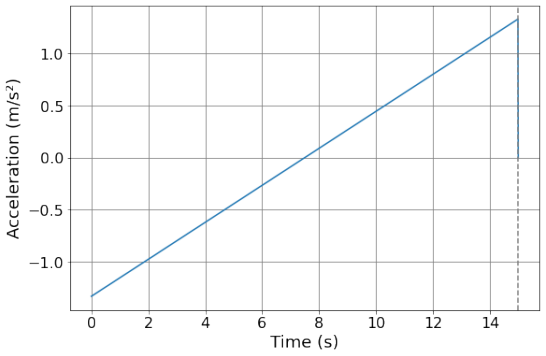
Figure 4: Position, velocity and acceleration of an arbitrary vehicle that minimizes the distance to the intersection.

Figure 5 illustrates the behavior of the vehicle when aiming to minimize the absolute acceleration of the vehicle. Again, the expected behavior is visible in Figure 5. First of all, notice in Figure 5a that the vehicle both enters and exits the control region at maximum velocity. This time however, the vehicle does not continue to drive at maximum velocity as long as possible, but starts to decelerate slightly after entering the control region, as can be seen in Figure 5b. When the

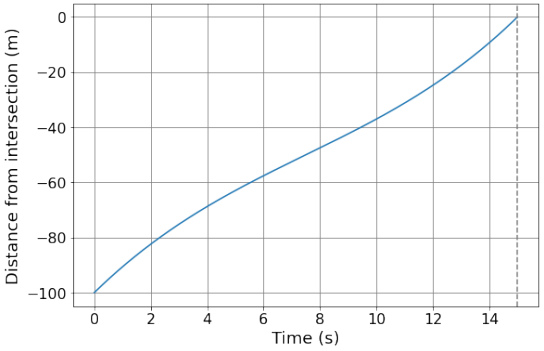
vehicle has driven halfway to the intersection, it slowly starts to accelerate again in order to reach the intersection at maximum velocity at time $t = 15$. Observe that again, all of the constraints corresponding to optimization problem 2 are satisfied; the vehicle enters and leaves the control region at the correct times at full speed, and the maximum and minimum acceleration and velocity are not exceeded. Observe that the trajectory computed for the vehicle when applying Algorithm 2 is indeed significantly smoother compared to the trajectory when applying Algorithm 1.



(a) Velocity



(b) Acceleration



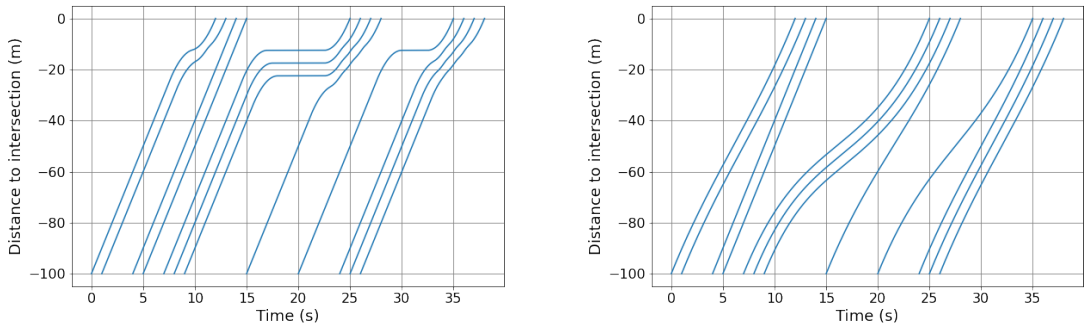
(c) Position

Figure 5: Position, velocity and acceleration of an arbitrary vehicle that minimizes the absolute acceleration.

Subsequently, a scenario will be investigated in which the vehicles do need to consider the position of their predecessors. Consider 12 vehicles V_i for $1 \leq i \leq 12$, that are scheduled to have arrival and crossing times as given in Table 1. Again, the control region has length $L = 100\ m$, vehicles have a maximum velocity of 10 m/s, and a minimum and maximum acceleration of -4 and 4 m/s^2 respectively. The required distance between two consecutive vehicles is set to 5 meters.

Vehicle V_i	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}	V_{11}	V_{12}
Arrival time (s)	0	1	4	5	7	8	9	15	20	24	25	26
Crossing time (s)	12	13	14	15	25	26	27	28	35	36	37	38

Table 1: Arrival and crossing times.



(a) SPA 1 (b) SPA 2

Figure 6: Trajectories of vehicles having arrival and crossing times as defined in Table 1 for both MotionSynthesize Algorithms.

Figure 6 shows the trajectories of these vehicles. Figure 6a shows the trajectories when applying MotionSynthesize Algorithm 1, while Figure 6b shows the trajectories when MotionSynthesize Algorithm 2 is applied. Notice that in both figures, all vehicles keep the same required distance to their predecessor.

4 Network of intersections

In the previous section, we have verified that the SPA's have been implemented correctly. In this section, the implementation that simulates an entire $m \times m$ network of intersections will be presented.

4.1 Routing matrix

A first algorithm that is necessary to simulate a network of intersections, is one that creates a matrix D describing the routing within the network. Each vehicle can either go left, straight, right or leave the network from each lane of each intersection. When this destination is known, we need to know whether either the vehicle exits the network, or we need to know the new intersection and lane number of the vehicle. Therefore, a matrix is created that provides this information. Thus, given that a vehicle is in intersection $I_0 \in \{0, 1, ..., m^2 - 1\}$ and queue $q \in \{0, 1, 2, 3\}$, we want to know the new destination of the vehicle if it were to go left, straight and right. This means that we want to know the corresponding new intersection and queue number of the vehicle. If a vehicle leaves the network, its intersection and lane number will be set to -1. Because of the way our lanes and intersections are numbered, the new destination of a vehicle follows a general structure. Since this structure is not that complicated, Algorithm 6 that provides the code creating the matrix is given in the Appendix.

4.2 Implementation

We have now presented both the SPA's as well as the routing matrix. These two are the first elements necessary for our simulation. We can now continue to the core of our simulation. The simulation consists of 5 classes, each having a specific purpose. They are presented below.

4.2.1 Vehicle class

First of all, our simulation contains a *Vehicle* class. This class contains and can alter all the characteristics of a vehicle. During the simulation, each time a vehicle arrives at the control region, a corresponding object *Vehicle* is created. This object knows this vehicle's arrival time, scheduled crossing time, current lane and intersection number, direction it will go, the intersections and lanes that the vehicle has visited called *route*, and whether it already has a departure scheduled. All of

these characteristics of the vehicle will be used when in a later class, the PFA schedules crossing times for vehicles. The class contains some functions that can alter the values of these parameters accordingly.

4.2.2 Event class

The *Event* class contains three different types of events; an arrival, departure and switch event. These events all come with several parameters. Each event has a corresponding type, time, vehicle and intersection number. A switch event has an additional parameter, namely a boolean *begin* denoting whether it is the beginning or the end of the switch-over time. An arrival event is accompanied by a boolean *external*, which denotes whether or not the arrival event corresponds to an external arrival.

4.2.3 Future Event Set

Throughout the simulation, the Future Event Set manages the ordering of the events. Whenever an event is created, it can be added to the Future event Set, which then places it at the correct chronological position. The Future Event Set also has a function that returns the event that will take place next, which is crucial in simulating our model.

4.2.4 Results class

Another crucial class is *Results*, which saves all the necessary data that is generated within the simulation. It contains and can return the route that each vehicle has followed, with the corresponding arrival and crossing times at each intersection. From this information, we can later derive the delays that the vehicles experienced at each intersection.

4.2.5 IntersectionSimulation class

The process of all vehicles arriving and crossing intersections throughout the network is simulated within the class *IntersectionSimulation*. First, it initializes certain parameters; the number of intersections, the total number of vehicles n that will cross the network, the arrival rate λ, the value k corresponding to the scheduling policy and the moving probabilities π, being the probability with which vehicles go left, straight, right or leave the network. Then, a function *simulate* is called that simulates the entire model.

Within *simulate*, first several arrays are initialized:

- An object *res* of the *Results* class is created, which will contain all the desired data of the simulated vehicles.
- An object *fes* will be created, which is the Future Event Set.
- A boolean array *working* is created, which contains if at each intersection the server is switching or not. If at time t the server is switching lanes in intersection I, *working* is set to *False* at the index of that intersection I at time t.
- An array *kPerIntersection* is created, containing for each intersection the number of vehicles that have started crossing the intersection since the last time the server switched lanes.
- Arrival times are sampled for the first arrivals in all the external lanes, the corresponding earliest possible crossing times are computed and vehicles are created accordingly. Arrival events are added to the future event set for all of these vehicles.

Then, we move on to the actual arrivals and departures of vehicles. *simulate* continues to sample the next event e from the future event set until there are no events scheduled anymore. For each event e, the time is updated to the time of that event. Then, the simulation checks if e is a switch, an arrival or a departure event. It then executes the appropriate procedure in the corresponding intersection I and queue q.

The switch procedure is relatively straight forward. There are two types of switch events, since each switch event e is accompanied by a parameter *begin*. If *begin=True*, this means the switch-over time will start. If this is the case, the boolean *working* needs to be set to *False* at intersection I, since no vehicles can cross during a switch-over time. Then, a new switch event with *begin=False* needs to be added to *fes* at time $t + S$, where S equals the switch-over time. If it is a switch event with *begin=False*, the algorithm starts by setting *working* to *True* at intersection I. It then checks if there is a non-empty queue besides q. If this is the case, the first non-empty queue $\tilde{q}$ in clock-wise direction will be allowed to cross the intersection. The procedure then finds the first car in this queue $\tilde{q}$ that does not have a departure scheduled, checks the destination and corresponding service time of the vehicle and schedules its departure accordingly. It then sets *kPerIntersection* to 1 for intersection I. If there was no non-empty queue $\tilde{q}$, the procedure still needs to check if there are vehicles waiting in queue q. If this is the case, the procedure will schedule the departure of the first car waiting that does yet not have a departure scheduled. *kPerIntersection* is then increased with 1 in I.

Algorithm 3 displays the procedure that handles the arrival event. Recall that the service time of a vehicle, being the time it takes a vehicle to cross the intersection, depends on the direction of the vehicle; going left and right takes longer than going straight. This is taken into account by having a parameter *prevServ* that keeps track of the service time of the last vehicle that crossed the intersection in each queue.

Algorithm 3 Handling the arrival event

```
1: Set v = e.vehicle, t = e.time, I = v.intersectionNr, q = v.queueNr
2: if I ≥ 0 then                                   ▷ Check if vehicle v is still in the system
3:     Add vehicle v to queue q
4:     Sample the destination of v from π, update v.direction and v.serviceTime accordingly
5:     if v is the first vehicle to arrive at intersection I then
6:         Schedule departure of v at t + v.serviceTime
7:         Set kPerIntersection to 1 in intersection I
8:     else
9:         if server is not switching in I and v does not have a departure scheduled then
10:            if all the other queues in intersection I are empty then
11:                if lane q is already allowed to cross then
12:                    Set depTime = max (t, prevTime + prevServ)
13:                else                    ▷ prevTime is the crossing time of predecessor of vehicle v
14:                    Set depTime = max (t, prevTime + prevServ + switchTime)
15:                    Set kPerIntersection to 0 in intersection I
16:                end if
17:                Schedule the departure of vehicle v at time depTime + v.serviceTime
18:                Increase kPerIntersection with 1 in intersection I
19:            end if
20:        end if
21:    end if
22:    if e was an external arrival and less than n vehicles have arrived in the network then
23:        Sample interArrTime from an Exponential(λ) distribution          ▷ λ is the arrival rate
24:        Set arr = max(interArrTime, minArrDiff) + v.arrivalTime
25:        Compute the earliest possible crossing time cross
26:        Create vehicle v_1 and add an external arrival event for v_1 to fes at time t=cross
27:    end if
28: end if
```

Lines $21-25$ describe the scheduling of external arrivals. These are taken into account by sampling the new arrival time in lane q each time an external arrival event is processed. Thus, if a vehicle v enters the system, it has an external arrival event, and within these lines of code the next arrival time of a vehicle in that lane is sampled. This arrival is again an external arrival, causing the next external arrival in this lane. This is executed for all external lanes of the network, therefore

correctly scheduling all of the arrivals. Notice that the inter-arrival times follow the described Renewal process of the maximum of an exponential distribution and a constant, where this constant *minArrDiff* equals the minimum difference between the arrival times of consecutive vehicles.

Last but not least, a procedure that handles a departure event is necessary to simulate vehicles crossing the network. This procedure is given in Algorithm 4.

Algorithm 4 Procedure that handles a departure event

```
1:  Set v = e.vehicle, t = e.time, I = v.intersectionNr, q = v.queueNr
2:  if I ≥ 0 then                                   ▷ Check if vehicle v is still in the system
3:      Remove vehicle v from queue q
4:      if v.direction = -1 then                    ▷ Vehicle has reached its destination
5:          v.leaveNetwork()                        ▷ Removes vehicle v from the system
6:      else
7:          Set newIntersection and newLane of vehicle v using the routing matrix
8:          if newIntersection = -1 then            ▷ By going his direction v leaves the network
9:              v.leaveNetwork()
10:         else
11:             if v is the first vehicle to cross lane q of intersection I then
12:                 set arrTime = t                 ▷ v can arrive at new control region at time t
13:             else                                ▷ v needs to keep enough distance to other vehicles
14:                 set arrTime = max (t, prevArrTime + minArrDiff)
15:             end if
16:             v.moveTo(newIntersection, newLane, arrTime)
17:             Schedule arrival event of vehicle v at time arrTime accordingly
18:         end if
19:     end if
20:                 ▷ Check according to the PFA if another vehicle can now cross intersection I
21:     if queue q is not empty and less than k vehicles have crossed the intersection then
22:         Find the first vehicle v_1 in the queue that does not have a departure scheduled
23:         Schedule the departure of v_1 at time t + v_1.serviceTime
24:         increase kPerIntersection with 1 for intersection I
25:     else                                        ▷ Switch to the next lane that can cross the intersection
26:         Schedule a begin switch event at time t for intersection I
27:     end if
28: end if
```

The first purpose of the procedure handling a departure event is described in lines $3-19$: it removes the vehicle v that has crossed the intersection from its queue, and ensures that its parameters are updated correctly. When the vehicle arrived at intersection I, the direction of the vehicle is determined (line 4 of Algorithm 3). Recall that a direction of $0, 1, 2, -1$ corresponds to going left, straight, right or arrive at destination respectively. Thus, if the direction of a vehicle equals -1, line 5 calls the function *v.leaveNetwork()*. This is a function defined in the *Vehicle* class that sets the intersection number of a vehicle to -1, which corresponds to removing it from the network. If the vehicle goes left, straight or right, lines $10-18$ adjust the parameters of the vehicle accordingly. The function *moveTo()* is also defined in the *Vehicle* class. This function alters the intersection number, queue number, arrival time, crossing time and boolean *departureScheduled* accordingly. Notice that going left, straight or right can also cause the vehicle to exit the network. In this case, *moveTo()* sets the intersection and queue number of the vehicle to -1, meaning that the vehicle has left the network.

Consequently, lines $21-27$ check if, after the departure of v, another car can cross the intersection in that lane, or that it is time to switch the lane that is allowed to cross the intersection. This is done by either scheduling a departure event or a switch event.

Throughout the entire simulation, *res* keeps track of the routes that vehicles take, the times at which they are scheduled to arrive and cross the intersection, and who their predecessors are. This

data can then be used to compute the trajectories of the vehicles when applying SPA's 1 and 2.

4.3 Trajectories

Using the described simulation, we can now investigate the behavior of vehicles throughout the network. Within this section, the optimal trajectories of the vehicles crossing the network will be shown. Since visualizing trajectories throughout the entire network would be very chaotic, the situation is simplified slightly. Instead of vehicles arriving at all of the external lanes, vehicles will now only be able to arrive in lanes 1 of the top row of the network and in lane 0 of the left column of the network. Additionally, every vehicle will always go straight at every intersection. This will allow us to understand the arrivals of vehicles at each intersection of the network. Figure 7 shows the structure of this simplified version in case of a 3×3 network. The arrows indicate the lanes where external arrivals can occur. A vehicle is shown that enters the network in lane 0 of intersection 6. Since the vehicle can only go straight, the route that the vehicle is going to drive is known. Its route is shown in the Figure; from intersection 6 it continues to intersection 7, then to 8 after which it leaves the network.

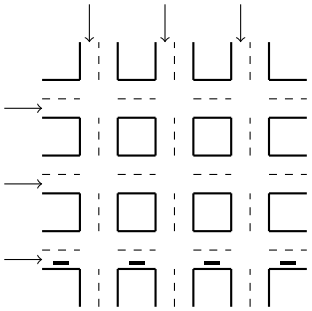


Figure 7: Illustration of simplified network in which vehicles always drive straight.

Consider 1000 vehicles crossing such a 9×9 network of intersections. The trajectories of vehicles crossing intersection $0, 40, 54$ and 80 will be visualized, when assuming SPA 1, minimizing the vehicles distance to the intersection. Intersection 0 corresponds to the top left intersection, which is the only intersection receiving external arrivals in two lanes. Intersection 54 is positioned in the most-left column of the network, meaning that it will receive external arrivals in row 0 and internal arrivals in lane 1. At intersection 40 as well as 80, the arriving vehicles always come from other intersections of the network. Whereas in intersection 40 they have passed 4 intersections, in intersection 80 they have already passed 8 intersections. Table 2 provides an overview of the values used for the parameters within the simulation.

L	l	w	v_{min}	v_{max}	a_{min}	a_{max}	s_{straight}	$minArrDiff$	S	k	λ
100	5	2	0	10	-4	4	1.5	1	1	5	$\frac{1}{3}$

Table 2: Values used for the parameters when simulating the trajectories.

Figure 8 shows trajectories of vehicles crossing intersection 0. Observe that indeed, starting from time $t = 0$ vehicles arrive in both lanes. Additionally, observe the pattern originating from using the k-limited PFA with $k = 5$; when 5 vehicles have subsequently crossed the intersection, the other lane is allowed to cross the intersection. Also, notice the slightly larger gap between the crossing times of vehicles when two vehicles in different lanes cross, compared to when two vehicles in the same lane cross. This is a result of the added switch-over time S. Notice that at the end of the simulation, corresponding to approximately $t = 210$, there is a slightly larger gap between crossing vehicles. Two vehicles in lane 1 cross the intersection, after which for about 10 seconds no vehicle crosses, followed by vehicles in lane 0 crossing. This behaviour is not what one would expect; one would think that first instead of 2, 5 vehicles in lane 1 would cross, after which lane 0 could cross after the switch-over time has passed. What happened in this case, is that the system became unstable. Within this 10 second gap, there were actually vehicles scheduled to cross the

intersection from lane 1. In reality however, when these vehicles arrived at the control region (around $t = 140$), there were too many vehicles already in the control region, resulting in those new vehicles not being able to come to a full stop from maximum velocity while maintaining enough distance to the vehicles in front. As a result, the vehicles were removed from the simulation, since they were not able to safely traverse the intersection. However, the structuring of our simulation is such that first all the crossing times of vehicles are scheduled, after which the trajectories are computed. Thus, if a vehicle does arrive minimally *minArrDiff* seconds later than the last car, but is not able to always maintain enough distance from its predecessor, this vehicle is included in the PFA, but its trajectory will not be shown as the vehicle will not actually cross the intersection.

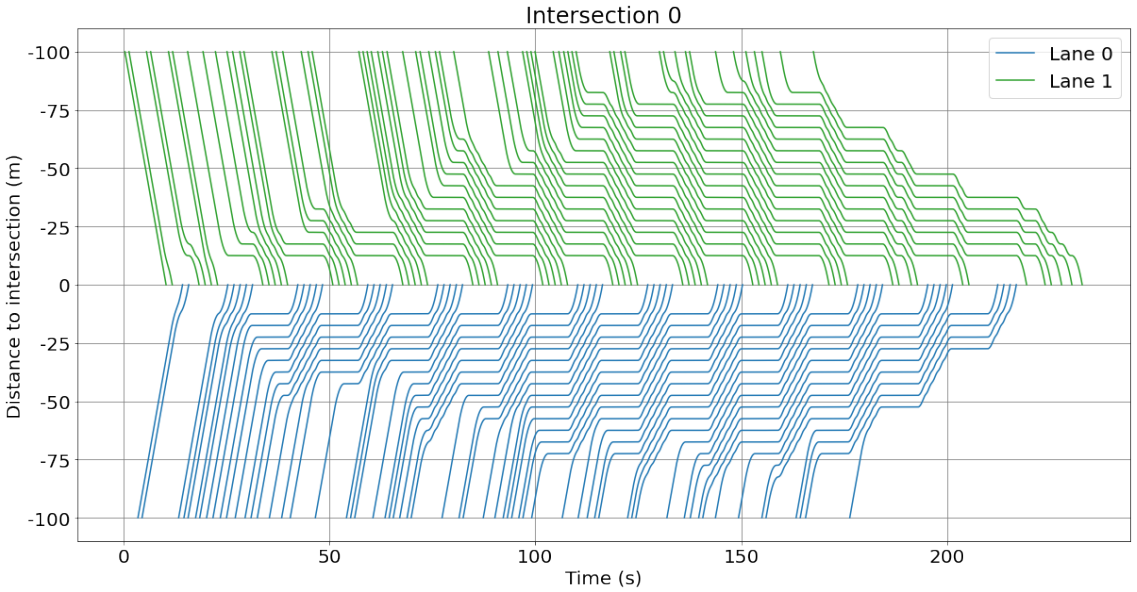


Figure 8: Trajectories of vehicles crossing intersection 0 in a 9×9 network.

Then, Figure 9 illustrates the trajectories of vehicles crossing intersection number 54. This intersection is positioned in the first column and 7^{th} row of the network. Therefore, it receives external arrivals in lane 0, but internal arrivals in lane 1. Since all the vehicles arriving in lane 1 first need to cross 6 other intersections, those vehicles cannot arrive earlier than at $6 \cdot (10 + 1.5) = 69$ seconds, which corresponds to a vehicle arriving at the first intersection at time 0 and having no delay before arriving at intersection 54. Indeed, Figure 9 shows that the earliest vehicle arriving in lane 1 arrives around $t = 75$. Before the arrival of this vehicle, the vehicles in lane 0 were always allowed to cross the intersection. Observe that indeed the only delay that they experience is due to having to keep enough distance from their predecessors. During the time that vehicles arrive in both lanes, the k-limited algorithm ensures that at most 5 vehicles cross the intersection consecutively. Observe however that around $t = 145$, more than 5 vehicles subsequently cross the intersection in lane 0. This is due to the fact that even though 5 vehicles have already crossed in lane 0, there is no vehicle ready to cross the intersection in lane 1. In lane 0 this is the case, therefore this vehicle is allowed to cross. Then, when there is a vehicle able cross the intersection in lane 1, the server switches and lane 1 can cross the intersection. Finally, observe the difference in the pattern of arrivals in lanes 0 and 1. Where vehicles in lane 1 arrive in platoons due to those vehicles crossing the previous intersection together, the arrivals in lane 0 are structured more randomly due to the Renewal arrival process.

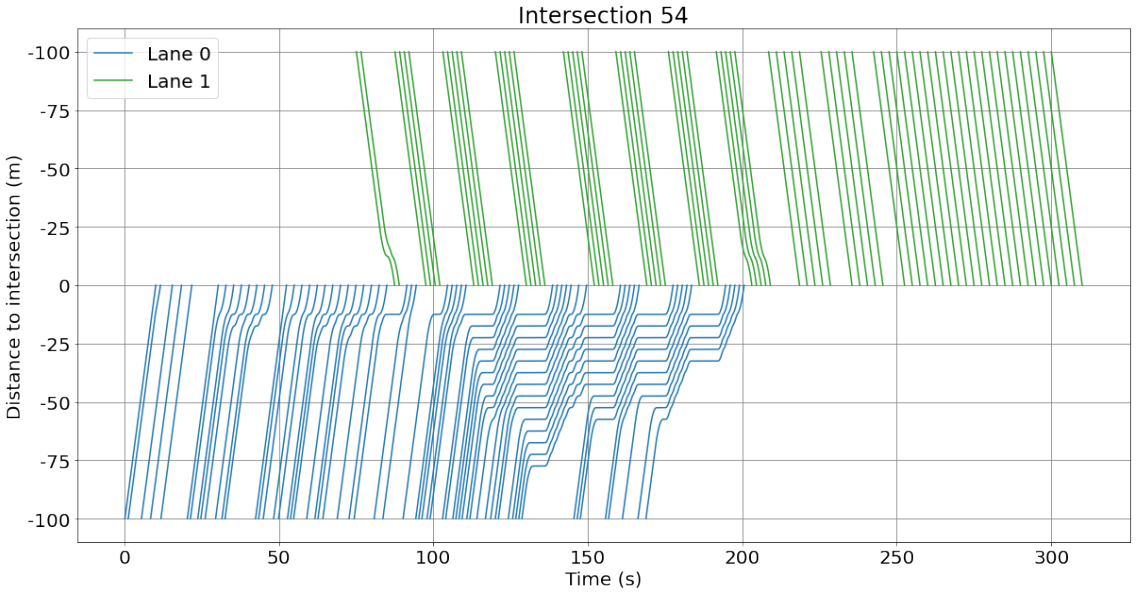


Figure 9: Trajectories of vehicles crossing intersection 54 in a 9×9 network.

Let us then investigate the trajectories of vehicles in the middle of the network, when traversing intersection 40, as illustrated in Figure 10. Here, vehicles generally arrive in both lanes. Notice however that relatively more vehicles cross the intersection in lane 0 than in lane 1. This is a result of the vehicles in intersection 36 having a lower average sampled inter-arrival time than the average sampled inter-arrival time of vehicles in intersection 4, being the two intersections receiving external arrivals that will eventually reach intersection 40. This difference in inter-arrival times is visible in Figure 10; within the time frame of $50 - 100$ seconds, only 8 vehicles arrive in lane 1 whereas 17 arrive in lane 0. Additionally, the vehicles in lane 0 almost always arrive in a platoon of size 5, whereas in lane 1, it occurs more often that vehicles arrive in smaller platoons.

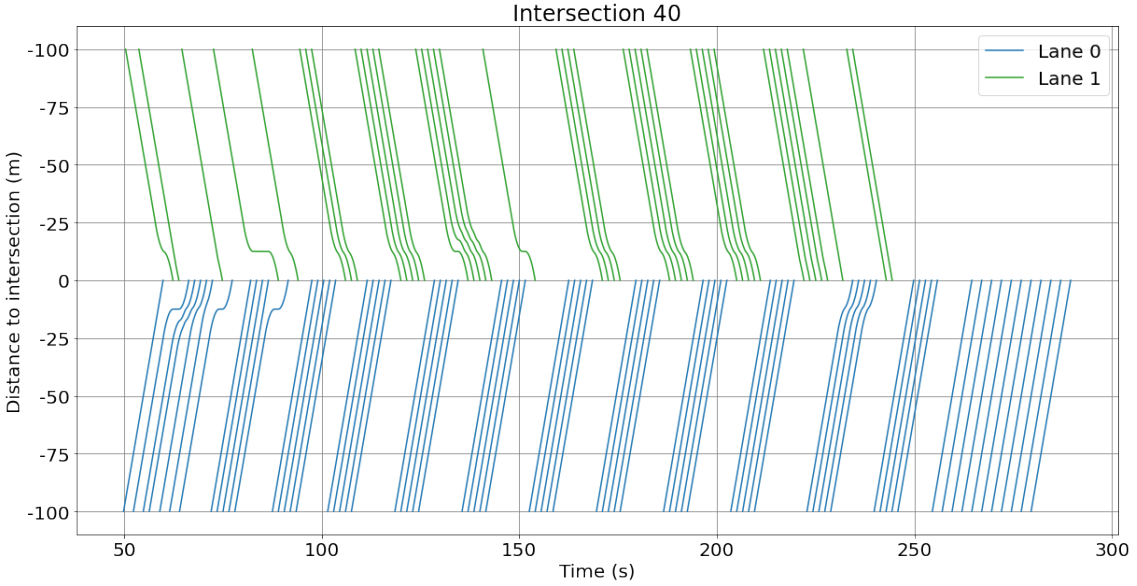


Figure 10: Trajectories of vehicles crossing intersection 40 in a 9×9 network.

Finally, the trajectories of vehicles crossing intersection 80 are considered. When vehicles arrive at intersection 80, they have already crossed 8 intersections. Therefore, it is expected that the vehicles arrive in platoons. Figure 11 shows that indeed, overall the vehicles arrive in platoons in both lanes. It is however not the case that the platoon a vehicle arrives in will remain the platoon with which a vehicle crosses intersection 80. Consider for example the time span $180 - 220$ seconds.

Here, first two vehicles cross the intersection in lane 1, after which there is no vehicle ready to cross the intersection. Then, a platoon of size 5 arrives in lane 1. During the time the first three vehicles of this platoon are crossing the intersection, there are vehicles in lane 0 ready to cross the intersection. Thus, after three vehicles of the platoon in lane 1 have crossed the intersection, 5 vehicles have crossed the intersection in lane 1 since they were allowed to cross. The k-limited discipline thus dictates that now lane 0 is allowed to cross the intersection. The two remaining vehicles in lane 1 that need to wait to cross the intersection then form a new platoon with newly arriving vehicles.

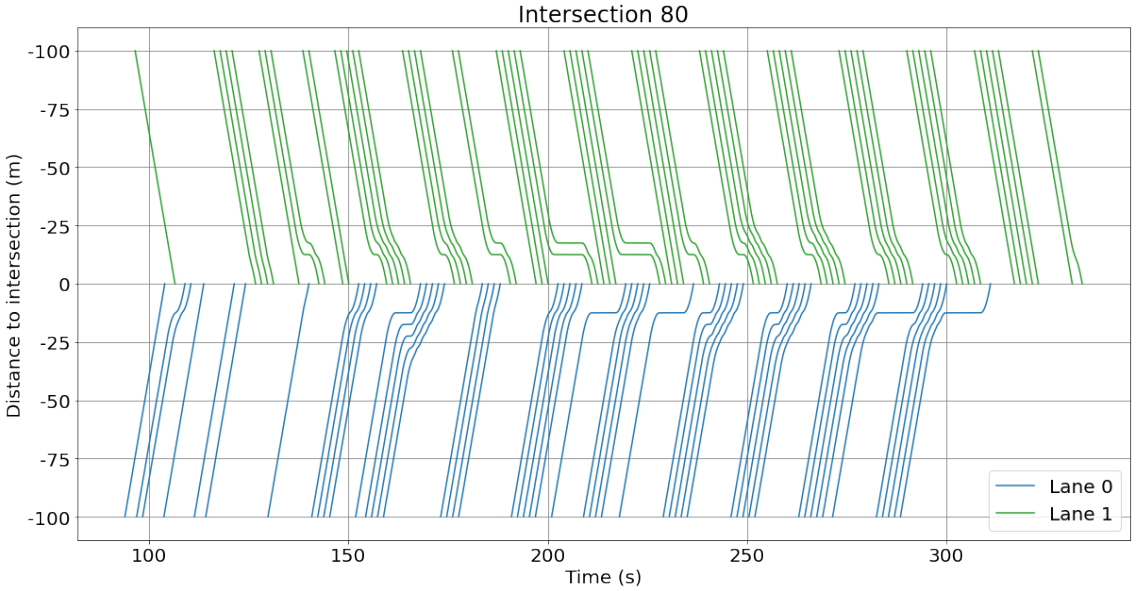


Figure 11: Trajectories of vehicles crossing intersection 80 in a 9×9 network.

5 Results

We return to the regular model of a network of intersections, as described in section 3.3. Thus, vehicles can arrive in all the external lanes, and can go left, right, straight and leave the network at every intersection. Using the simulation described in section 4, vehicles traversing the network under various circumstances can now be simulated, and the influences of the speed profiling and platoon forming algorithms on the delay and stability of the system can be investigated.

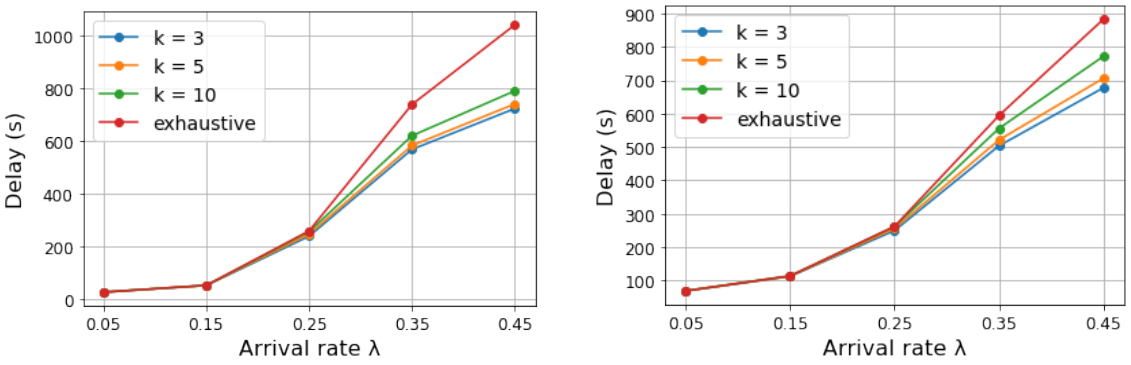
Due to the computational complexity of the simulation, all results that will be presented are derived over 1000 runs. Within each run, 1000 vehicles crossed a $m \times m$ network of intersections, where at each intersection, each vehicle had a probability of $\frac{1}{10}$ of leaving the network, and $\frac{3}{10}$ of going either left, straight or right. The value of the arrival rate λ, the parameter k and the size of the network are varied throughout the results, but the rest of the parameters are kept constant. When generating the results, a small error in the code resulted in the switch-over time not being taken into account. Therefore, the switch-over time S for these results is set to 0. Table 3 provides an overview of the values considered for these parameters.

L	l	w	v_{min}	v_{max}	a_{min}	a_{max}	s_{straight}	s_{turn}	$minArrDiff$	S
100	5	2	0	10	-4	4	1	1.5	1	0

Table 3: Values used for the parameters when simulating the network.

5.1 Platoon Forming Alorithms

We will start by analyzing the delays of the network when applying different PFA's. Recall that the two PFA's that will be considered are the exhaustive and k-limited policies. Figure 12 shows the average total delay that a vehicle experienced when traversing the entire network, for 4 different values of k. Notice that using the k-limited policy with the value $k = \infty$ is equal to applying the exhaustive policy.



(a) 3×3 network (b) 5×5 network

Figure 12: Average delay in the network when applying different platoon forming algorithms.

Notice that in both the 3×3 network as well as the 5×5 network, a higher value of k results in an equal or higher value of the average delay that a vehicle experiences. For an arrival rate $\lambda < 0.25$, the average experienced delay is approximately equal for all PFA's, but when λ exceeds 0.25, differences start to occur. It is to be expected that for a higher value of λ, a higher value of k results in higher delays. When λ becomes larger, this means that overall vehicles will arrive closer to each other. If the value of k is larger, one lane will be allowed to cross the intersection longer. Therefore, in a situation with a relatively higher λ and k, both more vehicles arrive at an intersection and a lane is allowed to cross the intersection during a larger time span. But then in the lanes that are not allowed to cross, multiple vehicles will have to wait longer before they can cross the intersection. This can result in higher average delays.

Additionally, observe that for each PFA, a higher value of λ results in a higher average experienced delay. This behavior is correct; as mentioned before, a higher value of λ results in more vehicles arriving during the same time span. As a result, more vehicles will arrive during the same time span at each intersection, therefore it will happen more often that a vehicle arriving at an intersection is not allowed to cross immediately, since it will have to wait for vehicles in other lanes of the intersection that are crossing the intersection. As a result, the average delay of a vehicle increases when the arrival rate λ increases.

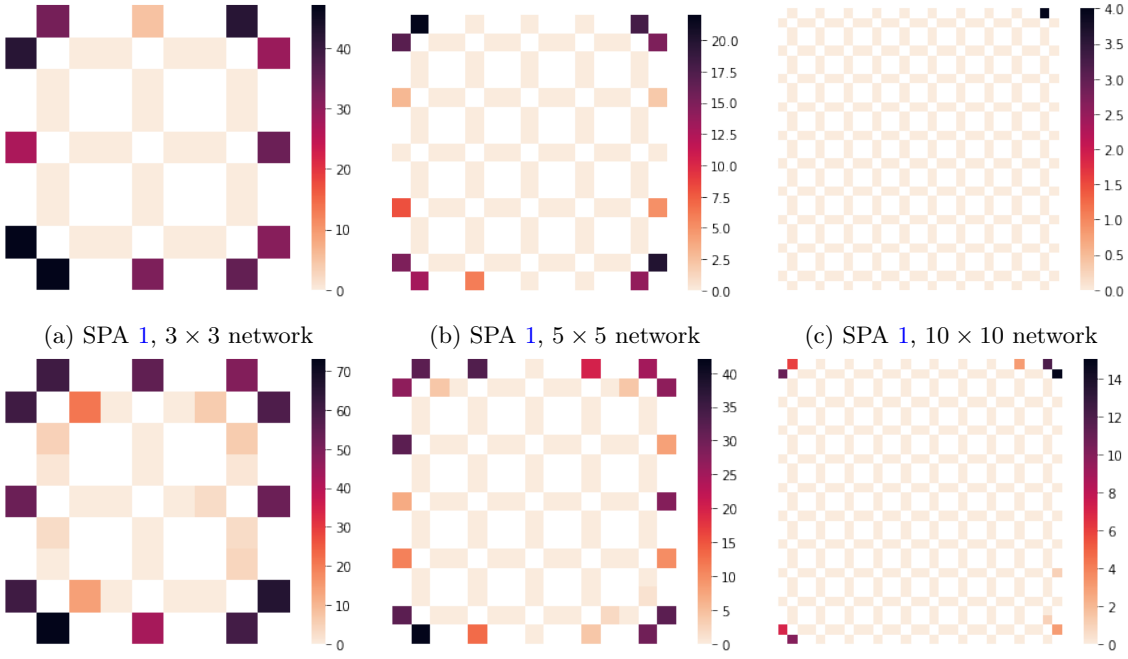
With a non-zero switch-over time S, one would have expected different results. For smaller values of λ, a higher value of k might result in a lower average delay. A reason for this is that for a higher value of k, the server generally switches lanes less frequently, therefore reducing the total switch-over time during which no vehicle can cross in any lane. When λ becomes higher however, this extra switch-over time might cancel out against the extra delays vehicles experience because they have to wait longer before crossing intersections.

Lastly, observe that a larger network overall seems to result in a lower average delay compared to a smaller network. This might be explained by the fact that each simulation ends when 1000 vehicles have entered the system. When the network is larger, these vehicles are spread out over more intersections, therefore on average less vehicles will arrive at each intersection. As a result, the experienced delays of vehicles can be lower.

5.2 Stability

Next, the stability of the system will be investigated. Our model already contains one feature that improves the stability of our system; each vehicle arriving at the control region of a lane arrives minimally *minArrDiff* later than the previous vehicle, meaning that at the moment it enters the control region it will maintain enough distance to a possible predecessor. However, this does not guarantee that the vehicle can always maintain enough distance. A vehicle arriving at the control region always does so at maximum velocity, which is 10 m/s, and has a minimum acceleration of -4 m/s^2. Thus, if a vehicle decelerates maximally starting from the moment it enters the control region, the vehicle will travel $5 \cdot 2.5 = 12.5$ m before coming to a full stop. In case another vehicle has stopped less than 12.5 meters from the end of the control region, the arriving vehicle will still bump into this vehicle even though it arrives later than the required *minArrDiff* seconds.

Figure 13 displays for three different sizes of the network and both SPA's, the total number of times that a vehicle was not able to safely cross each lane of the network. In all of these simulations, the arrival rate λ equals 0.7, and the k-limited policy is applied with $k = 5$. Notice that the total amount of times a vehicle was not able to safely cross a lane corresponds to all the times it occurred in all 1000 runs combined. The heatmaps represent the network; for each intersection, the 4 lanes where vehicles *arrive* are illustrated as a square. The color of the square then indicates the number of times a vehicle was not able to safely cross the intersection, which we will refer to as the total number of times the system became unstable.



(a) SPA 1, 3×3 network (b) SPA 1, 5×5 network (c) SPA 1, 10×10 network

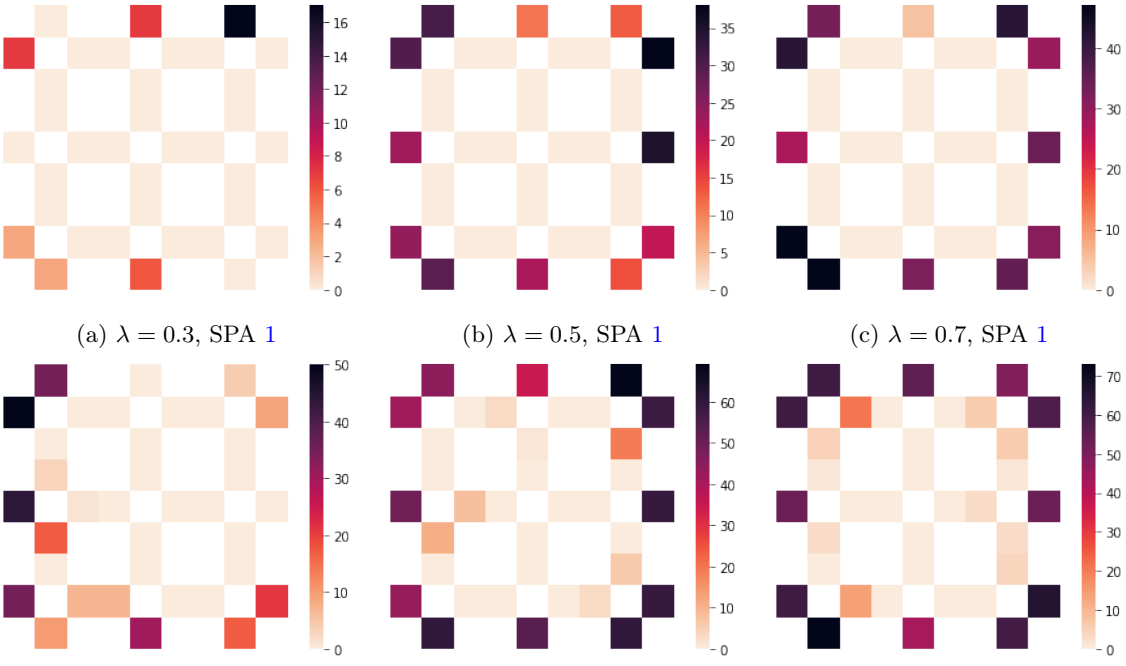
(d) SPA 2, 3×3 network (e) SPA 2, 5×5 network (f) SPA 2, 10×10 network

Figure 13: Total number of times a vehicle was not able to safely cross a lane.

First of all, observe that within all these scenarios, the system only becomes unstable at an intersection where external vehicles arrive (outer intersections). Because of the k-limited policy, within the inner intersections (without external arrivals), vehicles will mostly arrive in platoons of size 5. These vehicles cannot arrive constantly, since they need to cross an intersection first where they need to wait when other lanes are crossing the intersection. In the outer intersections however, there is either one or two lanes where vehicles can arrive constantly. As a result, overall more vehicles will arrive at an outer intersection compared to an inner intersection in a certain time span, resulting in vehicles stopping further from the intersection. Therefore, vehicles will more often not be able to safely enter the control region of an outer intersection, which results in more unstable instances in these intersections. Additionally, observe that within the outer intersections, the sys-

tem becomes unstable relatively more often in lanes actually receiving external arrivals compared to lanes receiving internal arrivals. This is a result of the vehicles not being able to constantly arrive at a lane receiving internal arrivals.

Furthermore, observe that a larger network results in fewer instances where the system becomes unstable. This is again explained by the fact that the simulation is ended when 1000 vehicles have entered the system, thus a larger network results in fewer vehicles arriving at each intersection. Lastly, Figure 13 shows the results concerning the stability of the system where both SPA's 1 and 2 are applied. Observe that for each network, Algorithm 2 results in more times that the system becomes unstable. This is a result of the different trajectories that vehicles follow with both algorithms. When applying Algorithm 2, vehicles aim to minimize their acceleration. As shown in sections 3 and 4, this results in vehicles immediately slowing down slightly when entering the control region. Algorithm 1 on the other hand aims to minimize the distance to the intersection. As a result, vehicles will slow down later in the control region, meaning that more vehicles will be able to safely enter the control region when applying Algorithm 1. As a result, the system will become unstable more often when applying Algorithm 2 compared to the case where Algorithm 1 is applied.



(a) $\lambda = 0.3$, SPA 1 (b) $\lambda = 0.5$, SPA 1 (c) $\lambda = 0.7$, SPA 1

(d) $\lambda = 0.3$, SPA 2 (e) $\lambda = 0.5$, SPA 2 (f) $\lambda = 0.7$, SPA 2

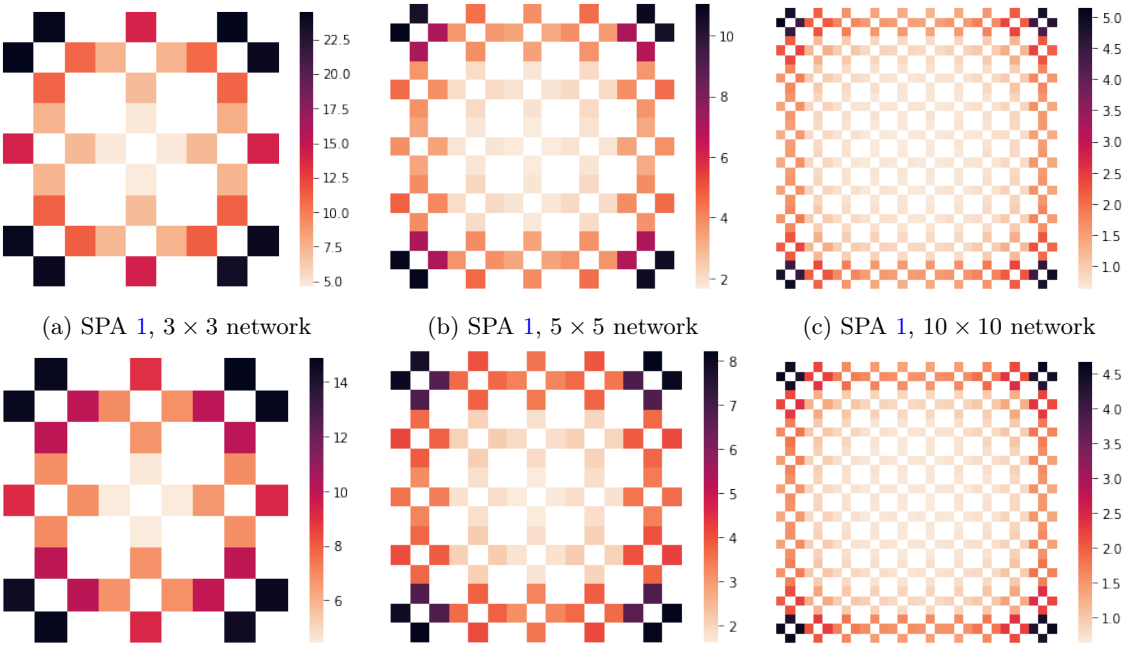
Figure 14: Total number of times a vehicle was not able to safely cross a lane.

Figure 14 then illustrates the stability of the system in a 3×3 network when applying both SPA's, where the arrival rate λ is varied. Again, one can observe that for each fixed arrival rate λ, the system becomes unstable more often when vehicles minimize their acceleration instead of their distance to the intersection. Also, for each arrival rate, a similar pattern arises as before, where the system only becomes unstable in outer intersections. This confirms that our results are not dependent on the arrival rate λ. Additionally, observe that the number of times the system becomes unstable increases as the arrival rate λ increases.

5.3 Delays throughout the network

Lastly, we will investigate the delays that vehicles experience throughout the network. Again, heatmaps are created that represent the intersection; for each lane in which vehicles arrive, a square is drawn whose color indicates the duration of the delay that a vehicle experienced on average in that lane.

Figure 15 illustrates the delays throughout a network with arrival rate $\lambda = 0.3$ for three different sizes, where both SPA's are applied. Notice first of all that within all three sizes of the network, the same pattern of the delays arises. The corners of the networks, being intersections where two lanes receive external arrivals, are the lanes where vehicles on average experience the highest delays. Then, the intersections where exactly one lane receives external arrivals on average have the highest delays. Vehicles experience an even lower delay in inner intersections, where the more intersections that separate a particular intersection from an outer intersection, the lower the delay that a vehicle on average experiences in that intersection. This is especially visible in Figures 15c and 15f, where the delays in a 10×10 network are displayed. As explained before, more vehicles arrive at the outer intersections in a given time span. Part of these vehicles then progress to inner intersections, where these vehicles will arrive in platoons and with larger inter-arrival times due to having to wait for other lanes that are crossing the intersection. The lanes where vehicles can arrive from an outer intersection however have slightly more arrivals in the same time span, since more vehicles have the opportunity to arrive in said lanes in that time span. As a result, the intersections that are connected to an outer intersection will be slightly more crowded than intersections that are positioned even more central in the network, therefore resulting in slightly larger delays in these less central intersections. This effect continues when going more and more central in the network, but it does fade out. Observe in Figures 15c and 15f that vehicles in the $4^{\text{th}}, 5^{\text{th}}$ and 6^{th} row and column on average experience an equal delay, illustrating that the influence of external arrivals on the delay experienced in the intersection has faded out. Additionally, observe again that as expected, a larger network results in lower average delays.



(a) SPA 1, 3×3 network (b) SPA 1, 5×5 network (c) SPA 1, 10×10 network

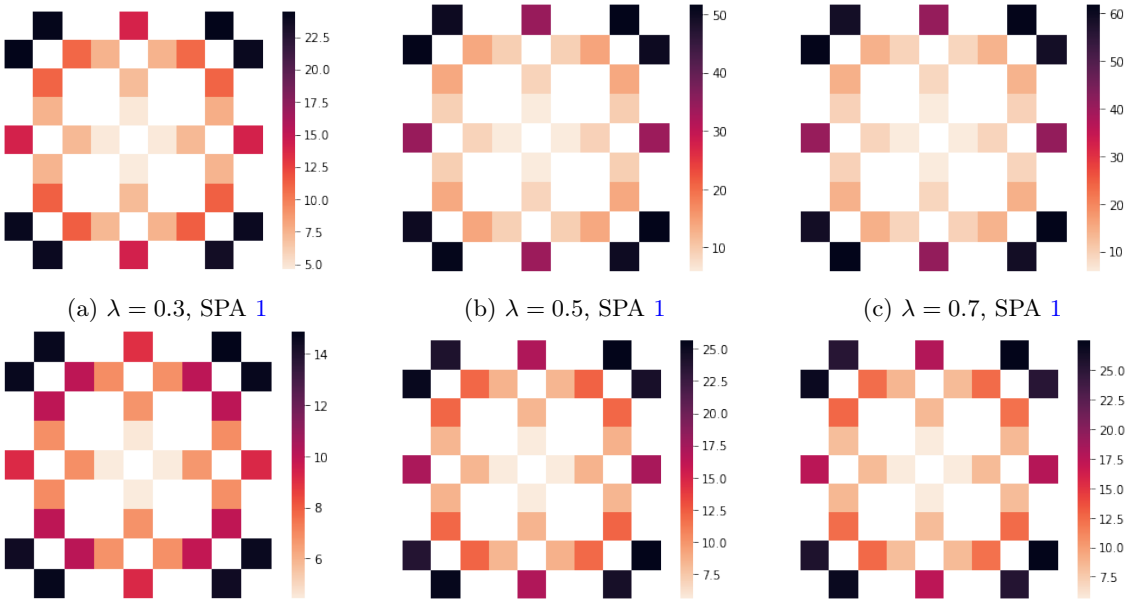
(d) SPA 2, 3×3 network (e) SPA 2, 5×5 network (f) SPA 2, 10×10 network

Figure 15: Average total delay in all lanes of a network with arrival rate $\lambda = 0.3$, given in seconds.

Figure 16 then illustrates the delays that vehicles experience in a 3×3 network with various arrival rates λ. Notice that the higher the arrival rate λ, the higher the average delay vehicles experience. This is to be expected, since a higher arrival rate means that more vehicles will arrive at the intersections in the same time span. Observe however that the difference in delays that vehicles experience when λ is increased from 0.3 to 0.5 is much larger than when λ is increased from 0.5 to 0.7. This is a result of the fact that the higher λ, the more often the system becomes unstable. When a vehicle is not able to safely cross an intersection, it is removed from the simulation and thus it does not experience a delay. Thus, λ can become as high as one would want, but only a certain maximum amount of vehicles will arrive at the network of intersections, and the other ones will simply be removed from the simulation. As a result, the delay that vehicles experience in a network is bounded from above. Notice that this upper bound is dependent on the applied SPA, since the SPA determines the maximum number of vehicles that can be present in the control region of a lane at the same time.

Furthermore, observe that for each arrival rate λ, the delays experienced by vehicles throughout the network follow the earlier described pattern. This implies that vehicles experiencing more delays in intersections closer to the edges of the network is not dependent on the arrival rate λ.

Additionally, observe that for all sizes of the network and a fixed arrival rate λ, vehicles on average experience less delay when SPA 2 is applied compared to when SPA 1 is applied. This is a result of the stability of the system; as shown before, for each arrival rate λ more vehicles are not able to safely cross a lane when minimizing their acceleration instead of their distance to the intersection. Therefore, when applying SPA 2, more vehicles will be removed from the simulation which will not contribute to the delay. Since the SPA does not influence the scheduled arrival and crossing times of vehicles, not counting certain delays when applying SPA 2 results in a lower average delay of vehicles traversing the network compared to the network in which vehicles apply SPA 1. Notice that vehicles are only removed from the simulation when it is relatively crowded at the intersection, meaning that the removed vehicles will have a relatively high delay. This strengthens the reasoning that the stability of the system influences the average total delay experienced in a certain lane. Notice that, if the system would be stable, meaning that no vehicle is removed from the simulation, one would expect that the average experienced delay of vehicles is equal for the two different SPA's. This is shown in Figures 15 and 16; within the central intersections, vehicles experience an approximately equal average total delay.



(a) $\lambda = 0.3$, SPA 1 (b) $\lambda = 0.5$, SPA 1 (c) $\lambda = 0.7$, SPA 1

(d) $\lambda = 0.3$, SPA 2 (e) $\lambda = 0.5$, SPA 2 (f) $\lambda = 0.7$, SPA 2

Figure 16: Average total delay in all lanes of a 3×3 network for different arrival rates λ, given in seconds.

6 Conclusion

Two different platoon forming and speed profiling algorithms were considered that determined the behaviour with which autonomous vehicles crossed a network of intersections. The optimal trajectories of the vehicles were visualized, the average total delays that vehicles experienced throughout the network was investigated and the stability of the system was analyzed.

First of all, it was shown that for a lower arrival rate λ, vehicles experienced approximately the same average delay when the crossing times of all vehicles in the network were either scheduled according to the exhaustive or k-limited policy. However, when the arrival rate increased, it was shown that the k-limited algorithm resulted in a lower average delay compared to the exhaustive algorithm. In particular, we found that for higher arrival rates, a lower value of k resulted in a lower average delay.

Furthermore, it was shown that the system is less stable in intersections where vehicles arrived from outside the network. Within these outer intersections, occasionally vehicles were not able to safely drive across the control region of a lane because there were too many vehicles already present. This occurred most in the corners of the network, which contain two lanes receiving external arrivals. Additionally, it was shown that a higher arrival rate λ resulted in a more unstable system. Also, when vehicles applied SPA 2, therefore minimizing their absolute acceleration, this resulted in a more unstable system compared to when SPA 1 was applied, where the vehicles aimed to minimize the distance to the intersection.

Last but not least, the delays that vehicles experienced throughout the network was analyzed. We found that vehicles on average experienced the most delay in intersections positioned at the edges of the network, in particular in the corners. Then, the further an intersection was positioned from the edge of the network, the smaller the delay a vehicle on average experienced in said intersection. Furthermore, it was shown that overall a larger arrival rate λ resulted in vehicles experiencing higher delays throughout the network.

These results are valid for a network in which there is no switch-over time accompanied with switching the lane that is allowed to cross the intersection. Future research could assume a non-zero switch-over time, and investigate whether the same results apply. Additionally, in this model only one lane can cross the intersection at the same time. There do however exist combinations of vehicles in different lanes that can cross the intersection simultaneously, based on the direction of these vehicles. This could be combined with also exploring different platoon forming algorithms, such as switching to the lane containing the most vehicles in the queue instead of simply switching to the next lane in clockwise direction that has a non-empty queue. Both these factors can result in the intersection being used even more efficiently, therefore resulting in even lower delays and a more stable system.

Furthermore, this simulation had a relatively long running time. Future research could improve the simulation to reduce its complexity, which would allow a simulation with more runs, therefore improving the reliability of the results. When the computational complexity is reduced, one could also simulate the network where vehicles do not arrive with the same arrival rate λ in each external lane. Also, different probabilities with which the vehicles go left, right, straight or leave the network at each intersection could be considered. The simulation is written such that these factors can already be included, but the long running time limits the opportunity to properly investigate their influences. Therefore, future research could include varying the arrival rate throughout the network and altering the probabilities with which vehicles move in a certain direction. By stopping the simulation after a certain time threshold has passed instead of a certain number of vehicles has arrived at the network, future research could also investigate whether the size of the network influences the delays and stability of the system. This could also be extended such that delays in networks that are shaped differently than our $m \times m$ network can be analyzed.

A last aspect that future research could consider, is changing the structure of our simulation. As of now, first the arrival and crossing times of all the vehicles that will cross the network are scheduled. After these times have been generated, the trajectories of the vehicles are computed accordingly. This structure cannot be implemented in real life, since the controller will then need to

compute the optimal trajectory of a vehicle while it is driving there. This will require an algorithm that constantly updates the trajectories of the vehicles every time their scheduled crossing time is altered.

References

[1] Sergey Chuprov et al. "Optimization of autonomous vehicles movement in urban intersection management system". In: *Conference of Open Innovation Association, FRUCT*. Vol. 2019-April. 24. Moscow, Russia, 2019, pp. 60–66. ISBN: 9789526865386. DOI: 10.23919/FRUCT.2019.8711967. URL: https://ieeexplore.ieee.org/abstract/document/8711967.

[2] Mohammad Pasha et al. "at Intersection Using V2I Communications for Road Safety". In: *Lecture Notes in Networks and Systems*. Vol. 103. Springer Singapore, 2020, pp. 23–31. ISBN: 9789811520433. DOI: 10.1007/978-981-15-2043-3. URL: http://dx.doi.org/10.1007/978-981-15-2043-3_4.

[3] David Miculescu and Sertac Karaman. "Polling-systems-based autonomous vehicle coordination in traffic intersections with no traffic signals". In: *IEEE Transactions on Automatic Control* 65.2 (2020), pp. 680–694. ISSN: 15582523. DOI: 10.1109/TAC.2019.2921659.

[4] *What Are the Levels of Automated Driving?* 2020. URL: https://www.aptiv.com/en/insights/article/what-are-the-levels-of-automated-driving.

[5] R. W. Timmerman and M. A.A. Boon. "Platoon forming algorithms for intelligent street intersections". In: *Transportmetrica A: Transport Science* 17.3 (2021), pp. 278–307. ISSN: 23249943. DOI: 10.1080/23249935.2019.1692962.

[6] Xi Lin et al. "Rhythmic Control of Automated Traffic—Part II: Grid Network Rhythm and Online Routing". In: *Transportation Science* 55.5 (2021), pp. 988–1009. ISSN: 0041-1655. DOI: 10.1287/trsc.2021.1061.

[7] Hideaki Takagi. "Analysis and application of polling models". In: *Performance Evaluation: Origins and Directions*. Vol. 1769. Springer, Berlin, Heidelberg, 2000, pp. 423–442. ISBN: 9783540671930. DOI: 10.1007/3-540-46506-5{_}18. URL: https://link.springer.com/chapter/10.1007/3-540-46506-5_18.

[8] Ivo Adan and Jacques Resing. *Queueing Systems*. Eindhoven University of Technology, 2015, pp. 26–27. DOI: 10.1016/0005-1098(66)90015-X.

[9] A. A. Boon Marko. "Polling models : from theory to traffic intersections". PhD thesis. Technische Universiteit Eindhoven, 2011. ISBN: 9038624492;9789038624495; DOI: 10.6100/IR702638. URL: https://research.tue.nl/en/publications/polling-models-from-theory-to-traffic-intersections%20http://tue.summon.serialssolutions.com/2.0.0/link/0/eLvHCXMwdV1dS8MwFL1s7EX0weG3TvIHNtYmaZuBiPtCVFCYVHwqzZKCIO1Yuwf_vfemc8zBIC8hIcmFJDfnJucEgPu9fndnT1CZsNIzSnJtNXo.

[10] Serguei Foss and Günter Last. "Stability of polling systems with exhaustive service policies and state-dependent routing". In: *The Annals of Applied Probability* 6.1 (1996), pp. 116–137. ISSN: 10505164. DOI: 10.1214/aoap/1034968068. URL: https://www.jstor.org/stable/2245319?seq=1#metadata_info_tab_contents.

[11] Marko Boon and Erik Winands. "Critically loaded k-limited polling systems". In: *EAI EndorsedTransactions on Collaborative Computing* 16.10 (2016). DOI: 10.4108/eai.14-12-2015.2262578. URL: https://doi.org/10.4108/eai.14-12-2015.2262578.

7 Appendix

Algorithm 5 Discretized MotionSynthesize Simulation

Input: $x_0, v_0, t_0^{(i)}, t_f^{(i)}, t_0^{(k)}, t_f^{(k)}, y$
Output: Trajectory of vehicle V_i

Initialize the number of discretization points N
Create $times$, a list containing all the times at which the position of vehicle V_i will be evaluated
Interpolate y and create pos_prev, a list containing the position of V_k at all times in $times$
Initialize m as a linear optimization model
Add continuous model variable x of dimension $3 \cdot N$ ▷ x = [position, velocity, acceleration]

if MotionSynthesize procedure minimizing distance to intersection **then**
    Set the objective to maximize the sum of $x[:N]$
else if MotionSynthesize procodure minimizing acceleration **then**
    Set the objective to minimize the dot product $< x[2N:], x[2N:] >$
end if

Make matrix A, vector b such that $Ax = b$ ensures ▷ Recursively define position and velocity
- $x[0] = x_0$ and $x[N] = v_0$
- $\forall i : N \leq i \leq 2N - 1 : x[i+1] = x[i] + x[N+i] \cdot dt$
- $\forall i : 0 \leq i \leq N - 1 : x[i+1] = x[i] + \frac{x[N+i]+x[N+i+1]}{2} \cdot dt$

$lb = [-L] \cdot (N-1) + [0] + [v_0] + [0] \cdot (N-2) + [v_{\max}] + [a_{\min}] \cdot N$

Create vector $prev$ of length $N-2$ by $prev[i] = \begin{cases} pos\_prev[i+1] - l & \text{if } times[i+1] < t_f^{(k)} \\ \infty & \text{otherwise} \end{cases}$

$ub = [-L] + prev + [0] + [v_0] + [v_{\max}] \cdot (N-1) + [a_{\max}] \cdot N$

Add constraints $Ax = b$, $x \geq lb$ and $x \leq ub$ to the model
Optimize the model
return the optimal position, velocity and acceleration at all times $t \in times$ of vehicle V_i

Algorithm 6 Routing matrix

Input: Number of intersections m^2
Output: Matrix D describing the flow through the network

Initialize matrix D = [[[[−1, −1], [−1, −1], [−1, −1]] for i in range(4)] for j in range(m^2)]
for Intersection number $I_0 \in \{0, 1, ..., m^2 - 1\}$ **do**
    for Queue number $q_0 \in \{0, 1, 2, 3\}$ **do**
        if $q_0 == 0$ **then**
            if $I_0 \geq m$ **then** ▷ The intersection is not in the first row
                $D[I_0][q_0][0] = [I_0 - m, 3]$ ▷ Set destination when going left
            end if
            if $I_0 \mod m \neq m - 1$ **then** ▷ The intersection is not in the last column
                $D[I_0][q_0][1] = [I_0 + 1, 0]$ ▷ Set destination when going straight
            end if
            if $I_0 < m^2 - m$ **then** ▷ The intersection is not in the last row
                $D[I_0][q_0][2] = [I_0 + m, 2]$ ▷ Set destination when going right
            end if
        end if

        if $q_0 == 1$ **then**
            if $I_0 \mod m \neq m - 1$ **then**
                $D[I_0][q_0][0] = [I_0 + 1, 0]$
            end if
            if $I_0 < m^2 - m$ **then**
                $D[I_0][q_0][1] = [I_0 + m, 1]$
            end if
            if $I_0 \mod m \neq 0$ **then** ▷ The intersection is not in the first column
                $D[I_0][q_0][2] = [I_0 - 1, 2]$
            end if
        end if

        if $q_0 == 2$ **then**
            if $I_0 < m^2 - m$ **then**
                $D[I_0][q_0][0] = [I_0 + m, 1]$
            end if
            if $I_0 \mod m \neq 0$ **then**
                $D[I_0][q_0][1] = [I_0 - 1, 2]$
            end if
            if $I_0 \geq m$ **then**
                $D[I_0][q_0][2] = [I_0 - m, 3]$
            end if
        end if

        if $q_0 == 3$ **then**
            if $I_0 \mod m \neq 0$ **then**
                $D[I_0][q_0][0] = [I_0 - 1, 2]$
            end if
            if $I_0 \geq m$ **then**
                $D[I_0][q_0][1] = [I_0 - m, 3]$
            end if
            if $I_0 \mod m \neq m - 1$ **then**
                $D[I_0][q_0][2] = [I_0 + 1, 0]$
            end if
        end if
    end for
end for
return matrix D