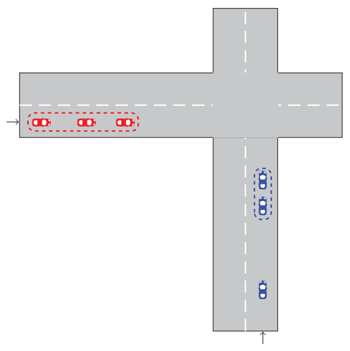
## Platoon forming algorithms for autonomous vehicles

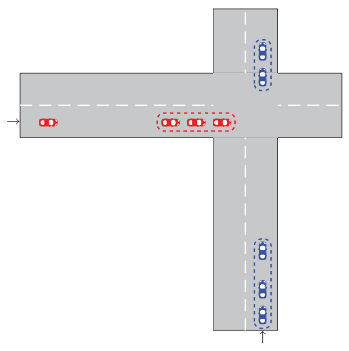
*This assignment is based on the paper by Timmerman and Boon (2021). It is recommended to read this paper (superficially) to get a better insight in the details of the model.*

In this assignment, we consider a futuristic setting in which all vehicles are fully autonomous, meaning that they are not operated by a human driver. These self-driving vehicles rely on communication among themselves *and* with an external central controller. One of the key features is that this central controller has full control over the vehicles, in particular the speed at which each vehicle drives. With traffic signals becoming obsolete and human behaviour no longer playing a significant role, it seems obvious that this new technology will open up entirely new ways to organise the process of giving vehicles access to traffic intersections. An example of such a novel method that has received much attention in the recent transportation literature, is a technique called *Platoon forming*. In recent papers papers by Miculescu and Karaman (2020); Tachet et al. (2016); Timmerman and Boon (2021), it is shown that the overall delay of vehicles can be minimised by letting the vehicles form so-called platoons, i.e. groups of vehicles in the same lane, driving at the shortest allowed distance from each other. The reason why this is so efficient, is easy to understand: *inside* a platoon, vehicles are allowed to drive at $B = 1$ second from each other. This time between cars is called the *headway*, which is measured from the front of one car to the front of the next car. For safety reasons, between platoons from *different* lanes, there needs to be at least $S = 2.4$ seconds. In more detail: the time between the last vehicle of platoon 1 and the first vehicle of platoon 2 (from another lane) crossing the intersection should be at least $S$. A graphical illustration of the model, for an intersection with two lanes, is given in Figure 1. Figures 1(a) and 1(b) show how vehicles inside the dashed rounded rectangles form platoons by appropriately slowing down and accelerating again. Figure 1(c) shows exactly how the speed of the individual vehicles is manipulated by the central controller to ensure they start crossing the intersection *at full speed* and *at the exact right time*.

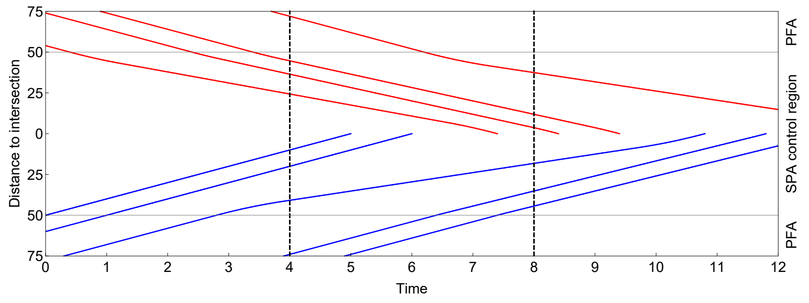
The paper by Timmerman and Boon (2021) consists of two parts. In the first part, so-called *Platoon Forming Algorithms* (PFAs) are developed to determine the crossing times of the vehicles, given their arrival times to the control region. The second part discusses several algorithms to compute the trajectory of each vehicle, taking into account that they need to cross the intersection at full speed. These algorithms are called Speed Profiling Algorithms (SPAs). As indicated earlier, the central controller has full control over the speed (and, as a consequence, over the full trajectories) of the autonomous vehicles. We assume that the central controller can look “ahead” and has full knowledge of the future arrival times of all vehicles at the start of the *control region*. We focus on PFAs first, in Section 1. Subsequently, we discuss the SPAs in Section 2.



(a) Intersection at time $t = 4$.



(b) Intersection at time $t = 8$.



(c) Trajectories for the red and blue traffic flows.

Figure 1: A schematic representation of the model discussed in Timmerman and Boon (2021). The platoon forming algorithms in this paper determine how the platoons are constructed. Fig (a) and (b) correspond to the situation in (c) at times $t = 4$ and $t = 8$ seconds, respectively. Note that all vehicles cross the intersection at maximum speed.

## 1 PFAs: Determining the crossing times

An important observation in Timmerman and Boon (2021) is that the whole system can be modelled as a queueing system. The vehicles represent the customers in this queueing system and $N$ queues represent the $N$ lanes. For simplicity, we assume that there is only one server, serving only one queue at a time. This means that no two lanes are allowed to cross the intersection simultaneously, even if they are non-conflicting (for example North-South and South-North). A graphical representation of this queueing system, is shown in Figure 2. This type of queueing system, with one server and multiple queues, is called a *polling model*.

Since the central controller has foll control over the autonomous vehicles, it determines their crossing times at the intersection. The reason why we can use a polling model to compute the optimal crossing times, becomes apparent when studying Figure 3. In this figure, the trajectories of two vehicles are plotted in blue. We assume that both vehicles are delayed because they have to wait for a platoon in one of the other lanes (not visualised). Without delay, the first vehicle would have started crossing the intersection at time 25 (following the black dashed line). Due to a 10 second delay, however, the vehicle can only cross the intersection at time $t = 35$. The second vehicle arrives 5 seconds later to the control region and would have started crossing the intersection at time $t = 30$. Since the first vehicle is delayed, this second vehicle forms a platoon with vehicle 1 and starts crossing the intersection $B = 1$ second later, at time $t = 36$. The key

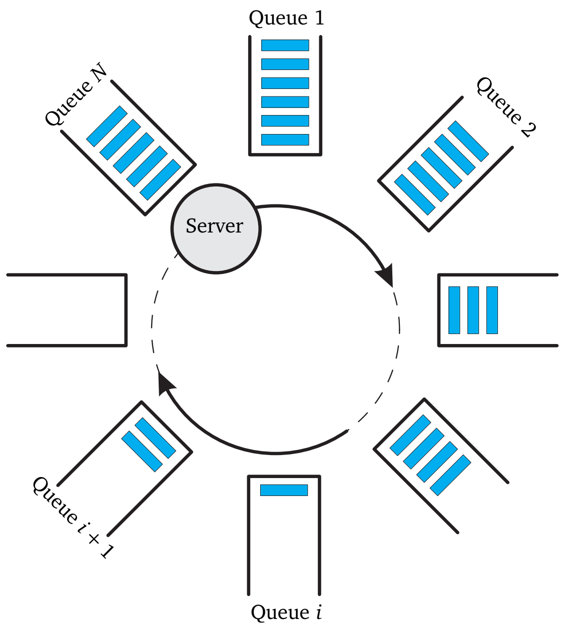


Figure 2: A polling model.

observation is that the trajectories are irrelevant to determine the optimal crossing times. Since all vehicles cross the intersection at the same speed (meaning they have the same *service time* in the queueing model), we only need to know when they would have started crossing the intersection *without any delay*. Those times will be the arrival times in our queueing model. The actual crossing times will be considered as the *start* of the service time of that particular customer. The service time ends $B$ time units later. In Table 1 we have given an example of 10 vehicles and their arrival times and crossing times. Note that the arrival times are those in the queueing system. The actual arrival time at the start of the control region would have been $L/v_m$ time units earlier, where $L$ is the length of the control region and $v_m$ is the maximum speed of the vehicles.

Table 1: An example of the arrival times and service beginnings (in the queueing model) of 10 arriving vehicles. The table is sorted by the arrival times of the vehicles.

| Lane | Arrival time | Start service Global FCFS | Start service Exhaustive |
|---|---|---|---|
| 0 | 1.000 | 1.000 | 1.000 |
| 0 | 2.000 | 2.000 | 2.000 |
| 1 | 2.309 | 4.400 | 4.400 |
| 1 | 3.309 | 5.400 | 5.400 |
| 0 | 4.816 | 7.800 | 10.800 |
| 1 | 5.169 | 10.200 | 6.400 |
| 1 | 6.985 | 11.200 | 7.400 |
| 1 | 8.051 | 12.200 | 8.400 |
| 0 | 9.158 | 14.600 | 11.800 |
| 1 | 9.996 | 17.000 | 18.200 |

We shall now describe the queueing model in more detail.

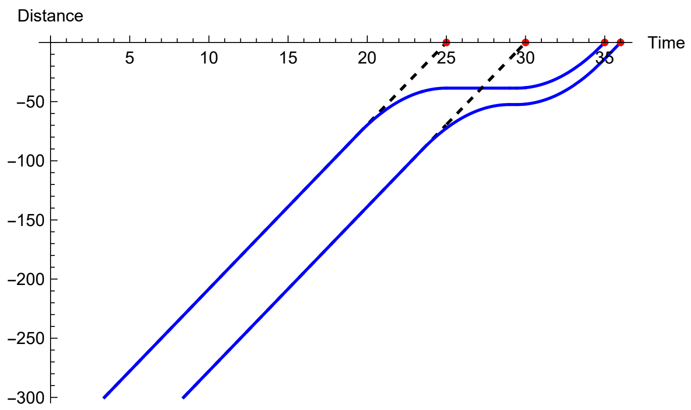


Figure 3: Trajectories of two vehicles forming a platoon (in blue). If there had been no delay, the vehicles would have followed the black dashed lines. Now they experience a delay of respectively $35 - 25 = 10$ and $36 - 30 = 6$ seconds. In the corresponding polling model, the arrival times of these two customers are $t_1 = 25$ and $t_2 = 30$. At time $t = 35$ the service of the first customer in this lane starts. One time unit later, this customer leaves the system and the next customer starts their service, which ends at time $t = 37$. Note that the distance is a negative number to be consistent with the definition in Section 2.

**Arrival process.** We have $N$ queues, each with its own arrival process. The interarrival times of customers in queue $i$ are i.i.d. random variables with a bunched exponential distribution (Cowan, 1975):

$$\mathbb{P}\left(A_{i,j} \leq x\right) = \begin{cases} 1 - \alpha e^{-\mu(x-B)} & \text{for } x \geq B, \\ 0 & \text{for } x < B, \end{cases}$$

This guarantees that vehicles do not arrive closer to each other than $B$ seconds, which is the minimum allowed headway.

**Service process.** The service times are all equal to $B$ seconds (with $B = 1$). If a customer from queue $i$ starts their service at time $T$, then the service of the next customer cannot start before time $T + S$ if this next customer is in any other queue than $i$. The time $S - B$ can be considered as a *clearance time*, required to clear the intersection for safety reasons. We assume that $S - B > 0$, otherwise the system would be *work conserving* and the order in which vehicles are served would have no effect on the overall mean waiting time (cf. Boxma and Groenendijk (1987)). In queueing theory, the clearance times are called switch-over times.

**Switching process.** We will consider two rules that the server may use to decide when to start serving another queue:

- **Global First-come-first-served (FCFS).** This policy considers all customers in the system, regardless of the queue they arrived in, and serves them in order of their arrival.
- **Cyclic Exhaustive service.** This policy dictates that the server keeps on serving customers in queue $i$ until that queue is completely empty. Then it switches to

queue $i + 1$ (or back to queue 1 if $i = N$). If a queue is empty, it will simply be skipped.

Timmerman and Boon (2021) show that cyclic exhaustive service leads to smaller delays, due to the fact that the server switches less frequently to another queue, wasting less time on clearing the intersection.

One of your tasks in this assignment is to write a (stochastic) discrete-event simulation for this queueing model. From this simulation you can obtain the customer arrival times and the departure times. Additionally, when performing multiple (long) runs, the simulation should give estimates and confidence intervals for the mean delay, for each of the lanes.

## 2 SPAs: Determining and visualising the trajectories

The second part of the paper by Timmerman and Boon (2021) elaborates on how the vehicles should approach the intersection, i.e., speed profile algorithms determining the optimal trajectories. In this assignment we focus on the algorithm that minimises the distance from vehicle to intersection. With this policy, an optimal trajectory always has one of the two shapes shown in Figure 4. Without loss of generality, we take $t = 0$ as the moment the vehicle enters the control region and the start of the trajectory. Then the vehicles continues driving at full speed $v_m$ (m/s) until time $t = t_{dec}$, when it starts to decelerate at rate $-a_m$ (m/s$^2$). In some cases, it will come to a full stop, at time $t = t_{stop}$. It starts accelerating at rate $a_m$ (m/s$^2$) until it reaches full speed $v_m$ again at time $t = t_{full}$. It continues driving at full speed until it starts crossing the intersection at time $t = t_f$. The trajectory $x(t)$ is given by Equation (1) in case the vehicle comes to a full stop, and by (2) in case the vehicle does not come to a full stop. We consider the trajectory of a vehicle that experiences no delay (i.e. a straight line) as a special case of (2) with $t_{dec} = t_{acc} = t_{\text{full}}$. We assume that $0 \leq t_{dec} \leq t_{stop} \leq t_{acc} \leq t_{\text{full}} \leq t_f$.

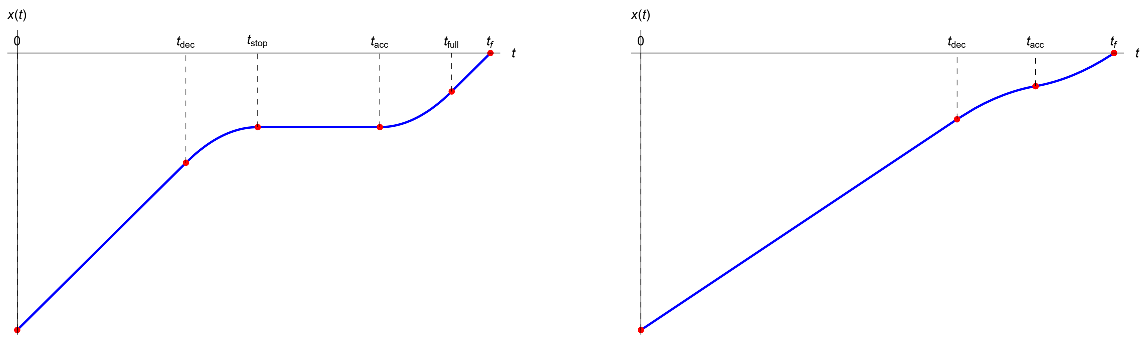


Figure 4: The two possible shapes of a trajectory: (a) with a full stop and (b) without full stop. Note that $t_{full} = t_f$ in (b).

$$x(t) = \begin{cases} (t - t_f)v_m & \text{for } t_{full} \leq t \leq t_f, \\ (t_{full} - t_f)v_m - \frac{v_m^2}{2a_m} + \frac{a_m}{2}(t - t_{acc})^2 & \text{for } t_{acc} \leq t \leq t_{full}, \\ (t_{full} - t_f)v_m - \frac{v_m^2}{2a_m} & \text{for } t_{stop} \leq t \leq t_{acc}, \\ (t_{full} - t_f)v_m - \frac{v_m^2}{2a_m} - \frac{a_m}{2}(t - t_{stop})^2 & \text{for } t_{dec} \leq t \leq t_{stop}, \\ x_0 + v_m t & \text{for } 0 \leq t \leq t_{dec}, \end{cases} \tag{1}$$

where $x_0$ is a negative number indicating (minus) the length of the control region.

Equation (1) is easiest to understand when starting at $t = t_f$ and constructing the trajectory backwards to $t = 0$, and using these auxiliary results:

$$t_{stop} - t_{dec} = t_{full} - t_{acc} = \frac{v_m}{a_m}, x(t_{stop}) - x(t_{dec}) = x(t_{full}) - x(t_{acc}) = \frac{v_m^2}{2a_m}.$$

Note that $t_{dec}$ follows from continuity of $x(t)$:

$$t_{dec} = \frac{|x_0|}{v_m} - (t_f - t_{full}) - \frac{v_m}{a_m}.$$

The case with no full stop is given by:

$$x(t) = \begin{cases} (t - t_f)v_m & \text{for } t_{full} \leq t \leq t_f, \\ (t - t_f)v_m + \frac{a_m}{2}(t - t_{full})^2 & \text{for } t_{acc} \leq t \leq t_{full}, \\ x_0 + v_m t - \frac{a_m}{2}(t - t_{dec})^2 & \text{for } t_{dec} \leq t \leq t_{acc}, \\ x_0 + v_m t & \text{for } 0 \leq t \leq t_{dec}. \end{cases} \tag{2}$$

We can eliminate the unknowns $t_{dec}$ and $t_{acc}$ by using the relations

$$t_{full} - t_{acc} = t_{acc} - t_{dec} = \sqrt{\frac{t_f v_m - |x_0|}{a_m}}.$$

Four important remarks about these trajectories:

1. Note that these trajectories contain at most *one* deceleration/acceleration phase. This means that a vehicle can come to at most one stop. This is perfect for exhaustive service (discussed earlier), because exhaustive service guarantees that vehicles will always be served in the cycle they arrive in, or in the next cycle. With global FCFS, however, the vehicles might have to decelerate and accelerate multiple times before crossing the intersection, meaning that trajectories with at most one deceleration/acceleration phase do not minimise the distance to the intersection.

2. The trajectories $x(t)$ given by (1) and (2) assume that the vehicle arrives at the control region at time $t = 0$. For arrivals at time $t = t_A$, take the translated trajectory $x(t - t_A)$.

3. There is one unknown left in (1) and (2): $x_{full}$, which represents the end of the acceleration period (i.e. the time at which the vehicle drives at full speed again). For a given vehicle, this value is determined by the trajectory of its predecessor (on

the same lane). For exhaustive service, it is quite easy to determine the value of $t_{full}$, see also Algorithm 4 in Timmerman and Boon (2021): for the first vehicle in a platoon, we have that $t_{full} = t_f$. All other vehicles in the same platoon have the exact same value for $t_{full}$!

4. Algorithm 4 in Timmerman and Boon (2021) contains explicit expressions for the unknowns $t_{acc}$, $t_{stop}$, and $t_{dec}$. You can use those directly, instead of computing them manually.

In this assignment, you will also write a program that visualises the trajectories for a two-lane intersection in the same way as they are drawn in Figure 5. Use the output of a single run of the simulation written in the previous part of this assignment as input for your program. From the above remarks it becomes apparent that exhaustive service simplifies these trajectories considerably. For this reason, we only ask you to compute and plot trajectories for the exhaustive service discipline.

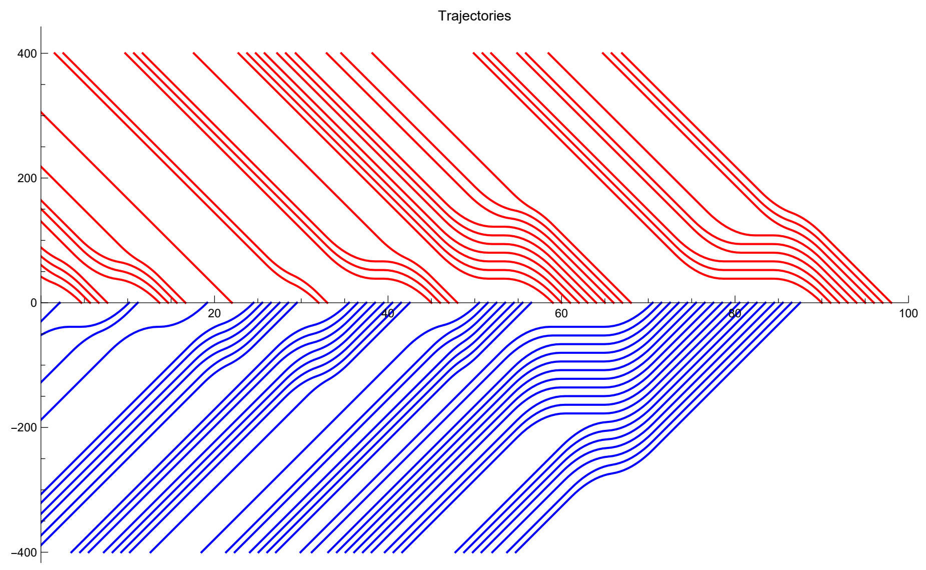


Figure 5: Trajectories.

## 3 Input data

Each group should study a two-lane intersection. The length of the control region is equal to 300 metres. The maximum speed of vehicles is $v_m = 13$ m/s. The maximum acceleration/deceleration rate is $a_m = 3$ m/s$^2$.

Each group receives a data set in Canvas that contains the arrival times of vehicles. Use this data set to estimate the parameters $\alpha_i$ and $\mu_i$ of the arrival process of lane $i$ ($i \in \{1, 2\}$). Verify that $B = 1$.
**Hint:** first try to interpret the parameter $\alpha_i$. That will make it easier to estimate it from the provided data. Subsequently, use the method of moments to also estimate $\mu_i$.

## 4 Deliverables

Programming:

1. Write a program that imports the data set and estimates the parameters for the arrival process in each lane.

2. Write a discrete-event simulation *with event scheduling*, using the techniques you learned in the lectures, for the queueing model described in Section 1. Implement both the Global FCFS and the Exhaustive service rules. You are allowed to write a separate program for each service rule, if you prefer. Your program(s) should produce the following results:

   - Waiting times for each lane: mean, standard deviation, histogram.
   - Queue lengths for each lane: mean, standard deviation, histogram.
   - 95% confidence intervals for the mean waiting times.
   - For exhaustive service only: export the arrival times (to the control region) and the crossing times (i.e. the *start* of the service time) of all the vehicles.

   You are allowed to write this simulation specifically for a two-queue model, but you can earn bonus points if you write it for a general number of queues ($N$).

   Note that this discrete-event simulation should be able to simulate the arrival process with the estimated parameters, but it should also be able to import the arrival times provided in the data set and run the platoon forming algorithm on this data set by converting the arrivals to the control region to arrival times for the queueing model.

3. Write a program that computes the trajectories according to (1) and (2) (for exhaustive service). This program should also visualise these trajectories by producing a plot similar to Figure 5. Produce this plot for the first 100 vehicles from your dataset *and* for 100 randomly generated vehicles in your discrete-event simulation.

Write a report, preferably in LaTeX, and hand in a PDF detailing your implementation and findings. The report must contain at least the following:

- A brief motivation of the problem and a short description of the model.
- A detailed description of your implementation of the discrete-event simulation(s). Use pseudo-code to most effectively communicate your implementation within the report. You do *not* need to discuss the implementation of the code that computes the trajectories.
- Well-designed and clear plots and tables. The more scientifically interesting your plots are, the better!
- A brief discussion and conclusion based on your plots.
- Exact copy of your code in the Appendix. Code in the Appendix does not count towards a page limit.

- Also in the appendix, for the first 100 vehicles of the data set provided to you: include a table with the arrival times to the queueing model, the crossing times when using exhaustive service, and a visualisation of the trajectories.

## 5 Programming and collaboration tips

In this course, you'll have to learn to work on the assignments with three people working in parallel. Of course, it is a good idea not to start coding right away. In the first phase of the project, you should discuss the problem together. Is the problem statement clear to everybody? Is it clear how to write code to answer the questions posed above? Note that multiple parts can be programmed separately:

- the discrete-event simulation, which could create an output file containing the arrival and departure times of all vehicles.
- a second part that should be able to import this file, compute the trajectories, and visualise them.
- possibly a third program that performs tasks like estimating the parameters of the arrival process, sampling interarrival times, computing confidence intervals, . . .

Before you start programming, discuss the **object oriented** setup of the program. Which classes do you need? In more detail: how do you represent the vehicles, the queues, the server, the trajectories? Please remember that you are not allowed to use libraries from Python that are not discussed in this course, unless you ask (and get) special permission. See the Canvas discussion forum for the most up-to-date list of allowed libraries.

Of course, it is always good to discuss (and check) each other's code. All of the students should understand each other's implementations and should understand the principles of stochastic simulation that are used in this assignment. In your report, include a simulation description where you detail the *relevant* classes, attributes and methods to simulate this system. We recommend using a Git repository for working on a coding project in a group. For obvious reasons, we ask you to keep this a private repository!

## 6 Grading and knock-out criteria

This assignment will be 25% of your final grade. The assignment is made in groups of three students. Each group should hand in a well-written report and the source code of their simulation programs in Canvas. The **page limit** of the report is **10 pages** excluding references and appendix. Ensure that **each** class is saved in its own **class file** and is named appropriately so that we immediately recognize the structure of your simulation program. Make sure to check Canvas for more information like deadlines or the rubric.

Please upload your report as a PDF file and upload the source code *in a separate zip file*.

## References

O. J. Boxma and W. P. Groenendijk. Pseudo-conservation laws in cyclic-service systems. *Journal of Applied Probability*, 24(4):949–964, 1987.

R. J. Cowan. Useful headway models. *Transportation Research*, 9(6):371–375, 1975.

D. Miculescu and S. Karaman. Polling-Systems-Based Autonomous Vehicle Coordination in Traffic Intersections With No Traffic Signals. *IEEE Transactions on Automatic Control*, 65(2):680–694, 2 2020. ISSN 0018-9286. doi: 10.1109/TAC.2019.2921659. URL https://ieeexplore.ieee.org/document/8732975/.

R. Tachet, P. Santi, S. Sobolevsky, L. I. Reyes-Castro, E. Frazzoli, D. Helbing, and C. Ratti. Revisiting Street Intersections Using Slot-Based Systems. *PLOS ONE*, 11 (3):e0149607, 3 2016. ISSN 1932-6203. doi: 10.1371/journal.pone.0149607. URL https://dx.plos.org/10.1371/journal.pone.0149607.

R. W. Timmerman and M. A. A. Boon. Platoon forming algorithms for intelligent street intersections. *Transportmetrica A: Transport Science*, 17(3):278–307, 2021. doi: 10.1080/23249935.2019.1692962. URL https://doi.org/10.1080/23249935.2019.1692962.