Diploma Thesis

Driving-Range Estimation of Electric Vehicles Using Graph Algorithms

by

Felix Gstrein
Matr. Nr.: 1026840

carried out for the purpose of obtaining the degree Master of Science
under supervision of

Assistant Prof. Dipl.-Inform. Dr.rer.nat. Martin Nöllenburg
Dominik Dvorak, MSc

E 186 - Institute of Computer Graphics and Algorithms

submitted at

Technische Universität Wien
Getreidemarkt 9/164-CT
1060 Wien

Vienna, April 23, 2017

This work was supported by the AIT Austrian Institute of Technology GmbH.

Affidavit:
I declare in lieu of oath, that I wrote this thesis and performed the associated research myself, using only literature cited in this volume.

Date *Signature*

Acknowledgements

It is not easy to find a diploma thesis whose results can be useful for further studies and whose topic is future-orientated and meets personal interests all at the same time. The dedicated team of the Center for Low-Emission Transport at the Austrian Institute of Technology and the Technical University of Vienna made it possible for me to work on such a topic.

At this point I would like to thank Assistant Prof. Dipl.-Inform. Dr.rer.nat. Martin Nöllenburg who paved the way for me to write this thesis, supplementing the work with his great knowledge in the field of graph algorithms.

My sincere thanks is owed to Dominik Dvorak, MSc, who was a great supervisor and workmate keeping track of the thesis and always knew what to do when problems arose. Both of them have sought to leave plenty of freedom for me to deal with the issues at all times. Appreciation is extended to all members of the Center for Low-Emission Transport for the pleasant working atmosphere and the willingness to help in the respective specialist fields. Particularly I would like to thank Dipl.-Ing Hansjörg Kapeller for his support in electrical engineering and Mag. Ulf Reisenbichler for an introduction into GUI-programming using C# and general IT services. I also thank DI(FH) Thomas Bäuml and DI Dr. Dragan Simic, MBA, for proofreading this work.

Deep gratitude goes to my family, especially to my parents Wolfgang and Elisabeth Gstrein and my sister Romana Gstrein who supported me with wise counsel and are a strong backing in my life. I am grateful to Stefanie Tomasch not only for her valuable help in engineering subjects but also for providing a retreat together with her family, such that i could take some time out whenever i needed to.

Abstract

Abstract

The goal of this study was to evaluate the energy consumption of battery electric vehicles in order to create a tool for estimating the maximum driving range. In the first part, a general model of a battery electric vehicle was implemented as C++ code using the parameters of the Mitsubishi i-MiEV for validation. Functions for the propulsion system, the battery, the facilities for driving assistance and comfort and the behaviour of the driver were established, describing the most important influences on the energy consumption, and hence the range of the vehicle. The impacts of two different battery models on the energy consumption were examined to justify the application of the simpler one in the remaining part of the work.

The outcome of the consumption estimation was then used for calculating cost functions to find an energy-optimised route for the vehicle. In the second part, where a graph model was developed for routing battery electric vehicles, these cost functions were assigned to the graph. For the purpose of routing, a graph algorithm has been implemented in the third part, which was adjusted to exploit the qualities of the developed energy-consumption model.

As final outcome, the developed program issues most energy-efficient paths from given start points to any destination in the graph. This is an advantage in relation to common navigation systems computing shortest distance paths. Moreover, the investigation of the results shows how different features of vehicle and driver affect the range. Thus, the user can assess the energy consumption of his driving style and the electronic devices of the vehicle when switched on.

Kurzfassung

Das Ziel dieser Arbeit war es, den Energieverbrauch von batteriebetriebenen Elektrofahrzeugen zu evaluieren und damit ein Werkzeug zur Abschätzung der verbleibenden Reichweite zu schaffen. Im ersten Teil wurde ein allgemeines Modell für ein batteriebetriebenes Elektrofahrzeug als C++ Code implementiert. Zur Validierung wurden die Parameter eines

Mitsubishi i-MiEV ausgewertet. Es wurden Funktionen zur Beschreibung des Antriebssystems, der Batterie, der Fahrassistenz- und -komforteinrichtungen und des Fahrers erstellt, welche sich als die wichtigsten Faktoren zur Reichweitenbestimmung herauskristallisiert hatten. Die jeweiligen Einflüsse zweier verschiedener Batteriemodelle auf den Energieverbrauch wurden untersucht, um die Verwendung des einfacheren Modells im weiteren Verlauf der Arbeit zu rechtfertigen.

Im zweiten Teil der Arbeit wurde ein zur Routenberechnung für batteriebetriebene Elektrofahrzeuge geeignetes Graphenmodell aufgestellt, dem mittels der zuvor implementierten Funktionen Kostenfunktionen für die Wegabschnitte zugeteilt wurden. Um eine energieoptimierte Reiseroute zu finden, wurde im dritten Teil ein Graphenalgorithmus implementiert, der dem entwickelten Energieverbrauchsmodell so angepasst wurde, dass er all dessen Ergebnisse berücksichtigen kann.

Letztendlich gibt das entworfene Programm die energieeffizientesten Routen von gegebenen Startpunkten zu jedem gewünschten Ziel, das innerhalb des betrachteten Graphen liegt, aus. Damit hat es einen Vorteil gegenüber herkömmlichen Navigationssystemen, die meist lediglich die Routen mit kürzester Distanz berechnen. Zusätzlich kann anhand der Ergebnisse der Arbeit darauf geschlossen werden, wie sich die verschiedenen Einflüsse von Fahrzeugkomponenten und Fahrer auf die Reichweite auswirken. Damit kann der Anwender den durch seinen Fahrstil und eingeschaltete elektronische Einrichtungen verursachten Energieverbrauch bewerten.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

One of the most frequent subjects of discussion in the 21^{st} century is the transition from fossil fuels to alternative energy sources. This is motivated by the climate change, the advancing environmental awareness of the human population and national governments and the effort of import countries to become independent from the main oil-producing countries. Concerning cars the usage of the Battery Electric Vehicle (BEV) has restricted popularity due to limited range and long battery-charging time, but the concept will be trendsetting once these problems can be mastered. Under these circumstances and involving the fact that there is still a lack of battery-charging infrastructure there is the demand to be able to assess which places are reachable considering the current charge level. Also assistance for economic driving integrated in cars is useful since people may try to save electricity by optimising their driving attitudes. For example, the most energy-efficient route can be chosen instead of taking the shortest way to a particular destination, or the acceleration and braking behaviour can be enhanced. This is, by the way, an important environmental aspect as well. Hence, the challenge is to predict energy consumption under real-world driving conditions and to combine a good forecasting with an algorithm that provides routes with low predicted energy consumption, based on the consumption model. With modifiable parameters such a simulation could be customised and the user might be assisted in saving energy.

In this thesis the goal was to establish a realistic model of the energy consumption for the Mitsubishi innovative Electric Vehicle, based on Mitsubishi i (i-MiEV), and to show how to combine it with a routing algorithm that is able find most energy-efficient paths exploiting consumption characteristics of the BEV. By changing the parameters accordingly the resulting tool should be able to simulate other BEVs too. The concerning program codes were written in C++. Related skills can be gained from [50]. The thesis can be structured into three parts. These are the estimation of the energy consumption for a general input of driving cycles, the development of a graph model for routing BEVs and its assignment with the cost functions using the consumption model of the first part and the implementation of a graph algorithm to do the final routing. The maximum driving range shall be indicated by the coordinates of the traffic point where the battery runs out of charge.

An endless number of phenomenons affect the driving range but only a few of them can be modelled for evaluating the energy consumption within reasonable time and effort. A further objective of this work was to filter out the most essential ones while keeping the model as realistic as possible. Finally, a sensitivity analysis of how a driver can affect the driving range is given.

1.1 Related Works

Generally speaking, there exist various types of vehicles that use electricity for propulsion, such as fuel-cell vehicles or plug-in-hybrid electric vehicles. Therefore, the term Electric Vehicle (EV) will be used in more universal issues.

Many models of BEVs and their energy consumption already exist. For this work, those are relevant that can be applied to construct cost functions for road sections. Most of the studies consider the basic driving resistances, such as air drag, rolling resistance, slope of the road and inertial resistance, to model the cost functions of a street map as can be seen in [12]. In that work, the cost functions were kept simple because of the high number of vehicles incorporated in a scenario describing BEVs moving in a road network and performing charging operations arbitrarily without deciding for a specific energy-efficient path. Some studies also include the impact of air conditioning. In [37] resistant forces and a lithium-ion battery are modelled to determine the remaining driving range. Further contributions to the energy consumption in [37] are only included using efficiency factors, so there is a lack of more exact models of rotating parts and gear transmissions for example. The driving profile in [37] is predicted using a stochastic approach instead of data from a street map and simulating a driving style and an energy-consumption behaviour of a user. The tool presented in [32] additionally incorporates rotating parts to the model of the BEV, but still lacks of a model for the driver and additional energy consumption due to facilities for driving assistance and comfort. It is tested for constant velocities and driving cycles, but no most energy-efficient path query is carried out. In [52] again only driving resistances and some power losses along the power train described by an efficiency factor are regarded. The force generated by the electric machine is established using a simplified equation for the motor torque, which is obtained from a motor characteristic in the present work. The simulation package Janus presented in [14] is able to evaluate the change in the State Of Charge (SOC) of a BEV, but does not consider as many influences as shall be included here, for example a driver model and the energy consumption of heating, air conditioning, ventilation and exterior lighting. Additionally, Janus accepts driving cycles as input and is not able to build cost functions for a graph. Also [48] only works with single driving cycles as input without considering a driver model. Both of the lastly addressed papers are outdated and still model lead-acid batteries instead of lithium-ion batteries now being state of the art. In [6] an existing model for the energy loss adopting the power-train components as “black boxes” is used to calculate a cost function in order to compare route choices of different vehicle types. In [8] a sensitivity analysis on various effects on energy demand is done without using any graph algorithm building a route. For this, slope, rolling resistance and air drag were modelled and some efficiency coefficients were implemented.

A key plank of routing BEVs is the graph model. Energy can be stored in the battery when braking the BEV. Thus, turn costs play a central role because the vehicle has to slow down when turning at a junction. Such turn costs are taken into account for example in [51], where the creation of a pseudo-dual graph is explained. The described graph was used with a modified structure in this thesis.

The goal of the present thesis was to implement a suitable routing algorithm for the realistic BEV model. A lot of research on energy-efficient routing for EVs has already been done. In [7] the challenges being posed particularly for routing BEVs are well defined. Lots of special phenomenons emerging only for such vehicles are addressed, some of the concepts noted in that work could further enhance the tool presented here. However, the main focus in the present thesis was on employing the developed energy-consumption model for the graph algorithm. Thus, the latter was customised for the routing of the designed vehicle and further special algorithms described in [7] were omitted. An algorithm called A^* search is introduced in [40], forgoing preprocessing techniques in order to be able to ask for specific parameters such as weight at query time. The graph models of the two lastly addressed papers do not consider turn costs.

There are also works that consider both energy consumption and routing of EVs. In [11] only the four resistant forces already mentioned are used to determine the energy consumption along a path and regenerative braking is not considered. That is because the used algorithm cannot predict brake events. Such events shall be regarded here, namely when a stop sign or a traffic light is expected at a junction. Another example for simple cost functions is given by [18], where a graph algorithm is developed regarding only the slope of a street and finite capacity of the battery. In [10] an existing model regarding driving resistances and transmission losses based on measurements is used. However, to make a really good approach for a cost function more than just general forces acting on an EV should be taken into account, so as to describe the attitudes of a vehicle type. As far as possible, “black boxes” should be avoided. Several factors may be overestimated and can be neglected, but others may show unexpected effects on the battery’s SOC. Environmental conditions, road and vehicle characteristics are considered in [9], but the recovery of energy when braking is not regarded. Regenerative braking is seen as one of the most remarkable features of BEVs and so its installation into the model played an important role in the present thesis.

Many existing search algorithms are listed in [16] and [41], but none of them regards the specific requirements on routing BEVs. The question of how far one is still able to get with a certain charge level is not referred to in the cited papers, except for [37]. The tool presented in this thesis shall give a feedback about that.

1.2 Contribution and Outline

Objective and Contribution

The aim of this study was to create a tool for estimating the range of a BEV for a given initial SOC and to find specific routes that minimise the energy consumption of the vehicle. Special attention was paid on the implementation of the vehicle model required to calculate the SOC considering various influences on its value over time. It was tried to keep the model of the vehicle as realistic as possible to close the gap in the state of the art. Doing so, the most significant contribution is the calculation of realistic edge weights describing the energy costs of road sections.

It was no objective to forgo preprocessing techniques or to be able to ask for specific parameters at query time, since a useful graph had to be composed during the preprocessing time. By contrast, an essential issue was the consideration of BEV-specific features like regenerative braking using turn costs. The added value to the state of the art concerning routing mechanisms is the development of an algorithm being applicable to a realistic consumption model. Graphs of large scale have not been considered in this work, the pre-existing graph used comprises the street map of Vienna. For large graphs containing several cities it would be useful to implement special techniques like introducing highway hierarchies as described in [41]. In future works such concepts could be brought in as well.

The significance of the study lies in the fact that a potential user of a BEV may have the fear of ending up somewhere on the way with a battery run out of charge. Regarding the lack of battery-charging sites, this anxiety is justified. The tool created shall address this problem and help to motivate people to employ BEVs. Moreover, answers to customer questions such as how the driver can affect the energy consumption and consequently the SOC by his driving style can be derived from the output of the tool.

Structure of the Thesis

The present thesis is basically structured in three main parts. As a first self-contained part of the work a vehicle model is implemented using driving cycles as input and issuing the SOC over the time as output. This is described in Chapter 3.

In the second part described by Chapter 4 a graph model for routing tasks on BEVs is developed. The difficulty is that the graph should be able to hold the costs for road sections and turn costs as well. Also the common graph algorithms should be able to work with the graph such that as few things as possible have to be changed on them. The functions and classes implemented in the first part are used to assign the costs to the graph.

Chapter 5 describes the third part, where a graph algorithm is implemented to do the final routing. Doing so, it uses the graph created in the second part as input and issues shortest paths and related SOC values. Some graph-theoretic preliminaries are stated in Chapter 2. The results are presented in Chapter 6 and Chapter 7 gives some concluding remarks.

1.3 Methodological Approach

The three self-contained parts were implemented as C++ code and the programs of part two and three were put together to form a tool that can find most energy-efficient paths and issue driving range and SOC of a BEV using the graph of a street map as input. The tool should work on any system including a C++ compiler. Representing an ordinary BEV the i-MiEV was selected to be the first object to provide specific parameters for the evaluation of the program. Since there is such a model at the company location used as business car, additionally depicting the attitudes of the i-MiEV has the benefit that the output can be verified easily. Measurement data of the business car was the most relevant reference to test the functionality. Other BEVs can be simulated with the tool by changing the parameters accordingly.

The input data and structures of the three parts are depicted in Figure 1.1. The blue boxes represent the classes implemented in the program code and the arrows outline the data flow. The latter describes which class uses information or functions provided by other classes. Sheet symbols signal the possibility of parameter entries by the user. The dashed arrows mark the data flow of the self-contained program for reading driving cycles and issuing the SOC described by the first part of this thesis. Thus, data is input from a class for reading the driving cycle (*DrivingCycle*), then the data is being prepared by a class providing functions for numerical mathematics (*NumericalMathematics*) and sent to a class describing the behaviour of the driver (*Driver*) which can be configured to describe the user-specific driving style. Parameters for the energy consumption of facilities for driving assistance and comfort and for the battery are accepted by a class describing the battery model (*Battery*). A file containing open-circuit voltage data is interpolated by *NumericalMathematics* before the data is used by *Battery*. Vehicle parameters can be changed in a class describing the mechanical model of the vehicle (*Propulsion*). *Battery* uses the information of *Propulsion* to calculate the SOC.

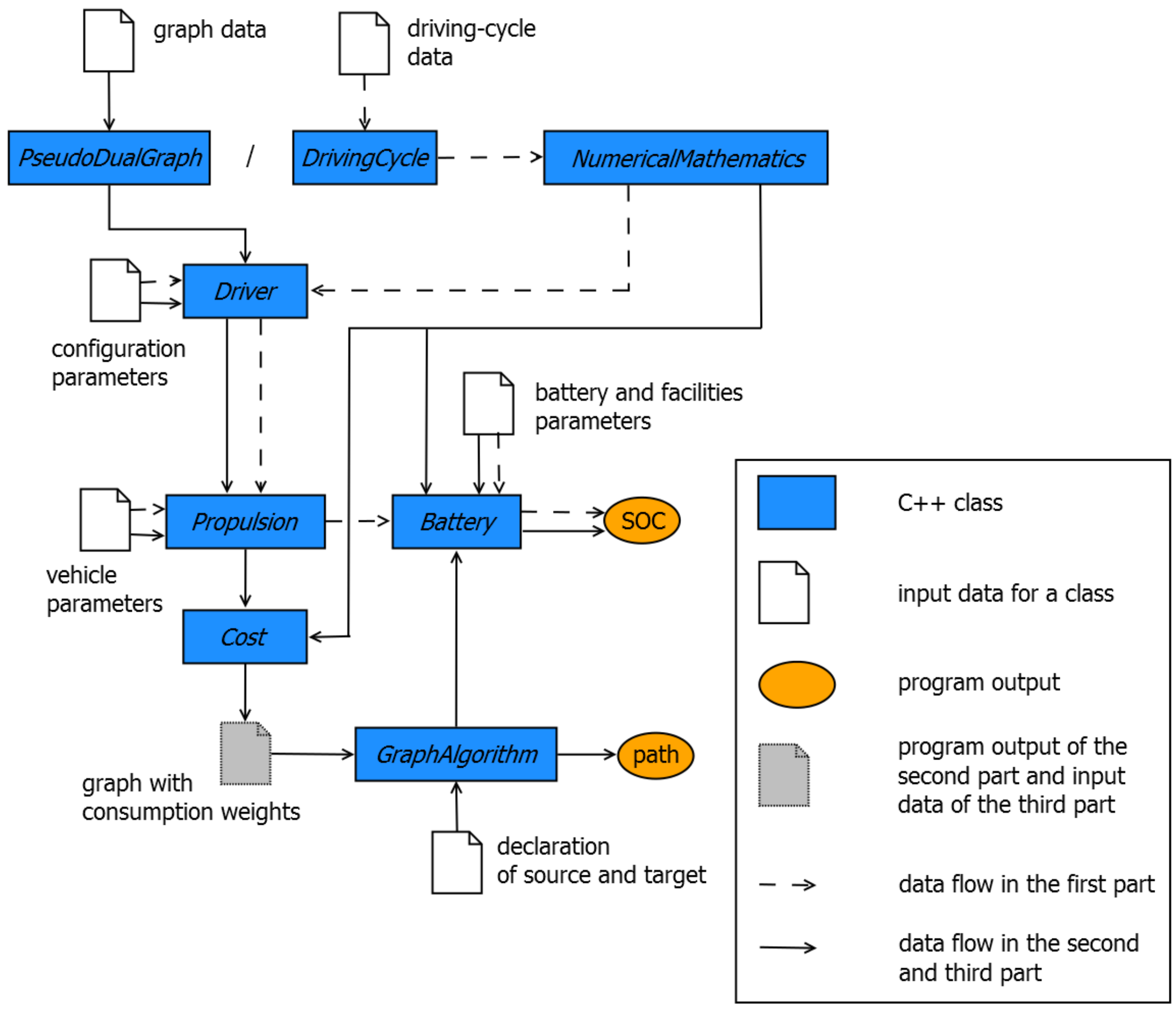

graph data
driving-cycle data
PseudoDualGraph
/
DrivingCycle
NumericalMathematics
configuration parameters
Driver
battery and facilities parameters
vehicle parameters
Propulsion
Battery
SOC
Cost
graph with consumption weights
GraphAlgorithm
path
declaration of source and target
C++ class
input data for a class
program output
program output of the second part and input data of the third part
data flow in the first part
data flow in the second and third part

Figure 1.1: Structure of the particular programs of the three different self-contained parts in the thesis

The data flow of the tool for finding the most energy-efficient path and the actual SOC of the BEV consisting of part two and three of this thesis is highlighted by solid arrows. The tool partly uses the program structure of the first part and complements it with new classes. Consequently, the data of a graph in a specific format, which will be explained in the following sections, is read in by a class for reading graph data and generating a pseudo-dual graph (*PseudoDualGraph*). Then the classes *Driver* and *Propulsion* process the data and hand over their evaluations to a class linking all program files to compute the costs of road sections (*Cost*). The class *Battery* is instantiated within *Cost* to create functions for determining the current SOC. These are called in a class performing all routing tasks (*GraphAlgorithm*) to compute and visualise the most energy-efficient path using the intermediate results of *Cost*. The intermediate results contain the pseudo-dual graph and the belonging calculated energy-consumption data. These are also the results of the self-contained program for creating the graph, described by the second part of this thesis. The graph algorithm finding the path forms the third part of this thesis and will be addressed in Chapter 5. Functions of *NumericalMathematics* are used for numeric calculations in *Battery* and *Cost*.

Chapter 2

Graph-Theoretic Preliminaries

This chapter may be skipped by the reader who is familiar with the concepts of graphs, pseudo-dual graphs and graph algorithms. The basic ideas and pseudo codes for graphs and graph algorithms are discussed in [33] and will also be briefly summarised here. A definition of the pseudo-dual graph can be found in [51].

2.1 Graphs and Graph Models for Specific Routing Tasks

A *graph* G consists of a *set* V *of vertices* and a *set* E *of edges*. Each edge connects two vertices. From a mathematical point of view, a graph is a pair of a vertex set $V = \{v_1, v_2, ..., v_n\}$ and an edge set $E = \{e_1, e_2, ..., e_m\}$,

$$G = (V, E). \tag{2.1}$$

In this context v_i are single vertices and e_j are single edges in the Graph. The total number of vertices in V is denoted by n and the total number of edges in E is denoted by m. A graph can be stated as a set of vertices and a set of edges, for instance

$$G = \big(\{v_0, v_1, v_2, v_3, v_4\}, \{(v_0, v_1), (v_1, v_2), (v_2, v_3), (v_3, v_4)\}\big). \tag{2.2}$$

In a *directed graph* an edge $e \in E$ connects two vertices $\{u, v\} \in V$ such that one of them is the source vertex and the other is the target vertex,

$$e = (u, v). \tag{2.3}$$

Working with *undirected graphs* it is understood that an edge $(u, v) \in G$ consisting of two vertices $\{u, v\}$ also considers the reverse edge (v, u) without explicitly indicating this. It is also possible to assign a *weight* or a *cost* $c : E \longrightarrow \mathbb{R}$ to an edge, such as the distance between the two vertices it connects.

The *indegree* of a vertex v is the number of edges (u, v) linking a vertex u to vertex v,

$$\text{indegree}(v) = |\{(u, v) \in E\}|, \tag{2.4}$$

whereas the *outdegree* of v is defined as the number of edges (v, u) leaving v,

$$\text{outdegree}(v) = |\{(v, u) \in E\}|. \tag{2.5}$$

A *path* is a sequence of consecutive vertices of a graph G that are connected in pairs by edges. That is for example

$$\langle v_0, v_1, v_2 \rangle = \langle (v_0, v_1), (v_1, v_2) \rangle \tag{2.6}$$

with the vertices $\{v_0, v_1, v_2\} \in G$. The vertex where a path starts is called the *source* s and the vertex where it ends is called the *target* t. The path is *simple* if its vertices are pairwise distinct except for s and t, which can also be the same. If the latter is the case, the path is called a *cycle.* A cycle is called a *negative cycle* if the cost for travelling once around it is negative. A directed graph G is said to be *strongly connected* if there always exists a path connecting any two vertices of G. The graph of a street map should always be strongly connected, such that the case of a vertex not being reachable from another vertex does not have to be considered. A path is a *shortest path* if the cost to get from s to t is minimal. Figure 2.1 shows a small example graph with weights, also a negative one. The graph is

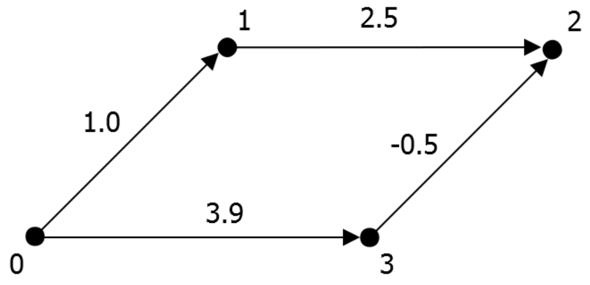


Figure 2.1: Small graph with weights

directed, that means it is only possible to travel in direction of the arrows. The vertices are numbered from zero to four, the edge weights are indicated as real numbers above the edges. In a street graph most of the vertices are connected such that it is feasible to get to the other from each of two linked vertices, unless the edge is a one-way street.

Within this thesis another type of graph is used, namely the *pseudo-dual graph.* Its definition is taken from [51] and is formulated in this thesis as follows. The graph $D(V_D, E_D)$ is the complete pseudo-dual graph of the primal graph $G(V, E)$ if three conditions are met. First, if $\bar{f}$ is a bijective function that converts an edge e to a vertex of the pseudo-dual graph D, for each edge $e_i \in G$ there has to be a vertex $\nu_i = \bar{f}(e_i)$ and $\bar{f}^{-1}(\nu_i) = e_i$ must apply. So, the edges of the primal graph are the new vertices of the pseudo-dual graph and the number of vertices n_D in the pseudo-dual graph equals m.

Second, for each pair of consecutive edges $(e_i, e_j) \in G$ there has to be a *pseudo-dual edge* $\varepsilon \in E_D$ connecting the vertices $\nu_i = \bar{f}(e_i), \nu_j = \bar{f}(e_j)$ such that ν_i is the source vertex and ν_j is the target vertex. Thus, the edges of the pseudo-dual graph connect two consecutive edges of the primal graph and the number of edges in the pseudo-dual graph m_D equals the sum of all pseudo-dual edges ε_i produced in this way.

Third, there has to be a cost function $c^{\varepsilon} : E_D \longrightarrow \mathbb{R}$. This function describes the costs for all edges of E of the primal graph and the costs it takes to pass from e_i to e_j for each consecutive pair of edges $(e_i, e_j) \in G$.

A *restricted pseudo-dual graph* R is a complete pseudo-dual graph D which has no edges connecting edges of the primal graph that link the same vertices of the primal graph in two directions. Doing so, turning on edges of the primal graph without passing the end vertices is prohibited.

The procedure to create a restricted pseudo-dual graph from a given primal graph is illustrated in Figure 2.2. Therein the primal graph is depicted as a junction of directed edges indi-

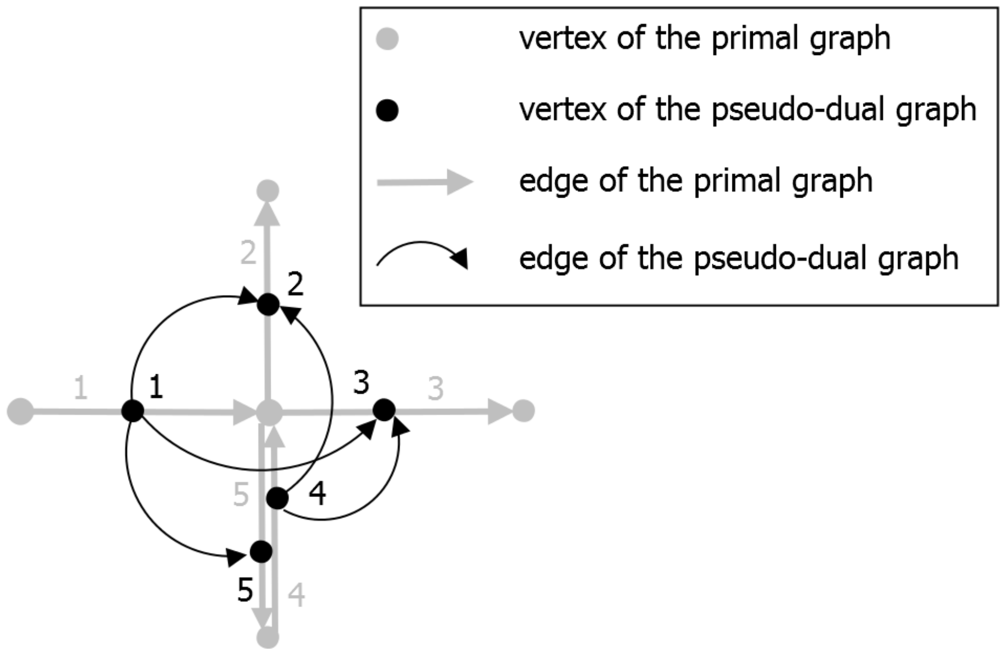

vertex of the primal graph
vertex of the pseudo-dual graph
edge of the primal graph
edge of the pseudo-dual graph
1
2
3
4
5
1
2
3
4
5

Figure 2.2: Primal graph with edge numbers (grey digits) and the related restricted pseudo-dual graph with vertex numbers (black digits)

cated by grey arrows and dots. The edges are labelled using grey digits. The related pseudo-dual graph connects consecutive edges as the pairs of edges $\{(1, 2), (1, 3), (1, 5), (4, 2), (4, 3)\}$. These are the edges of the pseudo-dual graph, which is indicated by black arrows and dots.

The edges of the primal graph are used as vertices of the pseudo-dual graph, which are labelled using black digits. Since the pseudo-dual graph is restricted, the primal edges 4 and 5 are not connected to a pseudo-dual edge. Thus, $n_D = 5$ and $m_D = 5$ hold.

2.2 Shortest Path Algorithms

As already stated before the cost of a shortest path from s to t determined by the cost function c is minimal. Since there may exist several paths with the same minimal cost, it is spoken of a shortest path and not of the shortest path. The *distance* of a vertex u is defined as the cost it takes to get there from the source s. A *shortest path algorithm* is a graph algorithm that finds shortest paths from a source s to a target t. Several of them yet have been established, the most famous are the Bellman-Ford algorithm from [13] and [19] and Dijkstra's algorithm from [17]. The pseudo codes to describe the algorithms are taken from [33] because clear versions of them are stated there. Accordingly, algorithm instructions are written bold, parameters are listed in parentheses and return or assigned values are stated after a colon. In this context, a *function* returns a certain value whereas a *procedure* has no return value.

Both algorithms use the so-called edge-relaxation procedure named *relax* within this thesis. The pseudo code for *relax* accepting an edge e as parameter is given in Figure 2.3. If $d[u]$ is

```
Procedure relax ($e = (u, v)$ : Edge)
    if $d[u] + c(e) < d[v]$ then $d[v] := d[u] + c(e)$;    $parent[v] := u$
```

Figure 2.3: Pseudo code for the edge-relaxation procedure

the distance from s to u and $c((u, v))$ is the cost of an edge coming out of u, the procedure works as follows. The distance of a vertex v consecutive to u is updated to the distance $d[u]$ plus the cost to get from u to v if the distance of v up to this time is larger than that. Doing so, the distance of v will be decreased as long as edges (u, v) with lower cost can be found. Additionally, a parent vector can be created to store the predecessors of v in the shortest path, in this case $parent[v] = u$. To *relax* an edge means applying the procedure *relax* to the edge.

The complete algorithm of Dijkstra is given in Figure 2.4. In Dijkstra's algorithm the distance vector d is declared and initialised with infinity on each position in d. The parent vector is declared and can be initialised with undefined values $\perp$. The parent of s is set as s. Then, an *addressable priority queue* for vertices is declared. An addressable priority queue is a data structure whose elements are assigned with *keys* that determine the processing sequence of the elements. The procedure *decreaseKey* supplied by the data structure removes the element just processed, decreases its key or increases its priority and reinserts the element.

```
**Function** $Dijkstra\ (s : VertexId) : VertexArray \times VertexArray$
    $d = \langle \infty, ..., \infty \rangle : VertexArray$ **of** $\mathbb{R} \cup \{\infty\}$
    $parent = \langle \perp, ..., \perp \rangle : VertexArray$ **of** $VertexId$
    $parent[s] := s$
    $Q : VertexPriorityQueue$
    $d[s] := 0; \quad Q.insert(s)$
    **while** $Q \neq 0$ **do**
        $u := Q.deleteMin$
        **foreach** $e = (u, v) \in E$ **do**
            **if** $d[u] + c(e) < d[v]$ **then**
                $d[v] := d[u] + c(e)$
                $parent[v] := u$
                **if** $\nu \in Q$ **then** $Q.decreaseKey(\nu)$
                **else** $Q.insert(\nu)$
    **return** $(d, parent)$
```

Figure 2.4: Pseudo code for Dijkstra's algorithm

The procedure *insert* inserts an element into the priority queue and the procedure *deleteMin* returns the element with the minimal key and deletes it from the priority queue. In the algorithm all reached vertices the edges of which are not all relaxed yet are stored in the adressable priority queue. Doing so, their tentative distance values are used as keys. Thus, the distance of s is set as zero and s is inserted into the addressable priority queue. While there are still vertices in the priority queue, the edges going out of the vertex with the minimal key are relaxed. If the distance of a vertex v is updated, the *decreaseKey* operation changes the key of v accordingly if it is an element of the priority queue already. Else if $d[v]$ is updated but is not an element of the priority queue yet, v will be inserted. The function returns the distance and the parent vector. In this way the shortest path is found quickly by considering the currently most promising path only. The algorithm can be stopped as soon as the element taken from the top of the priority queue is the target t. This technique is called early stopping.

The complete Bellman-Ford algorithm is given in Figure 2.5. In the Bellman-Ford algorithm

```
**Function** $BellmanFord\ (s : VertexId) : VertexArray \times VertexArray$
    $d = \langle \infty, ..., \infty \rangle : VertexArray$ **of** $\mathbb{R} \cup \{-\infty, \infty\}$
    $parent = \langle \perp, ..., \perp \rangle : VertexArray$ **of** $VertexId$
    $d[s] := 0; \quad parent[s] := s$
    **for** $i := 1$ **to** $n - 1$ **do**
        **forall** $e = (u, v) \in E$ **do** $relax(e)$
    **return** $(d, parent)$
```

Figure 2.5: Pseudo code for the Bellman-Ford algorithm

the vectors or arrays are declared and initialised in the same way as in Dijkstra's algorithm.

The edge-relaxation procedure is carried out for every edge $n-1$ times. Doing so, in the first step all shortest paths including at most one edge are calculated, in the second all shortest paths with at most two edges are calculated and so on until all shortest paths including $n-1$ edges are found. A simple path in G has at most $n-1$ edges, so the algorithm guarantees the correct solution if there are no negative cycles in G. If there is at least one negative cycle in the graph, the distance to the destination can be decreased by going around the negative cycle repeatedly, such that the algorithm will find a shorter path after each further iteration. It is unlikely that there exist negative cycles in the graph of a street map because the cost functions usually describe non-negative distances, slopes or energy consumption and negative cycles are physically impossible then.

The two presented algorithms pursue the same target, but each has its advantages and disadvantages. An important criterion is the runtime. Asymptotic growth rates of the runtimes of algorithms are usually stated with a capitalised O followed by a formula in parentheses, containing numbers of characteristic input parameters. For a more detailed explanation see [33]. If there are m edges and n vertices, the runtime of Dijkstra's algorithm grows with $O((m+n)\cdot\log n)$, while the runtime of the Bellman-Ford algorithm grows with $O(m\cdot n)$. Thus, Dijkstra's algorithm is faster.

Unfortunately, it is not possible to apply Dijkstra's algorithm if there are negative weights in a graph. This can be seen as follows. Using the pseudo code of Figure 2.4, the shortest path from $s=0$ to $t=2$ in the graph of Figure 2.1 is sought. It is determined as $0\to 1\to 2$ with the total cost of 3.5 even before travelling along $0\to 3$ is considered. In fact, the shortest path is $0\to 3\to 2$ with the total cost of 3.4 though. The Bellman-Ford algorithm iterates shortest paths from paths including at most one edge to paths with at most $n-1$ edges. Doing so, it has no problems to process a graph with negative weights, but takes more time than Dijkstra's algorithm.

Weighted graphs can be edited such that all negative costs disappear applying a modified Bellman-Ford algorithm. Then, Dijkstra's algorithm can be applied to find shortest paths. This is called the Johnson's algorithm or sometimes potential shifting. The benefit is that the slower Bellman-Ford algorithm has to be applied on the graph only once to shift the weights. Then, for any shortest path query the fast algorithm of Dijkstra can be applied. The detailed procedure of Johnson's algorithm is as follows. A new vertex w is added to the considered directed Graph $G=(V,E)$ and edges from w to all other vertices in G are created and weighted with zero. Then, the Bellman-Ford algorithm is applied to the resulting graph $G'=(V',E')$ using the vertex w as source s. Doing so, the distances from s to all vertices are determined. The distance $d[u]$ of a vertex $u\in V'$ is now defined as the vertex potential $pot(u)$ of u. With that, the weights of the original graph can be shifted. For an edge $e(u,v)$

the inequality

$$pot(v) \leq pot(u) + c(e) \tag{2.7}$$

holds, since the potential $pot(u)$ is the cost of the shortest path to the according vertex u and therefore the distance to v can not be larger than the distance of u plus the cost it takes to get from u to v. That leads to

$$0 \leq pot(u) + c(e) - pot(v) \tag{2.8}$$

and therefore the shifted costs have to be greater or equal to zero if they are defined as

$$\bar{c}(e) = pot(u) + c(e) - pot(v). \tag{2.9}$$

Now the added vertex w is deleted again and Dijkstra's algorithm can be applied to the graph with shifted costs. It remains to be proven that the shortest path with respect to the shifted costs stays the same as the shortest path with respect to the original costs. Therefore, let p and q be paths from u to v. Then, the shortest paths determined using $\bar{c}$ are the same as those determined using c if

$$\bar{c}(p) = pot(u) + c(p) - pot(v). \tag{2.10}$$

and

$$\bar{c}(q) = pot(u) + c(q) - pot(v). \tag{2.11}$$

hold. If p is a sequence of edges $p = \langle e_0, ..., e_{k-1} \rangle$ and for each edge $e_i = (v_i, v_{i+1})$ holds and if $u = v_0$ and $v = v_k$ hold, then also

$$\begin{aligned} \bar{c}(p) &= \sum_{i=0}^{k-1} \bar{c}(e_i) = \sum_{i=0}^{k-1} (pot(v_i) + c(e_i) - pot(v_{i+1})) \\ &= pot(v_0) + \sum_{i=0}^{k-1} c(e_i) - pot(v_k) = pot(v_0) + c(p) - pot(v_k) \end{aligned} \tag{2.12}$$

holds, and hence the Equations (2.10) and (2.11) are valid. This proof was taken from [33]. The ideas for Johnson's algorithm or potential shifting are noted in [33] and [3].

Johnson's algorithm shall be illustrated using the weighted example graph depicted in Figure 2.6 with solid arrows as edges and dots as vertices. The vertices are numbered from one to four and the edge weights are stated as black digits below the edges. A new vertex with ID zero is added and edges from zero to all other vertices are created and weighted with zero, indicated by dashed arrows. The vertex potentials after applying the Bellman-Ford algorithm with $s = 0$ are as listed in Table 2.1.

Table 2.1: Vertex potentials for the example graph from Figure 2.6

Vertex v_i	Potential $pot(v_i)$
1	0
2	0
3	-0.5
4	0

By applying Equation (2.9) on the vertex potentials the shifted costs are found. They are indicated as red digits above the edges in Figure 2.6. For example, the shifted weight for the edge $(2, 3)$ according to Equation (2.9) results in

$$\bar{c}(2,3) = 0 + 2.5 - (-0.5) = 3. \tag{2.13}$$

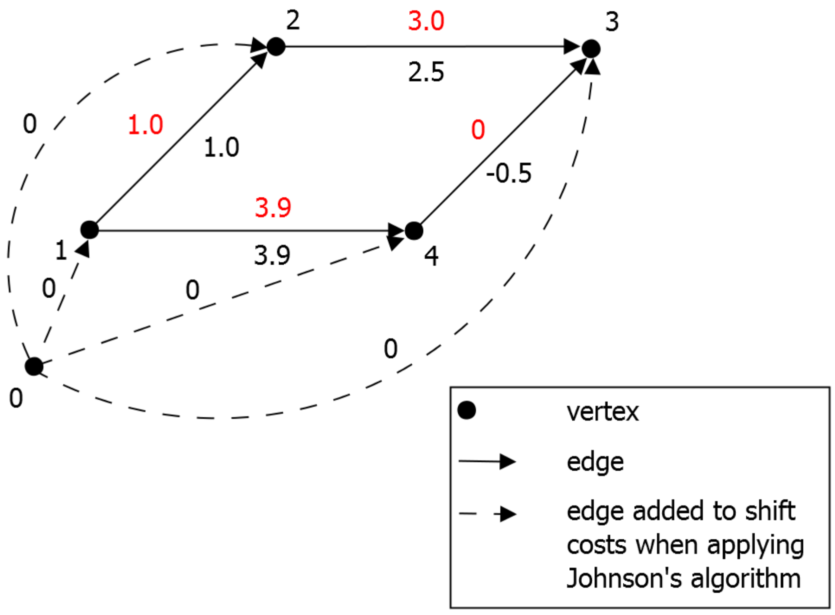


Figure 2.6: Small graph with edge weights (black digits) and shifted edge weights (red digits) found when applying Johnson's algorithm

Chapter 3

The Vehicle Model

As the first self-contained part of this thesis, this chapter describes the functionality of the classes for a program, implemented in C++, that aims to calculate the SOC of the i-MiEV over time using driving cycles as input. Figure 3.1 depicts the components of a BEV that were considered. These components were modelled within the presented classes. Most of the functions and procedures were generated as inline functions for a faster access.

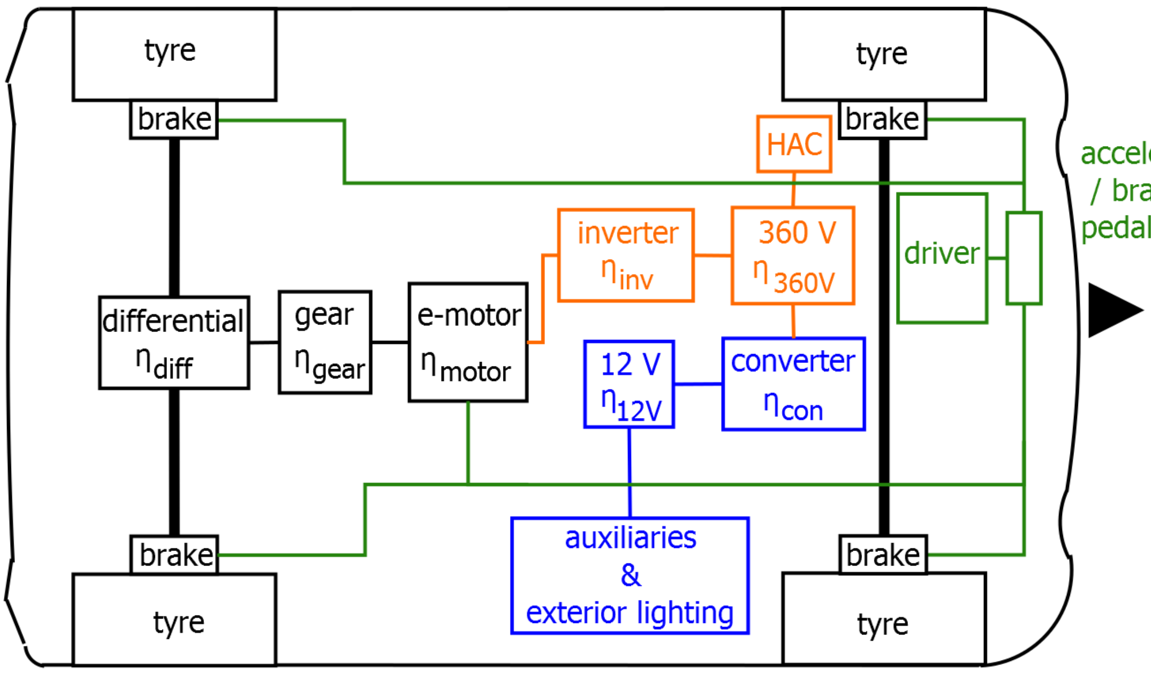


Figure 3.1: Considered components of a BEV and the belonging efficiency factors η, including control elements and signals (green), lithium-ion battery and direct electricity consumers (orange), lead-acid battery and direct electricity consumers (blue) and mechanic components (black)

A driver model was established in the class *Driver* by a controller that regulates the positions of the accelerator and the brake pedal and with that the torque of the electric motor and the brakes. Related components in Figure 3.1 are coloured green.

Power is requested from the motor when the vehicle is in use and has to be transferred over the differential with the efficiency factor η_{diff} and the gear with the efficiency factor η_{gear}. The electric motor itself also has an efficiency factor η_{motor}. Together with the external resistance forces and the forces of the mechanical brakes, these aspects were implemented in the class *Propulsion*. According modules are coloured black in Figure 3.1.

The power for the electric motor is provided by a lithium-ion battery with the usual voltage of $360\,V$. This battery provides Direct Current (DC) and the electric machine works with Alternating Current (AC). So, if the motor takes power from the lithium-ion battery, the voltage is converted from DC to AC voltage in the inverter with the efficiency factor η_{inv}. The lithium-ion battery also directly supplies the air-conditioning system and the heating, composed to Heating and Air Conditioning (HAC). It has the efficiency factor η_{360V}. All the direct electricity consumers and the battery itself are indicated as orange elements in Figure 3.1. If the BEV brakes, energy is stored in the lithium-ion battery due to a regenerative braking system. The surplus of energy in the motor is transferred to the battery via the inverter converting AC voltage to DC voltage this time. These procedures are implemented in the class *Battery*.

In this thesis, *auxiliaries* are defined as all electricity consumers except for the motor, the exterior lighting, the air-conditioning system and the heating. Hence, some remaining consumers are the interior lighting and the ventilation for example. The power for these auxiliaries and the exterior lighting is taken from a lead-acid battery, which usually has a voltage of $12V$, with the efficiency factor η_{12V}. It is charged by the lithium-ion battery and a DC-DC converter with the efficiency factor η_{con} reducing the voltage of the lithium-ion battery. The lead-acid battery and its direct electricity consumers are indicated as blue elements in Figure 3.1. The energy consumption of the exterior lighting, the air-conditioning system, the heating and the auxiliaries is specified in the class *Driver* since it is the driver who determines whether these facilities are turned on or off.

3.1 Driving Cycles as First Input Data

A class *DrivingCycle* was created to read in data of driving cycles. The according files contain specifications for the position in meters and the velocity in $\frac{km}{h}$ for a certain time reading in seconds in tabular form. A variable for the slope was introduced, expecting a specification in percent. Since the cycles are tested on flat terrain, it was set to zero. The velocity was converted to $\frac{m}{s}$ to get consistent Système International d'unités (SI) units. The memory for the data was dynamically allocated. For the processing of multiple cycles in a row, the row number of the appropriate matrix was multiplied with the number of cycles to be processed acting as new size for the matrix where the concatenated data was saved.

3.2 Functions for Numerical Mathematics

All basic data used to reproduce a driving cycle or a route is given as vectors of discrete values. This implies that numerical mathematics have to be used to calculate motion quantities.

3.2.1 Preliminaries

In [31] an introduction to numerical mathematics is given. These lecture notes contain all structures used within this thesis. Generally, the difference quotient

$$\frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} \tag{3.1}$$

is used to evaluate derivatives of a function f with respect to x at x_i. To evaluate numeric integrals, the trapezoidal rule

$$\int_{x_{i-1}}^{x_i} f(x) \cdot dx \approx (x_i - x_{i-1}) \cdot \frac{f(x_i) + f(x_{i-1})}{2} \tag{3.2}$$

has been used.

For the interpolation of data the formula

$$f(x_{inter}) = f(x_{i-1}) + \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} \cdot (x_{inter} - x_{i-1}) \tag{3.3}$$

has been used, where x_{inter} is between x_i and x_{i-1}. Hence, linear interpolation between the reference points has been used.

3.2.2 Implementation

The presented classes were instantiated and the object files were linked using a function called in the main file. In this function the desired *sample time T* as the interval between two sample values is defined. Since the data comes with a time vector without fixed intervals, it seemed to be reasonable to interpolate this data in order to get a fixed sample rate. The new number of elements arising from such an interpolation is the full duration of the driving cycle divided by the desired sample time and of course it has to be multiplied with the number of cycles to be processed as mentioned in Section 3.1. The new time vector was created multiplying the sample time with the progressing index of a loop being increased by one in every loop run until the final size of the vector is reached. All remaining data, which was extended for processing multiple cycles, was interpolated at the appropriate values of the new time vector using Formula (3.3). Consequently, the time vector is the *sample vector* and contains the times for which functional values are requested by interpolation. To find the correct index i referring to the value that is the first one following the desired sample value, in this case this would be the point in time ensuing the time to be interpolated, interpolation variables were introduced for each data vector. Such a variable is increased with unity steps as long as the sample value is greater than or equal to the variable and is used as index i in Formula (3.3). As an example, it is assumed that the functional value $f(x)$ at $x = 2.5$ has to be determined when the functional values are given as $f(1)$, $f(2)$, $f(3)$ and so on. Then if the interpolation variable var starts at 1 it is increased by one until $var = 3$, such that 2.5 lies between var and $var - 1$.

Another interpolation procedure has been implemented for a sample vector holding decreasing values with increasing index i. Doing so, while the sample value is smaller than the introduced interpolation variable starting as the last index number minus one, the variable is decreased by one and after that used as index i in Formula (3.3) again. It is supposed that $f(1.5)$ is searched when the functional values are given as $f(0)$, $f(1)$, $f(2)$ and $f(3)$. If the interpolation variable var starts at 3 it is decreased by one until $var = 2$ such that 1.5 lies between var and $var - 1$. This procedure was used to interpolate the function of the open-circuit voltage, see Section 3.5.

The introduced procedures and the numerical integral realised with the trapezoidal rule corresponding to Equation (3.2) have been implemented in the class *NumericalMathematics*. An implementation of a function for the difference quotient according to Equation (3.1) in *NumericalMathematics* did not prove useful since it would only enlarge the source code without having any impact on transparency.

3.3 Simulation of the Behaviour of a Human Driver

The driver of a car usually tries to follow an imposed velocity. That means he or she compares his actual velocity with the limit on a street or the velocity over time imposed by a driving cycle and corrects the difference by accelerating or decelerating. That complies with a controller regulating the positions of the accelerator and the brake pedal of the car to minimise the *control error*, which is the difference between the imposed and the actual velocity.

3.3.1 Preliminaries

The Equation for the continuous *control signal* $u(t)$

$$u(t) = K_P \cdot \left[e(t) + \frac{1}{T_N} \cdot \int_0^t e(\tau) \cdot d\tau + T_V \cdot \frac{de(t)}{dt} \right] \tag{3.4}$$

describes the output of a continuous-time PID-controller with the *controller-gain constant* K_P, the continuous *system deviation* or control error $e(t)$, the *reset time* T_N and the *derivative time* T_V. Using the trapezoidal rule (3.2) for integration, for the control signal $u(kT)$ of a discrete-time PID-controller

$$u(kT) = K_P \cdot \left[e(kT) + \frac{T}{T_N} \cdot \sum_{h=1}^{k} \frac{e((h-1)T) + e(hT)}{2} + \frac{T_V}{T} \cdot (e(kT) - e((k-1)T)) \right] \tag{3.5}$$

holds [27], where T is the sampling interval and k is a counter. Here, the system deviation $e(kT)$ is given as discrete value. Formula (3.5) can be modified to

$$\begin{aligned} u(kT) = u((k-1)T) + K_P \cdot \Bigg[&(e(kT) - e((k-1)T)) + \frac{T}{T_N} \cdot \frac{e((k-1)T)) + e(kT)}{2} \\ &+ \frac{T_V}{T} \cdot \big(e(kT) - 2 \cdot e((k-1)T) + e((k-2)T) \big) \Bigg] \end{aligned} \tag{3.6}$$

by subtracting $u((k-1)T)$ from $u(kT)$ calculated with Equation (3.5). Equation (3.6) is known as the position algorithm for a PID-controller using the trapezoidal rule. Within this thesis it is assumed that the controller generates the control signal as the input of an actuator. This actuator affects the variable to be controlled, the *control variable* $y(t)$. For example, if the liquid level in a container has to be controlled, the control variable is the liquid level and the control signal manipulates the position of a valve to adjust the liquid level. More information about digital control can be found in [27].

Furthermore the windup effect and countermeasures remain to be discussed. In practical applications the control signal $u(t)$ is limited due to limitation of the available controller-output power or due to input limitations of the system to be controlled. The limitation of the control signal $u(t)$ can be modelled by limiting $u(t)$ to upper and lower limits $\pm u_{sat,max}$ by means of a limiter function. This means, that if the control signal $u(t)$ ranges in the saturation region the saturated value $+u_{sat,max}$ or $-u_{sat,max}$ is forwarded to the system to be controlled. This leads for feedback systems with an integral term in the controller to the effect, that the control error is continually integrated and the absolute value of the output of the integrator $u_i(t)$ increases as can be seen in Figure 3.2. This is called the windup effect. When the control error changes its sign, $u_i(t)$ decreases and if it is small enough to be inside the restriction bounds, the control loop works in the desired way again. During the time when $u(t)$ is limited, the windup effect leads to an unwanted over- or undershoot of the control variable $y(t)$. As a countermeasure, the windup is counteracted proportional to $(u(t) - u_{sat}(t))$, where $u_{sat}(t)$ is the output of the limiter function, depicted in Figure 3.2. This leads to the equation

$$\frac{du_i}{dt} = \frac{K_P}{T_N} \cdot e(t) - \frac{1}{T_a} \cdot (u(t) - u_{sat}(t)) \tag{3.7}$$

for the change of $u_i(t)$ using the *tracking time constant* T_a as proportional factor. [25]

Figure 3.2 shows the part of a control loop with the controller and the discussed anti-windup measure. The auxiliary functions

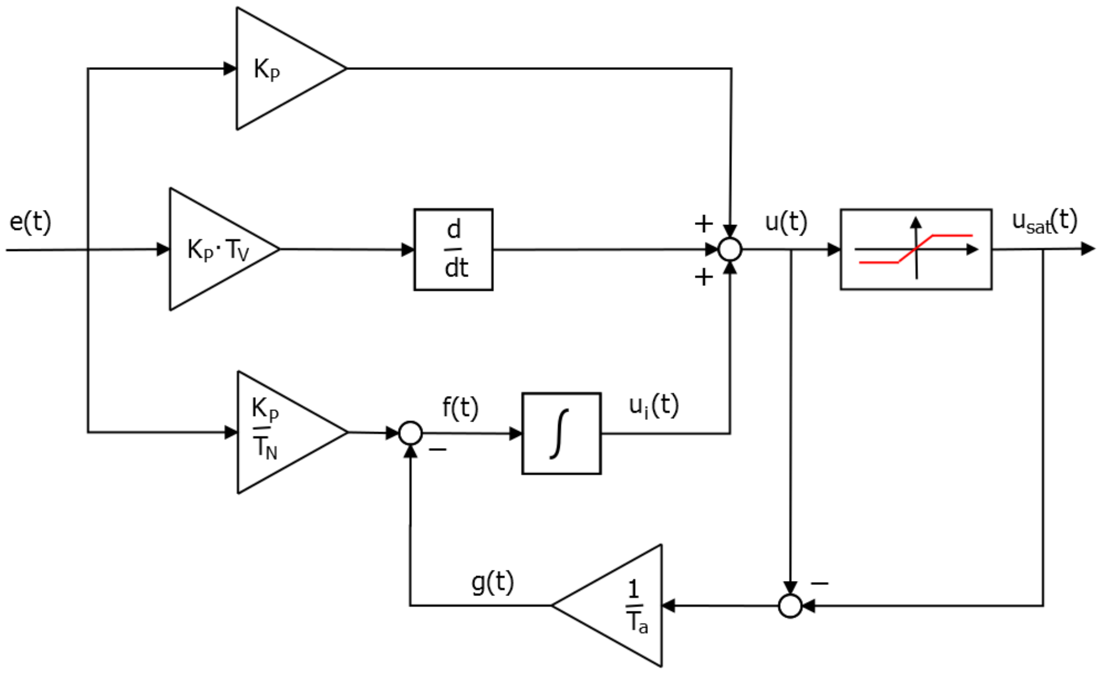


Figure 3.2: PID-controller and anti-windup measure

$$g(t) := \frac{1}{T_a} \cdot (u(t) - u_{sat}(t)) \tag{3.8}$$

and

$$f(t) := \frac{K_P}{T_N} \cdot e(t) - g(t) \tag{3.9}$$

are used to describe the intermediate steps in Section 3.3.2. The operating principle of a continuous PID-controller as described by Equation (3.4) is also illustrated in Figure 3.2. From the top down there are three branches describing the P-, D- and I-part of the controller. The branch for the anti-windup measure is only considered in the I-part of the controller. The figure can also be consulted for a discrete PID-controller if the variable t for the continuous time is replaced by kT.

3.3.2 Implementation

In the class *Driver* a function for the control error and another one for the controller returning the limited control signal $u_{sat}(kT)$ manipulating the positions of the accelerator pedal and the brake pedal have been implemented. The positions of the pedals take the part of an actuator adjusting the actual propulsion force and therefore the velocity of the car, as discussed in Section 3.4. Thus, the velocity of the car is the control variable. The controller was realised as a PID-controller with anti-windup measures as explained in Section 3.3.1. Figure 3.2 also holds for the implemented controller, only the continuous time t has to be replaced by discrete steps kT. Initially $g(kT)$ and $f(kT)$ are zero and from then on

$$g(kT) = \frac{u(kT) - u_{sat}(kT)}{T_a} \tag{3.10}$$

and

$$f(kT) = \frac{K_P}{T_N} \cdot e(kT) - g(kT) \tag{3.11}$$

hold. According to Equation (3.6) the control signal before being saturated is

$$\begin{aligned} u(kT) = u((k-1)T) + K_P \cdot (e(kT) - e((k-1)T)) + T \cdot \frac{f((k-1)T) + f(kT)}{2} \\ + K_P \cdot \frac{T_V}{T} \cdot \big(e(kT) - 2 \cdot e((k-1)T)) + e((k-2)T))\big). \end{aligned} \tag{3.12}$$

In a saturation block the control signal was set as $u_{sat}(kT) = 1$ if greater and as $u_{sat}(kT) = -1$ if smaller, else it just kept the value of $u_{sat}(kT) = u(kT)$. The values 1 and -1 were defined as the end positions of the accelerator pedal and the brake pedal. The controller-gain constant K_P was set as 0.4, the reset time T_N as $0.4667\,s$, the derivative time T_V as $0\,s$ and the tracking time constant T_a as $0.1\,s$. This configuration was set empirically by assessing the behaviour of a typical driver. Doing so, the method of the cumulative time constant described in [25] was used to get first estimates for the controller settings for a stable control. This method uses empirical tuning rules for a controller looking at the *step response* of the system to be controlled, meaning that the reaction of the system to a sudden change of the variable to be controlled has to be evaluated. In the prevailing circumstances the desired step response to a change of speed limits on roads of the system is a constant acceleration of the vehicle from the current velocity to the changed speed limit. As maximal acceleration and deceleration values found in the Worldwide harmonized Light-Duty vehicles Test Procedure (WLTP) the acceleration a_{ap} assumed for an acceleration process was set as $1.5\,\frac{m}{s^2}$ and the deceleration a_{dp} assumed for a braking process was set as $2\,\frac{m}{s^2}$. Figure 3.3 depicts the desired acceleration process of the vehicle for a speed change from 0 to $10\,\frac{km}{h}$ and the system parameters used for the method of the cumulative time constant. Thus, the system parameter K_S is the gain of the control variable in the step response. The *cumulative time constant* T_Σ can be defined as half the time it takes to reach K_S with constant acceleration or deceleration in these circumstances. Table 3.1 lists the formulas for setting either a PI- or a PID-controller. The first column states the different options for controller selection. Columns two to four give the empirical formulas for K_P, T_N and T_V. After calculating the controller parameters for a fast PI-controller the settings were experimentally fine tuned.

Table 3.1: control parameter settings according to the method of the cumulative time constant

setting options	K_P	T_N	T_V
“normal” settings for PI-controllers	$\frac{1}{2 \cdot K_S}$	$0.5 \cdot T_\Sigma$	-
“normal” settings for PID-controllers	$\frac{1}{K_S}$	$0.66 \cdot T_\Sigma$	$0.167 \cdot T_\Sigma$
“fast” settings for PI-controllers	$\frac{1}{K_S}$	$0.7 \cdot T_\Sigma$	-
“fast” settings for PID-controllers	$\frac{2}{K_S}$	$0.8 \cdot T_\Sigma$	$0.194 \cdot T_\Sigma$

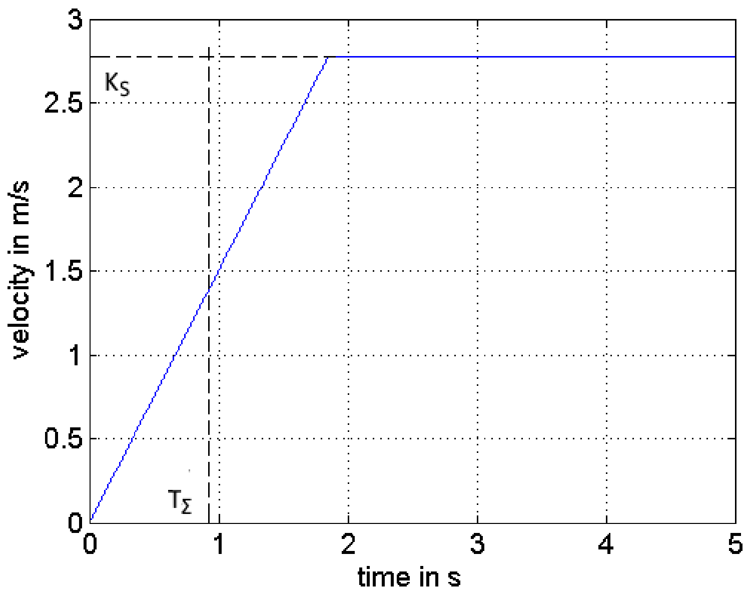


Figure 3.3: Desired step response of the system to a speed change from 0 to 10 $\frac{km}{h}$ and system parameters K_S and T_Σ

Additionally, the class *Driver* contains constants for the power consumption of exterior lighting, heating, air conditioning and auxiliaries defined as the remaining consumers besides the electric motor. These constants are added to the power consumption from the battery discussed in Section 3.5 if the according facilities are turned on, considering the appropriate efficiency factors. They were set up in this class, since it is the driver who decides whether these facilities are turned on. The consumption of the facilities just mentioned was evaluated experimentally for the i-MiEV at the AIT Austrian Institute of Technology GmbH (AIT) at the batteries. Figure 3.4 exemplary shows the measurements of the power consumption of heating and air conditioning from the lithium-ion battery if the air-conditioning system is turned on. As can be seen, the power consumption shows large fluctuations. Using MATLAB® the mean values of measurements such as the one depicted in Figure 3.4

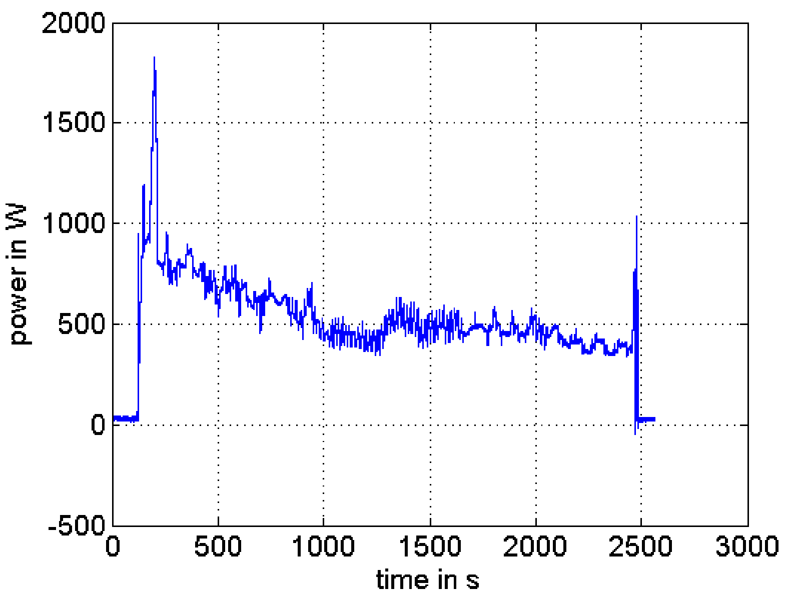


Figure 3.4: Power-consumption measurement of heating and air conditioning from the lithium-ion battery when turned on

were calculated and used for modelling the power consumption of the facilities. If the air-conditioning system is turned off, the auxiliaries consume $P_{aux} = 140\,W$ from the lead-acid battery on average. If the air-conditioning system is turned on, ventilation is assumed to be turned on too such that the average consume of the auxiliaries from the lead-acid battery increases to $P_{aux} = 450\,W$. Heating and air conditioning on average consume $P_{HAC} = 550\,W$ directly from the lithium-ion battery when turned on. Since the lead-acid battery is supplied with power by the lithium-ion battery and the DC-DC converter, the power consumption $P_{aux,360V}$ of the auxiliaries from the lithium-ion battery is

$$P_{aux,360V} = \frac{P_{aux}}{\eta_{12V} \cdot \eta_{con}}. \tag{3.13}$$

The efficiency factor of the inverter is used only for the power consumption of the motor and the efficiency factor of the lithium-ion battery is considered in the class *Battery*. Battery configurations have been created as general as possible such that the tool remains applicable to other vehicles. Efficiency factors have been estimated to $\eta_{12V} = 0.85$ for charging and discharging the lead-acid battery according to [45] and $\eta_{con} = 0.95$ since the efficiencies of DC-DC converters are in the range of $90\,\% - 95\,\%$, as can be found in literature such as [49]. If the traditional exterior lighting system for night-time driving is turned on, the power consumption of the exterior lighting P_{light} from the lithium-ion battery is determined by

$$P_{light} = \frac{2 \cdot P_{low} + 2 \cdot P_{tail}}{\eta_{12V} \cdot \eta_{con}}, \tag{3.14}$$

with the power consumption P_{low} of a low beam and P_{tail} of a tail lamp. Since the power for the exterior lighting is provided by the lead-acid battery, the same efficiency factors as for the auxiliaries have been considered in Equation (3.14). The variables were set as $P_{low} = 56.2\,W$ and $P_{tail} = 7.2\,W$ according to [43]. All the variable assignments are listed in Table 3.2 in Section 3.6 again.

3.4 Model of the Propulsion and the Resistances

This section describes the implementation of the generation of the propulsion force and the driving resistances. Additionally, the transfer of power via the power train is modelled.

3.4.1 Preliminaries

Mechanical forces acting against the motion of the vehicle are described by driving resistances. These are air drag, slope resistance, rolling resistance and inertial drag, occurring when moving in a fluid, driving on uneven terrain, rolling on a ground with friction and accelerating a mass. The corresponding formulas are listed in Section 3.4.2.

To understand characteristics of electrical motors it is helpful to understand the simplest example of the DC machine. For the induced voltage in a DC machine

$$U_i = z \cdot \frac{p}{b} \cdot \Phi \cdot n = k_1 \cdot \Phi \cdot n \tag{3.15}$$

applies, where z is the number of conductors, p is the number of pole pairs and b the number of current branches of a winding layer on the circumference of the armature. The symbol Φ describes the magnetic flux and n are the rotations per second of the armature, k_1 is a machine constant. Using

$$\omega_m = 2 \cdot \pi \cdot n \tag{3.16}$$

for the angular velocity ω_m of the machine,

$$U_i = \frac{k_1}{2 \cdot \pi} \cdot \Phi \cdot \omega_m = k_2 \cdot \Phi \cdot \omega_m \tag{3.17}$$

can be deduced. For the generated torque M_i

$$M_i = k_2 \cdot \Phi \cdot I_A \tag{3.18}$$

with the armature current I_A can be applied consequently. The mechanical power of the machine results in

$$P_m = k_2 \cdot \Phi \cdot I_A \cdot \omega_m. \tag{3.19}$$

[42]

Due to the high mechanical wear in DC machines, three-phase machines such as the Permanently excited Synchronous Machine (PSM) are used in EVs instead. For a PSM a controller demands a torque to be applied by the machine. The current is set to flow perpendicular to the magnetic flux, so that for an observer situated on the rotor of the machine the conditions are equivalent to the ones of a DC machine. [44]

Instead of modelling a motor for an EV directly, using equations like the ones above, it can be described by its *characteristic* that is based on those equations. In a simulation this saves execution time since data is read from the characteristic instead of being calculated. A typical characteristic of a synchronous motor is illustrated in Figure 3.5. The characteristic

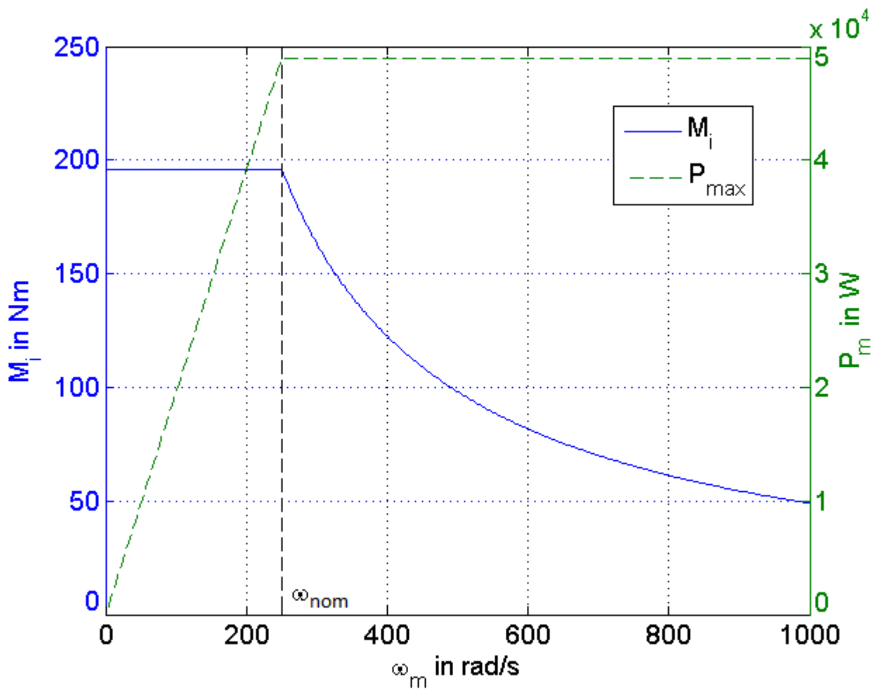


Figure 3.5: Characteristic curve of the motor torque and maximum available power dependent on the angular velocity of the motor

in Figure 3.5 applies to the motor of the i-MiEV and was generated at the AIT, described by the equations

$$M_i = M_{max} \quad \text{for} \quad \omega_m < \omega_{nom} \tag{3.20}$$

and

$$M_i = M_{max} \cdot \frac{\omega_{nom}}{|\omega_m|} \quad \text{for} \quad \omega_m \geqslant \omega_{nom} \tag{3.21}$$

respectively. Pursuant to Equation (3.17) the induced voltage increases and the armature current stays constant while ω_m is increased until the *nominal angular velocity* ω_{nom} is reached. The maximum motor power P_{max} is reached at this point such that the armature current and thus the torque of the machine decrease for higher ω_m according to Equations (3.18) and (3.19) respectively, while P_m remains constant. Detailed discussions of three-phase machines can be found in [42] and [44].

Recuperation is subject to both electrics and mechanics, since mechanical energy is transformed into electrical energy which can be stored in a battery. In this study it is counted to mechanics, considering the mechanical power that can be stored. Therefore, the motor's characteristic curve is regarded as a limitation and *serial recuperation* is assumed as it is described in [24]. Accordingly, a generator converts mechanical energy into electrical energy while braking until the brake demand exceeds the generator's torque pursuant to its characteristic curve. Then a friction brake helps to meet the full demand. Another strategy is called *parallel recuperation* where the brake demand is divided to the friction and the generator brake in a specific relation [24].

3.4.2 Implementation

The class *Propulsion* is the most important part to provide input for a graph algorithm as cost functions since it comprises the phenomenons with the biggest impact on the EV. As external forces acting on the EV the air drag

$$\frac{\rho \cdot A \cdot c_d}{2} \cdot v^2, \tag{3.22}$$

the slope resistance arising whenever there is a gradient

$$m_{tot} \cdot g \cdot \sin\alpha, \tag{3.23}$$

the rolling resistance

$$m_{tot} \cdot g \cdot c_{rr} \cdot \cos\alpha \tag{3.24}$$

and the inertial drag as the force needed to accelerate the total mass m_{tot}

$$m_{tot} \cdot a \tag{3.25}$$

have been implemented. Apart from the natural gravitational constant $g = 9.81\,\frac{m}{s^2}$, the air density assumed as $\rho = 1.25\,\frac{kg}{m^3}$ and some measurands that usually are not indicated by manufacturers, the factors of the formulas above originate from [4]. The frontal surface area A was assessed by

$$A = h \cdot w - h_c \cdot w + 2 \cdot w_t \cdot h_c, \tag{3.26}$$

slightly overestimating the value in order to be sure that the tool does not give too optimistic predictions of driving range. The height $h = 1.61\,m$, width $w = 1.475\,m$, ground clearance $h_c = 0.15m$ and tyre width $w_t = 0.145\,m$ were sourced from [4] as well as the curb weight $m_{chassis} = 1085\,kg$ of the car, the frontal surface area resulted in $A = 2.2\,m^2$ after rounding up the outcome of Equation (3.26). The total mass m_{tot} results from

$$m_{tot} = m_{chassis} + m_{load}, \tag{3.27}$$

where m_{load} is the additional mass of occupants and cargo set to $85\,kg$, such that $m_{tot} = 1170\,kg$ applies. The value for the load was set empirically, considering a driver with average weight and some additional cargo. It was assumed that usually some luggage is carried. Moreover, for measurement runs instruments have to be transported that have a certain weight. Referring to [36] the drag coefficient c_d was set to 0.35. Pursuant to literature, the rolling-resistance coefficient c_{rr} for car tyres on asphalt can adopt values from 0.013 to 0.015. As proposed in [53] $c_{rr} = 0.015$ was set, which is a conservative estimate.

To further take into account what happens in the EV the inertial losses of rotating parts were considered as well. Therefore, the angular velocity of the tyres ω_t is determined as the ratio of the increase of the actual velocity dv and the sampling interval T,

$$\omega_t = \frac{dv}{T}. \tag{3.28}$$

In doing so, dv has been specified in the main function as the difference of the actual velocity at the current loop index i and the preceding v at $i-1$. As a reminder it is noted that the passed time is identified by $i \cdot T$. The angular acceleration $\dot{\omega}_t$ was established as

$$\dot{\omega}_t = \frac{d\omega_t}{T}. \tag{3.29}$$

Inertial losses result in forces that arise when the torques of inner rotating parts are converted to the angular velocity of the tyres by transmission ratios and divided by the radius of a tyre r_t. The torque due to inertia of an inner rotating part is determined by its *moment of inertia* J, defined as the resistance of a rigid body to changing its rotational motion. The torques due to inertia recognised in this thesis are the torque of the motor M_m due to its inertia

$$M_m = J_m \cdot \dot{\omega}_t \cdot k_{gear}, \tag{3.30}$$

the torque of the gear M_g due to its inertia

$$M_g = (J_{g,t} + J_{g,m} \cdot k_{gear}) \cdot \dot{\omega}_t \tag{3.31}$$

and the torque of the tyres M_t due to their inertia

$$M_t = 4 \cdot J_t \cdot \dot{\omega}_t. \tag{3.32}$$

The transmission ratio k_{gear} describes the ratio of the angular velocity of the motor and the angular velocity of the tyres. So here

$$k_{gear} = \frac{\omega_m}{\omega_t} \tag{3.33}$$

applies. A torque produced by a moment of inertia is calculated by the multiplication of the moment of inertia with the related angular velocity. Hence, according to Equation (3.33), the moment of inertia of the machine J_m has to be multiplied with k_{gear} and ω_t as it is done in Equation (3.30). In most EVs there is only one gear. The gear has the moments of inertia $J_{g,m}$ at the shaft of the motor and $J_{g,t}$ at the shaft of the wheels and tyres. In Equation (3.31) $J_{g,m}$ is multiplied with k_{gear} again because its angular velocity is ω_m. There are four tyres, so in Equation (3.32) the moment of inertia of a tyre is multiplied by four. Since the force is transferred to the road at the tyres, the torques of Equations (3.30) – (3.33) were added up and divided by r_t, as a torque is the vector product of a radius and a force. For the i-MiEV the parameters are $r_t = 0.285\,m$ and $k_{gear} = 7.065$ according to [4], $J_m = 0.02\,kg\,m^2$, $J_{g,m} = 0.015\,kg\,m^2$, $J_{g,t} = 0.015\,kg\,m^2$ and $J_t = 1.23\,kg\,m^2$ based on measurements at the AIT. The tyre designation of [4] indicates the tyre width in mm, the ratio of height to width in percent and the inner diameter in inches. Therefore the diameter can be calculated by converting the inner diameter to meters by multiplying with 0.0254 and then adding twice the width in meters times the aspect ratio divided by 100. To get the radius, the diameter has to be divided by two. In this case there were two different tyre types, one for the front part and one for the rear part of the vehicle, but since the radii did not differ much the same value has been chosen for both of them.

The torque of the motor $M_i(\omega_m)$ available at a certain value of the the angular velocity was determined by the characteristic curve of the motor presented in Section 3.4.1. In [4] the maximum torque and the maximum power of the electric motor are quoted as $M_{max} = 196\,N\,m$ and $P_{max} = 49\,kW$ such that $\omega_{nom} = 250\frac{rad}{s}$ can be calculated from the formula

$$P_{max} = M_{max} \cdot \omega_{nom}. \tag{3.34}$$

The angular velocity of the machine was determined by the formula

$$\omega_m = \frac{v}{r_t} \cdot k_{mech}, \tag{3.35}$$

where the total transmission ratio k_{mech} is usually specified by the manufacturer. According to [4] k_{mech} is 7.065. Within this thesis it is assumed that the total transmission ratio k_{mech} can be calculated by the formula

$$k_{mech} = k_{gear} \cdot k_{diff} \tag{3.36}$$

where for the transmission ratio of the differential $k_{diff} = 1$ holds. The equations for the characteristic curve and ω_m were implemented in a function using v as parameter.

Another torque to be considered is the mechanical brake torque. Figure 3.6 illustrates the working principle of the mechanical brake. A function was introduced to get the maximum brake torque M_b determined by the equation

$$M_b = 4 \cdot F_N \cdot \mu \cdot r_m \cdot f, \tag{3.37}$$

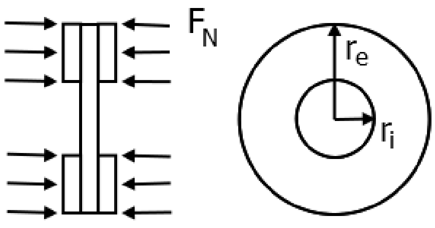


Figure 3.6: Sketch of the mechanical brake with the maximum normal force F_N applied on the brake disc from the block pads

where $F_N = 5\,kN$ is the maximum normal force from the block pads on a brake disc and $\mu = 0.3$ is the friction coefficient for brake disc and block pads. The number of friction faces f for such a brake is 2. Equation (3.37) arises from the fact that the friction force applies continuously at places with different distances to the rotation axis. Thus, the force per area has to be integrated over the total area of the brake disc according to the formula

$$M_b = \int_0^{2\cdot\pi} \int_{r_i}^{r_e} \frac{\mu \cdot F_N}{({r_e}^2 - {r_i}^2) \cdot \pi} \cdot r^2 dr \cdot d\varphi, \tag{3.38}$$

where $r_i = 0.08\,m$ is the internal and $r_e = 0.12\,m$ the external brake-disc radius. In the integral of Equation (3.38) the symbols r and φ have been used as integral variables for the radius of the disc and the rotation angle respectively. The mean radius of a brake disc r_m appearing in Equation (3.37) was then defined as

$$r_m = \frac{2}{3} \cdot \frac{{r_e}^3 - {r_i}^3}{{r_e}^2 - {r_i}^3}. \tag{3.39}$$

Data for the brake system such as maximum normal force, friction coefficient and brake-disc radii were already present at the AIT.

In the main function of the program the actual propulsion force F available from the motor is calculated using the currently present torques, projected by the accelerator pedal position u_{sat} of the last step times M_i if accelerating or u_{sat} times the sum of M_i and M_b if decelerating, and the SOC of the last step as input parameters. In this way, the force for accelerating or decelerating the BEV is described by its motor characteristic and the driver pressing the accelerator or the brake pedal. Since u_{sat} is negative when decelerating, no algebraic sign has to be added in any constellation described above. The function for the computation of F was implemented in the class *Propulsion*. If the SOC is smaller or equal to zero this function returns zero, else it determines F with the formula

$$F = \frac{\eta_{mech} \cdot k_{mech} \cdot M}{r_t}, \tag{3.40}$$

where the symbol M describes the torque passed to the function as a parameter. The mechanical efficiency factor η_{mech} is the product of the efficiency factors η_{motor} of the motor,

η_{gear} of the gear and η_{diff} of the differential,

$$\eta_{mech} = \eta_{motor} \cdot \eta_{gear} \cdot \eta_{diff}. \tag{3.41}$$

The efficiencies η_{motor} and η_{gear} were set as 0.95 and 0.97 as proposed in [23] as general values for modern BEVs. The efficiency factor η_{diff} was set as 0.97 according to [54], where a value for the efficiency of a differential was calculated exemplary. This calculated value was adopted, rounding off the result in order to stay conservative. Since in real vehicles there are power losses from the motor to the tyres, the power at the tyres is $\eta_{mech} \cdot P_m$. For rotating parts the power is the product of the torque and the angular velocity, so that the torque at the tyres is $\eta_{mech} \cdot k_{mech} \cdot M$. To get the according propulsion force the torque at the tyres is divided by r_t.

With the actual propulsion force of the previous time step obtained in this way, the slope of the previous time step, the velocity increase of the previous time step and the sample time, the actual velocity of the vehicle is determined. Doing so, the velocity of the current time step is calculated as the preceding velocity plus the product of T and the acceleration, which results in the difference of the propulsion force and all the external forces acting on the EV described by the Equations (3.22) – (3.25), divided by m_{tot}. Moreover, it is specified that the actual velocity can't be negative and that the slope and rolling resistance are zero if the velocity is zero. After that, the velocity difference dv is computed as the current velocity minus the velocity of the last time step and the acceleration is

$$a = \frac{dv}{T}. \tag{3.42}$$

The control error and u_{sat} are calculated as described in Section 3.3. In this way, the controller sets the position of the accelerator pedal and the force required to follow the imposed velocity of the driving cycle. As potential cost function of an edge in a graph and also as a transition value to the class *Battery* the mechanical power is determined using the actual force and the actual velocity and the maximum possible motor torque as parameters. The maximum power to be recuperated $P_{re,max}$ for the according value of ω_m was specified as

$$P_{re,max}(\omega_m) = |\omega_m| \cdot M_i(\omega_m), \tag{3.43}$$

according to Figure 3.5. The mechanical power is then computed as the product of the actual force and the actual velocity. If the result is greater than or equal to zero, mechanical power is needed to follow the imposed velocity and so the power withdrawn from the electric motor is set as the quotient of the mechanical power and η_{mech},

$$P_m = \frac{F \cdot v}{\eta_{mech}}, \tag{3.44}$$

since losses occur on the way from the motor to the tyres. Else if it is smaller than zero, mechanical power is gained and multiplied with η_{mech} to get the recuperated power P_{re},

$$P_{re} = F \cdot v \cdot \eta_{mech}, \tag{3.45}$$

as there is power lost on the way to the motor. The power to be recuperated was also defined to be greater than the maximum recuperated power $P_{re,max}$ multiplied with -1 times η_{motor}, because it still has to be transferred over the motor to the battery. In this way, the motor only recuperates as long as it can receive power, which is limited by its characteristic curve. In each of the stated cases the value of the appropriate formula is returned as the demand for mechanical power P_{mech}.

3.5 Model of the Battery and the Power Consumption

The power needed to move the BEV is provided by a rechargeable lithium-ion battery. Furthermore, all electronic components assisting the driver are supplied by it. In this section the implementation of a battery model and the power consumption of all electronic components and the electric motor from the battery is described.

3.5.1 Preliminaries

The principles of electronics used for the battery circuit can be acquired from [20]. The lithium-ion-battery circuit itself was designed using a common *electrical equivalent circuit* describing a simplified model of a circuit. According to [28] the voltage drops due to ohmic losses, described by the ohmic resistance R_s and the voltage U_s, and the double-layer capacitance and relaxation effects, described by the ohmic resistances R_{p1} and R_{p2}, the capacitances C_{p1} and C_{p2} and the voltages U_{p1} and U_{p2}, were considered as illustrated in Figure 3.7. In its idle state when no electricity consumer is connected to the battery, its *open-circuit voltage* is U_0. U_{bat} describes the *pin voltage* of the battery, which is the electric potential difference between its pins. The related differential equations are discussed in Section 3.5.2.

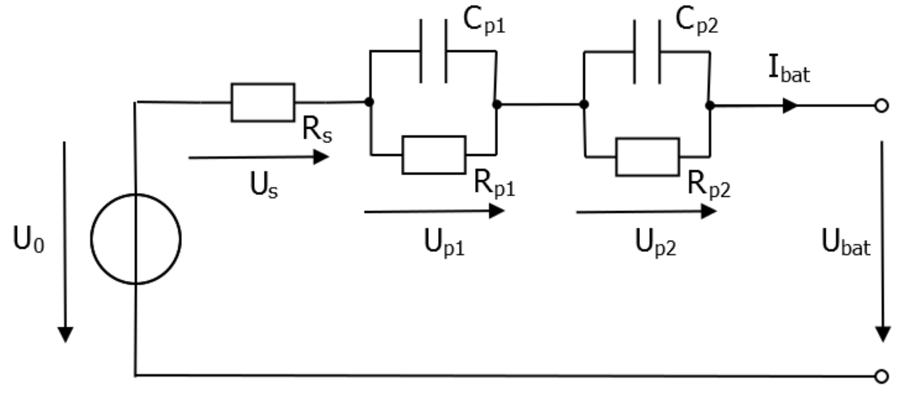


Figure 3.7: Electrical equivalent circuit of a lithium-ion battery

3.5.2 Implementation

To finally identify the SOC the demand for mechanical power P_{mech} established in the last section was used to calculate the power taken from or stored in the battery. When this was done, the current to or from the battery could be obtained so that after all, the SOC could be stated. The management of these tasks was put up into the class *Battery* that models a typical lithium-ion battery of modern BEVs. Also, the inverter between the lithium-ion battery and the electric motor is considered. The lead-acid battery supplying the auxiliaries and the exterior lighting and the DC-DC converter between the batteries are discussed in Section 3.3. In *Battery* the battery power P_{bat} is extrapolated from P_{mech} by applying either

$$P_{bat} = \frac{P_{mech}}{\eta_{inv} \cdot \eta_{360V}} + \frac{P_{HAC} + P_{light} + P_{aux,360V}}{\eta_{360V}} \tag{3.46}$$

if the power demand of the motor P_{mech} is greater than or equal to zero or

$$P_{bat} = P_{mech} \cdot \eta_{inv} \cdot \eta_{360V} + \frac{P_{HAC} + P_{light} + P_{aux,360V}}{\eta_{360V}} \tag{3.47}$$

if P_{mech} is smaller than zero. For a positive power demand of the motor from the battery, power has to be provided and the efficiency factors of the inverter η_{inv} and lithium-ion battery η_{360V} have to be considered. If energy is stored in the lithium-ion battery, the recuperated power, which is the surplus of power in the electric machine, narrows on the way to the battery which is described by a multiplication with the according efficiency factors. These are η_{inv} for the inverter where the AC voltage of the motor is converted to DC voltage when the battery is charged by the motor and η_{360V} for the efficiency of the lithium-ion battery. The power consumption of heating, air conditioning, exterior lighting and the auxiliaries are always positive, only P_{mech} can be negative and therefore supply power for charging the battery. Thus the power demand of the facilities for driving assistance and comfort from the battery is always positive and the efficiency factor of the battery has to be considered accordingly. Since the facilities get their power directly from the lithium-ion battery, the efficiency factor of the inverter is not used to describe their power consumption. Pursuant to [28] the efficiency of the lithium-ion battery was set as $\eta_{360V} = 0.95$. Similarly to [8] the efficiency factor of the inverter η_{inv} was set as 0.98.

The battery current I_{bat} is found by dividing P_{bat} by the battery voltage U_{bat} using the fact that electric, or battery power respectively, is defined by

$$P_{bat} = U_{bat} \cdot I_{bat}. \tag{3.48}$$

The open-circuit voltage U_0 of a battery is defined as the difference of the electric potential measurable between the two pins of the battery before being connected to any electricity consumer. Initially, the open-circuit voltage U_0 of the entire lithium-ion battery is interpolated from the battery characteristic shown in Figure 3.8 for the initial SOC using the class

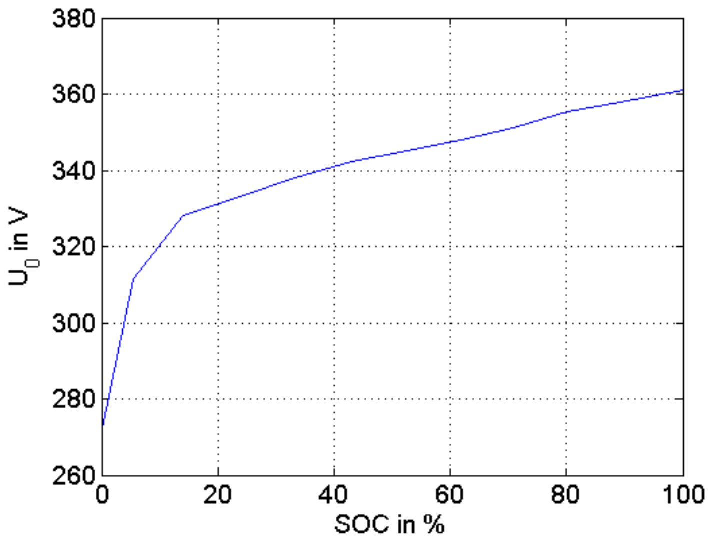


Figure 3.8: Open-circuit voltage of the entire battery pack of the i-MiEV for different SOC values

NumericalMathematics as explained in Section 3.2. As can be seen, U_0 varies for different values of the battery's SOC. Originally, Figure 3.8 is a plot of the characteristic of the open-circuit voltage for different SOC values constructed at the AIT for the GS Yuasa LEV50 battery cell. The original data table contains the voltage according to each SOC value at the inflexion points of the curve. A description of the LEV50 cell and its related battery module LEV50-4 with four LEV50 cells can be found in [30]. Figure 3.9 depicts the battery pack of the i-MiEV and its installed position, taken from [34]. It is put together of 22 LEV50-4

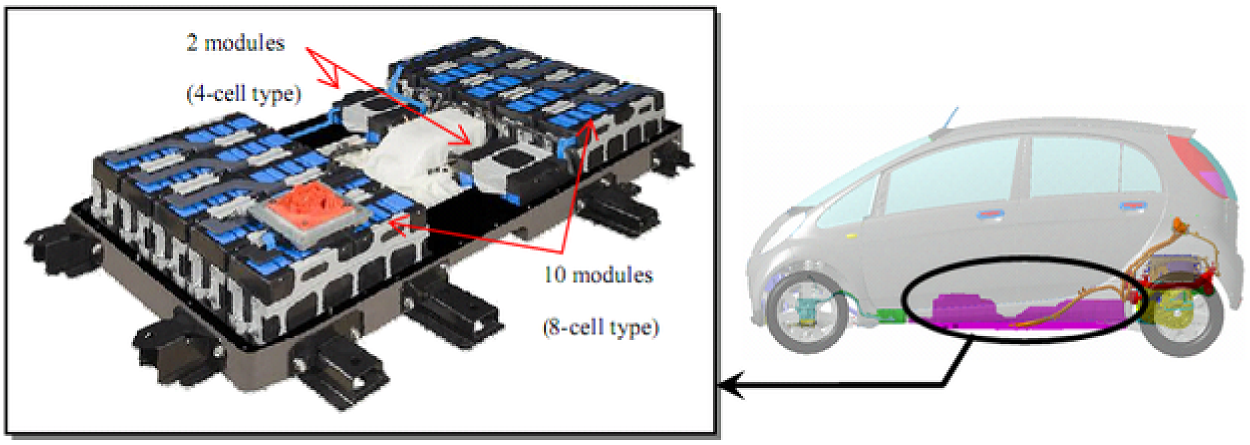


Figure 3.9: Battery pack of the i-MiEV and its installed position, taken from [34]

battery modules connected in series, so it contains 88 LEV50 cells connected in series, as stated in [34]. As can be seen there are two 4-cell modules installed vertically at the centre of the pack and ten more 8-cell modules, each of them put together of two 4-cell modules, placed around.

To work out the state of charge, the charge loss is determined by integrating the current over time using T as interval. It is then divided by the battery capacity C_{bat} and subtracted from the initial SOC. Moreover, the values the SOC can take are limited by zero and an upper limit $SOC_{lim} = 0.995$ given by the AIT as charge-capacity ratio as a precaution such that the battery may not to be overcharged. According to [30] C_{bat} was set as $50\,Ah$.

The battery voltage U_{bat} is calculated in compliance with Figure 3.7. With the current SOC U_0 is interpolated from the data of Figure 3.8. The voltage drop due to ohmic losses at metallic arresters, which conduct the electricity to the end poles when discharging, active material and electrolyte was described by Ohm's law

$$U_s = R_s \cdot I_{bat}. \tag{3.49}$$

The voltage drops U_p due to the double-layer capacitance are described by an ohmic resistance and a capacitance connected in parallel. *Double-layer capacitance* occurs during the charge transfer between the active material of an electrode and the electrolyte due to carriers of different polarisation [28]. For such an RC element the differential equation

$$I_{bat} = \frac{U_p}{R_p} + C_p \cdot \frac{dU_p}{dt}, \tag{3.50}$$

or

$$\frac{dU_p}{dt} = \frac{I_{bat}}{C_p} - \frac{U_p}{R_p \cdot C_p} \tag{3.51}$$

respectively, holds. The two differential equations for the two RC elements modelled were solved numerically using the formulas

$$U_{p1}[i] = U_{p1}[i-1] + \left(\frac{I_{bat}[i-1]}{C_{p1}} - \frac{U_{p1}[i-1]}{R_{p1} \cdot C_{p1}} \right) \cdot T \tag{3.52}$$

and

$$U_{p2}[i] = U_{p2}[i-1] + \left(\frac{I_{bat}[i-1]}{C_{p2}} - \frac{U_{p2}[i-1]}{R_{p2} \cdot C_{p2}} \right) \cdot T. \tag{3.53}$$

This is because of the numeric formula

$$U_p[i] = U_p[i-1] + \dot{U}_p[i-1] \cdot T. \tag{3.54}$$

In the end, the battery voltage results in

$$U_{bat}[i] = U_0[i] - U_s[i] - U_{p1}[i] - U_{p2}[i]. \tag{3.55}$$

Measurements at the AIT revealed $R_s = 92.59838\,m\Omega$, $R_{p1} = 54.85377\,m\Omega$, $R_{p2} = 108.11338\,m\Omega$, $C_{p1} = 84.917\,F$ and $C_{p2} = 1205.695\,F$.

3.6 Parameter Assignments

In this section a list of parameters and their assignments used for modelling the i-MiEV within this thesis is given in tabular form. The columns of the corresponding Table 3.2 hold the partitions to which the parameters can be assigned, the parameters, their assignments and the reference where the values are taken from. Only the basic parameters are listed, by which other variables stated in this thesis are calculated in the program. For example, the power consumption of the auxiliaries at the lithium-ion battery $P_{aux,360V}$ can be found with the parameters P_{aux}, η_{12V} and η_{con} and therefore it is not listed in this section. By contrast, parameters, by which variables used directly by the program were calculated, are stated. For example, the geometric parameters describing the frontal surface area A are listed. This was done in order to let the user keep an overview of the parameters he needs to describe his vehicle and driving style.

Table 3.2: Parameter assignments, meanings of parameter symbols are grouped in Section 8.1

Partition	Parameter	Assignment	Reference
chassis	r_t	$0.285\,m$	[4]
	h	$1.61\,m$	[4]
	w	$1.475\,m$	[4]
	h_c	$0.15\,m$	[4]
	w_t	$0.145\,m$	[4]
	A	$2.2\,m^2$	(3.26)
	c_d	0.35	[36]
	c_{rr}	0.015	[53]
	$m_{chassis}$	$1085\,kg$	[4]
moments of inertia	J_m	$0.02\,kg\,m^2$	AIT measurements
	$J_{g,m}$	$0.015\,kg\,m^2$	AIT measurements
	$J_{g,t}$	$0.015\,kg\,m^2$	AIT measurements
	J_t	$1.23\,kg\,m^2$	AIT measurements
transmission ratios	k_{gear}	7.065	[4]
	k_{diff}	1	empirical
motor parameters	M_{max}	$196\,N\,m$	[4]
	ω_{nom}	$250\,\frac{rad}{s}$	[4]
mechanical brake parameters	F_N	$5\,kN$	AIT data
	μ	0.3	AIT data
	f	2	AIT data
	r_i	$0.08\,m$	AIT data

	r_e	$0.12\,m$	AIT data
battery parameters	SOC_{lim}	0.995	AIT measurements
	C_{bat}	$50\,Ah$	[30]
	U_0	interpolated	AIT measurements
	R_s	$92.59838\,m\Omega$	AIT measurements
	R_{p1}	$54.85377\,m\Omega$	AIT measurements
	R_{p2}	$108.11338\,m\Omega$	AIT measurements
	C_{p1}	$84.917\,F$	AIT measurements
	C_{p2}	$1205.695\,F$	AIT measurements
driver parameters	m_{load}	$85\,kg$	empirical
	a_{ap}	$1.5\,\frac{m}{s^2}$	empirical from WLTC
	a_{dp}	$2\,\frac{m}{s^2}$	empirical from WLTC
	T	$0.1\,s$	empirical
	K_P	0.4	empirical
	T_N	$0.4667\,s$	empirical
	T_V	$0\,s$	empirical
	T_a	$0.1\,s$	empirical
further power consumers	P_{aux}	$140\,W$ if air conditioning is turned off or $450\,W$ if turned on	AIT measurements
	P_{HAC}	$550\,W$	AIT measurements
	P_{low}	$56.2\,W$	[43]
	P_{tail}	$7.2\,W$	[43]
efficiency factors	η_{motor}	0.95	[23]
	η_{gear}	0.97	[23]
	η_{diff}	0.97	[54]
	η_{12V}	0.85	[45]
	η_{con}	0.95	[49]
	η_{inv}	0.98	[8]
	η_{360V}	0.95	[28]
nature constants	ρ	$1.25\,\frac{kg}{m^3}$	nature constant
	g	$9.81\,\frac{m}{s^2}$	nature constant

Chapter 4

A Graph Model for Routing BEVs

The task in the second part of this thesis was to apply the program of the first part to road sections instead of single driving cycles and to produce edge weights for these sections. It is obvious that a driving cycle can be created for every kind of road section that can be followed by the already created driver model to calculate the costs for the BEV. The estimation of costs on road sections is more complicated than it seems to be at first glance. Due to regenerative braking energy is stored in the battery when decelerating the BEV. This makes it important to model the costs it takes to get from one road section to another, whereby the vehicle may get slower or even has to stop. The cost function assigned to the graph is the energy consumption of the BEV on different road sections, represented by edges.

4.1 General Specification of the Graph Model

Compared to a usual graph of a street map, a graph that shall be suitable for BEVs must meet special conditions. Since energy can be recovered by regenerative braking, turn costs defined as the costs it takes to get from one road section to another are an important feature that has to be considered. Therefore, conditions have to be specified that mark different ways to traverse a junction. This was achieved by the generation of a pseudo-dual graph from the primal graph. Figure 2.2 illustrates the procedure to create a restricted pseudo-dual graph from a given primal graph. The procedure is also described in Chapter 2. The primal graph represented by grey arrows and dots is replaced by the pseudo-dual graph represented by black arrows and dots. Doing so, the edges of the pseudo-dual graph store the costs of the edges of the primal graph and the turn costs. As already stated in Chapter 2 turning on road sections without reaching their end vertices was prohibited. For the implementation of turn costs the connection of two primal edges at a junction like the one depicted in Figure 2.2 has to be classified. That means that it has to be declared whether the vehicle has to slow down or even has to stop at the junction. This can be done by comparing the functional road classes of the according edges. Functional road classes describe the importance of edges for long-distance traffic.

Of course other predicates have to be regarded as well. The graph should be able to describe the terrain that is driven on, that is it must contain gradient data. A velocity profile has to be determined, so it is required to contain maximum allowed speed limits. For the output of the routing and visualisation tasks it has to comprise labels for the edges such as street names and coordinates. To estimate the costs, the lengths of the edges have to be known, the street length is also the usual edge-weight value a graph should include. It should also be marked if the edge is a one-way street or not.

In the primal graph the edges have a direction with which they are defined, that is they have a start vertex and an end vertex. For each primal edge a Boolean value exists that indicates whether the edge is directed or not. If it is an undirected edge, the Boolean value is false and to get the appropriate directed edges, such an edge was split into two edges. The start vertex of one of the resulting directed edges is the end vertex of the other one and vice versa. Else if the edge is directed, it is a one-way street and the Boolean value, which also is a specifier of one-way streets, is true. All further edges created in this work are directed, so that one-way streets did not have to be indicated as such any more in the final graph model. The edges of the primal graph contain street names, coordinates, lengths, speed limits, slope data, one-way-street specifications and functional road classes of all edges. Table 4.1 lists the attributes of an edge in the primal graph and gives brief explanations of their meanings. These attributes are essential for an input graph of the presented tool.

Table 4.1: Attributes of the edges of the primal graph and their meanings

Attribute	Meaning
street name	official name of the street which the edge belongs to
geographic coordinates	geographic coordinates of some reference points on a road section given in degrees
functional road class	categorisation of streets by means of their significance for long-distance journeys, given by numbers starting with 0 for the most important category and ending with 10 for the least important category
one-way street specification	Boolean value marking a road section as one-way street if set as true
speed limit	maximum speed allowed on the edge given in $\frac{km}{h}$
length	length of the edge given in m
ascent	altitude difference to be ascended when travelling along the edge into the direction the edge is defined with,

	given as a positive value of m
descent	altitude difference to be descended when travelling along the edge into the direction the edge is defined with, given as a negative value of m

The pseudo-dual edges connect two primal edges considering the turn costs also. The final pseudo-dual graph generated in this work contains the weights used to determine a shortest path, lengths of edges and the time it takes to travel along them as well as the belonging street names and coordinates.

4.2 Generation of Driving Cycles on Road Sections

The class *PseudoDualGraph* was introduced to read in all data of the primal graph, which is the basic input of the presented tool together with the source vertex s and the target vertex t, issuing a shortest path from s to t and the current SOC of the BEV. Moreover, four different procedures describing the transition from one edge to another for creating the pseudo-dual graph were implemented in this class. Instead of reading in a cycle from a file, an artificial cycle for each pseudo-dual edge was constructed assuming that an average acceleration process takes the constant acceleration a_{ap} and an average braking process takes the constant deceleration a_{dp}. These parameters have already been introduced in Section 3.3.2. Using the mentioned procedures, four different kinds of cycles were created. Doing so, a cycle for stopping the vehicle and then accelerating it to the speed limit of the consecutive edge was developed. This simulates a situation in which the vehicle has to traverse a junction with a stop sign or a traffic light for example. Another cycle for accelerating from a certain velocity to a higher velocity describes a change of the speed limit on a road. This is usually indicated by a transition from an edge with a certain velocity to an edge with an other velocity. A cycle for decelerating from a certain velocity to a lower velocity also describes a change of the speed limit on a road. One more cycle for driving on with constant velocity describes junctions that can be passed without a stop because the driver travels along a priority road for example.

In the cycle for crossing a junction with constant velocity the prescribed velocity of the car is set as the speed limit of the according edge. In every run of an inner loop creating the cycle, the variable for time is increased by T. The distance covered on the according street is determined by the integral of the constant velocity over time, resulting in the product of the actual time elapsed while driving on the street and the constant velocity.

The difficulty is exacerbated for the cycles describing an acceleration or a deceleration. The velocity is described by the integral of the constant acceleration a over time with the initial condition, that the velocity at the beginning is the speed limit of the source edge, $v(t=0)=v_1$. This results in

$$v(t) = \int a \cdot dt = v_1 + a \cdot t. \tag{4.1}$$

Integrating the velocity considering the initial condition $s(t=0)=0$ yields the distance $s(t)$ covered on the actual edge

$$s(t) = \int v(t) = v_1 \cdot t + \frac{a \cdot t^2}{2}. \tag{4.2}$$

The acceleration or deceleration process will last until the new speed limit v_2 is reached, that is until

$$t(v = v_2) = \frac{v_2 - v_1}{a}. \tag{4.3}$$

When the time is greater than the result of Equation (4.3), the prescribed velocity stays constant with the value v_2. The distance then has to be computed with the formula

$$s(t) = s(v = v_2) + v_2 \cdot (t - t(v = v_2)). \tag{4.4}$$

The time is increased by T again after each iteration. For the concrete case of accelerating from a lower to a higher velocity, a was replaced by a_{ap} in Equations (4.1) – (4.4). For the case of decelerating from a higher to a lower velocity a was replaced by $-a_{dp}$ in Equation (4.1) and analogous to the Equations (4.2) – (4.4) further cycle data was derived.
Stopping at a junction requires to decelerate from v_1 to zero. The according cycle data is computed analogously to the Equations (4.1) – (4.4) when v_2 is zero.

4.3 Account of the Slope

The class *Cost* was introduced to generate a cost function using the classes implemented in Chapter 3. Moreover, it contains a function determining the slope of an edge accepting the ascent, descent and length attributes of an edge as parameters. As already stated in Table 4.1 the ascent attribute is given as a positive height difference dy_{asc} in m and the descent attribute is given as a negative height difference dy_{desc} in m. These height differences are to be ascended or descended when travelling along an edge into the direction it is defined. It is not known for how long the road section is ascending or descending. Thus, the problem arises that the lengths of the ascending and the descending part of the road section may be assigned disadvantageously. For example, if the half of the length of the road section is defined to be ascending and the other half of the length is defined to be descending, road sections that ascend for a large part in reality may become too steep for a car when using this approach, because the ratio of ascent to length is bigger. Accordingly, a procedure to divide the lengths of the ascending and descending part has to be found, for which ascents

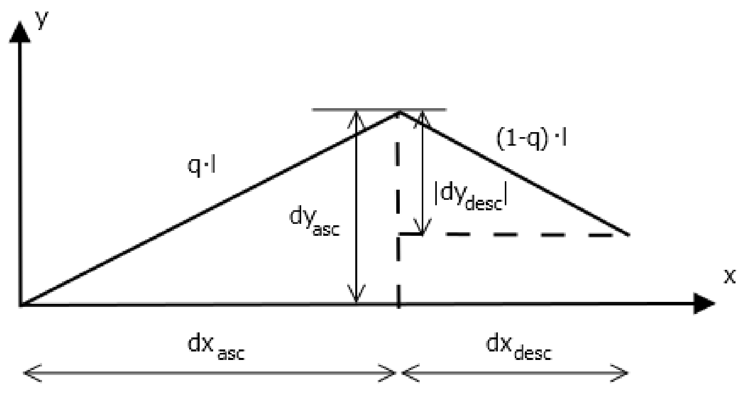


Figure 4.1: Division of the length l of an edge into an ascending and a descending part using the ascent ratio q

or descents do not become too steep. Figure 4.1 illustrates the approach used to resolve this issue and to determine the slope. An auxiliary coordinate system is used to draw in the height differences dy, the length l of the road section and the values dx for the adjacent legs of the resulting gradient triangles. The segments were divided using the ratio for the ascending part

$$q = \frac{dy_{asc}}{dy_{asc} - dy_{desc}}. \tag{4.5}$$

The minus in Equation (4.5) comes from the negative sign of the value of dy_{desc}. The ratio of the descending part is $1 - q$. The length of the ascending or descending part was specified as the product of the total section length l and the ratio of the ascending or the descending part. With the Pythagorean theorem the adjacent leg dx of the gradient triangle can be found as

$$dx_{asc} = \sqrt{(l \cdot q)^2 - dy_{asc}^2} \tag{4.6}$$

for the ascending part and

$$dx_{desc} = \sqrt{(l \cdot (1 - q))^2 - dy_{desc}^2} \tag{4.7}$$

for the descending part. Finally, the slope is established as the tangent of the pitch angle β for each case,

$$\tan\beta = \frac{dy}{dx}. \tag{4.8}$$

Using this approach a reasonable division of the ascending and descending part of a road section is likely to be found, since the length of a part grows with its related height difference value.

There were three cases considered for a road section. First, the section can have an ascending and a descending part. Then, there is a Boolean value that is set as true and that changes to false as soon as the distance covered on the section so far is greater than the product of the length of the road section and the ascent ratio $l \cdot q$. In the preceding time span the slope is regulated as ascending, afterwards as descending. Second, the section is ascending only. The slope is set as ascending the whole time in this case. Last, the section is descending only and the slope is set as descending the whole time. If the result of Equation (4.8) is greater than 0.3, the cost of the according edge is set as infinity. A tangent of 0.3 corresponds to a gradient of about 16.7° which is just too much for the simulated i-MiEV to follow the prescribed velocity profile.

4.4 Implementation of the Graph Model

Specifically for BEVs it is important to consider the energy costs when crossing a junction. It is distinguished whether it is possible to pass without braking or not or if it is even necessary to stop. A general BEV recuperates energy while braking as described and implemented in *Propulsion*. In order to take turn costs at junctions into account in a most possible realistic way, the original graph was converted to a pseudo-dual graph. This was realised by the class *PseudoDualGraph* generating driving cycles on edges as described in Section 4.2. The cycles constructed in *PseudoDualGraph* are handled equally to the read in cycles in the program of the first part of the thesis, meaning that the driver model still has to follow the prescribed velocity. In *PseudoDualGraph*, arrays for every edge variable of the graph data are allocated and filled with the data of the primal graph. The speed limits are converted from $\frac{km}{h}$ to $\frac{m}{s}$ again. The whole graph is then stored in an adjacency list of tuples with all the edge characteristics. An adjacency list holds a row with all vertices and each vertex holds a pointer to a linked list of all its associated or adjacent vertices inclusive of the attributes of an edge. It can for example be implemented using the class *list* from the Standard Template Library (STL). Figure 4.2 exemplary shows a small graph with edge weights, stated as real numbers above the edges representing edge attributes, and the according adjacency list. The list holds all vertices adjacent to any vertex of the edge and the according weight representing the attributes of an edge. Every time an edge is added to the adjacency list, an edge counter

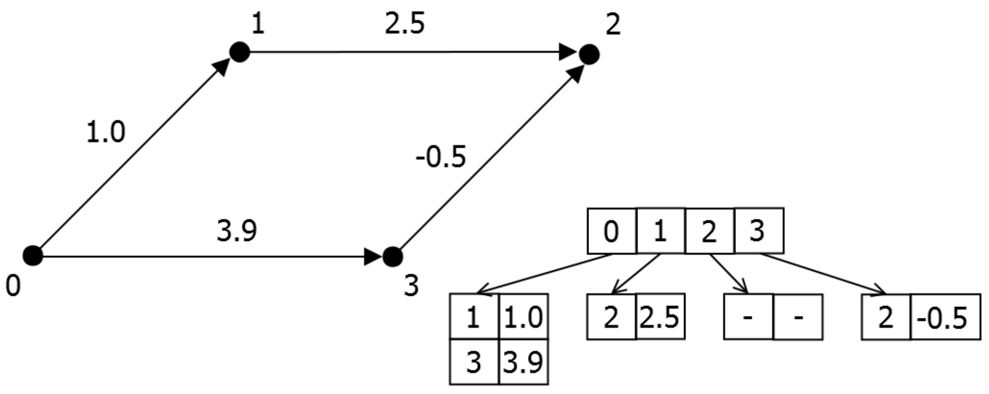


Figure 4.2: Small graph with edge weights and the related adjacency list holding adjacent vertices for each vertex of the graph and the according edge weights

is increased by one. The value of the counter starting with one and counting all edges is set as a new edge ID of each added edge. Additionally, a Boolean value was defined to determine whether an edge as connection from one vertex to another is incoming or outgoing for a certain vertex. In the adjacency list each edge is visible at two vertices, the source and the target vertex. So for its source vertex the edge is outgoing and the Boolean value is false, whereas for its target vertex the edge is incoming and the Boolean value is true. An array is defined that holds the total number of edges approaching or leaving the vertex ID, also called degree of the vertex, at the according index. That means the number of streets meeting at vertex i is stored at position i in the array. If an edge is no one-way street, another edge is created and is stored at the former source vertex as incoming and as outgoing at the former target vertex. So, if an edge of the primal graph is not a one-way street, it is split into two edges. Furthermore, the values for ascent and descent are swapped and multiplied by minus one since the descent is given as negative value initially. In a loop over the vertices of the primal graph the incoming and outgoing edges are stored in separated vectors for each vertex. Each incoming edge is connected with all outgoing edges that can be travelled along theoretically at this point. A turn that leads back to the source vertex of the incoming edge is restricted.

Now the conditions for whether the junction can be passed without stopping are laid down. Doing so, the functional road classes and the degrees of the vertices are used. This is somewhat artificial, since it is unknown if there is a traffic light, a stop sign or something comparable in reality. Anyway, the situations described by traffic lights and stop signs are considered in this work and if there was detailed data about which junctions are regulated by traffic lights or stop signs, the model would be easily adaptable. The functional road classes categorising the streets by means of their significance for long-distance journeys used in this thesis are stated in Table 4.2. The classes are described by numbers starting with 0 for the most important category and ending with 10 for the least important category. They are listed in the first column, the second holds the road types belonging to the classes.

Table 4.2: Functional road classes and the belonging road types

Class	Road Type
0	Motorway
1	Trunk Road
2	Primary Road
3	Secondary Road
4	Tertiary Road
5	Unclassified Road
6	Residential Road
7	Track

8	Pedestrian Ways or Shared Spaces
9	Pedestrian Ways and Cycle Paths
10	Stairs

The junction is assumed to be passed when the degree of the according vertex is smaller than three, meaning that the vertex is no junction at all but just a point where the speed limit changes for example. Moreover, it was defined to be passed when the degree is smaller than four and the incoming edge has a more important functional road class than the according outgoing edge. From and to Motorways and Trunk Roads the BEV is also dictated to drive on without stopping. From functional road class two or three respectively to classes greater than three or four respectively and from functional road class four or five respectively to classes greater than four or five respectively the BEV is regulated not to stop either. If the BEV does not stop at a junction, it either holds its current velocity, decelerates to a lower or accelerates to a higher velocity, depending on the speed limits of the edges.

Obviously the pseudo-dual graph $D(V_D, E_D)$ uses the edges of the primal graph as new vertices, the new vertex number n_D of the set V_D is found by the edge counter described above, that is increased by one every time an edge is added to its source and target vertex. The formation of V_D and E_D can also be observed in Figure 2.2. The new edge number m_D of the set E_D is determined by the size of the vectors that are used to store tuples with the data of the new edges after declaring the conditions for passing a junction. This information is also stored in these vectors as a Boolean value which is false if the junction can be passed and true if the vehicle has to stop. There is a vector holding all the source edges and another vector holding all the target edges. Together they contain all relevant data.

For future use the primal vertices are stored in a text file together with the IDs of all their incoming and outgoing edges and the belonging data. Each of the resulting weighted pseudo-dual edges contains the cost for traversing a junction and travelling along the following street. In this way, the costs of a route can be described completely, except for the energy used for accelerating from a certain starting point and stopping at the destination. The vertices and their in- and outdegrees are used in the graph algorithm for this purpose later on.

In *Cost* all presented classes are instantiated in the constructor. In a specific procedure the implemented functions are applied to compute the costs similar to the main function of the energy-consumption model, with the exception that only the energy consumption on the road segments is calculated and not the SOC. Doing so, the same strategy as in the former model of the BEV is used up to the calculation of the mechanical power, that is then multiplied with T to obtain the energy consumption. It is computed separately for each way of crossing a junction. In addition, the slope is determined with the according data.

Before calculating the cost for an edge the variables of the controller are reset and the initial velocity is fixed as the speed limit of the source edge or as zero, contingent on the distinguished ways for crossing a junction. Doing so, the auxiliary functions $f(0)$ and $g(0)$ as well as the control error were set as zero. The initial accelerator position was found using the accessible torque of the motor at the current velocity and the current resistance force applying on the i-MiEV consisting of air drag, slope resistance and rolling resistance. The actual motor torque M is determined by the multiplication of the position u_{sat} of the accelerator pedal represented by a value between 0 and 1 and the accessible torque of the motor M_i at the current speed. Thus, according to Equation (3.40) the pedal position can be found as

$$u_{sat} = \frac{F \cdot r_t}{\eta_{mech} \cdot k_{mech} \cdot M_i}. \tag{4.9}$$

The conditions for the controller have changed compared to the program of the first part of this thesis following the velocity profile of a single driving cycle. Now it can be better adapted to the situations the BEV has to deal with. An attentive driver was modelled with different controller settings for accelerating, braking and driving with constant velocity. The according parameters are listed in Table 4.3. The first column states the procedures modelled by the settings, the following columns give the numerical values of the controller parameters. The parameters were found with the approach already used and explained in Section 3.3.2.

Table 4.3: Controller settings for energy-efficient driving

Procedure	K_P	T_N	T_V	T_a
acceleration	0.65	$0.5\,s$	$0.1\,s$	$0.1\,s$
braking	0.15	$0.3\,s$	$0.2\,s$	$0.1\,s$
constant velocity	0.5	$0.4\,s$	$0\,s$	$0.1\,s$

Depending on the occurring situations for crossing a junction and covering ascending and descending parts of the edge, the computation procedure for the mechanical energy and with that the cost of the edge is executed for the appropriate driving cycle. In the procedure for calculating the costs there is a loop over all edges of the graph. Inside, each edge is evaluated whether the belonging junction is assumed to be passed or the vehicle has to stop. In the latter case three procedures are used for determining the weight. These will be described next.

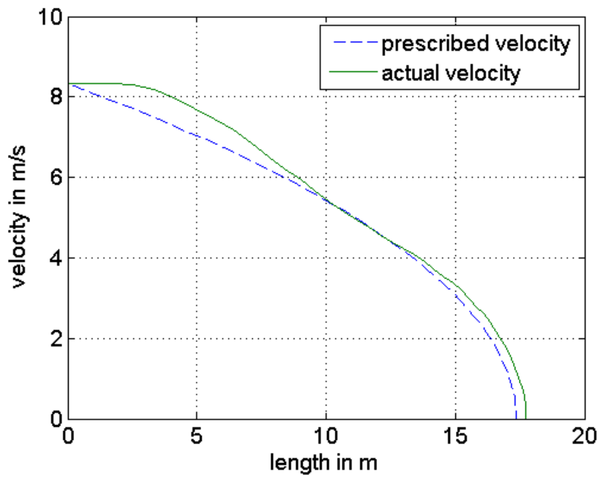
First, the model of the driver follows the cycle for decelerating to zero with the slope and velocity data of the source edge in the according inner loop, that is the BEV stops before the next road section begins. This usually leads to negative costs for a long deceleration phase. If the braking distance exceeds the length of the edge, formerly computed analogously to Equation (4.2) at $t = t(v = 0)$, it is assumed that the mechanical brake is used and therefore there is no recuperation of energy. Correspondingly, the cost is fixed as zero.

Second, if the cost for stopping was not set to zero, that is if the length of the source edge is greater than the distance needed to stop, for this distance the cost for driving with the constant velocity of the speed limit of the source edge is calculated in the according inner loop where the cycle is created and subtracted from the total edge cost. This is done because usually, only costs for the junction and the following street are considered, but here the stopping procedure is counted to the preceding road section and the energy consumption yet calculated for this edge is corrected. Therefore, the drive with constant velocity at the end of the road section, already assigned to the preceding edge, is replaced by the deceleration costs. The controller was not activated since it was assumed that the driver is able to drive with constant velocity after the acceleration process, and also to stay conservative in the consumption estimation. Furthermore, using the functions of *Driver* would take more runtime for the program.

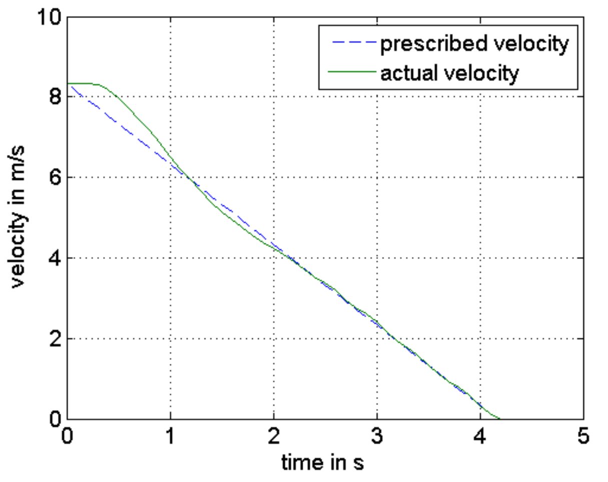
Third, the cost for following the presented cycle for accelerating from a velocity of $0\,\frac{km}{h}$ to the speed limit of the succeeding street are calculated in the according inner loop and added to the total edge cost.

Figure 4.3 shows the velocity profiles of the constructed cycles in comparison to the actual velocity of the BEV according to the described procedure for an edge. The cost for driving the cycle of Figure 4.3c is subtracted from the cost for driving the cycle of Figure 4.3a and the cost for driving the cycle of Figure 4.3d is added. The cycle of Figure 4.3d is driven on the second edge of the two consecutive primal edges the pseudo-dual edge describes. It has a length of $85.68\,m$. The cycles of Figures 4.3a – 4.3c are driven on the first edge. As can be seen in Figure 4.3a the actual distance needed to decelerate to zero is a little bigger than the calculated distance to stop. This is because the controller regulates the velocity over time and not over the distance. Since only the costs and not the actual position of the BEV are affected, there is no great problem with that. In Figure 4.3b the velocities are depicted over time to show that the controller works well.

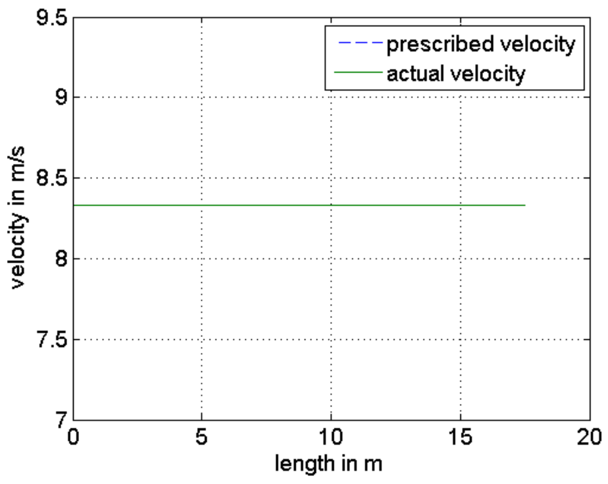
A similar procedure is carried out without considering the preceding edge if the junction can be passed and the speed limit of the preceding street is lower than the speed limit of the succeeding one. While the actual distance is smaller than the total length l of the edge, the driver model follows the cycle for accelerating from the lower to the higher velocity again using an inner loop, causing an energy loss that is computed as already described. The length of the edge is extended to the distance needed to accelerate to the speed limit on the street if l is smaller than this distance. There would be a velocity jump between some streets without doing so, because the succeeding edge starts with the speed limit of the actual street that would never have been reached then. If this exception occurs, the slope is set as zero when the actual distance is greater than l if the actual slope is negative. Else, if the actual



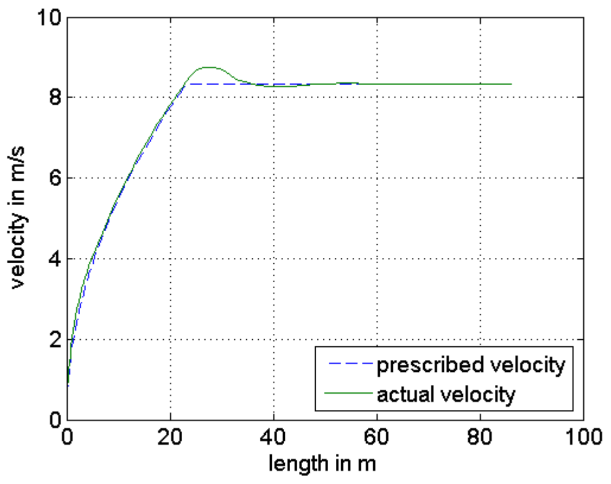
(a) prescribed cycle and actual velocity for stopping



(b) prescribed cycle and actual velocity for stopping over time



(c) prescribed cycle and actual velocity for driving the length needed to stop with constant velocity on the edge where the stopping procedure is performed



(d) prescribed cycle and actual velocity for accelerating

Figure 4.3: Different velocity profiles established for a BEV crossing a junction with a traffic light or a stop sign

slope is positive, its value stays the same. This is also a measure to stay conservative. Up to here, some exception handling had to be done using the given graph. In Section 6.2 some numbers are given for how often such exceptions occur. Lots of them could be avoided if the related problems were considered when making up a graph for routing in subsequent works.

Another procedure was established for decelerating from a higher to a lower velocity without stopping at a junction, following the cycle for passing a junction without stopping in an inner loop. Here, no exception handling for too short edge lengths is done, because it was assumed that the mechanical brake has to be used in such a case, so that energy is not recovered in form of negative costs.

If the speed limits of two connected edges are equal, the junction can be passed without a halt. Since the accelerator pedal is adjusted accordingly at the beginning of such an edge, the control error will only be different from zero when the slope changes somewhere in the middle of the edge. If this is the case, the velocity will slightly increase or decrease depending on the sign of the slope and then return to the speed limit of the edge again.

For the shortest path algorithm a text file was created containing the number of edges and vertices and the edge IDs of the connected edges as well as the energy-consumption values for travelling on the pseudo-dual edges. Additionally, the original length l of the second primal edge visited when travelling along a pseudo-dual edge, the time spent for driving on this primal edge, the street name and the geographic coordinates of this primal edge were stored for each pseudo-dual edge. The creation of a text file takes time but is also useful for structuring, since the included data does not have to be computed again. The user can be asked whether the data shall be recomputed with different parameters or the already existing data shall be used to find the most energy-efficient route.

Chapter 5

The Shortest Path Algorithm

A shortest path algorithm usually takes a graph, a source vertex s and a target vertex t as input and searches for a shortest path from s to t. As stated in Chapter 2 the shortest path is the path with minimal cost to get from s to t, so a shortest path may be a path that minimises distance, energy consumption, road tolls that have to be paid or other criteria depending on the definition of the cost function c. Dynamic graph algorithms that update the graph they get as input exist, but since there were no compelling reasons for using such an algorithm, none was implemented for this would go beyond the scope of this work. Instead, a graph algorithm was implemented that routes the BEV model defined in Chapter 3 on the graph created in Chapter 4.

5.1 Concepts Used in the Shortest Path Algorithm

The general concepts of Dijkstra's, Johnson's and the Bellman-Ford algorithm are discussed in Section 2.2. Applying the Bellman-Ford algorithm to a graph, negative cycles can be found by extending the algorithm with a loop over all edges of the graph, after the actual process for finding the shortest path is completed. Inside the loop it is searched for a path that is shorter than the path just found with the same method as in the original Bellman-Ford algorithm. If a shorter path is found, then there is a negative cycle in the graph. The Bellman-Ford algorithm implemented in this thesis computes shortest paths with at most n edges in the n^{th} step. In a simple path without cycles there are maximum n edges, so if in a $(n+1)^{th}$ step there is a shorter path found, the graph contains a negative cycle. The idea for such a loop is noted in [1] and [26]. As proposed in [26] additionally, the predecessor vertices of the vertex for which the first shorter path was found are added to a string until one of the vertices is visited the second time and thus marking a negative cycle. Then the string is written to a text file listing the negative cycle. The vertices are marked as visited or not using an array with the vertex IDs as indices filled with Boolean values. A true Boolean value at the i^{th} index marks vertex ID i as visited.

When adding vertices to a graph with shifted costs and connecting it with edges, the costs of these edges can be shifted retrospectively in certain circumstances, such that the shifted costs of the other edges are still valid. As long as the connection edge of an added source vertex is weighted positively and there is no edge leading to the source vertex, the vertex potentials of all other vertices remain unchanged. This was suggested by the fact that the potential of such an added vertex is zero, such that a positive weighted connection to another vertex cannot create the unique shortest path to this vertex. An acceleration from zero to a certain positive velocity practically never causes a negative cost. The case of an initial acceleration causing a negative cost can be prevented by setting the cost as zero if it is actually smaller than zero. When adding a destination vertex where no edges outgoing to other vertices exist, the potentials of all other vertices remain the same.

5.2 Implementation of the Shortest Path Algorithm

Since edges can have negative values for the energy consumption used as weight, Dijkstra's algorithm cannot directly be applied to the graph generated in Section 4.4. The Bellman-Ford algorithm could be used, but its runtime is just too long for a rooting program to work out shortest paths, applying it to the pseudo-dual graph of Vienna. Thus, Johnson's algorithm, which is introduced in Section 2.2, was implemented. The draft shall be explained in detail here. The implemented algorithms were inspired by [5], [1], [3] and [33] for a large part, they were improved by own ideas and concepts from [33] and [26].

All corresponding procedures were implemented in *GraphAlgorithm*. In the first place, a Bellman-Ford algorithm was used to shift the costs such that no negative value was left as illustrated in Section 2.2. To store the graph a simple structure was used holding the edge IDs of the pseudo-dual graph, which correspond to the order in which they were written to the text file. Each pseudo-dual edge stores its source-edge ID as source vertex, its target-edge ID as target vertex and its weight. These variables are read in using a procedure for potential shifting, generating a graph with n vertices and m edges. Even though a pseudo-dual graph is considered, the notions used for general graphs are taken to describe the used graph, because the algorithm can be used for any graph and the notation is easier. The values for n and m stem from the read in text file. The shortest path from a newly defined vertex with ID zero to all other vertices is worked out using the Bellman-Ford algorithm. This algorithm is implemented as a procedure that takes an array of size $n+1$ as parameter to store the total weight of every shortest path from the source vertex to each existing vertex. These weights are used as vertex potentials to shift the costs.

Another task of the Bellman-Ford algorithm was to detect negative cycles. The sum of the edge weights of a negative cycle is negative, meaning that it would be possible to completely recharge the battery of the BEV when driving around the cycle repeatedly. Apart from the fact that this is violating fundamental physical laws, the Bellman-Ford algorithm does not assess the correct result if this is the case. Thus, the graph was scanned for negative cycles using the approach described in Section 5.1. With the resulting text file negative cycles and with them mistakes in the algorithms could be found and corrected without tedious error search. Once all errors in the graph and the shortest path algorithm had been eliminated there was no negative cycle found in the final version of the program.

The positive costs are calculated according to Equation (2.9) and together with the vertex potentials found with this procedure they are stored in a text file, such that the following algorithm of Dijkstra can be run separately. Depending on the user request the calculation of the costs and the potential shifting can be performed or not. The potential shifting is executed within a procedure for all the routing tasks if the user decides to recalculate the edge costs. This emphasises the purpose of Johnson's algorithm to apply Dijkstra's algorithm for routing tasks without changing vehicle and driver parameters.

To succeed with the routing applying Dijkstra's algorithm the graph with all the belonging data including the shifted costs is read in. The IDs of the source s and target t are passed to the procedure for the routing tasks as parameters. Moreover, the degrees of the primal vertices are read in together with the newly defined edge IDs of the primal graph and the corresponding names, speed limits, lengths, ascents, descents and coordinates. This data is stored in a list array for the primal vertex IDs, such that at the index i the list holds all incoming and outgoing primal edges with the related data in tuples. Now, virtual edges are added to complete the route with the ways to the source and target vertices. This has to be done because the routing on a pseudo-dual graph results in a path that goes from a certain primal edge to another primal edge. The actual source and target vertices then have to be connected to these edges. As input parameters the procedure knows the IDs of s and t. So the original weights for all edges going out of s and all incoming edges at t can be calculated with the read in data. On all edges outgoing from s the cycle for accelerating according to Equations (4.1) – (4.4) with $v_1 = 0\,\frac{km}{h}$ is followed by the driver model and the emerging costs are calculated. The resulting costs were assumed to be positive or negligibly small according to amount, since an acceleration from $0\,\frac{km}{h}$ to a positive velocity usually produces a positive power consumption. To invalidate this assumption, larger gradients than these occurring in the considered graph would be required. Thus, for reasons described in Section 5.1 the cost is set as zero if it is actually smaller than zero as an assurance that the cost of the first edge is positive. On all edges incoming at t the already described cycle for stopping is used for the evaluation of the energy consumption. Since the last edge of a route leads to t by nature, the negative cost for stopping the vehicle is added to the cost of each edge that may be the last one. As a correction the cost for driving the distance needed to stop with constant velocity is subtracted analogously to the processes presented in Section 4.4. The just calculated weights are shifted according to Equation (2.9). The justification for this approach is defended in Section 5.1. Since the potentials for the added vertices are not known yet, they still have to be determined. Up to here, the IDs of the vertices start counting from one. The source vertex gets the ID zero and the target vertex the ID $n + 1$. As already mentioned, the potential of the vertex zero is zero. The potential of the added target vertex was found as the minimum of the sums of the potentials of the different vertices that have a connection to the added target vertex and their weights.

At this time, the final graph of $n + 2$ vertices can be passed to Dijkstra's algorithm finding the shortest path from vertex zero to vertex $n+1$. Dijkstra's algorithm is implemented using early stopping as described in Section 2.2. Analogously to the parent vector for storing the predecessors in the shortest path to a vertex, vectors for the distance covered, the original energy consumption and the shifted weights, time spent on the individual edges, street names and coordinates are introduced and handled. For the output of the result a recursive function was implemented that starts with the vertex ID of t and calls itself recursively with the ID of the predecessor vertex in the shortest path until the street name of the vertex is the name of the source vertex. Then the function returns all data of the shortest path from s to t that was stored on the function stack during the recursive calls. The idea for such a recursive function is noted in [21] and [15]. Also the SOC after each traversed street is computed in this function using the original, not shifted weight and the time spent on the appropriate edge as parameters for a procedure designed for this purpose in *Cost*. The procedure applies the functions of the now used class *Battery*. P_{bat} is computed according to the Formulas (3.46) and (3.47) using the quotient of the edge weight and the time spent on the edge instead of P_{mech}. The energy consumption on an edge is obtained by multiplying P_{bat} with the time spent on it. For being able to state the total energy consumption of a route, the energy consumption values of each edge travelled along when driving the path are summed. In the program of the first part of this thesis P_{mech} has a different value on every sample point of the driving cycle. Now, P_{mech} as the ratio of the edge weight to the time spent on the edge represents the power consumption for the whole driving cycle on the according edge. Since in *Battery* P_{bat} and with that I_{bat} also are calculated using P_{mech} as input parameter, I_{bat} has a constant value on an edge. Thus, instead of computing the charge-loss integral using the trapezoidal rule, the charge loss is now determined as the product of the battery current and the time spent on the according edge, because an integral method using more grid points would be useless in this case. Using this assumption, the current can no longer be seen as a discretisation of a continuous function and so the model for calculating the battery voltage according to Equations (3.55) and (3.51) makes no sense any more. Instead, the voltage drops U_p at the RC elements are neglected in the model and the battery voltage is now determined as

$$U_{bat}[i] = U_0[i] - U_s[i]. \tag{5.1}$$

Everything else in the usage of *Battery* remains the same. A Boolean value was introduced to check if the SOC gets less than zero somewhere along the path. If this is the case the algorithm returns the information that the desired destination can't be reached with the current SOC. The output format of the final result is presented in Section 6.3.

Chapter 6

Results and Discussion

The results are divided into the outcomes of the different parts of this thesis described by the previous chapters. The result of the second part is the pseudo-dual graph and was already discussed as input for the third part. Generating the graph it was shown how the outcomes of the realistic consumption model can be used as cost functions for its edges. Some statements about the primal graph as input for the tool routing the i-MiEV are given in Section 6.2. Hence, the outcomes of the evaluation of the energy consumption of the i-MiEV and the findings of the implemented shortest path algorithm remain to be discussed. Doing so, the model of the BEV, which claims to be as complete as reasonable, is discussed separately from the graph algorithm. The latter was introduced to show which criteria routing algorithms for a BEV have to meet. Several extensions of the graph algorithm would be conceivable, some of them are addressed in Chapter 7.

6.1 Validation of the Energy-Consumption Model

As the main contribution to the current literature, the energy consumption model of the BEV should generate more realistic results. To attest them, all parameters of the BEV have been adjusted to the i-MiEV according to the cited sources as described in the preceding chapters.

Measurement runs with the i-MiEV imitating different driving cycles performed by the AIT on a roller-test bench have been plotted together with the results of the presented tool. Doing so, text files describing a Worldwide harmonized Light-Duty vehicles Test Procedure (WLTP) and a New European Driving Cycle (NEDC) were used. These cycles were designed to assess fuel consumption and CO_2 emission of passenger cars with internal combustion engines. The cycles are also used to determine the range of conventional vehicles and BEVs. The according files were provided by the AIT.

The definition of the NEDC and its test procedure can be found in [47]. Accordingly,

$33.6\,km$ have to be covered in $1180\,s$ at $20 - 30\,°C$. There are two phases in the NEDC, the first phase reflects urban driving and consists of four so called *Urban Driving Cycles*, each lasting $195\,s$ for covering a distance of $994.03\,m$. The second phase shall simulate driving modes with higher speed outside the urban area. The total cycle is supposed to reflect the typical behaviour of drivers in Europe with modern cars. However, it was often criticised for being artificial and giving unreal results. For example, in [35] it is described as not being able to represent real-live driving conditions, considering low accelerations, constant speed cruises and idling events for the most part. Also in [29] the NEDC is criticised for its low accelerations and for not being able to represent real-live driving conditions.

As successor of the NEDC the WLTP shall correct the defects of the NEDC. The definition of the WLTP can be found in [46]. It contains three different cycles modelling different classes of vehicles with differing accelerations and speed curves. The Worldwide harmonized Light-Duty vehicles Test Cycle (WLTC) for vehicles of Class 1 was constructed for low power vehicles with a power-weight ratio smaller than or equal to $22\,\frac{kW}{t}$, the WLTC for vehicles of Class 2 for medium power vehicles with a power-weight ratio smaller than or equal to $34\,\frac{kW}{t}$ but greater than $22\,\frac{kW}{t}$ and the WLTC for vehicles of Class 3 for high power vehicles with a power-weight ratio greater than $34\,\frac{kW}{t}$. Since in [4] for the i-MiEV the maximum power of the electric motor is stated as $P_{max} = 49\,kW$ and the weight is stated as $m_{chassis} = 1.085\,t$, the WLTC for vehicles of Class 3 was the cycle to choose. The WLTC for vehicles of Class 3 consists of four parts for low, medium, high and extra high speed. The maximum speed in this cycle is $131.3\,\frac{km}{h}$, the total cylce lasts $1800\,s$ and covers a distance of $23.262\,km$.

Figure 6.1 shows the velocity over time, prescribed by the file data of the NEDC and the actual velocity of the car according to the C++ program. From the extract of the NEDC shown in Figure 6.2 it is easier to see the deviations of the actual from the prescribed velocity. The section that is zoomed in is also framed in Figure 6.1. It can be seen that the actual velocity controlled by the driver is not as edgy as the original data of the NEDC, but the driver is able to follow the prescribed velocity well.

Besides the comparison of NEDC passes, also a WLTC for vehicles of Class 3 has been run. The associated velocity over time of file data and driver model is shown in Figure 6.3. Again, an extract of the cycle framed in Figure 6.3 is shown in Figure 6.4 for a better observation of the deviations. The original data of the WLTC for vehicles of Class 3 seems to be more realistic than the data for the NEDC, since it is not as edgy. There are no long phases of constant speed any more and the accelerations are not as low. Again, the simulated driver is able to follow the prescribed velocity well.

Regarding the calculations only, that means neglecting the times for reading from files, declaring and initialising variables and writing to files, the presented tool needs $31\,ms$ for processing two consecutive NEDCs with a total duration of $2360\,s$ and a sample-time interval

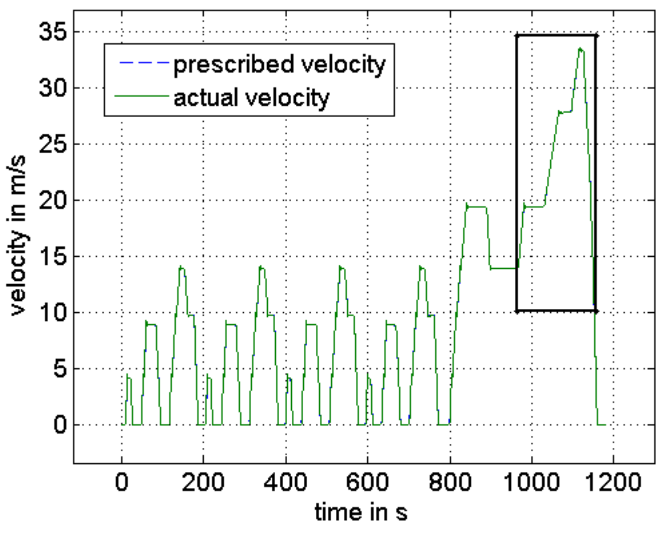


Figure 6.1: The NEDC and the implemented driver model trying to follow its prescribed velocity over time

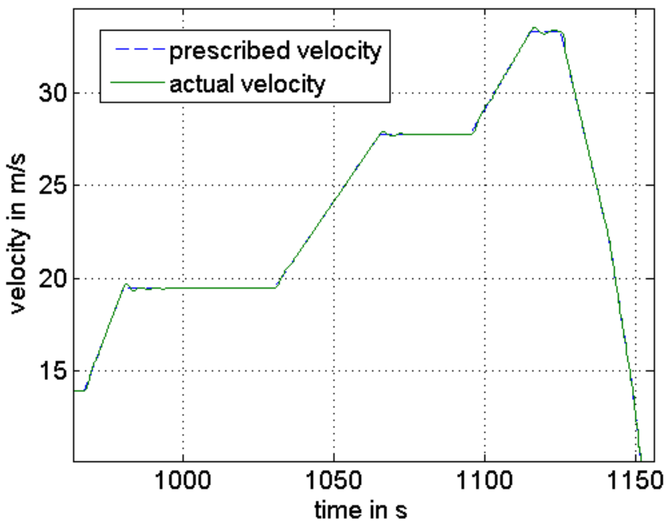


Figure 6.2: Extract of the NEDC and the implemented driver model trying to follow its prescribed velocity over time

of 0.1 s running on a computer with an Intel® Core^{TM} i3 CPU with 2.20 GHz. The C++ program takes 2.75 s in total, issuing a file with all data to plot. Time measurements were taken within a Windows® 10 Home system. The employed C++ compiler was the Microsoft® C/C++ optimising compiler version 18.00.31101 for $x86$. The runtimes of the preprocessing become important when a dynamical graph algorithm is used to do routing tasks using a graph varying in time.

The Figures 6.5 – 6.9 contrast the plots of the tool outputs and the data from the measurement run for driving two consecutive NEDCs when the air-conditioning system is turned

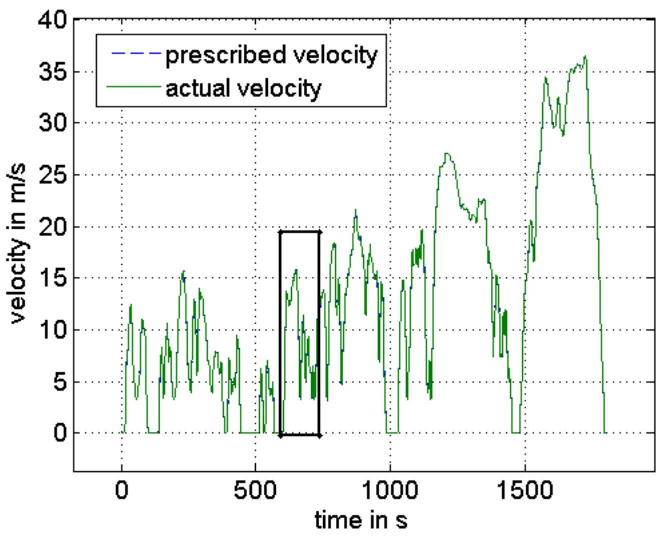


Figure 6.3: The WLTC for vehicles of Class 3 and the implemented driver model trying to follow its prescribed velocity over time

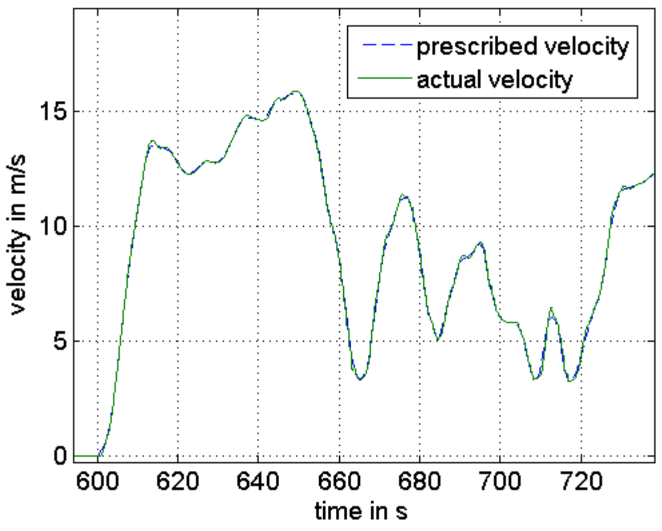


Figure 6.4: Extract of the WLTC for vehicles of Class 3 and the implemented driver model trying to follow its prescribed velocity over time

on and the exterior lighting is turned off. In Figure 6.5 the battery voltage over the time needed to complete two consecutive NEDCs is depicted. Since it is difficult to distinguish the voltages of the measurements and the output of the C++ program, Figure 6.6 shows the voltage deviations of the program output from the measurements at the appropriate times. For the most part, the deviations are low. Though, there are some spots where the deviations show large fluctuations up to about $\pm 10\,V$. Discretisation errors may play a role for the deviations. At the places of the large deviations, the velocity profiles of the program output and the measurements show some striking differences, which probably is the reason

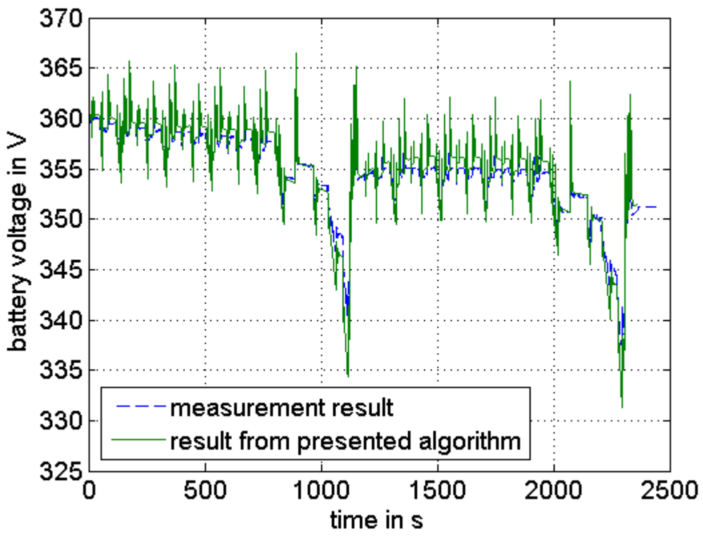


Figure 6.5: Comparison of the battery voltage over time between the C++ program and NEDC-measurements

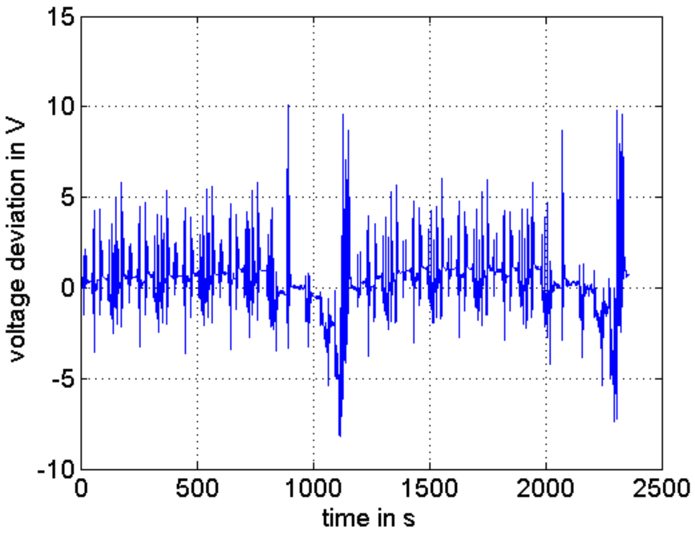
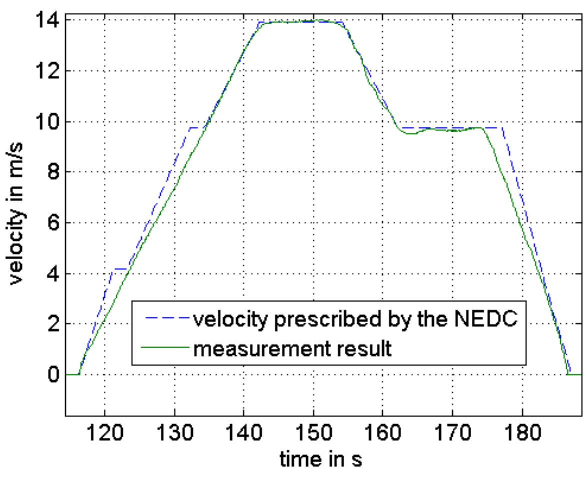
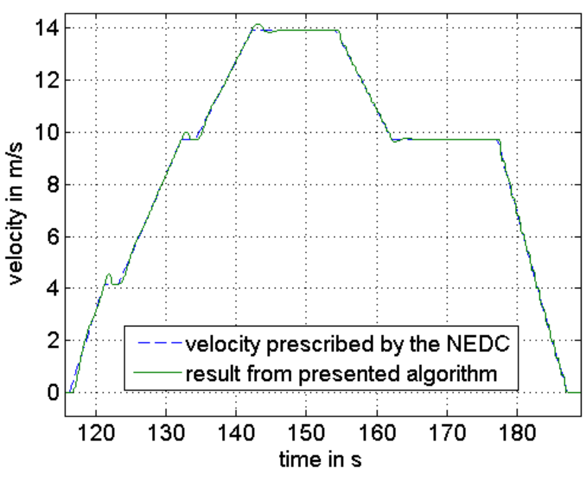


Figure 6.6: Voltage deviation of the output of the C++ program from the NEDC-measurements

for the large voltage deviation at these positions. On the one hand, the controller simulating the driver could not be set perfectly such that it reflects the behaviour of a real driver on highest level of detail. On the other hand, the measurements have noteworthy deviations from the original NEDC data. Figure 6.7 illustrates this issue. As can be seen, the driver modelled in the C++ program can follow the prescribed velocity more precisely than the real driver. Thus, voltage deviations were caused due to this fact, which does not mean that the battery model is invalid. Figure 6.8 shows the two different velocity profiles of the output of the C++ program and the measurements in a time interval where there are large



(a) real driver following the velocity over time prescribed by the NEDC performing measurements



(b) driver model implemented in the presented program following the velocity over time prescribed by the NEDC

Figure 6.7: Velocity profiles resulting when the simulated and the real driver follow the velocity prescribed by the NEDC

fluctuations in the voltage deviation depicted in Figure 6.6. It can be seen that there are

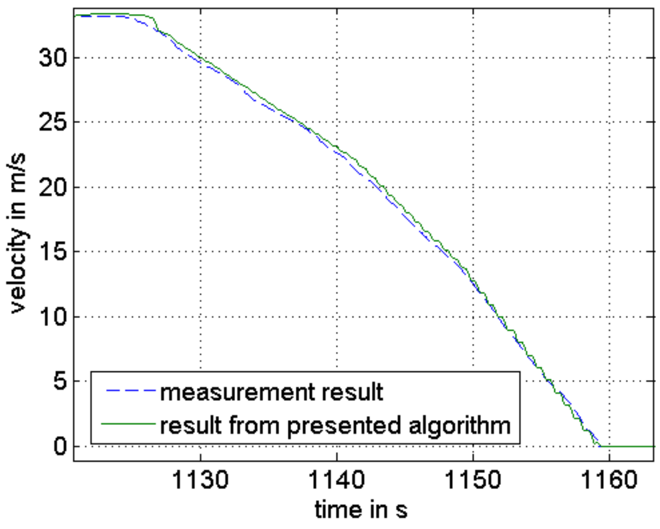


Figure 6.8: Extract of the velocity profiles of the output of the C++ program and the NEDC-measurements

noteworthy differences in the profiles that can explain the high voltage deviations. When decelerating, the simulated driver following the prescribed velocity has low variations in its velocity that probably produce the fluctuations in the voltage deviation.

Figure 6.9 shows the SOC over time issued by the C++ program and the measured SOC over time for two consecutive NEDCs. As can be seen, the presented tool estimates the SOC

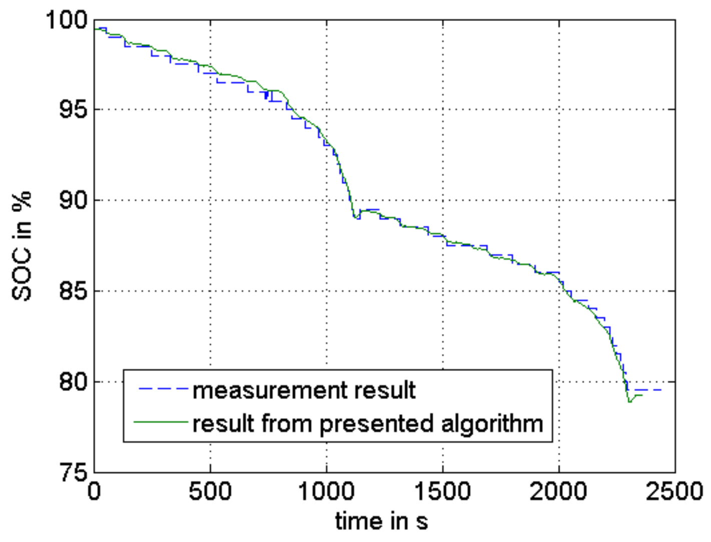


Figure 6.9: Comparison of the SOC over time between the C++ program and NEDC-measurements

in a realistic way, slightly underestimating it as intended.

The Figures 6.10 – 6.13 contrast the plots of the tool outputs and the data from the measurement run for driving a WLTC for vehicles of Class 3 when the air-conditioning system is turned on and the exterior lighting is turned off. In Figure 6.10 the battery voltage is depicted over the time needed for the run of a WLTC for vehicles of Class 3. Again, a look

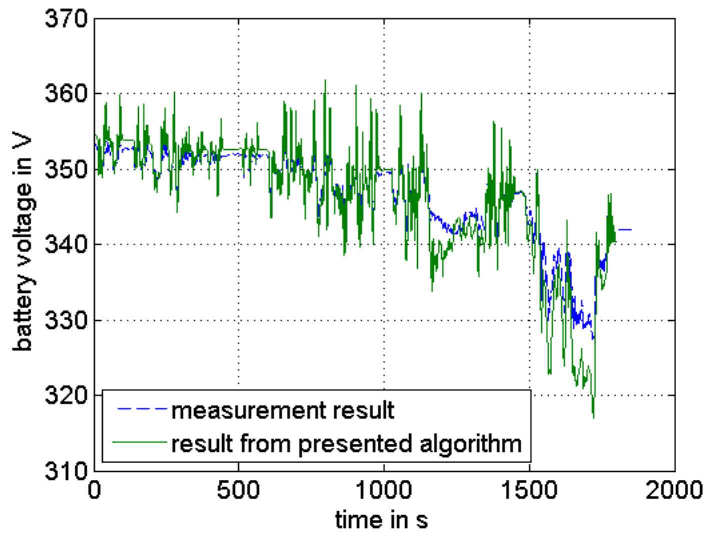


Figure 6.10: Comparison of the battery voltage over time between the C++ program and WLTC-measurements for vehicles of Class 3

at the voltage deviations over time depicted in Figure 6.11 gives more transparency. It looks similar to Figure 6.6. A representative area for the voltage deviation is located in the time

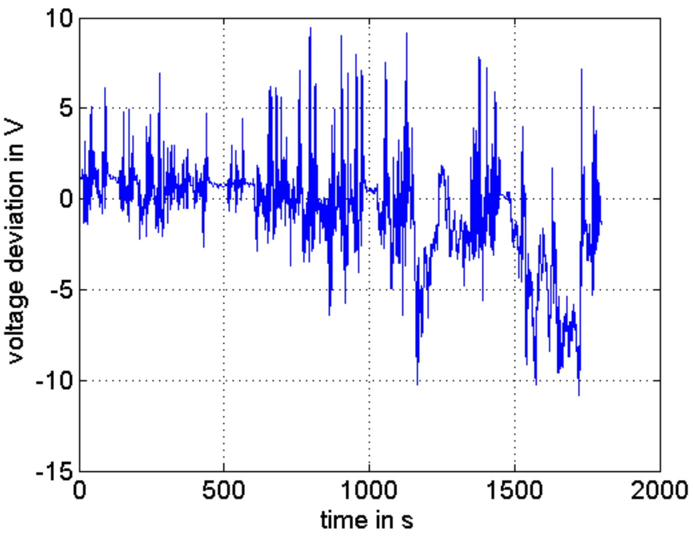
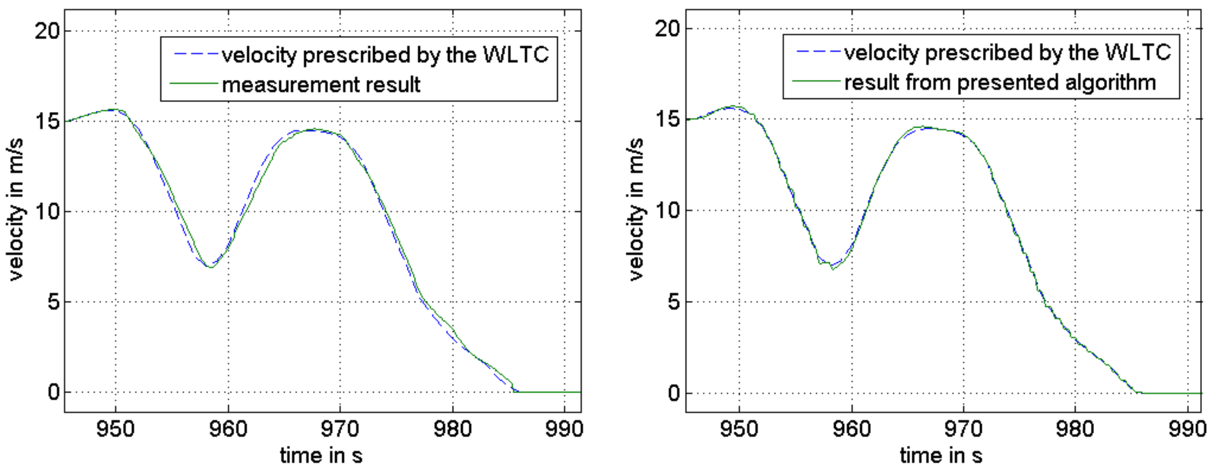


Figure 6.11: Voltage deviation of the output of the C++ program from the WLTC-measurements for vehicles of Class 3

interval from 950 s to 990 s, where the real driver is able to follow the prescribed velocity well without drifting far off course. Figure 6.12 shows the velocity profiles of the output of the C++ program and of the real driver performing the measurement run compared to the original data of the WLTC for vehicles of Class 3 in this interval. As can be seen, the



(a) real driver following the velocity over time prescribed by the WLTC for vehicles of Class 3 performing measurements

(b) driver model implemented in the presented program following the velocity over time prescribed by the WLTC for vehicles of Class 3

Figure 6.12: Extract of the velocity profiles resulting when the simulated and the real driver follow the velocity prescribed by the WLTC for vehicles of Class 3

difference between the velocity profile of the real driver and the original WLTC for vehicles of Class 3 is higher than that between the driver implemented in the C++ program and the original cycle, which can explain the voltage deviation at this time. Again, voltage fluctuations may be caused by the small oscillations in the velocity of the simulated driver when decelerating.

Figure 6.13 shows the SOC over time issued by the C++ program and the measured SOC over time for a WLTC for vehicles of Class 3. Although there are deviations of the battery

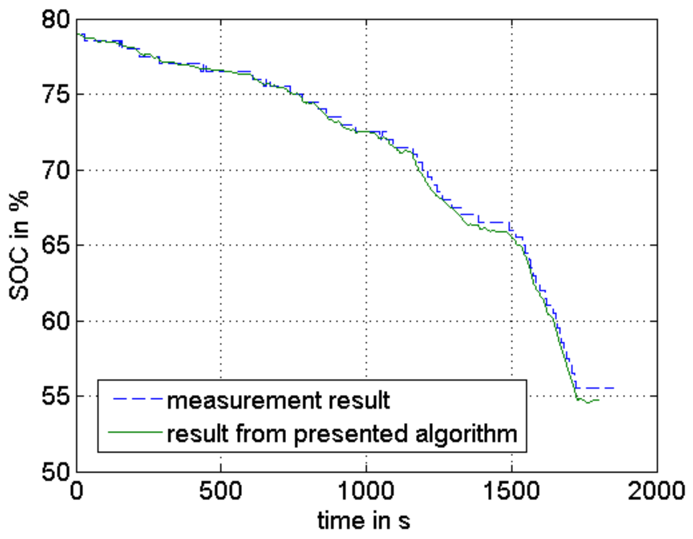


Figure 6.13: Comparison of the SOC over time between the C++ program and WLTC-measurements for vehicles of Class 3

voltage of the program output from the measured battery voltage, the values for the SOC match well. This may come from the fact that the deviations are small variations fluctuating around zero for a large part.

Generally, the tool was designed such that the SOC is always slightly underestimated. In this way it should be avoided that the user gets stranded due to inaccuracies in calculations or research data. In return, some phenomenons that may influence the SOC of a BEV were neglected. For example, ambient temperature, wind conditions and road conditions have not been considered. In comparison to the modelled phenomenons these factors were assumed to have rather small impacts on the SOC, unless there are extreme conditions like sub-zero temperatures in winter or high wind speeds above 60 $\frac{km}{h}$. Such conditions are also difficult to model since they are hardly predictable.

In the tool routing the i-MiEV, the battery model of Figure 3.7 was simplified to a model that just considers the voltage drops due to the ohmic resistance R_s of arresters, active material and electrolyte, as explained in Section 5.2. Thus, in what follows the results of the energy consumption model using the simplified battery model are discussed. In the Figures 6.14 and 6.15 the results of the presented program using the simplified battery model are compared to the measurements for two consecutive NEDCs when the air-conditioning system is turned on and the exterior lighting is turned off. Figure 6.14 depicts the battery voltage over the time needed to complete two consecutive NEDCs. Since the deviations of

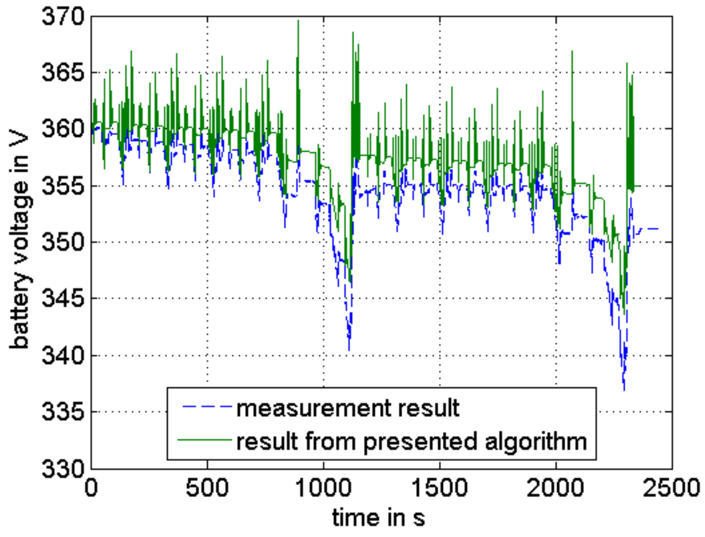


Figure 6.14: Comparison of the battery voltage over time between the C++ program using the simplified battery model and NEDC-measurements

the voltage of the modelled battery from the voltage of the measurements is apparent, it is not depicted in an extra figure. The voltage of the modelled battery is appreciably higher, because the voltage drops due to the double-layer capacitance and relaxation effects are neglected.

Figure 6.15 shows the SOC over time. The values for the SOC seem to match even better

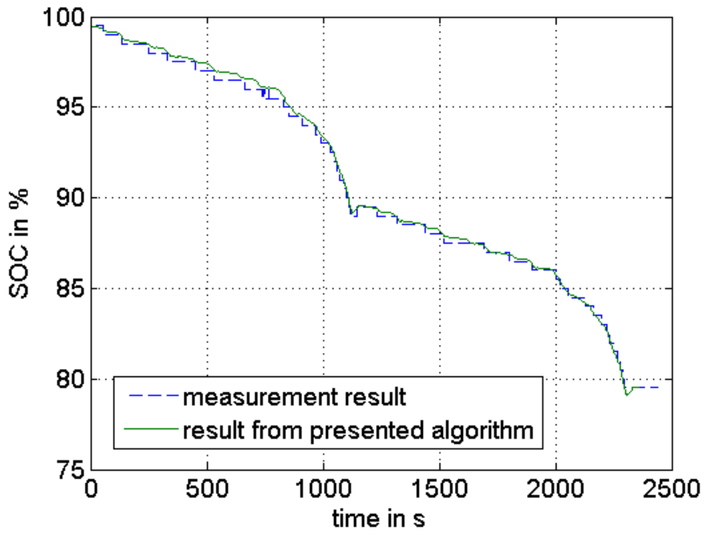


Figure 6.15: Comparison of the SOC over time between the C++ program using the simplified battery model and NEDC-measurements

than before. What really happens is that a part of the intended underestimation of the SOC gets lost when using the simplified battery model. For long distances the SOC is still slightly underestimated, such that this fact was accepted. The difference in the SOC between the two battery models is small but it is clearly identifiable.

In the Figures 6.16 and 6.17 the results of the presented program using the simplified battery model are compared to the measurements for a WLTC for vehicles of Class 3 when the air-conditioning system is turned on and the exterior lighting is turned off. Figure 6.16 depicts the battery voltage over the time needed to complete a WLTC for vehicles of Class 3. As in

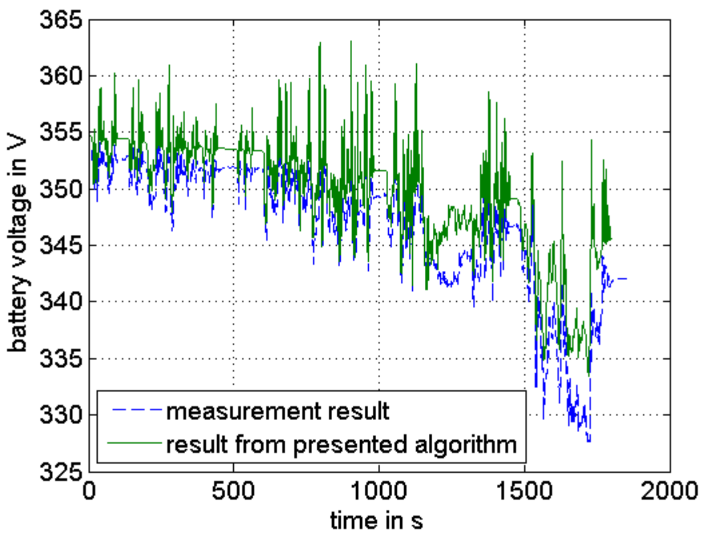


Figure 6.16: Comparison of the battery voltage over time between the C++ program using the simplified battery model and WLTC-measurements for vehicles of Class 3

Figure 6.14 the voltage of the modelled battery is higher because the voltage drops due to the double-layer capacitance and relaxation effects are neglected.

Figure 6.17 shows the SOC over time. Also for this figure, the same statements as for Figure 6.15 hold.

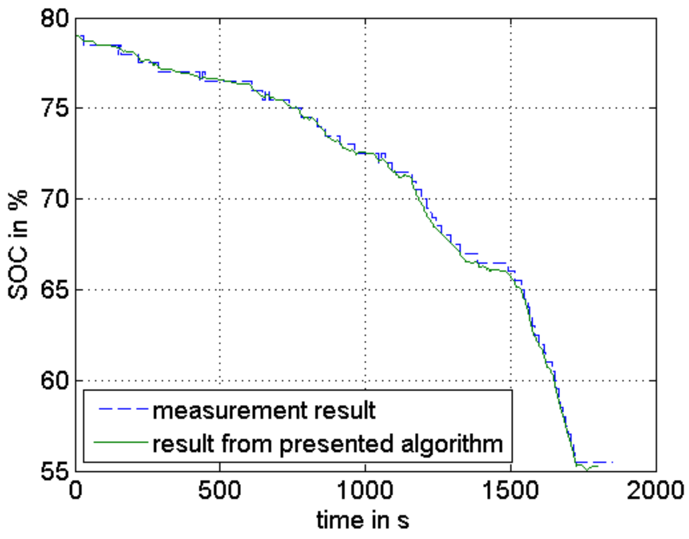


Figure 6.17: Comparison of the SOC over time between the C++ program using the simplified battery model and the WLTC-measurements for vehicles of Class 3

6.2 Input Graph for the Routing Tool

Besides the dependency of the results of the shortest path algorithm on the energy-consumption model there is a great reliance on the graph. All data used for the graph had already been aggregated by the AIT. This data contains the road network of Vienna only. Most of the data originally sources from OpenStreetMap [38]. That is vertex IDs, street IDs and the belonging names and coordinates. Since the AIT data is used for own researches, there are special link IDs for the edges, dividing OpenStreetMap edges into more edges. These AIT-link IDs were aggregated in a table together with vertex IDs for the vertices at the ends of an edge, street names, speed limits in $\frac{km}{h}$, lengths of the edges in m, altitude differences to be ascended and altitude differences to be descended when travelling along an edge into the direction it is defined with given in m, specifications of one-way streets as Boolean values, functional road classes and multiple geographic coordinate values for each edge given in degrees.

The vertex IDs originate from [38], but they were redefined for this work. This was done, because the implemented graph algorithm works with vertices continuously numbered from zero to the ID of the last vertex. Edges which are inaccessible for cars and the according vertices had already been marked as such in the used data set and were crossed out with the consequence that the numbering was not consistent any more. Therefore, all unique vertex IDs remaining were stored ordered ascending in a vector within MATLAB® and an unordered matrix with all utilised data was created. The vertex IDs of the matrix were compared with those of the ordered vector and when a value accorded with the i^{th} entry of the vector the new vertex ID was set as $i - 1$ since in MATLAB® the counter for vectors and matrices starts with one, but the IDs should begin with zero. A text file containing the vertex IDs after filtering out the edges inaccessible for cars, ordered ascending in a column, was created in order to find a new ID, given its ID from OpenStreetMap. The line number where the ID is found minus one is the ID used in the presented algorithms. While sorting a matrix with all edges by link IDs, missing speed limits were added. Since the lack of data came up only for residential and insignificant roads, all added velocities were set as $20\,\frac{km}{h}$. For the processing with C++, umlauts and delimiters between words were replaced by appropriate vowels and underscores. The numbers of vertices and edges in the graph were stored in a text file together with the complete table for the graph data as input for the tool routing the i-MiEV.

There are three more or less serious inconveniences of the graph that are dealt with in the consumption model. Of 329446 edges there are only 51 edges for which the gradient is to steep for the i-MiEV, most of them are mountain paths. So this problem is quite harmless. There are 30715 edges whose lengths are too short for recuperating when braking and 46781 edges whose lengths are too short to accelerate to the speed limit. So, for about every seventh edge there is a positive mistake due to too short lengths for reaching the speed limit and a positive mistake due to too short lengths for recuperating for about every tenth edge. On these edges, the program overestimates the energy consumption of the vehicle and thus underestimates its SOC. These problems could be considered when composing a new graph for the presented tool in future works.

6.3 Results of the Shortest Path Algorithm

Basing on the energy-consumption model the shortest path algorithm should deliver the result that is actually interesting for the user of the presented tool. Knowing his start point and the desired destination the driver of a BEV may want to know if he can reach it with the initial SOC or if he has to recharge the battery first or even has to make an intermediate stop to recharge the battery on the way. The algorithm determines the shortest path from the given source vertex s to the also given target vertex t and tells whether the destination can be reached on this path or not. It is also possible to find the place where the battery runs out of charge.

The final outcome of the tool is determined in the recursive function presented in Section 5.2. The program displays a list of the street names of all passed road sections sorted from s to t and the SOC at the end of each road section. The shortest path, defined as the path minimising a certain cost function, is visualised using an Application Programming Interface (API) for Google Maps from [22]. Therefore, a Hypertext Markup Language (HTML) document is created and the coordinates of the streets is pasted together with the according code from [22] in the correct order. All the shortest path queries were performed within the presented graph of the city of Vienna. Unless otherwise stated, the facilities for heating, air conditioning, ventilation and exterior lighting are considered to be turned off.

In Figure 6.18 the most energy-efficient path from a garage in Seestadt Aspern to a parking space in Ottakring computed by the presented program is visualised. It was found by using

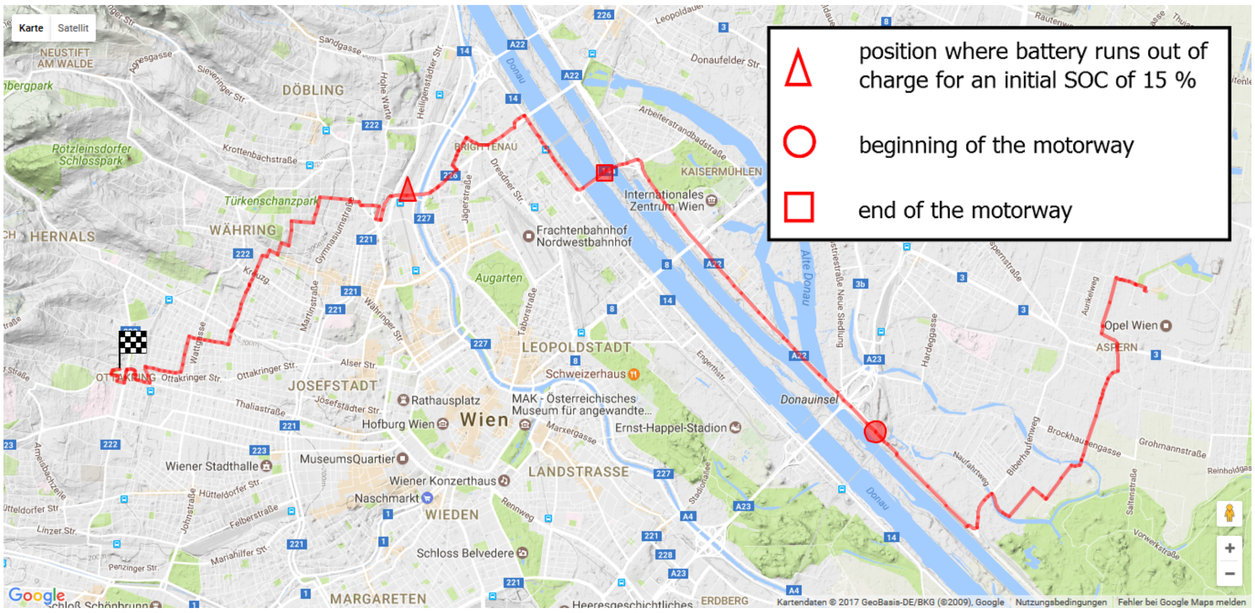


Figure 6.18: Most energy-efficient path from a garage in Seestadt Aspern to a parking space in Ottakring, visualised using [22]

the presented calculation of the energy consumption for a pseudo-dual edge as cost function in the shortest path algorithm. The initial SOC was chosen to be 70 % of the battery's capacity. Table 6.1 indicates the beginning and the end of the related list with street names of road sections.

Table 6.1: Beginning and end of a list of street names of the road sections in the most energy-efficient path from a garage in Seestadt Aspern to a parking space in Ottakring

Street Names	Distance Covered	SOC
Start_Point	$0\,m$	70 %
-	$29.2261\,m$	69.9684 %
Ilse-Arlt-Strasse	$75.8334\,m$	69.8558 %
Maria-Tusch-Strasse	$208.775\,m$	69.7826 %
An_den_alten_Schanzen	$220.372\,m$	69.8141 %
An_den_alten_Schanzen	$223.871\,m$	69.8101 %
.	.	.
.	.	.

.	.	.
Erdbrustgasse	24983.3 m	48.9487 %
Erdbrustgasse	25062.7 m	48.9529 %
Erdbrustgasse	25066.5 m	48.9487 %
Target_Point	25066.5 m	48.955 %

In the columns from left to right the street names, the distance covered so far and the SOC in percent of the battery's capacity are stated. The penultimate line gives values for reaching the destination without braking. These are the name of the street where the destination is located, the distance to it and the SOC when passing the destination. The last line gives the SOC when the vehicle has stopped and performed regenerative braking. Additionally, the program confirms that the destination can be reached with the current SOC. This path has a total length of 25.0665 km and the total energy consumption for covering it with the i-MiEV calculated by the presented program is 13.0925 MJ. That corresponds to 0.145 $\frac{kWh}{km}$. The results of an Allgemeiner Deutscher Automobil-Club (ADAC) test [39] state that the i-MiEV requires about 0.113 $\frac{kWh}{km}$ in town, 0.1503 $\frac{kWh}{km}$ out of town and 0.257 $\frac{kWh}{km}$ on a motorway. As mean value 0.177 $\frac{kWh}{km}$ is stated in [39]. The route depicted in Figure 6.18 is seen as a town route, although there is a short motorway section in it. Starting the route after the motorway section, marked with a square in Figure 6.18, the i-MiEV requires 0.177 $\frac{kWh}{km}$ for driving in town according to the C++ program, which is a higher value than before. This may come from the fact that from this point on, the i-MiEV has more stop-and-go phases which produces higher energy costs. Moreover, there are several trunk-road sections, where the vehicle has to accelerate to higher velocities as it is usual for urban driving conditions. From the garage in Seestadt Aspern to the start of the motorway, marked with a circle in Figure 6.18, there are no trunk road sections and the i-MiEV requires 0.108 $\frac{kWh}{km}$ according to the C++ program. This value matches well with the energy consumption when driving in town stated in [39]. On the motorway section the i-MiEV requires 0.152 $\frac{kWh}{km}$ according to the C++ program, whereas in [39] the according energy consumption is given by 0.257 $\frac{kWh}{km}$. The value of the simulation deviates from the measured one, because with 5.572 km length the section for simulating driving on a motorway may just be too short to be representative. To simulate driving on a motorway or driving out of town, an other graph than the one of Vienna should be considered. The comparison of simulation with measured data shows good correspondence considering the fact that road networks in towns differ from each other in terms of speed limits, road gradients and lengths, just to give a few examples.

Figure 6.19 depicts the elevation profile of the discussed path based on data provided by [2]. It shall just give an idea of the change in height along the route and was pointwise determined at six positions. On the way to the beginning of the motorway located around 6 km away from the start point and on the motorway that is travelled along for about 5.5 km there is

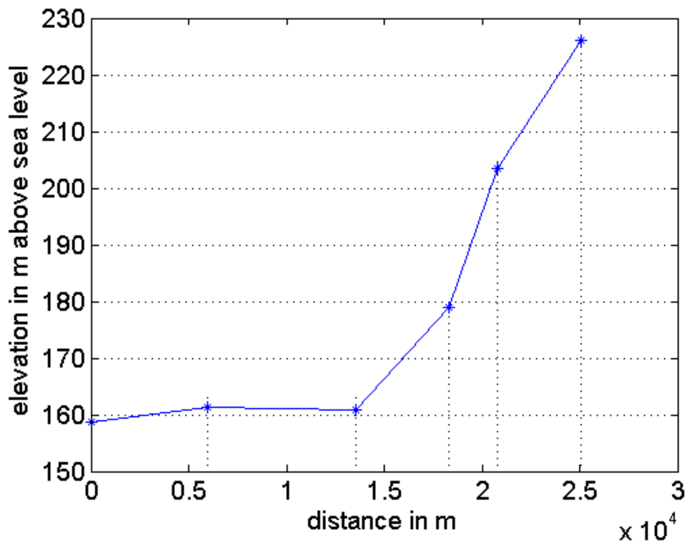


Figure 6.19: Meters above sea level given for six positions on the most energy-efficient path from a garage in Seestadt Aspern to a parking space in Ottakring

no significant height change. In the second half of the way, there is a height change of about $65\,m$. The larger gradient after the motorway section also contributes to the high energy consumption on the remaining distance to the destination.

The shortest path algorithm was run one more time with the same source and target as in the beginning, but starting with an initial SOC of 15 % to see what is the output of the program if the battery runs out of charge on the way. Table 6.2 indicates the beginning and the end of the program-output list of the road sections passed.

Table 6.2: Beginning and end of a list of street names of the road sections when the battery runs out of charge on the way from a garage in Seestadt Aspern to a parking space in Ottakring

Street Names	Distance Covered	SOC
Start_Point	$0\,m$	15 %
-	$29.2261\,m$	14.9684 %
Ilse-Arlt-Strasse	$75.8334\,m$	14.8482 %
Maria-Tusch-Strasse	$208.775\,m$	14.7699 %
An_den_alten_Schanzen	$220.372\,m$	14.8035 %
An_den_alten_Schanzen	$223.871\,m$	14.7993 %
.	.	.
.	.	.
.	.	.
Guertelbruecke	$17037.9\,m$	0.401613 %

Guertelbruecke	$17110.3\,m$	$0.359768\,\%$
Guertelbruecke	$17146.9\,m$	$0.36729\,\%$
Guertelbruecke	$17482.1\,m$	$0\,\%$

The algorithm stops outputting the road sections once the battery runs out of charge and issues that the destination can not be reached with the current SOC. According to Table 6.2 with the initial SOC of $15\,\%$ the i-MiEV can cover about $17.48\,km$ of the most efficient route from a garage in Seestadt Aspern to a parking space in Ottakring. On Figure 6.18 the place where the battery runs out of charge is marked with a triangle.

The calculation of the just presented path was performed when the costs of the graph were already computed and shifted. Thus, the graph had to be read in and Dijkstra's algorithm had to be executed. The process for reading the graph took $387.243\,s$ and Dijkstra's algorithm took $106.439\,s$, including the time to store the results. Some more time is needed to call the destructors of own classes and classes from the STL. The conditions of the hardware and software for these measurements were the same as mentioned in Section 6.1.

The added value of the presented tool to finding most efficient paths is the realistic energy-consumption model. Compared to traditional navigation systems searching for the shortest path minimising the distance between s and t the presented tool is more advantageous for the user who wants to save energy. This is illustrated using Figure 6.20 depicting the path with the least distance as a blue curve and the most energy-efficient path as a red curve from the company location of the AIT to Am Cobenzl. The difference in the altitude that

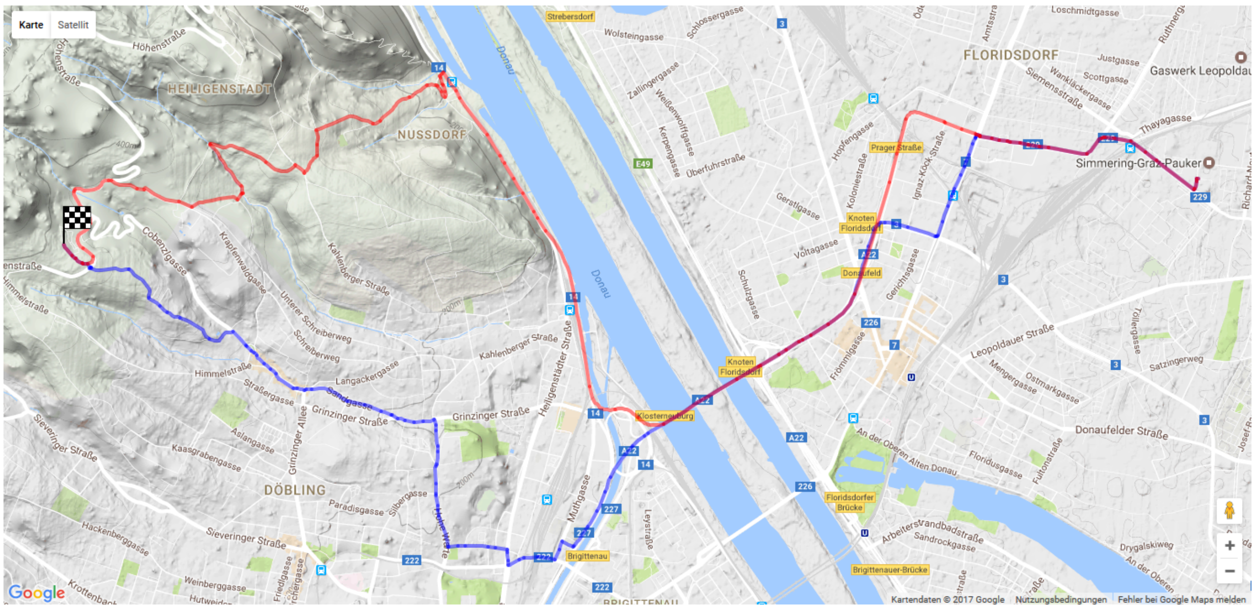


Figure 6.20: Most energy-efficient path (red curve) and path with the least distance (blue curve) from the AIT courtyard to Am Cobenzl, visualised using [22]

has to be climbed is $220\,m$, since according to [2] the AIT courtyard is situated $162.1\,m$ and Am Cobenzl is situated $382.1\,m$ above sea level. The distance from the start point to the destination is $13.246\,km$ using the most energy-efficient path and $11.3303\,km$ using the path with the least distance. Starting with an initial SOC of 70 %, the i-MiEV has a SOC of 53.7979 % at Am Cobenzl when taking the most energy-efficient path and 50.2651 % when taking the path with the least distance. The total energy consumption is $10.0858\,MJ$ on the most energy-efficient path and $12.2449\,MJ$ on the path with the shortest distance. Thus, $2.1591\,MJ$ can be saved when taking the energy-efficient path instead of the short path. The energy-consumption-distance ratio is $0.212\,\frac{kWh}{km}$ for the energy-efficient path and $0.3\,\frac{kWh}{km}$ for the short path. These values are higher than the average one given by [39] since the considered path is quite steep.

Another feature of the presented tool is that it can compare energy consumptions for different driving styles. To show this, three drivers with differing controller settings were simulated travelling along the most energy-efficient path from the Vienna airport to the Wolfersberg with an initial SOC of 90 %. The driver models are referred to as *Driver A*, *Driver B* and *Driver C*. There are different controller settings for accelerating, braking and driving with constant velocity. The settings for the different drivers are listed in Tables 6.3 – 6.5.

Table 6.3: controller settings for *Driver A*

Procedure	K_P	T_N	T_V	T_a
acceleration	0.65	$0.5\,s$	$0.1\,s$	$0.1\,s$
braking	0.15	$0.3\,s$	$0.2\,s$	$0.1\,s$
constant velocity	0.5	$0.4\,s$	$0\,s$	$0.1\,s$

Table 6.4: controller settings for *Driver B*

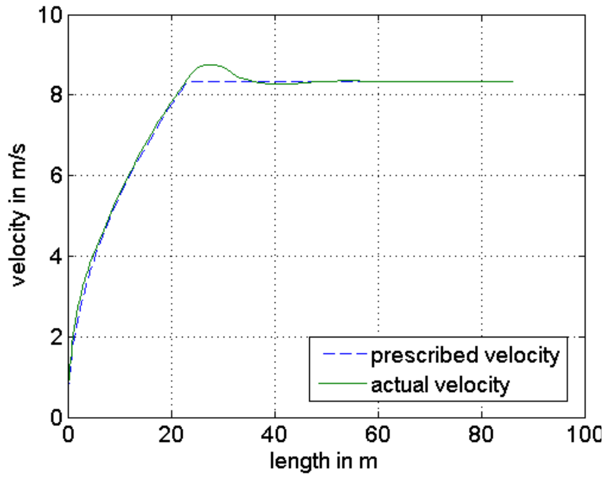
Procedure	K_P	T_N	T_V	T_a
acceleration	0.5	$0.4\,s$	$0\,s$	$0.1\,s$
braking	0.14	$0.6\,s$	$0\,s$	$0.1\,s$
constant velocity	0.5	$0.4\,s$	$0\,s$	$0.1\,s$

Table 6.5: controller settings for *Driver C*

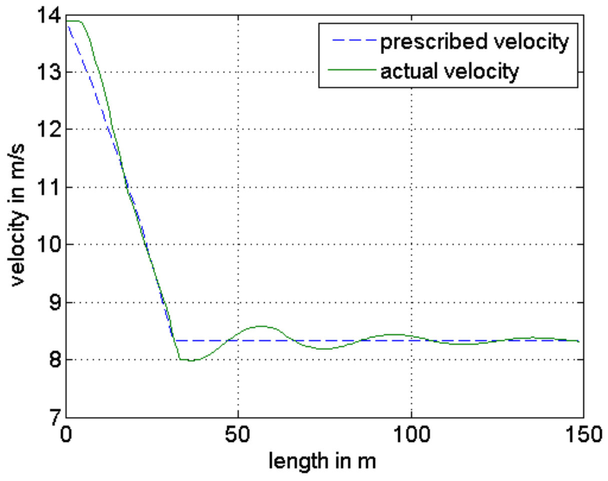
Procedure	K_P	T_N	T_V	T_a
acceleration	0.1389	$0.6\,s$	$0\,s$	$0.1\,s$
braking	0.1389	$0.6\,s$	$0\,s$	$0.1\,s$

constant velocity	0.5	0.4 s	0 s	0.1 s

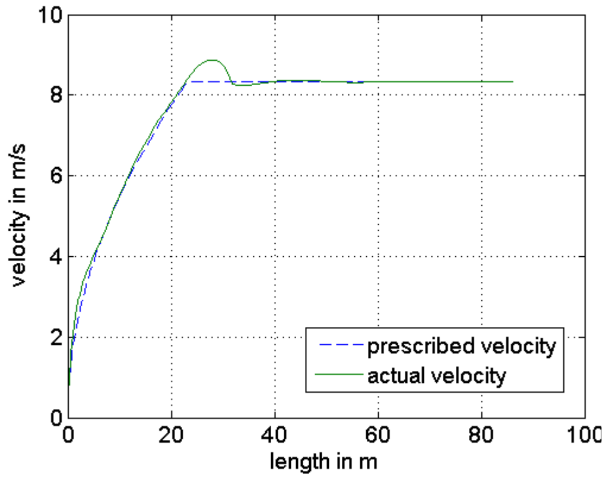
In Figure 6.21 it is shown how the *Drivers A – C* act when accelerating from a certain velocity to a higher one. Figure 6.21b exemplary illustrates the deceleration behaviour of *Driver A*. From A to C the energy efficiencies of the driving styles decrease. Thus, it is expected that



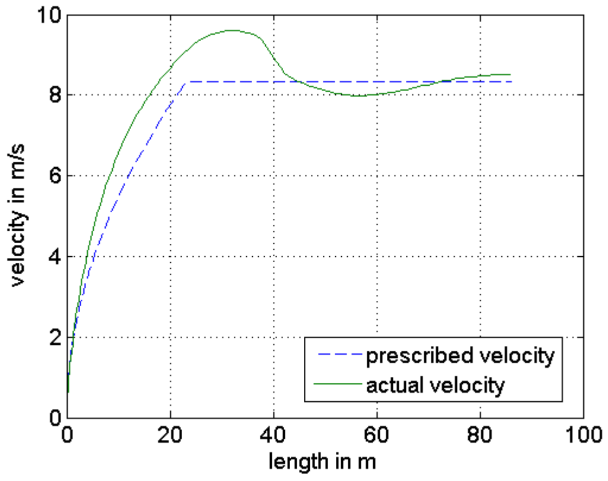
(a) acceleration behaviour of *Driver A*



(b) deceleration behaviour of *Driver A*



(c) acceleration behaviour of *Driver B*



(d) acceleration behaviour of *Driver C*

Figure 6.21: Simulated *Drivers A*, *B* and *C* trying to follow a prescribed velocity profile

from A to C the final SOC for each driver on a certain route decreases. Figure 6.22 gives the results of the shortest path algorithm for the costs calculated using the different driver models. The path courses alter slightly, because the different routes have their advantages for each driving style. Table 6.6 gives the route lengths, the energy consumption and the final SOC values for the different drivers.

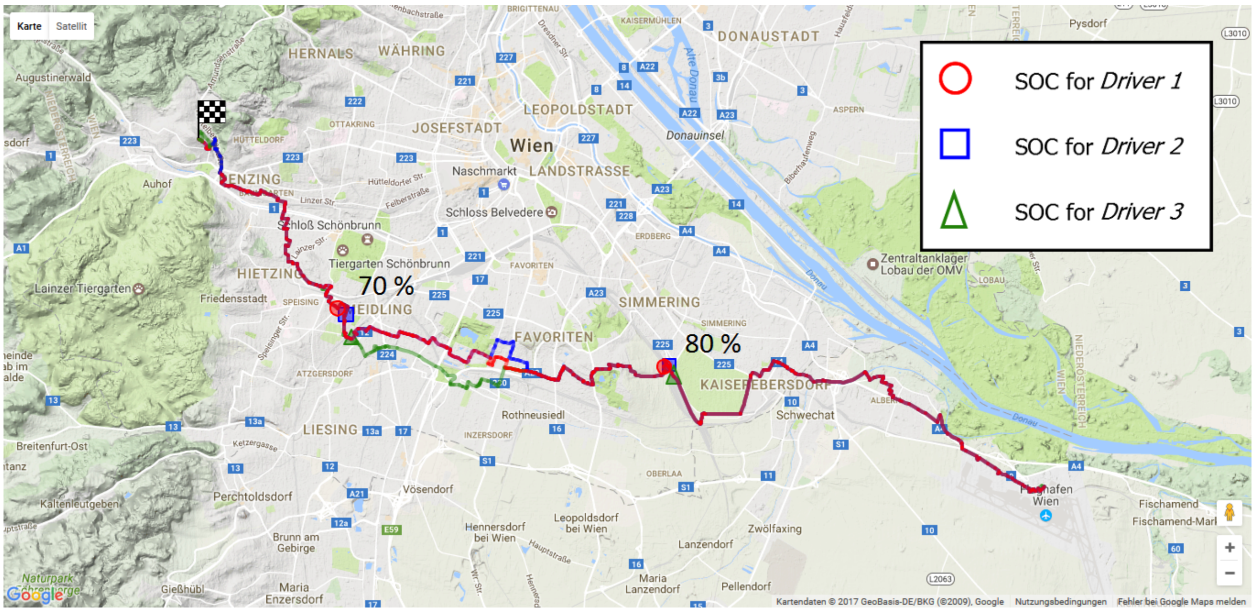


Figure 6.22: Most energy-efficient paths from the Vienna airport to the Wolfsberg for *Driver A* (red curve), *Driver B* (blue curve) and *Driver C* (green curve) and related intermediate SOC values, visualised using [22]

Table 6.6: Characteristic values for different driver models travelling along their most energy-efficient paths from the Vienna airport to the Wolfsberg

Driver Model	Route Length	Total Energy Consumption	Final SOC
Driver A	38.903 km	17.7949 MJ	61.9586 %
Driver B	39.3971 km	18.0264 MJ	61.5829 %
Driver C	39.1835 km	18.7707 MJ	60.3822 %

Comparing the drivers in terms of their energy consumption gives similar results. This implies that the driving style of a user has only limited impact. In contrast, the program output indicates that the additional consumption of facilities for driving assistance and comfort have a large influence on the SOC. This is shown in Figure 6.23. It depicts the most energy-efficient path for *Driver A* when the facilities for heating, air conditioning, ventilation and exterior lighting are considered to be turned on. Additionally, intermediate SOC values are given. The route is exactly the same as the route for *Driver A* in Figure 6.22, but the SOC decreases even faster than for the least energy-efficient driving style of *Driver C* when the facilities are turned off. The according total energy consumption is 24.6218 MJ and the SOC at the destination is 50.9589 %.

To simulate different drivers, the costs for the graph had to be recalculated several times. The average times required to perform the different tasks are stated in Table 6.7. The conditions of the hardware and software for these measurements again were the same as mentioned in Section 6.1.

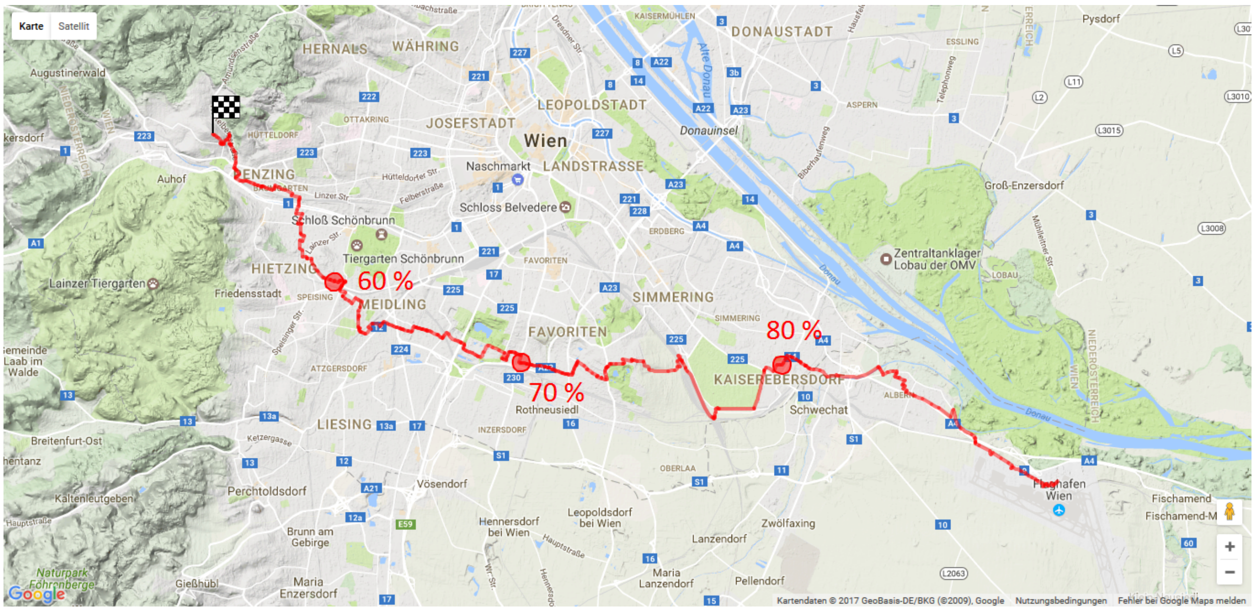


Figure 6.23: Most energy-efficient path from the Vienna airport to the Wolfsberg for *Driver A* when facilities for heating, air conditioning, ventilation and exterior lighting are turned on and related intermediate SOC values evaluated at the positions marked with the red circles, visualised using [22]

Table 6.7: Average times required for the different program tasks

Task	Average Time
read in primal graph data	36 s
create the pseudo-dual graph	128 s
calculate edge costs	630 s
intermediate Input/Output (I/O) operations	250 s
Bellman-Ford algorithm (potential shifting)	1460 s
intermediate I/O operations	435 s
Dijkstra's algorithm (including output operations)	115 s

Chapter 7

Conclusion and Outlook

In this thesis a realistic power-consumption model of a BEV that slightly overestimates its energy consumption to prevent inaccuracies has been introduced. Doing so, exterior forces resisting an acceleration, inertial forces of inner rotating parts, losses along the power train and energy consumptions of facilities for driving assistance and comfort have been considered. Detailed models of the battery, the electric motor and the chassis and its drag were created and put together to a model of a BEV. Its parameters were set in a way that the resulting program describes the energy consumption of the i-MiEV. First, the energy consumption and the SOC of the i-MiEV over time were evaluated for driving cycles. Then, a graph model suitable for routing a BEV taking into account its characteristic features was generated. The energy-consumption model was combined with a graph algorithm using the created static graph for finding paths with least energy consumption of the i-MiEV.

There are countless influences on a BEV that can be considered when simulating real-life-driving conditions. Most of them have a small impact and are rather difficult to describe. The final model of the i-MiEV established in this thesis is assumed to be realistic without regarding all possible influences on the battery's SOC. The power consumptions of the modelled phenomenons were estimated little higher than in reality to compensate the not simulated factors. For instance, the resistance coefficients were assigned with values of the upper boundary of a given range found in literature. Examples of neglected influences on the energy consumption are ambient air temperature, wind and road conditions. These are generally small, but can get notably higher at extreme conditions such as high wind speeds of $60\frac{km}{h}$ and higher, sub-zero temperatures or snow-covered roads. Moreover, environmental conditions like the ones just mentioned are often regional and hardly predictable. For being able to give reasonable results in such situations the tool has to be adapted accordingly. In normal circumstances, the energy consumption issues realistic results, as shown in Section 6.1. The implemented vehicle model is more detailed than the ones commonly used for energy consumption evaluations. All components for which data is usually given by the manufacturer are simulated. Additionally, the driving behaviour of different users can be reflected and used to adopt the values for the energy consumption.

A common graph is not able to hold costs that come up when passing from one edge to another. Therefore, the primal input graph has been converted to its pseudo-dual graph that can be assigned with the costs of the primal graph and additionally with the turn costs it takes for crossing or making a turn on junctions for example. The primal graph used in this thesis contains lots of very short road sections as edges. It is difficult to assign costs to such edges, because it is not possible to generate a complete driving cycle for accelerating or decelerating to the according speed limit with the maximum acceleration or deceleration rate of the vehicle since there is just not enough space to do so. In order not to underestimate the energy consumption on such edges when calculating the costs, the lengths of these edges have been extended or regenerative braking was not considered. The number of such short edges occurring in the graph is considerably high, it is stated in Section 6.2. This defect could be eliminated when creating further graphs that can be used as input for the tool. Moreover, a program getting geographical data and generating the primal graph of a street map automatically would be useful.

The implemented graph algorithm is able to exploit most of the potential of the energy consumption model. It includes turn costs since the pseudo-dual graph is used for the shortest path query. Routes can be found for any start point and destination applying Dijkstra's algorithm when the costs are shifted yet or a combination of the Bellman-Ford and Dijkstra's algorithm for the first path query when changes of the cost function are considered. Thus, the parameters can be varied such that different vehicle or driver models can be evaluated. The battery model of the original vehicle model had to be simplified, because as the cost of a certain edge the energy consumption is a constant value for the related road section, such that the battery current was considered to be constant there. However, the vehicle model was proved to still issue realistic results when using the simplified battery model in Section 6.1.

More sophisticated graph algorithms yet exist in literature, but implementing them would lead too far for this thesis and moreover it would not close any gap of knowledge any more. Nevertheless, some ideas of the appropriate scientific works given below would improve the presented tool and could be realised in future projects. For example, as a considerable improvement the algorithm could be dynamised, such that it would accept changes of the costs of the graph in time. Then, edge costs would be determined while driving along the belonging road section and the according energy consumption would not be a constant value any more. Thus, the original battery model could be used since the differential equations describing the double-layer capacitance and relaxation effects could be applied rationally. Furthermore, changing traffic conditions could be taken into account. In [7] an algorithm for the prefix-bounded shortest path problem is suggested, where a recursive function *absorp* accepts the charge level of the battery as a parameter at each vertex to set constraints for choosing a path. If further data would be available, the proposal of [51] to regard turn restrictions at junctions could be realised as well. Data for the exact positions of traffic lights or stop signs in the street map of a city could further enhance the results. Since the procedure for stopping at a junction is implemented in the presented tool, it could be easily adopted to consider this additional data and even evaluate average stoppage times.

Unlike common navigation systems, the presented tool indicates how a user can affect the driving range of the i-MiEV. According to Figure 6.20 the shortest path is not always the best way to choose if the driver wants minimise battery-charging intervals. So he or she may consider to accept a longer journey time and save energy in return. The results of this thesis reveal that the user can extend the driving range by improving his personal driving style. Figure 6.22 implies that this gets apparent only when covering a larger distance. The program output further indicates that the impact of the power consume of facilities for driving assistance and comfort on the SOC is even bigger. Thus, the user should switch on facilities only if necessary.

Chapter 8

Appendix

8.1 Formula Symbols

α number of current branches of a winding layer on the circumference of the armature of a DC machine

β pitch angle of a road segment

ε edge of the pseudo-dual graph D, $\varepsilon \in E_D$

η_{12V} efficiency factor of the low voltage lead-acid battery for charging and discharging

η_{con} efficiency factor of the DC-DC converter

η_{diff} efficiency factor of the differential

η_{360V} efficiency factor of the lithium-ion battery

η_{gear} efficiency factor of the gear

η_{inv} efficiency factor of the inverter

η_{mech} total efficiency factor of all considered mechanical parts, $\eta_{mech} = \eta_{gear} \cdot \eta_{diff} \cdot \eta_{motor}$

η_{motor} efficiency factor of the motor

μ friction coefficient for brake disc and block pads

ν vertex of the pseudo-dual graph D, $\nu \in V_D$

ρ air density

Φ magnetic flux

ω_{nom} nominal angular velocity of the machine

ω_t angular velocity of a tyre

ω_m angular velocity of the machine

A frontal surface area of the vehicle

a_{ap} constant acceleration assumed for an acceleration process

a acceleration

a_{dp} constant deceleration assumed for a braking process

b number of current branches of a winding layer on the circumference of the armature of a DC machine

c cost of a weighted edge, $c(u, v)$ is the weight of the edge (u, v)

$\bar{c}$ shifted cost of a weighted edge, $\bar{c}(u, v)$ is the shifted weight of the edge (u, v)

C_{bat} battery capacity

c_d drag coefficient

C_p capacitance due to double-layer capacitance

c_{rr} rolling-resistance coefficient

D pseudo-dual graph consisting of n_D vertices and m_D edges

d vector for path costs, $d[u]$ is the cost of a path from the source vertex to vertex u

dx adjacent leg in the gradient triangle

dy height difference on road segments and opposite leg in the gradient triangle

E edge set of a graph

e edge of a graph G, $e \in E$

$e(kT)$ discrete system deviation with sampling interval T and and counter k

$e(t)$ continuous system deviation

F actual propulsion force on the vehicle

f number of friction faces of a brake disc

$\bar{f}$ bijective function that converts an edge e_i to a vertex ν_i of the pseudo-dual graph D

F_N maximum normal force on a brake disc from block pads

$f(t)$ auxiliary function to describe intermediate steps

$f(x_i)$ function f evaluated at x_i

G Graph consisting of n vertices and m edges

g gravitational constant

$g(t)$ auxiliary function to describe intermediate steps

h overall height of the BEV

h_c ground clearance of the BEV

i index of a loop, array, vector or matrix

I_A armature current or effective value of alternating armature current

I_{bat} battery current

J moment of inertia

$J_{g,t}$ moment of inertia of the gear at the shaft of the tyres

$J_{g,m}$ moment of inertia of the gear at the shaft of the motor

J_m moment of inertia of the motor

J_t moment of inertia of a tyre

k_1 machine constant $k_1 = z\frac{p}{a}$

k_2 machine constant $k_2 = \frac{k_1}{2\pi}$

k_{diff} transmission ratio of the differential

k_{mech} total transmission ratio $k_{mech} = k_{gear} \cdot k_{diff}$

k_{gear} transmission ratio of the gear

K_P controller-gain constant

K_S gain of the control variable in the step response of a system to be controlled

l length of a road section

m total number of edges in an edge set E

M_b maximum brake torque

$m_{chassis}$ curb weight of the BEV

M_g torque of the gear due to its inertia

M_i generated torque of the DC machine

m_{load} mass of occupants and cargo

M_{max} maximum torque of the machine

M_m torque of the motor due to its inertia

m_{tot} total mass of the vehicle including curb weight and load

M_t torque on the tyres due to their inertia

n rotations per second or total number of vertices in a vertex set V

p number of pole pairs on the circumference of the armature of a DC machine or path in a graph G

$\boldsymbol{parent}$ vector to store the predecessors of a vertex in the shortest path to it

P_{aux} power demand of auxiliaries at the lead-acid battery

$P_{aux,360V}$ power demand of auxiliaries at the lithium-ion battery

P_{tail} power demand of a tail lamp

P_{bat} power demand at the lithium-ion battery

P_{HAC} power demand of heating and air conditioning

P_{light} power demand of exterior lighting

P_{low} power demand of a low beam

P_m power of the electric motor

P_{max} maximum power of the electric motor

P_{mech} demand for mechanical power

pot vertex potential

P_{re} recuperated power

$P_{re,max}$ maximum recuperated power

q division ratio for the ascending part of a road segment or path in a graph G

R restricted pseudo-dual graph

r_e external brake-disc radius

r_i internal brake-disc radius

r_m mean radius of a brake disc $r_m = \frac{2}{3} \cdot \frac{{r_e}^3 - {r_i}^3}{{r_e}^2 - {r_i}^3}$

R_p ohmic resistance due to double-layer capacitance

R_s ohmic resistance due to resistances in the battery

r_t radius of a tyre

SOC_{lim} upper limit of the SOC given as charge-capacity ratio of the battery

s distance covered on a road segment or the source vertex of a path

T sampling interval

t variable for time or the target vertex of a path

T_a tracking time constant, anti-windup constant

T_N reset time

T_Σ cumulative time constant

T_V derivative time

u vertex in a graph G, $u \in V$

U_0 open-circuit voltage of the lithium-ion battery

U_i induced voltage or effective value of alternating induced voltage

u_i output of the integrator in a controller

$u(kT)$ discrete control signal with sampling interval T and counter k

U_p voltage drop due to double-layer capacitance

U_s voltage drop due to ohmic losses

u_{sat} value of $u(t)$ or $u(kT)$ upon application of a saturation function

$u_{sat,max}$ saturation value of the control signal $u(t)$ or $u(kT)$

$u(t)$ continuous control signal with time t

U_{bat} total voltage of the lithium-ion battery

V vertex set of a graph

v velocity or vertex in a graph G, $v \in V$

w width of the car body or vertex in a graph G, $w \in V$

w_t width of the tyres

x_i discrete value of a vector with index i

$y(t)$ control variable

z number of conductors on the circumference of the armature of a DC machine

8.2 Acronyms

Battery a class describing the battery model

Cost a class linking all program files to compute the costs of road sections

Driver a class describing the behaviour of the driver

DrivingCycle a class for reading the driving cycle

GraphAlgorithm a class performing all routing tasks

NumericalMathematics a class providing functions for numerical mathematics

Propulsion a class describing the mechanical model of the vehicle

PseudoDualGraph a class for reading graph data and generating a pseudo-dual graph

AC Alternating Current

ADAC Allgemeiner Deutscher Automobil-Club

AIT AIT Austrian Institute of Technology GmbH

API Application Programming Interface

BEV Battery Electric Vehicle

DC Direct Current

EV Electric Vehicle

HAC	Heating and Air Conditioning
HTML	Hypertext Markup Language
i-MiEV	Mitsubishi innovative Electric Vehicle, based on Mitsubishi i
I/O	Input/Output
NEDC	New European Driving Cycle
PSM	Permanently excited Synchronous Machine
SI	Système International d'unités
SOC	State Of Charge
STL	Standard Template Library
WLTC	Worldwide harmonized Light-Duty vehicles Test Cycle
WLTP	Worldwide harmonized Light-Duty vehicles Test Procedure

Bibliography

[1] Dynamic Programming — Set 23 (Bellman–Ford Algorithm). Retrieved from http://www.geeksforgeeks.org/dynamic-programming-set-23-bellman-ford-algorithm/, 20/03/2017.

[2] Geodatenviewer der Stadtvermessung Wien. Retrieved from https://www.wien.gv.at/ma41datenviewer/public/, 18/03/2017.

[3] Johnson's algorithm for All-pairs shortest paths. Retrieved from http://www.geeksforgeeks.org/johnsons-algorithm/, 20/03/2017.

[4] MITSUBISHI Automobile in Österreich. Retrieved from http://www.mitsubishi-motors.at/i-miev/, 20/03/2017.

[5] S. Agrawal. Djkstra's Shortest Path Algorithm using priority_queue of STL retrieved from http://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-using-priority_queue-stl/, 20/03/2017.

[6] S. Agrawal, H. Zheng, S. Peeta, and A. Kumar. Routing aspects of electric vehicle drivers and their effects on network performance. *Transportation Research Part D: Transport and Environment*, 46:246–266, 2016.

[7] A. Artmeier, J. Haselmayr, M. Leucker, and M. Sachenbacher. The optimal routing problem in the context of battery-powered electric vehicles. In *Workshop: CROCS at CPAIOR-10, 2nd International Workshop on Constraint Reasoning and Optimization for Computational Sustainability*, pages 1–10, Bologna, Italy, 2010.

[8] J. Asamer, A. Graser, B. Heilmann, and M. Ruthmair. Sensitivity analysis for energy demand estimation of electric vehicles. *Transportation Research Part D: Transport and Environment*, 46:182–199, 2016.

[9] J. Barco, A. Guerra, L. Muñoz, and N. Quijano. Optimal routing and scheduling of charge for electric vehicles: Case study. *arXiv preprint arXiv:1310.0145*, 2013.

[10] M. Baum, J. Dibbelt, T. Pajor, and D. Wagner. Energy-optimal routes for electric vehicles. In *Proceedings of the 21st ACM SIGSPATIAL international conference on advances in geographic information systems*, pages 54–63. ACM Press, 2013.

[11] L. Bedogni, L. Bononi, A. D'Elia, M. Di Felice, M. Di Nicola, and T. S. Cinotti. Driving without anxiety: A route planner service with range prediction for the electric vehicles. In *2014 International Conference on Connected Vehicles and Expo (ICCVE)*, pages 199–206. IEEE, 2014.

[12] L. Bedogni, L. Bononi, M. Di Felice, A. D'Elia, R. Mock, F. Morandi, S. Rondelli, T. S. Cinotti, and F. Vergari. An Integrated Simulation Framework to Model Electric Vehicles Operations and Services. *IEEE Transactions on Vehicular Technology*, 65(8):5900–5917. IEEE, 2015.

[13] R. Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.

[14] J. R. Bumby, P. H. Clarke, and I. Forster. Computer modelling of the automotive energy requirements for internal combustion engine and battery electric-powered vehicles. *IEE Proceedings A (Physical Science, Measurement and Instrumentation, Management and Education, Reviews)*, 132(5):265–279, 1985.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms.* MIT Press, Cambridge, Mass., 3. edition, 2009.

[16] D. Delling, P. Sanders, D. Schultes, and D. Wagner. Engineering route planning algorithms. In *Algorithmics of large and complex networks*, volume 5515 of LNCS, pages 117–139. Springer Berlin Heidelberg, 2009.

[17] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271. Springer, 1959.

[18] J. Eisner, S. Funke, and S. Storandt. Optimal Route Planning for Electric Vehicles in Large Networks. In *Proceedings of the 25th AAAI Conference on Artificial Integligence*, pages 1108–1113. AAAI, 2011.

[19] L. R. Ford Jr. Network flow theory. No. P-923. RAND CORP SANTA MONICA CA, 1956.

[20] H. Frohne and F. Moeller. *Moeller Grundlagen der Elektrotechnik: Mit 36 Tabellen und 182 Beispielen.* Vieweg + Teubner, Wiesbaden, 22. edition, 2011.

[21] A. Goel. Printing Paths in Dijkstra's Shortest Path Algorithm retrieved from http://www.geeksforgeeks.org/printing-paths-dijkstras-shortest-path-algorithm/, 20/03/2017.

[22] Google developers. Google Maps APIs. Retrieved from https://developers.google.com, 2016.

[23] J. G. Hayes, R. P. R. de Oliveira, S. Vaughan, and M. G. Egan. Simplified electric vehicle power train models and range estimation. In *Vehicle Power and Propulsion Conference (VPPC)*, pages 1–5. IEEE, 2011.

[24] P. Hofmann. *Hybridfahrzeuge.* Springer-Verlag Vienna, Vienna, 2010.

[25] M. Horn. Regelungstechnik. Lecture Notes, TU Graz, 2015.

[26] X. Huang. Negative-Weight Cycle Algorithms. In *Proceedings of the 2006 International Conference on Foundations of Computer Science*, pages 109–115. CSREA Press, 2006.

[27] S. Jakubek. Digital Control. Lecture Notes, TU Wien, 2012.

[28] A. Jossen and W. Weydanz. *Moderne Akkumulatoren richtig einsetzen: 36 Tabellen.* Ubooks, Neusäß, 1. edition, 2006.

[29] P. Kågeson. Cycle-beating and the EU test cycle for cars. *European Federation for Transport and Environment (T&E), Brussels*, 1998.

[30] S. Kitano, K. Nishiyama, J.-i. Toriyama, and T. Sonoda. Development of large-sized lithium-ion cell 'LEV50'and its battery module "LEV50-4" for electric vehicle. *GS Yuasa, Technical Report*, 5(1):21–26, 2008.

[31] H. C. Kuhlmann. Numerische Methoden der Ingenieurwissenschaften. Lecture Notes, TU Wien, 2011.

[32] R. Maia, M. Silva, R. Araújo, and U. Nunes. Electric vehicle simulator for energy consumption studies in electric mobility systems. In *2011 IEEE Forum on Integrated and Sustainable Transportation System (FISTS)*, pages 227–232. IEEE, 2011.

[33] K. Mehlhorn and P. Sanders. *Algorithms and Data Structures: The Basic Toolbox.* Springer-Verlag Berlin Heidelberg, 2008.

[34] M. Millikin. The battery pack for mitsubishi's i-MiEV. 2008. Retrieved from http://www.greencarcongress.com/2008/05/the-battery-pac.html, 21/04/2017.

[35] P. Mock, J. German, A. Bandivadekar, and I. Riemersma. Discrepancies between type-approval and "real-world" fuel-consumption and CO. In *Annual conference of the prognostics and health management society*, volume 13, 2012.

[36] National Renewable Energy Laboratory. Mitsubishi iMiEV: An Electric Mini-Car in NREL's Advanced Technology Vehicle Fleet (Fact Sheet). 2011. Retrieved from http://www.nrel.gov/docs/fy12osti/48528.pdf, 21/04/2017.

[37] J. A. Oliva, C. Weihrauch, and T. Bertram. A model-based approach for predicting the remaining driving range in electric vehicles. In *Annual conference of the prognostics and health management society*, volume 4, pages 438–448. New Orleans, USA.

[38] OpenStreetMap contributors. Planet dump. Retrieved from http://planet.osm.org, 2017.

[39] M. Ruhdorfer. ADAC-Autotest: Mitsubishi i-MiEV, 2011.

[40] M. Sachenbacher, M. Leucker, A. Artmeier, and J. Haselmayr, editors. *Efficient Energy-Optimal Routing for Electric Vehicles*. AAAI, 2011.

[41] P. Sanders and D. Schultes. Engineering highway hierarchies. In *14th European Symposium on Algorithms*, volume 4168 of LNCS, pages 804–816. Springer Berlin Heidelberg, 2006.

[42] E. Schmidt and E. Rummich. Grundlagen der Elektrotechnik für MB(033 245), WI-MB(033 282). Lecture Notes, TU Wien, 2011.

[43] B. Schoettle, M. Sivak, and Y. Fujiyama. *LEDs and power consumption of exterior automotive lighting: implications for gasoline and electric vehicles*. University of Michigan, Transportation Research Institute, 2008.

[44] M. Schrödl. Drehstromantriebe mit Mikrorechnern. Lecture Notes, TU Wien, 1988.

[45] J. W. Stevens and G. P. Corey. A study of lead-acid battery efficiency near top-of-charge and the impact on PV system design. In *Conference of the 25th IEEE Photovoltaic Specialists*, pages 1485–1488. IEEE, 1996.

[46] UNECE Global Technical Regulation No. 15. Worldwide harmonized Light vehicles Test Procedure. UNECE, Geneva, Switzerland, 2014. Retrieved from http://www.unece.org/fileadmin/DAM/trans/main/wp29/wp29r-1998agr-rules/ECE-TRANS-180a15e.pdf, 05/04/2017.

[47] UNECE Regulation No. 101, Adendum 100, Revision 3. Agreement Concerning the Adoption of Uniform Technical Prescriptions for Wheeled Vehicles, Equipment and Parts which can be fitted and/or be used on Wheeled Vehicles and the Conditions for Reciprocal Recognition of Approvals Granted on the Basis of these Prescriptions. UNECE, Geneva, Switzerland, 2013. Retrieved from http://www.unece.org/fileadmin/DAM/trans/main/wp29/wp29regs/2015/R101r3e.pdf, 05/04/2017.

[48] L. A. van Dongen, R. van der Graaf, and W. H. Visscher. Theoretical prediction of electric vehicle energy consumption and battery state-of-charge during arbitrary driving cycles. In *EVC Symposium VI*, pages 1–13. EVC Paper No. 8115, Baltimore, USA, 1981.

[49] R.-J. Wai, C.-Y. Lin, R.-Y. Duan, and Y.-R. Chang. High-efficiency DC-DC converter with high voltage gain and reduced switch stress. *IEEE Transactions on Industrial Electronics*, 54(1):354–364, 2007.

[50] A. Willemer. *Einstieg in C++: [umfassender Einstieg in die Programmierung ; einfacher Zugang zur OOP und zur STL ; auf CD-ROM: Compiler, IDE und alle Beispiele]*. Galileo computing. Galileo Press, Bonn, 4. edition, 2013.

[51] S. Winter. Modeling costs of turns in route planning. *GeoInformatica*, 6(4):345–361, 2002.

[52] X. Wu, D. Freese, A. Cabrera, and W. A. Kitch. Electric vehicles' energy consumption measurement and estimation. *Transportation Research Part D: Transport and Environment*, 34:52–67, 2015.

[53] J. Yamakawa, A. Kojima, and K. Watanabe. A method of torque control for independent wheel drive vehicles on rough terrain. *Journal of Terramechanics*, 44(5):371–381, 2007.

[54] D. Yu. Mechanical Efficiency of Differential Gearing. *Gear Technology*, pages 9–16, 1986.