Baden-Wuerttemberg Cooperative State University

Faculty of Computer Science

Bachelor Thesis

"Inferring Sailing Race Course Layouts from Boat Satellite Tracking and Wind Data"

submitted by

Jan Luca Adams

2022

Baden-Wuerttemberg Cooperative State University

Faculty of Computer Science

Bachelor Thesis

"Inferring Sailing Race Course Layouts from Boat Satellite Tracking and Wind Data"

submitted by

Jan Luca Adams

2022

Submission Date:	29. August 2022
Editing Timespan:	06.06.2022 - 29.08.2022
Matriculation Number, Class	1012734, TINF19B2
Cooperate Partner:	SAP SE Dietmar-Hopp-Allee 16 69190 Walldorf, Germany
Supervisor of the Cooperate Partner:	Dr. Axel Uhl
Examiner of the Cooperate University:	Prof. Dr. Ralph Lausen

Declaration of Originality

I hereby confirm my bachelor thesis on the subject

> *Inferring Sailing Race Course Layouts from Boat Satellite Tracking and Wind Data*

has been written independently in accordance with § 5 of the "Studien- und Prüfungsordnung DHBW Technik" dated September 29, 2017 and that I have not used any sources or aids other than those specified. The work has not yet been submitted to any other examination authority and has not been published.

Furthermore, I assure that the submitted electronic version is identical to the printed version.

Karlsruhe, 29th August 2022

Jan Luca Adams
Adams, Jan Luca

Abstract

The aim of this bachelor thesis was the conception and implementation of a software solution for approximating information about the layout of a sailing race course in order to eliminate the necessity of individual mark tracking by solely using wind data and tracking data of the competitors as well as general knowledge of the course structure. It is investigated which information can be inferred from the available data, whether this is also possible live during the race and what approaches are suitable for this purpose. Through review of existing technologies for data analysis and the definition of requirements, a concept for the determination of waypoints using pattern recognition within the sailing behavior of the competitors was conceptualized. Core aspects of this concept were also evaluated in terms of their accuracy in the form of an early implementation. The designed solution is able to identify marks and determine their approximate positions in 85% of the tested waypoint to an average accuracy of 7 meters. A measured average time offset of mark roundings between actual and inferred marks less than one second.

Contents

List of Abbreviations

COG	Course over Ground
DBSCAN	A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise
GCS	Geograpical Coordinate System
GPS	Global Positioning System
GWT	Google Web Toolkit
GUI	Graphical User Interface
HDG	Heading
IOT	Internet of Things
OSGi	Open Services Gateway initiative
POS	Point of Sail
SOG	Speed over Ground
TWA	True Wind Angle
TWD	True Wind Direction
UML	Unified Modeling Language

List of Figures

List of Tables

List of Listings

1 Introduction

Automated analyses and visualizations in sailboat racing are often complex and cost-intensive. This thesis aims to present a solution to significantly reduce the effort required to set up the measurement infrastructure for analytical tools. A brief overview of the background, motivations and objectives for pursuing this approach is outlined in more detail below.

In sailboat racing, two or more boats sail simultaneously on a course and try to arrive at the finish line first. The dimensions and duration of races can vary greatly, from small races lasting less than an hour to offshore races, such as The Ocean Race, which can last up to several months.

Although the rules of different races may vary, most official competitions are conducted according to the racing rules [1] of World Sailing, the world governing body for the sport of sailing. Based on this set of rules, it is possible to determine some facts about the sport of sailing, which apply to most sailing races.

Unlike other racing sports, sail racing presents some unique challenges and characteristics when it comes to making it comprehensible and accessible to the general public. In a car race, for example, it is extremely easy to understand which competitor is in the lead and how the leaderboard is composed; to put it simply, you just look at who is furthest down the track. In sailing, however, this is not so simple, because another decisive factor must always be taken into account: The wind. This, combined with the fact that the races are usually several kilometers away from shore, makes it incredibly difficult for spectators and commentators to follow what is happening on the race track. Even for the candidates themselves on the water, it is often rather difficult to understand in detail what exactly happened during a race and where there is potential for improvement.

A remedy for this is provided by software solutions that are able to represent what is happening on the water and visualize it for the viewer based on a range of sensor readings from the boats themselves and also from the general environment. SAP Sailing Analytics goes one step further and delivers an entire suite of software solutions for analyzing,

monitoring, and managing sailing races and their results; and most of it even live during the race, with a sub-second delay.

The course layout of a sailing regatta course primarily consists of a series of waypoints that must be passed by the competitors in sequence. Each of the waypoints is represented in the real world by one or more marks. The shape and nature of the marks can vary greatly; for example, they can be temporary buoys, permanently installed marks, or even landmarks like lighthouses and islands. The quantity and composition of marks of each waypoint depend on its type.

- *Basic course mark:* Represented by a single mark, this checkpoint just has to be passed on a certain side. The side is specified by so-called passing instructions, which state to which side of the boat the mark needs to be left.
- *Gate:* Gates consist of two different marks located close to each other, which can be connected by an imaginary line. This line should be as perpendicular to the wind direction as possible. The competitors have to cross that line first and then round one of the two marks on the respective side. Circumventing the results of the mark from the fact that gates are always placed in such a way that a subsequent reversal of the direction of travel is required.
- *Line:* Similar to the gate, a line is defined by two marks, which then span a theoretical line. This line, however, only needs to be crossed in a specific direction. In the inner part of a sailing course, meaning between the start and finish line, this waypoint type is mainly used in long-distance races, which is why far distances between the two marks are not uncommon.
- *Course mark with bearing:* In addition to an ordinary mark, an additional bearing may be added to the mark. Along the bearing, starting from the mark itself, a line can be imagined which has to be crossed by the boats.

Additional waypoints in each course layout are the start and finish lines. These essentially also consist of two marks, which then span the theoretical start or finish line.

The schematic layout and composition of a course will be published by the race organizers prior to the race. This contains the sequence of waypoints that must be followed, including the instructions for rounding each mark. Several waypoints may also use identical marks

or share individual marks. However, the exact lengths of course-legs and positions of marks cannot be taken from the diagram.

1.1 Motivation

Before one is able to provide any visualization, comprehensive leaderboards, or detailed analysis of what is taking place on the water, you need some fundamental information. Among these are the current wind, the position of the boats on the water, and the course to be sailed, and thus the exact positions and rounding instructions of the course marks. While the former two do seem indispensable, the latter raises the question of whether this information can be derived from wind data and the tracking data of the boats as well.

This would greatly reduce the amount of equipment needed and the time and effort required to set up a race. Compared to other racing sports, obtaining accurate course information often proves to be extremely challenging or time-consuming. While in land-based races the course layout and the associated waypoint positions are always stationary requiring only a one-time fix, the fact that the race takes place on the water presents a major hurdle.

Fixed marks and landmarks may still have known positions, allowing them to be easily positioned. However, this is not the case with floating marks, representing the predominant type of course delimitation. They are often installed or relocated as needed and have the ability to be influenced by current or drift a few meters with the wind, depending on the depth of the water. During fully tracked events and races, one or more tracking devices are installed on them to monitor their exact position in the system on a continuous basis. Whereas for smaller races this is often not worth the effort or there are simply no additional tracking devices available that could be used for this purpose. The SAP Sailing Suite offers an app to use smartphones as tracking devices, which eliminates the need for professional tracking equipment. Nevertheless, this requires additional smartphones, which are usually already in use on the boats. Alternatively, another app may be used to record the static positions of the marks in the system while they are being laid out. This still involves additional effort and for the reasons mentioned above only provides a limited workaround often not feasible for smaller races. Likewise, a smartphone has

to be operated after the placement of each course mark, which sometimes proves to be difficult depending on weather conditions and when wearing gloves.

Therefore, a solution that would be able to infer the missing course information from the already existing wind and tracking data of the boats would be a significant improvement, which would likely expand the reached target group of the SAP Sailing Analytics. Furthermore, such a solution may also prove useful for large traditionally tracked races, where due to technical problems or human error tracking information of individual marks is temporarily missing and has to be approximated accordingly.

1.2 Assignment

The objective of the work is to develop an extension for the SAP Sailing Analytics platform, addressing the inference of possible information about the course layout from wind and tracking data. As part of this assignment, a series of relevant questions arise to examine this subject area in greater detail.

- What information about the course layout can be derived?
- Is it possible to derive the positions of the individual marks?
- Is it possible to derive the remaining waypoint information as well?
- How can the waypoints be sequenced to form a course?
- Are all processes also executable in real-time during the race, or is it only possible to execute them for past races?

In a preliminary assessment of the available data, it can already be said that even though a lot of information about the course layout is likely to be contained in the data, some information simply cannot be derived. Accordingly, restrictions are made concerning the scope of this assignment.

Waypoints that consist exclusively of marks that have to be rounded, as is the case with the standard mark and with the gate, should be identifiable based on the tracking data. With other waypoint types, this looks different. It should be difficult to identify a line based on the movement data since no or only slight changes in the movement pattern are apparent. Since the start and finish are also represented by lines, a similar issue may arise

here. The same applies to individual marks with an additional bearing. Determining the position of the mark should work in the same way as for its counterpart without an additional bearing. Determining the exact bearing or even the distinction from a conventional mark is not likely to be possible with the available data. For this reason, the work is limited exclusively to the interior waypoints of a course and assumes a given start and finish line. Likewise, the emphasis will be solely on the identification of simple marks and gates.

It will be assumed that tracking data of the boats as well as wind data are provided at all times.

Regarding the limitations in terms of already known information about the course layout, different degrees of restriction are applicable. The primary aim of the thesis is to achieve good results under the assumption that the general order and composition of the waypoints are already known. However, especially in the conception different approaches are explored, which allow for the inference of this data as well. Due to the limited time and resources of this thesis, however, these are not further investigated in the implementation and evaluation of the results. In a first approach, the assumption is made that the chosen course layout is known as well.

1.3 Outline

In chapter one, the necessary context and background of the subject have been presented to establish motivation, objectives, limitations, and the general course of action of the thesis.

In chapter two, the current state of technology is discussed in detail and the necessary technical knowledge about existing technologies and research will be presented, which will be required for an understanding of the following chapters.

In chapter three, the core concept is explained in detail. Possible challenges are investigated and requirements for the solution will be defined. Afterward, different approaches to solving the problem are presented and compared with each other on a theoretical level to be able to make a well-founded selection of the methods used.

In chapter four, the specific implementation of the previously explained theoretical concepts will be explained, as well as the integration into the existing environment. Specifically, the technical decisions regarding the execution, such as the choice of the implementation or the parameters used, will be presented. Likewise an evaluation of the results will be conducted and the generated results will be compared with real-world data to assess the accuracy of the solution.

In chapter five, a summary of the work with regard to the questions posed in chapter one will be presented. Likewise, a brief outlook will be given for subsequent research topics in this subject area, although they were not part of this thesis, possible approaches and ideas have already emerged in the course of the work.

2 Fundamentals

To properly follow the contents of this thesis, a fundamental knowledge of sailing and sailing regattas is required, which will be established hereinafter. Besides that, a lot of preliminary work has already been done in the area of efficient information extraction from large amounts of data. The following sections deal with existing technologies, equations, and approaches in the field of data processing, but also specifically in the handling of positioning data.

2.1 Sail Racing

Especially knowledge about the possible courses a sailboat is able to sail and the structure of race track layouts will be needed in the following chapters. Consequently, particular emphasis will be placed on this.

2.1.1 Sailing Basics

The True Wind Angle (TWA) indicates the angle of a boat in relation to the True Wind Direction (TWD), the current wind direction as perceived on land. The angle of the boat is measured by its keel line and not by the direction of its movement. Boats do not necessarily move along their keel line on the water, but displaced from it, depending on the wind and current. This is also referred to as Course over Ground (COG). In turn, the course that runs along the keel line is called the Heading (HDG). The difference between HDG and wind direction is the TWA [2]. The TWA may range from 0 degrees to 180 degrees, where 0 degrees would correspond to sailing directly into the wind and 180 degrees would have the wind completely from behind. Depending on which side the wind is on as seen from the bow, the TWA has a positive or negative prefix. It is negative if the wind direction is to port and positive if the wind direction is to starboard.

The position of the sails and the direction of the wind play a key role in generating speed with a sailboat. The actual speed as perceived from land or by tracking is also referred to

as Speed over Ground (SOG). No sailboat is able to sail directly into the wind, as there no tension is built up in the sails and they begin flapping. Depending on the boat and the prevailing wind, an TWA of approximately 45 degrees is required in order to pick up speed. In all orientations beyond this critical angle, in turn, sailing is possible [2]. That orientation with respect to the TWA is also called the "Point of Sail".

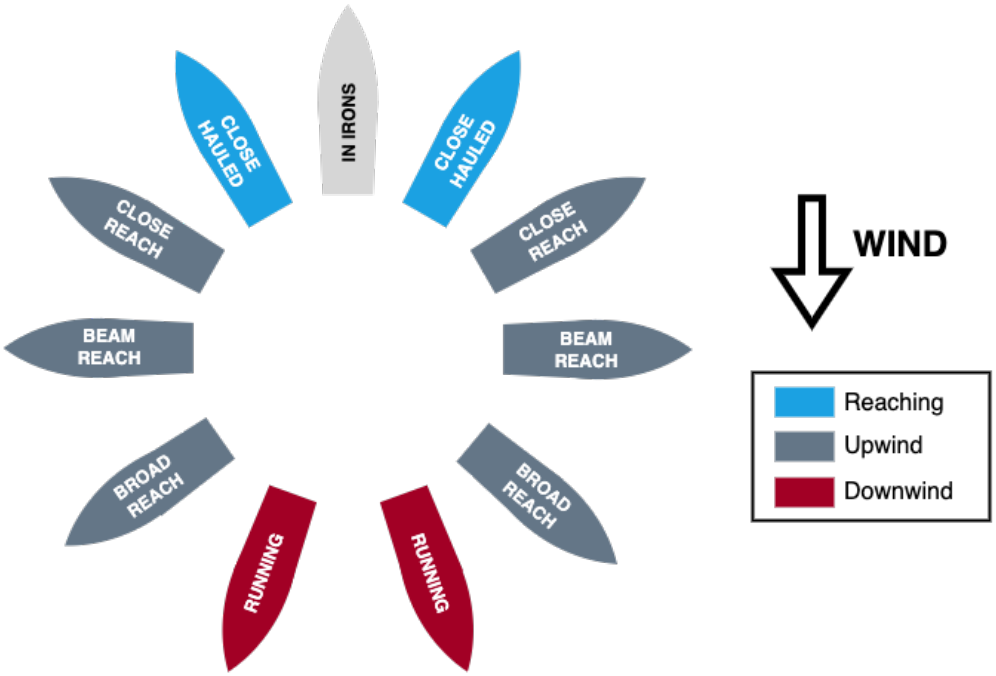

Figure 2.1: Schematic representation of the "Point of Sail"

As shown in Figure 2.1, the various orientations have different labels. In the scope of this thesis, however, such a detailed segmentation is not needed and therefore a less specific one will be used. Sailing into the wind is called upwind sailing, sailing with the wind at one's stern is called downwind sailing, and sailing perpendicular to the wind in either direction is called reaching. To differentiate between the two reaching orientations, the prefix of the TWA may be used. Each of these three labels is represented by a different color in the figure. The separation angles are only schematic, as they may vary depending on the boat used and the current wind speed [2].

To still be able to sail towards the direction from which the wind is coming, it is necessary to tack. Tacking refers to sailing alternating between the two upwind areas that are not

part of the no-go zone. The distance sailed in this way resembles a zigzag line as shown in Figure 2.2. Thus it is possible to sail straight upwind regardless of the no-go zone [2].

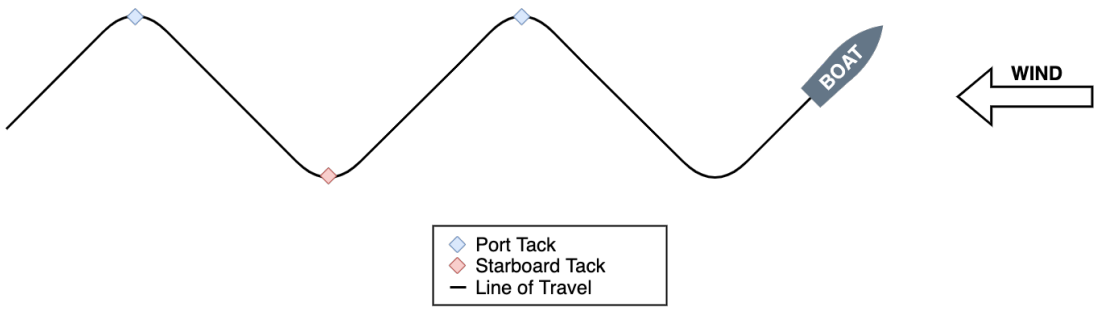

Figure 2.2: Zigzag line of travel when tacking into the wind

2.1.2 Race Course Layout

As already briefly explained in chapter 1, a sailing race course is composed of a sequence of waypoints, that on their end consist of one or more marks. For better identification, each mark is given a unique name within the depicted model. As a form of representation for the arrangement of the marks, a two-dimensional perspective of the race course from above is used, as illustrated as an example in Figure 2.3.

A sequence of waypoints belonging to the course layout schematically shown in Figure 2.3 is specified as follows [3]:

I2: *Start - 1 - 4s/4p - 1 - 2 - 3p - Finish*

Each of the dashes separates the waypoints from each other, while all the marks belonging to the waypoint are specified in between. In the example of *4s/4p* the representation of a gate is visible, where only one of the two waypoints has to be rounded. This notation does not provide any information on the passing instructions of the waypoints, as these can be taken from the accompanying diagram. One course diagram can also serve several variations of possible waypoint sequences, for example, other variations besides the previously presented I2 are the sequences I3 and I4 [3].

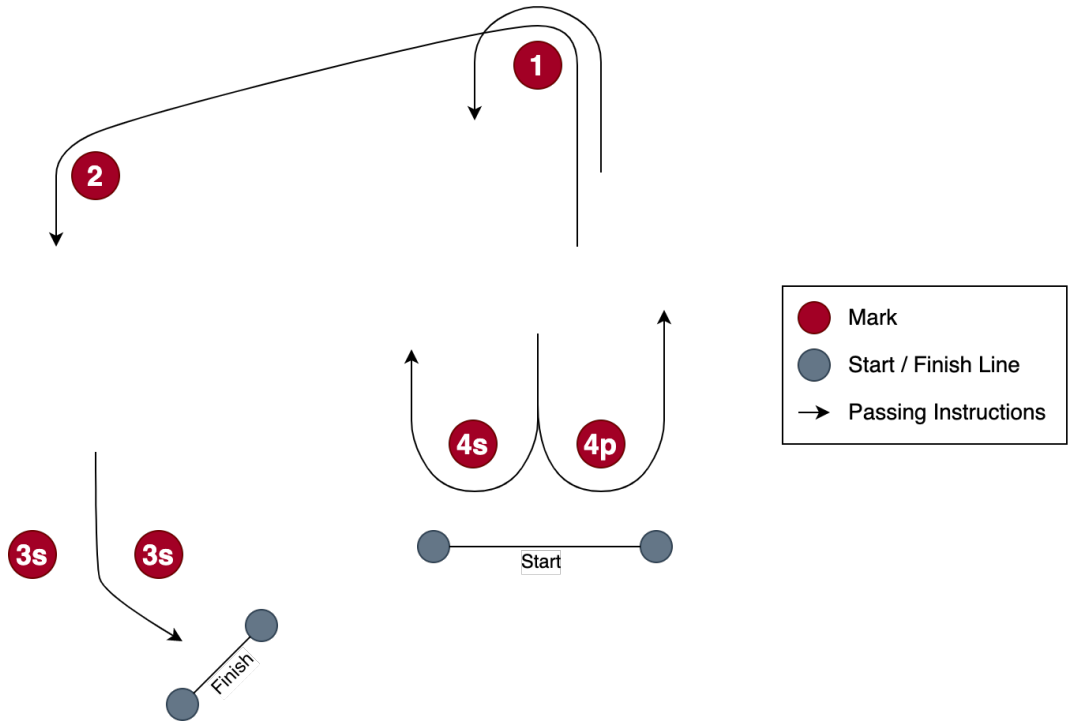

Figure 2.3: Layout diagram of an "Inner Trapezoid" course according to sailing instructions of the Kieler Woche 2022 [3]

I3: *Start - 1 - 4s/4p - 1 - 4s/4p - 1 - 2 - 3p - Finish*
I4: *Start - 1 - 4s/4p - 1 - 4s/4p - 1 - 4s/4p - 1 - 2 - 3p - Finish*

The subsequence *4s/4p - 1* forms a recurring part that requires mark 1 to be rounded several times. Each iteration is also referred to as a lap. Since in I2 the mark 1 is rounded twice, this sequence has two laps, while I3 and I4 have 3 and 4 laps respectively.

The entire race course can be subdivided into segments between the different waypoints. The stretch between two consecutive waypoints is called a leg. In Figure 2.4, a course is depicted schematically, in which the respective marks have to be traversed following the numbering. Whereby the legs of the course are represented by the black lines.

In accordance with the orientations to the wind described in subsection 2.1.1, each leg can also be assigned such an orientation, which indicates the general boat orientation towards the wind along the leg. In most cases, this assignment is unambiguous, but there are exceptions. An example would be the olympic triangle, where two course-legs have

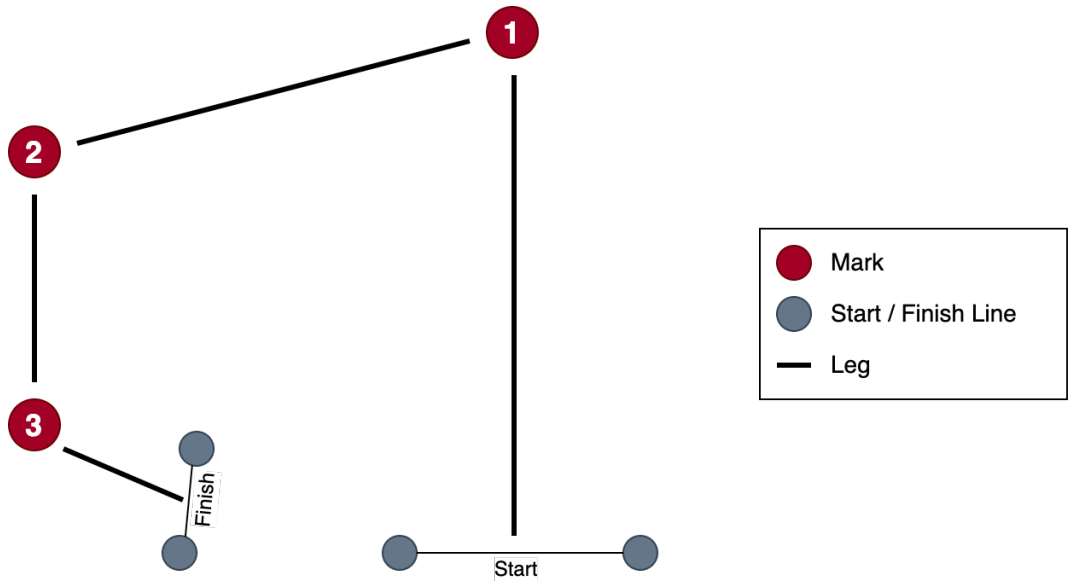

Figure 2.4: Separation of a course layout into different legs

an approximate angle of 135 degrees to the wind, which is why they might lie exactly on the boundary between reaching and downwind, depending on the conditions [1].

The so-called leg-type of a course leg refers to exactly this classification of the orientation with respect to the TWD. In the context of this thesis, the leg-type includes the classification into upwind, downwind, and reaching, but also the difference between positive and negative TWA in the case of reaching legs. This makes it possible to clearly distinguish the orientation of a leg in relation to the TWD in all four possible directions.

2.2 Big Data

Nowadays, Big Data seems to be present everywhere and it is hard to imagine large-scale analytical applications without it. However, it is difficult to say exactly how big data should be defined and where the line is to be drawn. Depending on whom you ask, you will certainly get very different answers. The obvious applications associated with this topic are perhaps Internet of Things (IOT) or self-driving vehicles, but analyzing sailing races is also a prime example. Based on virtually infinite amounts of sensor and measurement data, which are sometimes even collected over months, conclusions must be drawn for both past behavior and possible upcoming actions.

Certainly, as the name itself suggests, one definition would be by the volume of data. However, the general public seems to agree by now that even though volume is certainly the primary property, other characteristics are almost equally relevant to the definition of big data. A widely used and accepted golden standard for this are the so-called 3Vs by Douglas Laney [4]:

- *Volume:* As already mentioned, here it is a question of using a large amount of data. However, there is no unified or precise definition of what big means. In the scientific context, the term is often used to refer to a quantity that cannot be easily processed by the underlying computing system [5]. Since different systems have different amounts of computing power, this leaves a lot of room for interpretation.
- *Velocity:* In addition to the volume of data, the speed at which the data can be collected and processed is crucial. The faster the data can be processed, the more up-to-date analyses and conclusions can be made [4].
- *Variety:* Data can have a variety of different forms. Numeric, positional, or other metadata. Variety refers to the fact that a big data dataset will contain data from a variety of sources and in a variety of different forms. This variance induces the requirement that a corresponding analysis application must also be able to handle a wide range of data types and incorporate them into the results [5].

In addition to these three terms, at least two others have gained general acceptance in the scientific society, which is why it is commonly referred to as the 5Vs of big data. Again, there are numerous other proposals for extending these requirements for big data, but for the purposes of this thesis, it will be limited to further definitions of validity and value.

- *Validity:* To be able to make accurate statements about the information contained in the data, the correctness of the data is essential. Even if much of the data in the dataset is correct, incorrect data can quickly lead to distorted or even incorrect conclusions. Selecting the right data and rejecting the wrong data is therefore one of the most important but also most difficult tasks in working with big data.
- *Value:* The added value in data for a specific application or even the irrelevance of data for another can be the decisive step to enabling efficient and targeted analysis. A large volume of data initially contradicts fast data processing and thus *velocity*.

By carefully reducing the dataset used for the relevant information, the quantity can be reduced without impairing the quality.

2.2.1 Data Reduction

In big data projects, which rely on large amounts of data, it may seem counterproductive at first to reduce the amount of data, but this step in the preprocessing of data streams is one of the most important steps in modern data processing [5]. With a rapidly increasing amount of data from different sources, different sensors, and networks, a compromise has to be made to achieve a high-performance and efficient processing of this data, which ideally happens in real-time [6].

Depending on the area of application, it can be beneficial to reduce the data before it is stored in the dataset and to minimize the stored data to that relevant within one's industry [5, 6]. In other cases, the reduction is carried out before the data is further processed for a specific purpose, in order to keep the reduced data available, even if it is not currently required for an existing purpose; however, this may change in the future [7].

There are many different approaches in which the volume of data can be reduced [6]. Some only reduce the physical size of the stored data and keep the content identical, others actually reduce the data contained. The following section presents some of the fundamental techniques of data minimization.

Feature Selection

Concerning the value criterion of a good big data system (see section 2.2), arguably the most straightforward step to reduce the data set is to limit it to relevant and thus valuable data for the intended purpose by simply sorting out irrelevant dimensions and features of our dataset.

Entries in the dataset are typically stored as feature vectors [7]. An entry may represent an object, an event, or any other self-contained entity. The respective feature vector then consists of all metadata and information recorded about this entity including often those that have no relevance for further processing [8]. Simply stripping out additional

information reduces not only the size but also the dimensionality of the data, which may also be desirable as described in subsubsection 2.2.1 [7].

Data selection, as the name suggests, merely addresses the selection of the correct attributes and features. This may happen in a supervised manner through manual selection of criteria, or through unsupervised learning algorithms that autonomously decide on which data is of value [8].

Compression

Another approach to reducing the overall data volume, besides feature selection, is data compression. The techniques are by no means direct alternatives to each other but can be used simultaneously. In contrast to feature selection, data compression follows the basic idea of preserving the original form of the information while still reducing its size [5].

A wide variety of compression algorithms and methods are available, that produce vastly different results [9]. However, this is mainly due to the fact that different sorts of information and data types require different compression strategies. Some are more universal than others, but efficient compression requires targeted methods. And nevertheless, they are united by a common principle: "They compress data by removing redundancy from the original data in the source file" [9].

In some cases, it is possible to remove redundant data without changing the information contained in the data or its structure. This is the case when the same information is available multiple times or elements can be combined without changing the structure of the data format. In this case, we also speak of data summarization and the resulting data can be used directly wherever the original data would have been used. A good example of this is replacing multiple consecutive spaces in most source code compilers with a single space to reduce the number of characters [9]. But again, it quickly becomes apparent that this only works for certain programming languages where there is no semantic difference between one or more spaces. In most cases, however, compressed data cannot be simply interpreted and further processed, as it is stored in a separate new format. Therefore, decompression is required to restore the original state of the information, including any redundancies. Even if computing power and storage space can be saved during the storage and transport of the data, the additional compression and decompression step generate further computing overhead [5]. It is therefore a question of

deciding in which applications compression saves computing power and at which points in the processing stream it should be used.

Lossless Compression With lossless compression, the entire information of the original file is available again after decompression in the identical state as before the compression. Thus, only a smaller representation of the same content is chosen, which means that no information is lost [9].

Lossy Compression Lossy compression is the counterpart to lossless compression. This variant takes into account that some information is lost during compression, but in return often achieves a significantly better reduction of the original volume. It is mainly used where the transmission of the primary meaning is important and minor differences are barely noticeable, as is the case with audio, video, and images. After decompression, the data is no longer the same as before compression, but a reduced form of it, hence the name lossy [9].

Curse of Dimensionality

The so-called curse of dimensionality is one of the main reasons for reducing the features of its dataset and thus the dimensionality. It refers to the phenomenon of increasing complexity and difficulty in analyzing high-dimensional data. Not only are high-dimensional relationships significantly more challenging or even impossible to visualize and therefore to make tangible for the human imagination, but they can also lead unsupervised algorithms towards generalization and to missing features that are crucial. In the case of self-learning algorithms, this means that the amount of training data required increases exponentially with the number of dimensions [7].

2.2.2 Data Smoothing

Especially when working with sensor data, as is the case here, it is to be expected that there are outliers in the readings, which have no relevance in terms of the actual situation and are only due to technical errors or incorrect measurements. They are also generally

referred to as noise. A good practice is to sort out these outliers before any further processing and hence minimize the risk of them influencing the inferred results.

In terms of boat tracking data, we are faced with two different types of noise. Measurement noise refers to the measurement errors that occur due to inaccuracies in the triangulation of the position, technical interferences, or similar. In contrast, process noise can also be found in the data, which describes deviations due to natural behavior. For example, a human being and also the boat are probably not always able to maintain an optimal course under real circumstances.

For complex relationships or cases where no clear link can be derived manually, unsupervised methods can be used to filter out the noise, such as specialized clustering methods, which will be further described later in subsection 2.3.2.

In many cases, however, a clear linear connection is already apparent beforehand. The most common case is probably with a series of data, such as a time series, as is the case with sensor data. For these applications, so-called smoothening techniques can be utilized. Within a sequence, both preceding data items are available, as well as subsequent data items retrospectively. Thus, the local environment of a single entry in the dataset can be taken into consideration in order to conclude whether the entry fits in there or whether it represents an outlier.

These kinds of algorithms for processing signal data are called filters. One of the simplest filtering algorithms is known as moving average filtering.

The moving average filter looks at a constant set of sequential entries from the raw dataset in each step [10]. This is our sliding window. For temporal datasets, this is usually done by setting a duration in which all values before and after the observed timepoint are included. This duration is also called half-window-duration. To calculate the smoothed value at the point in time in question, the average value of all elements contained in the time window is calculated, which then forms the new smoothed output value [10]. A simplified visualization of this process can be seen in Figure 2.5. The half-time-window duration can be adjusted to specify how fine granular or how intense the smoothing should be. A larger halftime window results in a more intense smoothing, but also in a loss of resolution in the result. Since values before and after the timepoint are required, we cannot reliably smooth the outer regions. If the algorithm is used as an

online algorithm, which means on data that is added in real-time, its results always lag behind the real-time [11] by a duration of a half-time-window.

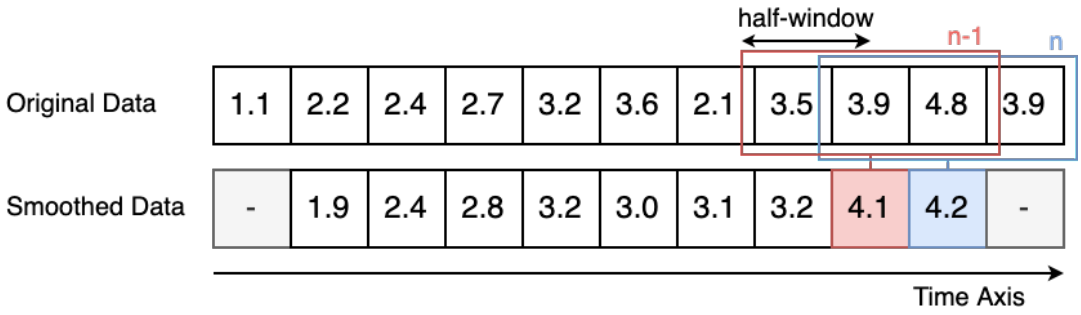

Figure 2.5: Simplified visualization of a moving average filter

Other approaches like the Kalman filter try to overcome this issue by estimating unknown values based on the assumption that noise appears as a Gaussian process [11]. However, different filtering methods often only work on specific datasets that are tailored to them or are very computationally intensive [11].

2.3 Clustering

The use case presents the unique problem of the necessity to aggregate position data from a multitude of competitors and to derive combined conclusions with respect to the position of marks. Data clustering provides unsupervised classification and grouping of entries in a dataset into groups, classes, or objects based on their particular features, attributes, and metadata. Thereby it does not matter whether they are physical or merely abstract objects, as long as they have properties by which they can be associated with each other [12]. Similar entities then form so-called clusters [13]. It should not be confused with supervised classification, which involves classifying a set of pre-labeled data; the focus of clustering is to classify solely by features of the data [14].

Clustering applications are numerous and range from exploratory data analysis, over image processing to machine learning. With this multitude of diverse use cases, however, also comes a wide range of specific requirements for the respective algorithm, which usually makes it a difficult and computationally intensive problem [14].

In order to determine the similarity of two objects in the dataset, most clustering algorithms require a metric by which the distance of the two objects can be measured. Choosing a relevant metric is fundamental to the success of the clustering process in relation to the respective area of application. Most commonly used for this purpose is the Euclidean distance, which allows for the calculation of a distance between any continuous feature vectors. However, the Euclidean distance has the disadvantage that scaling differences between the individual features are not taken into account, which is why it is always advisable to use a metric suited to the problem [14]. The formula of the Euclidean distance is given in Equation 2.1 [15], where d is the distance and p and q are the two elements whose distance is to be calculated. Both elements have n dimensions.

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \tag{2.1}$$

As a result of different approaches and techniques used to create clusters, clustering algorithms vary greatly in their characteristics, performance, and resulting output.

- Fuzzy: While hard clustering algorithms have a strict assignment of which cluster an element belongs to, fuzzy clustering algorithms are able to assign elements to several clusters. The assignment takes place with the help of confidence values, which indicate to what degree an element belongs to a certain cluster [16].
- Incremental: In terms of performance, significant differences can arise as to whether an algorithm allows incremental changes to the data set or requires a complete recalculation of all the clusters each time. Most of the popular algorithms today also have incremental versions.

Two of the most popular clustering approaches are broken down in more detail below.

2.3.1 Partitional Clustering

This subclass of clustering algorithms is probably the best-known class of clustering algorithms.

In essence, they all work by establishing a fixed number of clusters to be determined and then randomly populating them with groups of connected elements. The elements are

then moved between the clusters until an optimum is reached and no more improvement is foreseeable. For this purpose, a criterion is defined, which is attempted to be optimized [14].

This comes with the disadvantages that the number of clusters to be searched for needs to be known in advance and that the optimum found may only be a local optimum. To avoid finding only a local optimum, the algorithm can be run several times so that the chances of finding the global maximum are higher [14].

Since the data is only assigned to the clusters, it is not necessary to store all elements of the dataset in an additional data structure. The assignment and changes to it are therefore computationally inexpensive [14].

Most of the algorithms and applications of this category are based on the two best-known representatives, the k-Means and k-Metroid algorithms [17].

k-Means

K-means is one of the most widely used algorithms for dataset partitioning [18]. Each cluster is identified by its center, the so-called centroid, which corresponds to the mean value of all elements contained in the cluster. The overall goal of the algorithm is to distribute the elements among the clusters in the best possible allocation so that the internal variation of the elements among each other is minimized. The number of clusters k is predefined and it is simply a matter of finding an optimal placement of these centroids [18]; hence the name k-Means.

Initially, k elements of the dataset are now randomly selected as initial centroid elements. All other elements are now iteratively assigned to the centroid closest to them. For this, k-means is normally using the Euclidean distance, which has already been listed in section 2.3. This strategy is based on the "square error criterion" [19]. After all of the elements have been assigned, the mean values of the clusters are recalculated and the centroids are adjusted accordingly. The last two steps of assigning all points and recalculating the centroids are repeated until the assignment does not change any further [14].

The widespread use of the algorithm is probably because it is exceedingly straightforward to implement and understand in its simplest implementation, yet this poses the problem

that it is equally inefficient and inperformant. However, due to the popularity of the algorithm, there is an almost endless amount of improved variants, which aim to minimize redundant cycles and unnecessary work of the algorithm and to optimize it in terms of speed, computation and memory complexity [18, 20].

k-Metroid

This variant of the partitioning algorithms is also called partitioning by similarity, in contrast to partitioning by distance, to which k-means would belong. The process however is very similar, except that a different criterion is used to find the optimum [21].

Instead of the mean of all cluster elements, the centroid, one element is always used as the center of a cluster, the metroid. This is the element that is most similar to all other elements of the cluster [21]. Since this comparison does not require a distance metric with continuous values, k-Metroid also works on data sets for which the calculation of such a metric is not possible.

For the approximation of the optimum, similar to k-Means, a selection of k random elements is first used as metroids [17]. All further elements are now assigned to the most similar metroid based on the criterion. The convergence is done by randomly using a different metroid. If the general similarity of all elements to the metroid is higher than to the previous one, this metroid is used. This calculation is done with the help of a cost function. Similar to k-Means, this is repeated until there is no further improvement [17, 21].

Since k-Metroids uses actual data elements from the dataset as representatives for a cluster, it is less prone to error due to noise, in contrast to k-Means [17, 21].

2.3.2 Density Based Clustering

While many clustering algorithms, like k-Means, generate spherical clusters because of their technical design, this rarely reflects the form of identifiable groups in real-world datasets. Especially with spatial data, it is often much more likely to observe other arbitrary shapes in the point clouds. In the case of geodata, for example, this can be due to environmental constraints such as mountains, rivers, and other obstacles.

Similarly, it is feasible that one cluster partially or even completely encloses another cluster. Distance-based algorithms would presumably merge those clusters into one, even if clear boundaries are perceptible to the human eye. This problem is visible in Figure 2.6, where humans can clearly identify three clusters, but a conventional clustering algorithm has its difficulties. The clusters which were found are represented here by the different colors. With increasing multidimensionality of the data, this problem intensifies [21]. A solution to this problem is provided by density-based clustering algorithms.

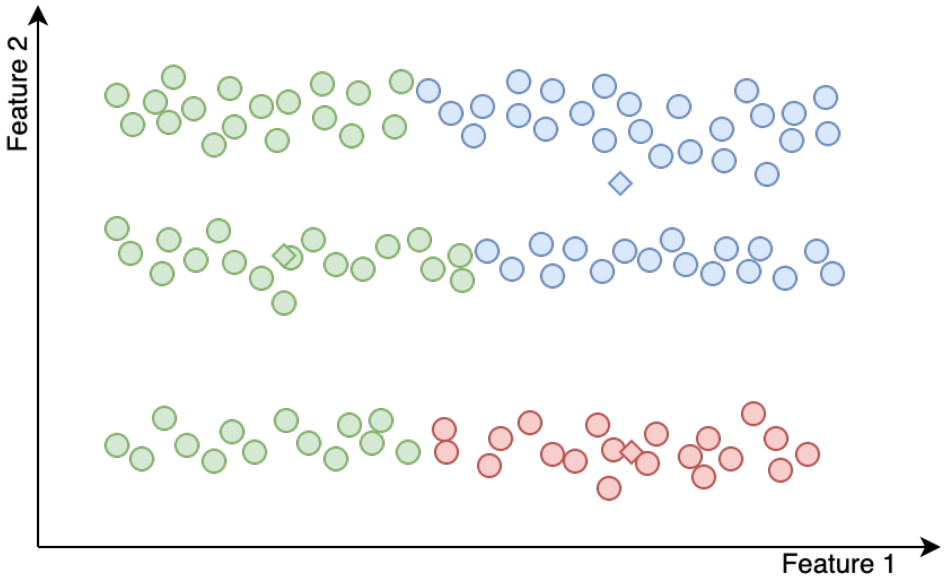

Figure 2.6: Non-density based clustering algorithm with non-elliptic clusters

Density-based clustering algorithms do not require a predefined number of clusters. They freely determine potential clusters based on the density of entries in the multidimensional space of the dataset. If there exist entries that exceed threshold values in terms of their density, it is assumed that they are clusters. Between the connected dense regions, or rather between the clusters that have been identified, some regions are not as dense. The data points contained in them are referred to as outliers or also more generally as noise [16]. This characteristic clearly distinguishes the density-based clustering techniques from the partitioning ones, in which the entire dataset is divided into clusters.

Through the technique of defining the clusters by connected dense areas and not by single center elements, the clusters can take on any shape [21]. They are therefore more adaptable to real-world situations.

Accepting noise in the dataset also helps in understanding data from the real world. Because of measurement errors or other inaccuracies, data points often occur that may be interfering or even misleading during further processing of the data. By accepting noise values that cannot be assigned to any cluster, this problem is reduced.

One of the most popular and well-known variants of density-based clustering algorithms is A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise (DBSCAN) [12, 21, 22], which we will further explore below.

DBSCAN

For the purpose of explaining the DBSCAN algorithm, a representation of the dataset as points in a two-dimensional space is chosen, showing two features plotted on the x and y axes. Accordingly, the distance in two-dimensional space is chosen as the distance metric. The algorithm of course also works with higher dimensional datasets and even is particularly efficient there in comparison to other clustering methods [12].

The algorithm is based on two parameters. The maximum distance of two neighbors for them to be considered connected and the minimum number of interconnected neighbors required to create a cluster. Based on these parameters, a *cluster* is composed as follows. The core of the cluster is formed by one or more elements that have at least $MinNum$ neighbors within a radius of Eps of them, according to the metric used. In addition, other elements can be part of a cluster, if they are neighbors of a core element, but do not have enough elements in their own neighborhood to be a core element themself [23, 24]. While these elements are not able to join clusters, they still belong to a cluster. Since the original implementation of DBSCAN is a hard and not a fuzzy clustering algorithm, these elements can only belong to one cluster and not several at the same time. The above-described structure is visualized in Figure 2.7.

In order to better understand how DBSCAN works, some formal definitions based on the previously explained circumstances need to be understood beforehand.

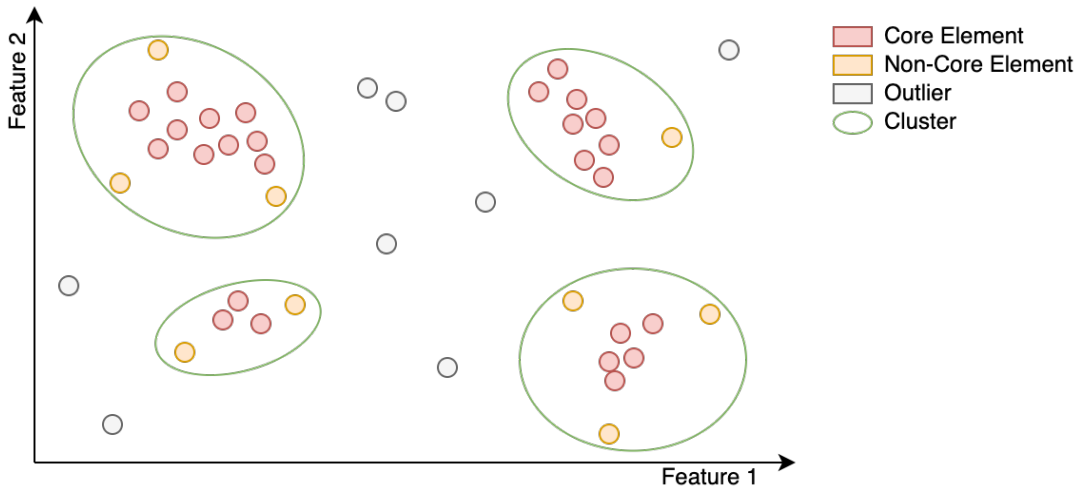

Figure 2.7: Example of a two-dimensional dataset classified by DBSCAN

- *Eps-neighbourhood of an element:* The Eps neighborhood denotes all elements of the dataset that lie within the distance Eps from the examined element [23]. The red marked element in Figure 2.8 is our considered element, while all orange elements are neighbors of the red element.

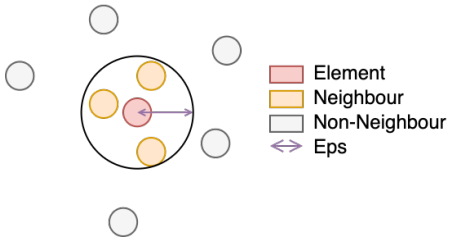

Figure 2.8: Eps-neighbourhood

- *Directly density-reachable:* An element p is directly density-reachable from element q if q is a core element and p is in its immediate neighborhood. For two core elements, this relationship is symmetric, which is not the case if p is a non-core element [23].
- *Density-reachable:* Similar to directly density-reachable, except that in this case p does not have to be in the immediate neighborhood of q, but rather a chain of arbitrarily many other connected core elements lies between the two elements [23].
- *Density-connected:* Density connected refers to cases where two non-core elements are connected by a cluster but are not density-reachable to each other because neither object is a core element. However, the elements are connected by one or more core elements. In this case, we call the elements density-connected. This is obviously a symmetrical relation [23].

A more detailed definition of these terms can be found in [23] including edge cases that may arise.

Below is a summary of the resulting algorithm from the original paper [23]. The algorithm starts with a random element from the dataset. First, all elements are retrieved which are directly density-reachable from the start element, hereafter referred to as neighbors. If this is less than $MinNum$ neighbors, then it is not a core object and the process is repeated with another object that has not been visited. The considered object is marked as already visited. If it is a core element, however, all density-reachable elements are determined by traversing the neighbors, which then together form a new cluster. The elements are also marked as visited. If a core element of the new cluster is density reachable to a core element of an existing cluster, the new cluster is merged into the existing one. This process is now repeated with all other elements that have not been visited yet. In the end, all elements are either core elements of a cluster, a border element of a cluster, or not assigned to a cluster, which means that they are noise.

An important factor for the accuracy and validity of the results of the clustering algorithm is the choice of a suitable distance metric and appropriate values for the parameters Eps and $MinNum$. A disadvantage of the algorithm is that an appropriate distance metric can often only be chosen with in-depth domain knowledge and that the parameters have to be equally applicable to the entire dataset [23]. Methods to manually find a suitable Eps are also presented in [23]. In addition, a $MinNum$ of 4 is recommended for all two-dimensional datasets.

Even though the basic concept remains the same, there are numerous improved implementations and additions to the original DBSCAN algorithm. In the following, an extension for the incremental use of the algorithm will be addressed.

Incremental DBSCAN

Especially in modern big data and data mining environments, such as data warehouses and data lakes, datasets are not static, but data is collected over time and thus the dataset is extended [25]. Dynamic datasets pose the problem that, when using DBSCAN, it would be necessary to perform the entire cluster calculation again with each insertion and deletion. However, due to the often enormous amount of data, this is computationally intensive and simply not desirable [12, 25]. A remedy for this is provided by incremental

implementations of the conventional DBSCAN algorithm. Instead of recalculating the connections of the entire dataset each time, they restrict their examination to elements that are in the immediate vicinity of the newly added or deleted element [25]. Therefore, updates of existing datasets are uncomplicated and computationally efficient, regardless of the size of the entire dataset [12].

Technically, this is implemented by keeping a dictionary that records for each element how many other elements are in its immediate neighborhood. This directory is filled as soon as an element itself is visited or updates in its neighborhood take place so that it is always up to date. With its help, it is possible to determine which elements would become core elements after the insertion of a new one, and which elements would lose their core element status after the deletion of an existing one. This way it is possible to calculate the number of elements that are actually affected by the update [25].

In both cases, the number of affected objects can then be used to decide as to which steps are necessary.

Insertion When inserting an additional element, only new density connections may be created, but in no case, any will be removed [25].

- *Is noise:* If no elements are affected, then the element remains alone, and it is accordingly an outlier [25].
- *Forms new cluster:* In case new core elements are created and none of the affected core elements is already part of a cluster, a new cluster is formed with all affected elements, that do not belong to a cluster yet [25].
- *Absorbed by existing cluster:* If all affected core elements belong to the same cluster or to no cluster at all, all affected elements, with no cluster, are absorbed by that cluster [25].
- *Merges two existing clusters:* However, if the influenced core elements belong to multiple clusters, those clusters and all affected elements, that do not belong to any cluster, are merged into one cluster. No further calculations need to be performed for the merge. It is sufficient to simply transfer all elements of a cluster to the new one [25].

Deletion When an existing object is removed from the dataset, existing density connections may be removed contrary to the insertion, but new ones will never be created [25].

- *Remove noise:* If no core element is significantly affected by the removal, it has to be an outlier or border element, which can be easily removed [25].
- *Remove from cluster:* On the other hand, if the object is a core element, meaning that all its neighbors are directly density-reachable to each other, or if a neighboring core element is thereby broken up, the surrounding elements may become noise or non-core elements in the case of previous core elements [25].
- *Split cluster:* In case not all influenced objects are density-reachable from each other, it has to be examined whether clusters have to be split into several clusters by checking the involved elements individually in regards to their density connections [25].

By handling these separate cases, the clusters retain the same properties as a non-incremental algorithm and allow for dynamic changes to the dataset without much computational overhead [12].

2.4 Geospatial Data

Geospatial data is information with a reference to a specific location. This means that metadata and the identity of an entity are linked with data relating to the spatial position. These entities may be physical objects, but they may also be events or abstract marks [26]. Examples from the context of the application include the boats themselves, course marks, or even maneuvers that have been sailed.

In the context of sailing, a spatial reference only contains data on the two-dimensional position, since the altitude is negligible in most cases.

2.4.1 Coordinates

The most commonly used system for specifying spatial references is a coordinate system. The entire earth is divided evenly into a coordinate system, which makes it possible to define exact positions, the so-called Geograpical Coordinate System (GCS).

Earth is subdivided by latitudes, lines parallel to the equator and longitudes, which are orthogonal to the latitudes and go from one pole to the other [27].

To specify a position, a tuple consisting of a longitude and a latitude value is used. In the following, we will also refer to this as a coordinate.

2.4.2 Distances

Due to the curvature of the earth and the resulting distorted representation of coordinates, the calculation of distances between two coordinates presents a certain difficulty. Just as nautical maps are distorted in such a way that every measured distance at every point on the chart corresponds to the same length [27], a corresponding conversion is also necessary for the algorithmic calculation of distances between two coordinates. In order to nevertheless be able to make approximate distance calculations between two positions on the earth's surface, the earth is modeled as a sphere [27, 28].

One of the most rudimentary approaches to calculating a distance is the so-called Rhumb Line Distance. In this case, the aforementioned distortion, which is common in nautical maps, is made use of in order to be able to use a straight line to measure the distance [27].

Another more practical approach for calculating distances on a sphere is the Great Circle Distance. The great circle is defined as all bearings along which the earth can be split exactly into two equal parts [29]. For the calculation of a distance, an arc can be found on any great circle that passes through the two positions. Using spherical trigonometry, the distance can be calculated, for example using the spherical law of cosines, shown in Equation 2.2 [27, 30]. It is the formula for the calculation of distance d in kilometers between two coordinates given in latitude (lat) and longitude (lon) on a sphere with the radius r in kilometers

$$d = r\,arccos(sin(lat1)\,sin(lat2) + cos(lat1)\,cos(lat2)\,cos(lon2 - lon1)) \tag{2.2}$$

However, for the calculation of short distances, as is predominantly the case in sailing, a less computationally intensive calculation exists with the help of the Haversine Formula stated in Equation 2.3 [28, 31]. The identifiers used are the same as before.

$$d = 2r\,sin^{-1}\left(\sqrt{sin^2\left(\frac{lat2 - lat1}{2}\right) + cos(lat1)\,cos(lat2)\,sin^2\left(\frac{lon2 - lon1}{2}\right)}\right) \tag{2.3}$$

3 Concept

First of all, the solution needs to be conceptualized before it can be implemented later on. For this purpose, a short analysis of the requirements and relevant use cases will be presented first. Afterward, the general idea behind the realization is presented, as well as detailed sub-concepts including the decisions taken in their execution.

3.1 Requirements

For the determination of the requirements, a consideration of possible utilizations is carried out first. Different usage scenarios require different kinds of information about the marks used in a sailing race track. Three concrete possible use cases are presented below:

1. A small sailing club for young people does not have the capacity for elaborate tracking, but every competitor in a club race has a GPS-enabled smartphone and the local pier is equipped with a wind measuring system. Thus competitors and wind are sufficiently tracked. Only the positions of the marks are unknown. In order to be able to offer the parents and teenagers analytical insights after the race, the missing positions have to be inferred from the remaining data.
2. During a fully tracked race with dedicated mark tracking, the tracker of one of the marks fails during the race. This is not noticed while the race is in progress, which is why the corresponding data is missing. The resulting analysis data of the race is erroneous as well. To recalculate the correct results, an estimated position of the mark needs to be entered first. Instead of guessing this by eye, the position may also be inferred from the rest of the data.
3. For a TV production, live data from a race is to be collected and analyzed. Both the tracking data and the real-time results are to be used for visualization in the overlay of the video feed. However, the organizer's project management failed to plan correctly and no trackers were allocated for the marks. Instead of showing

no tracking on the video, the time of transmission of the video is to be delayed by 30 minutes, so that the positions of the marks can be determined during this time using the remaining tracking data.

The required information that such a solution would have to provide can be derived from the concrete use cases.

For a simple visualization of the competitors and the race course on a map, it is sufficient to know the positions of the individual marks in order to display them alongside the competitors. However, a more sophisticated visualization may also benefit from semantic information about the race course, such as the composition of the waypoints and their order. This would, for example, enable highlighting of the upcoming waypoints. It might also be useful here to be able to display the respective passing instructions as well.

Especially when it comes to automated analysis of the tracked race, this additional information is essential. Without knowing how to go around a mark or how the respective waypoint is composed, it is not possible to give precise insights on which competitor was in the lead on which leg of the course or what the respective rounding times of the competitors are, and much more. Information on the succession, position, composition, and passing instructions of the waypoints is therefore indispensable.

In all applications, finding a position that is as close as possible to the actual position is desirable, but choosing a representative position is more important. This means that the estimated mark may be a few meters away from the actual mark, but the rounding times of the competitors are almost identical.

To be able to compete in a race, there have to be instructions on the course to be followed. Consequently, there is always a definition of the race course prior to the race. An exception to this may be unofficial races, where the competitors define the course as they please on the water. In all common scenarios, it is therefore also feasible to enter the general course specifications and merely automate the process of finding the appropriate positions. As explained in section 1.2, both of these cases will therefore be examined separately. The determination of the positions under the specification of the course layout stands is however the main focus and objective of the thesis.

Any use case might also be performed live during the race. In the case of visualization, this is probably only useful to a limited extent, since it is only possible to work with

historical data, but at least for analysis, an evaluation with the lowest possible latency is desirable.

In conclusion, the following tasks and requirements can be defined:

- Identification of marks
- Determination of the position for identified marks
- Assignment of the marks to waypoints
- Derivation of the passing instructions of the waypoints
- Execution with the shortest possible time delay after the arrival of new data

3.2 General Idea

The objective is to derive waypoints based on anomalies and patterns in the competitors' tracking data.

Anomalies that occur within the tracking of the competitors consist of unexpected maneuvers that take place instead of the logically expected maneuvers. These can be used as an indication of a motive other than simply sailing along the current leg and that this may involve bypassing a mark. To better understand this, it is first necessary to define which maneuvers are to be expected and which maneuvers are unexpected.

The expected behavior of a sailor on a sailing race course is to sail as directly as possible to the next waypoint. Even if the software does not yet know what the next waypoint is, the competitor is expected to maintain a roughly constant general direction towards it. In turn, unexpected behavior would be the divergence from this straight course, for example, in order to sail around a mark. In the case of the simple mark and gate, the passing of a waypoint is always accompanied by such a rounding. This behavior forms the basis of the concept developed in this thesis.

However, there are a number of other situations that lead to a deviation from the expected behavior, although no mark is present. Simply nonsensical sailing behavior in which a course is set that deviates momentarily from the actual destination. The capsizing of the boat and subsequent drifting away from the intended direction due to wind and

current is also considered an anomaly in tracking. Even racing rules such as sailing a circle as a penalty present such a deviation [1]. All these deviations can be summarized as short-term changes of course in which, unlike transitions between legs, a different course is taken only for a short time. However, another example would be the avoidance of obstacles. In the case of another boat, this is only a short-term change of course, but in the case of an island, the situation is different. In the latter case, however, they would usually be explicitly integrated into the course layout.

Additional risks of misjudgment are posed by unexpected maneuvers in the tracking, which did not actually take place. Due to the technical nature of the tracking systems, incorrect data may be present within the tracking data of the competitors. Various circumstances can be the cause of incorrect sensor data. Because a wide range of tracking services and methods are supported, all of which are based on different devices and systems, the causes can vary widely. The effects, on the other hand, are usually similar. Either the timestamps of the sent data are not quite correct or positions differ marginally from the actual ones. In both cases, slight jumps in the registered path are caused. Larger discrepancies have already been eliminated at a different stage. These jumps can lead to short-term leg-type changes, which are reversed again shortly afterward. Because of similar reasons as with incorrect tracking data, the received information may be missing completely. However, typical situations in which tracking data is missing for a period of time are when the tracker loses its connection to the communication network, regardless of whether it is using a cellular or a proprietary network. Or if the tracking device is underwater or otherwise shielded off and for this reason cannot send any data. In these cases, the missing data is usually delivered as soon as the tracker regains a connection and therefore not completely lost. In other cases, for instance, after the software of the tracker crashes itself and the tracker is rebooted, information for a part of the tracking path may be completely lost. If a maneuver or leg transition has occurred during this period, it is not possible to determine its position.

Another issue occurs with candidates who are participating in the wrong race, or have not competed in any race at all but are present on the water, producing irrelevant tracking data. And even if trackers never made it onto their designated boat and remain on auxiliary vessels or are still in port, they will generate false data as well.

A respective approach is based on the elimination of irrelevant deviations, by exclusively identifying long-term changes in the general direction of travel of the competitor. Likewise,

directional changes of various competitors should accumulate at the transitions between two-course legs. What is needed is a classification of the orientation of the course leg on which the competitor is currently situated. At positions where there is a change to another course leg, there should also be a change in the course leg orientation. As already explained in subsection 2.1.2, the orientation of a course leg is called leg-type. The position can be determined by the timepoint of the leg-type change, as well as the rounding direction by examining an area before and after the leg-type transition. By combining the position of the leg-type transition, rounding direction, semantic information regarding the order of the leg-type changes and nearby ones of other competitors, a sequence of waypoints corresponding to the course sailed can be determined.

Ultimately, the application to be developed represents an end-to-end data processing stream that generates estimates for the most likely waypoints from raw wind and competitor tracking data. This data handling process can be divided into three main sections; the preprocessing of tracking data, the classification of position candidates, and the mapping of candidates to a course layout.

3.3 Data Preparation

The preceding preparation of the data fulfills two important functions. On the one hand, it breaks down the enormous volume of data to the essential information that is needed afterward and eliminates possible noise and inaccurate data from the dataset.

3.3.1 Converting positions to directions

The first part is relatively straightforward. Given are the positions of each competitor as well as the COG and SOG, mapped on a time axis. For further processing, however, only the leg-type of the current course at the positions is relevant. By selecting and combining features strategically, the overall volume of data can be significantly reduced.

As described in subsection 2.1.1, the determination of the boat's current Point of Sail (POS) is based on its current TWA. The TWA, however, cannot be derived directly from the tracking and wind data, since only the COG is known, which is also influenced by current and drift and does not correspond to the actual keel line of the boat. Therefore, only an

approximation of the TWA of the boat can be made. For a possible determination of the leg-type, this should be sufficient, although it leads to a potential margin of error.

To derive the POS from the TWA, two threshold values are used, which are set as boundary values for the three directions of travel. All angles smaller than the first threshold indicate that the direction of travel is upwind, and all angles larger than the second threshold indicate a downwind direction of travel. Consequently, all angles between the two thresholds are reaching.

Choosing suitable threshold values, on the other hand, is less trivial. It is obvious that the downwind threshold has to be larger than the upwind threshold. However, the optimal values for this vary greatly depending on the respective boat type, but also the wind speed. In a first approximation, fixed threshold values can be defined initially, which will present acceptable results; more on this in subsection 4.4.1. However, it would be better to accumulate the values adjusted to the boat class and conditions from the data of past races and to make a choice of threshold values on the basis of these. Since there will be situations in which no data from the past is applicable, a fallback to reasonably chosen static values is advisable in any case.

By determining the TWA and looking up the vessel's direction, it is now possible to generate a time series in which the current leg-type is determined at each tracking fix of a competitor. Because in most cases the same course or at least the same POS is maintained for longer periods of time, a large number of successive entries with the same content will be generated. Those consecutive entries can be reduced to one entry using simple compression techniques without changing the meaning of the information [32]. The distance traveled in one direction in a single stretch remains the same, only the form of representation is reduced from several entries to one.

Only by selecting relevant features, converting them to information relevant for further processing, and compressing this information, the total volume of data is drastically reduced without reducing its underlying value.

3.3.2 Detecting and removing outliers

A variety of causes may lead to single incorrect fixes, which suggest that the competitor briefly changed direction, or the competitor in fact did but shortly thereafter changed

back to his original course. These outliers should be sorted out as accurately as possible before the data is evaluated, as otherwise incorrect conclusions could be drawn from them.

Since distinct states are used for the leg-types and it is to be assumed that course legs always have a certain minimum length, it is relatively easy to sort out outliers on the basis of their length. Complicated procedures for noise reduction are therefore not necessary since a simple relationship is already known which describes problematic entries. A corresponding data smoothing algorithm is therefore sufficient and will probably do this task just fine.

Due to the data being time-sequenced, a moving window filter, like the moving average filter, is suitable for this purpose [32]. Given the distinct values, however, it is not necessary to produce continuous smoothed outputs, but it is sufficient if one of the distinct values is returned. Two possible variations of the moving window filter designated for this specific application are described below.

- **Variation 1:** The simplest and therefore most efficient way to sort out short jumps from a data stream is to simply check in the time window under consideration whether all elements share the same direction. Due to our preprocessing in subsection 3.3.1, this is always the case if only a single entry lies within the time window. Only in this case, it is assumed that a uniform state has been reached and the respective entry is appended to the smoothed output [32]. However, this variant has the disadvantage that in the case of jumping states, the specific point in time of a rounding can be pushed back considerably if no new sustained state is reached beforehand.
- **Variation 2:** Alternatively, a methodology based on the moving average filter can be used, in which the predominant direction, based on its total duration of presence in the time window, is always determined and appended to the output as a smoothed value. The calculation of the total durations, in which all states were present, requires a more intensive computation than the criterion from variation 1. In edge cases, where the occurrence of multiple directions is identical and no unique state is reached, a smoothed result can now be appended, even though no stable state has been reached at all.

In any case, omitting individual incorrect entries or converting them to the correct state will result in consecutive identical entries. For this reason, the compression procedure already known from subsection 3.3.1 must be used here as well. The resulting output series only contains a single entry for each completed course leg. Accordingly, the transitions between two legs can be defined by the time points at which a new entry in the series begins.

3.4 Position Inference

At this point, a dataset is prepared on the basis of which the search for patterns that indicate the presence of marks can be initiated.

3.4.1 Grouping leg-type changes to individual marks

To get a general idea of where the marks are likely to be, it is possible to cluster the positions of all the leg-type transitions of all the candidates in a race together. The result would be a mapping of the transitions to each other, so that from each of the resulting clusters separately the expected position of a cluster's most likely mark position can be deduced.

Different clustering algorithms produce different results, which have both advantages and disadvantages for the intended application. They will be compared in the following.

k-Means Since we ultimately want to create an assignment of the leg-type transitions to the individual waypoints, the use of partitioning clustering is not far-fetched. If the overall layout of the course is given, the number of marks and therefore clusters is also known. However, this is a major obstacle if the solution is to work also without the course configuration being specified. The primary factor in determining related leg-type changes should be the distance between their positions. K-Means uses square-root-distance as a criterion for determining clusters. Following subsection 2.4.2, however, this only reflects the real distance on the water to a limited extent due to the unequal scaling of both coordinate axes.

k-Metroid In contrast to K-means, k-metroid also allows for alternative distance metrics. Since no new mean element has to be created, but one of the elements of the dataset forms the center point, it is also possible to include states like the direction of travel or rounding instructions in the similarity analysis. However, k-metroid also depends on a spherical shape of the cluster. For waypoints with sharp turning angles, this is also often approximately the shape of the incident positions and sufficient for an unambiguous classification. In cases, as shown in Figure 3.1, where the angle is flatter, a further scattering of the positions is to be expected increasingly, since the actual direction changes are spread more widely. In these cases, the shape of the desired cluster is more like an elongated ellipse or even a banana.

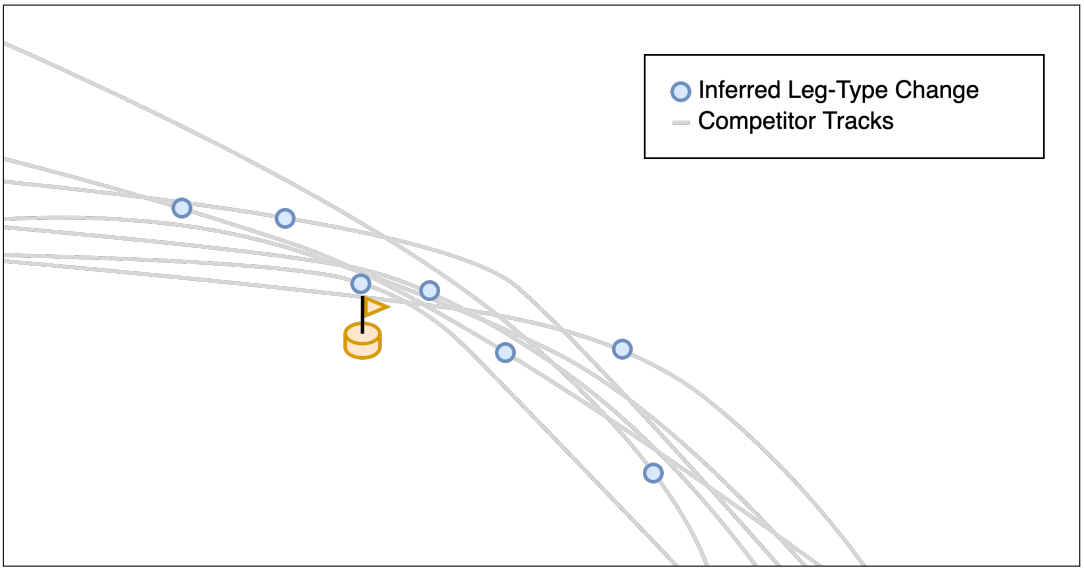

Figure 3.1: Widely scattered leg-type change positions of several competitors when rounding a mark in a flat transition angle

DBSCAN Due to the fact that DBSCAN clusters can take any shape, it is in this respect strongly advantageous to the other two algorithms. Furthermore, it offers the possibility to sort out further outliers directly during clustering. These are usually not noise, such as that filtered out in subsection 3.3.2, but predominantly valid leg-type changes. However, it can be advantageous to exclude individual valid leg-type changes from the result calculation if they are too far away from the overall volume of entries in the cluster. This occurs, for example, when individual competitors round the mark very far to the outside. Since this is a correctly recognized leg-type transition, we do not want

to exclude this in the previous step, nevertheless, they may complicate the calculation of the correct mark position with negative effects on the result. Moreover, DBSCAN is very versatile in choosing a suitable distance metric, as is the case with k-Metroids.

In terms of performance, differences are negligible with optimized modern implementations of the algorithms. Regarding accuracy, DBSCAN provides the most precise results in the required use case, due to the previously mentioned advantages.

Simply clustering all results together would mean that for marks that are used more than once, the data of all roundings would be taken into account. At first glance, this seems like a good idea, but it leads to several problems down the road.

In the case of marks that have to be approached or rounded in several laps from different directions, the leg-type transitions may be on one side in one lap and on the opposite side in another lap. This makes it very difficult to approximate a reasonable mark position based on the positions of the competitor tracks. If the leg-type transitions were always approximately equally distributed around the mark, one could simply take the mean value of these positions. This is not possible due to the fact that different distributions exist on different sides of the mark depending on the number of usages of the mark. If we consider only the leg-type changes associated with a single waypoint rather than all those associated with the mark, much more consistent patterns can be observed, which greatly simplifies the derivation of a potential mark position. More on this topic in subsection 3.4.2.

One way to implement this is to include the rounding direction and the transition type in the clustering of all leg-type transitions. Since this distinction would have to be unique and would not leave room for floating transitions, their priority in the distance metric would be even higher than the actual distance of the positions. Although this approach would certainly work, it is not necessarily the most performant. Since the distinction is obviously not unsupervised, but a hard case distinction based on domain knowledge, it is possible to perform it before clustering. If the clustering is performed separately for each combination of rounding side and transition type, this reduces the total amount of data for each clustering step considerably and limits the unsupervised inspection of other elements exclusively to those that are actually eligible.

By presorting the leg-type changes potentially belonging to a mark, the question arises of why clustering is needed at all and whether the set of relevant data has not already

been identified. As explained before, sorting out points that are too far away is still an important aspect. Depending on the choice of parameters for the clustering algorithm, it is also common for more than one cluster to be identified in the subset. This is by no means an error, but an adequate representation of the actual situation. If a small but nevertheless significant fraction of the competitors decide to take a somewhat unconventional path around the mark, as is the case in Figure 3.2, this can lead to the formation of a separate cluster in addition to the main cluster. In all the cases observed, however, the main cluster is still the one that should be decisive for finding the mark position. Therefore, the largest and thus most representative cluster of a subset is always selected in these cases.

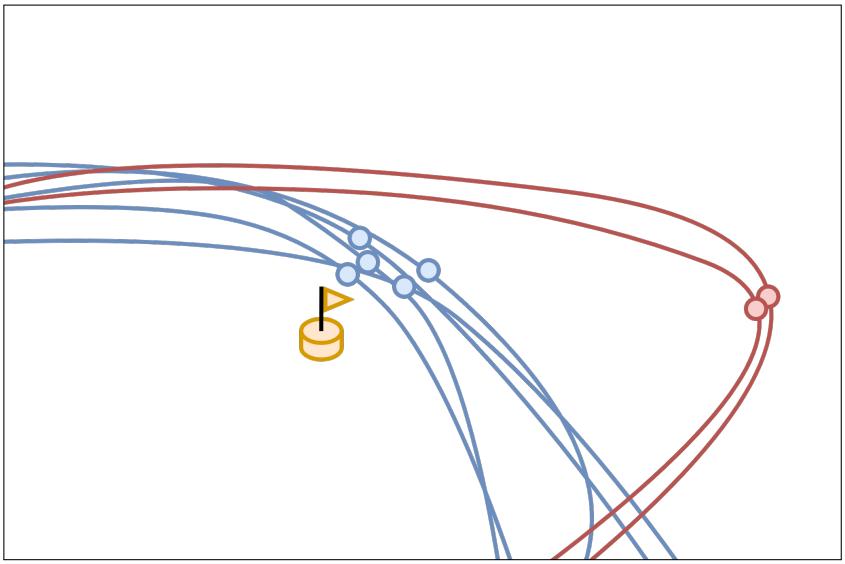

Figure 3.2: Positions of leg-type transitions of multiple competitor tracks form separated clusters for the same mark (The lines represent the tracks of the participants, the dots the respective leg-type transitions and the colors indicate various cluster affiliations)

To summarize the methodology, all leg-type transitions are first presorted by their side of rounding and by the type of transition. The DBSCAN algorithm is then applied to these subsets to identify potential clusters that could represent the respective mark. Likewise, outsiders that deviate too much are sorted out here. Subsequently, the largest cluster from each subset is selected, which then forms the set of meaningful data for further processing.

3.4.2 Estimation of a likely position

After a selection of all leg-type transitions relevant for a given waypoint has been made, a position at which the mark might actually be located must be determined separately for each mark.

The simplest way to obtain a single position representing a whole collection of positions is to take the mean of those individual positions. The position calculated in this way should also not be that far from the actual position. However, since all positions of the leg-type changes are located on the tracks sailed by the competitors themselves and since they usually sail past the mark and not exactly over it, the mean cannot be the exact position, but always an outward projection of it. So it is necessary to find a solution that now finds that offset to the inside of the rounding starting from the previously calculated mean.

The first step is to determine the direction of the offset. In addition to the mean position of all leg-type transitions, the mean of all travel directions at the time of the leg-type transition can also be calculated for this purpose. Now an average position including the corresponding average direction of travel is available. In order to determine a bearing that points to the inside of the rounding, it is sufficient to use an offset of 90 degrees to the direction of travel. Depending on the rounding side correspondingly positive or negative. The resulting mean direction, indicated in red in Figure 3.3, should roughly point to the center of all curves, where the position of the actual mark is expected to be.

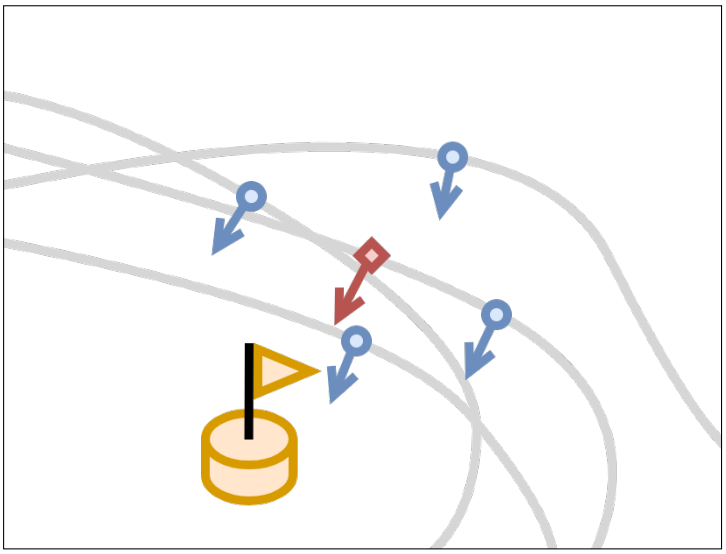

Figure 3.3: Normal directions to the rounding side of all leg-type transitions

All that remains is to determine how far the position must be offset inwards in order to be as close as possible to the original mark. Generally speaking, the closeness of the mean

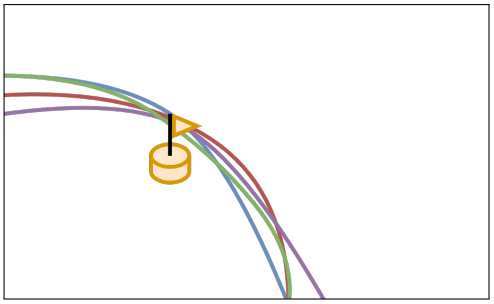

Figure 3.4: Tracks with low variety

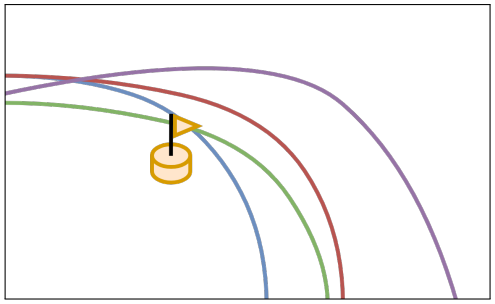

Figure 3.5: Tracks with high variety

position can vary greatly depending on the variance in the paths taken by the candidates. If all candidates are really close to the mark, as shown in Figure 3.4, the mean position is also close to the actual position. In many cases including the one pictured in Figure 3.5, however, some of the candidates also sail larger curves around the mark. For example, this may be due to right-of-way rules or simply poor performance. Because of this fact, however, it is not possible to simply offset the mean inward by a fixed value.

In order to still estimate where the position of the mark is, two circumstances are utilized. For one, it is in the interest of the competitors to round the mark as close as possible to save distance as well as the rule that it is forbidden to touch the mark. Assuming that none of the competitors will pass in front of the course mark, it can be assumed that the leg-type transition which lies furthest in the direction of the previously determined direction will probably be very close to the course mark. Thus, projecting the furthest inward mark onto the mean bearing, as seen in Figure 3.6, yields a fairly accurate position candidate. Since the competitors will not want to run over or touch the marks, the projected position needs to be moved slightly further inward. And even in the case that

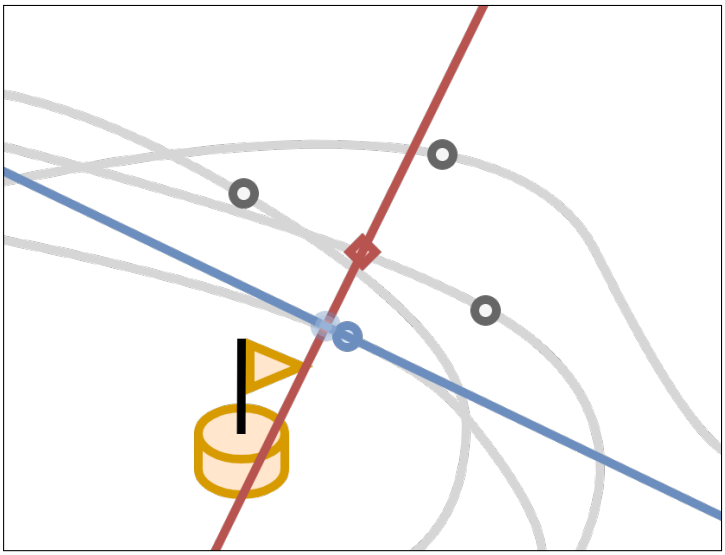

Figure 3.6: Projection of the innermost position onto the mean bearing

someone actually sails along the inside of the mark or all candidates sail very far around the outside of the mark, this method will still produce useful results, which may not be exactly at the location of the real mark, but instead, all tracks have run around the outside of it. And in general, it is more likely that the tracking data was inaccurate than that a competitor actually missed a mark. In these cases, it is safer to display a correct rounding in the tracking and let the race director judge whether the rounding was improper.

3.5 Course Mapping

From the dataset of individual leg-type changes, distinct positions of the likely marks are determined. These positions including their metadata, such as the rounding direction and their waypoint index, are hereinafter referred to as "mark candidates". The general objective now is to assign the positions of the mark candidates to the corresponding waypoints. In case a course configuration is given, the respective waypoints already exist. Otherwise, the waypoints have to be constructed prior to this using the inferred data.

For the allocation of the mark candidates, all candidates need to be arranged in sequence. Mark candidates may either follow one another or coexist parallel to each other as is the case with several marks of the same waypoint. The resulting sequence of marks or mark candidates for that matter will be referred to hereinafter just as the order of marks to simplify the terminology.

If no course layout is provided, further semantic information is derived from the mark candidates to assist in the creation of the waypoints. Even if the course layout is given, the semantic information helps to verify whether the subsequent assignment of the mark candidates to the waypoints makes sense, or whether further adjustments to the sequencing can remedy potential errors.

So once waypoints exist or have been created, the positions of the mark candidates can be assigned to them.

The individual steps are broken down in more detail below.

3.5.1 Mark order

There are three possible approaches to determine the order of the mark candidates; via the order in which the clusters were created, via timestamps of the leg-type transitions in them, or via the index of the leg-type transitions.

- **Cluster creation approach:** If the clusters are stored in a data structure ordered by the time of insertion, it is possible to implement the clustering algorithm in such a way that the order in which the clusters were created is retained. The problem with this approach is that in certain edge cases it is quite likely that the clusters are not created in the correct order. For example, if the tracking data is missing for the leading competitor during a rounding, it is possible that the cluster of the next mark is created first.
- **Timestamp approach:** Since the time points of the individual leg-type changes are known, it would be feasible to include them in the clustered dataset as well. They would not be used for the clustering itself but could be utilized in further processing. Thus it would be possible to determine the first rounding time or average rounding time of a cluster. This can then be used to sort the results. One problem with this approach is that there are sometimes large differences in the rounding times, even to the extent that individual competitors might be lapped. Because not every single mark is necessarily passed by every competitor, this can lead to distorted results and thus to wrong order.
- **Index approach:** Instead of trying to put the clusters in order based on the timestamps, the order can also be determined for each competitor individually. In this case, only the index of the leg-type transition in the competitor's individual sequence would be passed on to the clusters. As the majority of the leg-type transitions contained in a cluster can be assumed to have the same index, the average value over these can be determined here and rounded to the nearest whole number. This should then correspond to the actual sequence index of the majority of the competitors. An alternative approach would be to only use leg-type changes of the same index in the same sub-set for clustering. This would enable dividing different rounds in which the same mark is rounded. However, it comes with the disadvantage that potential mismatches in the individual sequencing would then no longer be taken into account in the clustering.

Since both the classification by creation date of the clusters and aggregated time stamps do not cover all edge cases and may lead to incorrect ordering, the choice falls on the index-based methodology.

To determine the index of the cluster, the accuracy of the indices of each leg-type change is taken into account. The more precise the assignment is, the more leg-type changes can be allocated to the correct sub-set, which also means that the potential accuracy of the positioning increases. Generally, the determination of the waypoint index is taken from the individual leg order of the competitor. However, if information about the course layout is available, this information can be used to correct possible errors within the leg-type changes based on semantics. A common example is a transition from upwind directly to downwind. In some cases, because the competitor makes a very large turn, this one leg-type transition becomes two; from upwind to reaching and only then to downwind. The incorrect leg-type changes may not be relevant for the determination of the position, but this also messes up the indexing of all subsequent leg-type changes. Knowing which transition is expected next, corrections can be made accordingly.

3.5.2 Mark semantics

In order to ensure the correct assignment and grouping of the identified mark candidates, semantic information about the relations between the marks is required. It needs to be determined whether it is a single mark that is passed once, whether it is a single mark that is part of several waypoints and therefore passed several times, or whether it is part of a gate that consists of two marks.

Marks passed multiple times

To determine whether several mark candidates belong to each other and in fact represent the same mark, an additional clustering operation can be applied to them, looking for any remarkable overlaps. The used metric refers solely to the positions of the mark candidates. This allows for identifying the same mark over different laps. However, if the mark is moved in between laps, this cannot be easily distinguished from a substitution mark, which is why it will be treated like one instead in the current approach. Only if information about the waypoints is available in advance, this could be extracted from it.

In case information about the waypoints in their sequence is available, which clarifies which marks are used more than once, the identification of the semantics is nevertheless useful, since this way the assignment can be checked to determine whether the data make sense and, in case of doubt, adjustments can be made.

Gates

Provided that the waypoints including their marks are already predefined, correct sequencing also makes it clear whether the determined mark candidates belong to a gate or not.

In the absence of this information, it can be assumed that a mark is part of a gate if the following properties are fulfilled. For one the associated leg-type transition is either upwind to downwind or vice versa. Besides that, the number of competitors who rounded the mark is often significantly lower than the total number of competitors. If the majority of the competitors round only one side of the gate due to an advantaged gate, there is no real possibility to distinguish this from a normal mark. Since these are primarily the cases where only one mark of a gate is recognized, this check is mostly also true, which is why no gate is recognized. Because of this, the estimation of unknown gates is prone to errors. Instead of provoking the recognition of non-existent gates, it makes more sense in these cases not to mark a gate at all, since the majority of competitors have sailed around the other mark anyway.

In case the second mark position needs to be estimated, the direction and distance from the existing mark candidate is required. Since the second mark should be placed somewhat orthogonal to the wind, the direction can be calculated using the rounding side. The distance of the laid down gates is usually roughly dependent on the hull length of the boat class and can therefore also be estimated.

However, even in the event of a given course layout, it may be advisable to check whether the marks assigned to a gate have suitable properties that suggest the presence of a gate. If this is not the case, this can be an indication that there is an error in the sequence of the mark candidates and, for example, an additional mark has been found or an actual mark has been missed. In these cases, the assignment of the marks may be shifted.

3.5.3 Waypoint assignment

Unless there are predefined waypoints and marks, these are constructed at this point with the help of the previously determined semantic information and general metadata of the associated mark candidates. Furthermore, it is possible that already existing waypoints have to be updated because a waypoint that was previously created as a single mark is now part of a newly recognized gate.

After this, the updated positions of mark candidates are assigned and added to the marks defined in the respective waypoints.

4 Implementation

Following on from the conceptual design of the solution described in chapter 3, the actual technical implementation of the concept is carried out. The necessary steps for the integration into the existing application environment, the general architecture of the solution, and individual technical details of the implementation will be explained in greater detail below.

4.1 Environment

Due to the integration of the solution as part of a comprehensive software suite, many technical details regarding the choice of technologies are already predetermined. The application is almost entirely written in Java and relies heavily on the Open Services Gateway initiative (OSGi) software platform and the Google Web Toolkit (GWT) web framework.

It is important that the solution integrates seamlessly with the existing application and does not cause existing parts of the application to stop working properly. Similarly, its performance should not be impaired significantly in any way. Newly created generic features, which may eventually be needed elsewhere as well, should be designed in such a way that they are modular and also work outside the specific use case.

The technical nature of the tracking systems also needs to be explained in more detail at this point. As already mentioned in section 3.2, most tracking systems, regardless of whether they are smartphones or dedicated trackers, are connected via a mobile network. So both heavy usage and conditions on the water can cause the connection to the server to be lost and cause tracking fixes to get interrupted. Usually, each fix is treated as a separate entity and also sent as such. Due to the implementation on the tracker side, it is possible that fixes do not arrive in chronological order, but individual fixes arrive before their predecessors; these are also referred to as out-of-order fixes and often require special attention.

4.2 Integration

Integrating with the existing application is fairly straightforward. There is plenty of interfacing with the existing services to retrieve details about the race and to react live to changes in it. This also applies to tracking and wind data. Therefore, only minor changes are necessary in this regard and most of the existing structures can be built upon.

Solely the race configuration and GUI must be supplemented by the respective option to activate the automatic waypoint inference. As can be seen in Figure 4.1, a checkbox is added for this purpose when connecting the race with the respective tracking provider, with which the function can be activated and deactivated. By default, it is deactivated.

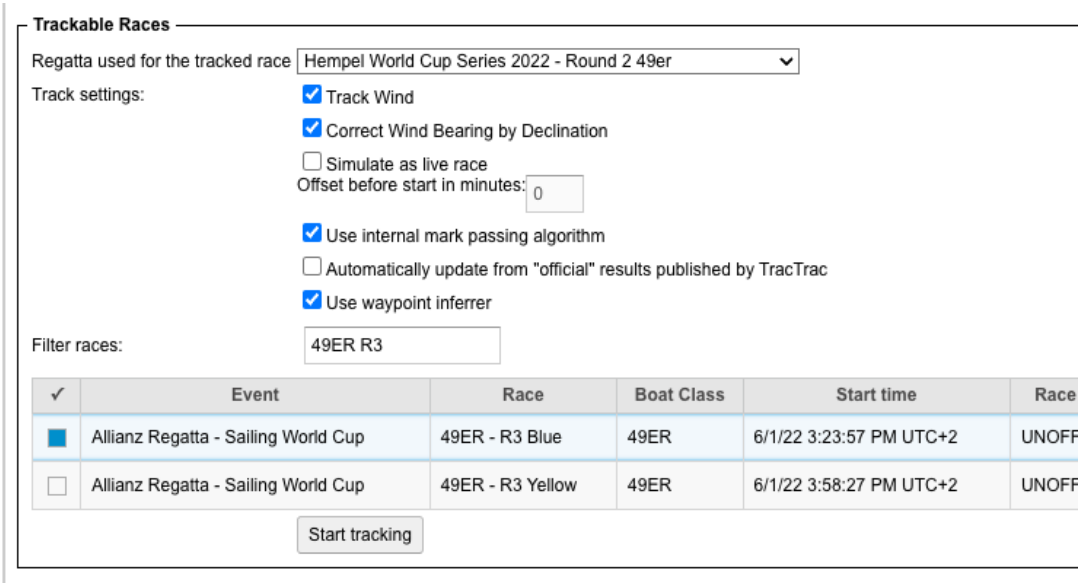

Figure 4.1: Checkbox in the GUI to enable the solution

4.3 Architecture

The actual architectural structure of the implementation differs from the theoretical structure explained in chapter 3. As can be distinguished in Figure 4.2 by the colored outlines, the new logic can be divided into three functional areas.

All entities outlined in red are part of the detection of leg-type transitions. While the underlying logic of this process is contained in the LegTypeChangeFinder class,

an interface that can be implemented to receive and process updates regarding leg-type changes is provided by LegTypeChangedListener. For this reason, the interface is also implemented by the WaypointInferrer class, containing all the necessary logic for inferring the actual waypoints from the collected leg-type transitions. As described in subsection 3.4.1 and subsubsection 3.5.2, the clustering of data is a large part of the steps required to accomplish that. For this reason, the entities for performing the actual clustering are framed in green. They are separated into their own independent classes according to the modularity required by the technical requirements. The same applies to the previously mentioned classes of the leg-type change detection. The class WaypointInferrerClusterer is a specialized implementation of the generic clusterer to allow for more efficient calculations for the specific application while clustering. So in terms of its context, it belongs to the WaypointInferrer.

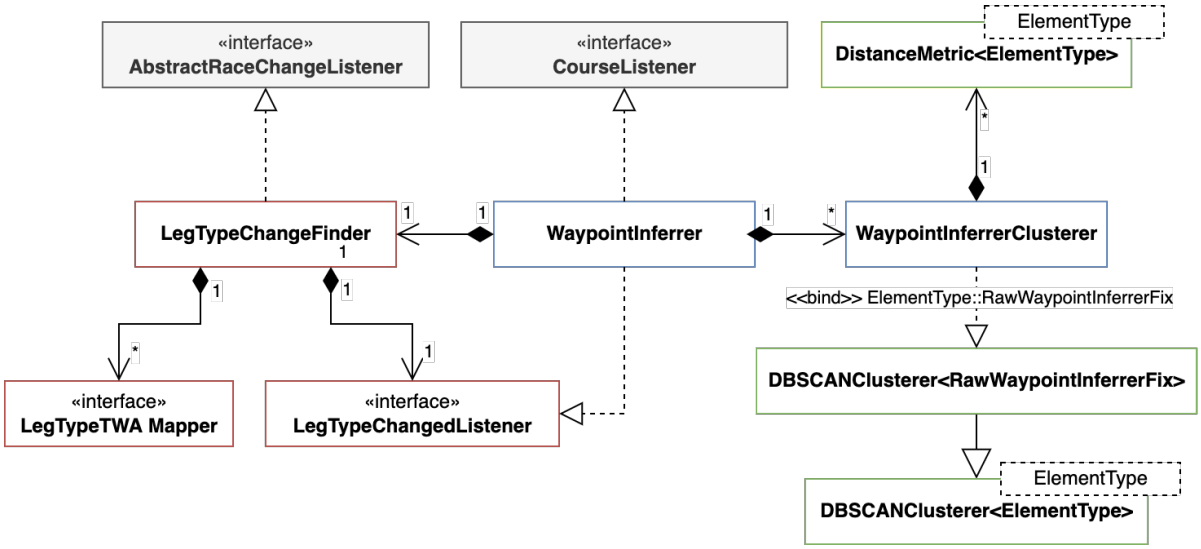

Figure 4.2: Simplified UML diagram of the solution architecture

Relevant technical details of the respective application areas will be explained in more detail below.

4.4 Leg-Type Detection

When a LegTypeChangeFinder is instantiated, it registers directly as a listener for changes to the tracking of the race. The respective race needs to be specified when the instance is

created. For this purpose, the class itself inherits from the `AbstractRaceChangeListener` class.

The basic functionality is based on maintaining a separate list of leg-type transitions sorted by time for all competitors in the race. This list exists in two versions. One list contains all recorded leg-type changes and one list contains all entries of the other list which have been smoothed with a filter.

Each time a new tracking fix for a competitor is added to the race, the corresponding leg-type is determined (more on that in subsection 4.4.1) and added to the set of raw leg-type changes for the competitor. Compression is already used here to merge the same consecutive entries and save them as one. For example, assuming that the registered leg-type changes, simplified as letters, would be the sequence `DDDDUUUURRRRDDDD`, this becomes `DURD`. It is important, however, that the correct point in time of the leg-type transition is not lost in the process. Even if the majority of tracking fixes arrive in the correct order, this cannot be guaranteed due to the technical characteristics of the tracking systems and may result in out-of-order fixes. Due to the use of a list sorted by timestamp, this is initially not problematic for the raw leg-type changes; however, special care must be given during compression to ensure that the correct timestamp is retained. This is established by a series of rules that have to be checked in order until one of them is true [32].

1. If no other leg-type has been saved so far, just add the new one.
2. If it is the most recent leg-type entry and the previous one is of the same type, remove the following entry and insert the new one instead. Otherwise, insert the new one without removing the following one.
3. If it is the first entry and the following one differs in type, simply add the new one.
4. Otherwise, if the following entry has a different leg-type, just add the element.

This ensures that each entry in the list is assigned to the last known time at which the competitor was measured on the respective leg. This is important for the correct filtering of the entries afterward. The time span in which the competitor has been on the respective leg is therefore always limited by the time of the previous and the current legs-type entry. During the entire time in between, the competitor has sailed into the respective direction [32].

After a change has been made to the raw leg-type changes, it is also necessary to check whether an update of the list for smoothed leg-type changes is necessary as well. For this, a distinction can be made between two specific cases; was the change at the end of the time series, or was it in the middle of it?

In case of an out-of-order fix, the integrity of the smoothed leg-type changes can no longer be guaranteed and the entire smoothed leg-type list of the competitor has to be regenerated based on its raw leg-type changes. For this purpose, the smoothed leg-type change list is cleared and reconstructed in sequence with the data of the raw leg-type changes until it is up to date again. The reconstruction follows the same procedure as for the arrival of new in-order fixes, which is explained below. In addition, a message is sent to the listeners informing them about the recalculation of the subscriber's leg-type changes. More about this in subsection 4.4.2 [32].

In most cases, the fix arrives in-order and updates on the smoothed leg-type change list can be carried out incrementally. For the application of the moving average filter, a half-window duration depending on the approximate maneuver duration of the respective boat class is used. This is necessary because different sampling rates are required for different course sizes and the best approximation to the course length, which is still unknown at the time, is the usual duration of maneuvers in the respective boat class. The approximate maneuver time of the boat classes can be easily accessed. A tenfold value of the maneuver time proved to be useful in tests, as this smoothed out any noise and short leg-type changes sufficiently to detect actual leg-type transitions, even if these were not particularly long.

For the actual implementation of the moving average filter, a subset of the raw leg-type changes of the competitor is taken, which contains all entries that lie within the half-window duration before and after the timepoint of the new fix. This is the filter window. No simple numerical operation can be performed to determine the average of the leg-type changes since the values are not continuous. Instead, the number of occurrences of all leg-type variants in the filter window is counted to determine the predominant leg-type. This then indicates the leg-type at the timepoint under consideration. This calculation is not very computationally intensive, since only a few entries are likely to be in the subset.

Only if the newly determined smoothed leg-type differs from the last stored one, a leg-type transition has actually taken place. In this case, the new leg-type is added to

the smoothed leg-type list at the corresponding point in time. An update message is also sent to the listeners. More about this in subsection 4.4.2. The reported time of the determined leg-type change is not that of the fix itself, but that of the fix minus the half-window duration, since as explained in subsection 2.2.2 only the smoothed element can be determined retroactively, but not the one directly at the edge of the dataset.

4.4.1 Determination of the leg type

The current wind at the location of the boat and at the respective time can be easily accessed through the corresponding service. From the returned object, direct access methods are available for the current TWA and wind speed.

As shown in Figure 4.3, the logic for determining the current leg-type from this data is hidden behind an interface, the `LegTypeTWAMapper`. The purpose of this is to allow fast and easy switching between methods of leg-type determination and to encapsulate this functionality for other use cases as well. This interface has two different implementations.

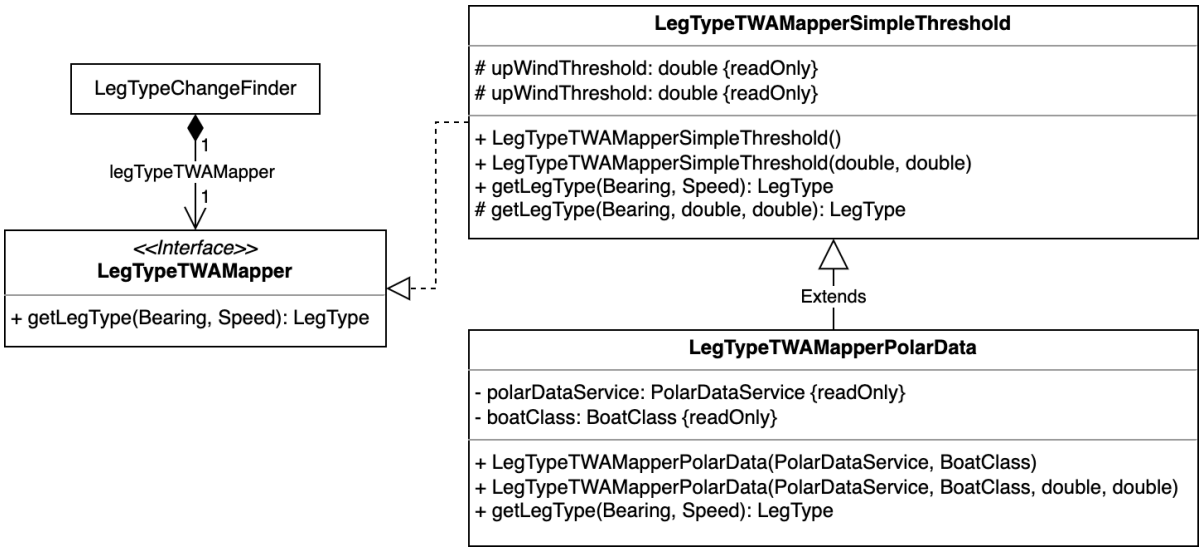

Figure 4.3: UML diagram of the leg-type determination

The class `LegTypeTWAMapperSimpleThreshold` contains the implementation for a simple distinction based on fixed threshold values, as described in subsection 3.3.1. Using trial and error on the live dataset, thresholds of 65 as upwind threshold and 120 as downwind threshold were initially found to yield acceptable results. The actual

separation of the three states is done using simple case separation. To allow additional implementations to build on this simple design, a helper method exists within the class, as shown in Figure 4.3, which allows the leg type to be determined by specifying the two threshold values. The actual `getLegType`-method simply invokes this auxiliary method with the fixed threshold values defined in the object itself [32].

As already mentioned, the second implementation expands on the previously explained one and adds the connection to the `PolarDataService` to obtain the threshold values. The `PolarDataService` analyzes all past races in order to gather aggregated data related to the polar characteristics of the different boat classes. It uses all historical races available on the respective SAP Sailing instance as well as the combined data of all races on the archive server. In this way, data is available for the assessment of almost all common boat classes in sailboat racing, even if the respective server instance is freshly set up. Among other things, the service also includes data on the maximum and minimum TWA achievable with the respective boat class during maneuvers, taking into account the given wind speeds. In turn, using a small tolerance margin, accurate thresholds can be derived for each of the known boat classes. The thresholds are thus dynamically adapted to the boat class of the race and the current wind strength.

4.4.2 Listener pattern

The observer pattern is used to notify other components of the application about leg-type changes within the examined race. This allows those other components to register as listeners of the `LegTypeChangeFinder` and to be informed accordingly as soon as new data is collected. The design pattern promotes abstraction in the application and decouples the `LegTypeChangeFinder` from possible usage locations of it. For the registration of a class as a listener, it has to implement the `LegTypeChangeListener` interface. This offers the two methods stated in Figure 4.4 [32].

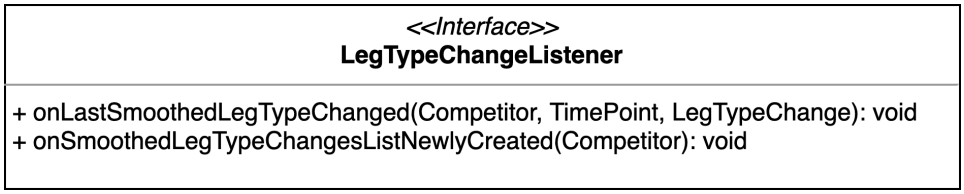

Figure 4.4: UML diagram of the LegTypeChangeListener interface

The standard scenario involves the `onLastSmoothedLegTypeChanged` method. This one is invoked whenever new leg-type changes occur at the end of the time series. It provides information regarding the affected competitor, the time of the transition, and the actual transition type. From this, both the previous and the new leg-type can be derived.

Since the arrival of an out-of-order tracking fix always entails the recalculation of all smoothed leg-type changes, every component that processes this data needs to be recalculated as well. In these cases the `onSmoothedLegTypeChangesListNewlyCreated` method is called, which only passes a competitor as its parameter. This informs the respective listeners that a recalculation has taken place and all leg-type changes should be reloaded.

With this interfacing, a listener can attach to and detach from the `LegTypeChangeFinder` at any time.

4.5 Clustering

In order to perform the clustering, an incremental DBSCAN algorithm is used. Its algorithmic implementation does not use any libraries other than the default java sources and is based on the two articles [23] and [25].

This implementation is kept generic, allowing it to work with any data type. The only prerequisite is that a metric for calculating the spatial distance between two elements are defined for the respective data type of the clustered elements. For this, the interface shown in Figure 4.5 is provided, which can be implemented.

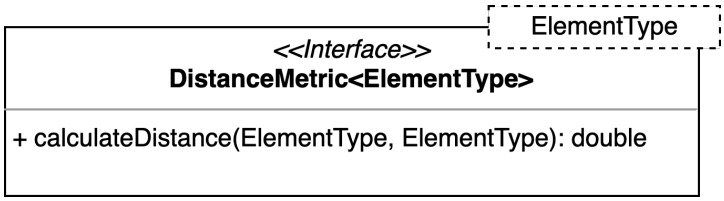

Figure 4.5: UML diagram of the distance metric interface

In Listing 4.1, the implementation for the calculation of the distance of two catesic coordinates is given as an example. The elements to be compared are specified as

parameters of the calculateDistance method. As return value, a double is expected as distance, which then represents a comparable length.

```
1 public class EuclideanDistanceMetric implements DistanceMetric<Coord> {
2
3     @Override
4     public double calculateDistance(Coord c1, Coord c2) {
5         double deltaX = Math.pow(c2.getX()-c1.getX(), 2);
6         double deltaY = Math.pow(c2.getY()-c1.getY(), 2);
7         return Math.sqrt(deltaX+deltaY);
8     }
9 }
```

Listing 4.1: Metric for calculating the eucildic distance of two coordinates

To increase the speed of the algorithm, the implementation uses a separate lookup HashMap storing the respective number of neighbors of each element in the dataset, as described in [25]. For the same reasons a lookup hashmap is used for the implementation in java, in order to be able to query efficiently for each element, to which cluster it belongs and vice versa for each cluster which elements are contained in it.

In addition, the class features the two not furtfher implemented methods `onCreateCluster` and `onInsertElement`. These can be overridden by more specific implementations that build upon this one to further extend the functionality.

4.6 Waypoint Inference

From a technical point of view, the `WaypointInferrer` class forms the core of the solution. Upon creation of the class, a new instance of the LegTypeChangeFinder is first created and the class itself is registered as a listener following the procedure described in subsection 4.4.2. With each incoming leg-type change, the inference of the waypoints advances incrementally.

4.6.1 Grouping the leg-type changes

As described in chapter 3, the grouping of leg-type changes takes place in two steps; presorting using rule-based patterns and unsupervised clustering afterward.

Deriving the rounding side is comparatively easy. The application already provides an implementation of a maneuver detector. Among other things, it offers a method to determine the course change around a certain `GPSFix`. Additionally, it requires the time span to be considered as a parameter. Because of the half-window duration defined within the `LegtypeChangeFinder` we know exactly which time span was relevant for detecting the leg-type change and therefore has to be considered now as well. As a result, the rounding side is returned, represented by the `NauticalSide` enum. Possible values here are `STARBOARD` and `PORT` [32].

For the determination of the sequence index of the leg-type changes, each competitor is assigned a separate time series containing its leg-type changes. This is implemented using a NavigateSet, which uses the timestamp of the leg-type change as a comparison operator. Thus all entries of the set are sorted immediately according to their time. Likewise, it is possible to easily check whether the new element is at the end of the time series, allowing one to judge whether it is an in-order or an out-of-order entry. This is because with in-order fixes only the new index of the added element has to be determined. For out-of-order fixes, the index of all subsequent fixes also needs to be updated.

Here, an advantage is taken of the fact that out-of-order fixes, even if they are technically possible and allowed, tend to be the exception and the majority of fixes arrive in their actual order. Hence, the most performant and effective processing can be chosen for in-order fixes, while out-of-order fixes require slightly more computational effort. Since these have already been assigned to a clusterer depending on the said index, it is necessary to first remove them from all affected clusters, correct the index and then add them to their new cluster. Due to the incremental implementation of the clustering algorithm, this is nevertheless possible without a major performance overhead.

Both the estimated index of the associated waypoint and the observed rounding direction together form the key for assigning the leg-type change to the correct subset of data and thus to the respective clusterer. For each such key combination that arises, a separate clusterer is created and stored in a hash map so that it can also be used for any subsequent new leg-type changes. The clusterer then takes over the further grouping and selection of relevant entries in the relevant sub-set of data. More about the implementation of this in section 4.5.

The metric listed in Listing 4.2 is used for clustering the leg-type changes. Primarily, the distance between the two positions of the leg-type changes is compared with each other. For this purpose, the great-circle-distance between the coordinates of the two positions is calculated using the Haversine formula. The position value object fortunately already provides the necessary methods, containing the implementation of the formula.

```
1 public class WaypointInferrerFixDistanceMetric implements ↙
      ↳ DistanceMetric<RawWaypointInferrerFix> {
2     @Override
3     public double calculateDistance(RawWaypointInferrerFix fix1, RawWaypointInferrerFix ↙
          ↳ fix2) {
4         return fix1.getPosition().getDistance(fix2.getPosition())
5             .getKilometers();
6     }
7 }
```

Listing 4.2: Metric for calculating the distance of two leg-type change positions

4.6.2 Determination of the exact position

Besides grouping the data, in this case, the clusterer is also responsible for determining the position estimates associated with the clusters. The implementation of this functionality directly within the clusterer has the advantage that the incremental nature of the algorithm can be utilized and the result is also adjusted incrementally. Thus, the estimated position does not have to be recalculated every time, but only in case of actual changes to the cluster in question.

Using the `onCreateCluster` and `onInsertElement` methods provided by the underlying `DBSCANClusterer` class, only the recalculation of previously modified clusters can be triggered independently. The resulting process corresponds to the procedure explained in subsection 3.4.2.

The mean position is calculated by simply taking the average of all longitude and latitude values and creating a new position from them.

In order to aggregate the bearing towards the center of the curved tracks of all competitors, rather than calculating this bearing individually for each competitor, instead, the mean bearing in the direction of travel is calculated first. This is done in the same way as for the positions by calculating the average of all bearings. The direction of rounding

is stored within the clusterer. Depending on this, either 90 degrees are added to the direction of travel or 90 degrees are subtracted. The resulting bearing now points starting from the mean position approximately into the middle of all curves of the competitors.

The next step is the determination of the element furthest in the direction of the mean bearing and the projection of this element onto the mean bearing. These two tasks are combined in a single step. To find out which element is furthest inside, all elements in the cluster are iterated. In each case, the projection of the element onto the mean bearing is calculated. This way the distance of the projection to the mean position can be measured to evaluate which element is furthest away along the bearing. Since this would also include elements that lie in the opposite direction of the bearing, these must be sorted out beforehand. For this purpose, first of all, the angle of the bearing pointing to the position of the respective element is measured. If this is greater than 90 degrees in either direction, the element is on the wrong side and can be ignored. All other elements are positioned accordingly in the correct direction. When iterating over the elements, the calculated distance is now compared with the previous largest one; if its own distance is larger, the comparison value is updated and the respective projection is stored temporarily. The project that is stored at the end is the one that lies furthest inwards. The position of the projection is now offset inwards by a further 1.5 beam lengths. This distance reflects the fact that the boats want to sail as close as possible to the mark but avoid colliding with it. One beam length often corresponds to the minimum possible distance. The additional half beam length then represents the additional space that is left. The factor also seems to provide reliable results with test data. Estimated mark positions of all clusters are cached locally. There are methods for accessing that cache externally or for directly retrieving the mark position of the largest cluster.

4.6.3 Updating the mark positions

For a better technical understanding of how the actual positions associated with the correct marks are entered, the underlying data structure needs to be understood first.

Data Structure

A reference to the tracked race is stored in the WaypointInferrer class. This in turn allows access to the existing course information, which also contains the sequence of waypoints that is being attempted to be inferred. In the case that a race course layout is already known in advance, this sequence of waypoints is already populated.

Essentially, a waypoint is composed of a control point and associated passing instructions. In the case of a single mark, the passing instructions contain the side on which the mark must be rounded, in all other cases general information about how the waypoint must be traversed. A control point, on the other hand, is simply a collection of marks that are grouped together under a collective name. In the focus of this work, only two different implementations of the control point are relevant; one with only one and one with two marks. In the case of the implementation with two marks, the left and the right mark can be accessed separately. In the context of this thesis, a mark is merely an identifiable entity, which in turn also has its own name. Waypoints and control points, however, are immutable. It is therefore not possible to make changes to them; it is necessary to remove them from or add them to the course object as a whole.

For each mark, a `DynamicTrack` is added to the tracked race, which stores the position data of the respective mark during the time of the race. Due to the current technical implementation of these tracks, it is only possible to add entries or replace existing ones, but not to delete them. Therefore, it is important to choose carefully at which points in time a `GPSFix` is added.

In order to enable the correct assignment of the `GPSFixes` to the individual marks later on, a dictionary is created for the marks within the `WaypointInferrer`. This is technically implemented using two nested `HashMaps` in order to guarantee faster access times than via the racecourse object itself. The outer hashmap allows the retrieval of all marks belonging to a waypoint; realized via the waypoint index as a key. These marks are in turn sorted according to their rounding side. As a result, the marks of waypoints with two marks can be uniquely identified.

Waypoint Initialization

When setting up the `WaypointInferrer`, this dictionary is filled with the mark instances corresponding to the already existing marks for faster access. This is done by iterating over the waypoints. If the associated `ControlPoint` is a `ControlPointWithTwoMarks`, both marks must be added to the dictionary. The left mark is stored as the port-side mark and the right mark as the starboard-side mark. In case it is not a `ControlPointWithTwoMarks` it is assumed that there is only one mark. In this case, the mark is stored to the respective side according to its passing instructions. In some races, no further information about the passing instructions is given. Since in this case it is not known which rounding side is to be expected, the same mark is simply stored for both rounding sides. The corresponding source code is provided in Listing 4.3.

```
1  private void attachMarksOfExistingWaypoints() {
2      for (int waypointIndex = 0; waypointIndex < existingWaypointData.size(); ↙
           ↳ waypointIndex++) {
3          Waypoint waypoint = existingWaypointData.get(waypointIndex);
4          ControlPoint controlPoint = waypoint.getControlPoint();
5
6          HashMap<NauticalSide, Mark> markMap = estimatedMarks
7                  .computeIfAbsent(waypointIndex, c -> new LinkedHashMap<>());
8
9          if (controlPoint instanceof ControlPointWithTwoMarks) {
10             Mark mark1 = ((ControlPointWithTwoMarks) controlPoint).getLeft();
11             Mark mark2 = ((ControlPointWithTwoMarks) controlPoint).getRight();
12
13             markMap.put(NauticalSide.PORT, mark1);
14             markMap.put(NauticalSide.STARBOARD, mark2);
15         } else {
16             Mark mark = controlPoint.getMarks().iterator().next();
17
18             boolean addToPort = false;
19             boolean addToStarboard = false;
20             if (waypoint.getPassingInstructions() == PassingInstruction.Port) {
21                 addToPort = true;
22             } else if (waypoint.getPassingInstructions() == ↙
                       ↳ PassingInstruction.Starboard) {
23                 addToStarboard = true;
24             } else {
25                 addToPort = true;
26                 addToStarboard = true;
27             }
28
29             if (addToPort) {
30                 markMap.put(NauticalSide.PORT, mark);
```

```
31                }
32                if (addToStarboard) {
33                    markMap.put(NauticalSide.STARBOARD, mark);
34                }
35            }
36        }
37    }
```

Listing 4.3: Algorithm for populating the quick lookup dictionary for the marks based on their waypoint index and their rounding side

Waypoint Updates

Using the previously created lookup dictionary, inserting a new position to the mark's fixes is relatively straightforward. Since timepoint, estimated waypoint index, and rounding side of the leg-type change that led to the update of that mark candidate position are all known, the affected mark can be easily retrieved from the dictionary.

Since no tracking fix can be removed, as mentioned in subsubsection 4.6.3, a check is performed first to see whether a `GPSFix` has already been stored for the respective waypoint. If not, a new `GPSFix` will be inserted at the time of the first rounding of the previous waypoint. Otherwise, the position of the existing fix is compared with the new one. If the position is still the same, no update is necessary and the process is therefore completed. But if there is a deviation, a new `GPSFix` with the updated position is created with the exact timestamp as the existing one, in order to overwrite it. For this purpose, the replace flag of the `DynamicGPSFixTrack` must be utilized.

4.7 Evaluation

In order to draw meaningful conclusions about the accuracy of the solution, a series of tests are carried out.

4.7.1 Test Environment

Each of the tests is performed on a wide range of different historical races. The selection includes 42 races with a wide range of characteristics and a total of 199 marks. The races are selected from different events of recent years, from different boat classes and dimensions. Based on the results of these tests, average key figures can be calculated across all tested races, but also individual results can be analyzed in order to identify possible relationships between race characteristics and better or worse results. For this purpose, the metadata of the analyzed race is also collected for each individual run.

The raw data for generating the results are collected in two separate runs. One run is done with the regular data on the marks obtained from the tracking; the second run is done without the tracking data of the marks, but instead with the waypoint inferrer activated. During each race, the following data is collected.

The chosen tests primarily look at and compare the inferred mark positions with those of the actual marks measured in tracking, but also the deviations in rounding times caused by the new marks. Separate data is collected for each mark of the race.

- The name of the mark for easier identification afterward.
- The timestamp of the first detected rounding of the waypoint. In order to have comparable data, we do not rely on external data for the mark passing times but use the internal mark passing algorithm instead. This avoids external influences from different mark passing detection methods.
- The position of the mark specified in latitude and longitude by degrees. The previously determined first mark passing time is used as the timepoint for capturing the position.
- Whether the mark is part of a gate or just an individual mark.

In addition to the data about the marks themselves, some metadata about the race is collected as well in order to allow for easier identification later on and to be able to draw more profound conclusions.

- Name of the regatta of which the race is a part.
- Name of the race itself.
- Boat class of the race.
- Number of competitors participating in the race.
- Total distance of the race course in meters.

From the raw data of both runs, a number of metrics can be calculated. For each mark, this includes the *distance in meters* between the two measurements and the *time difference in milliseconds* of the first mark passing. In hierarchical ascending progression, the average values of the marks can be calculated for each race and for all of the races. These values can also be correlated with the additional collected metadata.

4.7.2 Results

Of the 42 races examined, all the marks were correctly assigned to the waypoints in 33 of the races. Vice versa for the remaining races, this implies that the marks were either not found or were not assigned correctly on the basis of their waypoint indexes. Two patterns can be identified when manually examining the affected races. Races with few competitors are predominantly afflicted. In other cases, only a single side of a gate was used, which is why the missing waypoint could not be inferred. All in all, the assignment based on the waypoint index could be improved, although it provides a good foundation for many races. For the evaluation of the position accuracy, those races with an incorrect allocation were left out.

A crude examination of the total average values of all races as presented in Table 4.1 shows decent results. The average distance of 22.31 meters seems not particularly accurate at first. The fact that the time offset of the mark passing still only shows a minor offset of 0.71 seconds means that the selected positions are nonetheless reasonably chosen.

Average Position Offset	22.31 m
Average Absolute Time Offset	0.65 s
Average Time Offset	-0.08 s

Table 4.1: Total evaluation results for all marks in the test dataset

However, a closer look at the test results of the individual races quickly reveals that the majority of the races show significantly better values with regard to the distance of the marks. For 85% of the waypoints, the mark offset distance is less than 7 meters on average. Taking into account that tracking via Global Positioning System (GPS) also entails a level of inaccuracy, these values are actually quite convincing. Now, the question arises as to how the other 15% are composed and what are their characteristics correlating to the deviation.

It is noticeable that marks that are part of a gate tend to have a larger discrepancy than individual marks, as seen in Table 4.2. A direct comparison of the accuracy of gate marks and normal marks confirms this theory. While single marks have an average deviation of about 2 meters, the deviation of gate marks is above 20 meters. And yet the time offset of gate marks is nevertheless far less than one second. This indicates that the position determination also works correctly for the most part and that the marks are merely displaced too far inward.

Results by Waypoint Type			
Waypoint Type	Avg. Position Offset	Avg. Abs. Time Offset	Avg. Time Offset
SINGLE MARK	4.24 m	0.82 s	-0.10 s
GATE	39.19 m	0.49 s	-0.06 s

Table 4.2: Total evaluation results for all marks in the test dataset per waypoint type

The biggest error seems to be induced by a few very poorly inferred races. If we look at the 5 worst results among all 42 races shown in Table 4.3, there are some very strong tendencies. Noticeably, all of these races are races with a very small number of competitors. What is not apparent in the data itself, but on closer examination of the races themselves, is the fact that of those few competitors, about half of them did not actually take part in the race and therefore produced false fixes. Due to the overall low data density in

these races, wrong assumptions were made regarding the distribution of the marks. The reliability of races with few competitors is therefore overall quite limited. In general, races that have a higher number of competitors also seem to yield more accurate results.

Race #	1	2	3	4	5
Competitor Count	4	6	5	7	10
Course Length	6549 m	4783 m	4977 m	4918 m	6720 m
Average Position Offset	814.87 m	615.28 m	512.08 m	509.21 m	54.03 m
Average Absolute Time Offset	842.23 s	744.22 s	572.37 s	609.14 s	566.35 s
Average Time Offset	-837.42 s	-744.22 s	-572.37 s	-609.14 s	-566.35 s

Table 4.3: Evaluation results and details of the five worst inferred races

In addition to the average absolute time deviation of each first mark passing, it is also possible to calculate the average of the non-absolute values. In other words, positive or negative deviations are also taken into consideration. If there is a strong tendency in either a positive or negative direction, a statement can be made as to whether there is a general displacement of the marks' position along the competitors' lines of travel. However, since this value of −0.08 is very close to 0, this does not indicate a really noticeable preference for one of the sides. Conversely, it is evident that deviations in the time point of the leg-type changes seem to be due to general noise in the results and that the time point selected for the leg-type changes is very accurately chosen.

The inference of the marks can be performed live during the race, albeit with a slight delay. As soon as the first competitors have clearly passed a mark, the corresponding positions are determined and the rounding times are calculated by the internal mark passing algorithm. In case of initial wrong assumptions, the position will be updated as soon as more data is available.

5 Summary

In this thesis, a method was developed to indirectly determine the positions of waypoints without individual mark tracking, solely from wind and competitor tracking data. Different technologies for data processing, aggregating of data, and extraction of new features were examined and compared on a theoretical level, leading to the implementation of the solution mostly based on a mixture of rule-based systems and clustering techniques.

Besides competitor tracking and wind data, no further information is required to determine potential marks positions, as well as the respective passing instructions. Beyond that, the determination of associated marks is carried out on the basis of this information as well. Merely the selection and assignment of the marks to a waypoint sequence benefits considerably from information about the structure of the course layout. The quality and accuracy of the result increase with the amount of diverse data, which is available. An inference of the waypoints live during the race is also possible with a slight delay, but the first results are often imprecise and will improve continuously with the arrival of additional data.

Outlook

The current implementation provides a solid foundation for inferring the course layout of sail races. Due to the limited resources and time available for this bachelor thesis, however, it was only possible to focus on a partial aspect of the subject area. Likewise, other approaches arose during the development process, which could not be pursued further in the context of this thesis.

Since, in addition to wind and position data, speed measurements and data on the windward distance of the competitors are also available, it may be useful to include them in the inference as well. This may lead to more reliable results.

The inference of the other waypoint types is also not completely inapplicable. At least at the starting line, a pattern can be seen in the speed and HDG of the competitors.

Another interesting research topic is the issue of correctly assigning the mark candidates to the waypoints. Instead of the simple case distinction applied here, a complex network of probability functions is presumably necessary to make precise statements. The application of neuronal networks and artificial intelligence could also provide a promising remedy here.

References

[1] *Racing Rules of Sailing 2022-2024*. 2022. URL: https://www.sailing.org/tools/documents/RRS20212024Finalwithbookmarks-[27255].pdf (visited on 08/26/2022).

[2] Bark, A. *Sportküstenschifferschein & Sportbootführerschein See*. 19th ed. Bielefeld: Delius Klasing, 2020. ISBN: 978-3-667-11165-4.

[3] *Sailing Instructions Part I*. 2022. URL: https://www.kyc.de/fileadmin/content/Regatten/Kieler_Woche/2022/PDF/SI_KiWo_2022_Part1.pdf (visited on 08/26/2022).

[4] Laney, D. *3D Data Management: Controlling Data Volume, Velocity, and Variety*. Tech. rep. 02/2001. URL: http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf.

[5] Rehman, M. H. ur et al. "Big Data Reduction Methods: A Survey". In: *Data Science and Engineering* 1.4 (2016). URL: https://doi.org/10.1007/s41019-016-0022-0.

[6] Pioli, L. et al. "An overview of data reduction solutions at the edge of IoT systems: a systematic mapping of the literature". In: *Computing* (2022), pp. 1–23.

[7] Verleysen, M./ François, D. "The Curse of Dimensionality in Data Mining and Time Series Prediction". In: *Proceedings of the 8th International Conference on Artificial Neural Networks: Computational Intelligence and Bioinspired Systems*. IWANN'05. Barcelona, Spain: Springer-Verlag, 2005, pp. 758–770. ISBN: 3540262083. URL: https://doi.org/10.1007/11494669_93.

[8] García, S. et al. "Big data preprocessing: methods and prospects". In: *Big Data Analytics* 1.1 (2016), p. 9. URL: https://doi.org/10.1186/s41044-016-0014-0.

[9] Salomon, D. *Data Compression: The Complete Reference*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 1846286026.

[10] Proakis, J. G./ Manolakis, D. G. *Digital Signal Processing (3rd Ed.): Principles, Algorithms, and Applications*. USA: Prentice-Hall, Inc., 1996. ISBN: 0133737624.

[11] Morales, R. et al. “Online signal filtering based on the algebraic method and its experimental validation”. In: *Mechanical Systems and Signal Processing* 66-67 (2016), pp. 374–387. ISSN: 0888-3270. URL: https://www.sciencedirect.com/science/article/pii/S088832701500309X.

[12] Chakraborty, S. “Analysis and Study of Incremental DBSCAN Clustering Algorithm”. In: *International Journal of Enterprise Computing and Business Systems* 1 (07/2011).

[13] Likas, A./ Vlassis, N./ J. Verbeek, J. “The global k-means clustering algorithm”. In: *Pattern Recognition* 36.2 (2003). Biometrics, pp. 451–461. ISSN: 0031-3203. URL: https://www.sciencedirect.com/science/article/pii/S0031320302000602.

[14] Jain, A. K./ Murty, M. N./ Flynn, P. J. “Data Clustering: A Review”. In: *ACM Comput. Surv.* 31.3 (09/1999), pp. 264–323. ISSN: 0360-0300. URL: https://doi.org/10.1145/331499.331504.

[15] Danielsson, P.-E. “Euclidean distance mapping”. In: *Computer Graphics and image processing* 14.3 (1980), pp. 227–248.

[16] Kantardzic, M. *Data Mining: Concepts, Models, Methods, and Algorithms*. 2nd ed. New Jersey: Wiley - IEEE Press, 2011. ISBN: 978-0470890455.

[17] Madhulatha, T. S. “An Overview on Clustering Methods”. In: *CoRR* abs/1205.1117 (2012). arXiv: 1205.1117. URL: http://arxiv.org/abs/1205.1117.

[18] Celebi, M. E. *Partitional Clustering Algorithms*. Shreveport, LA, USA: Springer International Publishing, 2015. ISBN: 978-3-319-09258-4.

[19] Abu Bakar, Z./ Mat Deris, M./ Che Alhadi, A. “Performance Analysis of Partitional and Incremental Clustering”. In: 0 (01/2005).

[20] Ahmed, M./ Seraj, R./ Islam, S. M. S. “The k-means algorithm: A comprehensive survey and performance evaluation”. In: *Electronics* 9.8 (2020), p. 1295.

[21] Aggarwal, C. C./ Reddy, C. K. *Data Clustering: Algorithms and Applications*. Vol. 31. Data Mining and Knowledge Discovery. Chapman and Hall / CRC, 08/2013. ISBN: 978-1466558212.

[22] Hanafi, N./ Saadatfar, H. "A fast DBSCAN algorithm for big data based on efficient density calculation". In: *Expert Systems with Applications* 203 (2022), p. 117501. ISSN: 0957-4174. URL: https://www.sciencedirect.com/science/article/pii/S0957417422008302.

[23] Ester, M. et al. "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. Portland, Oregon: AAAI Press, 1996, pp. 226–231.

[24] Chen, Y. et al. "BLOCK-DBSCAN: Fast clustering for large scale data". In: *Pattern Recognition* 109 (2021), p. 107624. ISSN: 0031-3203. URL: https://www.sciencedirect.com/science/article/pii/S0031320320304271.

[25] Ester, M. et al. "Incremental Clustering for Mining in a Data Warehousing Environment". In: *Proceedings of the 24rd International Conference on Very Large Data Bases*. VLDB '98. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, pp. 323–333. ISBN: 1558605665.

[26] Stock, K./ Guesgen, H. "Chapter 10 - Geospatial Reasoning With Open Data". In: *Automating Open Source Intelligence*. Ed. by Layton, R./ Watters, P. A. Boston: Syngress, 2016, pp. 171–204. ISBN: 978-0-12-802916-9. URL: https://www.sciencedirect.com/science/article/pii/B9780128029169000105.

[27] Roy, M. "Great circle theorem and the application of the spherical cosine rule to estimate distances on a globe". In: (2022).

[28] Dauni, P. et al. "Implementation of Haversine formula for school location tracking". In: *Journal of Physics: Conference Series*. Vol. 1402. 7. IOP Publishing. 2019, p. 077028.

[29] Tuomi, M. "Spherical Trigonometry Handbook for Navigators". In: (2021).

[30] Chen, C.-L./ Liu, P.-F./ Gong, W.-T. "A simple approach to great circle sailing: The COFI method". In: *The Journal of Navigation* 67.3 (2014), pp. 403–418.

[31] Purbaningtyas, R./ Arizal, A./ Sholehuddin, M. "REGIONAL LEADING POTENTIAL RECOMMENDATIONS: IMPLEMENTATION OF HAVERSINE FORMULA IN SIDOARJO ON HANDS MOBILE APPLICATIONS". In: *Problems of Information Technology* 10 (07/2019), pp. 70–76.

[32] Stoltenberg, A. *Existing source code within the SAP Sailing Analytics application.* 2019.