Design and Implementation of an iPhone Application to Determine the Parking Position of a Car by Trajectory Analysis

*A **thesis** presented for the degree of **Bachelor of Science**.*

Benedikt Maria Beckermann, Freie Universität Berlin, Germany

Matriculation number: 5040175

benedikt.beckermann@fu-berlin.de

27th October 2019

Supervisor:

Nicolas Lehmann[1], Freie Universität Berlin, Germany

Reviewers:

Prof. Dr. Agnès Voisard[2], Freie Universität Berlin, Germany

Prof. Dr. Katinka Wolter[3], Freie Universität Berlin, Germany

Citation:

Benedikt Maria Beckermann, *Design and Implementation of an iPhone Application to Determine the Parking Position of a Car by Trajectory Analysis*, Freie Universität Berlin, Bachelor Thesis, 2019

Version: Final Version

[1]Dept. of Computer Science and Mathematics, Databases and Information Systems Group
[2]Dept. of Computer Science and Mathematics, Databases and Information Systems Group
[3]Dept. of Computer Science and Mathematics, Dependable Systems Group

Statutory Declaration

I declare that I have developed and written the enclosed Bachelor thesis completely by myself, and have not used sources or means without declaration in the text. Any thoughts from others or literal quotations are clearly marked.

The Bachelor thesis was not used in the same or in a similar version to achieve an academic grading or is being published elsewhere.

Berlin (Germany), 27th October 2019

Benedikt Maria Beckermann

Zusammenfassung

In Großstädten sind Parkplätze schwer zu finden, da die Anzahl privater Fahrzeugen zunimmt und die verfügbare Parkfläche nicht effektiv genutzt wird. Fahrer haben Probleme ihr geparktes Auto wiederzufinden und nutzen dadurch Parkplätze länger als nötig während sie ihr Auto suchen. Ein System, welches Fahrer dabei unterstützt ihr Auto zu finden, würde dabei helfen unnötig genutzte Parkplätze schneller freizugeben und folglich die Parkplatznot in Großstädten lindern. In der vorliegenden Bachelorarbeit wird eine mobile Applikation entworfen und implementiert, welche die Parkposition des Autos eines Fahrers automatisch anhand seiner Raumtrajektorie bestimmen kann. Um die Position des Autos zu determinieren, wird ein Algorithmus entworfen, der auf der Klassifikation der Transportationsmethode der Raumtrajektorie des Nutzers aufbaut. Die Applikation ist mit nutzerzentrierten Methoden entworfen.

Abstract

Parking spaces in major cities are hard to find as the number of private vehicles increases and the available land for parking spaces is not efficiently used. As drivers have sometimes difficulties to find their parked car and thus use a parking spot longer than necessary while searching for their car, a system to support users to find their parked car can help to reduce the lack of available parking spaces. In this Bachelor thesis an application to determine the parking position of a user's car is designed and implemented. To determine the position of the user's parked car, an algorithm is designed that is based on the classification of the transportation mode of the user's spatial trajectory. The application is designed using user-centric methods.

Table of Contents

1. Introduction **1**
- 1.1. Motivation . . . 1
- 1.2. Outline of Contribution . . . 2
- 1.3. Structure of Thesis . . . 2

2. Background **3**
- 2.1. Definitions . . . 3
- 2.2. Methodology . . . 4
- 2.3. Related Systems . . . 5
- 2.4. Related Work . . . 7

3. Analysis **10**
- 3.1. Background Interviews . . . 10
- 3.2. Requirements . . . 12
  - 3.2.1. Functional Requirements . . . 12
  - 3.2.2. Non-functional Requirements . . . 13

4. Design **15**
- 4.1. User Interface . . . 15
  - 4.1.1. Iterations . . . 15
  - 4.1.2. Final User Interface . . . 18
- 4.2. Determining the Parking Position of a Car . . . 23
  - 4.2.1. Noise Removal . . . 24
  - 4.2.2. Stay Point Detection . . . 25
  - 4.2.3. Trips . . . 25
  - 4.2.4. Determining Parking Position Candidate . . . 26
- 4.3. Transportation Mode Classification of Spatial Trajectories . . . 27
  - 4.3.1. Pre-processing . . . 27
  - 4.3.2. Feature Extraction . . . 27
  - 4.3.3. Data Validation . . . 29
  - 4.3.4. Model Training . . . 29

5. Implementation **33**
- 5.1. Architecture . . . 33
- 5.2. Technologies . . . 34
- 5.3. Details of Implementation . . . 35

6. Evaluation **41**
- 6.1. Evaluation of Machine Learning Models . . . 41
- 6.2. Accuracy of Determined Parking Position . . . 42
- 6.3. Fulfillment of Requirements . . . 43

7. Results **45**
7.1. Summary 45
7.2. Limitations 47

8. Conclusion **48**
8.1. Discussion of Results 48
8.2. Future Work 48

9. References **50**

A. User Interface Iterations **53**

B. Source Code of Implementation **62**

List of Tables

4.1. Visualization of Differences Between Windows and Moving Windows . 28
4.2. The Used Thresholds for Velocity and Acceleration [30] 29

6.1. Evaluation of XGBoost Model 41
6.2. Evaluation of Multilayer Perceptron Model 41
6.3. Evaluation of Parking Position Determination Accuracy 42

List of Figures

2.1. Apple Maps Parked Car, Image from Apple Support Page: https://support.apple.com/en-us/HT207227 (online: last accessed October 19, 2019) . . . 6

4.1. Iteration 1: MapView - Parking Position Determined (left), Iteration 2: MapView - Parking Position Determined (right) . . . 16
4.2. Iteration 3 - MapView - Parking Position Determined (left), Iteration 4 - MapView - Parking Position Determined (right) . . . 17
4.3. MapView - Loading . . . 19
4.4. MapView - Error . . . 20
4.5. MapView - Parking Position Determined . . . 21
4.6. Feedback . . . 22
4.7. Feedback - Confirmation . . . 22
4.8. Feedback - Failure . . . 23
4.9. Feedback - Success . . . 23
4.10. Basic Multilayer Perceptron Schema . . . 30

5.1. Architecture of Developed System . . . 33
5.2. The Three Main Views of the Implemented Application. MapView - Loading (left), MapView - Parking Position Determined (middle), and Report Accuracy (right). . . . 36

A.1. Iteration 1: Storyboard . . . 53
A.2. Iteration 1: MapView - Loading . . . 53
A.3. Iteration 1: MapView - Error . . . 53
A.4. Iteration 1: MapView - Parking Position Determined . . . 54
A.5. Iteration 1: MapView - Details . . . 54
A.6. Iteration 1: Feedback . . . 54
A.7. Iteration 2: Storyboard . . . 55
A.8. Iteration 2: MapView - Loading . . . 55
A.9. Iteration 2: MapView - Error . . . 55
A.10. Iteration 2: MapView - Parking Position Determined . . . 56
A.11. Iteration 2: Feedback . . . 56
A.12. Iteration 2: Feedback - Confirmation . . . 56
A.13. Iteration 3: Storyboard . . . 57
A.14. Iteration 3: MapView - Loading . . . 57
A.15. Iteration 3: MapView - Error . . . 57
A.16. Iteration 3: MapView - Parking Position Determined . . . 58
A.17. Iteration 3: Feedback . . . 58
A.18. Iteration 3: Feedback - Confirmation . . . 58
A.19. Iteration 3: Feedback - Failure . . . 58
A.20. Iteration 3: Feedback - Success . . . 59
A.21. Iteration 4: Storyboard . . . 59

A.22.Iteration 4: MapView - Loading 60
A.23.Iteration 4: MapView - Error 60
A.24.Iteration 4: MapView - Parking Position Determined 60
A.25.Iteration 4: Feedback 60
A.26.Iteration 4: Feedback - Confirmation 61
A.27.Iteration 4: Feedback - Failure 61
A.28.Iteration 4: Feedback - Success 61

List of Source Codes

4.1. Pseudocode: Determine Parking Position Candidate 23
4.2. Pseudocode: Noise Removal [1] 24
4.3. Pseudocode: Compute Mean Coordinate [1] 24
4.4. Pseudocode: Stay Point Detection [2] 25
4.5. Pseudocode: Determine Trips in a Trajectory 26
4.6. Pseudocode: Determine Parking Position Candidate 26
4.7. Pseudocode: Basic XGBoost 31
4.8. Pseudocode: XGBoost - Approximate Algorithm for Split Finding [3] . 32

5.1. Train XGBoost Classifier 36
5.2. Implementation of Algorithm to Determine Parking Position of the User's Car 37
5.3. Determine Stay Points - Implementation 38
5.4. Server: Handling of Reported Accuracy 39

1. Introduction

The first chapter, *Introduction*, introduces the domain of parking spaces, explains why it is useful to find a parked car in a short period of time and summarises the contribution of this Bachelor thesis for finding a parked car based on trajectory analysis. The first section, *Motivation*, outlines the problem of overused parking spaces in major cities. It shows why it can be difficult to find a parked car and how the period of time a driver needs to find their parked car affects the number of unoccupied parking spaces. The second section, *Outline of Contribution*, presents the main features of the developed mobile application to support a user in finding their parked car and the designed algorithm to determine the parking position of the user's car based on trajectory analysis. The third section, *Structure of Thesis*, summarised the content of each chapter of this Bachelor thesis.

1.1. Motivation

Finding an unoccupied parking space in a major city is becoming more difficult as the number of private vehicles increases and the available land for parking spaces is not efficiently used. It is estimated that 30% of driving vehicles in cities are searching for a parking space. A decrease in parking time leads to more available parking spaces. [4][5][6]

Drivers are more focused on the appointment they are heading to than the exact location where they park. The frustration caused by the difficulty to find an unoccupied parking space increases this lack of attention to the parking position. Out of frustration, any available space is chosen but its location may be forgotten while heading to ones appointment. Hence, it can be time-consuming to find the position where the car is parked after the appointment. Parking spaces are longer occupied than necessary.

Example 1 *Alexandra Smith has been invited for a job interview at Boho Technologies in London. The public transport in the village she lives in is not performing well. She decides to go to the interview by car. Half an hour before the appointment takes place, she arrived at the company and is looking for a parking space. After ten minutes she finds one in a small side street. She parks there and makes it just in time to the job interview. After the successful appointment, she cannot remember where her car is parked. She needs more than 30 minutes to find it. When she finds it, she notices an 80£ fine on her car because she stayed longer than allowed in the parking space.*

Example 1 presents the challenge of finding a parked car in a major city. Parking spaces are longer occupied than necessary. A system to support the drivers in finding their cars is needed to help reducing the unnecessary utilisation of highly needed parking spaces in major cities.

1.2. Outline of Contribution

As part of this Bachelor thesis an iPhone[1] application is developed that supports drivers of cars to locate their parked car. The application uses spatial trajectories, based on the Global Positioning System Standard Positioning Service, to model each trip of the driver to determine the parking position of the car. The application shows the location of the parked car and the location of the user on a map. All calculations involving the spatial trajectory of the user are computed on the mobile device itself to ensure privacy.

To enable the application to determine the parking position of a car, an algorithm is developed to determine the parking position of the user's car based on the user's spatial trajectory. The spatial trajectory of the user is split into trips. Based on the transportation mode of the user, the parking position of their car is determined. The transportation mode is classified using a machine learning model.

1.3. Structure of Thesis

Chapter two, *Background*, defines terms necessary to understand this Bachelor thesis, explains the methodology applied for analysing the problem and discusses related systems as well as related academic work. In the third chapter, *Analysis*, background interviews are conducted to identify the needs and expectations of users for an application that determines the parking position of a car. Based on the results of the interviews and the constraints of this Bachelor thesis, requirements are defined for the application. The fourth chapter, *Design*, presents the results of the user interface design, the created algorithm to determine the parking position of a user's car and the used machine learning workflow. The fifth chapter, *Implementation*, describes the implementation of the training of the machine learning model used for classifying the transportation of the user and the implementation of the developed application. The sixth chapter, *Evaluation*, evaluates the results of this Bachelor thesis. The seventh chapter, *Results*, summarises the outcome of this Bachelor thesis and presents its limitations. The eight chapter, *Conclusion*, discusses the results of this Bachelor thesis and potential future work. The ninth chapter, *References*, lists all used external sources. Appendix A, *User Interface Iterations*, presents all views of the four iterations of the user interface design. Appendix B, *Source Code of Implementation*, lists the content of the appended DVD and links to the source code repositories.

[1] iPhone: https://www.apple.com/iphone/, (online: last accessed October 19, 2019)

2. Background

The second chapter, *Background*, explains the background necessary to understand the Bachelor thesis. In the first section, *Definitions*, terms are defined that are used in this Bachelor thesis. The second section, *Methodology*, presents the methodology applied to analyse the problem of finding a parked car and used to improve the user interface design. In the third section, *Related Systems*, applications for iOS[1] are presented, that solve the problem of finding a parked car. In the fourth section, *Related Work*, academic work that relates to the bachelor thesis is discussed.

2.1. Definitions

Spatial Trajectory A spatial trajectory is a trace of an object through a geographical space. It is usually represented as a chronologically ordered list of points $P = p_1 \rightarrow p_2 \rightarrow \cdots \rightarrow p_n$. The points consist of a geospatial coordinate set and a timestamp $p_i = (lat, lon, t)$. [1]

Semantic Trajectory A semantic trajectory is defined as a finite spatial trajectory with a semantic meaning, i.e. the used transportation mode $((p_1 \rightarrow p_2 \rightarrow \cdots \rightarrow p_n), label)$. [1]

Trajectory Segmentation In trajectory segmentation, a trajectory is divided into fragments by a measure such as time, spatial shape, or semantic meaning. These fragments can be used for further processing such as classification. [1]

Trajectory Classification The process of predicting the class label of trajectories, or their segments, based on their features is called trajectory classification. [7]

Stay Point A stay point s is defined as a geographical location in which a user stays over a threshold time T_r within a threshold distance D_r. In a spatial trajectory, a stay point s is characterised by a set of consecutive points $P = p_m \rightarrow p_{m+1} \rightarrow \cdots \rightarrow p_n$ with $\forall m < i \leq n : \ Dist(p_m, p_i) \leq D_r$, $Dist(p_m, p_{n+1}) > D_r$, and $t(p_n) - t(p_m) \geq T_r$. The stay point s is then defined by $s = (lat, lon, t_a, t_l)$ where $lat(s) = \sum_{i=m}^{n} lat(p_i)/|P|$, $lon(s) = \sum_{i=m}^{n} lon(p_i)/|P|$, $t_a(s) = t(p_m)$, and $t_l(s) = t(p_n)$. [8]

Segment A segment is defined as the edge between any two consecutive points in a spatial trajectory. In a spatial trajectory $P = p_1 \rightarrow \cdots \rightarrow p_n$, each edge $p_i \rightarrow p_{i+1}, 1 \leq i < n$ is called a segment. [1]

[1] iOS: https://www.apple.com/ios/ios-13/, (online: last accessed October 19, 2019)

Trip A trip T_i is defined as the subset of a spatial trajectory between two consecutive stay points. $T_i = \{p \in P | \ s_i.l \leq p.t \leq s_{i+1}.a\}$, where T_i is the set of all spatial trajectories of a trip, P is the set of all recorded spatial trajectories of a user, and s_i, s_{i+1} are two consecutive stay points of the user. The index i is a natural number greater or equal zero. [9]

Stage A stage is a group of successive segments of the same transportation mode within a trip. A new stage starts where the transportation mode changes to another, or where the used vehicle is changed. [10]

Transportation Mode The transportation mode is the way in which an object moves through a space, such as walking, via car or via bus. [10]

Change Point A change point is defined as the location where the transportation mode of a trajectory changes within a trip. [9]

Global Positioning System Standard Positioning Service The Global Positioning System Standard Positioning Service is a positioning and timing service. It uses ranging signals transmitted by satellites. The signals contain a coarse/acquisition (C/A) code ranging signal and a navigation data message. [11]

Precision In the context of machine learning, the precision is the number of correctly positive labeled instances divided by the sum of the correctly positive labeled instances and the wrongly positive labeled instances. [12]

Recall The recall is the number of correctly positive labeled instances divided by the sum of the correctly positive labeled instances and the wrongly negative labeled instances. [12]

F1-Score The F1-score, also called the F-measure, is the harmonic mean of precision and recall. It is defined as $f_1 = (2 \cdot precision \cdot recall)/(precision + recall)$. [13]

2.2. Methodology

Four people are interviewed for background interviews. The goal of these interviews is to gain insight into the ”needs and expectations of users”. The interviews are performed using a semi-structured interview technique. In semi-structured interviews, the interviewer has a list of questions and the interviewee is encouraged to elaborate their answers. The interviewer can ask follow-up questions if more insight into a specific answer is needed. This technique is used because the detailed answers of the users allow insight into their expectations of the application and the option for asking follow-up questions avoids imprecise answers. A structured technique would

not allow the users to mention their own expectations and issues. An unstructured technique would make the answers of the interviewees difficult to compare. The answers given in the interviews are summarised. Based on the results of the interviews and the limitations of this Bachelor thesis, requirements for the application are defined. The requirements are separated into functional and non-functional requirements. [14] [15]

To ensure the usability of the application, the user interface is created iteratively. Best practises are applied by using design heuristics by Nielsen and conforming to the Human Design Guidelines by Apple Inc[2]. Iterative design is defined as ”progressive refinement through cyclical data-driven development”. During each iteration, users test the current design prototypes. For these tests, the Think-Aloud method is applied. Think-Aloud is ”a special kind of verbal protocol in which the user says out loud what she is thinking while she is carrying out a task or doing some problem solving”. The method is chosen because it can lead to less design interactions while still resulting in efficient systems that are accepted by users. In the tests themselves, a prototype of the user interface is provided to the users. The users are asked to perform five different actions in the prototype and thereby think aloud. First, they are asked to determine the parking position of their car. Second, they are asked to use the navigation to find to their car. Third, they are asked to imagine their car is parked in a multi-storey parking lot and to answer how the application would help them to find their car in this scenario. Fourth, the users are asked to report the accuracy of the determined parking position. Fifth, they are asked to refresh the determined parking position. They are encouraged to elaborate what they are thinking and reasoning while using the prototype. After the tests are conducted, the protocols of the tests are analysed for potential obstacles in the user interface. The results are aggregated and summarised. The obstacles in the user interface are removed in the next iteration. The tests are repeated with the next prototype to ensure the changes have a positive impact on the usability of the application. [14] [16] [17] [18] [19] [20]

2.3. Related Systems

Apple Maps[3], which is preinstalled on iOS devices, supports the determination of the parking position of a car on an iPhone. The parking position is determined via the phone’s connection to the car. When the Bluetooth connection or the CarPlay[4] connection, a proprietary way to connect cars with an iPhone, is interrupted, the phone logs the current position as the parking position of the car. The user can manually change and optimise the logged position, which can be seen in Figure 2.1, and add a photo to the saved location to simplify the process of finding the car. The approach used by Apple Maps is usable by users who connect their phone to their car regularly.

[2] Apple Inc: https://www.apple.com/ (online: last accessed October 19, 2019)
[3] Apple Maps, https://www.apple.com/ios/maps/, (online: last accessed October 19, 2019)
[4] CarPlay: https://www.apple.com/ios/carplay/ (online: last accessed October 19, 2019)

It is not usable by users who either do not or cannot connect their phone to their car as there is no alternative solution for these users in Apple Maps. [21]

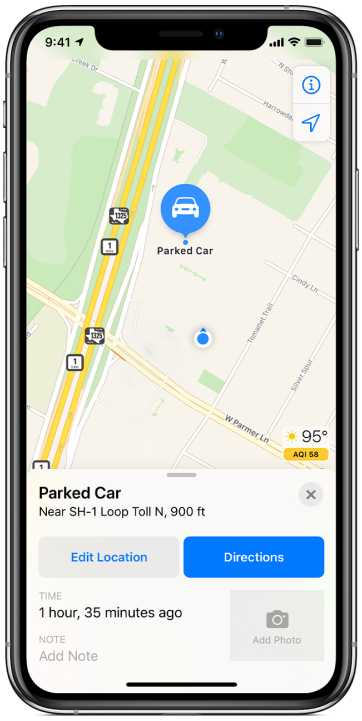

Figure 2.1.: Apple Maps Parked Car, Image from Apple Support Page: https://support.apple.com/en-us/HT207227 (online: last accessed October 19, 2019)

Google Maps[5] also offers the feature to determine the parking position of a user's car on an iPhone. Three different approaches are implemented. First, the application can determine the parking position automatically via the motion and fitness activity data of the user. The used process for the determination via the Motion and Fitness Activity data is not public. The mentioned motion and fitness activity data include data from the iPhone's accelerometer, gyroscope and magnetometer. The second approach is, similar to Apple Maps, based on the phone's connection to the user's car. The third approach is to let the user manually set the parking location. For this, the user taps their location on the map view and chooses to set the current position as parking location. Also, at the end of a navigation, the user can select their current position to be saved as parking position of their car. As Google Maps does not rely solely on the direct connection of the phone to the car, it is usable by any user, independent from the user's willingness or possibility to connect their phone to their car. [22] [23]

There are several application n the Apple AppStore[6] which are dedicated to finding the parked car of the user. Find my Parked Car[7], Find My Car - Car Locator[8], Find

[5] Google Maps, https://apps.apple.com/us/app/google-maps-transit-food/id585027354, (online: last accessed October 19, 2019)
[6] Apple AppStore, https://www.apple.com/ios/app-store/, (online: last accessed October 19, 2019)
[7] Find my Parked Car, https://apps.apple.com/us/app/find-my-parked-car/id987712319, (online: last accessed October 19, 2019)
[8] Find My Car - Car Locator, https://apps.apple.com/us/app/find-my-car-car-locator/id1295565906,

My Car - GPS Auto Parking Reminder & Tracker[9] and ParKing P[10] approach the determination of the user's car by relying on input by the user. ParKing P also offers an approach based on the Bluetooth connection of the phone to the car. During the research for this Bachelor thesis, no application is found that determines the parking position of a car based on the spatial trajectory of a user.

2.4. Related Work

[1] is a survey on the field of trajectory data mining. It presents a categorisation of trajectory data mining, ilustrates each category briefly and lists existing research of each category.
The paper explains the high level concepts of trajectory data mining, including stay-point detection, trajectory segmentation and trajectory classification that are used in the thesis to classify the transportation mode of a user's spatial trajectories. It is also referred to as a source of several definitions, such as spatial trajectory, trajectory segmentation, and segments.

[24] is a survey on approaches for transportation mode detection. It introduces a classification of the approaches into three classes and lists the most influential academic work for each class. The first class, Location-based Services (LBS), focuses on real time classification of a transportation mode and is mostly used for systems which provide direct feedback to the users. The second class, Transportation Science (TSc), focuses on travel patterns of one or more individuals. Thus, the data do not need to be processed in real time and more context can be taken into account. The third class, Human Geography (HG), focuses on enriching trajectories with more semantical meaning.
The algorithm to determine the parking position of a car based on trajectory segmentation used in this Bachelor thesis is to be categorised as LBS and HG, as it aims to provide the parking position of a car, which is additional semantical meaning, in near real time when the user requests the position of their parked car.

[9] presents an approach to infer the transportation mode only from GPS trajectories. For this, the authors first segment a spatial trajectory into trips. Next, the individual trips are segmented into walk segments and non-walk segments based on their average speed. The non-walk segments are then classified into different transportation modes: car, bus, and bike. In the last step, the probabilities of the transportation modes are adjusted based on the transportation mode of the previous segment.
In the context of the determination of the parking position of a car, this Bachelor thesis relies on the transportation mode classification of the user's spatial trajectory. This

(online: last accessed October 19, 2019)

[9] Find My Car - GPS Auto Parking Reminder & Tracker, https://apps.apple.com/us/app/find-my-car-gps-auto-parking-reminder-tracker/id815542507, (online: last accessed October 19, 2019)

[10] ParKing P, https://apps.apple.com/us/app/parking-p-find-my-parked-car/id1233326914?l=en, (online: last accessed October 19, 2019)

paper is one of the first academic work on the transport mode classification based on the spatial trajectories of a user. Thus, it builds the base for most further research in this domain.

[25] is a direct continuation of the author's previous work [9]. It improves the features and the post-processing process. The newly introduced features are heading change rate, velocity change rate and stop rate. The improved post-processing process segments the likelihood for a transportation mode into three categories and optimises the results based on the assigned category.
The newly introduced features are not applicable to the chosen approach in this Bachelor thesis, as they assume a pre-segmentation into stages, before the stages are classified.

[26] classifies the transportation mode of a spatial trajectory based on tree-based ensemble methods. It does not rely on a pre-segmentation into walk and non-walk segments. The used tree ensemble based machine learning models, such as Random Forest, Gradient Boosting Decision Tree, and XGBoost show better performance than the best models of previous work, such as in [9]. The authors present 111 different features to extract from trajectory segments and reach a maximum recall of over 90%. A limitation of the paper is the missing approach to segment unlabelled trajectories into stages which can then be classified. The paper assumes the data to be available in pre-segmented stages.
The paper can be understood as a continuation of the work described in [9] with more modern machine learning models and more exhaustive features. This Bachelor thesis uses the machine learning model XGBoost which has the best performance reported in the paper and also adapts the most important features shown in the paper which are applicable to the used approach. [3]

In [10] the transportation modes of spatial trajectories are classified by using a moving window approach. Based on these classifications, stages are defined. Only speed and acceleration are used as features as they are the most discriminatory. The paper introduces the idea of sparse data sets to reduce computation costs and to save battery power on the trajectory generating device. It is concluded that 30 to 60 second intervals seem to be sufficient.
The paper is highly influential to this Bachelor thesis. The moving window approach is used to avoid relying on heuristics to divide trips into segments of the same transportation mode.

[7] presents a framework to generate features for the classification of spatial trajectories. The framework distinguishes between region-based and trajectory-based features.
The results of that paper are only partly applicable for this Bachelor thesis, as the trajectory classification in the thesis solely focuses on trajectory-based features. Region-based features are avoided to ensure the independence of the transportation mode classification from the locations where the spatial trajectories are generated.

[27] presents the social networking service GeoLife by Microsoft Research Asia. The goal of the project is to improve the understanding of trajectories, users and locations.

One of the aspects of the trajectory understanding is the transportation mode classification. In the project, spatial trajectories of 182 people are recorded from April 2007 to August 2012.
The data from this project is used in this Bachelor thesis to train the machine learning model that classifies the transportation mode of a user's trajectory segments.

The data set [25][27][28][29] contains GPS trajectories collected for the GeoLife project from Microsoft Research Asia. 91% of the reported trajectories consist of relatively dense points which are reported every two to five seconds or five to ten meters. The data is created by 182 users over five years. 69 of the users labeled their trajectories with the used transportation mode.
For the purpose of this thesis, the data set is used to train a machine learning model which classifies the transportation modes of a user's trajectory. Based on the change points and the transportation modes of the adjacent trip segments, the parking position of the driver's car is determined.

[30] discusses a convolutional neural network approach to classify the transportation mode of spatial trajectories. It is argued that this approach avoids the manual creation of high level features that are error prone and not exhaustive. They reach an accuracy of 84.8%.
The paper describes the pre-processing steps of the training data for the machine learning model in detail. It introduces a limit for the average acceleration and speed of a stage for each transportation mode to filter wrongly labeled data. The values used are the basis of a pre-processing step in this Bachelor thesis.

3. Analysis

In the third chapter, *Analysis*, the needs and expectations of users for an application to determine the parking position of a car are identified and requirements for such an application are defined. In the first section, *Background Interviews*, users are interviewed to identify the needs and expectations they have for an application to determine the parking position of their car. In the second section, *Requirements*, the requirements of the application are defined based on the results of the background interviews and the constraints of this Bachelor thesis.

3.1. Background Interviews

To identify the needs and expectations of users, four users are interviewed for background interviews. The background interviews are semi-structured. All people who drive cars are potential users of the application. Because the application solves a specific problem that is applicable to most drivers, the target audience is not segmented into smaller groups. Four people are interviewed who are part of the target audience. The interviewed people are called participants. All participants volunteer for the interview without any extrinsic incentive, and agree to be recorded and quoted anonymously. [14] [15]

A nomenclature is introduced to identify the questions. The questions are identified with ”[Q.xx]”, where Q stands for question, ”.” separates the letter from the number and ”xx” is a natural number starting at 01. The brackets ”[]” limit the term.

[Q.01] In which situations in your past could you not find your parked car? The two most mentioned locations where users have trouble finding their car are big parking lots, mentioned by three of the four participants, and multi-storey parking lots, mentioned by two of the four participants.

[Q.02] Imagine you cannot remember where exactly you parked your car. Could you describe how you would approach finding your car? When asked to describe their approach to find their parked car if the exact location is not known, three out of four participants say they would walk around the area where the car should be. Two out of four would use the remote key of the car to unlock and lock the car to create visual and auditory feedback to indicate if the car is nearby.

[Q.03] Please list the two or three features which are most important for you in an application that helps you to find your parked car. The two most important aspects of the application, each mentioned by two of the four participants, are the accuracy of the determined parking position and the independence from manual user input, such as the manual logging of the parking position.

[Q.04] How accurate do you expect the application to be? All of the participants express that an accuracy of 10 meters is sufficient for the determined parking position. Two of the four participants are also satisfied with an accuracy of 50 meters.

[Q.05] Which information should the application present to you? All participants expect the application to show the position of their parked car. Two out of four participants also expect to see their current position and the position of their parked car on a map.

[Q.06] What would prevent you from using the application? Required manual user input or a complicated user interface would each prevent two of four participants to use the application. A lack of clarity about the use of the generated user data and bad reviews in the AppStore would each prevent one participant to use the application.

[Q.07] How do you feel about the application using your location data? For three of the four participants the use of the location data is not an issue. The remaining participant accepts the usage of their location data if the application openly communicates how the generated data is used.

[Q.08] Would it be reassuring for you to know that no location data is sent to the cloud? All participant would be reassured if the location data is only used on the device itself, but none of them define this aspect as a requirement. They would also use the application, if the data is uploaded and processed online.

[Q.09] Since when do you drive cars? Two of the participants drive cars since eighth years. The other two participants drive since six and twelve years.

[Q.10] How regularly do you drive cars? Three of four participants drive currently only a few times a year a car. One participant drives weekly. Two of four mention that they used to drive daily. The other two participants mention that they used to drive monthly or weekly.

[Q.11] Did you use applications which help you to find your car before? Only one of the four participants previously used an application to help them find their car. This person used Google Maps.

[Q.12] Are you aware that Google Maps and Apple Maps offer a feature to help finding a parked car? Only one of the participants is aware that Google Maps and Apple Maps provide the functionality to assist in finding a user's parked car.

[Q.13] Do you drive more often in cities or rural areas? A city is defined as a town with a population greater than 100.000 people. Two of the participants report that they drive more often in urban areas. The other two either drive in rural and urban areas equally or primarily in rural areas.

3.2. Requirements

The requirements are split into functional and non-functional requirements. A functional requirement ”specifies a function that a system or system component must be able to perform.”. A non-functional requirement is defined as a requirement that is not a functional requirement, such as data requirements, constraints and quality requirements. [31]

To identify the requirements, a nomenclature is introduced. The requirements are uniquely identified with ”[XX.yy]”. XX can either be equal to ”RF” or ”RN”. ”RF” stands for functional requirement and ”RN” stands for non-functional requirement. ”yy” is a positive natural number starting at 01. ”.” separates the letters from the number. The brackets ”[]” limit the term.

3.2.1. Functional Requirements

[RF.01] The application determines the parking position of the user’s car using spatial trajectory analysis The parking position is determined by analysing the spatial trajectory of the user. The spatial trajectory, based on the GPS sensor of the phone, is segmented and classified into the used transportation mode of the user and a heuristic is used to determine the parking position based on this segmentation. The requirement is built upon the constraints of this Bachelor thesis.

[RF.02] The application presents the determined parking position of the user’s car on a map The parking position of the user’s car is presented on a map to enable the user to easily orientate themselves in the environment. The requirement is based on the users’ answers to [Q.05].

[RF.03] The application presents the current position of the user on a map The current user location is presented on a map to enable the user to orient themselves in their current environment. The requirement is based on the users’ answers to [Q.05].

[RF.04] The application presents detailed information about the determined parking position, namely the address, the distance from the current location and the relative altitude The detailed information of the determined parking location is presented to the user to enable them to find the most fitting way to reach their parked car. The requirement is based on the users’ answers to [Q.05].

[RF.05] The application enables the user to report the accuracy of the determined parking position to the developer To enable the user to report the accuracy of the determined parking position, a view is implemented. The accuracy is defined as the distance between the determined parking position of the user's car and the actual parking position. This requirement is necessary to enable the quantitative evaluation of the parking position accuracy.

3.2.2. Non-functional Requirements

[RN.01] The application is easy to use The user interface of the application does not confuse the user and presents all options in an easy to understand way. The user is always aware of the state of the application. This requirement is based on the users' answers to [Q.06].

[RN.02] The application does not send any location data to server components of the implemented system To ensure the privacy of the user and to avoid collecting sensible information, the application does not send any location data to any server component of the implemented system. The requirement is based on the users' answers to [Q.08].

[RN.03] The application does not rely on manual user input for determining the parking position The application can determine the parking position of the user's car without the user being asked to manually input any data, such as the parking position or the transportation mode chosen. The process to determine the parking position is fully automated. The requirement is necessary due to half of the users reporting in [Q.06] that manual user input would prevent them from using the application.

[RN.04] The accuracy of the determined parking position is better than 50 meters The accuracy, defined as the distance between the determined parking position of the user's car and its actual parking position, is better than 50 meters. Users report the accuracy as one of the most important features of the application in [Q.03] and the majority agrees that an accuracy of better than 50 meters is sufficient in [Q.04].

[RN.05] The application is developed to run on iOS 13 The application is developed to run on iPhones with the latest iPhone operating system iOS 13. The requirement is based on the constraints of this Bachelor thesis.

[RN.06] The application is functional in big parking lots One of the most named locations in [Q.01] where users had problems finding their parked car are big parking lots, such as in front of major supermarkets or furniture stores. Thus, the application needs to support the user in finding their parked car in the mentioned environments.

[RN.07] The application is functional in multi-storey parking lots The other most named location in [Q.01] where users had problems finding their parked car are multi-storey parking lots. Thus, the application needs to support the user in finding their parked car in these environments by providing data to determine the floor on which the car is parked.

4. Design

The fourth chapter, *Design*, presents the results of the user interface design, the created algorithm to determine the parking position of a user's car and the applied machine learning workflow. The first section, *User Interface*, presents the four iterations of the user interface and explains the design decisions of the final user interface design. The second section, *Determine the Parking Position*, describes the algorithm used to determine the parking position of an user's car based on their spatial trajectory. The third section, *Transportation Mode Classification of Spatial Trajectories*, describes the process conducted to train a machine learning model that classifies the transportation mode of a spatial trajectory.

4.1. User Interface

The first section, *User Interface*, presents the designed user interface. First, the four iterations and the changes between them are described. Second, the final user interface design is discussed by considering usability heuristics by Nielsen and Apple.

4.1.1. Iterations

In the first part, *Iterations*, the different iterations of the user interface and their related Think-Aloud protocol results are presented. The full user interface of each iteration can be seen in Appendix A.

First Iteration

The first iteration consists of five views. The application starts in the "MapView - Loading" view (cp. Figure A.2). The view shows a map on which the location of the user is marked. At the top of the page a string indicates the current loading state of the system.

If the application fails to determine the parking position, a pop-up menu is shown (cp. Figure A.3) which notifies the user about the problem and explains the cause of the failure.

On successful determination of the parking position, the view "MapView - Determined Parking Postition" (cp. Figure 4.1) is shown. It shows a map with the user's location and the determined position of the user's car. The label of the user's car is clickable and opens "MapView - Details".

"MapView - Details" (cp. Figure A.5) presents more details about the determined parking position. It presents the address where the car is parked, the distance to the user's car, the altitude difference between the user and the user's car and the floor of

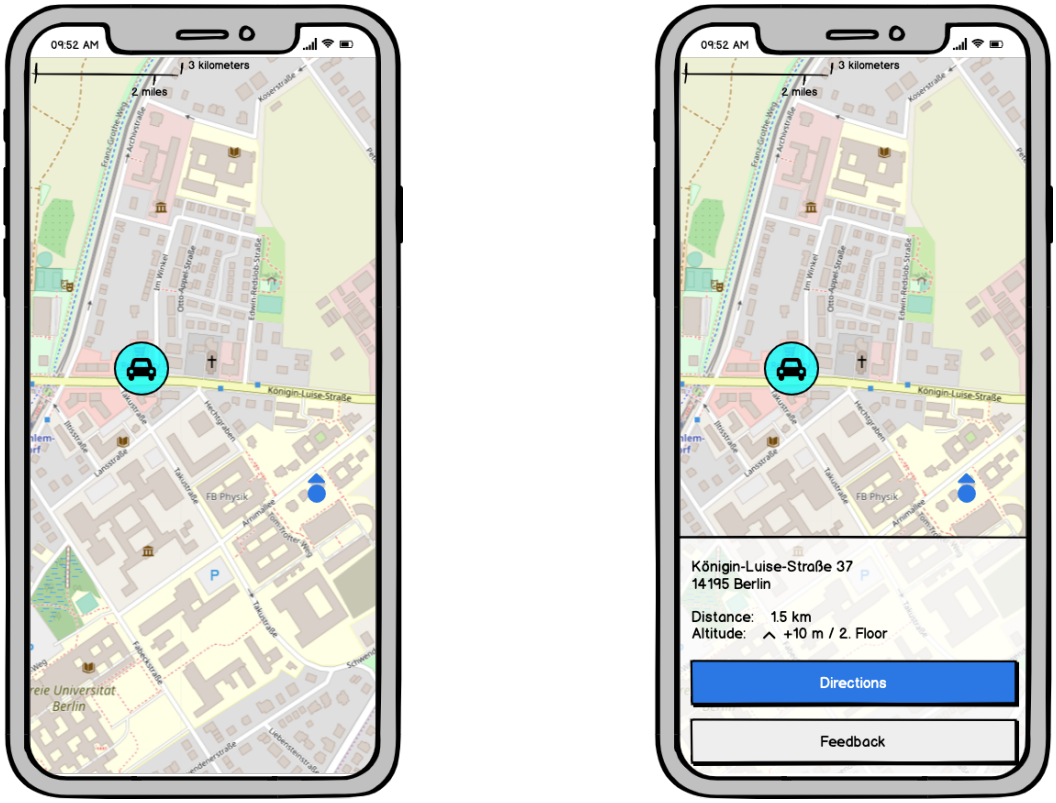

Figure 4.1.: Iteration 1: MapView - Parking Position Determined (left), Iteration 2: MapView - Parking Position Determined (right)

the parked car. Two buttons are presented: One button opens a third party navigation software to navigate to the determined location and the other button initiates the reporting of the accuracy of the determined parking position by opening the ”Feedback” view.

The ”Feedback” (cp Figure A.6) view presents the user instructions to report the accuracy of the determined parking position, the transmitted information and a button to report the information. When the button is clicked, the information is sent and the application returns to the previous view, ”MapView - Details”. The user can also go back to the previous screen without sending any information by using the ”Back” button.

The Think-Aloud protocol shows two key findings. First, in the view ”MapView - Parking Position Determined”, the user does not understand that the car icon is clickable. Second, the user expects to be able to give feedback in text form in the ”Feedback” view.

Second Iteration

In the second iteration of the user interface, the two problems of the first iteration are improved. The views ”MapView - Details” and ”MapView - Parking Position Determined” are merged. This change allows the user to see all details of the determined parking position directly. As the main view, ”MapView - Parking Position Determined”

(cp. Figure 4.1), now shows a partly transparent white box. This element is also added to the ”MapView - Loading” view to keep the user interface consistent. It contains a button to initiate the determination of the car. In the new ”MapView - Parking Position Determined” view, the call to action to give feedback is rephrased to ”Send Accuracy” to better represent the kind of feedback that can be given. The headline of the ”Feedback” (cp. Figure A.11) view is changed accordingly. A pop-up window is introduced in the ”Feedback” view to confirm the successful sending of the accuracy.

The Think-Aloud protocol of the second iteration shows three key findings. First, the altitude in the view ”MapView - Parking Position Determined” is understood as the total altitude of the car, but it shows the relative altitude of the car compared to the current altitude of the user. Second, the user is unsure if the presented address is the address of the location of the car, but assumes so. Third, the user does not notice in the ”Feedback” view that the accuracy is supposed to be reported at the actual parking position of the car.

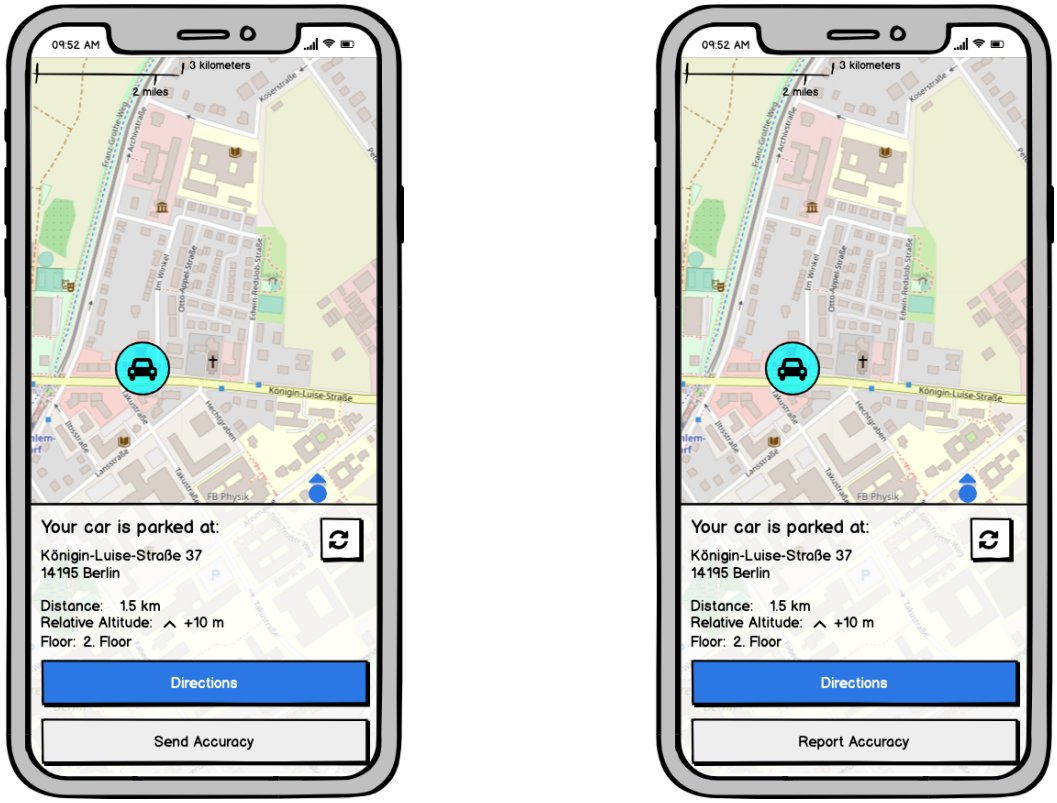

Figure 4.2.: Iteration 3 - MapView - Parking Position Determined (left), Iteration 4 - MapView - Parking Position Determined (right)

Third Iteration

The third iteration improves on the three obstacles of the previous iteration. First, in the view ”MapView - Parking Position Determined” (cp. Figure 4.2) the altitude information is split into two rows. The row ”Relative Altitude” shows the relative altitude between the user’s location and the car’s location. The row ”Floor” shows the floor of

the determined parking location, if the data is available. Second, the headline ”Your car is parked at:” is added to the view ”MapView - Parking Position Determined” to clarify that the presented information is about the determined parking position. Third, a pop-up window is introduced in the view ”Feedback” (cp. Figure A.18) to ensure the user is actually at the location of the parked car.

The Think-Aloud protocol shows one key finding. The call to action on the ”MapView - Parking Position Determined” view to report the accuracy of the determined parking position is still not clear. The user is confused because the wording is unusual for giving feedback to improve an application.

Fourth Iteration

To make the call to action to report the accuracy on the view ”MapView - Parking Position Determined” (cp. Figure4.2) more clear, it is rephrased to ”Report Accuracy”. The headline of the view ”Feedback” (cp. Figure A.25) is updated accordingly. To the view ”MapView - Parking Position Determined” a button is added to enable the user to refresh the determined parking position.

4.1.2. Final User Interface

The second part, *Final User Interface*, discusses the final user interface design. The different views are presented and design decisions argued based on the user interface heuristics by Apple Inc. and Nielsen. [17] [19]

MapView - Loading

The application starts in the ”MapView - Loading” view, shown in Figure 4.3. The view consist of a map and a button to initiate the determination of the parking position of the user’s car. The shown map supports the user to orientate themselves in their current environment by presenting their location on the map. Thus, the second heuristic from Nielsen, ”Match between system and the real world”, is implemented. The map is interactive and can be zoomed, moved and rotated. The button to determine the parking position of the user’s car is placed on top of a white and partly transparent box to ensure a sufficient contrast between the map and the button. The design fulfils Apples design guidelines for MapKit[1]. The basic design of a map with a white, partly transparent box is used for all of the applications views, except the ”Feedback” view, to keep the design simple and consistent, according to Nielsens fourth and eight heuristics. [17] [19]

[1] Apple Map Kit: https://developer.apple.com/documentation/mapkit, (online: last accessed October 19, 2019)

Figure 4.3.: MapView - Loading

MapView - Error

If the application fails to determine the parking position of the user's car, it shows the view "MapView - Error", shown in Figure 4.4. The view shows a map with the user's location. On the bottom, an error message is printed on top of a white, partly transparent box. The error message explains in an understandable way for the user, why the application failed to determine the parking position. Below the error text, a button is shown. When pressed, the application tries to determine the parking position again. With the understandable error message and the promoted button to retry, the application confirms to Nielsens ninth heuristic, "Help users recognize, diagnose, and recover from errors". [17]

MapView - Parking Position Determined

The view "MapView - Parking Position Determined", shown in Figure 4.5, is the main view of the application. It is shown when the application determined successfully the parking position of the user's car. A map and a partly transparent white box are shown. On the map, the user's current location and the position of the user's car are marked. The user's location is presented with a blue point. The location of the

Figure 4.4.: MapView - Error

car is marked with a stylised car surrounded by a transparent circle. The car icon is on the exact location, the application determined as the parking position and the circle represents the accuracy of the determined parking location. The map view is chosen to match the system to the surroundings of the user. On the bottom of the screen, detailed information of the determined parking location are presented. The address, the distance to the current location, the relative altitude and the floor in which the car is parked are shown. Above the details, the headline ”Your car is parked at” clearly identifies the listed information as related to the user’s car. Three buttons are shown. The first button, in the upper right corner of the white box, refreshes the determined parking location. The second button, labelled ”Directions”, opens a third party navigation software to navigate to the determined location. Because navigation apps, such as Apple Maps and Google Maps, use the color blue to indicate a button which starts a navigation, the color blue is also chosen for the ”Directions” button. The third button, labeled ”Report Accuracy”, initiates the process of reporting the accuracy of the determined parking location. The refresh button and the ”Report Accuracy” button are both colored white to not draw extra attention to them. Most users should not need to use them on a regular base. [17]

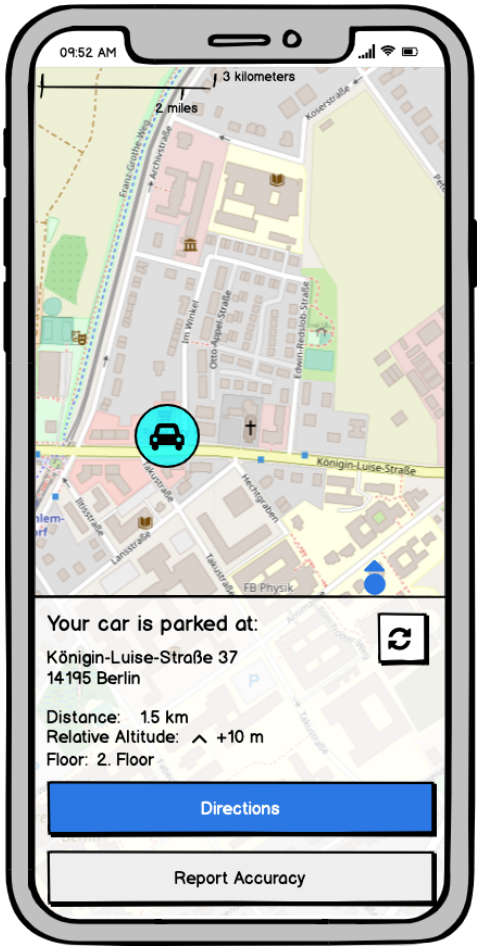

Figure 4.5.: MapView - Parking Position Determined

Feedback

The view ”Feedback”, shown in Figure 4.6, enables the user to report the accuracy of the determined parking location. This functionality enables the quantitative evaluation of the systems accuracy. The text in the upper half of the screen explains how to report the accuracy and what kind of data is transmitted. A complete list of the transmitted data is shown in the bottom half. The ”Back” button in the upper left corner navigates the system back to the view ”MapView - Parking Position Determined” without transmitting any data. The button on the bottom, labelled ”Report Accuracy”, brings up a pop-up window in which the user is asked, if they are at the location of the parked car (cp. Figure 4.7). The instructions in the explaining text are often skimmed by the users, thus this second check is necessary to keep the reported accuracy usable. If the user confirms their location, the data is send and a confirmation is shown (cf. Figure 4.9). The view changes back to the ”MapView - Parking Position Determined” view. If the user does not confirm their location, a pop up is shown (cp. Figure 4.8) which asks the user to report the accuracy at the location of their parked car. The ”Back” button enables the user to be in control if they want to share any information. The confirmation dialog prevents the user from accidental sending the accuracy at

Figure 4.6.: Feedback

Figure 4.7.: Feedback - Confirmation

the wrong location. Thus, the view fulfills the third and fifth heuristic by Nielsen. [17]

Figure 4.8.: Feedback - Failure

Figure 4.9.: Feedback - Success

4.2. Determining the Parking Position of a Car

The algorithm to determine the parking position of a user's car (Source Code 4.1) uses the heuristic that the user's car is parked where the user last traveled by car. To determine this location, the algorithm first removes the noise from the user's trajectory. The cleaned trajectory is segmented into individual trips based on the stay points of the trajectory. The trips are sorted from newest to oldest and examined for parking position candidates. The most recent of these candidates is returned as the determined parking position.

The algorithms applied in the parking position determination, namely the algorithms for noise removal, stay point detection, trip detection and parking position candidate detection are described in detail in the parts below.

```
def determineParkingPosition(trajectory, distTresh, timeTresh, car_window_size):
    trajectory <- removeNoise(trajectory)
    stayPoints <- detectStayPoints(trajectory, distTresh, timeTresh)
    trips <- cutIntoTrips(trajectory, stayPoints)

    for t in trips.reversed():
        labels <- classifyTransportationModes(t)
```

```
        parkPos <- detParkingPosCandidate(t, labels, car_window_size)

        if(parkPos != null):
            return parkPos
```

Source Code 4.1: Pseudocode: Determine Parking Position Candidate

4.2.1. Noise Removal

Due to sensor noise or poor positioning signals, spatial trajectories can be inaccurate. This Bachelor thesis uses a mean filter to clean the user's trajectory from this noise. The algorithm (Source Code 4.2) iterates through the trajectory of the user and calculates the mean coordinates (Source Code 4.3) of every 5 consecutive points in the trajectory. The resulting trajectory is used for further processing. [1]

```
def removeNoise(trajectory):
    cleaned <- Array()
    if len(trajectory) < 5:
        throw error

    for i in (2 ..< trajectory-2):
        c <- compMeanCoord(trajectory[ i-2 ... i+2 ])
        c.t <- trajectory[i]
        cleaned.append(c)

    return cleaned
```

Source Code 4.2: Pseudocode: Noise Removal [1]

```
def compMeanCoord(coordinates):
    lat <- 0; lon <- 0

    for p in coordinates:
        lat <- lon + p.lat
        lon <- lon + p.lon

    lat <- lat / len(coordinates)
    lon <- lon / len(coordinates)

    return (lat, lon)
```

Source Code 4.3: Pseudocode: Compute Mean Coordinate [1]

4.2.2. Stay Point Detection

The stay point detection algorithm iterates through the trajectory and detects if the user stayed within a given threshold of distance longer than a given threshold of time. The time and distance thresholds are necessary, as otherwise the algorithm could detect a cluster of points within an area where the user frequently travels through, such as a major interception, but which has no further semantic meaning. The algorithm is mostly adapted from [2], but differs slightly. In the original algorithm, line 26 is right before the break command in line 22. This could lead to an infinity loop in which the variable i is never assigned a value greater than pointNum. [2]

```
1  def detectStayPoints(trajectory, distTresh, timeTresh):
2      i <- 0; pointNum <- len(trajectory)
3      stayPoints <- Array()

5      while i < pointNum:
6          j <- i + 1
7          while j < pointNum:
8              p_i <- trajectory[ i ]
9              p_j <- trajectory[ j ]
10             dist <- Distance(p_i, p_j)

12             if dist > distTresh:
13                 time <- p_j.t - p_i.t

15                 if time > timeTresh:
16                     sP <- compMeanCoord(trajectory[ i ... j ])
17                     sP.arrival <- p_i.t
18                     sP.leave <- p_j.t

20                     stayPoints.append(sP)

22                 break;

24             j <- j+1

26         i = j

28     return stayPoints
```

Source Code 4.4: Pseudocode: Stay Point Detection [2]

4.2.3. Trips

The classification of the transportation mode of a trajectory is only relevant for the segments where the user is significantly moving. Thus, only the trajectory segments

between stay points are of interest. The segments at stay points can be ignored.

The algorithm (Source Code 4.5) segments the user's spatial trajectory into individual trips based on its stay points. It iterates through all stay points and selects the segments of the trajectory which are between the current and the following stay point. A segment is between two stay points if its timestamp is greater than or equal to the time of leaving the start stay point and if the timestamp is smaller than or equal to the first timestamp after the user reaches the destination stay point.

```
def cutIntoTrips(trajectory, stayPoints):
    trips <- Array()

    for i in (0 ..< len(stayPoints)-1):
        s_start <- stayPoints[ i ]
        s_stop <- stayPoints[ i+1 ]

        index_start = trajectory.firstIndexWhere( t >= s_start.leave)
        index_stop = trajectory.firstIndexWhere( t >= s_stop.arrival)

        trips.append(trajectory[index_start .. index_stop])

    return trips
```

Source Code 4.5: Pseudocode: Determine Trips in a Trajectory

4.2.4. Determining Parking Position Candidate

To determine a parking position candidate, a heuristic is used: Wherever a user travelled by car last, the car is parked. As a trip is the movement from one stay point to the next one, the last occasion of a user traveling by car in a trip is a potential parking position candidate. To reduce false positive errors, the transportation mode needs to be classified consecutive as car movement for a defined threshold of occurrences. The algorithm used to determine a parking position candidate (Source Code 4.6) iterates reversed through the classified transportation mode labels of the spatial trajectory of a trip. The start point of the temporally last segment of the windows, which are consecutively labeled car movement, is returned as a parking position candidate.

```
def detParkingPosCandidate(trip, labels, car_window_size):
    counter <-0

    for i in (len(labels) - 1  ... 0):
        if(labels[ i ] == "car"):
            counter <- counter + 1
        else
```

```
            counter = 0

        if(counter = car_window_size):
            return trip[ i + car_window_size ]
```

Source Code 4.6: Pseudocode: Determine Parking Position Candidate

4.3. Transportation Mode Classification of Spatial Trajectories

The third section, *Transportation Mode Classification of Spatial Trajectories*, describes the process applied to train a machine learning model that determines the transportation mode of a spatial trajectory. The first part, *Pre-processing*, describes the steps taken to clean and prepare the data of the GeoLife data set. The second part, *Feature Extraction*, names and defines all features generated based on the GeoLife data set. The third section, *Data Validation*, validates the generated data. In the fourth part, *Model Training*, the used machine learning models are presented and explained.

4.3.1. Pre-processing

The GeoLife data set is created by Microsoft Research Asia in the context of the GeoLife project. It consists of spatial trajectories which are created by 182 users over five years. 69 of the users label their trajectories with the used transportation mode. The labels are saved in separate files. [25] [27] [28] [29]

First, the trajectories of the users that labeled their transportation modes are joined with the reported transportation mode based on their timestamps. All trajectories without labels are discarded. Second, all transportation modes, which are not relevant for the context of this Bachelor thesis, are discarded. Only the trajectories of the transportation mode ”car” and ”walk” are used.

4.3.2. Feature Extraction

To enable the classification of the transportation mode, features are extracted from the pre-processed GeoLife data set. To enable the classification to use the temporal context of a segment, the segments are not classified individually but as moving window sets of size three. As shown in Table 4.1, a trajectory can be divided into moving window sets by combining each segment with their directly following segments into a new set. All features are created based on moving window sets of the same transportation mode. [10]

p_1	p_2	p_3
p_4	p_5	p_6
p_7	p_8	p_9

Windows of a Trajectory

p_1	p_2	p_3
p_2	p_3	p_4
p_3	p_4	p_5

Moving Windows of a Trajectory

Table 4.1.: Visualization of Differences Between Windows and Moving Windows

There are two kinds of features: First, individual features based on one single section, such as velocity, acceleration and bearing change. Second, aggregated features based on a moving window set. These are aggregations of the individual features of the segments, such as the minimum, the maximum, the range, the sum, the average, and the variance. Below, the used features are defined.

4.3.2.1. Individual Features

Velocity The velocity of a trajectory is the movement speed of the tracked object. It is defined as $v(p_n) = Dist(p_n, p_{n+1})/(t(p_{n+1}) - t(p_n))$ with $Dist$ as a function to calculate the distance between two coordinates and t being the function that returns the timestamp of a point of a trajectory. [9]

Acceleration The acceleration of a trajectory is the increase or decrease of velocity in the given time. It is defined as $a(p_n) = (v(p_{n+1}) - v(p_n))/(t(p_{n+1}) - t(p_n))$. [9]

Bearing Change Bearing is a measure for the angle between the direction a trajectory is heading and a reference, such as the magnetic or true north. The bearing change is the absolute change of the bearing between two consecutive coordinates. It is defined as $brCh(p_n) = 180 - ||brgn(p_n) - brng(p_{n+1})| - 180|$ with $brng$ as the function that returns the bearing of the segment. [30] Bearing is defined as

$$\begin{aligned} y =& \sin(lon(p_{n+1}) - lon(p_n)) \cdot \cos(lat(p_{n+1}) \\ x =& \cos(lat(p_n)) \cdot \sin(lat(p_{n+1})) - \sin(lat(p_n)) \\ & \cdot \cos(lat(p_{n+1})) \cdot \cos(lon(p_{n+1}) - lon(p_n)) \\ brng(p_n) =& arctan(y, x) \end{aligned}$$

4.3.2.2. Aggregated Features

The aggregation functions are applied to all three individual features. Thus, the aggregation features are described for the generic set X.

Maximum The maximum of a set is the greatest value in the set. It is defined as $\max(X) = \max(x \in X)$.

Minimum The minimum of a set is the smallest value in the set. It is defined as $\min(X) = \min(x \in X)$.

Range The range of a set is the difference between its minimum and its maximum. It is defined as $range(X) = max(X) - min(X)$.

Sum The sum of a set is the sum of its elements. It is defined as $sum(X) = \sum_{x \in X}(x)$.

Mean The mean value of a set is the sum of all its elements divided by the number of elements of the set. It is defined as $mean(x) = sum(X)/|X|$.

Variance The variance of a set it the squared deviation of the set's mean. It is defined as $var(X) = \sum_{x \in X}(x - mean(X))^2/|X|$.

All aggregated features are generated for each kind of individual feature. Thus, for each moving window with three segments, nine individual features and 18 aggregated features are defined.

4.3.3. Data Validation

To ensure that no invalid data is used for the classification of the machine learning model, the data is cleaned using thresholds for speed and acceleration based on the transportation mode of the given moving window set. The thresholds are adapted from [30] and can be seen in Table 4.2.

Transportation Mode	Maximum Velocity (m/s)	Maximum Acceleration (m/s^2)
Walk	7	3
Bike	12	3
Car	50	10
Train	34	3

Table 4.2.: The Used Thresholds for Velocity and Acceleration [30]

All trips with segments that violate the defined thresholds are discarded.

4.3.4. Model Training

In the fourth part, *Model Training*, the used machine learning models are presented and explained. First, *Multilayer Perceptron* is presented. Second, *XGBoost* is presented.

4.3.4.1. Multilayer Perceptron

Multilayer Perceptron is a class of feedforward neural networks. They consist of at least three layers: the input layer, the hidden layers and the output layer. It can be depicted as a diagram, as shown in Figure 4.10. This figure illustrates the input layer X of size P, one hidden layer Y of size M and the output layer Y of size K. Each node of a layer is connected to each of the nodes of the following layer. For the input data, an activation function is used. This function is often sigmoid $\sigma(v) = 1/(1 + e^{-v})$. [32]

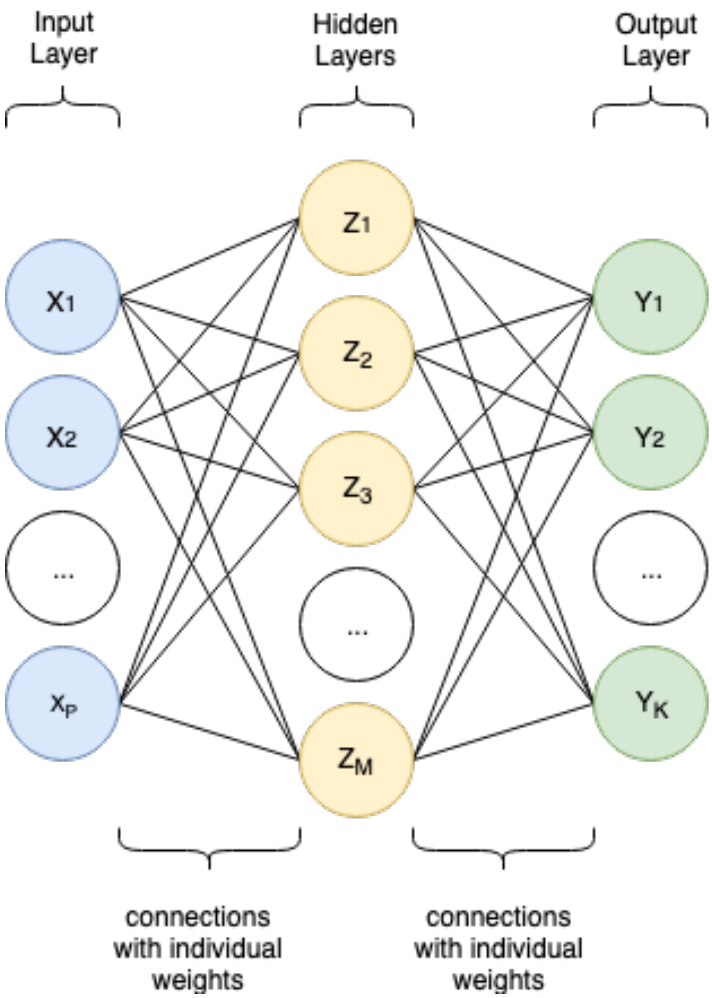

Figure 4.10.: Basic Multilayer Perceptron Schema

The neural network has unknown parameters, called weights. For each connection between two nodes exists one weight. The set of all weights is denoted with Θ. To generate these weights, the model is trained with a labelled training set. In each iteration, the error rate of the weights $R(\Theta)$ is being minimized using gradient decent, in this case called back-propagation. For classification, either the squared error or the deviance is used for the error function. [32]

$$R(\Theta) = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \log f_k(x_i) \tag{4.1}$$

For the classification of the transportation mode, 27 features are generated. Thus, the Multilayer Perceptron classifier has 27 input nodes. The trained model can classify the two transportation modes walk and car. Thus, the classifier has two output layer nodes, one for each class to predict.

4.3.4.2. XGBoost

XGBoost is a scalable tree boosting system. Since its release, it is successfully used in a number of machine learning competitions. Its success is mainly caused by its scalability. The scalability is due to a number of ”important systems and algorithmic optimizations”. Namely, these include a novel tree learning algorithm that can handle sparse data and also the introduction of approximate tree learning in its system. Due to this changes, the XGBoost ”runs more than ten times faster than existing popular solutions on a single machine”. [3]

The basic algorithm of XGBoost is similar to other gradient boosting tree ensemble systems (Source Code 4.7). The system iteratively tries to find the best decision tree for the given data, adds the created decision tree to a set of decision trees and increases the importance of the misclassified examples. In the following iteration, the system tries to find again the best decision tree, but due to the increased importance of the previously misclassified examples, these examples are more strongly considered. The algorithm stops if the maximum of allowed trees is generated. To classify an entry with the generated model, the entry is classified by each of the generated trees and their results are combined. [3]

```
def xgboostClassifyer(data, labels, maxTrees):
    i <- 0
    trees <- Array()
    while(i < maxTrees):
        decTree <- findBestDecisionTree(data, labels)
        trees.apped(decTree)
        increaseImportanceOfMissclassifiedData(decTree, data, label)
        i <- i + 1

    return trees
```

Source Code 4.7: Pseudocode: Basic XGBoost

To generate the individual decision trees, several splits need to be chosen. The possible splits can be evaluated by a loss function. In XGBoost, this loss function is based

on the difference between the prediction and target values of a split, as well as on the complexity of the model. The exact greedy algorithm is powerful in finding the next optimal split because it iterates over all possible next splits. The downside of the algorithm is that its performance suffers if the given data does not fits completely into the memory. To enable an efficient computation in this case, XGBoost implements an approximation algorithm for the split finding. [3]

The approximate algorithm for split finding (Source Code: 4.8) proposes splits for a percentile of each of the m features. The continuous features are sorted into buckets splits based on the generated candidate splits. The best solution among the proposed ones is aggregated. To calculate the score for each split, the algorithm uses the loss function of XGBoost, which is represented by the first and second order gradient statistics $g_j \in G$ and $h_j \in H$.

```
def approxSplitFinding():

    for k = 1 to m:
        Propose Splits S_k = {s_k1, S_k2, ... s_kl} by
    percentiles on feature k

    for k = 1 to m:
        G_kv <- all g_j, with s_kv >= x_jk > s_k,v-1
        H_kv <- all h_j, with s_kv >= x_jk > s_k,v-1

    return split with best score based on G and H
```

Source Code 4.8: Pseudocode: XGBoost - Approximate Algorithm for Split Finding [3]

5. Implementation

The fifth chapter, *Implementation*, describes the actual implementation of the training of the machine learning model used for classifying the transportation of the user and of the application to determine the parking position of the user's car. In the first section, *Architecture*, the architecture of the developed system is presented and explained. The second section, *Technologies*, outlines the technologies used in the implementation of the system. In the third section, *Details of Implementation*, several noteworthy code excerpts of the application, the implementation of the machine learning workflow and the server component are presented.

5.1. Architecture

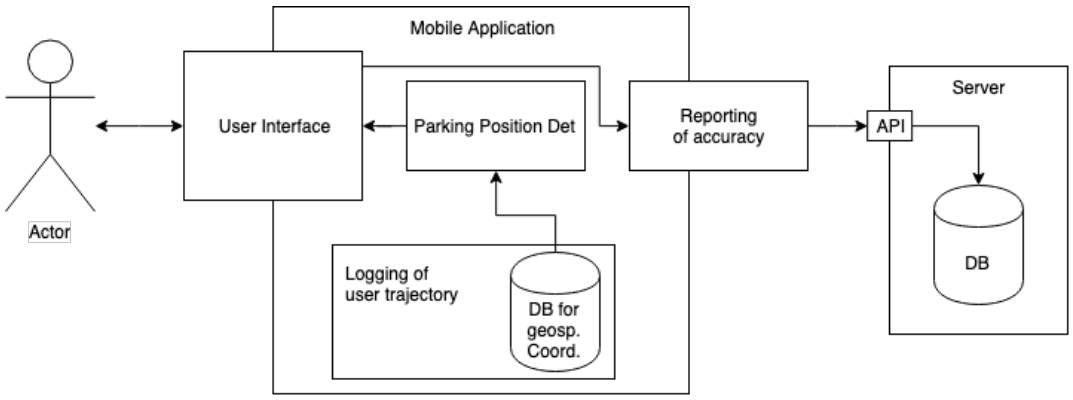

Figure 5.1.: Architecture of Developed System

The system is centered around the developed mobile application. Figure 5.1 visualises the architecture. The arrows represent the data flow. The mobile application consists of four components: the user interface, the logging of the user's trajectory, the determination of the parking position of the user's car and the reporting of the accuracy. The user interface presents all necessary data to the user and the user can perform actions through it. The component that logs the user's trajectory saves the coordinates together with the current timestamp of the coordinates and other meta information in a database on the phone. The component that determines the parking position of the user's car reads the logged trajectory of the user and performs the determination algorithm completely on device. The user can proactively report the accuracy of the determined parking position using the corresponding component. This component sends the accuracy and other meta data, but no sensitive data such as any actual location, to the server component. The data is securely transmitted via the HTTPS protocol. The server component consists of an API which accepts the data sent by the mobile application and stores it in a database.

5.2. Technologies

Swift Swift[1] is a general-purpose language developed by Apple Inc and being open-source since 2015. It is mainly developed to be used on Apple's platforms iOS, macOS, iPadOS, watchOS and tvOS, but can be used on other platforms as well. Its design focuses on safety, performance and expressiveness. It implements these by the avoidance of null values, direct compilation for each platform and a strict syntax. [33]

Alamofire Alamofire[2] is an HTTP networking library. It is written in Swift. In this Bachelor thesis it is used to report the accuracy of the determined parking position to the server component. [34]

Realm Database Realm Database[3] is an open source database management system with a focus on mobile platforms. Its core is written in C++ which enables a high performance. It can perform more than twice as many queries per second than the native solution of iOS called Core Data[4]. Realm Database is used in the application to store the trajectory of the user on device. [35]

MapKit MapKit[5] is a framework to embed maps in an iPhone application. It is developed by Apple. In this Bachelor thesis, the framework is used to present the user's location and the determined parking position on a map. [36]

Jupyter Notebook Jupyter Notebook[6] is an open-source web application that provides an environment to write code in a live context for a variety of languages and to present the code and its computational results. Jupyter Notebook is used in this Bachelor thesis to implement the machine learning training [37]

Node.js Node.js[7] is an asynchronous event-driven JavaScript runtime. It is used to build network applications. In this Bachelor thesis it is used to build the server component of the accuracy reporting. [38]

[1] Swift: https://swift.org/about/, (online: last accessed October 19, 2019)
[2] Alamofire: https://github.com/Alamofire/Alamofire, (online: last accessed October 19, 2019)
[3] Realm Database: https://realm.io/products/realm-database/, (online: last accessed October 19, 2019)
[4] Core Data: https://developer.apple.com/documentation/coredata, (online: last accessed October 19, 2019)
[5] MapKit: https://developer.apple.com/documentation/mapkit, (online: last accessed October 19, 2019)
[6] Jupyter Notebook: https://jupyter.org/, (online: last accessed October 19, 2019)
[7] Node.js: https://nodejs.org/en/about/, (online: last accessed October 19, 2019)

Scikit-learn Scikit-learn[8] is a Python library which implements a variety of machine learning algorithms. This Bachelor thesis uses Scikit-learns implementation of XGBoost and Multylayer Perceptron Neural Networks to classify the transportation modes of a user's spatial trajectory. [39]

5.3. Details of Implementation

The application is implemented for iPhones running the operating system iOS 13, the machine learning model training is implemented using Jupyter Notebooks and the server component is implemented using Node.js. All source code can be found in their repositories[9] [10] [11].

Figure 5.2 shows the three main views of the application. The left view presents the view the user sees when they start the application. It shows the current position of the user on a map and a button to initiate the determination of the parking position. The middle view pictures the status of the application if a parking position is determined. It shows the location of the car and the location of the user on a map. Also, it presents more detailed information about the location of the car, such as the address where it is parked and the relative distance and altitude between the user's location and the determined parking position. If data regarding the floor of the car is available, it is shown as well. The user can trigger three actions via buttons on this screen. First, they can start a navigation towards the position of the user's car in the standard navigation software Apple Maps. Second, they can go back to the previous screen, "Map View - Loading", via the reverse button. Third, they can report the accuracy of the application. The right view shows the interface of the application after the user initiates the reporting of the accuracy. An explaining text and the shared information are shown in the view. The user can report the presented data using the button at the bottom.

[8] Scikit-learn: https://scikit-learn.org/stable/, (online: last accessed October 19, 2019)

[9] Repository of implemented machine learning notebooks: https://github.com/greenBene/ba-ml-model, (online: last accessed October 19, 2019)

[10] Repository of implemented application: https://github.com/greenBene/ba-parking-position-determination/, (online: last accessed October 19, 2019)

[11] Repository of implemented server: https://github.com/greenBene/ba-server, (online: last accessed October 19, 2019)

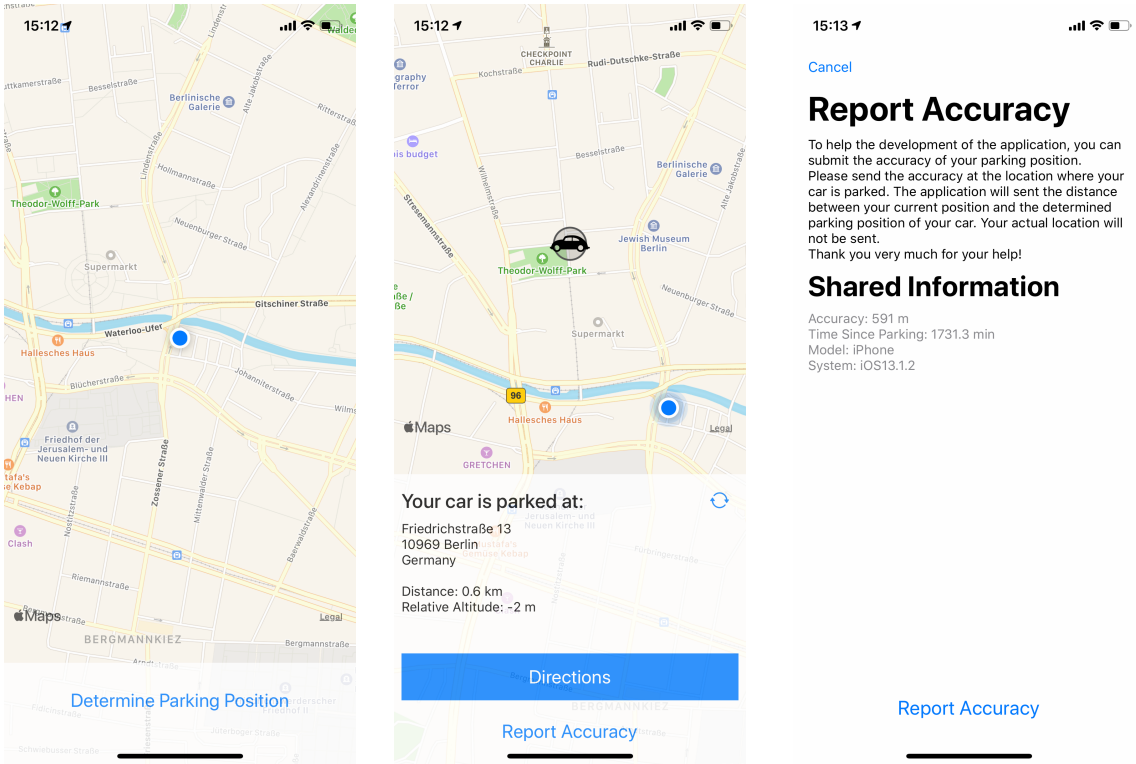

Figure 5.2.: The Three Main Views of the Implemented Application. MapView - Loading (left), MapView - Parking Position Determined (middle), and Report Accuracy (right).

Training of XGBoost Classifier Source Code 5.1 shows the implementation of the training of the XGBoost classifier. First, a XGBoost classifier object is created and possible parameters are defined in a dictionary. These parameters are chosen because they show the best results using three-fold cross-validation. The model is then trained for all possible combinations of the parameter space and the best model is chosen.

```
xgb = XGBClassifier()

parameter_space = {
    'objective': ['binary:logistic'],
    'learning_rate': [0.2],
    'max_depth': [10],
    'min_child_weight': [8],
    'subsample': [0.4],
    'n_estimators': [30]
}

from sklearn.model_selection import GridSearchCV

clf = GridSearchCV(xgb, parameter_space, n_jobs=-1, cv=3,
    verbose=10)
```

```
clf.fit(X_train, y_train)

print('Best␣parameters␣found:\n', clf.best_params_)
```

Source Code 5.1: Train XGBoost Classifier

Determine Parking Position Algorithm Implementation Source Code 5.2 shows the implementation of the algorithm to determine the parking position of the user's car. The whole trajectory of the user is loaded and the noise of the sensors is removed. The cleaned trajectory is cut into trips based on the stay points of the user. These trips are then labeled with their transportation modes and based on this, parking position candidates are computed. The most recent candidate is returned as the parking position.

```
func determineParkingPosition() throws -> (CLLocation, [StayPoint]){
        // Load trajectory
        var trajectory = loadTrajectory()

        // Remove noise
        trajectory = removeNoise(coords: trajectory)

        let stayPoints = getStayPoints(locations: trajectory, distThresh: DISTANCE_THRESHOLD, timeThresh: TIME_THRESHOLD)

        // Cut it into trips based on stay points
        let trips = cutIntoTrips(stayPoints: stayPoints, trajectory: trajectory)

        let classifier = trans_mode()

        for trip in trips.reversed() {
            let features = try featureExtraction(locations: trip)

            let output = try classifier.predictions(inputs: features)

            do {
                guard let loc = try determineParkingPosCandidate(trajectory: trip, labels: output) else {
                    continue
                }
                return (loc, stayPoints)

            } catch {

            }
```

```
29          }
30          throw ClassificationError.noParkingLocationDetermined
31      }
```

Source Code 5.2: Implementation of Algorithm to Determine Parking Position of the User's Car

Determine Stay Points - Implementation Source Code 5.3 shows the implementation of the stay point detection. This algorithm is adapted in most parts from [2] but it is adapted in two places. First, as mentioned in the fourth chapter, *Design*, the original algorithm has a potential infinity loop. This is fixed by moving the line i = j from the original line 25 to line 36. Also, when the algorithm finishes, the last potential start of a stay point is added to the list of stay points. This is implemented so that the most recent trip can be used for determining the parking position of a car.

```
1  private func getStayPoints(locations: [CLLocation], distThresh:
       Double, timeThresh:Double) -> [StayPoint] {
2          var i = 0; let pointNum = locations.count;
3          var stayPoints: [StayPoint] = Array()

5          while i < pointNum {
6              var j = i + 1
7              while j < pointNum {
8                  let p_i = locations[i]
9                  let p_j = locations[j]
10                 let dist = p_i.distance(from: p_j)

12                 if dist > distThresh {
13                     let timeDelta = p_j.timestamp.
       timeIntervalSince(p_i.timestamp)
14                     if timeDelta > timeThresh {
15                         let meanLoc = computeMeanCoordinate(
       coords: Array(locations[i...j]))

17                         stayPoints.append(
18                             StayPoint(lat: meanLoc.latitude, lon
       : meanLoc.longitude,
19                                                       arrivalTime:
        p_i.timestamp, leaveTime: p_j.timestamp,
20                                                       distThresh:
       distThresh, timeThresh: timeThresh)
21                         )
22                     }
23                     break
24                 }
25                 j = j+1
26             }
```

```
28              if(j >= pointNum){
29                  stayPoints.append(
30                      StayPoint(lat: locations[ i ].coordinate.
      latitude, lon: locations[ i ].coordinate.longitude,
31                                                  arrivalTime:
      locations[ i ].timestamp, leaveTime: locations[ i ].timestamp
      ,
32                                                  distThresh: 0,
      timeThresh: 0)
33                  )
34              }

36              i = j // Algorithm in source lacks this line.
      Otherwise potential infinity loop.
37          }

39          let firstLocation = locations[0]
40          stayPoints = [StayPoint(lat:firstLocation.coordinate.
      latitude , lon: firstLocation.coordinate.longitude,
41                                  arrivalTime: firstLocation.
      timestamp, leaveTime: firstLocation.timestamp, distThresh: 0,
       timeThresh: 0)] +
42                      stayPoints

44          return stayPoints
45      }
```

Source Code 5.3: Determine Stay Points - Implementation

Server: Handling of Reported Accuracy Source Code 5.4 shows the function of the server that handles incoming reports of accuracy. First, it parses the incoming message and retrieves the transmitted values. Second, it ensures that the necessary values are set. Third, if the data is valid, it inserts the values into a table of a MySQL database. After the successful insertion or after any error, the server sends an appropriate HTTP code to the client.

```
1  app.post("/post", function(req, res){
2    var body = req.body;

4    var model = body.model;
5    var os = body.os;
6    var timeSinceParking = body.timeSinceParking;
7    var accuracy = body.accuracy;
8    var test = body.test;

10   console.log("Received␣message");
```

```
12    if(model === undefined || os === undefined
13      || timeSinceParking === undefined
14      || accuracy === undefined){

16      res.sendStatus(400);
17      return;
18    }

20    var sql = "INSERT␣INTO␣accuracy␣VALUES␣(0,␣current_timestamp()
       ,␣'" + model +  "',␣'" + os + "',␣" + timeSinceParking +  ",␣
       " + accuracy +  ");"

22    con.query(sql, function (err, res) {
23      if (err){
24        console.log(err);
25        res.sendStatus(500);
26      }

28      });

30    res.sendStatus(200)
31  });
```

Source Code 5.4: Server: Handling of Reported Accuracy

6. Evaluation

The sixth chapter, *Evaluation*, evaluates the results of this Bachelor thesis. In the first section, *Evaluation of Machine Learning Models*, the performance of the trained machine learning models is compared. In the second section, *Accuracy of Determined Parking Position*, the accuracy of the determined parking position is evaluated. In the third section, *Fulfillment of Requirements*, the fulfillment of the defined requirements is discussed and evaluated.

6.1. Evaluation of Machine Learning Models

To find a machine learning model that classifies the transportation modes of a user's spatial trajectory, two models are trained. The parameters of both models are tuned by using the three-fold Cross-validation. The results of the model training are presented using the precision, the recall, the f1-score and the support.

The final parameters for the XGBoost Classifier are as follows: objective = binary:logistic, learning_rate = 0.2, max_depth= 10, min_child_weight = 8, subsample = 0.4, and n_estimators = 30. With these parameters the trained model reaches an accuracy of 90.486%. The precision, recall and f1-score differ between the two classes as seen in Table 6.1.

class	precision	recall	f1-score	Number of Instances
car	0.95	0.78	0.86	121972
walk	0.89	0.97	0.93	208984

Table 6.1.: Evaluation of XGBoost Model

The final parameters for the Multilayer Perceptron Classifier are as follows: hidden_layer_sizes = (20,20,20,20), activation= relu, solver = adam, alpha = 0.0025, learning_rate = adaptive, and max_iter = 200. The trained model reaches an accuracy of 89.837%. Again, the precision, recall and f1-score differ between the two classes as seen in Table 6.2.

class	precision	recall	f1-score	Number of Instances
car	0.96	0.76	0.85	121972
walk	0.87	0.98	0.92	208984

Table 6.2.: Evaluation of Multilayer Perceptron Model

As the XGBoost classifier shows the better accuracy, this classifier is chosen over the Multilayer Perceptron classifier for determining the transportation mode of a spatial

trajectory. A noteworthy aspect is the similarity between the results of both classifier. Not only does the total accuracy differ by less than 1%, but also the precision, recall and f1-score of the individual classes each differ at a maximum of 2%.

6.2. Accuracy of Determined Parking Position

To evaluate the accuracy of the determined parking position, users are asked to report the accuracy of the determined position while using the application in real world scenarios. The accuracy is defined as the distance between the determined parking position and the position at which the car is actually parked. To support the user in this task, the application lets the user report the accuracy via the app. For this, the user is asked to initiate the reporting of the accuracy when they arrive at their parked car. The accuracy is then automatically determined by calculating the distance between the user's current location and the location the application determined as parking location.

#	Seconds Since Parking	Accuracy in Meters
1	88188	13.72679121
2	2378	47.93333171
3	501	43.77836701
4	313	22.8430178
5	296	36.07612286
6	6687	31.54179313
7	1384	187.710812
8	3056	112.8413155
9	413	117.7779001
10	15384	177.9870509
11	963	208.0198427
12	6455	232.2154528
13	231	232.6976242

Table 6.3.: Evaluation of Parking Position Determination Accuracy

For the evaluation, 13 data points are collected by four people. The results can be seen in Table 6.3. The mean accuracy is 112.7 meters. The standard deviation is 85.12. The best measured accuracy is 13.7 meters, the worst measured accuracy is 232.7 meters. 46.2% of the data sets have an accuracy of 50 meters or better.

6.3. Fulfillment of Requirements

[RF.01] The application determines the parking position of the user's car using spatial trajectory analysis The requirement is fulfilled. The application logs the user's trajectory, classifies the transportation mode of each moving window segment within each trip of the trajectory and heuristically determines the parking position of the car based on the classified transportation mode

[RF.02] The application presents the determined parking position of the user's car on a map This requirement is fulfilled. When the application has determined the parking position of the user's car, the determined location is shown on a map view.

[RF.03] The application presents the current position of the user on a map This requirement is fulfilled. The application shows the location of the user when the application is started and also when the parking position of the user's car is determined.

[RF.04] The application presents detailed information about the determined parking position, namely the address, the distance from the current location and the relative altitude This requirement is fulfilled. When the application has determined the parking position of the user's car, the "MapView - Parking Position Determined" is shown, which presents detailed information of the determined parking position, such as the address, the relative distance and altitude from the user's location to the parked car and, if the information is available, the floor the car is parked on.

[RF.05] The application enables the user to report the accuracy of the determined parking position to the developer This requirement is fulfilled. The application offers the user the option to report the accuracy of the determined parking position. The user can initiate this process by clicking on the button labelled "Report Accuracy" on the "MapView - Parking Position Determined" view.

[RN.01] The application is easy to use This requirement is fulfilled. The design process of the user interface is user-centric. Each of the four user interface iterations is tested with new potential users for obstacles and irritations. This process results in an easy to use user interface.

[RN.02] The application does not send any location data to server components of the implemented system This requirement is fulfilled. The application has only one server connection which is used to report the accuracy of the application. In this report, no location data is sent. Only the relative distance between the user's location and the determined parking position and other meta data is sent. No concrete location can be determined based on this data.

[RN.03] The application does not rely on manual user input for determining the parking position This requirement is fulfilled. The application automatically determines the parking position of the user's car without any manual input of the user. The determination is based on the user's trajectory which is automatically logged by the application. The user does not need to give any manual input to enable the system to successfully determine the parking position of the user's car.

[RN.04] The accuracy of the determined parking position is better than 50 meters This requirement is fulfilled to 46.2%, as 46.2% of the tests show an accuracy of 50 meters or better.

[RN.05] The application is developed to run on iOS 13 This requirement is fulfilled. The application can successfully run on the operating system iOS 13.

[RN.06] The application is functional on big parking lots This requirement is partly fulfilled. The application can determine the parking position of a user's car in 42.6% of the cases accurately and thus can support the user in finding their car in a big parking lot in these cases.

[RN.07] The application is functional on multi-storey parking lots This requirement is fulfilled. If the parking position is determined, the application shows the relative altitude of the parked car and also the floor of the parked car, if the information is available. Therefore, the application can support a user to find their parked car in a multi-storey parking lot.

The application fulfills ten of the twelve requirements fully and two of the requirements partly. Thus, the requirements are reached to 91.6%.

7. Results

The seventh chapter, *Results*, presents the outcome of this Bachelor thesis. The first section, *Summary*, summarises the results of the chapters *Analysis*, *Design*, *Implementation* and *Evaluation*. The second section, *Limitations*, names and explains the limitations of this Bachelor thesis.

7.1. Summary

To better understand the problem of finding a parked car in an unknown environment, four users are interviewed about their experiences with the problem. The interviewees report that they most often have to look for their car in big parking lots and in multi-storey parking lots. When asked about a potential application to help users to find their parked car, the interviewees name most often two features. First, they expect such an application to be accurate. The majority agrees that an accuracy of better than 50 meters would be sufficient. Second, they expect the application to automatically determine the parking position of the car, without any manual user input. The need for manual user input would prevent some of the users from using the application. Based on the results of the interviews, five functional and seven non-functional requirements are defined for an application to support users to find their parked car.

A prototype for the user interface design is developed based on the defined requirements. It is improved iteratively in four iterations using the user-centric Think-Aloud method. Between each iteration, the current prototype is showed to users which are asked to perform five different tasks. The key results of the interviews are aggregated and summarised. Based on the results of the interviews, the next prototype is improved.

The final user interface consists of three main views: ”Map View - Loading”, ”Map View - Parking Position Determined”, and ”Feedback”. The first view, ”Map View - Loading”, is shown when the application is started. It shows a map with the current position of the user clearly marked and a button to initiate the determination of the parking position of the user’s car. The second view, ”Map View - Parking Position Determined”, is shown after the application successfully determined the parking position of the user’s car. It presents the determined position on a map together with the current position of the user. The view also shows detailed information about the determined position, such as the address, the relative altitude and distance from the user’s current location to the determined parking position. The user can initiate from this view a third party navigation with directions to the determined car position and they can initiate the reporting of the accuracy of the determined parking position. The view ”Map View - Loading” supports the user in reporting the determined parking position.

To determine the parking position of the user's car based on the analysis of their spatial trajectory, an algorithm is developed. This algorithm uses the user's spatial trajectory. The trajectory is cleared from noise. Based on the stay points of the trajectory, it is cut into individual trips. These trips are individually evaluated for parking position candidates of the user's car. To determine a potential parking position in a trip, the transportation mode for each moving window segment set is evaluated using a machine learning model. The location where the user last stopped traveling by car is a potential parking position. The most recent of these potential parking positions is used as the determined parking position of the user's car.

The developed algorithm to determine the parking position of a user's car depends on a machine learning model to classify the transportation mode of a user's trajectory. To create this model, data from the GeoLife project by Microsoft Research Asia is used. The data is prepared and cleaned. The features velocity, acceleration and bearing change are created for each individual segment of each trajectory. Aggregated features, such as the minimum, the maximum, the range, the sum, the average, and the variance, are created based on the individual features for each moving window set of segments. Based on the created data set, two machine learning models are trained: a Multilayered Perceptron classifier, which is a class of neural networks, and a XGBoost classifier, which is a greedy tree-boosting algorithm.

Based on the design described above, an application is implemented. The application is developed using the programming language Swift and runs exclusively on Apple iPhones. The application logs the spatial trajectory of the user and stores it locally on the iPhone using the database Realm. The designed algorithm to determine the parking position of the user's car is implemented in the application and is not dependent on any server structure. No private data, such as any location data, is sent to any server element to ensure the privacy of the user. To be able to evaluate the accuracy of the determined parking position, the application enables the user to report the accuracy of the determined parking position to a server. All sent information are shown to the user before they are sent and they do not contain any sensitive data, such as the location data or any information that could be used to identify the user.

The outcome of this Bachelor thesis is evaluated in three ways. First, the performances of the trained machine learning models are compared. The Multilayer Perceptron classifier reaches an accuracy of 89.8%. The XGBoost classifier reaches an accuracy of 90.48%. The parameters for both classifiers are optimised using threefold cross validation. The XGBoost classifier is used in the implementation of the application as it shows the better results. Second, the accuracy of the determined parking position is evaluated. Thirteen tests are performed by four people. The mean accuracy is 112.7 meters with a standard deviation of 85.12 meters. Third, the fulfillment of the defined requirement is evaluated. The implemented application fulfills all five functional requirements and 6 of the eight non-functional requirements. The missing two non-functional requirements are fulfilled partly.

7.2. Limitations

The analysis of the problem is conducted with semi-structured interviews. For these interviews four people are interviewed. For a more representative survey significantly more people need to be interviewed.

To design an user-friendly user interface Think-Aloud interviews are conducted between each of the four iterations. At each iteration only one person is interviewed. To ensure a good usability for a wide range of users, more people need to be interviewed.

The Think-Aloud interviews are performed in an office environment. To improve the results of these interviews even more they can be performed in the context in which the application is used.

For the classification of the transportation mode of the user's spatial trajectory, two machine learning models are trained. To find the best machine learning mode for this task, more models can be trained.

The trained classifiers can only distinguish between the transportation modes walking and car. In real world scenarios users might use other transportation modes, such as bikes, subways and buses, which are not covered by the developed system.

The developed application is implemented for Apple iPhones and cannot easily be adapted for Android devices. The designed approach of the application to determine the parking position of the user's car can be used to implement an Android application.

For the evaluation of the accuracy of the determined parking position, 13 data points are applied. To get a more accurate evaluation of the accuracy, more tests have to be conducted.

8. Conclusion

The eigth chapter, *Conclusion*, discusses the results of this Bachelor thesis and potential future work. In the first section, *Discussion of Results*, the results of this Bachelor thesis are discussed. In the second section, *Future Work*, possible future works related to this Bachelor thesis are presented.

8.1. Discussion of Results

The background interviews conducted in this Bachelor Thesis show that users experience the problem of finding a parked car but the majority of the users is not aware that there are existing solutions for this problem implemented in the most used navigation applications, such as Google Maps and Apple Maps. This shows that the developers of major navigation applications are aware that the problem exists and already started to develop systems to solve it, but may not be confident in their solutions as they do not present them prominently in their applications.

The developed algorithm to determine the parking position of a user's car based on the spatial trajectory of the user does successfully determine the parking position in 46% of the tests. This indicates that the spatial trajectory analysis of the user's trajectory is a viable way to determine the parking position of a user's car. Still, none of the major navigation applications uses this approach. This could be caused by the relative high battery impact of the constant trajectory recording.

The transportation mode classification in this Bachelor thesis is similar to the approaches used in the most influential related works. A major difference is the omission of any post-processing. The developed system still shows a similar accuracy compared to the related work. This shows that the classification of the transportation mode does not need any post-processing.

The developed application successfully implements the designed algorithm to determine the parking position of a user's car based on the spatial trajectory of the user. It performs the classification of the user's trajectory in near real time. This shows how performant modern smartphones are and underlines that more computation can be executed on the user's device itself.

8.2. Future Work

The developed approach focuses on the transportation modes car and walk. Future work can use a similar approach for other transportation modes, such as bike, more suitable for other environments. The developed approach is adaptable for such changes.

This Bachelor thesis constrains itself to the use of the user's spatial trajectory to determine the parking position of their car. Future work can use more information, such as the acceleration data of the user's smartphone, to determine the parking position.

In future work the developed algorithm to determine the parking position of the user's car can be implemented for other smartphones devices, such as Android devices. This allows the evaluation of the algorithm on a more diverse set of hardware.

9. References

[1] **Yu Zheng**. “Trajectory Data Mining: An Overview”. In: *ACM Trans. Intell. Syst. Technol.* 6.3 (May 2015), 29:1–29:41. ISSN: 2157-6904. DOI: 10.1145/2743025. URL: http://doi.acm.org/10.1145/2743025.

[2] **Quannan Li, Yu Zheng, Xing Xie, Yukun Chen, Wenyu Liu, and Wei-Ying Ma**. “Mining user similarity based on location history”. In: *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*. ACM. 2008, p. 34.

[3] **Tianqi Chen and Carlos Guestrin**. “Xgboost: A scalable tree boosting system”. In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM. 2016, pp. 785–794.

[4] **Qi Wu, Chingchun Huang, Shih-yu Wang, Wei-chen Chiu, and Tsuhan Chen**. “Robust parking space detection considering inter-space correlation”. In: *2007 IEEE International Conference on Multimedia and Expo*. IEEE. 2007, pp. 659–662.

[5] **Ph.D. Hossam El-Din I. S. Ahmed**. “Car Parking Problem in Urban Areas, Causes and Solutions”. In: *Ssrn* (2018). DOI: 10.2139/ssrn.3163473.

[6] **Yanfeng Geng and Christos G. Cassandras**. “A new “Smart Parking” System Infrastructure and Implementation”. In: *Procedia - Soc. Behav. Sci.* 54.may 1877 (2012), pp. 1278–1287. ISSN: 18770428. DOI: 10.1016/j.sbspro.2012.09.842.

[7] **Jae-Gil Lee, Jiawei Han, Xiaolei Li, and Hector Gonzalez**. “TraClass: trajectory classification using hierarchical region-based and trajectory-based clustering”. In: *Proceedings of the VLDB Endowment* 1.1 (2008), pp. 1081–1094.

[8] **Yu Zheng and Xie Xing**. “Learning Travel Recommendations From User- Generated GPS Traces”. In: *ACM Trans. Asian Lang. Inf. Process.* 6.3 (2007). ISSN: 15300226. DOI: 10.1145/1290002.1290003. URL: http://portal.acm.org/citation.cfm?doid=1290002.1290003.

[9] **Yu Zheng, Like Liu, Longhao Wang, and Xing Xie**. “Learning transportation mode from raw gps data for geographic applications on the web”. In: (2008), p. 247. DOI: 10.1145/1367497.1367532.

[10] **Adel Bolbol, Tao Cheng, Ioannis Tsapakis, and James Haworth**. “Inferring hybrid transportation modes from sparse GPS data using a moving window SVM classification”. In: *Comput. Environ. Urban Syst.* 36.6 (2012), pp. 526–537. ISSN: 01989715. DOI: 10.1016/j.compenvurbsys.2012.06.001. URL: http://dx.doi.org/10.1016/j.compenvurbsys.2012.06.001.

[11] **United States Department of Defense**. *Global positioning system standard positioning service performance standard*. 4th ed. Sept. 2008.

[12] **Jesse Davis and Mark Goadrich**. “The relationship between Precision-Recall and ROC curves”. In: *Proceedings of the 23rd international conference on Machine learning*. ACM. 2006, pp. 233–240.

[13] **Yutaka Sasaki et al.** “The truth of the F-measure”. In: *Teach Tutor mater* 1.5 (2007), pp. 1–5.

[14] **Chadia Abras, Diane Maloney-Krichmar, and Jenny Preece**. “User-Centered Design”. In: *Bainbridge, W. Encycl. Human-Computer Interact. Thousand Oaks Sage Publ.* 37.4 (2004), pp. 445–456.

[15] **Chauncey Wilson**. *Interview techniques for UX practitioners: A user-centered design method*. Newnes, 2013.

[16] **Elizabeth Goodman, Mike Kuniavsky, and Andrea Moed**. *Observing the user experience: A practitioner’s guide to user research*. Elsevier, 2012.

[17] **Jakob Nielsen**. *Usability engineering*. Elsevier, 1994.

[18] **Jakob Nielsen**. *10 Usability Heuristics for User Interface Design*. [Online; last accessed October 19, 2019]. 1995. URL: https://www.nngroup.com/articles/ten-usability-heuristics/.

[19] **Apple Inc.** *Human Interface Guidelines*. [Online; last accessed October 19, 2019]. 2019. URL: https://developer.apple.com/design/human-interface-guidelines/.

[20] **Monique WM Jaspers, Thiemo Steen, Cor Van Den Bos, and Maud Geenen**. “The think aloud method: a guide to user interface design”. In: *International journal of medical informatics* 73.11-12 (2004), pp. 781–795.

[21] **Apple Inc.** *How to find your parked car with Maps on your iPhone*. [Online; last accessed October 19, 2019]. 2019. URL: https://support.apple.com/en-us/HT207227.

[22] **Google LLC**. *Find & save parking locations*. [Online; last accessed October 19, 2019]. 2019. URL: https://support.google.com/maps/answer/7257797?co=GENIE.Platform%5C%3DiOS%5C&hl=en.

[23] **Apple Inc.** *Framework - Core Motion*. [Online; last accessed October 19, 2019]. 2019. URL: https://developer.apple.com/documentation/coremotion.

[24] **Adrian C. Prelipcean, Gyözö Gidófalvi, and Yusak O. Susilo**. “Transportation mode detection–an in-depth review of applicability and reliability”. In: *Transp. Rev.* 37.4 (2017), pp. 442–464. ISSN: 14645327. DOI: 10.1080/01441647.2016.1246489.

[25] **Yu Zheng, Quannan Li, Yukun Chen, Xing Xie, and Wei-Ying Ma**. “Understanding mobility based on GPS data”. In: *Proceedings of the 10th international conference on Ubiquitous computing*. ACM. 2008, pp. 312–321.

[26] **Zhibin Xiao, Yang Wang, Kun Fu, and Fan Wu**. “Identifying Different Transportation Modes from Trajectory Data Using Tree-Based Ensemble Classifiers”. In: *ISPRS Int. J. Geo-Information* 6.2 (2017), p. 57. DOI: 10.3390/ijgi6020057.

[27] **Yu Zheng, Xing Xie, and Wei-Ying Ma**. “Geolife: A collaborative social networking service among user, location and trajectory.” In: *IEEE Data Eng. Bull.* 33.2 (2010), pp. 32–39.

[28] **Yu Zheng**. *GeoLife GPS Trajectories 1.2.2*. [Online;last accessed October 19, 2019]. Aug. 2012. URL: https://www.microsoft.com/en-us/download/details.aspx?id=52367.

[29] **Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma**. "Mining interesting locations and travel sequences from GPS trajectories". In: *Proceedings of the 18th international conference on World wide web*. ACM. 2009, pp. 791–800.

[30] **Sina Dabiri and Kevin Heaslip**. "Inferring transportation modes from GPS trajectories using a convolutional neural network". In: *Transp. Res. Part C Emerg. Technol.* 86.August 2017 (2018), pp. 360–371. ISSN: 0968090X. DOI: 10.1016/j.trc.2017.11.021. URL: https://doi.org/10.1016/j.trc.2017.11.021.

[31] **Petter LH Eide**. "Quantification and Traceability of Requirements". In: *NTNU, Norwegian University of Science and Technology* (2005), p. 15.

[32] **Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin**. *The elements of statistical learning: data mining, inference and prediction*. 2nd ed. Springer, 2009.

[33] **Apple Inc.** *Swift*. [Online; last accessed October 19, 2019]. 2019. URL: https://swift.org/about/.

[34] **Alamofire Software Foundation**. *Alamofire*. [Online; last accessed October 19, 2019]. 2019. URL: https://github.com/Alamofire/Alamofire.

[35] **Marin Todorov**. *Realm is an Object-Centric Modern Database for Mobile Apps*. [Online; last accessed October 19, 2019]. 2019. URL: https://academy.realm.io/posts/realm-object-centric-present-day-database-mobile-applications/.

[36] **Apple Inc.** *MapKit*. [Online; last accessed October 19, 2019]. 2019. URL: https://developer.apple.com/documentation/mapkit.

[37] **Project Jupyter**. *Jupyter Notebook*. [Online; last accessed October 19, 2019]. 2019. URL: https://jupyter.org/index.html.

[38] **Node.js Foundation**. *Node.js - About*. [Online; last accessed October 19, 2019]. 2019. URL: https://nodejs.org/en/about/.

[39] **F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay**. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

A. User Interface Iterations

First Iteration

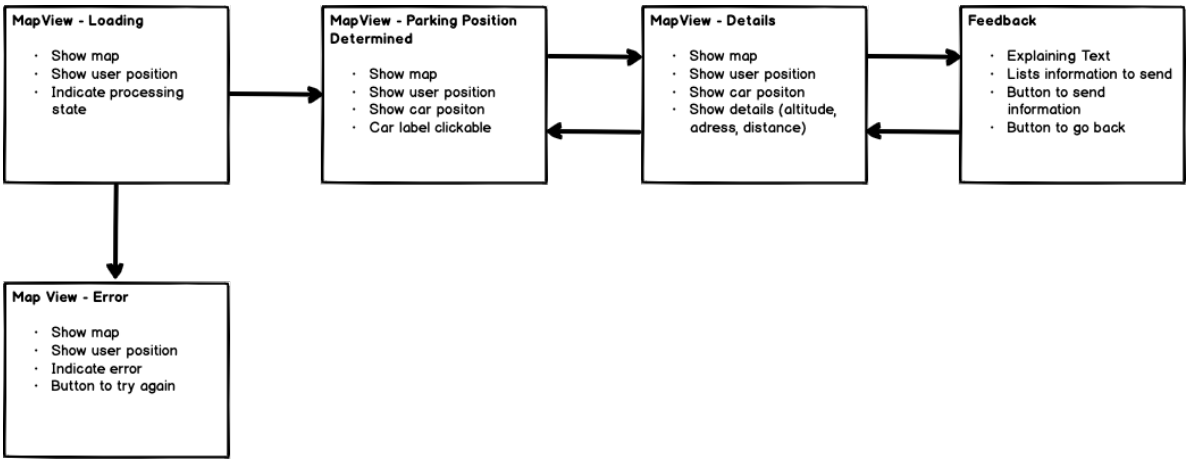

Figure A.1.: Iteration 1: Storyboard

Figure A.2.: Iteration 1: MapView - Loading

Figure A.3.: Iteration 1: MapView - Error

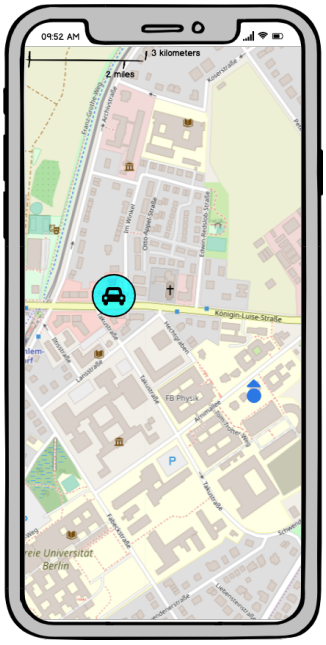

Figure A.4.: Iteration 1: MapView - Parking Position Determined

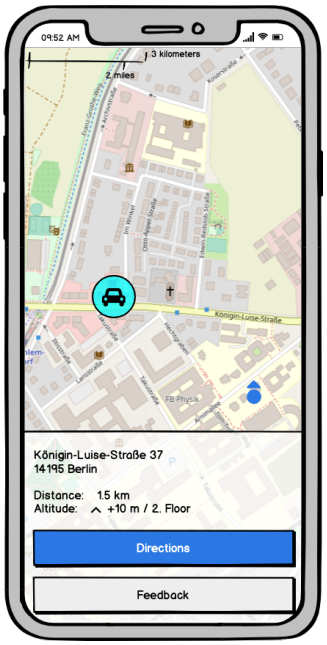

Figure A.5.: Iteration 1: MapView - Details

Figure A.6.: Iteration 1: Feedback

Second Iteration

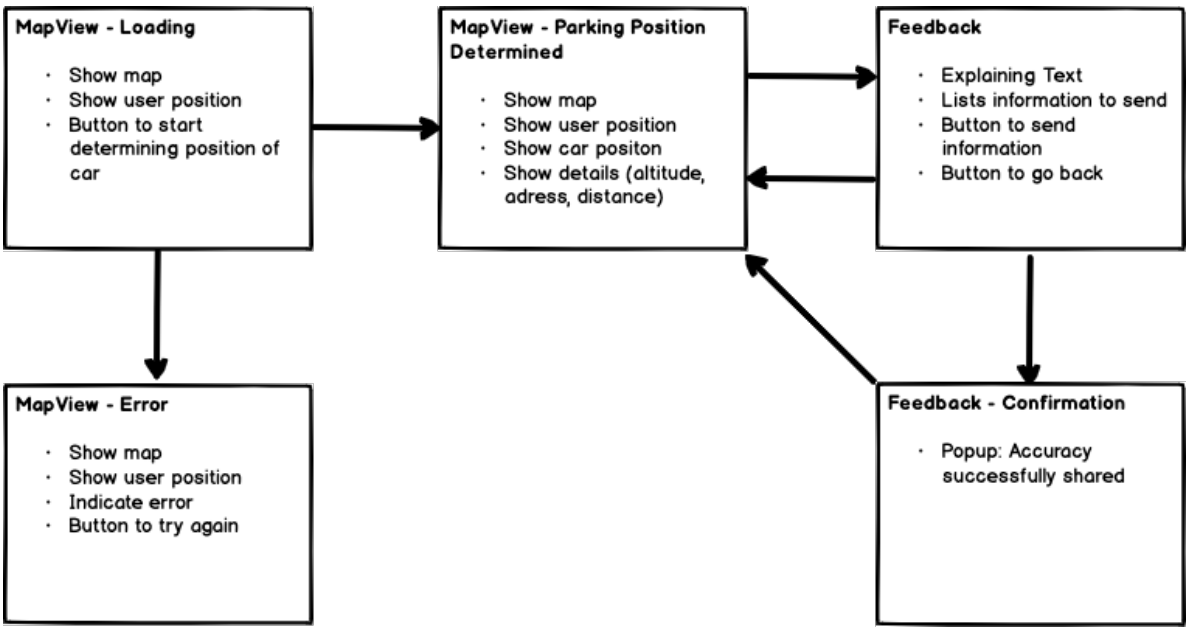

Figure A.7.: Iteration 2: Storyboard

Figure A.8.: Iteration 2: MapView - Loading

Figure A.9.: Iteration 2: MapView - Error

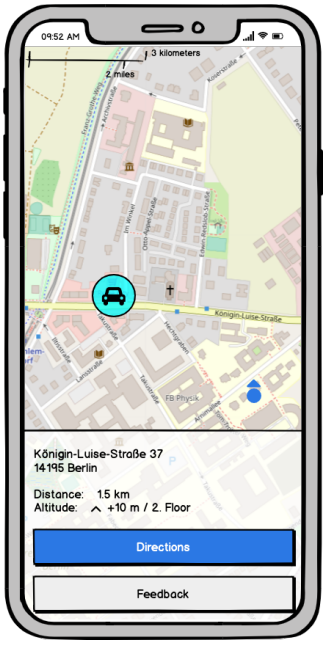

Figure A.10.: Iteration 2: MapView - Parking Position Determined

Figure A.11.: Iteration 2: Feedback

Figure A.12.: Iteration 2: Feedback - Confirmation

Third Iteration

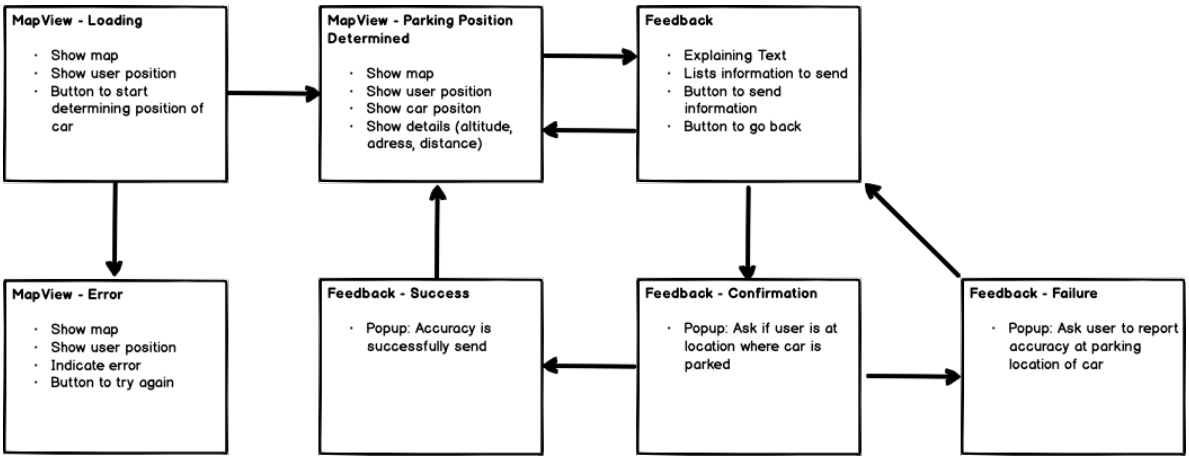

Figure A.13.: Iteration 3: Storyboard

Figure A.14.: Iteration 3: MapView - Loading

Figure A.15.: Iteration 3: MapView - Error

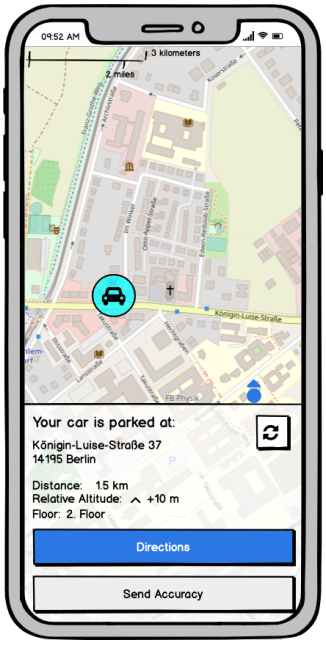

Figure A.16.: Iteration 3: MapView - Parking Position Determined

Figure A.17.: Iteration 3: Feedback

Figure A.18.: Iteration 3: Feedback - Confirmation

Figure A.19.: Iteration 3: Feedback - Failure

Figure A.20.: Iteration 3: Feedback - Success

Fourth Iteration

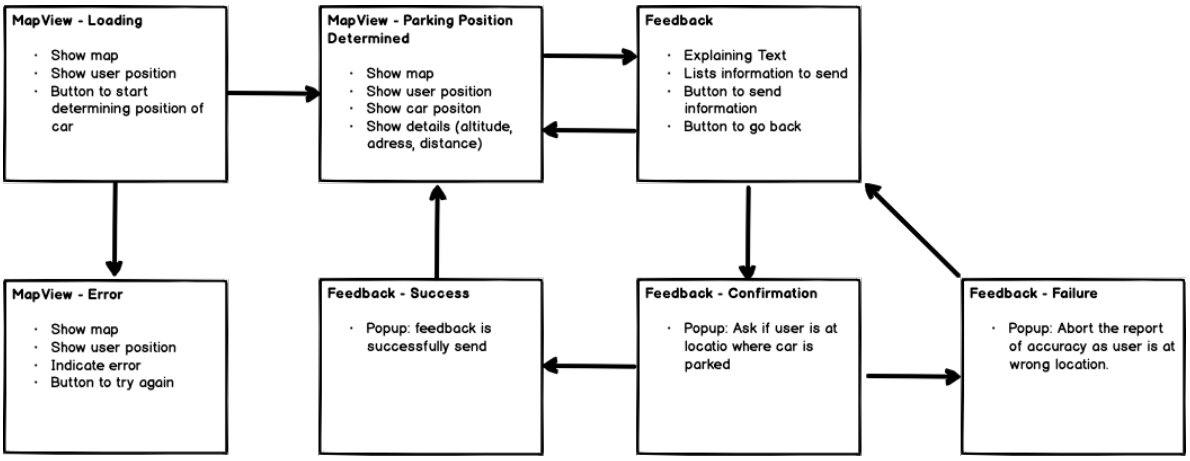

Figure A.21.: Iteration 4: Storyboard

Figure A.22.: Iteration 4: MapView - Loading

Figure A.23.: Iteration 4: MapView - Error

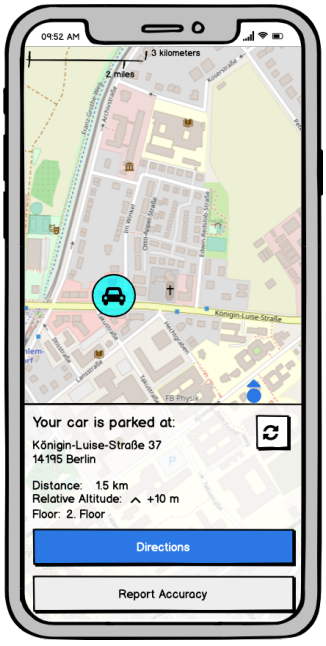

Figure A.24.: Iteration 4: MapView - Parking Position Determined

Figure A.25.: Iteration 4: Feedback

Figure A.26.: Iteration 4: Feedback - Confirmation

Figure A.27.: Iteration 4: Feedback - Failure

Figure A.28.: Iteration 4: Feedback - Success

B. Source Code of Implementation

DVD A DVD is added to this Bachelor thesis. It contains a PDF version of this Bachelor thesis and the source code of the developed iPhone application, the machine learning workflow and the server application.

Repositories The complete source code of the developed iPhone application, the machine learning workflow and the server application can be found online in GitHub repositories:

Repository of iPhone application:
https://github.com/greenBene/ba-parking-position-determination

Repository of machine learning workflow:
https://github.com/greenBene/ba-ml-model

Repository of server application:
https://github.com/greenBene/ba-server