Algorithms for finding a middle trajectory

Vermeulen, T.J.M.

Award date:
2013

Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science

Den Dolech 2, 5612 AZ Eindhoven
P.O. Box 513, 5600 MB Eindhoven
The Netherlands
www.tue.nl

Supervisor
prof.dr. Maike Buchin (RU Bochum)
prof.dr. Bettina Speckmann (TU/e)

Order issuer
Algorithms and Visualization

Date
October 23, 2013

Algorithms for finding a middle trajectory

Master's Thesis

Twan Vermeulen

Abstract

Advances in technology have caused trajectory data to become readily available, and analyzing these data has been the subject of research for several years now. We study the problem of finding a middle trajectory in a set of given trajectories. Several approaches to this problem have been presented. However, so far, the main focus of these has been on finding paths that lie spatially in the middle and ignore the time component of the trajectories. The main contribution of this thesis is two-fold. Firstly, we extend existing algorithms to include the temporal component and analyze these. Secondly, we implement and experimentally evaluate these algorithms.

Contents

CHAPTER 1

Introduction

In recent years, more and more devices have been equipped with tracking systems that broadcast their locations. Huge amounts of these geometric data on moving entities such as animals, people, vehicles and natural phenomena have now become available to be analyzed. It is no longer feasible to do this analysis by hand, increasing the need to automate the process.

Recent work in the area of movement analysis includes detecting movement patterns among trajectories [7, 17, 23] such as detecting commuting patterns [8], leadership [3], flocks [16] and relative motion [22]. Other topics include detecting stops and interesting locations [4, 19, 25, 27, 36], determining the similarity of trajectories and the clustering of subsets with a high similarity [24, 26]. More related to this thesis are the papers about finding and defining middle trajectories [1, 9, 34, 35].

Obtaining these geometric data is mostly done using space-based satellite navigation systems such as the Global Positioning System (GPS). Other methods of gathering these data, such as Radio-frequency identification (RFID) or information from camera feeds, exist as well. Using GPS has the advantage that the position in the three-dimensional space (latitude, longitude and altitude) is available at any moment in time, although it is often recorded (or transmitted) in specific intervals. Hence, besides the spatial component, the trajectory data also has a temporal component available.

1.1 Trajectories

A *trajectory* is a time-stamped path taken by a moving entity, represented by a sequence of $n+1$ points in time and space. Each point i is given as a tuple (p_i, t_i). Locations are typically modeled in a multi-dimensional Euclidean space that generally consists of two or three dimensions. A polygonal curve in $\mathbb{R}^d$ is a connected series of line segments. This curve can be described with a function $P : [0, n] \to \mathbb{R}^d$ such that each interval $[i, i+1]$ forms a line segment for $i \in \{0, \ldots, n-1\}$. From now on we assume that space is always two-dimensional and that spatial points are given as a tuple (x_j, y_j) for a point p_j with $0 \leq j \leq n$. A dataset of m trajectories $\tau_1, \ldots, \tau_m$ therefore gives rise to an input size of $\Theta(n \cdot m)$.

The positions are measured in intervals and are only accurate up to a few meters. A trajectory is therefore typically given as a discrete sample of a continuous motion path. The movement path is a polygonal curve that can self-intersect and have repeated vertices at the same location in space if the entity does not move. Moreover, the number of points defining one trajectory is typically much larger than the number of trajectories in the dataset, i.e. $n \gg m$. Such a discrete trajectory is often extended to a continuous motion by assuming that the moving entity is a point that moves with constant velocity from one time-stamped point to the next time-stamped point in a straight line. This assumption leads to an approximation of the real data, which becomes more inaccurate when sampled at longer intervals. An example of such a difference is shown in Figure 1.1. The problem of finding better estimates to fill the gaps between known positions has been the subject of research in many papers [11, 28, 31].

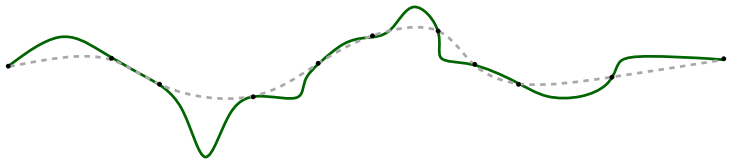


Figure 1.1: The calculated (dashed) trajectory need not be the same as the trajectory that was the source of the measurements (the solid trajectory). The real trajectory can be longer due to "detours" between two points.

1.2 Problem description

Interpreting large amounts of trajectory data is a challenge. Recently it has been argued that trajectories with high sampling rates have to be compressed to allow for fast computations [10, 15]. This compression can be done for each trajectory individually, but in many cases the data contains (parts of) trajectories that are similar to other trajectories in both space and time. A different, possibly additional method to compress the trajectories is to represent these similar trajectories with a single *middle trajectory*, although it would be possible use a graph or tree instead when no clear middle can be defined. An additional benefit is that using a single trajectory opens the possibility to explore the data in new ways. For example, the trajectory can be used to predict where an object will be at a specified time, how fast it will be going or detect anomalies. The latter includes measurements that indicate an object visiting a place that should not be possible, making very sharp turns or having an abnormal speed compared to the middle trajectory.

One application that we use to throughout this thesis is the prediction of the behavior of homing pigeons. They often follow the same route back to their loft when they encounter familiar terrain. Based on this information of individual *route fidelity* in solo homing flights, it is possible to predict leader-follower behavior in paired flights [14]. To measure this route fidelity, we need a notion of a middle trajectory to which we can compare the trajectories followed in the paired flights. This middle trajectory should not only contain information about the path taken in the solo flights, but also the time it takes to fly the path and at what speed (some part of) the trajectory was traveled.

Another application where a middle trajectory might be useful is with hiking. When data is available for many hikers that are walking roughly the same path, we can determine not only the route taken by most hikers, but also give an expected duration and an indication of the roughness of the terrain based on the speed of travel. Note that in this scenario the result should use parts of the input trajectories so that the middle trajectory does not go through inaccessible areas[1]. In other words, it should be a *median* trajectory, not a *mean* trajectory.

In recent years, several approaches have been proposed for finding a middle path. However, most of the existing solutions to the problem of finding a middle trajectory do not take the temporal component into account. There are scenarios where a time correspondence is not obvious to define or it seems to make more sense to ignore the time component. Suppose one wants to determine a middle trajectory for a set of tigers following a similar route. Since these are territorial and generally solitary animals, we cannot use the temporal component as they traveled on different days. If we would include the temporal component, then we might find a completely different trajectory that is not relevant to us. For example, spotting a prey might hold up the tiger for several hours. Likewise, for two cars following the same route it is not relevant that they may have to stop at different locations (e.g. traffic lights, bridges or intersections).

Next, imagine a city with lots of tourist attractions and someone wants to know the route taken by most tourists. Not all tourists visit all attractions and in this case we are not interested in the time spent at each attraction. Moreover, one might find a different route depending on the time of day, since people visit different locations during the day than they do in the evening in which case the time is relevant. Another scenario in which it makes sense to include time is for example when planning a hiking trip. Here one might not only be interested in the path taken by most hikers, but the time it takes to complete this path as well. This becomes more difficult if we want to ignore the stops (to rest), which can be at different locations or can have different lengths. In this case we might want some hybrid definition that normalizes the routes. On the other hand, if we would be interested in these stops, then it would make sense to only include those stops taken by the majority of hikers to avoid detecting stops everywhere.

1.3 Overview of the thesis

The main contribution of this thesis is two-fold. Firstly, we extend existing algorithms to include the temporal component and analyze these. Secondly, we implement and experimentally evaluate these algorithms. In Chapter 2 we discuss how to extend some of the algorithms that do not include the temporal component. Details of the implementation are given in Chapter 3. In Chapter 4 we experimentally compare the results of each algorithm to one another. To the best of our knowledge, we are the first to do such a comparison.

[1] Here we assume that previously accessible paths remain accessible.

CHAPTER 2

Finding a Middle Trajectory

WHEN we want to find a *middle* trajectory, we first have to define what it is. In the previous chapter we have seen a number of applications for such a trajectory, but its properties can be very different for each of the applications.

The goal is to find a time-constrained middle trajectory τ_{mid} of a set $T = \{\tau_1, \ldots, \tau_n\}$ that minimizes a certain distance to all trajectories τ_i for $1 \leq i \leq m$. First we look at some properties such a middle trajectory should satisfy. Then we discuss algorithms that produce trajectories that are spatially "in the middle" and finally we investigate how to incorporate time stamps.

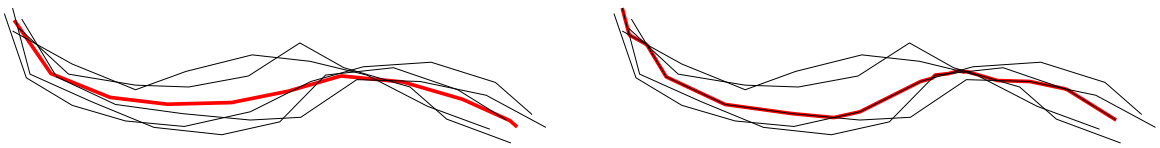
In this chapter we discuss three main approaches to compute a middle trajectory. Furthermore, several algorithms are proposed based on a notion of equal times, in which a middle trajectory is computed by finding a middle point for each time stamp. Finally, we discuss two methods to assign time stamps the middle trajectories.

2.1 Being in the middle: the intuition

There are two main types of interpretations of what a middle trajectory is for a set of input trajectories: a *median* and *mean* trajectory. They both produce a trajectory that is in the middle, either in space, in time or, preferably, both. For a collection of numbers, the definitions of the (arithmetic) mean and the (lower) median are well known. The arithmetic mean is the average of a set of values and the median is the numerical value separating the lower half of a data sample from the higher half. The mean need not be an element of the set, e.g. the mean of the set $\{3, 9, 7, 1\}$ is 5. The (lower) median however is an element contained in the set, in the previous example the lower median is the number 3.

When generalizing these definitions of a mean and a median to a set of trajectories, the intuition behind what these terms represent remains similar. In this thesis we consider a trajectory to be a connected sequence of points in space-time. The main difference between a mean trajectory and a median trajectory is that the latter only uses points of the input trajectories. Another definition of a median trajectory is to use only the edges present in the input set, or parts of the edges, and switch at intersections [9].

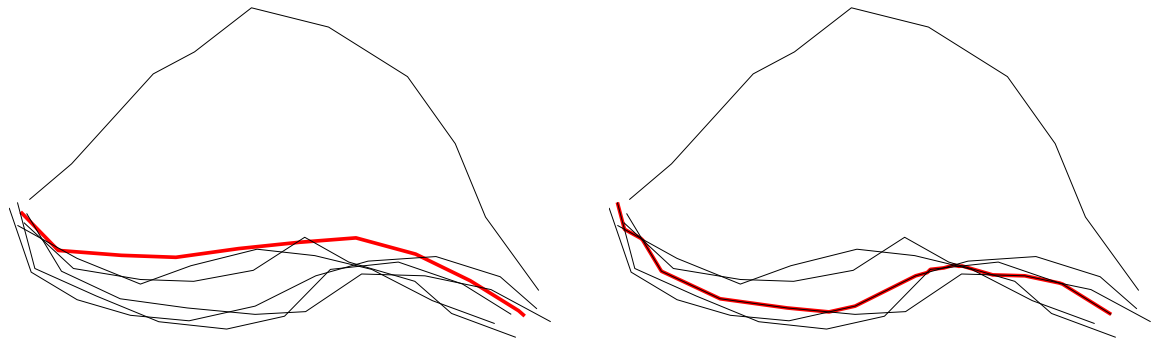
Although a mean trajectory that is based on the positions of the input trajectories might at first look more natural as opposed to a trajectory that is forced to one of the inputs, sometimes a median trajectory is more suitable. A common example is that when trajectories avoid obstacles, one wants the middle trajectory to avoid obstacles as well. If no information about the context (i.e. obstacles or the landscape) is known, then, under the assumption that there are no moving obstacles and that the previously traveled paths remain valid, the only way to make sure that the resulting path does not cross obstacles is to use parts of the original input trajectories. On the other hand if such contextual information is known, then it may also be possible to use this information and make the mean path go around the obstacles. Figure 2.1 shows an example of both a mean trajectory and a median trajectory on the same dataset; The median follows existing trajectories and thus avoids potential obstacles. The mean trajectory need not follow the input and can go through obstacles.



(a) Mean trajectory (b) Median trajectory

Figure 2.1: An example of two middle trajectories: left is a mean and right is a median trajectory.

One major disadvantage of using a mean value to calculate points on the middle trajectory is that this mean can be sensitive to outliers. A single "bad" measurement can negatively influence the calculated mean path if we do not filter these measurements out in a pre-processing step. An example is shown in Figure 2.2.



(a) Mean trajectory (b) Median trajectory

Figure 2.2: A single outlier can negatively influence the calculated mean trajectory, while the median is not affected.

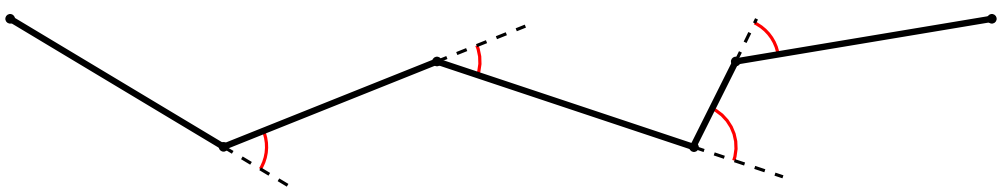


Figure 2.3: The angular change at each vertex is indicated by the red arcs.

2.2 Properties

To define a middle trajectory, we first need to know when a trajectory can be considered as being a good representative of the dataset. In the previous section we saw two approaches of defining what a middle trajectory is in general. Since we know of no straightforward way to incorporate time stamps in a middle trajectory, we first discuss some properties that the spatial part of a trajectory has to satisfy in order to be acceptable. We first review properties that are suggested in the literature and decide which of these properties to use. Next we propose two time-based properties that a middle trajectory should fulfill.

2.2.1 From the literature

Van Kreveld and Wiratma list eight properties [21] they suggest for a middle – in this case a median – trajectory with respect to a collection of input trajectories. Even though the properties are designed for a specific median trajectory, we think they are general enough to apply to *any* median trajectory. The spatial properties they require for their median trajectory (that does not include time stamps) are:

1. It should be a continuous polygonal curve (from the source to the destination of the trajectory).
2. It should be central in a local neighborhood with respect to the positions.
3. It should visit regions if and only if many trajectories do so.
4. The length of the trajectory should be comparable to the length of the input.
5. The total angular change of the trajectory should be comparable to the total angular change of the input. See Figure 2.3.
6. It should be robust against outliers.
7. It should be composed of parts of the input trajectories.

The final, general requirement for a middle trajectory that is mentioned in [21] is:

8. It should be efficient to compute.

In addition to these properties, Buchin et al. [9] suggest a number of desirable or even required properties for a similar median trajectory, which again does not include time stamps. Besides the properties 1 and 4–7, they list the following two properties that are not already mentioned in [21]:

9. Any edge of the median trajectory has the same direction as the edge of the input trajectory on which it lies.

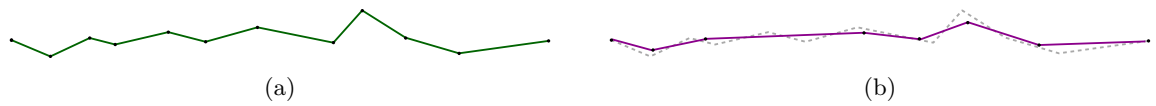
10. The number of vertices of the median should be comparable to the number of vertices on the input trajectories.

Intuitively, most properties can be classified into three groups: being spatially in the middle (2), properties related to the shape of the trajectory (1–3 and 7) and having similar attributes (4, 5, 9 and 10). Property 2 states that the median trajectory should not only be central when looking at the complete trajectories, but also for a part of them. In other words, if we would pick any point on the middle trajectory, then this point should be central to all sub-trajectories within some radius with respect to their spatial positions. Property 3 states that the median trajectory should not visit regions if this region is visited by only a few trajectories, but the trajectory should visit a region if a significant portion of the input trajectories do so.

We want the middle trajectory to represent the input trajectories as well as possible. Hence, if only a small number of trajectories (e.g. one or two out of ten) appear to deviate markedly from the other trajectories, then these outliers should not influence the median trajectory by much. In other words, the middle trajectory should be robust with respect to outliers (property 6). A consideration to be made here is whether such outliers should be taken into account at all (i.e. like a "mean" trajectory or simply having a lower influence), or should be left out completely during the calculation.

The main reason for property 7 to exist is best explained with an example. Suppose that we want to find the median trajectory of a set of hikes with the same source and destination, where at some point there is a lake on the path and about half of the trajectories go around the lake on the left side and the other half takes the right side around the lake. A mean trajectory would go through the lake, which is something we do not want. However, if for example we want to analyze the flight paths of birds, then there is no problem of flying over the lake if the birds do not clearly avoid flying over the lake. In other situations more information about the area might be known, showing that there is in fact a bridge that goes over the lake that has not been taken by any hiker. So it depends on context whether property 7 should be regarded as a requirement.

Mean trajectories do not necessarily contain edges directly corresponding to (parts of) edges of the input trajectories on which they are based. If we weaken property 9, then it can be translated to mean trajectories. The weakened property then becomes valid if and only if the direction of the edge itself or its inverse is similar (e.g. within 5 degree) and not considering all intersections with the edge. However, it is still unlikely that the property will be satisfied for mean trajectories. On the other hand, the property does make sense for trajectories that are based on a median.



(a) (b)

Figure 2.4: An animal does not usually move in a zig-zagging pattern like shown in (a). A smoother middle trajectory (b) is preferred in most cases.

The need for the middle trajectory to have the properties 5, 7 and 10 is debatable. Property 5 states that the total angular change should be comparable to the input. However, a *smooth* middle trajectory (as in not containing too many local detours) is in most cases more realistic and thus the sum of all angular changes (see Figure 2.4) is (much) lower. A better property might be to state that the total angular changes are not much higher than the input trajectories, but for trajectories with lots of intersections or many sharp angles we actually do want the original property to hold. A possible consequence of having a smooth middle trajectory is that we might be able to merge points, so a similar argument can be made for property 10. For property 2 the usefulness depends on what is considered the local neighborhood. In the example with the lake (of any size), the middle trajectory could be the outermost trajectory of the local group that goes around one side of the lake, but in the middle if we consider the trajectories that go around the other side of the lake as well. We do not consider such a scenario to be a bad thing.

A consequence of property 3 is that the length of the middle trajectory can be (much) shorter if the samples on which the trajectory is based make detours at different locations, like in the tourists example we mentioned before. Thus having a property that states the lengths should be comparable (property 4) can be contradicting. Instead, we implicitly assume the property implies that the length should be not (much) longer than the sample trajectories and can be shorter when these samples have long individual detours.

2.2.2 Adding time

Besides the properties 1–10 that are used for definitions not taking the temporal component into account, we propose two additional properties that do take time into account:

11. The duration of the median trajectory should be at most as long as the longest trajectory in the input set.[1]

12. The speed of the median trajectory should be comparable to the speed of the input trajectories.

Since trajectories are given as sequences of ordered points, the duration is simply the difference in time between the first and the last data point. Checking the speed (e.g. for property 12) on the other hand is more difficult if we base it on the original time stamps: a middle trajectory can be shorter than all of the input trajectories. This difference in lengths means that, in combination with the use of similar time stamps, we find (much) lower speeds. An example where we would find such low speeds is when the input trajectories contain a lot of intersections and make a local "detour" at some point, see Figure 2.6. In these situations the average speed is meaningless and both the minimum and maximum speeds are very low.

[1] To allow for shortcuts, a similar lower bound cannot be defined.

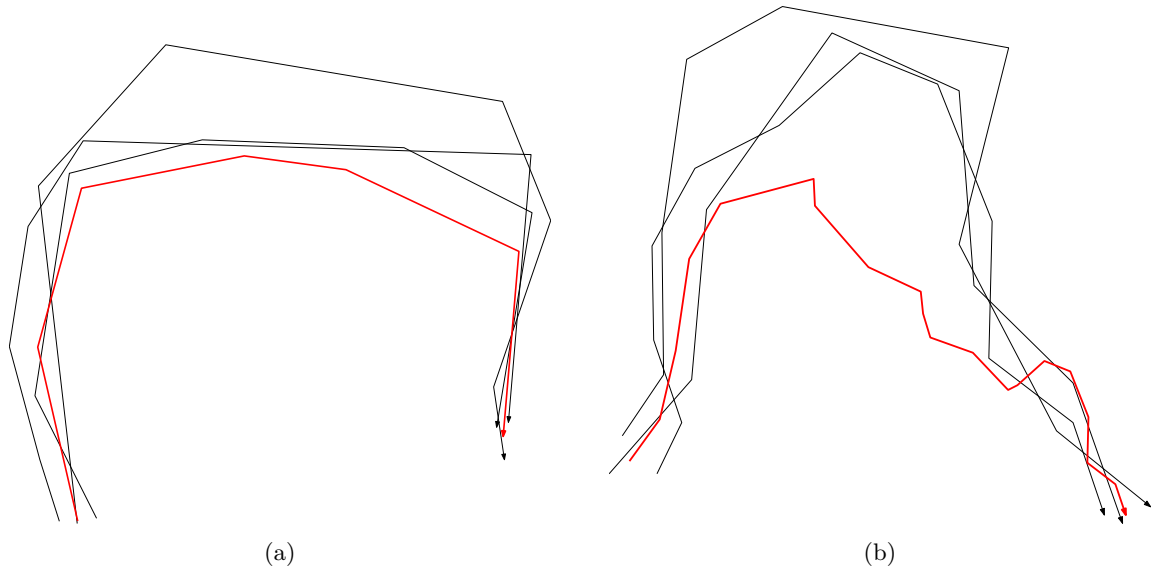
The way the speed is determined influences the duration of the middle trajectory. If the calculation of the speed is based on the input, then the total duration can be a lot lower (similar speeds in combination with a shorter distance). This means that the calculated trajectory cannot be used to determine at what time a certain location will be visited. On the other hand, if the calculation is based on the total duration, then the middle trajectory cannot be used to determine the speed. As a result, the middle trajectory should only be seen as a series of points where the time stamps represent the time at which the location will be visited, without having any information for what happens between two points.

Regardless of the method of choice, we can set an upper bound on the speed that is based on the highest speed in the input. If this bound is exceeded a negative speed is found, then the result is most likely not a realistic representation of the input and either needs to be refined or rejected as a usable result. An example of a scenario causing a trajectory to exceed these speed bounds is when a trajectory spans great distances in a really short time. In this thesis we refer to this problem as *jumps*. Negative speeds can be caused by using the time stamps of nearest neighbors that lie in the past with respect to the previous point.

The input should be filtered before any of the algorithms is applied to prevent situations where input trajectories and as a consequence also the middle trajectory contains implausible speeds as a result of measuring errors. For example, when the goal is to find a middle trajectory for a bird dataset, then we cannot expect speeds of 500 km/h to be realistic and should be prevented. Similarly, an average speed of 1 km/h is not what one would expect of a middle trajectory for a bird flight. However, checking for this average speed is quite meaningless if the middle trajectory is a lot shorter than the input and we base time stamps on expected arrival times: In this scenario the average speed will nearly always be (close to) zero.

When the calculated middle trajectory has a similar length and speeds as the input, the two different methods of determining the time stamps are not likely to have a major influence on the results; Either method can be used to give an indication of what to expect for a middle trajectory. Therefore one should keep in mind that either the duration or the speed might be invalid for the approach that was used. In the case where the speed does have some meaning, checking the validity of the time stamps comes down to comparing the average speed and the maximum speed to what is realistic for the type of data.

If we want to include the temporal aspect in property 2, then we could also add the requirement that the trajectory should be central to the sub-trajectories within a given time. However, finding a middle trajectory that satisfies the property both temporally and spatially can be difficult. The satisfaction of the spatial aspect does not guarantee the satisfaction of the temporal aspect and vise versa. This can be seen in the example shown in Figure 2.5, where we calculate the positions based on the time stamps. Even though the same time stamps are used to calculate the positions, therefore guaranteeing that the calculated positions are temporally in the middle, the trajectory is visibly not in the middle with respect to the spatial positions. With the exact same time stamps on each trajectory (a), the middle trajectory does have a similar shape. With different sampling rates – trajectories are sampled with respectively 30, 60 and 90 second intervals – the effects on the spatial aspect can be much worse (b).



(a) (b)

Figure 2.5: A trajectory where we use groups of equal time to determine the spatial position (based on the mean position of the coordinates). The result of an equal sampling rate is shown in (a) and the result of using different speeds is shown in (b).

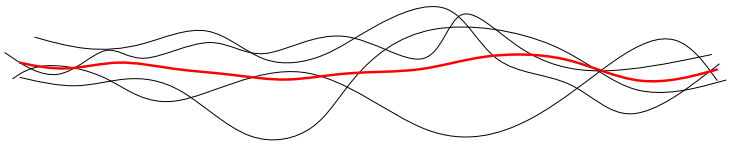


Figure 2.6: The middle trajectory is shorter when the trajectories in the sample set make local detours that are specific to only one trajectory or when these detours are "canceled out" by other trajectories.

2.3 Definitions

The notions mean and median are well known in the fields of mathematics and statistics. In the simplest form (i.e. arithmetic means and medians) they define what the middle value is in a collection of numbers, respectively the average numerical value and the value that separates the higher half of a sample from the lower half of a sample. For points in a plane on the other hand, there exist multiple definitions.

2.3.1 Point sets

In a multi-dimensional space, there are several notions of being a middle point. We first describe some that are based on points sets, i.e. based on either a mean or a median. Since our goal is to find points that lie in the middle for a given set of trajectories, we use the Euclidean distance metric whenever a distance has to be calculated. The definitions are not limited to this assumption however and can easily be replaced with some other metric, such as the Manhattan distance (also known as the rectilinear distance or the L_1 distance).

Means

The first definition is simply the notion of a mean extended to multiple, say d, dimensions. For simplicity we assume an arithmetic mean that is the sum of the coordinates divided by the number of coordinates in the set. The mean minimizes the average distance to all points in the set, but need not be one of the sample points. The calculation of an arithmetic mean can be done in linear time.

Minimizing the maximum distance

Instead of using the average distances, we can calculate the point that minimizes the maximal distance to the sample points. This point is the center of the smallest enclosing disk (SED) – called a ball in more than two dimensions – and, like the mean, is not necessarily part of the input. The problem can be solved in $\mathcal{O}(n)$ expected time by a randomized algorithm for a set of points in two or three dimension [30].

Projections

The previous two methods often have a solution that is not part of the input set. Sometimes it is desirable to find a point that is. One possible method of doing this in the plane is to draw a line through the set and project the points onto this line. Applying a standard algorithm to find the (lower) median in this one-dimensional projection gives a point that is part of the input and lies in the middle, however, this means we are ignoring all but the first dimension and hence the method is sensitive to outliers. Choosing a bad projection line can result in finding a point that lies (relatively) far away from the other points and therefore is unusable for a middle trajectory.

Weber points

One possible approach to overcome the problem of picking an outlier as the middle point is to apply the same approach as for finding a mean point, but to restrict this to the input points. That is, finding a point that minimizes the sum of distances to the sample points. Such a point is called a *Weber point*. The Weber point can be calculated in quadratic time.

Centerpoint

It is also possible to generalize the definition of a median to a higher-dimensional Euclidean space. A centerpoint of a set of n points in d space is a point such that any hyper-plane that goes through the point divides the set in two subsets of at least $\lceil n/(d+1) \rceil$ points. Like the standard definition of a median, the centerpoint need not be one of the sample points. However, we can use an approximation, though possibly of bad quality, that is part of the input points by picking the point that is nearest to point found by the algorithm.

Computing the centerpoint of a finite planar set of points of size n can be done in linear time by pruning the input [18]. The procedure for this is to choose four open half-planes such that each contains a minimum of $\lceil n/3 \rceil - 1$ points and their closure contains $\lceil n/3 \rceil$ points. Then pick one point in each of the four pairwise intersections of adjacent half-planes and replace these points by their Radon point (i.e. the point that lies at the intersection of a pair of lines through the

points) until one of the intersections is empty. This is repeated until the set is small enough to apply a bruteforce algorithm.

2.3.2 Trajectories

Not all definitions of mean and median trajectories are based on generalizations of their one dimensional definitions. In this section we describe several of the existing definitions for notions of middle trajectories that are based on some kind of a mean or a median [9, 13, 14, 21].

Simple median trajectory

The simple median trajectory as defined in [9] is a median trajectory that starts on the middle trajectory and always switches to another trajectory in the forward direction at an intersection point. Let s be the point incident to the outer face of the arrangement of curves corresponding to the trajectories in which all trajectories start. We can order the m edges adjacent to s with the first and last edge adjacent to the outer face. The middle trajectory then is the one whose first edge is the $\lceil m/2 \rceil$-nd edge in the order.

Buchin et al. show that under certain conditions, such a median is always in the middle. They prove that the median trajectory satisfies the property that for any point p on the trajectory, the minimum number of distinct trajectories that p must cross to reach the unbounded face (including the one(s) on which p lies) is $(m^* + 1)/2$, where m^* is the number of non-outlying trajectories in the set. Moreover, a path leaving s on the k-th edge that switches in the forward direction at every intersection will end at t arriving on the k-th edge as proven by Lemma 3.1 in the paper.

The main problem with the definition is that the simple median will nearly always fail to satisfy the properties 2–4, 6 and 10 when the trajectories are self-intersecting (as in Figure 2.7). This limitation can be a problem when we want to find the trajectory that most resembles the habitual routes of homing pigeons, where it is not uncommon for trajectories to self-intersect.

The simple median trajectory can be computed in $\mathcal{O}((nm)^2)$ time, where m is the number of given trajectories and n is the maximum complexity of any trajectory [9]. For practical situations, the algorithm has been improved to $\mathcal{O}((nm + k)\alpha(nm)\log(nm))$, where α is the inverse Ackermann function and k is the output complexity.

Homotopic median trajectories

Like the simple median trajectory, the calculation of the homotopic median [9] is based on switching trajectories. The difference is that the homotopic median does not switch at *every* intersection point. When trajectories self-intersect, a point p is placed in the face(s) around which it loops. Switching trajectories is only done if the median calculated up to the switching point is of the right homotopy type, determined by the signature of the (sub)trajectory. This signature is the intersection of the trajectory with the vertical line through each point p and side of the intersection (i.e. above or below p with the values p^+ and p^-, respectively). See Figure 2.7 for an example.

Van Kreveld and Wiratma give examples where the homotopic median trajectory fails (based on the eight properties defined in [21]): if all trajectories make a detour that is back-and-forth over the same path, then no relatively large face is present to place a point in. Such a detour is

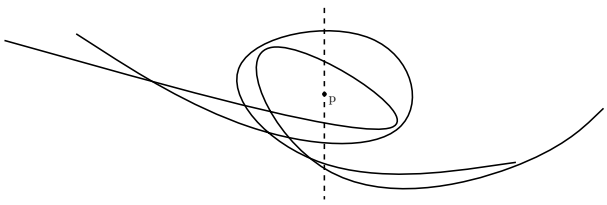


Figure 2.7: Two trajectories that loop around the same point p. The signature of both trajectories is $p^- p^+ p^-$: The first time the trajectory crosses the line through p, it is below p. The second time is above p and the third time is below p again.

incorrectly ignored by the homotopic approach. Another example is when all trajectories are of a different homotopy type (e.g. when no large subset of homotopically equivalent trajectories exists), which results in one of the input trajectories being the median. Using a single input trajectory as the median might result in the computed trajectory to include regions that are visited only by a single input trajectory.

The homotopic median trajectory can be computed in $\mathcal{O}((nm)^{2+\varepsilon})$ time for any $\varepsilon > 0$ [9].

Majority median trajectories

Another approach to computing a median trajectory is called the majority median trajectory [21] (or buffer median in [32]). In the paper, Van Kreveld and Wiratma present a completely different approach that uses a subset parameter m' with $m/2 < m' \leq m$. In this definition, an edge e of a trajectory is called *useful* – the edge might be a part of the resulting median trajectory – if at least m' trajectories are within a distance of at most δ from e, where δ is a given distance parameter. In other words, an edge e is called useful if a buffer of size δ around e intersects at least m' other trajectories.

A median is accepted when it only contains useful edges and all of these edges are either *happy* or *ignored*. A so-called happy edge is a useful edge that is either part of the median or is within a distance of at most 2δ from any edge in the median. The algorithm will try to compute a better median by including more unhappy (i.e. neither happy nor ignored) useful edges. This is done by including the edge f furthest from the median as long as such unhappy edges exist. This is done as follows. The shortest path from the source s to all endpoints of edges with distance at most δ from this furthest edge f. Then the shortest paths from these endpoints to the target t is computed. Finally, these are combined to find an overall shortest path from s to t that uses some edges near f (i.e. with a distance at most δ). If such a path does not exist, then f and all other nearby edges will be ignored.

The worst case running time is $\mathcal{O}(n^3 m^3 \log(nm))$ under the assumption that any edge of any trajectory intersects a constant number of edges of other trajectories (making a total of $\mathcal{O}(nm)$ edge-edge intersections).

Equal time based approach

Another approach of finding the middle trajectory out of a set of movement data was suggested by Etienne, Devogele, and Bouju [13]. This middle trajectory is called the *main route* in the paper. They employ an approach of analyzing movement data by extracting patterns of one

itinerary from a large dataset of mobile object. Their algorithm works in two steps: They first filter out erroneous trajectories that can be detected using the calculated speed of the position compared to the maximum speed of a moving object of the (known) type. Then a temporal normalization is applied. Next, they compute a median position for each point of the trajectories processed so far. This last step is done by having a subset of points (one for each input trajectory) for each position on a trajectory. The corresponding positions in the other trajectories are interpolated using their normalized time stamps.

Trajectories with a significant gap between two consecutive positions and erroneous positions are filtered out to improve statistical analysis. Moreover, positions that fall within the departure and arrival zones are filtered out to prevent the measurement of biases in the spatio-temporal patterns. Trajectories are then simplified (to speed up computations) using a spatio-temporal variant of the Douglas & Peucker-filter, with as goal to retain only significant positions of a trajectory while keeping information about both speed and heading changes.

Time stamps of positions are normalized in order to compare trajectories. The normalization procedure consists of simply calculating relative time stamps for each position such that all trajectories will start at $t = 0$. To avoid spatial distortions introduced by the (by assumption, slightly) different speeds of objects, positions of other trajectories are interpolated using a normalized time to ensure that they end at the same relative (normalized) time. Median positions are then computed for each of the coordinates in each position subset. The exact method to determine these median positions is not mentioned in [13], but we discuss this subject in more detail in Chapter 3. The next step is to order positions according to their normalized time to create the "main" route. Finally, this route is filtered by the same method as before to retain only significant positions.

The main part of the algorithm described in [13] is not the most efficient way of solving the problem in practice, but has asymptotically the same running time as more efficient alternatives. For a real world dataset it is unlikely to have exact locations known for all (equal) time stamps. Therefore, two steps are required to overcome this problem in general: First normalize the triplets to relative times (i.e. offset from first time stamp) and then interpolate positions for missing times. Since each trajectory contains $\mathcal{O}(n)$ points and there are m trajectories, each of which might contain unique time stamps, we have $\mathcal{O}(n \cdot m)$ data points to process in the worst case for each input trajectory. This gives for a total running time of $\mathcal{O}(nm \cdot T(m))$ for the main part of the algorithm, where $T(m)$ is the time it takes to process m points of equal time.

Mean-based approaches

So far we have seen several definitions of so-called median trajectories that are loosely based on the definition of a geometric median where one tries to minimize the sum of distances to the sample points taken from a discrete input set. Another possibility is to calculate the middle trajectory based on a mean value of the input trajectories.

Freeman et al. [14] use an iterative approach where they start with a straight thread – a polygonal line – between the source and target point and refine this path in each iteration. They first take a subset of the input data and place these points on the initial (straight-line) path. Then, at each iteration, each of the points on the thread is moved towards the average position of its nearest neighboring points in the original input. Points for which the minimal distance to the neighbors is greater than a given threshold are moved to their midpoint between

the current point on the thread and its neighbor to maintain an "even" distribution of the points along the path. This process is repeated a constant number of times and results in a sequence of points that lie at the average nearest neighbors of the original trajectories, a *mean path*.

The running time of the iterative algorithm is determined mostly by the time it takes to perform nearest neighbor queries for each point. More specifically, the time complexity of the algorithm can be described as $\mathcal{O}(n+m\cdot T(\textsc{BuildNnStructure}(n)+k\cdot T(\textsc{QueryNnStructure}(n)+1)))$, where k is the number of iterations to perform. The construction of the data structure can be done in $\mathcal{O}(n \log n)$ and finding the nearest neighbors can be done in $\mathcal{O}(\log n)$ time for each search, making the total running time of the algorithm $\mathcal{O}((n + k)m \log n)$. Since the problem is actually a variant of the nearest neighbor search problem, namely all nearest neighbors, it might in practice be more efficient to use an algorithm that exploits the information between searches [12, 29].

2.4 A middle trajectory with time

We have discussed many approaches, but only one of which incorporates time stamps in the result, viz. the algorithm by Etienne, Devogele, and Bouju [13]. The idea behind their approach is to interpolate the positions of the trajectories such that every time stamp in the dataset appears on all trajectories. A similar approach can be used in combination with the point set algorithms we have discussed in Section 2.3.

2.4.1 Equal time approach

The first possibility of adding time stamps to a trajectory is to consider points with the same time stamp of each trajectory as being in the same group. Then each group contains only points with the same time stamp and thus we only have to consider their location. For these points we calculate the middle point.

Let $\tau_i(t)$ and $\tau_j(t)$ be the positions at time t of the trajectories τ_i and τ_j respectively. We define the middle trajectory as the trajectory that connects the middle points of each group such that the point at time t is connected to the middle point of the group at time $t+1$. When the goal is to find a trajectory that has the shortest average distance to all points within a group, we want to find positions for the middle trajectory τ_{mid} such that the following formula is minimized:

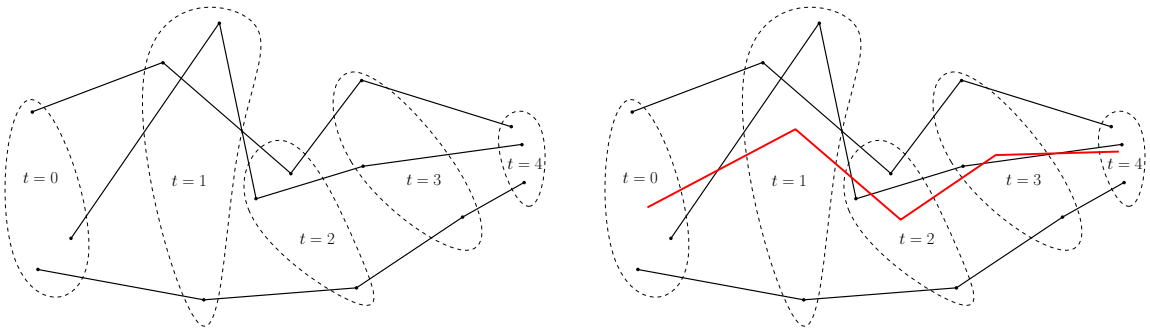
$$\sum_{0\le t\le n} \sum_{1\le j\le m} \frac{1}{m} d(\tau_{mid}(t), \tau_j(t)),$$

where $d(\cdot, \cdot)$ is the distance function between two points.

Similarly, computing the trajectory that minimizes the maximal distances involves optimizing the following formula:

$$\sum_{0\le t\le n} \max_{1\le j\le m} \frac{1}{m} d(\tau_{mid}(t), \tau_j(t)).$$

In fact, we can apply any of the point set based algorithms from Section 2.3 to each group to find a trajectory that lies in the middle with respect to the measure used by the algorithm. This approach is illustrated in Figure 2.8 where we have three trajectories with each five points,



(a) (b)

Figure 2.8: When using equal times, the middle trajectory can be obtained by connecting the middle points of each group. These can be the average points (a), centers of SEDs (b) or a middle point from any of the other methods. The thick lines represents the middle trajectory for each dataset.

grouped at equal time. It should be noted that equal time medians use a different definition of a median trajectory than used in [9]: these median trajectories from the equal time algorithms only use points of the input, but there is no such restriction for the edges.

The projected median trajectories are calculated as follows. For each group of points, a middle position is determined; We use the mean for this. When the position of two consecutive groups is known, the line segment connecting these points is determined and finally the points at the current time are projected onto the line perpendicular to the line segment we calculated.

Challenges

The equal time approach is meant for finding positions at a certain time, not to determine average speeds. This is due to the fact that a middle trajectory can be much shorter. However, there are situations where the middle path obtained by this approach is not the result that is desired. In the case where the trajectories are the result of objects moving along similar paths, but at completely different speeds, then the spatial part of middle trajectory can be far away from any of the input paths if we do not restrict the output to one of the data points of the sample set. An example is shown in Figure 2.5 where the mean path calculated by an equal time based approach does not lie spatially in the middle. One way to prevent this case from happening is to make sure that the trajectories have a similar duration. This can be accomplished by stretching (or compacting) some of the input. However, this has to be rolled back to determine the actual time for points on the trajectory. Moreover, multiplying the time stamps based on the start and end points alone is not enough when the speed of the trajectory is not constant: one section might be traveled very slow, while another was traveled very fast. Hence, a trajectory with non-constant speed can have the same duration, but still be problematic. In the pigeon example, this can happen when a fast bird makes several stops along the way.

On the other hand, using only points from the input might cause other undesired effects. Consider a set of points where the calculated middle is just in favor of one point p, with a very close second candidate q. For the next set of points this might be the other way around, causing

the algorithm to choose a completely different point q' as opposed to one close to p. For sets of points this is no problem, but when these are used in a sequence as with a trajectory, then some undesired jumping can occur. This problem is illustrated in Figure 2.9 where it would make more sense if all middle points would be on the same input trajectory.

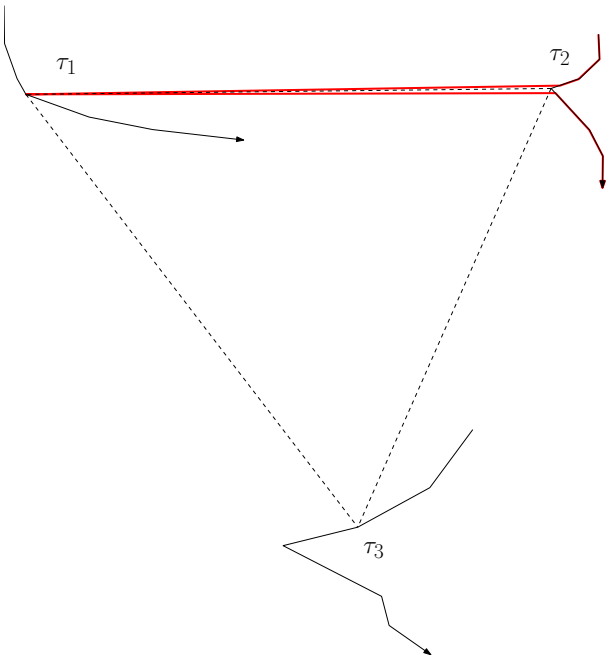


Figure 2.9: In this Weber point example the jumping is caused by having two sets of points with a nearly equal distance to the other points. The middle trajectory is shown in solid red using most of τ_2. The dashed line indicates the distances at the point on which the jump occurs, which are almost equal.

Nonetheless, the approach can be used if the input is roughly the same. For datasets that contain smooth paths, a meaningful average speed can be determined. When one wants to calculate a middle trajectory with highly dissimilar input paths (e.g. paths going in opposite directions), using a mean based approach may be the only of the approaches discussed in this thesis to get some sensible output.

2.4.2 Adding time stamps as a post-processing step

Instead of dealing with the temporal aspect during the calculation of the middle trajectory, we can also add time afterwards. The iterative algorithm by Freeman et al. [14] calculates a mean path based on the positions of nearest neighbors. These nearest neighbors also have time stamps. Hence, after the last iteration has completed and the spatial part of the middle trajectory has been calculated, these neighbors can be used to determine the time stamps for the middle trajectory. The most obvious approach is to assign these time stamps based on a mean of the inputs, for example by assigning the closest neighbor the highest weight. There is one slight problem with this approach though: Nearest neighbors can come from virtually anywhere on the input and therefore taking a mean need not guarantee that these time stamps are always non-decreasing. Calculating non-decreasing time stamps is not very difficult when

we are allowed to use any time stamp, however, our goal is to assign time stamps that can be used to calculate when to expect an object at a certain position.

One possibility to guarantee non-decreasing time stamps is to only use nearest neighbors that lie in the forward direction. Unfortunately, this approach does not work. While preventing the use of points that lie in the past, there still exists a possibility of using points that lie far in the future that cause parts of the input to be skipped. This is shown in Figure 2.10.

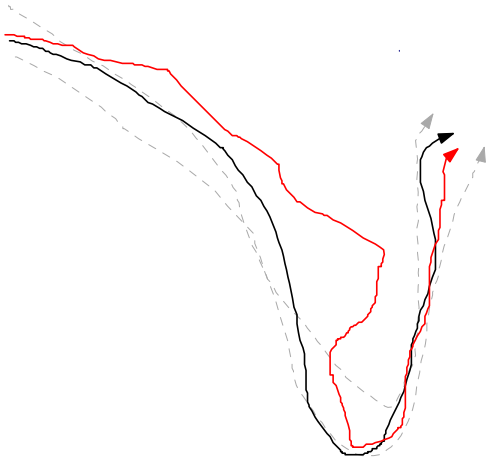


Figure 2.10: The dashed black trajectory represents an input trajectory and the red trajectory is the middle trajectory. After the turn at the bottom, the middle trajectory contains points close to end of the input trajectory and will therefore skip a large part of the input trajectory.

Using a single trajectory out of a set of m trajectories to determine the time stamps for the middle trajectory is not sufficient to get representative results. Once we have found the correct time stamps for all of the input trajectories, the next step is to use this information to calculate the time stamp for the middle trajectory. In this process, we are facing the same problem of potentially decreasing time stamps. Ideally, we want to use a middle value for the time stamps (i.e. the mean value).

Whichever method is used, simply using all these middle values does not guarantee non-decreasing values. Therefore, we should also keep track of the minimum and maximum value for each point to make corrections. However, this only works for most real world datasets. To always assign meaningful time stamps, we need another method. Using only nearest neighbors is not enough, since that does not remove the possibility of skipping large sections. Therefore, the shape of the path itself should be considered. An idea is to use a construction much like how the Fréchet distance is computed [2], but with the extra requirement to penalize the points based on the difference with the computed optimal value.

Instead of assigning time stamps afterwards, the temporal component of the points can be used as an extra dimension in the nearest neighbor queries. These neighbors can then be used to extract the time stamp for the position. Unfortunately, doing so is not trivial. The spatial aspect of a trajectory is not comparable to the temporal aspect and therefore one needs to assign a weight to each of the components to give scores to the points.

2.4.3 A second approach

When a middle trajectory is based on segments of the input, there is another way of calculating time stamps. For these segments we know the speed of travel (of the original trajectory) and therefore we can calculate how long it takes to travel a part of the segment. The difference with the previous approach is that we now know the average speed and the total duration. The only downside of this is that we assume the middle trajectory to be followed, while in fact this might not be true in reality in which an object does not travels from one point to the next in a straight line.

Both the simple and the homotopic median include all intersections in the resulting trajectory. A slight deviation is to skip these points and use only the data points that exist in the input sets. This modification reduces the number of points on the trajectory, but we lose the property that the trajectory always goes around (fixed) obstacles. The calculation of the speed is slightly different in this case, as using the same time stamps when the total distance is shorter – it might be shortcutted – can be considered incorrect. Instead, we calculate the average speed over the input segments and use this or some other interpolated speed to determine the next time stamp.

2.4.4 Freedom of choice

Calculating time stamps based on neighboring points is certainly not limited to the iterative algorithm. Likewise, using a speed-based method such as described in Section 2.4.3 is not limited to the homotopic medians. These two methods both compute time stamps, but not necessarily of the same "type". Using the nearest neighboring points to calculate time stamps works the same for all types of middle trajectory. With a speed-based approach we need to be more careful.

It is easy to calculate the speed of one segment when the original time stamps are known. However, when the middle trajectory is the result of a mean, it is not as easy as simply using the same speeds as used on a known segment. It seems obvious that one should do something with a weighted mean that uses either the nearest segments or points. Using segments of the input trajectories nearest to the computed middle point alone can lead to difficult situations when segments are in opposite directions. Moreover, since nearest segments need not be consecutive with respect to the input trajectories, the previous point (or segment) on the middle trajectory needs to be taken into account. An approach to calculate the speed-based time stamps for the mean approaches is the following. Let $\tau_{mid}(t)$ and $\tau_{mid}(t-1)$ be the current point and previous point on the middle trajectory, respectively. Moreover, let μ_j be the average speed of the path between the points used to determine the position of $\tau_{mid}(t-1)$ and $\tau_{mid}(t)$ for some trajectory j. A weighted mean $\sum_{j \in T} w_j \mu_j$ of the averaged speeds can then be used to calculate the time stamps.

2.4.5 Time complexity

The running time of either method depends on whether nearest neighbors and the corresponding data structure have to be calculated. Using the speed-based method on a median trajectory only takes linear time in the complexity of the output trajectory, for which we use the variable n'. Equal time mean-based approaches can use a mean of the values at the current time and thus is only a factor $\mathcal{O}(m)$ slower. Likewise, if the calculation of the time stamps of any type

is done in the last iteration of the iterative algorithm, it is sufficient to only compute mean of the time stamps of the points that are already required to determine the position, resulting in a time complexity of $\mathcal{O}(m \cdot n')$. To compute the nearest neighbor based time stamps of any other algorithm one cannot avoid finding all the nearest neighbors from scratch.

2.5 Evaluation of the properties

In Section 2.2 we have looked at twelve properties in total. Most are based on the spatial aspect of the trajectory, which is arguably the most important. Not all of the properties mentioned in the literature should be fulfilled without question. It really comes down to what the goal is to construct a middle trajectory in the first place.

Property 7 will, by definition, be satisfied by median trajectories and most likely not by mean trajectories. The property can be easily checked by looking at how a middle trajectory is constructed. Likewise, property 1 depends on the construction of the trajectory and will, for the approaches discussed in this thesis, always be satisfied. What remains are eight properties that we will check each computed middle trajectory against. These are the properties 2–4, 6, 8, 9, 11 and 12. In this section we evaluate which properties are satisfied by which algorithm and under what circumstances.

Since most properties were meant to be satisfied by a median trajectory, some properties only apply if the property is weakened and others do not apply at all to mean trajectories. Moreover, we only consider algorithms that are efficient to compute; A middle trajectory can be computed in $\mathcal{O}((nm)^2)$ with the algorithms consider for the evaluation, with m generally being quite small.

Recall that the properties we still had to evaluate are the following:

2. It should be central in a local neighborhood with respect to the positions.
3. It should visit regions if and only if many trajectories do so.
4. The length of the trajectory should be comparable to the length of the input.
6. It should be robust against outliers.
9. Any edge of the median trajectory has the same direction as the edge of the input trajectory on which it lies.
10. The number of vertices of the median should be comparable to the number of vertices on the input trajectories.
11. The duration of the median trajectory should be at most as long as the longest trajectory in the input set.
12. The speed of the median trajectory should be comparable to the speed of the input trajectories.

The properties 11 and 12 can always be satisfied individually when using the corresponding methods to assign time stamps. We therefore only consider the combination where both properties are satisfied using either of the two methods.

2.5.1 Equal time means

Since the averages based approach and the SED approach are very similar, we consider them both at the same time for the evaluation of the properties.

The satisfaction of property 2 depends on what is considered central. Although both find a point in the middle with respect to the points at the specific time stamp, this middle point need not be central with respect to the nearest neighbors of the middle point. Therefore property 9 can fail to be satisfied when trajectories come close to one another at different points in time.

Property 6 does not hold for the equal time based means. As a consequence, neither do the properties 3 and 4: an extreme outlying trajectory can pull the middle towards the outlying trajectory such that it can make a significant detour through a region not visited by any other trajectory. Likewise, a much longer input trajectory does not satisfy the properties 11 and 12.

Examples exist where none of the properties is satisfied. In fact, just a single outlying trajectory can make this happen. However, in most realistic cases we expect that the properties 4, 11 and 12 will be satisfied.

2.5.2 Equal time medians

All equal time based methods only consider a local group to determine the middle point. Therefore, property 2 may or may not be satisfied depending on how the neighborhood is interpreted. Although we consider it as a spatial property, the neighborhood can also be defined by the groups of equal time. Moreover, there is no way of determining whether selecting a middle point leads to a region that is visited by many of the input trajectories. This does not mean that the property does not hold when applied to a set of similar input trajectories. In this case, the median will not include outliers and what remains, if the trajectories are close together, are parts that are visited by many of the input trajectories.

Property 9 can be considered semi-satisfied: edges that are followed are always followed in the forward direction. However, when the middle trajectory jumps to another trajectory (i.e. without following an edge), the part between the two trajectories might overlap with (a part of) an existing edge.

Equal time projected median

With similar arguments as used for the equal time means, in combination with the long jumps, we can conclude that only property 9 can be satisfied. In most cases, the other properties do not hold even for real world data as we will see in Chapter 4. In particular property 2 need not hold in any case if there is no bound on the maximum distance.

Equal time Weber points and centerpoints

Unless there are few or no long jumps, property 4 does not hold for either the equal time median based on Weber points nor the one based on centerpoints. As a consequence, neither do the properties 11 and 12. The property that is satisfied for both algorithms, besides the general ones we already mentioned, is property 6.

2.5.3 Iterative mean

Not being based on existing segments, the iterative mean does not necessarily avoid regions that are not visited by the majority of the input trajectories. Moreover, it is negatively influenced by outliers. Although in many cases the total length is shorter than any of the input trajectories due to the placement of the initial thread, a single outlier can make the length much longer. Therefore, like the equal time means, the properties 4, 11 and 12 are not always satisfied.

2.5.4 Simple median

It should not come as a surprise that the simple median algorithm satisfies many of the properties it is designed to satisfy. Therefore we only consider the properties that are not (always) satisfied.

Van Kreveld and Wiratma give examples where the simple median (and the homotopic variant) do not satisfy property 3 and propose the majority median as alternative. Furthermore, in theory property 11 is not satisfied when the trajectory is constructed from edges that have a lower than average speed on the trajectory that the edge belongs to.

2.5.5 Evaluation

Theoretically, the algorithms evaluated in this chapter satisfy very few of the properties we mentioned. In reality, data can be filtered to avoid most of the problems and in most cases we probably get decent results for most algorithms. The next step is implement these algorithms and experimentally evaluate whether the methods actually work in practice and how they compare to one another in both performance and the quality of the generated trajectory.

CHAPTER 3

Implementation

In the previous chapter we have discussed three approaches to find a middle trajectory: an equal-time based approach, an iterative approach and the trajectory switching approach taken by the homotopic median. Moreover, several algorithms have been proposed based on slight deviations of existing algorithms, such as the five methods to find a middle point with the equal-time approach. Finally, we have looked at two methods to assign time stamps based on both estimated times of arrival and calculated speeds.

Implementing algorithms based on incomplete descriptions in papers can be a challenge. Even when pseudo-code is available, small details are often missing for seemingly trivial algorithms. Hence, we cannot guarantee that the conversion from these descriptions to a computer program is exactly the same as what the authors had in mind or as their own implementations. In this chapter we describe how we interpreted the algorithms, what problems we faced when implementing these in our program and how we tried to solve the problems. Furthermore, we discuss how time stamps are added in the non-straightforward cases.

3.1 Main goals

Although a major task of the program is to run the algorithms to find a middle trajectory from a set of m input trajectories, it is definitely not the only one. The main purpose is to provide a tool to easily see what a certain method does and to have the ability to compare the output of one method to the output of another. Since it is easier for the human brain to process images than large amounts of data points, we try to make this data more comprehensible by visualizing the trajectories and to provide the user with a means to interact with them.

3.1.1 Data sets

Nowadays there is a lot of trajectory data available due to the advances in technology. Unfortunately, there is no standard format in which these datasets are stored: not only the file format can be different (e.g. stored in the XML format or as a CSV file), but also the data itself can be different. For our purposes, we are only interested in three values: the position, which can be stored as latitude, longitude and possibly altitude, or as a mapping; the trajectory to which a

data point belongs to, which may or may not be sequentially stored in a single or a set of files in which every trajectory has its own file; and the time stamp, which can be stored as an absolute value (e.g. the number of seconds since epoch or a time (and date) stored as hours, minutes and seconds) or a relative value since the start of a trajectory. It might even be omitted, in which case we have to resort to the index of the data point.

In the implementation of the algorithms that are based on the work of others, we tried to use the exact same input as was used to generate the results in the papers. This makes it possible to verify that our implementation of the algorithms produce the same result as what was published in the papers. It is easy to see how data points are linked to trajectories and what time stamps to use, however, the same does not always hold for the positions, which is arguably the most important data. If possible, we used the same mapping as described in the paper, otherwise we chose the most suitable based on the data that was available (e.g. for data collected in Great Britain [14] we used the Ordnance Survey National Grid (OSNG 36) reference system).

3.2 Implementation

The main implementation of the program has been done in C++ according to the C++03 standard, together with version 4.8 of the Qt framework[1] that provides support for a large number of platforms to simplify the development of a GUI-based application. Rendering of the trajectories and all related graphics was done with OpenGL to get the best performance and have full control over the output.

3.2.1 Visual feedback

One of the drawbacks of most programs that (visually) manipulate data is that it is not possible to see what happens during the calculation. While this is by far the easiest method to implement, it does not give the user the ability to see the behavior. Even if everything is displayed directly on the screen, in many cases this calculation happens a lot faster than one can follow.

Debuggers have the ability to interrupt a program to inspect its current state. We implemented a similar function that allows the user to go through each of the algorithms in a step-wise fashion: Process a single point and then pause the calculation to more carefully inspect the state of the algorithm if something unexpected happens (e.g. picking a point that does not appear to be in the middle). For the iterative algorithm one also has the ability to pause the computation after each iteration. The feature to examine any state of the algorithm can be seen in Figure 3.1.

Implementation wise, we chose to model the algorithms in a separate thread and use semaphores such that the algorithm has to wait for the availability of a slot before processing each point. This implies full control over how many points are processed and in how much detail the computations are shown. Moreover, it is possible to abort (or restart) a computation if there is need for it.

Another issue that occurs when working with large datasets is that a user can be overloaded with too much information or have several points in a small space. To overcome this, we added

[1] More information on the framework can be found at qt-project.org.

features to zoom, highlight trajectories and the ability reduce the amount of information shown on the screen.

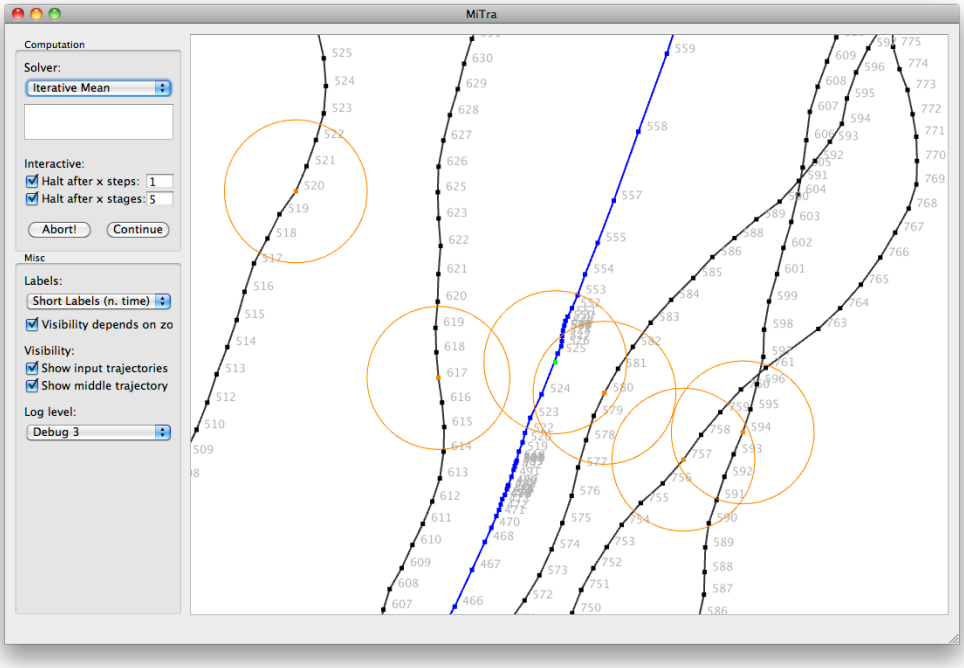


Figure 3.1: A screenshot of the user interface showing the nearest neighbors to the green middle point for each of the trajectories.

3.2.2 Equal time algorithms

The idea behind the equal time algorithms is to compare the positions at equal times. It is clear how to do this if all trajectories contain exactly the same time stamps. If they do not, then the following situations can and will arise:

1. Time stamps on trajectory a are shifted by x seconds compared to trajectory b.
2. A time stamp is available on one trajectory, but not on another.
3. The duration of trajectory a is shorter than the duration of trajectory b.

It is not uncommon to compare trajectories with completely different time stamps. The first step is to normalize these time stamps such that the relative times can be compared. We chose not to scale any trajectory, so this normalization step is to simply use the number of seconds from the start of the trajectory as the normalized time stamp.

In the previous chapter we already mentioned how we handle the case when a time stamp does not exist on another trajectory, namely by using a linear interpolation of the endpoints of the segment in which the time stamp would fall. We could have used a more sophisticated interpolation method, such as a cubic (spline) interpolation, but our goal is not to find the

exact positions in the input based on discrete measurements. Moreover, since we try to find a middle trajectory, choosing a different interpolation method has only a small influence. This is especially true for algorithms that determine the positions on the middle trajectory based on the mean location of the input points, since these points are not (directly) seen in the output.

How to solve situation 3 is a matter of taste. One cannot expect that all input trajectories have the same duration. There are several solutions to resolve this problem for normalized trajectories, namely the following:

1. Use the longest duration.
2. Use the shortest duration.
3. Use a mean or a median duration.
4. Make the duration depend on the positions.

The first three solutions should be clear. For the fourth case, when the duration depends on the positions of the input, it essentially is a special case of the first solution where the duration is equal to the duration of the input trajectory with the longest duration. It means that we "cut off" the middle trajectory if we find that the points either no longer move or the distance between two consecutive points is below a given threshold. So it can be seen as a post-processing step for the first solution.

When one trajectory has a shorter duration than another, then without making any assumptions, a middle trajectory that is based on the equal times method can take at most as long as the shortest duration. This is because we do not have more information available and therefore we do not know if the trajectory continues after the final position. However, we can (and do) make the assumption that once a trajectory reaches its final point, it remains at that point indefinitely. This allows us to use the final position for any time stamp after the last known time stamp and thus we can compare positions from all trajectories to all positions of trajectories with the longest durations.

In our implementation of the equal time algorithms we use the longest duration as the duration of the middle trajectory. We do not apply any post-processing other than computing time stamps, so even if a location does not change, two duplicate locations are retained with different time stamps.

3.2.3 Simple median algorithm

The simple median is the trajectory found by switching at each intersection. For our rather small datasets, we find these intersections by testing each of the $\mathcal{O}(mn)$ segments for each segment on the middle trajectory and do not use a specialized data structure. Once we have found these intersections, the next step is to pick the closest one to the current point, which is either the beginning of a segment or the point of a previous intersection. In the latter case we have to make sure that the previous point is not used in order to prevent a loop. After an intersection, the segment to which it belongs is followed in the forward direction. Unfortunately, we were not able to make the algorithm work correctly when a segment of the middle trajectory overlapped with a segment from the input other than the one on which it was based; The middle trajectory kept looping.

An assumption in [9] is that the simple median starts at the middle trajectory defined by two trajectories that are adjacent to the outer face of the arrangement of the trajectories. Since this is a requirement that we cannot guarantee (without pre-processing) with real life datasets – in the datasets we tested, pigeons for example nearly always flew in a circle to orientate themselves –, we chose to determine the initial trajectory based on the first two points of all trajectories without constructing an arrangement. Since this approach does not guarantee that the simple median always start at the middle trajectory, we also implemented a manual override such that we could enforce the correct start trajectory. An alternative would be to pick several locations to test whether a line through the trajectories intersects each trajectory at most once and find the median from there. This is possible, since Buchin et al. have proven that the simple median always stays in the middle. Hence, if a trajectory is in the middle at the start, it is in the middle over the whole set.

To determine the time stamps on the middle trajectory, we use the time stamps assigned to the endpoint of the segments included in the middle trajectory. If a segment is completely included, then its full duration to the calculated previous time stamp can be added. Suppose for example that x segments have already been included in the middle trajectory and that the latest time stamp that is assigned has value $t(x-1)$. The algorithm decided to add a segment s in its totality to the middle trajectory. This segment s has the time stamps $t(s_0)$ and $t(s_1)$ for respectively the start and endpoint of the segment. Let s' be the segment s after it is included in the middle trajectory. Then $t(s'_0)$ will be assigned t and $t(s'_1)$ will be assigned $t(x-1) + (t(s_1) - t(s_0))$, the "duration" of the segment added to the original time stamp $t(x-1)$. When only a part of a segment is used in the median, then we assume a constant speed over the segment and let the duration correspond to a fraction on the duration determined by the length of the part. In our implementation we did the calculation of the time stamp not as a post-processing step, but during the calculation of the middle trajectory.

3.2.4 Iterative algorithm

At first glance, the description of the mean path in [14] does not give the impression that it is hard to reproduce. However, a closer look reveals some issues when combined with the datasets that were used for the analysis of the data presented in the paper. One of the first steps that is mentioned is normalizing the input trajectories – of which only the last five are used – using piecewise cubic-spline interpolation. Since the algorithm determines positions on the middle trajectory based on the average position of its nearest neighboring points, we chose to use a linear interpolation instead. This should not have a large impact.

A closer look at the data taught us that each of the trajectories consists of more than 500 data points, so in order to get the same number of points as used in [14], sub-sampling was required. Since this can be done in many ways, we chose to use either the median of the lengths or the maximum length as the number of points to place on the initial trajectory.

The algorithm was designed to produce a satisfactory result for the input the authors used. Therefore, if we would follow its description literally, we would not always get the desired result. When creating the initial thread between the release site and the home loft, we assumed that the points are evenly distributed. A problem with using nearest neighbors to determine locations is that points may get clustered. The following is stated by the authors:

> At each iteration, each point was moved to the average position of its nearest neighbouring points on the five original tracks. To maintain an even distribution of the points along the thread, points that were more than 5m from their neighbouring points were moved to their midpoint.

One interpretation of the sentences is to move the point to the midpoint of its neighbors. This means that, in the first iteration, the midpoint between the two nearest neighbors will always be the point we started with (we do not move the two endpoints). This is not the only case in which using more than one of its nearest neighbors is problematic: When the nearest points are all on the same side of the thread (i.e. happen either both earlier in time or later in time), using the midpoint of these neighbors will cause the path to be pulled in that direction. Something that makes more sense is to choose one of these neighbors and move it to the midpoint of the current position and the position of the neighboring point. If we would choose the next point on the thread, we would move it based on an old position. However, consistently choosing one side might create a cluster of points and we still may not get an even distribution. To prevent these scenarios from happening, we chose to alternate using the previous and the next position by using the last point of the previous iteration as the first point on the next iteration.

Instead of using a fixed number of iterations, one could argue that it might be better to detect changes and stop when there are only few of them. However, a downside of using nearest neighbors is that there can always be changes. In some experiments we have seen that after a while – mostly when the distribution of points started to stabilize – only two or three states of the middle trajectory remained. Besides small changes caused by the methods to prevent clustering, significant changes between these states existed. It should be noted however that the effect was noticed with a small number of input trajectories (at most ten). We did not do any extensive tests, nor did we try to find a solution. An example can be seen in Chapter 4

The last deviation to the original algorithm is that we chose not to hard-code the values mentioned in the paper (500 points, 100 iterations, 5 meters, etc.), but to make these variables that can be set by the user.

3.3 Adding the temporal component

The original algorithms do not deal with the temporal aspect of the trajectories. The main focus of the experimental evaluation is that we also compare the time stamps of trajectories, hence we need to have a method to add these time stamps. In Section 2.4.2 we discussed two methods to add time: using the average time stamp of nearest neighbors and using the speed of the original segment (for median trajectories) or nearby segments (for mean trajectories) to compute a new time stamp. The obvious approach to compute the time stamps that we cannot directly derive from the input is to use the time stamps of the nearest neighboring points. However, using these neighboring points has the disadvantage that we might find time stamps that lie in the past with respect to the previous time stamp we found. One possibility to prevent these erroneous time stamps is to add the durations as we go along, much like using the speed of single segments with the simple medians. Unfortunately, time stamps calculated by this approach might no longer correspond with arrival times at these locations, but assume that the path is followed without any detours. This is an assumption we cannot always make.

In our implementation, time constraints forced us to make an assessment of either trying to find the best neighbors using a complicated algorithm or to use an approximation that works in most cases. Although we briefly described an algorithm to solve the problem in the previous chapter, we chose to implement the latter one and used a standard nearest neighbor graph provided as a library. Specifically, we used ANN, a library written by Mount and Arya that supports data structures and algorithms for both exact and approximate nearest neighbor searching. Whenever we detect a time stamp in the past, we simply re-use the previous time stamp, which in real world datasets is a close neighbor as well. This does not detect closer "future" neighbors, but there are only a few cases where these bad choices were made.

For the median trajectories that follow the edges of the input trajectories, we determine the speed of the middle trajectory based on a single input trajectory for each (partial) edge. We cannot use a single edge with mean trajectories, since there is no edge that we follow directly. We take a mean approach instead. For each point on the middle trajectory, we determine the nearest neighbor in the forward direction for each of the input trajectories. The speed is then calculated based on the average of the previous points and the average of the current points. A more accurate approach would be to take both the direction of the segment and the distance to the calculated middle segment into account, but it is not straightforward what the correct speed would be in some instances, for example when segments go in the opposite direction.

CHAPTER 4

Experimental Evaluation

While all of the approaches we have mentioned in the previous chapters produce a trajectory, their results can be vastly different. In this chapter we present experimental results and evaluate these results by comparing measurements, visual inspections and reviewing expectations.

We have two main classes of algorithms that we use in all of the experiments. The first class is based on processing each data point at equal times. The second class is based on algorithms that were originally designed to compute a path without time stamps. The seven main algorithms we used to generate the middle trajectories are the following:

1. Equal time based:
   - average position
   - minimizing the maximal distance
   - projected medians
   - Weber point
   - centerpoint
2. Path based:
   - Iterative algorithm
   - Simple medians.[1]

We used two types of datasets: one with real world data, the other with synthetic data. The datasets we used contain a discrete sampling of both a position in at least two dimensions and the time corresponding of the measurement. Time stamps on the middle trajectory include all of the $\mathcal{O}(nm)$ found in the samples when equal time algorithms are used. Measurements of the

[1] Due to the fact that the majority of the tested input trajectories contains no significant self-intersections, the result of the homotopic median [9] and the majority median [21] would be nearly identical in most cases. Therefore, we chose not to implement these.

time stamps and of the speeds are done for both the nearest neighbor (NN) approach and the speed-based approach. In these experiments we used the median size as the initial number of points for the iterative algorithm.

For each dataset we will describe the results using the following measures:

- The number of points on the middle trajectory.
- The maximal, total, average and standard deviation of the length of a segment defined between two consecutive datapoints. These are measured in meters.
- The maximal, total, average and standard deviation of the angular changes at each point.
- The total duration of the middle trajectory, measured as the number of minutes between the first and the last point. Time stamps higher than the time stamp at the last point or lower than the time stamp at the first point are ignored.
- The maximal, total and standard deviation[2] of the speed of each segment, and the average of the speed over the whole trajectory measured in kilometers per hour.

We ran experiments on several types of datasets. Each dataset, except for the ones used in the first experiment, contains at least one problematic case for one of the algorithms to seek simple improvements and to evaluate the quality of the other algorithms. These datasets include trajectories with mostly temporal differences, interesting spatial properties (e.g. very sharp angles or many intersections) and inputs that exploit certain properties. We start with a simple case to get some sort of a control sample from a real world. These datasets contain mostly straight lines and the average speeds of the input trajectories are similar to one another. The input is not filtered and the algorithms are applied as they are described in Chapter 3. Therefore, seemingly obvious improvements to algorithms are not done in these experiments.

4.1 Pigeon data

The first types of datasets we use to evaluate the algorithms are based on pigeon flights from Freeman et al. [14]. There are twelve datasets in total and each dataset contains five trajectories. Each trajectory is the result of releasing a single pigeon at a specific site and tracking its flight back to the home loft. A dataset only contains trajectories for the same bird. All birds have the same home loft and are released at the same site.

Out of the twelve available datasets, we took averages of the seven most similar ones that were not problematic with the tested algorithms. These sets contain trajectories that are roughly straight paths from the release site to the home loft with times stamps that indicate the bird was continuously flying. An example of a dataset is shown in Figure 4.1. We discuss some of these remaining datasets later in the thesis. Two of these remaining datasets contain trajectories where the pigeons do not fly in a straight line to the home loft and another set contains two flights where the bird appears to have not moved for some time. In the remaining two sets there are segments in the middle that overlap with other segments, causing loops that are not easily fixed.

[2] Compared to the average speed over the whole trajectory.

All of the included trajectories follow roughly the same path. However, the speed in which the bird has traveled this path can be vastly different, ranging from about 50 km/h up to 76 km/h for an average speed. Sustained peeks in speed were seen both directions (i.e. speeds that were a lot lower or higher) for parts of the input. Since the trajectories cover only a small distance of approximately 11 kilometers, one trajectory can have finished at time t, while another trajectory is only about halfway to the end at that same time. Since the sampling rate is the same, roughly one measurement each second, slower trajectories have more datapoints than the faster trajectories. This difference in speed can be a problem for the equal time algorithms as similar spatial paths are compared on individual locations. Therefore, the shape of the resulting middle trajectory need not be the same as the shape of the input trajectories. However, since we know that the trajectories are relatively straight and we left out the extreme cases, the generated middle trajectories are still acceptable.

4.1.1 Results

We ran our algorithms on seven different datasets of a similar type. The results described in Table 4.1 show the averages of these data for all spatial measurements, while Table 4.2(a) and Table 4.2(b) show the results for the temporal aspects of these data, respectively based on the speeds on the segments and time stamps of the nearest neighbors.

Number of vertices

The datasets have an input size of 586 points on average. Due to the way we implemented the equal time algorithms (see Section 3.2.2), these all have a size equal to the longest trajectory in the input. The iterative approach determines the size beforehand, either as a parameter to the algorithm, or based on the input sizes.

The simple median is the only algorithm where the output size depends on the positions of the input points. Typically it has more points than the input trajectories [9], which is confirmed by our (mostly straight) input trajectories.

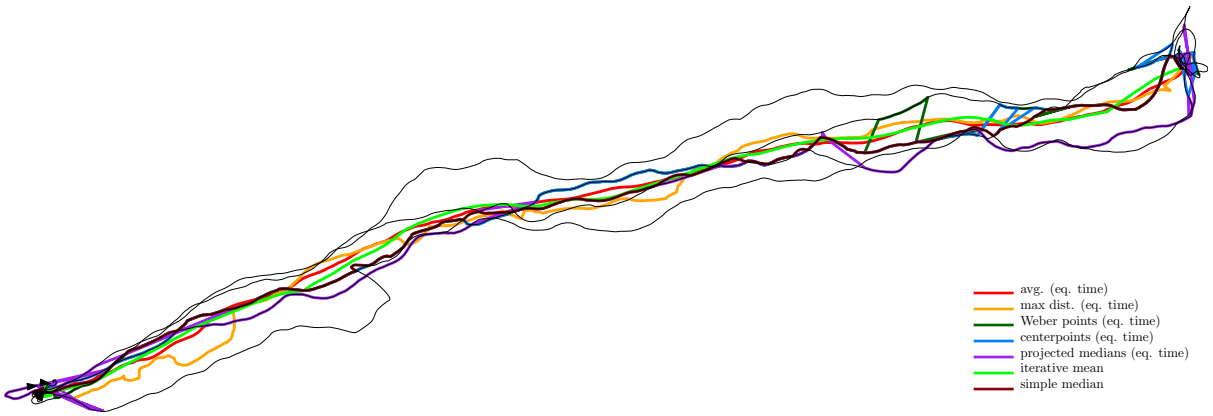


Figure 4.1: An example dataset where the bird flies in a relatively straight line to its home loft. The thinner black lines show the input trajectories.

type	n	length (m)				angle (rad)			
		max	total	μ	σ	max	total	μ	σ
input	518	80	10 132	20	6	1.3	60	0.1	0.1
Equal time									
avg.	661	24	9 111	14	6	1.0	49	0.1	0.1
max dist.	661	45	9 758	15	6	2.5	94	0.1	0.2
Weber points	661	742	13 643	21	42	3.1	107	0.2	0.4
centerpoints	661	1 088	22 553	34	101	3.1	154	0.2	0.6
projected med.	661	1 995	54 614	83	282	3.1	239	0.4	0.8
Path based									
iterative mean	509	205	9 023	18	31	0.8	22	0.0	0.1
simple median	547	36	9 810	18	6	2.0	91	0.2	0.2

Table 4.1: Average results for the spatial aspect of the 7 tested pigeon sets.

Length of the segments

A middle trajectory is expected to lie in the middle. Hence, for a dataset with mostly straight trajectories, the length of this middle trajectories should only be slightly shorter than the sample trajectories, but certainly not longer. The average total length is about 11 km for the input with most segments around 19 meters. The mean-based equal time algorithms and the simple median generate similar results, though with a slightly lower total length. This is what we expected, since a trajectory that lies in the middle of a set of relatively straight trajectories should have a comparable length in total. Moreover, having a similar length implies that the average length of a segment depends on the number of segments on the trajectory.

The iterative algorithm is similar with respect to the total length, although points seem to be pulled apart when using a distance threshold based on the total length of the input and checked against both neighbors. This causes both maximum segment length and the variation to be much higher, which is somewhat unexpected for an input with straight trajectories. Lastly, there are the equal time algorithms that are based on a median. These methods produce a trajectory that is (much) longer and has segment lengths that can be more than ten times longer than what is present in the input. This is caused by the jumping (explained in Section 2.2.2) and, as expected, this is worst with the projected medians. Even using Weber points results in a lot of jumping, indicated by the large standard deviation of the segment lengths.

Angular changes

The results shown in Table 4.1 are for the complete trajectories. For most statistics this is useful, but when analyzing angular changes in the pigeon dataset, this might not be the best approach as they tend to orientate themselves first and circle around the home loft at the end. However, since this is only a small part of the total trajectory, the effects are neglectable, which we verified by running the same tests after manually filtering the data. The results show that indeed for the sample trajectories the change in angle is very low both on average and for the value of the standard deviation.

type	Δt (min)	speed (km/h)			
		max	total	μ	σ
input	9.1	109	35 378	68	15
Equal time					
avg.	8.2	372	64 620	67	76
max dist.	8.6	368	64 264	68	74
Weber points	12.2	145	39 468	67	27
centerpoints	20.5	158	38 411	67	30
projected med.	50.4	181	34 990	65	33
Path based					
iterative mean	7.3	499	29 626	74	86
simple median	8.7	115	38 034	68	17

(a) Speed-based approach

type	Δt (min)	speed (km/h)			
		max	total	μ	σ
input	9.1	109	35 378	68	15
Equal time					
avg.	11.1	85	32 774	50	20
max dist.	11.1	162	35 067	53	22
Weber points	11.1	2 673	49 109	74	153
centerpoints	11.1	3 915	81 186	120	365
projected med.	11.1	7 182	196 599	297	1 016
Path based					
iterative mean	9.1	306	20 178	60	53
simple median	9.1	395	46 675	65	60

(b) Nearest neighbor approach

Table 4.2: Average results of the temporal aspect of the 7 tested pigeon sets.

By design, the iterative algorithm produces a trajectory with small angular changes. The simple median on the other hand can generate many angular changes, and possibly even very sharp angles. The results show this to be the case (a relatively high standard deviation with a slightly higher average angular change), although limited for this dataset.

The equal time algorithms show interesting results. Although the variant based on averages has results that can be expected for a middle path (i.e. comparable to the input, but with lower extremes), the other methods show results that indicate many angular changes. For the medians this can be explained by jumping from one trajectory to another one, sometimes in almost the opposite direction. For the variant based on the center of the smallest enclosing disk (SED) that minimizes maximal distances for each group of equal time, it indicates a lack of smoothness: Most of the points are somewhat on a straight line, but a large number of points can be considered outliers that are not on this path.

Time stamps and speeds

The last measurements on the assigned time stamps and the speed of the trajectories. In Section 2.4 we described two different methods to determine what time stamps to assign to the points on the calculated trajectory. Since in most cases the results are quite different, we consider the two methods as separate measurements. The results of the measurements are shown in Table 4.2. Note that while we calculated a range for each point in which the time stamps should fall, we did not include these in the tables.

In general, the duration found by the speed-based method should be lower than that found by the nearest neighbor approach. These expectations mostly correspond to the results we obtained. The only exception is that it is the other way around for the equal time based medians, which suits with the extreme segment lengths we have found in the spatial analysis of the trajectories.

A closer look at Table 4.2(a) reveals that the equal time algorithm based on the maximal distances assigns time stamps really close to the average of the time stamps on the input trajectories. For most datasets, for example like the one shown in Figure 4.1, the average speed we calculated for the equal time means is actually lower than that of the input trajectories, which is quite strange. The most likely explanation is that the lower average speeds are caused by the fact that we calculate positions for all available time stamps in the dataset, which means that the speed is close to zero when all but one trajectory have finished. This assumption is further supported by the fact that the iterative algorithm has a much higher average speed, which is calculated using the same method.

Using average segment speeds to calculate time stamps seems to give reasonable results on average. However, the measurements of the standard deviation and the maximum speed seem to indicate that the speeds are not stable. It is difficult to reason about these values without knowing on which segment(s) the maximum speeds are reached: A visual inspection did not reveal strange results. For instance, it might be possible that these segments are really short, but we did not do any further analysis to investigate this possibility.

Trajectory speeds that are based on the segments of a single input trajectory show more consistent results: a lower standard deviation and far more reasonable maximum speeds. Moreover, out of all algorithms we tested, the simple median shows results most similar to the input.

The NN based assignment of time stamps shown in Table 4.2(b) makes it easier to see that the centerpoints and the maximal distance minimization algorithms do not produce desirable results. The input shows that we can expect a maximum speed of about 110 km/h and fly with an average speed around 64 km/h. Anything that is significantly different from these is considered incorrect. On the other hand, our approach of not using true nearest neighbors, but instead use the time stamp for which the positions are calculated for the equal time based methods hides information about the main part of the trajectory. Nonetheless, most of the time stamps appear to be assigned as expected, except for the occasional outlier.

Since the tables do not show all information, we included additional plots to show how the speed progresses over time for both the speed-based and the nearest neighbor based methods. These plots are shown in Figures 4.2 and 4.3 for the nearest neighbors approach and the speed-based approach respectively. Such a plot makes it much easier to see that the projected medians algorithm does not generate useful trajectory at all. Moreover, the plot makes identifying jumps much easier. It should be noted that the two plots use a different scaling on the horizontal axis due to the longer duration of some middle trajectories in combination with the speed-based approach. Using the nearest neighbor approach, we implicitly set a bound on the maximal duration, even if the total length of the middle trajectory is much greater than that of the input trajectories. Therefore, the average speed of a middle trajectory can be much higher than what is found for the input when using the nearest neighbor approach.

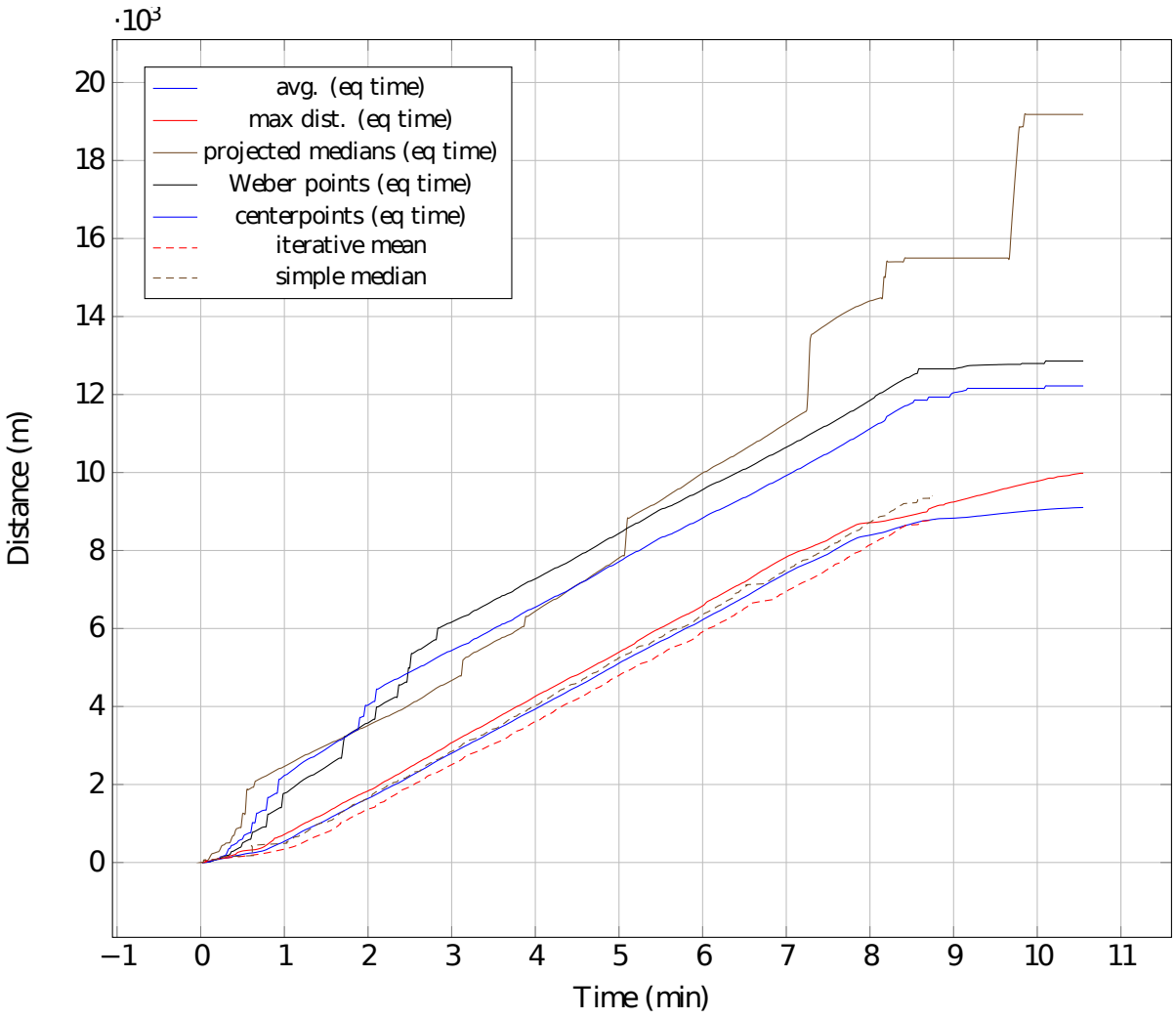


Figure 4.2: A visualization that shows a progression in time for the nearest neighbor method.

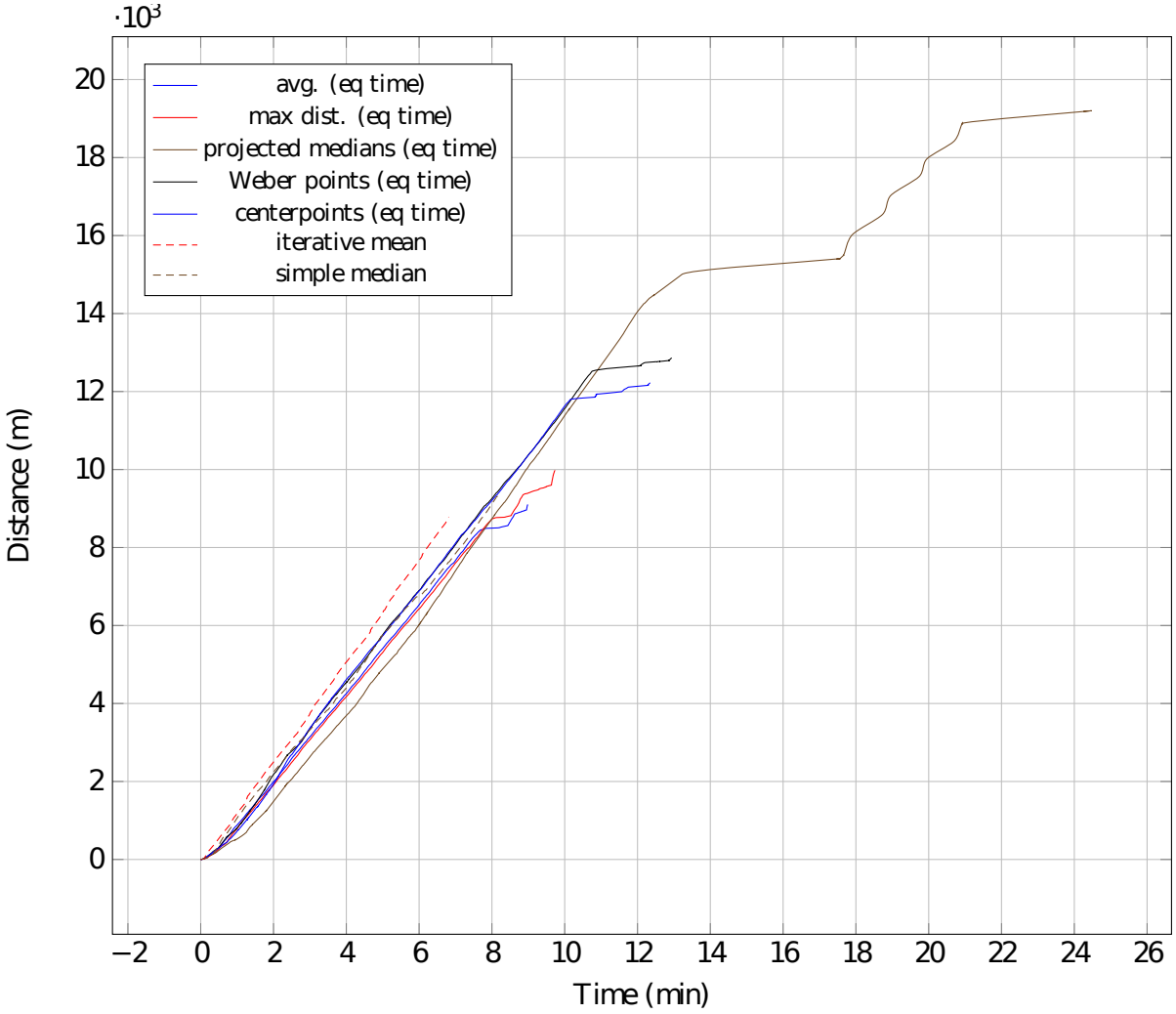


Figure 4.3: A visualization that shows a progression in time for the speed-based method.

4.1.2 Challenges

Although all methods seem to produce a trajectory that is visually in the middle for datasets with similar, nearly straight paths and different speeds, we have already seen some issues with respect to the segment lengths and maximum speeds. So far we tested seven datasets. The five remaining datasets contain more extreme examples of these jumps. One dataset contains a large detour and two sets gave problems with the simple median algorithm. In this section we discuss some of these problems.

Jumps

The algorithms that stand out the most in the earlier analysis are the equal time median algorithms. These suffer from a problem we call “jumping”: the next point in the sequence need not be close to the previous point, even though a closer point exists. An object following this trajectory would travel enormous distances in a short span of time (e.g. traveling around 9000

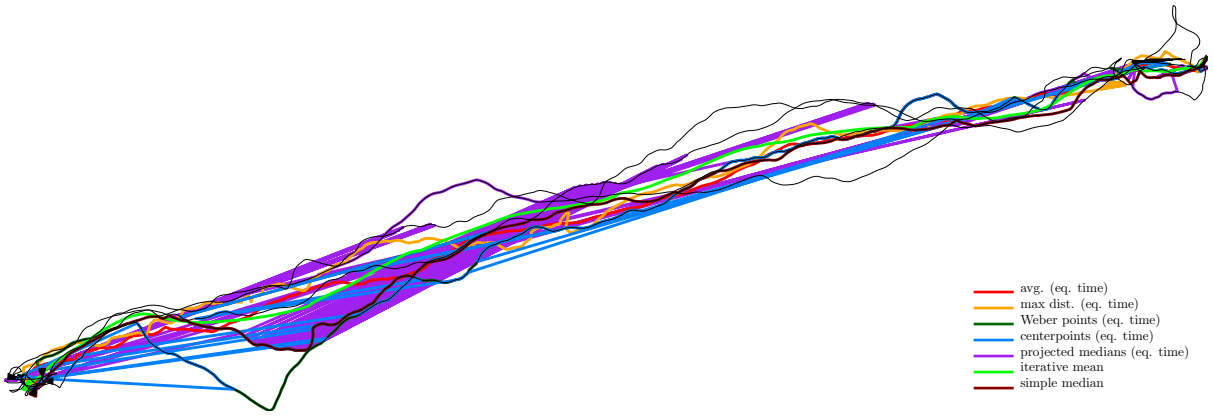


Figure 4.4: An example dataset where the equal time based medians produce undesired results where it is no longer possible to make sense of the data.

km/h such as the projected median with the nearest neighbor time stamp assignment approach). In some instances, consecutive points even alternate between two (or more) trajectories that do not intersect in the region. For example, the point at t_1 is on the first trajectory, t_2 is on the second, t_3 on the first, and so on.

The problem with these jumps is even worse when more than one of the trajectories in the set do not appear to go forward for several minutes. Figure 4.4 shows such an example where a computed middle trajectory – based on projected medians – is based on point sets with the points really far apart, resulting in unrealistic jumps.

Detours

In general, pigeons tend to fly in a relatively straight line from the release site to the home loft using an internal compass. This is exactly what the iterative algorithm by Freeman et al. assumes. However, pigeons also follow landmarks [5] (and cues provided by sound waves [20, 33]) and thus their paths are not always straight. The dataset in Figure 4.5 shows the flights for a bird that does not go directly to its home loft, which yields interesting results for the equal time based algorithms. The problem with these algorithms is that they compute their values based on points at equal time, which can be at vastly different locations if there is a large difference in the speeds. When trajectories do not follow the same path this is not a problem, since these trajectories have to be compared in some way and in these situations using their time stamps is the most obvious one. However, when the trajectories follow the same path, then one might expect that the middle trajectory does so as well. Therefore, the equal time approach does not seem to be fit for these kinds of datasets.

The simple median algorithm and pigeon data

Buchin et al. [9] use an arrangement to determine where to start and end the median trajectory: both must be adjacent to the outer face. Unfortunately, not all datasets meet this requirement. When pigeons are released, they often fly in a circle to orient themselves. Therefore, the first

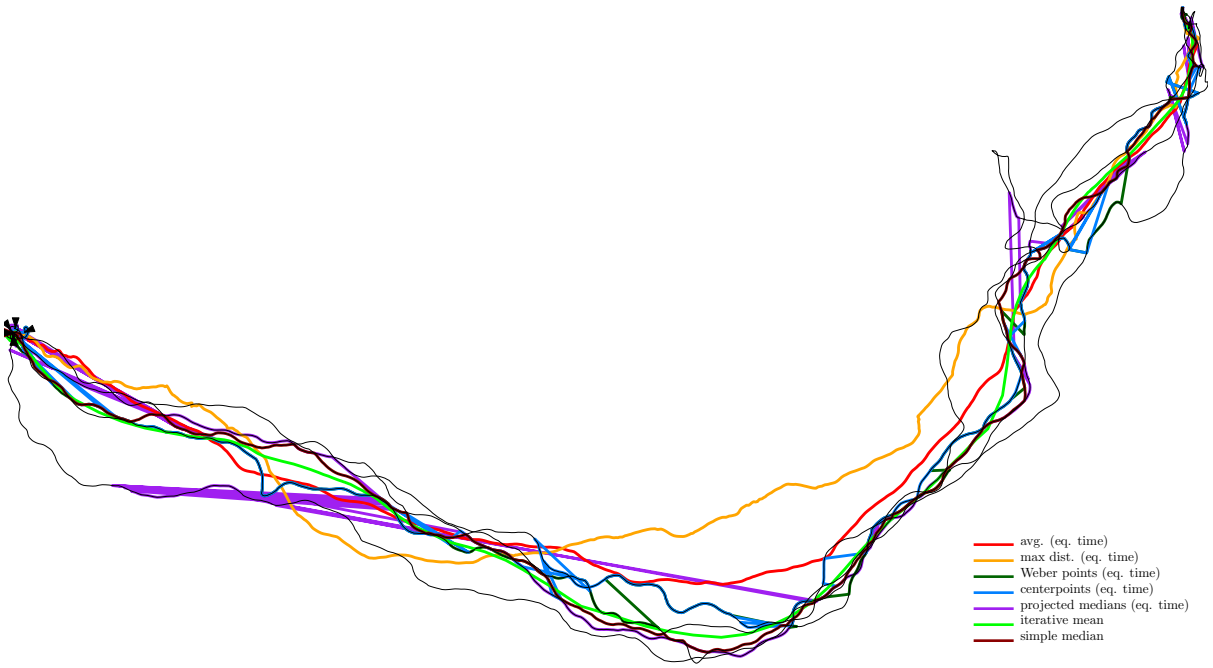


Figure 4.5: Pigeons do not always fly directly to their home loft. When these trajectories are flown at different speeds the equal time methods tend to make shortcuts.

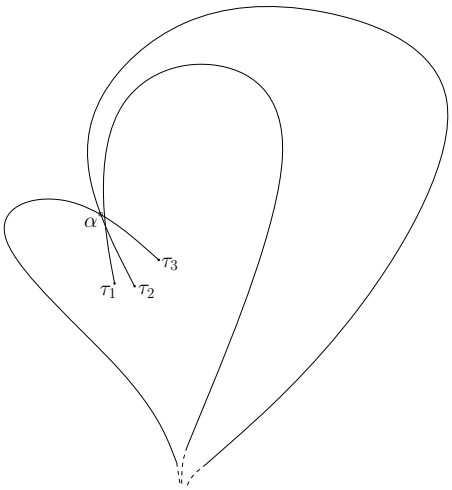


Figure 4.6: A set of trajectories not suited for the standard definition of a simple median.

points of these trajectories do not start adjacent to an the outer face of the arrangement as we can see in the simplified example in Figure 4.6.

For the three (partial) trajectories in Figure 4.6 it is obvious which one starts in the middle (i.e. τ_2). If we would ignore the requirement that the middle trajectory should start on an edge adjacent to the outer face, then we would stay in the middle until we reach intersection point α. After this point, we would be on the outermost trajectory and as a consequence, we will never switch back to the middle (except when the same thing happens at the endpoint). In most cases

this can be solved by either starting at some other point where it is easy to determine what the middle is or adding a point on the outside of the trajectories and start from there. For the latter solution, what remains is most likely a small region in which we have to find a middle path to this point. We made sure that all experiments used the true (lower) median trajectory before running the simple median algorithm on the datasets.

A real problem for the simple median algorithm occurred when we tried to run it on two of the remaining datasets. Both of these contain a segment that partially overlaps with another segment, causing it to switch indefinitely between several input trajectories. Since there was no easy fix for the problem, we decided to leave out these two sets altogether. We did apply pre-processing to retain only the main part of the trajectories, but none of the results we describe in this chapter use those data. Moreover, these sets did not reveal any new problems or other interesting results.

A note about the iterative algorithm

In Section 3.2.4 we mentioned the possibility to stop the algorithm as soon as no more changes are detected. Experiments show however there will most likely always be changes in two consecutive iterations as a consequence of using nearest neighbors. This can be seen in Figure 4.7. Interestingly, a closer look at the figure reveals that not the next iteration (iteration $i+1$), but the one that follows (iteration $i+2$) is most similar. We arbitrarily picked the iterations 30, 31 and 32, but any triplet after approximately the 20th iteration would do.

We now know that the most similar results do not always occur in the following iteration. It would therefore be possible to stop the computation as soon as we detect that changes are below a certain bound when we compare the resulting trajectory of the current iteration to the trajectory from two iterations ago. Doing so would require keeping track of only $\mathcal{O}(nm)$ extra points[3] in this instance, but we do not always know in advance in what pattern these similar results occur in general.

4.1.3 A more natural result

The three equal time based median algorithms suffer from what we call jumping as can be seen in Section 4.1.1. These algorithms are not really usable in the shape we have described them so far, which is especially true for the projected medians algorithm. In an attempt to improve the algorithm we can exploit a property of a trajectory, namely that we can extract the maximum speed. (With low sampling rates it might be better to require the maximum speed as a parameter for the algorithm.) Under the assumption that the maximum speed of an object is approximated by the maximum speed that was reached in the input trajectories, we can use this property to set a bound on what points can be considered reachable[4]. In other words, we can filter out all points that cannot be reached from the previous point on the middle trajectory without breaking the speed 'limit'. Using such a bound, we cannot prevent alternating jumps from one trajectory to another (illustrated in Figure 4.9), but the extremely long jumps can be prevented. The results are depicted in Figure 4.8, where the projected medians are compared

[3] More precisely, it depends on the output complexity. In our experiments this is always linear.

[4] For some groups, none of the points were within the range of valid points using our simple algorithm. In these cases, we simply took our original approach. Alternatives are to choose the nearest point in such cases, or to apply backtracking to prevent these scenarios altogether.

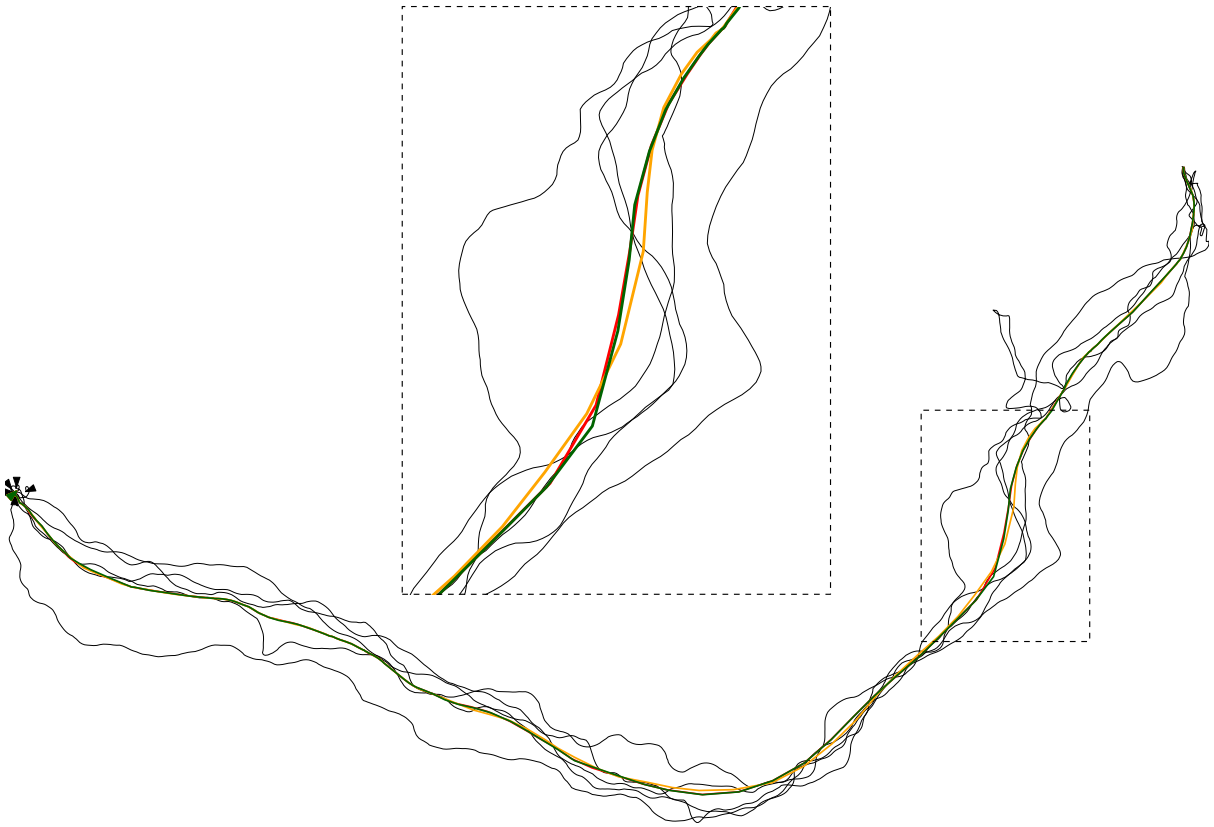


Figure 4.7: Small changes exist between iterations i and $i+2$ for $i \gtrapprox 20$. The arbitrarily chosen iterations 30, 31 and 32 are shown in respectively red, orange and green.

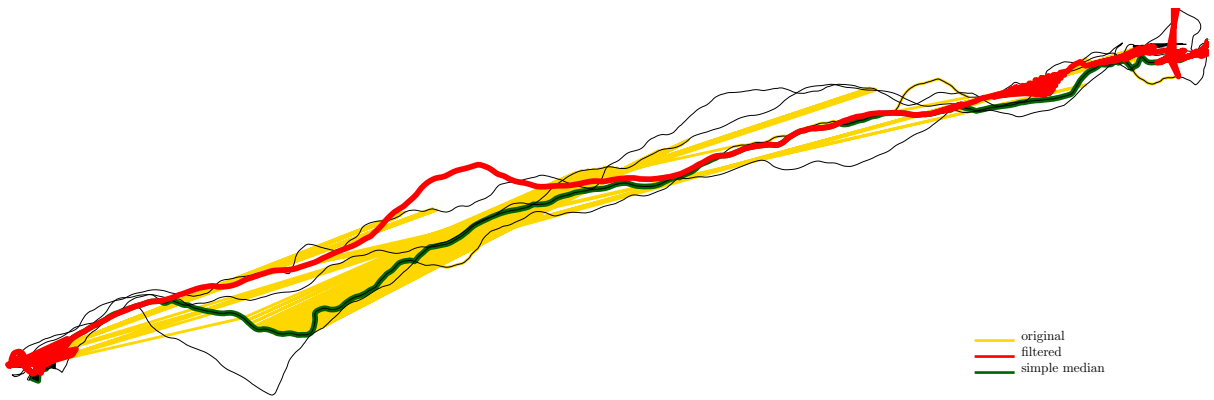


Figure 4.8: The same dataset as before, but using a bound to filter invalid points.

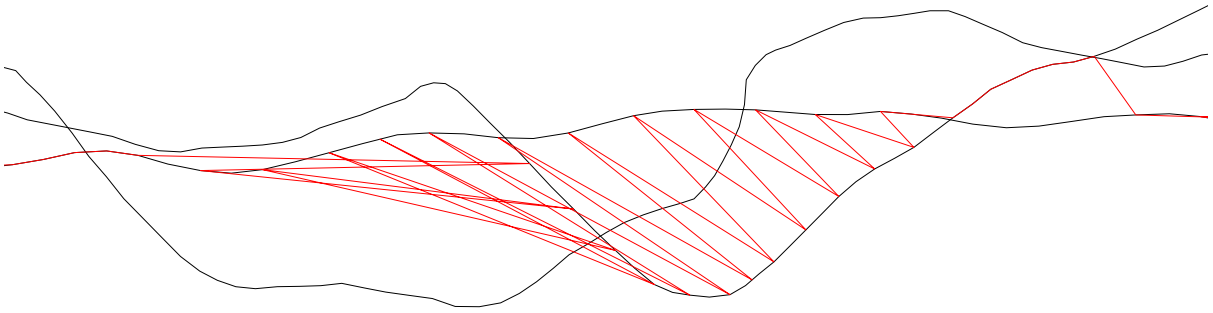


Figure 4.9: With a bound on the speed it is still possible to have a median trajectory that alternates between two or more input trajectories.

type	Δt (min)	speed (km/h)			
		max	total	μ	σ
input	32.1	95	37 572	20	27
input	8.6	105	35 723	71	23
input	14.4	1 338	42 044	51	93
input	9.4	140	33 163	60	16
input	25.3	80	37 710	25	26
original	723.0	168	35 779	23	30
filtered	96.3	136	36 496	20	32
eq. time avg.	26.0	475	117 171	22	98

(a) Speed-based approach

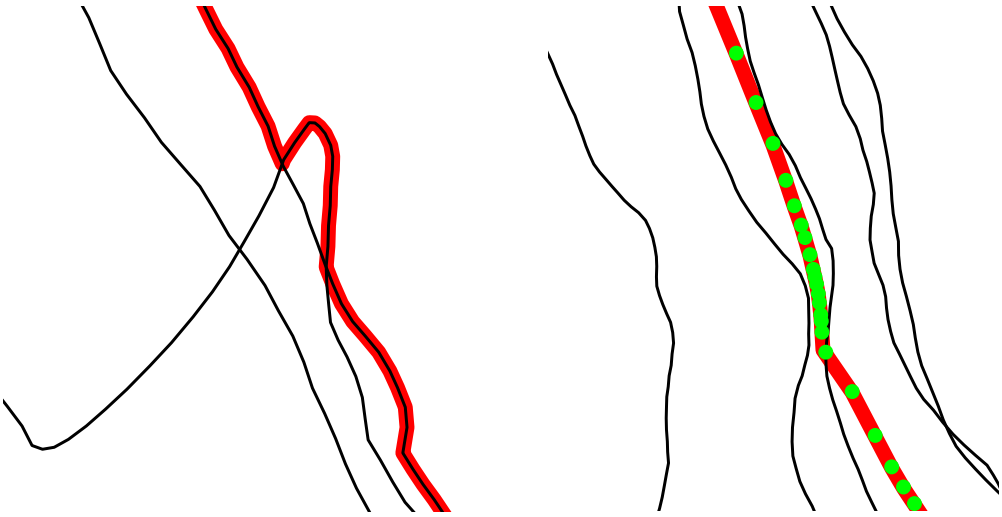
type	Δt (min)	speed (km/h)			
		max	total	μ	σ
input	32.1	95	37 572	20	27
input	8.6	105	35 723	71	23
input	14.4	1 338	42 044	51	93
input	9.4	140	33 163	60	16
input	25.3	80	37 710	25	26
original	32.1	22 270	982 476	510	2 345
filtered	32.1	1 454	114 758	60	182
eq. time avg.	32.1	305	34 280	18	21

(b) Nearest neighbor approach

Table 4.3: Results for the dataset that causes extreme jumps with the unfiltered projected medians algorithm.

before and after applying the filter. Simple medians are used as a reference to see what the actual median is.

We use the dataset shown in Figure 4.4 to compare the original projected medians algorithm before and after applying the filter. The results are shown in Table 4.3. Despite being a huge improvement, the results still are not as good as we would like to see. Unfortunately, the localized jumping problem is much harder to solve: Let h_{ij} be the (largest) number of consecutive points that are not included in the middle trajectory directly after a point τ_{ij} for some trajectory i and point j. In this example h_{ij} equals 1 and a fix would therefore involve checking the next two points. A more general fix would be to set a lower bound β and consider all sequences with a length smaller than β to be bad jumps. However, jumping can also occur between multiple trajectories (with $h_{ij} \leq \beta$) and we might only consider a sequence as being bad when the jump leaves in a direction similar as where it came from, which need not be a single trajectory. Further research should be done to find a solution that works in all (practical) cases.



(a) Unnatural switching (b) Clustering

Figure 4.10: Results are not always realistic. Some of the challenges to overcome is to skip certain switches or to remove certain points.

The simple median is proven to stay always in the middle. This implies that the input trajectory that is in the middle has to be followed, even if this requires an unnatural action such as a sharp turn. An example can be seen in Figure 4.10(a), where it might be better to have a less strict requirement so that the switch does not have to be made.

4.1.4 Evaluation

A good middle trajectory should represent the length, shape and speed of the input trajectories. Moreover, for a middle trajectory to be realistic it should not have many angular changes on straight stretches. The length of the middle trajectory should not be longer than that of the majority of the input trajectories and the speed should be similar. The shape is harder to measure, since it should not follow detours made by only one trajectory (or a small subset of the trajectories). Intuitively the trajectory should follow the input trajectories and stay in the middle. Based on these criteria, there is no clear algorithm that gives the best results on all of these aspects. On the other hand, there are algorithms that clearly perform very badly.

The biggest flaw with the iterative approach seems to be the clustering of the points as visible in Figure 4.10(b), although this could be fixed in a post-processing step. These clusters of points appear throughout the plane, while at other places there is a lot of space between two consecutive points. This uneven spread results in high maximal segment lengths and thus decreases the precision of the middle trajectory, because it is not recorded what happens in between. Moreover, the number of points is based on the input and is not influenced by the choices made during the construction of the output trajectory. Choosing these numbers beforehand is not always the best approach, for example when we require a significant reduction of the number of points if the output is a set of line segments that resemble a straight line. A similar post-processing step can be used to overcome this behavior as the one required to replace the clusters by only a few points.

The equal time based approach using averages seems to work with the first datasets, but fails when the difference in speed for individual input trajectories is too great. In fact, all equal time based methods suffer from this problem. Even when the speeds are similar, equal time based medians can suffer from a problem we called jumping. The only algorithm that performed well on the pigeon datasets on both the length and speed is the simple median algorithm. Unfortunately, the implementation can be tricky and a solution has to be found to deal with overlapping segments, i.e. to make sure the calculation finishes. Furthermore, a filter should be applied afterwards to simplify the shape in order to prevent high angular changes.

When it comes to the assignment of the time stamps, both methods seem to work reasonably well, even though our implementation of the nearest neighbor approach can be improved (see Chapter 2). The main problem with both approaches is that both the maximum speed and the standard deviation that are found are often much higher than what is realistic, with the exception of the speed-based methods that directly follow an input segment.

4.2 Synthetic data

Besides looking at real world data where either the movement paths or the time stamps of the individual trajectories are (completely) different, it is also interesting to see what happens when the input is similar. We constructed several datasets such that they are interesting to more than one algorithm by using similar speeds, but at the same time sharp angles. Moreover, similar trajectories could be the result of real world data, such as climbing a steep hill or sigh seeing.

In this section we look at the result of running the algorithms on five datasets constructed using the trajectory generator described by Wiratma [32]. Multiple datasets are generated using the same waypoints for each experiment[5]. Each of these generated sets contains eleven similar trajectories, made up of a similar number of segment lengths with a comparable length. Since this generator does not add time stamps, we chose to use the index as the time stamp (i.e. index i is the location after i minutes). This implies that the unit of the speed does not have a real meaning, but it allows us to compare the numbers and spot potential bad behavior of the algorithms when these values are completely different from the input.

4.2.1 Zigzagging

The first group of datasets we look at contains zigzagging trajectories. An example of such dataset is shown in Figure 4.11. The trajectories have similar speeds and positions. Hence, an optimal middle trajectory has similar properties as the input, but possibly fewer datapoints (only two are needed for a straight line) and thus longer individual segment lengths. Moreover, unlike with the real world data we have used, the equal time based methods stay within the bounds of the outer trajectories with these datasets. The measurements for this dataset are shown in Table 4.4 for spatial measurements, and Table 4.5 for the temporal measurements.

[5]It takes several hours to generate a single dataset with $n < 1000$ points for each trajectory, hence these datasets are relatively small.

type	n	Δt (min)	length (m)				angle (rad)				speed (km/h)	
			max	total	μ	σ	max	total	μ	σ	max	μ
input	35	35.2	57	1 675	48	7	2.5	15	0.4	0.6	3	3
Equal time												
avg.	38	37.0	50	1 557	42	9	2.2	10	0.3	0.5	3	3
max dist.	38	37.0	57	1 573	43	9	2.3	14	0.4	0.5	3	3
Weber points	38	37.0	71	1 616	44	13	2.4	19	0.5	0.5	4	3
centerpoints	38	37.0	74	1 618	44	14	2.4	18	0.5	0.5	4	3
projected med.	38	37.0	141	1 827	49	34	3.1	35	1.0	1.0	8	3
Path based												
iterative mean	36	35.2	69	1 093	31	11	0.8	7	0.2	0.2	8	2
simple median	159	34.8	52	1 685	11	11	2.8	115	0.7	0.7	11	3

Table 4.4: Experimental results for the zigzagging type of generated data.

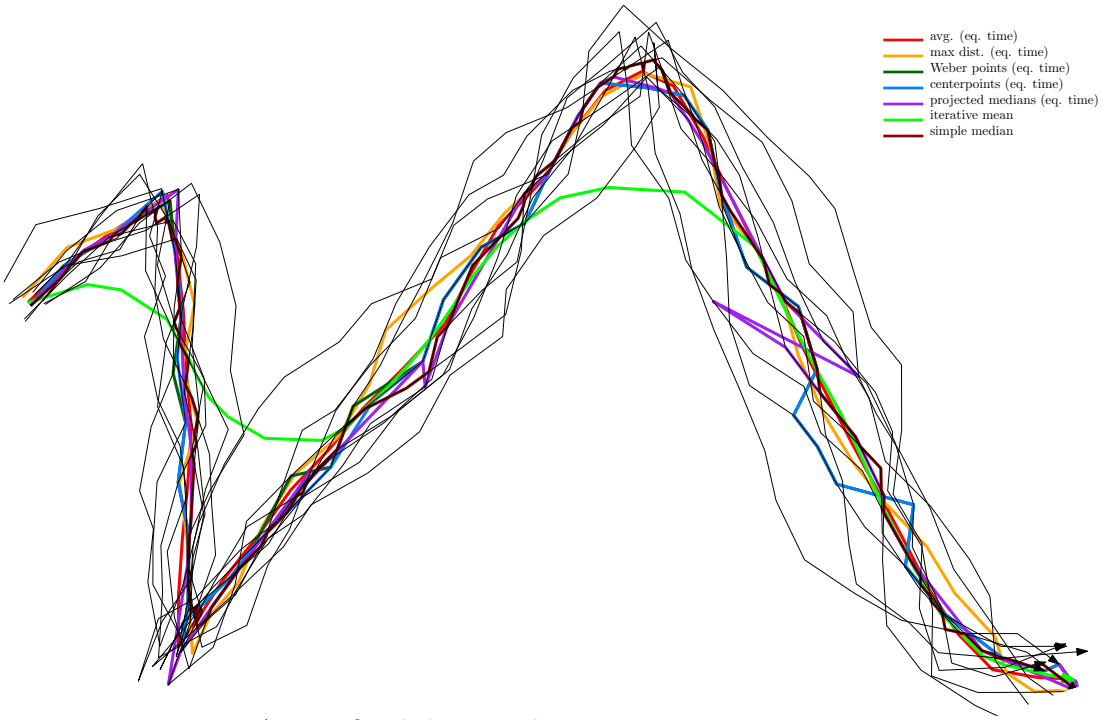


Figure 4.11: An artificial dataset that contains zigzagging trajectories.

A visual inspection of Figure 4.11 shows that all trajectories stay in the middle, except for the iterative algorithm. Similar speeds cause the equal time based methods to generate more "natural" results (points at equal time are confined to a small region). The iterative algorithm, which starts with a straight line between the endpoints, is pulled towards the locations with the most points, namely the peaks. Lastly, the endpoints are adjacent to the outer faces of the arrangement, so we can use the original definition the simple median.

Number of vertices

Although containing much fewer points than the previous sets, it is clearly visible that the simple median generates much more points than the other algorithms. For this dataset, it generates more than four times as many datapoints as the input. We expect to see this for all datasets generated by the generator, as it is designed to produce trajectories that contain many intersections with one another. Moreover, since the data is generated to have a difference of at most 5 points in the size, the sizes of the other trajectories are similar.

Length of the segments

We expect the generated middle trajectory to be at most as long as the input. But this is not what we see with two of the methods, namely the projected median and the simple median. We already know that the projected medians are sensitive to outliers. Even for this dataset where the distances between points at the same time is small due to the similar speeds, the effects are clearly noticeably. A closer inspection reveals that this is caused by switching from the foremost point to the rearmost point at some time stamps. In the dataset shown in Figure 4.11 this happens just after the second turn. The fact that the simple median generates a longer trajectory (albeit slightly) does come as a surprise. Despite having generated trajectories that consist of mostly straight lines between the peaks, taking the average of an input with mostly longer trajectories should give rise to a shorter middle trajectory. It seems the input does not contain many crossings in a single direction, nor do individual trajectories zigzag much between peaks.

A consequence of the amount of points generated by the simple median, together with the total path length, is that the average segment lengths decrease. The standard deviation however does not seem to be affected by much, making it equal to the average value after rounding. Regions that do not contain intersections do not generate additional points and therefore have segments lengths that are equal to the trajectory that is followed at that time, so there can still be segments with similar maximal length as what appears in the input.

The only approach that produces a significantly shorter trajectory is the iterative one. It is also the only of the middle trajectories that really stands out in Figure 4.11 due to the shortcuts it takes. Being based on an even distribution of points, the variance is relatively high. One of the reasons for this is that the initial path does go not near the peaks and thus has long segments at these areas. We would expect the variance to increase when these peaks would be higher.

Angular changes

Another side-effect of having many segments is that all of them attribute to the total angular change count. Therefore, like the size, this is highest for the simple median. In general, for an input with sharp angles we naturally expect the maximal angle to be high. Moreover, since the other parts are mostly straight, both the average and standard deviation of the angular changes are expected to be low. This is what we see for both the input and the results, except for projected medians that still shows many abrupt angular changes.

With these datasets, all of the input trajectories contain angles of about 2 radians, so a middle trajectory should do the same. However, all of the measurements for the iterative approach related to the angular changes are much smaller than the others, which is another indication

type	Δt (min)	speed (km/h)			
		max	total	μ	σ
input	35.2	3	100	3	0
Equal time					
avg.	32.8	19	127	3	3
max dist.	32.9	19	127	3	3
Weber points	33.9	6	109	3	1
centerpoints	34.1	5	107	3	1
projected med.	39.2	6	107	3	1
Path based					
iterative mean	15.9	21	195	4	4
simple median	34.9	3	464	3	0

(a) Speed-based approach

type	Δt (min)	speed (km/h)			
		max	total	μ	σ
input	35.2	3	100	3	0
Equal time					
avg.	37.0	3	93	3	1
max dist.	37.0	3	94	3	1
Weber points	37.0	4	97	3	1
centerpoints	37.0	4	97	3	1
projected med.	37.0	8	110	3	2
Path based					
iterative mean	35.2	8	98	2	2
simple median	34.8	11	278	3	2

(b) Nearest neighbor approach

Table 4.5: Average results of the temporal aspect of the generated zigzagging dataset.

that the result of the iterative approach is taking shortcuts. For some application this might not be a problem, but we want the middle trajectory to be similar to the input and hence it should include these sharp angles.

Speeds

The temporal aspect of the experiments is shown in Table 4.5. The measurements yield similar results to the pigeon datasets for the maximum speed with the speed-based method; These are much higher for the algorithms where we could not directly use one of the input segments. However, since the speeds of the input trajectories were similar this time, we do not see the extreme results like we did before with the nearest neighbor approach. Moreover, the highest standard deviation for the iterative mean algorithm is still very low, which is confirmed by the near-constant speeds for all calculated trajectories.

A hybrid iterative approach

The results that stand out in Table 4.4 are the ones produced by the iterative algorithm. Clearly, using just the endpoints for the initial thread only works if the trajectories are almost straight. The difference in lengths increases when trajectories are closer together, which can be seen in Figure 4.12. One option to prevent the undesired behavior of skipping whole regions is to force the initial thread to go through the regions containing these peaks. A simpler, although less enforcing option is to use the result of one of the other algorithms as the initial thread and continue from there. This produces the results shown in Figure 4.13 and Table 4.6 when we use the averages based equal time algorithm as the initial thread.

The total length did not increase by much. Despite the fact that the result is a mean, we would expect it to be closer to the algorithm that was used for the first iteration. This seems to be caused by the shortcuts that still happen near the peaks (one of which is visible at the bottom peak in Figure 4.13). Again, this results in high speeds compared to the results of the other algorithms.

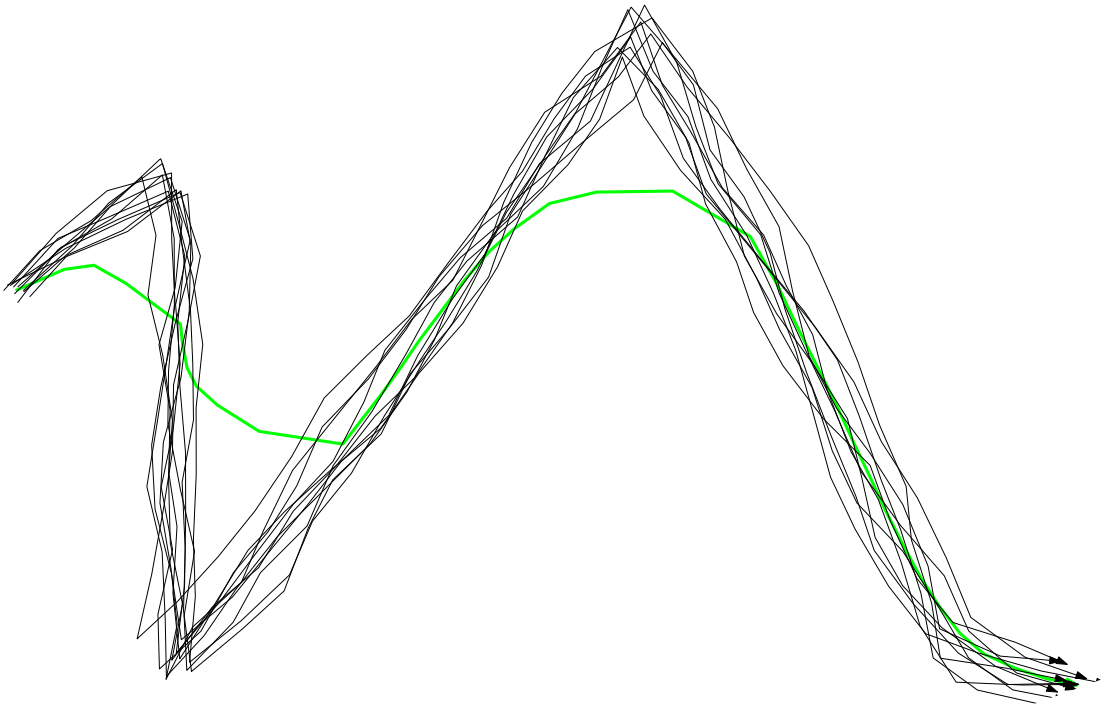


Figure 4.12: When the trajectories are closer to one another, the length of middle trajectory computed by the iterative algorithm differs even more compared to the other algorithms.

type	n	Δt (min)	length (m)				angle (rad)				speed (km/h)	
			max	total	μ	σ	max	total	μ	σ	max	μ
input	35	35.2	57	1 675	48	7	2.5	15	0.4	0.6	3	3
hybrid iterative	38	35.2	53	1 184	32	8	1.1	8	0.2	0.3	5	2
iterative mean	36	35.2	69	1 093	31	11	0.8	7	0.2	0.2	8	2

Table 4.6: Results for the hybrid iterative algorithm on a similar dataset using nearest neighbors to assign time stamps.

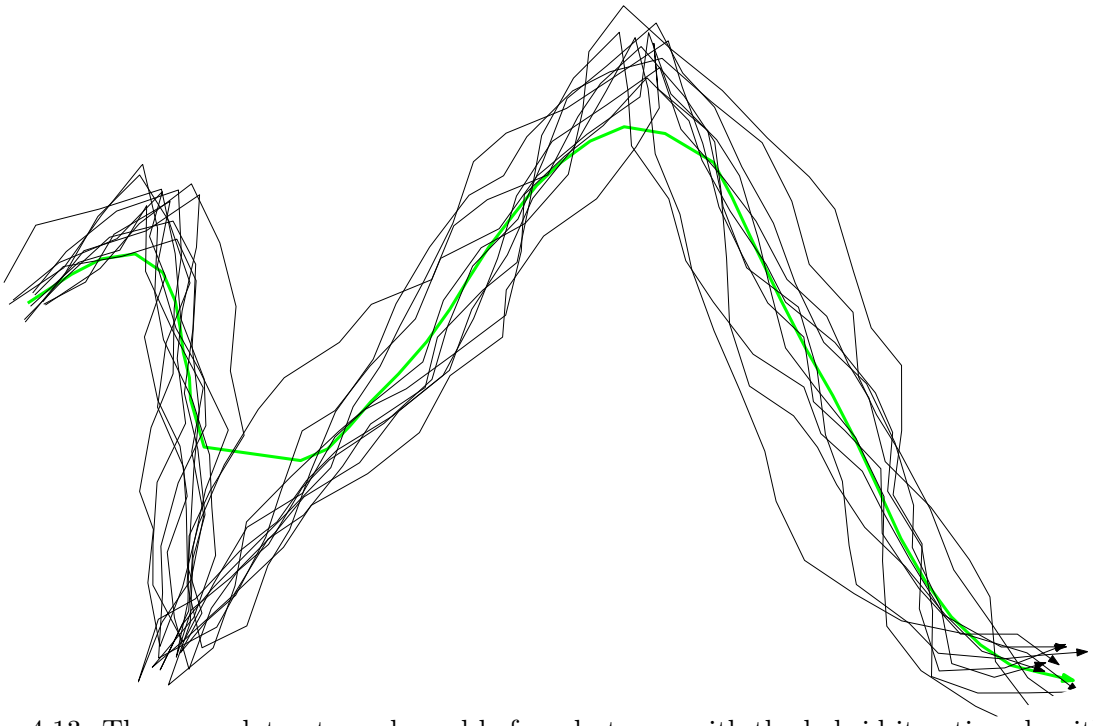


Figure 4.13: The same dataset we showed before, but now with the hybrid iterative algorithm.

type	n	Δt (min)	length (m)				angle (rad)				speed (km/h)	
			max	total	μ	σ	max	total	μ	σ	max	μ
input	47	46.6	50	2 204	47	3	2.8	17	0.4	0.6	3	3
hybrid iterative	49	46.6	51	1 558	32	6	0.7	9	0.2	0.2	5	2
iterative mean	48	46.6	107	1 193	25	18	1.0	6	0.1	0.2	16	2

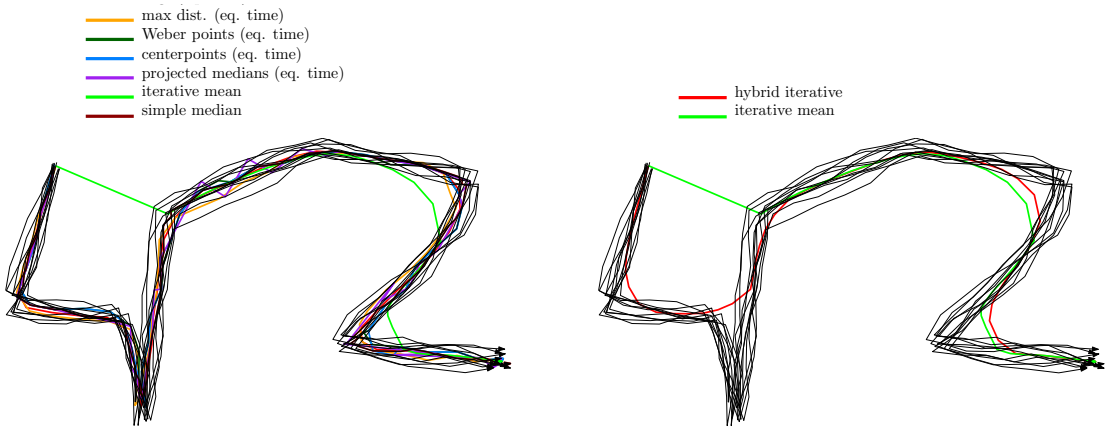
Table 4.7: Results for the iterative algorithm, both with and without the hybrid approach.

Based on these findings, we expect to get a more extreme example if the trajectories are closer together. Figure 4.14 shows the results for the algorithms with the results of the original algorithms shown in Figure 4.14(a). The modified iterative algorithm is shown in Figure 4.14(b). We can see that even after this modification, complete regions are skipped.

As we can see in the results described in Table 4.7, the length increased by more than 30% compared to when a straight thread is used. Moreover, there is a significantly lower variance in the segment lengths and has less extreme maximum and minimum lengths. However, compared to the input, the length still about 30% less. This difference is again caused by shortcuts.

4.2.2 Evaluation

In contrast to the pigeon data the equal time based medians are comparable to the input in all aspects. In fact, the only approach that creates a significantly different (in this case shorter)



(a) Original algorithm (b) Hybrid algorithm

Figure 4.14: The iterative algorithm uses shortcuts. All other algorithms follow the input.

type	cpu time (s)
Equal time	
avg.	0.0055
max dist.	0.0086
Weber points	0.0112
centerpoints	0.0287
projected med.	0.0099
filtered projected med.	0.0105
Path based	
iterative mean	0.4325
hybrid iterative mean	0.5405
iterative mean (k = 20)	0.0934
simple median	0.2916

(a) Speed-based approach

type	cpu time (s)
Equal time	
avg.	0.0049
max dist.	0.0079
Weber points	0.0108
centerpoints	0.0275
projected med.	0.0094
filtered projected med.	0.0098
Path based	
iterative mean	0.4291
hybrid iterative mean	0.5514
iterative mean (k = 20)	0.0928
simple median	0.3007

(b) Nearest neighbor approach

Table 4.8: Running times for the algorithms.

middle trajectory is the iterative algorithm. This difference was to be expected, since we chose to generate the datasets with only a small difference in both the number of points and the distance between these points at equal time stamps.

4.3 Performance

Not only should the algorithms find a middle trajectory that is a good representative of the input, it should also find this in a reasonable amount of time. In Chapter 2 we mentioned the asymptotic running times of each algorithm. To see how these relate to one another in practice, we now look at the running time for one of the pigeon sets, namely the one that was also discussed in Section 4.1.1. These results are shown in Table 4.8.

All measurements are the result of running all algorithms seven times and taking the average of the five middle results. It should not come as a surprise that fastest algorithm is the equal time algorithm that uses averages. In fact, all equal time based algorithms are much faster than the path based algorithms. Therefore, it can also be expected that the slowest algorithm is the hybrid iterative mean, which essentially is the simple median followed by equal time algorithm and increasing the time to process the already slowest individual algorithm. It should however be noted that the calculation of the speed-based time stamps was done during the construction of the middle trajectory. If we would have done this calculation afterwards, we would have seen a significant performance impact when the original segments need to be found.

It is interesting to see how much of a difference there is between the equal time based algorithms and the path based algorithms, even for a small dataset containing five trajectories with approximately 500 points. Not having to search for nearest neighbors significantly reduces the time required to find a trajectory. Unless the algorithms are designed to compute time stamps, we see no obvious way of avoiding these NN searches when not using the speed-based approach.

4.4 Evaluation

We ran experiments on several datasets to measure the quality of the various algorithms and analyzed the results. Depending on the data and the desired results, each can have a different preferable method of finding a middle path. When the goal is to find a mean path, then the equal time based methods give the best results if either the speeds on the input trajectories are comparable like in Section 4.2.1. The iterative algorithm is preferred in the remaining cases, provided some post-processing will be performed to reduce the clustering, as indicated in Section 4.1.1. Moreover, it can be unclear at what position the trajectory is (or should be) at a given time due to potential non-ascending or even descending time stamps when approximating the time stamps.

On the other hand, when we want to find a middle trajectory based on the input points, the best results are found with the simple median algorithm and related algorithms such as the homotopic median. Still, a solution needs to be found for both the problem of endless loops mentioned in Section 4.1 and the case when one of the endpoints is not adjacent to the outer face of the arrangement of the input trajectories to be usable for all datasets. A major disadvantage of the approach however is that the resulting middle trajectory is constructed from individual segments. This segmentation can result in rapid changes in velocity and many angular changes, however, it might be possible to smoothen the result in a post-processing step.

Most problems can be avoided by (manually) pre-processing the input trajectories. Moreover, strange loops such as the one just above the zoomed rectangle in Figure 4.7 can be removed to reduce their effects on the middle trajectory. However, one has to consider whether 1. pre-processing has to be done at all in order to say something meaningful about the input (e.g. when the middle trajectory is used to analyze the behavior of the animal), and 2. whether or not these adaptations should be incorporated in the algorithm, although this makes these algorithms more specific to those scenarios.

CHAPTER 5

Conclusion and Future Work

The outset of this thesis was to evaluate the performance of known algorithms that generate middle trajectories and compare the quality of the results in both their spatial and temporal aspects. The latter, as turned out, is often not part of the result. Therefore, part of this thesis focuses on two methods of assigning time stamps as a post-processing step on the output of the algorithms.

As a first step of the evaluation twelve properties were defined that should be satisfied by either the algorithm or the result of that algorithm. These properties, most of which are based on existing works, assure similarity of the shapes and attributes, running time, and for a trajectory to be in the middle if satisfied. The results of this theoretical evaluation did not produce satisfactory results, as counter-examples could be found that violate many properties for most algorithms. For the experimental evaluation we ended up with three main algorithms in total, and several smaller algorithms based on these three main algorithms such as some hybrids and using different methods to compute an equal time algorithm. Modifications to existing algorithms also included the use of different – in our implementation – parameters and the exclusion of intersections in case of the simple median algorithm. This experimental evaluation yielded interesting results. While some approaches clearly do not work for some real world datasets, they might perform reasonably well when the input is composed of trajectories that are similar in both their spatial positions and speed. The conclusion is that no single algorithm is a clear winner in all scenarios, and therefore an evaluation is required for each application.

Assigning time stamps can be difficult. The (most) correct assignment depends on a number of factors, including the goal (e.g. whether one wants to know likely speeds or whether the time of arrival is important), the complexity of the input and the method that is used to calculate the path: one cannot directly follow segments using a mean-based algorithm. For example, when using a nearest neighbor based approach to assign time stamps, one has to be careful about decreasing values when. We have examined the results of both a speed-based method and a method using nearest neighboring points. The results indicate that both approaches assign time stamps that look “natural” for the goal they were designed, although some fine-tuning might be required to get realistic maximum speeds.

In this thesis we tested three different algorithms. A fourth option is to implement an algorithm based on finding a middle trajectory that minimizes the Fréchet distance to all other trajectories, much like the work by Alt and Godau [2]. Buchin, Buchin, and Gudmundsson [6] suggested a constrained variant of the Fréchet distance that can be used to restrict the valid regions to be within a restricted time interval. This restricted variant can not only be useful for finding a middle trajectory with a temporal component, but also to improve the assignment of time stamps in a post-processing step as explained in Section 3.3. Other algorithms we did not include in the experiments are the homotopic median [9] and the majority median [21]. By implementing these, a fairer comparison can be made for trajectories that contain significant loops. Moreover, having an algorithm (i.e. the simple median) that sometimes fails on real world data is not acceptable.

Mean trajectories as defined in this thesis are sensitive to outliers. More sensible approaches are to use a *weighted mean* to reduce the impact of the outliers, or to filter the input data to exclude these outlying trajectories. The former can be easily implemented and requires only small modifications to the algorithms.

All of the experiments were performed on two-dimensional spatial data, despite the fact that there is no dimensional restriction for most algorithms and many datasets contain another spatial dimension, namely the altitude. It would be interesting to extend our experiments to include the altitude. The leader-follower behavior in pigeons can for example be extended to answer the question if the altitude flown by the birds is influenced when the pigeons fly in pairs. Even though we tried to test as many of the "bad" cases as we could that are more or less realistic, there almost certainly are more cases in which some of the algorithms fail. What remains to be done is therefore to test more and larger datasets. For the latter one might have to develop a trajectory generator similar to the one described by Wiratma [32] that allows for the generation of large datasets in a reasonable amount of time.

Despite all the things that can still be done, we think that knowing what aspects to consider when either designing or using an algorithm to compute a middle trajectory can be very helpful. We showed that even simple algorithms based on equal times have decent results under some conditions, while being much faster to compute. Moreover, adding time stamps can be more difficult than one might think, but it is not impossible. We hope this work makes people think twice before ignoring the temporal aspect of trajectories.

Bibliography

[1] P. Agarwal, M. de Berg, J. Gao, L. Guibas, and S. Har-Peled. "Staying in the middle: Exact and approximate medians in $\mathbb{R}^1$ and $\mathbb{R}^2$ for moving points". In *Proceedings of the Canadian Conference on Computational Geometry*. 2003, pp. 42–45. DOI: 10.1.1.114.8625.

[2] H. Alt and M. Godau. "Computing the Fréchet distance between two polygonal curves". In *International Journal of Computational Geometry & Applications* 5.01n02 (1995), pp. 75–91. DOI: 10.1142/S0218195995000064.

[3] M. Andersson, J. Gudmundsson, P. Laube, and T. Wolle. "Reporting leadership patterns among trajectories". In *Proceedings of the Symposium on Applied Computing*. SAC. ACM, 2007, pp. 3–7. DOI: 10.1145/1244002.1244004.

[4] P. Bak, E. Packer, H. Ship, and D. Dotan. "Algorithmic and visual analysis of spatiotemporal stops in movement data". In *Proceedings of the International Conference on Advances in Geographic Information Systems*. SIGSPATIAL. ACM, 2012, pp. 462–465. DOI: 10.1145/2424321.2424390.

[5] D. Biro, R. Freeman, J. Meade, S. Roberts, and T. Guilford. "Pigeons combine compass and landmark guidance in familiar route navigation". In *Proceedings of the National Academy of Sciences* 104.18 (2007), pp. 7471–7476. DOI: 10.1073/pnas.0701575104.

[6] K. Buchin, M. Buchin, and J. Gudmundsson. "Constrained free space diagrams: a tool for trajectory analysis". In *International Journal of Geographical Information Science* 24.7 (2010), pp. 1101–1125. DOI: 10.1080/13658810903569598.

[7] K. Buchin, M. Buchin, and J. Gudmundsson. "Detecting single file movement". In *Proceedings of the International Conference on Advances in Geographic Information Systems*. GIS. ACM, 2008, pp. 288–297. DOI: 10.1145/1463434.1463476.

[8] K. Buchin, M. Buchin, J. Gudmundsson, M. Löffler, and J. Luo. "Detecting commuting patterns by clustering subtrajectories". In *Algorithms and Computation*. Vol. 5369. Lecture Notes in Computer Science. Springer, 2008, pp. 644–655. DOI: 10.1007/978-3-540-92182-0_57.

[9] K. Buchin, M. Buchin, M. van Kreveld, M. Löffler, R. Silveira, C. Wenk, and L. Wiratma. "Median trajectories". In *Algorithms–ESA* (2010), pp. 463–474. DOI: 10.1007/978-3-642-15775-2_40.

[10] H. Cao, O. Wolfson, and G. Trajcevski. "Spatio-temporal data reduction with deterministic error bounds". In *The VLDB Journal* 15 (3 2006), pp. 211–228. DOI: 10.1007/s00778-005-0163-7.

[11] R. Cheng, D. Kalashnikov, and S. Prabhakar. "Querying imprecise data in moving object environments". In *IEEE Transactions on Knowledge and Data Engineering* 16.9 (2004), pp. 1112–1127. DOI: 10.1109/TKDE.2004.46.

[12] K. Clarkson. "Fast algorithms for the all nearest neighbors problem". In *IEEE Annual Symposium on Foundations of Computer Science* (1983), pp. 226–232. DOI: 10.1109/SFCS.1983.16.

[13] L. Etienne, T. Devogele, and A. Bouju. "Spatio-temporal trajectory analysis of mobile objects following the same itinerary". In *Advances in Geo-Spatial Information Science* 10 (2012), pp. 47–57.

[14] R. Freeman, R. Mann, T. Guilford, and D. Biro. "Group decisions and individual differences: route fidelity predicts flight leadership in homing pigeons (Columba livia)". In *Biology Letters* 7.1 (2011), pp. 63–66. DOI: 10.1098/rsbl.2010.0627.

[15] J. Gudmundsson, J. Katajainen, D. Merrick, C. Ong, and T. Wolle. "Compressing spatio-temporal trajectories". In *Computational Geometry* 42.9 (2009), pp. 825–841. DOI: 10.1016/j.comgeo.2009.02.002.

[16] J. Gudmundsson and M. van Kreveld. "Computing longest duration flocks in trajectory data". In *Proceedings of the International Symposium on Advances in Geographic Information Systems*. GIS. ACM, 2006, pp. 35–42. DOI: 10.1145/1183471.1183479.

[17] J. Gudmundsson, M. Kreveld, and B. Speckmann. "Efficient detection of patterns in 2D trajectories of moving points". In *GeoInformatica* 11.2 (2007), pp. 195–215. DOI: 10.1007/s10707-006-0002-z.

[18] S. Jadhav and A. Mukhopadhyay. "Computing a centerpoint of a finite planar set of points in linear time". In *Discrete & Computational Geometry* 12.1 (1994), pp. 291–312.

[19] N. Kami, N. Enomoto, T. Baba, and T. Yoshikawa. "Algorithm for detecting significant locations from raw GPS data". In *Discovery Science*. Vol. 6332. Lecture Notes in Computer Science. Springer, 2010, pp. 221–235. DOI: 10.1007/978-3-642-16184-1_16.

[20] M. Kreithen and D. Quine. "Infrasound detection by the homing pigeon: A behavioral audiogram". In *Journal of Comparative Physiology* 129.1 (1979), pp. 1–4. DOI: 10.1007/BF00679906.

[21] M. van Kreveld and L. Wiratma. "Median trajectories using well-visited regions and shortest paths". In *Proceedings of the International Conference on Advances in Geographic Information Systems*. SIGSPATIAL. ACM. 2011, pp. 241–250. DOI: 10.1145/2093973.2094006.

[22] P. Laube, S. Imfeld, and R. Weibel. "Discovering relative motion patterns in groups of moving point objects". In *International Journal of Geographical Information Science* 19.6 (2005), pp. 639–668. DOI: 10.1080/13658810500105572.

[23] P. Laube, M. van Kreveld, and S. Imfeld. "Finding REMO-detecting relative motion patterns in geospatial lifelines". In *Developments in Spatial Data Handling*. Springer, 2005, pp. 201–215. DOI: 10.1007/3-540-26772-7_16.

[24] J.-G. Lee, J. Han, X. Li, and H. Gonzalez. "TraClass: trajectory classification using hierarchical region-based and trajectory-based clustering". In *Proceedings of the VLDB Endowment* 1.1 (2008), pp. 1081–1094. DOI: 10.1.1.140.6708.

[25] B. Moreno, V. Times, C. Renso, and V. Bogorny. "Looking inside the stops of trajectories of moving objects." In *Proceedings of the Brazilian Symposium on Geoinformatics*. Vol. 11. 2010, pp. 9–20.

[26] M. Nanni and D. Pedreschi. "Time-focused clustering of trajectories of moving objects". In *Journal of Intelligent Information Systems* 27.3 (2006), pp. 267–289. DOI: 10.1007/s10844-006-9953-7.

[27] A. Palma, V. Bogorny, B. Kuijpers, and L. Alvares. "A clustering-based approach for discovering interesting places in trajectories". In *Proceedings of the ACM Symposium on Applied Computing*. SAC. ACM, 2008, pp. 863–868. DOI: 10.1145/1363686.1363886.

[28] D. Pfoser and C. Jensen. "Capturing the uncertainty of moving-object representations". In *Proceedings of the 6th International Symposium on Advances in Spatial Databases*. SSD. Springer-Verlag, 1999, pp. 111–131. DOI: 10.1.1.17.4724.

[29] P. Vaidya. "An $\mathcal{O}(n \log n)$ algorithm for the all-nearest-neighbors problem". In *Discrete & Computational Geometry* 4.1 (1989), pp. 101–115. DOI: 10.1007/BF02187718.

[30] E. Welzl. "Smallest enclosing disks (balls and ellipsoids)". In *New Results and New Trends in Computer Science*. Vol. 555. Lecture Notes in Computer Science. Springer, 1991, pp. 359–370. DOI: 10.1007/BFb0038202.

[31] E. Wentz, A. Campbell, and R. Houston. "A comparison of two methods to create tracks of moving objects: linear weighted distance and constrained random walk". In *International Journal of Geographical Information Science* 17.7 (2003), pp. 623–645. DOI: 10.1080/1365881031000135492.

[32] L. Wiratma. "Following the majority: a new algorithm for computing a median trajectory". INF/SCR-09-103. MA thesis. Department of Information and Computing Sciences, Utrecht University, 2010.

[33] M. Yodlowski, M. Kreithen, and W. Keeton. "Detection of atmospheric infrasound by homing pigeons." In *Nature* (1977). DOI: 10.1038/265725a0.

[34] Y. Zhu and J. Xu. "On the 2-central path problem". In *Computing and Combinatorics*. Vol. 7434. Lecture Notes in Computer Science. Springer, 2012, pp. 543–555. DOI: 10.1007/978-3-642-32241-9_46.

[35] Y. Zhu and J. Xu. "On the central path problem". In *Combinatorial Optimization and Applications*. Vol. 7402. Lecture Notes in Computer Science. Springer, 2012, pp. 138–150. DOI: 10.1007/978-3-642-31770-5_13.

[36] M. Zimmermann, T. Kirste, and M. Spiliopoulou. "Finding stops in error-prone trajectories of moving objects with time-based clustering". In *Intelligent Interactive Assistance and Mobile Multimedia Computing*. Vol. 53. Communications in Computer and Information Science. Springer, 2009, pp. 275–286. DOI: 10.1007/978-3-642-10263-9_24.