Masterarbeit

zur Erlangung des akademischen Grades „Master of Science (M.Sc.)“ im Studiengang
Wirtschaftswissenschaft der Wirtschaftswissenschaftlichen Fakultät der Leibniz Universität Hannover

Development of a Car-Sharing Simulator Including Electric Vehicles

Jan Isermann, B.Sc.

Prof. Dr. Michael H. Breitner
Wirtschaftswissenschaftliche Fakultät
Institut für Wirtschaftsinformatik

Hannover, 30.09.2014
Contents

List of Figures iii
List of Tables iv
List of Abbreviations iv
Listings iv

1 Introduction 1

2 Simulation & Car Sharing 3
   2.1 Systems, Models, and Simulation . . . . . . . . . . . . . . . . . . . . . . . 3
   2.2 Common Simulation Approaches in the Domain of Car Sharing . . . . . 6

3 Conception of the proposed E-Car Sharing Simulator 17
   3.1 General Features and Design Decisions . . . . . . . . . . . . . . . . . . . . 18
   3.2 Specific Concept . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22

4 Documentation 32
   4.1 Simulator Class . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
   4.2 Customer Class . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55
   4.3 Vehicle Class . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 67
   4.4 Station Class . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 70

5 Scenario-Analysis 73
   5.1 Description of Influence Factors, Performance Indicators, and the Scenarios 73
   5.2 Scenario 1-6 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 79
   5.3 Interpretation & Implications . . . . . . . . . . . . . . . . . . . . . . . . 98

6 Critical Reflection 100
   6.1 System-Related Issues . . . . . . . . . . . . . . . . . . . . . . . . . . . . 100
   6.2 Process-Related Issues . . . . . . . . . . . . . . . . . . . . . . . . . . . . 102

7 Conclusion 106

References 108
List of Figures

1  Simulation Distinction .................................................. 4
2  Simulation Domains in Car Sharing ................................... 7
3  Basic Mechanics of a Petri Net ....................................... 12
4  Simulation-Based-Optimization ....................................... 17
5  To Be Simulated Car Sharing Process ................................. 19
6  Structure of the Proposed Simulator ................................. 22
7  Reversed Identity Matrix .............................................. 33
8  Plotting the Station Grid .............................................. 39
9  Simulation Time .......................................................... 40
10 Driving Customers Scenario 1 ......................................... 79
11 Waiting Times Scenario 1 ............................................... 80
12 Profit Scenario 1 .......................................................... 81
13 Average Charge Scenario 1 ............................................. 82
14 Driving Customers Scenario 2 ......................................... 83
15 Waiting Times Scenario 2 ............................................... 84
16 Average Charge Scenario 2 ............................................. 85
17 Driving Customers Scenario 3 ......................................... 86
18 Waiting Times Scenario 3 ............................................... 87
19 Average Charge Scenario 3 ............................................. 88
20 Driving Customers Scenario 4 ......................................... 89
21 Waiting Times Scenario 4 ............................................... 90
22 Average Charge Scenario 4 ............................................. 91
23 Driving Pattern Scenario 5 ............................................. 92
24 Waiting Times Scenario 5 ............................................... 93
25 Average Charge Scenario 5 ............................................. 94
26 Driving Pattern Scenario 6 ............................................. 95
27 Waiting Times Scenario 6 ............................................... 96
28 Average Charge Scenario 6 ............................................. 97
29 Appendix 1: Driving Customers ...................................... vii
30 Appendix 2: Peaks during the morning hours ....................... viii
31 Appendix 3: Quadrants describing the place of living .......... ix
List of Tables

<table>
<thead>
<tr>
<th>Appendix</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>Appendix 3: A Fully Set Customer</td>
<td>x</td>
</tr>
<tr>
<td>33</td>
<td>Appendix 4: A Fully Set Vehicle</td>
<td>x</td>
</tr>
<tr>
<td>34</td>
<td>Appendix 5: A Fully Set Station</td>
<td>x</td>
</tr>
<tr>
<td>35</td>
<td>Appendix 6: Elastic price elasticity, Source: <a href="http://www.bized.co.uk">www.bized.co.uk</a></td>
<td>xi</td>
</tr>
</tbody>
</table>

Listings

<table>
<thead>
<tr>
<th>Simulator.m/lines</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-41</td>
<td>32</td>
</tr>
<tr>
<td>43-58</td>
<td>34</td>
</tr>
<tr>
<td>60-94</td>
<td>35</td>
</tr>
<tr>
<td>95-106</td>
<td>37</td>
</tr>
<tr>
<td>108-119</td>
<td>38</td>
</tr>
<tr>
<td>124-147</td>
<td>38</td>
</tr>
<tr>
<td>148-160</td>
<td>39</td>
</tr>
<tr>
<td>162-173</td>
<td>40</td>
</tr>
<tr>
<td>175-191</td>
<td>41</td>
</tr>
</tbody>
</table>
List of Abbreviations

FPT  Full-port-time
ZVT  Zero-vehicle-time
State-of-charge SOC
Level of Service LOS
Optimization-Trend-Simulation OTS
1 Introduction

In the last century, the worldwide population has increased steadily and is about to reach an astounding 8.5 billion people in 2030. Since individuals strive for both material security and a certain level of wealth, the need for basic as well as consumption goods is expected to consistently rise. However, as this additional demand can only be satisfied by a boosting production leading to higher emissions of green house gases, thus promoting global warmth, both the public and private sectors are inclined to search viable and innovative solutions to manage the growingly scarce resources while simultaneously decreasing the level of green house gases.

One of these solutions can be found in the concept of car sharing. Car sharing, in short, allows one to temporarily rent and therefore gain access to vehicles which they could use to perform daily tasks (cf. Stillwater et al. 2008, p. 1). By joining a car sharing service, the users gain the mobility and flexibility of having a car at their disposal, while also avoiding the responsibilities and costs resulting from regular insurance fees and maintenance costs or the initial purchase (cf. Shaheen and Cohen 2013, p. 5; Markel 2010). Consequently, sharing a vehicle not only offers economic advantages but also indicated positive environmental impacts, such as for traffic-related issues. Hence, a decrease in the overall number of privately owned vehicles helps to reduce congestion or the lack of parking spaces (cf. Shaheen and Cohen 2013, p. 7; Lee et al. 2012, p. 89; Parent and Gallais 2002, p. 827).

Moreover, since global warming has become more and more an issue, individuals are developing an environmental consciousness and starting to actively think about their role in the process of climate change. As car sharing offers a possibility to both reduce financial costs and one’s carbon footprint, car sharing has become more and more popular as an alternative means of transportation. This development is also reflected in the numbers of active car sharing members. Since its first appearance in 1965, where usually only a handful of people shared a small pool of cars, car sharing programs have grown impressively. Nowadays 1.25 million persons regulary use car sharing services and the fleet-size has almost doubled in the last fifteen years to 31,000 vehicles in 2010 (cf. Shaheen and Cohen 2013, p. 7).

Although alternative means of transportation are becoming more and more popular, worldwide greenhouse gas emissions, fuel and energy costs still continue growing, thus promoting and intensifying the search for further means of optimization (cf. Alli et al. 2012, p. 1; Barth et al. 2003, p. 2; Figueiredo et al. 2001, p. 1206). Being one of these solutions, car sharing companies have deployed electric vehicles worldwide in order to test their applicability and economic viability in real environments (cf. Shaheen and Cohen 2013, p. 9; Alli et al. 2012, p. 1).
However, due to very expensive field testing for car sharing systems, other methods have been developed by researchers allowing for the evaluation of the performance of car sharing system prior to their implementation. One of these methods is found in the field of simulation. Simulation tools offer possibilities to simulate the consequences of planning decisions and initial assumptions of decision-makers on the overall system performance by creating multiple scenarios depicting the potential future. Thus, decision-makers can explore various car sharing concepts accruing trade-offs through different scenarios, thereby improving the overall understanding of the system and interactions therein (cf. Farina 2013, p. 3).

Although simulations are a viable and common method in the domain of car sharing, none have ever been applied in the field of electric car sharing. Despite this significant lack of research, researchers have mostly confined themselves to analyzing, describing and predicting user behavior in casual car sharing systems, subsequently creating a research gap (cf. Jorge et al. 2012, p. 205). In order to fill this gap, this paper proposes a car sharing simulator that incorporates electric vehicles and their peculiarities while also simulating the entirety of the car sharing process, consisting of system, user, station, and vehicle-inherent processes. It is intended to serve as a decision support tool supporting decision-makers, researchers, and other users given its convenient application, simplicity, customization opportunities.

After the introduction and the description of the purpose and value of this paper in the first section, section two deals both with explaining the fundamentals of simulations, as well as presenting and categorizing various simulations in the domain of car sharing. Whereas in subsection 2.1 components of simulation such as models and systems are introduced, the second subsection 2.2 will consist of a small literature review of previous simulation approaches to car sharing. The concept of the proposed car sharing simulator is presented in the third section. In the first subsection of the respective section, the various general features the simulator is supposed to encompass are derived by a visualization of the underlying car sharing process. Furthermore, certain design decisions made by the author are explained. Subsection 3.2 introduces the specific concept of the car sharing simulator in detail, including the description of the various entities of the simulation system and their interactions in the general process. In section 4, the code of the various simulation classes, which are the entities of the system, is presented and explained in detail. In the fifth and second to last section, the actual simulation will be conducted. For that purpose, a variety of scenarios are introduced and possible simulation outcomes are described in subsection 5.1. In subsection 5.2, the actual scenario analyses are conducted, whereas the interpretation of the results follows in subsection 5.3. A critical reflection and discussion of the respective simulation results is performed in section 6. Possible limitations will be identified and matched to either system-related issues or process-related issues, where the former are described in subsection 6.1 and the
latter in subsection 6.2. Section 7, being the last of the paper, provides a final conclusion of the various sections and results of the paper.

2 Simulation & Car Sharing

In order to gain a better understanding of the meaning and purpose of simulations and its possible applications in the domain of car sharing, the following chapter starts by giving a comprehensive overview of the various components researchers have to decide upon when thinking about setting up simulations. Moreover, basic terms will be explained briefly and the various common simulation models will be categorized and introduced.

After this initial examination, a brief literature review concerning the application of simulations in car sharing will be conducted. In this consecutive subsection, some of the respective studies will be introduced and filed according to their domain of application, namely demand, relocation, and performance simulations. However, as these studies resemble each other process-wise, solely a selection of studies will be introduced in detail in each domain. Anyone who is further interested, can consult table 1 which contain additional studies.

2.1 Systems, Models, and Simulation

Over the last decades, simulations have become the most popular tool in the domain of operations research being most commonly applied in the area of manufacturing (cf. Law and McComas 1999, p. 56; Kelton and Sadowski 2010, p. 3). In general, the term simulation “...refers to a broad selection of methods and applications to mimic the behavior of real systems...” and its environment (Kelton and Sadowski 2010, p. 3). More specifically, simulations can be regarded as the “...process of designing and creating a computerized model of a real or proposed system for the purpose of conducting numerical experiments to give us a better understanding of the behavior of that system for a given set of conditions...” (Kelton and Sadowski 2010, p. 7).

Therefore, simulating systems, which can briefly be defined as a set of interacting entities, components, or elements that follow certain predefined rules, enable researchers and decision-makers alike to test their performance and consequently allow for their optimization before they are actually implemented (cf. Backlund 2000, p. 448; Hachicha et al. 2010, p. 2). In particular, simulations are used to determine the need for and the quantity of equipment and personnel. Therefore, they help during the initial set-up or to evaluate operational procedures and the overall performance of the systems to be, thus being a viable tool for their continuous improvement (cf. Law and McComas 1999,
This, in turn, is not only necessary to ensure the safety of these systems, but also mitigates costly redesign procedures during operations which could further lead to a significant loss in production.

Simulation models are abstract simplifications of the initial system that are designed for experimental purposes through simulation (cf. van Berkum et al. 1991, p. 232). These generalizations are usually made, since it is more feasible, that is more efficient and reliable, to focus on the most important features of a complex system instead of depicting every process and entity involved. Nonetheless, one should proceed carefully when considering simplifying assumptions. Although these assumptions reduce the system's complexity, they can also lead to an oversimplification, which in turn hampers the validity of the model leading to unmeasurable errors and questionable results (cf. Kelton and Sadowski 2010, p. 9).

Basically, simulation models can be matched to three different dimensions that are depicted in the figure 1. However, it has to be mentioned that models can but do not have to be a mixture of each of these three categories.

First of all, models can be distinguished according to the system’s behavior, as to whether they are static or dynamic in operation. Static simulation models depict a certain state that does not change overtime or, in other words, static systems are not subject to changes. Although the system can adopt various predefined states, its balance sheet keeps being constant meaning that nothing enters or leaves the system, therefore inclining that no growth or decrease takes place. Classic stochastic experiments, such as Monte-Carlo simulations, are static simulation models, since the experiment is being repeated, while the conditions and thus, the origin of the experiment are constants. Moreover, while the states of dynamic models keep on changing due to their subjugation to time,
static models always remain time independent and thus, can be seen as snapshots of the
time. Dynamic models are therefore more flexible and show
the behavior of a certain object and how it reacts to the various possibilities and events
that might arise in the course of the simulation.

Besides their static or dynamic nature, simulation models can either be *continuous*,
discrete, or both. In *continuous* models, the system is evaluated and its states change
continuously. This continuity means that the system always advances with a specified
interval between each iteration; so if the simulation always advances one minute at a time,
irregardless of the events or the changes in the systems state, it is called a *continuous*
simulation model (cf. Kelton and Sadowski 2010, p. 9). In other words, the time step
is determined at the start of the simulation and the time itself advances accordingly
throughout the whole simulation process.

Unlike *continuous* simulations, *discrete* simulations are, although they also include the
factor time, often event-oriented. In these *event-oriented* simulations, the system’s state
changes due to the occurrence of certain events. Therefore, the system’s variables do
not change simply because of the passing time, but solely when certain actions are
performed. An example could be a casual assembly line. Although time passes, the
state of the system only changes, when the unfinished goods are passed on to the next
station. Besides *event-oriented* simulations, discrete models can also be *process-oriented*.
Process-oriented models differ from *event-oriented* ones in the regard that the systems
solely advance, when certain sections of the whole process are completed. Hence, the
focus lies on the process rather than single events. However, *time-controlled* models are
relatively close to continuous ones. The main difference is that the simulation time in
every simulation step is advanced by a given time step $\Delta t$.

Furthermore, if these time steps are small enough that it behaves like a continuous
simulation, it is called a *quasi-continuous* model.

The decision whether a model is deterministic or stochastic is determined by the way
the input data is passed on to the system. Deterministic models strictly depend on
input data that has been given by the user beforehand. Hence, the behavior of the
system is relatively predictable (Kelton and Sadowski 2010, p. 9). Unlike the latter,
stochastic models use random inputs that are determined by probability distributions
such as normal or log normal distributions. However, since the system depends on
random inputs, the results of the simulation are also going to be more or less uncertain.
This should be considered cautiously not only while modeling the system, but also while
interpreting the simulation’s output (cf. Kelton and Sadowski 2010, p. 10).
2.2 Common Simulation Approaches in the Domain of Car Sharing

Although the field of research concerning car sharing concepts is fairly old, the idea of using simulations to preemptively determine their viability or other key parameters, such as traveling demand, is comparably new as it first emerged in 1979. Due to a considerable decrease in investment costs and rising computational power, researchers began using computers to facilitate their respective studies (cf. Bonsall 1979, p. 1; Kelton and Sadowski 2010, p. 8). Forecasting and simulation systems play an integral role in the success of car sharing systems. The simulations can help to drastically improve the efficiency of the car sharing system, since they can both aid in planning and determining the best configuration of a car sharing system in advance or by constantly analyzing the overall performance of the system (cf. Xu and Lim 2007, p. 1671; Kitamura 2002, p. 86; Barth and Todd 1999, p. 237). Moreover, through forecasting the demand and supply of the shared-use vehicles, these simulations could help to save initial set-up costs emerging from their purchase. Thus, by determining the optimal fleet size in advance, car sharing companies could avoid the acquisition of too many cars in the first place, which then would supposedly be idle most of the time due to the excessive supply (cf. Cepolina and Farina 2012b, p. 242).

Most of the studies that are introduced in this section and their simulations in-use solely concentrate on small fractions of the car sharing business-process that as depicted in figure 2.

Accordingly, these studies are matched to either demand, relocation, or station placement simulations. However, since station placement is mostly tackled by optimization and rarely being backed up by simulation, only demand and relocation simulations will be presented in detail (cf. Nourinejad and Roorda 2014, p. 49; Kek et al. 2009, p. 158). Moreover, in each of these domains, solely two studies will be presented, as these simulations resemble each other process-wise. Further simulation-related studies can be drawn from table 1.
Figure 2 Simulation Domains in Car Sharing

<table>
<thead>
<tr>
<th>Authors</th>
<th>Year</th>
<th>Topic</th>
<th>Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barth &amp; Todd</td>
<td>1999</td>
<td>Performance Analysis</td>
<td>Discrete-Event Simulation</td>
</tr>
<tr>
<td>Barth et al.</td>
<td>2004</td>
<td>User-Based Relocation Strategies</td>
<td>Simulation</td>
</tr>
<tr>
<td>Uesugi et al.</td>
<td>2007</td>
<td>User-Based Relocation Strategies</td>
<td>Simulation</td>
</tr>
<tr>
<td>Febbraro et al.</td>
<td>2010</td>
<td>User-Based Relocation Strategies</td>
<td>Discrete-Event Simulation</td>
</tr>
<tr>
<td>Wang et al.</td>
<td>2010</td>
<td>Operator-Based Relocation Strategies</td>
<td>Micro-Simulation</td>
</tr>
<tr>
<td>Ciari et al.</td>
<td>2011</td>
<td>Estimation of Car Sharing Demand</td>
<td>Activity-Based Simulation</td>
</tr>
<tr>
<td>Nourinejad et al.</td>
<td>2014</td>
<td>Relocation Strategies</td>
<td>Discrete-Event Simulation</td>
</tr>
</tbody>
</table>

Table 1 Further Studies of Simulation in Car Sharing
Demand

One of the earliest researchers who applied computer-aided simulation in the domain of car sharing was Peter Bonsall. The main objective of his study was to develop a simulation model and thus, to provide policy makers with a tool that would help and guide them either in designing or modifying car sharing schemes, by allowing them to establish direct connections between the nature and the performance of the scheme, as well as its immediate environment with respect to policies (cf. Bonsall 1979, p. 1). By doing so, Bonsall intended to bridge a significant gap between theoretical modeling and investigative field studies that was common back then, since most of the previous research concerning car sharing has been solely focused on examining either the behavior of the respective customers directly, such as their desire to use or join the service, or mathematical models that would predict the latter (cf. Bonsall 1979, p. 2).

For that purpose, Bonsall chose to apply a micro-simulation approach that would allow him to determine an individual’s readiness to join a car sharing system as well as the probability for finding a suitable trip. The particular strength of respective micro-simulation models is that they allow researchers to evaluate how changes to the system might impact the behavior at the individual level such as the customer himself. Subsequently, the main advantage of micro-simulation models lies in the detailed depiction of decision-making processes of the participants in these systems while still being relatively simple in their application (cf. Bonsall 1979, p. 3).

However, as users and their decision to join the car sharing scheme were to be simulated, the simulator required a detailed and unique description of each person’s trip maker in the system. Therefore, each of them was represented by a set of properties including age, sex, location of residence and workplace, or the normal time of arrival or departure of the workplace (cf. Bonsall 1979, p. 9). After describing each participant, their decision making processes for joining the service had to be modeled as well. In the first step, a value representing the approximate probability of a customer applying for the service was designed as a function based on the above-mentioned individual characteristics. Each of these attributes were multiplied by certain balancing coefficients, that were iteratively determined later on by comparing the simulation results with real life findings, thus ensuring the validity of the simulation results (cf. Bonsall 1979, p. 11ff.). Afterwards, the ratio between this probability and a randomly generated number between 0 and 1, drawn from a rectangular distribution, would then be determined (cf. Bonsall 1979, p. 13). This ratio, in turn, would then be seen as the likelihood of an individual to join the service. Furthermore, the region in which the car sharing scheme was to be applied, was separated in both residential areas and work locations and each of these regions was assigned a so-called ‘Threshold of interest’ (cf. Bonsall 1979, p. 8). Finally, this threshold would then be compared with the formerly mentioned ratio, or, in other words,
the likelihood of an individual to join the service. In case this likelihood exceeded the ‘threshold of interest’, the individual would eventually join the car sharing scheme (cf. Bonsall 1979, p. 13).

As Bonsall examined not only the probability of joining the car sharing scheme, but also the likelihood of finding a suitable carpooling partner, he implemented a so-called matching algorithm. This algorithm was designed to compare the various individuals and their respective characteristics, consequently searching for persons that exhibit, among others, compatible home and work locations, as well as matching work hours. The simulator would then create a list with the most advisable partners, where there would be a minimum diversion from the driver’s shortest route and timetable (cf. Bonsall 1979, p. 14).

However, since Bonsall chose a micro-simulation approach and more specifically a Monte-Carlo-Simulation, the simulator was inflexible in its application and more focused on the initial set-up and evaluation of the basic car sharing concept. As mentioned beforehand, Monte-Carlo-Simulations are static in nature. Therefore, the simulation only depicted the situation at a single point in time, namely their initial decision to join the car sharing scheme. Consequently the possibility to examine the effects that certain policies or decisions might have in the operating business was neglected.

One of the latest simulation approaches regarding the prediction of traveling demand in car sharing was proposed by Ciari et al. 2008 who introduced an activity- and agent-based micro-simulation approach to examine and determine the demand and furthermore, the viability of a large scale car sharing system, while also considering the overall transportation supply (cf. Ciari et al. 2008, p. 3; Ciari et al. 2013, p. 71).

However, in contrast to Bonsall’s micro-simulation modeling approach, the individuals of their proposed system are not only represented by certain attributes, but can be seen as actors in an environment that also exhibits a certain, goal-directed behavior and learning capabilities in accordance with predefined rules, while also featuring the ability to communicate with one another (cf. Ciari et al. 2008, p. 15; Farina 2013, p. 64). Additionally, whereas the system’s behavior of Bonsall’s approach did not change over time due to its static nature, Ciari et al. ’s proposed dynamic system and its respective behavior allows for global reactions directly resulting from the various behavioral patterns of the single agents and the interactions among them (cf. Ciari et al. 2008, p. 15). Thus, their modeling approach can be seen as far more detailed and sophisticated, although this is mainly due to the advanced technologies the researchers had at their disposal.

The core of the system’s behavior and the overall travel demand is made up of each agents’ desires to perform certain activities, thus the term activity-based system. In order to fulfill these desired activities, the agents are required to rent vehicles for certain
periods of time, that are directly linked to the activities itself (cf. Ciari et al. 2008, p. 15f.). However, the choice of initializing an activity, such as going to work, or which activities to pursue is determined by a so-called scoring function which is, in principle, an ordinary utility function (cf. Ciari et al. 2013, p. 74). This function is used to maximize the utility of the various plans an individual can go after, which, besides containing a series of activities an individual intends to perform during the day, also specifies the time and place as well as the means of transportation that are required (cf. Ciari et al. 2008, p. 16). The overall utility of the plan is calculated by the sum of the activities and their respective utility values. However, the utility of each activity is determined both by regarding each unique activity and the individual’s gain, as well as the disutility that is ascribed to the respective travel process (e.g. financial costs and required time). Finally, the plan with highest score is retained, while the others are discarded (cf. Ciari et al. 2013, p. 74).

While the traveling demand is determined by the individual agents’ desire to maximize the utility of their plans, the supply of the system is governed by the car sharing operators. These operators are also modeled as agents and control the whole car sharing system by their ability to modify its general characteristics. Similar to the customers, the car sharing operator agent tries to maximize certain key values that are part of an utilitarian function such as the revenue, the number of customers, and their general satisfaction with the service. However, while the customers can solely influence the utility of the plan by choosing the time when to pursue their activity or their means of transportation, the car sharing operator agent could either increase or decrease the size of the vehicle fleet, its composition, the number of stations and their location, or the pricing schedule (cf. Ciari et al. 2008, p. 17).

Nevertheless, as Ciari et al.’s proposed system is designed to grant open access to all vehicles as long as an individual owns a valid driver’s license, the number of potential customers in the area of their study is comparably high and mounts up to a million individuals, whereas casual car sharing companies generally have but a fraction of that number as active customers (cf. Ciari et al. 2013, p. 77). Since each individual has a set of plans and even more activities at their disposal, the simulator is required to perform staggering amounts of computations to calculate and maximize each agents’ utility accrued to their potential plans. As a consequence, simulating a single day already requires about 4.5 hours. Therefore, their simulation approach is unfeasible to simulate the long-term effects of certain operator design decisions, for instance changes in the fleet size, the customers’ preferences concerning different vehicle models or even their willingness to wait for a vehicle. Moreover, depicting the latter is not possible in Ciari et al.’s simulator, as they assumed a limitless supply of vehicles, thus further decreasing the general applicability of their findings.
For further studies dealing with predicting the travel demand in car sharing systems by modeling and simulating consult the following table. However, it has to be noted that in some of these studies, estimating the demand by simulation is just one task among others.

**Relocation**

Moreover, besides predicting the travel demand, simulation models can also be used to actively ease vehicle imbalances in one-way car sharing systems by evaluating and suggesting relocation strategies (cf. Clemente et al. 2013, p. 253f.). According to the literature, these relocation strategies can be divided into user-based and operator-based strategies. Whereas the former strategy solely depends on customers and giving incentives for relocating the company’s vehicles, the latter strategy includes the staff of the respective company (cf. Cepolina and Farina 2012a, p. 423ff.; Xu and Lim 2007, p. 1671).

For example, Clemente et al. 2013 proposed that by applying a user-based relocation strategy drawing from suggestions of a simulation using real-time monitoring of shared-use vehicles, the availability and distribution of the operator’s fleet could be improved significantly (cf. Clemente et al. 2013, p. 254). By simulating the future demand and monitoring the current distribution, the simulation system could provide users with incentives (e.g. discounts on their rental fees) that would influence their travel behavior to such a degree that they would eventually follow the system suggestions by parking their cars at less frequented stations, thus effectively minimizing distribution imbalances (cf. Clemente et al. 2013, p. 267).

As a basis for their simulator Clemente et al. 2013 chose a timed petri net that can, among others, be used to depict and simulate whole business processes. In general, these petri nets consist of three components that are called "places", "arcs" and "transitions". Both places and transitions, that figuratively can be seen as nodes, are connected by these arcs. Thus, petri-networks are so-called 3-tuples, where there is a finite set of places and transitions and a multi set of arcs that connect them, although the latter can solely connect places with transitions and vice versa but never two places or transitions with one another. Additionally, a certain weighing factor is assigned to each of these arcs. Another important component of petri nets are so-called tokens. These tokens represent the input of the model and are required for the system to advance from one place to another. This advance is conducted via the aforementioned transitions. Each transition checks the amount of incoming tokens and advances the process when the directly connected input places possess the required number of tokens. A small example is provided in figure 3.
In addition, because the authors used a timed petri-network, which is a special form of casual petri nets, the factor time was introduced. As the whole process advances with each firing transition, the transition itself henceforth requires a certain amount of time. So instead of solely having the possibility to check whether the whole process finished successfully or reached a certain place, the amount of time it took the system to do so could be precisely determined as well. In addition, the financial incentives to induce the desired relocation behavior of the customers were depicted by the arcs and more specifically, by their respective weights. By assigning higher or lower values, the required number of input tokens for a transition to fire could be increased or decreased. Therefore, giving the incentive to relocate was modeled by lowering the weight of an arc, therefore increasing the likelihood that the incoming number of tokens would suffice for the transition to fire (cf. Clemente et al. 2013, p. 252f.).

In their case, Clemente et al. 2013 designed three independent petri nets representing the different relocation scenarios and their respective business-processes, namely "to-be-offline", "to-be-online" and "as-is". In their "to-be-offline" scenario, the distribution of vehicles is not monitored actively but in regular intervals and the financial incentives to relocate the vehicles are determined at each of these intervals. In their "to-be-online" scenario, the distribution of vehicles is regularly monitored and thus, the financial incentives are determined constantly. In their last scenario, "as is", the system and certain key factors are evaluated at the end of each day, the incentives are determined accordingly and take effect the next day (cf. Clemente et al. 2013, p. 258ff.).
After the simulation, Clemente et al. 2013 evaluated several key factors that allowed them to assess the effectiveness of their proposed strategies. Since they modeled the factor time, they were able to assert how long it took for the customers and their vehicles to arrive at the two parking areas after each rental process (cf. Clemente et al. 2013, p. 264). Additionally, they introduced three performance indices, namely Level of Service as well as the companies revenue and monetary gain. The Level of Service (LOS) expressed the number of users who were successfully served. This could be measured by counting the number of times the transitions representing a customer leaving the station with a vehicle fired, thus advancing the car sharing process. The company’s revenue was composed of the hourly price for each rented vehicle, as well as the distance dependent fees minus the financial incentives needed to achieve a relocation behavior that would reduce the vehicle imbalance. The last measure of effectiveness, the companies monetary gain, also took into consideration the otherwise intangible image damage in case a customer is not served. This was done by introducing a monetary penalty totaling up to five dollar each time a vehicle was needed but unavailable (cf. Clemente et al. 2013, p. 264f.)

As their results suggested, constantly monitoring and adjusting the financial incentives reaps the best results, consequently increasing the number of customers served and providing economic benefits. Encouraging customers to relocate the vehicle as soon as possible, however, was no appropriate solution, since it did not significantly lower the overall vehicle imbalance and led to economic losses (cf. Clemente et al. 2013, p. 267).

Although dealing with the same topic of decreasing vehicle imbalances by relocation, Kek et al. 2009 focused on an operator-based instead of user-based relocation strategies. The authors main goal was to develop a three-phased optimization-trend-simulation (OTS) decision support system that would help to effectively optimize certain operating parameters of the car sharing system such as staff strength, shift hours or station related criteria (Kek et al. 2009, p. 151). Thus, by optimizing and recommending the ideal system configuration, car sharing operators would be able to both increase the operational efficiency and service levels while also reducing the overall costs (cf. Kek et al. 2009, p. 158).

As mentioned above, the decision-support-system consists of three interdependent phases. The first phase consists of an optimizer that requires the user to enter a Setting of system specific characteristics. Among others, these inputs encompass station-specific parameters such as number of vehicles and parking spaces, vehicle-usage and maintenance patterns as well as relocation and staff costs (cf. Kek et al. 2009, p. 152). After retrieving the respective input, the optimizer proceeds to minimize the objective function. This function aims at minimizing the total costs consisting of movement, relocation, and staff related costs while also considering imaginary costs, such as penalties for the
inability to serve customers. Although the vehicle relocation and staff movement costs remain fixed, the optimizer nonetheless can take various sets of costs into consideration (cf. Kek et al. 2009, p. 154). However, objective functions are always subject to various constraints which characterize the systems peculiarities. Therefore, Kek et al. 2009 proposed constraints that would restrict each staff member to one relocation process at a time, subsequently ensuring that each of them would solely begin a new relocation after the previous had been completed. Besides introducing constraints solely concerning the staff, further restrictions were proposed that involved the systems vehicles. Some of them assured that the number of both available and unavailable vehicles never exceeded the respective station’s capacities, while others ascertained that the rejected relocation attempts never surpassed the number of requested relocations (cf. Kek et al. 2009, p. 154).

In the next phase, the output of the optimization process, specifically the staff strength, staff activities, relocations, and station status including vehicle allocations were passed onto a so-called Trend Filter. This filter uses a set of heuristic methods to filter and extract respective key information including trends. By doing so, the authors were able to analyze the proposed values in greater detail, therefore finding possible discrepancies between theory and practice. For instance, the optimizer suggested employing one staff member between 8 am and midnight to get all the relocations done, while the Trend Filter found that most of the relocations were to be performed between 12 a.m and 10 p.m. As a consequence, the shift hours were adjusted accordingly, thus reducing the overall working hours notably at the cost of a minor decrease in vehicle availability (cf. Kek et al. 2009, p. 155).

The last phase constitutes the actual simulation phase where the simulator receives the operating parameters obtained in phase two and starts the evaluation of the system's effectiveness. These input parameters are, however, handled differently according to whether they are operational set-up or dynamic event parameters (cf. Kek et al. 2006, p. 7). Whereas operational set-up parameters remain consistent throughout the simulation, dynamic events may differ between every time step which the authors chose to be a minute each (cf. Kek et al. 2009, p. 156). Operational set-up parameters encompassed the station (i.e. number of vehicles), job (type of task, e.g. maintenance), and staff parameters (staff strength) as well as the relocation technique to be used (cf. Kek et al. 2006, p. 7). Dynamic events, on the other hand, referred to either trip data, staff status, or basic job status. Trip data contained information on when vehicles were to be picked up or returned by the customers, while staff status entailed the whereabouts and tasks of the staff. Lastly, basic job status described the frequency and occurrence of when basic tasks, such as maintenance, were to be expected from the staff (cf. Kek et al. 2006, p. 8). After the classification of input data, the simulator initiated the simulation. At the beginning of each iteration, the availability of the staff was checked. In case a staff
member was idle, the simulation checked whether a basic task needed to be performed. These basic tasks were prioritized, since they were solely required, when a vehicle malfunctioned. If this were not the case, the system checked for pending relocation requests and, if needed, assigned a staff member and initiated the relocation, thereby updating the staff and station status parameter, which were, as mentioned before, dynamic throughout the whole process. As soon as all assignments were appointed successfully, the system advanced to the next time step until, eventually, the simulation ceased (cf. Kek et al. 2006, p. 8).

The effectiveness of the proposed parameters obtained in phase two was then measured by the introduction of three distinct performance indicators. First of all, the total amount of time the stations spent without vehicles was captured by the indicator zero-vehicle-time (ZVT). Conversely, full-port-time (FPT) measures the time the stations were fully occupied, whereas the indicator number of relocation speaks for itself. From the customer’s perspective, both ZVT and FPT are to be kept as low as possible, since they reduce the perceived attractiveness of the car sharing service; when the station lacks appropriate vehicles, the customer cannot be served and has to go to another station. To the contrary, when all parking spaces are filled, the customer can neither drive nor return to his desired station. As for the operators, although FPT is to be avoided, solely ZVT has direct negative impacts, since potential customers are unable to rent a vehicle, consequently decreasing the potential revenue (cf. Kek et al. 2006, p. 9).

Finally, by using their decision support system the authors were able to significantly improve the initial system configuration, thus enabling the operators to reduce all three performance indicators while also decreasing the staff costs by about 50% (cf. Kek et al. 2009, p. 158).

Station-Placement

Contrary to the domain of testing relocation strategies, simulations are rarely used when it comes to station-placement in car sharing systems. In general, respective challenges are tackled by their transformation into optimization problems and the subsequent appliance of solvers. This might mainly be due to the nature of station placement concerns, since they mostly require and consider rather static key values, such as the demographic structure of the population in the vicinity of the stations. Therefore, the simulation of dynamic environments is not explicitly required. However, using simulations paired with optimization, which is also-called simulation-based-optimization, to validate the viability of the proposed parameters was shown to be an effective approach.

Some of the researchers that conducted simulations to test for the validity of their optimization results in the field of station placement were Lam et al. 2013. As the placement of charging stations is especially critical for e-Car sharing systems and car sharing op-
erators are forced to find a balance between accessibility and economic feasibility, the authors developed an Electric Vehicle Charging Station Placement Problem. Furthermore, they introduced four independent solution methods namely Iterative and Effective Mixed-Integer Linear Program, Greedy Approach, and Chemical Reaction Optimization (cf. Lam et al. 2013, p. 5f.).

Both the iterative and the effective mixed integer linear program belong to the field of linear programming. Linear programming, however, is a method that minimizes or maximizes a certain outcome under the pretext of exhibiting linear relationships both in linear equality and linear inequality constraints. Greedy Approaches use a certain kind of algorithm called greedy algorithm. Respective algorithms have the advantage of decreasing computation times, since they solely search for local instead of global optima during each computational step. Lastly, in the chemical approach the entities to be examined are seen as molecules. Each of these molecules carry a solution within them which is determined by exploring "...the solution space of the problem through a random series of elementary reactions taking place in a container" (Lam et al. 2013, p. 6).

To test for the effectiveness of each of these solutions, Lam et al. 2013 proceeded to apply a simulation-based-optimization approach. Corresponding techniques permit integrating optimization methods into simulation environments and thus, simulation analysis (cf. Deng 2007, p. 1). In general, simulations are used to examine the effects of changing the initial setting of the system. By simulating various system settings, the user can, eventually, compare these different scenarios and thus, is able to manually determine the best initial setting (cf. Nguyen et al. 2014, p. 128; Farina 2013, p. 63). The simulation-based-optimization aims, however, at automating this selection of the "best" alternative by applying an algorithm using an underlying simulation model. For that purpose, the various possible set-up scenarios are modeled as simulation variables. Subsequently, the simulation-based-optimization algorithm then proceeds to vary these simulation variables. Afterwards, the simulator simulates the system with the respective changes and creates certain outputs which then are passed back to the algorithm. The latter, serving as an objective function, then evaluates the output stemming from the various scenarios and finally, returns the best solution. So instead of a simulation solely examining the effects of variables, a simulation-based-optimization iteratively optimizes and thus, determines the most viable solution by simulating certain scenarios (cf. Nguyen et al. 2014, p. 1044ff.). The whole process ends, when the stopping criteria, for instance the maximum number of iterations, have been met. The general procedure is depicted in figure 4.
After performing the simulation-optimization of their four approaches, Lam et al. 2013 compared them by introducing the five different categories \textit{Solution Quality}, \textit{Computation Time}, \textit{Problem Size}, \textit{Algorithmic Nature}, and \textit{Prerequisites}. According to their results, none of the suggested approaches dominates in all of the aforementioned categories. Consequently, each has certain strengths and weaknesses and thus, should be used for different situations with varying requirements (cf. Lam et al. 2013, p. 10).

3 Conception of the proposed E-Car Sharing Simulator

After giving this detailed insight on common simulation approaches in the domain of car sharing, the following chapter will deal with the conceptualization and design of the proposed simulation tool. Therefore, the first subchapter focuses on the introduction of the general concept of the proposed simulation tool and its underlying system. For that purpose, a business-process is being depicted in 3.1 based on which the desired features will be derived. The section ends with subsection 3.2 that deals with a detailed conceptualization of the simulation tool, thus giving both knowledge about the various interactions between the different entities and the process of their implementation. Additionally, each entity and their respective properties are explained in greater detail.
3.1 General Features and Design Decisions

As it can be derived from chapter two, most simulations in car sharing are highly specialized and focus mostly on single problems of respective systems such as demand, station placement, and relocation strategies, where the latter is to be seen as the main field of research. However, while narrowing down the focus on single elements may reap more detailed and thus, better examinable results in that domain, but omitting many aspects and skipping various steps of the rental process also may slightly compromise the overall quality of the results.

For that reason, the author of this paper proposes an easy to use simulator covering the entirety of the car sharing process while providing users with numerous opportunities to change the various system parameters at their leisure. Thus, this simulator not only allows users and researchers alike to examine, for instance, the effects of changing the rental fees on travel demand, but also how respective decisions might influence key performance indicators such as vehicle availability, average time of rental or the revenue of the car sharing system. Furthermore, applying such a simulation approach might allow for the examination of possible cross dependencies between these performance indicators and consequently can help in answering questions such as how the total number of served customers influences the revenue of the car sharing program. As mentioned in the introduction of this chapter, the systems underlying car sharing process is depicted in figure 5 to the provide for a visualization and thus, a better understanding of the features offered by the simulator.

The whole process starts with a customer wishing to rent a vehicle. This desire will be determined stochastically for each customer, thus rendering the simulation model as partly stochastic. By applying a probability density function, a value representing the minimum time interval having to pass between each rental is set. As soon as the simulation time exceeds this given minimum interval, the customer checks for a station in his immediate vicinity. As each customer has a certain maximum walking distance, they will solely go to a station that lies within these given limits. So in case there is no accessible station nearby, the customers will simply refrain from using the car sharing service all together. Otherwise, the starting station is set and the customer proceeds to search for an appropriate vehicle. The choice of vehicle, however, depends on the type of customer, which is, besides other properties, set at the beginning of the simulation. If the station does not have a vehicle with the desired type, the customer either waits or completely terminates the rental process.

If his conditions are met and he found a car, the customer will start driving. At this point, the rental starts to generate revenue. However, since the simulation will advance minute-by-minute, hourly fees are adjusted accordingly. Whereas revenue is only generated while
Customer wants to rent a vehicle
Customer checks for the nearest station
Station is close enough
Customer checks availability of vehicles
Desired vehicle is available
Customer claims vehicle
Customer drives
Customer reaches destination
Customer performs tasks
Customer finished tasks
Customer leaves
End

Station is too far away
Customer checks availability of vehicles
No vehicle available
The customer waits
Vehicle still unavailable
Customer drives back
Customer reaches starting station

Figure 5 To Be Simulated Car Sharing Process
customers are driving, operational costs arise during the whole simulation. These costs will be composed of maintenance, station, insurance, and administrative costs, which are distributed evenly throughout the entire simulation process, thus being separately computed every minute. Although a map is used to visualize the trip and the station grid, the customers will not use the actual, real streets depicted on the map, but will drive the Euclidean distance, or in other words, a direct line between both stations. However, as a respective distance inevitably underrepresents the real distance, a fixed percentual value is added to account for this simplified approach.

Yet the customer’s trip will be determined by choosing a destination station, he will never actually reach it, since the proposed car sharing system only supports two-way rentals. A stopping criterion is introduced forcing each customer to stop at a random portion of the maximum distance between both stations. The place he stops at symbolizes his actual destination, figuratively being a shopping mall and else. The customer then proceeds to perform the chores he initially intended to carry out while renting the vehicle. The time he needs to pursue these activities depends on the type of customer as well. Whereas students usually rent vehicles for shorter trips, families might need longer, since they use the vehicle for an extended family trip or something similar. While the customer performs his tasks, the vehicle is parked. However, as it is common in many car sharing systems that the rental fees are lower while a vehicle is being parked, the rental fees will also be lowered correspondingly. Additionally, the time required for the customer’s chores partly depends on the hourly rental fees, as one is inclined to hurry up, when paying rental fees.

Upon finishing his business, the customer will start driving again. Since the car sharing system is designed to solely allow for two-way rentals, the customer will automatically drive to his initial starting station. The rental ends, when the customer reaches the station; yet, at the time of his arrival, the customer automatically will connect the vehicle to the charging station. Moreover, there will be two types of charging stations: a fast charger and a casual charger. For simplification, it is assumed that customers will always tend to, if available, use fast chargers, although no incentives are set to promote a respective behavior.

As previously mentioned, the proposed car sharing system will solely allow two-way rentals. Two-way rentals strongly reduce the flexibility provided by car sharing services, since customers will be forced to return the vehicle at the station they initially rented the vehicle (cf. Jorge et al. 2012, p. 139. Although this decreases the attractiveness of the service notably, in case of e-Car sharing, a respective approach is advisable and therefore chosen by the author, since many respective issues can be avoided thereby. One of the biggest issues of one-way rentals in e-Car sharing is relocation. Although relocation is also an issue when it comes to casual car sharing services, balancing the number of
vehicles across all stations is more challenging in e-Car sharing systems. This is mainly due to the difference between casual vehicles depending on fuel and electric vehicles using electricity, as the latter can only be recharged at car sharing stations, whereas the former can refill fuel at every gas station. However, as electric vehicles require chargers and their amount at each station is inevitably restricted, having more vehicles parked at the station than the station can provide chargers for is to be avoided. Casual car sharing operators, however, usually refrain from using static stations, as renting parking lots all over the city is more convenient and saves respective investment costs. Furthermore, as it is far cheaper to rent and distribute numerous parking lots all over the map than building stations, it is less likely that the majority of vehicles will be clustered at single locations, thus needing further relocations. In addition to that, the distance between each station is likely to be higher than between parking lots. Therefore, when an e-Car sharing station has no vehicles available, it is unlikely that the customer will simply change the station, as they are too distant from each other. Consequently, it is important that each station always has at least one vehicle available.

Additionally, reservations features will be omitted. This is necessary, as a respective feature would require further mechanisms. One of these would be travel time and time of arrival estimation techniques which are individual fields of research and therefore, too complex to be easily implemented in this simulator.
3.2 Specific Concept

This subsection deals with the introduction of the specific concept of the simulator, thus encompassing the entirety of the entities involved, their interactions and properties, as well as the required input data and the final outputs. For that purpose, a scheme will be presented in figure 6 that shows the overall structure of the micro-simulator and all the dependencies between each component. As this scheme aids visualizing and understanding the function of the simulator, this sections structure will be derived accordingly.

![Figure 6 Structure of the Proposed Simulator](image)

Generally, the simulation will use a dynamic instead of static model. Furthermore, it is time-controlled and more specifically, due to the small number of time steps, quasi-continuous. The data being used is both determined stochastically and deterministically. Therefore, a dynamic, discrete-continuous, mixed logic is applied. The simulation allows to analyze and even to graphically follow the activities of each user and vehicle in detail.¹ The simulator is used to perform scenario analyses when changing core parameters of the system such as demand, customer preferences and the station set-up. By choice of the author, the entities of the model will be adopted into the code as so-called classes, since it keeps the structure of the code as close as possible to previously depicted business-process and thus, to operating business of car sharing companies. The classes, each having a different set of properties and functions, representing the system are:

---
¹for an image of the driving customers see appendix 1
- Simulator
- Customer
- Vehicle
- Station

Each class is comprised of descriptive properties and influence properties. The descriptive properties solely describe the system as is, whereas the influence properties offer opportunities to directly influence the simulation and to examine the effects of respective changes. It has to be noted that due to the interconnections the classes are having, influence properties of one class can be the descriptive properties of another. Furthermore, this distinction is solely applied by the author to allow for a better understanding of the concept, as the simulation tool does not differentiate accordingly.

**Simulator**

The simulator class constitutes the core of the simulator. It encompasses the definition of main variables, the conduct of certain main functions and the initialization of the system’s entities. Additionally, it will be used to both calculate and display the performance of the system by introducing performance indicators such as revenue and number of rentals. It involves the following properties:

<table>
<thead>
<tr>
<th>Properties</th>
<th>Influential</th>
<th>Descriptive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Users</td>
<td></td>
<td>Simulation Time</td>
</tr>
<tr>
<td>Number of Stations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vehicle Capacity</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2 Simulator Properties

**System Set-Up**

The purpose of the simulator class is, among others, to deal with the set-up of the station grid. First of all, the number of stations, which is one of the main properties, needs to be determined by the user. However, a default number of stations will be set by the author, as the station grid is supposed to be based on Volkswagen’s Quicar car sharing service. In addition, an array containing the respective station coordinates will be implemented, thus setting Quicar’s station grid as the default. The user can proceed to use these default options or is free to increase or decrease the number of stations. In case the user enters a number higher than the default values, a function was designed to allow him to conveniently place the remaining stations. For that purpose, a window containing the map in-use is opened automatically. By clicking on the desired station locations and
pushing the enter button, stations are added to the system up to the point that the actual number of station matches the desired number of stations. In the opposite case, namely when the user reduces the number of stations, the length of the station coordinates array is shortened by the difference between the default and the desired number of stations.

As mentioned before, the map and its roads are solely used for visualization purposes, as the customers will not follow the course of the actual roads. The paths customers can take are determined by the set-up of the station grid. Each station is directly connected with one another by roads; therefore the road network consists of straight lines ranging from one station to another. However, as the system to be modeled is a two-way rental system, the customer’s trip will never cover the whole distance but a random portion of it. Besides determining the number and locations of stations, the simulator class will also be used to define various additional station characteristics, as the user can set the number of vehicle capacities as well as the number of fast and normal chargers of each station. Therefore, the total number of vehicles, set by the number of vehicles, is multiplied by the number of stations. Furthermore, the vehicles will be assigned to the stations, the vehicle class is being initialized, and the distances between each station will be determined. Finally, the station class with all its properties will be initialized by passing on the aforementioned properties from the main class to the station class.

Besides the set-up of the station grid, the simulator class will also be used to determine the number of customers, being one of its main properties. For that purpose, the user can simply choose and enter any desired amount of customers. Following up, some of the customer class’ functions, which are used to initiate many of its primary properties, will be called. These, however, are defined in the customer class itself.

**Simulation**

The main reason for the simulator class to be constituted as the core of the simulation is that it contains the simulation process besides being the class initiator. Whereas the other classes mainly encompass functions setting their properties, and therefore can be seen as static, the simulator class encompasses a process continuously being performed until the simulation is ended either by force or by meeting the stopping criterion. The simulation process is initiated by determining the start and the end time, where the latter is also the aforementioned stopping criterion. The simulation will then advance minute by minute, starting with validation whether a customer is driving or not. Nevertheless, by definition, all customers will be set as idle at the start.

If the customer is not driving, the simulator checks whether he wants to drive. This is determined by comparing the minimum time interval a customer requires to pass between each rental and the current simulation time minus the time of his last rental. In case the former is lower than the latter, a customer wants to rent a vehicle. However, separate
minimum intervals have been set for day and night rentals. The next step simulates
the customer searching for his nearest station. This is done by determining the walking
distances between his place of living and all existing stations, and by picking the lowest
respective value. Afterwards the distance is controlled as to whether it is within the
bounds of his maximum walking distance. In case it is, the respective station will be
set as the starting station. Otherwise the customer will refrain from driving throughout
the entire simulation which will be counted and saved as a performance indicator in a
separate variable for a later examination.

Subsequently, it will be evaluated whether a suitable vehicle having sufficient charge
and the right vehicle type is available at the station which is done separately in the
customer class. If there is none, the customer is set to wait. As soon as a vehicle
becomes available, it is assigned to the customer who immediately starts driving to
his destination, which is determined randomly by the simulator. However, if the current
waiting time becomes higher than the customer’s maximum wait time, the rental process
will be terminated. If this occurs, the variable describing his last time of rental will be
updated to the current simulation time, since a failed rental implies that the customer
wanted to use the car sharing service but could not. Therefore, his desire to drive
is restarted, because otherwise, the customer would automatically attempt to rent a
vehicle in the next iteration. In any case, the waiting time will be averaged out for
all customers and saved as a performance indicator. Additionally, the number of times
customers could not rent a vehicle due to the vehicles being unavailable or their charge
being too low will be counted separately for evaluation purposes.

As soon as the customer is set to drive, the charger the vehicle was plugged in will
become available once again, depending on whether the vehicle was previously charged
by a fast or a normal charger. Moreover, the vehicle will be removed from the current
station’s vehicle pool in order to avoid multiple customers renting the same vehicle.
While driving, the vehicle will constantly lose charge. The extent of this depletion will
be determined by the vehicle type and its respective average consumption. Moreover,
the customer’s state will be changed from idle to driving. Alternatively, if the state
could not be changed, the whole process would start anew in the next iteration which
is commencing with the validation of whether a customer is driving or not (see above).
In addition, while driving, the customer’s current position will be perpetually updated
and revenue is generated equaling 1/60 of the hourly fees. By constantly updating all
customer’s positions, their trip can and, if wanted, will be visualized on a separate map
containing the station grid and their connections. Besides showing the customer’s route,
his ID and current vehicle will be depicted as well, therefore allowing for an identification
of each and every driving entity.

Although the destination of the customers will be randomly determined assigning a
destination station, his actual destination will be on his route. Thus, the customer will be stopped somewhere along the way, although there is a minimum and a maximum distance for each rental. These boundaries are introduced so that no one will solely drive, for instance, 100 meters, since in reality nobody would rent a vehicle for such a distance. The driving speed will also be set randomly, although once again, maximum and minimum speeds are introduced. Upon arrival, the vehicle will be stopped and the rental process is interrupted symbolizing the customer going about his business (i.e. shopping). The length of the break depends on the customer type and varies randomly within these given limits in order to simulate a realistic behavior. As mentioned in the previous subsection, the revenue generated in this period will be lower than when the customer is driving.

When the customer has finished his tasks, the driving function will be called once more. However, the current position will be set as starting point and his starting station will be set as the destination. Therefore, the customer will turn around and proceed to drive until he reaches his initial station. At the time of his arrival, the vehicle will be added to the stations vehicle pool once again. Furthermore, a charging station will be occupied by the returning vehicle, depending on whether fast or a normal chargers are available. However, presumably customers will generally will use fast chargers before taking normal chargers. Afterwards, the customer’s position will be set as unavailable, so that he will not be shown on the map any longer. Finally, the current simulation time will be set as the customer’s last time of rental and the successful rental will be recorded, as the sum of all successful rentals will be used as a performance indicator.

Visualization of Results

Besides the initial set-up of the system, its properties and the simulation process, the simulator class is used to compute and display some of the various results of the simulation. These encompass the revenue and costs of the car sharing operation, number of driving customers, average waiting times, successful as well as unsuccessful rentals, and the total number of drop-out customers. The revenue is solely generated by the driving customers, since no monthly fees are assumed. The costs compose of initial costs and operating costs, where the former are investment costs for both vehicles and stations and the latter consists of vehicle related insurance and maintenance costs. Additionally, expenses for the administration and employees are considered as well. In addition, some of the graphs depicted in the figures will change colors depending on whether it is night or day, consequently allowing for a better distinction and a more precise evaluation. However, since constantly plotting the various outputs requires a lot of computational power and thus, significantly decreases the simulation performance, the user is able to turn these features off.
Customer

As the most important entity of the simulation, the user class enacts of both functions being called in the course of the simulation as well as the initial set up of customer-related properties. It comprises of mostly influence properties and therefore offers many opportunities for the user to alter the customer behaviour and thus, the overall simulation set up. The most important properties describing this class are to be taken from table 3.

<table>
<thead>
<tr>
<th>Influential</th>
<th>Descriptive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum Interval</td>
<td>ID</td>
</tr>
<tr>
<td>Customer Type</td>
<td>State</td>
</tr>
<tr>
<td>Current Position</td>
<td>Last Time of Rental</td>
</tr>
<tr>
<td>Place of Living</td>
<td>Sum of Rentals</td>
</tr>
<tr>
<td>Max. Walk. Distance</td>
<td></td>
</tr>
<tr>
<td>Max. Duration of Stay</td>
<td></td>
</tr>
<tr>
<td>Maximum Wait Time</td>
<td></td>
</tr>
<tr>
<td>Vehicle Preferences</td>
<td></td>
</tr>
<tr>
<td>Vehicle Tolerance</td>
<td></td>
</tr>
</tbody>
</table>

Table 3 Customer Properties

Setting Customers

These properties, however, were renamed or pooled for this subsection in order to improve the comprehensibility of the conceptual design. The customer’s ID ranges from one to the total number of customers, which is set by the user in the simulator class, and allows for their direct identification. Furthermore, the ID is used to determine the customer’s type. This is done by setting certain thresholds and comparing the customer’s ID with the number of customers multiplied by this threshold. In case the ID is below one of these thresholds, for instance lower than 33% of the total number of customers, the customer will belong to customer type one. In this sense, a customer having an ID higher than 33% of the total number of customers would be assigned customer type two. The distinction between customers and their respective types also has important implications for the other properties of the customer class, as all except for the current position, state, and sum of rentals vary notably between each customer type. The customer type is implemented to allow for the development of different customer profiles, each having different rental patterns and preferences. For example, a family will probably rent vehicles more often than a student.

As described in the simulator class, the desire to rent a vehicle depends on the minimum
time interval having to pass between each rental. The respective minimum interval is also set in the customer class, being one of its main properties. In general, these values are determined by a logarithmic normal distribution which is why the simulation model is to be seen as partly stochastic. A logarithmic distribution is chosen to verify that the minimum interval will always adopt positive values. Likewise, when using normal distributions, the intervals could also become negative. A negative minimum interval, however, would result in customers starting to drive immediately after their last rental being a result of determining the desire to drive that is described in the simulator class. When comparing the minimum interval with the current simulation time minus the last time of rental, the former would always be smaller than the latter and thus, a customer would want to drive immediately. Additionally, depending on which type the customer belongs to, both the mean and the standard deviation are increased or decreased, therefore leading to higher or lower minimum intervals. Besides the customer type, the fees for each vehicle also have an impact on the mean, thus enabling the user to examine the effects of pricing policies.

After the minimum interval is set, the time of the customer’s last rental needs to be determined. This is done just once at the beginning of the simulation and serves as a starting condition. If no last rental was set beforehand, the customers would wait for the whole minimum interval to pass until they desire to rent a vehicle. Thus, nobody would start driving at the beginning of the simulation, which is to be avoided. The last rental will be set randomly by the simulator, although it is aligned with the minimum interval and therefore close to it. By multiplying a value close to the minimum interval with a random number between zero and one, the last rental is to be located within these given boundaries.

In the next step, the customer’s place of living is to be assigned. The respective approach resembles the method of determining customer types. Consequently, thresholds are defined according to which the customers are distributed into four separate quadrants on the map. As of now, these quadrants are supposed to have the same dimensions and therefore, all customers are distributed equally all over the map.

Afterwards, the maximum walking distance and the vehicle preferences are determined. Both values are set deterministically, and are consequently consistent across all customers. However, each customer type implies a different maximum walking distance and vehicle preference.

The last properties being defined in the customer class are the maximum wait time, the duration of stay, and the tolerance of renting another vehicle. Whereas the maximum wait time determines how long a customer will wait for a vehicle at best, the duration of stay defines the amount of time a customer requires to perform his tasks. The tolerance of renting another vehicle is introduced to simulate cases in which customers are willing
to switch to another vehicle not matching their actual vehicle preferences.

**Choice of Vehicle**

Besides these functions setting the various customer-related properties, the customer class also encompasses a function that simulates the customer’s vehicle decision process. In the first step, it is checked if vehicles are available at the customer’s closest station. Afterwards, the state-of-charge (SOC) of the vehicles is examined in order to verify whether one of them has enough energy for the journey. The minimum charge is determined by predicting the consumption during the trip and adding a buffer equaling a certain, user-defined, percentage of this prediction. Thus, certain occurrences such as traffic congestions, are considered in the simulation as delaying the customer’s travel which in turn would increase the overall consumption. In case a vehicle with enough charge and the matching vehicle type is available, it is assigned to the customer and the simulator class proceeds simulating the rental process. If not, it is distinguished whether the vehicle did not have enough charge, did not match the type, or no vehicle was available in the first place. These can be examined later in the course of the evaluation.

**Resetting Customer Properties**

At the end of each rental process, various customer-related properties need to be reset. As mentioned in the simulator class, during the rental the customer’s current state will be set as driving, whereas between each rental it needs to be set as not driving. Moreover, the customer’s current position and his start and destination station have to be set as not available or zero. Finally, the current simulation time is checked to verify whether it was a day time or a night time rental, as the simulation distinguishes between them. In both cases, a variable recording the customer’s total amount of either day and or night time rentals is increased by 1. The respective variable is then used in the simulator class after the simulation to determine and visualize the total number of rentals of all customers.
Vehicle

In comparison to the simulator and the customer class, the vehicle class is fairly small. First of all the properties being depicted in table 4 are declared.

<table>
<thead>
<tr>
<th>Properties</th>
<th>Influential</th>
<th>Descriptive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vehicle Type</td>
<td>ID</td>
<td></td>
</tr>
<tr>
<td>Maximum Charge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nominal Consumption</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4 Vehicle Properties

Setting Vehicles

The total number of vehicle IDs is determined by multiplying the number of stations in the system with their individual vehicle capacities. The IDs are used to identify and display the driving vehicles on the map or when vehicles are assigned to customers.

The property vehicle type is introduced to allow for the simulation of different vehicles having distinct technical properties, thus giving the user further opportunities to influence the simulation and to test, for example, the applicability of specific vehicles in the car sharing system. The method of determining the vehicle type resembles the method of assigning customer types. In this case, thresholds are determined up to which a vehicle will have a certain vehicle type. For instance, the user defines that the car sharing system offers three distinct vehicles. Afterwards, the user determines how many percent of the stations total number of vehicles will be vehicle type one. If the user defines that each vehicle type makes up 33% of the stations vehicle capacities and there are three vehicles total at the station, the first vehicle will be type one, whereas the second will be type two and so on.

The maximum charge, being one of these varying technical properties, describes the vehicle’s charge capacities, whereas the property nominal consumption sets the amount of charge a vehicle requires for traveling a distance of 100 kilometers.

Battery Management

Besides these property setting functions, the vehicle class encompasses two additional functions that are used during the simulation. First of all, a function was implemented that depicts the deterioration of the vehicle’s battery charge capacities. Therefore, when performing lengthy simulations, the impacts of this degradation can be examined in detail.

The last function of the vehicle class is run throughout the entire driving process. It
computes the consumption during the trip by multiplying the driving distance with the nominal consumption and subtracting it from the current SOC. Thus, the function simulates the vehicle’s battery consumption process.

**Station**

The station class is the last class in the simulation and mostly consists of descriptive properties, as many station-related properties are defined in the simulator class. Thus, it does not offer any opportunities for the user to influence the simulation and test for different scenarios. The class is represented by the following properties:

<table>
<thead>
<tr>
<th>Properties</th>
<th>Influential</th>
<th>Descriptive</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capacities</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vehicles IDs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vehicles</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chargers</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5 Station Properties

**Setting Stations**

Just like the customers and vehicles, a station is assigned an ID and coordinates that allow for its identification and display on the general map, or to determine the distance of trips. Furthermore, each station has a maximum vehicle capacity that is determined in the simulator class which is why it is not listed as an influential property here. As the car sharing system solely supports two-way rentals, this capacity is never exceeded. The property vehicle IDs solely contains the specific IDs of all vehicles that are currently parked at the station, whereas the property vehicles is comprised of the actual vehicle objects that were assigned to the station. The last property chargers only describes the number of chargers the user set in the simulation class. There, the user can vary the composition of the station’s chargers and decide on how many normal and fast chargers are deployed. As fast chargers significantly improve the charging times, they could constitute as a strong influence factor regarding the general vehicle availability.

The class comprises of four important functions that are used throughout the simulation. First of all, it contains a function that removes a vehicle from the stations vehicle pool as soon as the customer starts the driving process. This is done by checking which vehicle the customer chose and removing the respective entry from the list of station vehicles.

Additionally, as the vehicles are removed when the customer starts the rental, they also have to be reinserted once the rental is over. This is performed by another function that
solely adds the rented vehicle at the end of vehicle list.

Charge Management

Lastly, two separate functions are introduced simulating the occupation of chargers when a customer ends his trip. As mentioned in the simulator class, it is assumed that customers prefer fast chargers. Therefore, when a trip ends, the number of available fast chargers is reduced by one. However, if they are all occupied, the number of normal chargers decreases.

Finally, a function describing the charging of vehicles is introduced. Vehicles are being charged solely when chargers are available and the current SOC is lower than the actual battery capacities.

4 Documentation

This paper aims at providing researchers and users alike with a convenient and easy to use car sharing simulator that allows for a thorough simulation of the car sharing process and opportunities for an uncomplicated customization and a user driven extension of the existing functions. Therefore, it is advisable to describe the implementation of the various functions which have been portrayed in the previous section, thus providing the base for a broad understanding of how the simulation functions. Furthermore, the assumptions concerning the numerous values in-use are introduced and, if available, verified by existing studies. This introduction begins with the simulator class, proceeds with the customer class, vehicle class and ends with the station class.

4.1 Simulator Class

As mentioned in the previous chapter, the simulator class constitutes the core of the simulation, as it contains the simulation process. However, it also deals with the initialization of the system which starts with the definition of the station grid:

```matlab
1 nStat = 22;
2 statCoords = zeros(2,nStat);
3 defaultstatCoords = [40 9; 50 376; 468 592; 365 472; 576 504; 361 129; 733 752; 681 572;
                      1247 703; 458 712; 748 445; 898 562; 804 486; 771 357; 843 314; 788 289; 835 152;
                      1001 459; 884 449; 1001 459; 1031 848; 893 694].’;
4 statIncMatrix = ones(nStat, nStat)-eye(nStat);
5 if length(statCoords) == length(defaultstatCoords)
6    statCoords = defaultstatCoords;
7 elseif length(statCoords) < length(defaultstatCoords)
8    statDiff = length(defaultstatstatCoords) - length(statCoords);
9    ```
Listing 1 Simulator.m/lines 6-41

It has to be noted here that all listings in this subsection refer to the simulator class named Simulator.m. The lines being referenced are the actual lines in the simulator class, thus allowing the reader to follow the text in the respective class on his computer.

The total number of stations is set by the variable \( nStat \), whereas their coordinates, being represented by \( x \) and \( y \) values, will be saved in the array called \( \text{statCoords} \) containing zeros, whose length is determined by the number of stations (line 1-2). However, since the station grid is supposed to be based on Volkswagen’s e-Car sharing service Quicar, default station coordinates were introduced (line 3). The variable \( \text{statIncMatrix} \) represents the connections between the stations which will be used to depict the roads of the customers (line 4). It is assumed that all stations are connected with one another, except for itself, since customers will never choose their starting station to be their destination. This is done by defining the variable \( \text{statIncMatrix} \) as an identity matrix and reversing its entries. As a consequence, all values on the main diagonal are set to 0, whereas all other elements will assume the value 1:

\[
\begin{bmatrix}
0 & 1 & \cdots & 1 \\
1 & 0 & \cdots & 1 \\
\vdots & \vdots & \ddots & \vdots \\
1 & \cdots & 1 & 0
\end{bmatrix}
\]

Figure 7 Reversed Identity Matrix
Since the user should be able to add or remove stations from the existing station grid, three possible cases are introduced. If the number of stations set by the user matches the number of default stations, the variable `statCoords` assumes the coordinates of the variable `defaultStatCoords` (line 6). In case the user picks a number of stations being lower than the default station grid, the length of the `statCoords`-array will also be lower than the length of the `defaultStatCoords`-array. Therefore, the number of default station coordinates will be reduced by the difference between both values, represented by the variable `statDiff`, consequently deleting the last entries in the respective array (lines 9-11). If the number of stations set by the user exceeds the number of default stations, a map of Hanover will be opened by the `imread`-command in Matlab’s Data Cursor Mode which is invoked in line 27. Since the map is too big to display in its entirety, it is resized to 97.5% of its original size. The Data Cursor Mode allows users to pick certain points in a figure and in this case, the map figure. Doing so will prompt Matlab to close the Data Cursor Mode and to save the coordinates in a variable called `cursor` (line 30). In the next step, a station variable called `newStat` is added, having the x and y values of the cursor’s last position (line 31). Subsequently, the array containing the default station coordinates is extended by 1 and this last entry assumes the values of `newStat` (line 32). Since the whole procedure is written in a for-loop, it will be repeated until the length of both station coordinate and default station coordinate arrays match. As soon as this happens, the `statCoords`-array assumes all values of the `defaultStatCoords`-array, thus assigning the coordinates to all stations (line 35).

Subsequently, many station and vehicle-related variables defining the car sharing system will be initialized:

```matlab
1 statDist = zeros(nStat);
2 statX1X2 = cell(nStat);
3 stations = cell(nStat,1);
4 capacities = 3;
5 sumCap = sum(capacities);
6 ccounter = 0;
7 cars = cell(sum(capacities),1);
8 vTypes = 3;
9 avCharge = zeros(vTypes,1);
10 cCars = 30000*length(cars);
11 nCharger = 3;
12 maxnCharger = nCharger;
13 fCharger = 1;
14 maxfCharger = fCharger;
15 priceperHour = 12;
```

Listing 2 Simulator.m/lines 43-58

Variable `statDist` is an array describing the distances between each station (line 1). As they are solely introduced here and computed later, it only contains zeros. `statX1X2` is a cell containing the vectors from one station to another, as these vectors will be used to display the roads in the car sharing network later on (line 2). Likewise to the
variable `stations` containing all station objects, it is set to 0 at the beginning and will be filled later (line 3). The variable `capacities` defines the vehicle capacities adopted by all stations (line 4). The variable `sumCap` contains the sum of all vehicles, thus it counts all vehicles at each station and sums them up (line 5). It will be needed during the simulation process and therefore, its purpose will be clarified later. Likewise, the variable `ccounter` is solely initialized here and assumes the value 0 (line 6). It will be used later on serving as a counter during each iteration. The variable `cars` is an empty cell containing all the vehicle objects being used in the simulation (line 7). `vTypes` sets the number of different vehicle types deployed in the car sharing system (line 8). The variable `avCharge` will serve as a performance indicator and describes the average state-of-charge of all vehicles throughout the simulation (line 9). `cCars` defines the average investment costs of each vehicle (line 10). Here, it is assumed that an electric vehicle costs about 30,000 euros, which is the average price of both the Mitsubishi I-MIEV and the BMW I-3 (Mitsubishi 2008; BMW 2013). The variables `nCharger` and `fCharger` define the number of normal chargers and fast chargers the stations will have (lines 11-14). `maxnCharger` and `maxfCharger` assume the values of the previous variables and are needed later, when the occupation of the chargers is modeled. `priceperHour` is the rental fee customers have to pay when renting a vehicle (line 15). It is set at 12 euros per hour, since the proposed car sharing system is supposed to match Quicar’s and therefore, its price policy.

Afterwards, the station objects are created and vehicles are assigned:

```matlab
x1 = statCoords(:,ii);
statCars = cell(capacities(ii),1);
carIds = zeros(capacities(ii),1);

for cc = 1:capacities(ii)
    ccounter = ccounter + 1;
cars{ccounter} = Car(ccounter);
statCars{cc} = cars{ccounter};
carIds(cc) = ccounter;
end

if cc <= ceil((1/3)*capacities(ii))
    vType = 1;
    statCars{cc}.setvehicleType(vType);
    statCars{cc}.setmaxCharge();
elseif cc <= ceil((2/3)*capacities(ii)) && cc >= floor((1/3)*capacities(ii))
    vType = 2;
    statCars{cc}.setvehicleType(vType);
    statCars{cc}.setmaxCharge();
elseif cc > ceil((2/3)*capacities(ii))
    vType = 3;
    statCars{cc}.setvehicleType(vType);
    statCars{cc}.setmaxCharge();
end

for jj = 1:nStat;
    x2 = statCoords(:,jj);
```
d = x2 - x1;
statDist(ii,jj) = norm(d,2);
statX1X2(ii,jj) = [x1, x2];
end
stations(ii) = Station(ii, x1, statCars, carIds, capacities(ii), nCharger, fCharger);
end

Since each station is created separately, a for-loop is introduced ranging from one to the maximum number of stations. Variable \(x1\) describes the coordinates of the station in the current iteration (line 1). \(\text{statCars}\) is a cell containing the number of capacities of this station, whereas \(\text{carIds}\) is an empty array having the length of the stations capacities (lines 2-3).

Both of the previous values are filled in the subsequent for loop (lines 5-25) ranging from one to the number of the individual stations vehicle capacities. \(\text{Ccounter}\) serves, as mentioned previously, as a counter (line 6). It increases each time the respective iteration advances. In line 8, the vehicle objects are created. Since the vehicle class requires an ID as input (see vehicle class), the counter \(\text{ccounter}\) will serve as such, i.e. when the iteration was performed five times, \(\text{ccounter}\) would equal that number. Thereby, a vehicle with ID = 5 would be created. Afterwards, the variable \(\text{statCars}\) is filled with vehicle objects, as it equals \(\text{cars}\) which contain, as previously described, the respective vehicle objects. Likewise, the variable \(\text{carIds}\), containing the vehicle IDs that are parked at the station, is set (line 9).

In the next step, the vehicle’s type is determined (line 11-23). As described in the previous section, percentages symbolizing thresholds are defined. In this case, it was assumed that each vehicle type makes up 33% of the stations total number of vehicles, because the proposed car sharing system uses three distinct vehicles. If the station has, for instance, a capacity of six vehicles, the first threshold would be two \((0.33*6)\). If it is the first iteration, iteration variable \(\text{cc}\) would equal one. Therefore, \(\text{cc}\) would be smaller than two and the respective vehicle is assigned vehicle type 1 (line 12) by calling the respective function in the vehicle class (line 13). Moreover, since the vehicle types were assigned, the function setting the vehicles maximum charge is also-called (line 14).

The next for-loop (line 26-31) is used to determine the distances between the stations. As this for-loop is always one iteration ahead of the whole for-loop, and therefore, the currently observed station, the station coordinates \(x2\) will be the coordinates of the following station (line 27). By subtracting both coordinates from one another, a vector \(d\) describing the distance on the x- and y-axis is obtained (line 28). Since vectors as station distances are unsuitable to predict the battery consumption during the trip, \(d\) is normed, transforming it into a one-dimensional value such as kilometers (line 29).
Thus, variable \( \text{statDist} \) which was only initialized as an empty array beforehand, is filled with the required values.\(^2\) Likewise, \( \text{statX1X2} \) containing the connections between the stations is set, although in this case, each entry of the array encompasses the coordinates of both stations (line 30). Consequently, each entry describes one connection.

Finally, the constructor method of the station class is called (line 33). It creates the station objects that need, as can be drawn from the previous chapter or the station class itself, IDs (\( ii \)), coordinates (\( x1 \)), station cars (\( \text{statCars} \)), station car IDs (\( \text{carIds} \)), station capacities (\( \text{capacities}(ii) \)), number of normal (\( n\text{Charger} \)), and fast chargers (\( f\text{Charger} \)).

After setting up the station grid and creating the station objects, customer-related variables are determined or initialized:

```
1 nCust = 1000;
2 sumdayrentals = zeros(1,1);
3 sumnightrentals = zeros(1,1);
4 customers = cell(nCust,1);
5 currentpos = NaN*zeros(2,nCust);
6 custcars = cell(1,nCust);
7 custids = cell(1,nCust);
8 custminInts = cell(2,nCust);
9 sumtotaldayrentals = zeros(1,nCust);
10 sumtotalnightrentals = zeros(1,nCust);
11 walkdistances = zeros(1,nStat);
12 durStay = zeros(1,nCust);
```

Listing 4 Simulator.m/lines 95-106

Variable \( n\text{Cust} \) determines the number of members of the car sharing system and is one of the influence variables which was described in the previous section (line 1). \( \text{sumdayrentals} \) and \( \text{sumnightrentals} \) are variables that describe either the sum of day or night rentals of each customer (lines 2-3). \( \text{Customers} \) is a cell containing the customer objects whereas \( \text{currentpos} \) is an array filled with all the customers current positions (lines 4-5). However, it is initialized here being filled with NaN as entries and is updated later on during the simulation process when the customers are driving. \( \text{Custcars} \) is an array containing the IDs of the vehicles the customers are using throughout their rental (line 6). It is used during the simulation to depict the car IDs next to the customer IDs on the map. \( \text{CustminInts} \) is an array that consists of the customer’s minimum time intervals that have to pass between each rental although it is solely used for observational purposes (line 8). \( \text{sumtotaldayrentals} \) and \( \text{sumtotalnightrentals} \) are performance indicators that calculate the sum of each day and night rentals among all customers (lines 9-11). The array \( \text{walkdistances} \) contains the distances a customer has to walk to a station which will be needed later on when the customer checks for suitable car sharing stations (line 11).\(^3\) Likewise \( \text{durStay} \) describes the durations of the customer’s stops during the trip.

---

\(^2\)see listing 2

\(^3\)see listing 11
However, both variables are solely initialized here and will be filled later on (line 12).4

In the next step, the customer class objects are created:

```matlab
for ii = 1:nCust;
    customers(ii) = Customer(ii);
    customers(ii).setcustTypeDistrib(nCust);
    customers(ii).setfirstDrive();
    customers(ii).setplaceofLiving(nCust);
    customers(ii).setmaxwalkDist();
    customers(ii).setmaxdurStay();
    customers(ii).setprefCar();
    customers(ii).setmaxwaitTimetolerance();
    custminInts{1, ii} = customers(ii).minIntervalday;
    custminInts{2, ii} = customers(ii).minIntervalnight;
end
```

Listing 5 Simulator.m/lines 108-119

First of all, the constructor method of the customer class is called (line 2). It requires the customer IDs which are drawn from the iteration variable of the for-loop (lines 1-12) and subsequently, creates the customer objects and each customer with all his properties. All the following functions (lines 3-11) could as well be avoided and handled by the constructor method. However, the author chose to implement separate functions so that the user of the simulator can easily distinguish between them. Each of these functions sets an individual property of the customer class such as the maximum walking distance. However, these functions will be presented in their respective class.

The following segment deals with plotting the stations and their connections:

```matlab
plotting = 1;
hf = figure();
[B, map] = imread('Hannoverosm.png');
C = imresize(B, 0.975);
imshow(C, map)
tempEdges = statX1X2(:,);
plotEdges = zeros(2,3*length(tempEdges));
for ii = 1:length(tempEdges);
    plotEdges(:,ii*3-2) = tempEdges{ii}(:,1);
    plotEdges(:,ii*3-1) = tempEdges{ii}(:,2);
    plotEdges(:,ii*3) = NaN;
end
hold on;
plot(plotEdges(1,:), plotEdges(2,:), 'Color', [0 0 0], 'Linestyle', ':');
for gg = 1:nStat;
    plot(statCoords(1,:), statCoords(2,:),...
    'LineStyle', 'none', 'Marker', 'o', 'MarkerFaceColor', 'g');
end
posplot = plot(NaN,NaN,'LineStyle', 'none', 'Marker', 'o', 'MarkerFaceColor', 'w');
hold off;
```

Listing 6 Simulator.m/lines 124-147

4see listing 15
As mentioned in the previous section, a variable called `plotting` is introduced to allow customers to turn off the live-plotting features (line 1). Since they require most of the computational power, users can disable them to accelerate the simulation process. Afterwards, a figure is initialized and the map of Hanover is read and resized (lines 2-5). The hold-command is a special command being used when plotting data (line 13). It is required to save the values during each iteration. Otherwise, each station and their connections would be plotted only once in the respective iteration step. Consequently, the graphical output of the station grid would eventually consist of solely one station which is to be avoided. The plot-commands are used to plot the connections between the stations (line 14) and the stations themselves (line 17).

From line six to eleven, the connections between the stations are set. First of all, a variable called `tempEdges` is introduced. Each entry in this array contains the coordinates of two stations stemming from the variable `statX1X2` (line 6). In the next step, a for-loop is introduced that calculates the vectors between the stations in each entry in the respective variable (lines 8-12). However, since the map in-use has only two axes, the third vector value describing the z-axis is set as NaN. In both cases, two values are plotted (lines 14+17). The first value represents the x-coordinates, whereas the second value describes the y-coordinates. The logic can be derived from the following figure:

Figure 8 Plotting the Station Grid

Variable `posplot` (line 22) is initialized here being filled with solely NaN-entries. It is used and updated thereafter, when the driving processes of the customers are depicted on the map by displaying their current positions. Thus, it will encompass the current positions of all customers.

Following the plotting of the stations and their connections, the plot visualizing the revenue of the car sharing system will be initialized:

```plaintext
if plotting == 1
    hold on;
    hf2 = figure();
    revplot = plot(NaN, NaN);
    title('Revenue');
    xlabel('time');
    ylabel('revenue');
    hf3 = figure();
    hold on;
    dayPlot = plot(NaN,NaN,'Color','g');
    nightPlot = plot(NaN,NaN,'Color','k');
    hold off;
```

39
Like the previously mentioned posplot, the revenue plot is solely initialized here, as the revenues accrue during the simulation process (line 4). The x-axis will depend on the simulation time, whereas the y-axis displays the revenue being generated (line 5-7).

Furthermore, to facilitate the examination of results, so-called day and night plots are initialized (line 10-11). These permit to graphic differentiations between day and night times during plotting. This feature is required later on when the plot depicting the number of currently driving customers is issued. The graph is defined here as assuming the color green during the day and black throughout the night.

Afterwards, diverse variables needed for the simulation process are initialized:

```plaintext
1 revenue = 0;
2 costs = 0;
3 ttnight = 0;
4 ttday = 540;
5 waittime = 0;
6 startT = 0*24*60;
7 endT = 14*24*60;
8 time = startT:endT;
9 countDriving = 0;
10 carDeficit = 0;
11 carChargeDeficit = 0;
12 revperMin = 0;
```

Both the revenue and the costs of the car sharing system assume the value 0, as none of them have occurred before the simulation begins (lines 1-2). Ttnight and ttday are two very important variables of the system that are required when the customer’s desire to rent a vehicle is determined (lines 3-4). The simulation time tt advances by one minute independently whether it is day or night, whereas ttnight and ttday only increase, when it is either day or night. The respective scenario is depicted in figure 8.

![Simulation Time](image)

Figure 9 Simulation Time

Because the simulation starts at midnight, ttnight is set as zero (line 3). ttday is initialized as 540, since the first day of the simulation starts at 9 a.m. (line 7). Variable
waittime sets the customer’s waiting time to zero, since at the beginning of the simulation no customer would have waited. StartT and endT define the start and end time of the simulation (line 6-7). The simulation starts at minute 0, which is midnight and ends after 14 days, since the simulation is designed to simulate two weeks of operating the car sharing system. Variable time defines the time interval during which the simulation will be running, as it describes the interval between the start and end time (line 8). Countdriving, carDeficit, and carChargeDeficit are variables that will be used to display the current number of driving customers, the number of customers that could not drive because no vehicle was available and the number of customers that could not drive because no vehicle had enough charge for the journey (lines 9-11). RevperMin describes the revenue being generated each minute. As it is solely initialized here, it will be updated during the simulation.\footnote{see listing 13}

After the set-up of the car sharing system and the initialization of the plotting features, the actual simulation process starts:

\begin{verbatim}
for ii = 1:length(time)-1
    tt = time(ii);
    minuteOfDay = mod(tt,60*24);
    hourOfDay = floor(minuteOfDay/60);
    fprintf('Time : %f years , %i months , %i days , %i hours
', floor(tt/365/24/60), floor(tt/730/60), floor(tt/24/60), hourOfDay)

    if floor((tt-1)/730/60) < floor(tt/730/60)  % 730 hours per month
        revenue = revenue + nCust*0;
    end

    deltaT = time(ii+1)-tt;

    if hourOfDay > 20 || hourOfDay < 9
        ttnight = ttnight + 1;
    else
        ttday = ttday + 1;
    end
end
\end{verbatim}

Listing 9 Simulator.m/lines 175-191

As mentioned before, the whole process is depicted in a for-loop that advances one minute at a time during each iteration. It starts at minute one and ends when the simulation time interval time is reached (line 1). Variable tt records the current simulation time, whereas MinuteOfDay determines the minute of the day the simulation is currently in (lines 2-3). This is done using the so-called modulus of tt. Mod(tt, y) is:

\[
\text{mod}(tt, y) = tt - n \times y
\]  \hfill (1)

where

\[
n = \text{floor}(tt/y) \text{ if } y = 0.
\]  \hfill (2)
For instance, if the simulation time $tt=2300$ and $y=60*24$, since the minute is to be determined, $\text{minuteOfDay}$ would be 906. Dividing $\text{minuteOfDay}$ by 60 returns the exact hour of the day $\text{hourOfDay}$. In this case, it would be 3 p.m.

Afterwards, by using the fprintf command, the year, month, day, and hours are displayed in Matlab’s command window (line 7). Thus, the user can always examine the respective values during the simulation, even when plotting is completely turned off.

From line seven to nine, the monthly usage fees are calculated. At the beginning of each month which is also determined by using the modulus, each customer will pay a fixed fee that differs between each car sharing company. Since Quicar does not charge their users for being members of their service, it is set to 0. However, as this simulator should be easily customizable for the application in other car sharing services, the feature was implemented.

In the next step, variables $\text{ttnight}$ and $\text{ttday}$ are increased. As mentioned before, the former only increases during the night, whereas the latter increases during the day (lines 13-17). This is determined by checking the variable $\text{hourOfDay}$. If the hour of day is higher than 20 and lower than 9, $\text{ttnight}$ increases. Therefore, it is assumed that between 9 p.m. and 9 a.m. it is night, whereas the day period lasts from 9 a.m. to 8 p.m.

Subsequently, the customer’s desire to rent a vehicle is determined:

```
for cc = 1:nCust
    customers(cc).setminInt(hourOfDay, priceperHour);
    custminInts(1, cc) = customers(cc).minIntervalday;
    custminInts(2, cc) = customers(cc).minIntervalnight;
    if ~customers(cc).isDriving
        if hourOfDay > 20 || hourOfDay < 9
            hasNotDriven = customers(cc).minIntervalnight < ttnight - customers(cc).lastNightDrive;
        else
            hasNotDriven = customers(cc).minIntervalday < ttday - customers(cc).lastDayDrive
        end
    end
```

Listing 10 Simulator.m/lines 193-203

First of all, the customer class function is called where the minimum time intervals having to pass between each rental are set (line 2). Furthermore, the respective values of all customers are saved in an array for examination purposes (line 3-4). The minimum intervals are split between day and night, since the customers are supposed to have different driving patterns (i.e. frequency of driving) during both.

If the current customer is not driving, the simulator checks if he wants to rent a vehicle or not. This is done by determining whether the current minimum interval is lower than the current simulation time minus the last time of rental (line 7-11). The strict differentiation between $\text{ttnight}$ and $\text{ttday}$ is required to avoid undesirable driving behaviors. In case the simulator would solely use $tt$ instead of the other two variables, many of the customers
would immediately start driving as soon as it either turns day or night, thus creating unwanted peaks.\footnote{see appendix 2 for example}

The reason for this behavior is that $tt$ increases both during day and night. Therefore, it might come to pass that during the night, the $minintervalDay$ becomes smaller than $tt$ minus $lastDayDrive$. Therefore, the customer would want to rent a vehicle, but since it is still night, he is required to wait until the system switches from night to day. As soon as this happens, he would immediately start driving.

If the previous conditions are met, variable $hasNotDriven$ assumes 1 and the process proceeds:

```matlab
if hasNotDriven
    for hh = 1:nStat
        walkdistances(1, hh) = norm(customers(cc).pLiving - stations(hh).coords); % computes distances between cust. and all stations
    end
    minwalk = min(walkdistances);
    if minwalk/85 < customers(cc).maxdistKM % if the distance to the nearest station is lower than the max walk distance, the customer found his station
        [fS] = find(walkdistances==minwalk);
        tS = randi(nStat);
        fromStat = stations(fS);
        toStat = stations(tS);
    
Listing 11 Simulator.m/lines 205-217
```

As soon as the customer wants to drive, the distances between his residence and all stations is calculated (line 4). This is done by subtracting the station coordinates from the customers residence coordinates. Since the resulting vector is unfeasible to describe the distance, because a distance is a one dimensional value, it is normed. Afterwards, the lowest value from the array $walkdistances$ is picked and saved in the variable $minwalk$ (line 7).

Subsequently, it is compared with the customer attribute $maxdistKM$ describing his maximum walking distance (line 9). Since coordinates in figures and therefore, the map in-use, are displayed in pixels, the walking distance $minwalk$ is converted to kilometers by dividing it by 85. Thus, 85 pixels are set to be one kilometer. If the distance to the nearest station $minwalk$ is lower than the maximum walking distance $maxdistKM$, the customer is willing to go there.

In lines 10-11 the indices $fS$ and $tS$ are introduced. They are used to identify the exact station objects from the list of station objects. The starting station index $fS$ is determined by finding the number of the entry, where the previously calculated walking distance $walkdistances$ matches the minimum walking distance $minwalk$. For instance,
if the minimum walking distance is between the customer’s residence and station 6, \( fS \) would assume the value 6. In this case, his starting station fromStat would be stations\{6\} and thus, stations object six with all its respective properties (line 12). The destination station index \( tS \) and thus, the destination station object, is chosen at random (line 13).

After determining the starting point and the customer’s destination, the driving process is initiated:

```java
if fromStat.id ~= toStat.id;
    successDriving = customers(cc).letDrive(fromStat, toStat, statDist(fS,tS));
    if successDriving == -1
        customers(cc).actualwaitTime = customers(cc).actualwaitTime + 1;
    if hourOfDay > 20 || hourOfDay < 9
        daydriving = 0;
    else
        daydriving = 1;
    end
carDeficit = carDeficit + 1;
    if customers(cc).actualwaitTime > customers(cc).maxwaitTime
        customers(cc).setlastdrive(tt,daydriving);
        customers(cc).dDrive = 0;
        sumCarDeficit = sumCarDeficit + 1;
    end
elseif successDriving == 0
    customers(cc).actualwaitTime = customers(cc).actualwaitTime + 1;
    if hourOfDay > 20 || hourOfDay < 9
        daydriving = 0;
    else
        daydriving = 1;
    end
carChargeDeficit = carChargeDeficit + 1;
    if customers(cc).actualwaitTime > customers(cc).maxwaitTime % if the customer waited longer than his max wait time, he loses his intention of driving
        customers(cc).setlastdrive(tt,daydriving);
        customers(cc).dDrive = 0;
        sumChargeDeficit = sumChargeDeficit + 1;
    end
```

Listing 12 Simulator.m/lines 219-251

First of all, the system checks that the ID of the customer’s starting station and destination match do not match (line 1). This is done to ensure that customers do not choose their starting station to be their destination. Following that, a customer class function named letDrive is called (line 2). In short, this routine is used to determine the availability of cars. Three different cases are defined within this separate customer function whose outcomes are saved in the variable successDriving.\(^9\) In case no vehicle is available because they are all in use, successDriving will assume the value -1 (line

\(^9\)see listing 37
3). Likewise, when all vehicles do not have enough charge for the journey, successDriving assumes the value 0 (line 19). Lastly, if the customer finds an appropriate vehicle, successDriving assumes the value 1 (line . However, if successDriving equals either -1 or 0, the customer starts to wait and actualwaittime increases by one in each iteration (lines 5+21). As soon as the actualwaittime exceeds the customer’s maximum wait time maxwaitTime, the customer aborts the rental process (lines 14+30). In these cases, it is assumed that the customer’s desire to rent a vehicle expires and his desire to drive dDrive is set to 0. Therefore, the current simulation time will be set as the time of his last rental by calling the customer class function setlastDrive. However, because the last rental differentiates between last day rental and last night rental, the respective function requires an input whether it is day or night. This is done by introducing the variable daydriving which assumes the value 1 during the day and 0 if its night (line 7-11 and 23-27). Finally, in each of the two unsuccessful cases, counters are introduced that count the number of times each case occurred. Thus, sumChargeDeficit and sumCarDeficit serve as performance indicators which can be used during the evaluation (lines 16+32).

If an appropriate vehicle is found by using the customer class function letdrive and successDriving assumes the value 1, the driving process is initiated:

```matlab
1 % elseif successDriving == 1
2     if hourOfDay > 20 || hourOfDay < 9
3         daydriving = 0;
4     else
5         daydriving = 1;
6     end
7
8     if fromStat.nCharger < maxnCharger && fromStat.fCharger == 0
9         fromStat.nCharger = fromStat.nCharger + 1;
10     elseif fromStat.fCharger < maxfCharger
11         fromStat.fCharger = fromStat.fCharger + 1;
12     end
13
14     fromStat.removeCar(customers{cc}.myCar);
15     countDriving = countDriving +1;
16     x1 = fromStat.coords;
17     x2 = toStat.coords;
18     d = x2-x1;
19     distKM = d/85;
20     v = (50+20*randn(1))/2;
21     drivedistancevector = v*d/norm(d,2)*deltaT;
22     currentpos(1:2, cc) = x1 + drivedistancevector;
23     customers{cc}.updatepos(currentpos(1:2, cc));
24     custids{cc} = sprintf('%i\n%i', customers{cc}.id, customers{cc}.myCar.id,
25         customers{cc}.custType);
26     custcars{cc} = customers{cc}.myCar.id;
27     revperMin = revperMin + priceperHour/60;
28     customers{cc}.myCar.drive(norm(drivedistancevector));
29     end
30
31 % Customer drop-out the system
32 else
33     % Customer drop-out the system
```
Yet again, it is determined whether the driving was initiated during the day or night (line 2-6). This is done to ensure that a day rental is always identified as such, even when the rental starts during the day and ends during the night. Without the `daydriving` variable, such a trip would be counted as a night rental, since the customer’s last rental would be set during the night. As a consequence, the customer would then immediately start driving again when it turns day, since the condition whether he wants to drive or not is still met. This, in turn, leads to unrealistic driving peaks during the morning hours, like the ones being depicted in appendix 2 which are to be avoided.

As mentioned in the previous section, it is assumed that the vehicles are automatically charged when they are parked at a station. Therefore, the chargers have to be set as available, as soon as the vehicle leaves the station. If the number of available normal chargers is lower than the number of normal chargers the station actually has `maxnChargers` and all fast chargers are occupied, the number of normal chargers will increase by one (lines 8-9). If the number of available fast chargers is lower than the station’s maximum number of fast chargers `maxfCharger`, the number of available fast chargers will increase by one (lines 10-11).

In line 14 the customers current vehicle is removed from the list of the starting station’s available vehicles by calling the station class function `removeCar` which will be described in the corresponding section later on. Like `chargeCarDeficit` and `carDeficit`, `countDriving` is a counter, but in this case it keeps track of the number of driving vehicles (line 15). It is both used for display in Matlab’s Command Window and during the live-plotting.

From line 16-18 the distance of the trip is predicted. In line 20, the velocity of the vehicles in each iteration is determined. It is assumed that the general speed limit in cities and thus, the driving speed is set to 50 km/h. However, some roads have higher speed limits, consequently increasing the driving speed, and some factors, such as congestion, lead to lower driving speeds. This variability is taken into account by multiplying a fixed value, in this case 20, with a normally distributed number drawn from the standard normal distribution and adding it to the fixed driving speed (line 20). As normally distributed values can either assume positive or negative values, the fixed driving speed decreases or increases variably. In addition, a corrective factor that divides the previous results by two is introduced. It is needed to adjust the driving speed to a more realistic level because generally, vehicles drive about 25-30 km/h in urban traffic (cf. André and Hammarström 2000, p. 328). However, orienting oneself towards the speed limits and using this corrective factor, as it is done in this case, allows for the easy implementation

\[\text{10 see listing 50}\]
of various road types with varying speed limits.

Afterwards, the distance being covered in one iteration is determined by calculating the `drivedistancevector` (line 21). Subsequently, the customer’s current position is first determined and then updated by adding the traveled distance to the starting station’s coordinates $x1$ (line 22-23). `Custids` and `custcars` are arrays containing the IDs of the driving customers and vehicles. The former is used to display the customers on the overview map allowing the user to see who is currently driving to which destination, whereas the latter is solely implemented for observational purposes overview. `RevenueMin`, however, describes the revenue generated each minute and is calculated by dividing the hourly rental fees by 60 (line 26).

Finally, the `drive` function from the vehicle class is called (line 27). It is used to simulate the consumption of electricity and the lowering of the SOC throughout the trip. The else-case in line 31-33 references to the determination whether a station is in range or not. If it is not, the customer property `dropOut` is set to true, indicating that the customer will never use the car sharing system. The respective property is used later on as a performance indicator.

However, if `hasNotDriven`=0, meaning that the customer does not want to drive, three variables need to be updated:

```matlab
else % customer does not want to drive
    currentpos(1:2, cc) = NaN;
    custids(1,cc) = NaN;
    custcars(1,cc) = NaN;
end
```

Listing 14 Simulator.m/lines 287-291

First of all, the current position `currentpos` is set to NaN to ensure that the customers will not be shown on the map. Moreover, the IDs of the driving customers `custids` and vehicles `custcars` are set to NaN.

Up to this point it was defined what happens, when a customer is not driving. The following code describes the simulation behavior while the customer is already driving. However, there are three possible driving cases being relatively similar to the previous scenario and to each other. Therefore, some lines of the following code that are repeatedly used will not be described again:

```matlab
else % if driving
    myCar = customers(cc).myCar;
    countDriving = countDriving +1;
    x1 = customers(cc).fromStation;
    x2 = customers(cc).toStation;
    fprintf('Customer %i: Driving\n', cc);
    d = x2-x1;
```

11 see listing 11
12 see listing 10
The first case describes the situation when a customer drives to his destination and reaches it. It is assumed that each customer drives a random portion of the maximum distance between the stations. Here, the maximum drivable distance is set as 90% of the complete distance. As soon as the customer’s current position matches the coordinates of his destination, he stops driving and his current position is updated (lines 11-13). While the customer is performing his tasks, the revenue being created is decreased by 30% and the variable durstay that describes the customer’s duration of stay, increases by 1 (line 14-15). When durstay matches a random portion of the customer’s maximum duration of stay, thus simulating a variable behavior, the customer class function returnDrive is called (line 16-17). The function is is similar to the letDrive function except for the destination station and the starting point which are reversed. Therefore, the starting station will be set as the customer’s destination, whereas his current position defines the starting point of the return trip. Additionally, the respective function sets the property return from the customer class to 1.

The behavior during the return trip is defined in the following lines:

```matlab
if ~customers(cc).ret && norm(currentpos(:, cc)-x1) > max(rand(1)*norm(d), 0.9*norm(d))
    currentpos(1:2, cc) = currentpos(1:2, cc);
    customers(cc).updatepos(currentpos(1:2, cc));
    revperMin = revperMin + (priceperHour/60)*0.7;
    durStay(cc) = durStay(cc) + 1;
    if durStay(cc) >= rand(1)*customers(cc).maxdurofStay
        customers(cc).returnDrive(currentpos(:, cc));
        v = (50+20*randn(1))/2;
        drivedistancevector = v*d/norm(d, 2)*deltaT;
        currentpos(1:2, cc) = currentpos(1:2, cc) + drivedistancevector;
        customers(cc).updatepos(currentpos(1:2, cc));
        revperMin = revperMin + priceperHour/60;
        customers(cc).myCar.drive(norm(drivedistancevector));
    end
end
```

Listing 15 Simulator.m/lines 293-316

If the customer is returning (customers.ret = 1) and his current position does not match his destination coordinates yet, he will drive a certain distance drivedistancevector with

---

13 see listing 39
a certain velocity $v$ (line 2-3). Again, his positions will be determined and updated (line 4-5), revenue is being generated, and the $drive$ function from the vehicle class simulating the consumption of battery charges is called (line 7-8).

The final case describes the situation, when the customer terminates his trip upon reaching his starting station:

```
% customer reaches starting station
elseif customers{cc}.ret && norm(currentpos(:, cc)-x1) > norm(d)
   customers{cc}.setlastdrive(tt, daydriving);

% reduces number of chargers when car is parked and charged
if fromStat.fCharger > 0
   fromStat.fCharger = fromStat.fCharger - 1;
elseif fromStat.nCharger > 0
   fromStat.nCharger = fromStat.nCharger - 1;
end
fromStat.addCar(myCar);
customers{cc}.stopDrive(sundayrentals, sumnightrentals, hourOfDay);
```

Listing 17 Simulator.m/lines 327-339

First of all, the customer class function $setlastDrive$ is called once again (line 3). Depending on whether the trip started during the day or night, which is determined by the variable $daydriving$, the customer property $lastdayrental$ or $lastnightrental$ is set to the current simulation time $tt$. Afterwards, the occupation of charges is handled. If the station has fast chargers available, the number of fast chargers will be reduced by 1, as it is assumed that customers will always choose them first (lines 6-7). However, if no fast chargers are available, a normal charger is occupied and the corresponding variable decreases by one (lines 9-10).

Finally, the customer class function $stopdrive$ and the station class function $addCar$ are called (lines 12-13). The former resets many customer-related properties, whereas the latter adds the current vehicle object to the starting stations vehicle pool, thus terminating the whole rental process.

At this point, the customer’s interactions with the system end and the simulation process is almost complete except for various plotting features and battery related functions:

```
for hh = 1:sumCap
   cars{hh}.detCharge();
   vCharges(hh,tt+1) = cars{hh}.currcharge;
end

% charging + plotting
for gg = 1:nStat
   stations{gg}.chargeCars(deltaT, fCharger, nCharger);
   set(0, 'CurrentFigure', hf)
   hold on;
```

\[14\text{ see listing 40 + 51}\]
if plotting == 1
    % show availability of cars
    if isempty(stations(gg).cars) == 1;
        plot(stations(gg).coords(1,:), stations(gg).coords(2,:),...
            'LineStyle', 'none', 'Marker', 'o', 'MarkerFaceColor', 'g');
    else
        plot(stations(gg).coords(1,:), stations(gg).coords(2,:),...
            'LineStyle', 'none', 'Marker', 'o', 'MarkerFaceColor', 'r');
    end
end
hold off;

As mentioned in the specific concept, it is intended to depict the continuous deterioration of the battery capacities amongst all vehicles. This is done by writing a loop encompassing all vehicles and calling their vehicle class function \texttt{detCharge}, which decreases the vehicle object’s property \texttt{maxCharge} in each iteration by a fixed value (lines 1-2). Moreover, \texttt{vCharges}, being an array, continuously saves the current charges of all vehicles (line 3). It is used later to determine the average charges among all iterations.\footnote{see listing 25}

Afterwards, the charging of vehicles at all stations is initiated by calling the station class function \texttt{chargeCars} (line 8).\footnote{see listing 53} As the charging speed depends on the number of fast and normal chargers, as well as the time interval between each iteration, \texttt{deltaT}, \texttt{fCharger} and \texttt{nCharger} are required as input variables.

From lines 10 to 20 a traffic light system is introduced that gives visual information about the availability of vehicles. While plotting the overview map depicting the stations and driving customers, the colors of the stations change according to whether they are currently empty or not. In case that a station is not currently empty (means not) the station will be displayed in green (lines 14-15). Likewise, the stations turn red, when all vehicles are currently in use and therefore, not parked at the station (lines 17-18).

In the next step, the total sum of both day and night rentals is determined:

\begin{verbatim}
for hh = 1:nCust
    sumtotaldayrentals(1, cc) = sum(customers(cc).sumdayrentals); % saves the sum of each customers rentals
    totalrentals = sum(sumtotaldayrentals); % sums up the sum of all customers rentals (daytime)
    sumtotalnightrentals(1, cc) = sum(customers(cc).sumnightrentals);
    totalnightrentals = sum(sumtotalnightrentals);
    if customers(cc).dropOut == true;
        dropOuts = dropOuts + 1;
    end
    if customers(cc).dDrive
        waiting = waiting + 1;
    end
    sumWaitTime = sumWaitTime + customers(cc).actualwaitTime;
end
\end{verbatim}
First, the sum of day and night rentals of each customer is evaluated. The respective values are saved individually in the array `sumtotaldayrentals` or `sumtotalnightrentals` (lines 2+4). In order to receive the total number of rentals among all customers, the sum of both distinct arrays has to be determined. Doing so returns the variables `totaldayrentals` and `totalnightrentals` that can be used as performance indicators later on (lines 4-5).

Furthermore, a counter is introduced that examines all customers and increases according to whether customers dropped out of the system due unacceptable walking distances. If the respective customer property is set to `true`, the counter increases by 1 (line 7). Additionally, the number of waiting customers is determined. The customer property `dDrive` describes, whether a customer requests a vehicle or not (line 9). It is set to 0 as soon as the customer found an appropriate vehicle. However, if he did not find a vehicle during the iteration, the property remains 1. In this case, the counter `waiting` increases by 1 (lines 9-10). In line 12, the total sum of wait times of all customers is determined by checking the respective customer properties. From lines 15-19, the average wait time is calculated. For that purpose, both variables `sumWaitTime` and `waiting` are required. By dividing the sum of the wait times `sumWaitTime` by the number of waiting customers `waiting`, the average wait time `avgWaitTime` is acquired. However, if no one is waiting, the average wait time is set to 0.

In this system, it is assumed that the costs will consist of both fixed and variable costs. The fixed costs are mainly marketing, administrative and maintenance costs (line 2-4). The values of the marketing and maintenance costs are fictional, whereas the maintenance costs account for 10.73 euros per vehicle per day (line 2) (cf. Falko Koetter 2012, p. 51).

---

% hourly cost
maintenanceCost = sum(capacities)*10.73/24/60;
administrationOverhead = 10000/30/24/60;
marketingOverhead = 10000/30/24/60;
stationRent = sum(capacities) * 30/30/24/60;
nEmployees = 30;
labourCost = nEmployee/60000/365/24/60;
sumCost = cCars + maintenanceCost + administrationOverhead + marketingOverhead +
stationRent + labourCost;
costs = costs + sumCost;
32). Regarding the station rent, it is assumed that it is directly connected to its vehicle capacities accounting for 30 euros for each vehicle (line 5). The more vehicle capacities a station has, the bigger the station needs to be, thus increasing the rent. The labor costs are directly connected with the number of employees \( n_{Employees} \). It is assumed that each employee earns 60,000 euros a year, as car sharing companies supposedly employ mainly academicians especially when no operator-based relocation strategy is applied (line 7). Finally, the overall costs of the system consist of both the investment costs for the vehicles \( c_{Cars} \) and the running expenses that were described previously (line 9).

Since most of the plotting features were solely initialized before, the following lines describe the code being used to display the IDs and the current positions of the customers:

```matlab
if plotting == 1
    set(0,'CurrentFigure',hf)
    hold on;
    try
        cIdx = ~isnanCell(custids(1,:));
    catch ex
        disp(ex);
    end
    set(posplot,'xdata',currentpos(1,:),'ydata',currentpos(2,:));
    if any(cIdx)
        if exist('cTextPlot') > 0
            delete(cTextPlot);
        end
        cTextPlot = text(currentpos(1,cIdx)+10, currentpos(2,cIdx)+10, custids(cIdx),'
        FontWeight', 'bold', 'BackgroundColor', [1 1 1]);
    else
        cTextPlot = text();
    end
    hold off;
end
```

Listing 21 Simulator.m/lines 408-427

First of all, the figure displaying the map is called (line 2). Next, a variable \( cIdx \) is introduced retrieving all entries from the \( custids \) array that are not set to NaN (line 6). In line 10, the previously initialized \( posplot \) is used to display the customer’s current positions.\(^\text{18}\) However, if any customers are driving (line 11), their IDs will be displayed next to their current position by the \( cTextPlot \) (line 15). Subsequently, as the the customers are changing their position during each iteration, the position of the IDs also has to be adjusted repeatedly. Therefore, in each iteration it is checked if any \( cTextPlot \) already exists (line 12). If it does, it gets deleted, thus allowing the simulator to update it for the newer one (line 15). If no vehicle is currently driving, an empty and therefore invisible text is depicted on the map by the \( cTextPlot \) (line 17).

Besides showing the vehicle and customer IDs on the map, the profit plot \( profPlot \) is to be filled with data:

\(^{18}\)see listing 6
if plotting == 1
    set(0, 'CurrentFigure', hf2)
    hold on;
    set(profPlot, 'xdata', [get(profPlot, 'xdata') tt], 'ydata', [get(profPlot, 'ydata')
        revenue - costs]);
    hold off;
end

Listing 22 Simulator.m/lines 429-434

The x-axis on the corresponding figure being shown in Matlab will display the simulation
time tt. The y-axis will depict the profit of the car sharing company during each iteration
which is determined by subtracting the variable costs from variable revenue (line 4).

Next, the plotting of the number of driving customers is settled:

if plotting == 1
    set(0, 'CurrentFigure', hf3)
    hold on;
    if mod(tt, 10) == 0
        if hourOfDay > 20 || hourOfDay < 9 % night
            set(nightPlot, 'xdata', [get(nightPlot, 'xdata') tt], 'ydata', [get(nightPlot, '
                ydata') countDriving]);
        else
            set(dayPlot, 'xdata', [get(dayPlot, 'xdata') NaN], 'ydata', [get(dayPlot, '
                ydata') NaN]);
        end
    end
    else % day
        if floor(mod(tt - 1, 24/60) / 60) > 20 || floor(mod(tt - 1, 24/60) / 60) < 9 % if last
            set(nightPlot, 'xdata', [get(nightPlot, 'xdata') tt NaN], 'ydata', [get(nightPlot, '
                ydata') NaN]);
        else
            set(nightPlot, 'xdata', [get(nightPlot, 'xdata') NaN], 'ydata', [get(nightPlot, '
                ydata') NaN]);
        end
        set(dayPlot, 'xdata', [get(dayPlot, 'xdata') NaN], 'ydata', [get(dayPlot, 'ydata'
                ) NaN]);
    end
    drawnow;
end
hold off;

Listing 23 Simulator.m/lines 436-461

As mentioned before, the plot will differentiate between day and night times by changing
the colors of the figure. During the day, the number of driving customers will be depicted
in green, whereas during the night, it is displayed in black.\textsuperscript{19} The \textit{nightPlot} is used

\textsuperscript{19}see listing 7

53
throughout the night. It returns a figure where the x-axis displays the simulation time \(tt\), whereas the y-axis shows the number of driving customers (line 7). Likewise, the \textit{dayPlot} is structured (line 20).

In order to prevent the simulation from displaying either the \textit{nightPlot} during the day or the \textit{dayPlot} during the night, the values of the respective plot are switched to NaN. Thereby, one of the plots has no values to present and thus, is invisible while the other plot is in-use (lines 9-13 & 15-19).

Furthermore, various outputs that are going to be displayed during the simulation are defined:

\begin{verbatim}
1 fprintf ('Number of driving customers: %i\n', countDriving);
2 fprintf ('Number of requests: %i\n', wishToDrive);
3 fprintf ('Number of waiting customers: %i\n', waiting);
4 fprintf ('Number of customers who didn\'t find a car: %i\n', noSuccess);
5 fprintf ('Number of customers turned away: %i due to missing cars, %i due to too little charge\n', carDeficit, carChargeDeficit);
6 fprintf ('Hourly revenue %f, hourly cost: %f\n', revperMin, sumHourlyCost);
7 fprintf ('Total revenue: %f, total cost: %f\n', revenue, costs);
8 fprintf ('Total profit: %f\n', revenue - costs);
9 fprintf ('Sum of day time rentals: %f\n', totalrentals);
10 fprintf ('Sum of night time rentals: %f\n', totalnightrentals);
11 revenuePlot(tt+1) = revenue - costs;
\end{verbatim}

Listing 24 Simulator.m/lines 462-472

As mentioned before, the fprintf-command is not used to create and display figures, but to return current results by showing them in Matlab’s Command Window. As the simulation solely has to return the latest values, the fprintf-command does not consume any computational power and therefore, does not slow down the simulation process like the live plotting features. As a consequence, the respective features will be provided throughout the simulation period.

In this case, the number of driving and waiting customers (\textit{countDriving}, \textit{waiting}), failed rentals (\textit{carDeficit}, \textit{chargeCarDeficit}), revenue (\textit{revenue}), costs (\textit{costs}), profit, and the total number of both night (\textit{totalnightrentals}) and day rentals (\textit{totaldayrentals}) are displayed (line 1-7).

Finally, various plots that are used to create the figures after the simulation are set:

\begin{verbatim}
1 avgChargePerTime = sum(vCharges)/size(vCharges,1);
2 figure();
3 plot(time(1:length(time)-1),avgChargePerTime);
4 figure();
5 plot(time(1:end-1),profitPlot);
6 figure();
7 plot(time(1:end-1),avgWaitTime);
\end{verbatim}

Listing 25 Simulator.m/lines 476-482
AvgChargePerTime displays the average charge in each iteration. For that purpose, the sum of all charges from all vehicles \( vCharges \) is divided by the total number of vehicles which is, in this case, determined by the size-command that checks the length of the \( vCharges \)-array from top to bottom (line 1). Subsequently, the plot is created (line 3). The x-axis of the plot displays the time and ranges from the start time to the end time of the simulation. The y-axis displays the previously determined variable \( \text{avgChargePerTime} \). Afterwards, the plot containing the profit is set. Yet again, the x-axis displays the time, whereas the y-axis displays the profit in each iteration (line 5). Finally, the plot creating the figure displaying the average wait time is set. Likewise, the x-axis displays the time, and the x-axis displays the average waiting time of all customers (line 7).

### 4.2 Customer Class

In comparison to the simulator class, the customer class is fairly simple, as no real processes are simulated. Since it is one of the objects being used throughout the simulation, it encompasses various properties:

```matlab
classdef Customer < handle

    properties
        id;
        isDriving = false;
        ret = 0;
        lastDayDrive;
        lastNightDrive;
        fromStation = 0;
        fromId = 0;
        toStation = 0;
        toId = 0;
        currentpos = 0;
        myCar;
        sumdayrentals;
        sumnightrentals;
        custType;
        minIntervalday;
        minIntervalnight;
        pLiving = NaN*zeros(1,2);
        lQuad;
        maxdistKM;
        maxdurofStay;
        actualwaitTime = 0;
        maxwaitTime;
        prefCar;
        initdiffcarTol = 0;
        toleranceThreshold;
        dropOut = false;
        dDrive = 0;
    end
```

Listing 26 Customer.m/lines 1-31
However, most of these properties and their purposes were already described in the specific concept or the simulator class. Therefore, they are solely listed here, whereas new ones will be explained in the corresponding functions of this class.\textsuperscript{20}

**Function constructor**

The first function of the customer class is the so-called constructor function:

\begin{verbatim}
function obj = Customer(id)
    obj.id = id;
end
\end{verbatim}

Listing 27 Customer.m/lines 34-36

The constructor function is used to assign customer properties when the customer objects are created for the first time. Usually, these constructor methods are used to assign all class-related properties. However, the author decided to write unique functions for each property, as this greatly improves the clarity of the structure of the program and thus, facilitates the customization by the user. In this case, the constructor function solely defines the customer’s IDs. In order to do so, it requires an input variable called ID being defined in the simulator class.\textsuperscript{21}

**Function setcusttypeDistrib**

The second function determines the customer type of each customer:

\begin{verbatim}
function obj = setcustTypeDistrib(obj, ncust)
    if obj.id < ceil(ncust/3) \% 33 percent of the customers are type 1
        obj.custType = 1;
    elseif floor(ncust/3) < obj.id && obj.id < ceil(0.75*ncust)
        obj.custType = 2;
    elseif ceil(0.76*ncust) < obj.id
        obj.custType = 3;
    end
end
\end{verbatim}

Listing 28 Customer.m/lines 38-48

The customer type depends on the customer’s ID. If the customer’s ID is below a certain portion of the total number of customers $nCust$, his type is set accordingly. In this case, it is assumed that there are three customer types each comprised of 33\% of the total number of customers $nCust$ (lines 2, 5, and 8). Customer type 1 will represent families; customer type 2 describes students, whereas customer type 3 depicts singles. Moreover, ceiling and floor commands are used. $Ceil$ rounds up the respective value, whereas floor rounds the value down. Both commands are highly important, as otherwise some of the customers would not get a type assigned. As mentioned in the specific concept, the

\textsuperscript{20}For an image of a completely set customer, see appendix 3
\textsuperscript{21}see listing 5
customer types have very important implications for the rest of the customer class, since
the properties being set in the various functions differ in accordance.

Function setminInt
The dependency on the customer type can be seen in the following function:

```matlab
function obj = setminInt(obj, hourOfDay, priceperhour)

if hourOfDay > 20 || hourOfDay < 9 % night

    if obj.custType == 1 % families
        meanInterval = 45*24*60+0.1*priceperhour*24*60;
        stDev = meanInterval/2;
        MU = log(meanInterval^2/sqrt(stDev^2+meanInterval^2));
        SIGMA = sqrt(log(stDev^2/meanInterval^2 + 1));
        obj.minIntervalnight = lognrnd(MU, SIGMA);
    elseif obj.custType == 2 % students
        meanInterval = 30*24*60+0.2*priceperhour*24*60;
        stDev = meanInterval/2;
        MU = log(meanInterval^2/sqrt(stDev^2+meanInterval^2));
        SIGMA = sqrt(log(stDev^2/meanInterval^2 + 1));
        obj.minIntervalnight = lognrnd(MU, SIGMA);
    elseif obj.custType == 3 % singles
        meanInterval = 25*24*60+0.05*priceperhour*24*60;
        stDev = meanInterval/3;
        MU = log(meanInterval^2/sqrt(stDev^2+meanInterval^2));
        SIGMA = sqrt(log(stDev^2/meanInterval^2 + 1));
        obj.minIntervalnight = lognrnd(MU, SIGMA);
    end
else
    if obj.custType == 1 % families
        meanInterval = 12*24*60+0.1*priceperhour*24*60;
        stDev = meanInterval/7;
        MU = log(meanInterval^2/sqrt(stDev^2+meanInterval^2));
        SIGMA = sqrt(log(stDev^2/meanInterval^2 + 1));
        obj.minIntervalday = lognrnd(MU, SIGMA);
    elseif obj.custType == 2 % students
        meanInterval = 20*24*60+0.2*priceperhour*24*60;
        stDev = meanInterval/9;
        MU = log(meanInterval^2/sqrt(stDev^2+meanInterval^2));
        SIGMA = sqrt(log(stDev^2/meanInterval^2 + 1));
        obj.minIntervalday = lognrnd(MU, SIGMA);
    elseif obj.custType == 3 % singles
        meanInterval = 15*24*60+0.05*priceperhour*24*60;
        stDev = meanInterval/9;
        MU = log(meanInterval^2/sqrt(stDev^2+meanInterval^2));
        SIGMA = sqrt(log(stDev^2/meanInterval^2 + 1));
        obj.minIntervalday = lognrnd(MU, SIGMA);
    end
end
```

Listing 29 Customer.m/lines 50-96

The setminint-function determines the minimum time interval that has to pass between
each rental. As mentioned in the specific concept, these intervals are set by a logarithmic
normal distribution (i.e. line 10). A logarithmic normal distribution is a continuous
distribution in which the logarithm of a variable has a normal distribution:

\[ P(x) = \frac{1}{\sqrt{2\pi\sigma x}} e^{-\frac{(ln(x)-\mu)^2}{2\sigma^2}} \quad (3) \]

It solely requires the user to define a mean interval and a standard deviation, as \( \mu \) and \( \sigma \) depend on both (lines 6-10). In this simulation, it is assumed that families drive once every 45 days during the night and every 12 days during the day. Thus, variable \( \text{meaninterval} \) is set accordingly (lines 6+26). Students however, drive every 20 days during the day and once every 30 days during the night (lines 12 + 32). Finally, singles are assumed to drive every 15 days during the day and every 25 days during the night (lines 18+38). Furthermore, it is assumed that the hourly rental fees \( \text{priceperHour} \) play a minor role regarding the frequency of driving, thus allowing the user to test for the viability of various price policies. Concerning customer type 1, it is assumed that 10% of the hourly rental fees multiplied by 24 hours is added to the mean interval (line 6). In case the fees add up to 12 euros an hour, \( \text{meaninterval} \) would increase by 1.2 days or 1728 minutes. As students generally have less money than families and singles, they are more price-sensitive and the corrective factor constitutes 20% of the price per hour (line 12). Singles are the least cost sensitive customer type with 0.05%, as they usually have the best ratio between income and cost of living, and consequently have the most money at their disposal (line 18).

**Function setfirstDrive**

In the next step, the starting condition setting the customer’s last rental is set:

```matlab
function obj = setfirstDrive(obj)
    if obj.custType == 1
        obj.lastDayDrive = -(10.234*24*60)*rand(1);
        hourOfDay = floor(mod(obj.lastDayDrive,60*24)/60);
        while hourOfDay > 20 || hourOfDay < 9 % reiterates until lastDayDrive was during the day
            obj.lastDayDrive = -(10.234*24*60)*rand(1);
            hourOfDay = floor(mod(obj.lastDayDrive,60*24)/60);
        end
        obj.lastNightDrive = -(40.234*24*60)*rand(1);
        hourOfDay = floor(mod(obj.lastDayDrive,60*24)/60);
        while hourOfDay < 20 && hourOfDay > 9 % reiterates until lastNightDrive was during the day
            obj.lastNightDrive = -(40.234*24*60)*rand(1);
            hourOfDay = floor(mod(obj.lastDayDrive,60*24)/60);
        end
    elseif obj.custType == 2
        obj.lastDayDrive = -(16.5*24*60)*rand(1);
        hourOfDay = floor(mod(obj.lastDayDrive,60*24)/60);
        while hourOfDay > 20 || hourOfDay < 9 % reiterates until lastDayDrive was during the day
    end
```
Listing 30 Customer.m/lines 97-145

First of all, a value close to the mean interval is chosen (i.e. 10.234 being close to 12) and multiplied with a random number between zero and one, thus returning an initial last drive (line 5). However, as the simulation distinguishes between day and night rentals, a while-loop is introduced that examines whether the respective value, for instance the last day rental lastDayDrive, was really during the day. If not, the while-loop reassigns values until this condition has been met (lines 7-10). Once again, this procedure is necessary to avoid driving peaks during the early morning and evening hours, as customers would immediately start to drive in the morning, if the last day rental lastDayDrive was set to be during the night. The function setfirstDrive is only called once, since it is solely used to determine the customer's last rentals before their first actual rental.

Function setplaceofLiving

In this function, the customer’s place of living is determined. However, the following code is shortened, as the procedure is repeated multiple times:

```
function setplaceofLiving(obj, ncust) %distributes customer’s place of living
    % workaround for generating random numbers between two fixed
    % values, since matlab has no respective commands
    highxQ23 = 1730; % highest x value in quadrant 2 and 3
    lowxQ23 = 850;
    rx = lowxQ23+(highxQ23-lowxQ23)*rand(1); % produces a random nr. between 1730 and 850
```
highyQ34 = 850;
lowyQ34 = 400;
ry = lowyQ34+(highyQ34-lowyQ34)*rand(1);

if obj.id < ceil(0.08333*ncust) %since all customers up to 33%*ncust are custType
 1, we have to divide 33% by 4 and distribute the customers accordingly
  obj.pLiving(1,1) = 850*rand(1);
  obj.pLiving(1,2) = 400*rand(1);
  obj.lQuad = 1;
else if obj.id > ceil(0.08333*ncust) && obj.id < ceil(0.16666*ncust)
  obj.pLiving(1,1) = rx;
  obj.pLiving(1,2) = 400*rand(1);
  obj.lQuad = 2;
else if obj.id > ceil(0.16666*ncust) && obj.id < ceil(0.25*ncust)
  obj.pLiving(1,1) = rx;
  obj.pLiving(1,2) = ry;
  obj.lQuad = 3;
else if obj.id > ceil(0.25*ncust) && obj.id < ceil(0.33333*ncust)
  obj.pLiving(1,1) = 850*rand(1);
  obj.pLiving(1,2) = ry;
  obj.lQuad = 4;

Listing 31 Customer.m/lines 146-173

The customer’s place of living \textit{pLiving} is a two dimensional coordinates variable consisting of both x- and y-values. In order to equally distribute the customer types over the map, the latter is divided into four quadrants. The first quadrant ranges from 0 to 850 pixels on the x-axis and from 0-400 pixels on the y-axis. The second quadrant ranges from 850-1735 pixels on the x- and 0-400 pixels on the y-axis. The third quadrant ranges from 850-1735 pixels on the x- and 400-850 pixels on the y-axis, whereas the fourth quadrant ranges from 0-850 pixels on the x- and 400-850 pixels on the y-axis.\footnote{for a figure depicting the city quadrants see appendix 3}

Consequently, the place of living of customers living in quadrant 1 is determined by multiplying the respective quadrant’s borders with a random number between zero and one (lines 13-14). Setting the customer’s place of living in the other 3 quadrants, however, requires a workaround in Matlab. Since the x-axis borders of quadrant 2 are 850 and 1735, a random number between both values has to be found. However, Matlab has no commands to find random numbers between two fixed values. Thus, the workarounds in line seven and eleven are applied. Variable \textit{rx} will return a random number between 850 and 1735, whereas \textit{ry} will return a random number between 400 and 850 (line 7+ 11).

Additionally, as each customer type should be distributed equally over the map, the procedure is divided into three parts. In the previous code, solely the first part is shown. It makes sure that customers having customer type 1, which makes up 33\% of the total number of customers, are distributed equally over the four quadrants (line 15, 19, 23, and 27). Likewise, the place of living of the two remaining customer types is set.
Function setmaxwalkDist

The function \textit{setmaxwalkDist} deterministically sets the maximum walking distance customers are willing to walk to their closest station:

\begin{verbatim}
function setmaxwalkDist(obj)
  if obj.custType == 1
    obj.maxdistKM = 0.83;
  elseif obj.custType == 2
    obj.maxdistKM = 1.5;
  elseif obj.custType == 3
    obj.maxdistKM = 1.3;
  end
end
\end{verbatim}

Listing 32 Customer.m/lines 211-221

It is assumed that families have the lowest readiness to walk and will at best go 0.83 kilometers (line 3). Students, being customer type 2, have a higher threshold and will walk 1.5 kilometers (line 5). Customers with customer type 3 will walk 1.3 kilometers maximum to reach the nearest station (line 9). These assumptions are incongruent with the current car sharing literature, as the acceptable walking distance to a vehicle is usually set to 1/3 of a mile being 536 meters (cf. Barrios 2012, p. 6). However, these values apply for common car sharing systems where the vehicles are parked throughout the city on rented parking lots. As the proposed system requires car sharing stations and since it is financially unfeasible to place as many stations as other car sharing systems rent parking lots, the number of stations is significantly lower. Therefore, it is assumed that customers accept higher walking distances when they want to use e-Car sharing systems.

Function setmaxdurStay

Function \textit{setmaxdurStay} sets the maximum and minimum length of time customers will need to perform the tasks they rented the vehicle for:

\begin{verbatim}
function setmaxdurStay(obj)
  if obj.custType == 1
    obj.maxdurofStay = 30+(4*60-30)*rand(1);
    obj.maxdurofStay = 20+(2.5*60-30)*rand(1);
    obj.maxdurofStay = 30+(6*60-30)*rand(1);
  end
end
\end{verbatim}

Listing 33 Customer.m/lines 223-231

In order to simulate a random but nonetheless predictable customer behavior, the customer property \textit{maxdurofStay} will always range between a minimum and a maximum
value that also differs notably between the various customer types. The maximum time a family will require to perform their tasks is set to 240 minutes, whereas they will always need at least 30 minutes (line 3). Students, however, will need less time, as presumably they always try to minimize the length of rental in order to save money. Thus, the property \textit{maxdurofStay} is set to 150 minutes maximum and 20 minutes minimum (line 5). Singles, being customer type 3, will need between 30 minutes and 360 minutes to perform their tasks (line 7).

\textbf{Function setmaxwaitTimenTolerance}

The function \textit{setmaxwaitTimenTolerance} sets two important properties that are required during the vehicle decision process:

\begin{verbatim}
function setmaxwaitTimenTolerance(obj)
    if obj.custType == 1
        obj.maxwaitTime = 20+(30-20)*rand(1); % maximum time of waiting lies between 20 and 30 min
        obj.toleranceTreshold = 0.35 +(0.5-0.35)*rand(1);
    elseif obj.custType == 2
        obj.maxwaitTime = 30+(40-30)*rand(1);
        obj.toleranceTreshold = 0.2 +(0.3-0.2)*rand(1);
    elseif obj.custType == 3
        obj.maxwaitTime = 10+(20-10)*rand(1);
        obj.toleranceTreshold = 0.55 +(0.7-0.5)*rand(1);
    end
end
\end{verbatim}

Listing 34 Customer.m/lines 233-245

First of all, it determines the maximum time a customer is willing to wait for a vehicle. As soon as the waiting time exceeds his maximum wait time \textit{maxwaitTime}, he will cancel the rental process.\textsuperscript{23} Furthermore, the functions set a so-called tolerance threshold. It is needed in another function to simulate the behavior that customers are willing to switch to a vehicle having another vehicle type than their actual preferences. Yet again, these values differ between customer types and the customers themselves. Families wait for 20-30 minutes at best, whereas students wait for about 30-40 minutes (line 3+6). singles have the lowest tolerance concerning wait times, as they solely wait between 10-20 minutes. As opposed to this, singles have the highest tolerance regarding the type of vehicle, ranging between 0.55 and 0.7 (line 10). Families, however, have a tolerance between 0.35 and 0.5, whereas the student’s tolerance varies between 0.2 and 0.3 (line 4+7). The latter is assumed to have the lowest tolerance, as students generally want the cheapest vehicle and thus, are not very willing to switch to a more expensive type. However, solely \textit{maxwaitTime} will be used later in the vehicle decision process, as the function depicting the customers switching to another vehicle did not make it to the final version if this simulation.

\textsuperscript{23}see listing 12
Function setprefCar

The customer’s preferences regarding the vehicle types are determined by the function setprefCar:

```matlab
function setprefCar(obj)
    if obj.custType == 1
        obj.prefCar = 2;
    elseif obj.custType == 2
        obj.prefCar = 1;
    elseif obj.custType == 3
        obj.prefCar = 3;
    end
end
```

Families are assumed to prefer vehicles with vehicle type 2, as the respective vehicles depict family cars having a bigger trunk and sufficient space for children (line 3). Students will prefer vehicle type 1 which is the smallest and least expensive vehicle (line 6). Finally, singles prefer vehicle type 3. Vehicles of the corresponding type are neither compact cars (type 1) nor station wagons (type 2) but sedans such as the BMW-I3.

Function letDrive

The function letDrive is one of the few functions of the customer class that are actively used during the simulation process, since the previous functions were solely setting customer-related properties:

```matlab
function success = letDrive(obj, fromStation, toStation, dist)
    distKM = norm(dist/85);
    obj.fromStation = fromStation.coords;
    obj.toStation = toStation.coords;
    obj.toId = toStation.id;
    obj.fromId = toStation.id;
    nCars = length(fromStation.cars);
    % waitTime = 10;
    obj.currentpos = obj.fromStation;

    if ~isempty(fromStation.cars)
        foundCar = 0;
        for ii = 1:nCars
            fCar = fromStation.cars(ii);
            buffer = distKM * fCar.nominalConsump * 0.3; % 30% of the complete trip length as buffer
            minCharge = distKM * fCar.nominalConsump + buffer; % mincharge = actual mileage + buffer
            if fCar.carType == obj.prefCar
                if fCar.currcharge > minCharge
                    foundCar = 1;
                    nCar = fCar;
                else
                    success = 0;
                end
            end
        end
    end
end
```

Listing 35 Customer.m/lines 246-256
Listing 36 Customer.m/lines 258-295

The task of the function consists of finding suitable vehicles for the customer’s trip. In order to do so, the function requires information about the customer’s starting station, his destination and the distance between both which is received from the simulator class while the function is called. Variable \textit{distKM} is introduced to convert the distance being measured in pixels to kilometers. As each kilometer consists of 85 pixels, the distance is divided accordingly (line 2). \textit{NCars} is required to define the number of iterations in the subsequent for-loop. It checks and saves the number of vehicles that are currently parked at the customers starting station (line 7). Moreover, the starting stations coordinates are set as the customer’s current position (line 9). In line 11, it is examined whether the starting station is currently empty or not. In case it is, variable \textit{success} is set to -1 (line 36). If not, \textit{foundCar} is initialized which is a binary variable that describes whether the customer found a vehicle or not (line 12). Afterwards, the previously mentioned for-loop is introduced. It starts at 1 and ends when all currently parked vehicles were examined or when a suitable vehicle was found (line 34). Variable \textit{Fcar} contains the vehicle object that is currently examined (line 14). The variable \textit{buffer} defines the amount of charge that is required in addition to the predicted trip consumption. In this case, it is assumed that the buffer should at least equal 30\% of the total consumption (line 15). The minimum charge a vehicle has to have in order to be considered for the rental consists of the actual mileage of the trip and the previously described buffer (line 16). In the next step, the program checks whether the currently examined vehicle has both enough charge and the right vehicle type (line 18). If both parameters fit, \textit{foundCar}, which is used in line 29, is set to 1. However, if the vehicle does not have the right vehicle type \textit{success} is set to 0 which in turn will initiate the waiting in the simulator class. Consequently, when the customer reaches his maximum wait time, the rental process stops. If the customer does not find a suitable vehicle due to the battery charges, variable \textit{foundCar} remains 0 as set in line 12. If that happens,
variable *success*, which is needed during the simulation process in the simulator class, assumes value -1, thus signaling the program that the customer did not find a vehicle due an inappropriate vehicle type or the station lacking vehicles (line 26 + 36). In case the customer found a suitable vehicle and *foundCar* = 1, the customer’s state changes to driving (line 30). Furthermore, his property *myCar* describing the customer’s vehicle is set to the current vehicle object (line 31). Thus, for the time of the rental, each customer has a vehicle object assigned. Finally, variable *success* is set to 1, therefore signaling the program that a vehicle has been successfully found.

**Function updatePos**

The function *updatePos* is used throughout the whole rental process while the customer is driving:

```matlab
function obj = updatepos(obj, currentposition)
    obj.currentpos = currentposition;
end
```

Listing 37 Customer.m/lines 324-326

Each time the customer advances a portion of the trip, his position property *currentpos* needs to be updated to both allow for its depiction on the map and the driving procedure. In order to do so, the function requires the current position to be passed on from the simulator class, and overwrites *currentpos* with the respective position.

**Function returnDrive**

The function *returnDrive* is called in the simulator class when the customer starts returning after performing his tasks:

```matlab
function returnDrive(obj, haltPosition)
    obj.ret = 1;
    obj.toStation = obj.fromStation;
    obj.fromStation = haltPosition;
    obj.toId = obj.fromId;
    obj.fromId = 0;
end
```

Listing 38 Customer.m/lines 297-303

First of all, it sets the property *ret* to 1. The property is used in various if-cases during the simulation process to determine whether the customer is driving to or returning from his destination. Furthermore, since the customer is returning, his destination station is set to be his starting station and vice versa (lines 3-4).

---

26 see listing 15
Function stopDrive

As already mentioned in the simulator class, the *stopDrive* function is used to reset various customer-related properties after each rental:

```
function obj = stopDrive(obj, sumdayrentals, sumnightrentals, hourOfDay)
    obj.isDriving = false;
    obj.fromStation = 0;
    obj.toStation = 0;
    obj.fromId = 0;
    obj.toId = 0;
    obj.currentpos = NaN;
    obj.ret = 0;
    if ~strcmpi(class(obj.myCar), 'Car')
        % obj.fromStation.addCar(obj.myCar);
        obj.myCar = NaN;
    end
    if daydriving == 0
        obj.sumnightrentals(end + 1) = sumnightrentals + 1;
    else
        obj.sumdayrentals(end + 1) = sumdayrentals + 1;
    end
end
```

Listing 39 Customer.m/lines 305-322

Firstly, the state of the customer changes from driving to not driving by changing the property *isDriving* from true to false (line 2). Afterwards, the customer’s starting and destination station (*fromStation, toStation*), as well as their IDs (*fromId, toId*) are set to 0, since the *stopDrive* function is called, when the rental process is terminated (line 3-6). By setting the property *currentPos* to NaN, the customer’s disappear from the overview map, whereas the property *myCar* is set to NaN, because the customers left their vehicle at the station. Finally, it is checked whether the customer started his trip during the day or night, which is done by examining whether the variable *daydriving* equals 1 or 0. In the former case, *sumdayrentals*, describing the current customer’s sum of day rentals, increases by 1 (line 16). Likewise, *sumnightrentals* increases by 1, if *daydriving* equals 0 (line 14).

Function setlastdrive

The last function of the customer class is used to set the time of the customer’s last rental:

```
function obj = setlastdrive(obj, tt, daydriving)
    if daydriving == 0
        obj.lastNightDrive = tt;
    else
        obj.lastDayDrive = tt;
    end
end
```

Listing 40 Customer.m/lines 328-335

66
Yet again, it is solely examined whether the customer started the trip either during the day or night by checking daydriving (line 3). If he started during the day, the property lastDayDrive, which describes the time of his last day rental, is set to the current simulation time \( tt \) (line 6). Similarly, lastNightDrive is determined (line 4).

### 4.3 Vehicle Class

The vehicle class describes the vehicle objects and functions being used in the simulation:

```plaintext
properties
    id;
    maxcharge; % maximum charge
    currcharge; % current charge
    state; % 0 = parked, 1 = drives
    carType;
    nominalConsump;
end
```

Listing 41 Vehicle.m/lines 5-12

Each vehicle has an ID, a maximum charge \( \text{maxcharge} \), a property defining the current SOC \( \text{currcharge} \), a certain vehicle type \( \text{carType} \), and a nominal consumption \( \text{nominalConsump} \). Additionally, each vehicle is in a certain \( \text{state} \), which differs between parking and driving (lines 6-11).

#### Function constructor

The constructor function of the vehicle class is equivalent to the one of the customer class:

```plaintext
function obj = Car(id)
    obj.id = id;
    obj.state = 0; % 0: parked, 1: driving, 2: charging, 3: damaged
end
```

Listing 42 Vehicle.m/lines 15-18

Thus, an ID is assigned to each vehicle allowing for an easy identification and addressing in the code. Besides determining the ID, the vehicle’s property \( \text{state} \) is set to 0 by default, as the vehicles are supposed to be parking before the simulation is initiated.

#### Function setvehicleType

The function \( \text{setvehicleType} \) defines the type of the vehicle:

```plaintext
function setvehicleType(obj, vType)
    if vType == 1
        obj.carType = 1;
    end
```

\(^{27}\) For an image of a completely set vehicle, see appendix 4
Listing 43 Vehicle.m/lines 20-30

However, the property is solely set here, as the decision of which vehicle having which type is made in the simulator class. By examining the input variable \( vType \), the property \( carType \) assumes either 1, 2, or 3. Like mentioned before, vehicle type 1 describes compact cars, whereas type 2 stands for station cars. Vehicles of type 3 are sedans such as the BMW-I3.

**Function setmaxCharge**

The function \( \text{setmaxCharge} \) determines the maximum charge and default SOC of each vehicle type:

Listing 44 Vehicle.m/lines 32-46

Vehicles of type 1 are set to have a maximum charge \( \text{maxcharge} \) of 16 kWh and nominal consumption \( \text{nominalConsump} \) of 12.9 kW / 100 km, as respective vehicles are based on the Mitsubishi I-MIEV (Mitsubishi 2008) (lines 2-5). Vehicle type 2 depicts station wagons. As of now, there are solely prototypes of electric station wagons. Therefore, it is assumed that the battery of such a vehicle enacts of 23 kWh and requires 13.3 kW / 100 km (lines 6-9). Finally, since vehicle type 3 depicts the BMW-I3, \( \text{maxcharge} \) is set to 25 kWh and \( \text{nominalConsump} \) amounts to 13.5 kW / 100 km (line (BMW 2013)).

\[\text{setmaxCharge}(\text{obj})\]

\[
\begin{align*}
\text{if} \ & \text{obj.carType == 1} \\
\text{obj.currcharge} &= 16; \% 16: 100\%, 0: 0\% \rightarrow \text{Mitsubishi I-Miev} \\
\text{obj.maxcharge} &= 16; \\
\text{obj.nominalConsump} &= 12.9/100; \\
\text{elseif} \ & \text{obj.carType == 2} \\
\text{obj.currcharge} &= 23; \% \text{Station Wagon} \\
\text{obj.maxcharge} &= 23; \\
\text{obj.nominalConsump} &= 13.3/100; \\
\text{elseif} \ & \text{obj.carType == 3} \\
\text{obj.currcharge} &= 25; \\
\text{obj.maxcharge} &= 25; \\
\text{obj.nominalConsump} &= 13.5/100; \% \text{BMW-I3}
\end{align*}
\]

28 see listing 3
Function detCharge

The purpose of detCharge is to model the continuous deterioration of the battery capacities. In so doing, the function allows the user to examine the effects on the vehicle availability or other performance indicators in long-term simulations:

```
if obj.maxcharge > 0
    obj.maxcharge = obj.maxcharge - 0.0000148;
end
```

Listing 45 Vehicle.m/lines 50-54

As long as the maximum charge maxcharge of the vehicle is greater than 0, it will decrease by 0.0000148 kWh per minute. The respective value is derived from the official value being stated by BMW, declaring that the vehicle will lose 15% of its battery capacities within a year of active driving (BMW 2013). Since the battery technology is the same and as there is no further information on how much capacity Mitsubishi’s I-MIEV is losing per year, the value for the BMW-I3 is applied to all vehicle types.

Function drive

The drive-function does not, as the name suggests, advance the customer during the rental process, but manages the depletion of charge:

```
obj.state = 1;
distKM = (dist/85)*1.39;
obj.currcharge = obj.currcharge - obj.nominalConsump * distKM;
end
```

Listing 46 Vehicle.m/lines 57-61

Initially, the vehicle’s state is being changed from 0 to 1 because the customer starts driving and state 0 solely applies to parked vehicles (line 2). Furthermore, the distance of the trip, which is drawn from the simulator class, is divided by 85, since 85 pixels make up 1 kilometer, thus returning the variable distKM. This variable is multiplied by a factor of 1.39, as the true driving distances would otherwise be underestimated. By comparing multiple real life driving scenarios in which the direct distances were compared to the actual driving distances, an average factor, equaling the previously mentioned 1.39, depicting this discrepancy was computed. Finally, the current charge currCharge is determined by subtracting nominalConsump from the level of charge from the previous iteration. It has to be noted that this function is called multiple times throughout the simulation. Consequently, the vehicles lose charge continuously while the customers are driving.

29see listing 13
4.4 Station Class

The station class contains the properties and functions of the station objects:

```matlab
classdef Station < handle

    properties

        id;
        carIds;
        cars;
        coords;
        capacity;
        nCharger;
        fCharger;

end
```

Listing 47 Station.m/lines 1-12

As with the other classes, each station has a particular ID. Moreover, each station has certain vehicle capacities `capacity` which also determine the number of vehicle objects `cars` and their IDs `carIds`. Apart from that, each station is placed at certain coordinates `coords` and encompasses a number of fast `fCharger` and normal chargers `nCharger`.

Function constructor

The constructor function is called when the station objects are created:

```matlab
function obj = Station(id, coords, cars, carIds, capacity, nCharger, fCharger)
    obj.cars = cars;
    obj.id = id;
    obj.coords = coords;
    obj.carIds = carIds;
    obj.capacity = capacity;
    obj.fCharger = fCharger;
    obj.nCharger = nCharger;
end
```

Listing 48 Station.m/lines 16-24

It defines the station objects properties with the input stemming from the simulator class.

Function removeCar

The `removeCar` function is used to remove the vehicles each time they leave the station:

```matlab
function obj = removeCar(obj, car)
    remainingCars = ~logical(obj.carIds == car);
    obj.cars = obj.cars(remainingCars);
    obj.carIds = obj.carIds(remainingCars);
end
```

For an image of a completely set station, see appendix 5

See listing 3
In order to work properly, the function needs the exact vehicle object that was chosen by the customer (line 1). In the next step, the array of car IDs `carIDs` of the station is compared to the vehicle object, that was chosen by the customer in the customer class, by using the logical-command. The logical-command converts the elements of an array into logicals. It returns an array that can be used for logical indexing or logical tests. Logicals can have the values 0 and 1 corresponding to false and true, respectively.

For instance, the station has the following vehicles \( \text{cars} = [1 2 3] \) with the carIds-array \( \text{carIds} = [4 5 6] \) and the customer chose the vehicle with \( ID = 4 \). The logical-command in line 2 will now check which entry of the `carIDs`-array matches the respective vehicle object. In this case, it would return \( \text{remainingCars} = [1 0 0] \). However, since the program should delete the entry where both values match, logical is applied, therefore returning \( \text{remainingCars} = [0 1 1] \) (line 2). This array is then used as an index for the `cars` and `carIDs`-array. Indexing the arrays respectively will return \( \text{carIDs} = [5 6] \) and \( \text{cars} = [2 3] \). Thus, the vehicle is deleted from the pool of available vehicles.

**Function addCar**

The `addCar` function is used to add the customer’s vehicle to the station’s vehicle pool once the rental is finished:

```
function obj = addCar(obj, car)
    obj.cars{end+1} = car;
    obj.carIds(end+1) = car.id;
end
```

For that purpose the `cars` and `carIds`-array is extended by one element. The vehicle object will be added to `car`-array in its entirety, whereas the `carIDs`-array is solely extended by the vehicle object’s ID.

**Function removeCharger**

The `removeCharger` function is needed when the customer reaches his initial station. It manages the availability of both fast and normal chargers:

```
% removes fast charger when available (see main)
function obj = removeCharger(obj, fromStation)
    obj.fromStation = fromStation;
    obj.fromStation.fCharger = fromStation.fCharger - 1;
end

% removes normal charge unit
function obj = removeNormalCharger(obj, fromStation)
Every time one of the two functions is called, the number of either fast \textit{fCharger} or normal chargers \textit{nChargers} at the customer’s starting station decreases by 1 (lines 4+10).

\textbf{Function chargeCars}

The \textit{chargeCars} function is one of the few functions that are performed throughout the entire simulation process:

\begin{verbatim}
function chargeCars(obj, tt)
    for kk = 1:length(obj.cars);
        for count = 1:tt
            if obj.cars(kk).currcharge < obj.cars(kk).maxcharge && obj.fCharger > 0 && obj.cars(kk).state == 0;
                obj.cars(kk).currcharge = obj.cars(kk).currcharge + 0.026*obj.cars(kk).maxcharge; % a fast charger needs 30 min to recharge 80% of a 16 Kwh battery -> 2.6% per minute
            elseif obj.cars(kk).currcharge < obj.cars(kk).maxcharge && obj.nCharger > 0 && obj.cars(kk).state == 0;
                obj.cars(kk).currcharge = obj.cars(kk).currcharge + 0.002083*obj.cars(kk).maxcharge; % a fast charger requires 6-8 hours to recharge 100% of the batteries of a 16 Kwh battery -> 0.2% per minute
            else
                end
            end
        end
    end
end
\end{verbatim}

It uses nested for-loops, ensuring that each vehicle is charged in each iteration, as long as their state is set to 0, signifying that they are currently parked at the station, and that their current charge \textit{currCharge} is lower than the vehicle’s maximum battery capacities \textit{maxCharge} (line 6). Furthermore, as described in the specific concept, the customers always prefer fast chargers over normal chargers. Thus, if a fast charger is available, the vehicle’s charge \textit{currCharge} increases by 2.6% per minute (line 7). Otherwise, the vehicle will be charged by the normal chargers, thus increasing \textit{currCharge} by 0.2083% per minute (lines 9-10). The respective values are derived from the technical data of the BMW-I3 that requires half an hour to charge 80% of the capacities when using fast chargers, and 6-8 hours when using normal chargers (BMW 2013).
5 Scenario-Analysis

In the following section, a scenario analysis will be performed. Scenario analyses are utilized as an effective tool for dealing with uncertainties, as they allow researchers to create and examine multiple alternative images of the future and its external environments by alternating certain key factors of the system or its environment (cf. Postma and Liebl 2005, p. 161).

In this case, the system to be examined is the car sharing system proposed in the paper. The aim of the analysis consists of finding the best configurations of the system, thus maximizing the customer satisfaction and the profitability of the system while minimizing its overall costs. Consequently, the analysis does not aim at evaluating certain active measures that are supposed to optimize an already existing system, but basic decisions regarding the set-up of the car sharing service.

Nevertheless, the future scenarios of the car sharing system and its profitability cannot be thoroughly analyzed without taking the customer’s behavior into account. Besides the number of stations, vehicles, and their distribution throughout the city, individual characteristics such as the frequency of driving, vehicle preferences, tolerance regarding waiting times, or in other words, characteristics shaping the interaction between the customer and the system, are also highly important for the success of car sharing services. Therefore, a dualistic approach is chosen, according to which the multiple scenarios vary in both customer behavior and the system set-up. The analysis itself is performed by using the simulation tool that was meticulously elucidated in the previous section.

5.1 Description of Influence Factors, Performance Indicators, and the Scenarios

There are numerous performance indicators by which the scenarios are finally assessed:

<table>
<thead>
<tr>
<th>Performance Indicator</th>
<th>Desired Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sum of Successful Rentals</td>
<td>High</td>
</tr>
<tr>
<td>Vehicle Availability</td>
<td>High</td>
</tr>
<tr>
<td>Average Charge</td>
<td>High</td>
</tr>
<tr>
<td>Profit</td>
<td>High</td>
</tr>
<tr>
<td>Dropout Customers</td>
<td>Low</td>
</tr>
<tr>
<td>Wait Times</td>
<td>Low</td>
</tr>
</tbody>
</table>

Table 6 Key Performance Indicators and Desired Outcomes

First of all, the total number of rentals will be measured. The more successful rentals were performed throughout the simulation, the more profit, which also is used as a
performance indicator, will be generated. Subsequently, as the system deploys electric vehicles, the average charge of the vehicles and their general availability is also examined. A high average charge amongst all vehicles is desirable, since it directly impacts the likeliness of a successful rental, which is depicted by the overall vehicle availability. Likewise, high vehicle availability is of high importance, as it directly impacts customer satisfaction. In order to evaluate the accessibility of the car sharing system and the distribution of stations, the number of drop-out customers never using the service due to the high walking distances, is to be measured and minimized. As mentioned before, the influence factors are to be differentiated according to whether they are system or customer-related.

**System-Related Influence Factors**

The number of system-related influence factors that are being varied is comparably small:

<table>
<thead>
<tr>
<th></th>
<th>Setting 1</th>
<th>Setting 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Stations</td>
<td>22</td>
<td>32</td>
</tr>
<tr>
<td>Normal Chargers</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Fast Chargers</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 7 Possible System Settings

The first one which will be applied in two distinct settings is the distribution and number of stations. No further varieties are introduced, as the number of stations will almost exclusively influence the number of drop-out customers. Therefore, examining the differences once sufficiently provides an indication of the respective impacts. Furthermore, the number of both fast and normal chargers is altered in two cases. In the first setting, the station has a capacity of 3 normal chargers, whereas it will operate with three fast chargers in the second setting.
Customer-Related Influence Factors

The proposed simulation tool offers numerous possibilities to model the behavior of the customers, thus influencing the whole outcome of simulation. However, as solely six distinct scenarios are going to be examined, various customer-related properties are assumed to be consistent to improve the comparability of the distinct scenarios. The respective values can be taken from table 8:

<table>
<thead>
<tr>
<th>Customer Type</th>
<th>Max. Walk Distance</th>
<th>Duration of Stay</th>
<th>Vehicle Preferences</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.83 km</td>
<td>30-120 min</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1.5 km</td>
<td>20-90 min</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1.3 km</td>
<td>30-110 min</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 8 Consistent Customer Properties

The first customer property to be varied is the minimum time interval between each rental, or, in other words, the frequency of driving:

<table>
<thead>
<tr>
<th>Day</th>
<th>Customer Type</th>
<th>Setting 1</th>
<th>Setting 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12 days</td>
<td>9.6 days</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>20 days</td>
<td>16 days</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>15 days</td>
<td>12 days</td>
<td></td>
</tr>
<tr>
<td>Night</td>
<td>Customer Type</td>
<td>Setting 1</td>
<td>Setting 2</td>
</tr>
<tr>
<td>1</td>
<td>45 days</td>
<td>36 days</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>30 days</td>
<td>24 days</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>25 days</td>
<td>20 days</td>
<td></td>
</tr>
</tbody>
</table>

Table 9 Settings Concerning Frequency of Driving

In order to establish comparability between the scenarios, solely two distinct settings concerning the frequency of driving are introduced. As can be drawn from the table, the frequency of driving in Setting 2 is increased by 30% compared to Setting 1.

Moreover, the maximum wait time is altered, again, between two settings:

<table>
<thead>
<tr>
<th>Customer Type</th>
<th>Setting 1</th>
<th>Setting 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20-30 min</td>
<td>60-90 min</td>
</tr>
<tr>
<td>2</td>
<td>30-40 min</td>
<td>90-120 min</td>
</tr>
<tr>
<td>3</td>
<td>10-20 min</td>
<td>50-80 min</td>
</tr>
</tbody>
</table>

Table 10 Settings Concerning the Customer’s Maximum Wait Time

As the proposed simulation tool does not allow for vehicle reservations through customers, the waiting time is almost tripled. Doing so slightly resembles a reservation
process, since the accepted waiting times are far higher than a customer probably would physically wait at a station. Thus, the customers are depicted to wait "at home".

As mentioned before, the customer property *place of living* is to be used to model the demographic structure of the city and its various districts. The demographic distribution is altered between two settings:

<table>
<thead>
<tr>
<th>Customer Type</th>
<th>Setting 1</th>
<th>Setting 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Equally Distributed</td>
<td>Mainly Quadrant 2</td>
</tr>
<tr>
<td>2</td>
<td>Equally Distributed</td>
<td>Equally Distributed</td>
</tr>
<tr>
<td>3</td>
<td>Equally Distributed</td>
<td>Mainly Quadrant 1</td>
</tr>
</tbody>
</table>

Table 11 Settings Concerning the Place of Living

Regarding the distribution of the customers throughout the map, the distribution of both customers of type 1 and 2 is altered in the second setting. This is done to examine the effects of changing the distribution on the number of successful rentals and the vehicle availability.

**Vehicle-Related Influence Factors**

Finally, the vehicle properties are to be altered between two settings. In order to simulate the effects the seasons and declining temperatures have on the battery charges, the battery capacities are lowered by 30%. Furthermore, the nominal consumption is increased, as electrical heating is required during winter. The different settings can be drawn from table 12.

<table>
<thead>
<tr>
<th>Vehicle Type</th>
<th>Setting 1</th>
<th>Setting 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16 kWh 12.9 kW/100 km</td>
<td>11.2 kWh 14.9kW/100 km</td>
</tr>
<tr>
<td>2</td>
<td>23 kWh 13.5 kw/100 km</td>
<td>16.1 kWh 15.5 kw/100 km</td>
</tr>
<tr>
<td>3</td>
<td>25 kWh 12.7/100 km</td>
<td>17.5 kWh 14.7 kW/100 km</td>
</tr>
</tbody>
</table>

Table 12 Vehicle Settings

Changing the capacities respectively is expected to lower the vehicle availability, average charge and total revenue.
Configuration of Scenarios

The combination of these customer and system-related settings leads to a set of possible scenarios. In total, six different scenarios are going to be examined; their characteristics are presented in table 14. Afterwards, the scenarios are shortly introduced, followed by the presentation of the results in the consecutive subsection 5.2. Finally, in subsection 5.3, the various implications being derived from the respective results are going to be discussed, thus describing crucial factors when setting up a car sharing service.

<table>
<thead>
<tr>
<th>System Settings</th>
<th>Scenario 1</th>
<th>Scenario 2</th>
<th>Scenario 3</th>
<th>Scenario 4</th>
<th>Scenario 5</th>
<th>Scenario 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Stations</td>
<td>Setting 1</td>
<td>Setting 2</td>
<td>Setting 1</td>
<td>Setting 1</td>
<td>Setting 1</td>
<td>Setting 1</td>
</tr>
<tr>
<td>Number of F. Chargers</td>
<td>Setting 1</td>
<td>Setting 2</td>
<td>Setting 1</td>
<td>Setting 1</td>
<td>Setting 1</td>
<td>Setting 1</td>
</tr>
<tr>
<td>Customer settings</td>
<td>Scenario 1</td>
<td>Scenario 2</td>
<td>Scenario 3</td>
<td>Scenario 4</td>
<td>Scenario 5</td>
<td>Setting 6</td>
</tr>
<tr>
<td>Driving Frequency</td>
<td>Setting 1</td>
<td>Setting 1</td>
<td>Setting 2</td>
<td>Setting 1</td>
<td>Setting 1</td>
<td>Setting 1</td>
</tr>
<tr>
<td>Max. Wait time</td>
<td>Setting 1</td>
<td>Setting 1</td>
<td>Setting 1</td>
<td>Setting 2</td>
<td>Setting 1</td>
<td>Setting 1</td>
</tr>
<tr>
<td>Place of Living</td>
<td>Setting 1</td>
<td>Setting 1</td>
<td>Setting 1</td>
<td>Setting 1</td>
<td>Setting 2</td>
<td>Setting 1</td>
</tr>
<tr>
<td>Vehicle settings</td>
<td>Scenario 1</td>
<td>Scenario 2</td>
<td>Scenario 3</td>
<td>Scenario 4</td>
<td>Scenario 5</td>
<td>Scenario 6</td>
</tr>
<tr>
<td>Battery Capacities</td>
<td>Setting 1</td>
<td>Setting 1</td>
<td>Setting 1</td>
<td>Setting 2</td>
<td>Setting 1</td>
<td>Setting 2</td>
</tr>
</tbody>
</table>

Table 13 Combination of Influence Factors

Scenario 1 is to be seen as the basic scenario that is to be compared to the other scenarios. A moderate number of stations are deployed, each having three vehicles of every kind and a total of three normal chargers. The customer’s exhibit an average frequency of driving and have a comparably low maximum wait time and are equally distributed all over the map.

Each of the consecutive scenarios examines the effects of solely one of the various influence factors, thus allowing insights on the importance of each of them. However, in scenario 2, two system settings are altered, as both effects can be examined separately since the settings do not affect each other.

In scenario 2, the number of stations is increased to see the respective effects on the number of dropout customers, successful rentals, vehicle availability, and average charge. It is expected that increasing the number of stations will decrease the number of drop-out customers, consequently increasing the number of rentals and decreasing vehicle availability as well as the average charge. Additionally, the number and composition of each stations chargers are adjusted. The number of fast chargers and normal chargers is mirrored so that each station operates with 3 fast chargers instead of normal chargers, thus allowing to examine the effects of the permutation of the number of chargers on the vehicle availability and the average vehicle charge. It is expected that applying solely fast chargers will significantly increase the average charge among all vehicles.

In scenario 3, the driving frequency of the customers is raised. This is done to reflect
the current trend of ever increasing numbers of active car sharing members which was observed during the last decades. This, in turn, allows the user to evaluate whether the current car sharing system will still be suitable if the previously described development of increasing customer numbers continues to persist.

Scenario 4 will alter the maximum waiting time of each customer. As the simulation does not provide any reservation features, increasing the waiting time is used to feign its existence. If the customer intends to reserve a vehicle at a certain time in the near future and it is already occupied, he will probably delay the reservation. Since he is not actively waiting on-site, his readiness to wait is probably going to be higher. A higher maximum wait time, in turn, will presumably increase the number of successful rentals, thus increasing the profit as well as decreasing the average charge.

In scenario 5, the distribution of customer types across the map, or in other words, their place of living is changed. The distribution of customer type 2 is to be kept unchanged. However, customers of type 1 and 3 will mainly live either in quadrant 2 or 1. This is done to examine whether certain city quarters with a particular demographic structure, such as student quarters around universities, will notably affect the overall performance of the system. Supposedly, the number of successful rentals is going to decrease, as one vehicle type is increasingly demanded, although the station only operates with one vehicle of each kind.

The last scenario constitutes as a special scenario, since neither customer nor system-related influence factors are altered. Instead, the vehicle’s battery capacities are significantly reduced, as the scenario is supposed to simulate the car sharing system during winter. Thereby, it is examined whether the car sharing system needs adjustments as the weather turns colder.
5.2 Scenario 1-6

Scenario 1

The first scenario describes the basic scenario of the simulator, and discusses the results stemming from the default values of the simulator. Basically, each consecutive scenario will be compared to this scenario in order to allow for an examination of the effects of the various influence properties on the diverse key performance indicators of the simulation. Moreover, by comparing each scenario with the basic scenario, it can be evaluated whether the influence factors act as predicted thus verifying that the simulator works as intended.

As can be seen in figure 1, the majority of the customers are renting their vehicles within the first two days. These two days depict the initialization phase of the simulation, in which many customers desire to rent a vehicle due to the starting conditions set in the customer class. As of now, this issue is hard to address and can solely be mitigated through specific functions described in the documentation of the customer class. In the course of the simulation, the respective effects quickly recede, thereby being replaced by a more realistic driving behavior.

In general, 220 day time rentals were conducted throughout the simulation period, although it has to be mentioned that a notable portion of these rentals were allotted to the
first two days of the simulation. However, it is to be expected that by finding more exact starting conditions, the distribution of rentals would spread out evenly, thus illustrating a natural rental behavior within the first days of simulation. Likewise, the sum of night rentals adds up to 131 throughout the 14 days of simulation. Furthermore, as intended, the number of both night and day rentals differs significantly, thus reflecting a different driving behavior during either day or night.

![Figure 11 Waiting Times Scenario 1](image)

The customers average waiting times also reflect and correlate with the current driving behavior. As can be seen in figure 11, the highest waiting times are recorded throughout the first two days, mounting up to a maximum of 35 minutes during the first day time period. However, it can also be seen that afterwards, the average waiting times sporadically increase to a maximum of 20 minutes mostly occurring during the day time intervals. At night, waiting times are rarely recorded, as the customer’s desire to rent a vehicle is also significantly lower. Therefore, it is relatively unlikely that two customers want to rent the same vehicle at the same station at the same time.

The revenue being generated totals to 6,075.67 euros. On the other hand, the costs accruing to the various fixed and variable costs described in the simulator class, add up to 90,839.72 euros. As a consequence, the whole car sharing service suffers financial losses in the amount of -84,710.38 euros. It also has to be noted that there are short periods in which the revenues actually surpass the costs and by that the car sharing
service generates profits. However, most of the time, even during the day, the revenue being generated is too low to counteract the costs and thus, solely reduces the slope of the decline. The figure depicting the profit is solely shown once here in the basic scenario, as the graph will not differ notably between the various scenarios due to the high operating costs of the car sharing system which were assumed and determined in the section about the documentation. These costs are so high that the respective visual differences between the scenarios are negligible and barely visible.

As can be seen from the previous figure, the average charge of the vehicles is constantly declining. By the start of the simulation, the average charge adds up to 21.33 kWh, as it consists of the individual maximum charges of the three vehicle types. During periods of high usage, especially at the beginning of the simulation, the average charge significantly drops. As soon as the number of actively driving customers decreases, which occurs especially during the night hours, the average charge increases. This is due to the cars being recharged at the charging stations. However, it does not reach its initial height, but shows a continuous decline which indicates that the normal chargers are insufficient to maintain consistently high levels of battery charges. This can also be seen by examining the total average charge throughout the simulation period which mounts up to 20.9591 kWh, being a decline of 0.17%.

The vehicle availability of the proposed car sharing system is comparably high. Through-
out the 14 days of simulation, 35 customers were unable to rent a vehicle due to the vehicles being unavailable. This number has to be further mitigated because of the driving peaks during the first two days. The majority of these 35 unsuccessful rentals emerge during the first day, as the number of desired rentals surpasses the number of available vehicles. Therefore, a lot of customers are forced to wait. In many cases, the actual wait times surpass the customer’s maximum wait times and thus, lead to a termination of the rental process, successively increasing the number of unsuccessful rentals. However, the number of rentals that were rejected due to insufficient vehicle charges is negligible. In total, solely three requested rentals were unsuccessful due to vehicle’s SOC.

The number of customers that never desired to rent a vehicle due unacceptable walking distances is relatively high. In total, 602 customers dropped out of the system, thus indicating that an increase in stations is required to boost the car sharing system’s attractiveness.
Scenario 2

In scenario 2, the effects of increasing the number of stations while also changing the station’s composition of chargers were examined. First of all, it is to be noted that the number of active customers increased notably, since adding stations also entails that more stations are within the maximum walking distance of previously unavailable customers. As a consequence, the number of so-called Drop-Outs decreased from 602 in scenario 1 to 441 in this scenario which is a reduction of 34.67%.

Comparing the driving patterns of this scenario with the one being depicted in scenario 1, indicates that the overall number of rentals increased significantly. While the number of driving customers mounted to 33 during the peaks within the first day period, it now shows that at best, 46 customers were driving at the same time.

This trend was also illustrated by the number of both successful day and night rentals. Reducing the amount of drop-out customers by deploying more stations boosted the total number of successful day rentals to 373 which is an increase of 69.54%. Furthermore, the number of successful night rentals, increased to 236 in total, thus being a growth of 80.1% compared to scenario 1. As can be seen, the successful night and day rentals increased proportionally to the decrease in drop-out customers. Presumably, all of the new customers drove within the 14 days of the simulation due to the starting conditions set in the customer class. 161 new customers became active and the amount of successful
day rentals increased by 153 in comparison 1. Therefore, the previous assumption is backed by actual numbers.

![Figure 15 Waiting Times Scenario 2](image)

While the number of customers increased due to the higher number of stations, the waiting times altogether remained the same. Since the 'new' customers were distributed evenly, they would solely choose the stations that were newly added to the system. Consequently, the total number of customers using the 'old' stations remains unchanged. However, as more customers are waiting, more cases of customers waiting occur, especially throughout the first day. Accordingly, the bars displaying the waiting times are broader during the respective time period.

The effect of using solely fast chargers is rather drastic. Although the average charge drops during the time of the customer's rentals, it is quickly restored to its maximum of 21.33%. Altogether, the average charge amounts to 21.25 kWh throughout the entire simulation which is a minor decrease of 0.03%. Compared to scenario 1, the overall reduction is decreased by a factor of 58.93.

As total revenue is directly linked to the number of successful rentals, the former increases significantly from 6075.67 to 10558.31 euros which is an incline of 73.78%. However, as each new station entails monthly costs, the total costs increased as well. In scenario one, the total costs of operating the car sharing service for two weeks amounted to 90,839.72 euros, whereas in this scenario, the overall costs grew to 100,443.72 euros, thus being an
increase of 10.57%.

Finally, the vehicle availability was not affected by deploying more stations. In scenario 1, 35 cases occurred in which customers were unable to rent a vehicle due to the latter being unavailable. In this scenario, the number increased marginally to 37 vehicles in total which might be ascribed to the stochastic and thus, random nature of various input data. However, the number of unsuccessful rentals due to vehicles having insufficient charges was almost completely dissolved to 1 vehicle in total which is a decrease of 66%.
Scenario 3

In scenario 3 the driving frequencies were increased by 20% to examine the impacts on the vehicle availability, number of rentals, and the average charge.

First of all, the driving pattern changed slightly, as both the driving peaks at the beginning as well as the general number of rentals increased. However, most notably the peak during the first night did not increase but decrease from 13 to 10 driving customers which might be accrued to the stochastic nature of the input. In comparison to scenario 1, the peak at the very beginning of the first day period raised notably. Whereas it mounted up to 32 customers in scenario 1, it changed to 37 customers driving at the same time in scenario 3 which is an increase of 8.57%.

The total number of rentals also increased from 220 day time rentals to 298. Therefore, changing the driving frequencies by 20% led to a total increase of 26.17%. Regarding the night time rentals, they increased from 131 in scenario 1 to 151 in scenario 2. Thus, the percentage increase solely added up to 13% in total. This relatively small increase might result from the lower standard deviation during nights that is applied in the process of determining a customer’s desire to rent a vehicle at night.

Besides affecting the number of rentals, an increase in the driving frequencies also led to a gain in waiting times. In scenario 1 waiting times were mostly restricted to the
beginning of the simulation and some sporadic occurrences during day time periods. In scenario 3, however, the average waiting times mounted up to 20 minutes and higher on multiple occasions.

The revenue in scenario 3 mounts up to 7131.24 euros in total. Compared to scenario 1, this constitutes as an increase of 17.38%, thus almost matching the increase in driving frequency of 20%. The costs of the car sharing system increased by 3.94% equaling 3588.35 euros; hence, in comparison to the increase in revenue, the rising costs are negligible.

Moreover, changing the driving frequencies had a rather small but noticeable effect on the average charge. In scenario 1, the average charge amongst all vehicles was to be set at 20.9591, whereas it decreased by about 1 point to 19.3536 in scenario 3. Therefore, increasing the driving frequencies by 20% lead to an 8.29% decline in battery charges. Probably, the charge decline is lower than the increase in driving frequencies, since the increase of rentals is distributed all over the simulation period. Therefore, the chargers generally have enough time to alleviate the respective effects.

As the number of successful rentals also depends partly on the average charge, the number of unsuccessful rentals increased. In scenario 3, the number of times rentals were rejected due to insufficient vehicle charges doubled from 3 to 6 compared to scenario 1,
thus implying an increase of 100%. However, compared to 298 successful rentals in total, this change is rather imperceptible. Nevertheless, in 54 cases requests were turned down due to stations not having vehicles. Compared to scenario 1, this is an increase of 54%. Presumably, this drastic change is ascribed to the increase of customers driving at the beginning of the simulation. Since the number of vehicles is kept consistent, even more customers can not be served during the starting peaks on day 1.

Scenario 4

In scenario 4, the maximum wait times of the customers were significantly increased. On the one hand, this was done to examine the respective effects on the number of successful rentals, profit, and the average charge and on the other hand, to see whether the customers would act as intended.

As can be derived from figure 20, the driving pattern changed notably in comparison to scenario 1. Both maximum values of the driving peaks throughout the first day remain the same, as the driving frequency, or, in other words, the minimum intervals, as well as the standard deviations of the lognormal distribution are kept consistent. However, figure 18 also shows that more customers are driving throughout both day and nights, as the driving peaks across the entire simulation are visibly higher.

This visual assessment is backed by the actual number of both day and night rentals. In
total, 271 customers rented vehicles during the day time periods within the 14 days of simulation. Compared to scenario 1, this amounts to an increase of 23.18%. Considering that the maximum wait times were almost tripled, this change is rather minor. Yet, the result was expectable, since the maximum waiting times are generally only exceeded during the very beginning of the simulation. Therefore, the positive effects of increasing the waiting times mostly applied solely within the first two days of the simulation. This explanation is further supported by the total number of night time rentals. 159 night time rentals were examined, whereas in scenario 1, 151 rentals were recorded, thus constituting as an increase of 5.29%. Since the total number of driving customers during the first night is comparably low to the number day time rentals, most of the customers will find a suitable vehicle almost immediately. As a consequence, the number of waiting customers is also lower, thus increasing everyone’s chance to find a vehicle within a shorter period of time.

The previous figure indicates that an increase in maximum waiting times mostly affects the average waiting times during the initialization phase of the simulation. As described before, most of the customers want to drive at the beginning of the simulation. However, as the number of driving requests exceeds the number of vehicles, many customers will have to wait. Furthermore, as the customers that successfully found a vehicle are going to use it for a while, only a small number of vehicles will become available again within
the first hours of simulation. Consequently, the majority of the customer’s actual waiting times will exceed their maximum waiting time which is why the average waiting times during the first peak on day one almost matches the maximum waiting time of 120 minutes. Throughout the following days, the average waiting times will reach 40 minutes on multiple occasions.

The total revenue reflects the increase of successful rentals. As expected, it lies between the revenue of scenario 1 and 3, since the total number of successful rentals in scenario 4 is located between the ones of the respective scenarios. In total, 6708.17 euros of revenue were generated throughout the entire simulation. The costs, on the other hand, match the costs of scenario 3; the total profit, or, since the company actually made losses, the total loss mounts up to 88,728.06 euros.

The average charge and its progression throughout the simulation mostly match the one from scenario 2. Compared to scenario 1, the average charge decreased from 20.9591 kWh to 19.6754 kWh or 6.55%. Yet again, the decline is relatively small, as the chargers supposedly mitigate the effect of an increase in rentals during night times, when the majority of customers are sleeping.

Finally, the number of unsuccessful rentals is significantly reduced. Whereas in scenario 1, the number of rentals being terminated due to stations not having vehicles was 35,
the respective number decreased to a total of 19. Therefore, increasing the maximum wait times led to a decline of 84.21% compared to scenario 1. This was to be expected, since almost tripling the waiting times especially helps during the initialization of the simulation where many customers want to but cannot rent a vehicle. Likewise, the number of times vehicles did not have enough charge for the various trips solely increased from 3 to 4. Although this is an increase of 33% compared to scenario 1, it is still considerably lower in comparison to scenario 2. As the customers were waiting up to two hours for a vehicle, many of the latter were recharged up to the point where they became suitable for the customer’s desired trip.

Figure 22 Average Charge Scenario 4
Scenario 5

In scenario 5, the distribution of customer types was adjusted to simulate the effects of varying demographic structures throughout the diverse city districts.

First of all, the driving pattern hints at one of the major issues of changing the distribution of customer types, as the driving peaks throughout the first day are significantly lower compared to all other scenarios. Whereas in scenario 1, 33 customers were driving at the same time during the first day, the amount decreased 26, indicating that not enough vehicles were available. This was to be expected, as the customers prefer a certain type of vehicle. Since customers of type 1 mainly live in quadrant 2, the demand for vehicles of type 2 is especially high in that district. As the vehicle capacities are kept the same, many requests cannot be served. Likewise, in quadrant 1 vehicles of type 3 are requested more frequently, as customers of type 3 mainly live there.

Most notably is that the number of successful day and night rentals did not decrease in comparison to scenario 1. In total, 225 day time rentals and 141 night time rentals were conducted, thus being an increase of 1.8% and 7.63%. However, this discrepancy between expectation and reality might yet be explainable. The strongest effects of certain customer types living in specific districts of the map are measured during the first day. Additionally, since during the consecutive days solely five customers are driving at the same time at best, the chance of two customers using the same station at the same time
is relatively low. Even if this is the case, one of both customers will be able to rent the vehicle. As a consequence, the number of both successful and unsuccessful rentals increases, as one customer finds a vehicle whereas the other terminates the rental process.

In comparison to scenario 1, the number of occasions when customers had to wait for a vehicle increased. The reasons for this behavior are the same as stated above. More customers want to rent the same type of vehicle, although the stations solely possess one of each kind. Accordingly, these customers have to wait for the vehicle to become available once again.

As for the revenue, it slightly increased compared to scenario 1, since the number of successful rentals increased as well. Altogether, the car sharing service generated 6617.45 euros of revenue, thus being a growth of 8.92%. The costs, however, also grew by about 4.43% stemming from the slight increase in rentals.

During the simulation of this scenario, the average charge did not change noticeably compared to the other scenarios. Although some vehicle types were used far more frequently and thus, had a lower average charge throughout the 14 days of simulation, many vehicles were not used at all and therefore, did not lose any charge. For instance, as customers of type 1 solely lived in quadrant 2, vehicles of type 2 that were parked at stations in the other quadrants were never used. As a consequence, the higher vehicle utilization in one district was compensated by the lower utilization in the other districts. This can

![Figure 24 Waiting Times Scenario 5](image-url)
also be seen by the overall average charge of 20.9459 kWh. Compared to scenario 1, the average charge decreased by 0.06%, thus being negligible.

Most noteworthy is the number of unsuccessful rentals due to vehicles being missing or not having the right vehicle type. In scenario 1, solely 35 respective cases occurred and those mainly took place during the first day of simulation. However, in this scenario, the number increased to 91 in total which is a raise of 260%, consequently emphazising the importance of demographic structures.
Scenario 6

Scenario 6 is a special case, since it examines the effect seasons are having on the success of car sharing systems. As mentioned before, it is assumed that it is winter, thereby reducing the vehicle charge by about 30% due to the batteries dependency of ambient temperatures. Doing so helps to evaluate whether the existing car sharing system is equally suitable in winter as it is in summer. Moreover, the nominal consumption is increased, since the heating is powered by electricity.

As can be derived from the previous figure, the driving pattern does not change notably compared to scenario 1. The spikes throughout the various day time periods are higher and more pointed. This, in turn, hints at customers waiting more often for their vehicle, as many customers start waiting when the day time period begins. By mid-day, many of the customers find a suitable vehicle, as, for instance, the latter were charged beforehand and have the appropriate SOC.

In general, the number of successful rentals decreased in comparison to scenario 1. In total, 205 day time rentals and 122 night time rentals were recorded which constitutes as a total decline of 7.33%. This rather small decline is to be noted, as the vehicle charges were reduced by 30%, whereas the consumption of each vehicle was increased by about 15%.
The waiting times during the winter noticeably climbed but not as significant as expected. In comparison to scenario 3, they even diminished. However, customers were forced to wait more often, as presumably, they had to wait for the vehicles to be charged sufficiently. Altogether, when considering the winter season, customers had to wait 22 times, whereas they solely had to wait 12 times when it was summer. Therefore, the increase accounts for 83%.

Due to an overall decrease in successful rentals, the revenue was also notably lower than it was in scenario 1. During the simulation period, solely 5769.69 euros were generated. Therefore, the revenue was diminished by 305.31 euros, being a decline of 5.2% in comparison to scenario 1. As these 5.2% come close to the decrease in overall rental (7.33%), the respective result seems appropriate.

The average charge during the winter is, as stated above, set by default as 30% lower than the usual battery capacities. However, as can be seen in the figure 22, the steepness of the decline of the average charge is significantly higher, as the electric heating systems increase the consumption by 15%. Overall, the average charge of the three vehicle types amounts to 14.93 kWh. However, during the course of the simulation, it receded to 14.5467 kWh being a reduction of 2.65%. In scenario 1, the average charge was set to 21.33 kWh per default. As it decreased to 20.9591, the overall decline was solely 1.76%.

In conclusion, increasing the consumption by 2 kWh/100 km does not affect the system...
as predicted, since a higher rate was to be expected.

In total, the number of successful rentals declined as stated above. However, the reasons for this failure shifted, as in this case, the number of unsuccessful rentals because of too low charge levels increased to 9 in comparison to scenario 1, thus constituting a 300% gain. The amount of rentals that were terminated due to vehicles being unavailable was found to be quite similar to scenario 1, as this case occurred 38 times throughout the simulation gaining 8.57%. 

Figure 28 Average Charge Scenario 6
5.3 Interpretation & Implications

The previous subsection was conducted both to investigate the effects of changing the diverse system variables, thus allowing to determine the importance of each individually, and to ensure that the simulator behaves as intended. It can now be concluded that the system not only operates as intended, but also that most of the various customer and system-related properties are highly important for the performance of the system. That being said, the respective results can be used to provide guidance and to derive various implications for car sharing operators and researchers alike.

First of all, the most crucial factor for the success of car sharing systems is either the total number of active customers or the frequency of them driving. As can be seen in scenario 3, the frequency of rentals has the biggest impact on the revenue of the car sharing system. Increasing the frequency by 20% led to a 17.38% gain in revenue. Although higher frequencies imply higher variable costs, as, amongst others, they increase maintenance costs due to the abrasion of vehicle components, the growth in revenue highly outweighs the growth in costs. Whereas the revenue grew by 17.38%, the costs solely increased by 3.57%. As a consequence, car sharing operators are inclined to search for options of increasing the driving frequencies. This can be either done by offering financial incentives, i.e. discounts, for renting vehicles multiple times a month, or by decreasing the hourly fees. However, the respective effects can hardly be predicted, as both in reality and in the simulation, customers react according to their individual price elasticity.

A further possibility of improving the attractiveness of the car sharing system consists of deploying many car sharing stations across the map. As can be derived from a comparison between scenario 1 and 2, the number of active car sharing users directly depends on the distance each of them has to walk to their nearest station. Whereas in scenario 1, 602 of 1000 potential customers fell out of the system because of unacceptable walking distances, deploying more stations in scenario 2 increased the number of active members by 155 to a total of 553. Furthermore, most of these users rented a vehicle within the two weeks of simulation, thus increasing the revenue by 73.78%, whereas the costs solely increased by 10.57%. Consequently, the financial gains yet again outweigh the losses. However, as each new station entails high investment costs, car sharing operators are inclined to deploy station placement algorithms, thus optimizing the distribution of stations before their actual implementation.

However, scenario 2 not only showed the importance of the number of stations, but also the significance of the type of chargers being used at each station. Compared to all the other scenarios, solely in this scenario the average charge among all vehicles retained its initial level. In every other scenario, it continuously declined, although hardly any impact on the number of unsuccessful rentals was recorded. However, in longer simulations, this
continuous decline could lead to more rentals being unsuccessful. Therefore, each station should at least operate with one or more fast chargers to counter this development.

Nevertheless, not only the number of customers or their driving frequencies were shown to significantly influence the performance of the car sharing system. In addition, the demographic structure of city districts is to be kept in mind when deciding upon the distribution of stations and vehicles. As the type of customer defines his vehicle preferences and maximum walking distances, it is crucial for car sharing operators to learn about the specific demographic structure of each city. According to these structures, car sharing operators should adjust both the number and distribution of vehicle types correspondingly. For instance, if primarily families or students live in a specific city district, either the number of compact cars or station wagons are to be increased. Additionally, if a city district has a high population density, the station capacities need to be increased, as having one vehicle of each kind might be insufficient to handle all requests.

Finally, it was shown in scenario 6 that seasons and the temperature-related decline of battery capacities also have to be considered when setting up a car sharing system. By reducing the battery capacities for about 30% and increasing the consumption of the vehicles by 2 kWh/100 km due to electric heating, the vehicle availability and the average charge was decreased, whereas the waiting times increased notably. Consequently, car sharing operators should either increase the station capacities or deploy more fast chargers.

Nonetheless, the scenario analysis has shown that the general profitability of car sharing services is to be questioned. In none of the simulation runs, profits were generated, as the total costs highly outweighed the total revenues. Although the investment costs of setting-up the system were always taken into account, thus causing the simulated car sharing system to start with negative profits right at the beginning of each simulation run, a steady decline was nevertheless noted in all scenarios. Even more so, as the proposed car sharing solely allowed for two-way rentals, no expensive relocation strategies were required. As the general tendency goes towards one-way car sharing system, it is expected that future E-car sharing systems will also provide customers with the option to start their trip at one station and to terminate it at another. Therefore, respective car sharing operators would need to either apply user- or operator-based relocations strategies, whereas the former would reduce the revenue being created by the trips due to financial incentives being offered. The latter, however, would require specialized staff members solely relocating the vehicles, thus implying monthly wages. Furthermore, as the electric vehicle’s total battery capacities decline steadily, mounting up to 15% per 100,000 km, car sharing operators would be forced to continuously renew their vehicle fleet in order to retain a constant vehicle availability. Thus, the total replacement costs would be higher compared to a common car sharing system using vehicles depending on
fuel. As a consequence, it is not to be expected that car sharing operators will become more profitable in the long-term.

In conclusion, the scenario analyses have shown that using electric vehicles in car sharing systems is a viable option. Only in rare occasions, the vehicle charges were insufficient for handling the individual requests although the required, minimum charges were actively increased by 30% serving as a buffer for cases of unexpected delays or detours. Even during winter, when the battery capacities were temporarily decreased due to the ambient temperatures, most rentals were unsuccessful solely as a result of missing vehicles.

6 Critical Reflection

This section deals with a critical acclaim regarding the approach and the results of this paper. It is divided into subsections 5.1 and 5.2, namely system-related issues and process-related issues. In the subsection about system-related issues, problems are described that arise due to the choice and modulation of the simulation system and input data. The subsection about process-related issues discusses problems and uncertainties that emerge due to the manner in which the car sharing process was coded.

6.1 System-Related Issues

One of the major drawbacks regarding the choice of the system model and its behavior is that a quasi-continuous model, as well as continuous models, significantly decrease the performance of the simulator. Although the simulator was intended to simulate longer time periods such as multiple months or more, it is now solely feasible to examine a single month, as even that requires a couple of hours. This is mainly due to two factors. First of all, the time steps that were chosen are relatively short, as the simulation advances one minute at a time. As a consequence, each day alone requires the simulator to perform 1,140 iterations mounting up to 43,200 iterations for a single month. Furthermore, due to its focus on single customers, the simulation is to be seen as a so-called micro-simulator in which the interactions between the system’s entities are modeled in great detail. Therefore, the simulator checks and changes each customer’s status, whereabouts, and desire to rent a vehicle including other corresponding properties during every iteration. As long as only a handful of customers are simulated, the performance is acceptable, whereas the performance rapidly decreases as soon as the number of customers exceeds two-digit numbers. Besides handling the customer, station, and vehicle objects, the simulator is also required to display various key performance indicators and the progress of each customer’s trip on the overall map. In its entirety, the visualization of these outputs requires even more computational power than the simulation taken by itself.
Additional issues might arise from the modeling of the input data. Although most of the input data is set deterministically, some of it is determined stochastically. Therefore, the system is affected by uncontrollable and random inputs causing the results to be more or less random, too. Therefore, running this stochastic or partly stochastic simulation model is like performing random physical experiments, as the user will probably see different results each time the simulation is conducted, even though the input remains unchanged. However, this cannot be changed, as the behavior of the entirety of customers is supposed to be mostly unpredictable. Predefining fixed values for each of the customer’s properties would either mean that all the customers act identically, or the user would have to determine individual values for every single customer object which is unfeasible for a high number of customers.

Moreover, much of the input data regarding the customer behavior was solely deduced and assumed by the author, as it is near impossible to find respective values for their numerous properties. Hence, the validity of the results cannot be guaranteed, but, as the simulator was more or less designed to be a proof of concept that entire e-Car sharing processes can be simulated, researchers can build upon this foundation. By studying the customer behavior in car sharing systems, researchers can find valid input data for each and every entity, thus constantly improving the capabilities of the simulator.

A further disadvantage concerns simulations in general. As mentioned before, the simulator is designed to serve as an easily customizable decision-support system for researchers and conventional users alike. Finding the ideal values for the initial set-up of the car sharing system, however, requires a lot of effort. In general, the user defines various scenarios, evaluates these with the help of the simulation program and consecutively chooses the best option. Hence, simulations require numerous runs to evaluate a variety of possible combinations of the various input parameters which is both time-consuming and tedious.

Another problem might arise due to the relatively short simulation period. As can be seen in the various scenarios, the average charge of the vehicles is constantly declining if mostly normal chargers are used. If the simulation time is increased to last a month or even longer, the average charges of the vehicles will supposedly reach more critical levels, where the number of unsuccessful rentals will notably rise due to insufficient charges.

Finally, the size of the current map might lead to further problems. As of now, it depicts a relatively small portion of the city’s map. Therefore, the distance the customers can and will drive are also relatively small. As a consequence, the charges of the vehicles will in most cases suffice for each individual trip; accordingly, the number of trips that fail due to insufficient charges is possibly largely underestimated.
6.2 Process-Related Issues

One of the main advantages of this simulation is that it simulates the whole process, thus offering various opportunities to influence all variables which is, as of now, also one of its major weaknesses. Due to the lack of specialization, most events of the car sharing process are modeled quite intuitively, thus oversimplifying much coherence. One of these over-simplifications can be found in the modeling of the travel demand. As mentioned in subsection 2.2, researchers often develop and apply utility functions to predict the travel demand of a system. During these simulations each customer’s utility drawn from a utility function is maximized by determining the customer’s driving schedule, consequently simulating a realistic decision process that could have as well taken place in the real customer’s minds.

Although it is undeniably an oversimplification, it is still to be questioned whether a highly detailed travel demand prediction using utility functions truly is needed. In the end, it solely models the customers desire to rent a vehicle; why and how that wish came about is probably of minor interest. As long as the utilization of the log normal distribution leads to realistic driving patterns and frequencies, it is a viable option. Additionally, it is easily accessible for any user being experienced in the field of statistics, as he is only required to change mean intervals and standard deviations in order to adjust the driving patterns to his liking. Thus, using utility functions might be more precise and closer to the customer’s actual decision processes, but in the end, the results probably will not differ drastically.

A further weakness of the current travel prediction and driving behavior is that customers will stop solely once throughout the trip upon reaching a certain portion of the total distance between two stations in order to perform their tasks. After finishing these tasks, the customers simply drive back to their starting station and hence complete the rental process. However, in reality it would be highly likely that many customers instead have a variety of destinations and a chain of activities to be performed throughout a single rental. For instance, if a vehicle is rented for the weekly shopping, it is unlikely that all required goods could be bought at a single store. Therefore, the driver would stop multiple times whilst driving all over the map instead of once driving a straight line between two stations. This in turn, would both have an impact on the profit, as parking is less expensive than driving, and the overall battery consumption of the trip, because the actual length of the trip would inevitably increase. However, the respective effects are to be mitigated, as the current length of the customer’s duration of stay might as well count for the whole trip and thus, is to be seen as the sum of all stays. Hence, mostly the trip length is affected by the current depiction of the process.

Furthermore, it could be reasonable to introduce different driving patterns for each
customer type, since families probably use their vehicles differently compared to either singles or students. For that purpose, various activities, such as shopping or sports, could be introduced to the simulation. The probability to perform either one or more of these activities would then differ between the various customer types, thus creating a more flexible and realistic driving behavior of each customer.

Besides varying the trips between the customer types, it could further be advisable to introduce different driving patterns throughout the week and the weekends. During the weekdays, the rental times could be lowered on average, as many of the customers would probably rent vehicles either during their lunch breaks or after work, thus significantly decreasing the time span during which they could actually perform the tasks they need the vehicles for. During the weekend, however, the rental times and the distance of the trip should increase, as some customers would rent vehicles to go on lengthy excursions such as visiting amusement parks or spending a day at the lake. Furthermore, many people wait for the weekend to do their shopping which is why it could be reasonable to increase the driving frequency by either lowering the standard deviation or decreasing the minimum interval between rentals. Additionally, it is more likely that customers would rent a vehicle during the night when they are not required to work the morning after, that is, during the weekend. Thus, the minimum interval of the night rentals should also be decreased respectively. As of now, these possibilities are not considered in the simulation.

Another factor hampering the validity of the simulation results is the way the road network is modeled. As mentioned in the specific concept, the road network and thus, the customer’s trips are determined by the connections between the stations. Since these constitute as the drivable roads, customers can solely drive from one station to another. Consequently, it is impossible to drive to every location on the map, except for when the number of stations is increased drastically, which is economically unfeasible. Furthermore, as the customers will always drive the Euclidian Distance or in other words the flying distance, the distance is significantly underestimated. To counter this, a corrective factor generally increasing the trip distance by 39% was introduced. The factor itself was determined by comparing the actual driving distance with the flying distance of various trips in and around Hanover city and calculating the average difference thereof. However, in some cases the difference between the flying and driving distance was comparably negligible or far higher than the average. In these cases, the factor would have been set as either too high or too low, therefore over- or underestimating the true distance of the trip.

The current depiction of the process of customers choosing their starting station might also negatively influence the validity of the results. In its current form, customers will always choose the station that is closest to their place of living. If the station is within
their walking distances, the customers will check the very station for available vehicles. In case there are no vehicles available and the customer waits longer than his individual maximum wait time, he will abort the rental process. However, in reality customers would then look for another station that also is within their walking limits and check for vehicles there. As a consequence, the likelihood of finding a vehicle, and thus a successful rental, would proportionally increase to the number of stations in the customer’s vicinity, as the other possible starting stations might have some appropriate vehicles at their disposal. However, as the stations are sparsely placed among the map, it is not very likely that more than one station is within the customers walking limits. Thus, the accruing effects on the results are probably to be mitigated.

A further issue with the process of choosing a starting station arises from the assumption that the customers will solely start the rental at a station close to their homes. In the current state, the customer will never use the car sharing service when there is no station near their homes. The simulator will then count the respective individuals as drop-out-customers, which is used as a performance indicator. However, in reality some of the customers probably never intended to use the car sharing service for exclusively renting vehicles close to their homes. Instead, they also require the vehicles, for instance, during their lunch break at work or in between the lectures at their university. Thus, the number of drop-out customers might be overestimated compared to the number of actual Drop-Outs.

Another problem might arise from the behavior of the simulator when the customer does not find a suitable vehicle. In this case, the program behaves the same way like it does when the customer successfully finishes his trip, as the time of his last rental is set to the current simulation time. As a consequence, the customers will wait the full minimum time interval until their next rental, or in other words, their desire to rent a vehicle is completely reset. However, in reality many of the tasks the customers rent the vehicle for could as well be carried out within the subsequent days, consequently leading to a successful rental. The current system’s behavior thus might have a significant influence on the overall profitability of the car sharing service, since it directly decreases the total amount of rentals.

Most of the common car sharing services allow customers to reserve vehicles prior to the actual rental. However, as described in the subsection 3.1, it is assumed that the proposed car sharing system does not dispose of any reservation features. This in turn might lead to an overestimation of both the waiting times and the number of unsuccessful rentals. Currently, customers will try to rent a vehicle as soon as the respective conditions are met and thus, the desire is tied to a particular point in time. As soon as they want to do so, the customers will check for available vehicles and wait if there are none. However, if the car sharing service offers reservations, the customers will check the vehicle availability before
they actually want to rent the vehicle. In case no vehicles match the requirements, the customer will simply refrain from using the car sharing service, thus completely avoiding any potential waiting times.

As of now, the vehicle availability might further be overestimated due to the lack of a process depicting the cases of vehicles having malfunctions. Usually, the respective vehicles would be required to be driven to the next car service station, where they eventually would get repaired. As a consequence, the vehicles would drop-out of the pool of available vehicles until the malfunction is fixed which could either take a long or short period of time according to the severity of the damage.

Another problem occurs during the initialization of the simulation. Throughout the first few days, the number of driving customers is overestimated, as many last rentals are set lower than the minimum interval by the starting condition. As of now, there is no obvious way to completely avoid this problem, although functions were implemented to mitigate this effect to a certain level. As a consequence, the waiting times and the number of unsuccessful rentals are higher at the beginning. Furthermore, many of the customers that drove during the first day would have been distributed over the subsequent days. Therefore, the driving pattern is distorted throughout the first week of simulation. However, the respective effects can be ignored and omitted from the examination, as the overall number of rentals remains the same. Nonetheless, the number of unsuccessful rentals is overestimated, as too many customers request vehicles at the same time and terminate the rental process when they waited for too long.

Also, as of now the user cannot define the number of vehicles at each station individually. Therefore, in its current form, the simulator cannot be used to individually alter the number of vehicles or the distribution of vehicles when needed. For instance, if the demographic structure is to be altered, it would be advisable to increase the number of vehicles that have the vehicle type preferred by the customer type which is mainly settled in the respective district.

Finally, the current modeling of the coherence between price and travel demand is oversimplified. Currently, the simulator assumes a linear relationship between both, as the demand decreases proportionally with the price, thus implying an isoelastic price elasticity. Yet, it is far more likely that the price elasticity is greater than one, consequently indicating an elastic demand. Elastic demands are characterized by a progressively diminishing total demand. In other words, the change in the demanded quantity is greater than the percentage change in price. In this manner, the percentage decline in travel demand would grow with an increase in the price.
7 Conclusion

Due to the continuous research and subsequent improvements of battery technology and a general decline in the acquisition cost of electric vehicles, the latter’s popularity has been shown to grow steadily. Besides a general increase in private use, electric vehicles have become more and more popular in the domain of car sharing. Despite this increasing popularity, researchers have almost exclusively confined their studies either on field tests of already existing e-Car sharing systems, or on simulating car sharing services using common vehicles. In order to fill this research gap, the aim of this paper consisted of developing a car sharing simulator including electric vehicles and their peculiarities, thus allowing researchers or other users alike to preemptively test for the viability of e-Car sharing systems by conducting various scenario analyses.

To provide the reader with a convenient access to the topic, the thesis began with describing the diverse basics in the domain of simulation. The various types of simulation models were categorized and simulation systems and their components were explained. Subsequently, a literature review of simulations in the domain of car sharing was conducted. It was found that most of the respective simulations focus on certain fractions of the car sharing process and none of them included electric vehicles in their observations. After finding and describing this research gap, the concept for the e-Car sharing simulator was developed. For that purpose, a car sharing process was visualized, according to which the features of the simulator were derived. Furthermore, as a simulation system is always a simplified depiction of reality, simplifying assumptions, including omitting of reservations and one-way-rentals, were made. Afterwards, these general features and the interactions therein were translated into a specific concept that included a description of all entities, their properties, and their part in the simulation.

In the following section, the specific implementation, that is the translation into code, was explained in great detail. The simulator consists of the four classes Simulator, Vehicle, Station, and Customer. Whereas the last three classes play a passive role in the simulation, the simulator class defines the behavior of the simulation system, since it encompasses the actual simulation process.

In order to test for the applicability of the e-Car sharing simulator, multiple scenarios were defined and distinguished. Furthermore, expected tendencies and respective outcomes of these scenarios were suggested, in order to check whether the actual simulation results meet these expectations, and thus, if the simulation behaves as intended. Afterwards, the simulation of the scenarios was conducted and the results were presented. Subsequently, the results were interpreted and recommendations for car sharing operators were derived. It was found that both the number and the type of the charging stations play a significant role for the success of the e-Car sharing companies, as they
have a critical influence on the average charge and thus, on the vehicle availability. The vehicle availability, in turn, has been found to notably influence the profitability of the car sharing service, because it directly impacts the number of successful rentals, which in turn are responsible for generating revenue. However, it was also found that these impacts are comparably small as long as the car sharing service is quite new and the frequency of rentals is low. In these cases, the intervals between each rental were mostly long enough that the vehicle had enough time to become sufficiently charged for the next trip. Moreover, it has been found that the demographic structure of a city district also plays an important role for the success of car sharing systems, as different customer structures require different vehicle distributions due to their vehicle preferences. Thus, car sharing services are required to ensure that the distribution of vehicle types matches the customer distribution and their preferences. In addition, the number of stations was found to be a critical success factor as well. The car sharing companies are required to reflect on the optimal number of stations. On the one hand, they are expensive to build but on the other hand, having not enough stations leads to a significant decrease in active customers, as the customers will not walk greater distance to rent vehicles.

Finally, the results of the simulation were reflected critically, because both during the design of the simulation and its translation into code, simplifying assumptions were made by the author. These are expected to hamper the validity of the results, since they lead to either an over- or underestimation of certain key performance indicators. For instance, omitting a reservation feature was discussed. It was expected to significantly increase the waiting times, since currently, customers will go to their next station and, if no vehicle is available, will wait until either they waited too long, or a suitable vehicle arrives.

However, one of the biggest issues of the proposed simulator is the significant lack of scientifically backed input data. Therefore, the current customer behavior is more or less based on assumptions, as car sharing operators and researchers alike are very secretive when it comes to customer data and especially driving behavior such as number of rentals and length of stay. Furthermore, the validity of the simulation could not be confirmed by comparing it with other simulators, because no similar systems and e-Car sharing simulators exist.

As a consequence, future users, be it researchers or car sharing operators, are inclined to find out the actual usage and customer driving patterns, thus allowing for an appropriate customization and calibration of the current simulator. Doing so would increase its applicability and reliability, consequently allowing users to use it both for supporting their decision-making regarding either existing or conceptual car sharing systems.
References


Cepolina and Farina (2012a) : Urban Transport XVIII. WITPRESS.


Figure 29 Appendix 1: Driving Customers
Figure 30 Appendix 2: Peaks during the morning hours
Figure 31 Appendix 3: Quadrants describing the place of living
Figure 32 Appendix 3: A Fully Set Customer

```
Customer with properties:

  id: 1
  isDriving: 0
  rent: 0
  lastDayDrive: -3.2405e+05
  lastNightDrive: -5.3068e+04
  fromStation: 0
  fromId: 0
  toStation: 0
toId: 0
  currentpos: 0
  myCar: []
  sumdayrentals: []
  sumnightrentals: []
  custType: 1
  minIntervalDay: 1.7582e+04
  minIntervalNight: 7.2140e+04
  pLiving: [2x1 double]
  lQuad: 1
  maxdistKM: 0.8300
  maxdurationStay: 82.2579
  actualwaitTime: 0
  maxwaitTime: 20.6836
  prefCar: 1
  initdiffcarTo1: 0
  toleranceThreshold: 0.4473
  dropOut: 1
dDrive: 0
```

Figure 33 Appendix 4: A Fully Set Vehicle

```
Car with properties:

  id: 1
  maxCharge: 15.9901
  currCharge: 1.0381
  arcage: 0
  lastDrive: 0
  carType: 1
  nominalConsump: 0.1290
```

Figure 34 Appendix 5: A Fully Set Station

```
Station with properties:

  id: 1
carIds: [8x1 double]
cars: {3x1 cell}
coord: [2x1 double]
capacity: 3
nChargers: 5
eChargers: 0
```
Figure 35 Appendix 6: Elastic price elasticity. Source: www.bized.co.uk
Simulator.m

1 clear;
2 close all;
3 addpath(genpath('./'))
4 clc;
5
6 nStat = 22;
7 statCoords = zeros(2,nStat);
8 defaultstatCoords = [40 9; 50 376; 468 592; 365 472; 576 504; 361 129; 733 752; 681 572;
1247 703; 458 712; 748 445; 898 562; 804 486; 771 357; 843 314; 788 289; 835 152;
1001 459; 884 449; 1001 459; 1031 848; 893 694].';
9 statIncMatrix = ones(nStat, nStat)-eye(nStat);
10
11 if length(statCoords) == length(defaultstatCoords)
12 statCoords = defaultstatCoords;
13 elseif length(statCoords) < length(defaultstatCoords)
14 statDiff = length(defaultstatCoords) - length(statCoords);
15 statCoords = defaultstatCoords(1:2,1:(length(defaultstatCoords)-statDiff));
16 else
17 statDiff = length(statCoords)- length(defaultstatCoords);
18 for count = 1 : statDiff
19
20 hf5 = figure(5);
21 set(0,'CurrentFigure',hf5)
22 [B, map] = imread('Hannoverosm.png');
23 C = imresize(B, 0.975);
24 imshow(C, map)
25 hold on
26 for gg = 1:length(defaultstatCoords);
27 plot(defaultstatCoords(1,:), defaultstatCoords(2,:),...'
28LineStyle', 'none', 'Marker', 'o', 'MarkerFaceColor', 'g');
29 end
30 dcmObject = datacursormode;
31 pause
32 datacursormode off
33 cursor = getCursorInfo(dcmObject);
34 newStat = [cursor.Position(1), cursor.Position(2)].';
35 defaultstatCoords(:,end + 1) = newStat;
36 hold off
37 end
38 statCoords = defaultstatCoords;
39 end
40
41 statDist = zeros(nStat);
42 statX1X2 = cell(nStat);
43 stations = cell(nStat,1);
44 capacities = 3;
45 sumCap = sum(capacities);
46 counter = 0;
47 cars = cell(sum(capacities),1);
48 vTypes = 3;
49 avCharge = zeros(vTypes,1);
50 cCars = 30000*length(cars);
51 nCharger = 3;
52 maxnCharger = nCharger;
53 fCharger = 1;

xii
maxfCharger = fCharger;
priceperHour = 12;

for ii = 1:nStat;
    x1 = statCoords(:,ii);
    statCars = cell(capacities(ii),1);
carIds = zeros(capacities(ii),1);

    for cc = 1:capacities(ii)
        ccounter = ccounter + 1;
cars{ccounter} = Car(ccounter);
        statCars{cc} = cars{ccounter};
carIds(cc) = ccounter;

        if cc <= ceil((1/3)*capacities(ii))
            vType = 1;
            statCars{cc}.setvehicleType(vType);
            statCars{cc}.setMaxCharge();
        elseif cc <= ceil((2/3)*capacities(ii)) && cc >= floor((1/3)*capacities(ii))
            vType = 2;
            statCars{cc}.setvehicleType(vType);
            statCars{cc}.setMaxCharge();
        elseif cc > ceil((2/3)*capacities(ii))
            vType = 3;
            statCars{cc}.setvehicleType(vType);
            statCars{cc}.setMaxCharge();
        end
    end

    for jj = 1:nStat;
        x2 = statCoords(:,jj);
        d = x2-x1;
        statDist(ii,jj) = norm(d,2);
        statX1X2(ii,jj) = [x1, x2];
    end

    stations(ii) = Station(ii, x1, statCars, carIds, capacities(ii), nCharger, fCharger);
end

nCust = 1000;
sundayRental = zeros(1,1);
sunmNightRental = zeros(1,1);
customers = cell(nCust,1);
custcars = cell(1,nCust);
custids = cell(1,nCust);
custminInts = cell(2,nCust);
sumTotalDayRental = zeros(1,1);
sumTotalNightRental = zeros(1,1);
walkdistances = zeros(1,nStat);
durStay = zeros(1,nCust);

for ii = 1:nCust;
customers(ii) = Customer(ii);
customers(ii).setcustTypeDistrib(nCust);
customers(ii).setfirstDrive();
customers(ii).setplaceofLiving(nCust);
customers(ii).setmaxwalkDist();
customers(ii).setmaxdurStay();
customers(ii).setprefCar()
customers(ii).setMaxWaitTimeTolerance();
custminInts(1, ii) = customers(ii).minIntervalday;
custminInts(2, ii) = customers(ii).minIntervalnight;
end

% creates array for containing the distances from the customers place
% of living to all stations
plotting = 1;
hf = figure();
[B, map] = imread('Hannoverosm.png');
C = imresize(B, 0.975);
imshow(C, map)
tempEdges = statX1X2(:);
plotEdges = zeros(2,3*length(tempEdges));
for ii = 1:length(tempEdges);
    plotEdges(:,ii*3-2) = tempEdges(ii)(:,1);
    plotEdges(:,ii*3-1) = tempEdges(ii)(:,2);
    plotEdges(:,ii*3) = NaN;
end
hold on;
plot(plotEdges(1,:), plotEdges(2,:), 'Color', [0 0 0], 'Linestyle', ':');
for gg = 1:nStat;
    plot(statCoords(1,:), statCoords(2,:),...'
        'LineStyle', 'none', 'Marker', 'o', 'MarkerFaceColor', 'g');
end
posplot = plot(NaN,NaN,'LineStyle', 'none', 'Marker', 'o', 'MarkerFaceColor', 'w');
hold off;
if plotting == 1
    hold on;
    revplot = plot(NaN, NaN);
title('Revenue');
xlabel('time');
ylabel('revenue');
hf3 = figure();
hold on;
    dayPlot = plot(NaN,NaN,'Color','g');
    nightPlot = plot(NaN,NaN,'Color','k');
hold off;
end

revenue = 0;
costs = 0;
ttnight = 0;
ttday = 540;
waittime = 0;
startT = 0*24*60;
endT = 14*24*60;
time = startT:endT;
countDriving = 0;
carDeficit = 0;
carChargeDeficit = 0;
revperMin = 0;
for ii = 1:length(time)-1
tt = time(ii);
minuteOfDay = mod(tt,60*24);
hourOfDay = floor(minuteOfDay/60);
fprintf('Time : %f years, %i months, %i days, %i hours
', floor(tt/365/24/60), floor(tt/730/60), floor(tt/24/60), hourOfDay)

if floor((tt-1)/730/60) < floor(tt/730/60) % 730 hours per month
    revenue = revenue + nCust*0;
end

deltaT = time(ii+1)-tt;

if hourOfDay > 20 || hourOfDay < 9
    ttnight = ttnight + 1;
else
    ttday = ttday + 1;
end

for cc = 1:nCust
    customers(cc).setminInt(hourOfDay, priceperHour);
customInts(1, cc) = customers(cc).minIntervalday;
customInts(2, cc) = customers(cc).minIntervalnight;
if ~ customers(cc).isDriving
    if hourOfDay > 20 || hourOfDay < 9
        hasNotDriven = customers(cc).minIntervalnight < ttnight - customers(cc).lastNightDrive;
    else
        hasNotDriven = customers(cc).minIntervalday < ttday - customers(cc).lastDayDrive
    end

    if hasNotDriven
        for hh = 1:nStat
            walkdistances(1, hh) = norm(customers(cc).pLiving - stations(hh).coords); % computes distances between cust. and all stations
        end

        minwalk = min(walkdistances);

        if minwalk/85 < customers(cc).maxdistKM % if the distance to the nearest station is lower than the max walk distance, the customer found his station
            [fS] = find(walkdistances==minwalk);
            tS = randi(nStat);
            fromStat = stations(fS);
            toStat = stations(tS);
            if fromStat.id ~= toStat.id;
                successDriving = customers(cc).letDrive(fromStat, toStat, statDist(fS,tS));
            if successDriving == -1
                customers(cc).actualwaitTime = customers(cc).actualwaitTime + 1;
            if hourOfDay > 20 || hourOfDay < 9
                daydriving = 0;
            else
                daydriving = 1;
            end
            carDeficit = carDeficit + 1;
        if customers(cc).actualwaitTime > customers(cc).maxwaitTime
            xv
```
customers{cc}.setLastDrive(tt, daydriving);
customers{cc}.dDrive = 0;
sumCarDeficit = sumCarDeficit + 1;
end

elseif successDriving == 0

customers{cc}.actualwaitTime = customers{cc}.actualwaitTime + 1;
if hourOfDay > 20 || hourOfDay < 9
daydriving = 0;
else
daydriving = 1;
end
carChargeDeficit = carChargeDeficit + 1;
if customers{cc}.actualwaitTime > customers{cc}.maxwaitTime % if the
customer waited longer than his max wait time, he loses his intention
of driving
customers{cc}.setLastDrive(tt, daydriving);
customers{cc}.dDrive = 0;
sumChargeDeficit = sumChargeDeficit + 1;
end

elseif successDriving == 1
if hourOfDay > 20 || hourOfDay < 9
daydriving = 0;
else
daydriving = 1;
end
if fromStat.nCharger < maxnCharger && fromStat.fCharger == 0
fromStat.nCharger = fromStat.nCharger + 1;
elseif fromStat.fCharger < maxfCharger
fromStat.fCharger = fromStat.fCharger + 1;
end

fromStat.removeCar(customers{cc}.myCar);
countDriving = countDriving +1;
x1 = fromStat.coords;
x2 = toStat.coords;
d = x2-x1;
distKM = d/85;
v = (50+20*randn(1))/2;
drivedistancevector = v*d/norm(d,2)*deltaT;
currentpos(1:2, cc) = x1 + drivedistancevector;
customers{cc}.updatepos(currentpos(1:2, cc));
custids{cc} = sprintf('%i
%i', customers{cc}.id, customers{cc}.myCar.id, customers{cc}.custType);
custcars{cc} = customers{cc}.myCar.id;
revperMin = revperMin + priceperHour /60;
customers{cc}.myCar.drive(norm(drivedistancevector));
end
end
else % Customer drop-out the system
customers{cc}.dropOut = true;
end
else % customer does not want to drive
currentpos(1:2, cc) = NaN;

custids{1,cc} = NaN;
custcars{1,cc} = NaN;
end

else % if driving
    myCar = customers(cc).myCar;
    countDriving = countDriving +1;
    x1 = customers(cc).fromStation;
    x2 = customers(cc).toStation;
    fprintf('Customer %i: Driving
', cc);
    d = x2-x1;

    % Case when reaching the station
    if ~customers(cc).ret && norm(currentpos(:, cc)-x1) > max(rand(1)*norm(d),0.9*norm(d))
        currentpos(1:2, cc) = currentpos(1:2, cc);
        customers(cc).updatepos(currentpos(1:2, cc));
        revperMin = revperMin + (priceperHour/60)*0.7;
        durStay(cc) = durStay(cc) + 1;
        if durStay(cc) >= rand(1)*customers(cc).maxdurofStay % customer finished his tasks
            customers(cc).returnDrive(currentpos(:, cc));
            v = (50+20*randn(1))/2;
            drivenistancevector = v*d/norm(d,2)*deltaT;
            currentpos(1:2, cc) = currentpos(1:2, cc) + drivenistancevector;
            customers(cc).updatepos(currentpos(1:2, cc));
            revperMin = revperMin + (priceperHour/60);
            customers(cc).myCar.drive(norm(drivedistancevector));
        end

    % customer returns from his initial destination
    elseif customers(cc).ret && ~(norm(currentpos(:, cc)-x1)) > norm(d)
        v = (50+20*randn(1))/2;
        drivenistancevector = v*d/norm(d,2)*deltaT;
        currentpos(1:2, cc) = currentpos(1:2, cc) + drivenistancevector;
        customers(cc).updatepos(currentpos(1:2, cc));
        revperMin = revperMin + 12/60;
        customers(cc).myCar.drive(norm(drivedistancevector));

    % customer reaches starting station
    elseif customers(cc).ret && ~norm(currentpos(:, cc)-x1) > norm(d)
        customers(cc).setlastdrive(tt, daydriving);

    % reduces number of chargers when car is parked and charged
    if fromStat.fCharger > 0
        fromStat.fCharger = fromStat.fCharger - 1;
    elseif fromStat.nCharger > 0
        fromStat.nCharger = fromStat.nCharger - 1;
    end
    fromStat.addCar(myCar);
    customers(cc).stopDrive(sundayrentals, sumnightrentals, hourOfDay);

else % initial drive
    v = (50+20*randn(1))/2;
    drivenistancevector = v*d/norm(d,2)*deltaT;
    currentpos(1:2, cc) = currentpos(1:2, cc) + drivenistancevector;
    customers(cc).updatepos(currentpos(1:2, cc));
    revperMin = revperMin + priceperHour/60;

xvii
% revenue = rate * time * 0.001
% fprintf('%d', revenue);
customers{cc}.myCar.drive(norm(driveldistancevector));
end
end
end
deficit of customers{cc}.sumRentals;
end
end
for hh = 1:sumCap
cars(hh).detCharge();
vCharges(hh,tt+1) = cars(hh).currcharge;
end
end

% charging + plotting
for gg = 1:nStat
stations(gg).chargeCars(deltaT, f Charger, n Charger);
end

set(0,'CurrentFigure',hf)
hold on;
if plotting == 1
% show availability of cars
if ~isempty(stations(gg).cars) == 1;
plot(stations(gg).coords(1,:), stations(gg).coords(2,:),...'
LineStyle', 'none', 'Marker', 'o', 'MarkerFaceColor', 'g');
else
plot(stations(gg).coords(1,:), stations(gg).coords(2,:),...'
LineStyle', 'none', 'Marker', 'o', 'MarkerFaceColor', 'r');
end
end
hold off;

for hh = 1:nCust
sumtotaldayrentals(1, cc) = sum(customers{cc}.sumdayrentals); % saves the sum of
each customers rentals
totalrentals = sum(sumtotaldayrentals); % sums up the sum of all customers rentals (daytime)
sumtotalnightrentals(1, cc) = sum(customers{cc}.sumnightrentals);
totalnightrentals = sum(sumtotalnightrentals);
if customers{cc}.dropOut == true;
dropouts = dropouts + 1;
end
if customers{cc}.dDrive
waiting = waiting + 1;
end
sumWaitTime = sumWaitTime + customers{cc}.actualWaitTime;
end
else
avgWaitTime(tt+1) = 0;
end

% hourly cost
maintenanceCost = sum(capacities) * 10.73/24/60;
administrationOverhead = 10000/30/24/60;
marketingOverhead = 10000/30/24/60;
stationRent = sum(capacities) * 30/30/24/60;
nEmployees = 30;
labourCost = nEmployee/60000/365/24/60;
sumCost = cCars + maintenanceCost + administrationOverhead + marketingOverhead + 
stationRent + laborCost;
costs = costs + sumCost;
if plotting == 1
    hold on;
    try
        cIdx = ~isnanCell(custids(1,:));
catch ex
        disp(ex);
    end
    set(posplot,'xdata',currentpos(1,:),'ydata',currentpos(2,:));
    if any(cIdx)
        if exist('cTextPlot') > 0
            delete(cTextPlot);
        end
        cTextPlot = text(currentpos(1,cIdx)+10, currentpos(2,cIdx)+10, custids(cIdx),'
                      FontWeight', 'bold', 'BackgroundColor', [1 1 1]);
    else
        cTextPlot = text();
    end
    hold off;
end
if plotting == 1
    set(0,'CurrentFigure',hf)
    hold on;
    set(profPlot,'xdata', [get(profPlot,'xdata') tt], 'ydata', [get(profPlot,'ydata')
                      revenue-costs]);
    hold off;
end
if plotting == 1
    set(0,'CurrentFigure',hf2)
    hold on;
    set(nightPlot,'xdata', [get(nightPlot,'xdata') tt], 'ydata', [get(nightPlot,'ydata')
                      countDriving]);
    if hourOfDay > 20 || hourOfDay < 9 % night
        set(nightPlot,'xdata', [get(nightPlot,'xdata') tt NaN], 'ydata', [get(nightPlot,'ydata')
                      countDriving NaN]);
    else
        set(nightPlot,'xdata', [get(nightPlot,'xdata') NaN], 'ydata', [get(nightPlot,'ydata')
                      NaN]);
    end
else % day
    if floor(mod(tt-1,24/60)/60) > 20 || floor(mod(tt-1,24/60)/60) < 9 % if last was night
        set(nightPlot,'xdata', [get(nightPlot,'xdata') tt NaN], 'ydata', [get(nightPlot,'ydata')
                      countDriving NaN]);
    else
        set(nightPlot,'xdata', [get(nightPlot,'xdata') NaN], 'ydata', [get(nightPlot,'ydata')
                      NaN]);
    end
    if floor(mod(tt-1,24/60)/60) > 20 || floor(mod(tt-1,24/60)/60) < 9 % if last was night
        set(nightPlot,'xdata', [get(nightPlot,'xdata') tt NaN], 'ydata', [get(nightPlot,'ydata')
                      countDriving NaN]);
    else
        set(nightPlot,'xdata', [get(nightPlot,'xdata') NaN], 'ydata', [get(nightPlot,'ydata')
                      NaN]);
    end
end
set(dayPlot,'xdata', [get(dayPlot,'xdata') tt], 'ydata', [get(dayPlot,'ydata') countDriving]);

end
drawnow;
end
hold off;
end
fprintf('Number of driving customers: %i
', countDriving);
fprintf('Number of requests: %i
', wishToDrive);
fprintf('Number of waiting customers: %i
', waiting);
fprintf('Number of customers who didn’t find a car: %i
', noSuccess);
fprintf('Number of customers turned away: %i due to missing cars, %i due to too little charge
', carDeficit, carChargeDeficit);
fprintf('Hourly revenue %f, hourly cost: %f
', revperMin, sumHourlyCost);
fprintf('Total revenue: %f, total cost: %f
', revenue, costs);
fprintf('Sum of day time rentals: %f
', totalrentals);
fprintf('Sum of night time rentals: %f
', totalnightrentals);
revenuePlot(tt+1) = revenue - costs;

avgChargePerTime = sum(vCharges)/size(vCharges,1);
figure();
plot(time(1:length(time)-1),avgChargePerTime);
figure();
plot(time(1:end-1),profitPlot);
figure();
plot(time(1:end-1),avgWaitTime);

Listing 53 Simulator.m
Customer.m

clear;
close all;
addpath(genpath(’./’))
clc;

nStat = 22;
statCoords = zeros(2,nStat);
defaultstatCoords = [40 9; 50 376; 468 592; 365 472; 576 504; 361 129; 733 752; 681 572;
1247 703; 458 445; 898 562; 804 486; 771 357; 843 314; 788 289; 835 152;
1001 459; 884 449; 1001 459; 1031 848; 893 694]. ’;
statIncMatrix = ones(nStat, nStat) - eye(nStat);

if length(statCoords) == length(defaultstatCoords)
statCoords = defaultstatCoords;
elseif length(statCoords) < length(defaultstatCoords)
statDiff = length(defaultstatCoords) - length(statCoords);
statCoords = defaultstatCoords(1:2,1:(length(defaultstatCoords)-statDiff));
else
statDiff = length(statCoords) - length(defaultstatCoords);
for count = 1 : statDiff
hf5 = figure(5);
set(0,’CurrentFigure’,hf5)
[B, map] = imread(’Hannoverosm.png’);
C = imresize(B, 0.975);
imshow(C, map)
hold on
for gg = 1:length(defaultstatCoords);
plot(defaultstatCoords(1,:), defaultstatCoords(2,:), ’LineStyle’, ’none’, ’Marker’, ’o’, ’MarkerFaceColor’, ’g’);
end
dcmObject = datacursormode;
pause
datacursormode off
cursor = getCursorInfo(dcmObject);
newStat = [cursor.Position(1), cursor.Position(2)]. ’;
defaultstatCoords(:,end + 1) = newStat;
hold off
end
statCoords = defaultstatCoords;
end

statDist = zeros(nStat);
statX1X2 = cell(nStat);
stations = cell(nStat,1);
capacities = 3;
sumCap = sum(capacities);
ccounter = 0;
cars = cell(sum(capacities),1);
vTypes = 3;
avCharge = zeros(vTypes,1);
cCars = 30000*length(cars);
nCharger = 3;
maxnCharger = nCharger;
fCharger = 1;
maxfCharger = fCharger;
priceperHour = 12;

for ii = 1:nStat;
    x1 = statCoords(:,ii);
    statCars = cell(capacities(ii),1);
    carIds = zeros(capacities(ii),1);
    for cc = 1:capacities(ii)
        ccounter = ccounter + 1;
        cars{cc} = Car(cccounter);
        statCars{cc} = cars{cccounter};
        carIds(cc) = ccounter;
        if cc <= ceil((1/3)*capacities(ii))
            vType = 1;
            statCars{cc}.setvehicleType(vType);
            statCars{cc}.setmaxCharge();
        elseif cc <= ceil((2/3)*capacities(ii)) && cc >= floor((1/3)*capacities(ii))
            vType = 2;
            statCars{cc}.setvehicleType(vType);
            statCars{cc}.setmaxCharge();
        elseif cc > ceil((2/3)*capacities(ii))
            vType = 3;
            statCars{cc}.setvehicleType(vType);
            statCars{cc}.setmaxCharge();
        end
    end

    x2 = statCoords(:,jj);
    d = x2-x1;
    statDist(ii,jj) = norm(d,2);
    statX1X2{ii,jj} = [x1, x2];
end

stations(ii) = Station(ii, x1, statCars, carIds, capacities(ii), nCharger, fCharger);
end

nCust = 1000;
sundayrentals = zeros(1,1);
sumnighrentals = zeros(1,1);
customers = cell(nCust,1);
custcars = cell(1,nCust);
custids = cell(1,nCust);
custminInts = cell(2,nCust);
sumtotaldayrentals = zeros(1,nCust);
sumtotalnighrentals = zeros(1,nCust);
walkdistances = zeros(1,nStat);
durStay = zeros(1,nCust);

for ii = 1:nCust;
    customers(ii) = Customer(ii);
    customers(ii).setcustTypeDistrib(nCust);
    customers(ii).setfirstDrive();
    customers(ii).setplaceofLiving(nCust);
    customers(ii).setmaxwalkDist();
    customers(ii).setmaxdurStay();
    customers(ii).setprefCar()
customers{ii}.setMaxWaitTimeTolerance();
custminInts(1, ii) = customers{ii}.minIntervalday;
custminInts(2, ii) = customers{ii}.minIntervalnight;
end
%
creates array for containing the distances from the customers place % of living to all stations
plotting = 1;
hf = figure();
[B, map] = imread('Hannoverosm.png');
C = imresize(B, 0.975);
imshow(C, map)
tempEdges = statX1X2(:,);
plotEdges = zeros(2,3*length(tempEdges));
for ii = 1:length(tempEdges);
    plotEdges(:,ii*3-2) = tempEdges{ii}(:,1);
    plotEdges(:,ii*3-1) = tempEdges{ii}(:,2);
    plotEdges(:,ii*3) = NaN;
end
hold on;
plot(plotEdges(1,:), plotEdges(2,:), 'Color', [0 0 0], 'Linestyle', ':');
for gg = 1:nStat;
    plot(statCoords(1,:), statCoords(2,:), ... 'LineStyle', 'none', 'Marker', 'o', 'MarkerFaceColor', 'g');
end
posplot = plot(NaN,NaN,'LineStyle', 'none', 'Marker', 'o', 'MarkerFaceColor', 'w');
hold off;
if plotting == 1
    hold on;
    revplot = plot(NaN, NaN);
title('Revenue');
xlabel('time');
ylabel('revenue');
hf3 = figure();
hold on;
dayPlot = plot(NaN,NaN,'Color','g');
nightPlot = plot(NaN,NaN,'Color','k');
hold off;
end
revenue = 0;
costs = 0;
ttnight = 0;
ttday = 540;
waittime = 0;
startT = 0*24*60;
endT = 14*24*60;
time = startT:endT;
countDriving = 0;
carDeficit = 0;
carChargeDeficit = 0;
revperMin = 0;
for ii = 1:length(time)-1
tt = time(ii);
minuteOfDay = mod(tt,60*24);
hourOfDay = floor(minuteOfDay/60);
fprintf('Time: %f years, %i months, %i days, %i hours\n', floor(tt/365/24/60), floor(tt/730/60), floor(tt/24/60), hourOfDay)

if floor((tt-1)/730/60) < floor(tt/730/60) % 730 hours per month
    revenue = revenue + nCust*0;
end
deltaT = time(ii+1)-tt;

if hourOfDay > 20 || hourOfDay < 9
    ttnight = ttnight + 1;
else
    ttday = ttday + 1;
end

for cc = 1:nCust
    customers(cc).setminInt(hourOfDay, priceperHour);
custminInts(1, cc) = customers(cc).minIntervalDay;
custminInts(2, cc) = customers(cc).minIntervalNight;
    if ~ customers(cc).isDriving
        if hourOfDay > 20 || hourOfDay < 9
            hasNotDriven = customers(cc).minIntervalNight < ttnight - customers(cc).lastNightDrive;
        else
            hasNotDriven = customers(cc).minIntervalDay < ttday - customers(cc).lastDayDrive
        end
    if hasNotDriven
        for hh = 1:nStat
            walkdistances(1, hh) = norm(customers(cc).pLiving - stations(hh).coords); % computes distances between cust. and all stations
        end
    minwalk = min(walkdistances);
    if minwalk/85 < customers(cc).maxdistKM % if the distance to the nearest station is lower than the max walk distance, the customer found his station
        [fS] = find(walkdistances==minwalk);
tS = randi(nStat);
        fromStat = stations(fS);
toStat = stations(tS);
        if fromStat.id ~= toStat.id;
            successDriving = customers(cc).letDrive(fromStat, toStat, statDist(fS,tS));
            if successDriving == -1
                customers(cc).actualwaitTime = customers(cc).actualwaitTime + 1;
        if hourOfDay > 20 || hourOfDay < 9
            daydriving = 0;
        else
            daydriving = 1;
        end
        carDeficit = carDeficit + 1;
        if customers(cc).actualwaitTime > customers(cc).maxwaitTime
customers{cc}.setlastdrive(tt,daydriving);
customers{cc}.dDrive = 0;
sumCarDeficit = sumCarDeficit + 1;
end

elseif successDriving == 0
    customers{cc}.actualwaitTime = customers{cc}.actualwaitTime + 1;
    if hourOfDay > 20 || hourOfDay < 9
daydriving = 0;
    else
daydriving = 1;
    end
carChargeDeficit = carChargeDeficit + 1;
    if customers{cc}.actualwaitTime > customers{cc}.maxwaitTime % if the customer waited longer than his max wait time, he loses his intention of driving
        customers{cc}.setlastdrive(tt,daydriving);
customers{cc}.dDrive = 0;
        sumChargeDeficit = sumChargeDeficit + 1;
    end
end

elseif successDriving == 1
    if hourOfDay > 20 || hourOfDay < 9
daydriving = 0;
    else
daydriving = 1;
    end
    if fromStat.nCharger < maxnCharger && fromStat.fCharger == 0
        fromStat.nCharger = fromStat.nCharger + 1;
    elseif fromStat.fCharger < maxfCharger
        fromStat.fCharger = fromStat.fCharger + 1;
    end
    fromStat.removeCar(customers{cc}.myCar);
countDriving = countDriving +1;
x1 = fromStat.coords;
x2 = toStat.coords;
d = x2-x1;
distKM = d/85;
v = (50+20*randn(1))/2;
drivedistancevector = v*d/norm(d,2)*deltaT;
currentpos(1:2, cc) = x1 + drivedistancevector;
customers{cc}.updatepos(currentpos(1:2, cc));
custids{cc} = sprintf('%i\n%i',customers{cc}.id,customers{cc}.myCar.id,
customers{cc}.custType);
custcars{cc} = customers{cc}.myCar.id;
revperMin = revperMin + priceperHour / 60;
customers{cc}.myCar.drive(norm(drivedistancevector));
end

end% Customer drop-out the system
customers{cc}.dropOut = true;
else % customer does not want to drive
    currentpos(1:2, cc) = NaN;
XXV
custids{1,cc} = NaN;
custcars{1,cc} = NaN;
end

else % if driving
    myCar = customers(cc).myCar;
    countDriving = countDriving +1;
    x1 = customers(cc).fromStation;
    x2 = customers(cc).toStation;
    fprintf('Customer %i: Driving\n', cc);
    d = x2-x1;

    % Case when reaching the station
    if ~customers(cc).ret && norm(currentpos(:, cc)-x1) > max(rand(1)*norm(d),0.9*norm(d))
        currentpos(1:2, cc) = currentpos(1:2, cc);
        customers(cc).updatepos(currentpos(1:2, cc));
        revperMin = revperMin + (priceperHour/60)*0.7;
        durStay(cc) = durStay(cc) + 1;
        if durStay(cc) >= rand(1)*customers(cc).maxdurofStay % customer finished his tasks
            customers(cc).returnDrive(currentpos(:, cc));
            v = (50+20*randn(1))/2;
            drivedistancevector = v*d/norm(d,2)*deltaT;
            currentpos(1:2, cc) = currentpos(1:2, cc) + drivedistancevector;
            customers(cc).updatepos(currentpos(1:2, cc));
            revperMin = revperMin + priceperHour/60;
            customers(cc).myCar.drive(norm(drivedistancevector));
        end
    end

    % customer returns from his initial destination
    elseif customers(cc).ret && ~norm(currentpos(:, cc)-x1) > norm(d)
        v = (50+20*randn(1))/2;
        drivedistancevector = v*d/norm(d,2)*deltaT;
        currentpos(1:2, cc) = currentpos(1:2, cc) + drivedistancevector;
        customers(cc).updatepos(currentpos(1:2, cc));
        revperMin = revperMin + 12/60;
        customers(cc).myCar.drive(norm(drivedistancevector));
    end

    % customer reaches starting station
    elseif customers(cc).ret && norm(currentpos(:, cc)-x1) > norm(d)
        customers(cc).setlastdrive(tt, daydriving);
    end

    % reduces number of chargers when car is parked and charged
    if fromStat.fCharger > 0
        fromStat.fCharger = fromStat.fCharger - 1;
    elseif fromStat.nCharger > 0
        fromStat.nCharger = fromStat.nCharger - 1;
    end
    fromStat.addCar(myCar);
    customers(cc).stopDrive(sundayrentals, sumnighrentals, hourOfDay);
end

else % initial drive
    v = (50+20*randn(1))/2;
    drivedistancevector = v*d/norm(d,2)*deltaT;
    currentpos(1:2, cc) = currentpos(1:2, cc) + drivedistancevector;
    customers(cc).updatepos(currentpos(1:2, cc));
    revperMin = revperMin + priceperHour/60;

%rev(tt) = rTime * 0.001;
fprintf('%d', revenue);
customers(cc).myCar.drive(norm(drivedistancevector));
end
end

for hh = 1:sumCap
cars(hh).detCharge();
vCharges(hh,tt+1) = cars(hh).currcharge;
end

% charging + plotting
for gg = 1:nStat
stations(gg).chargeCars(deltaT, fCharger, nCharger);
set(0,'CurrentFigure',hf)
hold on;
if plotting == 1
%show availability of cars
if isempty(stations(gg).cars) == 1;
plot(stations(gg).coords(1,:), stations(gg).coords(2,:),...
'LineStyle', 'none', 'Marker', 'o', 'MarkerFaceColor', 'g');
else
plot(stations(gg).coords(1,:), stations(gg).coords(2,:),...
'LineStyle', 'none', 'Marker', 'o', 'MarkerFaceColor', 'r');
end
end
hold off;

for hh = 1:nCust
sumtotaldayrentals(1, cc) = sum(customers(cc).sumdayrentals); % saves the sum of each customers rentals
totalrentals = sum(sumtotaldayrentals); % sums up the sum of all customers rentals (daytime)
sumtotalnightrentals(1, cc) = sum(customers(cc).sumnightrentals);
totalnightrentals = sum(sumtotalnightrentals);
if customers(cc).dropOut == true;
dropOuts = dropOuts + 1;
end
if customers(cc).dDrive
waiting = waiting + 1;
end
sumWaitTime = sumWaitTime + customers(cc).actualwaitTime;
end

if waiting >= 1
avgWaitTime(tt+1) = (sumWaitTime/waiting);
else
avgWaitTime(tt+1) = 0;
end

% hourly cost
maintenanceCost = sum(capacities) * 10.73/24/60;
administrationOverhead = 10000/30/24/60;
marketingOverhead = 10000/30/24/60;
stationRent = sum(capacities) * 30/30/24/60;
nEmployees = 30;
labourCost = nEmployee/60000/365/24/60;
405  sumCost = cCars + maintenanceCost + administrationOverhead + marketingOverhead +
406  stationRent + laborCost;
407  costs = costs + sumCost;
408  if plotting == 1
409     hold on;
410     try
411         cIdx = ~ isnanCell(custids(1,:));
412         catch ex
413             disp(ex);
414         end
415         set(posplot,'xdata',currentpos(1,:),'ydata',currentpos(2,:));
416         if any(cIdx)
417             if exist('cTextPlot') > 0
418                 delete(cTextPlot);
419             end
420             cTextPlot = text(currentpos(1,cIdx)+10, currentpos(2,cIdx)+10, custids(cIdx),'
421                 FontWeight', 'bold', 'BackgroundColor', [1 1 1]);
422         else
423             cTextPlot = text();
424         end
425     end
426  end
427  if plotting == 1
428     set(0,'CurrentFigure',hf)
429     hold on;
430     try
431         cIdx = ~ isnanCell(custids(1,:));
432         catch ex
433             disp(ex);
434         end
435         set(posplot,'xdata',currentpos(1,:),'ydata',currentpos(2,:));
436         if any(cIdx)
437             if exist('cTextPlot') > 0
438                 delete(cTextPlot);
439             end
440             cTextPlot = text(currentpos(1,cIdx)+10, currentpos(2,cIdx)+10, custids(cIdx),'
441                 FontWeight', 'bold', 'BackgroundColor', [1 1 1]);
442         else
443             cTextPlot = text();
444         end
445     end
446  end
447  if plotting == 1
448     set(0,'CurrentFigure',hf2)
449     hold on;
450     set(profPlot,'xdata', [get(profPlot,'xdata') tt], 'ydata', [get(profPlot,'ydata')
451                 revenue-costs]);
452     hold off;
453  end
454  if plotting == 1
455     set(0,'CurrentFigure',hf3)
456     hold on;
457     if mod(tt, 10) == 0
458         if hourOfDay > 20 || hourOfDay < 9 % night
459             set(nightPlot,'xdata', [get(nightPlot,'xdata') tt], 'ydata', [get(nightPlot,'ydata')
460                 countDriving]);
461             if "(floor(mod(tt-1,24/60)/60) > 20 || floor(mod(tt-1,24/60)/60) < 9) % if last
462                 was day
463                 set(dayPlot,'xdata', [get(dayPlot,'xdata') tt NaN], 'ydata', [get(dayPlot,'ydata')
464                     countDriving NaN]);
465             else
466                 set(dayPlot,'xdata', [get(dayPlot,'xdata') NaN], 'ydata', [get(dayPlot,'ydata')
467                     NaN]);
468             end
469         else % day
470             if floor(mod(tt-1,24/60)/60) > 20 || floor(mod(tt-1,24/60)/60) < 9 % if last was
471                 night
472                 set(nightPlot,'xdata', [get(nightPlot,'xdata') tt NaN], 'ydata', [get(nightPlot,'ydata')
473                     countDriving NaN]);
474             else
475                 set(nightPlot,'xdata', [get(nightPlot,'xdata') NaN], 'ydata', [get(nightPlot,'ydata')
476                     NaN]);
477             end
478         end
479     end
480
Listing 54 Customer.m
classdef Car < handle

% The car class solely sets car-related properties and manages the
% battery management

properties
    id;
    maxcharge;  % maximum charge
    currcharge;  % current charge
    state;  % 0 = parked, 1 = drives
    carType;
    nominalConsump;
end

methods
    function obj = Car(id)
        obj.id = id;
        obj.state = 0;  % 0: parked, 1: driving, 2: charging, 3: damaged
    end

    function setvehicleType(obj, vType)
        if vType == 1
            obj.carType = 1;
        elseif vType == 2
            obj.carType = 2;
        elseif vType == 3
            obj.carType = 3;
        end
    end

    function setmaxCharge(obj)
        if obj.carType == 1
            obj.currcharge = 16;  % 16: 100%, 0: 0% -> Mitsubishi I-Miev
            obj.maxcharge = 16;
            obj.nominalConsump = 12.9/100;
        elseif obj.carType == 2
            obj.currcharge = 23;  % Station Wagon
            obj.maxcharge = 23;
            obj.nominalConsump = 13.3/100;
        elseif obj.carType == 3
            obj.currcharge = 25;
            obj.maxcharge = 25;
            obj.nominalConsump = 13.5/100;  % BMW-I3
        end
    end

    function detCharge(obj)
        if obj.maxcharge > 0
            obj.maxcharge = obj.maxcharge - 0.0000148;
        end
    end

    function obj = drive(obj, dist)
        obj.state = 1;
end

% Deterioration of maximum charges over time -> 100000 KM = - 15% capacities

% vary the mileage of the different vehicles

XXX
distKM = (dist/85)*1.39;
obj.currcharge = obj.currcharge - obj.nominalConsump * distKM;
end
everify
end
everify

Listing 55 Vehicle.m

Station.m

classdef Station < handle

    properties
        id;
        carIds;
        cars;
        coords;
        capacity;
        nCharger;
        fCharger;
    end

    methods

        function obj = Station(id, coords, cars, carIds, capacity, nCharger, fCharger)
            obj.cars = cars;
            obj.id = id;
            obj.coords = coords;
            obj.carIds = carIds;
            obj.capacity = capacity;
            obj.fCharger = fCharger;
            obj.nCharger = nCharger;
        end

        function obj = removeCar(obj, car)
            remainingCars = ~logical(obj.carIds == car);
            obj.cars = obj.cars(remainingCars);
            obj.carIds = obj.carIds(remainingCars);
        end

        function obj = addCar(obj, car)
            obj.cars(end+1) = car;
            obj.carIds(end+1) = car.id;
        end

        % removes fast charger when available (see main)
        function obj = removenfCharger(obj, fromStation)
            obj.fromStation = fromStation;
            obj.fromStation.fCharger = fromStation.fCharger - 1;
        end

        % removes normal charge unit
        function obj = removennCharger(obj, fromStation)
            obj.fromStation = fromStation;
            obj.fromStation.nCharger = fromStation.nCharger - 1;
    end

    % xxi
function chargeCars(obj, tt)

    for kk = 1:length(obj.cars);
        for count = 1:tt
            if obj.cars(kk).currcharge < obj.cars(kk).maxcharge && obj.fCharger > 0 && obj.cars(kk).state == 0;
                obj.cars(kk).currcharge = obj.cars(kk).currcharge + 0.026*obj.cars(kk).maxcharge; % a fast charger needs 30 min to recharge 80% of a 16 Kwh battery --> 2.6% per minute
            elseif obj.cars(kk).currcharge < obj.cars(kk).maxcharge && obj.nCharger > 0 && obj.cars(kk).state == 0;
                obj.cars(kk).currcharge = obj.cars(kk).currcharge + 0.002083*obj.cars(kk).maxcharge; % a fast charger requires 6-8 hours to recharge 100% of the batteries of a 16 Kwh battery --> 0.2% per minute
            else
                end
            end
        end
    end
end

Listing 56 Station.m
Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht sind und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

Hannover, den