Delft University of Technology
Faculty Electrical Engineering, Mathematics and Computer Science
Delft Institute of Applied Mathematics

Using an Agent-Based Formalism to Simulate Passenger Flows (Dutch title: Simulatie van passagiersstromen met een individu-gebaseerd model)

A thesis submitted to the
Delft Institute of Applied Mathematics
in partial fullment of the requirements

for the degree

BACHELOR OF SCIENCE
in
APPLIED MATHEMATICS

by

Tobias Christiaan Molenaar

Delft, the Netherlands
20 December 2017

BSc thesis APPLIED MATHEMATICS

"Using an Agent-Based Formalism to Simulate Passenger Flows (Dutch title: Simulatie van passagiersstromen met een individu-gebaseerd model)"

TOBIAS CHRISTIAAN MOLENAAR

Delft University of Technology

Responsible Professor

Dr.ir. F.J. Vermolen

Other members of the thesis committee

Drs. A.T. Hensbergen

Dr. J.G. Spandaw

20 December, 2017

Delft

Abstract

In this report two Agent-Based Models, the Passenger Model and the Transfer Model, will be constructed and analyzed. The objective is to make a realistic model inspired by a cell-based model to simulate passenger flows on a train platform. We will start by studying the cell-based model and we will explain which aspects have been applied to our model. Then the Passenger Model, which can also be applicated to different environments, will be introduced. After that the Transfer Model, which is an even more realistic model for a train platform, is constructed. The models are used to analyze different types of platforms to investigate what an optimal platform would look like.

These models can help visualizing passenger flows and can be used to analyze the performance of existing platforms and platforms that have yet to be designed.

Contents

Chapter 1

Introduction

Rush hour in London, one of the biggest metropolises in the world, means that Tubes and trains are almost always packed. Therefore Underground and railway stations are overcrowded, which results in long queues for escalators and exits. Some entrances are closed dozens of times a year, simply because there are too many people on the Tube stations [20]. After a while the amount of people in the station is reduced and the entrances can be reopened, such that people can carry on with their journey with a lot of delay. Figure 1.1 confirms these findings and gives us a cause of this problem, it shows that the number of passengers during the morning peak increases over the years (although there is a small fall around 2009 and 2010).

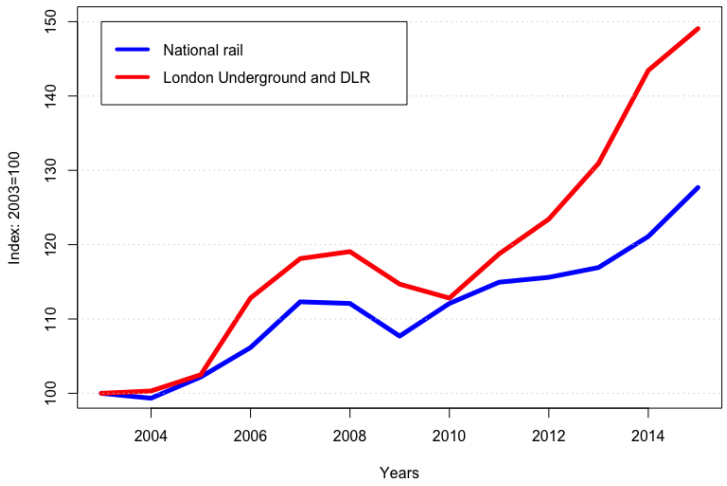


Figure 1.1: Passengers entering central London during the morning peak between 2003 and 2015 according to Transport for Londen (TfL) [28]. It can be seen that the amount of passengers is increasing. From 2003 to 2015, the number of passengers using the London Underground and DLR has increased by 50%.

London is not the only city struggling with this problem, lots of big cities all over the world are having issues with overcrowdness, especially during rush hour [5]. If we do not react, the problem will become bigger and stations have to be closed more times. By 2050, 66 per cent of the world's population is expected to live in urban areas [29]. A short term solution to this problem is to reduce the number of passengers during rush hours such that the number of passengers is more evenly distributed throughout the day, but eventually this will not entirely solve the problem. Another solution would be expanding the stations to make more space to move, so more passengers can use the station at the same time, yet this is rather expensive and

difficult to accomplish in big cities. A long term solution and a less expensive solution would be designing a platform such that the passengers can migrate over the platform in an optimal way to cause a minimal amount of delay. There are a lot of other locations where such crowds are, take for example an airport during peak season, major sports events or music festivals. For safety reasons there must be an evacuation plan for all these places, to get everyone as fast as possible out of a building or away from a specific place. To make this possible there needs to be no unnecessary obstacles causing delay in these types of situations. Recently there was panic in the Oxford Circus subway station (London) which resulted into many people being injured [13]. The Deputy Mayor for Transport, Val Shawcross, said: [14] "Clearly there is an issue about getting people out of stations very quickly, and that is something I think is worthy of more examination." This type of crowd panic happens often and sometimes even with fatal outcomes like in Turin this year [12] or seven years ago during the Love Parade in Duisburg [33].

This sums up that there are serious problems making a platform or other type of environment safe in these type of situations. Therefore we will study the evacuation of train platforms as well as the migration of people over these platforms.

1.1 Several Models

There exist several methods to model passenger flows. For example using partial differential equations or Agent-Based Modelling. Recently, the Delft University of Technology has come up with another model called NOMAD [2] which is based on the microscopic behaviour of humans. This indicates that there is still some interest in this kind of model. For this study we will model the passenger flow using an Agent-Based Model. An Agent-Based Model is often used for biological purposes, to simulate the migration, deformation, division and death of cells are a few examples. This way of modelling makes it possible to describe and simulate the complex behaviour of cells. Because of these models it is possible to look on a micro level at cells and on a macro level at an entire system, such as a tumor. These models are not used for cells only, they are applied on other levels as well, like on DNA-, tissue- and cell colony level. Cell-based modelling is another way to describe these kinds of individuals. You can roughly divide cell-based models into two categories: with or without lattice. Some models with a lattice make use of a grid to state the location of cells such as cellular automata (CA) models, here cell positions are allowed to take a predefined set of gridpoints only [3]. Models without a grid make it possible to model continuous migration of cells, using equations of motion depending on time, which in our case is more realistic.

1.2 Goal of this Study

The main goal of this study is to investigate whether Agent-Based Modelling inspired by cell-based modelling can be used to model passenger flows. Realistic assumptions must be made to construct such a model. Moreover, the results will be evaluated to see whether they are realistic. Another aspect is to design the model in a way that it is applicable to other environments. Hence a general model is aimed for. Besides this, we will look at the optimization of platforms. A good platform would require both comfort for waiting passengers like benches, and a good passenger flow such that the passenger migration time interval is as short as possible. For safety reasons we also aim for a platform with a small evacuation time interval.

1.3 Thesis Structure

First of all the cell-based model from biology is explained in detail and we will explain which aspects are used to make a realistic model and which aspects are abandoned here. Subsequently we explain in depth how our model, the Passenger Model, is implemented and how we model human behaviour as well as possible. In Chapter 4 we will analyze the data obtained from the Passenger Model. Some initial values will be altered to see whether the model is realistic and how these values will influence the travel and evacuation time. A few different platforms will be simulated and analyzed here to finally conclude what a good platform requires. Chapter 5 contains an even more realistic model, the Transfer Model, to simulate a platform with arriving and departing trains. This model will be simulated to analyze platforms on their performances regarding passenger flows. Finally we will combine our main results in a conclusion and we will try to answer our main questions.

Chapter 2

Cell-Based Model

Since our model for passenger flows has been constructed based on a cell-based model, we first need to understand how a cell-based model works. To describe the behaviour of cells we need to know what influences the movement of cells. Before we begin to explain this model, there are a few assumptions that are made. We assume that the cells are on a two dimensional substrate and that the projections of the cells are perfect circles with a fixed radius R. This radius can differ for different cells. There are some aspects for the cell-based model that we do not use, for the sake of simplicity or to make the model more realistic for passengers on a platform. More information regarding the migrational behaviour of cells can be found in [4][30][31].

Migration of cells depends on several factors, such as the presence of other cells and their movements, mechanotaxis[1]. Other factors are for example chemotaxis[2] and random walk. All these factors will give a component to the velocity. If we combine all the factors we find the total velocity. We will describe the components that are responsible for the velocity in the next section of this chapter and we will explain which aspects will be used in our model and why we incorporate them.

2.1 Mechanotaxis

Imagine n cells with respectively a radius of $R_1, R_2, \ldots, R_n$, all strictly positive, in a two dimensional substrate $\Omega \subset \mathbb{R}^2$. Let N be the set of cells, then we have $N = \{1, 2, \ldots, n\}$. A cell will move in a certain direction that is determined by a traction force exerted by another migrating cell. The traction force causes deformations in this substrate and therefore a strain that other cells can sense. Both cells exert this force that causes this strain and therefore they will start migrating towards one another. We can calculate the migration of cells from the equations for this strain. We denote the strain energy density from cell with index $i \in N$ by M_i^0. Let $\boldsymbol{r} \in \Omega$ be the coordinates of a position on the substrate and $\boldsymbol{r}_i$ the location of cell i on the substrate. The strain energy density caused by cell i at location $\boldsymbol{r}$ is equal to:

$$M_i(\boldsymbol{r}) = M_i^0 \exp\left(-\lambda_i \frac{|\boldsymbol{r} - \boldsymbol{r}_i|}{R_i}\right). \tag{2.1}$$

[1]The movement of a cell along a rigidity gradient.
[2]Oriented movement toward or away from a chemical stimulus.

Here λ_i is a damping factor caused by the characteristics of the substrate with index i and the substrate. Using Equation (2.1) we can determine the strain energy density at $\boldsymbol{r}_i, \forall i \in N$:

$$M(\boldsymbol{r}_i) = \sum_{j \in N} M_j(\boldsymbol{r}_i) = \sum_{j \in N} M_j^0 \exp\left(-\lambda_j \frac{|\boldsymbol{r}_i - \boldsymbol{r}_j|}{R_j}\right). \tag{2.2}$$

Equation (2.2) gives us the strain energy density at every position on the substrate. Hence we know the strain energy density that each cell detects. To determine the direction of migration for cell i we need the unit vector from cell i to another cell j, $\frac{\boldsymbol{r}_j - \boldsymbol{r}_i}{|\boldsymbol{r}_j - \boldsymbol{r}_i|}$, and the strain cell i detects from cell j: $M_j(r_i)$. The direction for cell i depends on all cells, hence we get:

$$\boldsymbol{z}_i = \sum_{j \in N} M_j(\boldsymbol{r}_i) \frac{\boldsymbol{r}_j - \boldsymbol{r}_i}{|\boldsymbol{r}_j - \boldsymbol{r}_i|}.$$

Using the numerical Forward Euler [32] method, we can determine the new location for every cell by using the following formula:

$$\boldsymbol{r}_i(t + \Delta t) = \boldsymbol{r}_i(t) + \Delta t \alpha_i M(\boldsymbol{r}_i) \hat{\boldsymbol{z}}_i.$$

Here $\hat{\boldsymbol{z}}_i = \frac{\boldsymbol{z}_i}{\|\boldsymbol{z}_i\|}$ and α_i depends on the viability of a cell and depends on the force that is exerted by cell i, as well as by the resulting friction between the cell and the substrate. Moreover α_i depends on the radius of cell i and the mobility of the surface. We will not use these parameters in the passenger flow model for sake of simplicity (so we set $\alpha_i = 1$).

However, using these equations to model a passenger flow on a platform will lead to strange outcomes. In reality, passengers will not walk towards each other since their goal is to reach the exit. Moreover passengers do not exert any force which can be felt by other passengers across the platform. However, we still want to use these equations to describe the migrational behaviour of the passengers, hence we will incorporate this into the model by treating exits to exert the traction force as described above to attract the passengers. All passengers on the platform will notice the exits, and based on the distance between the exit and the passenger we can determine their direction. Let U be the set containing all exits then there is always one exit $\hat{u} \in U$ which is the closest exit to a passenger. For now, the location of this exit is denoted by $\hat{\boldsymbol{u}}$, as if it is one point. This will be specified and explained in detail in Section 3.2.1. The equation for the new location for all passengers i based on the closest exit $\hat{u} \in U$ is the following:

$$\boldsymbol{r}_i(t + \Delta t) = \boldsymbol{r}_i(t) + \Delta t v_i M(\boldsymbol{r}_i) \hat{\boldsymbol{z}}_i, \tag{2.3}$$

$$M(\boldsymbol{r}_i) = A \exp\left(-B \frac{\|\boldsymbol{r}_i - \hat{\boldsymbol{u}}\|}{R_i}\right). \tag{2.4}$$

Here $\hat{\boldsymbol{z}}_i$ is the unit vector from $\boldsymbol{r}_i$ to $\hat{\boldsymbol{u}}$. v_i is the prefered walking speed of passenger i. Constants $A, B \in \mathbb{R}$ can be chosen such that there is a realistic function for the migration towards exits. To estimate these constants we need to compute $M(\boldsymbol{r}_i)$ for different values of A and B. First it is important to note that $M(\boldsymbol{r}_i)$ is a number that indicates how much movement there is caused by the closest exit. After combining all components we multiply it by the velocity of the passenger and then we can derive the new location of passenger i. If there is one passenger and one exit on a platform with no obstacles, then you would expect that this passenger can walk with his or her desired speed (say $v_i = 1.4$ [m/s]) regardless of his or her size. According to Equation (2.4) $M(\boldsymbol{r}_i)$ will increase in value when this passenger comes closer to the exit, with a maximum of A. This is assumed because a passenger will search for exits first which causes a relatively low walking speed and as the passenger comes closer to the exit the passenger will have a relatively

high walking speed. Since you would not walk (or run) twice as fast as your desired speed v when you are close to an exit, we want A to be slightly above 1. The further you are away from the exit the smaller $M(\boldsymbol{r}_i)$ becomes, with its limit to zero. So we need to take into account the size of platforms. Although there are platforms longer than one kilometre [16], we will model platforms with a size around 50 metres to speed up the computation time. So a long distance to the exit would be around 50 metres, if you are on the other end of the platform (with one exit). Then $M(\boldsymbol{r}_i)$ close to 1 is desired because you would not walk very slow to an exit if there are no other passengers around you. Figure 2.1 shows us values of $M(\boldsymbol{r}_i)$ for different values of A and B, here $R_i = 0.3$ [m] is fixed. $M(\boldsymbol{r}_i)$ is initially intended to be close to 1, even if a passenger

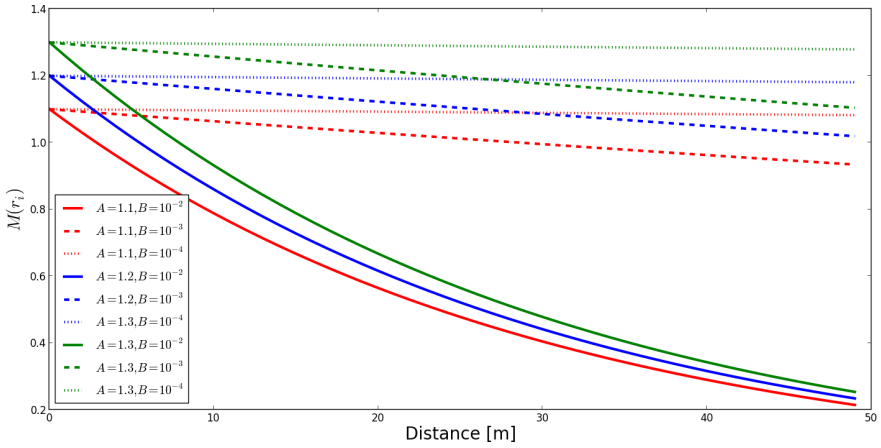


Figure 2.1: Value of $M(\boldsymbol{r}_i)$ with $R_i = 0.3$ [m] for different values of A and B. On the x-axis is the distance from the exit. You can see that the further away a passenger is from an exit, the smaller the value $M(\boldsymbol{r}_i)$ is.

is far away from the exit. The choice $B = 10^{-2}$ leads to rapidly decreasing values for $M(\boldsymbol{r}_i)$ if passenger i is far away from the exit. This is in contrast with the choice $B = 10^{-4}$, here $M(\boldsymbol{r}_i)$ is nearly constant for every distance to the exit. Therefore a value $B = 10^{-3}$ is chosen. For A however it is a bit arguable which value we need to take, a realistic model is desired and we do not think that people will walk (or run) a lot faster, nor will they walk (or run) much slower than their desired speed. Therefore the value $A = 1.1$ is chosen. For sake of completeness, here is the initially chosen equation for the new location of passenger i depending on the closest exit $\hat{j}$:

$$\boldsymbol{r}_i(t + \Delta t) = \boldsymbol{r}_i(t) + \Delta t v_i \hat{\boldsymbol{z}}_i \cdot 1.1 \exp\left(-10^{-3} \frac{\|\boldsymbol{r}_i - \hat{\boldsymbol{u}}\|}{R_i}\right).$$

2.2 Random Walk

The migration of a cell is not only influenced by other cells, it is also influenced by random factors. At each timestep the next location is determined by either the random factor or using the mechanotaxis described as above. This random component can be determined by using the normal distribution [26]. However in the model for passenger flows a different method has been chosen. Choosing either the random factor or using the mechanotaxis as before can be done when movements are very small and/or the timesteps are very small. Modelling lots of cells and, in our case, passengers leads to a long computation time. To reduce this computation time we want to take a relatively large timestep. Instead of taking only the random factor

or the mechanotaxis, we add the random facter every timestep in the calculation of the new location. In Equation (2.2) you can see the added component $\boldsymbol{W}(t)$, which is a Wiener Process [15]. Here $\boldsymbol{W}(t)$ contains two independent and identically distributed (iid) samples from the standard normal distribution for every time t.

$$\boldsymbol{r}_i(t+\Delta t) = \boldsymbol{r}_i(t) + \Delta t v_i \hat{\boldsymbol{z}}_i \cdot 1.1 \exp\left(-10^{-3}\frac{\|\boldsymbol{r}_i - \hat{\boldsymbol{u}}\|}{R_i}\right) + \boldsymbol{W}(t).$$

2.3 Colliding Cells

Cells will migrate towards each other due to the mechanotaxis, therefore they will eventually collide with each other. Colliding cells deliver less strain energy which leads to a lower total strain from cell i. A colliding cell on a two dimensional substrate has the following total strain energy density function:

$$\hat{M}_i(\boldsymbol{r}) = M_i(\boldsymbol{r}) - M^{ij}, \tag{2.5}$$

$$M^{ij} = \frac{16}{45}\frac{\sqrt{R_i}E_i h^{\frac{5}{2}}}{\sqrt{2}\pi R_i^2}. \tag{2.6}$$

For the derivation, we refer to [30]. In Equation (2.2) E_i denotes the elasticity modulus of cell i and h is the sum of the two radii of both cells minus the distance between the two cells and then divided by two. So if cell i and j with radius equal to R are colliding then h is equal to $h = \frac{2R - \|\boldsymbol{r}_i - \boldsymbol{r}_j\|}{2}$. However, passengers do not deliver any strain energy and therefore this formula is not used in the Passenger Model. Some rules must apply for colliding passengers, because two passengers cannot stand or walk on the same position. In Section 3.2.5 it is further explained how we prevent passengers from crashing into one another.

2.4 Chemotaxis

We make a small excursion to the biology of skin tissue. When there is a small wound on your skin, some cells move towards the wound area or are already there. This is no coincidence, since cells can react to chemical signals. An example is that bacteria will move towards high concentrations of food (glucose). This movement is called chemotaxis. It is also possible that cells move away, from poison for example. This is referred to as negative chemotaxis. A passenger will likely move to a space where there is a low concentration of passengers in order to have more room for himself. In real life no one likes crowded spaces and we are much more inclined to go to a space where there are fewer people. However, the main goal of the passenger remains to get to an exit. People react on passengers close to them, if someone from the right is getting close, it is likely that you will move slightly to the left to get some space. The area around a passenger such that he or she will react to someone else in this area is called the comfort zone. This also holds for walls and objects (e.g. benches or shops). More details about this type of movement can be found in Sections 3.2.3 and 3.2.4.

2.5 Life Cycle of a Cell

Cells are living organisms and therefore they will die eventually. A dead cell will not migrate on its own anymore, only exterior forces can effect the location of this cell. Looking at passengers on a platform, there is a chance that a passenger dies on a platform. However in the model for passengers we do not take this into account, because this rarely happens on platforms. Another

aspect we do not encounter in our model is cell division, a so called mother cell generates a daughter cell. Applying this on our model, it would be equivalent to a woman giving birth to a child, as this also rarely happens on a platform, we leave this out of our model.

2.6 Conclusion

Looking at all different components based on the behaviour for cells, we think that this model can be used to model passenger flows as well. Still some changes have to be made in order to make it more realistic. The largest difference between the cell-based model and the Passenger Model that we consider in this manuscript, is that there is no mechanotaxis between the passengers. Instead we incorporate mechanotaxis between passengers and exits. Some minor changes would be that we need to adjust the scale, if we model passengers as circles then the radius would be orders of magnitude larger than the radius of cells. Some changes are made for the sake of simplicity and because they are rare events on a platform, such as dying passengers and passengers giving birth. Another difference between the cell-based model and the model for passengers is the presence of benches, shops and other obstacles on the platform. Passengers normally do not walk into these things so we need to describe this behaviour as well.

Chapter 3

The Passenger Model

To look whether the cell-based model including the adjustments stated in Chapter 2 is working, we will construct a computer program that can be used to model the passenger flow on a train platform. A general program is needed so we can alter the program in terms of properties of the platform and properties of the passengers. The programming language of choice for this is Python [10][18][21] because of personal preference. The basic idea for the program is to determine the direction of every passenger and move them from their original place to their new place using these directions. Many other things happen during this process: new passengers arrive on the platform and sometimes passengers bump into each other. To make a realistic model, the Passenger Model needs to take into account that there are objects of different shapes and some natural human traits such as herd behaviour. We will call the adapted cell-based model as the Passenger Model and we will explain in detail how this model works.

3.1 Arrival of Passengers

When a train arrives at the train station there are multiple doorways from the train to the platform. We will call these entrances, because these are the entrances to the platform. For every entrance we need an algorithm to determine when and how much passengers appear on the platform. We will use a Poisson distribution [1][27] to model this process, because it is a discrete probability function and one can use this distribution to say how often an event occurs within a predefined time-interval. Besides this distribution we need to take the width of the entrance into account, for example, when an entrance has a width of one metre, only one passenger can go through the door when he/she has a width between 25 and 50 centimetres. The Poisson distribution needs a rate parameter $\lambda > 0$ to determine the probability of k occurrences within a time interval. This parameter λ can be increased if we want to model an entrance where a lot of passengers come from per second. The probability for $X = k$ is given in Equation (3.1) (see also A.4).

$$\mathbb{P}(X = k) = e^{-\lambda}\frac{\lambda^k}{k!}, \qquad \text{for } k = 0, 1, 2, \ldots. \tag{3.1}$$

Now that we know how many passengers arrive we only need to know when and where.

When we change the timestep of the calculations we need to make sure that passengers still arrive after each other within a few seconds, Thus we determine every second how many passengers alight from the train using the Poisson distribution. These passengers are put in front of the entrance on the platform and are ready to walk around. When there are multiple passengers at once we set them next to each other, just like in reality, see Figure 3.1.

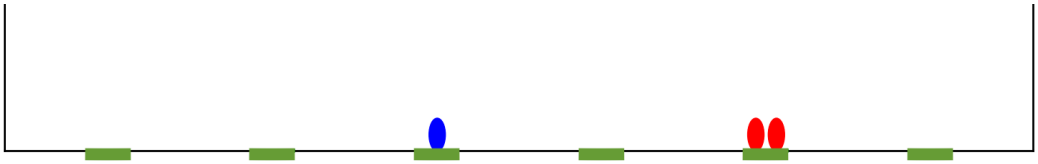


Figure 3.1: Example of alighting passengers, green lines: entrances, blue circles: passengers, red circles: group of passengers, black lines: walls.

3.2 The Migration

The most important part of the model is to determine the direction in which passengers will walk, this depends on many different factors. From the cell-based model we know that there is mechanotaxis for every cell. For passengers we can use this mechanotaxis to determine the movement towards the exits.

If there is more than one exit, we need to decide which exit a passenger prefers to go to. Often a passenger wants to go to the nearest exit to for example transfer to another train platform. However, sometimes the passenger has reached his final destination and therefore he or she wants to go to a specific exit on the platform. Because of this, a random factor needs to decide how many passengers on average go to the nearest exit. To find the closest exit we need to calculate the shortest distance to every exit for each passenger. We can do this by parametrising the exit, if exit u is a straight line with endpoints $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ we can parametrize exit u as follows:

$$\boldsymbol{u}(\tau) = \boldsymbol{u}_1 + \tau(\boldsymbol{u}_2 - \boldsymbol{u}_1), \tau \in [0,1].$$

Let $\boldsymbol{r}_i(t)$ be the location of passenger i on time t, now the distance between passenger i and all points for exit u is equal to:

$$\delta_i^u(\tau, t) = \|\boldsymbol{u}(\tau) - \boldsymbol{r}_i(t)\|.$$

We can square this function and then minimize this by taking the derivative with respect to τ and set it to zero. We can eliminate the optimal $\hat{\tau}_i^u$, we find:

$$\hat{\tau}_i^u(t) = \frac{< \boldsymbol{r}_i(t) - \boldsymbol{u}_1, \boldsymbol{u}_2 - \boldsymbol{u}_1 >}{\|\boldsymbol{u}_1 - \boldsymbol{u}_2\|^2}.$$

Here $< \boldsymbol{a}, \boldsymbol{b} >= \boldsymbol{a} \cdot \boldsymbol{b}$ is the dot product. We defined exit u by $\boldsymbol{u}(\tau)$ with $\tau \in [0,1]$, but $\hat{\tau}$ does not necessarily need to be in the interval $[0,1]$. Therefore we define $\bar{\tau}_i^u = \min(\max(\hat{\tau}_i^u, 0), 1)$. Now we find the minimal distance between passenger i and exit u to be equal to $\delta_i^u(\bar{\tau}_i^u(t), t)$. Likewise we find that the point of exit u closest to passenger i is $\boldsymbol{u}(\bar{\tau}_i^u(t))$.

3.2.1 Mechanotaxis

Now that we know which exit the passenger wants to go to, we can calculate the direction for each passenger to the exit. The direction from passenger i to the closest point of exit u is equal to $\boldsymbol{u}(\bar{\tau}_i^u(t)) - \boldsymbol{r}_i(t)$. We normalise this expression and multiply it by $M(\boldsymbol{r}_i)$, see Equation (2.4). We can now use the following formula for every passenger i to determine the direction of passenger i towards the closest exit $\hat{u}$:

$$\boldsymbol{d}_i(t) = A \exp\left(-B \frac{\|\boldsymbol{r}_i(t) - \hat{\boldsymbol{u}}(\bar{\tau}_i^{\hat{u}}(t))\|}{R_i}\right) \frac{\hat{\boldsymbol{u}}(\bar{\tau}_i^{\hat{u}}(t)) - \boldsymbol{r}_i(t)}{\delta_i^{\hat{u}}(\bar{\tau}_i^{\hat{u}}(t), t)}.$$

However only this formula is not enough, sometimes people will cross their paths and the platform is almost never totally empty. Beside these things we also have a random factor due to the fact that passengers never walk in perfectly straight lines.

3.2.2 Random Walk

This random factor can be modelled by a normal distribution $\mathcal{N}(\mu, \sigma^2)$ just like in the cell-based model [26]. It is possible to alter the parameters μ and σ as we like. If we would like a high random factor with a small variance then we can take a high μ and a small σ. We denote this change in direction by $\boldsymbol{n}$ which contains two iid samples of the normal distribution, so for every time step and each passenger this will be a different vector. The formula for the next location based on only this random walk and mechanotaxis is:

$$\boldsymbol{r}_i(t + \Delta t) = \boldsymbol{r}_i(t) + \boldsymbol{d}_i(t) + \boldsymbol{n}.$$

3.2.3 Interaction with other Passengers

When it is rush hour on a busy platform, there are a lot of passengers. Often people do not prefer to walk close to other people, so we allow some distance between the individuals. When we want to determine the next location at time $t+\Delta t$ for passenger i, we need the location of all other passengers at time t. However the direction a passenger will take does not depend on all other passengers. It will mostly depend on other passengers close to him or her. So we need a kind of comfort zone around passenger i and whenever another passenger j is inside this comfort zone we will alter his or her direction to move away from passenger j. Let $d_i^1 > R_i$ be the radius around passenger i, the circle around passenger i with radius d_i^1 is now the comfort zone. The following formula for the next location based on other passengers is obtained:

$$\boldsymbol{r}_i(t + \Delta t) = \boldsymbol{r}_i(t) + \sum_{\substack{j \in N \\ i \neq j}} \boldsymbol{b}_i^j(t), \qquad \boldsymbol{b}_i^j(t) = \begin{cases} \boldsymbol{r}_i(t) - \boldsymbol{r}_j(t) & \text{if } \|\boldsymbol{r}_i(t) - \boldsymbol{r}_j(t)\| < d_i^1, \\ 0 & \text{elsewhere.} \end{cases}$$

For comparing platforms, it may be good to determine the total interaction time with passengers for a simulation. This is the total time that passengers are in each other's comfort zones. Hypothetically, platforms with a high total interaction time will lead to a longer travel times.

3.2.4 Interaction with Walls and Objects

For walls and other objects on the platform we can use the function $\delta_i^w(t)$ from the previous section to determine the minimal distance between passenger i and wall or object w. Here we assume that walls and all objects can be seen as straight line segments. Like with other passengers, passenger i will not alter his direction for a bench which is far away. So we need another comfort zone to check whether walls or objects are close to passengers. However, a wall or object does not move, so walking along the wall does not need to be effecting the direction of the passenger. Walking straight onto a wall will effect the direction, so we need the direction of the previous timestep to determine the direction for this timestep. For this model we determine the minimal distance to every object, when this distance is inside the comfort zone with radius $d_i^2 > R_i$, we determine the minimal distance between the passenger and object again but now a timestep ahead using the last direction of this passenger. If the new distance is smaller than the old distance we know that the passenger is walking towards the wall or object and therefore alter his or her direction. Using the angle of the walking direction towards the wall or object we can determine how much we alter the walking direction, the direction is always in opposite towards the wall but the intensity changes due to the angle of walking direction. We note this direction for passenger i by $\boldsymbol{w}_i$.

Again for analyzing different platforms, it seems to be useful to calculate the total interaction time with walls. It makes it possible to see if a high total interaction time with walls has a

positive or negative influence for the travel time of passengers. Hypothetically, it will be a negative influence, interaction with a wall will lower the speed of passengers which causes a longer travel time.

3.2.5 Collisions

However, since we are dealing with a discrete problem it sometimes occurs that two passengers will overlap with one another. Therefore we need to adjust the location of passengers for those who collide with other passengers or objects. We do this by resetting the location for those who collide. Let d_3 be the distance between two passengers, for example i and j, so $d_3 = \|\boldsymbol{r}_i - \boldsymbol{r}_j\|$. Now we can determine the adjusted location for passenger i, denoted by $\boldsymbol{r}_i^\star$ and in the same way the adjusted location for passenger j.

$$\boldsymbol{r}_i^\star = \boldsymbol{r}_i + \frac{R_i + R_j - d_3}{2} \cdot \frac{\boldsymbol{r}_i - \boldsymbol{r}_j}{\|\boldsymbol{r}_i - \boldsymbol{r}_j\|}, \quad \boldsymbol{r}_j^\star = \boldsymbol{r}_j + \frac{R_i + R_j - d_3}{2} \cdot \frac{\boldsymbol{r}_j - \boldsymbol{r}_i}{\|\boldsymbol{r}_i - \boldsymbol{r}_j\|}.$$

For walls and objects the formula is very much alike, if passenger i overlaps with a wall of object w we can only change the position of passenger i to make sure that he or she does not hit the wall. Again, we denote the adjusted location by $\boldsymbol{r}_i^\star$. The direction has to be away from the wall or object so that the adjusted location for passenger i is equal to:

$$\boldsymbol{r}_i^\star = \boldsymbol{r}_i + (\boldsymbol{r}_i - \boldsymbol{w}(\bar{\tau}_i^w)) = 2\boldsymbol{r}_i - \boldsymbol{w}(\bar{\tau}_i^w).$$

Here $\boldsymbol{w}(\bar{\tau}_i^w)$ is the closest point of wall or object w for passenger i.

3.2.6 Herd Behaviour

On top of all these formulas for changing the position of passengers, we need to take into account the average direction and speed of passengers. When everyone around you is heading to exit $e \in E$ (E is the set of all exits), you probably would go to exit e as well. Furthermore, you would adjust your walking speed a bit when everyone around you is walking faster or slower than you. This can be done after we determine the movement for every passenger. This behaviour depends on a few parameters. Firstly, the choice of the passengers around you for whom you alter your speed, can vary. This is been done by making a circle around every passenger $i \in N$ with radius h. Every other passenger that is inside this circle, that is if his or her centre is inside this region, influences the speed of passenger i. We use $\boldsymbol{m}_i$ to denote the migration vector for passenger i, this is difference between the new location and the old location based on mechanotaxis, random walk, and interaction with passengers, walls and objects. The average movement $\boldsymbol{a}_i$ of the passengers around passenger i is the sum of these migration vectors divided by the number of passengers around him or her. The new migration vector will depend on the amount of herd behaviour that is encountered, say that your direction and speed is based on a fraction p of the movement around you then the altered migration vector $\hat{\boldsymbol{m}}_i$ will be:

$$\hat{\boldsymbol{m}}_i = (1-p)\boldsymbol{m}_i + p\boldsymbol{a}_i.$$

If there is nobody near you, then according to this formula, you would walk slower than you normally would. Therefore the formula needs to be improved:

$$\hat{\boldsymbol{m}}_i = \begin{cases} (1-p)\boldsymbol{m}_i + p\boldsymbol{a}_i & \text{if there are passengers near you,} \\ \boldsymbol{m}_i & \text{if there is no one near you.} \end{cases}$$

3.3 Group of Passengers

Lots of people travel in groups, for example with their family or friends. Usually they are close to each other in the train because they want to talk to one another and they often stay together when they arrive on the platform. Therefore we use the option to set multiple passengers as a group when they arrive at the same time through the same entrance. Taking the same transfer or having your bicycles next to each other outside the station are a couple of reasons why these groups will take the same exit. If there are multiple exits and one exit is closer to a part of the group but another exit is closer to another part of the group, then we need to make sure that they will go to the same exit. By taking the centre of the group we can determine the closest exit to this centre, by using the function $\delta_i^u(\tau, t)$ as before. To determine the centre of a group we take the sum of the coordinates of the passengers and then divide it by the number of passengers in this group (method 1). Another method would be to determine the point such that the distance between each passenger from the group and that point is equal (method 2). However, only method 1 is applicable because a group with 3 passengers on a straight line has no point such that the distance between each passenger and that point is equal. Furthermore, there is only one such point for three passengers not being on a line. This gives a requirement for more members of a group. For a graphical explanation see Figure 3.2. Here you can see that the method 1 is closer to the two passengers and in reality it would seem that the two passengers on the left have bigger influence on choosing the exit than the other passenger on the right.

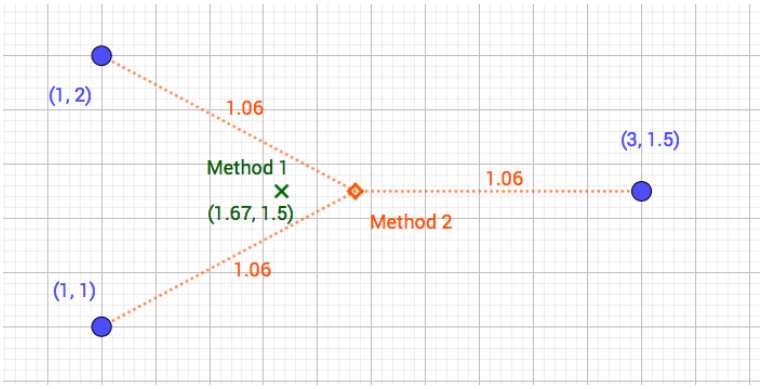


Figure 3.2: Difference between two methods to determine the centre of a group. Three passengers (blue circles). Method 1 (green), mean coordinate. Method 2 (orange), point such that distance from point to every passenger is equal. Method 2 is not applicable because these three passengers define the circumference of a circle. A fourth member must be on this circumference to enable the existence of such a point.

3.4 Own Will

The closest exit is not always the exit you want to go to, mainly because your total route will be shorter if you would take another exit. Therefore some passengers will walk towards a specific exit, regardless of the distance to the exit. Taking this into account the mean travel time of a passenger will increase, due to the fact that the distance to the specific exit will always be greater or equal to the distance to the closest exit. We will analyze this using numerical experiments in Chapter 4.

If one member of a group of passengers has a particular will, then the whole group will go to

the specific exit of this member. If there are multiple members of a group with this characteristic, then they will go to the exit of the leader of the group. For the leader, we will use the member with the lowest index number.

3.5 Departure of Passengers

The final part of the model is when passengers arrive at one of the exits. A passenger is at the exit when the minimal distance between him or her and an exit is smaller than his radius. So $\delta_i^u(\tau, t) < R_i$, here i is the passenger, u is an exit and R_i is the radius of passenger i. The minimal t for which this equation is true is the time that passenger i arrives at the exit and therefore departs from the platform. From that moment we stop calculating the next location for passenger i. This location does not appear in the calculations for other passenger's locations either. We determine the time interval this passenger needed to get from the train to the exit. This is called T_{exit}, this is the time interval we want to minimize. We will analyze this in Chapter 4. The moment that the last person leaves the platform is denoted by T_{evac}, which is the time interval that is necessary to get all passengers off the trains and platform. This is an important value, as stated before, because we aim at an evacuation process that proceeds as quickly as possible.

3.6 Conclusion

The Passenger Model is inspired by cell-based models to simulate passenger flows on a train platform. However, this model can be applied to different situations as well, such as concerts (outside or inside) and sports events. Basically an area with multiple exits and entrances where a lot of people are walking can be simulated with this model. Passengers will arrive on the platform using a Poisson-distribution, walk towards one of the exits and depart when they arrived at an exit. The migration is based on multiple features, namely mechanotaxis, chemotaxis, random walk, collisions and herd behaviour. However this list does not contain all aspects that influence the migration. If a passenger is with someone else in a group, they want to stay close to each other. Some passengers want to go to a specific exit because of their location of work or home. Finally, when all passengers have left the trains and found their way off the platform, the stochastic variables T_{exit} and T_{evac} will be used to analyze different types of platforms. Now a program can be made to simulate multiple passengers on a platform using a combination of all these apsects. Subsequently we can analyze the obtained data which will be done in Chapter 4. As an example of a platform with passengers and multiple objects, a freeze-frame of an animation has been made, see Figure 3.3. Here it can be seen that there are two triangle shaped objects (red lines) and a lot of passengers on the platform. Note that the size of each passenger differs. From the top and bottom, passengers alight from the trains and they will find their way to their preferred exit. Passengers colored in red are part of a group, blue means that they travel alone. The black lines represent the edges of the platform and the green lines represent the entrances of the trains. It can be seen that most space of the platform is occupied by passengers. In particular the number of individuals is high near the exits of the platform.

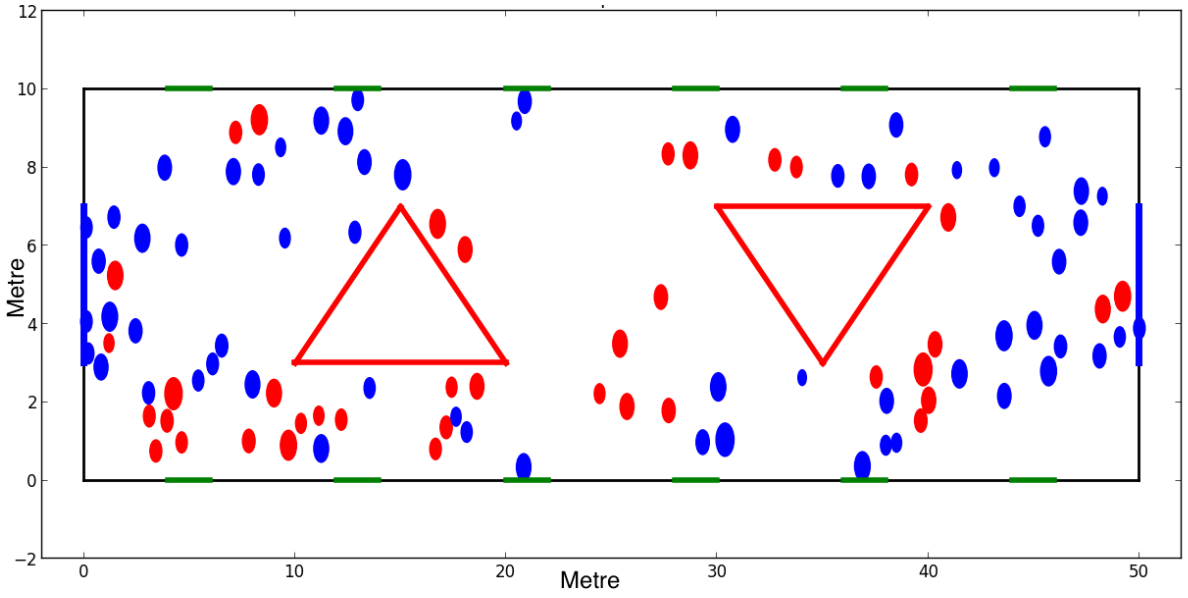


Figure 3.3: A freeze-frame of an animation with 2 trains (their entrances are colored green), 2 exits (blue lines), 200 passengers (red and blue circles) and triangle shaped objects (red lines). The red colored passengers are members of a group and the blue passengers travel alone.

Chapter 4

Statistical Analysis

To analyze the data we used the program RStudio [24]. From our model discussed in Chapter 3 we can produce a lot of data. We have many parameters to change the situation on the platform, we will call these parameters the explanatory variables. Using these explanatory variables we obtain some output variables, the response variables. The focus of this project is to get an optimal platform such that the travel time for the passengers is as low as possible. So the most important explanatory variables are the location and number of obstacles on the platform. The two most important response variables are, the mean travel time of the passengers T_{exit} and the time, T_{evac}, for which all passengers have reached one of the exits. To find the optimal platform we want to have these response variables as low as possible, therefore we want to decide which explanatory variables significantly affect the response variables. Before we begin to analyze the data, some initial choices must be made.

4.1 Initial Values

Humans are unique for a lot of characteristics and for these features we use a normal distribution to create a lot of different values. A lot of these features need to be (strictly) positive and therefore we will use the Lognormal distribution as it does not take negative values [25]. Let $X \sim \mathcal{N}(0,1)$ and $Y = \exp(\kappa + \xi X)$, then $\ln(Y) \sim \mathcal{N}(\kappa, \xi^2)$ and $Y \sim \text{Lognormal}(\nu, \lambda^2)$, with the following probability density function:

$$f(y) = \begin{cases} \frac{1}{y\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma^2}(\ln(y)-\mu)^2\right) & \text{if } y > 0, \\ 0 & \text{if } y \leq 0. \end{cases}$$

The mean of the lognormal distribution is equal to: $\mathbb{E}[Y] = \exp(\nu + \frac{\lambda^2}{2})$ and the variance is equal to: $\text{Var}[Y] = [\exp(\lambda^2) - 1]\exp(2\nu + \lambda^2)$. Say that we want a mean equal to μ and a variance equal to σ^2, then we have these equations:

$$\mu = \exp\left(\nu + \frac{\lambda^2}{2}\right)$$
$$\sigma^2 = [\exp(\lambda^2) - 1]\exp(2\nu + \lambda^2)$$

Using substitution we get:

$$\nu = \ln(\mu) - \frac{\lambda^2}{2} \tag{4.1}$$
$$\lambda^2 = \ln\left(\left(\frac{\sigma}{\mu}\right)^2 + 1\right) \tag{4.2}$$

Using Equations (4.1) and (4.2) we can construct a lognormal distribution with mean μ and variance σ^2.

4.1.1 Passenger Characteristics

For the speed of the passengers we want the mean to be the average walking speed for humans, which is around $\mu = 5\ [km/h] \approx 1.4\ [m/s]$. Furthermore, we take as standard deviation $\sigma = 1.4/5$ to allow passengers to run or to walk slowly according to their preferences. A lot of factors influence this velocity, e.g. age, culture, gender and the purpose of travelling. Similarly we set the sizes for the passengers with the lognormal distribution. The size is fixed by the radius for a passenger, since we are modelling passengers as circles. We take the mean as $\mu = 0.3\ [m]$ and a standard deviation of $\sigma = 0.05$ such that there are very few extremely small or large passengers. Another value we want to vary for passengers is the comfortzone, a circle around a passenger such that he or she moves away for passengers who are inside this circle. This value cannot be negative either, hence we will use the lognormal distribution again. This time with mean $\mu = 2\ [m]$ and standard deviation $\sigma = 0.5$. The comfortzone for walls and obstacles, which we call the wallzone, is set equal to $0.5\ [m]$ for every passenger and works similarly as the comfortzone for passengers. The same holds for the meanzone, this is the zone around a passenger for which he or she will adapt the direction and velocity for passengers in this zone (herd behaviour). We set the radius of this zone equal to $3\ [m]$. How much we take this value into account, is fixed by the value meanvalue, which is set to 0.1. The last zone we need to discuss is the zone for groups. This is the zone for which the centre of a group needs to be inside for every member of the group. The radius of this zone is set to $1.5\ [m]$. The number of passengers entering the platform from one train is set as 100 passengers, the larger the platform and thus the train, the larger this number can be. The parameter for the Poisson distribution to model how many passengers enter the platform per second per entrance is set to $0.5\ [\mathrm{s}^{-1}]$ this means that on average one passenger alights from an entrance of a train per 2 seconds.

4.1.2 Platform and Train Dimensions

The size of the platform is also something we have to talk about. As discussed in Section 2.1 we will model a platform with a length equal to 50 metres and width equal to 10 metres. Although a platform would be larger in reality [19], a smaller platform is simulated to speed up the simulation. The number and size of entrances and exits are based on reality as well. The number of the entrances of a train is set to six with a width of two metres. So out of the 50 metre, about 25% is occupied by the entrances. We will model the platform with two exits, both with a width of four metres. Unless it is specifically otherwise stated, these values will be used during the simulations.

4.2 Influence of the Time Step

Before we start producing and analyzing the data, we need to check whether the time step influences the response variables. As stated earlier, a large time step is desired because otherwise the program has a long running time. First an empty platform is modeled. Simulating this situation 1000 times with time steps equal to $\Delta t = 0.125, \Delta t = 0.25, \Delta t = 0.5$ and $\Delta t = 1$ second, gives us the probability and cumulative density functions that can be seen in Figure 4.1. The figures on the left are the probability and cumulative density function for T_{exit}. It can be seen that decreasing the value of Δt will result in a lower value for T_{exit}, it can also be seen that there is convergence for the mean and variance. We even observe convergence in probability, which implies convergence in distribution. This suggests that for a relatively small Δt we get the true

values for the response variables, see Table 4.1. The models with $\Delta t = 0.125$ and $\Delta t = 0.25$ look very similar, for T_{exit} it is only shifted a bit. If we shift the distribution of $\Delta t = 0.25$ to the left with the difference in mean (which is equal to: $\mu_{\Delta t=0.25} - \mu_{\Delta t=0.125} \approx 0.276$), then the Kolmogorov-Smirnov test [22] gives us a p-value of 0.9356 which strongly suggests that these distributions indeed are samples from the same distribution.

Similar results are for T_{evac}, however in this case the distribution with $\Delta t = 0.5$ seems to fit the distributions with $\Delta t = 0.125$ and $\Delta t = 0.25$. $\Delta t = 1$ is still not usefull.

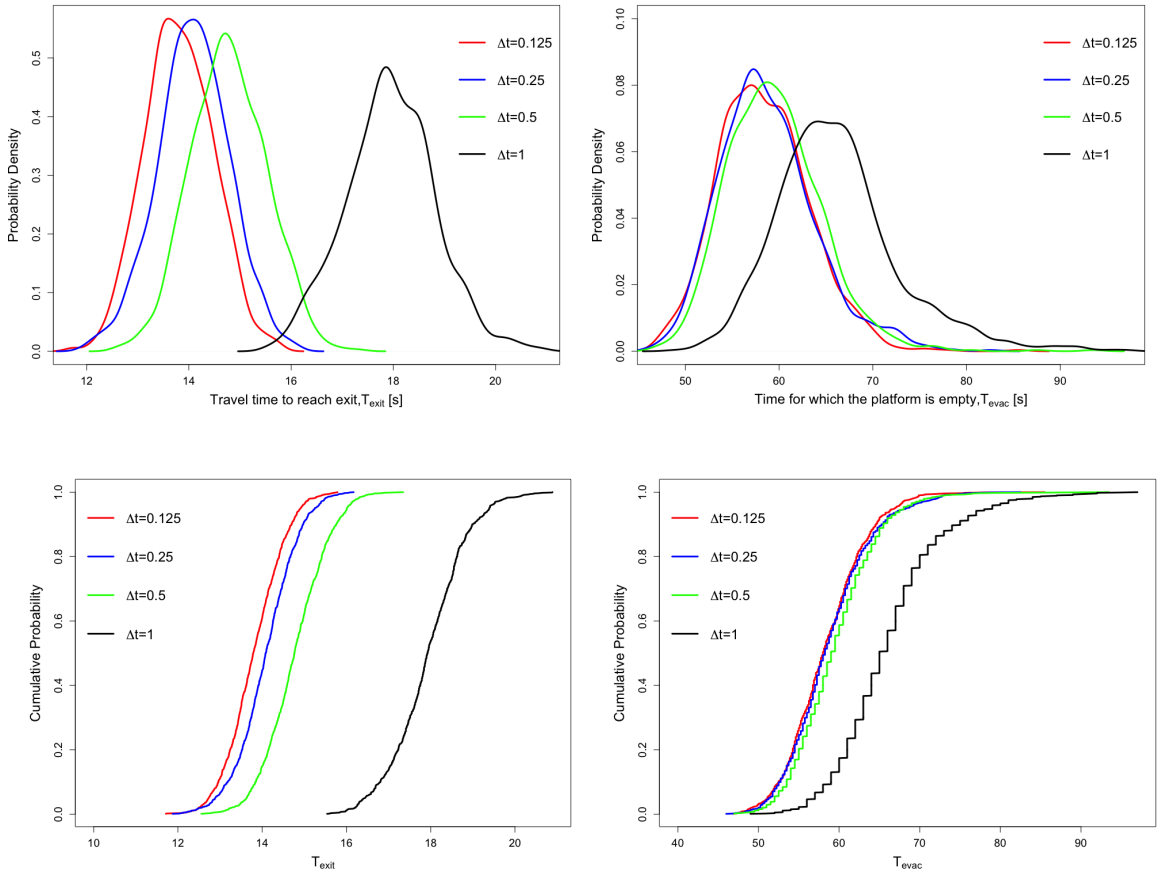


Figure 4.1: Probability density functions (top) and cumulative density functions (bottom) for T_{exit} (left) and T_{evac} (right) for 4 different time steps.

	T_{exit}		T_{evac}	
Δt	mean	variance	mean	variance
1	17.94	0.76	66.13	43.66
0.5	14.78	0.54	59.50	25.80
0.25	14.09	0.50	58.77	26.52
0.125	13.81	0.45	58.38	23.43

Table 4.1: Mean and Variance for T_{exit} and T_{evac} using 4 different time steps (seconds). It can be seen that there is some convergence as Δt decreases, especially for T_{exit}.

Using Richardson's extrapolation [32] it is possible to estimate the rate of convergence, here we assume that the error of the approximation has the form $c_p h^p + \mathcal{O}(h^{p+1})$. Here $c_p \neq 0$ and

$p \in \mathbb{N}$. Let p_1 denote the rate of convergence for T_{exit} and p_2 the rate of convergence for T_{evac}. Using formula (4.3) and (4.4) gives us an estimation of these convergence rates p_1 and p_2.

$$p_1 = \log_2\left[\frac{T_{exit}(\Delta t = \frac{1}{4}) - T_{exit}(\Delta t = \frac{1}{2})}{T_{exit}(\Delta t = \frac{1}{8}) - T_{exit}(\Delta t = \frac{1}{4})}\right] \approx \log_2\left[\frac{14.09 - 14.78}{13.81 - 14.09}\right] \approx 1, \tag{4.3}$$

$$p_2 = \log_2\left[\frac{T_{evac}(\Delta t = \frac{1}{4}) - T_{evac}(\Delta t = \frac{1}{2})}{T_{evac}(\Delta t = \frac{1}{8}) - T_{evac}(\Delta t = \frac{1}{4})}\right] \approx \log_2\left[\frac{58.77 - 59.50}{58.38 - 58.77}\right] \approx 1. \tag{4.4}$$

Since we estimated the error with $c_p h^p$ we can calculate this error:

$$c_{p_1} h^{p_1} = \frac{T_{exit}(\Delta t = \frac{1}{8}) - T_{exit}(\Delta t = \frac{1}{4})}{2^{p_1} - 1} \approx 13.81 - 14.09 = -0.28,$$
$$c_{p_2} h^{p_2} = \frac{T_{evac}(\Delta t = \frac{1}{8}) - T_{evac}(\Delta t = \frac{1}{4})}{2^{p_2} - 1} \approx 58.38 - 58.77 = -0.39.$$

To receive a better approximation for T_{exit} and T_{evac} we can add this error estimate to the original estimation:

$$T_{exit}\left(\Delta t = \frac{1}{8}\right) + c_{p_1} h^{p_1} \approx 13.81 - 0.28 = 13.53\ s,$$
$$T_{evac}\left(\Delta t = \frac{1}{8}\right) + c_{p_2} h^{p_2} \approx 58.38 - 0.39 = 57.99\ s.$$

For better approximations, a smaller h can be used and can be done in further research.

4.3 The Monte Carlo Error

From a mathematical point of view we want to look at the convergence of the mean of the response variables. Since we are using Monte Carlo methods to estimate the mean, variance and the cumulative probability of T_{exit} and T_{evac}, we know that convergence depends on the sample size. Increasing the sample size four times will reduce the error by half [9]. To see if this is correct we need to estimate the mean and the variance of the response variables. Let N be the sample size, then unbiased estimators for the mean $\bar{X}$ and variance σ^2 are respectively $\bar{X}_N = \frac{1}{N}\sum_{i=1}^{N} X_i$ and $s_N^2 = \frac{1}{N-1}\sum_{i=1}^{N}(X_i - \bar{X}_N)^2$, here X_i is for example T_{exit} for simulation i. According to the strong law of large numbers we know that $\bar{X}_N \to \bar{X}$ if $N \to \infty$ and similar for the variance: $s_N^2 \to \sigma^2$ if $N \to \infty$. Imagine that we have k batches of Monte Carlo simulations and each Monte Carlo simulation contains N samples. We denote this by $\{X_N^{(i)}\}_{i=1}^{k}$, so this set contains N times k values (so we need N times k simulations). For every batch of Monte Carlo simulations we know from the central limit theorem that:

$$\sqrt{N}(X_N^{(i)} - \bar{X}) \sim \mathcal{N}(0, \sigma^2),$$

where σ^2 represents the variance of the iid stochastic variables X_i. From this we can calculate the variance of $X_N^{(i)}$:

$$Var(\sqrt{N}(X_N^{(i)} - \bar{X})) = N \cdot Var(X_N^{(i)}) = \sigma^2 \Rightarrow Var(X_N^{(i)}) = \frac{\sigma^2}{N}.$$

We have already an estimation for σ^2, namely s_N^2. So as result we have: $Var(X_N^{(i)}) \approx \frac{s_N^2}{N}$. The Monte Carlo Error (MCE) is defined by the standard deviation of $X_N^{(i)}$, so this means:

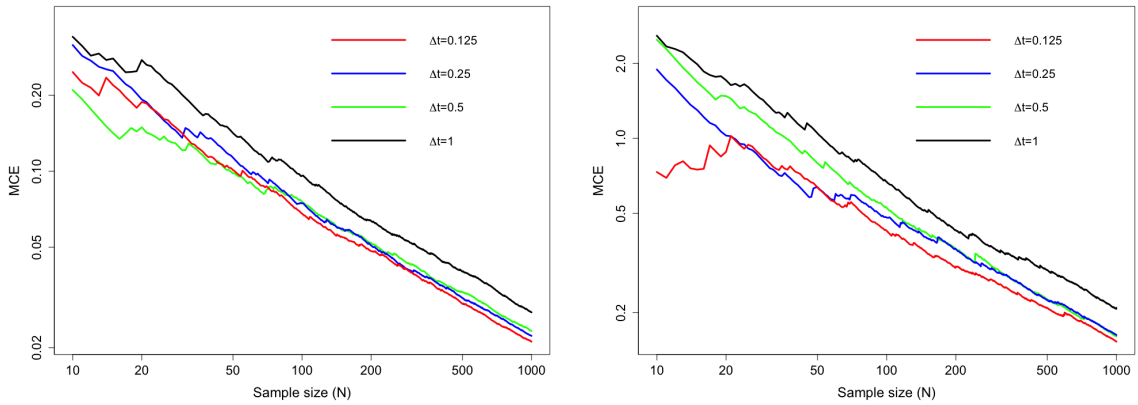


Figure 4.2: The Monte Carlo Error (MCE) plotted against the sample size on a double logarithmic scale for T_{exit} (left) and T_{evac} (right).

$MCE_N \approx \frac{s_N}{\sqrt{N}}$. From this it can be seen that the error reduces with a factor 10 if the sample size increases with a factor 100. The same four different time steps have been used as in the previous section to determine this Monte Carlo Error. In Figure 4.2 the error for T_{exit} and T_{evac} have been plotted versus the sample size N. It can be seen that all time steps follow roughly the same line, however you can also see that a higher Δt results in a higher error. This suggests that multi-level Monte Carlo [7] may be an opportunity to make simulations much faster. Multi-level uses the following formula:

$$\mathbb{E}[T_M] = \mathbb{E}[T_0] + \sum_{i=1}^{M} \mathbb{E}[T_i - T_{i-1}].$$

Here T_i denotes a stochastic variable using time step $h_i = 2^{-i}[s]$. It can be seen that if $\mathbb{E}[T_i - T_{i-1}]$ is small, then a small error has been made if you estimated $\mathbb{E}[T_M]$ with $\hat{T}_M = \mathbb{E}[T_0]$. This has not been done in this study due to lack of time, however it is good to keep in mind that multi-level Monte Carlo simulation can reduce the simulation time.

With the estimates for the mean and the variance we can make 95% confidence intervals for different sample sizes:

$$\mathbb{P}\left(\bar{X}_N - \frac{1.96 s_N}{\sqrt{N}} \leq \bar{X} \leq \bar{X}_N + \frac{1.96 s_N}{\sqrt{N}}\right) = 0.95.$$

Again, this formula shows you that quadrupling the sample size will reduce the confidence interval with 50%. From now on we will simulate every model with a time step $\Delta t = 0.25$. From 4.3 we can see the predicted length which is the observed length using 10 simulations multiplied by $\sqrt{10}$ and then divided by the square root of the sample size N. It can be observed that the expected convergence rate is approximately the same. Note that at the beginning the sample size is very small and that the convergence is not very good. We can also observe that the variance of T_{evac} is much larger than the variance of T_{exit}.

It can be seen that the 95% confidence interval for $\bar{T}_{exit}$ is: $[12.62, 12.70]$, and for $\bar{T}_{evac}$: $[56.49, 57.08]$.

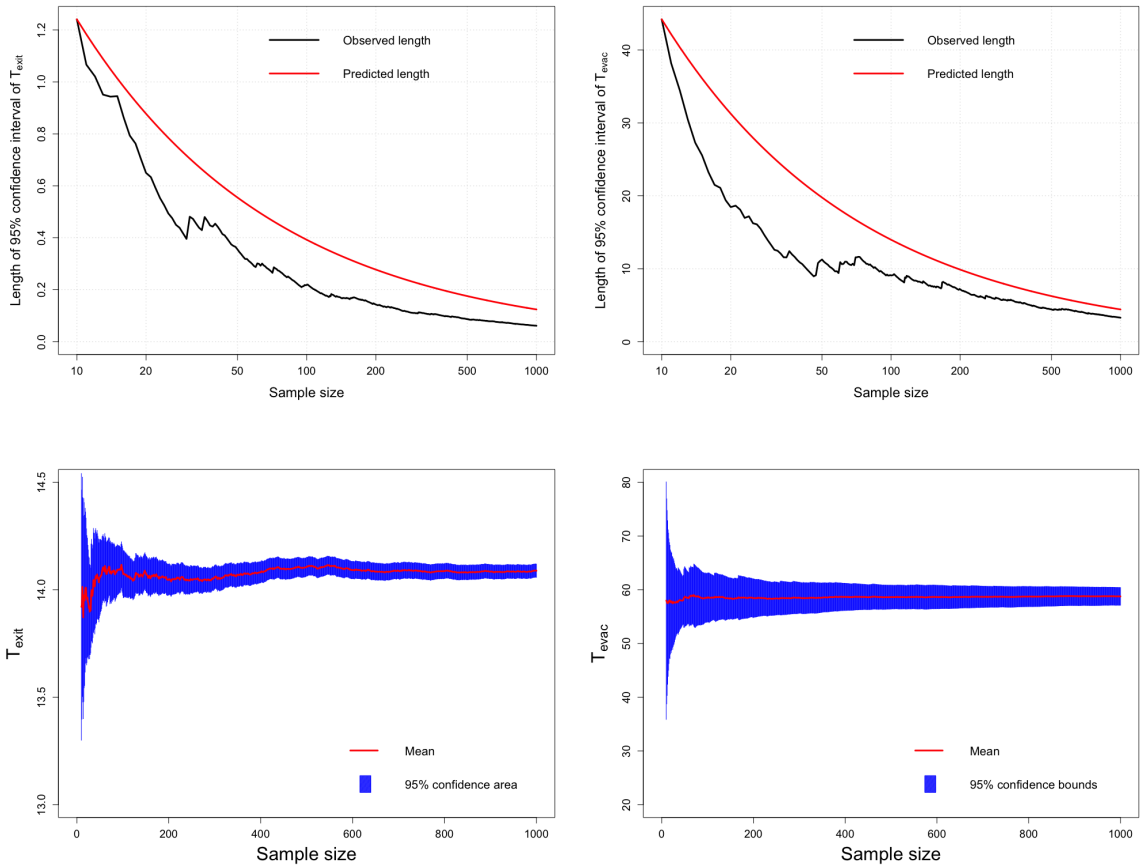


Figure 4.3: Length of confidence interval (observed and predicted) plotted against the sample size for T_{exit} (left) and T_{evac} (right).

4.4 Mean Free Path of Passengers

An interesting value for platform designing engineers is the mean free path, which respresents the mean distance an individual can move without colliding with another individual. To determine this value in our situation, we need to assume that all passengers have the same size, radius $R = 0.3$. Hence the area for which a passenger would collide with another passenger has area equal to $A = \pi(2R)^2$. The total distance a passenger travels on average during time interval t is equal to $\bar{v}t$, here $\bar{v}$ represents the mean velocity of all passengers, which in our case is equal to $\bar{v} = 1.4\ [m/s]$. The area that a passenger uses in time t is equal to: $A + 4R\bar{v}t$. This is the area for which a passenger will collide if the centre of another passenger is inside this area. However, we did not encounter the case that other passengers are moving as well, therefore the mean relative velocity is used: $\bar{v}_{rel} = \sqrt{2}\bar{v}$. Furthermore we need to know how many passengers are on the platform. If there are no passengers on the platform we would have a high mean free path, whereas if the amount of passengers increases, then the mean free path decreases. We will denote the number of passengers per square metre by n_v. The complete formula for the mean free path, λ, is now [17]:

$$\lambda = \frac{\bar{v}t}{n_v(A + 4R\sqrt{2}\bar{v}t)} = \frac{\bar{v}t}{4Rn_v(R\pi + \sqrt{2}\bar{v}t)} \approx \frac{1}{2Rn_v\sqrt{8}}.$$

The last approximation holds if $\bar{v}t\sqrt{2} >> \pi R$, where time t is the time a passenger travels and we saw in our model that a passenger needs 10 to 20 seconds approximately. Let $t = 10$ $[s]$ then we have: $\bar{v}t\sqrt{2} \approx 1.4 \cdot 10\sqrt{2} = 14\sqrt{2} >> 0.3\pi \approx 0.94$. Therefore we think the estimation is reasonable. Using the area of the platform, which is $A_{plat} = 10 \cdot 50 = 500$ $[m^2]$, we can determine n_v at every time step, since we know the number of passengers at every time step. When a train with six entrances arrives at a platform with two exits and we let 1000 people walk from the train to the exits, then there is a time interval in which the number of passengers on a platform hardly changes, therefore the mean free path hardly changes as well. In Figure 4.4 you can see on the left that in the beginning and at the end of the simulation the value for the amount of passengers varies a lot and therefore the value for the mean free path changes rapidly accordingly. On the right you can see a figure where the right y-axis is scaled such that the value for the mean free path can be seen more clearly. As one can see, the value for the mean free path is close to 10 metres. This is, of course, on a platform with no obstacles.

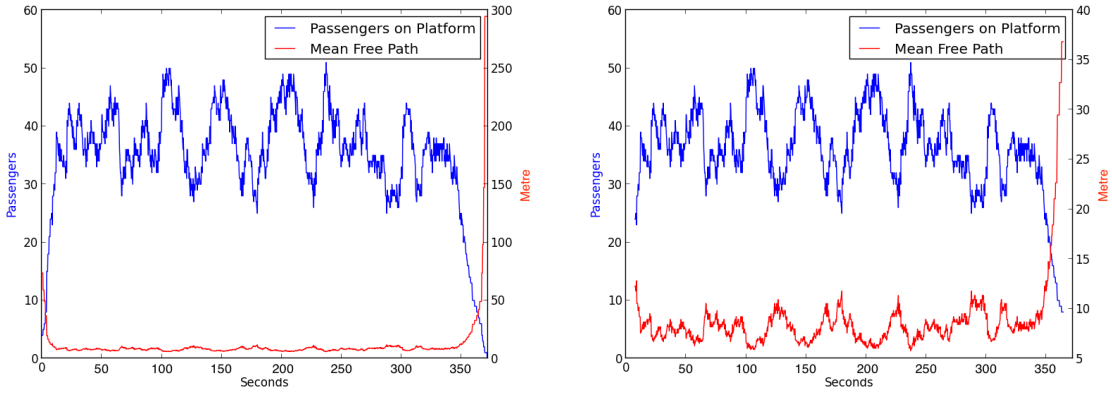


Figure 4.4: The mean free path (red) and the number of passengers (blue) can be seen. On the right a right y-axis is scaled to see more details in the mean free path.

4.5 Sprinter and Intercity Trains

We investigate the influence of the entrances at the trains on the values for T_{exit} and T_{evac}. The size of the entrances could be important as well as the positions of the entrances. Different traintypes or metrotypes have different kind of doors as well as different spacings between doors. Sometimes the floor of the trains are on the same height of the platform, but there are also trains with small steps what will cause a minor delay in the passenger flow.

In the Netherlands there are roughly two types of trains, Sprinters and Intercities. Sprinters are for short distances and therefore the level of comfort is lower than in an Intercity. However they have relatively more doors than Intercities. Furthermore the doors are more spread over the length of the Sprinter in comparison to the Intercity. This is because the Intercity is made for long distances and therefore large coaches are needed. An Intercity repeatedly has a pair of doors close to each

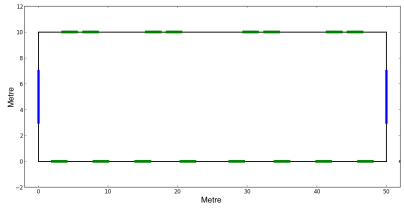


Figure 4.5: Difference between the entrances of an Intercity (top) and a Sprinter (bottom).

other, however the pairs are far away from each other. In Figure 4.5 the difference between the two trains can be seen, the entrances above belong to an Intercity and the ones below belong to a Sprinter. We choose them to be two metres wide to test only if the location matters for T_{exit} and T_{evac}. To test whether there is a difference between these two trains, we simulate a platform with only sprinter doors and a platform with only intercity doors and then look at their distribution functions for T_{exit} and T_{evac}. We will simulate this a 1000 times with 100 passengers. The results can be found in Figure 4.6.

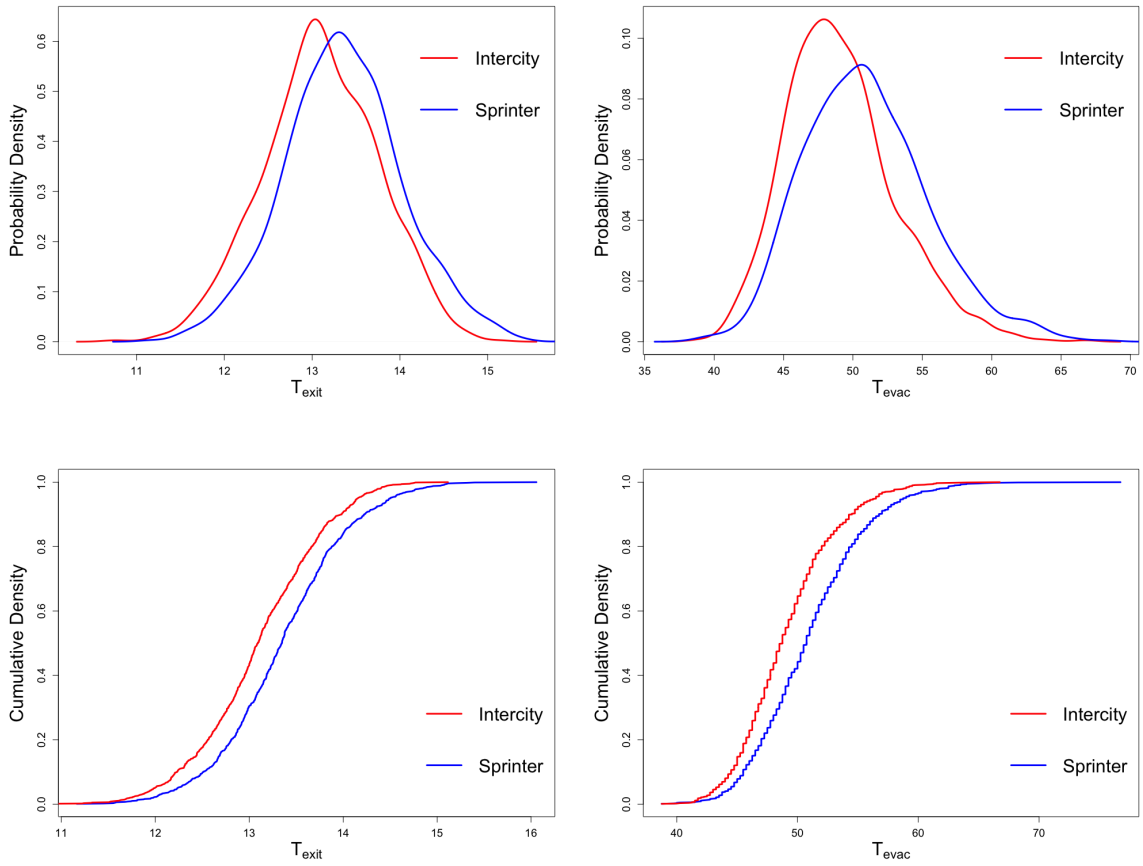


Figure 4.6: Probability (top) and Cumulative (bottom) Density Functions for the T_{exit} (left) and T_{evac} (right). Here two different type of trains are compared, the Sprinter (blue) and the Intercity (red). It can be seen that for both T_{exit} and T_{evac}, the Intercity has better results.

The cumulative distribution function (cdf) of the Intercity compared to the cdf of the Sprinter is more to the left. Therefore the Intercity doors are 'better' than the Sprinter doors, since in general the passengers are faster from the platform with the Intercity doors. It should be noted that the difference between these two doors is not very large, the difference in means for T_{exit} and T_{evac} is respectively 0.26 and 1.89 seconds. Testing whether the datasets come from the same distribution function (the null-hypothesis) with the Kolmogorov-Smirnov test gives us for T_{exit} a p-value equal to $2.837 \cdot 10^{-13}$ and for T_{evac} a p-value $< 2.2 \cdot 10^{-16}$. From these p-values ($p = \mathbb{P}\{\text{reject } H_0 | H_0 \text{ is true}\}$) we can see that the probability to wrongfully reject the null-hypothesis is very low. Therefore it can be concluded that there is indeed enough evidence to say that there is a statistically significant difference in the distributions of the two trains. A

platform with an Intercity will likely be evacuated more quickly than a platform with a Sprinter. Passengers from an Intercity will likely find their exit sooner than passengers from a Sprinter.

4.6 Group of Passengers and Own Will

To measure the influence of the "groups" and "own will" on both response variables, multiple simulations have been done with "groups" and without "groups" and similar for "own will". First some probability density functions have been obtained, which can be seen in Figure 4.7. Here it can be seen that the distributions with both "groups" and "own will" switched off and the one with only "own will" switched on is approximately the same. Similarly, the distribution with both "groups" and "own will" on and the one with only "groups" switched on are roughly the same. This suggests that the group behaviour has an influence on the response variables

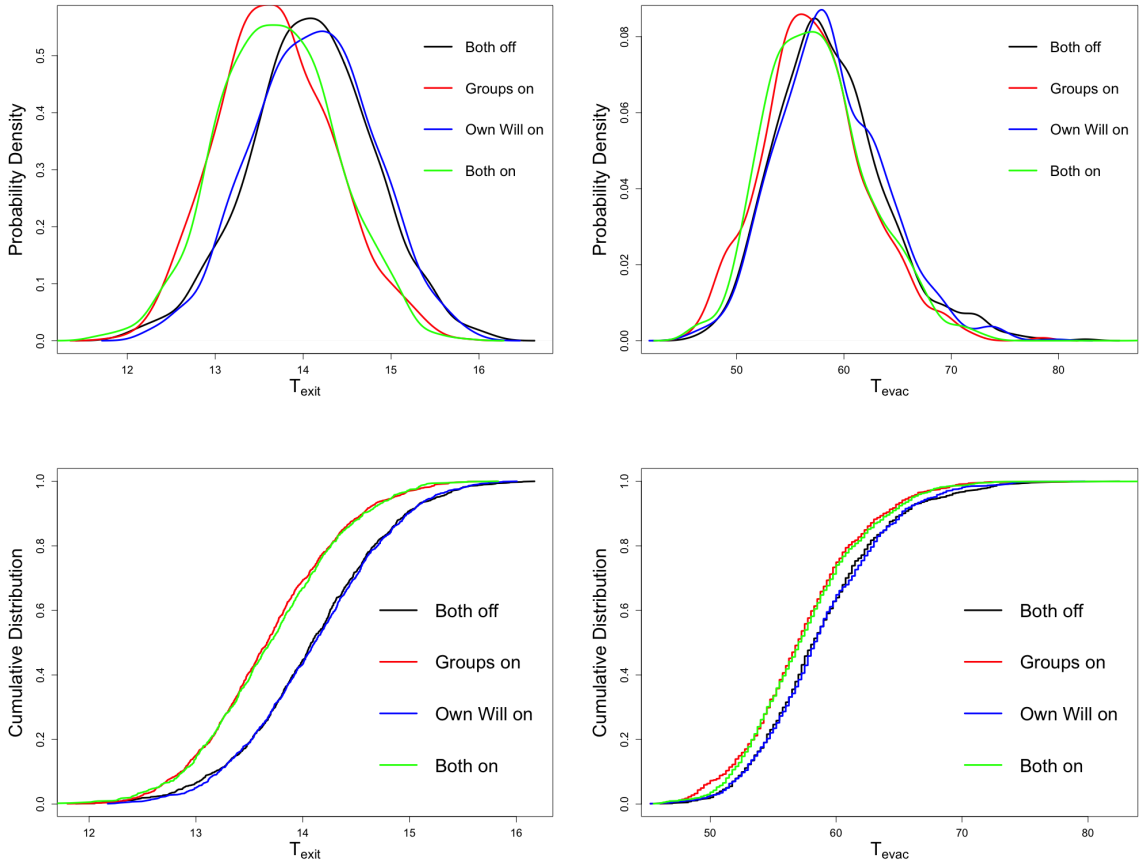


Figure 4.7: Probability density functions (top) and cumulative density functions (bottom) for T_{exit} (left) and T_{evac} (right). "Groups" and "own will" switched off and on.

whereas the factor "own will" has not. A linear model $T = \alpha + \beta \cdot G + \varepsilon$ makes it possible to test whether the factors "groups" and "own will" do influence the response variables [6]. We can test the null hypothesis $H_0 : \beta = 0$ against the alternative hypothesis $H_1 : \beta \neq 0$. Here T is the response variable, α the intercept and β the slope, G is the factor "groups" or "own will" and ε is the error that this model contains. For G equal to the factor "own will" give for both response variables T_{exit} and T_{evac} respectively $\beta = 0.02$ and $\beta = 0.13$ with the following corresponding p-values for the Student's t-test (t-test) [23]: 0.47 and 0.398. Hence the likeli-

hood to wrongfully reject H_0 is large. This implies that switching on or off "own will" does not significantly influence the results. However, fitting the model with G equal to the factor groups gives for both response variables T_{exit} and T_{evac} respectively $\beta = -0.4$ and $\beta = -1.35$ with the following corresponding p-values for the t-test: $< 2 \cdot 10^{-16}$ and $< 2 \cdot 10^{-16}$. This points towards that the factor "groups" has a significant statistical influence.

For further simulations we will still model with both variables switched on because these variables do not influence the computation time. Furthermore these variables are taken into account because they have been introduced to make a platform more realistic.

4.7 Parameter Variation

To verify whether this model is a good representation of reality we alter some of our initial values. Hypothetically, the radius of the passengers will influence T_{exit} and T_{evac} not that much. Smaller passengers do not feel the presence of other passengers as much as larger passengers, so their direction will not alter that much. If the radius is large, then there is a high probability that passengers are in their comfort zones. If two passengers are inside each other's comfort zone, then they will react to this. However the amount of time to find their exit will not alter that much. Simulating a platform with 100 passengers with different radii gives us the opportunity to see whether there is some difference, see Figure 4.8. Here we used the graphical method

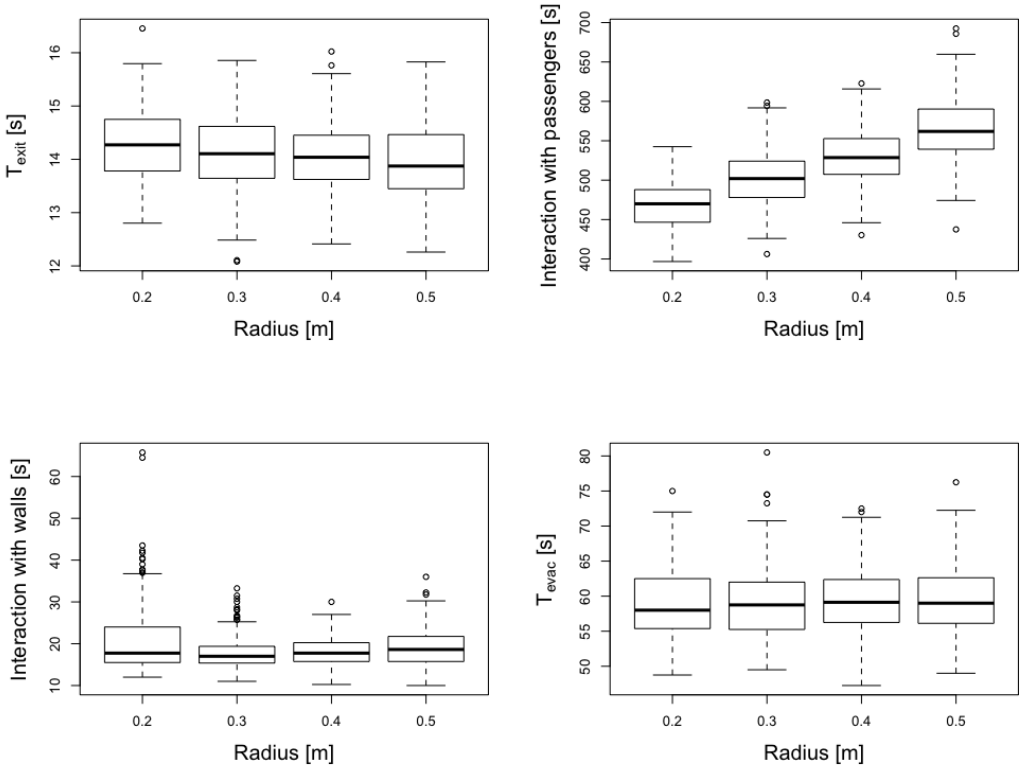


Figure 4.8: Boxplots for the response variables T_{exit}, interaction with passengers, interaction with walls/objects and T_{evac} for different values of width R of passengers.

boxplot to represent a distribution. Sample values are sorted and divided into four groups. The

median (black line) divides the sample values into two equal groups, 50% of the sample size has a sample value greater or equal to this median and the other half has a sample value smaller than this median. The top of the box is the upper quartile, which divides the top 50% into two groups. The bottom of the box is the lower quartile, which divides the bottom 50% into two groups. The dotted line (called whisker) and dots above the box represents the top 25%, where the dots are called outliers. Similar for the dotted line and dots underneath the box, which represents the bottom 25%, where the dots are called outliers again.

You can see that especially the time that passengers are in each other comfort zones increases when the radius increases. It can also be seen that a radius equal to 0.2 metre gives a high spread for the time of interaction with walls. This is mainly because of the fact that we model with a time step of $\Delta t = 0.25$ seconds, this causes a relative big jump for passengers with a small radius which results in 'missing' the exit. Hence this error is caused by a relatively large time step. For both T_{exit} and T_{evac} there is no significant difference. To test whether a different radius significantly changes the mean for the response variables we use the t-test. For the interaction with passengers we get a really low p-value, $< 2 \cdot 10^{-16}$, which means that indeed the radius influences the interaction with passengers. We can also find an estimation for the slope, which is equal to 31.9 [s/dm] with standard error equal to 1.08. This means that increasing the radius of the passengers with 10 centimetres results in 31.9 extra seconds of interaction with passengers (with a small deviation). However for T_{exit} and T_{evac} there is no clear relation. This seems to be a realistic result as stated before.

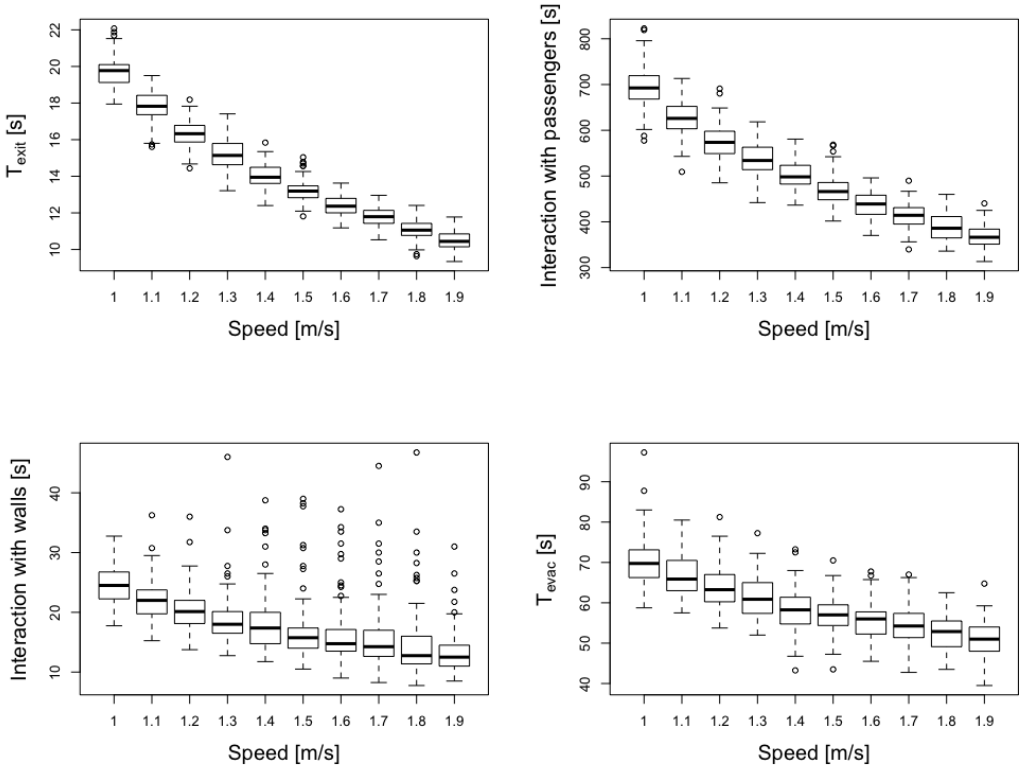


Figure 4.9: Boxplots for the response variables T_{exit}, interaction with passengers, interaction with walls/objects and T_{evac} for different values of the speed of passengers.

Simulating passengers with a different speed will, hypothetically, result in a decreasing relation for all four response variables. Passengers who have a high walking speed will likely find their exit faster than passengers with a low speed, this explains a decreasing relation for T_{exit} and T_{evac}. The passengers are in a short time frame on the platform which results in a low interaction time for both walls and passengers. To see whether this is a correct hypothesis we perform, again, a lot of simulations with different sampled values for the speed. The results can be seen in Figure 4.9.

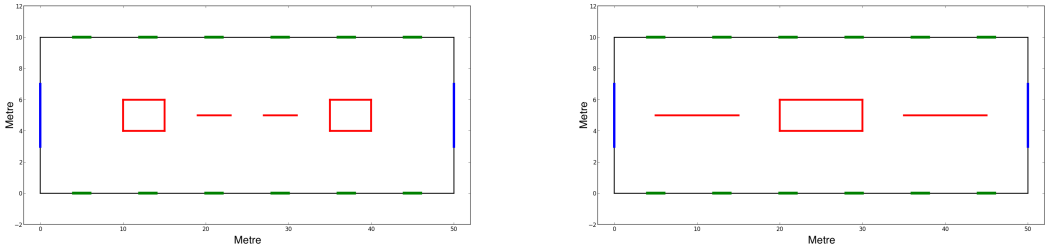
Examining whether the speed influences the response variables using a t-test indeed results into the observation that the speed is of significant importance (all p-values are $< 2 \cdot 10^{-16}$), see Table 4.2. Notice that T_{exit} reduces by roughly one second while T_{evac} reduces with 2 seconds. This is probably due to the fact that T_{evac} mainly depends on the last few passengers and because of the fact that there are fewer passengers on the platform which results into more walking space. These findings seems to be realistic as well.

Response Variable	Slope	Std. Error	p-value
T_{exit}	−9.82	0.10	$< 2 \cdot 10^{-16}$
Interaction with Passengers	−348.7	4.2	$< 2 \cdot 10^{-16}$
Interaction with Walls	−11.22	0.50	$< 2 \cdot 10^{-16}$
T_{evac}	−20.46	0.55	$< 2 \cdot 10^{-16}$

Table 4.2: Testing with the t-test whether the slope is equal to zero. Here, the slope is the change in seconds for the response variables, if the speed is increased with 1 [m/s], with the corresponding standard error and p-value.

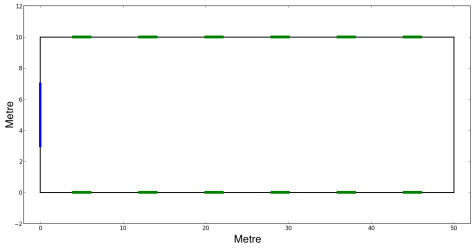
4.8 Platform Design

From an architecture point of view, we want to determine what kind of obstacles influences the transfer and evacuation times. Further, one of our main goals of this study is to make a platform such that T_{exit} and T_{evac} are as low as possible. Using Monte Carlo methods we can model this. We can measure the increase in time but we can also measure how many passengers are close to each other and obstacles or walls. One may expect that the travel and evacuation time is lower if passengers only spend a short time close to walls, obstacles or other passengers and vice versa. We will investigate whether this hypothesis is correct. First we need to place a few obstacles on the platform. Based on maps of platforms and experience, you could have benches and little shops for comfort on a platform. Furthermore, small obstacles such as trash cans, signs and pillars can be added to the platform. We will model four different platforms with obstacles and one platform with no obstacles at all as a reference. The four chosen platforms can be seen in Figure 4.10. Note that the third platform has no obstacles, but the difference is that it only has one exit. Simulating this with this input data, we will have four different response variables, which can be analyzed to choose which platform is preferred in comparison with the other platforms. First probability density functions have been made for T_{exit} and T_{evac}, which can be seen in Figure 4.11. Clearly platform C, with only one exit, stands out. Every measurement of T_{exit} is higher than the maximum value of all other platforms. Also for T_{evac} it can be seen that platform C is clearly not a handy platform for an evacuation in comparison with the other platforms. For all other platforms T_{exit} and T_{evac} are distributed nearly around the same values and the maximum difference between the means is less than half a second. To take a closer look at the difference between platforms A,B and D we construct a cumulative probability distribution function, see Figure 4.12. Here it can be seen that platform B clearly is a better platform

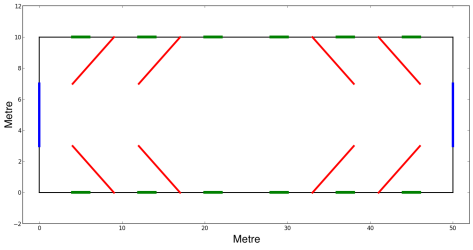


(a) Platform A

(b) Platform B



(c) Platform C



(d) Platform D

Figure 4.10: 4 different platforms, we will call these (from natural reading order), platform A,B,C and D.

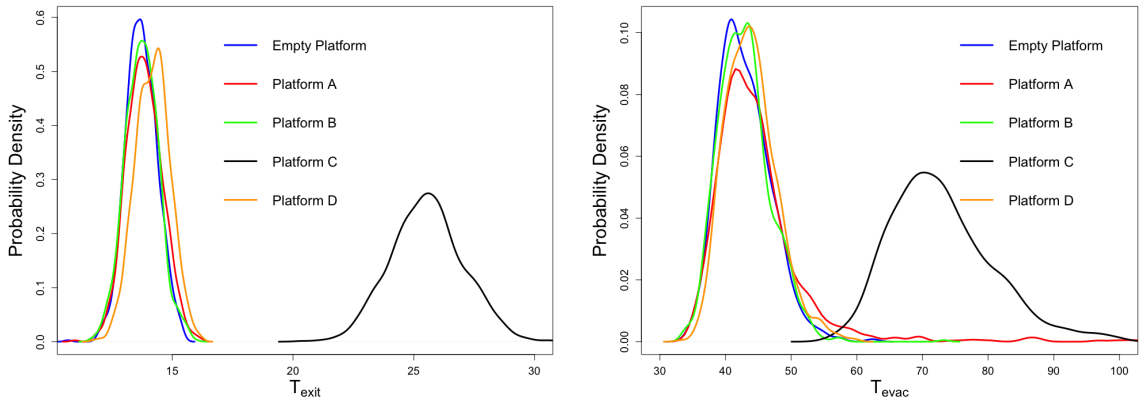


Figure 4.11: Probability density functions for T_{exit} and T_{evac}, for different platforms.

for what concerns T_{exit} and T_{evac}. Moreover in comparison with an empty platform, there is not much difference between the two. The difference between platform A and D is remarkable, for T_{exit}, platform A is clearly a better than platform D. However, if we look at T_{evac} it can be seen that in particular the end of the cdf platform A has a long tail. $\mathbb{P}(T_{evac} \leq 56.75) = 0.95$ for platform A and $\mathbb{P}(T_{evac} \leq 51.0) = 0.95$ for platform D, so we can see a 5 second difference here, which is roughly a 10% increase. From these findings and the Kolmogorov-Smirnov test which confirms our findings we can say that an empty platform has the same distribution as platform B and gives the best results for T_{exit} and T_{evac}. Platform A and D look a bit similar however the results do not follow the same distribution. The p-values from the Kolmogorov-Smirnov test are below 0.01. Platform C has the worst results for T_{exit} and T_{evac}, see Figure 4.11.

To determine whether the interaction time with passengers and walls or objects are significant

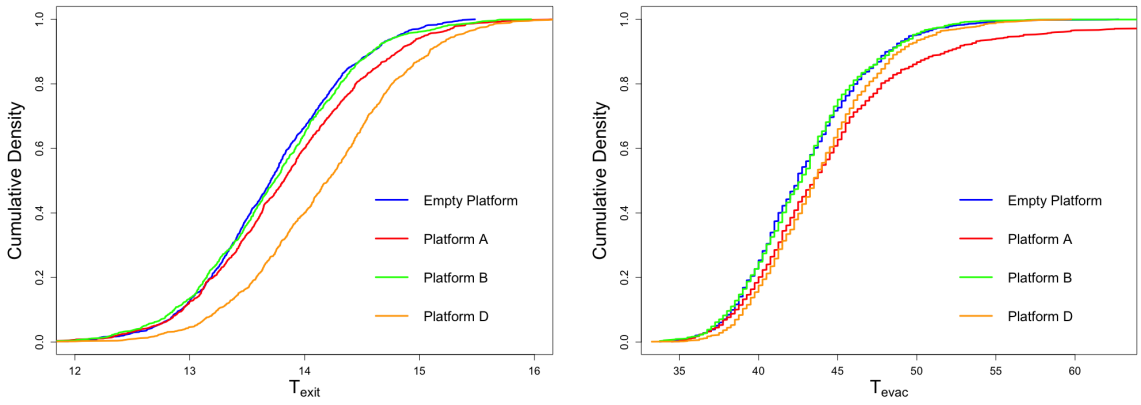


Figure 4.12: Cumulative density functions for T_{exit} and T_{evac}, for different platforms.

for the results, the mean, standard deviation and correlation between these interaction variables with T_{exit} and T_{evac} has been calculated. These values can be seen in Table 4.3. First of all, it

	Interaction Passengers		Correlation		Interaction Walls		Correlation	
Platform	mean	sd	T_{exit}	T_{evac}	mean	sd	T_{exit}	T_{evac}
Empty	358.4	36.97	0.540	0.019	17.44	4.42	0.096	0.100
A	379.3	41.22	0.596	0.010	39.09	19.09	0.362	0.838
B	374.9	38.75	0.541	0.013	17.60	4.27	0.163	0.172
C	602.4	59.79	0.615	0.010	16.81	7.02	0.039	0.038
D	377.9	40.39	0.552	0.040	115.26	12.00	0.738	0.167

Table 4.3: For every platform the mean and standard deviation of the interaction with passengers and interaction with walls/objects is shown (in seconds). Furthermore the correlation coefficients between the interaction times and T_{exit}, T_{evac} are given.

can be seen that the mean value (and standard deviation) for the interaction with passengers is roughly the same for platforms A,B and D. The mean value for an empty platform is the lowest because passengers have more space to walk on and therefore there is less interaction. Notice that the interaction with passengers has some correlation with T_{exit}, but for T_{evac} the correlation is very small. Hence the interaction time with passengers does not influence the evacuation time. From the information of the interaction with walls it can be said that the more interaction the higher the travel time will be, however this does not hold for the evacuation time. For platform A there is a clear relation between the interaction with walls and T_{evac}. For platform D there is a clear relation between the interaction with walls and T_{exit}. Again the difference between platform B and an empty platform is small, there is only a minor increase in correlation for platform B in comparison with an empty platform. From these data, it can be observed that a platform with obstacles increases the interaction time with passengers and walls. A platform with only one exit will have a higher interaction time with passengers but a bit lower interaction time with walls. For a good platform it requires to have a low interaction time with passengers and walls, this is hard to accomplish because adding obstacles will cause more interaction. Therefore platform B has been chosen cleverly because there is only a small increase in the interaction time with walls.

4.9 Conclusion

The Passenger Model described in Chapter 3 has been analyzed. First some realistic assumptions have been made for the initial values of the characteristics of passengers and the dimensions of the platform and trains. The size of the platform is a bit small but this speeds up the computation time and the results will be similar if the platform is larger. To simulate a passenger flow on a platform we need a time step to determine each time step the new location of all passengers. As result we saw that a time step equal to $\Delta t = 0.25$ [s] gives approximately the true values for T_{exit} and T_{evac}. Analyzing these response variables has been done using Monte Carlo methods. Therefore there are few types of errors made. The first error is the error of the model which uses Forward Euler to determine the next location of passengers. The second error is the Monte Carlo Error, this is the error we make if we analyze the data and we found out that the error decreases with the factor $\frac{1}{\sqrt{N}}$, where N is the sample size. Further, we observed that multi-level Monte Carlo may be an opportunity to decrease the simulation time. We also found out that it is possible to determine the mean free path for passengers on a platform and that this value fluctuates between five and ten metres if there is one train on a platform. Changing the location of entrances may be a part of a way to reduce the travel and evacuation time, as we have seen in Section 4.5. Subsequently the factors "group" and "own will" have been implemented to make the model more realistic and after analyzing these factors we found out that the factor "group" has a postive influence on T_{Exit} and T_{evac}, as these response variables decreases when we switch "groups" on.

To test whether the model works properly we altered some of our initial values to see if the results are realistic. Increasing the size of passengers will increase the interaction time with passengers, but the other response variables remain somewhat equal. Increasing the walking speed of the passengers will lead to a decrease for all response variables, because passenger will spend less time to reach an exit. These results satisfy our hypotheses which indicates that our model can be used to analyze different types of platforms. Four different platforms have been compared with an empty platform as reference, simulating 100 passengers and two trains. A platform with only one exit is clearly not suitable. The other three platforms seems to have similar results in comparison with an empty platform, however platform B shows better results than platform A and D, if we take the interaction with passengers and walls into account.

Chapter 5

Transfer Model

To make a more realistic model for passengers on a platform, we need to add more aspects to the model. A lot of passengers want to transfer to another train to reach their destination, therefore the entrances of trains are also treated as exits. However, passengers will not enter the train when there are alighting passengers as this is common courtesy. Furthermore we need to specify where a passenger wants to go to. Moreover, if there is no train at the other side of the platform, transferring passengers need to go to their preferred entrance in a train (so not only to the closest one). This is due to the fact that this particular entrance is close to the exit of the station of their next destination, mostly useful if there is only one exit on a platform which is far to the right or left of a platform. Instead of only passengers that are alighting from trains, there are also passengers that are entering the station and therefore enter the platform via the exits. These passengers want to go to a train as well as the transferring passengers, so the same rules apply for them as well.

With these guidelines making a more realistic model, we can determine the time of walking on the platform and the time of total evacuation again. However, we can determine the dwell time as well. This is very interesting for metro and train services, because on basis on the dwell time of a train a schedule is made for the arriving and departure times. A low dwell time is desired, because this means that more trains per unit time can arrive and this will reduce travel time of the train and thus is beneficial to the passengers. We can determine in what way obstacles on a platform influence these times.

5.1 Description of the Transfer Model

The frame of the model is already there: the Passenger Model as described in Chapter 3. However, we need to adapt a lot of things to make it work. Trains will arrive at different times and will depart at different times and therefore we need to make our own timetable such that there will never be two trains at one side of the platform at the same time. For every train the number of passengers can be adjusted and this can also be done for the amount of arriving passengers, which are the passengers that arrive at the platform through the exits. Furthermore, the size and number of doors, in other words, the type of train can be altered if desired.

Passengers who are entering the platform through the exits have a probability of 50% that they will take the train on their left hand and 50% chance that they will take the train on their right hand, this is regardless of the presence of a train. Thirty percent of the passengers alighting from the trains are set as transfer passengers and will walk to the other side of the platform (if there is still a train that has yet to arrive). But where do the passengers go to if

there is no train? The answer to this question is based on reality and experience on a platform. Many passengers will stand on the side of the platform waiting for the train to arrive, in many countries it is not known on forehand where the entrances of the train are. However in some countries (mainly in East-Asia and Europe), it is known where you can get on a train, because they use platform screen doors [11]. The number of used platform screen doors is increasing and therefore we will model a platform where it is known on forehand where the entrances of the train are located, because then the destination of a passenger is known before the train has arrived. The only thing left to do is to determine which entrance every passenger will go to. This is done using the normal distribution with the expected value equal to the x coordinate and standard deviation equal to the length in metres of the platform (which in this case is 50). The outcome defines the chosen entrance of a train, however values outside the interval $[0, 50]$ are not desired, because this will lead to a strange behaviour. Therefore the normal distribution is used again as long as the value is inside the interval $[0, 50]$. This type of distribution is called the truncated normal distribution [8], see Figure 5.1 for a sample distribution of a passenger with x coordinate equal to 50. When the entrance of a train is chosen there is

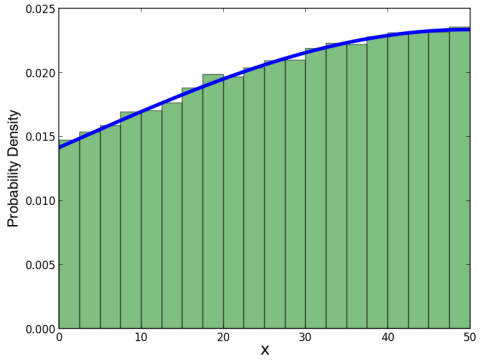


Figure 5.1: Green histogram is produced using 100.000 samples of the truncated normal distribution, blue line is the corresponding probability density function. Here $\mu = 50$ and $\sigma = 50$.

another 50% chance that the passenger will wait on the right hand side of the entrance and 50% to stand on the left hand side. When a passenger has arrived at this area he or she will stay there untill the train arrives and all passengers are alighted from this entrance. However it can also happen that the train arrives during the walk to this area, in this case the passenger will go to the nearest exit and again wait untill all passengers are alighted from this entrance.

When all passengers got off the train, this train still has to wait untill everybody is inside. However, when is this true? There is a continious flow from the exits of the platform to the trains, so it will take very long to wait untill everyone is inside. Therefore it is stated that a train leaves if there are no passengers who alight and if there are no passengers within one metre off the train. The first time that this condition holds for a train, it leaves the train station and the dwell time is then equal to the difference between the time of departure and time of arrival.

Finally, when the last train has left the train station, each passenger needs to leave the platform through the exit (there is no other possibility of leaving the platform). The time at which the last passenger has left the platform is again denoted by T_{evac}. The overall average travel time for all passengers, that is the passengers going from train to another train, passengers going from a train to an exit of the platform and the passengers going from the exit of the platform to a train is denoted by T_{exit}.

5.2 Analysis of the Travel, Evacuation and Dwell Time

Again the same types of platforms have been simulated as in Chapter 4, however only 10 simulations have been done per platform due to long computation time. Three trains have been modeled each with 100 passengers and the timetable can be seen in Table 5.1. The arrival time is fixed, but the departure time can vary due to the fact that not everybody has alighted or due to passengers who are still within one metre of the train. The number of passengers that arrive at the platform is set to 100 passengers per exit. For every train, the dwell time is measured.

	Bottom		Top
	Train 1	Train 3	Train 2
Arrival	0	120	60
Departure	40	160	100

Table 5.1: Timetable of all simulations for three trains in seconds.

T_{exit}, T_{evac} and the interaction times are measured as well. For the travel and evacuation time the boxplots in Figure 5.2 have been made. It can be seen that platform C, with only one exit, is still not useful as it gets for both the exit time and the evacuation time a high value. An empty platform, platform A and platform B give similar results, however an empty platform still has a smaller spread in comparison with the other platforms. Platform D clearly stands out for both variables, as it takes low values for T_{exit} but high values for T_{evac}. This is in contrast to the results in Section 4.8 where it was the other way around. Again a table has been made

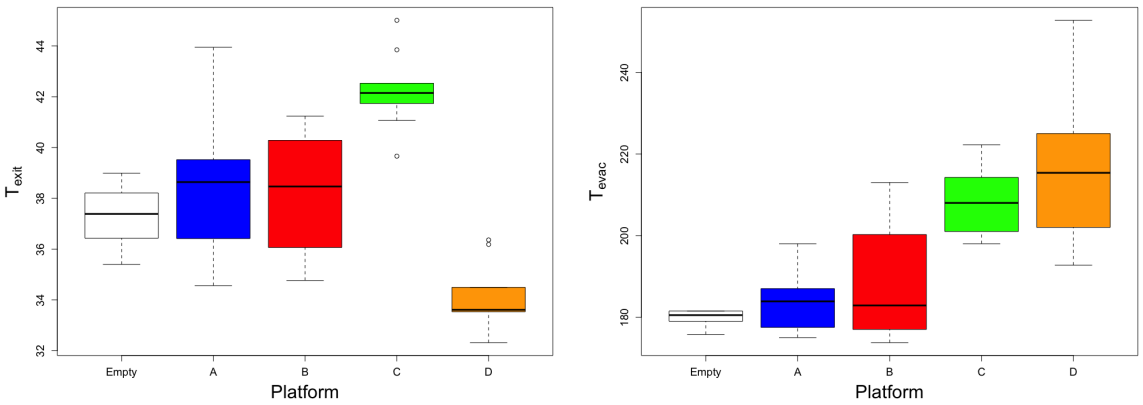


Figure 5.2: Boxplot for T_{exit} and T_{evac} for different platforms.

to investigate whether we can say something about the interaction times, see Table 5.2. For the interaction with passengers we can see that there is a huge increase in comparison with Table 4.3. This is mainly because passengers are close to each other when they are waiting for the next train. Adding obstacles to a platform causes even more interaction time with passengers and the correlation between this interaction and T_{exit} is higher than in case of a platform without any obstacles. Even some negative correlations can be seen, this may be the result of doing only 10 simulations, the corresponding p-values are all above 0.1 so we do not reject the null hypothesis that the correlation ρ is equal to zero (so $H_0 : \rho = 0$ and $H_1 : \rho \neq 0$). Although it can also be the result of a high interaction time which is mainly based due to waiting passengers. Similarly, the platforms with obstacles have a higher interaction time with walls in comparison with platforms without obstacles. Furthermore, the correlation with T_{exit} is much higher for

platforms with obstacles.

	Interaction Passengers		Correlation		Interaction Walls		Correlation	
Platform	mean	sd	T_{exit}	T_{evac}	mean	sd	T_{exit}	T_{evac}
Empty	3833	131	0.239	-0.230	72.18	6.96	0.117	-0.288
A	4084	315	0.788	0.022	904.8	140	0.715	-0.019
B	4247	172	0.447	0.391	966.7	133	0.571	0.413
C	3888	122	-0.260	0.135	49.88	5.20	-0.080	-0.440
D	4386	149	0.496	-0.156	2426	191	0.591	0.461

Table 5.2: For each platform the mean and standard deviation of the interaction with passengers and interaction with walls/objects is shown (in seconds). Furthermore you can see the correlation between these interaction times with T_{exit} and T_{evac}.

Lastly we will look at the dwell times. First we combined the dwell times for all three trains and looked at the difference between the platforms, this can be seen in Figure 5.3. The dwell time for platform D is on average much higher than for all other platforms, with some measurements almost four times higher than measurements from other platforms. This type of platform is

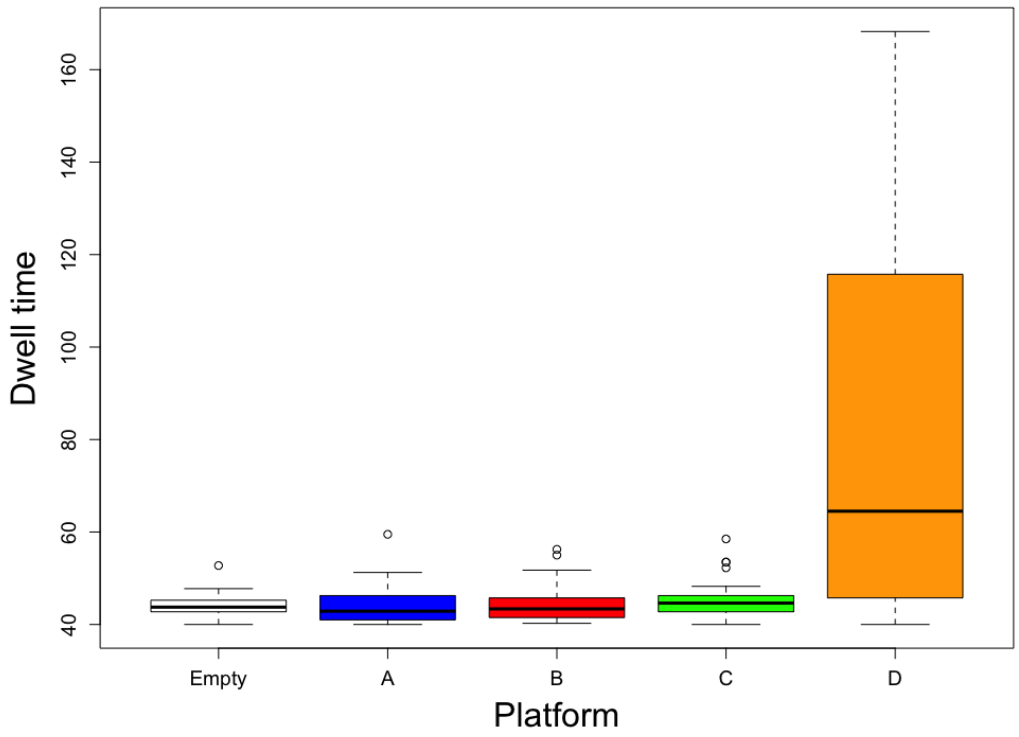


Figure 5.3: Dwell times of 3 trains combined for different platforms (in seconds)). It can be seen that only platform D is really different from the rest.

clearly not useful in real life because of the many obstacles that makes it difficult to transfer. The other platforms give roughly the same observations. In Figure 5.4 it can be seen that the dwell time for the last train, which is train 3, has on average the lowest dwell time. This is due

to the fact that at the end of the simulation, only a few passengers will be at the platform which will cause a low dwell time. An empty platform and platform B seem to give similar results for

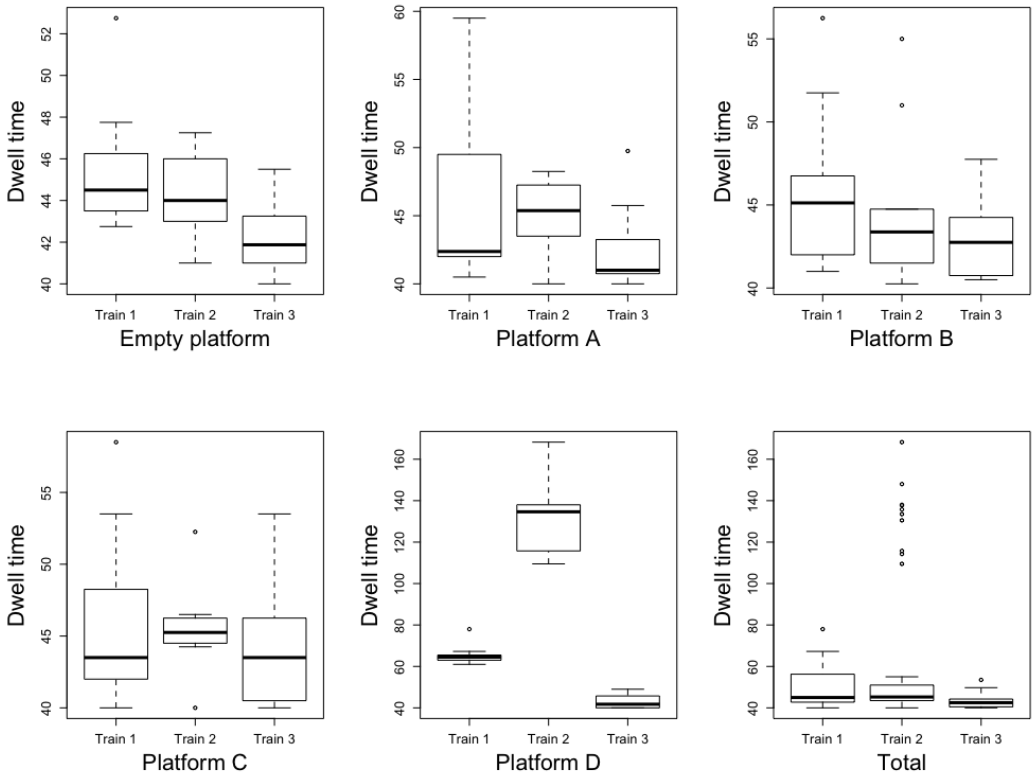


Figure 5.4: Dwell times of the 3 trains for different platforms (in seconds).

the dwell time just like before. Further, these two platforms seem to have the lowest and least varying dwell times of all different platforms. Although platform B contains some objects it gives comparable results for the dwell times as an empty platform. This is a remarkable result, as adding objects would give less walking area for passengers.

To test these two categorials variables we make the following model:

$$Y = \alpha + \gamma_1 D_1 + \gamma_2 D_2 + \beta_1 E_1 + \beta_2 E_2 + \beta_3 E_3 + \beta_4 E_4 + \delta_1 D_1 E_1 + \delta_2 D_2 E_1 + \delta_3 D_1 E_2 + \delta_4 D_2 E_2 + \delta_5 D_1 E_3 + \delta_6 D_2 E_3 + \delta_7 D_1 E_4 + \delta_8 D_2 E_4 + \varepsilon.$$

Here D_i $(i = 1, 2)$ and E_j $(j = 1, 2, 3, 4)$ are dummy variables defined as in Table 5.3. Now α

	D_1	D_2
Train 1	0	0
Train 2	1	0
Train 3	0	1

,

	E_1	E_2	E_3	E_4
Empty Platform	0	0	0	0
Platform A	1	0	0	0
Platform B	0	1	0	0
Platform C	0	0	1	0
Platform D	0	0	0	1

Table 5.3: Defenition for dummy variables D_i $(i = 1, 2)$ and E_j $(j = 1, 2, 3, 4)$.

is the estimation for the dwell time for train 1 on an empty platform and $\alpha + \gamma_1$ for train 2

on an empty platform, moreover $\alpha + \gamma_1 + \beta_3 + \delta_5$ is the estimation for the dwell time of train 2 on platform C. We can estimate the coefficients with this linear model in Rstudio **??**. For every estimation we can test the null hypothesis that this estimation is equal to zero against the alternative hypothesis that it differs from zero. A summary of all these tests is represented in Table 5.4. Here it can be seen that the dwell time is not significantly different for different trains. Also the type of platform does not significantly influence the dwell time, except for platform D which increases the dwell time hugely. Also from the interaction coefficients δ_7 and δ_8 it can be seen that the dwell time for train 2 on platform D is very high in comparison with train 1 on an empty platform. The negative estimation of δ_8 is due to the fact that β_4 has a high positive estimation. If we ignore platform D, it can be seen that every other coefficient (except the intercept α) does not have any stastical significance to the dwell time. So adding obstacles to a platform, platform A and B, do not need to influence the dwell time. However, these high p-values are probabily the effect of doing only 10 simulations per platform which results in only 10 sample values per train per platform. To investigate whether a platform with obstacles has a lower dwelltime than an empty platform, more simulations must be made.

Coefficient	Estimate	p-value	Coefficient	Estimate	p-value
α	45.4	$< 2 \cdot 10^{-16}$	δ_1	0.575	0.877
γ_1	−1.100	0.676	δ_2	0.175	0.963
γ_2	−3.250	0.218	δ_3	−0.275	0.941
β_1	0.175	0.947	δ_4	0.100	0.979
β_2	0.575	0.827	δ_5	0.750	0.840
β_3	0.400	0.879	δ_6	1.675	0.653
β_4	20.125	$3.26 \cdot 10^{-12}$	δ_7	68.700	$< 2 \cdot 10^{-16}$
			δ_8	−19.225	$8.15 \cdot 10^{-7}$

Table 5.4: Estimations for the coefficients with their corresponding p-value for the null hypothesis that they are equal to zero.

5.3 Conclusion

To make the Passenger Model even more realistic, transferring passengers have been taken into account. Again some assumptions have been made to simulate these transferring passengers. A timetable has been made, as we are dealing with multiple trains. Again we analyzed the travel time and evacuation time of the passengers. The interaction with passengers has next to no meaning, because transferring passengers are close to each other when they are waiting to get on the train. Platform C, a platform with only one exit, is still not ideal in reality because of a high travel time and a high evacuation time. Platform D has the best results for T_{exit} but the worst results for T_{evac}, so the obstacles on platform D are not convenient for an evacuation. Platform A and B seem to have the same results as an empty platform, but there is a wider spread. Furthermore, the interaction time with passengers and walls are higher for platform A and B than for an empty platform.

The Transfer Model gives us the opportunity to analyze the dwell time for trains on different platforms. From Figure 5.3 we can see that platform D is clearly not ideal concerning the dwell time as it takes relatively large values. We could not see any differences between the other platforms and an empty platform. This implies that adding obstacles does not necessarily to increase the dwell time. However it should be noted that we only used 10 simulations to measure the dwell time.

Chapter 6

Conclusion and Recommendations

6.1 Conclusion

In this thesis, we studied the possibility of constructing an Agent-Based Model inspired by the cell-based model to simulate passenger flows. Furthermore, we analyzed this model whether it has realistic results and whether it can be used for simulating passenger flows. Two models have been made: the Passenger Model and the Transfer Model. First the cell-based model has been studied to determine which aspects can be used in these models. We adjusted these models even more by using a Poisson distribution for the arrival of passengers and taking "groups" and "own will" into account. Even the herd behaviour of humans have been included in the model. The travel time and the evacuation time have been analyzed as we want to minimize these variables. A good platform requires some objects, because they have a positive influence on the comfort of passengers, however a low interaction time with passengers and walls is desired. We found out that the location of doors, the factor "groups", the size of passengers and the speed of passengers have an influence on the response variables. Assumptions that we made seem to be realistic and all the results satisfied our hypotheses, which indicates that these models indeed represent the reality.

Analyzing four different types of platforms with two trains on each side of the platform has been done with the Passenger Model. A platform with only one exit is clearly not ideal as the travel and evacuation time are relatively high. Furthermore, it seems that a platform with a large object in the centre of a platform gives better results than large objects closer to the exits. The same four platforms have been analyzed with the Transfer Model. This model is even more realistic as passengers can transfer to other trains and these trains can arrive and depart according to our own choice. We analyzed again the travel and evacuation time as well as the dwell time. We found out that a platform with objects near the entrances of trains has a negative influence on the evacuation and the dwell time. Adding objects in the centre of a platform do not have significantly different results than an empty platform. This is an interesting result, because this indicates that it is possible to add objects to a platform without getting worse results for the travel, evacuation and dwell time. Perhaps even better results can be obtained by adding objects to a platform.

6.2 Recommendations

While constructing the models, we encountered several things that can be adjusted to make the models even more realistic. From the cell-based model we used the mechanotaxis to determine the movement from passengers towards exits. The constants A and B can be chosen

differently to simulate a more realistic passenger flow. The interaction with passengers and walls/objects happens instantaneously, however it would be more realistic if this happens more gradually. Furthermore, to implement the herd behaviour we used a fraction p, this is the amount of movement of a passenger which is based on passengers around you. This p does not depend on the number of passengers around a passenger. It is more likely that passengesr will adapt their velocity more if there are more passengers around them. For the Transfer Model we used the truncated normal distribution to determine the movement of a transferring passenger. Adjusting the parameters of this distribution can ensure that the reality is better simulated.

Simulating both models led to a few problems. The simulation time may be reduced using multi-level Monte Carlo so more simulations can be made to make more precise estimations. More Monte Carlo techniques can be used to reduce the variance, the control variates method [9] has been implemented but led to a very small decrease. Especially for the Transfer Model, more simulations are required to analyze the data better. This can lead to different results or that results can be substantiated more strongly. For further studies we also recommend to simulate more kinds of platforms to get a better picture for an optimal platform. Lastly it would be very useful to compare these findings with real data. This may be the best way to check if the models do represent reality.

Bibliography

[1] Bruce Brooks, E. (2007, August 24). The Poisson Distribution. Retrieved December 5, 2017, from http://www.umass.edu/wsp/resources/poisson/index.html

[2] Delft University of Technology. (2016). NOMAD. Retrieved December 5, 2017, from https://www.tudelft.nl/en/ceg/about-faculty/departments/transport-planning/research/competence-centres/pedestrians/simulation-models/nomad/

[3] Dijkstra, J., Jessurun, J. & Timmermans, H. (2001). A Multi-Agent Cellular Automata Model of Pedestrian Movement. Retrieved from http://alexandria.tue.nl/openaccess/Metis209483.pdf

[4] Dudaie, M., Weihs, D., Vermolen, F. J. & Gefen, A. (2015). Modeling migration in cell colonies in two and three dimensional substrates with varying stiffnesses. Retrieved from https://link.springer.com/content/pdf/10.1186%2Fs40482-015-0005-9

[5] Fitzsimmons, E. G., Fessenden, F. & Rebecca Lai, K. K. (2017, June 28). very New York City Subway Line Is Getting Worse. Here's Why. *The New York Times*. Retrieved from https://www.nytimes.com/interactive/2017/06/28/nyregion/subway-delays-overcrowding

[6] Fox, J. (2016). Applied Regression Analysis & Generalized Linear Models (3rd ed.). Thousand Oaks, United States: SAGE.

[7] Giles, M. B. (2008). Multilevel Monte Carlo Path Simulation. Retrieved from http://people.maths.ox.ac.uk/gilesm/files/OPRE_2008.pdf

[8] Grimmett, G. & Welsh, D. (2014). Probability an introduction (2nd ed.). Oxford, United Kingdom: Oxford University Press.

[9] Higham, D. J. (2013). An Introduction to Financial Option Valuation (9th ed.). Cambridge, United Kingdom: Cambridge University Press.

[10] Hunter, J. D. & Droettboom, M. (2015). Matplotlib, Version 1.2.0 [Software]. Retrieved from http://matplotlib.org

[11] Intelligent Transport. (2007, April 19). Platform Screen Doors: No barrier to success. Retrieved December 5, 2017, from https://www.intelligenttransport.com/transport-articles/1725/platform-screen-doors/

[12] Jones, S. (2017, June 4). More than 1,500 Juventus fans in Turin injured after stampede. Retrieved December 5, 2017, from https://www.theguardian.com/football/2017/jun/03/juventus-fans-injured-turin-square-panic-firecrackers

[13] Katz, G. (2017, November 25). 2 Men Questioned by Police After Causing Panic at London Subway Station. Retrieved December 5, 2017, from http://time.com/5036947/men-questioned-london-panic/

[14] Mackintosh, T. (2017, November 28). 'Gunshot' claim sparked Oxford Circus 'security alert'. Retrieved December 5, 2017, from http://www.bbc.com/news/uk-england-london-42149542

[15] Moehlis, J. M. (2001, December 6). A Standard Wiener Process. Retrieved December 10, 2017, from https://me.ucsb.edu/~moehlis/APC591/tutorials/tutorial7/node2.html

[16] Mwaniki, A. (2017, April 25). Longest Railway Platforms In The World. Retrieved from http://www.worldatlas.com/articles/the-longest-railway-platforms-in-the-world.html

[17] Nave, R. (2017). Mean Free Path. Retrieved December 5, 2017, from http://hyperphysics.phy-astr.gsu.edu/hbase/Kinetic/menfre.html

[18] NumPy developers. (2016). Numpy, Version 1.7.2 [Software]. Retrieved from http://www.numpy.org

[19] ProRail. (n.d.). Perrons: welk station heeft de langste? Retrieved December 10, 2017, from https://www.prorail.nl/nieuws/perrons-welk-station-heeft-de-langste

[20] Proto, L. (2016, June 22). Revealed: London's most overcrowded Tube stations. *London Evening Standard*. Retrieved from https://www.standard.co.uk/news/transport/revealed-londons-most-overcrowded-tube-stations-a3277706

[21] Python Software Foundation. (2014). Python Version 2.7.9 [Software]. Retrieved from https://www.python.org

[22] R Documentation. (n.d.). Kolmogorov-Smirnov Tests. Retrieved November 10, 2017, from https://stat.ethz.ch/R-manual/R-devel/library/stats/html/ks.test.html

[23] R Documentation. (n.d.). Student's t-Test. Retrieved November 10, 2017, from https://stat.ethz.ch/R-manual/R-devel/library/stats/html/t.test.html

[24] RStudio Team. (2016). RStudio Version 1.1.383 [Software]. RStudio: Integrated Development for R. RStudio, Inc., Boston, MA. Retrieved from http://www.rstudio.com/

[25] The Scipy community. (2008). numpy.random.lognormal. Retrieved December 5, 2017, from https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.random.lognormal.html#numpy.random.lognormal

[26] The Scipy community. (2008). numpy.random.normal. Retrieved December 5, 2017, from https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.random.normal.html#numpy.random.normal

[27] The Scipy community. (2008). numpy.random.poisson. Retrieved December 5, 2017, from https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.random.poisson.html#numpy.random.poisson

[28] Transport Statistics Great Britain (TSGB), Department for Transport (DfT) & Transport for London (TfL). (2017, February). People entering central London during the morning peak, since 1996. Retrieved from https://www.gov.uk/government/statistical-data-sets/tsgb01-modal-comparisons

[29] United Nations, Department of Economic and Social Affairs (UN DESA). (2014, July 10). Worlds population increasingly urban with more than half living in urban areas. Retrieved from `http://www.un.org/en/development/desa/news/population/world-urbanization-prospects-2014`

[30] Vermolen, F. J. (2015). Particle Methods To Solve Modelling Problems in Wound Healing and Tumor Growth. Retrieved from `https://link.springer.com/content/pdf/10.1007%2Fs40571-015-0055-6.pdf`

[31] Vermolen, F. J. & Gefen, A. (2011). A semi-stochastic cell-based formalism to model the dynamics of migration of cells in colonies. Retrieved from `https://link.springer.com/content/pdf/10.1007%2Fs10237-011-0302-6`

[32] Vuik, C., Vermolen, F. J., Van Gijzen, M. B. & Vuik, M. J. (2016). Numerical Methods for Ordinary Differential Equations. (2nd ed., pp. 67-68). Delft, The Netherlands: Delft Academic Press.

[33] Wolf, H. & Fischer, A. (2017, December 4). Loveparade 2010: ein Rückblick vor dem Prozessbeginn. Retrieved December 5, 2017, from `https://www.waz.de/staedte/duisburg/loveparade/loveparade-2010-ein-rueckblick-vor-dem-prozessbeginn-id212735643`

Appendices

Appendix A

Distributions

A.1 Normal Distribution

The probability density function of a random variable X is:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right).$$

With μ and $\sigma^2 > 0$ denotes the mean and the variance of the distribution. Notation: $X \sim \mathcal{N}(\mu, \sigma^2)$.

A.2 Log-Normal Distribution

Let X be normally distributed with mean μ and variance $\sigma^2 > 0$ (i.e. $X \sim \mathcal{N}(\mu, \sigma^2)$). Then $Y = e^X$ has the log-normal distribution with the following probability density function:

$$f(y) = \begin{cases} \frac{1}{y\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma^2}(\ln(y)-\mu)^2\right) & \text{if } y > 0, \\ 0 & \text{if } y \leq 0. \end{cases}$$

Vica verca, $\ln(Y) \sim \mathcal{N}(\mu, \sigma^2)$. The mean and variance are respectively $\exp(\mu + \frac{\sigma^2}{2})$ and $\exp(2\mu + \sigma^2)[\exp(\sigma^2) - 1]$. One of the main properties of this distribution is that it only takes positive real values.

A.3 Truncated Normal Distribution

Let X be normally distributed (i.e. $X \sim \mathcal{N}(\mu, \sigma^2)$) with $\mu \in \mathbb{R}$ and $\sigma^2 > 0$. If $X \in (a, b)$ for some $a, b \in \mathbb{R}$ with $a < b$, then X has a truncated normal distribution ($a < X < b$). The probability density function for $a \leq x \leq b$ is:

$$f(x) = \frac{\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right)}{\sigma\left(\frac{1}{2}\left(\operatorname{erf}\left(\frac{b-\mu}{\sigma\sqrt{2}}\right) - \operatorname{erf}\left(\frac{a-\mu}{\sigma\sqrt{2}}\right)\right)\right)}.$$

If $a \to -\infty$ and $b \to \infty$ you can see that f will approach the probability density function for the normal distribution.

A.4 Poisson Distribution

A random variable X is Poisson distributed with parameter λ if X takes values in $\{0, 1, 2, \ldots\}$ and their probability is equal to:

$$\mathbb{P}(X = k) = e^{-\lambda} \frac{\lambda^k}{k!}, \qquad \text{for } k = 0, 1, 2, \ldots.$$

Here λ is the average rate of an event happening in a fixed time interval. If such events occurs independently from each other, then the Poisson distribution can be used. The mean and variance of X are both equal to λ.

Appendix B

Python Code

B.1 Passenger Model

Here the main program (B.1.1) uses the functions from B.1.2 to do 1000 simulations with 100 passengers. The output is a matrix with response variables T_{exit}, T_{evac}, the interaction time with passengers and the interaction time with walls and objects. Initial values can be adjusted to adapt the characteristics of passengers or the dimensions of the train and platform. Obstacles can be added to simulate different types of platforms.

B.1.1 Main Program

```
import numpy as np
import matplotlib.pyplot as plt
import sys
import time
from Passengermodel import poissoncalculate, initialvalues, determinewalls, progression, determinematrix
##Initial Values##
#These values can be adjusted#
width = 10.                 #metre
length = 50.                #metre
dt = 0.25                   #secondes
sprinter = np.array([[[2,0],[4,0]], [[8,0],[10,0]],[[14,0],[16,0]], [[20.5,0],[22.5,0]], [[27.5,0],[29.5,0]],
    - [[34,0],[36,0]], [[40,0],[42,0]], [[46,0],[48,0]]])
intercity = np.array([[[3.5,width],[5.5,width]],[[6.5,width],[8.5,width]],[[15.5,width],[17.5,width]],[[18.5,
    - width],[20.5,width]],[[29.5,width],[31.5,width]],[[32.5,width],[34.5,width]],[[41.5,width],[43.5,width
    - ]],[[44.5,width],[46.5,width]]])
entrance = np.array
    - ([[[4,0],[6,0]],[[12,0],[14,0]],[[20,0],[22,0]],[[28,0],[30,0]],[[36,0],[38,0]],[[44,0],[46,0]]])
exits = np.array([[[0,3],[0,7]],[[length,3],[length,7]]])
obstacles = np.zeros((0,2,2))
numberofpassengers = 100
numberofsimulations = 1000
poissonvalue = 0.5          #/second
speedinit = 1.4             #metre/second
Rinit = 0.3                 #metre
wallzone = 0.5              #metre
comfortzoneinit = 2.        #metre
groupzone = 1.5             #metre
meanzone = 3.               #metre
meanvalue = 0.1
grouptrue = False
ownwilltrue = False
AB = [1.1, 10**-3]
##Main Program##
edges = np.array([[0.,0.],[0.,width],[length,width],[length,0.]])
Walls = np.array([[edges[0],edges[1]],[edges[1],edges[2]],[edges[2],edges[3]],[edges[3],edges[0]]])
time1 = time.time()
X = np.zeros((numberofsimulations,23))
for a in range(numberofsimulations):
    progression(a,numberofsimulations)
    numberofsteps = []
    people = []
    grouplst = []
    Walls = determinewalls(Walls,np.zeros(shape = (0,2,2)),exits)
    speed, R, peopleonplatform, comfortzone, ownwill = initialvalues(numberofpassengers,comfortzoneinit,Rinit,
        - speedinit,exits,ownwilltrue)
    poisson = poissoncalculate(numberofpassengers,entrance,poissonvalue,Rinit)
    mean, evac, interpas, interwall = determinematrix(Walls,entrance,poisson,peopleonplatform,dt,length,width,R
        - ,people,numberofpassengers,obstacles,wallzone,exits,comfortzone,speed,numberofsteps,a,grouplst,
        - groupzone, grouptrue,meanzone,ownwill,meanvalue,AB)
    X[a,:] = np.array([mean,interpas, interwall, evac, length,width, dt, comfortzoneinit, poissonvalue,
        - speedinit,Rinit,wallzone,groupzone, meanzone, meanvalue, AB[0], AB[1], numberofpassengers, len(exits)
        - , len(entrance), len(obstacles), grouptrue, ownwilltrue])
print 'It took ' + str(int(time.time()-time1)/60) +' minutes and ' + str(int(time.time()-time1)%60) + ' seconds
    - to determine the matrix.'
##Save Data##
string = ''
for i in range(len(obstacles)):
    string += str(obstacles[i][0])+str(obstacles[i][1])
if string == '':
```

```
    string = 'no'
np.savetxt("matrix_" + string +"_obstacles_" + str(len(entrance)) + "_entrance_" + str(dt) + "_dt.csv", X,
    - delimiter=",")
```

B.1.2 Functions

```
import numpy as np
import matplotlib.pyplot as plt
import sys
##Initial Values##
def normaltolognormal(mu,sigma):
    #determines the parameter for the lognormal distribution with mean mu and sd sigma#
    l = np.sqrt(np.log((sigma/mu)**2+1))
    v = np.log(mu)-l**2/2
    return v,l
def initialvalues(numberofpassengers, comfortzoneinit,Rinit, speedinit, exits,ownwilltrue):
    #determines the intial values for all passengers#
    v,l = normaltolognormal(speedinit,speedinit/5); speed = np.random.lognormal(v,l,numberofpassengers)
    v,l = normaltolognormal(Rinit,0.05); R = np.random.lognormal(v,l,numberofpassengers)
    peopleonplatform = range(numberofpassengers)
    v,l = normaltolognormal(comfortzoneinit,0.5);comfortzone = np.random.lognormal(v,l,numberofpassengers)
    ownwill = (np.random.uniform(0,10,numberofpassengers)<2)*np.random.randint(1,1+len(exits),
        - numberofpassengers) if ownwilltrue == True else np.zeros(numberofpassengers)
    return speed, R, peopleonplatform , comfortzone, ownwill
def exitinwall(exits,wall):
    #determines whether there is an exit in a wall#
    for exit in exits:
        if wall[0][1] != wall[1][1] and exit[0][1] != wall[0][1] and wall[0][1] != exit[1][1]:
            ratio = (wall[0][0] - wall[1][0])/(wall[0][1]-wall[1][1])
            if (wall[0][0] - exit[0][0])/(wall[0][1]-exit[0][1]) == ratio and (wall[0][0] - exit[1][0])/(wall
                - [0][1]-exit[1][1]) == ratio:
                if min(wall[0][1],wall[1][1])<min(exit[0][1],exit[1][1]) and max(wall[0][1],wall[1][1])>max(
                    - exit[0][1],exit[1][1]):
                    return True, exit
        elif wall[0][1] == wall[1][1] and exit[0][1] == exit[1][1] and wall[0][1] == exit[0][1]: #als muur en
            - exit horizontaal lopen
            if min(wall[0][0],wall[1][0])<min(exit[0][0],exit[1][0]) and max(wall[0][0],wall[1][0])>max(exit
                - [0][0],exit[1][0]):
                return True, exit
    return False, exit
def determinewalls(Walls,walls, exits):
    #makes a set of all walls (without exits)#
    while Walls.size !=0:
        wall = Walls[0]
        boolian, exit = exitinwall(exits, wall)
        if boolian == True:
            xwall = [wall[0][0],wall[1][0]]
            xexit = [exit[0][0],exit[1][0]]
            ywall = [wall[0][1],wall[1][1]]
            yexit = [exit[0][1],exit[1][1]]
            if xwall[0] == xwall[1]:
                wall1 = np.array([[[wall[np.argmin(ywall)][0],min(ywall)],[exit[np.argmin(yexit)][0],min(yexit)
                    - ]]])
                wall2 = np.array([[[exit[np.argmax(yexit)][0],max(yexit)],[wall[np.argmax(ywall)][0],max(ywall)
                    - ]]])
            else:
                wall1 = np.array([[[min(xwall),wall[np.argmin(xwall)][1]],[min(xexit),exit[np.argmin(xexit)
                    - ][1]]]])
                wall2 = np.array([[[max(xexit),exit[np.argmax(xexit)][1]],[max(xwall),wall[np.argmax(xexit)
                    - ][1]]]])
            Walls = np.append(Walls,wall1, axis = 0)
            Walls = np.append(Walls,wall2, axis = 0)
            Walls = np.delete(Walls,0, axis = 0)
        else:
            Walls = np.delete(Walls, 0, axis = 0)
            walls = np.append(walls, np.array([wall]),axis=0)
    return walls
##Functions to determine the new location of passengers##
def normalize(X, boolian = True ):
    #Normalizes a vector X#
    if boolian == True:
        X = X/np.linalg.norm(X) if np.linalg.norm(X) != 0 else np.array([0,0])
    return X
def centregroup(matrix, step, group):
    #Determines the centerpoint of a group#
    middle = np.zeros(2)
    for i in group:
        middle += matrix[i][step][:]
    middle = middle/len(group)
    return middle
def passengersaroundyou(matrix,i,step,people,boundary):
    #Returns the passengers around you and the amount of passengers around you with radius=boundary#
    passengers = []
    for j in [x for x in people if x!=i]:
        d = np.linalg.norm(matrix[i][step][:] - matrix[j][step][:])
        if d<boundary:
            passengers.append(j)
    return passengers, len(passengers)
def groupofpassenger(i,grouplst):
    #Returns the group of passenger i, if i isn't in a group it returns []#
    if i in sum(grouplst,[]):
        for group in grouplst:
            if i in group:
                return group
    else:
        return []
def peopledirection(matrix, i, step,comfortzone,R,people, grouplst,interpas,dt):
    #Returns the direction for passenger i based on passengers in his/her comfortzone#
    passengers, total = passengersaroundyou(matrix,i,step,people,comfortzone[i])
    group = groupofpassenger(i,grouplst)
    direction = np.zeros(2)
    for j in passengers:
        if np.linalg.norm(matrix[i][step][:] - matrix[j][step][:])<comfortzone[i]/max(1,total) + R[i]:
            if j not in group:
                interpas += dt
                direction += matrix[i][step][:]-matrix[j][step][:]
    return normalize(direction), interpas
```

```
def whichexit(matrix, step, point, exits, i, grouplst,ownwill):
    #Returns the exit passenger i will go to#
    if i in sum(grouplst,[]):
        for group in grouplst:
            if i in group:
                if sum(ownwill[group])>0:
                    index = np.argmin((ownwill[group]>0)==True)
                    point = group[index]
                else:
                    point = centregroup(matrix, step, group)
    distances = []
    for exit in exits:
        distances.append(distance(point, exit))
    index = np.argmin(distances); exit = exits[index]
    return exit, index

def exitdirection(matrix, step, point, i, exit,R, AB):
    #Returns the direction towards the preferred exit of passenger i#
    boundary = R[i]/np.linalg.norm(exit[1]-exit[0])
    t = tbar(point,exit)
    if t<=1-boundary and t>=boundary:
        direction = directionline(point,exit)
    else:
        direction = (exit[0]+exit[1])/2-point
    return normalize(direction)*AB[0]*np.exp(-AB[1]*distance(point,exit)/R[i])

def tbar(point, line):
    #Determines t such that the distance between point and line is the smallest#
    t = np.dot(point-line[0],line[1]-line[0])/(np.linalg.norm(line[0]-line[1])**2)
    return t
def directionline(point, line):
    #Returns the direction from point to linesegment#
    t = tbar(point, line)
    direction = line[0]+ min(max(t,0),1)*(line[1]-line[0])-point
    return direction
def distance(point, line):
    #Returns the minimum distance from point to line#
    d = np.linalg.norm(directionline(point,line))
    return d
def walldirection(point,i,walls,wallzone,R,interwall,r0,dt):
    #Returns the direction for passenger i based on walles and obstacles#
    wallboundary = wallzone+R[i]; Wall = []
    direction = np.zeros(2)
    for wall in walls:
        d = distance(point, wall)
        if d<wallboundary:
            Wall=wall
            d1 = distance(point + normalize(r0)*d,wall)
            r = directionline(point,wall)
            if d1<d:
                hoek = np.arccos((d-d1)/d)
                direction -= abs(1-hoek/(np.pi/2))*r
                interwall += dt
            else:
                direction -= 0.1*r
    return normalize(direction), interwall, Wall

def randomdirection():
    #Returns a random vector#
    xy = np.random.normal(0,0.5,2)
    return xy

def collidinggroups(matrix, i, step, grouplst, groupzone,R):
    #Determines whether members of a group are still close to each other#
    group = groupofpassenger(i,grouplst)
    d = np.linalg.norm(matrix[i][step][:]-centregroup(matrix, step, group)) - (groupzone+R[i])
    if d>0:
        matrix[i][step][:] += 1.1*d*normalize(centregroup(matrix, step, group)-matrix[i][step][:])
    return matrix
def collidingwalls(matrix, i, step, walls,R):
    #Determines whether passengers are colliding with walls and replaces them#
    for wall in walls:
        d = distance(matrix[i][step][:], wall)
        if d<R[i]:
            matrix[i][step][:] -= directionline(matrix[i][step][:],wall)*1.1
    return matrix[i][step][:]

def collidingpeople(matrix, i, step, people,R):
    #Returns the new location for passengers who are colliding with other passengers#
    for j in people:
        if i == j:
            return matrix
        else:
            d = np.linalg.norm(matrix[i][step][:]- matrix[j][step][:])
            if d<R[i]+R[j]:
                matrix[i][step][:] += (R[i]+R[j]-d)/1.9*normalize(matrix[i][step][:]-matrix[j][step][:])
                matrix[j][step][:] += (R[i]+R[j]-d)/1.9*normalize(matrix[j][step][:]-matrix[i][step][:])
    return matrix
def meandirection(matrix, i, step, people, directionmatrix, meanzone,speed):
    #Returns the mean direction for passenger i using passengers around him#
    direction = np.zeros(2)
    pas, tot = passengersaroundyou(matrix,i,step,people,meanzone)
    for j in pas:
        direction += directionmatrix[j]*speed[j]
    if tot>0:
        return direction/tot
    else:
        return direction
def poissoncalculate(numberofpassengers,entrances,poissonvalue,Rinit):
    #Returns a vector for each entrance with how many passenger alight every second using the Poisson
    - distribution#
    poisson = np.zeros(shape = (0,len(entrances)), dtype = int)
    while numberofpassengers>0:
        lst = np.random.poisson(poissonvalue, len(entrances));
        for i in range(len(lst)):
            lst[i] = min(lst[i],int(np.linalg.norm(entrances[i][0]-entrances[i][1])/(2*Rinit)))
            if sum(lst[:i+1])>=numberofpassengers and sum(lst[:i])<numberofpassengers:
                lst[i] = lst[i]-sum(lst[:i+1])+numberofpassengers
            elif sum(lst[:i+1])>=numberofpassengers:
                lst[i]=0
        value = sum(lst)
        numberofpassengers -= sum(lst)
        poisson = np.append(poisson, [lst],axis = 0)
    return poisson

def entrancelocation(matrix, step, entrances, poisson, peopleonplatform, dt, length, width,R,people, grouplst,
    - speed, grouptrue,ownwill):
```

```
    #Determines for every alighting passenger their initial position#
    for ingang in range(len(entrances)):
        instappers = poisson[step/int(1/dt),ingang]
        if entrances[ingang][0][0] == length or entrances[ingang][0][1] == width:
            normaal = normalize(np.array([-abs(entrances[ingang][1][1]-entrances[ingang][0][1]),-abs(entrances[
                -ingang][0][0] - entrances[ingang][1][0])]))
        else:
            normaal = normalize(np.array([abs(entrances[ingang][1][1]-entrances[ingang][0][1]),abs(entrances[
                -ingang][0][0] - entrances[ingang][1][0])]))
        if instappers>1 and grouptrue == True:
            grouplst.append(peopleonplatform[:instappers])
            ownwill[peopleonplatform[:instappers]] = 0
            speed[peopleonplatform[:instappers]] = sum(speed[peopleonplatform[:instappers]])/float(len(
                -peopleonplatform[:instappers]))
        for i in range(instappers):
            t = (2*i+1.)/(instappers*2.) + np.random.uniform(-0.1,0.1)
            x = entrances[ingang][0]+t*(entrances[ingang][1]-entrances[ingang][0])
            matrix[peopleonplatform[0]][step][:] = x+normaal * np.array([R[peopleonplatform[0]],R[
                -peopleonplatform[0]]])
            people.append(peopleonplatform[0])
            peopleonplatform.remove(peopleonplatform[0])
    return matrix, people, grouplst, speed

def exitreached(matrix, i, step, exits, people,R,numberofsteps, grouplst,evac,dt):
    #Checks whether passenger i has reached an exit#
    for exit in exits:
        if distance(matrix[i][step][:], exit)<R[i]:
            for group in grouplst:
                if i in group:
                    group.remove(i)
            people.remove(i)
            numberofsteps.append(step-np.argmax(matrix[i]>0)/2)
            if people == []:
                evac=step*dt
    return people, grouplst, evac, numberofsteps

def progression(count, total):
    #Shows the progression of the simulation#
    length = 60
    filled_len = int(round(length * count / float(total)))
    if filled_len>int(round(length *(count-1)/float(total))):
        bar = '='
        sys.stdout.write('%s' % (bar))
        sys.stdout.flush()

def direction(matrix,step,people,numberofpassengers,walls,obstacles,wallzone,R,exits,comfortzone,grouplst,
    -interpas, interwall,ownwill,dt,AB):
    #Determines the direction for each passenger on the platform based on every aspect#
    directionmatrix = np.zeros((numberofpassengers,2))
    for i in people:
        point = matrix[i][step][:]
        exit, index = whichexit(matrix, step, point, exits, i, grouplst,ownwill)
        r0 = exitdirection(matrix,step,point,i,exit,R,AB)
        r1, interpas = peopledirection(matrix,i,step,comfortzone,R,people,grouplst, interpas,dt)
        r2, interwall, Wall = walldirection(point,i,np.append(walls,obstacles,axis=0),wallzone,R, interwall, r0
            -,dt)
        r3 = randomdirection()
        direction = r0+r1+r2+r3
        directionmatrix[i] = normalize(direction,np.linalg.norm(direction)>1)
        if np.linalg.norm(direction)<0.5 and Wall !=[]:
            if np.linalg.norm(Wall[0]-point)<np.linalg.norm(Wall[0][1]-point):
                directionmatrix[i] = normalize(Wall[0]-Wall[1])
            else:
                directionmatrix[i] = normalize(Wall[1]-Wall[0])
    return directionmatrix, interpas, interwall

##Simulation##

def determinematrix(walls,entrances,poisson,peopleonplatform,dt,length,width,R,people,numberofpassengers,
    -obstacles,wallzone,exits,comfortzone,speed,numberofsteps,a,grouplst,groupzone, grouptrue, meanzone,
    -ownwill, gemiddeldevalue,AB):
    #Here everything is combined, passengers alight, walk and depart from the platform. It returns the response
        - variables
    matrix = np.zeros((numberofpassengers,1,2)); step = 0; interpas = 0; interwall = 0; evac=0
    while people != [] or peopleonplatform !=[]:
        matrix = np.append(matrix, np.zeros((numberofpassengers, 1, 2)),axis=1);
        if peopleonplatform != [] and step%(int(1/dt)) == 0: matrix, people, grouplst, speed = entrancelocation
            -(matrix,step, entrances, poisson, peopleonplatform, dt, length, width,R,people,grouplst, speed,
            -grouptrue, ownwill)
        directionmatrix, interpas, interwall = direction(matrix,step,people,numberofpassengers,walls,obstacles,
            -wallzone,R,exits,comfortzone,grouplst, interpas, interwall,ownwill,dt,AB)
        for i in people[:]:
            vect = (1-gemiddeldevalue)*directionmatrix[i]*speed[i]+gemiddeldevalue*meandirection(matrix,i,step,
                -people,directionmatrix, meanzone,speed)
            matrix[i][step+1][:] = matrix[i][step][:] + dt*vect
            people, grouplst, evac, numberofsteps = exitreached(matrix, i, step+1, exits, people,R,
                -numberofsteps,grouplst,evac,dt)
            matrix = collidingpeople(matrix, i, step+1, people,R)
            matrix[i][step+1][:] = collidingwalls(matrix, i, step+1, np.append(walls,obstacles, axis = 0),R)
        if grouptrue == True:
            for i in sum(grouplst,[]):
                matrix = collidinggroups(matrix,i,step+1,grouplst,groupzone,R)
        step+=1
    return dt*sum(numberofsteps)/float(len(numberofsteps)), evac, interpas, interwall
```

B.2 Transfer Model

Here the main program (B.2.1) uses the functions from B.1.2 and some extra functions from B.2.2 (where functions with the same name need to be replaced), to do 10 simulations with 100 passengers per train. The output is a matrix with response variables T_{exit}, T_{evac}, the dwell time for every train, the interaction time with passengers and the interaction time with walls and objects. Initial values can be adjusted to adapt the characteristics of passengers or the dimensions of the train and platform. Further, the number of trains and passengers can be changed and the entire timetable can be adjusted as well.

B.2.1 Main Program

```
import numpy as np
import matplotlib.pyplot as plt
import sys
import time
from Transfermodel import poissoncalculate, initialvalues, progression, determinematrix, animation
##Initial Values##
#Platform Choice#
empty = np.zeros((0,2,2))
A = np.array([[[10,  4],[10,  6]],[[15,  4],[15,  6]],[[10,  4],[15,  4]],[[10,  6],[15,  6]],[[40,  4],[40,
    - 6]],[[35,  4],[35,  6]],[[40,  4],[35,  4]],[[40,  6],[35,  6]],[[19,  5],[23,  5]],[[27 , 5],[31 , 5]]])
B = np.array([[[5 ,5],[15,  5]],[[45,  5],[35,  5]],[[20 , 4],[30,  4]],[[20,  6],[30,  6]],[[20,  4],[20,
    - 6]],[[30,  4],[30,  6]]])
D = np.array([[[9 , 0],[4 ,3]],[[ 9, 10],[4, 7]],[[17,  0],[12,  3]],[[17, 10],[12,  7]],[[33,  0],[38,
    - 3]],[[33, 10],[38,  7]],[[41,  0],[46,  3]],[[41, 10],[46,  7]]])
platform = empty
#These values can be adjusted#
width = 10.          #metre
length = 50.         #metre
dt = 0.25            #second
sprinter0 = np.array
    - ([[[4,0],[6,0]],[[12,0],[14,0]],[[20,0],[22,0]],[[28,0],[30,0]],[[36,0],[38,0]],[[44,0],[46,0]]])
sprinter1 = np.array([[[4,width],[6,width]],[[12,width],[14,width]],[[20,width],[22,width]],[[28,width],[30,
    - width]],[[36,width],[38,width]],[[44,width],[46,width]]])
entrances0 = np.concatenate(([sprinter0],[sprinter0]))
entrances1 = np.concatenate(([sprinter1],))
exits = np.array([[[0,3],[0,7]],[[length,3],[length,7]]])
obstacles = platform
numberofpassengers = np.array([[100, 100], [100],[200]])
arrival = np.array([[0, 120], [60]])
departure = np.array([[40, 160], [100]])
numberofsimulations = 10
poissonvalue = 0.5   #/second
speedinit = 1.4      #metre/second
Rinit = 0.3          #metre
wallzone = 0.5       #metre
comfortzoneinit = 2.#metre
groupzone = 1.5      #metre
meanzone = 3.        #metre
meanvalue = 0.1
grouptrue = False
ownwilltrue = False
transfertrue = True
AB = [1.1, 10**-3]
##Main Program##
entrances = [entrances0,entrances1]
timetable = np.array([arrival,departure])
edges = np.array([[0.,0.],[0.,width],[length,width],[length,0.]])
walls = np.array([[edges[0],edges[1]],[edges[1],edges[2]],[edges[2],edges[3]],[edges[3],edges[0]]])
time1 = time.time()
X = np.zeros((numberofsimulations,23+len(entrances0)+len(entrances1)))
for a in range(numberofsimulations):
    progression(a,numberofsimulations)
    numberofsteps = []
    parenlst = []
    dwelltime = [[],[]]
    for i in range(len(arrival)):
        for j in range(len(arrival[i])):
            dwelltime[i].append(departure[i][j] - arrival[i][j])
    speed,R,peopleonplatform,comfortzone,ownwill,transfer,train,people,whichtrain = initialvalues(
        - numberofpassengers,comfortzoneinit,Rinit,speedinit,exits,ownwilltrue,transfertrue)
    poisson,timetable = poissoncalculate(numberofpassengers,entrances,poissonvalue,Rinit,timetable,exits)
    mean, evac, interpas, interwall, matrix,dwelltime = determinematrix(walls,entrances,poisson,
        - peopleonplatform,dt,length,width,R,people,numberofpassengers,obstacles,wallzone,exits,comfortzone,
        - speed,numberofsteps,a,parenlst, groupzone, grouptrue,meanzone,ownwill,meanvalue,transfer,train,
        - timetable,whichtrain,dwelltime,AB)
    X[a,:] = np.concatenate((np.array(sum(dwelltime,[]))),np.array([mean, interpas, interwall, evac, length,
        - width,dt,comfortzoneinit,poissonvalue,speedinit,Rinit,wallzone,groupzone,meanzone,meanvalue, AB[0],
        - AB[1], sum(map(sum,numberofpassengers)),len(exits), len(np.concatenate((entrances[0],entrances[1])))),
        - len(obstacles),grouptrue,ownwilltrue])))
print 'It took ' + str(int(time.time()-time1)/60) +' minutes and ' + str(int(time.time()-time1)%60) + ' seconds
    - to determine the matrix.'
##Save Data##
string = ''
for i in range(len(obstacles)):
    string += str(obstacles[i][0])+str(obstacles[i][1])
if string == '':
    string = 'no'
np.savetxt(str(versie) +"-Transfer.csv", X, delimiter=",")
animation(length, width, walls, exits, entrances, obstacles, evac, sum(map(sum,numberofpassengers)), matrix,R,
    - [],dt,timetable,transfer)
```

B.2.2 Functions

```
def initialvalues(numberofpassengers, comfortzoneinit,Rinit, speedinit, exits,ownwilltrue,transfertrue):
    #determines the intial values for all passengers#
    numberofpassengers = sum(map(sum,numberofpassengers))
    v,l = normaltolognormal(speedinit,speedinit/5); speed = np.random.lognormal(v,l,numberofpassengers)
    v,l = normaltolognormal(Rinit,0.05); R = np.random.lognormal(v,l,numberofpassengers) #meter
    peopleonplatform0 = [range(numberofpassengers[0][i]) for i in range(len(numberofpassengers[0]))];
        - peopleonplatform1 = [range(numberofpassengers[1][i]) for i in np.arange(len(numberofpassengers[1]))];
    peopleonplatform = [peopleonplatform0,peopleonplatform1,[range(numberofpassengers[2][0])]]
    v,l = normaltolognormal(comfortzoneinit,0.5);comfortzone = np.random.lognormal(v,l,numberofpassengers)
    prob = np.random.uniform(0,10,numberofpassengers)
    ownwill = (prob<2)*np.random.randint(1,1+len(exits),numberofpassengers) if ownwilltrue else np.zeros(
        - numberofpassengers)
    transfer = (prob>7) if transfertrue else np.zeros(numberofpassengers)
    train = np.zeros(numberofpassengers) if transfertrue else []
    people = []
    train[-numberofpassengers[2][0]:] = (prob[-numberofpassengers[2][0]:]>5); transfer[-numberofpassengers
        - [2][0]:] = np.ones(numberofpassengers[2][0])
    whichtrain = np.zeros((2,numberofpassengers))
    return speed,R,peopleonplatform,comfortzone,ownwill,transfer,train,people,whichtrain

def areaexit(exit,point,whichtrain,i):
    #Returns the area a transfer passengers wants to go to and whether he has arrived or not#
    area = np.zeros((2,4,2)); hoogte = 1.4;
```

```
    plusmin = 1 if exit[0][1] == 0 else -1
    area[0] = np.array([exit[0],2*exit[0]-exit[1],exit[0]+plusmin*np.array([0,hoogte]),2*exit[0]-exit[1]+
        plusmin*np.array([0,hoogte])])
    area[1] = np.array([exit[1],2*exit[1]-exit[0],exit[1]+plusmin*np.array([0,hoogte]),2*exit[1]-exit[0]+
        plusmin*np.array([0,hoogte])])
    if whichtrain[1][i] == 0:
        index = np.random.randint(0,2)
        area = area[index]
        whichtrain[1][i] = index+1
    elif whichtrain[1][i] == 1:
        area = area[0]
    else:
        area = area[1]
    if all(point>np.min(area,axis=0)) and all(point<np.max(area,axis=0)):#blijf staan
        arrived = True
    else:
        arrived = False
    return area, arrived, whichtrain

def exitdirection(matrix, step, point, i, exit,R, groupslst,AB):
    #Returns the direction towards the preferred exit of passenger i#
    if exit.shape !=(2,):
        boundary = R/np.linalg.norm(exit[1]-exit[0])
        t = tbar(point,exit)
        if t<=1-boundary and t>=boundary:
            direction = directionline(point,exit)
        else:
            direction = (exit[0]+exit[1])/2-point
        return normalize(direction)*AB[0]*np.exp(-AB[1]*distance(point,exit)/R)
    else:
        return normalize(exit-point)*AB[0]*np.exp(-AB[1]*np.linalg.norm(exit-point)/R)

def walldirection(point,walls,wallzone,R,interwall,r0,dt,exit):
    #Returns the direction for passenger i based on walles and obstacles#
    wallboundary = wallzone+R; Wall = []
    direction = np.zeros(2)
    if exit.shape == (2,):
        if np.linalg.norm(exit-point)<wallboundary+2*R:
            return np.zeros(2), interwall, Wall
    else:
        if distance(point,exit)<wallboundary+2*R:
            return np.zeros(2), interwall, Wall
    for wall in walls:
        d = distance(point, wall)
        if d<wallboundary:
            Wall=wall
            d1 = distance(point + normalize(r0)*d,wall)
            r = directionline(point,wall)
            if d1<d:
                angle = np.arccos((d-d1)/d)
                direction -= abs(1-angle/(np.pi/2))*r
                interwall += dt
            else:
                direction -= 0.1*r
    return normalize(direction), interwall, Wall

def poissoncalculate(numberofpassengers,entrances,poissonvalue,Rinit,times,exits):
    #Returns a vector for each entrance with how many passenger alight every second using the Poisson
        distribution#
    p0 = [[] for _ in range(len(entrances[0]))]; p1 = [[] for _ in range(len(entrances[1]))]; p2 = [[]]; ast =
        copy.deepcopy(numberofpassengers); p = [p0,p1,p2]
    for platform in range(len(entrances)):
        for train in range(len(p[platform])):
            for tijd in range(times[0][platform][train]):
                p[platform][train].append([0]*len(entrances[platform][train]))
            while ast[platform][train]>0:
                lst = np.random.poisson(poissonvalue, len(entrances[platform][train]))
                for i in range(len(lst)):
                    lst[i] = min(lst[i],int(np.linalg.norm(entrances[platform][train][i][0]-entrances[platform
                        ][train][i][1])/(2*Rinit)))
                    if sum(lst[:i+1])>=ast[platform][train] and sum(lst[:i])<ast[platform][train]:
                        lst[i] = lst[i]-sum(lst[:i+1])+ast[platform][train]
                    elif sum(lst[:i+1])>=ast[platform][train]:
                        lst[i]=0
                ast[platform][train] -= sum(lst)
                p[platform][train].append(list(lst))
            if len(p[platform][train])>times[1][platform][train]:
                times[1][platform][train]+=5
    while ast[2][0]>0:
        lst = np.random.poisson(2*poissonvalue, len(exits))
        for i in range(len(lst)):
            lst[i] = min(lst[i],int(np.linalg.norm(exits[i][0] - exits[i][1])/(2*Rinit)))
            if sum(lst[:i+1])>=ast[2][0] and sum(lst[:i])<ast[2][0]:
                lst[i] = lst[i]-sum(lst[:i+1])+ast[2][0]
            elif sum(lst[:i+1])>=ast[2][0]:
                lst[i]=0
        ast[2][0] -= sum(lst)
        p[2][0].append(list(lst))
    return p,times

def entrancelocation(matrix, step, entrances, poisson, peopleonplatform,numberofpassengers,dt,length, width,R,
    people, groupslst, speed, grouptrue,exits,train):
    #Determines for every alighting passenger their initial position#
    for platform in range(len(peopleonplatform)): #2
        for train in range(len(peopleonplatform[platform])): #0
            if peopleonplatform[platform][train]!=[]:
                if platform == 2:
                    passageways = exits
                else:
                    passageways = entrances[platform][train]
                for ingang in range(len(passageways)):
                    alighters = poisson[platform][train][int(step*dt)][ingang]
                    if alighters>0:
                        toevoeg = [0,sum(numberofpassengers[0]),sum(np.sum(numberofpassengers[:2]))]
                        if len(train)>0 and platform!=2:train[np.array(peopleonplatform[platform][train][:
                            alighters])+sum(numberofpassengers[platform][:train])+toevoeg[platform]] = np.
                            zeros(alighters) if entrances[platform][train][ingang][0][1] ==0 else np.ones(
                            alighters)
                        if passageways[ingang][0][0] == length or passageways[ingang][0][1] == width:
                            normaal = normalize(np.array([-abs(passageways[ingang][1][1]-passageways[ingang
                                ][0][1]),-abs(passageways[ingang][0][0] - passageways[ingang][1][0])]))
                        else:
                            normaal = normalize(np.array([abs(passageways[ingang][1][1]-passageways[ingang
                                ][0][1]),abs(passageways[ingang][0][0] - passageways[ingang][1][0])]))
                        if alighters>1 and grouptrue == True:
                            groupslst.append(peopleonplatform[platform][train][:alighters])
```

```
                                speed[peopleonplatform[platform][train][:alighters]] = sum(speed[peopleonplatform[
                                    platform][train][:alighters]])/float(len(peopleonplatform[platform][train][:
                                    alighters]))
                            for i in range(alighters):
                                t = (2*i+1.)/(alighters*2.) + np.random.uniform(-0.1,0.1)
                                x = passageways[ingang][0] + t*(passageways[ingang][1]-passageways[ingang][0])
                                matrix[peopleonplatform[platform][train][0]+sum(numberofpassengers[platform][:train
                                    ])+toevoeg[platform]][step][:] = x+normaal * np.array([R[peopleonplatform[
                                    platform][train][0]+sum(numberofpassengers[platform][:train])+toevoeg[
                                    platform]],R[peopleonplatform[platform][train][0]+sum(numberofpassengers[
                                    platform][:train])+toevoeg[platform]]])
                                people.append(peopleonplatform[platform][train][0]+sum(numberofpassengers[platform
                                    ][:train])+toevoeg[platform])
                                peopleonplatform[platform][train].remove(peopleonplatform[platform][train][0])
    return matrix, people, groupslst, speed, train

def walk(matrix,step,people,numberofpassengers,walls,obstacles,wallzone,R,exit,comfortzone,groupslst, interpas,
    interwall,ownwill,dt,i,AB):
    #Determines the direction for each passenger on the platform based on every aspect#
    point = matrix[i][step][:]
    r0 = exitdirection(matrix,step,point,i,exit,R[i],groupslst,AB)
    r1, interpas = peopledirection(matrix,i,step,comfortzone,R,people,groupslst, interpas,dt)
    r2, interwall, Wall = walldirection(point,np.append(walls,obstacles,axis=0),wallzone,R[i], interwall, r0,dt
        ,exit)
    r3 = randomrichting()
    direction = r0+r1+r2+r3
    if np.linalg.norm(direction)>1:
        return normalize(direction), interpas, interwall
    elif np.linalg.norm(direction)<0.5 and Wall !=[]:
        if np.linalg.norm(Wall[0]-point)<np.linalg.norm(Wall[0][1]-point):
            return normalize(Wall[0]-Wall[1]), interpas, interwall
        else:
            return normalize(Wall[1]-Wall[0]), interpas, interwall
    else:
        return direction, interpas, interwall

def wexit(point,i,train,length,whichtrain):
    #Returns the exit of a train, that has not arrived yet, for passenger i#
    if whichtrain[0][i]==0:
        x=np.random.normal(point[0],50)
        while x<0 or x>length:
            x=np.random.normal(point[0],50)
        index = np.argmin(abs(train[:,:,0][:,0]-x))
        exit = train[index]
        whichtrain[0][i] = index+1
    else:
        exit = train[whichtrain[0][i]-1]
    return exit, whichtrain

def direction(matrix,step,people,numberofpassengers,walls,obstacles,wallzone,R,exits,comfortzone,groupslst,
    interpas, interwall,ownwill,dt,poisson,transfer,train,entrances,Trains,whichtrain,length,AB):
    #Determines what the goal of every passenger is and uses 'walk' to calculate the direction#
    directionmatrix = np.zeros((sum(map(sum,numberofpassengers)),2))
    for i in [x for x in people if transfer[x]==False]:
        point = matrix[i][step][:]
        exit,index = whichexit(matrix,step,point,exits,i,groupslst,ownwill)
        directionmatrix[i], interpas,interwall = walk(matrix,step,people,numberofpassengers,walls,obstacles,
            wallzone,R,exit,comfortzone,groupslst, interpas, interwall,ownwill,dt,i,AB)
    for i in [x for x in people if transfer[x]==True]:
        point = matrix[i][step][:]; trainotherside = Trains[int(train[i]+1)%2][0]
        if trainotherside==[]:
            if Trains[int(train[i]+1)%2][1] == []:
                exit,whichtrain = wexit(point, i, entrances[int(train[i]+1)%2][0], length, whichtrain)
                area,arrived,whichtrain = areaexit(exit,point,whichtrain,i)
                if arrived: directionmatrix[i] = np.zeros(2)
                else:directionmatrix[i],interpas,interwall=walk(matrix,step,people,numberofpassengers,walls,
                    obstacles,wallzone,R,centregroup(area),comfortzone,groupslst, interpas, interwall,ownwill
                    ,dt,i,AB)
            else:
                if (Trains[int(train[i]+1)%2][1][-1]+1)==len(entrances[int(train[i]+1)%2]):
                    exit,index = whichexit(matrix,step,point,exits,i,groupslst,ownwill)
                    directionmatrix[i], interpas,interwall = walk(matrix,step,people,numberofpassengers,walls,
                        obstacles,wallzone,R,exit,comfortzone,groupslst, interpas, interwall,ownwill,dt,i,AB)
                else:
                    exit,whichtrain = wexit(point, i, entrances[int(train[i]+1)%2][Trains[int(train[i]+1)
                        %2][1][-1]+1], length, whichtrain)
                    area,arrived,whichtrain = areaexit(exit,point,whichtrain,i)
                    if arrived: directionmatrix[i] = np.zeros(2)
                    else:directionmatrix[i],interpas,interwall=walk(matrix,step,people,numberofpassengers,walls
                        ,obstacles,wallzone,R,centregroup(area),comfortzone,groupslst, interpas, interwall,
                        ownwill,dt,i,AB)
        else:
            exit,index = whichexit(matrix,step,point,entrances[int(train[i]+1)%2][trainotherside[0]],i,
                groupslst,ownwill)
            if sum([row[index] for row in poisson[int(train[i]+1)%2][trainotherside[0]][int(step*dt-2):]])==0:
                directionmatrix[i],interpas,interwall = walk(matrix,step,people,numberofpassengers,walls,
                    obstacles,wallzone,R,exit,comfortzone,groupslst, interpas, interwall,ownwill,dt,i,AB)
            else:
                area,arrived,whichtrain = areaexit(exit,point,whichtrain,i)
                if arrived: directionmatrix[i] = np.zeros(2)
                else:directionmatrix[i],interpas,interwall=walk(matrix,step,people,numberofpassengers,walls,
                    obstacles,wallzone,R,centregroup(area),comfortzone,groupslst, interpas, interwall,ownwill
                    ,dt,i,AB)
    return directionmatrix, interpas, interwall

def whichtrains(entrances,arrival,peopleonplatform,step,Trains,dt,dwelltime,matrix):
    #Here it is determined whether trains have to arrive or have to depart depending on time#
    for platform in range(len(entrances)):
        for train in range(len(entrances[platform])):
            if train in Trains[platform][0]: #train is nu op het platform
                if peopleonplatform[platform][train]==[] and dt*step>=arrival[1][platform][train] and np.sum(
                    abs(matrix[np.nonzero(matrix[:,step][:,1]),step][:,:,1]-entrances[platform][train
                    ][0,0,1])<1)==0:
                    arrival[1][platform][train] = step*dt
                    dwelltime[platform][train] = dt*step-arrival[0][platform][train]
                    Trains[platform][0].remove(train)
                    Trains[platform][1].append(train)
            elif train not in Trains[platform][1]: #train moet nog komen
                if Trains[platform][0] ==[] and dt*step>=arrival[0][platform][train]:
                    Trains[platform][0].append(train)
    return Trains,dwelltime,arrival

##Simulation##

def determinematrix(walls,entrances,poisson,peopleonplatform,dt,length,width,R,people,numberofpassengers,
    obstacles,wallzone,exits,comfortzone,speed,numberofsteps,a,groupslst,groupzone, grouptrue, meanzone,
```

```
    ownwill, meanvalue, transfer, train, arrival, whichtrain, dwelltime, AB):
    #Here everything is combined, passengers alight, walk, transfer and depart from the platform. It returns
        the response variables
    matrix = np.zeros((sum(map(sum, numberofpassengers)),1,2));
    step = 0; interpas = 0; interwall = 0; evac=0; Trains = [[[],[]],[[],[]]]
    while people != [] or np.sum(np.sum(peopleonplatform)) !=0:
        matrix = np.append(matrix, np.zeros((sum(map(sum, numberofpassengers)), 1, 2)),axis=1);
        Trains, dwelltime, Tijden = whichtrains(entrances, arrival, peopleonplatform, step, Trains, dt, dwelltime,
            matrix)
        if np.sum(np.sum(peopleonplatform)) != 0 and step*dt%1 == 0: matrix, people, groupslst, speed, train =
            entrancelocation(matrix, step, entrances, poisson, peopleonplatform, numberofpassengers, dt, length
            , width, R, people, groupslst, speed, grouptrue, exits, train)
        directionmatrix, interpas, interwall = direction(matrix, step, people, numberofpassengers, walls, obstacles,
            wallzone, R, exits, comfortzone, groupslst, interpas, interwall, ownwill, dt, poisson, transfer, train,
            entrances, Trains, whichtrain, length, AB)
        for i in people[:]:
            vect = (1-meanvalue)*directionmatrix[i]*speed[i]+meanvalue*meandirection(matrix, i, step, people,
                directionmatrix, meanzone, speed)
            matrix[i][step+1][:] = matrix[i][step][:] + dt*vect
            matrix = collidingpeople(matrix, i, step+1, people, R)
            if transfer[i]:
                if len(Trains[int(train[i]+1)%2][1]) == len(entrances[int(train[i]+1)%2]):
                    people, groupslst, evac, numberofsteps = exitreached(matrix, i, step+1, exits, people, R,
                        numberofsteps, groupslst, evac, dt, transfer, length, width)
                elif Trains[int(train[i]+1)%2][0] != []:
                    people, groupslst, evac, numberofsteps = exitreached(matrix, i, step+1, entrances[int(train
                        [i]+1)%2][Trains[int(train[i]+1)%2][0][0]], people, R, numberofsteps, groupslst, evac, dt,
                        transfer, length, width)
            else:
                people, groupslst, evac, numberofsteps = exitreached(matrix, i, step+1, exits, people, R,
                    numberofsteps, groupslst, evac, dt, transfer, length, width)
        for i in people[:]:
            matrix[i][step+1][:] = collidingwalls(matrix, i, step+1, np.append(walls, obstacles, axis = 0), R)
        if grouptrue == True:
            for i in sum(groupslst,[]):
                matrix = collidinggroups(matrix, i, step+1, groupslst, groupzone, R)
        step+=1
        print step
        if step == 2000:
            break
    return dt*sum(numberofsteps)/float(len(numberofsteps)), evac, interpas, interwall, matrix, dwelltime

##Animation##

def animation(length, width, walls, exits, entrances, obstacles, evac, numberofpassengers, matrix, R, parenani,
    dt, times, transfer):
    plt.axis([-2, length+2, -2, width+2]) #randen
    for wall in walls:
        plt.plot([wall[0][0], wall[1][0]], [wall[0][1], wall[1][1]], linewidth=2, c='k')
    for exit in exits:
        plt.plot([exit[0][0], exit[1][0]], [exit[0][1], exit[1][1]], linewidth=5, c='b')
    for obstacle in obstacles:
        plt.plot([obstacle[0][0], obstacle[1][0]], [obstacle[0][1], obstacle[1][1]], linewidth = 4, c = 'r')
    lst = []
    for step in range(int(evac/dt)+1):
        for k in range(len(lst)):
            x = lst[k]
            x.remove()
        lst = []
        for platform in range(len(entrances)):
            for train in range(len(entrances[platform])):
                if times[0][platform][train]<step*dt and times[1][platform][train]>step*dt:
                    for ingang in entrances[platform][train]:
                        line = plt.Line2D([ingang[0][0], ingang[1][0]], [ingang[0][1], ingang[1][1]], linewidth=4,
                            c='r')
                        plt.gca().add_line(line)
                        lst.append(line)
        plt.title(str(int(step*dt)) + ' seconds')
        for i in range(numberofpassengers):
            if np.array_equal(matrix[i][step][:], np.array([0,0])) == False:
                color = 'r' if i in sum(parenani,[]) else 'b'
                color = 'r' if transfer[i] else 'b'
                p = plt.Circle(matrix[i][step][:], radius = R[i], color=color)
                lst.append(p)
                plt.gca().add_patch(p)
        fig = plt.gcf()
#       fig.savefig('Animatie'+ str(step) + '.png')
        plt.pause(0.0001)
    plt.show()
```

Appendix C

R code

Here the code in R code is showed for only a few statistical analysis due to the fact that much of the analysis is done in the same way. For the Passenger Model we included the analysis of the time step and the platform design. For the Transfer Model we included also the analysis of the platform design.

C.1 Passenger Model

C.1.1 Time Step

```
setwd("~/Documenten/Thesis/RUN4/Time Step")
M8 = read.csv("dt=0.125.csv",header = FALSE,sep=",");M8=M8[,c(1,4)]
M4 = read.csv("dt=0.25.csv",header = FALSE,sep=",");M4=M4[,c(1,4)]
M2 = read.csv("dt=0.5.csv",header = FALSE,sep=",");M2=M2[,c(1,4)]
M1 = read.csv("dt=1.0.csv",header = FALSE,sep=",");M1=M1[,c(1,4)]
names(M8) = c("exit","evac");names(M4) = c("exit","evac"); names(M2) = c("exit","evac"); names(M1) = c("exit","
    evac")
######Probability Density Functions#########
plot(lwd = 3, cex.lab = 1.5,cex.axis = 1.2,density(M8$exit),col="red", xlim=c(min(M8$exit),max(M1$exit)),xlab=
    expression(paste("Travel time to reach exit ",T[exit]," [s]")),ylab="Probability Density",main="
    Probability Density Function for different Timesteps"); lines(lwd = 3,density(M4$exit),col="blue");lines(
    lwd = 3,density(M2$exit),col="green");lines(lwd = 3,density(M1$exit),col="black")
legend("topright",lty=1,lwd=3,cex=1.5,bty="n",seg.len=2, legend=c(expression(paste(Delta,t, "=0.125")),
    expression(paste(Delta,t, "=0.25")),expression(paste(Delta,t,"=0.5")),expression(paste(Delta,t,"=1"))),
    col=c("red", "blue","green","black"))
plot(lwd = 3, cex.lab = 1.5,cex.axis = 1.2,density(M8$evac),col="red",xlim=c(min(M8$evac),max(M1$evac)),ylim=c
    (0,0.1), xlab=expression(paste("Time for which the platform is empty,",T[evac]," [s]")),ylab="Probability
     Density",main="Probability Density Function for different Timesteps");lines(lwd = 3,density(M4$evac),col
    ="blue") ;lines(lwd = 3,density(M2$evac),col="green");lines(lwd = 3,density(M1$evac),col="black")
legend("topright",lty=1,lwd=3,cex=1.5,bty="n",seg.len=2,legend=c(expression(paste(Delta,t, "=0.125")),
    expression(paste(Delta,t, "=0.25")),expression(paste(Delta,t,"=0.5")),expression(paste(Delta,t,"=1"))),
    col=c("red", "blue","green","black"))
######Cumulative Density Functinos#########
cdf.m1.sec.x = sort(M1$exit);cdf.m2.sec.x = sort(M2$exit);cdf.m4.sec.x = sort(M4$exit);cdf.m8.sec.x = sort(M8$
    exit);cdf.m1.evac.x = sort(M1$evac);cdf.m2.evac.x = sort(M2$evac);cdf.m4.evac.x = sort(M4$evac);cdf.m8.
    evac.x = sort(M8$evac)
par(mar=c(5,6,5,1)+.1)
plot(lwd = 3,cex.lab = 1.5,cex.axis = 1.2,cdf.m8.sec.x,seq(0.001,1,0.001),col="red",type="l",xlim = c(10,max(
    cdf.m1.sec.x)), xlab=expression(T[exit]),ylab="Cumulative Probability",main="");lines(lwd = 3,cdf.m4.sec.
    x,seq(0.001,1,0.001),col="blue");lines(lwd = 3,cdf.m2.sec.x,seq(0.001,1,0.001),col="green");lines(lwd =
    3,cdf.m1.sec.x,seq(0.001,1,0.001),col="black")
legend("topleft", lty=1,lwd=3,cex=1.5,seg.len=2,bty="n", legend=c(expression(paste(Delta,t, "=0.125")),
    expression(paste(Delta,t, "=0.25")),expression(paste(Delta,t,"=0.5")),expression(paste(Delta,t,"=1"))),
    col=c("red", "blue","green","black"))
plot(lwd = 3,cex.lab = 1.5,cex.axis = 1.2,cdf.m8.evac.x,seq(0.001,1,0.001),col="red",type="l",xlim = c(40,max(
    cdf.m1.evac.x)),xlab=expression(T[evac]),ylab="Cumulative Probability",main="");lines(lwd = 3,cdf.m4.evac
    .x,seq(0.001,1,0.001),col="blue");lines(lwd = 3,cdf.m2.evac.x,seq(0.001,1,0.001),col="green");lines(lwd =
    3,cdf.m1.evac.x,seq(0.001,1,0.001),col="black")
legend("topleft", lty=1,lwd=3,cex=1.5,seg.len=2,bty="n", legend=c(expression(paste(Delta,t, "=0.125")),
    expression(paste(Delta,t, "=0.25")),expression(paste(Delta,t,"=0.5")),expression(paste(Delta,t,"=1"))),
    col=c("red", "blue","green","black"))
######Boxplots and tests#######
boxplot(M1$exit,M2$exit,M4$exit,M8$exit);boxplot(M1$evac,M2$evac,M4$evac,M8$evac)
ks.test(M8$exit,M4$exit);ks.test(M8$evac,M4$evac)
ks.test(M4$exit,M2$exit);ks.test(M4$evac,M2$evac)
ks.test(M2$exit,M1$exit);ks.test(M2$evac,M1$evac)
```

C.1.2 Platform Design

```
setwd("~/Documenten/Thesis/RUN4/Model")
AA = read.csv("EmptyPlatform.csv",header = FALSE,sep=",")
A = read.csv("PlatformA.csv",header = FALSE,sep=",")
B = read.csv("PlatformB.csv",header = FALSE,sep=",")
C = read.csv("PlatformC.csv",header = FALSE,sep=",")
D = read.csv("PlatformD.csv",header = FALSE,sep=",")
AA=AA[,1:4];A=A[,1:4];B=B[,1:4];C=C[,1:4];D=D[,1:4]
names(AA) = c("exit","interpas","interwall","evac"); names(A) = c("exit","interpas","interwall","evac");names(B
    ) = c("exit","interpas","interwall","evac");names(C) = c("exit","interpas","interwall","evac");names(D) =
     c("exit","interpas","interwall","evac")
```

```
########Distribution for Texit and Tevac#########
plot(lwd=3, cex.lab = 2,cex.axis = 1.2,density(AA$exit),xlim=c(11,30),col="blue",xlab =expression(T[exit]),ylab
    = "Probability Density",main="");lines(lwd=3,density(A$exit),col="red");lines(lwd=3,density(B$exit),col=
    "green");lines(lwd=3,density(C$exit),col="black");lines(lwd=3,density(D$exit),col="orange")
legend("top",lty=1,lwd=3,cex=1.5,bty="n",seg.len=2, legend=c("Empty Platform","Platform A","Platform B","
    Platform C","Platform D"),col=c("blue", "red","green","black","orange"))

plot(lwd=3, cex.lab = 2,cex.axis = 1.2,density(AA$evac),xlim=c(30,100),col="blue",xlab=expression(T[evac]),ylab
    ="Probability Density",main="");lines(lwd=3,density(A$evac),col="red");lines(lwd=3,density(B$evac),col="
    green");lines(lwd=3,density(C$evac),col="black");lines(lwd=3,density(D$evac),col="orange")
legend("topright",lty=1,lwd=3,cex=1.5,bty="n",seg.len=2, legend=c("Empty Platform","Platform A","Platform B","
    Platform C","Platform D"),col=c("blue", "red","green","black","orange"))

ks.test(AA$exit,B$exit); ks.test(AA$evac,B$evac)
qqplot(AA$exit,D$exit)

par(mar=c(5,6,5,1)+.1)
plot(lwd=3, cex.lab = 2,cex.axis = 1.2,xlim = c(12,16),sort(AA$exit),seq(0.001,1,0.001),'l',col="blue",xlab =
    expression(T[exit]),ylab="Cumulative Density");lines(lwd=3,sort(A$exit),seq(0.001,1,0.001),col="red");
    lines(lwd=3,sort(B$exit),seq(0.001,1,0.001),col="green");lines(lwd=3,sort(D$exit),seq(0.001,1,0.001),col=
    "orange")
legend("bottomright",lty=1,lwd=3,cex=1.5,bty="n",seg.len=2, legend=c("Empty Platform","Platform A","Platform B"
    ,"Platform D"),col=c("blue", "red","green","orange"))

plot(lwd=3, cex.lab = 2,cex.axis = 1.2,sort(AA$evac),seq(0.001,1,0.001),'l',col="blue",xlab =expression(T[evac
    ]),ylab = "Cumulative Density");lines(lwd=3,sort(A$evac),seq(0.001,1,0.001),col="red");lines(lwd=3,sort(B
    $evac),seq(0.001,1,0.001),col="green");lines(lwd=3,sort(D$evac),seq(0.001,1,0.001),col="orange")
legend("bottomright",lty=1,lwd=3,cex=1.5,bty="n",seg.len=2, legend=c("Empty Platform","Platform A","Platform B"
    ,"Platform D"),col=c("blue", "red","green","orange"))

#######Relation between Texit/Tevac and Interaction walls/Interaction passengerss##########
cor(AA[,1:4]); cor(A[,1:4]); cor(B[,1:4]); cor(C[,1:4]); cor(D[,1:4])
pairs(AA[,1:4]); pairs(A[,1:4]); pairs(B[,1:4]); pairs(C[,1:4]); pairs(D[,1:4])
plot(AA[,1],AA[,2],xlab=expression(t[exit]),ylab="Time of interaction with obstacles",main=bquote("Correlation
    = " ~.(round(cor(AA[,1],AA[,3]),digits=3))));
plot(A[,1],A[,2],xlab=expression(t[exit]),ylab="Time of interaction with obstacles",main=bquote("Correlation =
    " ~.(round(cor(A[,1],A[,3]),digits=3))));
plot(B[,1],B[,2],xlab=expression(t[exit]),ylab="Time of interaction with obstacles",main=bquote("Correlation =
    " ~.(round(cor(B[,1],B[,3]),digits=3))));
plot(C[,1],C[,2],xlab=expression(t[exit]),ylab="Time of interaction with obstacles",main=bquote("Correlation =
    " ~.(round(cor(C[,1],C[,3]),digits=3))))
plot(D[,1],D[,2],xlab=expression(t[exit]),ylab="Time of interaction with obstacles",main=bquote("Correlation =
    " ~.(round(cor(D[,1],D[,3]),digits=3))))
```

C.2 Transfer Model

C.2.1 Platform Design

```
setwd("~/Documenten/Thesis/RUN4/TransferModel")
AA = read.csv("EmptyPlatform.csv",header = FALSE,sep=",")
A = read.csv("PlatformA.csv",header = FALSE,sep=",")
B = read.csv("PlatformB.csv",header = FALSE,sep=",")
C = read.csv("PlatformC.csv",header = FALSE,sep=",")
D = read.csv("PlatformD.csv",header = FALSE,sep=",")
AA=AA[,1:7];A=A[,1:7];B=B[,1:7];C=C[,1:7];D=D[,1:7]
names(AA) = c("T1","T2","T3","exit","interpas","interwall","evac"); names(A) = c("T1","T2","T3","exit","
    interpas","interwall","evac");names(B) = c("T1","T2","T3","exit","interpas","interwall","evac");names(C)
    = c("T1","T2","T3","exit","interpas","interwall","evac");names(D) = c("T1","T2","T3","exit","interpas","
    interwall","evac")

boxplot(cex.lab = 2,cex.axis = 1.2,AA$exit,A$exit,B$exit,C$exit,D$exit,names = c("Empty","A","B","C","D"),ylab=
    expression(T[exit]),xlab="Platform",col = c("white","blue","red","green","orange"))
boxplot(cex.lab = 2,cex.axis = 1.2,AA$evac,A$evac,B$evac,C$evac,D$evac,names = c("Empty","A","B","C","D"),ylab=
    expression(T[evac]),xlab="Platform",col = c("white","blue","red","green","orange"))

cor(AA[,4:7]); cor(A[,4:7]); cor(B[,4:7]); cor(C[,4:7]); cor(D[,4:7])
mean(AA$interwall); sd(AA$interwall); cor(AA$exit,AA$interwall); cor(AA$evac,AA$interwall)
mean(A$interwall); sd(A$interwall); cor(A$exit,A$interwall); cor(A$evac,A$interwall)
mean(B$interwall); sd(B$interwall); cor(B$exit,B$interwall); cor(B$evac,B$interwall)
mean(C$interwall); sd(C$interwall); cor(C$exit,C$interwall); cor(C$evac,C$interwall)
mean(D$interwall); sd(D$interwall); cor(D$exit,D$interwall); cor(D$evac,D$interwall)

op = par(mfrow=c(2,3))
boxplot(cex.lab = 2,cex.axis = 1.2,AA$T1,AA$T3,AA$T2,ylab = "Dwell time",xlab = "Empty platform",names = c("
    Train 1", "Train 2", "Train 3"))
boxplot(cex.lab = 2,cex.axis = 1.2,A$T1,A$T3,A$T2,ylab = "Dwell time",xlab = "Platform A",names = c("Train 1",
    "Train 2", "Train 3"))
boxplot(cex.lab = 2,cex.axis = 1.2,B$T1,B$T3,B$T2,ylab = "Dwell time",xlab = "Platform B",names = c("Train 1",
    "Train 2", "Train 3"))
boxplot(cex.lab = 2,cex.axis = 1.2,C$T1,C$T3,C$T2,ylab = "Dwell time",xlab = "Platform C",names = c("Train 1",
    "Train 2", "Train 3"))
boxplot(cex.lab = 2,cex.axis = 1.2,D$T1,D$T3,D$T2,ylab = "Dwell time",xlab = "Platform D",names = c("Train 1",
    "Train 2", "Train 3"))
boxplot(cex.lab = 2,cex.axis = 1.2,xlab = "Total",c(AA$T1,A$T1,B$T1,C$T1,D$T1),c(AA$T3,A$T3,B$T3,C$T3,D$T3),c(
    AA$T2,A$T2,B$T2,C$T2,D$T2),names = c("Train 1", "Train 2", "Train 3"),ylab="Dwell time")
op = par(mfrow=c(1,1))

boxplot(cex.lab = 2,cex.axis = 1.2,c(AA$T1,AA$T2,AA$T3),c(A$T1,A$T2,A$T3),c(B$T1,B$T2,B$T3),c(C$T1,C$T2,C$T3),c
    (D$T1,D$T2,D$T3),names = c("Empty","A","B","C","D"),ylab="Dwell time",xlab="Platform",col = c("white","
    blue","red","green","orange"))

row1 = c(AA$T1,AA$T2,AA$T3,A$T1,A$T2,A$T3,B$T1,B$T2,B$T3,C$T1,C$T2,C$T3,D$T1,D$T2,D$T3)
rowAA = c("1AA","1AA","1AA","1AA","1AA","1AA","1AA","1AA","1AA","1AA","1AA","1AA","1AA","1AA","1AA")
rowA  = c("A","A","A","A","A","A","A","A","A","A","A","A","A","A","A")
rowB  = c("B","B","B","B","B","B","B","B","B","B","B","B","B","B","B")
rowC  = c("C","C","C","C","C","C","C","C","C","C","C","C","C","C","C")
rowD  = c("D","D","D","D","D","D","D","D","D","D","D","D","D","D","D")
rowT1 = c("T1","T1","T1","T1","T1","T1","T1","T1","T1","T1","T1","T1","T1","T1","T1")
rowT3 = c("T2","T2","T2","T2","T2","T2","T2","T2","T2","T2","T2","T2","T2","T2","T2")
rowT2 = c("T3","T3","T3","T3","T3","T3","T3","T3","T3","T3","T3","T3","T3","T3","T3")
row2 = c(rowAA,rowAA,rowAA,rowA,rowA,rowA,rowB,rowB,rowB,rowC,rowC,rowC,rowD,rowD,rowD)
row3 = c(rowT1,rowT2,rowT3,rowT1,rowT2,rowT3,rowT1,rowT2,rowT3,rowT1,rowT2,rowT3,rowT1,rowT2,rowT3)
df = data.frame(row1,row2,row3)
fit = lm(df$row1~factor(df$row3)*factor(df$row2))
summary(fit)
anova(fit)

par(mar=c(5,6,5,1)+.1)
row1 = c(AA$T1,AA$T2,AA$T3,A$T1,A$T2,A$T3,B$T1,B$T2,B$T3,C$T1,C$T2,C$T3)
row2 = c(rowAA,rowAA,rowAA,rowA,rowA,rowA,rowB,rowB,rowB,rowC,rowC,rowC)
row3 = c(rowT1,rowT2,rowT3,rowT1,rowT2,rowT3,rowT1,rowT2,rowT3,rowT1,rowT2,rowT3)
```

```
df = data.frame(row1,row2,row3)
boxplot(cex.lab = 1.5, cex.axis = 1.5, ylab = "Dwell Time [s]",df$row1~df$row3+df$row2,xlab = "Empty Platform,
    → Platform A, Platform B, Platform C",names = c("T1", "T2", "T3","T1", "T2", "T3",
    → "T1", "T2", "T3","T1", "T2", "T3"),col = c("white","white","white","blue","blue","blue","red","red","red"
    → ,"green","green","green"))
```