TECHNISCHE UNIVERSITÄT
ILMENAU

Department of Computer Science and Automation
Systems and Software Engineering Group

Master Thesis

Identification of Trajectory Anomalies in Uncertain Spatiotemporal Data

Registration Date:	1 April 2020
Submission Date:	XX August 2020
Supervisor:	Prof. Dr.-Ing. Kai-Uwe Sattler (TU Ilmenau)
Supervisor:	Prof. Igor Anikin (KNRTU-KAI)

Submitted by: Mardanova Aigul
Matriculation Number 62106
aigul.mardanova@tu-ilmenau.de

Abstract

Abstract content

Contents

List of Figures **3**

List of Tables **4**

List of Algorithms **5**

List of Listings **6**

1 Introduction **7**

1.1 Problem Statement 8

1.2 Contribution 9

1.3 Thesis Structure 9

2 Basic Knowledge **11**

2.1 Input Data Sources 11

2.2 Trajectory Definition 12

2.3 Trajectory Anomaly Definition 12

2.3.1 Trajectory Anomalies Classification 13

2.4 Challenges 14

3 Background and State of the Art **17**

3.1 Anomalies Detection Techniques 17

3.2 Clustering Approaches 23

3.3 Distance and Similarity Measures 28

3.4 Related Work 32

4 Framework 35

4.1 Framework Conceptual Model 35

4.1.1 Input Data Description (Nature of Data) 36

4.1.2 Framework Architecture 39

4.1.3 Agglomerative Hierarchical Clustering 40

4.1.4 Trajectories similarity 42

4.1.5 Measuring the Clusters Validity 42

4.1.6 Trajectories Approximation 44

4.2 Framework Implementation 47

4.2.1 Stack of Technologies 47

4.2.2 Input Data Processing 47

4.2.3 Trajectories Approximation using Polynomial Regression 49

4.2.4 Similarity measure calculation 53

4.2.5 Clustering 55

4.2.6 Clusters' modeling 55

5 Evaluation & Results 56

6 Conclusion & Perspectives 57

Bibliography 58

ToDo List I

Appendix II

A. Input trajectories parsing algorithm II

B. Polynomial Regression initiation VII

C. Agglomerative Hierarchical Clustering VIII

Abbreviations

GIS	Geographic Information System
ITS	Intellectual Transport System
ST	Spatio-Temporal (data)
TVS	Traffic Video Surveillance
GPS	Global Positioning System
DTW	Dynamic Time Warping
LCSS	Longest Common SubSequence
FARS	Fatality Analysis Reporting System
ID	Identificator
SVM	Support Vector Machine
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DI	Dunn’s Index

List of Figures

4.1.1 Pravo-Bulachnaya / Puschkina intersection 37
4.1.2 Nesmelova / Kirovskaya Damba intersection 37
4.1.3 Moskovskaya / Galiaskara Kamala intersection 38
4.1.4 Moskovskaya / Parizhskoy Kommunyi intersection 38
4.1.5 Output of a tracking system for video the first intersection 39
4.1.6 Two-phased proposed approach 39
4.1.7 Architecture of an implemented framework 41
4.2.1 Trajectories approximated with 4th-degree polynomial functions . 52

List of Tables

4.2.1 R^2 values for different degrees of polynomials 50
4.2.2 Overview of min, avg and max speeds of vehicles 52

List of Algorithms

1 Description of Agglomerative Hierarchical Clustering 41
2 Description of LCSS distance calculation 43

List of Listings

4.1 Speed calculation 48

4.2 LCSS calculation 53

6.1 Input trajectories parsing algorithm II

6.2 Polynomial Regression initiation VII

6.3 C. Clustering implementation VIII

Chapter 1

Introduction

Nowadays spatiotemporal (ST) data analytics plays an important role in different applications, based on Geographic Information Systems (GIS). Recent advances in GIS and, in particular, in GIS technologies and infrastructure have made cities smarter. And Intellectual Transport Systems (ITS) with urban traffic analysis are one of the most attractive applications in a smart city [1]. Intelligence surveillance in smart cities has rapidly progressed in last decade [2]. More and more roads and public areas are getting equipped with monitoring video cameras, amount of publicly available video data increases further [3]. Automatic analysis in Traffic Video Surveillance (TVS) receives increasingly more attention [4].

Nowadays there are many tasks and applications of urban traffic analysis and, according to [2], tracking vehicles behavior using image processing of videos is one of the promising approaches. One of the main research approaches in urban traffic analysis, which works with data from monitoring video cameras, is mining frequent trajectory patterns from the ST data representing a traffic flow, because extracted trajectories can be afterwards applied to automatic visual surveillance, traffic management, suspicious activity detection, etc. [5][6]. Another important sub-category of traffic analysis, which has become a commended task in many applications in smart cities, is an identification of trajectory anomalies [2]. Anomaly is traditionally described as a data instance that remarkably deviates from the majority of data instances in a data set [7]. In TVS domain an anomalous activity refers to events violating the common rules [4]. Such unusual traffic patterns, which do not conform to expected behavior, reflect abnormal traffic streams on road networks and thus provide useful, important and valuable information [2]. For instance, when a traffic incident or jam happens, traffic flow changes suddenly, and this will be reflected by deviations from the normative activity patterns. That means that recognizing outliers can be useful in detecting traffic incidents. However, in the context of huge amounts of data to be processed, or information overload in other words, manual solutions are infeasible nowadays due to high complexity and high time consumption, and researchers look for automatic or semi-automatic intelligent methodologies to solve these tasks to minimize the required involvement of the human operator [8].

As stated in recent researches in the field of traffic data analysis, it is significant in many applications, including ITS, to take into consideration uncertainty of data. The reasons of data uncertainty can be imprecisions in measurements and inexactitude of observations. In case of acquiring trajectory data from video enforcement cameras data uncertainty can be caused by limitations of used devices [9].

1.1 Problem Statement

As it was mentioned above, ST data analytics plays an important role in everyday life, and the process of extracting useful information from ST data is one of the most significant challenges in traffic data mining. Since ST trajectory data is multi-dimensional and spatiotemporally related, traditional data mining approaches, proposed for static, single and independent data, are inefficient and inappropriate in that case [10].

The main objective of the work in this thesis is to implement a framework for frequent trajectory patterns mining and identification of trajectory outliers in a three dimensional ST trajectory data, extracted from video surveillance cameras. A video from surveillance cameras will be processed in a tracking system, which extracts vehicle trajectories and converts them into vectors containing tracking points. The implemented method needs to be evaluated in terms of accuracy, performance, and an improvement to increase the accuracy of results in context of input data particularities needs to be suggested.

In order to achieve the main objectives, following sub-tasks need to be performed:

- Perform state-of-the-art review of existing approaches and choose a method to implement frequent trajectories extraction and anomalies detection;
- Investigate and suggest an improvement of the chosen algorithm to increase accuracy of results for data from video surveillance cameras;
- Implement a framework with the selected algorithm;
- Perform evaluation of implemented algorithm in terms of performance and accuracy.

In this thesis, we will focus on following types of anomalies:

- Anomalous trajectories with anomalous spatial information. This category covers trajectories with abnormal spatial behavior, such as illegal U-turns on the intersection, double solid line crossing, driving in an opposite direction.
- Anomalous trajectories with anomalous spatiotemporal information. This type corresponds to situations where the spatial information can be considered as a normal, but adding a temporal information converts the trajectory into an abnormal one, for example: moving with an anomalously high or low speed, unexpected, emergency stops.

1.2 Contribution

The main contribution of this work is ...

include or not?

1.3 Thesis Structure

The rest of this thesis work is structured as follows. The whole paper is organized into 6 parts. Chapter 2 introduces the background and terminologies used in the thesis work. Chapter 3 performs the State-of-the-art analysis of existing approaches. Chapter 4 presents the concept of an implemented framework, describes input data structure and input data processing and provides the detailed description of the implementation part. Chapter 5 presents experimental results and evaluation of implemented approach. Chapter 6 gives conclusion and discussions on possible further perspectives.

Chapter 2

Basic Knowledge

This chapter is intended to give a background information, introduce useful definitions and basic concepts of approaches used in following chapters. Input data sources and related challenges will be discussed.

2.1 Input Data Sources

Tasks of frequent trajectories identification and outliers detection can be applied to different data sources, for example: GPS (Global Positioning System) devices and sensor networks, then trajectory data is collected by sensors on moving objects, which periodically transmit information about location over time, or video traffic surveillance cameras. This work will focus on working with latter type of input data sources.

Video data from enforcement cameras is considered as a raw data and is not used directly as an input for implemented system. Raw video processing is done in a stand-alone tracking system. Tracking system takes raw video from enforcement cameras and handles it to perform the objects detection and converting the trajectory into a number of tracking points on images. Tracking points, containing such information as vehicle ID, timestamp, spatial coordinates, are used as an input.

2.2 Trajectory Definition

Trajectories can be described as multi-dimensional sequences containing a temporally ordered list of locations along with any additional information [7]. So, since a trajectory, denoted as τ, represents consecutive positions of a moving target object in temporal domain, in a case of a single-camera surveillance data, it can be defined as:

$$\tau = (x_1, y_1, t_1), (x_2, y_2, t_2), \ldots, (x_n, y_n, t_n), \tag{2.2.1}$$

where (x_i, y_i) denotes the position of the target object in the image at time t_i [5]. According to this, trajectories can be represented as a sequence of 3D points, where 2D object is for geometric coordinates and the third dimension stores the time [11].

Generally, trajectory data is raw and contain only minimum information such as position and time as well as the identifier of the tracking object. This information can be easily augmented by such detailed information as speed, acceleration and direction, since they can be extracted from the initial trajectory data [12].

2.3 Trajectory Anomaly Definition

Twenty-four-hour recording video surveillance cameras produce massive amounts of data about moving objects, and that increases the possibility that along with the normally behaving objects some of the moving objects will demonstrate abnormal behavior. Such exceptional behaviors can also be named as outliers, anomalies, abnormalities, exceptions, novelties or deviants [13][14]. Notwithstanding that no standardized way of deviation characterization exists, in statistics following definition can be found [15]:

> “An outlying observation, or outlier, is one that appears to deviate markedly from other members of the sample in which it occurs”.

Trajectory anomalies can be described as traffic flow patterns, which significantly deviate from some normal behavior pattern or, in other words, inconsistent with the rest of traffic behavioral patterns. Anomalous trajectories are supposed to have great local or global difference with the majority of trajectories in terms of a chosen similarity metric [6].

The process of outlier detection is intended to reveal unusual patterns that drastically differ from majority of samples in order to process them further in an appropriate way [13]. Also anomalous to normality activity patterns ratio should be relatively small in order to be able to distinguish abnormalities from the dominating normal patterns.

2.3.1 Trajectory Anomalies Classification

According to the literature, trajectory anomalies can be categorized as follows [14][16][17]:

- *Point anomaly* - represents the simplest type of anomalies. Corresponds to an individual data instance which is considered as an abnormal one with respect to the rest of data, since it deviates significantly from all other samples in a data set. For example, a non-moving car on a busy road.

- *Contextual anomaly* - corresponds to a data instance which is considered as anomalous in a specific context, but as normal otherwise. It can be also understood as a point anomaly in a neighboring of the data point itself. Contextual anomalies are also referred as conditional anomalies and represent the most common group of categories applicable to spatial and time-series data. For example, trajectories can be classified based on the spatial data (coordinates) in the scope of a time. Examples of a contextual anomaly may be trajectories of a vehicle moving with a much higher speed comparing to others in the same traffic flow or of a vehicle driving in the opposite direction.

- *Collective anomalies* - a set of data instances, cooccurence of which as a group is considered as an anomaly with respect to the whole data set, while each data instance individually does not necessarily represent an anomaly. The given definition can be simplified to a set of neighboring point anomalies or context anomalies. Collective anomalies can only be applied to data sets with a relation between data instances.

The other way of trajectory outliers systematization may be dividing them into following categories according to the properties, which were used to perform classification:

- *Spatial trajectory anomaly* - classification process takes into consideration only spatial information of moving object trajectories, such as position coordinates. Examples of spatial anomalies can be illegal U-turns, double line crossing or moving in an opposite direction.

- *Temporal trajectory anomaly* - corresponds to anomalies detected by analyzing only temporal characteristics of trajectories, such as duration, time of moving. For example, a trajectory with significantly long duration or a trajectory appearing at an anomalous time.

- *Spatiotemporal trajectory anomaly* - can be detected by analyzing spatial and temporal information in aggregate. Examples of ST anomalies can be vehicles moving with a considerably high speed comparing with majority of trajectories. Also such anomalies can be detected in the case of a contraflow traffic systems with a reversing traffic light anomalous trajectories: since for such a line allowed direction changes according to some known or learned schedule, classifier can analyze the trajectory direction together with a temporal information.

In accordance with the second classification, the current work will focus on determining trajectory anomalies of the first and third types (spatial and spatiotemporal trajectory anomalies).

2.4 Challenges

Since ST data differs from other types of data in many aspects, challenges are related to the used data type. A unique quality of it is that ST data instances are not independent and identically distributed, as it is usually assumed to be in many of existing data mining approaches. On the contrary, ST data instances, related to observations made by nearby locations and time, are structurally correlated with each other in context of space and time, and it is important to take into consideration the presence of dependencies among measurements in these dimensions. Consequently, many of the existing data mining approaches are not applicable to ST data, since ignoring the aforementioned characteristics can result in poor accuracy of results. This leads to the necessity of investigating and using different methods for processing such a data to preserve all the relations between information domains [7].

It should be noted that the chosen type of an input source leads to difficulties in processing. Since trajectory data is acquired by video enforcement cameras, the first problem is an uncertainty of a location as a result of limitations in measurement accuracy of the used cameras, resolution and the quality of the received images or frame jitter [3]. Moreover, enforcement cameras are placed at some fixed locations on an intersection, because of that one of the particularities of used data are pose and perspective, which can cause challenges while dealing with input video data [16]. The view angle of the camera in respect to the scene ground plane and distance between the tracked object and camera can affect the performance by decreasing the accuracy of objects detection and tracking: the smaller the angle - the bigger is the problem of determining the center of an object [2][3]. Tracked objects can drive in and out of the camera view, but be still detected while partially visible. This can cause trajectory changes on the borders of the camera scene: displacing and shifting of vehicle trajectories depending on the location of an object in respect to the camera [3]. Quality of the performed trajectory analysis also depends on the input trajectory data, including: quality of used cameras, quality of a tracking system, which converts a video data into a list of trajectories consisting of tracking points.

Moreover, in the current thesis work input data contains trajectories extracted from video cameras without sorting and analyzing, so:

1. input data set can contain both examples of normal and anomalous trajectories;

2. input data does not contain labels.

Aforementioned limitations lead to the necessity to use unsupervised methods to automatically extract normality and abnormality rules from unlabeled data [18].

Chapter 3

Background and State of the Art

This chapter presents state of the art analysis of existing approaches and discuss advantages and disadvantages of existing approaches. Each section is concluded with a summary, in which the chosen approach is specified and a short argumentation is given.

3.1 Anomalies Detection Techniques

According to [14], the task of outliers detection, which is a main objective task of this work, has been an object of interest and studies of research community originating from 19th Century. Nowadays a huge variety of different techniques for solving the task of detecting outliers and abnormalities in video traffic data are presented, and these approaches can be classified in various ways.

For example, data mining techniques in general and anomaly detection techniques as well as clustering approaches, which are one of the ways to solve the task of outliers identification, particularly can be classified as supervised, semi-supervised and unsupervised on the grounds of the manner of labeling the input data] [14][17]:

1. *Supervised*. Input data used for training contains labels for both normal and anomalous instances. As a result the algorithm can build models for both normal and abnormal classes;

2. *Semi-Supervised [14]* or *Weakly-Supervised [5].* Input training data set contains class labels only for normal data instances. Such techniques are more widely used than supervised approaches, since anomalous data instances are usually not predictable and random and it is difficult to provide examples to cover all possible anomalous events;

3. *Unsupervised.* Does not require input data to be labeled neither for normal nor anomalous data. Such algorithms are based on the expectation that normal data instances are significantly more frequent than anomalous ones in the test data set and therefore are not applicable when this assumption is disrupted.

Alternatively, according to surveys done by Chandola in [14], Kumaran in [16] and Malik in [17], anomaly detection techniques can be classification based, nearest-neighbor based, clustering based, statistical and etc. The following offers the short overview of mentioned groups.

Classification based

The main concept of these methods lies in using a classifier which firstly learns to distinguish inliers and outliers and then classifies each input instance [19]. Such techniques consist of training and testing phases. Training, or learning, phase supposes learning a classifier model from a training data set, containing labeled data instances. The learned classifier is then used to classify an input trajectory as normal or anomalous by assigning a class label in a testing phase.

Depending on how testing data instances are labeled, all classification based anomaly detection techniques can be one-class or multi-class. The first type assumes that all training data instances are normal and are labeled as one class. During training phase model learns a discriminative boundary around normal instances, and a trajectory, which is not aligned with the learned normal class description, is considered as an anomalous. Single-class Support Vector Machines (SVMs) is the most commonly used classification based approach, which is applicable to the task of anomalous trajectory detection, as it was proposed by Piciarelli *et al.* [20][21]. However, this approach requires trajectory vectors to be the same length. Since raw trajectory data is usually contains different amount of trajectory points due to different speed of moving objects, it is necessary to preprocess raw trajectories to normalize them to vectors of the same length [21]. Moreover, SVMs become highly time and memory consuming while working with huge amounts of multi-dimensional data [22].

The latter category supposes learning multiple classes during training step and then using a classifier to review the input trajectory for compliance with each learned class. In literature different descriptions of training phase and training data labels are given. According to [14], training data contains only normal data instances with corresponding normal class labels, and during training phase model learns multiple discriminative boundaries around each class of normal instances. A trajectory, which is aligned with none of the learned normal class descriptions, is considered as an anomalous. In other words, an anomalous trajectory will not be accepted by neither of the classifiers. In [16] it is assumed that model is learned using training data containing labels for normal and anomalous classes. Therefore, a classifier can classify an input trajectory as belonging to a normal or anomalous class.

The advantage of two-phased classification based algorithms is a fast testing phase due to precomputed classifier model used to classify each input instance. Also such algorithms can perform well in cases when anomalous data instances form a class or cluster [19]. However, the training step requires accurately labeled training data, which is often not available.

Nearest-neighbor based [14] or Proximity / Distance based [16][19]

Proximity based approaches decide whether a data instance is normal or anomalous based on how close or far is it located with respect to neighbors [16]. Nearest-neighbor and density based approaches are based on the assumption, that «normal data instances have dense neighborhood, while anomalous data instances occur far from their closest neighbors» [14].

In order to be able to compare the surrounding density for an instance under consideration with the density around its local neighbors, a distance (dissimilarity) or similarity measure between two data instances needs to be specified [19]. By virtue of an anomaly score calculation method, techniques can be grouped into two categories: 1) the anomaly score is calculated as a distance of a data instance to its k^th nearest neighbor and 2) to compute the anomaly score the relative density of each data instance is being computed [14].

These approaches has several disadvantages. First of all, in comparison with classification based anomalies detection techniques, the computational complexity of the testing phase is considerably higher, since nearest neighbors are computed by computing the distance for each test data instance with all instances from either testing and training data. In case of multi-dimensional trajectory data, the task of distance computation becomes even more complicated. Moreover, the accuracy of labeling decreases when the main assumption is violated: when normal instances have sparse neighborhood or anomalous instances have dense [14].

Clustering based

Clustering is an efficient approach aimed to group data instances into different classes, called clusters, based on their similarity in such a way, that objects in one cluster are similar to each other and dissimilar to objects in other clusters [10][22]. ST clustering supposes grouping objects on the ground of their spatial and temporal similarity. To compare data instances before grouping them into clusters, similarity or distance between them needs to be measured.

There are three types of clustering based anomalies detection techniques with following assumptions: 1) normal data instances are associated with a cluster, while anomalous data instances are not associated with any cluster, 2) normal data instances are close to the cluster center, while abnormal instances lie far away from the closest cluster center and 3) normal data instances lie in large and dense clusters, while anomalies are associated with sparse clusters or clusters with a small cardinality [14][16]. Techniques of first type can be implemented using one of the clustering methods which do not require every data instance to belong to some cluster, for example DBSCAN [23]. Algorithms from second group consist of two phases: 1) data clustering and 2) calculating an anomaly score for each data instance. Techniques of the latter type require a threshold for cardinality size and/or density of a cluster to be defined to decide whether a cluster refers to normal or anomalous data.

The necessity to compute distance between trajectories in some of the clustering based approaches makes them similar to neighbor based approaches. As it is stated in [14], techniques are different in the way they process instances: in clustering based techniques each instance is evaluated with respect to the corresponding cluster, while in neighbor based techniques each instance is being inspected with respect to its proximate neighborhood. Consequently, the selection of distance computation method plays an important role and affects results and performance significantly.

On the other side, dividing all training data into groups makes clustering based algorithms similar to classification based algorithms. Though in classification based approaches class is assigned based on given labels, while in clustering based approaches classification is not given in advance [19].

One of the main advantages of clustering based techniques is the ability of majority of them to run in an unsupervised manner. For the case of TVS-based trajectory data acquisition the unsupervised learning methods are the most appropriate, because labeling hours of video data is a highly time-consuming task. Also, manual labeling of input data can lead to errors due to human operator intervention.

Moreover, clustering based techniques are adjustable to work with complex data types because of adaptability of clustering algorithms. However, at the same time they are computationally expensive and are used primarily for relatively low dimensional data, highly dependent on the used clustering algorithm and can not effectively deal with situations when anomalies form significant separate cluster groups [14].

Model based [16][19] or statistical [14]

The main concept of model based algorithms is that they represent the data as a set of parameters to create the model of a normal behavior. As an advantage, model based approaches do not ask the user to provide any input parameters, because all the parameter values can be derived from the data. Statistics based approaches can be considered as a subcategory of model based approaches, they are treated as one of the earliest algorithms and can be used as a basis by the various outlier detection techniques [17]. As it is stated in [14], the main idea of statistical approaches is that data instances occurring in high probability regions of a stochastic model assumed to be normal, while data instances from the low probability regions refer to anomalies. So, statistical approaches are based on using statistical stochastic model to fit to the given data and then applying a statistical inference test, also called discordance test, to decide if a data instance is normal or anomalous. It comes from the main concept that «based on results of applied statistical test, anomalies have low probability to be generated from the learned stochastic model» [14].

Statistical techniques in turn can be parametric or non-parametric [17]. In parametric approaches the normal data is supposed to fit the parametric distribution and probability density function with estimated from the given data parameters [16]. One of the advantages of parametric techniques is that the data size does not affect the model: models grow only depending on a model complexity. However, the necessity to fit the data into some preselected distribution model complexifies and limits the application of such approaches: it is difficult to fit the data to one distribution. In this case it is possible to use a multiple-distribution model to match some clusters of the data with particular distributions [19]. One of the most known examples of parametric methods is Regression Method [17].

By contrast to this, non-parametric approaches are based on using non-parametric statistical models with structures, which are not defined in advance: the given data is used to determine the structure dynamically. Such approaches do not suppose making assumptions about the statistical distribution of the data [17].

Since statistical approaches are based on fitting a statistical model, the choice of it significantly affects results, computational complexity and performance. Nevertheless, the main assumption of statistical approaches that the data comes from a particular distribution can not be always satisfied, specifically for the case of a multi-dimensional data [14].

Summary

Based on the given description of different approaches and their advantages and disadvantages, it was decided to focus on clustering based anomalies detection approaches for several reasons:

1) they can work in an unsupervised mode without a human intervention and do not require the input data to contain labels,

2) input data is allowed to contain anomalous trajectories,

3) clustering method can be easily applied to such a multi-dimensional data as trajectories by defining a suitable similarity measure.

That means that a clustering method and a similarity measure need to be specified.

3.2 Clustering Approaches

Clustering is a highly researched form of data mining, and huge variety of clustering methods has already been proposed in literature [10]. State-of-the-art analysis of related research papers revealed that all traditional clustering approaches are usually categorized into five types: partitioning, hierarchical, density-based, model-based and grid-based methods [5][10]. Next paragraphs will briefly discuss each of the categories with highlighting main assumptions and concepts.

Partitioning, or Partition-based, methods

Such methods are based on partitioning the trajectories data set randomly and then regrouping clusters by reassigning objects from one partition to another to minimize the objective function. They require the predefined parameter, usually denoted as k, which determines the amount of final clusters, or partitions, to be created. The main requirement is that number of partitions must be smaller than number of initial data points, since each partition forms a cluster, that means that it must be non-empty and contain at least one data instance, and each data instance must be included into exactly one cluster.

One of the most well-known examples of partitioning clustering algorithms is a K-Means algorithm, where firstly k cluster centers are initialized randomly and then data points are iteratively reassigned to the closest clustering center based on the discrepancy to minimize the clustering error [24]. The clustering error is defined as the sum of the squared Euclidean distances between each data set point and the corresponding cluster center [25]. The process is stopped when there are no more changes in clustering centers.

The disadvantages of the traditional K-Means clustering method are inability to form clusters of arbitrary form, dependence on initial random cluster centers initialization and high memory consumption [10]. Also finding an appropriate partitioning technique is a challenging task.

Hierarchical methods

In hierarchical based methods the given data set is decomposed into multiple levels to organize a hierarchical tree of clusters. The resulting hierarchical structure can be depicted as a tree [24].

There are two different ways of hierarchical decomposition: 1) the bottom-up (combining) and 2) the top-down (split, divisive) decomposition. They refer to agglomerative and divisive (split) clustering approaches respectively [26].

Agglomerative hierarchical clustering algorithms start by assigning each data instance to a distinct singleton-cluster, so the number of initial clusters is equal to the exact amount of data instances in input data, and then continue uniting clusters based on theirs similarity until all the initial clusters are merged into one single cluster or into predefined amount of clusters [27]. This is done by repeatedly executing following two steps: 1) identifying the two closest clusters and then 2) merging these two clusters [26]. Proximity matrix is used to store similarity measurements between clusters and is being updated on each step by computing distances between the new cluster and the other clusters.

The divisive hierarchical clustering algorithms work in a reverse manner: initially all data instances belong to one cluster and then step by step clusters split into smaller clusters until all of them become singleton clusters or until satisfying some predefined end condition.

Hierarchical clustering is supposed to be simple, but it is necessary to choose between agglomerative and split methods. Divisive clustering is more expensive in computation, therefore, it is less common than agglomerative approaches. Irreversibility of both splitting or uniting processes in traditional hierarchical clustering algorithms is also a particularity of such algorithms [10].

Since approach includes clusters joining, a significant task of agglomerative clustering algorithms is defining and computing the similarity or distance between clusters. This similarity can also be referred to as an inter-cluster or between-cluster distance. For the case of single-trajectory clusters the similarity between them is simplified and is equal to the similarity between respective trajectories. For multiple-trajectory clusters the similarity is computed according to a chosen linkage method. In literature following linkage methods are given as mostly common: single link, complete link, average link [24][28]. The choice of the linkage method depends on the application domain [26]. In the case of the single link distance between two clusters is defined as the minimum distance between two trajectories in these clusters, that means that the similarity between of two clusters is determined by two closest trajectories. The complete link linkage method implies taking the maximum distance between two trajectories in two clusters as an inter-cluster distance, so it is defined using the farthest distance of trajectory pairs. The average link supposes calculating averaged paired distance between all trajectory pairs in these two clusters.

A convenience of agglomerative hierarchical clustering approaches is that they do not require the number of resulting clusters to be predefined, so they are appropriate for clustering vehicle trajectories, because number of clusters of normal or anomalous trajectories is not known in advance. However, the most well-known disadvantage of hierarchical clustering algorithms is that they are not robust and can suffer from noise and anomalies.

Density-based methods

In comparison with the partitioning and hierarchical clustering approaches, density-based methods objects inspect similarity based on the density of the data [22]. The area is being added to the nearest cluster, while density of the points in the area remains greater than the predefined threshold [10]. Clusters form dense regions of objects and they are separated by sparse regions with low density.

The main advantage of density-based clustering approaches is that they are able to form clusters of arbitrary forms, extend beyond spherical [10]. Also they are appropriate for clustering huge data sets of trajectories in an unsupervised manner and do not require the amount of clusters to be known in advance [5][22]. However, the results quality highly dependent on the amount of trajectories in training data set, available for analysis.

The most well-known and commonly used density-based algorithm is a DB-SCAN, proposed by M. Ester *et al.* in [23]. According to it, input data points are categorized as follows: core data, density-reachable data and outliers based on parameters ε, *minPts* and the density threshold. Neighbor parameter ε and *minPts* specify the maximum remoteness and minimum amount of satisfying points while choosing the core points: at least *minPts* points must be present within distance ε from the core point, these points are marked as directly reachable from the chosen core point. Aforementioned parameters need to be predefined by the user, but it is difficult to determine them correctly. Each cluster must contain at least one core point. Points are denoted as anomalous if they are not reachable from any of the other points.

Shrinkage-based or Grid-based methods

The main idea of grid-based algorithms lies in applying a multi-resolution grid data structure: the data space is quantized into a finite number of cells (units) that form a multi-resolutional grid structure. Each cell stores summary information about data objects within its subspace [22]. Since clustering operations are performed on the created grid, and also important trajectories characteristics can be computed in each of the spatial grid cells, the quality of data compression influences the quality of results significantly [7]. Density of closely located dense cells can help to determine clusters. A trajectory can be considered as an anomalous if it differs from the expected trajectory in a number of covered grid cells [22].

The main advantage of grid-based clustering algorithms is an improved performance: increased processing speed and processing time becomes independent on the size of the data set, only the number of cells in each dimension in the quantized space affects the processing time [10].

Model-based methods

In comparison with the above methods, which analyze distance among data objects, in model-based approaches data is supposed to be generated by a mixture of probability distributions, where each component of mixture represents a cluster. So a mathematical model is assigned to each cluster, and then method attempts to find the best fitting data for the chosen model. In this way such methods seek to increase the adaptability between given data and some statistical models [10][22]. The idea of model-based algorithms is that in order to locate clusters they describe the spatial distribution of the input data points by building density functions. The model-based approaches are typically used in feature-specific clustering and depend on the selected features and model [5].

It is emphasized, that model-based approaches show good performance while working with complex data types. This category usually includes statistical and neural network methods [10].

Graph-based methods [7]

Another category of clustering methods in application to vehicle trajectories data. Liu *et al.* in [29] presented a graph-based approach to solve the problem of detection of outliers in traffic data streams. A graph structure was used to store the traffic: nodes represent regions while edge weights depict the traffic flow. Edge anomalies in the graph denote the traffic abnormalities, and causal outlier tree can then be used to further analyze these outliers to find causal interactions.

Another higher-level classification of clustering methods can consist of only two sub-classes on the ground of properties of generated clusters: hierarchical and partitioning approaches [24]. Hierarchical algorithms group objects into clusters from singleton cluster to cluster containing all data instances or in a reverse direction. While partitioning clustering algorithms divide given data set into a predefined number of clusters in a single-layer structure.

In order to perform clustering, the similarity between two trajectories needs to be defined. Different existing distance measures will be reviewed in following paragraphs.

Summary

Based on the given description of clustering approaches, their limitations, advantages and disadvantages, it was decided to focus on a hierarchical clustering approach, more specifically on an agglomerative hierarchical clustering, because it can deal with limitations of a given input data, that are: absence of input labels, unknown number of resulting clusters, presence of both normal and anomalous trajectories in input data.

3.3 Distance and Similarity Measures

As it was mentioned before, clustering based approaches require a similarity measure to be defined between two trajectories. Apart from that, distance and similarity measures are also used to compare a trajectory with a cluster or a pair of clusters between each other. A similarity measure highly dependent on the format of a trajectory. A trajectory data, represented as a multidimensional data, can contain quantitative or qualitative features, continuous or binary. In such a classification, distance measure functions are more appropriate to work with continuous features, while similarity measures – with qualitative features [24]. Input trajectory-vectors in this work contain spatial information along with temporal, which can be termed as qualitative continuous data. That means that distance measure functions are more appropriate in this case. Moreover, distance and similarity functions can be classified as 1) working with raw representations of trajectories without any preprocessing steps and 2) working with preprocessed trajectories representations. Preprocessing can include unifying the length of trajectories or reducing the dimensionality of trajectory-vectors [28].

Some of the most known and widely used traditional similarity measures are following: Euclidean distance, Fréchet Distance, DTW, LCSS.

Euclidean distance

Euclidean distance between two trajectory vectors is calculated as a sum of squared differences of corresponding spatial coordinates [18]:

$$d_{ij} = ||T_i - T_j||_E = \sqrt{\sum_{k=1}^{m}((t^k_{i_x} - t^k_{i_x})^2 + (t^k_{i_y} - t^k_{j_y})^2)}, \tag{3.3.1}$$

where both trajectories consist of *m* tracking points and are represented by two-dimensional vectors $T_i = \{t_i^1, t_i^2, \ldots, t_i^m\}$ and $T_j = \{t_j^1, t_j^2, \ldots, t_j^m\}$. Tuples $(t^k_{i_x}, t^k_{i_y})$ represent spatial coordinates for a *k*-th tracking point of *i*-th trajectory from a data set.

However, Euclidean distance works only with trajectories with equal number of tracking points. Since usually vehicles move with different speed and behavior, trajectory length is always different and that means that raw trajectories need to be preprocessed and reduced to the same size [28]. Also, traditional Euclidean distance requires two-dimensional data, meaning that it is not able to process temporal information, and is dependent on the trajectory direction: the reversed direction can cause incorrect distance measurement, that in its turn leads to errors in clustering. Also, it fails while working with trajectories moving in a similar way but with different speeds and in the case of different sampling rates [30].

Fréchet Distance

Fréchet Distance is based on Euclidean distance. It considers the positional and sequential relationship of trajectory points while calculating the similarity. The main idea of this approach is computing Euclidean distance for each pair of points from two trajectories and then designating the maximum Euclidean distance as a Fréchet Distance between them [10][31]. However, since only the maximum among distance is considered, the approach is sensitive to the presence of outliers.

DTW

Dynamic Time Warping (DTW) is one of the algorithms for measuring the similarity between two temporal time series sequences, which may vary in speed. The objective of time series comparison methods is to produce a distance metric between them two. DTW method aims to find an alignment between time-dependent sequences, such as trajectories, and is able to process trajectories of different lengths [10].

According to [10], DTW distance is calculated as follows (Formula 3.3.2):

$$D_D(T_i, T_j) = \begin{cases} 0 & m = n = 0 \\ \infty & m = 0 \text{ or } n = 0 \\ dist(a_i^k, b_j^k) + min \begin{cases} D_D(Rest(T_i), Rest(T_j)) \\ D_D(Rest(T_i), T_j) \\ D_D(T_i, Rest(T_j)) \end{cases} & \text{others} \end{cases} \tag{3.3.2}$$

where $D_D(T_i, T_j)$ refers to DTW distance between two trajectory segments with lengths m and n, $dist(a_i, b_j)$ means the Euclidean Distance between two trajectory points. Function $Rest(T_i)$ takes the remaining part of a trajectory after excluding the point a_i. It can be seen, that in case of zero-length trajectories the DTW distance is equal to 0, for the case then only one of two trajectories is non-empty, the distance between them is considered to be infinite. For two non-empty trajectories, the minimum distance between them is calculated in a recursive way.

Though the important advantage of the DTW method is its ability to process trajectory vectors of distinct lengths, DTW distance is not robust to noise and requires trajectory points to be continuous. Also DTW distance computation is highly time consuming and complex due to necessity to compare distances between each pair of trajectories.

LCSS

Longest Common SubSequence (LCSS) distance tries to match two trajectory sequences based on the longest common sub-sequence between them. The LCSS algorithm works with discrete values and calculates the largest number of equivalent points between the two trajectories. The task of finding the longest common sub-sequence is usually solved recursively [10]: possible translations, or shiftings, are calculated in each dimension and used to provide the maximum LCSS [32]. The basic idea of an LCSS distance is that it allows two trajectories to stretch. In comparison with DTW and Euclidean distances, LCSS enables some elements to remain unmatched [33] and, in comparison with DTW, LCSS is more robust against presence of outliers [34].

The LCSS distance is calculated according to the Formula 3.3.3 [28]:

$$D_{LCSS}(T_1, T_2) = 1 - \frac{LCSS_{\delta,\epsilon}(T_1, T_2)}{min(m, n)} \tag{3.3.3}$$

where m and n are lengths of trajectories T_1 and T_2 respectively. $LCSS_{\delta,\epsilon}(T_1, T_2)/min(m, n)$ can be also referred to as an LCSS similarity and takes value between 0 and 1.

The $LCSS_{\delta,\epsilon}(T_1, T_2)$, the longest common sub-sequence between trajectories, represents the number of matched trajectory points between trajectories T_1 and T_2 and is defined as follows (Formula 3.3.4):

$$LCSS_{\delta,\epsilon}(T_1, T_2) = \begin{cases} 0 & \text{if } m = 0 \text{ or } n = 0 \\ & (\text{if } |t_{1_{x,m}} - t_{2_{x,n}}| < \epsilon \\ 1 + LCSS_{\delta,\epsilon}(Head(T_1), Head(T_2)) & \text{and } |t_{1_{y,m}} - t_{2_{y,n}}| < \epsilon \\ & \text{and } |m - n| \leq \delta) \\ max\begin{cases} LCSS_{\delta,\epsilon}(Head(T_1), T_2) \\ LCSS_{\delta,\epsilon}(T_1, Head(T_2)) \end{cases} & \text{otherwise} \end{cases} \tag{3.3.4}$$

As it can be seen, LCSS calculation depends on two constant parameters: δ (point spacing [32]) and ϵ (point distance [32] or matching threshold [33]):

- parameter δ defines the maximum remoteness in terms of time between two trajectory points in which we can look to match a given point from one trajectory with another. Also can be defined as a value representing the maximum index difference between two input trajectories allowed in calculation [32].
- constant ϵ defines the size of proximity to look for matches in terms of spatial information. According to [32] it is a floating point number which represents the maximum allowed distance between trajectory points in each dimension to consider them as equivalent: difference between X- and Y-coordinates less than ϵ value means that points are relatively close to each other and can be considered as equivalent. LCSS distance is increased by 1 in this case.

Parameters δ and ϵ affect results significantly, therefore, the task of choosing the optimal values for them is challenging and important [28][30]. The $Head(T)$ function is defined to return the first $M - 1$ points from the trajectory T, representing the trajectory with the last trajectory point removed. According to implementation given in [32] the LCSS computation, based on a dynamic programming approach, has a complexity of $O((m+k)\delta)$. However, the algorithm requires predefined constant δ and ε parameter values as an input to a method. Also, due to the recursive way of computations, LCSS has a high computational cost [35].

Summary

The LCSS distance is the most appropriate in this work, since it allows the trajectories to contain noise, have different length, objects speed and sampling rates (local time shifts in trajectories) [28]. Moreover, among the aforementioned methods, the LCSS distance is the most robust approach against noises.

3.4 Related Work

The aforementioned objective has been investigated and solved in numerous works using different methods. Since in fact normal events are common and dominate the data, and abnormal events are rare and difficult to describe explicitly, many approaches are based on an unsupervised clustering of trajectories. For this thesis work the approach proposed by Ghrab, Fendri, Hammami in [28] was chosen as a basis. It is focused on detection of abnormalities based on a trajectories clustering.

The proposed approach can be described as a two-phase approach with offline clustering to extract frequent trajectories and an online classification of an input trajectory to label it as a normal or anomalous.

The clustering is done in an unsupervised manner using an agglomerative hierarchical clustering algorithm operating on a distance matrix between trajectories. To perform clustering, the LCSS distance is used as a similarity measure. The formulas and description of the LCSS distance are given in the previous section (Formulas 3.3.3 – 3.3.4).

Advantages

One of the advantages of the proposed method is that the chosen similarity measure does not require the trajectories to be of the same length, so that the preprocessing of the trajectories, which is a high complexity process, can be avoided. Moreover, the training data is allowed to contain normal trajectories as well as anomalous: algorithm will extract both normal and anomalous clusters. Dense clusters will represent normal trajectories classes, sparse clusters – anomalous trajectories classes.

Disadvantages

However, the disadvantage of the proposed method is that LCSS distance does not take into consideration such problems of video surveillance as a view perspective and a position of a moving object.

Summary

This thesis work will be intended to investigate an opportunity of increasing the accuracy of results by making epsilon and sigma parameters, which are used to calculate the sigma, adaptable and dependent on the perspective and a distance from the camera. This includes:

- exploring a functional dependency between epsilon and sigma parameters and a distance from the camera,
- evaluating algorithm with different values.

Chapter 4

Framework

In this chapter the description of the framework development is given. In first sections the conceptual model and the architecture of the solution are discussed. Further sections provide the details of the solution implementation.

4.1 Framework Conceptual Model

The main objective of current thesis work is analyzing ST trajectory data extracted from videos from surveillance cameras and solving the task of identifying outliers. To solve this task the clustering approach was chosen as a basis: firstly trajectory data is used as a training data to define clusters and model them; clusters are denoted as a normal or abnormal ones based on their density; then classifier marks an input trajectory as a normal or anomalous one.

The contribution of the work is in making an attempt to solve the problem coming from data uncertainty and increase the results accuracy by adapting the algorithm of measuring trajectories similarity: take into consideration the position of moving objects in respect to the camera. For this purpose a framework covering mentioned tasks was implemented with an ability to extract frequent trajectories and then detect anomalous trajectories.

The basic workflow of the framework consist of processing the input data, performing trajectories approximation using a polynomial regression, calculating the similarity matrix between trajectories, then clustering the trajectories and modeling the extracted clusters, identifying normal and anomalous clusters based on their density, visualization of modeled clusters, and finally taking an input trajectory and classifying it as a normal or abnormal one according to the built clusters' models.

4.1.1 Input Data Description (Nature of Data)

According to the research done by the US Department of Transportation based on data of Fatality Analysis Reporting System (FARS) and National Automotive Sampling System, nearly 40 percents of all the reported in 2008 year crashes were road intersection related [36]. Consequently, cross-road transport activity analysis is significantly important nowadays in context of safety, and identifying unsafe vehicular trajectories, which violate traffic rules, may be one of the steps towards improving the statistics.

In the presented work video from enforcement cameras is used for training and testing. Test videos are captured using the Intellectual Transportation Systems implemented on four different Kazan crossroads:

1. An intersection of Pravo-Bulachnaya and Puschkina streets (Figure 4.1.1).
2. An intersection of Nesmelova and Kirovskaya Damba streets (Figure 4.1.2).
3. An intersection of Moskovskaya and Galiaskara Kamala streets (Figure 4.1.3).
4. An intersection of Moskovskaya and Parizhskoy Kommunyi streets (Figure 4.1.4).

Each crossroad corresponds to a 4-way intersection and is equipped with a single monitoring camera. Sample pictures from surveillance cameras are given below on Figures 4.1.1 – 4.1.4.

Input data files contain 624, 211, 231, 237 vehicular trajectories for the each of the aforementioned intersections respectively.

By a trajectory anomaly we understand vehicle trajectories through the crossroad, which remarkably differ from majority of common, known trajectories. For

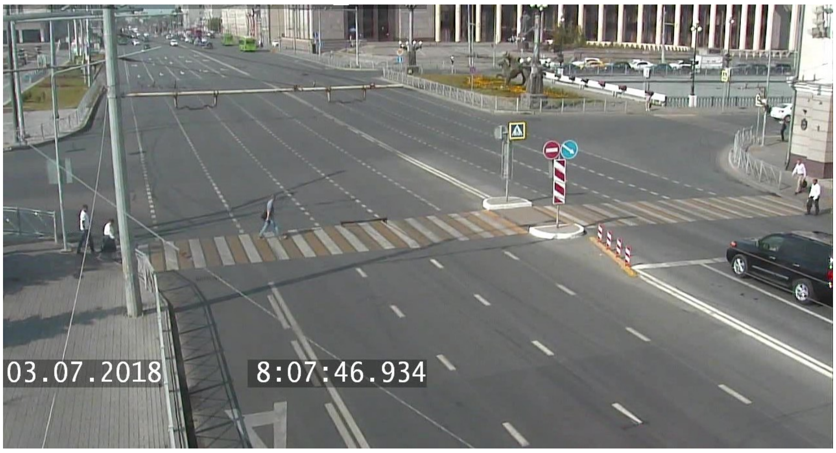

Figure 4.1.1: Pravo-Bulachnaya / Puschkina intersection

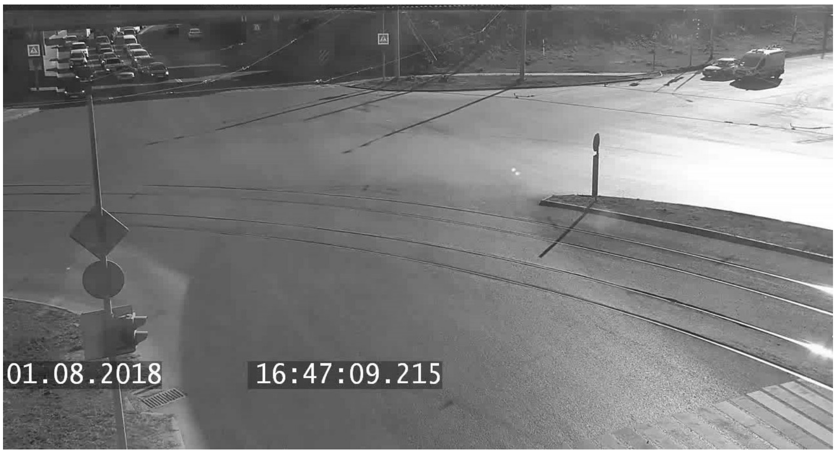

Figure 4.1.2: Nesmelova / Kirovskaya Damba intersection

example, if no turning to the right from the left line is allowed, such a behavior will be unknown and such a trajectory must be considered as an anomaly.

Input data file structure

Tracking system, as it was described before, handles video from enforcement cameras and prepare it for further analysis: converts video stream into a set of vectors with tracking points on images (Figure 4.1.5).

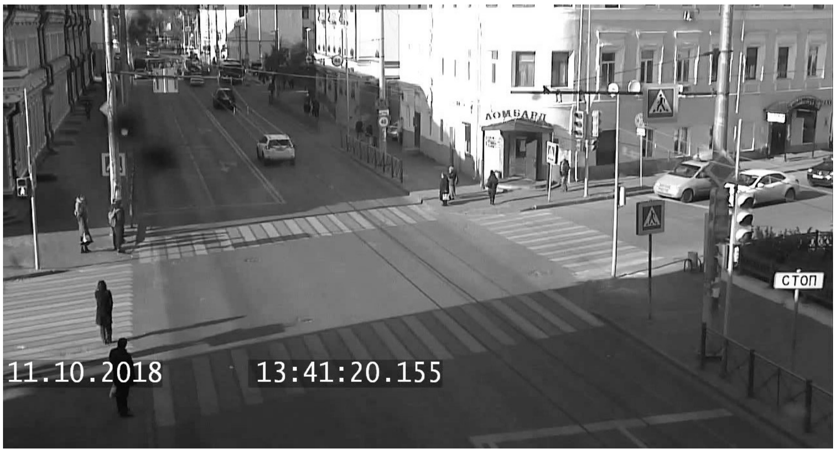

Figure 4.1.3: Moskovskaya / Galiaskara Kamala intersection

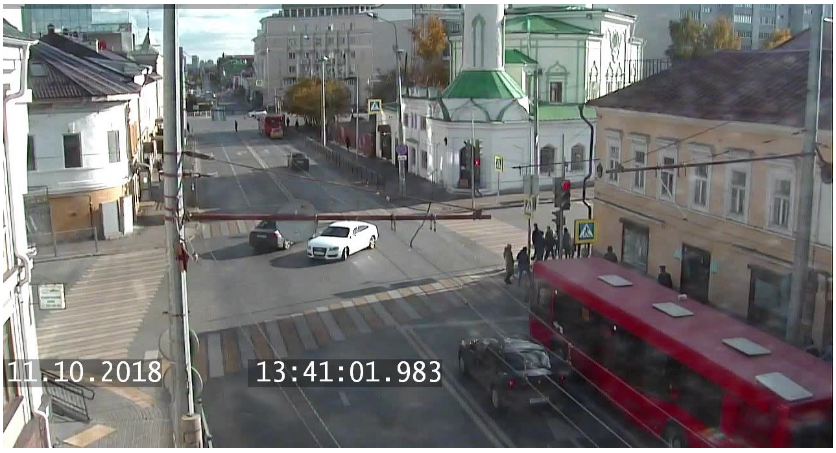

Figure 4.1.4: Moskovskaya / Parizhskoy Kommunyi intersection

Input data files have the following structure:

$$[[[(x_1^1, y_1^1), ..., (x_1^n, y_1^n)], [t_1, ...t_n]], [[(x_2^1, y_2^1), ..., (x_2^m, y_2^m)], [t_1, ...t_m]], ...] \quad (4.1.1)$$

As it can be seen from the input data file structure, each trajectory is represented by a two-element array, where first array stores coordinates as an array of two-tuples (x_i^j, y_i^j) and second array contains timestamps for each spatial point in the corresponding order (t_i). The extracted x- and y-coordinates correspond to pixels on input images. In Formula 4.1.1 the lower index of the spatial coordinates indicates the ordering number of a trajectory, while the upper index indicates

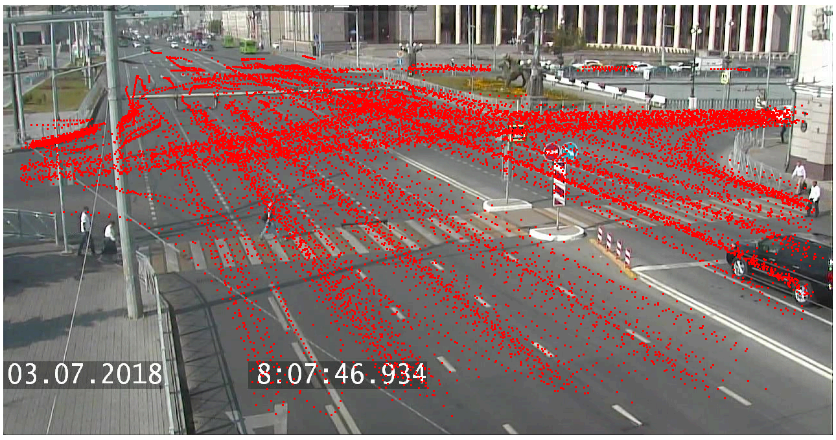

Figure 4.1.5: Output of a tracking system for video the first intersection

the ordering number of a tracking point. The outer array refers to the array of trajectories.

4.1.2 Framework Architecture

The architecture of the framework is based on an already discussed related work [28] and consists of two phases (4.1.6):

- *offline* to perform clustering and extract frequent trajectories, and
- *online* to classify the new trajectory as a normal or abnormal one.

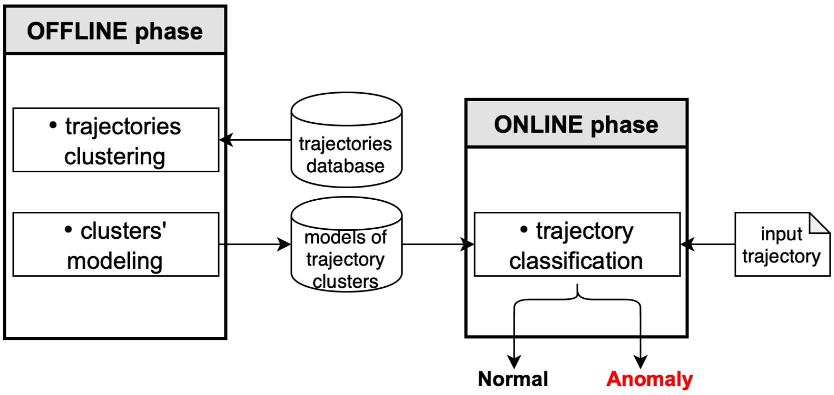

Figure 4.1.6: Two-phased proposed approach

The implemented framework consists of several modules which are responsible for performing particular steps of the aforementioned workflow (Figure 4.1.7):

- *entity* – contains entity-classes for $Trajectory$, $TrajectoryPoint$, $Cluster$ and etc. objects;
- *parsing* – reading a 'txt'-file with input trajectories and parsing them to create $TrajectoryPoint$ and $Trajectory$ objects;
- *csv* – contains logic for reading and writing from/into 'csv'-files, is used to save calculated LCSS measures and load to proceed with clustering;
- *approximation* – performs an approximation of trajectories using a Polynomial regression;
- *visualization* – is responsible for visualization and saving the results, contains methods to read, edit, save $BufferedImage$'s;
- *clustering* – consists of a $Clustering$ class which contains methods to compute LCSS metric values, perform clustering of $Trajectory$ objects and create $Cluster$ objects;
- *exception* – contains exceptional classes hypothetically thrown in the framework (e.g. $TrajectoryParserException$);
- *misc* – contains utility classes needed to store constant values and basic methods.

Add some text about architecture, how should it work

4.1.3 Agglomerative Hierarchical Clustering

Clustering is done using an unsupervised agglomerative hierarchical clustering approach. The description of this approach is given in Algorithm 1 [28].

As it was already mentioned, agglomerative hierarchical clustering methods suppose clusters joining, which requires the inter-cluster distance measure to be defined. In [28] authors have performed evaluation of different linkage methods,

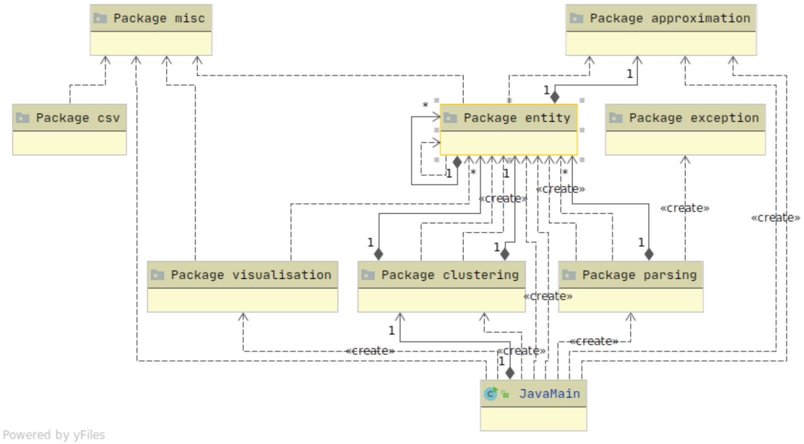

Figure 4.1.7: Architecture of an implemented framework

Algorithm 1: Description of Agglomerative Hierarchical Clustering

```
Input: A Database of Trajectories: trajectories
Output: Clusters of Trajectories: clusters
Initialization:
- initialize the clusters with one trajectory in each cluster
Clusters merging:
while number of clusters is greater than 1 do
    - calculate similarity matrix D between pairs of clusters based on
      single linkage approach using LCSS similarity measure;
    - find the smallest distance between clusters in D;
    - merge two clusters with the corresponding smallest distance into a
      single cluster;
    - remove two merged clusters;
end
```

including single link, complete link and average link. According to the performed tests, the single link method showed the best results and, in view of this, will be used as a linkage method in the current work.

Single link linkage method considers a minimum distance between two trajectories as an inter-cluster distance and can be summed up as [28]:

$$D_{min}(C_i, C_j) = \min_{T_1 \in C_i, T_2 \in C_j} D_{LCSS}(T_1, T_2), \tag{4.1.2}$$

where (C_i, C_j) denote two clusters and (T_1, T_2) correspond to two trajectories from two clusters respectively.

4.1.4 Trajectories similarity

It was mentioned before, that LCSS distance is used as a distance measure between trajectories to perform clustering. LCSS distance implies computing the Longest Common SubSequence between two input trajectories using two parameter values: δ and ε.

Traditionally δ and ε parameters are constant and defined in advance. However, in the developed framework in order to handle uncertainty of trajectory data coming from different position in respect to camera adaptive values of parameters are implemented. Parameters are functionally dependent on position of a moving object on a scene in respect to the camera.

While considering the visualization of trajectories on sample images taken from the cameras, following can be deduced: since the bottom part of the image represent the region located closer to surveillance camera, moving objects on the upper part of the image are more distant from the camera and as a result are more densely located in respect to representation of each other on the image. ε is responsible for the threshold controlling spatial remoteness of trajectory points while computing similarity distance. Consequently, it must be adapted to the remoteness and decrease as a trajectory point gets farther from the camera.

δ behaviour?

LCSS calculation is described in Algorithm 2.

include δ and ε as an input or calculate inside?

4.1.5 Measuring the Clusters Validity

Since the goal of the current work is finding an optimal adaptive parameter values for similarity measure computation, it is necessary to analyze and compare the results after performing clustering.

According to [37] cluster validity measures can be classified as follows:

Algorithm 2: Description of LCSS distance calculation

```
Input: First trajectory: t1,
       Second trajectory: t2,
       Temporal remoteness threshold: $\delta$,
       Spatial remoteness threshold: $\varepsilon$
Output: LCSS distance for two trajectories
begin
    // Initialization
      - calculate length of t1;
    - calculate length of t2;
    // LCSS similarity calculation
      if t1 or t2 is empty then
          return 0;
    else
        if difference between X-coordinates < $\varepsilon$
          AND difference between Y-coordinates < $\varepsilon$
          AND difference between trajectory lengths < $\delta$ then
            - increase LCSS by 1;
            - call recursive for trajectories excluding last points;
        else
            - calculate LCSS for first trajectory and second trajectory
              excluding last point;
            - calculate LCSS for first trajectory excluding last point and
              second trajectory;
            - take maximum between these LCSS values;
        end
    end
    // LCSS distance calculation
      LCSS distance = 1 - LCSS similarity / minimum(input lengths)
end
```

- **Internal cluster validation** – the result of performed clustering is being evaluated based on the input data clustered. It is based on an internal information and does not include references to external information.

- **External cluster validation** – evaluation of clustering results is performed in accordance with externally known results, e.g. given class labels. Such validation is not appropriate for unsupervised clustering then no input labels are provided.

- **Relative cluster validation** – evaluation of the clustering results is done by running the same algorithm using different input parameters, such as number of clusters, etc..

At the same time clustering is primarily an unsupervised data mining technique and the input data does not contain data labels. That leads to the necessity to test the resulting clusters in an unsupervised manner.

One of the most widely used and known measures for evaluating clustering algorithms is a Dunn's Validity Index (DI), which was introduced by J. C. Dunn in 1974 in [38]. It is an internal evaluation metric which is intended to identify compact clusters with a small variance between cluster members which are well-separated between each other, meaning clusters are sufficiently distant from surrounding clusters in comparison with inter-cluster variance [39]. Dunn's index is calculated as the ratio between the minimum inter-cluster distance d_{min} to the maximum intra-cluster diameter d_{max} and for k number of clusters can be defined as follows (Formula 4.1.3) [40]:

$$DI = \frac{d_{min}}{d_{max}} = \frac{\min\limits_{\substack{1 \leq i \leq k \\ i+1 \leq j \leq k}} dist(c_i, c_j)}{\max\limits_{1 \leq l \leq k} diam(c_l)}, \tag{4.1.3}$$

where minimum inter-cluster distance d_{min} in accordance with the single linkage method refers to the minimal distance between two trajectories from different clusters. Maximum intra-cluster diameter d_{max}, or the largest within-cluster distance in other words, supposes computing the diameter of a cluster as the distance between its two farthermost trajectories [41].

Higher values of the DI indicates the better results of clustering. However, the computational cost of the DI шs highly dependent on the data: the computation cost increases with the increase of number of clusters and dimensionality of the data [37].

4.1.6 Trajectories Approximation

However, notwithstanding that the LCSS similarity distance works with trajectories of arbitrary lengths and does not natively require the preprocessing of trajectories, the calculation of LCSS measure becomes extremely computationally expensive and time consuming with the growth of the trajectory length because of the recursiveness. For that reason it was decided to decrease the size of trajectories in the current work by approximation of trajectory data. That leads to the lose of accuracy but allows to get acceptable results in adequate amount of time.

The curve fitting concept is one of the standard approaches to perform approximation [42]. The main task is finding an appropriate relation or law possibly existing between the input (independent) and output (dependent) variables from a given input data set of observed values. And the curve fitting is the process of expressing a relationship between variables in terms of algebraic equations. The main goal of the curve fitting is to find parameters for a model (equation or function) to fit to the experimental data.

Regression Analysis

One of the widely used approaches appearing from curve fitting is a Regression Analysis, which is also considered as a form of predictive modeling approach and, according to the traditional definition, studies the relationship between a dependent variable (response) Y and one or more independent variables X's and tends to find trends in data. In other words it supposes "using the relationship between variables to fit the best fit line or regression equation that can be used to make predictions" [43].

In order to simplify the relationship fitting procedure, it is usually assumed that the independent X variables are measured without an error while the dependent Y variables values are measured with some random error. For the data with a small ratio of the measurement error in an independent variable to the range of values of that variable, it is possible to use the least squares regression analysis with legitimacy [42].

A regression can be linear or polynomial (nonlinear, curvilinear) depending on the function the data is approximated with: linear regression refers to a relationship approximated by a straight line whereas curvilinear regression refers to a relationship following a curve. Due to a broader range of functions the polynomial regression can work with, it provides better approximation of the input relationship in comparison to linear regression [43]. Even if it is impossible to guess the type of function to use for approximation in advance, plotting the data and analyzing it to find some behavioral pattern, such as linear, quadratic or higher-order dependency, can be useful [42].

Polynomial Regression

The visualization of input trajectory data is given in Figure 4.1.5. As it can be seen from the picture, neither the linear or 2nd order functions can not fit the data properly due to complexity of trajectory forms. For that reason it was decided to focus on approximation using higher-order polynomial regression. The evaluation of polynomial regression with different degrees will be given further and the following discussion and implementation will be intended to find a suitable n^{th} order polynomial equation and parameter values to represent each input trajectory as a 'trajectory function'. Since trajectory data is represented by two-dimensional spatial data along with temporal data and it is necessary to approximate spatial information, x- and y-coordinates will be considered as dependent variables and $time$ will be used as an independent variable. Consequently, polynomial regression will be performed twice with two output polynomial functions representing $x(t)$ and $y(t)$ for each of the input trajectories T:

$$\forall\, T = [\ldots(x_i, y_i, t_i)\ldots] => T(t) = \begin{cases} x = x(t) \\ y = y(t) \end{cases} \tag{4.1.4}$$

Trajectories will be converted from a shape of a list of trajectory points into equations (time functions) defined in a geometrical space, which can represent approximately all of them. Taking key points of the representative polynomial can decrease the size of the trajectory therethrough reducing the total operational cost and computational complexity of LCSS calculation. Moreover, mathematical equations are able to store information in a dense form and apart from other advantages such a data reduction leads to consuming less amount of space and increasing the storage efficiency [42]. Also so called build 'trajectory function's can provide interpolation and discover the missing data points.

4.2 Framework Implementation

This section will give implementation details of a presented concept based on a chosen stack of technologies. Based on a workflow of the framework outlined above, separate modules will be implemented. Detailed description of each of them will be presented in following sections.

4.2.1 Stack of Technologies

For implementation part of the work Java programming language along with Apache Maven as a build automation tool were used with following versions:

- Java - 11 OpenJDK
- Apache Maven - 3.6.3

Add stack of technologies, concrete versions and description

4.2.2 Input Data Processing

Since chosen algorithm requires trajectories in a form of multi-dimensional vectors, the initial input data needs to be converted into the required form. For that reason, a custom parser was implemented. It takes a 'txt' file with trajectories as an input and as a result it returns a list of Trajectory objects. Trajectory object consists of a number of TrajectoryPoint objects with following information: x-coordinate, y-coordinate, time t. The source code of the parsing method is presented in Appendix A.

As it was mentioned before, the current work is focused on detecting two types of abnormalities: spatial and spatiotemporal. To detect the outliers of the first group it is sufficient to analyze spatial information of trajectories. Detecting outliers of the second group, which is formed by trajectories of vehicles moving with an anomalously low or high speed, requires taking into consideration the temporal information along with spatial. For that reason the average constant speed υ is being calculated for each of the input trajectories t at the end of the parsing step using the following equation (Formula 4.2.1):

$$v_{avg}(t) = \frac{distance_{total}}{time_{total}}, \tag{4.2.1}$$

where $distance_{total}$ refers to the total distance between the first and last trajectory points and $time_{total}$ refers to the time elapsed. The total distance can be computed as a sum of Euclidean distances between trajectory points on neighboring frames. Since it is known that frames are taken with an inter-frame interval 0,01 second, the speed calculation can be implemented as follows (Listing 4.1):

Listing 4.1: Speed calculation

```java
/**
* Calculates average speed for the trajectory in ’pixels per sec’
*/
public double calcSpeed(Trajectory t) {
  double dist = 0.0;
  for (int i = 0; i < t.length() - 1; i++) {
    dist += t.get(i).distanceTo(t.get(i + 1));
  }

  double time = (t.get(length() - 1).getTime() - t.get(0).getTime()) * interFrameTime;

  double avgSpeed = dist / time;
  return avgSpeed;
}

/**
* Calculates the Euclidean distance between two trajectory points
*
* @param this        first (current) trajectory point
* @param other       second trajectory point
* @return            Euclidean distance
*/
public double distanceTo(TrajectoryPoint other) {
  if (this == other) {
    return 0;
  }
  double d = Math.pow(this.x - other.x, 2) + Math.pow(this.y - other.y, 2);
  return Math.sqrt(d);
}
```

4.2.3 Trajectories Approximation using Polynomial Regression

As it was discussed before, the polynomial regression will be used to approximate input trajectories. The implementations of a polynomial entity class[1] (needed to further analyze the approximation equations and find key points) and a polynomial equation solver [2] from the Apache Commons Math 3.4.1 library were used. To perform a polynomial regression[3] the implementation provided by R. Sedgewick and K. Wayne for Java language was taken. All the ready-to-use implementations were extended by utility methods.

The $PolynomialRegression$ class takes as an input the desired degree of a polynomial (d) and two data sets of N data points consisting of real numbers: array of independent variables ($double[]t$), temporal data in this case, and array of dependent variables ($double[]x$, $double[]y$), spatial x- or y-coordinates. Then it performs a polynomial regression on an input set of N data points (t_i, x_i) or (t_i, y_i) and tries to fit a polynomial $x = \beta_0 + \beta_1 t + \beta_2 t^2 + \ldots \beta_d t^d$, where β_i are the regression coefficients, with an aim to minimize the sum of squared residuals of the multiple regression model. Finding the best solution for polynomial parameters is based on a Least Squares method [42].

In order to achieve better approximation the evaluation of polynomial regression results was performed using the Coefficient of Determination denoted by R^2

[1] Polynomial implementation https://javadoc.io/doc/org.apache.commons/commons-math3/3.4.1/org/apache/commons/math3/analysis/polynomials/PolynomialFunction.html

[2] Polynomial Function Solver implementation https://www.javadoc.io/doc/org.apache.commons/commons-math3/3.4.1/org/apache/commons/math3/analysis/solvers/LaguerreSolver.html

[3] Polynomial Regression implementation https://algs4.cs.princeton.edu/14analysis/PolynomialRegression.java

(also known as a R-$squared$ score, *Pearson's coefficient of regression*) [44]. R^2 measures the proportion of the response dependent variable variance that can be explained by the regression model with given parameters and is predictable from the independent variable and can be calculated as follows:

$$R^2 = 1 - \frac{SSE}{TSS} = 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \overline{y})^2} \tag{4.2.2}$$

where SSE (Sum of Squares due to Error) is calculated as a sum of squared differences between actual y_i, and predicted $\hat{y}_i$ dependent variable values and TSS (Total Sum of Squares) is calculated as a sum of squared deviations of an actual value y_i from a mean $\overline{y}$.

R^2 takes a value between [0, 1] and value of 1 indicates that the model (polynomial equation in this case) predicts the data perfectly [45].

In this work the polynomial regression was performed for all the input trajectories (Appendix B). The resulting regression models were compared in terms of R^2 score and analyzed with respect to a trajectory: shape, speed. Following will give the comparison of evaluation results and discuss the obtained results.

Approximation Results

To make a decision about a degree of an approximation polynomials, several experiments were run trying to fit the input trajectories into polynomials of 3rd, 4th, 5th degrees respectively. The following table will depict minimum and average values of R^2 metric for each of the experiments (Table 4.2.1).

Table 4.2.1: R^2 values for different degrees of polynomials

Degrees of polynomials	R^2 score			
	X		Y	
	min	avg	min	avg
{3}	0.66	0.994	0.466	0.989
{3, 4}	0.897	0.997	0.823	0.994
{3, 4, 5}	0.949	0.998	0.864	0.995

Last column formatting...

It can be seen from the table that the average values of R^2 score are acceptable for all the experiments. However, the minimum R^2 values, which are equal to 0,66 and 0,466 are poor and unsatisfactory, meaning that model can predict barely half of the points for some trajectories correctly. This can affect following analysis and worsen further analysis. For that reason approximation using several degrees was performed in a following way:

1. firstly perform approximation using the lowest degree of a polynomial as a starting point,
2. compare the obtained R^2 with a predefined threshold (0,97 in this case); if the obtained value is less than the threshold value, increase the degree and reperform polynomial regression,
3. continue till the acceptable R^2 is obtained or till reaching the limit for a polynomial degree to check (5 in this case).

Approximation using 3rd and 4th degree polynomials in conjunction improved both minimum and average values of R^2 drastically. Though adding a 5th degree polynomial into consideration did not affect results significantly and improved the average coefficient only for 0,01 in comparison with the previous experiment. In view of this it was decided to focus on approximation using 3rd and 4th degrees.

For the sake of simplicity the trajectories are classified into two groups depending on a degree of polynomials used to approximate: first group contains trajectories approximated with a 3rd degree polynomial functions while the second group consists of trajectories approximated with a 4th degree polynomial functions. Both groups were analyzed in terms of a shape and an average speed. Figure 4.2.1 depicts second group of trajectories and Table 4.2.2 gives the minimum, average and maximum speed of trajectories for both groups.

Thereinafter the trajectories approximated with 3rd- and 4th-degree polynomial functions will be referred to as a first and second groups of trajectories respectively. It can be observed that trajectories approximated with 3rd- and 4th-degree polynomial functions have the widely different speeds. The first group includes trajectories with much higher speeds, particularly striking is that the maximum

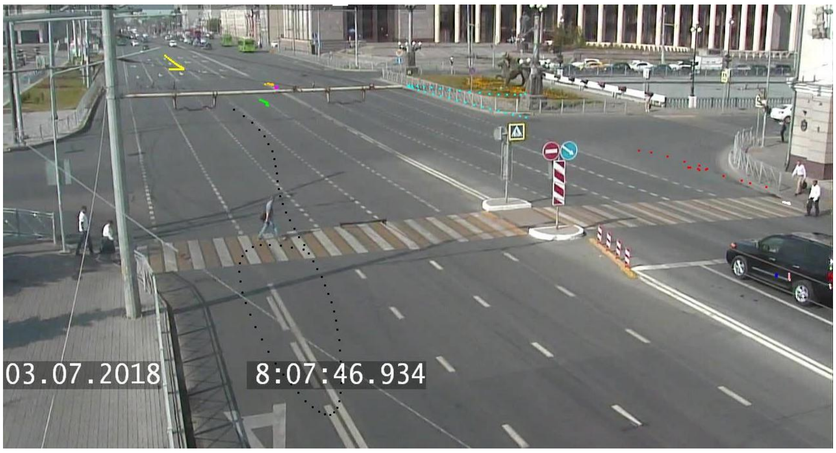

Figure 4.2.1: Trajectories approximated with 4th-degree polynomial functions

Table 4.2.2: Overview of min, avg and max speeds of vehicles

Degree of a polynomial	Speed *(pixels per sec)*		
	min	avg	max
{3}	18.555	331.299	1721.499
{4}	1.206	84.86	374.396

speed for the second group is almost equal to average speed for the first group and the average speed for the first group is almost four times as much as for the second group. Also the picture depicts that 4th-degree polynomial functions were used to approximate trajectories of complex shape or trajectories with densely located trajectory points.

Hence, it follows up that the higher-order polynomial functions are preferred to approximate trajectories of following groups:

- slow-moving or inactive vehicle trajectories (including trajectories of vehicles waiting at the intersections),
- trajectories of complex shapes (sharp turns, Pascal snails).

Choosing key points from approximated trajectories

Using the approximated trajectories in further calculation was aimed to decrease the complexity of LCSS calculation. For that reason the length of trajectories must be reduced by choosing several key representative points from the trajectories by analyzing the approximation polynomials.

It is known from Mathematics that critical points of a polynomial $f(t)$ refer to points where the polynomial function is not differentiable or the derivative at that point is equal to zero (stationary points). Stationary points, including local minimum and maximum, rising and falling inflection points, can be found by analyzing the first derivative of a function and solving the $f'(t) = 0$ equation. The inflection points can be found by further analysis of a second derivative: they correspond to the solutions of $f''(t) = 0$ equation.

In the case of trajectories analysis inflection points are very significant, because they carry an important information about the shape of a trajectory: such key points can denote the main turns or changes in the trajectory.

4.2.4 Similarity measure calculation

As it was mentioned before, LCSS measure will be used as a similarity measure. Consequently, LCSS distance will be calculated based on a LCSS similarity according to above mentioned formulas. It is worth noting that LCSS distance is symmetric and for pair of trajectories can be computed just once [30].

Notwithstanding that the implementation of LCSS similarity measure exists in R package [32], it does not allow δ and ε parameters to be dynamic. For that reason the custom implementation was written. The method for LCSS calculation is presented in Listing 4.2.

Listing 4.2: LCSS calculation

```
/**
* Calculates LCSS for two input trajectories
*
* @param t1     first trajectory
* @param t2     second trajectory
* @param δ      δ parameter: how far we can look in time to match a given point from one T to a point in another T
* @param ε      ε parameter: the size of proximity in which to look for matches
* @return       LCSS for t1 and t2
*/
private Double calcLCSS(Trajectory t1, Trajectory t2, Double δ, Double ε) {
  int m = t1.length();
  int n = t2.length();

  if (m == 0 || n == 0) {
    return 0.0;
  } else

  if (abs(t1.get(m - 1).getX() - t2.get(n - 1).getX()) < ε
      && abs(t1.get(m - 1).getY() - t2.get(n - 1).getY()) < ε
      && abs(m - n) <= δ) {
    return 1 + calcLCSS(head(t1), head(t2), δ, ε);
  } else {
    return max(
      calcLCSS(head(t1), t2, δ, ε),
      calcLCSS(t1, head(t2), δ, ε));
  }
}

/**
* Calculates shortened trajectory by excluding last trajectory point
*
* @param t trajectory
* @return trajectory without last trajectory point
*/
private Trajectory head(Trajectory t) {
  Trajectory tClone = t.clone();
  tClone.getTrajectoryPoints().remove(tClone.length() - 1);
  return tClone;
}
```

4.2.5 Clustering

Since no appropriate implementation of hierarchical clustering for trajectories with the use of LCSS distance and capable of taking an adaptable parameters values were found, the clustering as well as LCSS similarity calculation was written from scratch guided by Algorithm 1 outlined above. The source code of clustering is given in Appendix C.

Clustering is done in an iterative way of joining two closest clusters into one with following recalculation of a clusters similarity (proximity) matrix. The clustering method takes as an input the $OUTPUT\_CLUSTERS\_COUNT$ parameter which controls when the clustering will stop. If no value is passed, it will be considered as 1 and the clustering will be done till all clusters are merged into one in concordance with the basic algorithm of agglomerative hierarchical clustering.

4.2.6 Clusters' modeling

Chapter 5

Evaluation & Results

Chapter 6

Conclusion & Perspectives

In this work following results were achieved:

- task1,
- task2.

The implemented algorithm is designed in an offline-learning manner, that means that models of normal trajectories are learned offline beforehand and are not updated with new upcoming data on an on-going basis. The future researches can include investigating an opportunity of updating normal trajectories database in order to make the framework more adaptable to actual traffic data.

Bibliography

[1] Y. Djenouri, A. Belhadi, J. C. Lin, D. Djenouri, and A. Cano. A Survey on Urban Traffic Anomalies Detection Algorithms. *IEEE Access*, 7:12192–12205, 2019.

[2] F. Mehboob, M. Abbas, R. Jiang, A. Rauf, S. A. Khan, and S. Rehman. Trajectory Based Vehicle Counting and Anomalous Event Visualization in Smart Cities. *Cluster Computing*, 21:443–452, March 2018.

[3] C. Koetsier, S. Busch, and M. Sester. Trajectory Extraction for Analysis of Unsafe Driving Behaviour. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 42(2/W13):1573–1578, June 2019.

[4] R. Ranjith, J. J. Athanesious, and V. Vaidehi. Anomaly Detection using DBSCAN Clustering Technique for Traffic Video Surveillance. In *2015 7th International Conference on Advanced Computing (ICoAC)*, pages 1–6, December 2015.

[5] S. A. Ahmed, D. P. Dogra, S. Kar, and P. P. Roy. Trajectory-Based Surveillance Analysis: A Survey. *IEEE Transactions on Circuits and Systems for Video Technology*, 29(7):1985–1997, 2019.

[6] F. Meng, G. Yuan, S. Lv, Z. Wang, and S. Xia. An overview on trajectory outlier detection. *Artificial Intelligence Review*, February 2018.

[7] G. Atluri, A. Karpatne, and V. Kumar. Spatio-Temporal Data Mining: A Survey of Problems and Methods. *ACM Computing Surveys*, 51(4), 2017.

[8] F. Tung, J. S. Zelek, and D. A. Clausi. Goal-Based Trajectory Analysis for Unusual Behaviour Detection in Intelligent Surveillance. *Image Vision Comput.*, 29(4):230–240, March 2011.

[9] Y. Li, J. Bailey, L. Kulik, and J. Pei. Mining Probabilistic Frequent Spatio-Temporal Sequential Patterns with Gap Constraints from Uncertain Databases. In *2013 IEEE 13th International Conference on Data Mining*, pages 448–457, December 2013.

[10] G. Yuan, P. Sun, J. Zhao, D. Li, and C. Wang. A Review of Moving Object Trajectory Clustering Algorithms. *Artificial Intelligence Review*, 47(1):123–144, January 2017.

[11] A. d'Acierno, A. Saggese, and M. Vento. Designing Huge Repositories of Moving Vehicles Trajectories for Efficient Extraction of Semantic Data. *IEEE Transactions on Intelligent Transportation Systems*, 16(4):2038–2049, August 2015.

[12] V. Bogorny V. C. Fontes. Discovering Semantic Spatial and Spatio-Temporal Outliers from Moving Object Trajectories. *ArXiv*, abs/1303.5132, 2013.

[13] H. Liu, X. Li, J. Li, and S. Zhang. Efficient Outlier Detection for High-Dimensional Data. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 48(12):2451–2461, December 2018.

[14] V. Chandola, A. Banerjee, and V. Kumar. Anomaly Detection: A Survey. *ACM Computing Surveys*, 41(3), July 2009.

[15] F. E. Grubbs. Procedures for Detecting Outlying Observations in Samples. *Technometrics*, 11(1):1–21, February 1969.

[16] S. K. Kumaran, D. P. Dogra, and P. P. Roy. Anomaly Detection in Road Traffic Using Visual Surveillance: A Survey. *ArXiv: Computer Vision and Pattern Recognition*, January 2019.

[17] K. Malik, H. Sadawarti, and G. Kalra. Comparative analysis of outlier detection techniques. *International Journal of Computer Applications*, 97:12–21, July 2014.

[18] D. Kumar, J. Bezdek, S. Rajasegarar, C. Leckie, and M. Palaniswami. A Visual-Numeric Approach to Clustering and Anomaly Detection for Trajectory Data. *The Visual Computer*, 33(3):265–281, March 2017.

[19] S. W. T. T. Liu, H. Y. T. Ngan, M. K. Ng, and S. J. Simske. Accumulated Relative Density Outlier Detection For Large Scale Traffic Data. In *Electronic Imaging*, volume 9, pages 1–10, 2018.

[20] P. Batapati, D. Tran, W. Sheng, M. Liu, and R. Zeng. Video Analysis for Traffic Anomaly Detection using Support Vector Machines. In *Proceedings of the 11th World Congress on Intelligent Control and Automation (WCICA)*, pages 5500–5505, March 2014.

[21] C. Piciarelli, C. Micheloni, and G. L. Foresti. Trajectory-Based Anomalous Event Detection. *Circuits and Systems for Video Technology, IEEE Transactions on*, 18(11):1544–1554, December 2008.

[22] H.-L. Nguyen, Y.-K. Woon, and W. K. Ng. A Survey on Data Stream Clustering and Classification. *Knowledge and Information Systems*, 45:535–569, December 2014.

[23] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231. AAAI Press, 1996.

[24] R. Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, May 2005.

[25] G. F. Tzortzis and A. C. Likas. The Global Kernel k-Means Algorithm for Clustering in Feature Space. *IEEE Transactions on Neural Networks*, 20(7):1181–1194, July 2009.

[26] T. Bock. DisplayR Blog. What is Hierarchical Clustering? `https://www.displayr.com/what-is-hierarchical-clustering/`. Internet Resource, Accessed: 2020-07-05.

[27] C. R. Patlolla. Understanding the Concept of Hierarchical Clustering Technique. `https://towardsdatascience.com/understanding-the-concept-of-hierarchical-clustering-technique-c6e8243758ec`, December 2018. Internet Resource, Accessed: 2020-07-05.

[28] N. B. Ghrab, E. Fendri, and M. Hammami. Abnormal Events Detection Based on Trajectory Clustering. In *2016 13th International Conference on Computer Graphics, Imaging and Visualization (CGiV)*, pages 301–306, 2016.

[29] W. Liu, Y. Zheng, S. Chawla, J. Yuan, and X. Xing. Discovering Spatio-Temporal Causal Interactions in Traffic Data Streams. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 1010–1018, New York, NY, USA, 2011. Association for Computing Machinery.

[30] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering Similar Multidimensional Trajectories. In *Proceedings 18th International Conference on Data Engineering*, pages 673–684, February 2002.

[31] T. Eiter and H. Mannila. Computing Discrete Fréchet Distance *. In *Technical report CD-TR 94/64, Technische Universitat Wien*, 1994.

[32] K. Toohey. R Package Documentation. Similarity Measures. LCSS. `https://rdrr.io/cran/SimilarityMeasures/man/LCSS.html`, May 2019. Internet Resource, Accessed: 2020-06-30.

[33] K. Toohey and M. Duckham. Trajectory similarity measures. *SIGSPATIAL Special*, 7(1):43–50, May 2015.

[34] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh. Indexing multidimensional time-series. *The VLDB Journal*, 15:1–20, July 2006.

[35] Zhang Zhang, Kaiqi Huang, and Tieniu Tan. Comparison of similarity measures for trajectory clustering in outdoor surveillance scenes. volume 3, pages 1135–1138, January 2006.

[36] E.-H. Choi and National Highway Traffic Safety Administration. Crash Factors in Intersection-Related Crashes: An On-Scene Perspective. In *NHTSA Technical Report DOT HS 811 366*, September 2010.

[37] D. Dey. Dunn index and DB index – Cluster Validity Indices. `https://www.geeksforgeeks.org/dunn-index-and-db-index-cluster-validity-indices-set-1/`. Internet Resource, Accessed: 2020-07-07.

[38] J. C. Dunn. Well-separated clusters and optimal fuzzy partitions. *Journal of Cybernetics*, 4(1):95–104, 1974.

[39] DataCamp. Hierarchical Clustering in R. `https://www.datacamp.com/community/tutorials/hierarchical-clustering-R`. Internet Resource, Accessed: 2020-06-30.

[40] Z. Ansari, M.F. Azeem, W. Ahmed, and A. Babu. Quantitative evaluation of performance and validity indices for clustering the web navigational sessions. *World of Computer Science and Information Technology (WCSIT) Journal*, 1(5):217–226, 2011.

[41] B. Desgraupes. Clustering indices. 2016.

[42] I. Hadi and M. Sabah. Behavior formula extraction for object trajectory using curve fitting method. *International Journal of Computer Applications*, 104:28–37, 10 2014.

[43] A. Pant. Introduction to Linear Regression and Polynomial Regression. `https://towardsdatascience.com/introduction-to-linear-regression-and-polynomial-regression-f8adc96f31cb`, January 2019. Internet Resource, Accessed: 2020-07-15.

[44] Y. A. W. Shardt. *Statistics for Chemical and Process Engineers: A Modern Approach*, chapter 3.2 Regression Models, pages 90–104. Springer International Publishing, Cham, Switzerland, 2015.

[45] Minitab Blog Editor. Regression Analysis: How Do I Interpret R-squared and Assess the Goodness-of-Fit? https://blog.minitab.com/blog/adventures-in-statistics-2/regression-analysis-how-do-i-interpret-r-squared-and-assess-the-goodness-of-fit, May 2013. Internet Resource, Accessed: 2020-07-17.

Todo list

include or not? 9

Add some text about architecture, how should it work 40

δ behaviour? 42

include δ and ε as an input or calculate inside? 42

Add stack of technologies, concrete versions and description 47

Last column formatting... 51

Appendix

A. Input trajectories parsing algorithm

Listing 6.1: Input trajectories parsing algorithm

```java
package ru.griat.rcse.parsing;

import ru.griat.rcse.entity.Trajectory;
import ru.griat.rcse.entity.TrajectoryPoint;
import ru.griat.rcse.exception.TrajectoriesParserException;
import org.apache.commons.io.FilenameUtils;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;

/*
* Parser to parse input trajectories from text file
*
* stop symbols:
* if meet number - read until ']' or ',' or ')'
* [ - check for next, if [ - check for next, if [ - isX=true,
    if value - save x,
* */
public class TrajectoriesParser {

  private int openingSqBracketNumber;

  private boolean trajectoryStarted = false;
  private boolean trajectoryCoordinatesStarted = false;
  private int indexOfT;
  private int indexOfTP;

  private StringBuilder x;
  private StringBuilder y;
  private StringBuilder t;

  private List<TrajectoryPoint> trajectoryPoints;
  private List<Trajectory> trajectories;

  public TrajectoriesParser() {
    openingSqBracketNumber = 0;
    indexOfT = 0;
    indexOfTP = 0;

    x = new StringBuilder();
    y = new StringBuilder();
    t = new StringBuilder();

    trajectoryPoints = new ArrayList<>();
    trajectories = new ArrayList<>();
  }

  /**
  * Parses input 'txt'-file
  *
  * @param fileName  full path to the input data file with trajectories
  * @return          list of extracted trajectories
  */
  public List<Trajectory> parseTxt(String fileName) throws IOException, TrajectoriesParserException {

    InputStream reader = new FileInputStream(FilenameUtils.normalize(fileName));
    int intch;
    while ((intch = reader.read()) != -1) {
      char nextChar = (char) intch;
      while ((nextChar == ',' || nextChar == ' '))
      nextChar = (char) reader.read();
      while (nextChar == '[') {
        increaseOpeningSqBracketsCount();
        nextChar = (char) reader.read();
      }
      while (trajectoryCoordinatesStarted) {
        if (nextChar == '(') {
          readCoordinates(reader);
        }
        nextChar = (char) reader.read();
        if (nextChar == ’]’) {
          increaseClosingSqBracketsCount();
        }
      }
      nextChar = (char) reader.read();
      while ((nextChar == ’,’ || nextChar == ’ ’))
      nextChar = (char) reader.read();
      if (trajectoryStarted) {
        if (nextChar == ’[’) {
          increaseOpeningSqBracketsCount();
          readTime(reader);
        } else {
          throw new TrajectoriesParserException(”After coordinates array with timestamps was expected”);
        }
        finishProcessingTrajectory();
      }
    }

    reader.close();
    return trajectories;
  }

  private void processBracketsCount() {
    if (openingSqBracketNumber == 1) {
      trajectoryStarted = false;
      trajectoryCoordinatesStarted = false;
    }
    if (openingSqBracketNumber == 2) {
      trajectoryStarted = true;
      trajectoryCoordinatesStarted = false;
    }
    if (openingSqBracketNumber == 3) {
      trajectoryCoordinatesStarted = true;
    }
  }
```

```java

/**
* Reads an x and y values from file after '(' and before next ')'
*/
private void readCoordinates(InputStream reader) throws IOException {
  char nextChar = (char) reader.read();
  while (nextChar != ',') {
    if (nextChar >= '0' && nextChar <= '9')
    x.append(nextChar);
    nextChar = (char) reader.read();
  }
  while (nextChar != ')') {
    if (nextChar >= '0' && nextChar <= '9')
    y.append(nextChar);
    nextChar = (char) reader.read();
  }
  processTrajectoryPoint();
}

/**
* Reads time and saves it into already initialized trajectory by updating trajectoryPoint at indexOfTP position in a current trajectory
*/
private void readTime(InputStream reader) throws IOException {
  char nextChar = (char) reader.read();
  while (nextChar != ']') {
    while (nextChar != ',' && nextChar != ']') {
      t.append(nextChar);
      nextChar = (char) reader.read();
    }
    if (nextChar == ']') {
      increaseClosingSqBracketsCount();
    }
    trajectoryPoints.get(indexOfTP).setTime(Integer.parseInt(t.toString().trim()));
```

```java
      indexOfTP++;
      t = new StringBuilder();
      nextChar = (char) reader.read();
    }
  }

  private void increaseOpeningSqBracketsCount() {
    openingSqBracketNumber++;
    processBracketsCount();
  }

  private void increaseClosingSqBracketsCount() {
    openingSqBracketNumber--;
    processBracketsCount();
  }

  /**
  * Adds parsed trajectory into an array of output
   trajectories and prepares for the next input trajectory by
    resetting to 0 indexes and buffers
  */
  private void finishProcessingTrajectory() {
    trajectories.add(new Trajectory(indexOfT,
   trajectoryPoints));
    trajectoryPoints = new ArrayList<>();
    indexOfT++;
    indexOfTP = 0;
    trajectoryStarted = false;
    increaseClosingSqBracketsCount();
  }

  /**
  * Creates a new TrajectoryPoint with collected x and y
  * Clear the buffer
  */
  private void processTrajectoryPoint() {
    TrajectoryPoint point = new TrajectoryPoint(
      Integer.parseInt(x.toString().trim()),
      Integer.parseInt(y.toString().trim())
    );
    trajectoryPoints.add(point);

    x = new StringBuilder();
    y = new StringBuilder();
  }

}
```

B. Polynomial Regression initiation

Listing 6.2: Polynomial Regression initiation

```
// initialization
double[] t, x, y;
int degree = 3;
double thresholdR2 = 0.97;
double minR2forX = 1.0, minR2forY = 1.0;
int minR2forXid = -1, minR2forYid = -1;

Invocation of the polynomial regression for each trajectory
for (int tId = 0; tId < trajectories.size(); tId++) {
  PolynomialRegression regressionX;
  PolynomialRegression regressionY;

  Trajectory currentTr = trajectories.get(tId);
  t = currentTr.getTrajectoryPoints().stream()
    .mapToDouble(TrajectoryPoint::getTime).toArray();
  x = currentTr.getTrajectoryPoints().stream()
    .mapToDouble(TrajectoryPoint::getX).toArray();
  y = currentTr.getTrajectoryPoints().stream()
    .mapToDouble(TrajectoryPoint::getY).toArray();
  regressionX = new PolynomialRegression(t, x, degree);
  regressionY = new PolynomialRegression(t, y, degree);

//    if regression results are not satisfactory (means that degree of polynomial is not enough)
//    try to obtain an equation with a higher degree
  if (regressionX.R2() < thresholdR2)
    regressionX = new PolynomialRegression(t, x, degree + 1);
  if (regressionY.R2() < thresholdR2)
    regressionY = new PolynomialRegression(t, y, degree + 1);

  currentTr.setRegressionX(regressionX);
  currentTr.setRegressionY(regressionY);

//     calculation of minimum $R^2$
  if (regressionX.R2() < minR2forX) {
    minR2forX = regressionX.R2();
    minR2forXid = tId;
  }
  if (regressionY.R2() < minR2forY) {
    minR2forY = regressionY.R2();
    minR2forYid = tId;
  }
}

// calculation of average $R^2$
double avgR2forX = trajectories.stream()
  .mapToDouble(tr -> tr.getRegressionX().R2())
  .average().getAsDouble();
double avgR2forY = trajectories.stream()
  .mapToDouble(tr -> tr.getRegressionY().R2())
  .average().getAsDouble();

// print results
LOGGER.info(”min R2 for X is for trajectory {}: {}”,
    minR2forXid, minR2forX);
LOGGER.info(”avg R2 for X is: {}”, avgR2forX);
LOGGER.info(”min R2 for Y is for trajectory {}: {}”,
    minR2forYid, minR2forY);
LOGGER.info(”avg R2 for Y is: {}”, avgR2forY);
```

C. Agglomerative Hierarchical Clustering

Listing 6.3: C. Clustering implementation

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import ru.griat.rcse.entity.Cluster;
```

```java
import ru.griat.rcse.entity.Trajectory;
import ru.griat.rcse.entity.TrajectoryPoint;

import java.util.ArrayList;
import java.util.List;

import static java.lang.Math.*;

public class Clustering {

    private static final Logger LOGGER = LoggerFactory.getLogger(Clustering.class.getName());
    private static final int OUTPUT_CLUSTERS_COUNT = 17;

    private List<Cluster> clusters;

    private Double[][] trajLCSSDistances;
    private Double[][] clustLCSSDistances;
    private int minX, maxX, minY, maxY;
    private TrajectoryPoint cameraPoint;

    public Clustering(List<Trajectory> trajectories) {
        clusters = new ArrayList<>();
        trajLCSSDistances = new Double[trajectories.size()][trajectories.size()];
        clustLCSSDistances = new Double[trajectories.size()][trajectories.size()];
    }

    public Double[][] getTrajLCSSDistances() {
        return trajLCSSDistances;
    }

    public void setTrajLCSSDistances(Double[][] trajLCSSDistances) {
        this.trajLCSSDistances = trajLCSSDistances;
        for (int i = 0; i < trajLCSSDistances.length; i++) {
            System.arraycopy(
                trajLCSSDistances[i], 0,
                clustLCSSDistances[i], 0,
                trajLCSSDistances.length);
        }
    }

    /**
  * set borders for an input image in terms of pixels
  * calculate the position of a camera
  */
    public void setBorders(int minX, int maxX, int minY, int
   maxY) {
        this.minX = minX; this.maxX = maxX;
        this.minY = minY; this.maxY = maxY;
        this.cameraPoint = new TrajectoryPoint(
            (int) Math.round(0.25 * maxX),
            (int) Math.round(0.95 * maxY));
    }

    /**
     * Single linkage
     * LCSS similarity measure
     *
     * @param trajectories  database of trajectories
     * @return        clusters of trajectories
     */
    public List<Cluster> cluster(List<Trajectory>
   trajectories) {
        initClusters(trajectories);
        whileCluster(OUTPUT_CLUSTERS_COUNT);
        return clusters;
    }

    /**
  * initialize clusters with each trajectory singly
  */
    public void initClusters(List<Trajectory> trajectories) {
        trajectories.forEach(trajectory ->
                clusters.add(new Cluster(trajectory.getId(),
   trajectory)));
```

```
    }

    /**
     * stopPoint - desired number of clusters to stop:
     * if null - stop when 1 cluster is left
     * if no joins are possible, stop.
     */
    public void whileCluster(Integer stopPoint) {
        if (stopPoint == null)
            stopPoint = 1;
        int numOfClusters = clusters.size();
        int id1;
        int id2;
        double minClustDist;
        while (numOfClusters > stopPoint) {
            id1 = -1;
            id2 = -1;
            minClustDist = Double.MAX_VALUE;
            for (int i1 = 0; i1 < clusters.size(); i1++) {
                for (int i2 = i1 + 1; i2 < clusters.size(); i2++) {
                    if (i1 != i2
                            && clustLCSSDistances[clusters.get(i1).getId()][clusters.get(i2).getId()] != null
                            && clustLCSSDistances[clusters.get(i1).getId()][clusters.get(i2).getId()] < minClustDist) {
                        minClustDist = clustLCSSDistances[clusters.get(i1).getId()][clusters.get(i2).getId()];
                        id1 = i1;
                        id2 = i2;
                    }
                }
            }
//            join i1 and i2 clusters, add i1 traj-es to cluster i2
            clusters.get(id1).appendTrajectories(clusters.get(id2).getTrajectories());
//            recalculate D for i1 and i2 lines -> set i2 line all to NULLs
            recalcClustersDistMatrix(id1, id2);

//            remove i2 from 'clusters'
            clusters.remove(id2);

            numOfClusters--;
        }
        printClusters();

    }

    /**
     * Calculates LCSS distance for two input trajectories
     * Smaller the LCSS distance - the better (0.0 - equal trajectories)
     *
     * @param t1 first trajectory
     * @param t2 second trajectory
     * @return LCSS distance for t1 and t2
     */
    public Double calcLCSSDist(Trajectory t1, Trajectory t2) {
        int m = t1.length();
        int n = t2.length();

        double delta = getDelta(m, n);
        double epsilonX = getEpsilonX(m, n);
        double epsilonY = getEpsilonY(m, n);

        double dist = 1 - calcLCSS(t1, t2, delta, epsilonX, epsilonY) / min(m, n);
        trajLCSSDistances[t1.getId()][t2.getId()] = dist;
        clustLCSSDistances[t1.getId()][t2.getId()] = dist;
        return dist;
    }


    /**
     * Calculates LCSS for two input trajectories
     * Bigger the LCSS - the better
     *
     * @param t1       first trajectory
     * @param t2       second trajectory
     * @param delta    δ parameter: how far we can look in time to match a given point from one T to a point in another T
     * @param epsilonX ε parameter: the size of proximity in which to look for matches on X-coordinate
     * @param epsilonY ε parameter: the size of proximity in which to look for matches on Y-coordinate
     * @return LCSS for t1 and t2
     */
    private Double calcLCSS(Trajectory t1, Trajectory t2, Double delta, Double epsilonX, Double epsilonY) {
        int m = t1.length();
        int n = t2.length();

        if (m == 0 || n == 0) {
            return 0.0;
        }

//      check last trajectory point (of each trajectory-part recursively)
//      delta and epsilon as thresholds for X- and Y-axes respectively
//      Then the abscissa difference and ordinate difference are less than thresholds (they are relatively close to each other), they are considered similar and LCSS distance is increased by 1
        else if (abs(t1.get(m - 1).getX() - t2.get(n - 1).getX()) < epsilonX
                && abs(t1.get(m - 1).getY() - t2.get(n - 1).getY()) < epsilonY
                && abs(m - n) <= delta) {
            return 1 + calcLCSS(head(t1), head(t2), delta, epsilonX, epsilonY);
        } else {
            return max(
                    calcLCSS(head(t1), t2, delta, epsilonX,
   epsilonY),
                    calcLCSS(t1, head(t2), delta, epsilonX,
   epsilonY)
            );
        }
    }

    /**
     * Calculates shortened trajectory by excluding last
   trajectory point
     *
     * @param t trajectory
     * @return trajectory without last trajectory point
     */
    private Trajectory head(Trajectory t) {
        Trajectory tClone = t.clone();
        tClone.getTrajectoryPoints().remove(tClone.length() -
    1);
        return tClone;
    }

    /**
     * calc δ
     *
     * @param m length of first trajectory
     * @param n length of second trajectory
     * @return δ value
     */
    private Double getDelta(int m, int n) {
        return 0.5 * min(m, n);
    }

    /**
     * calc ε for X
     *
     * @param m length of first trajectory
```

```
     * @param n length of second trajectory
     * @return ε value
     */
    private Double getEpsilonX(int m, int n) {
        return 0.1 * (maxX - minX);
    }

    /**
     * calc ε for Y
     *
     * @param m length of first trajectory
     * @param n length of second trajectory
     * @return ε value
     */
    private Double getEpsilonY(int m, int n) {
        return 0.1 * (maxY - minY);
    }

    /**
     * At each step calc a distance matrix btwn clusters
     * Merge two clusters with a min dist -> requires an update of the dist matrix
     * because of the implementation: clusterId1 < clusterId2
     *
     * @param clusterId1 index of left joined cluster in clusters list (remained cluster)
     * @param clusterId2 index of right joined cluster in clusters list (removed cluster)
     */
    private void recalcClustersDistMatrix(int clusterId1, int clusterId2) {
        for (int i = 0; i < clusterId1; i++) {
            clustLCSSDistances[clusters.get(i).getId()][clusterId1] =
                calcClustersDist(clusters.get(i), clusters.get(clusterId1));
        }
        for (int j = clusterId2; j < clusters.size(); j++) {
```

```
      clustLCSSDistances[clusterId2][clusters.get(j).getId()] =
       calcClustersDist(clusters.get(clusterId2), clusters.get(j));
    }
    clustLCSSDistances[clusters.get(clusterId1).getId()][clusters.get(clusterId2).getId()] = null;
  }

  /**
   * Calculates inter-clusters distance for two input clusters
   * using 'single-link' linkage method:
   * the between-cluster distance == the min distance btwn two trajectories in the two clusters
   *
   * @param cluster1 first cluster
   * @param cluster2 second cluster
   * @return distance between clusters
   */
  private Double calcClustersDist(Cluster cluster1, Cluster cluster2) {
    double dist = Double.MAX_VALUE;
    for (Trajectory trajectory1 : cluster1.getTrajectories()) {
      for (Trajectory trajectory2 : cluster2.getTrajectories()) {
        Double lcssDist = trajLCSSDistances[trajectory1.getId()][trajectory2.getId()];
        if (lcssDist != null && lcssDist < dist)
          dist = lcssDist;
      }
    }
    return dist;
  }

}
```