



Transport Mode Detection - Preprocessing and Segmentation Analysis

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Brigitte Withalm

Registration Number 01126733

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer

Second advisor: Stefan Ohrhallinger, PhD

Vienna, 17th August, 2023

Brigitte Withalm

Michael Wimmer

Erklärung zur Verfassung der Arbeit

Brigitte Withalm

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 17. August 2023

Brigitte Withalm

Acknowledgements

I would like to take this opportunity to thank all the people who supported me during the work on my bachelor's thesis. First and foremost, I want to thank my advisor Mr. Stefan Ohrhallinger, for his professional expertise, valuable advice, and patience. I am also grateful to my colleague Mr. Jakob Lang for his great cooperation. I would also like to thank the technicians of the Institute of Visual Computing and Human-Centered Technology, especially Mr. Stephan Bösch-Plepelits and Mrs. Simone Risslegger for their serverside support. I really appreciate the participation of the volunteers who supported this work: thank you all! A special thank you of course goes out to my family for believing in me and giving me mental support.

Kurzfassung

Jede Bewegung, die ein Mensch macht, bietet durch den Zugriff auf verschiedene Sensordaten von Smartphone-Geräten eine Vielzahl von Informationen über die Art des gewählten Fortbewegungsmittels. Das Reiseverhalten beeinflusst viele Bereiche wie Stadtplanung und Umwelt.

Ziel dieses Projektes ist es, genauere Ergebnisse bei der, von einer Künstlichen Intelligenz (KI) unterstützten, Reisemodus-Erkennung zu erhalten. Dazu werden Methoden verwandter Studien in der Vorverarbeitungsphase kombiniert. Die analysierten Forschungsartikel basieren entweder auf dem Global Positioning System (GPS) oder verschiedenen anderen Sensordaten wie Beschleunigungsmessern oder Gyroskopen und verwenden Datensegmente mit festgelegter Länge oder Änderungspunkt-basierte Segmentierung. Diese Arbeit profitiert von der gemeinsamen Verwendung verschiedener Input-Typen, d. h. Sensordaten, GPS und Indikatoren, die aus einem eingebundenen Geo-Informationen-System (GIS) geliefert werden, und der Implementierung einer Kombination verschiedener Segmentierungsansätze.

Um eine automatisierte Information über die verwendeten Reisemodi zu erhalten, müssen der KI im ersten Schritt gekennzeichnete Trainingsdaten bereitgestellt werden. Dazu wurden Mobilitätsdaten von 19 Nutzern über einen Zeitraum von etwa zwei Monaten von einer Android-Anwendung (TMD-App) aufgezeichnet. Nach dem Hochladen der Dateien vom Gerät auf einen Server werden die geografischen Daten mittels Analyse der Reisegeschwindigkeit und verwandter Parameter wie Beschleunigung, Zeit und Distanz zum Trennen der Wege in „Gehen“- und „Nicht-Gehen“ Segmente verwendet. Diese werden anschließend in Abschnitte fester Länge, einschließlich Sensordaten sowie Navigationsdaten, aufgeteilt. Im nächsten Schritt werden diese Segmente mit einem neuronalen Netz (nicht Bestandteil dieser Arbeit) verarbeitet, das dann die genaue Transportart bestimmen soll. In dieser Arbeit unterscheiden wir zwischen sieben Arten der Fortbewegung, nämlich Gehen, Radfahren, Autofahren, Bus, U-Bahn, Zug und Straßenbahn.

Das in dieser Studie verwendete Gefaltete Neuronale Netzwerk (CNN) benötigt eine Datensequenz mit einer definierten Länge von einer Minute. Daher werden die gespeicherten Tracks in nicht überlappende Samples von 60 Sekunden Länge zerteilt, also 6000 Datenzeilen, die mit einer Frequenz von 100 Hertz aufgezeichnet wurden. Um jedoch falsche Zuordnungen innerhalb der Ein-Minuten-Zeitfenster zu den entsprechenden Labels zu

reduzieren, wird ein Vorsegmentierungsschritt durch Änderungspunkterkennung angewendet. Da Geh-Phasen einen Wechsel der Fortbewegungsarten nahelegen, wurde Gehen als Wechselpunkt in dieser Arbeit ausgewählt.

Des Weiteren werden verschiedene weitere nützliche Indikatoren wie Entfernung, Dauer, Geschwindigkeit und Höhenangaben auf Grundlage der Daten berechnet. Im Gegenzug liefert der Server den CO₂-Fußabdruck der Mobilität an die TMD-App.

Abstract

Every movement people make offers a variety of information about the type of chosen transportation mode by accessing different sensor data of smartphone devices. Travel behavior strongly influences many fields such as city planning and travel impacts.

This project aims to get more precise results in travel mode detection by combining methods in the pre-processing stage of related surveys. The reviewed research is either based on the global positioning system (GPS) or various other sensor data such as accelerometers or gyroscopes and uses fixed length - or changepoint-based segmentation. This study benefits from using various input types i.e. sensor data, GPS, and Geographic-Information-System (GIS) together, and implementing a compound of different segmentation approaches.

To get the information on which travel mode is used automatically, a set of labeled training data has to be provided in the first step. Therefore mobility data of 19 users over a period of about three months was tracked by an Android Application (TMD-App). After uploading the files from the device to a server, the geographical data is used for splitting tracks into "walk-" and "non-walk" segments by Analysis of traveling speed and accompanying parameters, such as acceleration, time, and distance. These newly determined segments are divided into fixed-length parts including sensor data as well as navigation data. In the next step, these segments are processed with a neural network (not part of this paper) which should then determine the exact mode of transportation. In this work, we differ between seven types of locomotion i.e. walking, biking, car driving, bus, subway, train, and tram.

Since a data sequence with a defined length of one minute is a precondition in this study for the downstream convolutional neural network (CNN), the stored tracks are cut into non-overlapping samples of 60-seconds length, i.e. 6000 data lines recorded with a frequency of 100 Hz. However, to reduce incorrect allocations within the one-minute time windows to the corresponding labels, a pre-segmentation step by changepoint detection is applied. As there is evidence that walking phases suggest a transition in modes of transportation, walking was selected as a changepoint in this paper.

Besides that various useful indicators and features such as distance, duration, speed, and elevation are calculated based on the data coming from the Android App and stored

in a SQL database on the server. Further conclusions such as the carbon footprint of mobility, are provided from the server to the TMD-App as well.

Contents

Kurzfassung	vii
Abstract	ix
Contents	xi
Glossary	xiii
1 Introduction	1
2 Related work	5
3 Methodology	7
3.1 Architecture	7
3.2 Client	9
3.3 Server	14
4 Evaluation	31
4.1 Settings	31
4.2 Evaluation Approach	31
5 Conclusions and future work	37
List of Figures	39
Listings	41
Bibliography	43

Glossary

Certain Segment

A Walking- or non-Walking segment is a Certain Segment if its distance lies above a lower limit (default: 200 m). Otherwise, it is considered to be an Uncertain Segment.

Changepoint

The position when switching from a Walk- to a non-Walk segment or vice versa.

Fixed-size Segment

A single mode segment with a defined duration of n seconds (default: 60 s).

GPS coordinates

Latitude and longitude, which are acquired from a global positioning system.

GPS log

A sequence of GPS points in chronological order.

GPS point

Every hundredth data line of the 100 Hz recorded track including latitude, longitude, and timestamp.

Non-Walk Segment

A segment with the mode of transportation biking, car driving, bus, subway, train, or tram.

Rest period

Part of a track, in which no traveling is assumed with an upper threshold for the speed (default: 2 km/h) within a lower threshold for the duration (default: 20 min).

Segment

A Data set of a trip that includes only one type of transport mode.

Track

The recorded path from the TMD App, the entire data set included. A track may contain more than one trip, different modes of transportation as well as lengthy stops.

Trip

Part of a track without rest periods. In the event of interrupted travel, a track is separated into trips, excluding the data points, which form together the enduring break. Like a track, one trip can have several transportation modes.

Trivial Segment

A Walking- or non-Walking segment is trivial if its distance is below a threshold (default: 20 meters) or the period between its start- and end time lies below an upper limit (default: 10 seconds).

Walk Segment

A segment traveled on foot.

Introduction

How behavior in traveling changes over time and what the trends regarding transportation modes used by humans are, can provide the foundation for future mobility policies as well as urban- and traffic planning. Mobility is a key factor in smart city planning, which aims to accomplish the best quality of life by conserving nature's resources [Wie14]. Not only sustainability [BAT16] but also prosperity can be ensured by traffic- and transport management [NDC⁺14].

The basic categories of modes of transportation are road, railroad, air, and waterways. Since the preferred way of getting around in cities is walking, riding a bicycle, or car-driving, and considering that public transport of course also holds an important share, especially in big cities, this paper focuses mainly on the road and public transport.

To get insight into people's travel behavior, the kind of transportation modes people choose, to get from a starting point to their desired destination must be identified. For this purpose, cities all over the world frequently carry out surveys. Residents often still get invited to provide information about their travel habits by telephone and/or (online) surveys, which use conventional questionnaires [Aus20]. The participants are asked to describe which mode of transportation they used on a certain day for their trips. The obvious drawbacks are among others enormous administrative management and limited coverage. - If only one day is taken into consideration, other influencing factors such as weather conditions or variations in selecting the means of transport in the course of a week are ignored. The results of these studies provide the most characteristic fact in the transport policy, which is represented by the modal split (see Figure 1.1 for an example).

As in other disciplines, reliable and complete data based on frequent and thorough acquisition is essential to understand the needs of people. Modern technology offers the possibility of gathering such data collections conveniently and much more effectively than traditional procedures. Ubiquitous mobile devices all contain motion sensors required for

Figure 1.1: Modal Split 2022 in Vienna



Source: Wiener Linien

obtaining useful information about the user's surroundings. Considering multiple sensor readings including an accelerometer (the rate of change in velocity), gyroscope (sense of angular rotational velocity and acceleration), magnetometer (measures the intensity of Earth's magnetic field), etc., we can observe movement as a reflection of the environment. A smartphone application is used to record such a wide range of information, which then can be used for further processing.

It is in the nature of things that each transportation mode has its typical properties such as speed or acceleration, which can disclose if a person is more likely walking or driving a car for instance. However, some activities may differ less from each other than others. To allow a distinction between closely related categories, especially in motorized transportation modes, hints from a Geographic-Information-System (GIS) for public transportation can be used as well. By distinguishing public transportation routes from a GIS and the local position of a user, which is recorded by a built-in Global Positioning System (GPS) sensor of the smartphone's device, conclusions about the transportation mode can be drawn.

To exploit the many data collected, and to identify the exact modes of traveling correctly, methods of human activity recognition (HAR), of which transport mode detection is a sub-field, are applied. The common process of HAR will follow the steps of (1) pre-processing of the raw data, (2) segmentation, (3) feature extraction, (4) dimensionality reduction, and (5) classification, the core machine learning, and reasoning [RMM16]. This work has to provide appropriate input data for a Convolutional Neural Network (CNN) scheme for deep learning, which is again a subset of machine learning. A CNN

is capable of handling very large datasets as well as many different input parameters. Furthermore, features from the raw signal input data can be generated automatically by this method. Because segments in a fixed size (e.g. segments that each hold 60 seconds of a track) are required by a CNN, it is possible that within such a segment a change of transportation mode can take place. To reduce errors caused in the classification thereby, often overlapping of neighboring segments is applied in the reviewed literature. This study tries to solve the problem of redundant data caused by this design by introducing a pre-segmentation step using changepoints.

In this project, a smartphone application TMD-App, which collects sensor and location data, was extended to upload these collected data to a server. The server pre-processes the raw data, stores it in the database, and prepares structured data that includes various indicators and features of movements. By combining different methods described in related works, this project is trying to find an improved way for data preparation to provide a better basis for deep learning algorithms. For the distribution of the TMD-App and to give users information about usage and project background, a small responsive, and accessible website was built. Also, features that could give some incentives for users to regularly interact with the application were added. For example, a feature that shows the CO₂ footprint regarding the mobility of the user. Privacy is another important aspect of working with a large amount of data. To ensure data privacy, various additional functionalities had to be implemented in the app as well as in the server component.

This bachelor's thesis is structured into five chapters: in the beginning, related work focusing on different approaches of segmentation gets analyzed. The next chapter introduces the description of the project's implementation. The results are covered in chapter four. And, finally, a conclusion and future work is offered.

Related work

Human activity recognition (HAR) finds its application in various fields such as healthcare, ambient assisted living, or security. Numerous studies have already addressed the possibility of identifying the movement of a person based on vision- or sensor information. At the present time, where telecommunication networks have developed into a significant and widely accepted market, methods in transport mode detection rely mainly on gathering different data from smartphone devices such as GPS or accelerometer-based information.

As an example of transport mode detection from GPS trajectory data, Dabiri et al. [DH18a] present a model based on GPS-based features computed from the collected raw GPS data such as speed, acceleration, jerk, and bearing rate with the aim to predict five classes which are walking, bike, bus, driving, and train. The data pre-processing steps include the clearance of possible outliers and inaccurate GPS points and the division of GPS tracks into trips that are further split into segments of the same length. For determining the uniform segment size, the median of the number of GPS points in all trips is used. After the segmentation into single-mode parts of the same size, Convolutional Neural Network (CNN) architectures are applied to the input samples to recognize patterns in the data. The best model in this study reaches 84.8 percent test accuracy.

In contrast, the acceleration sensor of a smartphone is used by Nick et al. [NCGG10] for differentiating three modes of transportation: cars, trains, and pedestrians. Before splitting the sensor data into windows of four seconds with a 50 percent overlap features like mean value, standard deviation, or maximal and minimal value of a window are calculated either on the normalized acceleration values or the cumulated sensor data during the pre-processing stage. In their work, they manage to achieve over 97 percent prediction accuracy by the use of a Support Vector Machine which is used to analyze the data for classification.

While the papers mentioned above are making use of a uniform-based segmentation approach Zheng et al. [ZCL⁺10] introduce a method drawing upon change-point detection. They suggest dividing a GPS trajectory into alternating Walk- and Non-Walk segments depending on upper threshold values of velocity and acceleration. For keeping the resulting single-mode sequences as long as possible respectively to avoid an over-segmentation segments are merged into their previous ones because of short distance or time span. Besides basic features such as the average velocity a set of advanced features which is stable to real-world traffic conditions consisting of change rate in direction and in velocity, and stop rate are determined for each segment then. The deployment of a Decision tree-based inference model is used for the classification into four means of transportation (driving, walking, taking a bus, and riding a bicycle). By the additional application of a graph-based post-processing algorithm, a result of 75 percent inference accuracy is presented.

Whereas this work relies on different sensor information (acceleration, gyroscope, magnetometer) as well as on GPS data of smartphone devices for the users' transport mode inference, Li et al. [LYL⁺22] employ a smartwatch-based segmentation approach to realize long-term activity monitoring and recognition of a wider range of human activities such as staying, walking, climbing stairs, jogging, running, and jumping. Only the regular movement of the smartwatch on the wrist enables an arm-swing-wise segmentation. After extracting the start and end points of each arm swing from the square wave resulting from the raw data an adaptive sliding window method is used which changes the size of the window and step along with the amplitude and speed of arm swings.

However, segmentation methods are not only used in the context of human recognition activity but find wide applications in other fields like digital image processing also. Here, for example, related regions are created by grouping together neighboring pixels that correspond to a certain criterion with the aim of identifying objects that are logically related. Cheng et al. [CJW02] propose a method for image segmentation using two steps: homogram thresholding and region merging. First, local thresholds are defined to categorize pixel values with the aim to find coherent objects. Depending on whether a threshold is exceeded or not reached homogeneous clusters are composed in a color image. Then, the region merging approach is applied to join clusters that are sited very close together on the one hand and take into account clusters that include too few pixels to be relevant on the other.

Methodology

This chapter is composed of three sections: Architecture, Client, and Server. An overview of the main modules, their interaction, and used tools for implementation is provided in the first part (Architecture 3.1). The following Section (Client 3.2) gives the background on the Client module, introducing the functioning of the TMD-App. Server-side components are explained in the final section (Server 3.3), which includes the major part of this work's efforts. This passage will describe the server-side infrastructure, the implementation of the segmentation, and its preparatory steps.

3.1 Architecture

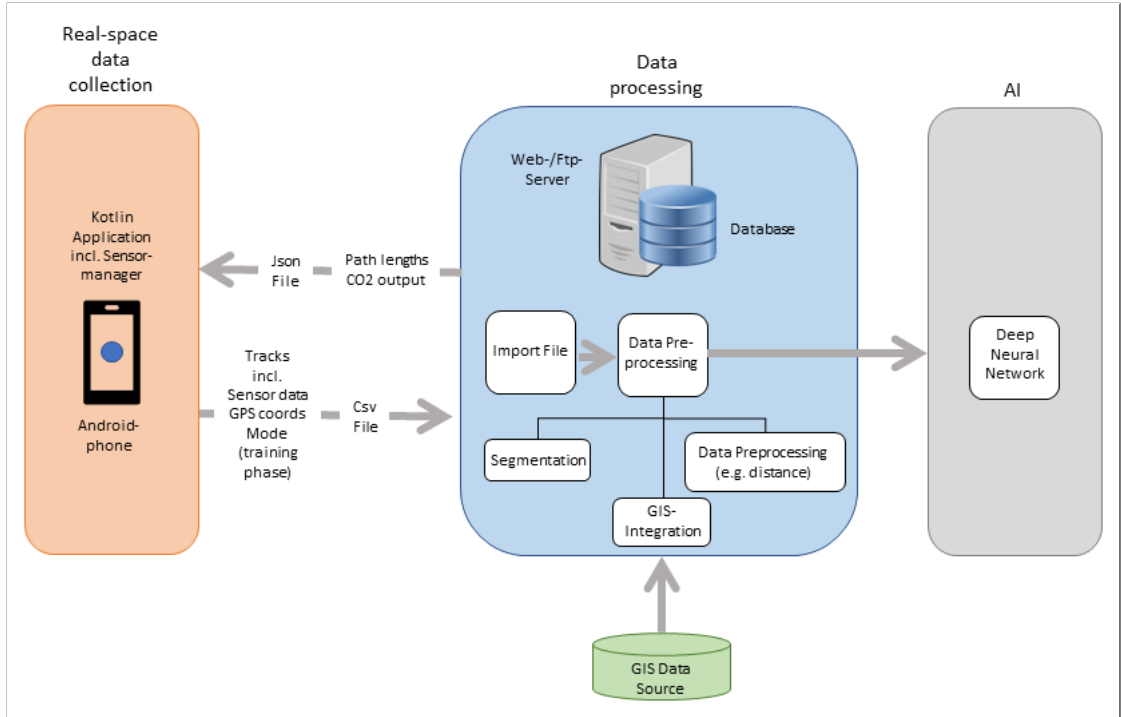
The implemented software consists of two main modules:

1. the Client Module which is responsible for recording the tracks and uploading the collected data to the server side,
2. the Server Module where the data gets processed and stored in a database.

This chapter will further explain the interaction of processes and used tools.

As shown in Figure 3.1 in the first step users have to record their tracks by using the TMD-App via an Android Smartphone. The client Android application is written in Kotlin [Fou22a], which is the favored programming language for such purpose since Google I/O in 2019. The recorded tracks include various sensor data and GPS coordinates. During the AI training process, the travel mode, which has to be manually selected by the user, is attached as well. The data of each track gets temporarily saved into a new Comma-separated value (CSV) file and uploaded via HTTP post request to the Server. The status of the upload process is sent from the Server as an HTTP response, which is displayed in the App.

Figure 3.1: TMD Architecture Diagram



For the server-side implementation Python (v3.10) [Fou22b] was chosen. For this choice, there are many reasons why. Among others, Python is considered to be highly productive and can be used to support machine learning. As server software, Apache/2.4.41 on Ubuntu Linux and as a database MySQL is used.

The server applications consist of 3 main components:

1. The main component "tmdlearn.py" implements the main logic of importing the data, applying various calculations (features) like determining the length of the track, elevation, etc. as well as segmenting the track. The prepared data is stored in a MySQL database.
2. The watcher is a component that uses the watchdog Python library to watch the upload folders and monitor whether new files have been uploaded. If the watcher finds new files they are sent to the main module to be processed. The watcher also shows basic information about up-time and status on a basic web page.
3. Django Rest framework [Inc22b]: Django was used for the communication with the Client, handling the uploads of the CSV files as well as providing information for the user, like for example the covered distances per travel mode and CO2 emissions accrued thereby, which can then be shown in the Client App.

In order to separate the application layer from the infrastructure level the Python Application was "dockerized" [Inc22a]. Docker of course helps by segregating and isolating the dockerized app, minimizing the impact on the host system. Also - and that's the main advantage - it makes moving the application to another infrastructure very easy. Docker compose was used for creating a script that sets up two interacting containers, one for the Python app and another one for the database.

The prepared segments can be fetched from the database by a Deep Neural Network as implemented in the thesis of my colleague Jakob Lang [Lan23], which determines the transport modes then. The outcome may be used to represent the Modal split for example.

Tools that were used:

- Server Development: Pycharm (Python IDE)
- Client Development: Android Studio (Android IDE)
- Various: Putty (SSH client for Windows), Filezilla (SFTP utility), PhpMyAdmin (interface to the SQL database), Postman (tests API endpoints based on HTTP), Bootstrap v5.1.3

Used Python libraries:

Django 4.0.4, djangorestframework 3.13.1, geopy 2.2.0, mysql_connector_repackaged 0.3.1, numpy 1.22.3, requests 2.27.1, scipy 1.8.1, watchdog 2.1.8, etc.

3.2 Client

A basic Android Application named "TMD App" for recording sensor and GPS data was built by my colleague Jakob Lang using Kotlin as the programming language. The TMD App was expanded in this work with a function for the automated upload process of the stored tracks from the Android device to the server (Subsection 3.2.2). Furthermore, the TMD App was extended through the addition of two more features (Subsection 3.2.3).

3.2.1 Overall functioning and Data gathering

After the installation of the TMD App, which is available for download on the TMD website (<https://kontor.cg.tuwien.ac.at/tmdlearn/>), the main activity is presented as shown in Figure 3.2. The user first will be asked to select the training label that indicates the used transportation mode. After starting the recording of a track by tapping the "Start Tracking" button, the following data from the user's smartphone device is recorded into a stream:

- GPS coordinates (longitude and latitude)
- Transport Mode label (as selected by the user)
- other sensor data (acceleration, gravity, gyroscope, rotation, magnetic field, pressure) with a sampling period of 100 Hz including the current timestamp
- a numeric value based on the Android Device ID.

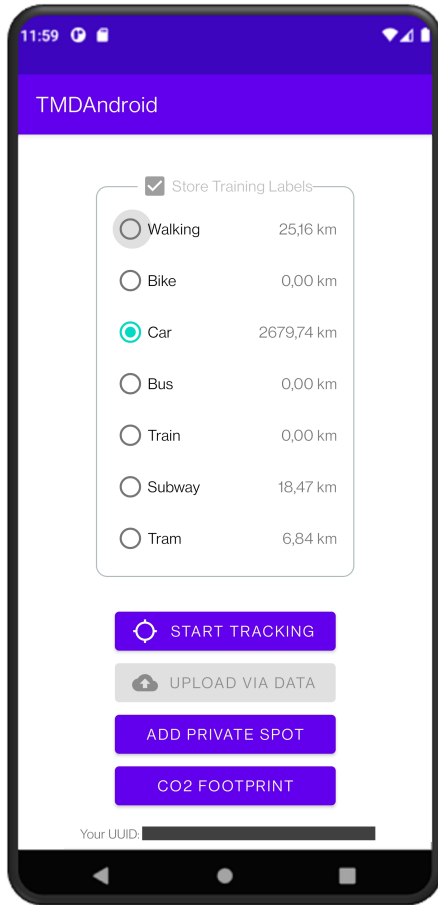


Figure 3.2: Main screen

At all times during the recording of a track, the user may change the transportation mode by tapping the corresponding option. In this case, the transport-mode label of the current track will be changed but the recording of the track continues otherwise uninterrupted. When the recording of the track is stopped by the user by tapping on the "Stop tracking" button, the stream is saved into a CSV file. As soon as a WIFI connection is available, all the recorded tracks are uploaded to the server, where the datasets are further processed. When the user stops a recording and the device has only internet access via cellular there is the possibility to start the file upload manually by clicking on the button "Upload via Data". If the client receives a successful response from the server (HTTP 200), the corresponding track is deleted from the device's file system. After the successful upload and processing of the data, the overall distances tracked by the user per transport mode are calculated and will be queried by the client on the next refresh of the activity. This data is then displayed next to each transportation mode as an additional incentive for the user to collect more data.

In order to make the TMD App accessible for potential test users it was distributed by sending the link to the TMD website where among other things the information is provided that no individual-related data gets recorded and a

list of all data types used can be found. The TMD App was used by 19 volunteers over about three months. More than 360 tracks with a total of 26.741.588 data points were recorded so far. Please refer to Figure 3.3 for more details on that.

Transport-Mode	Duration (h)	Distance (km)
BIKE	36.2	236.8
BUS	9.5	104
CAR	97	4739.3
SUBWAY	18.6	165.6
TRAIN	6.9	405.7
TRAM	3.1	44.2
WALKING	73.3	357.6

Figure 3.3: Recorded distance and duration per label

3.2.2 Client-side file upload

As soon as a WIFI-connection is available on the smartphone device or the Button "Upload via Data" was activated, the class UploadUtility.kt, which includes information about the designated server URL and the upload path of the server, is called. For each stored file in the tracks directory of the device, an HTTP-POST request is generated by `okhttp3.RequestBody (MultipartBody)`, which consists of the track file in CSV format and its source path. During the transfer of the files to the server URL (`https://kontor.cg.tuwien.ac.at/tmdlearn/tmd_upload/upload/`), the user gets information about the upload status using the Android toast notification system ("Currently Uploading...", "Upload Successful!"/"Upload failed!"). The upload directory is monitored by the "Watcher" module (see Subsection 3.3.6) for file changes. If the Watcher detects a new file, it will be sent to the main TMDLearn module for pre-processing and segmentation of the data. For more information about the server-side upload process, please see Subsection 3.3.1.

3.2.3 Additional App activities

In order to create additional incentives for the app users and satisfy their requirements concerning privacy aspects, two extra activities were added to the main functionality of the TMD App. To meet the standard for a responsive design, the UI of these views was built with ConstraintLayout, a support library for Android systems.

CO2 Footprint

Transport produces almost a quarter of carbon emissions and is the main reason for air pollution in cities. Understandably, people may be curious about the impact caused by their own choice of transportation. Knowing and comparing the carbon footprint can encourage the user to adapt travel behavior towards more conscious travel choices. On that account, the TMD App tracks automatically transport emissions and delivers

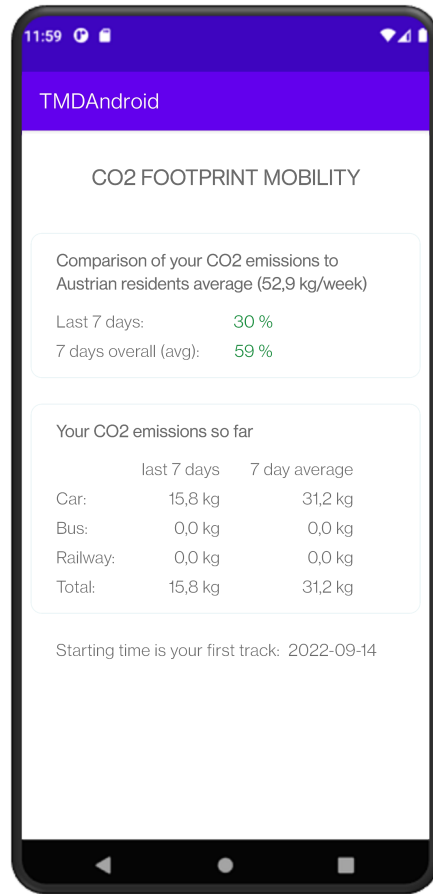


Figure 3.4: CO2 footprint

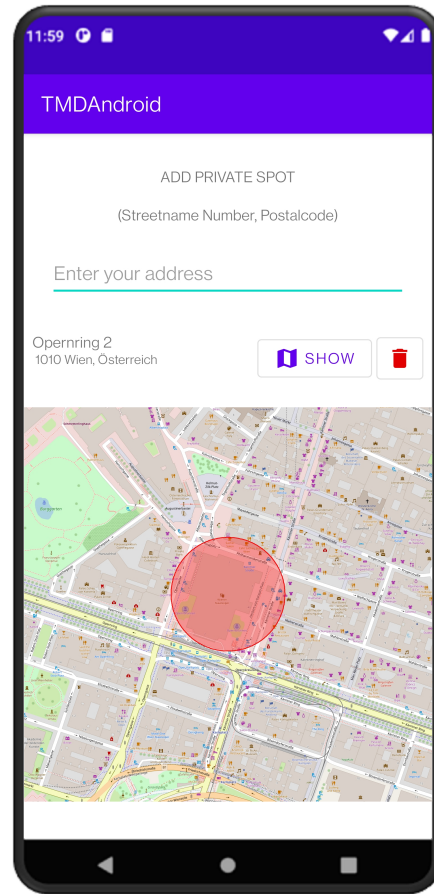


Figure 3.5: Privacy spot

insights about carbon pollution from the mobility of daily life through the CO2 Footprint section (Figure 3.4).

The carbon footprint including average wastage per travel mode is represented on an extra view, which is divided into two main parts:

1. The user's last seven days and the seven days average overall of the produced carbon are displayed in relation to a reference value, which was 2.760 kg for an Austrian resident in 2019 [VCO21] (52.9 kg/week, respectively). If the value exceeds the 100 percent threshold, its color will change from green to red.
2. The CO2 wastage in kg caused by the App-User is displayed for the last seven days and the overall seven-day average per travel mode. The calculations take into account the different travel modes car, bus, and railway (train, tram, and subway together) and are based on emission values 2022 of the Austrian Federal Environmental Agency [Umw22].

The time of the first track recorded by the App-user is defined to be the start of the calculations, which is also shown in the status line.

All the values are updated every time the button "CO2 footprint" is tapped in the main activity by calling the `updateFootprintUI()`-method of the class `FootprintActivity.kt`. This method calls the `emissions(request, UUID)`-method on the server, passing the User device ID, which then queries the database (e.g. 3.1), getting the result of produced emissions simply by multiplying the covered distances per travel mode with the respective matching CO2 values (stored in the `config.ini` file on the server). The result is returned to the client as a JSON file.

Listing 3.1: SQL statement CO2 emission for travel mode car

```

1 SELECT 'last_seven_days',
2 COALESCE((SUM(length_car)/1000*{co2_car}),0) AS s_car,
3 MIN(tracking_date) AS start_time
4 FROM tracks
5 WHERE tracking_date IS NOT NULL
6 AND uuid LIKE '{uuid}'
7 AND augment_factor = 0
8 AND tracking_date > DATE_SUB(NOW(), INTERVAL 7 DAY)

```

Privacy spots

In order to design a privacy-friendly application, an overview of the transmitted data is listed on the TMD website. The app does not track any personal data except for the user ID, which is linked to the recorded tracks. This ID is hashed from the device ID, which can not - or at least not easily - be linked to a person. However, the users are offered an additional possibility to assign geographical areas, in which no coordinates are recorded at all. Therefore, an according feature was implemented (Figure 3.5), which can be activated by tapping the button "Add private spot".

When entering an address, providing street name and postal code, the proper values of latitude and longitude are found by Geocoder, a geocoding library written in Python, using its method `getFromLocationName(String locationName, int maxResults)`. The determined coordinates are stored as a key-value pair locally, using Kotlin's Shared-Preferences interface. If such a private spot was specified before starting tracking, a random point displaced within a distance threshold (default: 50m) in x- and y-direction is calculated per track. The generation of a uniform, random, and independent point within a circle was implemented based on this post <https://stackoverflow.com/questions/36905396/randomly-generating-a-latlng-within-a-radius-yields-a-point-out-of-bounds> provided by Markus Kauppinen. That way identification of the private circles' center is impossible. On every service update then, the distance between the current and the randomly offset location gets checked. In the case where the current coordinates of the device's position are inside a radius threshold

value (default: 100 m) of this/these random point(s), they are not recorded by the TMD App. Instead, a new track starts if a user travels between a private and a non-private sector.

This activity was extended to save more than one address and to show the chosen private zones on a map by my colleague Jakob Lang.

3.3 Server

A multi-container application was created for server-side development using Docker and Docker Compose. The latter automates the creation of the two containers that are used for our project: “web” and “db”.

```
docker-compose.yml:
version: "3.3"

services:
  web:
    build: .
    command: python manage.py runserver 0.0.0.0:8000
    volumes:
      - ../code
    ports:
      - "8001:8000"
  db:
    image: mysql
    restart: always
    environment:
      MYSQL_USER: 'tmdlearn'
      MYSQL_ROOT_PASSWORD: '****'
      MYSQL_PASSWORD: '****'
      MYSQL_DATABASE: 'tmdlearn'
    ports:
      - "3307:3306"
    volumes:
      - ../database:/var/lib/mysql
volumes:
  my-db:
```

The compose file includes instructions to build the database service by downloading and installing the latest MySQL image, mapping the appropriate ports, and creating the environment by setting up a database and the credentials. Additionally, a local folder is mapped to the `var/lib/mysql` folder where the database files are stored. In upgrading the

database docker, these files will not be overwritten. For the web service, the compose file includes instructions to first build the contents of a Docker file, which installs Python as well as all the libraries as described in the requirements.txt file.

```
Docker-File:
FROM python:3
ENV PYTHONDONTWRITEBYTECODE=1
ENV PYTHONUNBUFFERED=1
WORKDIR /code
COPY requirements.txt /code/
RUN pip install -r requirements.txt
COPY . /code/
```

Like with the database container, a local folder is mapped, where all the Python files will be accessible from the local file system. After building the Docker file, the docker-compose file includes a command to start the Django Server (“python manage.py runserver 0.0.0.0:8000”).

3.3.1 Server-side file upload

The main methods for communication with the client can be found in the view.py file in the tmd_upload folder. The post() method gets passed a request object when it is called by the client. Some basic security checks are being done, a filename based on the user Id and current date time is generated, and the file is saved into the settings.MEDIA_ROOT folder as defined Django settings file (settings.py). Based on the success of the operation, a JSON response, that includes a basic message as well as the status code (201 for success or 500 for failure), is returned to the calling client.

3.3.2 Server process overview

The main() method of the main program "tmdlearn.py" handles a few basic tasks: reading the config.ini file that defines the required paths (uploads, archive, augmentations, augmented), parsing the two possible arguments (-include_gis: whether GIS Data should be used, -offset: in case you want to augment the file), connecting to the TMDLearn database, and setting up the logging. Then a list including the names of all uploaded and not yet processed files of the designated directory is created. In case this list returns at least one file, the GIS Data Map gets opened (if the -include_gis parameter was set to 1 when starting the program). - The GIS transport network of Vienna was integrated for better differentiation between motorized vehicles. In that regard, the value of the distance to the nearest public transportation station for a given latitude and longitude is saved in the database. Looking at the file size of the GIS Data Map, which is more than 300 MB, it is necessary to make sure, that this file is opened only once since it is a quite lengthy process.

After that, a for loop iterates over the list of files, calling the method "process_one_file". This method checks first the number of lines of the imported file. If the duration of a track is below a defined threshold value (default: 120 seconds, resp. 12000 lines at a record rate of 100 Hz) it gets discarded from further processing, because constant time is required for CNN, besides recognition may not be good enough. The next steps include the import of the logged track into the database (Subsection 3.3.3), computations of several indicators and features (Subsection 3.3.4), and segmentation (Subsection 3.3.5). After successful processing, the uploaded track file is moved to an archive directory.

During the whole processing, the Python logging module is used for providing proper info-, debug- and error messages. The logger uses Streamhandler for displaying the log in the console and Filehandler for saving it in a file. Three different log files are created: execution via console with no offset given ("TMDLearn.log"), "TMDLearn_Augmenter.log" in the case an offset greater than zero is passed (Subsection 3.3.6), and "TMDLearn_Watcher.log" (Subsection 3.3.6).

Besides that, the use of global exception handling shall lead either to timely termination at fault or proper error information, depending on the severity of the problem. In case of a keyboard interruption or an abortion after an error, a clean-up routine should be called, which tries to delete the incomplete track in the database.

3.3.3 Importing samples

The python class "CsvImporter" is responsible for reading an incoming CSV file with its included User ID, filename, timestamp, GPS (latitude and longitude), and sensor data (linear acceleration, gravity, gyroscope, rotation, magnetic, pressure), and optional label and loading it into the TMD database. Firstly, the User ID, which was checked for validity before, and the filename are inserted into a new entry in the databases table "tracks" (please reference Figure 3.6). This SQL INSERT statement generates a unique track ID for the related data record, which can be accessed using the `MySQLCursor.lastrowid` Property.

In the next step, all the logged values, including the newly created track ID as a foreign key, are stored in the table "trackpoints". The import is done line per line using Python's Built-in CSV Library for parsing the CSV File. To identify invalid records of the track, the expected file structure gets checked and an invalid line is discarded if necessary. At this stage, some basic calculations are done, like time- and distance difference and totals, checking whether the line contains valid coordinates and the elevation (details: Subsection 3.3.4). If a line is valid, the sensor as well as GPS data, and computed values are collected into a multidimensional array. For every 50.000 lines, the contents of the array are committed to the trackpoints table. Figure 3.7 shows by way of example an excerpt from such a result of an import into the database table "trackpoints".

Figure 3.6: Excerpt of TMD database, table "tracks"

trackID	uuid	tracking_date	length_walk	length_car	length_bus	length_subway	augment_factor
385	514425f9eb45c17b838abd019bb9c3ef	2022-11-26	0	0	0	4358.85	0
386	514425f9eb45c17b838abd019bb9c3ef	2022-11-26	0	33776.2	0	0	0
387	7dd94bf53c29d09ceae367d869c540fb	2022-11-27	1987.78	0	0	0	0
388	514425f9eb45c17b838abd019bb9c3ef	2022-11-27	667.335	0	0	0	0
389	d6562c430a16e3f0c76b2a1fa55f3d08	2022-11-28	0	0	4022.98	0	0
390	d6562c430a16e3f0c76b2a1fa55f3d08	2022-11-28	0	0	3439.86	0	0

Figure 3.7: Excerpt of TMD database, table "trackpoints": import of logged values

trackID	pointID	latitude	longitude	label	IAccX	IAccY	IAccZ	gravX	gravY	gravZ
386	24948155	48.41082985	16.53773645	CAR	0.5269108	-0.31047368	0.75868654	7.3833904	-0.03625105	6.4540386
386	24948156	48.41082985	16.53773645	CAR	-0.14207315	-0.0993703	0.1960082	7.401994	-0.022868143	6.4327555
386	24948157	48.41108793	16.53791032	CAR	-0.25802612	0.12100858	-0.5845237	7.4206767	-0.013798178	6.4112215

3.3.4 Indicators and features

Speed related characteristics

Since speed-related attributes are instrumental in detecting travel modes, movement attributes based on GPS data are provided in addition to the sensor records in this work also. As recent papers, describing transport mode detection methods based on GPS tracking data, use mainly speed, acceleration, jerk, and bearing rate [LZZ⁺20, MJ20, DLHR19, DH18b], these very same features have been selected to be implemented in advance. For a list of the features use, see the thesis of my colleague Jakob Lang. However, speed and acceleration are needed for change-point segmentation anyway.

Whenever work is performed with GPS data, the occurrence of signal loss must be considered in the first step. If latitude and longitude have zero values, or consecutive coordinates have the exact same values, a lost GPS signal can be assumed. To assure correct results related to the distance traveled of the time elapsed, lines containing such invalid GPS points are treated specially. To this end, the latitude and longitude values of each data row are screened during the import processing while keeping the last valid GPS position and the so-far cumulative duration memorized.

While reading the recorded track, distance, and duration, derived from the location, are computed per line. These indicators are on the one hand the basis for further calculations such as speed or acceleration and on the other hand supply interesting aspects for users and analysis. To obtain the cumulative sum of the track distance, the distances between

3. METHODOLOGY

each of two consecutive GPS points are determined by the Python Geopy library, using the geodesic distance based on the Vincenty formula and added up then:

```
distance_diff_m = geopy.distance.geodesic(current_coords,
last_valid_coords).m
```

In order to speed up calculations and subsequent changepoint segmentation steps, only every hundredth line of the 100 Hz recorded track is taken into consideration. A flag named "is_gps_point" is set in the database to identify only the affected lines after computation.

After these preparatory measures, the speed is calculated by dividing the differences in the distance through time:

```
speed_unfiltered = distance_diff_m / time_diff_sec
```

Since errors in GPS logging can lead to a considerable source of noise, center-weighted averaging by a Gaussian distribution is applied using the 1-D Gaussian filter of the scipy library. This frequency filter turns a distribution of a large number of measurement points into the Gaussian distribution curve (bell curve). By attenuation of the high signal components, noise and details are removed. The definition of the one-dimensional Gaussian filter is:

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$$

Where x is the distance from the origin on the horizontal axis and σ is the standard deviation of the distribution.

As higher sigma values blur over a wider radius on the one hand but mean a slowdown in computation time on the other hand, sigma = 1 seemed practical for this work. For recognizing the effect of filtering a car-labeled track recorded by the TMD App, see Figures 3.8 and 3.9 as an example.

The computations of the remaining characteristics, which are acceleration/deceleration, jerk of a point (the rate of change in the acceleration), and bearing rate (angle between the line connecting two successive points and a reference), are based on the work of Sina Dabiri et al. [DH18b] and applied by converting the suggested calculations into proper SQL statements.

Figure 3.8: Speed (unfiltered)

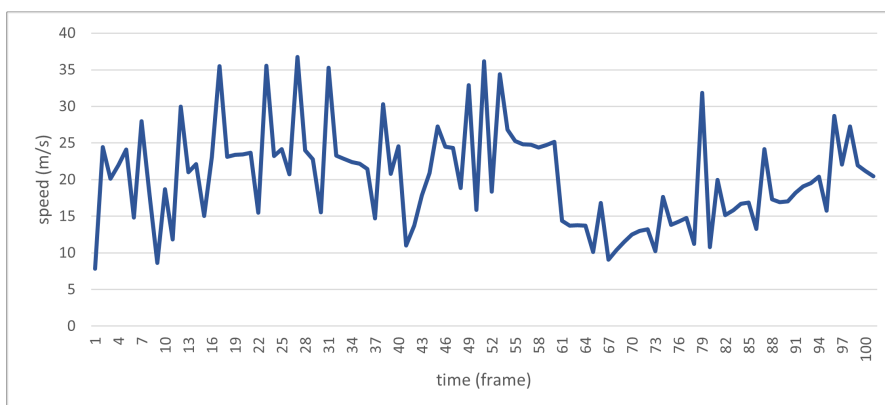
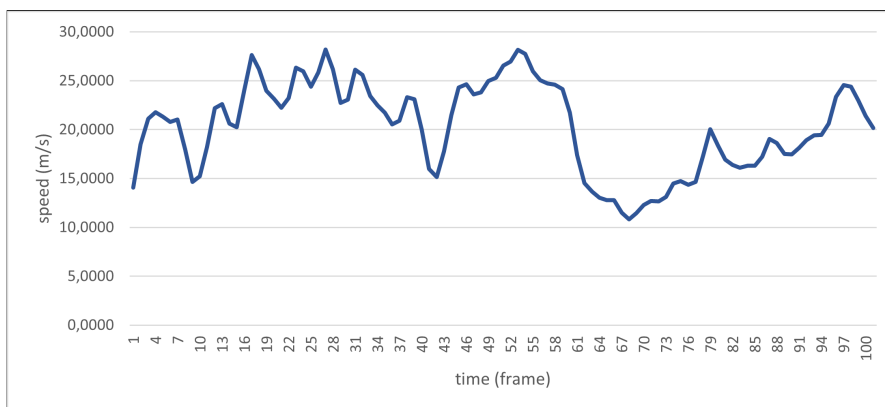


Figure 3.9: Smoothed speed



This is illustrated by the extract of the class `feature.py` below:

```
import logging
import mysql.connector

#  $a(P1) = (speed(P2) - speed(P1)) / time\_diff$ 
def get_acceleration(self):
    self.my_logger.info("Start calculating acceleration")

    sql: str = f"
    UPDATE track_points A, track_points B
    SET A.acceleration = (B.speed - A.speed) / B.time_diff_sec
    WHERE B.pointID = (A.pointID + 100) AND A.is_gps_point = 1
    AND A.trackID = {str(self.track_id)}"
```

```
AND B.time_diff_sec > 0"

    try:
        mycursor = self.db.cursor()
        mycursor.execute(sql)
        self.db.commit()
    except mysql.connector.Error as e:
        self.my_logger.error(f"Could not commit acceleration:
            mysql Error: {e.errno}, {e}")
        return 0
    self.my_logger.info("Committed acceleration")
    return 1
```

Elevation

Having tracked the GPS coordinates by the TMD App, the elevation can be determined for each GPS point as an additional feature. There are several methods for retrieving the desired data. One option would be to apply the `getAltitude()` function on the location object, which is obtained from the Android location listener on the client. This approach is practicable but involves the major drawback, that the altitude is either retrieved from pressure sensors, which are not built in on devices generally, or by Google. In the latter case, there could arise privacy issues. Another widespread possibility would be to use an Elevation API Web Service, which responds in general to HTTP queries. The drawback: this could cause heavy data traffic and potential payment requirements.

However, in this work, a digital terrain model (DTM) of Vienna and Lower Austria was made available as a Geo Tiff file according to the WGS84 coordinate system and with an accuracy of ten meters. When processing an incoming track, the RAW image of the Geo Tiff file (converted by the software suite ImageMagick) is loaded by using the `imread()` method of the OpenCV-Python library, which returns a NumPy N-dimensional array containing the coordinate and elevation values for each pixel. If the tracked coordinates of the TMD App have valid values, the method for determining the elevation of this GPS point is called already during the import process. Knowing the width and height as well as the corner points of the grid from the metadata of the Geo Tiff file, OpenCV enables access to the designated elevation value by passing the determined pixels for a given latitude and longitude. To ensure the accuracy of the results, random checks were performed, using the height service of the Austrian Geo service [bÄdL22].

If an exception is thrown during calculating indicators or features further computation and segmentation steps are discarded.

3.3.5 Segmentation

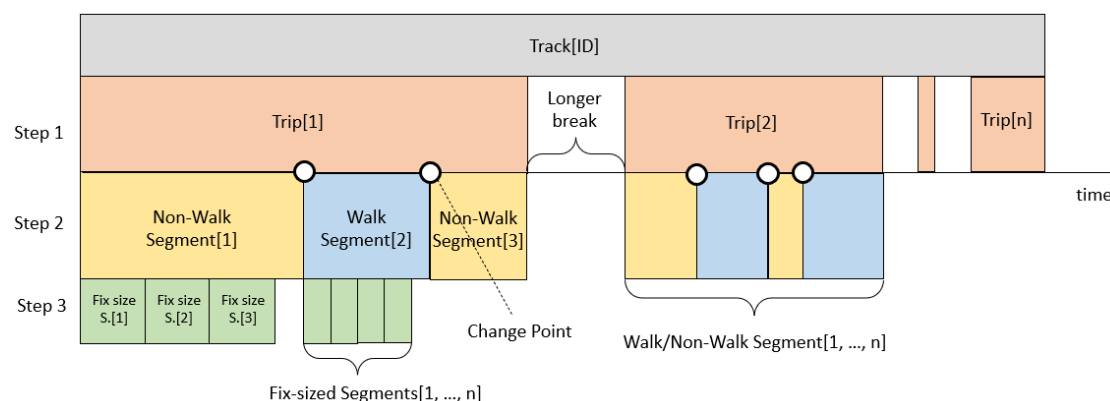
Essentially, the aim of segmenting the movement data is to provide single-mode data sets, which help the machine learning to create a reliable Transport Classifier during the training phase and help the downstream classification process.

The segmentation process is done in three basic, consecutive steps:

1. A **track**, respectively the entire GPS-Log, is divided into **trips**, where longer interruptions in the logging are omitted (see the next Subsection 3.3.5 for details).
2. **Trips** are classified into **single-mode segments** in the case of a labeled record during the training phase or by change point detection otherwise. Detailed information about the implementation of the change point detection can be found in subsection 3.3.5.
3. Finally, these alternate **Walk-/Non-Walk segments** are split into **fixed-size segments** to meet the requirement of the Neural Network. Please see the description below 3.3.5.

In Figure 3.10 you will find the individual phases of the segmentation as described above illustrated.

Figure 3.10: Understanding Segmentation



Step 1: Segmentation of a track into 1, ..., n trips separated by longer pauses. Step 2: Change-Point Segmentation of a trip into 1, ..., n alternate Walk-/Non-Walk segments per trip. Step 3: Segmentation of a Walk-/Non-Walk Segment into 1, ..., n fixed-size Segments.

Each step of the segmentation process further details the data in the track points table by setting various flags and fields instead of moving the data to different tables. For example, if a track point is detected to be part of a trip, the field "trip" will be set to the number of the trip. Or, if a track point is considered to be part of a trivial segment, the flag trivial_segment is set to 1, etc. All segmentation steps are implemented in the Segmentation class which can be found in the segment.py module.

Segmentation of tracks into trips

The division of a track into trips, split by and omitting longer pauses, is used by segmentation methods using fixed-sized segments as well as using segments, which shall be as long as possible. Contrary to longer interruptions, waiting- or stop time is considered an attribute of the transport mode and therefore remains part of a trip.

To verify an effective pause, a time frame is used, in which either a certain distance (as suggested by Paulo Ferreira et al. [FZV20]) or velocity is not exceeded. Looking at other eligible papers [ZCL⁺10]), tracks are split up into trips if the time difference between two consecutive GPS points exceeds twenty minutes. This seemed a reasonable time span and was therefore also used for this work. As 3,6 km/h can be understood as the average walking speed or even only 3 km/h unhurried average, a speed threshold of 2 km/h is set in this thesis. This value is mentioned for the stop-change point by P. Sadeghian et al. [SZGH22] as well. In other terms, a track gets separated into trips if the speed is less than 2 km/h over more than 20 minutes. Under those conditions, it may be assumed that no traveling takes place.

This first step of the segmentation is implemented in the method `tracks_to_trips()`, which is the first method called in the Segmentation class. It identifies all windows over a minimum of 20 minutes inside a track, in which the speed limit of 2 km/h is not exceeded. In order to do that, the point IDs with their speed and time differences (which were calculated in one of the previous steps) of the track to be processed are fetched from the `track_points` table. The results are iterated through with a for-loop. If the absolute speed drops below the threshold of 2 km/h for the first time, we assume this point to be a potential starting point of a rest period, and the subsequent entries are examined. As long as the speed stays less than the threshold, we assume these points to be part of a longer pause and store them in an array ("rest list") temporarily. If the time gap of collected points exceeds 20 minutes in the end, there is an actual rest period. The data points of the "rest list" are marked accordingly by updating the corresponding field `rest_period` to 1 in the database. The rest list is cleared and the next points are examined until another potential starting point is found.

After this, the method `trip_numbers()` again fetches the track points but leaves out the rest periods, that were set in the previous method. The result is ordered by pointID. If the difference between two subsequent pointIDs in this result set is greater than 1 it means that the track points in between have been set as a rest period and therefore are not present in the result set. All track points after this point will be updated with a new trip number until the next gap in pointIDs occurs. Figure 3.11 shows an example of the outcome of the procedure described above. In the subsequent steps, only the trips are used for further processing.

Figure 3.11: Example of the partition of a track into trips

Track ID	Trip	Rest period	Point ID	Speed in km/h	Time diff in sec	Duration in min
433	1	NULL	27148486	0	0	8.21
			
			27167786	2.19	9.09	
	NULL	1	27167886	1.19	5.04	34.18
			
			27201886	1.66	2.02	
	2	NULL	27201986	3.09	2.02	15.82
			
			27222686	3.89	2.02	

Segmentation of trips into segments by changepoint detection

After extraction of the trips, they get separated into segments, which are logically connected by the classification type and consecutive timestamps. In the case of labeled trips, which are used to train the Neural Network, a new segment starts based on the change in the labeled transport mode. The segmentation of unlabeled data records (after the training process) raises the question regarding the choice of the appropriate change point to use. For instance, there can be found papers describing change point detection by stop and move segmentation. But this approach has a major drawback in that a transport mode could be often changed without stopping. It seems more likely that a starting- or endpoint of walking marks such a change point. This assumption is supported by the analysis of the behavior of mobility patterns amongst the Viennese population ([Wie21]), which documents that the most commonly used form of travel on the way to a stop of public transport is by walking (97,5 percent). Furthermore, this study proves that Vienna is a pedestrian town, as 99,9 percent of the city's population is walking daily, and 37 percent of journeys are taken on foot in the year 2020. For these reasons, walking was chosen to be a change point between different modes of transportation in this work.

The change point segmentation was realized based on the work by Zheng et al. [ZCL⁺10], where segments are categorized into two classes (Walk, Non-Walk) first, trivial segments are merged into the previous segment then, and finally, segments of short distances are concatenated together as one segment. In the following, the steps of the implementation are described more in detail:

1. First, the data lines of each trip of a track, which were marked as GPS points beforehand (3.3.4), are grouped into 1, ..., n alternate walking and non-walking segments. To achieve this, each GPS point is assigned to the correspondent category Walk or non-Walk using an upper bound of velocity and acceleration

(default: $v = 9 \text{ km/h}$, $a = 1.5 \text{ m/s}^2$). - In comparison, the average speed of cycling is 16-22 km/h at a beginner's level.

2. Due to possible inaccuracies of the GPS signal a walking segment could be wrongly detected as not walking. On the other hand, in the case of a vehicle, the presence of heavy traffic may lead to a false identification as walking. In order to improve the result of the segmentation, an upper distance- and time threshold are introduced and used to describe trivial segments. Zheng et al. demonstrate in their work, that the best results could be achieved by using 20 meters as a Minimal Distance Bound and 10 seconds as minimal time. To indicate such a trivial segment, the distance, and duration of each Walk and non-Walk segment retrieved from the first step of this method are checked against their thresholds. If one of the parameters is lower than its boundary value, the respective segment gets merged into the previous one by updating the segment numbers 1, ..., n per trip accordingly.
3. It seems unlikely that people change their transport type several times over a relatively short distance. Therefore the mode of transportation in a segment below a particular distance can not be assessed with certainty. To counteract a trivial partition, uncertain segments are concatenated as one non-Walk segment, in the case of occurring successively more often than a threshold. As Zheng et al. realized the most acceptable outcome for their experiments by selecting 200 meters for the distance threshold (DT) and the value of 2 for the number of consecutive Uncertain Segments, these settings were also taken over by this work. In Figure 3.12 the concatenation of uncertain segments and concluding segment numbers is demonstrated in an exemplary manner. In this example, a track t consists of two trips, which indicates a longer pause (length = j) in between. During the first trip, six alternating phases of walking/non-walking were determined (already after applying step 2 of this method, which is not listed separately in this Figure), from which four had less than the traveling distance bound. These parts are recognized as Uncertain Segments and are merged as one non-Walk segment. The Segment number is the result of the fourth step in the method, which combines sequences of the same walking/non-walking categories.
4. In a post-processing stage, successive sequences with the same label are merged. Ascending segment numbers per trip, which increase at each change of walk/non-Walk segments are finally set.

Creation of fixed size segments

The segments created by change point detection, as outlined above, are not directly meeting the requirements of the convolutional Neural Network that we use in the thesis of my colleague Jakob Lang, as they expect an input of fixed size. To this end, the previously determined walk/non-Walk segments are split into segments of equally sized parts of 60 seconds in this work, which corresponds to several 6.000 lines at a record

Figure 3.12: The result of getting consecutive segment numbers after merging uncertain segments

Track ID	Trip	Point ID	Walking (0/1)	Distance	Certain Segment (0/1)	Segment Name	Segment Nr.
t	1	i	0	> DT	1	NonWalk	1
		i + 100	0				
		i + 200	0				
		i + 300	0				
		i + 400	0				
		i + 500	1	< DT	0	NonWalk	
		i + 600	1				
		i + 700	0	< DT	0		
		i + 800	0				
		i + 900	1	< DT	0		
		i + 1000	0	< DT	0		
		i + 1100	1	> DT	1	Walk	
		i + 1200	1				
	pause	i+1300+j					
	2	i+1300+j+100	1	> DT	1	Walk	1
		i+1300+j+200	1				
		i+1300+j+300	1				
		i+1300+i+300	1				

rate of 100 Hz. The required fixed size (number of data lines) is passed as a parameter "segments-size" when calling the method "set_fix_size_segments()" and can easily be adjusted, if necessary. During the process of breaking down the data into fixed-size segments, they are numbered 1, ..., n per label respectively walk/non-walk segment.

3.3.6 Tools

Watcher

One possibility to handle receiving upload requests from the URL of the server is to execute the method process(request) of the class views.py, which calls the tmdlearn.main method. Since preprocessing a track is heavy on the processor and memory and takes quite some time to be finished, this method can lead to the server becoming unresponsive and not accepting new files. To facilitate the execution of continuous incoming files, this work takes advantage of monitoring file system events via a Python library, called Watchdog (<https://pypi.org/project/watchdog/>). For that purpose, the program watcher.py was created, which includes the watchdog's FileSystemEventHandler and Observer. The Observer is configured with the upload directory to watch and

a handler. If a change is happening on the monitored filesystem, the event handler object gets notified. It is first checked whether the event was triggered by a deletion, modification, or the creation of a new file. After ensuring that the new file is a CSV file, the method "process_one_file" of the main program tmdlearn.py, as described in the previous Chapter 3.3.2, is called. The watcher runs continually on the server until it is quit by killing the process.

To trace the watchers' activities, two watcher URLs were added. One shows the current status of the watcher, including its PID and which files were processed, including the corresponding track_IDs: <https://kontor.cg.tuwien.ac.at/tmdlearn/watcher>. The second provides detailed log- and debug information by reading the log file created by the logger (https://kontor.cg.tuwien.ac.at/tmdlearn/watcher_log).

Augmenter

During the phase of data gathering by test users, it became apparent, that some transport modes - especially city buses and trams - were currently not used as much as other ones. To compensate for the comparatively short distances covered by these, an opportunity is provided to create augmented tracks by using already existing ones, just with another offset. If an offset is specified as "3.000" for instance, the first 3.000 lines (e.g. 30 seconds at a rate of 100 Hz) of the track are not taken into account when processing the file. This creates additional tracks on the same data that are sufficiently different from the original and therefore provide a bigger sample for the training of the neural network. The solution was implemented through an offset counter during the import process of the files.

The augmentation of tracks can be enabled in the terminal in two ways:

1. through directly calling the main program tmdlearn.py with the desired offset as an input parameter:

```
docker exec -it tmdlearn_web_1 python tmdlearn.py
--include_gis 1 --offset 3000
```

This approach processes all currently existing track files within the upload folder with the given offset.

2. through the augmenter script with a given augmentation factor and label as input parameters:

```
docker exec -it tmdlearn_web_1 python3 augmenter.py
--augm 4 --label bus
```


The following applies if a_i is the augmentation factor with $i = 1, \dots, n$, and x is the number of lines in the fixed-size segments:

$$Offset(a_i) = \frac{x}{n}(i - 1)$$

This method provides the possibility to batch files with a given label and create $n-1$ augmentations of them. When starting the augmenter script, tracks, containing the label that was passed as an argument, are selected from the database, and the corresponding archived CSV files are temporarily copied from the archive- to the augmentations folder. The augmenter then loops through the list of files and first checks whether the file includes also transportation modes that are not needed for the augmentation. If this is the case the unnecessary lines are deleted. After that, the augmenter cycles through the n factors by calling the `main_process_one_file()` method of `tmdlearn.py` and passing the corresponding offset.

Anytime an offset different from zero is passed, the additional tags `augment_offset`, `augment_factor`, and `augment_label` are stored in the database table "tracks", indicating that this track represents an augmentation. In this way, it is ensured, that augmented data can be filtered in the database for any queries easily. So that they are only used for the training of the neuronal network. To ensure server responses to requests of the TMD App, regarding covered distance and produced CO2 emissions, are not corrupted, tracks with an augmentation factor greater than one are excluded in these cases.

Patcher

Using a database to store the analysis provides the possibility to easily make changes in specific parts of the pre-processing without having to completely re-import the CSV file. For example, a better method of creating the fixed-size segments was discovered after most of the training data had already been imported. The patcher program made it possible to apply the changes in the segmentation logic to all tracks that are stored in the database in this case. To take another example, after analyzing the imported tracks considerable variations in the recording frequency were evidenced. For this purpose, a new method, which interpolates possible missing values, was implemented. To change the already existing tracks in the database automatically, the patcher was used as well.

On the console, the program `patcher.py` can be started by specifying the following two parameters:

1. "mode": an improved method, which is implemented in the main program, can be reapplied to any set of the database. Currently, changes in the CSV import- and segmentation module are implemented. If "interpolate" is selected as the mode, the data points of the specified track are deleted in the first step. After that, the associated file is accessed and manipulated by running the revised method for the import before applying the preprocessing steps in consideration of the interpolated

lines. When "Segmentation" is set as a parameter for the mode, the entries of the database, which are related to segmentation steps, are reset before the enhanced method is executed on the chosen track(s). This method could also be expanded for other stages of pre-processing.

2. "trackID": a single track ID or a range between two given track IDs, to which the chosen mode should be applied.

3.3.7 Database

All the information and records of a track, which was transmitted to the server, are stored in a mySQL database. After the computation of various features and division into appropriate segments, the database is updated accordingly. By querying the database, the relevant segments including all the required indicators are made available to the AI. This database consists of two tables "tracks" and "track_points" forming a one-to-many relationship and having an actual storage location consumption of about 16 GB for the 245 hours of data.

The table "tracks" supplies the global data, which are: track ID (unique primary key, auto_increment), filename, labeled (boolean), tracking date, date time of import, user ID, the cumulative distance of each label, and the augmentation tags.

The second table "track_points" contains the track ID as a foreign key and a unique point ID as the primary key. As this table has a size of 27 Mio lines so far, BigInt was used for the point ID. All recorded sensor and GPS data, as well as the mentioned features and results of the segmentation, are stored in this table. In order to improve the databases' query performance, indexes of the type B-tree were created using the columns pointID, point_time, longitude, latitude, and combined trackID with segment and segment_fix_size.

3.3.8 Website

In order to distribute the TMD App to test users, a static website was built. After giving a basic insight into the purpose of transport mode detection information is offered on how the recorded data is used by this project. A detailed list of the data recorded by the TMD App can be found and a description regarding privacy settings. Besides a detailed explanation of the TMD-App installation and usage additional benefits such as the CO2 footprint are presented. For users with any more questions, the website contains the project's contact information. In the section "Preliminaries" a download button for the TMD App can be found. The TMD website can be accessed under the following URL: <https://kontor.cg.tuwien.ac.at/tmdlearn/>.

As the implemented Django framework embraces the Model-View-Controller (MVC) pattern, the Hypertext Markup Language (HTML) page is stored as a template in

Django views. The website was built using the Bootstrap framework, which supports responsive web design, mobile-first sites, and high browser compatibility. Of course, in the implementation accessibility aspects are taken care of as well.

Evaluation

The evaluation is focused on the efficiency of the change point segmentation method, which splits the trips into Walking- and Non-Walking segments including the accuracy of the detected change points. Also, the subsequent fixed-size segmentation is checked for correctness. As there is no ground data present a very costly manual acquirement of additional records which meet the requirement for the evaluation was necessary. Although it has only been tested on a small scale it could be confirmed already that the method is very efficient. The evaluation approach described below is demonstrated with a single track (Subsections 4.2.1, 4.2.2) while in Subsection Results 4.2.3 the findings of a total of ten trips consisting of 27 segments and 17 change points are presented.

4.1 Settings

For recording the tracks to be evaluated the TMD App on a smartphone device OnePlus 7TPro equipped with version 12 of Google's Android operating system was used. Like other ubiquitous mobile devices, it features a built-in GPS sensor whose accuracy should be around less than 5 meters. At each change of transportation mode, a timestamp with a precision of one second was noted separately.

During the segmentation step of processing a GPS log, the data set of each trip is split into Walk- and Non-Walk segments based on change point detection in the first step. Next, each of the resulting intervals is divided into fixed-size segments.

4.2 Evaluation Approach

4.2.1 Ground Truth

In Figure 4.1 a track is shown which consists of four alternating Walk and Non-Walk segments (when the transportation mode is "bike" or "car" we are dealing with a Non-Walk

4. EVALUATION

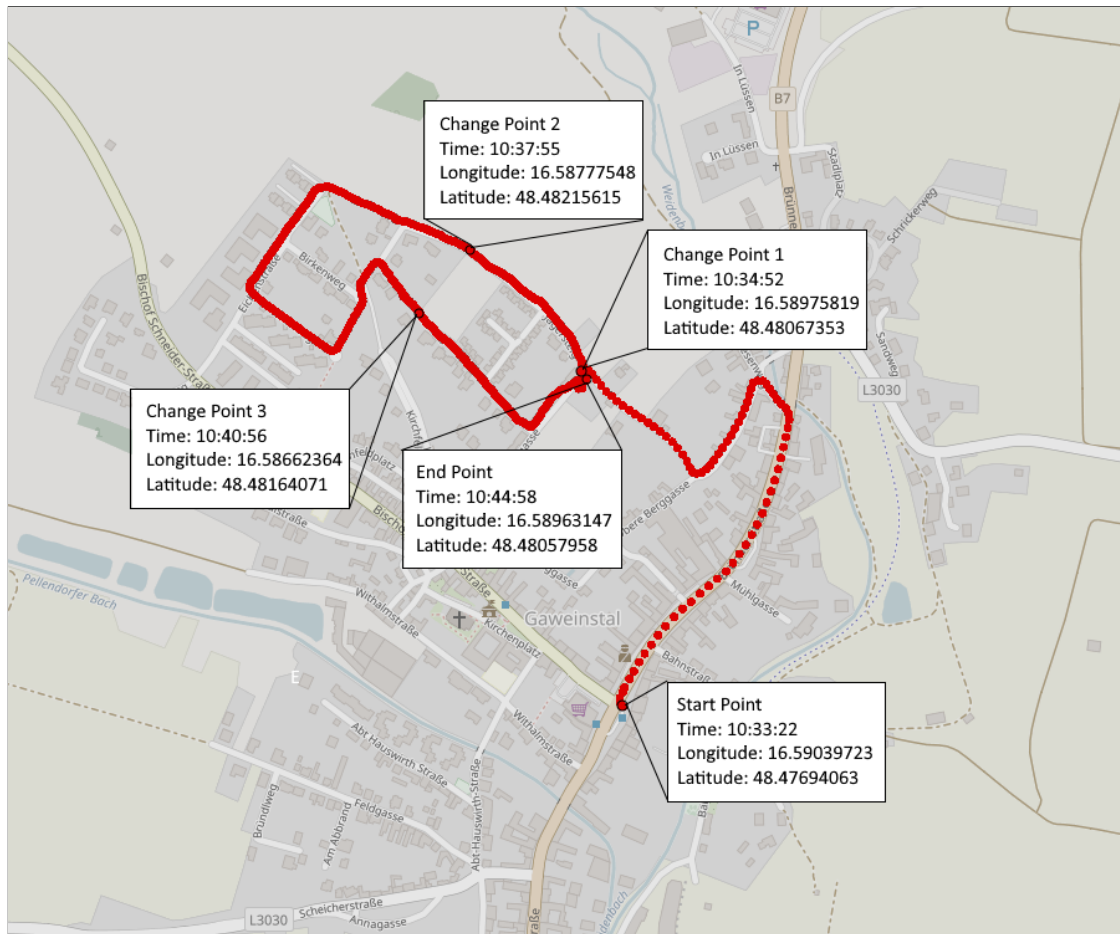
segment). A user drives by car from 10:33:22 followed by walking from 10:34:52 before transferring to biking from 10:37:55 and traveling on foot from 10:40:56 to 10:44:42 at the end.

The labeled ground truth is supposed to be as follows:

"2023-05-09 10:33:22 to 2023-05-09 10:34:52 Non-Walk 0.869 km";
"2023-05-09 10:34:52 to 2023-05-09 0:37:55 Walk 0.241 km";
"2023-05-09 10:37:55 to 2023-05-09 10:40:56 Non-Walk 0.695 km";
"2023-05-09 10:40:56 to 2023-05-09 10:44:58 Walk 0.332 km".

In this case, three change points should be detected.

Figure 4.1: Ground truth data of track ID 518



4.2.2 Criteria

This section includes three aspects that are examined: the accuracy of detected Walk- and Non-Walk segments (Subsection 4.2.2), the accuracy of identified change points (Subsection 4.2.2), and the correctness of the creation of fixed-size segments (Subsection 4.2.2).

Detection of Walk- and Non-Walk mode

We first identify the success of the segmentation into Walk- and Non-Walk segments. This entails defining the Accuracy by Segment (AS) and Accuracy by Distance (AD) as suggested by Y. Zheng et al. [ZCL⁺10]:

$$AS = m/N$$

$$AD = \frac{\sum_{j=0}^m CorrectSegment[j].Distance}{\sum_{i=0}^N Segment[i].Distance}$$

where N is the total number of segments and m is the number of correctly estimated segments.

The SQL statement as shown in Listing 4.1 is applied to the database to obtain the required information of the exemplary track for the previously defined criteria.

Listing 4.1: SQL statement for retrieving values needed for computing AS and AD values

```

1 SET @groupNumber := 0;
2 SET @prev_status := NULL;
3 SELECT MIN(pointID), MAX(pointID),
4     round(sum(distance_diff_m)/1000,3) as km, segment_name
5 FROM (SELECT track_points_511_520.*,
6     @groupNumber := IF(@prev_segment_name != segment_name,
7     @groupNumber + 1, @groupNumber) AS gn,
8     @prev_segment_name := segment_name
9     FROM track_points_511_520
10    WHERE is_gps_point = 1
11    ORDER BY pointID) sq
12 WHERE segment_name IS NOT NULL
13 AND longitude > 0.0 AND latitude > 0.0
14 AND trackID = 518 AND is_gps_point = 1
15 GROUP BY gn, segment_name
16 ORDER BY max(PointID);

```

Comparing the outcome of this query which is presented in Figure 4.2 to the ground truth data of the related track we can recognize that $AS = 1$ is reached in this case.

MIN(pointID)	MAX(pointID)	km	segment_name
2953123	2962223	0.869	NonWalk
2962323	2981523	0.251	Walk
2981623	2998323	0.679	NonWalk
2998423	3022923	0.352	Walk

Figure 4.2: Result of segmentation into Walk- and Non-Walk segments of track ID 518

The result of the AD values in this example can be seen in Figure 4.3.

	Distance (km)	
	Walking	Non-Walking
Ground Truth	0,573	1,564
Assumed Result	0,603	1,548
AD	1,052	0,990

Figure 4.3: The resulting AD-value of track ID 518

Accuracy of Change points

After switching the transportation mode, the first data line is defined as a change point. The accuracy of the derived change points is determined by comparing the distance between the GPS coordinates at the time of occurrence and their equivalent ground truth (whereas for checking the above-mentioned AS- and AD values the calculated distances are independent of the time recorded). A change point counts as a correct conclusion if the resulting difference is less than a given threshold (default: 150 meters, cf. [ZCL⁺10]).

Functioning of uniform duration segments

After the change point segmentation into walking- and non-walking parts, each resulting segment gets further split into segments of the same duration respectively an equal number of data lines of the recorded track. To prove the correctness of the retrieved number of fixed-size segments (SegmentFixSize.Count) the following applies:

$$\frac{\sum_{i=0}^n \lfloor \frac{Segment[i].Duration}{t} \rfloor}{SegmentFixSize.Count} = 1$$

where n is the total number of walk- and non-walk segments and t is the committed duration of the fixed-size segment (default: 60 sec. respectively 6.000 data lines at a recording rate of 100 Hz). Any leftover track points from the first division are eliminated by the floor function and will not be included in any fixed-size segment. The result of another SQL query (please see Figure 4.4) shows the comparison between duration and

the number of fixed-size segments per segment. Looking at Figure 4.4 segment number 2 has a duration of between 3 and 4 minutes and therefore consists of 3 fixed-size segments. The database holds 6.000 data lines for each of these parts.

Figure 4.4: Result of a SQL-Query showing the fixed-size segments of trackID 518

MIN(pointID)	MAX(pointID)	segment	segment_name	minutes	nr_fixed_size_segments
2953423	2962223	1	NonWalk	1.52	1
2962323	2981523	2	Walk	3.22	3
2981623	2998323	3	NonWalk	2.8	2
2998423	3022923	4	Walk	4.1	4

segment	segment_fix_size	Count
2	1	6000
2	2	6000
2	3	6000

4.2.3 Results

The achieved success of dividing the recorded paths of the TMD App into Walk- and Non-Walk segments can be seen in Figure 4.5. As presented here, the accuracy by segment (AS) result demonstrates that each Walk and Non-Walk segment could be predicted correctly. In the Accuracy by Distance (AD) outcome, only minor variations were detected.

Figure 4.5: Result of AS and AD trackID 518-527

Segment Name	Ground Truth		Inferred Results		AS	AD
	Amount of segments	Distance (km)	Amount of segments	Distance (km)		
Walking	14	5,539	14	5,645	1,00	1,02
Non-Walking	13	14,745	13	14,59	1,00	0,99

The distance of Change points between ground truth and those which were identified by the segmentation method is shown in Figure 4.6. As all distance values are clearly less than the threshold all examined Change points are assumed to be predicted correctly.

4. EVALUATION

Figure 4.6: Result of Change points Accuracy applied to track ID 518 - 527

trackID	Change Point	Ground Truth		Inferred Results		Distance (m)
		Longitude	Latitude	Longitude	Latitude	
518	1	16.58975819	48.48067353	16.58975819	48.48067353	0,00
	2	16.58777548	48.48215615	16.58764373	48.48221213	11,54
	3	16.58662364	48.48164071	16.58660366	48.48164819	1,69
519	1	16.59208071	48.48161622	16.59197649	48.48176402	18,14
520	1	16.58976668	48.48067919	16.58973695	48.48073732	6,83
	2	16.58841724	48.47235644	16.58845053	48.47234832	2,62
521	1	16.59100736	48.46208137	16.59098222	48.46207805	1,89
	2	16.59100751	48.46209751	16.59095308	48.46213169	5,53
522	1	16.59340701	48.47681689	16.5935627	48.47683245	11,61
523	1	16.58723996	48.48575492	16.58724492	48.48575758	0,47
	2	16.59044552	48.48334991	16.59046918	48.48330076	5,74
	3	16.59291638	48.47954617	16.59288068	48.47952934	3,23
524	1	16.59409703	48.48103725	16.59416604	48.48097215	8,84
525	1	16.59104502	48.47046319	16.59101572	48.47045149	2,52
526	1	16.58788261	48.47537306	16.58787577	48.47545182	8,77
	2	16.58072638	48.47906504	16.58074381	48.47906399	1,29
527	1	16.59285841	48.48473875	16.5928486	48.48473326	0,95

The evaluation of the retrieved fixed-size segments of the examined tracks (trackID 518-522) led to the following results for the criteria laid down by the above-mentioned equation:

$$\sum_{i=0}^n \lfloor \frac{Segment[i].Duration}{t} \rfloor = 43 \text{ and } SegmentFixSize.Count = 43$$

It therefore follows: $\frac{43}{43} = 1$ is valid.

Conclusions and future work

This paper proposes a client- and server-side implementation which pre-processes raw GPS and sensor data of people's locomotion for further usage by a transport mode inference model. To this end, single-mode segments of recorded trajectories are provided in a fixed size by a database. A particular emphasis was put on the pre-processing steps and segmentation.

Adequate features were calculated using GPS data for a change point-based segmentation leading up to the creation of altering walk and non-walk sections keeping them as long as possible to avoid over-segmentation. By applying this methodology a very high test accuracy in the detection of walking and non-walking sequences could be recognized. Taking these results into consideration by the appliance of further inference models the complexity for the identification of other transportation types can be reduced considerably. Because the fixed-size segments - as required by a CNN - being generated from the result of the change point segmentation are single-mode window-overlapping can be neglected. Thereby, redundancy in the data basis is avoided. Together with the GPS data including several features sensor data recorded in high frequency was stored in the database which enables the use of the desired combination for the applied inference model.

The next step could include data gathering of people's movements not only at a higher scale but also focusing on achieving a reliable quality in labeling the training data as the accuracy in the classification is highly dependent on this. In the future traditional questionnaires could become obsolete by using smartphone data for analyzing the travel behavior of different individuals.

List of Figures

1.1	Modal Split 2022 in Vienna	2
3.1	TMD Architecture Diagram	8
3.2	Main screen	10
3.3	Recorded distance and duration per label	11
3.4	CO2 footprint	12
3.5	Privacy spot	12
3.6	Excerpt of TMD database, table "tracks"	17
3.7	Excerpt of TMD database, table "trackpoints": import of logged values	17
3.8	Speed (unfiltered)	19
3.9	Smoothed speed	19
3.10	Understanding Segmentation	21
3.11	Example of the partition of a track into trips	23
3.12	The result of getting consecutive segment numbers after merging uncertain segments	25
4.1	Ground truth data of track ID 518	32
4.2	Result of segmentation into Walk- and Non-Walk segments of track ID 518	34
4.3	The resulting AD-value of track ID 518	34
4.4	Result of a SQL-Query showing the fixed-size segments of trackID 518	35
4.5	Result of AS and AD trackID 518-527	35
4.6	Result of Change points Accuracy applied to track ID 518 - 527	36

Listings

3.1	SQL statement CO2 emission for travel mode car	13
4.1	SQL statement for retrieving values needed for computing AS and AD values	33

Bibliography

- [Aus20] Statistik Austria. Traffic statistics, 2020. Available at https://www.statistik.at/fileadmin/publications/Verkehrsstatistik_2020.pdf.
- [bÄdL22] Österreichische Länder bzw. Ämter der Landesregierung. Geo Services, 2022. Available at <https://www.geoland.at/site/geoservices.html>.
- [BAT16] Manfred Boltze and Vu Anh Tuan. Approaches to achieve sustainability in traffic management. *Procedia Engineering*, 142:204–211, 01 2016.
- [CJW02] H.D. Cheng, X.H. Jiang, and Jingli Wang. Color image segmentation based on homogram thresholding and region merging. *Pattern Recognition*, 35(2):373–393, 2002.
- [DH18a] Sina Dabiri and Kevin Heaslip. Inferring transportation modes from gps trajectories using a convolutional neural network. *Transportation Research Part C: Emerging Technologies*, 86:360–371, 2018.
- [DH18b] Sina Dabiri and Kevin Heaslip. Inferring transportation modes from gps trajectories using a convolutional neural network. *Transportation research part C: emerging technologies*, 86:360–371, 2018.
- [DLHR19] Sina Dabiri, Chang-Tien Lu, Kevin Heaslip, and Chandan K Reddy. Semi-supervised deep learning approach for transportation mode identification using gps trajectory data. *IEEE Transactions on Knowledge and Data Engineering*, 32(5):1010–1023, 2019.
- [Fou22a] Kotlin Foundation. Kotlin language, 2022. Available at <https://kotlinlang.org/docs/android-overview.html>.
- [Fou22b] Python Software Foundation. Python, 2022. Available at <https://www.python.org/>.
- [FZV20] Paulo Ferreira, Constantin Zavgorodnii, and Luís Veiga. edgetrans - edge transport mode detection. *Pervasive and Mobile Computing*, 69:101268, 2020.
- [Inc22a] Docker Inc. Docker, 2022. Available at <https://www.docker.com/>.

- [Inc22b] Docker Inc. Django REST Framework, 2022. Available at <https://www.django-rest-framework.org/>.
- [Lan23] Jakob Lang. Transport mode detection in vienna using android sensor data and gis information. *TU Vienna, Faculty of Informatics*, 2023.
- [LYL⁺22] Yande Li, Lulan Yu, Jun Liao, Guoxin Su, Hashmi Ammarah, Li Liu, and Shu Wang. A single smartwatch-based segmentation approach in human activity recognition. *Pervasive and Mobile Computing*, 83:101600, 2022.
- [LZZ⁺20] Linchao Li, Jiasong Zhu, Hailong Zhang, Huachun Tan, Bowen Du, and Bin Ran. Coupled application of generative adversarial networks and conventional neural networks for travel mode detection using gps data. *Transportation Research Part A: Policy and Practice*, 136:282–292, 2020.
- [MJ20] Christos Markos and JQ James. Unsupervised deep learning for gps-based transportation mode identification. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6. IEEE, 2020.
- [NCGG10] Theresa Nick, Edmund Coersmeier, Jan Geldmacher, and Juergen Goetze. Classifying means of transportation using mobile sensor data. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6, 2010.
- [NDC⁺14] Paolo Neirotti, Alberto De Marco, Anna Corinna Cagliano, Giulio Mangano, and Francesco Scorrano. Current trends in Smart City initiatives: Some stylised facts. *Cities*, 38:25–36, 2014.
- [RMM16] Suneth Ranasinghe, Fadi Al Machot, and Heinrich C Mayr. A review on applications of activity recognition systems with regard to performance and evaluation. *International Journal of Distributed Sensor Networks*, 12(8):1550147716665520, 2016.
- [SZGH22] Paria Sadeghian, Xiaoyun Zhao, Arman Golshan, and Johan Håkansson. A stepwise methodology for transport mode detection in gps tracking data. *Travel Behaviour and Society*, 26:159–167, 2022.
- [Umw22] Umweltbundesamt. Emissionskennzahlen, 2022. Available at https://www.umweltbundesamt.at/fileadmin/site/themen/mobilitaet/daten/ekz_pkm_tkm_verkehrsmittel.pdf.
- [VCO21] VCOE. Pro Kopf Co2 Austria, 2021. Available at <https://vcoe.at/presse/presseaussendungen/detail/vcoe-oesterreichs-verkehr-hat-zweithoechsten-pro-kopf-co2-ausstoss-der-eu>.
- [Wie14] Stadt Wien. Smart City Wien, 2014. Available at <https://www.wien.gv.at/stadtentwicklung/studien/b008380.html>.

- [Wie21] Stadt Wien. Aktive Mobilität in Wien, 2021. Available at https://blog.stadtentwicklung.wien.gv.at/wp-content/uploads/sites/57/2021/03/Vert_Ausw_Aktiv_Mobili_Endb_21.01.2021.pdf.
- [ZCL⁺10] Yu Zheng, Yukun Chen, Quannan Li, Xing Xie, and Wei-Ying Ma. Understanding transportation modes based on gps data for web applications. *ACM Transactions on the Web (TWEB)*, 4(1):1–36, 2010.