Segment-Based Map Matching

Sudarshan S. Chawathe

***Abstract*— We address the problem of determining the path of a vehicle on a given vector map of roads, based on tracking data such as that obtained from onboard GPS receivers. We describe a method that is based on a piecewise matching of track segments to map features. A notable feature of our method is that it is applicable to a large class of existing methods. We discuss metrics for evaluating the output of map-matching methods and briefly describe our implementation of a map-matching system based on our methods.**

I. Introduction

The map-matching problem is, in general, the problem of correlating the path of a vehicle to a vector map of roads or other features. There are several variants of this problem, based on the kinds of input data and the desired output. On the output side, we may be interested in either the instantaneous (current) position of a vehicle or a sequence of recent or historical positions. On the input side, we may use data from sources such as onboard GPS receivers, dead-reckoning systems, and computer vision systems [12], [8]. For concreteness, we focus on GPS data in this paper; however, our method is also applicable to other systems.

In the case of GPS-based map-matching, the dynamic input consists of a sequence of time-stamped geo-coordinates (*track points*), which we shall henceforth call the *trajectory* data. The static input consists of a map of geographical features in geocoded vector format, henceforth simply *map*. Intuitively, our task is that of plotting the trajectory on the map. This problem is nontrivial because the trajectory data is typically subject to considerable errors, often with magnitudes much larger than the distance between geographical features in the map. Thus, the 90% confidence region surrounding a track point may encompass several map features. Further, an implicit requirement is that the vehicle path produced as output be consistent with topological and other constraints induced by the map features. For example, the path cannot jump from a highway to a local road unless there is a suitable connection between the two, such as an exit ramp. Similarly, travel along local roads must be consistent with their interconnections. Thus, although the distance between two parallel roads may be very small, the path cannot transition from a position on one road to that on the other if there is no connecting cross street.

There are two major sources of errors in the trajectory data. The first is GPS measurement error, which arises from the inherent limitations of GPS methods. Typically, measurement error in time and the three spatial dimensions is modeled using random variables with a Gaussian distribution. The positional accuracy may be described using a bivariate normal distribution corresponding to the two horizontal spatial dimensions, as depicted by Fig. 1. The standard deviation of this distribution provides a measure of the accuracy. Although the standard deviation can be quite low in the best case, around 3 meters, it can increase several-fold due to tree cover, urban canyons, and other problems. The second source of errors in the trajectory data is the limited sampling rate. A vehicle moving at highway speeds may cover a considerable distance between two consecutive readings from the GPS receiver, often crossing multiple features in a map.

Early methods for map-matching may be broadly classified into those based on geometry and those based on topology. The former use primarily geometric calculations of distance and orientation to determine a vehicle's path on a map. Although intuitively appealing, these methods are known to suffer from significant drawbacks because they do not consider the topological constraints induced by a map [6]. Topological methods use the topology of map features to constrain the set of potential matches for a track point. For example, given prior positions on a road, several geometrically close roads may be removed from consideration because there is no way to go from one to the other or, more generally, because the shortest route from one to the other is longer than some threshold based on vehicle speed and other factors. Although these methods are less likely to generate topologically infeasible paths, they suffer from two related problems: First, determining a globally consistent and optimal (best match) path using such methods is very resource-intensive and often practically infeasible. Second, when used in a local manner (to improve efficiency), these methods may not always yield a solution unless unlimited backtracking is allowed, in which case the methods are again likely to be very resource-intensive. For example, consider a road that splits into two almost parallel, but slowly diverging roads, as suggested by Fig. 2. An initial error of choosing the wrong road can remain undetected for a long time, until the distance between the diverging roads grows large, requiring unbounded backtracking for a guaranteed solution. An actual case produced by our map-matching system is depicted in Fig. 3.

For efficiency reasons, many methods for map-matching, whether based on geometry, topology, a combination, or other hybrid techniques, operate on the trajectory in a piece-

S.S. Chawathe is with the Computer Science Department, University of Maine, Orono, Maine 04469-5752, USA. chaw@cs.umaine.edu

This work was supported in part by the U.S. National Science Foundation with grants IIS-0081860 and CNS-0426683.

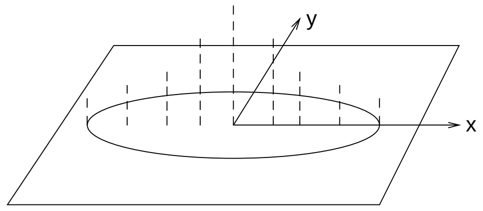

Fig. 1. Positional accuracy model. The horizontal axes are labeled x and y, with x being the direction of travel. The ellipse outlines a 90% confidence region. A projection of the bivariate Gaussian probability density function on the $y = 0$ plane is suggested by the dashed lines.

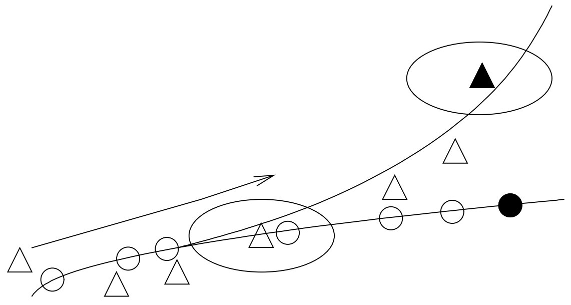

Fig. 2. The need for unlimited backtracking due to topological constraints. The bifurcated line represents roads diverging from a fork and the arrow denotes the general direction of travel. Triangles represent track points while circles represent map points. The filled triangle represents the most recent track point. Although this point lies very close to the upper fork of the road, a topology-guided method may be forced to map it to a point in the lower fork (filled circle) well outside the elliptical error region (solid ellipse) of the track point due to an erroneous decision much earlier, when the two forks were well within the error region (dashed ellipse). The situation cannot be remedied in a topologically consistent manner without backtracking that changes the earlier choice of the upper fork.

wise manner. For example, a geometry-based method may compute the best fit between trajectory points and map roads based on trajectory pieces that are limited using trajectory points (e.g., 10 points) or the induced distance (e.g., 100 meters). In this manner, the complexity of computing a best fit can be controlled. Similarly, a topology-based method may limit the complexity of the combinatorial process of matching points to line segments in a map by computing the match 10 points at a time. We shall henceforth refer to such pieces of the trajectory as *segments*.

In this paper, we do not propose a new, stand-alone method for map-matching. Instead, we propose a method that can be used to improve many existing segment-based methods, as well as other methods adapted to segments. Although our modification is applicable to a variety of such methods, for concreteness, we shall present our work in the context of one specific method: a simple incremental method that uses a combination of geometric and topological ideas.

The rest of this paper is organized as follows. Section II describes two simple, point-wise methods for mapping a trajectory to paths in maps. The first method is based on making decisions by examining one track point at a time while the second uses a limited amount of look-ahead. Our main purpose in presenting these methods is to establish a

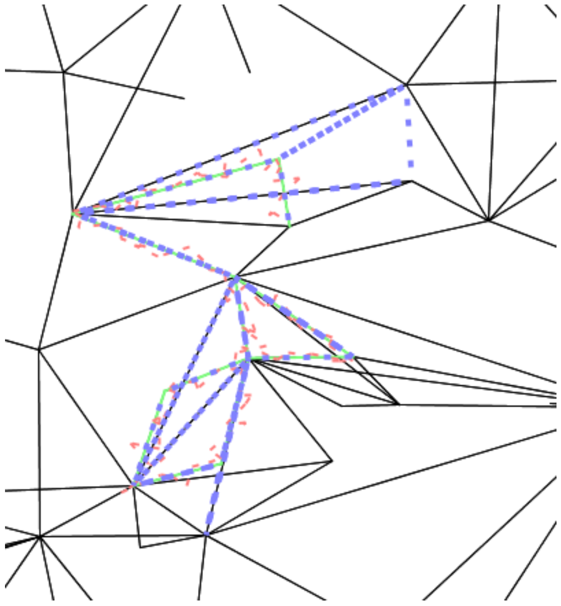

Fig. 3. A screenshot of a portion of the display area of the GeoTrackMapper system (Section V) illustrating a real case similar to the conceptual example of Fig. 2. The solid line represents the path of actual travel. The dashed line that meanders around this solid line is trajectory of track points. The thicker dashed line is the map path computed by our method when used without backtracking. We note the large error near the top right: The computed path wanders off into an area that is very far from the nearest track points. This error is caused by the topological constraints induced by the road network. The incorrect choice was made at an earlier point in the path: the crowded corner near the top left.

concrete context for describing our segment-based improvements. These improvements are the topic of Section III, which describes a static segmented path-matching method followed by a dynamic version that enables the underlying map-matching method to update accuracy estimates based on partial results. Section IV addresses the topic of measuring the quality of a map-path produced by map-matching methods by quantifying its divergence from the real path, if known, and the input trajectory. Our implementation of the *GeoTrackMapper* system based on these ideas is described briefly in Section V. Related work is described in Section VI and Section VII summarizes our work and outlines ongoing work.

II. Map-Matching

In this section, we describe a simple map-matching method that is based on a measure of similarity defined between track points and candidate edges, where candidate edges are the line segments that represent the roads of a map. Briefly, the similarity between a point and an edge is composed of two parts. The first quantifies how close the point is to the edge, while the second quantifies the similarity in orientation (direction). The specific details of these calculations, which are found in prior work [11], [7], are immaterial for our purposes, and we therefore skip them in this paper. For our purposes, all that is needed is an encapsulation of the similarity measure between a track point

x and a map feature f in a function SIM(x, f).

Using the similarity function SIM, we can match each track point x in a path $P = P[1], P[2], \ldots, P[m]$ in order by selecting the candidate feature that maximizes the similarity score. The candidate features are those that are connected to the feature matched to the previous track point. This procedure is summarized as POINTMATCH in Listing 1. In that listing, P is the input trajectory (sequence of track points), G is the map, i is the index of the track point to be matched, and M is an array storing the matching. That is, the feature matched to track point $P[i]$ is stored in $M[i]$. The function CONN(f) returns the set composed of the given feature f along with all features connected to it, based on the topology of the underlying map. For example, if a track point $P[3]$ is matched to a road r, then CONN$(M[3])$ returns r and all roads connected to r. As is the case with SIM, our method does not depend on the details of how CONN is implemented. The implementation may be a simple test for connectivity or a more sophisticated test that incorporates direction and other attributes. If there is no previous track point, or if the feature matched to the previous track point is unavailable, then the candidate features are those within a distance T from the track point. Such features can be located efficiently using standard techniques [25].

Listing 1 Match the ith point in path P to a feature in map G, storing results in M.

```
1: procedure PointMatch($P, G, i, M$)
2:     $P = P[1] \ldots P[m]$;  $M[0] = \perp$
3:     if $M(P[i-1]) \neq \perp$ then
4:         $C \leftarrow$ CONN$(M(P[i-1]))$
5:     else
6:         $C \leftarrow \{f \in G \mid d(P[i], f) < T\}$
7:     end if
8:     $M[i] \leftarrow \mathrm{argmax}_{f \in C}$ SIM$(P[i], f)$
9: end procedure
```

Procedure POINTMATCH considers each track point only once. The CONN function is implemented to run in time proportional to the number of connected features [9]. Over the entire run of POINTMATCH, each connection is accessed at most twice (once from either end). Thus the total time required by CONN is $O(|G|)$, where $|G|$ denotes the size of the map. We can locate the features as indicated on line 6 using multi-dimensional orthogonal range queries in $O(\log |G| + |C|)$ time using a data structure that uses $O(|G| \log^{1+\epsilon} |G|)$ space [2]. Here $|C|$ denotes the size of the result, as on line 6, and is expected to be small. The time spent on line 8 is $m \cdot |C|$ times the time required by the similarity function SIM. (Recall that m is the number of track points in the input path $P = P[1], P[2], \ldots, P[m]$.) Thus, the overall running time is $O(m \cdot \log |G| + m \cdot |C| \cdot S)$, where S is the time required by an invocation of SIM.

Although POINTMATCH provides a simple method for matching track points to map features, it does not provide satisfactory results on typical real data. Intuitively, the main problem is that it matches each point almost individually, considering only the feature matched to the immediately preceding point, if any. In particular, it does not consider the ramifications that matching a point to a feature has on the feasible matchings of later points. As indicated by our example in Fig. 2, a locally optimal choice may force the choice of a poor solution later in the trajectory.

Listing 2 summarizes a method that matches a track point to a map feature using a strategy of limited look-ahead. Instead of simply picking the feature that maximizes the point-wise similarity function, as done by POINTMATCH, the for loop of line 11 recursively computes, for each candidate feature f, the aggregate similarity of matching the next l track points, subject to the assumption that the current point is matched to f. The matching M' is used to hold a working copy of M during these computations and the final choice of a feature to match $P[i]$ is the one, f^*, that yields the maximum aggregate similarity. Although we compute the aggregate similarity by assuming a matched feature for each of the next l points, only the matching of the point $P[i]$ is finalized. That is, the points are still matched one at a time.

Listing 2 Match the ith point in path P to a feature in map G, storing results in M and using a lookahead of l points in the path. The function returns the similarity between the matched point and the track point.

```
 1: function LocalMatch($P, G, i, M, l$)
 2:     $P = P[1] \ldots P[m]$;  $M[0] = \perp$
 3:     $M' \leftarrow M$
 4:     if $M'(P[i-1]) \neq \perp$ then
 5:         $C \leftarrow$ CONN$(M'(P[i-1]))$
 6:     else
 7:         $C \leftarrow \{f \in G \mid d(P[i], f) < T\}$
 8:     end if
 9:     $f^* = \perp$
10:     $v^* = -\infty$
11:     for all $f \in C$ do
12:         $M'[i] \leftarrow f$
13:         $v \leftarrow$ SIM$(P[i], f)$
14:         $v \leftarrow v +$ LocalMatch$(P, G, i+1, M', l-1)$
15:         if $v > v^*$ then
16:             $v^* \leftarrow v$
17:             $f^* \leftarrow f$
18:         end if
19:     end for
20:     $M[i] = f^*$
21:     return $v^*$
22: end function
```

In offline map-matching, the entire trajectory is available and the LOCALMATCH method can be applied directly as described above. In online map-matching, the trajectory is available as it develops in real time. In this case, we may interpret the limited lookahead as an option for limited backtracking by reversing the direction of the lookahead. That is, instead of examining the next l track points to determine the matching of the current one, we reexamine the last l track-points to determine if we should alter an earlier match based on the newly arrived track points. Using an analysis similar to that for POINTMATCH, we can determine that the running time of LOCALMATCH is at most l times that of POINTMATCH.

III. SEGMENT-WISE MATCHING

The description of the previous section implicitly assumed that map-matching benefits from considering more than one point at a time. When the track points in the input have very similar positional accuracy, this assumption is valid. However when, as is often the case, the positional accuracy of track points varies significantly over the trajectory, this assumption is not valid. For example, it is very common for the positional accuracy to drop significantly (i.e., the standard deviation to rise significantly) when a vehicle enters an area with tall buildings or dense tree cover. In such cases, a look-ahead that includes the low-accuracy track points may degrade the quality of the solution by causing mismatches for near-by high-accuracy track points. Intuitively, we would like to strike a balance between using a larger number of track points (larger look-ahead) of, in general, lower accuracy and using a smaller number of more accurate track points.

Our segment-based matching methods build on the above idea. We use a function SCORE to encapsulate the varying effectiveness of track points for the purpose of accurate matching. As with functions SIM and CONN of the previous section, our methods do not depend on any specific implementation of the SCORE function. A simple version of this function assigns to each track point a score that is inversely proportional to its positional accuracy. However, a more sophisticated version of this function may also use other factors that affect the likelihood of a correct match, such as the sampling frequency and the number of candidate features. For instance, a track point with a low positional accuracy may nevertheless be assigned a high score because there is only one candidate feature in its vicinity. A track point on a remote highway with dense tree cover, and a resulting low positional accuracy, may thus receive a high score because there are no other roads nearby and therefore no risk of matching it to the wrong road.

We refer to a sequence of contiguous track-points as a segment. The main idea of segment-based matching is to match track-points belonging to high-score segments before those belonging to low-score segments. In contrast, the methods of the previous section match track points in sequential order by time.

The first of these methods is summarized as procedure SEGPATHMATCH in Listing 3. This procedure uses a heap (priority queue [9]) Q to organize segments of length K in nonincreasing order of aggregate scores. The first part of the procedure computes the aggregate scores and builds the heap Q and the second part matches all points in each segment as it is removed from Q in heap order. The MATCH function is very similar to the LOCALMATCH function of Listing 2, with a lookahead of K. The main difference is that while one invocation of LOCALMATCH results in matching only one track point (although others are considered), a single invocation of MATCH matches all track points in the given segment. A simple option is to implement MATCH by invoking LOCALMATCH (Listing 2) K times. However, SEGPATHMATCH does not assume such an implementation; therefore, MATCH may also be implemented using other methods, such as one that determines the geometric best-fit of the K points in the segment to the underlying map.

Listing 3 Segmented Path Matching (Static) Match trajectory P to map G by matching segments of K track points in order of nonincreasing segment scores.

```
 1: procedure SEGPATHMATCH($P, G$)
 2:     $P = P[1] \dots P[m]$
 3:     $S[0] \leftarrow 0$
 4:     for all $i = 1 \dots m$ do
 5:         $S[i] \leftarrow$ SCORE($P[i], G$)
 6:     end for
 7:     $v \leftarrow 0$
 8:     for all $i = 1 \dots K - 1$ do
 9:         $v \leftarrow v + S[i]$
10:     end for
11:     for all $i = K \dots m$ do
12:         $v \leftarrow v - S[i - K] + S[i]$
13:         $Q$.INSERT($v, i$)
14:     end for
15:     $T[1 \dots m] \leftarrow 0 \dots 0$
16:     for all $j = 1 \dots \lceil m/K \rceil$ do
17:         repeat
18:             $n \leftarrow Q$.DELETEMAX()
19:         until $T[n] = 0$
20:         $T[n] \leftarrow 1$
21:         MATCH($P[n - K + 1 \dots n], G$)
22:     end for
23: end procedure
```

Lines 3 through 14 build the heap Q linear time by making two passes over the track points $P[1] \dots P[m]$. The first pass (first for loop) computes the score of each point while the second pass (second and third for loops) computes the weight $\sum_{i=B-K+1}^{B} Score(P[i], B)$ for $B = K \dots m$ by maintaining a running total, and uses it as a key for inserting segment $P[B - K + 1] \dots P[B]$ (identified by B) in the heap. Lines 15 through 22 repeatedly dequeue and match the unmatched segment with the largest weight. Unmatched segments are identified using the array T.

Other than line 21, the running time of Procedure SEGPATHMATCH is dominated by the heap operations on lines 13 and 18, which are invoked $O(m)$ times. Given the standard $O(\log |m|)$ implementation of heaps, we have $O(m \log m)$ plus the time spent in MATCH as the total running time. Our implementation of a segment-based match is very similar to that of procedure LOCALMATCH of the previous section, and each track point is considered only once. Thus the overall running time is $O(m \cdot \log m + m \cdot \log |G| + m \cdot |C| \cdot S)$, where, as before, S is the time required by an invocation of SIM.

As suggested earlier, a sophisticated SCORE function, which is used to gauge the expected accuracy of matching a track point, may depend on not only the positional accuracy of a track point, but also other factors. Some of these factors may depend on the partial matching of track points to map features. In such cases, computing the scores of all track points at once, before any matches have been made, as done in Listing 3 is not suitable. Instead, we use a dynamic segment-wise matching method as summarized in Listing 4 as the recursive procedure DYNSEGPATHMATCH. In this procedure, we first determine a segment with maximum aggregate score. We use a different score function, SCORE2, to indicate that the scores now depend on the current partial matching M, which is its third argument. Similarly, the maximum-score segment is matched using a function MATCH2 that takes the partial matching as an argument. After the segment is matched, the portions of the trajectory before and after the segment, if nonempty, are matched recursively. On average, we may expect the recursive invocations of DYNSEGPATHMATCH on lines 22 and 25 to occur on trajectories of roughly half the size of the parent invocation. Thus, this modification to SEGPATHMATCH adds at most a factor of $\log m$ to the expected running time. In worst case, it may add a factor of m.

IV. METRICS

One measure of the quality of the solution produced by a map-matching method is, intuitively, the closeness between the input trajectory, or real trajectory, if known, and the output path. To quantify this idea, we use the idea of a *distance* between the two curves representing the input trajectory and the output path. Several definitions of such a distance have been proposed in prior work [4]. An obvious choice is the Hausdorff metric between two paths, using the Euclidean space as the underlying metric space. Intuitively, the Hausdorff distance between paths p_1 and p_2 is the smallest number d such that every point in p_1 is within a distance d of some point in p_2. We may think of the Hausdorff distance as the thickness of the padding that we must add to each path so that the other will lie completely inside the padded area. More precisely, the Hausdorff distance between paths p_1 and p_2 is given by

$$d_H(p_1, p_2) = \sup_{x_1 \in p_1} \inf_{x_2 \in p_2} d(x_1, x_2)$$

where $d(x_1, x_2)$ is the Euclidean distance between points x_1 and x_2.

Although the Hausdorff distance is popular, it is not ideal for map-matching applications because, intuitively, it does not take the position along the paths into account. For example, consider the two paths suggested by Fig. 4. It is clear that every point on the path AB is at most

Listing 4 Dynamic Segmented Path Matching. Match trajectory P to map G segment-wise, as in Listing 3, but with updates between segment matches.

```
1:  procedure DYNSEGPATHMATCH($P, G, M$)
2:      $P = P[1] \dots P[m]$
3:      $S[0] \leftarrow 0$
4:      for all $i = 1 \dots m$ do
5:          $S[i] \leftarrow$ SCORE2($P[i], G, M$)
6:      end for
7:      $v \leftarrow 0$
8:      for all $i = 1 \dots K - 1$ do
9:          $v \leftarrow v + S[i]$
10:     end for
11:     $v^* \leftarrow -\infty$
12:     $i^* \leftarrow 0$
13:     for all $i = K \dots m$ do
14:         $v \leftarrow v - S[i - K] + S[i]$
15:         if $v > v^*$ then
16:             $v^* \leftarrow v$
17:             $i^* \leftarrow i$
18:         end if
19:     end for
20:     $M \leftarrow M \cup$ MATCH2($P[i^* - K + 1 \dots i^*], G$)
21:     if $i^* > K$ then
22:         DYNSEGPATHMATCH($P[1 \dots i^* - K], G, M$)
23:     end if
24:     if $i^* < m - 1$ then
25:         DYNSEGPATHMATCH($P[i^* + 1 \dots m], G, M$)
26:     end if
27: end procedure
```

0.5m from the nearest point in path CD, and vice versa; thus, $d_H(\text{AB}, \text{CD}) = 0.5$. However, the two paths are very dissimilar to each other for map-matching purposes.

The above shortcoming of the Hausdorff distance may be addressed by using the Frechet distance, which takes the position along paths into account. Intuitively, the Frechet distance between two paths is the length of the shortest possible rope that could be tied between two cars while permitting the cars to travel on the two paths, respectively, moving only in the forward direction. That is, the cars can adjust their speeds to try to accommodate for the rope, but they cannot move backward along the path at any time. More precisely, the Frechet distance between paths p_1 and p_2 is given by

$$d_F(p_1, p_2) = \inf_{m_1, m_2 : [0,1] \to [0,1]} \max_{t \in [0,1]} d(m_1(t), m_2(t))$$

where the paths p_1 and p_2 are parameterized using t and the infimum ranges over all possible reparameterizations m_1 and m_2, for p_1 and p_2, respectively, and where d is, as before, the Euclidean distance. The reparameterizations m_1 and m_2 are required to be continuous and nondecreasing, and to map the points 0 and 1 to themselves.

Returning to the two paths suggested by Fig. 4, it is clear

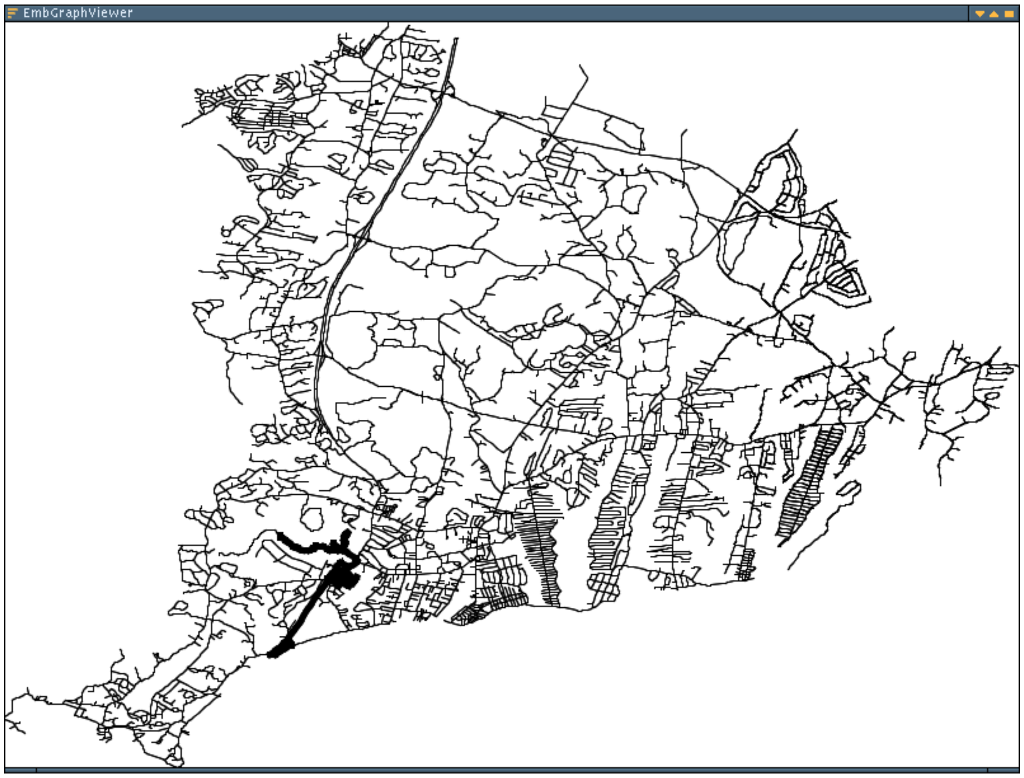

Fig. 5. A screenshot of the visualization component of the *GeoTrackMapper* map-matching system. The window shows a map of streets in Falmouth, Massachusetts. The path composed of thick edges, near the bottom right corner of the screenshot, depicts a trail that has been correctly mapped based on track points.

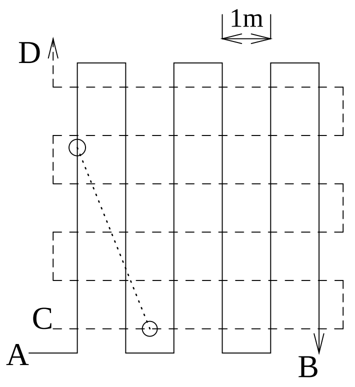

Fig. 4. Dissimilar paths AB and CD with low Hausdorff distance and high Frechet distance.

that the Frechet distance is much higher than the Hausdorff distance. For example, if the dotted line represents a rope of length approximately 5m, then the two circles represent a configuration in which no car can make forward progress without breaking the rope. It is sometimes useful to use a variant of the Frechet distance, called the weak Frechet distance, in whose definition the reparameterizations m_1 and m_2 are not required to be nondecreasing. In the intuitive interpretation of a car traveling along each paths with a rope connecting the cars, this variant permits the cars to back up along the paths in order to accommodate the rope. Other variants, such as the average Frechet distance, have also been proposed [7]. Although not obvious from the definition, the Frechet distance between paths p_1 and p_2, with n_1 and n_2 points, respectively, can be computed in $O(n_1 n_2 log^2(n_1 n_2))$ time [3].

V. Implementation

Based on the methods described in this paper, we have built a map-matching system called *GeoTrackMapper*. The implementation uses pure Java and therefore runs on any platform supported by the *Java2 SE 5.0* runtime environment. The graphical features use the standard *Swing* libraries, and some file formats are supported using the *GeoTools* libraries [10]. The system has a modular design that supports multiple uses, ranging from time-sensitive in-vehicle map-matching to large batch simulations using a combination of real and synthetic datasets. For example, not only can the system evaluate various map-matching methods on real map datasets, but it can also transform those datasets in various ways in order to test the sensitivity of the methods to properties such as road-segment lengths, number and density of intersections, ratio of road-segments to intersections, and the resolution of lines modeling the roads.

For our testing, we have made extensive use of real road data provided by a number of U.S. agencies. For example, Fig. 5 depicts a screenshot of a map dataset from the *MassGIS* collection, which contains detailed road data for the major towns in Massachusetts [20]. We have found some of the map-transformation features of GeoTrackMapper to be quite useful in dealing with some problematic features of such real data. For instance, we found that in order to properly model intersections so that topology-based matching methods work properly, it was useful to smooth the map dataset by merging the endpoints of features that are closer than a specified threshold.

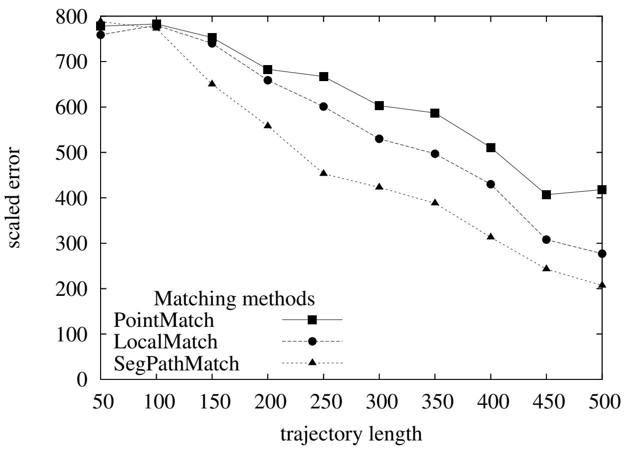

Fig. 6. Matching error as a function of trajectory length.

Figure 6 summarizes the results of some experiments using *GeoTrackMapper*. For this set of experiments, we used the map of Falmouth, Massachusetts, depicted in Fig. 5, as the basis. Trajectories were generated by first generating a path in the network of roads and then perturbing the geo-locations of this path using a random variable with a Gaussian distribution of varying parameters. To generate a path in the road network, we first randomly select an intersection in the network using a uniform distribution. Edges (road segments) are added to the path one at a time by randomly selecting one edge among those incident on the current intersection, with a uniform distribution with one exception: If the edge selected is the one that was taken to arrive at this intersection then we make another random selection, up to a total of five times. This correction generates paths that reflect real trajectories more accurately than paths selected uniformly at random, because the latter tend to have a very high degree of redoubling. To obtain a sequence of geo-locations from a path, the path is sampled with $10 \times l$ points uniformly separated along the path, where l is the number of edges in the path.

For each trajectory thus generated, we find a matching path in the network using three methods: POINTMATCH of Listing 1, LOCALMATCH of Listing 2, and SEGPATHMATCH of Listing 3. For each resulting path, we compute the error as the sum of the distances between the trajectory points, before addition of Gaussian errors, and the mapped points. The scaled error, which is plotted in Fig. 6, is obtained by dividing the error by the number of trajectory points. We note that, in general, the scaled error decreases with increasing trajectory lengths. This behavior is expected, because the aggregate errors are typically caused by a few large errors, whose effect is larger for shorter trajectories. We also note that the segmented method exhibits an improvement in accuracy and that this improvement is more pronounced for longer trajectories. This result is also expected, because the potential for large errors in non-segmented methods is greater for longer paths. That is, once a non-segmented method makes a matching decision, the longer the path, the greater the likelihood that that decision leads to a very large error later in the trajectory. For the segmented method, on the other hand, the errors are limited by segment length and, further, the ordering of segment-matching by the score functions helps reduce the likelihood of errors.

VI. RELATED WORK

An early paper by Bernstein and Kornhauser provides a good introduction to the general problem of map-matching [6]. A more recent survey appears in Quddus's thesis [22]. As indicated in Section II, we have presented our segment-based method in the context of a simple local-matching method proposed by Brakatsoulas, Pfoser, Salas, and Wenk [7], which in turn uses similarity measures described by Greenfeld [11]. Alt, Erfat, Rote, and Wenk describe a method for the efficient evaluation of the Frechet distance discussed in Section IV [5]. Alstrup, Brodal, and Rauhe describe methods for multi-dimensional orthogonal range queries in $O(\log |G| + |C|)$ time and $O(|G| \log^{1+\epsilon} |G|)$ space [2]; these allow efficient location of map features in the proximity of track points, as required for bootstrapping the map-matching method of Section II.

Aigong, Voon, and Look describe the use of map-matching in a GPS/GIS ERP system in Singapore [1]. Quddus, Noland, and Ochieng study the effect of map quality on map-matching algorithms [23]. As noted in Section V, our implementation of the *GeoTrackMapper* system allows us to study such issues by performing various transformations, such as smoothing and reduction of detail, on real [20] and synthetic maps. The modular design of *GeoTrackMapper* is suitable for use in real-time vehicle-location problems, as studied by Jagadeesh, Srikanthan, and Zhang [15].

Quddus, Ochieng, Zhao, and Noland describe an application of map matching that monitors vehicle performance and emissions [24]. Their algorithm combines GPS data with data from low-cost dead reckoning sensors using an extended Kalman filter. Pyo, Shin, and Sung use a multiple-hypothesis technique that consists of generating pseudo-measurements on roads in the vicinity of the location indicated by a GPS measurement, along with a Kalman filter to estimate the bias errors . They report experimental results indicating consistent performance despite signal degradation in urban environments. Lamb and Thiebaux describe a method that uses closely coupled Kalman filters and Markov models . The former are used to process the metric aspects of the map-matching problem, while the latter is used for the topological aspects. The output of the Kalman filters is used to update the belief states of the Markov model. In turn, the Markov model provides a probability distribution for over the Kalman filters. Hummel and Tischler present a map-matching method based on a Bayesian classifier that incorporates a Hidden Markov Model in order to model topological constraints of the road network [14]. Their method is based solely on GPS data, without inputs from addition in-vehicle sensors, but nevertheless has been shown experimentally to be robust in urban environments that are challenging for GPS. It should be interesting to combine probabilistic methods such as these with the segment-based scheme we describe in this paper.

As suggested earlier, there is a trade-off between the length of a segment and the average score of its constituent points. In general, the longer the segment, the lower the average score, which could lead to lower-quality results. On the other hand, longer segments provide for more look-ahead, so that it is possible to make locally nonoptimal decisions that result in better overall results. We may achieve a judicious trade-off by selecting segments in order of nonincreasing *density*, where the density of a segment is defined as the average score of track points in that segment. It is possible to use recent work on efficient computation of maximum-density substrings for this purpose [18], [17], [13], [19].

VII. Conclusion

We presented techniques for modifying existing methods for map-matching based on the idea of segment-wise matching, where a segment is contiguous sequence of track points. Track points and segments are assigned scores that quantify the expected accuracy of matching them to map features. A notable feature of our techniques is that they do not make strong assumptions about the specific low-level methods used for matching points to features. For concreteness, we described our work in the context of a simple map-matching method based on geometric and topological matching. However, the ideas apply to other, more sophisticated methods as well. We briefly discussed some issues related to quantifying the quality of the output of map-matching methods based on the Frechet distance. We also briefly described the *GeoTrackMapper* map-matching system that we have implemented based on the methods of this paper. We have tested our work on several real and synthetic datasets, such as those from *MassGIS* [20]. In continuing work, we are adding additional map-mapping methods to *GeoTrackMapper* and evaluating its performance on additional datasets, both real and synthetic. In particular, we are investigating the effect of non-Gaussian noise and nonuniform sampling.

References

[1] Xu Aigong, Ling Keck Voon, and Law Choi Look. Research on the GPS/GIS based ERP system in Singapore. *IEEE Intelligent Transportation Systems Society Newsletter*, 7(2):16–20, June 2005.

[2] Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. New data structures for orthogonal range searching. In *Proceedings of the 41st IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 198–207, Redondo Beach, California, November 2000.

[3] H. Alt, A. Efrat, G. Rote, and C. Wenk. Matching planar maps. *Journal of Algorithms*, 49:262–283, 2003.

[4] H. Alt and L. Guibas. Discrete geometric shapes: Matching, interpolation, and approximation—a survey. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*. Elsevier, 1999.

[5] Helmut Alt, Alon Erfat, Günter Rote, and Carola Wenk. Matching planar maps. *Journal of Algorithms*, 49(2):262–283, July 2003.

[6] David Bernstein and Alain Kornhauser. An introduction to map matching for personal navigation assistants. Technical report, New Jersey TIDE Center, New Jersey Institute of Technology, Newark, New Jersey, August 1996.

[7] Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. On map-matching vehicle tracking data. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB)*, pages 853–864, Trondheim, Norway, August 2005.

[8] W.C. Collier. In-vehicle route guidance systems using map matched dead reckoning. In *Proceedings of the IEEE Position Location and Navigation Symposium*, pages 359–363, 1990.

[9] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, June 1990.

[10] GeoTools: The open source Java GIS toolkit. `http://geotools.codehaus.org/`, 2007.

[11] Joshua Greenfeld. Matching GPS observations to locations on a digital map. In *Proceedings of the 81st Annual Meeting of the Transportation Research Board*, Washington, D.C., January 2002.

[12] B. Hofmann-Wellenhoff, H. Lichtenegger, and J. Collins. *GPS: Theory and Practice*. Springer-Verlag, New York, 1994.

[13] Sun-Yuan Hsieh and Ting-Yu Chou. Pseudo-polynomial time algorithms for the maximum-density subtree problem and related problems (extended abstract). In *Proceedings of the 23rd Workshop on Combinatorial Mathematics and Computation Theory*, pages 24–27, Changhua, Taiwan, April 2006.

[14] Britta Hummel and Karin Tischler. Robust, GPS-only map matching: Exploiting vehicle position history, driving restriction information and road network topology in a statistical framework. In *Proceedings of the GIS Research UK Conference (GISRUK)*, pages 68–77, April 2005.

[15] G. R. Jagadeesh, T. Srikanthan, and X. D. Zhang. A map matching method for GPS based real-time vehicle location. *The Journal of Navigation*, 57(3):429–440, September 2004.

[16] Peter Lamb and Sylvie Thiébaux. Avoiding explicit map-matching in vehicle location. In *Proceedings of the 6th ITS World Congress*, Toronto, Canada, November 1999.

[17] Rung-Ren Lin, Wen-Hsiung Kuo, and Kun-Mao Chao. Finding a length-constrained maximum-density path in a tree. *Journal of Combinatorial Optimization*, 9(1):145–156, 2005.

[18] Yaw-Ling Lin, Xiaoqiu Huyang, Tao Jiang, and Kun-Mao Chao. MAVG: locating non-overlapping maximum average segments in a given sequence. *Bioinformatics*, 19(1):151–152, 2003.

[19] Yaw-Ling Lin, Tao Jiang, and Kun-Mao Chao. Efficient algorithms for locating the length-constrainted heaviest segments, with applications to biomolecular sequence analysis. *Journal of Computer and System Sciences*, 65:570–586, 2002.

[20] MassGIS database. Office of Geographic and Environmental Information (MassGIS), Commonwealth of Massachusetts Executive Office of Environmental Affairs. `http://www.mass.gov/mgis/`, December 2005.

[21] Jong-Sun Pyo, Dong-Ho Shin, and Tae-Kyung Sung. Development of a map matching method using the multiple hypothesis technique. In *Proceedings of the 4th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 23–27, Oakland, California, August 2001.

[22] Mohammed A. Quddus. *High Integrity Map Matching Algorithms for Advanced Transport Telematics Applications*. PhD thesis, Imperial College, London, U.K., January 2006.

[23] Mohammed A. Quddus, Robert B. Noland, and Washington Y. Ochieng. The effects of navigation sensors and digital map quality on the performance of map matching algorithms. In *Proceedings of the 85th Annual Meeting of the Transportation Research Board*, Washington, D.C., January 2006.

[24] Mohammed A. Quddus, Washington Y. Ochieng, Lin Zhao, and Robert B. Noland. A general map matching algorithm for transport telematics applications. *GPS Solutions Journal*, 7(3):157–167, 2003.

[25] Hanan Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, San Francisco, California, 2006.