Data Mining and Knowledge Discovery, 9, 59–87, 2004

Mining GPS Traces for Map Refinement

STEFAN SCHROEDL schroedl@rtna.daimlerchrysler.com
KIRI WAGSTAFF wagstaff@rtna.daimlerchrysler.com
SETH ROGERS rogers@rtna.daimlerchrysler.com
PAT LANGLEY langley@rtna.daimlerchrysler.com
CHRISTOPHER WILSON wilson@rtna.daimlerchrysler.com
DaimlerChrysler Research and Technology North America, 1510 Page Mill Road, Palo Alto, CA 94304

Editors: Fayyad, Mannila, Ramakrishnan

Received May 16, 2001; Revised August 23, 2002

Abstract. Despite the increasing popularity of route guidance systems, current digital maps are still inadequate for many advanced applications in automotive safety and convenience. Among the drawbacks are the insufficient accuracy of road geometry and the lack of fine-grained information, such as lane positions and intersection structure. In this paper, we present an approach to induce high-precision maps from traces of vehicles equipped with differential GPS receivers. Since the cost of these systems is rapidly decreasing and wireless technology is advancing to provide the communication infrastructure, we expect that in the next few years large amounts of car data will be available inexpensively. Our approach consists of successive processing steps: individual vehicle trajectories are divided into road segments and intersections; a road centerline is derived for each segment; lane positions are determined by clustering the perpendicular offsets from it; and the transitions of traces between segments are utilized in the generation of intersection models. This paper describes an approach to this complex data-mining task in a contiguous manner. Among the new contributions are a spatial clustering algorithm for inferring the connectivity structure, more powerful lane finding algorithms that are able to handle lane splits and merges, and an approach to inferring detailed intersection models.

Keywords: digital maps, spatial data mining, GPS, floating car data

1. Introduction

Today's commercially available digital maps have achieved an accuracy in the range of a few to a few ten's of meters as well as a coverage for the major highway road network and urban regions that suffices to enable useful automotive applications; e.g., navigation and route guidance systems are becoming more and more common. However, the combination of enhanced digital road maps and precise positioning systems enables a much wider range of novel in-car applications that improve safety and convenience (Wilson et al., 1998). This is demonstrated by a number of national and international research efforts under way to investigate the promising potential of these technologies, like the U.S. Department of Transportation project *EDMap* (2002) and its European Union counterpart *NextMap* (Ertico, 2002), which are similar in scope. A partial list of possible applications for which detailed digital maps can lay the foundations includes:

- Precise information about road curvature and elevation enable **roll-over warning** and **adaptive cruise control** systems. Moreover, they can support fuel-saving applications for intelligent gear shifting and power train management for hybrid cars.
- **Lane departure warning/lane-keeping** applications track a car's current offset from the road/lane centerline. Different orders of driver interaction are conceivable, including issuing a warning signal, unobtrusive driving assistance, and even full automatic control.
- **Lane-level navigation** enhances standard road-level navigation systems to let them advise the driver about which specific lane he should choose to reach his destination without excessive or last-minute lane changes.
- **Dynamic lane closure warning** uses aggregate data on lane occupancy, available online through wireless communication, to compare current occupancies for a road segment with past lane occupancies. If a lane is particularly underrepresented, the application may conclude that it is closed due to an accident or construction, then pass this inference to navigation applications, which can take it into account when calculating routes.

For many automatic control tasks, it is beneficial to combine the generated map with different types of input using *sensor fusion* methods. For example, vision systems have been developed to recognize the offsets from lane markings. However, without an absolute positional sensing system, there is no way to register the data spatially and, consequently, no way to look ahead around corners and over hills. On the other hand, vision systems can maintain stability if the satellite signal is temporarily unavailable. Thus, due to their different failure modes, an integrated system can be made more reliable than one based on only one of these techniques.

Currently commercially available digital maps lack the necessary accuracy and detail for many useful applications, like those listed above; for example, a precondition for lane departure warning systems is information about lane positions within the accuracy of centimeters or decimeters, which is beyond the scope of most of today's available maps. Moreover, acquiring and maintaining maps with this amount of additional information would significantly increase their cost, if done in the conventional way by specialized personnel and dedicated surveying vehicles.

In this paper, we give an overview of a system that automatically generates digital road maps that are significantly more precise and contain descriptions of lane structure, including the number of lanes and their locations, along with detailed intersection structure. Our approach is statistical: we combine a large quantity of possibly noisy data from *global positioning systems* (*GPS*) for a fleet of vehicles, as opposed to a small number of highly accurate points obtained from surveying methods. We assume that the input probe data is obtained from vehicles that go about their usual business unrelated to the task of map construction, possibly piggybacking on other applications based on positioning systems. Automated processing can be much less expensive. The same holds for the price of *differential* (*DGPS*) devices; within the next few years, most new vehicles will likely have at least one DGPS receiver, and wireless technology is rapidly advancing to provide the communication infrastructure. The result will be maps with higher accuracy that are at the same time cheaper and faster to construct and maintain.

Parts of this work have been published previously (Wilson et al., 1998; Rogers et al., 1999). The main purpose of the current article is to provide a contiguous description of how the different algorithmic building blocks fit together to refine digital maps. Sometimes we will discuss alternative algorithms and heuristics whose use might depending on the final application. However, since standard formats for enhanced maps are not yet available, we strive to keep our treatment as general as possible.

New contributions of this article include (1) a spatial clustering algorithm for inferring the connectivity structure of the map from scratch, making the map refinement process completely independent of an initial input map, (2) more powerful lane clustering algorithms that can handle lane splits and merges, and (3) an approach to inferring detailed intersection models.

2. Steps in the map refinement process

Commercially available digital maps are usually represented as graphs, where the nodes represent intersections and the edges are unbroken road *segments* that connect the intersections. Each segment has a unique identifier and additional associated attributes, such as the number of *shape points* that approximate its geometry roughly, the road type (e.g., highway, on/off-ramp, city street), name, speed information, and the like. Generally, no information about the number of lanes is provided. The usual representation for a a two-way road is by means of a single segment. However, we depart from this convention and view segments as unidirectional links, essentially splitting those roads into two segments of opposite direction. This will facilitate the generation of precise geometry.

A *trace* is a trajectory generated by the motion of a single vehicle, as recorded by a GPS receiver. Each vehicle may generate more than one trace, at different times. Over time, each road segment is covered by multiple vehicle traces, and the accumulated data forms the input to our map refinement approach.

Traces are divided into subsections that correspond to the road segments as described above, and the geometry of each individual segment is inferred separately. Each segment, in turn, is transformed into a subgraph structure that captures its lanes, which might include splits and merges. We assume that the lanes of a segment are mostly parallel. In contrast to commercial maps, we view an intersection as a structured region, rather than as a point. Within these regions, traces may diverge and follow more or less constrained trajectories that constitute transitions between individual lanes in adjacent segments.

In summary, the overall approach involves six successive, dependent processing stages:

1. Collect raw DGPS traces from vehicles as they drive along the roads. Currently, commercially available DGPS receivers output positions (given as longitude/latitude/altitude coordinates with respect to a reference ellipsoid) at a regular frequency between 0.1 and 1 Hz. Optionally, if available, gather measurements for the purpose of electronic safety systems (anti-lock brakes or electronic stability program), such as wheel speeds and accelerometers, integrating them into the positioning system through a *Kalman filter* (Harvey, 1990). In this case, Step 2 (filtering or smoothing) can be accomplished in the same procedure.

2. Filter and resample the traces to reduce the impact of DGPS noise and outliers. If no error estimates are available, some errors can be detected by additional indicators provided by the receiver, relating to satellite geometry and availability of the differential signal; one can infer other (*multipath*) errors from additional plausibility tests, such as knowledge about the vehicle's maximum acceleration. Resampling balances the bias of traces recorded at high sampling rates or at low speed. Details of the preprocessing are beyond the scope of the current paper, but they can be found in textbooks (e.g., (Parkinson et al., 1996).
3. Partition the raw traces into sequences of segments by matching them to an initial base map.[1] This might be a commercial digital map, such as that of Navigation Technologies, Inc. (NavTech, 1996). Since we are not constrained to a real-time scenario, it is useful to consider the *context* of sample points when matching them to the base map, rather than one point at a time. We implemented a map matching module that is based on a modified best-first path search algorithm based on Dijkstra's (1959), where the matching process compares the DGPS points to the map shape points and generates a cost that is a function of their positional distance and difference in heading. The output is a file which lists, for each trace, the traveled segment IDs, along with the starting time and duration on the segment, for the sequence with minimum total cost. A detailed description of map matching is beyond the scope of this paper.
4. For each segment, use spline fitting to generate a *road centerline* that captures the accurate geometry to serve as a reference line for lanes once they are found.
5. Within each segment, cluster the perpendicular offsets of sample points from the road centerline to identify the number and location of *lanes*.
6. Refine the *intersection geometry* and model lane transitions between adjacent segment boundaries.

In the sections that follow, we describe each of these steps in detail. We start with the segmentation process in Section 3, after which we discuss the generation of road centerlines in Section 4, lane finding in Section 5, and intersection modeling in Section 6. After this, Section 7 reports the results of experiments with the map refinement process on a known test track. In the closing section, we discuss the implications of our results and prospects for future research. We also provide a brief glossary of GPS-related terms in the appendix.

3. Map segmentation

The first Step of the map refinement process decomposes traces into a sequence of sections that correspond to road segments. To this end, an initial base map is needed for *map matching*. This can either be a commercially available map, such as that of Navigation Technologies, Inc. (1996), or one can infer the connectivity through a *spatial clustering algorithm*, as we will describe shortly.

These two approaches both have their respective advantages and disadvantages. The dependence on a commercial input map has the drawback that, due to its inaccuracies (Navigation Technologies advertises an accuracy of 15 meters), traces sometimes are incorrectly assigned to a nearby segment. We experienced this problem especially with highway on ramps, which can be close to the main lanes and have similar direction. As we discuss in Section 6, inaccurate node positions can produce partitions of road intersections in which one segment is "too short" and another is "too long", essentially embedding all turning lanes in that intersection within the latter segment. A further disadvantage is that roads omitted from the input map cannot be learned at all. One cannot process regions if no previous map exists or the map is too coarse.

On the other hand, using a commercial map as the initial baseline associates additional attributes with the segments, such as road classes, street names, posted speeds, and house numbers. Some of these could be inferred from traces by related algorithms on the basis of average speeds and lane numbers. For example, Pribe and Rogers (1999) describe an approach to learning traffic controls from GPS traces, whereas Handley et al. (1998) present a method for predicting travel time. However, obviously not all such information can be recovered independently. Moreover, using the commercial map used for segmentation will make the refined map more compatible and comparable with applications based on existing databases.

3.1. Road segment clustering

Now we are ready to describe an approach to *inferring* road segments from a set of GPS traces. This approach relieves us of the requirement that an existing map be provided. Our algorithm can be regarded as a *spatial clustering* procedure. This class of methods is often applied to recognition problems in image processing; for example, Doucette (2001) applies clustering to find roads in aerial images. In our case, the main subtasks are to identify common segments used by several traces, and to locate the branching points (intersections). We should use a procedure that exploits the contiguity and temporal order of the trace points in order to determine the connectivity graph. We divide the method into three stages: cluster seed location, seed aggregation into segments, and segment intersection identification.

3.1.1. Cluster seed location. To automatically identify road segments, we first search for *road cluster seeds*, which act as initial anchor points on segments that will be subsequently extended into the final road segments. It is sufficient to identify one seed per segment. Locating cluster seeds means finding a number of corresponding sample points on different traces that belong to the same longitudinal position on a road. Assume we have already identified a number of trace points belonging to the same cluster; from these, we can derive mean values for position and heading. In view of the later refinement Step described in Section 4, we can view such a *cluster center* as one point of the road centerline.

Based on the assumption of lane parallelism, we measure the distance between traces by computing their intersection point with a line through the cluster center that runs perpendicular to the cluster heading; this is equivalent to finding those points on the traces whose projection onto the tangent of the cluster coincides with the cluster location, as in figure 1.

Our similarity measure between a new candidate trace and an existing cluster is based both on its minimum distance to other member traces belonging to the cluster, computed as described above, and on the difference in heading. If both of these indicators are below

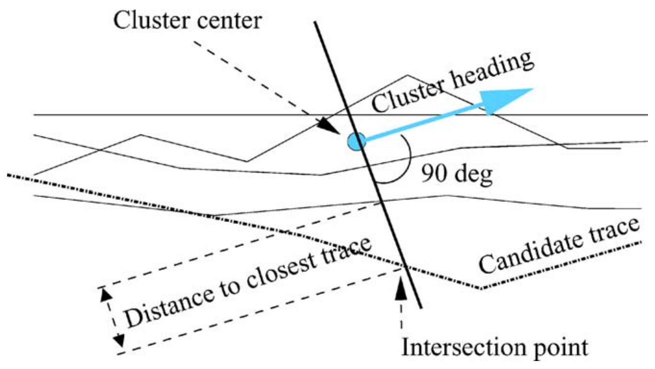

Figure 1. Distance between candidate trace and cluster seed.

suitable thresholds (call them θ and δ, respectively) for two sample points, they are deemed to belong to the same road segment.

The maximum heading difference δ should be chosen according to the accuracy of the data, such as to exclude sample points significantly above a high quantile of the error distribution. If such an estimate is unavailable, but a number of traces have already been found to belong to the cluster, the standard deviation of these members can give a clue. In general, we found that the algorithm's behavior is not very sensitive to variations in δ.

The choice of θ, the maximum allowed distance from a candidate trace to the traces already present, introduces a trade-off between two kinds of segmentation errors: if it is too small, wide lanes will be regarded as different roads; in the opposite case, nearby roads would be identified as the same. In any case, the probability that the GPS error exceeds the difference between the distance to the nearest road and the lane width is a lower bound for the segmentation error. A suitable value depends on the expected GPS errors, characteristics of the map (e.g., the relative frequencies of four-way intersections vs. freeway ramps), and on the relative risks associated with both types of errors that are ultimately determined by the final application. As a conservative lower bound, θ should be at least as large as the maximum lane width, plus a tolerance (estimated standard deviation) for driving off the center of the lane, plus a considerable fraction of an estimated standard deviation of the GPS error. Empirically, we found the results with values in the range of 10–20 meters to be satisfying and sufficiently stable.

Using this similarity measure, the algorithm proceeds in a fashion similar to the k-means algorithm (MacQueen, 1967). First, it initializes the cluster with some random trace point. At each step, it adds the closest point on any of the traces not already in the cluster, unless θ or δ is exceeded. Then, it recomputes the average position and heading of the cluster center. Due to these changes, it can sometimes occur that trace points previously contained in the cluster no longer satisfy the conditions for being on the same road, in which case they are removed. This process is repeated until no more points can be added.

In this manner, the method repeatedly generates cluster seeds at different locations, until each trace point has at least one of them within reach of a maximum distance threshold $d_{\max}$. This threshold should be such that it does not miss any intersection (say, 50 meters).

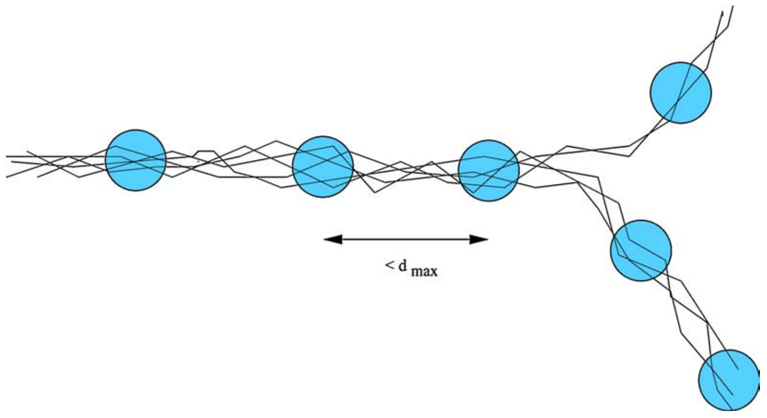

Figure 2. Example of traces with cluster seeds.

A simple greedy strategy would follow each trace and add a new cluster seed at regular intervals of length $d_{\max}$ when needed. Figure 2 shows an example section of traces, together with the generated cluster centers.

3.1.2. Segment merging. The next Step is to *merge* instances of the previously obtained cluster centers that belong to the same road segment. Based on the connectivity of the traces, our criterion for two clusters C_1 and C_2, where C_1 precedes C_2, is that all the traces belonging to C_1 subsequently pass through C_2, and all traces that belong to C_2 originate from C_1. All possible adjacent clusters that satisfy this condition are merged. We denote a resulting maximum chain of clusters $C_1, C_2, \ldots, C_n$ as a *segment*, and we refer to C_1 and C_n as the segment's *boundary clusters*.

At this stage of the map refinement process, only a crude geometric representation of the segment is sufficient; its precise shape will be derived later in the centerline generation. Hence, as an approximation, adjacent cluster centers can either be joined by straight lines, polynomials, or a representative part of the trace (as used in our algorithms). Figure 3 illustrates an approach in which lines connect the merged segments.

3.1.3. Intersections. The only remaining problem is to represent intersections. To capture the extent of an intersection more precisely, the method first tries to advance the boundary clusters in the direction of the split or merge zone. This can be done by selecting a point from each member trace at the same (short) travel distance away from the respective sample point belonging to the cluster, and then again testing for contiguity as described above. The algorithm extends the segment iteratively in small increments until the test fails.

The set of adjacent segments of an intersection is the union of all incoming segments that connect to an outgoing segment in one of the member boundary clusters. Each adjacent segment should be joined to each other segment for which connecting traces exist.

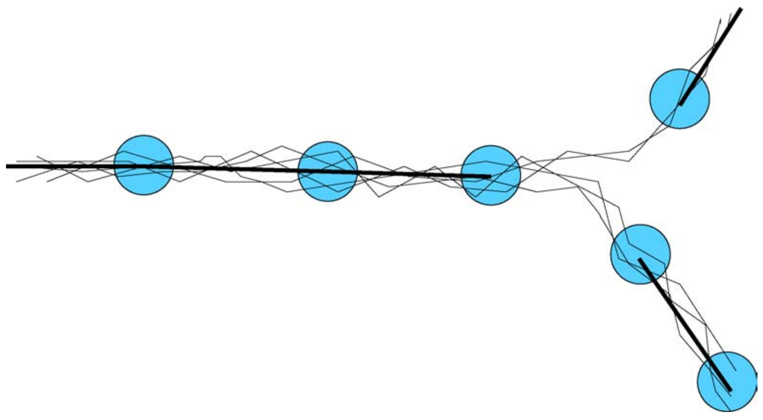

Figure 3. Merged cluster seeds result in segments.

If we adhere to the intersection region model as described later in Section 6, it would suffice to compute a common approximation of the path of connecting traces between each pair of boundary clusters (if they exist), and then to regard the intersection as the collection of all these transitions. However, a number of existing applications prefer a more compact but less accurate map representation which uses a unique *intersection point* that connects all adjacent segments simultaneously. Hence, for compatibility, our method also computes an approximation for the intersection point. After this, it merges all segments with their adjacent approximations up to the intersection point.

We borrow the concept of a *snake*—a contour model that is fit to (noisy) sample points—from the domain of image processing (Kass et al., 1988). In our case, a simple star-shaped contour suffices, with the end points held fixed at the boundary cluster centers of the adjacent segments. Conceptually, each sample points exert an attractive force on it closest edge. Without any prior information on the shape of the intersection, we can define the 'energy' to be the sum of the squared distances between each sample point and the closest point on any of the edges, and then iteratively move the center point, as in k-means, in order to minimize this measure. The dotted lines in figure 4 correspond to the resulting snake for our example.

3.2. *Dealing with noisy data*

Gaps in the GPS receiver signal can be an error source for the road clustering algorithm. Due to obstructions, it is not unusual to find gaps in the data that span a minute. As a result, interpolation between distant points is not reliable, so we need other solutions to this problem.

As mentioned above, checking for parallel traces depends crucially on *heading* information. For certain types of positioning systems used to collect the data, the heading might have been determined from the direction of differences between successive sample points.

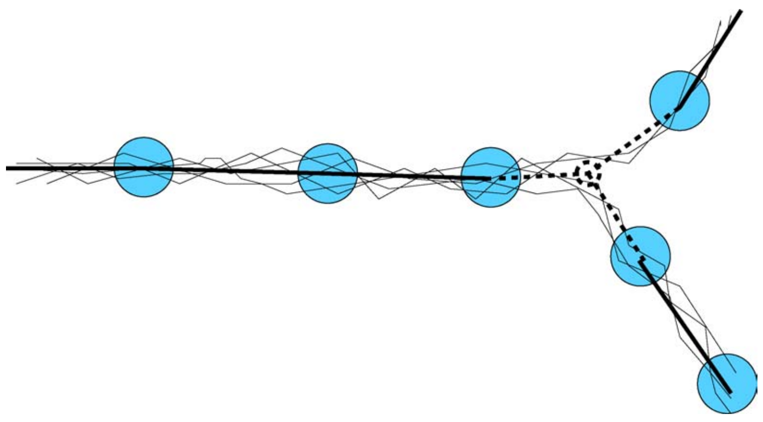

Figure 4. Traces, segments, and intersection contour model (dotted).

In this case, individual outliers, as well as lower speeds, can lead to even larger errors in direction.

Therefore, the first stage of our segmentation inference algorithm performs filtering by disregarding trace segments in cluster seeds that have a gap of more than $d_{\max}$ or that fall outside a 95 percent confidence interval in the heading or lateral offset from the cluster center. Cluster centers are recomputed only from the remaining traces, and only they contribute to the subsequent steps of merging and intersection location with adjacent segments.

Another issue concerns the start and end parts of traces. Considering them in the map segmentation could, for example, introduce segment boundaries at each parking lot entrance. To avoid too detailed a breakup, our method disregards initial and final sections. Different heuristics are possible, but the system currently uses a threshold that combines minimum trip length with speed.

4. Road centerline generation

We now turn our attention to the refinement of individual segments. The *road centerline* is a geometric construct whose purpose is to capture the road geometry. The centerline can be viewed as a weighted average trace, hence subject to relative lane occupancies and not necessarily a trajectory that any specific vehicle would ever follow. However, we assume that lanes are parallel to the road centerline at a (constant) perpendicular offset. For the subsequent lane clustering, the road centerline helps cancel out the effects of curved roads.

For illustration, figure 5 shows a section of a segment in our test area, along with sample points that stem from different traces. Clearly, by comparison, the shape points of the NavTech segment exhibit a systematic error. The figure also shows the centerline derived from the sample points.

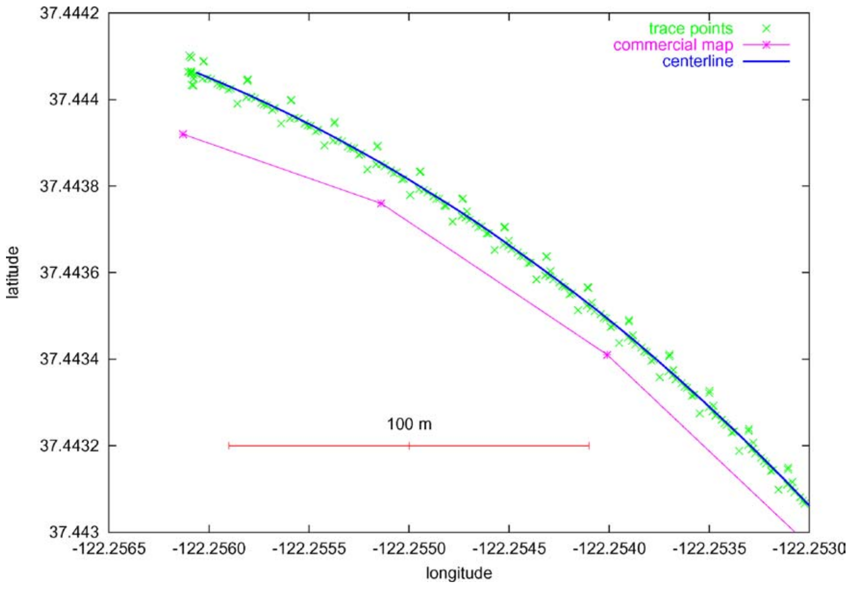

Figure 5. Portion of a road segment: NavTech shape points (lower solid line), GPS trace points (crosses), estimated centerline (upper solid line) [blank].

4.1. *Representation of the optimization problem*

It is useful to represent curves in *parametric form*, that is, as a vector of coordinate variables $C(u) = (x, y, z)(u)$ that is a function of an independent parameter u, for $0 \leq u \leq 1$. The centerline is estimated from a set of sample points using a *weighted least squares fit* in which $Q_0, \dots, Q_{m-1}$ are the m data points given, $w_0, \dots, w_{m-1}$, are associated weights (dependent on an error estimate), and $\bar{u}_0, \dots, \bar{u}_{m-1'}$ are their respective parameter values. The task can be formulated as finding a parametric curve $C(u)$ from a class of functions $\mathcal{S}$ such that the Q_k are approximated in the weighted least square sense, so that

$$s := \sum_{k=0}^{m-1} w_k \cdot \|Q_k - C(\bar{u}_k)\|^2$$

is minimized with respect to $\mathcal{S}$, where $\|\cdot\|$ denotes the usual Euclidean distance. Optionally, in order to guarantee continuity across segments, the algorithm can easily be generalized to take into account derivatives; if heading information is available, we can use coordinate transformation to arrive at the desired derivative vectors.

Our class $\mathcal{S}$ of approximating functions is composed of non-uniform *B-Splines*, which are piecewise defined polynomials with continuity conditions at the joining knots, as described in detail by Piegl and Tiller (1997) and Schroedl et al. (2000). As we explain shortly, the polynomial's degree must be at least three to ensure continuous curvature. If each sample point is marked with an estimate of the measurement error (standard deviation), which is usually available from the receiver or a Kalman filter, then we can use its inverse to weight the point, since we want more accurate points to contribute more to the overall shape.

4.2. Choice of parameter values for trace points

For each sample point Q_k, one must choose a parameter value $\bar{u}_k$. This parameter vector affects the shape and parameterization of the spline. If we were given a single trace as input, we could apply the widely used *chord length* method, in which d is the total chord length $d = \sum_{k=1}^{m-1} |Q_k - Q_{k-1}|$, and one sets $\bar{u}_0 = 0$, $\bar{u}_{m-1} = 1$ and $\bar{u_k} = \bar{u}_{k-1} + \frac{|Q_k - Q_{k-1}|}{d}$ for $k = 1, \dots, m-2$. This gives a good parameterization, in the sense that it approximates a *uniform* parameterization proportional to the arc length.

For a set of k distinct traces, we must impose a common ordering on the combined set of points. To this end, we utilize the polyline of shape points from the original NavTech map segment, s, as an initial rough approximation. If no such map segment is available, one of the traces can serve as a rough baseline. Each sample point Q_k is *projected* onto s by finding the closest interpolated point on s and choosing $\bar{u}_k$ to be the chord length (cumulative length along this segment) up to the projected point, divided by the overall length of s. It is easy to see that, for the special case of a single trace identical to s, this procedure coincides with the chord length method.

4.3. Choice of the number of control points

The least squares procedure (Piegl and Tiller, 1997) expects the *number of control points n* as input, the choice of which turns out to be crucial in the calculation of the centerline. The control points define the shape of the spline, even though they need not lie on the spline themselves.

For a cubic spline, the number of control points can be chosen freely in the valid range $[4, m-1]$, where m is the number of sample points. Figure 6 shows the centerlines for one segment as computed for four, ten, and twenty control points. Note that a small number of control points may not capture the shape of the centerline sufficiently well ($n = 4$); on the other hand, too many degrees of freedom causes the result to “fit the error”. Also observe how the spacing of sample points influences the spline for the case $n = 20$. From the latter observation, we can derive an upper bound on the number of control points: it should not exceed the average number of sample points per trace, multiplied by a small factor, say $2 \cdot m/k$.

Although the appropriate number of control points can be easily estimated by human inspection, its formalization is not trivial. We found empirically that two measures are useful in the evaluation. The first is related to the *goodness of fit*. Averaging the absolute offsets of the sample points from the spline is a feasible approach for single-lane roads, but otherwise depends on the number and relative occupancies of lanes, and we do not expect this offset to be zero even in the ideal case. Intuitively, the centerline is supposed to stay roughly in the middle between all traces. If we project all sample points on the centerline, and imagine a fixed-length window moving along the centerline, then the average offset of all sample points whose projections fall into this window should be close to zero. Thus, we define the *approximation error* ϵ_{fit} as the average of these offsets over all windows.

The second measure checks for overfitting. As illustrated in figure 6, using many control points renders the centerline “wiggly” and increases the *curvature* (cf. Section 4.4) in that

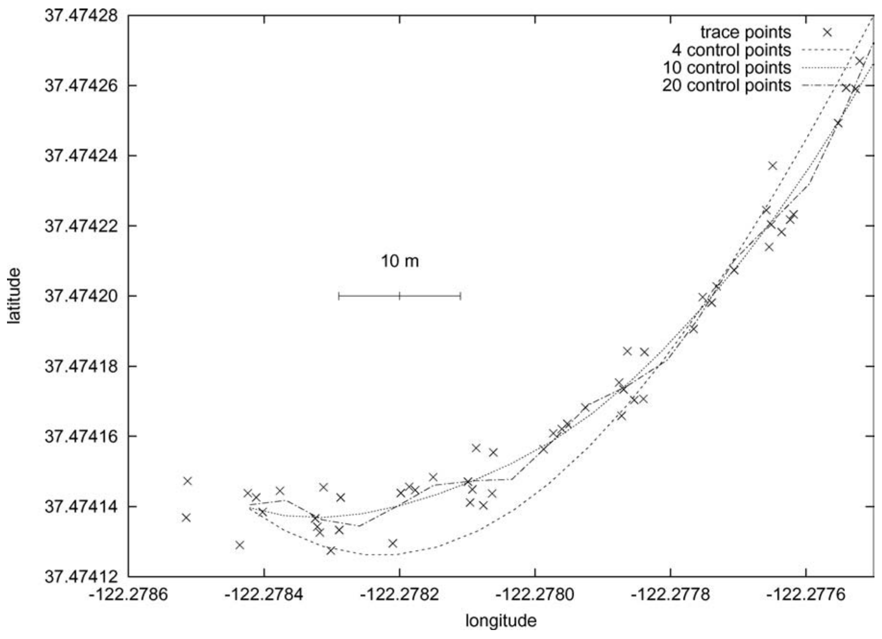

Figure 6. Trace points and centerlines computed with varying number of control points.

it changes direction frequently. However, according to construction guidelines, roads are designed to be piecewise segments of either straight lines or circles, with clothoids in between as transitions. These geometric concepts constrain the curvature to be *piecewise linear*. As a consequence, the second derivative of the curvature should be zero nearly everywhere, with the exception of the segment boundaries, where it might be singular. Thus, we evaluate the curvature of the spline at constant intervals and calculate the second derivative numerically. We refer to the average of these values as the *curvature error* ϵ_{curv}.

Figure 7 plots the respective values of ϵ_{fit} and ϵ_{curv} for the trace from figure 6 as a function of the number of control points. There is clearly a tradeoff between ϵ_{fit} and ϵ_{curv}, with the former decreasing rapidly as the latter increases. However, their relation is not completely monotonic. Searching the space of possible solutions exhaustively can be expensive, since a complete spline fit must be calculated in each step. To save computation time, the current approach heuristically picks the largest valid number of control points for which ϵ_{curv} lies below an acceptable threshold.

4.4. Computation of curvature

Any algorithm for centerline approximation should also estimate the road's *curvature*. This important quantity is directly proportional to the steering wheel angle necessary for driving on the road. The curvature at parameter u of a curve $C(u)$ is defined as the differential change in angle per unit length. Thus, it is defined as the limit of the inverse radius of

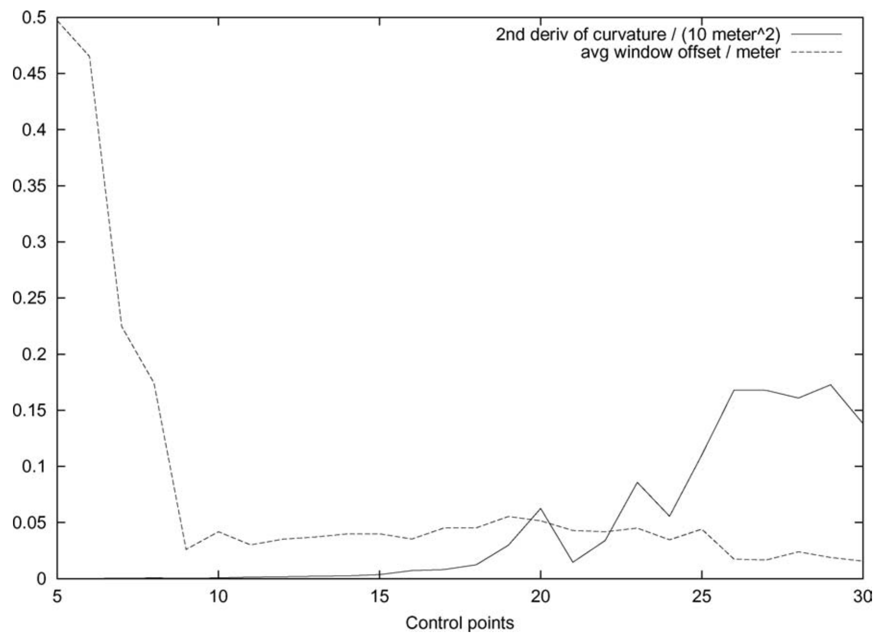

Figure 7. Error measures ϵ_{fit} and ϵ_{curv} vs. number of control points.

a circle passing through the three points at $u - \Delta u$, u, and $u + \Delta u$, as Δu approaches zero.

With our chosen spline fitting approach, the curvature $\kappa(u)$ can be conveniently calculated using the first and second derivative with respect to u. In our implementation, these values are readily available from the same procedure that evaluates the position of the spline. The relation is given as

$$\kappa(u) = \frac{\|C'(u) \times C''(u)\|}{\|C'(u)\|^3} ,$$

where $\times$ denotes the vector product and $\|\cdot\|$ denotes the usual Euclidean distance.

Usually, we assume that curvature is continuous across a road segment. According to the formula above, the second derivative must also be continuous. To achieve this at the knots on the spline, the polynomial must be at least third degree. When generating three-dimensional centerline models (including altitude), this curvature measure will also include the *vertical* component. For control applications, only the horizontal curvature is relevant. Thus, we must replace C' and C'' in the above formula by the projections $C' - (C' \cdot v)v$ and $C'' - (C'' \cdot v)v$, respectively, where v is a unit normal vector to the Earth surface at this point.

5. Lane finding

After computing the approximate geometric shape of a road in the form of the road centerline, the next processing Step infers the number and positions of its *lanes*. The task is simplified

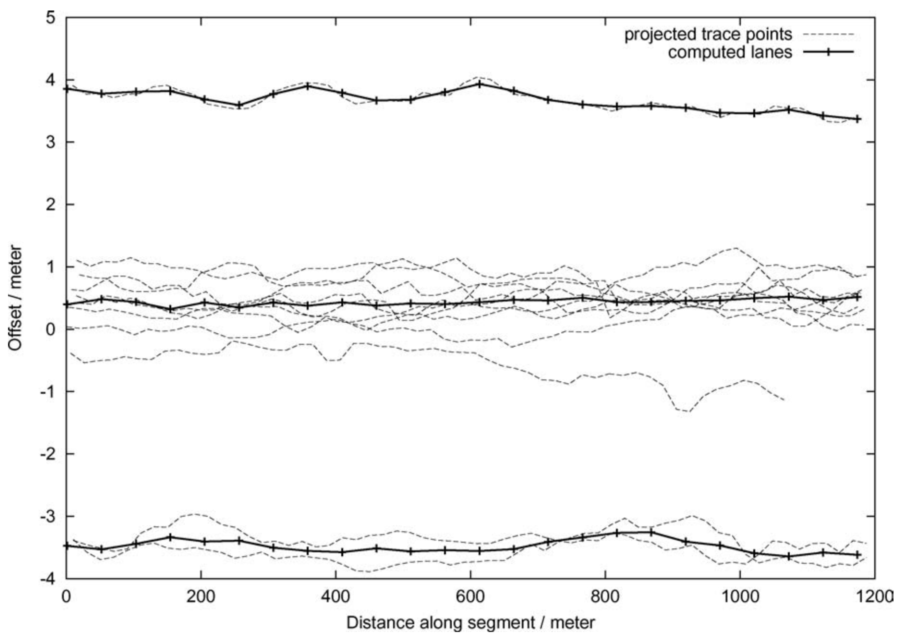

Figure 8. Traces projected on centerline (dashed) and computed lane centers (solid).

by canceling out road curvature by a transformation in which each trace point P is *projected* onto the centerline for the segment, i.e., its nearest interpolated point P' on the map. Again, the arc length from the first centerline point up to P' is the *distance along the segment*; the distance between P and P' is referred to as its *offset*. Figure 8 shows an example of the transformed data.

In general, *clustering* means assigning n data points in a d-dimensional space to k clusters such that some distance measure is minimized within a cluster (e.g., between pairs of data belonging to the same cluster or to a cluster center) and maximized between clusters. For the problem of lane finding, we are considering points in a plane that represent the flattened face of the earth, so the Euclidean distance measure is appropriate.

Since clustering in high-dimensional spaces is computationally expensive, methods like the *k-means* algorithm use a hill-climbing approach to find a (local) minimum solution. Initially, k cluster centers are selected, and two phases are repeated until cluster assignments converge. The first phase assigns all points to their nearest cluster center, whereas the second phase recomputes the cluster center based on the respective constituent points (e.g., by averaging) (MacQueen, 1967).

5.1. Segments with parallel lanes

If we assume that lanes are parallel over the entire segment, and that they never merge or split, then clustering is essentially one-dimensional, taking into account only the offset from the road centerline. In our previous approach (Rogers et al., 1999), we used a hierarchical agglomerative clustering algorithm that terminated when the two closest clusters were more than a given distance apart (which represented the maximum width of a lane). However, this algorithm requires $O(n^3)$ computation time. More recently, we have found that one can explicitly compute the *optimal* solution in $O(k \cdot n^2)$ time and $O(n)$ space using a dynamic programming approach, where n denotes the number of sample points and k denotes the maximum number of clusters (Schroedl et al., 2000).

5.2. *Segments with lane splits and merges*

Up to now, we have assumed the ideal case of a constant number of lanes at constant offsets from the road centerline along the whole segment. However, lane widths may gradually change; in fact, they usually get wider near an intersection. Moreover, new lanes can start at any point within a segment (e.g., turn lanes) and lanes can merge (e.g., on ramps). Here we present two algorithms that can accommodate the additional complexity introduced by lane merges and splits.

5.2.1. One-dimensional windowing with matching. One solution to this problem is to augment the one-dimensional algorithm with a windowing approach. We divide the segment into successive windows with centers at constant intervals along the segment. To minimize discontinuities, we use a Gaussian weighting to generate the windows. The points inside each window are clustered separately as described above, so that clustering remains essentially one-dimensional. If the number of lanes in two adjacent windows remains the same, we can associate them in the order given. For lane splits and merges, however, lanes in adjacent windows need to be *matched*.

To match lanes across window boundaries, we consider each trace individually, without using the information about which trace a data point belongs to in either centerline generation or lane clustering. Following the trajectory of a given trace through successive windows, we can classify its points according to the computed lanes. Accumulating these counts over all traces yields a matrix of *transition frequencies* for any pair of lanes from two successive windows. Each lane in a window is matched to that lane in the next window with maximum transition frequency.

5.2.2. Two-dimensional clustering with constraints. It is also possible to view lane finding as a two-dimensional problem by incorporating the distance along the segment as an additional feature for each point. Here, we used a k-means variant to cluster the points into k distinct lanes. Applied naively, the original k-means algorithm (MacQueen, 1967) does not perform well on this task. True lane clusters tend to be extremely elongated and, for the most part, oriented horizontally with respect to the centerline. In contrast, k-means by default seeks compact, spherical clusters. For example, figure 9 shows the output of the regular k-means algorithm for a sample road segment (entry 6 from Table 1). There are four true lanes. The points for each of the four clusters found by k-means are represented by different symbols. Clearly, these lanes do not correspond to the true lanes.

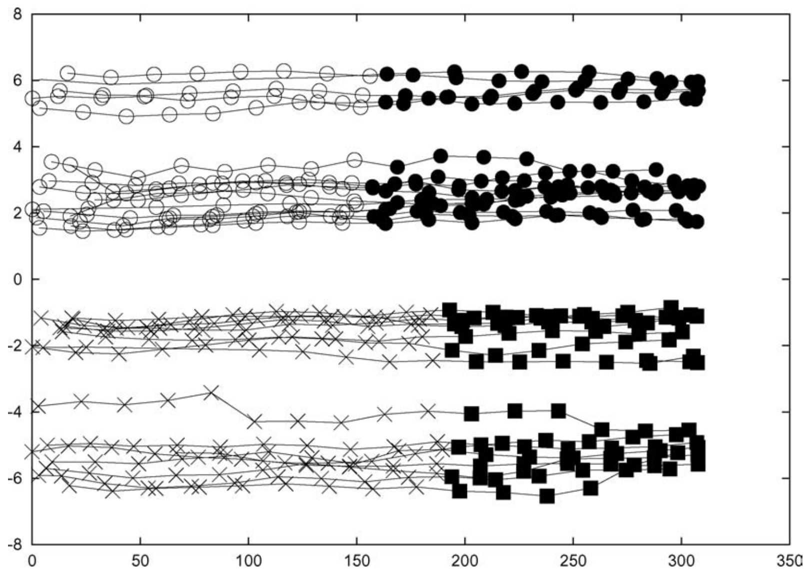

Figure 9. k-means output for data set 6, $k = 4$, with nearest clusters marked with different symbols.

Thus, we modified the cluster center representation to better reflect lane structure. The usual way to compute the center of a cluster is to average all of its constituent points, so that the center of a cluster is itself a point. The drawback of this representation is that the center of a lane is a point halfway along its extent, which commonly means that points inside the lane but at the far ends of the segment appear to be extremely far from the center. Consequently, we instead represented each lane cluster with a line parallel to the centerline. This more accurately models what we conceptualize as "the center of the lane" and provides a better basis for measuring the distance from a point to its lane cluster center.

Moreover, we need not blindly cluster the sample points according to the basic k-means algorithm; we can exploit domain knowledge about what a valid lane cluster looks like. More precisely, we have identified two important concepts that constrain how lane clusters can be formed:

- *Trace contiguity* means that, in the absence of lane changes, all of the points generated from the same vehicle in a single pass over a road segment should end up in the same lane. In other words, traces are contiguous in that all of the points from a single trace are linked.
- *Maximum separation* refers to a limit on how far apart two points can be (in terms of centerline offset) while still being in the same lane. If the difference in offset of two points is too large, then both points cannot be in the same lane.

These two general concepts can be used to generate a set of instance-level pairwise constraints. The trace contiguity heuristic generates a constraint for every pair of points that

Table 1. Lane finding performance (Rand Index).

Road segment	Windowing	COP-KMEANS
1	100	100
2	100	100
3	100	100
4	99.7	100
5	100	100
6	99.8	100
7	100	100
8	100	100
9	100	100
10	99.9	100
11	99.9	100
12	96.6	96.6
13	99.5	99.8
14	99.9	100
15	99.6	100
16	100	96.6
17	100	98.9
18	100	100
19	100	100
20	100	100
Average	**99.7**	**99.6**

come from the same trace and are not separated by a lane change. Each such *must-link* constraint requires that those two points end up in the same lane cluster.

In contrast, the maximum separation heuristic generates a *cannot-link* constraint for every pair of points that are separated by at least four meters (in centerline offset). These prevent those points from being placed in the same lane cluster.

We made use of the COP-KMEANS algorithm (Wagstaff et al., 2001) to incorporate these constraints into the clustering process. This algorithm is a variation of the generic k-means algorithm that can make use of instance-level pairwise constraints while clustering, which it does by modifying the data reassignment Step that reassigns each point to its nearest cluster. Instead of selecting the closest cluster, the algorithm chooses the closest *valid* cluster. A cluster is considered valid if assigning the point under consideration to it does not violate any must-link or cannot-link constraints.

5.3. Choosing the number of clusters

Most clustering algorithms, like the optimal clustering described above or the k-means algorithm, expect the number of clusters k as input. However, for the lane generation process we would like to have the most likely number of lanes automatically determined. This requires some method for comparing solutions that use different values of k.

Milligan and Cooper (1985) compared 30 such indices and found the *Calinski-Harabasz* measure to have the best performance on various synthetic data sets. However, it assumes at least two clusters and, in our experiments, it always preferred the solution with the most clusters. One possible reason is that, due to systematic position errors, distinct clusters are discernible inside a true lane.

In response, we developed another approach that uses constraints to select the number of clusters. For the *windowing* method, we experimented with estimating the road width from the total range of offsets, after reassessing possible outliers. This yielded acceptable results for city roads and highway segments. However, when the procedure is applied without prior knowledge of the road type, the distinction for more than three lanes becomes hard due to varying lane width. Thus, different parameters are required for each road type, which does not comply with our general approach. An additional challenge is the aggregation of data with different levels of noise, due to the use of different receivers. Such errors restrict the applicability of the criterion of the average standard deviation criterion per lane, too.

The current windowing approach successively generates clusters using different values for k ($k = 5 \ldots 1$), removing leftmost and rightmost lanes with low coverage (two or fewer traces). The first generated solution that satisfies a minimum distance requirement between lane centers is accepted.

On the other hand, to select the correct value of k, COP-KMEANS uses a measure that trades cluster cohesiveness against simplicity (number of clusters). More precisely, it calculates the average squared distance from each point to its assigned cluster center and penalizes for the complexity of the solution, using the equation

$$\frac{\Sigma_i\, dist(d_i, d_i.cluster)^2}{n} * k^2 .$$

The goal is to minimize the value of this expression. In our experiments, COP-KMEANS selected the correct k value for all but one road segment.

5.4. *Experimental comparison*

To compare the behavior of the windowing and COP-KMEANS algorithms, we evaluated their performance over 20 road segments. As part of this evaluation, we asked the drivers to indicate which lane they occupied and any lane changes, which let us label each data point with its correct lane. Table 1 presents accuracy results computed via the Rand Index (1971). In this experiment, the algorithms were required to select the best value of k from 1 to 5.

Both algorithms performed very well, exhibiting near-perfect performance on the lane clustering problem. The windowing algorithm achieved 99.7% average accuracy, while the COP-KMEANS algorithm averaged 99.6%. The difference is slight and appears largely due to an error COP-KMEANS made on road segment 16, where it decided there were three lanes rather than four. This road segment contains very few traces, which may have contributed to the difficulty, although the windowing algorithm achieved 100% accuracy on the same section.

We are encouraged by the high performance of both algorithms on these road segments. However, these data sets were all taken from highway roads, where lane merges and splits are rare. To further test the ability of these algorithms to handle such situations, our plans for future work include applying them to city roads, which are much less regular.

6. Refined intersections

Now that we have described computational tools that infer the lane structure of road segments, we can turn to the remaining part of the map refinement process. This deals with modeling the connections among segments in terms of *intersections*.

Currently available commercial digital maps usually represent an intersection as a node connecting all adjacent segments, that is, a point feature. However, the actual structure of an intersection contains much more information (cf. figure 10): a number of straight and turn lanes may exist for each possible direction, some of which may only start near the junction. Traffic rules can restrict the legal possibilities for each lane, and divides may separate lanes physically. There may not be any lane markings on the road in the range of the intersection, and sometimes the notion of a lane is not even appropriate for an intersection. Nevertheless, we can observe actual driving trajectories between segments. Although not being lanes in the narrow sense, these *transition paths* can be an important data source for automated control applications.

As already briefly mentioned in Section 2, we regard a map as being subdivided into two classes of regions: interior *segments*, where lanes and traces are roughly parallel, and

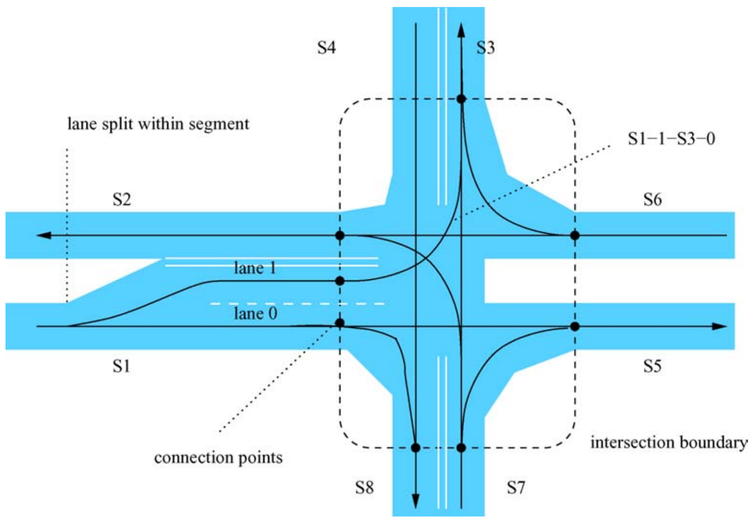

Figure 10. Schematic intersection model.

intersection regions, where adjacent segments are connected by arbitrarily shaped paths. In the following section, we describe a method that infers such a representation when a commercial map was used for segmentation. Next, in Section 6.2, we outline an approach that views the latter class as a collection of transitions between all legal pairs of lanes in adjacent segments. Finally, in Section 6.3 we present results for an example intersection from a test area.

6.1. Detecting segment-intersection boundaries

The first subtask consists in identifying the boundaries between segments and intersections. If the segmentation of traces has been carried out using our spatial clustering algorithm of Section 3.1, there is nothing left to do: the intersection regions consist of the spaces between boundary clusters of adjacent segments.

Instead, we assume here that the trace segmentation is based on an external input map. Since the size of the intersection is zero, all data points, including those belonging to turning lanes, are assigned to some segment. As depicted in figure 11, inaccurate node positions result in a segmentation of traces in which one participating segment is "too short" and the other one is "too long", essentially mapping all turning lanes within the intersection to the latter one. As a consequence, the centerline of the segment will be inaccurate and curvy.

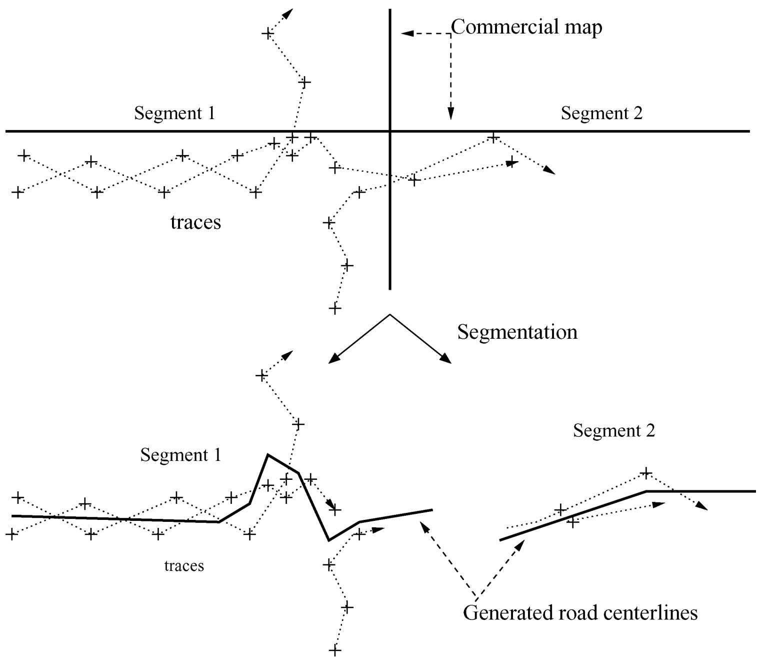

Figure 11. Inaccurate node position in the input map and resulting segmentation. Due to turning lanes, the centerline of segment 1 is not straight.

Thus, our map refinement system must search for the boundaries. To determine if a number of trace points belong to the segment, we can apply essentially the same contiguity test as in Section 3.1.1, which is based on the distance between traces and their difference in heading. However, the computation is significantly simplified, since the previous stages have already computed the road centerline (comprising all the cluster centers) and the projections of the sample points onto it.

The resulting modification is straightforward. Starting from both ends of a segment, the method repeatedly steps through the segment along the centerline (cf. figure 8 on p. 17) in small increments. At each distance, it inspects the interpolated perpendicular trace offsets from the road centerline. If 95 percent or more of these offsets are transitively reachable from zero (i.e., the cluster center) using steps to neighboring traces that are no larger than the threshold defined in Section 3.1.1, and at the same time have a sufficiently similar heading (according to the same criterion), then it assumes that the segment boundary has been reached and stops.

The lane offsets of the segment at the intersection boundary, as computed by the clustering algorithm, yield the *connection points* between the segment and the intersection region (cf. figure 10). When computing intersection paths, the system imposes these positions and associated headings as end-point constraints in order to guarantee smooth transitions.

6.2. Classification of transition paths

Once the system has finished detecting segment-intersection boundaries, it identifies feasible transitions and groups input traces accordingly. Usually the two possible driving directions are represented by separate segments. The system subdivides each of these enter-exit combinations further according to the structure of parallel lanes. For example, segment S_1 in figure 10 gives rise to the transition submatrix

	S2-L0	S3-L0	S5-L0	S8-L0
S1-L0	X	–	X	X
S1-L1	U-turn	X	–	–

where the symbol "*X*" denotes a legal transition and where "–" denotes an illegal one.

Most intersections are *T*-shaped or cross-shaped (as in figure 10). Without one-way restrictions, there are six segments involved in a *T* intersection and eight in a cross intersection. At most three different turning options exist for each direction, or four if we allow for U-turns, which makes 12 ways to leave one segment and entering the next for *T* shapes and 16 for cross shapes. Of course, turning restrictions may apply for a given intersection, but the system can check the number of times each turn has occurred in the matrix and infer these prohibitions from zero counts.

Each of these intersection combinations is further subdivided according to the lane structure. The system lumps together the sections of all the traces that leave segment S_1 on lane

l_1 and successively entering segment S_2 on lane l_2, for all lanes l_1 and l_2 present at the intersection boundary of S_1 and S_2, respectively.

When turning at intersections, drivers sometimes "cut corners", in that they stay temporarily in one lane in order to end up in an adjacent one. Traces of such maneuvers could impair inference of the intersection geometry. Thus, in order to compute a reliable lane classification of a trace that enters or leaves segment S, the system identifies the i-th data point P_i of the trace for which the projection d onto (distance along) the centerline is closest to the connection point (i.e., the segment-intersection boundary). Recall that the clustering algorithm produces a set of lane structure descriptors at regular window intervals along the segment, each of which comprises a list of lane offsets from the centerline. The algorithm interpolates between these lane offsets at the window positions enclosing d, assuming that the car used the lane l which is nearest to P_i.

However, classifying this point alone is not reliable, as there may be ambiguity for traces that cut lanes; for instance, the driver may have arrived in the leftmost lane but immediately shifted to the right. Therefore, the system tries to confirm the classification by successively testing trace points P_{i-1}, $P_{i-2}, \ldots$ and P_{i+1}, $P_{i+2}, \ldots$ until either a lane change occurs, the end of the trace or lane is reached, or the lane disappears due to a merge or split. The system only retains the trace as a training example for the transition path if the region extended in this way exceeds some minimum length. Depending on the sampling rate of the receiver and the speed of the car, there may be few data points. Moreover, short turning lanes may only include one window. These cases may rule out a number of traces.

Once the system has grouped the traces according to the transition matrix, it can easily infer the path by applying the same spline fitting algorithm as for road centerline generation in Section 4. However, it must constrain the end points of paths to match their respective lane connection points. The set of all trajectories that result for a given intersection constitutes the corresponding model of that intersection.

6.3. An example intersection

Now that we have described the intersection refinement method, let us illustrate it with an example from our test area. Figure 12 shows the intersection of Page Mill Road and Hanover Street in Palo Alto, California. We collected five traces using a code differential GPS receiver at a sampling rate of 1 Hz. No additional training data besides the trace points shown in the diagram were used for map making.

From the figure, one can see that the horizontal segments in the NavTech map exhibit an offset of approximately 15 meters from the true road centerline. As a consequence, since the map matching process uses this map for breaking up the traces into segments, the resulting centerline endpoints of the lower vertical segments lie on the opposite side of the intersection. Notice also that the centerlines sway to the sides, which comes from traces of drivers turning left or right.

Figure 13 depicts the lane offsets computed by the clustering algorithm, in addition to the centerlines. For this run, we set the interval between interior sample windows along the centerline to 15 meters. All first and last offsets lie on the respective segment-intersection boundaries and represent the connection points. Notice that this boundary is computed

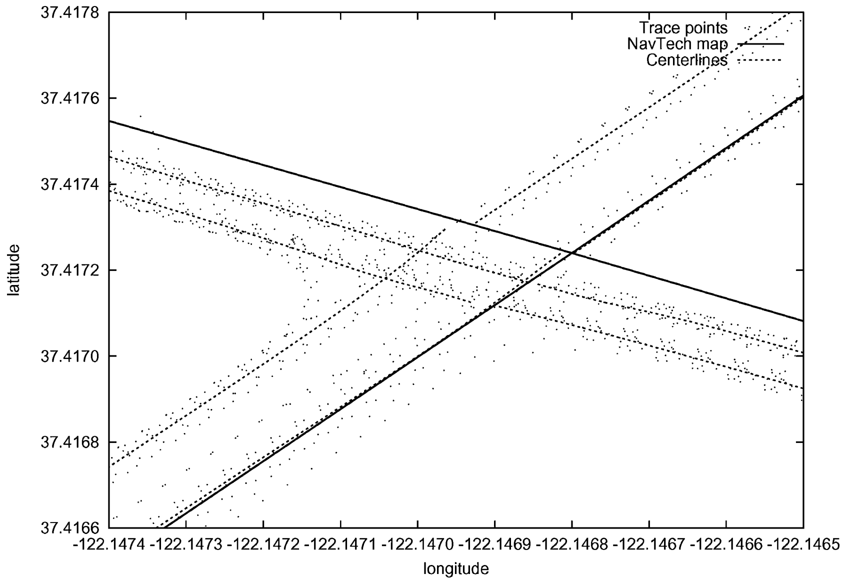

Figure 12. NavTech map (solid), trace points (dots), and generated centerlines (dashed) for adjacent road segments, without intersection modeling; centerline ends confounded by turn lanes, gaps due to segmentation inaccuracy.

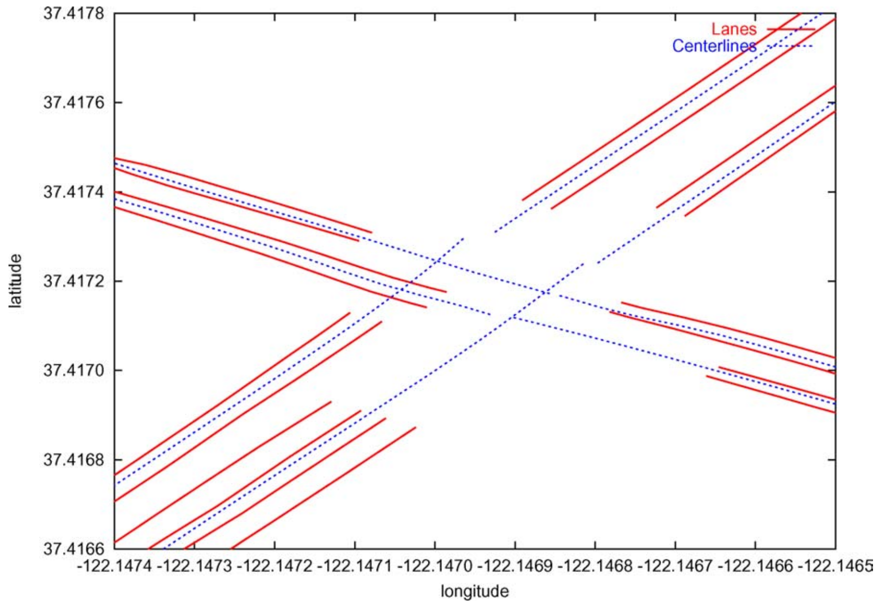

Figure 13. Road centerlines (dashed) and associated lanes (solid) generated by the clustering algorithm.

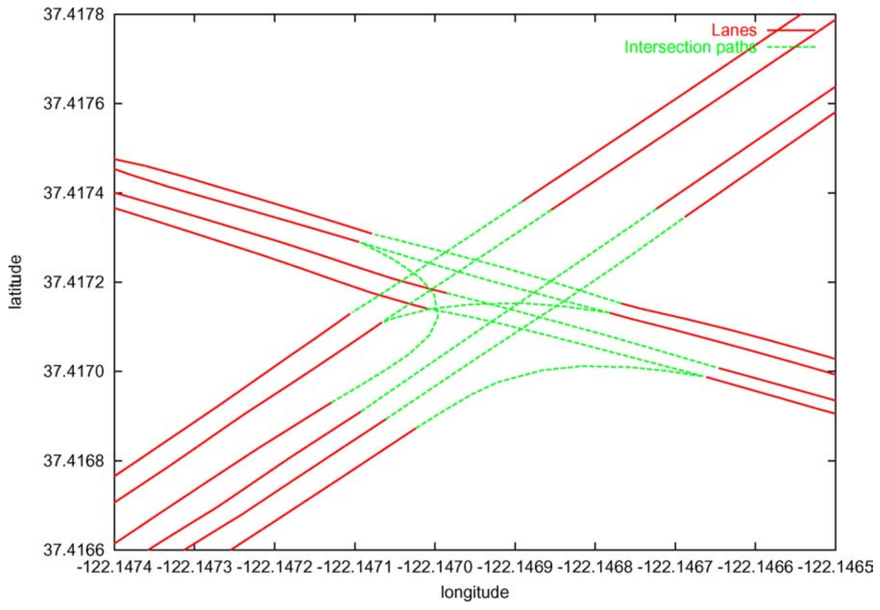

Figure 14. Segment lanes (solid) and connecting intersection paths (dotted).

separately for each segment, so that it cuts the segment earlier when traces start to diverge at a greater distance from the intersection. For example, this holds for the segment approaching the intersection from the lower left corner of the diagram, as compared to the opposite road side.

Finally, figure 14 illustrates the intersection paths that model the transitions between segments. The traces comprise two left-turn lanes and a right turn. In the latter case, most vehicles entered a bike lane immediately before turning. The straight continuations are handled in the same way as turning lanes. Notice that by grouping the input traces according to their destinations, these transitions reflect the actually driven paths more accurately and do not suffer from confounding the centerlines near the intersection with turning traces, as in the case of figure 12. In summary, our method can infer detailed and accurate descriptions of intersections and relate them to the road segments they connect.

7. Experimental results

We have now completed our account of the individual steps in the map refinement process. In this section, we report on experiments designed to evaluate the system's behavior, focusing especially on its rate of learning and its robustness in the presence of noise. Our test area in Palo Alto, California, covered 66 segments with a combined length of approximately 20 km of urban and freeway roads of up to four lanes, with an average of 2.44 lanes.

One principal problem we faced was the lack of a ground truth map with lane-level accuracy for comparison. In response, we used a high-end real-time kinematic carrier phase

DGPS system to generate a base map from a few traces (Wang et al., 2001). Based on the advertised accuracy (about 5 cm) and visual inspection of the map, we decided to define the map obtained in this manner as our baseline. Specifically, the input consisted of 23 traces generated at different sampling rates between 0.25 and 1 Hz.

Subsequently, we artificially created more traces of lower quality by adding varying amounts of Gaussian noise to each individual sample position ($\sigma = 0.5 \ldots 2\ m$) of copies of the original traces. For each combination of error level and training size, we generated a lane level map as described above, and measured its accuracy.

Figure 15 shows the resulting error in the *number of lanes*, that is, the proportion of instances in which the number of lanes in the learned map differs from that in the base-line map at the same position. The learning rate is very rapid for the two lowest noise conditions, with error reaching the asymptote (five percent) before the system has processed 50 traces, and dropping below ten percent by 100 traces for the third noise level. At the maximum noise level $\sigma = 2\ m$, the error is substantially higher, but, obviously, when the noise is greater than half a lane width, traces from different lanes will overlap and it is difficult to distinguish clusters. However, based on the total spread of the traces, the number of lanes can still be estimated. These remaining differences arise mainly from splits and merges, where in absence of the knowledge of lane markings it is hard to determine the exact branch points, whose position can heavily depend on single lane changes in traces.

Figure 16 plots the mean absolute difference of the offsets for corresponding lanes in the learned map and the base map as a function of the number of traces. Again, the condition $\sigma = 2\,m$ requires substantially more training data to achieve low error. For $\sigma <= 1.5\,m$, the lane offset error decreases rapidly; it is smaller than 15 centimeters after $n = 92$ traces, and

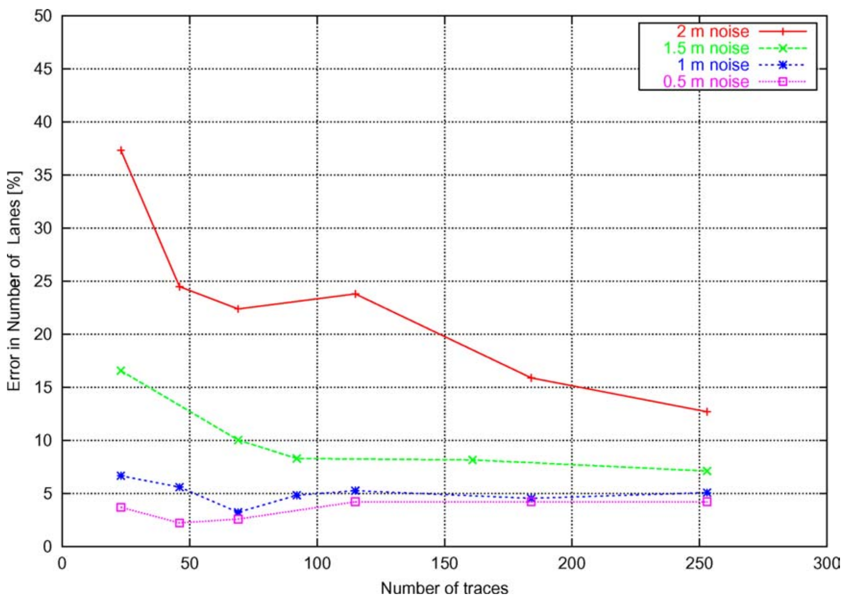

Figure 15. Error in determining the number of lane clusters.

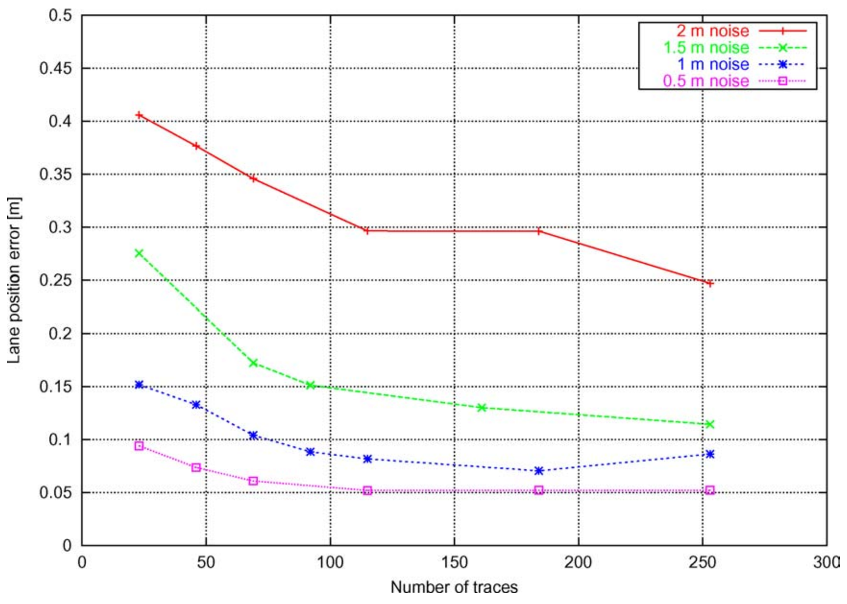

Figure 16. Average error in lane offsets.

thus in the range of the base map's accuracy. In summary, the experiments suggest that our approach to map refinement can learn both the number of lanes and their positions, requiring few training traces when noise is low but still behaving reasonably when noise is higher.

8. Concluding remarks

In this paper, we have described an approach to the automatic induction of high-precision road maps from global positioning data obtained from probe vehicles. With the decreasing cost of positioning systems and advances in wireless communication technology, we expect that such data will be available inexpensively in the near future.

Our approach segments traces into road segments and intersections. This *partitioning* can either make use of a commercially available map or be inferred by a trace clustering algorithm. The latter method builds maps from scratch, as required in areas where no previous digital maps exist or where they are coarse grained or outdated.

The method estimates a *road centerline* for each segment using a weighted least-squares spline that represents its geometrical shape. Lane positions are determined by a clustering algorithm. We have developed two such methods, one that uses a moving window across the segment and another that uses domain-specific constraints. In contrast to previous work, both algorithms can detect lane splits and merges.

The system determines the extent of *intersection regions* by detecting the divergence of trace transitions between segments. It then constructs an *intersection model* that includes a set of separate paths for each possible pair of entrance and exit lanes, which it infers using a spline-fitting technique similar to that for road centerline generation.

We might increase the accuracy of the inferred maps through an iterative refinement approach. For example, after having computed initial road centerlines, the system could repeat the process, using them as better approximations than the original ones for inferring the spline parameter values (cf. Section 4). We could adopt a similar scheme to better characterize lane splits and merges. Having encountered them in a map generated on one run, the system would revert to the segmentation stage and split the affected segments at these occurrences. Essentially, this would require a fixed number of lanes along each segment, so that splits and merges would be represented as detailed "intersections" in the form of transition paths. We are currently experimenting with such wrapper schemes to further improve accuracy.

Obviously, our system cannot take into account the "real" boundaries of lanes due to markings, but it is based on observed driver behavior. If some location exhibits a consistent bias toward one side of a lane, the learned map will reflect this fact as well. Moreover, intersection transition models help to determine transition paths even if there are no lane markings at all. It depends on the final application to decide which one of these two aspects should be preferred. E.g., for automatic control applications, a smooth and "human-like" trajectory could be more desirable than an "ideal" one, while for lane departure warning, the actual lane marking should be regarded as the ground truth. For the latter preference, we are investigating algorithms that use additional data from a visual lane tracker in order to correct for driving behavior.

In summary, the methods we have described can both automatically construct digital road maps from GPS races and improve on initial maps when they are available. These abilities have great potential for increasing the accuracy and detail of future maps, while substantially reducing the cost of creating and maintaining them. However, incorporating other sources of data and additional processing should produce even better maps that can be used in a variety of control applications that enhance driver safety and convenience.

Appendix: Glossary

B-Spline: A sequence of piecewise Bézier polynomials joined together in a 'smooth' way. The B-spline $C(u)$ of degree p is defined on $u \in [0, 1]$ as $C(u) = \sum_{i=0}^{n} N_{i,p}(u)P_i$, where the P_i are called the control points, the $N_{i,p}$ are the *blending functions*, and u_k is a sequence of increasing real numbers in [0, 1] called the *break points*. B-splines have the property of *local control*, since each curve segment is only influenced by the p closest control points. More precisely, if $u \in [u_i, u_i + 1)$, the only non-vanishing blending functions are $N_{i-p,p}(u), \ldots, N_{i,p}(u)$. For details, see Piegl and Tiller (1997).

Coordinate systems: Usually, GPS receivers refer to the ellipsoidal earth model *WGS-84*. The Prime Meridian and the Equator are the reference planes used to define *longitude* and *latitude*, respectively. The geodetic *altitude* at a point is the distance from the reference ellipsoid to the point in a direction normal to the ellipsoid.

Our calculations are based on a transformation of WGS-84 to the three-dimensional Euclidean space of so-called *Earth Centered, Earth Fixed Cartesian* (*ECEF*) coordinates. The origin lies in the center of mass of the reference ellipsoid. The Z-axis points toward the North Pole. The X-axis is defined by the intersection of the plane defined by the prime meridian and the equatorial plane. The *Y*-axis completes a right handed orthogonal system by a plane 90 degrees east of the *X*-axis and its intersection with the equator. ECEF coordinates are independent of the earth shape.

DGPS: Differential GPS. The position error of a stationary receiver at a known surveyed position is broadcast within a nearby region, e.g., by radio. A moving receiver can then filter out some atmospheric errors by subtracting the stationary error.

GPS: Global Positioning System. A system for determining position on the earth's surface by comparing radio signals from several satellites. The GPS receiver needs data samples from at least four satellites, calculates the time taken for each satellite signal to reach the GPS receiver, and from the difference in time of reception, determines your location.

Heading (a.k.a. azimuth): The geographical compass direction of travel, as a clock-wise angle with north being zero.

Kalman Filter: A numerical algorithm to estimate the past, present, or future state of a parameter vector, such as a user's position and heading. It is based on (1) a time sequence of (possibly several different) measurements of the system behavior, e.g., GPS and inertial sensors; (2) a (linear) statistical model that characterizes the system and measurement errors; and (3) initial condition information.

Polyline: An ordered series $p_0, \ldots, p_n$ of two- or three-dimensional points in space connected by straight lines. p_0 and p_n are called the start point and end point, respectively. *The cumulative length* of p_i is defined as $\text{length}(p_k) = \sum_{i=1}^{i=k} \|p_i - p_{i-1}\|$, where $\|\cdot\|$ denotes the usual Euclidean distance. An *interpolated point* is any point of the p_i, or any point p' on a line segment (p_i, p_{i+1}). The cumulative length of an interpolated point is defined in the straightforward way, namely as $\text{length}(p') = \|p' - p_k\| + \sum_{i=1}^{i=k} \|p_i - p_{i-1}\|$.

Projection of a point p onto a polyline P. Let p' be the closest interpolated point on P (i.e., $\|p' - p\|$ minimal). Then the cumulative length of p' is called the *distance along the segment* of p, or the *parallel component*, and the distance $\|p' - p\|$ is called the *offset*, or *orthogonal component*. In fact, if p' does not coincide with any of the polyline points, then the line through p and p' is perpendicular to the line segment which contains p'.

Segment: Elementary road segment between two intersections; i.e., the map os regarded as a graph, where the edges are the segments, and the nodes are the intersections represented as points. While the term segment is often used to refer to both directions of travel, in this article we understand it unidirectionally; i.e., a two-way road is composed of two parallel segments of opposite direction. The geometry of a segment is often represented by a polyline; in this case, the points of the polyline are called *shape points*.

Trace: Contiguous sequence of time-stamped positions collected during a road trip.

Note

1. In Section 3.1, we present an alternative algorithm for inferring the network structure from a set of traces alone.

References

Ertico, 2002. NextMap project. http://www.ertico.com/activiti/projects/nextmap/home.htm.

Harvey, A.C. 1990. Forecasting, Structural Time Series Models and the Kalman Filter. Cambridge University Press.

Kass, M., Witkin, A., and Terzopoulos, D. 1988. Snakes: Active contour models. International Journal of Computer Vision, 321–331.

Lavoie, P. 1999. The Nurbs++ Library Reference. Ottawa, Canada, University of Ottawa.

MacQueen, J.B. 1967. Some methods for classification and analysis of multivariate observations. In Proceedings of the Fifth Symposium on Math, Statistics, and Probability, vol. 1, Berkeley, CA: University of California Press, pp. 281–297.

NavTech, 1996. Software Developer's Toolkit. Navigation Technologies, Sunnyvale, CA, 5.7.4 Solaris edition.

Parkinson, B.W., Spilker, J.J., Axelrad, P., and Enge, P. 1996. Global positioning system: Theory and applications. American Institute of Aeronautics and Astronautics.

Piegl, L. and Tiller, W. 1997. The Nurbs Book. Springer Verlag.

Rogers, S., Langley, P., and Wilson, C. 1999. Mining GPS data to augment road models. In Proceedings of the Fifth International Conference in Knowledge Discovery and Data Mining, San Diego, CA: ACM Press, pp. 104–113.

Schroedl, S., Rogers, S.O., and Wilson, C.K.H. 2000. Map refinement from GPS traces. Technical Report RTC Report Number 6/2000, DaimlerChrysler Research and Technology North America, Palo Alto, CA.

Wagstaff, K., Cardie, C., Rogers, S., and Schroedl, S. 2001. Constrained *k*-means clustering with background knowledge. In Proceedings of the Eighteenth International Conference on Machine Learning (ICML-2001), Morgan Kaufmann, San Francisco, CA: Williams College, Williamstown MA, pp. 577–584.

Wang, J., Rogers, S., Wilson, C., and Schroedl, S. 2001. Evaluation of a blended DGPS/DR system for precision map refinement. In Proceeedings of the ION Technical Meeting 2001, Long Beach, CA: Institute of Navigation.

Wilson, C.K.H., Rogers, S., and Weisenburger, S. 1998. The potential of precision maps in intelligent vehicles. In Proceedings of the IEEE International Conference on Intelligent Vehicles, Stuttgart, Germany, pp. 419–422.