JOHANNES KEPLER
UNIVERSITY LINZ

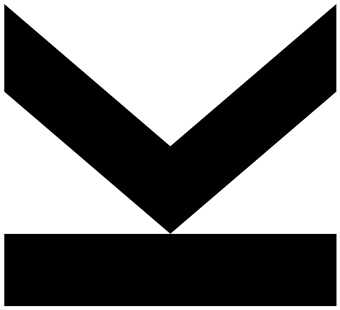
Author
Markus Hiesmair

Submission
Institute of Telecooperation

Thesis Supervisor
Ass. Prof. Dr. Karin Anna Hummel

March 2018

MACHINE LEARNING BASED DRIVE-BY ROAD SIDE PARKING DETECTION USING OPTICAL DISTANCE SENSING



Master's Thesis

to confer the academic degree of

Master of Science

in the Master's Program

Computer Science

JOHANNES KEPLER UNIVERSITY LINZ
Altenberger Str. 69
4040 Linz, Austria
www.jku.at
DVR 0093696

Affidavit

I hereby declare that the following dissertation "Machine Learning based Drive-by Road Side Parking Detection using Optical Distance Sensing" has been written only by the undersigned and without any assistance from third parties.

Furthermore, I confirm that no sources have been used in the preparation of this thesis other than those indicated in the thesis itself.

Linz, on April 13, 2018

Markus Hiesmair

Acknowledgment

Abstract

Abstract ...

Contents

Abbreviations

CNN Convolutional Neural Network

DBSCAN Density-Based Spatial Clustering of Applications with Noise

DL Deep Learning

DWG Design Web Format (file format)

DXF Drawing Interchange Format (file format)

GNNS Global Navigation Satellite System

GPIO General Purpose Input Output

GPS Global Positioning System

I2C Inter Integrated Circuit

kNN k-Nearest Neighbour Classifier

LIDAR Light detection and ranging

NaN Not a Number

ROC Receiver operating characteristic

SMOTE Synthetic Minority Over-sampling Technique

SVM Support Vector Machine

List of Figures

List of Tables

Chapter 1

Introduction

Traffic congestion in urban areas became a bigger problem every year in the past decades. Increasing traffic causes several issues, for example high monetary and environmental costs due to gasoline consumption and by emitting CO_2 to the environment. There are several strategies to reduce urban traffic to mitigate these problems like for example new investments in public transport infrastructure. However, according to a study by Hao et al. [10], car sales will continue to grow in the upcoming decades and therefore also traffic will continue to become worse in big cities. Thus, it is important to find ways to reduce traffic in urban areas.

With more vehicles driving in urban areas, there also comes the need for a sufficient number of parking spaces. Finding parking spaces in urban areas can be really difficult, frustrating and time consuming. There often exists some information about the availability of parking spaces in parking garages, but in most cities the situation of road side parking is unknown. This not only leads to frustrated drivers, who are searching for parking spaces a long time, but again contributes to urban traffic congestion as many cars have to drive around close to their destination while searching for free parking spaces.

There are many studies, which show that the searching for parking spaces adds a lot of traffic. A study by Nawaz et al. [19] shows that about 30% of traffic congestion is created by drivers looking for free parking spaces. Another study [11] concludes that alone in 2007 searching for parking spaces caused costs of about 78 billion US dollars by using 2.9 billion gallons of wasted gasoline and 4.2 billion lost hours only in the United States. Furthermore, cruising causes a lot of greenhouse emissions which is not only bad for the environment and contributes to climate change, but also lowers the quality of living in big cities through the significant amount of air pollution.

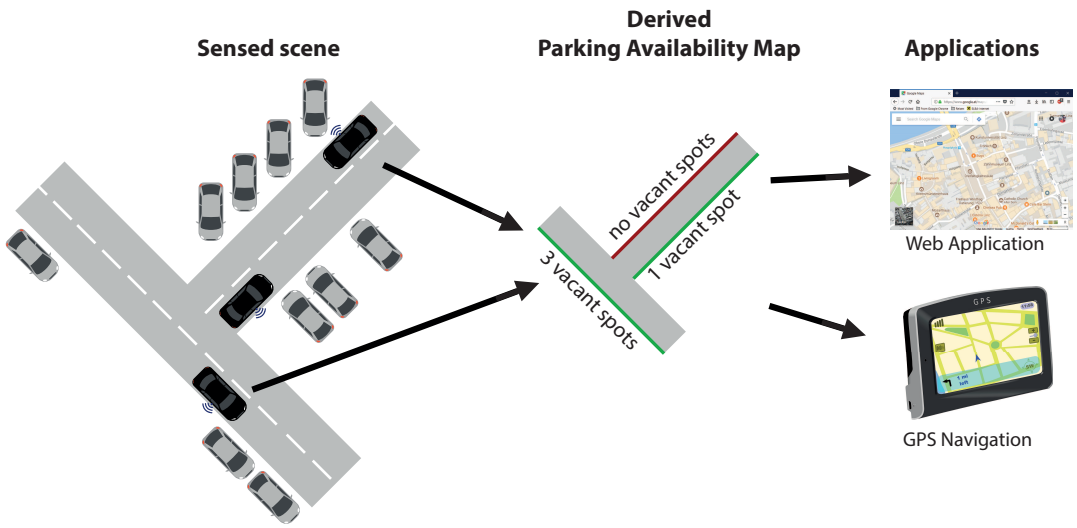


Figure 1.1: Envisioned system of several vehicles driving through the city and sensing its parking availability situation. The resulting parking availability map can be shown via a Web application or can be used by GPS navigation devices.

One of the most important contributors to long search times for parking spaces is not only the lack of vacant parking spaces, but also the lack of information about free parking spaces. Therefore, one way to mitigate many of the above stated problems is to determine the current parking space situation in the city and make it accessible to the public (for instance via a Web application), so that drivers can efficiently navigate to a vacant parking space, or even decide if they want to go by car or use public transportation, depending on the number of parking spaces available close to their destination.

1.1 Envisioned Parking Information System

Detection of road side parking spaces and their states is a challenging task. Of course an obvious approach to the problem would be to put stationary sensors to every parking space in the city, which check, if the corresponding parking space is occupied or vacant. This, however, has the drawback to be very expensive as, for big cities, thousands of sensors would have to be bought, installed, and maintained. Moreover, because the state of parking parking spaces does not often change, the high frequency of sensing with such a system would be rather inefficient.

The envisioned system, which is shown in Figure 1.1, uses a mobile crowd sensing approach to determine the current parking availability situation in urban areas. Several driving vehicles continuously sense their environment using multiple sensors while driving through the streets of the city. These measurements are then used for the analysis of the parking situation, so that parking cars and vacant parking spaces are being detected.

The parking situation analysis can be performed by each sensing vehicle for itself. After obtaining the results, all the derived information may be sent to a central server, which then combines the data of all sensing vehicles and computes a parking availability map. This map of free and occupied parking spaces can then be used for showing users the current parking situation (e.g., through a Web application) so that they can decide if they want to go by car or by public transport. Furthermore, the information of vacant parking spaces could be used effectively by GPS navigation systems which could direct the driver to a vacant parking space close to his end destination and therefore maybe prevent long parking search times.

1.2 Drive-By Parking Space Sensing

The envisioned system is using a promising novel approach, called drive-by park sensing, to sense a city's parking situation. Drive-by park sensing appraoches are using mobile sensors instead of static ones to derive a city's current parking availability situation. Several prototypes, including the ParkNet prototype by Mathur et al. [17], which will be discussed later on, show that mobile crowd sensing has the advantage to be usually more cost effective and can provide sufficient accuracy for the purpose of providing parking space availability maps.

The idea of drive-by approaches is that there are sensing vehicles which are driving through the city and collecting data of their environment through mounted sensors. Using the collected data, parking cars and vacant parking spaces can be detected. There already exists a prototype implementation of such a system. In 2010, Mathur et al. [17] presented their system, called ParkNet, which continuously measures the distance to the nearest obstacle on the side of the road, as well as the location of the sensing vehicle through a GPS sensor. Using these data, they used a distance threshold to detect parking cars and vacant parking spaces. A more detailed description of their work can be found in section 2.1.6.

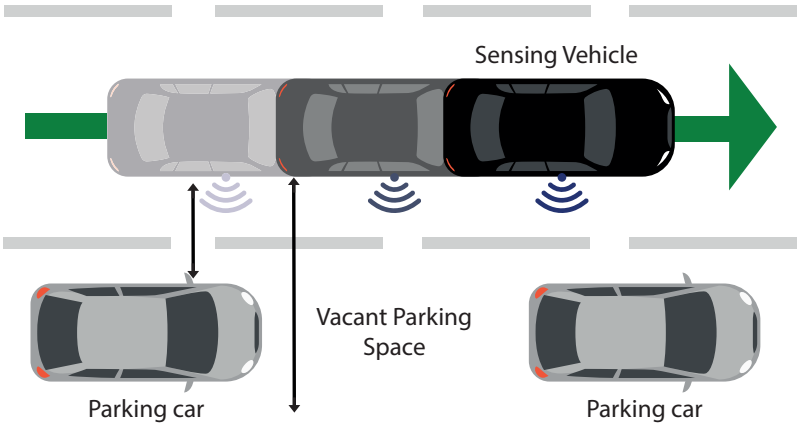


Figure 1.2: The sensing vehicle passes two parked cars and should identify a vacant spot in between using distance and location measurements.

Figure 1.2 shows a standard drive-by scenario of a sensing vehicle that passes two parallel parked cars and a vacant parking space in between them. The distance measurements while passing the parked cars will be much shorter than the measurements taken while passing the vacant parking space. This should allow a basic algorithm to recognize parking cars and vacant parking spaces.

However, the situation shown in Figure 1.2 is an idealistic one. In real life traffic, there will be much more complex situations to face, which are not as easily detectable and which might influence the success of the detection. For instance, the sensing vehicle might not drive in the right-most lane, therefore the measured distances will be much longer. Another possible issue are other driving cars, motorcycles or bicycles which the sensing car overtakes. Many of such distractions have to be filtered out to counteract false detection results.

The high complexity of urban traffic and the many distractions during sensing make it nearly impossible to create a rule set based on the sensor measurements which would be able to detect parking situations at a sufficient accuracy. Therefore, simple thresholding will not work in real life traffic scenarios. Furthermore, such rule sets would have to be created for each city individually because of the different nature of the roads and the parking spaces. For instance, the distance between lanes and parking spaces can vary highly in two different cities. Thus, to be able to detect parking situations, new methods have to be found, that are more flexible to distractions and changes in the environment.

1.3 Research Goal and Methodology

The overall goal of this thesis is to study and increase the accuracy of detecting the parking space situation in urban real world traffic scenarios using drive-by park sensing on multi lane roads. Vacant parking spaces and parking cars are detected and distinguished from other common objects beside the road like trees, houses and other objects. To counteract false detections of parking cars while sensing in real world traffic, several distractions are being identified, detected and filtered out. Common distractions include overtaken cars, bicycles and motorcycles as well as passed cars waiting at traffic lights.

To determine if it is possible to derive parking availability statistics from drive-by sensing in complex road scenarios the following steps of the used research methodology will be taken:

Building a test bed To retrieve the required data, a test bed is being built which is able to access the distance and GPS sensors as well as the camera. It records these data continuously while driving by and saves them for the later analysis. The detailed description of all hardware parts and the data collection process can be found in the sections 3.1 and 3.2.

Acquiring a dataset As soon as the test bed is ready, the sensors are mounted on the prototype car to be able to start recording the dataset. Test drives are being done in some selected streets in Linz, Austria with the focus of variety of the recorded situations. All measured distances, GPS locations and ground truth images are saved to files including the according timestamps to be able to evaluate the results of different approaches later on. Furthermore, using the images taken by the camera, ground truth values are manually labelled in different classes (Parked car, free space, overtaken car, ...). A detailed description about the dataset and the ground truth tagging can be found in the sections 3.3 and 4.1.

Data processing and segmentation As next step the measured sensor values have to be preprocessed and filtered in order for the following algorithms to work. Sensing and overflow errors as well as outliers in the measurements should be identified and removed before further processing. Section **??** is discussing all filter and preprocessing techniques used. After the raw data has been filtered, the sensor data has to be segmented. As parking cars and other objects which should be classified consist of several sensor measurements, the corresponding measurements should be grouped together and merged to segments which can

later be classified. A detailed description of the segmentation process can be found in section **??**.

Classification using machine learning and deep learning Features on the created segments are calculated (for instance length, average distance and variation of the distance measurements) and are being used to train and evaluate several machine learning algorithms which will be compared on their performance. Furthermore, some deep learning models will also be evaluated on the raw sensor data of the segments and will be compared to common machine learning results. The used features are described in section **??**, whereas the tested machine learning and deep learning approaches can be found in the sections **??** and **??**. The results of all experiments are evaluated and compared in section **To-Do:** ref **??**.

Deriving basic parking space maps To support parking detection accuracy a parking space map using the sensor measurements are being derived. Parking space maps are necessary for the overall system to determine if it is legal to park at a certain location, i.e., if any space is a parking space. As for our test use case there is no parking space map with sufficient quality available at city authorities, a basic parking space map is derived using the acquired dataset. The process of deriving the parking space map can be found in section **??**. **To-Do:** reference to parking space map to improve accuracy

Further improvements of the machine learning approaches Several strategies for further improving the accuracy of the machine learning performance are being tested and evaluated. Section **??** describes all of these attempts to improve the detection accuracy.

Chapter 2

Related Work

This chapter will cover work related to park sensing and machine learning. Section 2.1 will discuss different approaches to sensing current parking situations in cities. Furthermore, a comparison of them will be given as well as advantages and disadvantages of the specific approaches. In Section 2.2 the need of parking space maps will be discussed as well as how they can be obtained using several sensing vehicles.

2.1 Approaches to Park Sensing

There already exist numerous approaches for detecting the states of parking spaces as well as the parking situation in a city. In this section parking detection approaches will be categorized in six different categories and in the following subsections several reference papers for all categories will be discussed. The six categories are stationary park sensing with dedicated sensors (Section 2.1.1), park sensing using stationary cameras (Section 2.1.2), counting in- and outgoing vehicles (Section 2.1.3), event detection based park sensing (Section 2.1.4), and drive-by park sensing using either a camera (Section 2.1.5) or a distance sensor (2.1.6). After all detection strategies have been discussed an overall comparison is presented in Section 2.1.8.

2.1.1 Stationary Park Sensing with Dedicated Sensors

The most obvious and technically simple solution to park sensing is to use stationary sensors to determine the state of parking spaces. Usually one sensor per parking space is used which determines its state (occupied or vacant) and sends this state to a central server. Several reference projects already exist implementing this technique[21, 20].

In this section as a representation of all the similar systems, the SFpark project will be examined.

In San Francisco a first prototype of the SFpark project [21] has been employed from 2008 to 2011. Wireless stationary sensors have been placed at about 8.200 road side parking spaces in specific down town areas in San Francisco with high amounts of traffic. These sensors are able to detect the state of one parking space in real time and send this information to a central server. Furthermore, parking lots also count the in- and outgoing vehicles and share this information, so that the parking situation in the areas with park sensing can be derived. The gained information is being shared as open data, so third party developers and researchers can also use the dataset for all kind of projects and purposes. Furthermore, there exists an App from SFpark itself to help drivers find available parking spaces nearby, navigate to them, and pay as they go with their phones.

A top priority goal of the SFpark project is to increase the availability of parking spaces in every block throughout the city. To achieve this goal, they are using demand responsive pricing. If (almost) all parking spaces in a neighbourhood are occupied for a long time, they raise the price in this specific area and vice versa if no parking spaces are occupied they lower prices. This leads to high overall parking space availabilities (20 - 40%) and also to lower traffic congestion and lower greenhouse gas emissions. However, besides the advantages of high accuracy and being a real-time system, there also are disadvantages. First of all, only metered parking spaces can be tracked, as sensors have to be installed per parking space. Thus, areas where parking is allowed but there are no clearly marked parking spaces cannot be sensed with a reasonable accuracy. Another drawback are the high overall costs. The about 8.200 stationary sensors have to be bought, installed and maintained, which obviously causes high costs while only covering a tiny fraction of the overall San Francisco down town. So if the system should be available in the whole city, costs would increase dramatically.

2.1.2 Stationary Park Sensing using Cameras

Another approach while using stationary sensors is to use fixed deployed cameras which continuously record images of parking areas and analyze them for vacant parking spaces. Cameras can monitor up to one hundred parking spaces simultaneously with an accuracy of up to 96%. Challenges of image detection are of course different lightning and weather conditions as well as occlusions depending on the angle which the camera records the parking scene. Detection algorithms using standard digital

image processing will be discussed in this section as well as approaches using deep learning and convolutional neural networks (CNNs).

Parking detection using digital image processing

There exist a few common approaches using digital image processing to detect the state of parking spaces using a captured image of a parking area. First of all, edge detection is often used for parking space classification. Common edge detectors such as the Canny Edge Detector [4] or the Sobel Operator [7] can be used to derive the edge pixels of an image. The edges or edge pixels are counted and if the amount is above a certain threshold, the space will be detected as occupied. The assumption behind this approach is, that usually a vacant parking space has a plain surface and therefore a low amount of edges whereas an image of a parking car should have a lot of edges. Blumer et al. [2] and Liu et al. [15] both used this technique as part of their algorithms.

A slightly different yet related approach object counting [15]. The edges of the image segment of a parking space are analyzed and closed contours (treated as objects) are detected and counted. Then, depending on a threshold which has to be set first, a parking space is classified as either vacant or occupied depending on the object count.

Another common image processing technique is to use foreground/background information of the images. Here, the main background color is identified and compared to the whole image. The background of a parking space can either be defined via extracting a certain part of an image, which should always represent the main background color of a parking space's pavement (done by Blumer et al. [2]), or via an histogram of the image assuming that the background uses the most pixels in a recorded image (done by Liu et al. [15]). After the background color is available it is being subtracted from the original image and using thresholding foreground and background pixels are identified and counted. Depending on the count of the respective pixels, the parking space is then classified as vacant or occupied.

Liu et al. [15] use all of the above mentioned techniques in combination to build a more stable prototype. They only test their prototype indoors which is why weather and lightning conditions do not cause problems. With sensing only a maximum of seven cars, it provids an ensemble technique which should be more reliable. However, the work does not include an evaluation of how well their algorithm performed on a bigger dataset. Another ensemble method is developed by Blumer et al [2]. They use

edge counts and background/foreground information as input for different machine learning techniques, which then classify a parking space as vacant or occupied. The plain algorithm achieves an accuracy of about 77.8%. By improving the algorithm using frames of preceding and following images to identify parking/unparking events, an accuracy of about 88.8% is achieved.

Parking detection using deep learning and CNNs

In recent years deep learning gained a lot of significance in the field of machine learning. Deep learning is a specific variant of neural networks with a high amount of hidden layers. For example in image recognition, convolutional neural networks (CNNs) nowadays perform as well as humans in classifying everyday objects in digital images. A CNN is a neural network with a possibly large amount of hidden layers of which some of them are convolutional layers. In the case of image recognition, CNNs take the spatial relationships between neighbouring pixels into account. This rise in learning power also leads to projects which try to train CNNs in classifying parking situations while using camera images of a parking area.

Amato et al. [1] train two different CNNs on classifying parking space states and compare their performances afterwards. They use the publicly available PKLot dataset as well as a dataset collected by themselves to train and evaluate the neural networks. In total over 700.000 images in different weather conditions and with different levels of quality are processed (in some situations parking lots are almost fully occluded by trees or lamp posts, etc.). The results of the CNN's classifications are promising. Despite the fact that some of the images do not exactly match the parking space and there are a lot of occlusions, the best performing deep neural network achieves an accuracy of above 91% on all subsets of the datasets.

Another work in the field of deep learning is reported by Di Mauro [6]. They also use CNNs and the PKLot dataset as well as a self acquired dataset. However, a slightly different CNN is employed together with pseudo-labelling of the data, which can be seen as semi-supervised approach. When using pseudo-labelling both labelled and unlabelled data are used at the same time to train the network. For unlabelled data the label which was computed by the CNN in the forward pass is used, that is also why it is called pseudo-labelling. Furthermore, a different loss function has to be used as the pseudo labelled data has much less significance than the ground truth labelled data. Di Mauro et al. trained the CNN with about 5% of the data and achieved an accuracy of above 96% with a the fully supervised approach. The pseudo-labelling

approach reached about the same level of accuracy, however, the dataset had to be balanced for it to work properly.

Advantages and disadvantages of using stationary cameras

Advantages of cameras are the high level of accuracy they reach while covering a lot of cars at once. However, for all of the above discussed approaches to work, cameras would have to be calibrated manually before their usage to know where parking spaces are located. Furthermore, most of the research focused on the use of cameras for detecting the parking space situation of a single parking lot with many cars rather than for road side parking spaces. For this approach to work at a city wide scope, cameras would have to be mounted at least at every block. Similar to the dedicated sensors per parking space (section 2.1.1) this would cause high costs not only in the form of hardware but also in installation and maintenance costs. Furthermore, mounting cameras throughout the city obviously also brings up privacy issues.

2.1.3 Counting in- and outgoing Vehicles

A rather simple approach of estimating the number of free parking spaces is to count in- and outgoing vehicles at parking zones. Zadeh et al. [24] present a prototype which counts all in- and outgoing vehicles using a Raspberry Pi and two ultrasonic range finders at each gate. Ultrasonic sensors measure the distance from the side of the road to a potentially passing vehicle. Two ultrasonic sensors in sequence are needed to detect if a vehicle is going in or out. According to the order in which the ultrasonic sensors detect a passing vehicle it is decided whether a vehicle is going in or out. Furthermore, the two sensors are also used to differentiate a passing person from a vehicle. Both sensors are as far apart as they can detect a vehicle at the same time but not a person. Misleading detections of passengers are filtered out using this technique.

As described this approach is technically simple to develop and the hardware costs are also very low. However, this system is only designed for parking lots which have a low number of gates where cars go in and out and not for a city wide deployment. For closed parking areas it is easy to detect in- and outgoing vehicles whereas it is hard to do so in open traffic and street scenarios. Furthermore, the counting also will not work with ultrasonic sensors in open street scenarios. For example, there are often several lanes, which would make the sensing impossible as passing cars would possibly occlude each other. Another issue is that with the installation of the required sensors

they also have to be calibrated for the specific scene to reach a sufficient performance (e.g. the distance to the road). The calibration effort raises the costs for this approach and also makes it inflexible to context changes.

2.1.4 Event Detection based Park Sensing using Smartphones

Detecting parking and unparking events using a driver's smartphone is another approach to estimate parking space availabilities, which is especially interesting because nowadays almost everybody in the developed world has a personal smartphone. As soon as these parking related information is available for a high enough rate of drivers, a city's parking availability situation can be roughly estimated. There already exist a few research papers which discuss different approaches to this topic. Three of these will be discussed in this section.

In 2013 Nawaz et al. developed ParkSense [19] which is an application for Android phones to detect certain events which relate to the change of parking situations. Using the app a user can log his parking events and furthermore the costs of parking. The app is released to the Google Play Store and recorded 59 parking traces in four different cities at the time their paper was released. However, the app not only records the times when parking and unparking takes place, but also records a detailed profile of Wi-Fi access points while parking as well as Wi-Fi profiles and GPS locations after unparking takes place. Using this dataset, the goal of the prototype was to evaluate how accurately unparking events can be detected using Wi-Fi fingerprints. ParkSense takes several of such fingerprints when parking takes place and then continuously scans the Wi-Fi signal and compares it to the signal while parking. Using this technique ParkSense can detect when users return to their vehicle. However, it cannot be assumed that when a user returns close to his vehicle, that he will leave with his car, therefore Nawaz et al. also implement an activity detection for driving using the change of Wi-Fi signals over a specific timespan. The idea behind this approach is, that while driving, Wi-Fi access points in range will change much more frequently than while walking or while staying at the same place. Out of 41 cases where unparking actually took place ParkSense could detected 38 user returns correctly and 35 times out of the 41 it could detect that the vehicle started driving. Thus, ParkSense reached an overall accuracy of about 85% in detecting unparking events.

A similar approach to the problem of parking related event detection is described in [16]. The overall goal of this research paper is to robustly identify parking and unparking events using different sensors of a typical smartphone. In total, nine indicators are identified which can give hints whether a parking/unparking event occurred. For

instance, they use several accelerometer measurements to identify the change of the user from walking to driving and vice versa. Other indicators are the Bluetooth signal strength of the car, sounds of a started motor sensed via a smartphone's microphone, Wi-Fi fingerprints and accessing parking payment apps. To decrease the amount of energy consumed by the application, certain low power sensors (e.g. accelerometer) measure periodically while other high consumption sensors (e.g. microphone) only confirm or reject events which were already sensed by low power sensors. Using all indicators, Ma et al. build a model based on conditional probabilities which should be able to show the probability of an event at a certain time. To evaluate the approach, an Android application is used to manually collect ground truth events. 40 parking samples are collected to train the model and 60 other samples are used for testing. The best configuration achieves 93% recall and 90% precision on parking events and 81% recall and 93% precision on unparking events.

Estimating events by non-tracked drivers

All of the above mentioned approaches only work if almost all drivers in a specific city have the respective smartphone app installed, so that all parking and unparking events can be detected. That way the states of all parking spaces would be known and parking availability could be derived. However, this is highly unlikely as people would have to know that such an app exists in the first place and furthermore there is no instant positive effect of having such an application installed for a single user. Such an app might also have a negative influence on the battery life of a user's smartphone even if all of the above applications are designed for a low power consumption.

Facing this problem Nandugudi et al. propose PocketParker [18], a smartphone application which not only detects parking and unparking events but also estimates the amount of so called "hidden drivers" - drivers that do not have the application installed. PocketParker uses a simple model to detect parking and unparking events. It uses the Google Play Services activity recognition library to detect changes from driving to walking and vice versa and thus derives parking and unparking events. After this information is available, PocketParker tries to estimate the amount of drivers having the application installed, so that all detected events could be scaled to the right proportion and taken into the overall parking estimation process. The PocketParker app not only identifies the parking space location, but also the user's end target after parking his vehicle. That way they want to identify parking spaces which would be nearer to the end destination of the user, but probably are occupied since the user parked further away and did not take the closest parking space. Using this approach PocketParker derives parking space counts at dedicated parking lots. The

overall parking space count of these parking lots is known before. To evaluate its performance Nandugudi et al. performed an experiment at their university's parking lot. 105 users generated 10.827 events over 45 days, therefore an average of 241 events per day. They use cameras to identify the ground truth and then evaluate their results against it. With an estimated driver fraction (drivers which had the app installed) of about 20%, PocketParker achieves a parking space count accuracy of about 94% for the university's parking lot.

Advantages and disadvantages of using an event based approach

There are a number of advantages using smartphones to detect parking availabilities. Nowadays smartphones are wide-spread among people in the developed world. Therefore, smartphones are the optimal sensing devices for detecting a user's activities throughout the day. Also parking and unparking activities can be detected and parking availabilities can be derived. The whole detection process causes no additional costs for hardware installations and maintenance for city authorities. Furthermore, the detection process can all be done without the user even noticing, because the detection process can theoretically run fully as a background process.

However, a big concern of mobile applications is power usage. Activity detection requires certain sensors to run periodically which prevents the phone from being idle and therefore uses more battery. Even if all of the discussed approaches intentionally use high power consuming sensors like GPS sensors or microphones only in rare cases, the immanent sensing of low power consuming sensors will also cause decreased battery hours. Another disadvantage is that a lot of users would have to install the app for accurate parking availability estimations. Even if Nandugudi et al. [18] show that the so called "hidden drivers" can be estimated to a high degree, it will still be challenging to get a user base of even 20% as they propose in their paper. The first obstacle would be to let users even know that there exists such an app, so that they can install it. The following problem would be the cold start problem. After the app is released, usually only a few people will have it installed and they will probably not see an advantage of such an app because the availability estimates will not be accurate if there is only a really small user base.

2.1.5 Drive-by Park Sensing using Cameras

In contrast to approaches with stationary sensors which were discussed in sections 2.1.1 and 2.1.2, drive-by sensing approaches use sensors which are mounted on vehicles

driving through the city to sense vacant and occupied parking spaces. Using such mobile sensors has the goal of achieving a high enough accuracy at a fraction of the costs of stationary solutions. In this section the usage of cameras mounted on driving vehicles to detect parking spaces is discussed.

There exist a lot of different approaches in detecting parking spaces from a picture captured by a camera which is mounted on a vehicle. Most of are were designed for the purpose of park assistance, so that parking is being made easier for drivers or that the vehicle can even park itself. However, almost all of such approaches could also be applied for camera-based drive by sensing.

The first discussed approach is to use parking markings. Xu et al. [23] introduce a color vision based parking space detection algorithm which tries to identify parking markings on the street. They use neural networks to learn to distinguish parking marking pixels from others and they use stereo vision to identify obstacles on parking spaces. A similar approach is followed by Jung et al. [12] who applies a hough transform on the captured images to detect lines (parking markings) on the images. However, the accuracy of color vision based approaches, as just discussed can vary highly due to different lightning conditions, conclusions or shadows. Furthermore, this approach obviously only works with parking spaces where markings are present.

Another approach is to use two or more images to create depth maps to estimate vacant parking spaces. Kaempchen et al. [13] developed a stereo camera based approach which estimates depth maps based on two images of calibrated cameras placed on the vehicle next to each other. By finding features in both images and correlating them, the disparity and thus the depth of points can be estimated and vehicles as well as free spaces can be detected using the 3D information. However, a stereo camera solution has the drawbacks of relying on sensitively calibrated cameras which should have a reasonable resolution to be able to match the features in both images.

Depth maps can also be generated using a single camera with motion stereo-based 3D reconstruction. In 2010 Suhr et al. [22] developed an approach using a fish eye camera mounted on a vehicle, which is facing backwards. It takes multiple images in a sequence and searches for point correspondences in adjacent images to reconstruct 3D structures and to estimate depth maps. This eliminates the need of a second camera while still providing similar results. Suhr et al. reported a 90% success rate of their approach in detecting vacant spaces.

All of the above discussed camera based drive-by approaches are not designed to detect parking spaces while driving through the city and would have to get evaluated to know

if they are suited for the task. However, there exists a recent work of Grassi et al. [8] about drive-by sensing to detect parking space availability at a road segment accuracy level. They introduce a smartphone application, called "ParkMaster". The user has to mount his smartphone behind the wind shield of his vehicle so that the smartphone camera is able to capture images of the area in front of the vehicle. ParkMaster tries to detect parking cars using the camera's video stream and counts them. In combination with parking space counts at a road segments level the number of vacant parking spaces then gets estimated. For the detection of the parking cars ParkMaster uses the video stream of the smartphone camera as input for a Viola-Jones feature-based cascade classifier, which is a machine learning technique to detect complex objects within images. Several positive and negative examples in different lightning conditions were used to train the classifier in an offline process. While driving, the smartphone searches for the learned features in the captured images and the classifier reports the bounding boxes of detected vehicles. However, as the vehicle is driving, it might detect the same vehicle in multiple subsequent frames of the captured video sequence. ParkMaster handles this problem by estimating the GPS coordinates of detected parked vehicles and then compares their position to identify multiple detections of the same vehicle. For the calculation of the sensed vehicle's GPS coordinates, ParkMaster uses the coordinates of the bounding box in the image in combination with self calibrated parameters of the smartphone camera.

Grassi et al. evaluated their prototype with real world experiments in Paris, Los Angeles and a small village in Italy[1]. During their test drives, they recorded a dataset containing 5.896 parking cars and 2.280 vacant parking spaces. They manually assigned ground truth values, optimized the variables of their algorithm and tested their approach on the dataset which produced an overall accuracy of close to 90%.

Advantages and disadvantages of camera based drive-by park sensing

As only the last approach discussed in this section, is designed for parking space availability estimation, this paragraph will be focused only on ParkMaster's advantages and disadvantages. The main arguments, Grassi et al. give to promote their approach is the high enough accuracy and that high hardware costs can be avoided, because of the use of smartphones of end users as processing devices. If smartphones are used, costs would be lower, however, as already discussed in section 2.1.4, smartphone apps would need to have a high enough distribution for enough drivers to sense available parking space. Furthermore, drivers would have to mount their phone each time they drive through the city and this obviously also costs a lot of battery hours. Therefore,

[1] Sant' Angelo in Vado

it is not sure if a smartphone based system would give a good enough coverage. Of course dedicated cameras could be used as part of a hardware installation in a car, but this would of course introduce new costs, while still keeping costs lower than with stationary sensor deployments. Further limitations of drive-by park sensing in general will be discussed section 2.1.7.

2.1.6 Drive-by Park Sensing using Distance Sensors

Another approach of drive-by sensing is to use a distance sensor instead of a camera as discussed in the previous section. The approaches discussed in this section are closely related to the work presented in this thesis. Two different approaches will be discussed in this section. First an approach using a 1D ultrasonic sensor will be described and furthermore another prototype using two 2D LiDAR scanners will also be examined.

Drive-by park sensing using a 1D ultrasonic distance sensor

Mathur et al. [17] developed a drive-by prototype, called ParkNet, in 2010. It consists of an ultrasonic range finder which continuously measures the distance to the nearest obstacle on the right side of the road (at an interval of about 50 ms - 20 measurements per second) and a GPS receiver which records the corresponding locations. Furthermore, a camera is used for obtaining ground truth information through manual tagging of the captured images. Mathur et al. deployed their system on a standard car, which collected test data in drives during daily commuting throughout 2 months in selected areas in Highland Park, New Jersey. Altogether, about 500 miles of street parking scenes were collected during their test period.

ParkNet's detection algorithm is based on thresholding. It tries to detect so called "dips", which are parts of the sensor signal caused by parking cars or other objects. Sensor readings are separated from each other by overflow measures which show that at that point the distance is higher than the range of the ultrasonic sensor. In a second step, these dips are compared to two thresholds for the distance and the length of the dip. Both thresholds have been derived by a subset of the data and produce an overall error rate of 12.4% on this training set. ParkNet assumes that detailed parking space maps are available. They distinguish between a slotted and unslotted model for their parking space detection. In the slotted model, they assign detected parking cars to single parking spaces using the GPS position. However, GPS positions are not always as accurate as they need to be to correctly identify a parking space. Therefore,

ParkNet uses an environmental fingerprinting algorithm which corrects GPS positions using multiple test runs on the same street. They identify static objects which are there on all sensor traces and then cluster their positions. Mathur et al. use the first cluster center as true position and correct the surrounding GPS measurements. In the unslotted model, ParkNet estimates the number of free parking spaces by checking for all free spaces if the length is high enough for a car to park in between the two enclosing objects.

Mathur et al. used their self acquired dataset to evaluate their algorithm. On the unslotted model, they tried to estimate the number of vacant parking spaces in a specific street. Their algorithm turned out to have an overall accuracy of about 95%. For the slotted model they reach an accuracy of 91% using their environmental fingerprinting approach to correctly assign detected parked cars to parking spaces.

To show that their system is more cost effective than stationary sensor deployments, Mathur et al. also present a mobility study which has been done in the city of San Francisco, California. They try to estimate the interval in which a sensing vehicle would visit a street. The GPS locations of 536 taxi cabs which were collected during one month are used and the analysis of these data shows that while in the outer San Francisco areas visiting intervals can be as high as hundreds of minutes, in the down town areas 80% of the streets are visited every 10 minutes. Furthermore, they estimate the costs of running their system to be more than 12 times cheaper than with the use of stationary sensors at each parking space.

Drive-by park sensing using two 2D LiDAR distance sensors

Bock et al. [3] use two 2D LiDAR distance scanners to approach the problem of drive-by park sensing. They are both deployed on a regular car facing to the right side of the road with an angle of 60 degrees in between them. The position of the vehicle is measured via a GNSS sensor. The LiDAR scanners measure at a frequency of up to 300.000 measurements per second and produce 3D point clouds of the sensed scenes. These point clouds are used to detect parking cars. In a first step the data is preprocessed and segmented. For the segmentation Bock et al. use a ”similar region growing approach” to differentiate between multiple objects in a single scan. They use a random forest classifier to classify sensed objects into two distinct classes: non-moving objects and other objects. The random forest classifier is being used because it is less prone to overfitting, fast and easy to interpret. As input for the classifier, several calculated features based on the geometry of the sensed objects are used. To distinguish moving cars from non-moving ones, Bock et al. use two LiDAR scanners

instead of one. Because of the angle of 60 degrees in between the two sensors, moving vehicles can be identified as they appear at different times in both sensors and therefore at different positions.

For the evaluation of the performance of their prototype, Bock et al. obtained a dataset in Hannover, Germany. They gathered data of 1.313 parallel and perpendicular parking cars as well as 1.360 other objects. Altogether, their approach reaches a high accuracy of 95.8% recall and 98.4% precision for parallel parking cars and 93.7% recall and 97.4% precision for perpendicular parking cars. However, despite the good results for the detection, 2D LiDAR systems are very expensive in comparison to one dimensional sensors as used by ParkNet (described in the previous paragraph).

Advantages and disadvantages using distance sensor based drive-by park sensing

As already discussed the use of 1D distance sensors for a drive-by approach can be much cheaper than stationary sensor deployments while providing a sufficient accuracy with enough sensing vehicles. In contrast to camera based drive-by sensing a higher accuracy can be achieved, due to the fact that detailed distance information is instantly available and does not have to be derived from camera images. The presented camera based drive-by sensing approach can only detect cars and then estimate the vacant parking spaces using an overall count of parking spaces per road segment. Using distance information distance sensor based drive-by approaches are also able to detect free spaces which improves the overall result. The use of 2D LiDAR distance scanners brings another gain in accuracy, however, such systems are still very expensive and therefore, not suited for the use of a lot of vehicles sensing the parking situation in a city. Further limitations of drive-by park sensing in general will be discussed in the next section.

2.1.7 Limitations of current drive-by Park Sensing Approaches

In this section limitations which apply to both camera based- and distance sensor based drive-by park sensing will be discussed. The first limitation is based on the cars which are detectable. Most drive-by sensing approaches currently only try to detect limited types of parking cars. Even if in most cities parallel parking spaces are the most prominent type of parking spaces, obviously there exist a lot of other parking spaces (e.g. perpendicular parking cars and angular parking cars). Camera-based drive-by sensing would have to also learn to detect images of such parking cars and

distance sensor based drive-by sensing would have to be much more variable in terms of the length of detected cars as well as the patterns of the detected distance to the obstacle. For instance, in the case of angular parking spaces, the discussed approach of the ParkNet prototype would not work as they only apply thresholding and therefore it is likely that such parking spaces would not be segmented and detected correctly.

Another limitation of drive-by sensing are multi-lane roads. Most of the discussed drive-by sensing approaches only work when the sensing vehicle drives in the right-most lane. In modern cities, there are often multiple lanes and GPS is too inaccurate to estimate the lane, where a vehicle is going. Therefore, to work properly, lane detection would have to performed for the systems to work on multi lane roads. However, it would be preferable to have a algorithm which also works in other lanes than the right most one as then the coverage of detections would be greater. For algorithms to work on non-right lanes, they also have to face several distractions. Of course, the distances to parking cars will be greater if the sensing vehicle is going on a left lane. Additionally, when a sensing vehicle overtakes another car which is going slower, it will record the distances to this vehicle instead of the distance to parked cars. Furthermore, also distractions such as traffic lights or traffic jams have to be taken into account for a real life installation of such prototypes.

2.1.8 Comparison of the existing Park Sensing Approaches

Table 2.1 shows a comparison of the park sensing approaches which were discussed in this section. All of the above discussed approaches show high accuracy (>= 90%) but not all of them are suited for all kinds of park sensing. For instance, the drive-by approaches are not able to detect parking availabilities in dedicated parking lots or parking garages whereas vehicle counting cannot detect the parking situation of road side parking spaces and thus a city's overall parking situation.

The costs of the different approaches also vary highly. Approaches with stationary sensors usually have the highest accuracy, but also bring the highest costs. Event-based sensing operates on very low costs because it usually uses smartphones as sensing and processing devices. However, to reach a high accuracy a lot of people have to sense parking events. Furthermore, such apps will also drain the user's battery faster than usual.

Drive-by park sensing approaches are in the medium cost range because the vehicles which are sensing have to be equipped with the required sensors, but in contrast to stationary sensing, much fewer sensors can reach a suffcient degree of accuracy.

	Accuracy	**Costs per sensor setup**	**Limitations / Drawbacks**
Stationary Sensors	100% [21]	$250 - $800	- High amount of sensors needed - Therefore high overall costs
Stationary Cameras	88.8% [2] 91% [1] 96%[6]	about $85	- Cameras can only cover limited area - A lot of cameras needed for city-wide deployments - affected by lightning and occlusions
Counting Vehicles	100% [24]	about $45	- hardly applicable in open street scenarios - gates needed for in- and outgoing vehicles
Event-based Sensing	85% [19] 89% [16] 94% [18]	$0	- high user base required (crowd sensing) - drains battery of the user's smartphones
Drive-by Sensing (Camera)	90% [8]	$0	- high user base required (crowd sensing) - detection of different types of parking cars - multi lane and overtaking situations - detecting standing cars (e.g. at traffic light) - avoiding multiple detections of the same car
Drive-by Sensing (1D-Distance sensor)	93% [17]	about $400	- detection of different types of parking cars - multi lane and overtaking situations - detecting standing cars (e.g. at traffic light)
Drive-by Sensing (2D-Distance sensor)	96% [3]	$1000 - $5000	- high costs per sensor setup - detecting standing cars (e.g. at traffic light)

Table 2.1: Comparison of all park sensing approaches

	Accuracy	**Dataset**
Convolutional Neural Networks	91% [1]	about 700.000 images [1]
(Stationary cameras)	96% [6]	about 970.000 images [6]
Viola-Jones feature-based	90% [8]	5.896 parked cars and
cascade classifier		2.280 available parking spaces
(camera-based drive-by sensing)		
Random Forest Classifier	96% [3]	1.313 parking cars and
(2D distance sensor based drive-by sensing)		1.306 other objects

Table 2.2: All used machine learning models to detect a city's parking space availability.

However, there are certain limitations of the current approaches as described in the previous section. In this thesis, a prototype using distance sensor based drive-by park sensing will be developed and evaluated, as the ratio from cost to accuracy seems to be the best of the investigated approaches. To improve accuracy in real life scenarios, machine learning will be used to identify all types of parking cars as well as different distractions such as overtaking while driving on multi lane roads.

Used Machine Learning Approaches

Table 2.2 shows an overview of all machine learning algorithms which were used for the classification of parking situations in cities in the related work. Convolutional neural networks are used to classify images of parking lots obtained by stationary cameras (Section 2.1.2), whereas a Viola-Jones feature-based classifier is used in camera-based drive-by sensing (Section 2.1.5) and a random forest classifier is used in the 2-dimensional distance-sensor based drive-by sensing (Section 2.1.1).

2.2 Acquiring Parking Space Maps

Parking space maps are of great importance when sensing a city's parking availability. Knowing where parking is allowed and how parking is possible is crucial information for park sensing approaches. Several of the in section 2.1 discussed approaches use some kind of parking space map. Coarsely grained parking maps in the form of parking space counts for road segments/parking lots are necessary for several approaches, like the counting all in- and outing vehicles approach discussed in section 2.1.3. Furthermore, such information is also necessary for event-based park sensing (section 2.1.4)

as well as drive-by sensing (2.1.5 and 2.1.6). Of course parking space maps with a higher granularity can be beneficial in most cases and are sometimes even necessary. For instance, the exact position and orientation of parking spaces can increase the performance of drive-by sensing.

Most previously discussed papers assume that parking space maps are already available, for example at city authorities. However, even if a city authority has a parking space map of their road side parking spaces, that does not mean that it is ready for processing and may not contain the necessary information. That is why Coric et al. [5] developed an algorithm to derive parking space maps from the data stream of an ultrasonic range finder. They used the same equipment as ParkNet [17] to record GPS and distance information to the right side of the road while driving through streets in their test zones. Using their testbed, they acquired two datasets in Highland Park, New Jersey and in Brooklyn, New York City. After obtaining the data, they try to predict areas where parking is legal and such where it is not. Their hypothesis is that spaces which are almost never filled will be most likely illegal spots, whereas spaces which are often occupied will be likely legal parking spaces.

Using sensor readings from several runs they try to aggregate this data to derive their results. They first assign the sensor readings to one meter cells and then try to classify these cells as legal or illegal parking spots. Using their algorithm, called ”Weighted occupancy rate thresholding approach”, they not only use occupancy rates per cell, but also take into account, that runs with almost only occupied cells provide better information than runs with a lot of vacant spaces, as usually only the illegal spaces stay vacant, when the parking availability is low in a certain street. They compute a occupancy rate per cell and weight it using the overall occupancy of the street. Then they apply thresholding to the occupancy rates to classify whether parking in the range of a cell is legal or not. Coric et al. evaluated their approach using the obtained dataset and found that their acquired parking space maps have a false negative rate of 5.21% and a false positive rate of 5.89%. Thus, using the specified setup, parking space maps can be derived at a good accuracy and may be used for the approaches stated in section 2.1.

To-Do: short paragraph about deriving parking space map in this thesis?

Chapter 3

Testbed - Prototype Implementation

As already discussed in Chapter 1, sensing a city's parking availability and thus making the parking situation more transparent would be highly beneficial for reducing traffic congestion and greenhouse gas emissions as driver's looking for parking spaces in urban areas could navigate directly to vacant parking spaces close to their destinations. To support this goal, a prototype of drive-by sensing using an optical distance sensor has been designed, implemented, and evaluated. This chapter introduces the prototype and testbed which have been developed to acquire the necessary dataset to run the machine learning experiments. First, the used hardware and sensors are described (Section 3.1), then the software to collect test data is being discussed (Section 3.2) and finally the acquired raw sensor dataset is shown (Section 3.3).

3.1 Used Hardware and Sensor Parts

Figure 3.1 shows the sensing car and its mounted sensors. For collecting, processing, and saving the sensed data a Raspberry Pi 2 Model B[1] is used. A Raspberry Pi was chosen because of its simplicity to connect and access sensors and moreover because it is really easy to program as it is just a regular Linux-based computer. Furthermore, the price of a Raspberry Pi is also quite low (about €35), therefore the overall cost of the system will remain low.

To determine the location of the sensing vehicles while driving through the city, a Navilock USB GPS receiver[2] is used. The receiver is connected to the Raspberry Pi

[1] `https://www.raspberrypi.org/products/raspberry-pi-2-model-b/`
[2] `http://www.navilock.de/produkte/G_61840/merkmale.html?setLanguage=en`

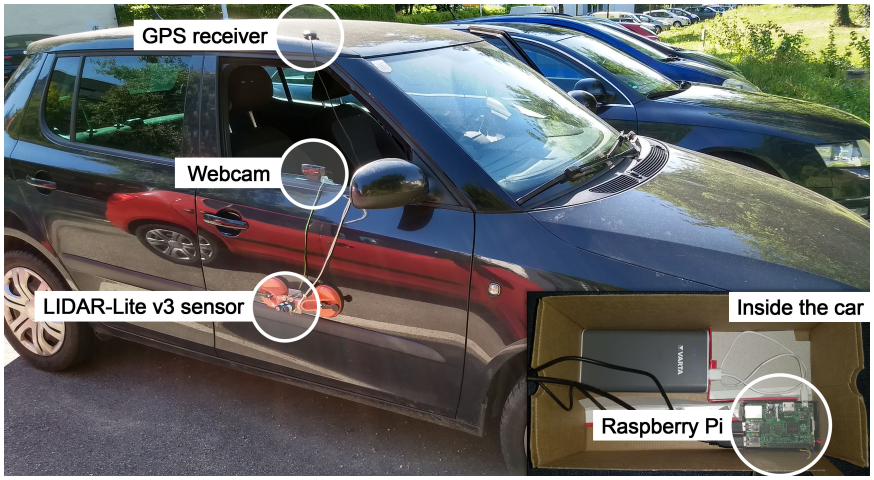


Figure 3.1: Prototype of the sensing car, which is composed of a LIDAR Lite v3 optical distance sensor, a GPS receiver, a camera for ground truth collection and a Raspberry Pi as processing device.

using a USB port and measures the GPS location (latitude and longitude) at a rate of about 1 Hz. According to the manufacturer the acquired positions are correct in the range of 2.5 meters. The GPS receiver is mounted on the top of the sensing car using a magnet in order to receive more accurate sensor readings. The Navilock USB GPS sensor costs about €50.

Two major distance sensor technologies can be differentiated: an ultrasonic range sensor and an optical laser distance sensor. While ultrasonic sensors are cheap and widely used in commercial parking assistance systems, they also have limitations in terms of sensing frequency (about 20 Hz) and range (a few meters). These specifications limit the use of ultrasonic sensors for drive-by sensing on multi-lane roads or high speed. Due to these limitations an optical distance sensor, the "Lidar Lite v3[3]" is used in our testbed. It measures the time of flight of an emitted laser signal which is reflected by an object in its way. The Lidar Lite v3 can measure distances from a few centimeters up to 40 meters at a frequency of 1 to 500 Hz. Furthermore, it is able to measure distances with an accuracy of about 2.5 centimeters. However, in comparison to ultrasonic sensors, optical sensors are more expensive. While the costs of ultrasonic sensors are often below €10, the Lidar Lite v3 and other comparable sensors cost about €150.

The distance sensor is mounted on the co-driver's door using a sucker handle in order to face the right side of the road and to take measures at about 62 cm above the ground. It is connected to the Raspberry Pi via GPIO (General Purpose Input Output) pins,

[3] `https://www.sparkfun.com/products/14032`

	Ultrasonic Range Finder	**Lidar Lite v3**
Costs	from about €5,00 to €100,00	about €150,00
Sampling Frequency	up to 20 Hz (at 10 m distance)	up to 500 Hz
Range	2 cm - 10 m	30 cm - 40 m
Distance between Measurements at 50 km/h	about 70 cm	about 3 cm

Table 3.1: Comparison of ultrasonic sensors and the used Lidar Lite v3 optical distance sensor.

communicating using an I2C (Inter Integrated Circuit) interface, according to the manufacturer's specifications.

Table 3.1 shows a comparison of the characteristics of ultrasonic sensors and the Lidar Lite v3 sensor. The most significant limitation is the sampling frequency. While optical laser sensors are based on the speed of light, ultrasonic sensors are based on the speed of sound, which is much lower. Therefore, also the distance between consecutive measurements varies highly from an ultrasonic to an optical laser sensor. At a speed of 50 km/h, the distance between two consecutive measurements is up to 70 cm using an ultrasonic sensor while it is only about 3 cm when using an optical distance sensor.

A camera is leveraged to capture images of the ground truth, here, a Logitech C922 Pro Stream[4] is used. Images with a resolution of 352 x 288 pixels are captured at a frequency of about 30 Hz while the car is sensing parking spaces. The camera is mounted on the co-driver's door on the same horizontal position as the sensor. This way the center of the image will be the position where the distance measurement is taken. The Logitech camera is connected to the Raspberry Pi using an USB port and is available at about €90.

3.2 Collecting Sensor Measurement Data

A Python script has been developed which saves all sensor readings during the test runs into a text file for later analysis and evaluation. Furthermore, all images captured by the camera are saved into a specific folder on the Raspberry Pi to be able to derive the ground truth information later on (see Section 4.1). The sensing of all three devices (GPS, Lidar Lite v3, Logitech camera) is performed asynchronously in separate

[4] `https://www.logitech.com/en-us/product/c922-pro-stream-webcam`

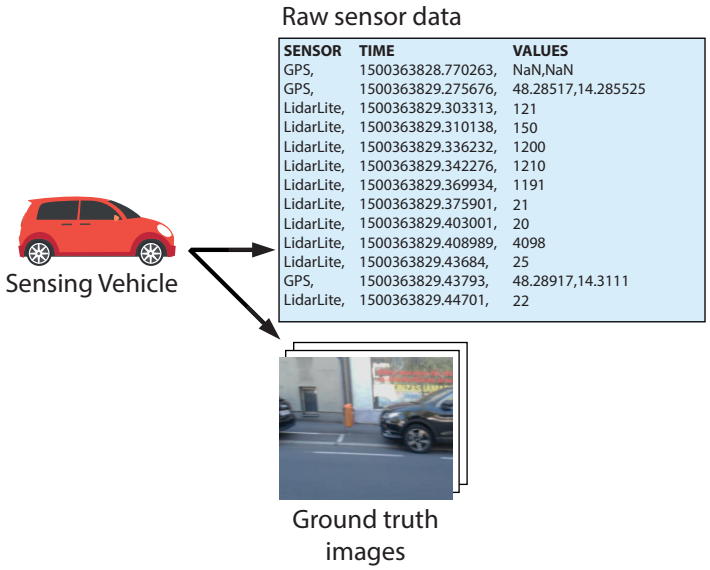


Figure 3.2: Acquiring raw sensor data.

threads so that none of them influence the other sensor's readings. For accessing the GPS receiver the Python GPS library is used whereas for measuring the distance using the Lidar Lite v3 sensor, an open source library available on github.com is used[5].

Figure 3.2 shows how the sensor data gets collected and furthermore presents a sample of raw sensor data derived during a test drive. The sensor data text file contains GPS and distance measurements. The first column contains the sensor type. The second column gives the time stamp of the Raspberry Pi's system time in seconds. GPS measurements also result in latitude, longitude and speed values while the Lidar Lite measurements result in the measured distance in centimeters.

3.3 Raw Data Set

To obtain a dataset, in total 32 test drives were completed in the city of Linz, Austria. The street scenarios have been selected aiming at a high variability of road scenarios. The test scenes include single lane as well as multi lane streets. While collecting the test measurements the sensing car was driving as it would in regular traffic (not only in the right most lane, etc.) and the scenes also include high and low traffic scenarios. For all street scenarios, measurements have been repeated multiple times. Figure 3.3 shows an aggregated GPS trace of all test drives on a map of Linz. Red dots

[5] `https://github.com/Sanderi44/Lidar-Lite`

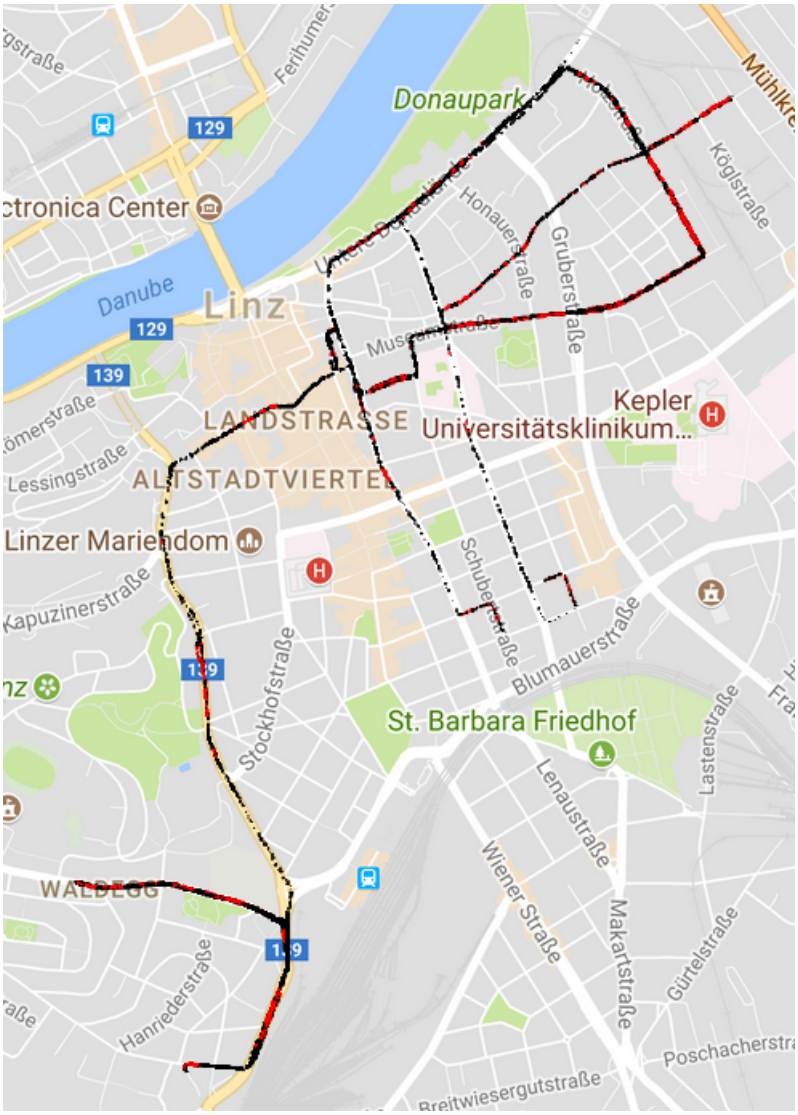


Figure 3.3: GPS trace of all recorded samples in the acquired dataset, while 32 test drives in Linz, Austria. Red dots represent parking cars.

represent parking cars while black dots represent other objects. In total **444.427 distance measurements** and **14.997 GPS measurements** were taken, which makes up about 15,9 megabytes of distance and GPS data. Additionally, **266.309 images** were captured to record the ground truth of the sensed scenes. The images, which make up about **5.072 megabytes**, are used to determine the ground truth of the recorded scenes. Sensed data and ground truth data are applied to supervised machine learning algorithms later on. Ground truth tagging is described in the following section.

Chapter 4

Data Processing and Machine Learning Experiments

This chapter describes all the steps which are necessary to prepare the raw dataset for the use in machine learning. The required steps include manual ground truth tagging (Section 4.1), data preprocessing (Section 4.2), data segmentation (Section 4.3), and feature calculation (Section 4.5.1). Furthermore, the use of parking space maps is being discussed to possibly improve the dataset and also the classification results (Section 4.4). The derived datasets are described in Section 4.5, providing an analysis of the datasets and the features. Finally, the machine learning- and deep learning experiments (Sections 4.6 and 4.7) are discussed including the used machine learning models, used tools and the evaluation process.

4.1 Ground Truth Tagging

The recorded raw dataset contains distance and position measurements as well as images showing the ground truth at a specific point in time. To be able to process the ground truth information included in the images, a manual ground truth tagging task has to be performed. Figure 4.1 shows the user interface of the program which has been implemented to manually tag an image with one of nine pre selected class labels. The file name of each image is the time stamp of the recording. After the images are loaded, the user has to tag the situation which is present at the center of the image (red line). For instance, Figure 4.1 shows a parking car at the center of the image. In addition, on the right side of the image, a car which has been overtaken by the sensing vehicle is also visible, but as it is not in the center, the image is labelled as ”parking car”. Possible class labels are:

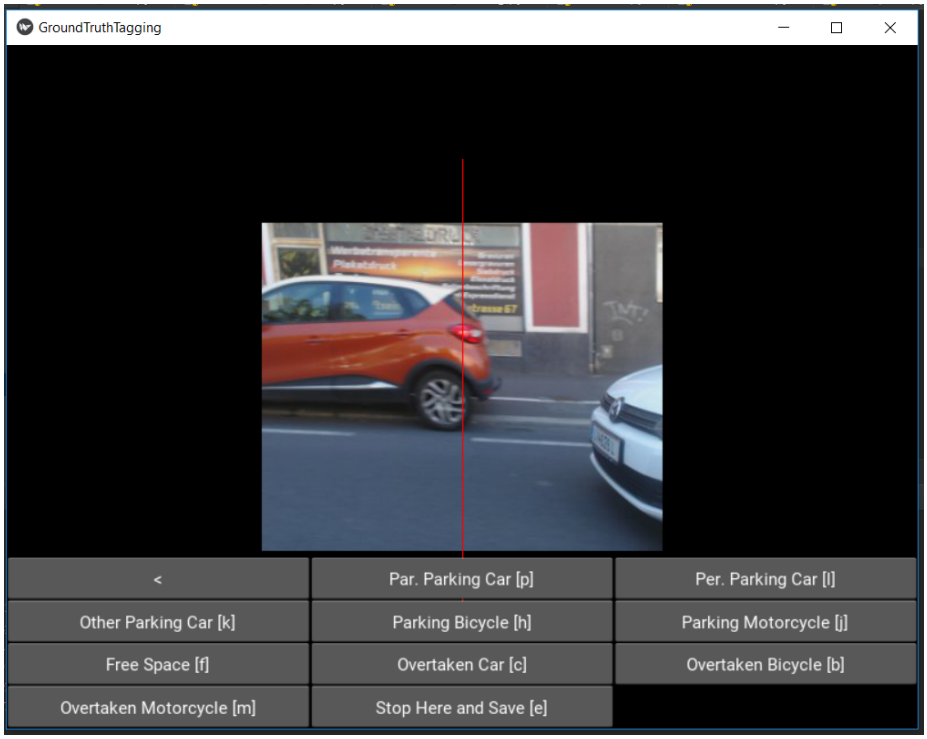


Figure 4.1: User interface for manual tagging of parking situations.

Parking car: A parking car is a non-moving vehicle which is parking at the right side of the road. There are three kinds of parking vehicles: ***Parallel parking cars***, ***perpendicular parking cars***, and ***angular parking cars***. Figure 4.2 shows the different types of parking spaces.

Overtaking situation: This class label means that the sensing vehicle is overtaking another road vehicle. The user can choose between ***overtaken cars***, ***overtaken motorcycles*** and ***overtaken bicycles***.

Other parking vehicle: The user can also tag ***parking motorcycles*** and ***parking bicycles***. This class represents obstacles which can be located on parking spaces and therefore make it unavailable for parking.

Free space: This class label should be used, when none of the above listed class labels are applicable. It represents a free space where it is maybe possible to park a vehicle. Yet only with the help of a parking space map, a free space can be marked as vacant parking space as a random free space may be an illegal parking space (e.g. a driveway).

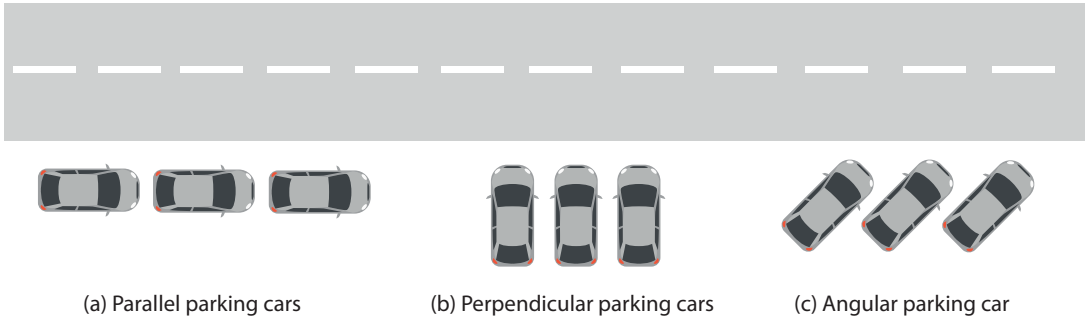


Figure 4.2: Different types of parking cars.

The output of the ground truth tagging process is persistently saved as a text file where each line contains an entry with a time stamp and its corresponding tagged ground truth class label. This ground truth file will be further processed during the data segmentation process which is described in 4.3.

4.2 Data Preprocessing

Before the raw dataset can be used to create the input features for machine learning models, a few preprocessing steps have to be taken. First of all, error values of the GPS sensor and the LiDAR Lite v3 sensor are deleted. The GPS sensor usually shows erroneous output (NaN values) when it starts sensing due to the fact that it is not yet receiving signals from enough satellites. The distance sensor can provide overflow measurements when the target object is too far away or too close to the sensor. In both cases it will result in a distance of less than 10 centimeters. Furthermore, outliers where a single measurement differs greatly (more than 1 meter) from the preceding and following measurements are also detected to avoid over-segmentation in the data segmentation process (discussed in Section 4.3). All of the detected error/overflow/outlier cases are deleted from the raw dataset before further processing to ensure the quality of the sensor measurements.

Another necessary preprocessing step is the filtering of standing situations of the car. For instance, when the sensing vehicle is waiting at a traffic light or in a traffic jam, it most likely measures other waiting vehicles and not parking spaces. Furthermore, the distance measurements are constant for a long time, i.e., measuring the same object. Such situations would lead to misleading samples for the machine learning process and would decrease the accuracy of the classification results. Therefore, all situations where the car is driving at a speed lower than 1 m/s are deleted from the dataset.

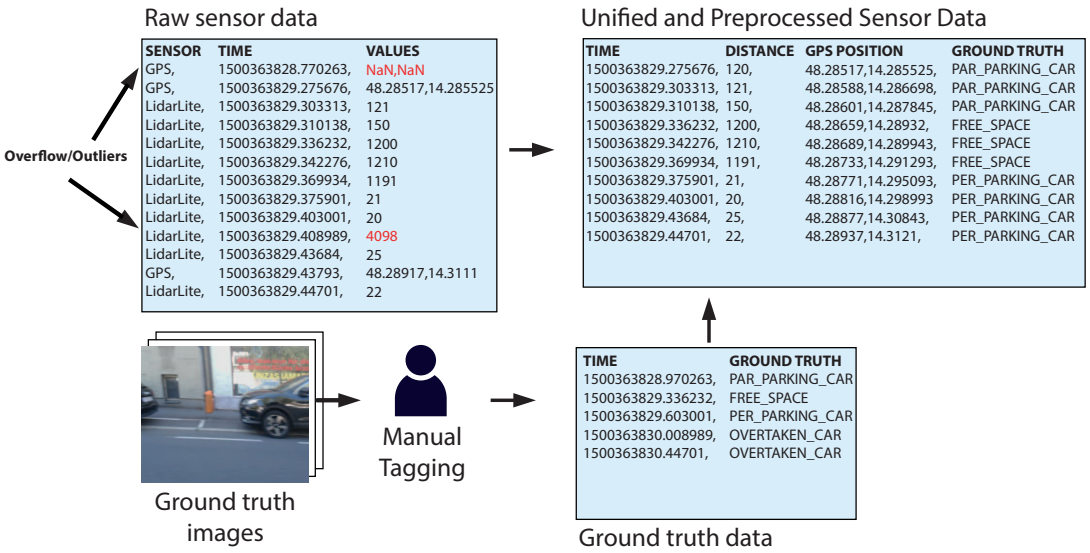


Figure 4.3: **To-Do:** todo All sensor measurements are being merged to obtain a dataset where all samples are containing time, distance, location and ground truth information.

Due to the fact that all used sensors measure at different frequencies, the sensor data of all sensors has to be unified before the next steps can be taken. The distance sensor measures at a much higher frequency (about 100 Hz) than both the GPS sensor (about 1 Hz) and the camera (about 30 Hz). Therefore, the GPS location as well as the ground truth at each distance measurement have to be approximated. Linear interpolation using the timestamp of the measurements is being used to calculate the approximate location and ground truth at the times of all distance measurements. Figure 4.3 shows how the raw sensor data and the ground truth data get unified so that a list of data points containing distance, GPS position and ground truth can be derived.

4.3 Data Segmentation

As the used distance sensor measures at a high frequency, it usually collects several distance measurements belonging to the same object. For example, if the sensing vehicle passes a parallel parked car (having a length of 4 meters) at a speed of 30 km/h it will collect approximately 24 measurements belonging to one passed car. It is necessary to group all measurements concerning this car; this is done by data segmentation of the preprocessed data.

There are several strategies how sensor data can be segmented, for example using sliding windows. However, such approaches will not work accurate enough in the case

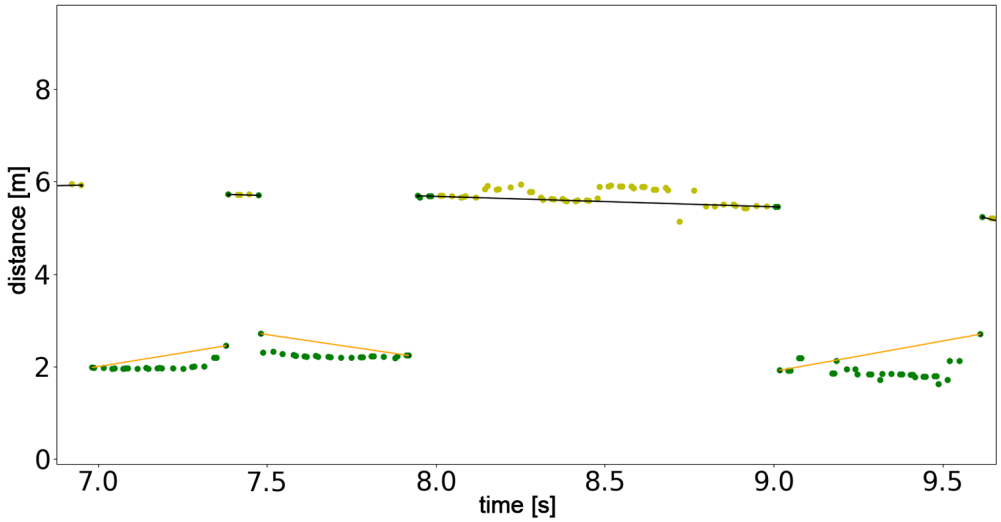


Figure 4.4: Result of the segmentation algorithm. Three parking cars (green dots and orange lines) and two free space segments (yellow dots and black lines) have been detected.

of street side parking detection, as the objects which should be segmented range from a few centimeters to a length of tens of meters. Therefore, a new strategy to segment the sensor data points is proposed. The used segmentation algorithm searches for high frequent changes in the distance signal and starts a new segment when the difference in distance between two consecutive measurements is above a certain segmentation threshold. Furthermore, if the timestamp of two measurements differ more than one second the start of a new segment will also be detected (this can be the case if the sensing vehicle was standing or driving at a very low speed and many measurements have been deleted in the preprocessing steps).

To configure the threshold, experiments have been conducted with varying thresholds. The best threshold to detect the segmentation points in the distance sensor signal has been identified by manually reviewing the resulting segments for under- and over-segmentation. The resulting segmentation threshold is $1,05$ meters. This means that if two consecutive measures differ more than $1,05$ meters, the algorithm will assume that the sensor measurement belongs to a new object (e.g., a parking car or an overtaken car) and will detect a new segment.

Figure 4.4 shows an example of the segmentation process. All points represent distance measurements. The green points represent measurements belonging to a parking car whereas the yellow points show free space measurements. The lines represent detected segments where belong to the same ground truth class. Black lines represent free space and orange lines represent a parking car. Thus, the segmentation shown in Figure 4.4 has successfully detected three parking car- and two free space segments.

Another challenge of the segmentation task is to estimate the ground truth classes of the segments. So far, only single sensor reading got a ground truth class assigned. However, as described earlier, each segment consists of multiple sensor measurements and it should be avoided that one segment consists of multiple different ground truth classes. This phenomenon is visible in Figure 4.4 where both "Free space"-segments are overlapping with measurements marked as "Parking Car". To overcome this problem, the segments are tagged with the label of the majority of the sensor readings. This approach turned out to work well as in most cases only the first and last ground truth values are assigned wrong.

4.4 Parking Space Maps

When sensing a city's parking availability situation parking space maps are of great importance. As already discussed, the machine learning models only classify road segments as "free spaces" (rather than "vacant parking spaces") which means that there are no parking cars. However, it is not clear whether parking is allowed. For example, a free space might just be a driveway or a bus station. Vacant parking spaces can be identified when their location is compared to a parking space map matching parking spaces there.

The use of parking space maps will also possibly improve the performance of the classification task. If only segments in areas, where parking spaces are located, are used as training set, the overall data will be much more balanced (there will be less "free space" segments, but still almost the same amount of parking cars) and moreover the machine learning model will only be trained with relevant samples because only segments which are close to parking spaces have to be classified when generating a parking space availability map. Other segments do not add more information to such a map because it is known that in such areas it is not allowed to park.

4.4.1 Parking Space Map of Linz, Austria

As it is crucial to have accurate parking space maps, the goal was to obtain such maps for the area of the experiment. As a first try, the APIs of OpenStreetMap[1] have been used to obtain map data of the city of Linz, Austria. Parking spaces are listed in the downloaded map data but unfortunately, only parking spaces of big parking lots (of shopping malls, etc.) are available but most of the city's road side parking spaces,

[1] `https://www.openstreetmap.org`

which are being sensed in this work are not present. Therefore, the parking space maps of OpenStreetMap cannot be used due to insufficient data.

Another possibility to obtain a parking space map is to use data from city authorities in Linz. City authorities keep detailed digital maps of the streets as well as the road side parking spaces. However, the only format available are several DXF- or DWG-files, created using the program AutoCAD[2], which is commonly used for designing and drawing as well as for geographical applications. There exist libraries which can read all components of DXF-files and their attributes but unfortunately there are no geographical position data attached to the components and therefore the GPS location of all parking spaces is unknown. This fact makes it impossible to use the parking space map in the park sensing prototype as the parking spaces of the map cannot be matched with the sensed segments obtained during the experiments.

4.4.2 Approximating Parking Space Maps

As parking space maps are unavailable in a sufficient quality from outside sources, an algorithm is introduced to derive a coarse parking space map from the dataset which was acquired during the test drives. The dataset contains the GPS positions of over 2.000 parking cars and most streets have been sensed several times. As of the limited amount of data, the goal is to create a coarse parking space map which contains areas where it is likely that vehicles park (parking zones).

The proposed algorithm uses clustering to group parked cars to parking zones. The idea is that if there are several cars parking at the same location in several test drives then it is likely that there are parking spaces at that position. A DBSCAN[3] clustering algorithm (part of the scikit learn[4] python framework for machine learning) is being used to cluster the position of the parked cars. The DBSCAN algorithm searches for high density core samples and expands clusters from them. It takes the a similarity function which compares two data points and a minimum similarity as input parameters.

As similarity measure for the clustering algorithm a custom function has been implemented. Two parked cars are considered to be similar if the sensing car is driving in the same direction while sensing and if both cars are close to each other in terms of their GPS position. The first check is applied to ensure that both compared cars are

[2] `https://www.autodesk.eu/products/autocad/overview`

[3] Density-Based Spatial Clustering of Applications with Noise - `http://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html`

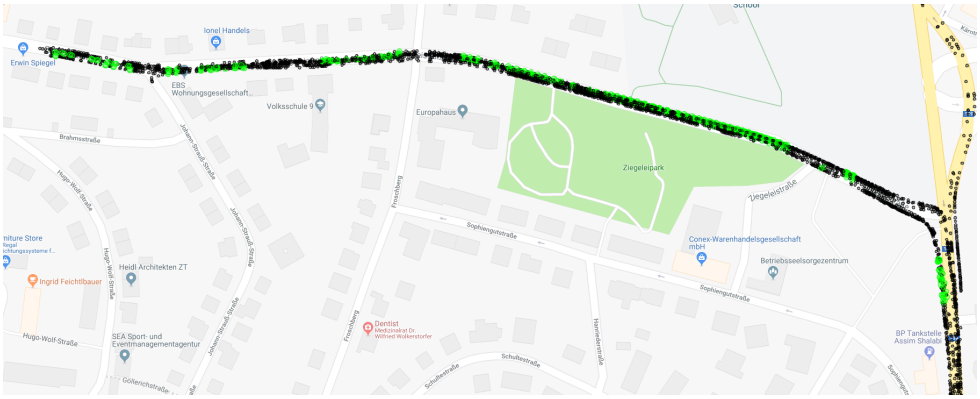
[4] `http://scikit-learn.org/`

parking at the same side of the street. Else it would be possible that two cars which park at the opposite side of the street are considered as belonging to the same ”parking zone”. If the first check is passed, the similarity function will compute the distance in meters between the GPS positions of both parked cars which are compared. The described similarity function is used by the DBSCAN clustering algorithm to identify core samples and to cluster multiple parking cars to parking zones. The maximum distance between two cars to be considered to be similar is set to eight meters. If two parking cars are further apart they are not considered to be able to be in the same parking zone. The result of the algorithm are clusters of parking cars which are considered as ”parking zones” or ”area where it is likely that cars are parking”. To be able to determine if a specific GPS positions is in such a parking zone (needed later on for filtering), a bounding box is created for each cluster. Such a bounding box is created by taking the GPS position of the parking cars which are parking apart from each other the greatest distance. Around these two GPS positions a rectangle (the bounding box) is created by giving a ten meter error distance to all sides. The end result of the parking space map approximation is therefore a collection of parking zones, which are implemented as bounding boxes.

Figure 4.5 shows two images of the same test region in Linz, Austria. In Figure 4.5 (a) the GPS positions of all sensed segments, which are part of the dataset, while in Figure 4.5 (b) only the GPS positions of segments which are in parking zones are shown. In both pictures green dots represent parking cars, while black dots show other segments. The semitransparent boxes which are shown in (b) show the clusters (parking zones) which were derived by the proposed algorithm. Only areas where there are lots of parking cars are extracted to be parking zones. In total 131 parking zones are derived from the whole dataset containing 97.2% of all parking cars in the dataset. The remaining 62 parking cars are classified as noise by the clustering algorithm because there are no other parking cars in the dataset nearby. A larger dataset with more test drives during different times would be needed to get a more accurate parking space map and to reduce the parking cars which are classified as noise. However, because the most parking cars are contained in the derived parking zones, the parking space map can be used without a great loss to obtain a filtered and more balanced dataset as described in the following section.

4.4.3 Using Parking Space Maps to improve Classification Results

As argued in Section 4.4, it does not make sense to classify segments which are not close to parking zones. Such segments will not make a difference to the end result as it is known that at these positions parking is not possible (or legal) and there are not valid



(a) All GPS points

(b) Detected parking zones

Figure 4.5: Approximation of parking zones from the dataset: (a) showing all GPS points at the selected region while (b) showing the derived parking zones (clusters of parked cars).

	Full Dataset	**Filtered Dataset**
Free Space	11.811 (82.6%)	6.345 (73.8%)
Parking Car	2.216 (15.5%)	2.131 (24.8%)
Overtaking Situation	207 (1.4%)	55 (0.6%)
Other Parking Vehicle	69 (0.5%)	60 (0.7%)
Total	14.303	8.591

Table 4.1: Number and percentage of the segments per class in the two used datasets.

possible parking spaces. To improve the classification results, only segments which are close to parking zones will be used for training and for evaluating the machine learning models. The full dataset is further filtered by matching the GPS positions of the segments with the bounding boxes of the parking zones. All segments which are contained in at least one bounding box of a parking zone (and which are on the right side of the road) will remain in the dataset while all others are deleted. The full dataset as well as the filtered dataset and their differences are described in more detail in the following section.

4.5 Derived Datasets

Two main datasets are being available for the use of machine learning experiments. The ”full dataset” contains all segments which have been sensed during the test drives while ”filtered dataset” has been filtered to only contain segments which are close to areas where parking spaces are located (described in the previous section). Both datasets are made up of samples with all calculated feature values (discussed in Section 4.5.1) and the corresponding class label.

Table 4.1 shows a comparison of the number of instances per class for both datasets as well as the percentage of each class. One can clearly see that both datasets are highly imbalanced because in both cases the ”free space” class is by far the dominating class with 83.6% and 73.8%, respectively. However, the filtered dataset contains only about half of the ”free space” instances which are in the full dataset and thus the filtered dataset will probably lead to a better classification result as it is more balanced. Of course, this has to be proven by the machine learning experiments which will be covered in the following sections.

Another fact which can be observed is that the classes ”overtaking situation” and ”other parking vehicle” only have a really small percentage of the whole dataset size in both cases (both classes occur in the two datasets only in less than 2% of the samples). For a good classification result it will probably be necessary to gain more data for these classes as with only such few instances the variety of the different instances will not be good enough for good classification results.

4.5.1 Feature Calculation

To conduct machine learning experiments with common machine learning models, feature values for each segment have to be computed using the sensor measurements. These feature values serve as input values of the machine learning model to help classifying the corresponding segment to one of the available class labels. All computed features which were used are described below and a further analysis of the usefulness of the features for the classification tasks will be provided in the next section.

Average distance to the sensing vehicle: The average of the segment's distance sensor measurements to the closest object on the right side of the road in meters.

Length: The distance the sensing vehicle drove during sensing the segment in meters (the distance between the first and last GPS location).

Duration: The duration in seconds the sensing vehicle needed to sense the segment (the last time stamp - the first time stamp).

Number of distance measurements: The number of distance measurements assigned to the segment.

Variance of the distance measurements: The variance of all distance measurements belonging to the segment.

Speed: The speed of the sensing vehicle while sensing the segment in m/s.

Acceleration: The acceleration of the sensing vehicle while sensing the segment in m/s^2.

Distance difference to the next/previous segment: Difference of the average distance measurements to the next/previous segment in meters.

4.5.2 Feature Analysis

The goal of each feature is to help identify the true class of a sample which should be classified. The feature analysis presented in this section should determine if all features are necessary and beneficial to this goal. Too many features maybe do not produce better results and also increase the learning time of all models. Using the tool Weka[5] two analytic approaches have been used, which show for each feature how useful it is for the classification tasks:

Information Gain Analysis: Weka provides a tool called "InfoGainAttributeEval" which calculates the information gain (also known as entropy) of all the features for the classification task. The features which will contribute more to an accurate classification result will have a higher information gain than others. The resulting values are in the range of 0 to 1.

Learner based Feature Analysis: This approach tests several feature subsets on a real machine learning algorithm (a J48 decision tree is used) and identifies the best feature subset. Cross-validation is used for the experiment with different feature subsets and the result is a list of features which works best for the classification task using the tested machine learning algorithm.

Table 4.2 shows the results of the feature analysis process for the full and filtered datasets, respectively. The information gain shows that most features of the filtered dataset have a higher score, therefore probably the classification result will lead to a higher accuracy using the filtered dataset. Looking at the information gain of both dataset, six important features can be identified, having an information gain value over 0.1 (average distance, different to previous and next segment, length, number of measures and duration). The remaining three features have a much lower information gain and therefore probably will not contribute as much to the classification result.

The learner based analysis confirms the results of the information gain analysis. All of the six most important features have been selected for the best feature subset in both datasets. It is also shown that the distance variance is not used in any case while the average speed and average acceleration are used in one case while being left out in the other.

[5] `https://www.cs.waikato.ac.nz/ml/weka/`

	Information Gain Full/Filtered Dataset	**Contained in best subset (Learner based Analysis) Full/Filtered Dataset**
Average distance	0.361 / 0.502	yes / yes
Diff to next	0.232 / 0.338	yes / yes
Diff to prev.	0.233 / 0.337	yes / yes
Length	0.223 / 0.290	yes / yes
Nr. of measures	0.178 / 0.235	yes / yes
Duration	0.175 / 0.234	yes / yes
Distance variance	0.055 / 0.067	no / no
Avg speed	0.019 / 0.022	no / yes
Avg acceleration	0.011 / 0.008	yes / no

Table 4.2: Feature analysis showing the information gain and learning based analysis results of all calculated features on the full and filtered dataset.

4.5.3 Comparison to the Datasets of existing Parking Detection Systems

The existing prototype of a parking detection system which is closest related to the prototype presented in this thesis is the ParkNet prototype developed by Mathur et al [17] (see Section 2.1.6). The dataset of all other parking detection systems, which are discussed in Section 2.1, cannot be compared to our dataset as their approaches are too different. Thus only the dataset of ParkNet is compared to our dataset in this section.

ParkNet also uses the drive-by park sensing approach. It measures the distance to the right side to the road using an ultrasonic distance sensor and tries to find ”dips” (segments) in the sensor signal, which represent parking cars. However, ParkNet was only developed, designed and trained for a very restrictive scenario of traffic. It only has been trained and evaluated on single-lane streets with parallel parking cars. The main goal of this thesis is to develop a drive-by parking detection system, which can handle various different situations in city traffic, like for example driving on multi-lane streets, overtaking other vehicles, traffic jams and waiting at traffic lights. While gathering the dataset, all of these situations were faced and our prototype has been trained to detect all of these scenarios.

ParkNet only uses two features to detect parking spaces in their approach, namely the length of segments and the distance from the sensing vehicle. Mathur et al. derived

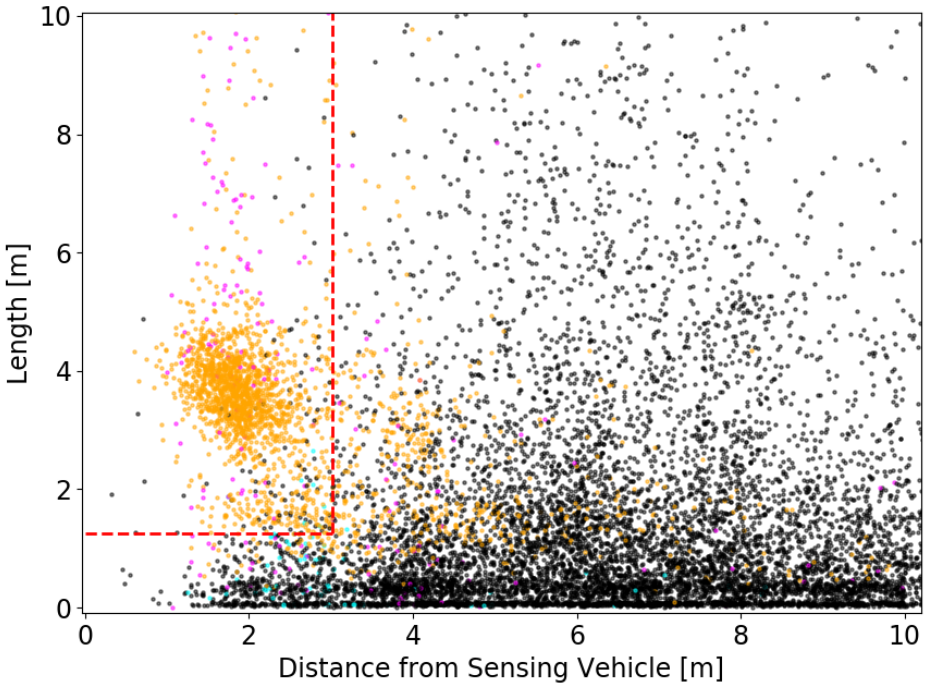


Figure 4.6: A plot of the length and distance of the segments of the full dataset. Orange points show parking cars, while black points show free spaces. The red line shows the learned decision tree boundaries.

thresholds for both features creating a minimum error calculated on 20% of their dataset. The thresholds are then used to classify the segments into the classes *parking car* and *other object*. This technique is sufficient for their simple scenario because of their restricted test data. However, on a more complicated dataset which is used in this thesis the results will probably be worse.

To support that hypothesis, a similar technique as in the ParkNet prototype was used to classify segments of our dataset using only the two features *length* and *distance to the sensing vehicle*. A decision tree with a maximum depth of two has been learned using our dataset and evaluated using 10-fold cross-validation. Similar to the two threshold used in ParkNet, our decision tree also learned to divide the two target classes using two thresholds. However, while the ParkNet prototype reaches about 87% recall on parking cars on their dataset, our approach only gets an recall of about 75%. The author believes that this can be explained because of the various traffic situations which are included in our dataset but were not in the ParkNet dataset. These more complex scenarios make it harder to classify a parking situation using only the two selected features.

Figure 4.6 shows a plot of our full dataset using both used features as well as the learned decision tree boundaries. The black dots represent free spaces, while the orange ones represent parking cars. Magenta and cyan dots are overtaking situations and other parking vehicles, respectively. The dashed red lines represent the decision boundaries which have been learned by our decision tree, which is a similar technique to the ParkNet prototype. All segments in the upper left rectangle are classified as *parking cars* while all others are classified as *free spaces*. Figure 4.6 shows that there are a lot of missed parking car detections (parking cars which are classified as other objects). About 25% of parked cars are not detected. Thus, it can be concluded that only leveraging the two features is insufficient for the dataset which has been gathered in this thesis.

4.6 Machine Learning Experiments

There are a lot of different machine learning approaches, which can be chosen from. All of them have their strengths and weaknesses and are applied in different scenarios. To find out which machine learning model works best for the parking space sensing scenario, experiments are being performed and the results are compared so that the best model can be identified. The overall goal of the experiments is to find the feature subset and the machine learning model which in combination produce the highest accuracy in classifying all segments. Additionally, the precision and recall values of the "parking car" class are of interest in our scenario, because detecting parking cars is crucial for creating accurate parking space availability maps.

4.6.1 Tested Machine Learning Models

This section discusses machine learning models which are investigated on their performance on our datasets. Furthermore, all parameters which can be set for the different machine learning models will be presented. The tested machine learning models are:

Naïve Bayes Classifier

The Naïve Bayes classifier is based on the theory of conditional probabilities. For each segment the probability of all classes are calculated, given the calculated feature values. Let $P(c|F)$ be the probability of a segment being a class c given the features $F = \{l, d\}$ (length and distance). According to Bayes' rule the probability can be

calculated as $P(c|F) = \frac{P(F|c) \times P(c)}{P(F)}$, where $P(F|c)$ is the probability of the features F given the class c, $P(c)$ is the overall probability of a segment to be the respective class and $P(F)$ is the probability of the occurrence of the features F. Using Bayes' rule, probabilities of each class can be calculated given the related features. However, calculating the different probabilities in the equation is a non-trivial task and requires a lot of computing resources (especially if there are a lot of features). Thus, Naïve Bayes introduces the assumption that all features in F are independent to each other. Using this assumption the probabilities can be calculated with a much less complex equation: $P(c|F) = \frac{P(l|c) \times P(d|c) \times P(c)}{P(F)}$. $P(l|c)$ and $P(d|c)$ are the probabilities of the *length*- and *distance*-features given the class c. Both of these probabilities can be easily computed using the dataset. The independence-assumption greatly simplifies the solution, however, it also introduces error to the result as the features in most problems cannot be treated independently. In spite of its simplicity, the Naïve Bayes classifier can compete with more sophisticated classifiers in a lot of domains [9].

kNN Classifier

A kNN classifier (k nearest neighbours) searches for the k samples in the training set which are most similar to the sample which should be classified and then predicts the class which is contained in a majority of the found k samples. To find similar samples, a similarity function has to be computed which calculates the similarity between the features of two samples. Such similarity functions are for example Euclidian distance, Manhattan distance or Chebyshev distance. For the classification of each sample, the classifier has to calculate the similarity to each sample in the training set and find the most similar k samples. For big datasets this process can take a long time and another disadvantage is that the whole training set has to be saved for the classifier to work. Parameters for the kNN classifier are the number k of nearest neighbours to search for and the similarity function [9].

Support Vector Machines (SVMs)

Support vector machines try to find a linear separation between the samples of different classes in the training set in order to predict unknown samples. A hyperplane which maximizes the separation of different classes is computed. In order to do so, the distance of the hyperplane to the closest samples, which are also called support vectors, is maximized. Figure 4.7 shows the so called maximum margin hyperplane as well as the support vectors which are needed to find it. However, in real world problems it is rarely the case that different classes can be separated using a linear

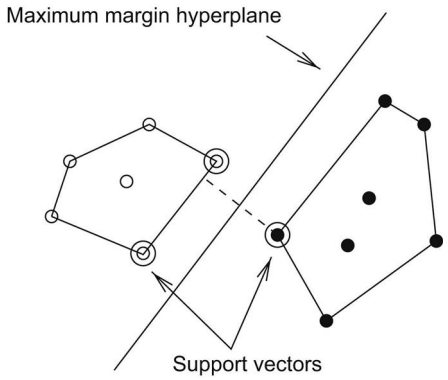


Figure 4.7: Maximum margin hyperplane of a support vector machine [9]. **To-Do:** kann ich die grafik aus dem buch verwenden oder soll ich sie nachzeichnen?

hyperplane. To be able to find such an hyperplane, the feature space is transformed into a higher dimensional one using non-linear mappings. This process increases the chance of finding a hyperplane drastically. However, such transformations need a lot of computational performance and therefore usually kernel functions, such as a polynomial kernel, are used to speed up this process [9].

Neural Networks

A Neural Network (also called MultiLayerPerceptron in some frameworks) is a popular classification technique nowadays, because theoretically neural networks can learn every function possible, given a large enough network and enough training time. Simple neural networks are composed of three layers: the input layer, one hidden layer and the output layer. Figure 4.8 shows an example of such a neural network.

The output of the neural network is calculated by a process called forward propagation. The resulting value of each unit is obtained by a non-linear formula using all the previous units' values. $f(x) = a(b + \sum_{i=0}^{n} input_i \times w_i)$ calculates the output of a hidden layer unit. It computes a sum of all input values $input_i$ multiplied by their weights w_i and adds a constant bias value b. Subsequently, this intermediate result is applied to a non-linear activation function a, such as a sigmoid or ReLU function. The same formula used to compute the result of the hidden units is also used for the output unit to calculate the final result of the neural network. The forward propagation of the neural network represents a function $f(x)$ which maps the input features x to the output value $f(x)$.

To use neural networks for classification, the function $f(x)$ has to be learned according to provided training samples. As $f(x)$ is calculated using all weights w_x and bias values

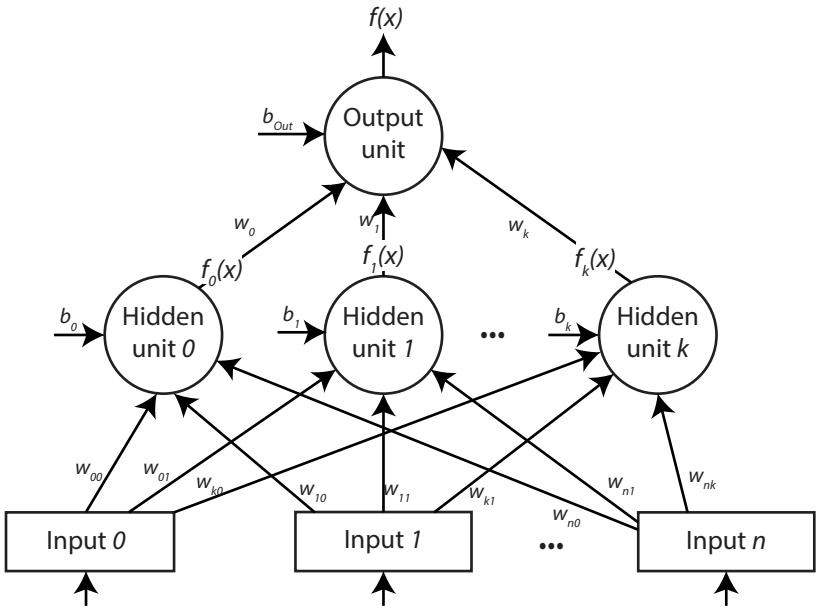


Figure 4.8: Example of a simple neural network showing the units, the weights w_x and the biases b_x.

b_x, these values have to be learned. This is done using back propagation. As first step all weights and biases are initialized randomly. Then for every training sample the result of the neural network is computed and an error to the desired output is calculated using a loss function, such as the squared-error loss function. Using gradient decent, the weights and biases are then adapted slightly to reduce the loss function. This process is repeated and finally the loss function should be minimized to obtain good classification results. The most important parameters which can be set are the number of hidden layers and hidden units, the learning rate, momentum and the activation function. Further information about neural networks and their parameters can be found in [9].

Decision Trees

A decision tree is a powerful classification approach. Figure 4.9 shows an example of a decision tree which has been learned using the full dataset acquired in this thesis with the features length and distance. Each node of the tree divides the dataset into several sub-groups using separation values of specific features (in Figure 4.9 the root node divides the dataset at the value 3.0217). This division into sub-groups goes on and at the leave nodes the class with the majority of the samples is the classification result of the respective branch of the tree. The important task to learn a decision tree is to select the feature which should be used to split and at what value the split should

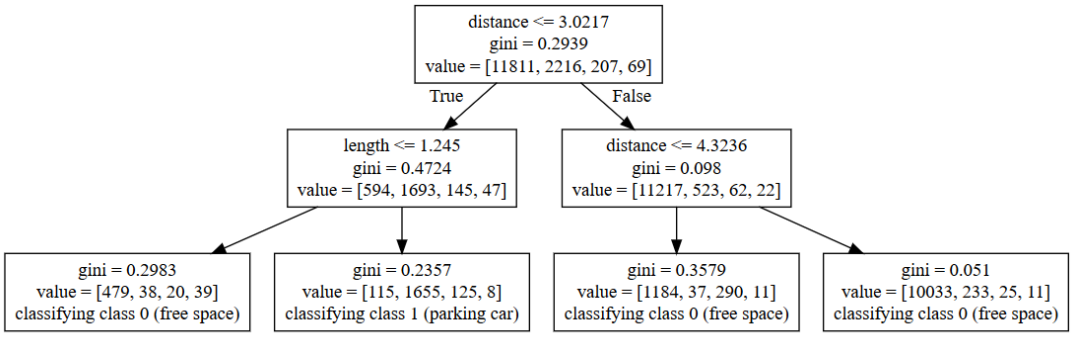


Figure 4.9: Example of a decision tree with a depth of two, which has been trained with the full dataset and with the features length and distance.

be performed. This is done at each node by trying different features and values in order to find the combination which maximizes certain measures, such as information gain (entropy) or Gini impurity[6]. To overcome the risk of an over-fitting decision tree, the splitting process is stopped if there is a big majority or a single class at a node. This process is called ”pruning”. The most important parameters are the criterion function (entropy, gini impurity, ...) and the pruning configuration. [9].

Random Forest Classifier

Decision trees can be rather unstable, meaning they can change their output drastically if their input is only altered slightly. To overcome this shortcoming, an ensemble technique, called ”random forest” is introduced. A random forest classifier is made up of multiple different trees which have been learned using random subsets of features at each node of the trees during the training process. All trees are classifying using the input features and are then voting to determine the classification result of the random forest classifier. Random forests are usually creating more accurate predictions due to using many different decision trees which in conjunction produce better results than single decision trees. The most important parameters are the number of trees which should be generated, the random seed and the maximum number of features per subset [9].

[6] Gini impurity represents how often a randomly chosen sample would be assigned into the wrong class if it was randomly labeled

4.6.2 Tested Datasets and Handling Imbalance

All discussed machine learning models are being tested on multiple different datasets. The two main datasets (full and filtered datasets) which contain all calculated features have been discussed in Section 4.5. Both of these datasets are being tested as well as datasets which only contain the most important features (distance, length, difference to next/prev segment, number of measures, duration), as discussed in Section 4.5.2. The experiments should test which dataset performs best and if the datasets with less features can compete with the datasets containing all of them.

Dealing with imbalance in the data is another important factor to handle. As discussed in Section 4.5 our datasets are highly imbalanced. The *free space*-class has in all cases more than four times as many samples as all other classes combined. That fact can create shortcomings in the performance of predicting classes with the minor number or samples in the dataset, therefore it might be beneficial to prepare the dataset to be more balanced. Imbalanced datasets are quite common in a lot of domains. For example, in medical data there are much more negative examples than positive ones when testing a patient's data on a specific disease. There exist a few techniques to handle such imbalances. Under-sampling randomly takes only a fraction of the class with the majority of the sample to create a more balanced dataset, whereas over-sampling operates the opposite way. Furthermore, there also exist combinations of both techniques. All the mentioned approaches should be tested on their impact on the classification performance on the different classes to address the imbalanced dataset.

4.6.3 Evaluation Methods

To be able to get reliable results for the performance of the different classifiers it is important to have a comparable evaluation method. The used method in this thesis is 10-fold cross validation. Using this technique, all samples of the dataset are split into 10 equally sized parts (folds). Then one fold is selected as test data while all the other folds are used as training data for the classifier. This process gets repeated 10 times, so that every fold has been used as test data exactly once and has been used for training all the other nine times. Cross validation ensures, that there is no sample in the test data which has been previously learned by the classifier in the training process. Moreover, it is important that every sample gets predicted once, so that the performance measures are realistic.

Another technique which helps to get better performance is to shuffle the dataset, before it is divided into folds for cross validation. The shuffling ensures that it is likely that only one of two consecutive segments is in the test data, while the other one is in the training data. This helps to have more diverse training data as consecutive segments usually are more alike than others. For example, usually perpendicular parking spaces are in the same area. Therefore, if the dataset gets shuffled the ratio between perpendicular parking spaces and other parking spaces will be approximately the same in the test- and training data and lead to a better classification performance.

4.6.4 Used Tools and Frameworks

To conduct the machine learning experiments the dataset is processed using the Python programming language. Furthermore, the classifiers of the "Scikit learn" machine learning framework are used for the experiments and several different configurations of all of them are tested. Scikit learn is a framework for Python which offers all of the above discussed classifiers as well as tools to support the evaluation process (cross validation, shuffling, etc.).

The implementation of re-sampling techniques, to tackle the dataset imbalance, is done with a python framework called *imbalanced-learn* [14]. The imblearn framework is compatible with the Scikit learn framework and is able to perform under-sampling, over-sampling and a combination of both. Under-sampling is done by clustering the data in the majority class to select the best segments which represent the full dataset. Over-sampling is implemented using the SMOTE (Synthetic Minority Over-sampling Technique) algorithm. As the name suggests, SMOTE creates new "synthetic" samples in the feature space rather than by repeating samples several times. The creation of such new samples is done by joining five neighbouring samples to create a new synthetic sample. Finally, imblearn provides a combination of over- and under-sampling.

4.7 Deep Learning

Deep learning gained a lot of attention in the last few years because of its rising classification accuracy results. For example, deep learning models became the first classifiers which could compete with humans on simple classification tasks. A deep learning network is a special form of a neural network with a possibly large amount of hidden layers between the input layer and the output layer. Furthermore, in many

applications raw sensor measurements are used as input instead of pre-computed feature values, which are derived from the raw sensor values. In theory, a simple neural network containing only one hidden layer can derive any function given enough hidden elements and long enough training time. However, that does not mean that such a simple neural network will find a optimal solution and that it will find it in a finite amount of time. Adding more hidden layers (to get a deep neural network) can help to find a solution more easily and with less training time. Figure 4.10 shows an example of a deep learning network with several hidden layers which are densely-connected.

Due to its classification power, deep learning should also be tested for the use in classifying parking situations using our datasets. It should be investigated if deep learning can be used to do this kind of classification on raw sensor data and the results should be compared to classical machine learning algorithms (described in Section 4.6.1) using pre-computed feature values.

As input for the deep learning experiments a dataset containing all sensor measurements for each segment is needed. In a pre-processing step a dataset for the deep learning experiments is created which is made up of a vector of sensor measurements for each segment. Each vector has a fixed size of 256 sensor measurements (which will be enough for most segments). In most cases the segments have a much lower number of measurements. In these cases the measurements will be zero padded to the required size. If a segment has more than 256 measurements, only the first 256 will be used. This step is necessary to have a fixed length input for the experiments as the input size for all segments has to be equal. A vector represents the input layer of the neural network.

4.7.1 Tested Deep Learning Models

As baseline deep learning network a dense neural network with 5 hidden layers and 64 hidden elements per layer is used. This basic deep learning network should show how a densely-connected deep learning model performs on the raw sensor dataset. Figure 4.10 shows the illustration of the created deep learning network containing of an input layer with 256 elements (one for each sensor value), 5 hidden layers containing 64 hidden elements per layer and four output elements which calculate the probability of each class for the given input. The class which gets predicted is the one with the greatest probability out of the four classes.

In addition to the basic deep neural network, several known strategies should be tested to improve the performance of the deep learning approach. First of all, dropout

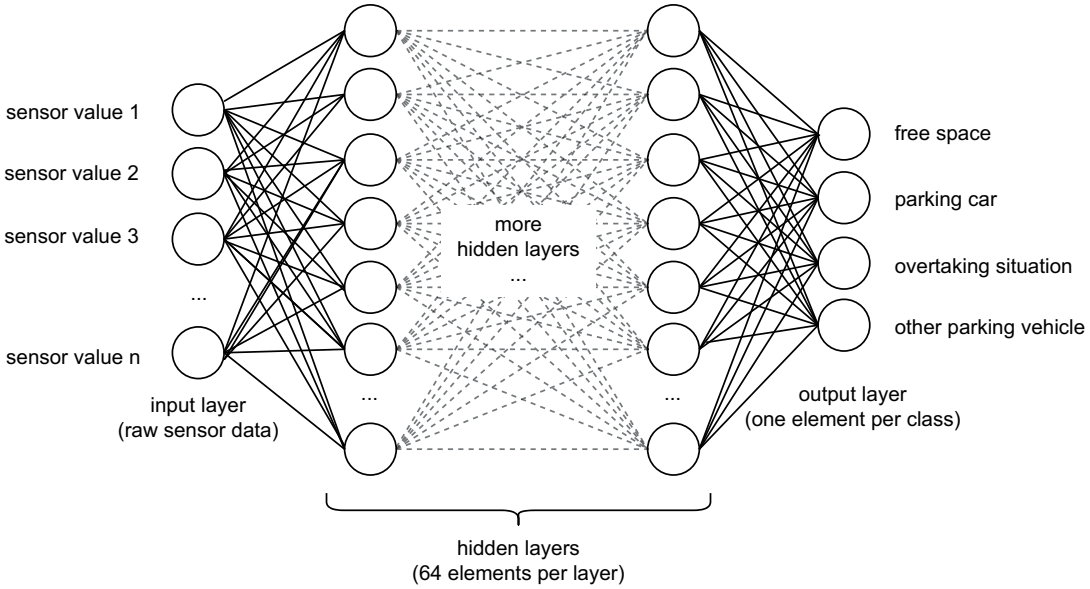


Figure 4.10: A basic deep learning network having sensor values as input and the probability of all classes as output values. All layers are densely connected with each other.

between the layers should be tested. Dropout is a technique in deep learning to handle over-fitting of the neural network. If dropout with a prability p is introduced between two layers, a connection between two elements is maintained only at a probability of $1 - p$. The result is a more sparsely connected deep neural network which should be less prone to over-fitting. Dropout should be tested between several layers and with different values for the deletion-probability p.

Another popular technique to improve deep learning models is the use of convolution. Convolutional neural networks are especially popular for image classification. However, convolution can also be applied on the one-dimensional distance sensor data available in our dataset. Different configurations for the configuration of such a convolutional neural network should be tested and the performance should be compared to the other approaches.

4.7.2 Tools and Frameworks used for developing Deep Neural Networks

To implement the deep learning models, the programming language Python[7] (version 3.6.4) is used as well as the Keras[8]- and the TensorFlow[9] frameworks. Keras is used

[7] `https://www.python.org/`
[8] `https://keras.io/`
[9] `https://www.tensorflow.org/`

as a wrapper which is using the TensorFlow framework internally and helps to create easy to read representations of deep neural networks. TensorFlow is built for creating neural network models and especially for building deep learning models using a lot of pre-built components.

Listing 4.1 shows an example of a deep learning network which has been defined using the Keras framework. It defines a neural network with five hidden layers with each having 64 hidden elements using the ReLU activation function. Moreover, between every hidden layer there is a 20% dropout of the connections between the hidden elements. To get a result, the output layer which consists of four elements is using the softmax activation function which calculates the probabilities for each class, therefore the output of the deep learning network is a vector with 4 numbers in the range of 0 to 1 adding up to 1. The listing furthermore shows the loss function (categorical crossentropy) as well as the optimizer (adam) and the used metric (accuracy).

Listing 4.1: Example of a densly-connected deep learning network containing 5 hidden layers with dropout defined with Keras

```
def simple_dense_model(dataset, x_train, y_train):
    hidden_dims = 64

    model = Sequential()
    model.add(Dense(hidden_dims, activation='relu',
                    input_dim=len(dataset.x[0])))
    model.add(Dropout(0.2))
    model.add(Dense(hidden_dims, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(hidden_dims, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(hidden_dims, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(hidden_dims, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(len(dataset.class_labels),
                    activation='softmax'))

    model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])

    model.fit(x_train, y_train, epochs=200)
    return model
```

4.8 Improving Classification Results using the Segments' Surroundings

So far the machine learning- and deep learning models are only classifying using the segment's properties, but its surroundings are ignored. The used features (described in Section 4.5.1) are all calculated using only the segment's sensor data. However, the surrounding segments' properties also include information about the class to which the current segment belongs to. For example, one fact which could help in the classification task is that if the segment under investigation has a similar distance to the sensing car as surrounding segments which are labelled as *parked car*, then it is likely that the segment under investigation is also a *parked car*.

Figure 4.11 shows a scene where segments from the same class are all approximately at same distance to the sensing vehicle. Segments of the *parking car*-class are all about 2 meters from the sensing vehicles whereas *free spaces* are about 6 meters or more away from the sensing vehicle. An investigation on the full dataset confirms this observation. The standard deviation of the distances of all groups of 5 subsequent parking cars has been calculated on the whole dataset. The result was a value of 0.8234 meters. It shows that subsequent parking cars are usually at the same distance from the sensing vehicle. As one would expect, the more surrounding parking cars are taken, the higher is the value for the standard deviation. For groups of 11 subsequent parking cars, it is 0.9821 meters.

To investigate if such information can in fact benefit the classification results, a custom classification model is implemented. First of all, a basic classification task is performed by one of the classifiers mentioned in Section 4.6.1. Then, the dataset is being enhanced by the features and predicted class labels of the surrounding segments. The number of surrounding segments used is tested in several experiments to find the best amount. In the final step, another classification task is performed using the enhanced dataset. This process should help to include information of surrounding segments and it should improve the classification results. The results of the approach can be viewed in Section **To-Do:** todo.

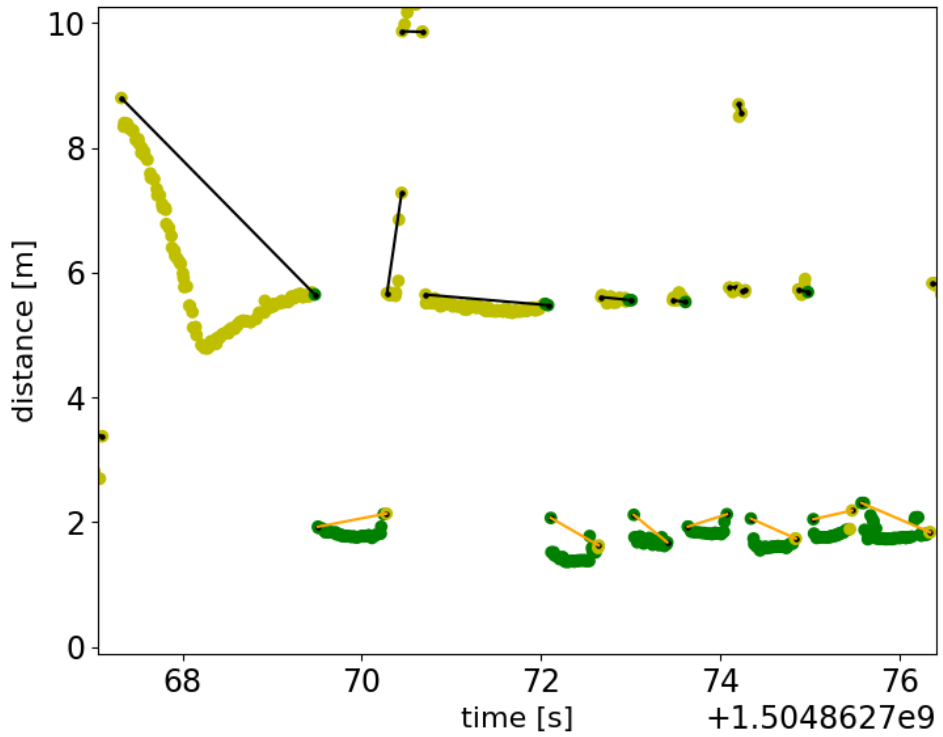


Figure 4.11: Samples from the same class are approximately the same distance away from the sensing vehicle. Black lines (yellow dots) represent free spaces, whereas orange lines (green dots) represent parking cars.

Chapter 5

Results and Discussion

This chapter summarizes the results of the machine learning experiments and compares different machine learning models and datasets on their performance. First of all, Section 5.1 discusses the measures which should be optimized to derive most accurate parking space availability estimates. Furthermore, Section 5.2 shows the results of classical machine learning model, while Section 5.3 will discuss the results of the deep learning neural networks operating on raw sensor data. Finally, Section 5.4 describes how the surroundings of the samples have been used to further improve the machine learning results.

5.1 Optimization Goal

This section should outline the processes and decisions which have been taken to determine the best performing system which would produce the most accurate parking space availability estimates. Of course, the first interesting measure when investigating the results of machine learning experiments is the accuracy. However, that measure is not always the most important one, when it comes to solving a specific problem.

In the parking space detection scenario, which is tackled in this thesis, the overall goal is to derive accurate parking space availability estimates using drive-by park sensing. It is assumed that detailed parking space maps are available, thus it is sufficient to know where cars are parking to also be able to derive vacant spots (detecting a *free space*-class is not enough as it does not determine if it is allowed to park at this spot). Therefore, the most important measures, which should be optimized, are precision and recall of the *parking car*-class. As secondary goals, precision- and recall-measures of the classes *overtaking situation* and *other parking vehicle* should be optimized. *Overtaking situations* are important to detect distractions which prevent

	Accuracy (full / filtered)	**Recall Parking Car** (full / filtered)	**Precision Parking Car** (full / filtered)
Random Forest (entropy, 1000 trees)	0.9600 / 0.9608	0.8799 / 0.9239	0.9129 / 0.9377
Decision Tree (gini impurity)	0.9486 / 0.9492	0.8492 / 0.8987	0.8753 / 0.9176
kNN classifier (5 nearest neighbors)	0.9459 / 0.9457	0.8641 / 0.9175	0.8618 / 0.8941
Neural Network (5 layers - 50 units each, 1 million epochs)	0.9440 / 0.9415	0.8479 / 0.9005	0.8529 / 0.8891
Support Vector Machine (kernel: radial basis function)	0.9423 / 0.9359	0.7951 / 0.8273	0.9259 / 0.9450
Naive Bayes	0.4957 / 0.5886	0.8763 / 0.8822	0.4622 / 0.5915

Table 5.1: Overall accuracy and results for the *parking car*-class of the best configuration (best set of parameters) of all tested classic machine learning models applied on the full and filtered dataset.

deriving accurate parking space availability estimates whereas *other parking vehicles*, such as parking motorcycles, should be detected because they also show that a specific parking spot is occupied. However, these classes have only a minor occurrence in the dataset. This shows that such situations occur relatively rare and therefore are not as important.

To be able to compare two classification results containing recall- and precision measurements, the $f - measure$ (or also called $F_1 - score$) has been chosen as comparison measurement. Because it is the harmonic mean of both recall and precision it is well suited for the task. The formula to calculate the $f - measure$ is: $F_1 = 2 \times \frac{precision \times recall}{precision + recall}$.

5.2 Machine Learning Results

This section discusses the results of the machine learning experiments with common machine learning models using computed feature values as input. As evaluation-method 10-fold cross validation has been chosen (see Section 4.6.3). Table 5.1 shows the classification results of the best configurations of all tested machine learning models on the full and filtered dataset sorted on their performance on the f-measure of the *parking car*-class. For each machine learning model, many different parameter combi-

Predicted class → / True class ↓	**Free Space**	**Parking Car**	**Overtaking Situation**	**Other Parking Vehicle**
Free Space	11718	90	0	1
Parking Car	244	1950	21	1
Overtaking Situation	63	82	62	0
Other Parking Vehicle	54	14	0	1

(a) full dataset

Predicted class → / True class ↓	**Free Space**	**Parking Car**	**Overtaking Situation**	**Other Parking Vehicle**
Free Space	6598	81	0	1
Parking Car	164	2017	2	0
Overtaking Situation	14	35	17	0
Other Parking Vehicle	37	18	0	5

(b) filtered dataset

Table 5.2: Resulting confusion matrices of the random forest classifier (containing 1000 trees and using entropy as criterion for node-splitting) applied on (a) the full dataset and (b) the filtered dataset.

nations have been tested to find the best performing parameter set of each model. The best performing classifier on the *parking car*-class is a random forest classifier using 1000 trees and entropy as criterion for node-splitting at the individual decision trees. It leads to an accuracy of about 96% and a f-measure value of 0.8961 and 0.9307 for the *parking car*-class on the full and filtered dataset, respectively.

Table 5.2 shows the confusion matrices of the random forest classifier applied on both datasets. The results clearly indicate that all tested models perform better on the parking space map filtered dataset than on the full dataset with all sensed segments. This can be explained due to a more balanced dataset, as a lot of free space segments are filtered out because they are not close to parking zones and thus not relevant.

Another conclusion which can be drawn is that the performance is much worse for the classes *overtaking situation* and *other parking vehicle* than for the *free space*- or *parking car*-class. The best random forest classifier reaches a recall of 0.2575 and a precision of 0.8947 on *overtaking situations*. Therefore, only about a quarter of the overtaking situations are detected. On the other hand, the high precision shows that only a low number of false positives are detected as overtaking situations. The *other parking vehicle*-class has similar results: a really low recall of 0.0833 and a relatively

high precision of 0.8333. The bad results on the two classes are probably explainable because of the low number of segments of both classes included in the datasets. Thus, the variety of the data is low and a lot of different situations are not included in the dataset. Therefore, the classifiers cannot learn all different variations of overtaking situations or other parking vehicles. Another explanation is the imbalance in both datasets.

5.2.1 Results for different Types of Parking Cars

As mentioned in earlier sections, the datasets consists of three different types of parking cars: Parallel parking cars, perpendicular parking cars and angular parking cars. All of these different types are included in the broader *parking car*-class which gets predicted by the classifiers. In this section the performance of the classification on the different types of cars is being investigated.

Overall the filtered dataset contains 1647 parallel parking cars, 339 perpendicular parking cars and 197 angular parking cars. This shows that most of the parking cars in the dataset are parallel parking cars, which can be explained because of a majority of parallel road side parking spaces in Linz, Austria. The best classifier achieves a recall of 0.9581 on parallel parking cars, 0.7964 on perpendicular parking cars and 0.8578 on angular parking cars. Obviously, the performance of parallel parking cars is the best. That is probably caused because parallel parking cars are in comparison longer than the other types. This fact can make it easier to detect them Furthermore, the majority of parallel parking cars also enhances the performance of that type in comparison the the other types of parking cars. Perpendicular parking cars are detected less often because they are shorter in length which may lead to a more likely classification as *free space*. In general, parking cars which are classified falsely are almost exclusively assigned to the *free space*-class. Only two parallel parking cars have been detected as *overtaking situations* while no parking cars have been detected as *other parking vehicles*.

5.2.2 Impact of Sampling Techniques in Order to Handle the Dataset Imbalance

As the datasets are highly imbalanced, sampling techniques have been tested on their effect to improve the overall accuracy of the classifier and to improve the precision and recall for classes with a low number of samples. Table 5.3 shows a comparison

	No sampling	**Over-sampling**	**Under-sampling**	**Combination Technique**
Overall accuracy	0.9608	0.9479	0.6101	0.9349
Recall Free Space	0.9877	0.9664	0.5752	0.9456
Precision Free Space	0.9684	0.9772	0.9738	0.9815
Recall Parking Car	0.9239	0.9225	0.7091	0.9257
Precision Parking Car	0.9377	0.9117	0.8336	0.8813
Recall Overtaking Sit.	0.2575	0.3484	0.6060	0.4393
Precision Overtaking Sit.	0.8947	0.3066	0.0417	0.2989
Recall Other p. vehicles	0.0833	0.4666	0.9000	0.6166
Precision Other p. vehicles	0.8333	0.2828	0.0242	0.2269

Table 5.3: Comparison of different sampling techniques and their effect on the performance of a random forest classifier on the filtered dataset.

of different sampling techniques and their impact on different performance measures using the best classifier shown in the previous section (random forest). Experiments with other classifiers have been conducted as well but yielded in worse results on the overall accuracy.

All sampling techniques alter the training data by adding or deleting samples so that it is more balanced. The test data is not altered to get comparable results to the experiments where no sampling has been performed. Over-sampling creates new synthetic samples from the existing ones, while under-sampling uses clustering techniques to find representatives for a lot of different samples. The combination technique uses both techniques in conjunction to find a compromise sampling solution. For a more detailed description of the different sampling techniques see Section 4.6.2. The goal of all these sampling techniques is to create a dataset where all classes have an equal number of samples.

The results in Table 5.3 show that all sampling techniques slightly decrease the performance of the classifier as the overall accuracy is the highest with no sampling at all. Another fact which is shown is that the sampling techniques improve the recall of the classes with a minority of samples (*overtaking situations* and *other parking vehicles*) while at the same time lowering the precision of these classes. This leads to the conclusion that the different sampling techniques make it more likely for the classifier to predict a class which has been a minority class, but using the sampling techniques has an equal amount of samples in the new training data.

When comparing the different sampling techniques, under-sampling clearly performs worst. This can be explained because the under-sampling algorithm deletes most samples of the majority classes in the dataset to create a balanced new dataset. The under-sampled dataset has a size of only 240 samples in total whereas the filtered dataset has a size of 8989 samples. This process removes so much information when deleting these samples that the overall performance decreases drastically. Over-sampling and the combination technique show similar results in terms of accuracy.

To conclude the results of the tests, all sampling techniques decrease the overall accuracy as well as the f-measure of the majority classes (*free spaces* and *parking cars*). Furthermore, the recall of the minority classes is increased while the precision is decreased. Thus, sampling techniques do help to detect more samples of the minority class while detecting less samples of the majority class. Because the primary goal of this thesis is to find a classifier with high accuracy on the *parking car*-class (see Section 5.1), performing over- or under-sampling is counterproductive to the this goal and therefore is not considered to help in the creation of accurate parking space availability estimates.

5.3 Deep Learning Results

As deep learning outperforms classical machine learning on many different tasks, it has been evaluated and compared to classical machine learning approaches on the drive-by park sensing task, which is tackled in this thesis. Several different deep learning networks have been created with different amounts of layers and units. All of these neural networks have the raw and unprocessed sensor data as input. Table 5.4 shows the results of the different configurations of deep learning networks as well as the best performing classifier from Section 5.2 to be able to easily compare the deep learning result to the classical machine learning results obtained during this thesis.

The results shown in Table 5.4 clearly indicate that all deep learning approaches perform worse than the random forest classifier, which was the best performing model out of the classical machine learning classifiers. The best performing deep neural network (5 hidden layers, each containing 128 units, trained in 200 epochs) gains an overall accuracy of about 93.6% whereas the random forest classifier gets about 96.0%. Furthermore, the recall / precision on the *parking car*-class also are also worse than for the random forest classifier (0.8987 / 0.9078 for the best performing deep neural network and 0.9239 / 0.9377 for the random forest classifier). For the classes with a minority of samples, the results are similar to the random forest classifier. *Overtaking*

	Accuracy (full / filtered)	**Recall Parking Car** (full / filtered)	**Precision Parking Car** (full / filtered)
Random Forest (entropy, 1000 trees)	0.9600 / 0.9608	0.8799 / 0.9239	0.9129 / 0.9377
Dense Deep Network (5 layers - 128 units each 200 epochs)	0.9361 / 0.9365	0.7946 / 0.8987	0.8524 / 0.9078
Dense Deep Network (5 layers - 64 units each, 500 epochs, 20% dropout)	0.9379 / 0.9313	0.8434 / 0.8836	0.8147 / 0.8634
1D Convolutional Deep Network (1 convolutional layer, 2 dense layers, 200 epochs, 20% dropout)	0.9296 / 0.9311	0.7563 / 0.8027	0.8248 / 0.8488

Table 5.4: Overall accuracy and results for the *parking car*-class of different configurations of deep learning models applied on the full and filtered dataset compared to the best classical machine learning model (random forest).

situations show a recall of 0.2272 and a precision of 0.4687 whereas *other parking vehicles* get a recall of 0.1833 and a precision of 0.3793.

Deep learning gains respectable results when considering that all tests have been performed using raw sensor data. However, all of the tested configurations do not nearly perform as well as the classical machine learning approaches. Therefore, it can be concluded that for the drive-by parking detection task, classical machine learning models using pre-computed feature values are better suited than deep neural networks operating on raw sensor data.

5.4 Using the Segment's Surroundings to improve the Classification Results

TODO

Chapter 6

Conclusions and Future Work

Bibliography

[1] G. Amato, F. Carrara, F. Falchi, C. Gennaro, and C. Vairo. Car parking occupancy detection using smart camera networks and deep learning. In *2016 IEEE Symposium on Computers and Communication (ISCC)*, pages 1212–1217, June 2016.

[2] Katy Blumer, Hala R. Halaseh, Mian Umair Ahsan, Haiwei Dong, and Nikolaos Mavridis. *Cost-Effective Single-Camera Multi-Car Parking Monitoring and Vacancy Detection towards Real-World Parking Statistics and Real-Time Reporting*, pages 506–515. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[3] F. Bock, D. Eggert, and M. Sester. On-street parking statistics using lidar mobile mapping. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 2812–2818, Sept 2015.

[4] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, Nov 1986.

[5] V. Coric and M. Gruteser. Crowdsensing maps of on-street parking spaces. In *2013 IEEE International Conference on Distributed Computing in Sensor Systems*, pages 115–122, May 2013.

[6] Daniele Di Mauro, Sebastiano Battiato, Giuseppe Patanè, Marco Leotta, Daniele Maio, and Giovanni M. Farinella. *Learning Approaches for Parking Lots Classification*, pages 410–418. Springer International Publishing, Cham, 2016.

[7] R. C. Gonzales and R. E. Woods. *Digital Image Processing*. 2002.

[8] Giulio Grassi, Kyle Jamieson, Paramvir Bahl, and Giovanni Pau. Parkmaster: An in-vehicle, edge-based video analytics service for detecting open parking spaces in urban environments. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, SEC '17, pages 16:1–16:14, New York, NY, USA, 2017. ACM.

[9] Mark A. Hall, Ian H. Witten, Eibe Frank, and Christopher J. Pal. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 4th edition, 2016.

[10] Han Hao, Yong Geng, and Joseph Sarkis. Carbon footprint of global passenger cars: Scenarios through 2050. *Energy*, 101(Supplement C):121 – 131, 2016.

[11] Texas Transportation Institute. Urban mobility report. 2007.

[12] Ho Gi Jung, Dong Suk Kim, Pal Joo Yoon, and Jaihie Kim. Parking slot markings recognition for automatic parking assist system. In *2006 IEEE Intelligent Vehicles Symposium*, pages 106–113, 2006.

[13] N. Kaempchen, U. Franke, and R. Ott. Stereo vision based pose estimation of parking lots using 3d vehicle models. In *Intelligent Vehicle Symposium, 2002. IEEE*, volume 2, pages 459–464 vol.2, June 2002.

[14] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017.

[15] J. Liu, M. Mohandes, and M. Deriche. A multi-classifier image based vacant parking detection system. In *2013 IEEE 20th International Conference on Electronics, Circuits, and Systems (ICECS)*, pages 933–936, Dec 2013.

[16] Shuo Ma, Ouri Wolfson, and Bo Xu. Updetector: Sensing parking/unparking activities using smartphones. In *Proceedings of the 7th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, IWCTS '14, pages 76–85, New York, NY, USA, 2014. ACM.

[17] Suhas Mathur, Tong Jin, Nikhil Kasturirangan, Janani Chandrasekaran, Wenzhi Xue, Marco Gruteser, and Wade Trappe. Parknet: Drive-by sensing of road-side parking statistics. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 123–136, New York, NY, USA, 2010. ACM.

[18] Anandatirtha Nandugudi, Taeyeon Ki, Carl Nuessle, and Geoffrey Challen. Pocketparker: Pocketsourcing parking lot availability. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '14, pages 963–973, New York, NY, USA, 2014. ACM.

[19] Sarfraz Nawaz, Christos Efstratiou, and Cecilia Mascolo. Parksense: A smartphone based sensing system for on-street parking. In *Proceedings of the 19th Annual International Conference on Mobile Computing & Networking*, MobiCom '13, pages 75–86, New York, NY, USA, 2013. ACM.

[20] Vehicle Sense. http://www.vehiclesense.com (accessed on november 9th). 2017.

[21] SFMTA. http://sfpark.org (accessed on november 9th). 2017.

[22] Jae Kyu Suhr, Ho Gi Jung, Kwanghyuk Bae, and Jaihie Kim. Automatic free parking space detection by using motion stereo-based 3d reconstruction. *Machine Vision and Applications*, 21(2):163–176, Feb 2010.

[23] Jin Xu, Guang Chen, and Ming Xie. Vision-guided automatic parking for smart car. In *Proceedings of the IEEE Intelligent Vehicles Symposium 2000 (Cat. No.00TH8511)*, pages 725–730, 2000.

[24] N. R. N. Zadeh and J. C. D. Cruz. Smart urban parking detection system. In *2016 6th IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, pages 370–373, Nov 2016.