Scalable, Distributed, Real-Time Map Generation

In this application, many vehicles participate in the collection, processing, and dissemination of data to automatically generate digital road maps.

Jonathan J. Davies, Alastair R. Beresford, and Andy Hopper
University of Cambridge

Modern vehicles have a plethora of onboard computing equipment. Today's cars have up to 50 microprocessors governing various aspects of their operation. We foresee a future in which vehicles' sensor, communication, and computational resources are harnessed to improve the transportation infrastructure and have a positive impact on society.

To start working toward that vision, we can exploit modern vehicles' computing facilities to provide a host of participative mobile applications. Such uses include vastly improved collection and dissemination of weather data and real-time measurement of road surface conditions. In particular, the sharing of movement data from vehicles might help facilitate city planning, improve fleet management, and enforce congestion charging (charging fees for driving in certain areas at certain times).

The map data in a vehicle's navigation unit forms the basis of the unit's routing decisions. This data is prone to cartographic errors and to inaccuracies due to recent changes in the road network, including temporary road closures. These problems can frustrate drivers when they cause the unit to give impossible driving instructions.

So, organizations that produce digital maps, such as Navteq and Tele Atlas, must invest significant effort in maintaining and improving their data's accuracy. Details about new roads and the modification or closure of existing roads must be incorporated into the databases in a timely fashion. Mapping organizations obtain data about road network changes from various sources, including local authorities and building contractors, but this data tends to be highly inaccurate. Aerial photographs can help mapmakers deduce roads' presence and shape but are prohibitively expensive to update frequently. Instead, mapping companies typically own fleets of probe vehicles that investigate discrepancies and explore new roads. Tele Atlas spends tens of millions of dollars each year in North America to keep its databases up-to-date, while in 2004 Navteq employed more than 500 analysts who drove a total of 3.5 million miles throughout North America and Europe.

To simplify this mapmaking process, vehicles could use their computational resources to directly generate accurate map data. We envisage vehicles on the road network forming a wide-scale mobile sensing and computing platform. Toward that goal, we've developed an algorithm for keeping digital maps up-to-date, using ordinary vehicles making normal journeys rather than fleets of dedicated probe vehicles. However, further development is required to address several remaining challenges, including the choice of architecture to support the algorithm.

Automatically generating digital maps

Information about a road network can be represented as a directed graph with metadata associated with its edges. Such a graph can be used to both render a graphical depiction of the road network and serve as an input to in-vehicle navigation systems. The graph's edges represent roads (or road lanes), and its vertices represent junctions.

1536-1268/06/$20.00 © 2006 IEEE ■ *Published by the IEEE CS and IEEE ComSoc*

Related Work in Map Generation

Discretizing the space covered by sensor readings into a grid of cells is an established practice. Robotics researchers use certainty grids[1] or occupancy grids[2] to store the probability that an obstacle exists in any particular cell in an environment that robots will explore.[3] Unlike that research, we don't have the luxury of being able to determine obstacles' presence. Robotics research also heavily employs Voronoi graphs,[4] particularly because they describe an environment's topological features, making them suitable for route planning.

Robert Harle investigated generating descriptions of an indoor environment purely from records of location fixes.[5] While it's unrealistic to expect exhaustive coverage of a typical indoor environment, we can expect vehicles to exhaust all available road space. This is because the road network imposes greater constraints on the users' movement.

REFERENCES

1. H.P. Moravec and A. Elfes, "High Resolution Maps from Wide Angle Sonar," *Proc. 1985 IEEE Int'l Conf. Robotics and Automation*, IEEE Press, 1985, pp. 116–121.
2. A. Elfes, "Using Occupancy Grids for Mobile Robot Perception and Navigation," *Computer*, June 1989, pp. 46–57.
3. S. Thrun, "Robotic Mapping: A Survey," *Exploring Artificial Intelligence in the New Millennium*, G. Lakemeyer and B. Nebel, eds., Morgan Kaufmann, 2002, pp. 1–36.
4. H. Choset and K. Nagatani, "Topological Simultaneous Localization and Mapping (SLAM): Toward Exact Localization without Explicit Localization," *IEEE Trans. Robotics and Automation*, vol. 17, no. 2, 2001, pp. 125–137.
5. R.K. Harle, "Maintaining World Models in Context-Aware Environments," doctoral thesis, Laboratory for Communications Eng., Dept. of Eng., Univ. of Cambridge, 2004.

Generating a directed graph

The application based on our algorithm transforms GPS traces from multiple vehicles into a road map. Processing involves four basic stages:

1. Generate a 2D histogram indicating the number of GPS fixes found in each cell.
2. Deduce the road edges' positions.
3. Compute the positions of the roads' centerlines.
4. Determine the direction of travel permitted along each road.

We now examine these stages in detail, using figure 1 as a running example.

A trace of location fixes obtained from a vehicle's GPS unit will show which roads the vehicle has traveled along. The trace will contain errors due to uncertainty in the location fixes and missed sightings when the GPS satellite signal is obscured. From a single trace, distinguishing between junctions and bends in roads is difficult. Moreover, the GPS readings' inherent errors might mean that the trace misrepresents the roads' true positions. However, if we superimpose the traces from several vehicles traveling along different routes, junctions will soon become apparent. Also, the roads' true positions will become clearer as noise due to errors becomes less significant.

Creating a histogram. Splitting up 2D space in the horizontal plane into *cells*—small, tessellating, square units of area—we wish to determine the likelihood that each cell is part of a road. The Nyquist-Shannon sampling theorem dictates that the cell width should be at most half the minimum road width to prevent aliasing. In practice, this equates to a few meters. A GPS reading falling in a cell is a good indication that the cell might be part of a road. So, if we associate with each cell the number of GPS points that fall in it, cells with higher frequencies will more likely be parts of a road. In this way, we group our 2D real-valued GPS fixes into discrete cells. (For a brief look at similar research and other research related to map generation, see the "Related Work in Map Generation" sidebar.)

At highway speeds, with GPS fixes obtained at 1 Hz, consecutive fixes will fall approximately 30 meters apart. Thus, with a cell size of a few meters, successive fixes won't lie in adjacent cells, leaving us with disjoint regions of road. But if the GPS fixes are temporally ordered, we know that road exists between consecutive fixes. So, we can also increment the value in the cells that are between those cells in which the fixes lie. If the frequency of readings is greater than a few Hertz, then linear interpolation between the GPS fixes will likely be acceptable. However, higher-order interpolation might yield more realistic results.

The value attached to each cell represents the confidence that the cell is part of a road. Our application increments a cell's value by an amount proportional to the length of the line that passes through the cell. In this way, if the line traverses only a cell's corner, then that cell's value is incremented by only a small amount.

After the application has processed all the available GPS fixes in this way, we have a 2D histogram that estimates the confidence of each cell constituting part of a road, based on all the journeys traced out in the data (see figure 1a).

Despite the interpolation between successive GPS fixes, gaps will likely exist because a cell with a low frequency might be surrounded by cells with high frequencies. This might be due to a random paucity of data collected in that cell; systematic errors intrinsic in the original GPS data; real-world features that

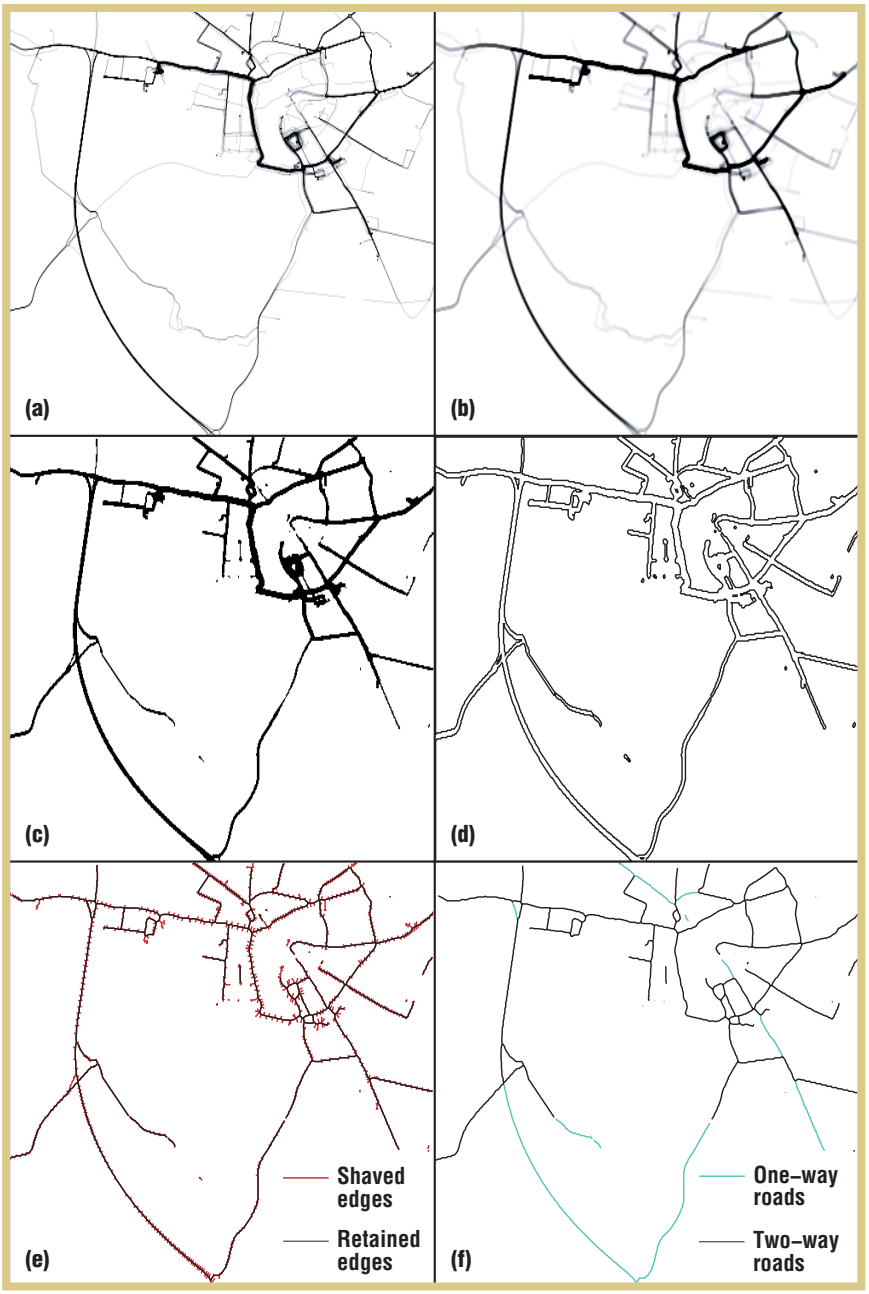

Figure 1. Generating a map of the city center of Cambridge, UK: (a) a histogram, with one pixel per cell, (b) a blurred histogram, (c) a thresholded histogram, (d) contours, (e) a Voronoi graph, and (f) a directed graph.

obstruct vehicles, such as lampposts or road barriers; or blackspots (areas without GPS coverage) such as under bridges. In any case, these gaps are undesirable and should be removed because we're merely aiming for a directed graph of the road network's topology where connectivity is important.

To remove such gaps in the histogram, we apply a blur filter. This removes small gaps because the filter averages cell values with neighboring cells, but larger gaps will persist. Similarly, the filter will smooth jagged edges. However, performing a blur might undesirably merge two nearby but distinct roads. Figure 1b depicts the data convolved with a three-by-three-cell uniform blur convolution filter.

Deducing edge positions. At this processing stage, we have a good idea of whether a road exists in any given cell. So, we binarize this metric to a Boolean value: is there a road or not? To do this, we apply a global threshold to our cells. The threshold's value will relate to the degree of confidence we want in the road network deduced from our algorithm. A lower threshold will be more susceptible to noise due to GPS errors. Figure 1c shows the result of thresholding the data.

After thresholding, we apply a contour follower[1] to the image. This extracts a set of closed polygons describing the road regions' outline (see figure 1d). This outline might not coincide precisely with the roads' real-life edges, owing to errors in the original GPS data. However, if these errors are symmetrically distributed, the centerline between the edges will coincide with the road's actual centerline.

Computing centerlines. A Voronoi graph is the set of points equidistant from the nearest two points on the boundaries of a set of closed polygons.[2] We can compute the road centerlines by producing the Voronoi graph of the contours describing the roads' edges and discarding the resulting edges that lie outside the roads.

However, because the roads' edges aren't convex, many short edges of the Voronoi graph will be attached to the main trunk of each road. This gives the graph a rather "hairy" appearance. These edges don't correspond to real-life roads; they're artifacts from our initial discretization of GPS fixes into square cells. We can remove the edges that are shorter than a threshold representing the minimum permitted road length, leaving just the main edges running the roads' lengths. Figure 1e depicts the Voronoi graph generated from the contours, with the short edges in red.

Determining road direction. The final stage is to deduce which edges of the

undirected Voronoi graph represent unidirectional roads and which represent bidirectional roads. If the original GPS data was temporally ordered, we can produce a further data structure that will help us determine this. Again splitting space into cells, in each cell we now keep track of the number of journeys embodied in the GPS traces that pass through the cell in each of the eight compass directions. We generate this by quantizing the bearing of the displacement vector between each successive pair of fixes and incrementing the count for that direction.

For each compass direction, we sum the counts of journeys associated with each cell on a road. Unidirectional roads have significantly more cars traveling parallel to the road in one direction than in the other. Figure 1f shows which edges the algorithm deduced as unidirectional and as bidirectional.

Complexity. The algorithm's overall time complexity is $O(n + m)$, where n is the number of GPS readings and m is the total number of cells. Individually, the stages involving the processing of GPS readings are $O(n)$ and the stages involving image processing are $O(m)$. However, if we can store the latest versions of the histogram and direction data, the time complexity of generating the new map incorporating an incremental set of GPS readings of size δn is merely $O(\delta n + m)$.

Reflecting road changes

We wanted our application to produce digital maps that reflect the creation of new roads, the closure of old roads, and the change in geometry of existing roads. Our algorithm makes it trivial to establish that a new road has been opened: it acquires GPS traces of vehicles using the road and regenerates the map to show the new road. However, we can't say that about the other two goals. Once we have some GPS traces traveling down a particular road, the digital maps that our basic application produces will always contain that road.

> For the map to be suitable for navigation, we need to associate metadata with each edge in the directed graph.

To reflect changes to existing roads over time, we must place lower trust in older data than in more recent data. This means that vehicles must continue to travel down roads regularly to maintain our level of trust in the roads' existence. To adjust the trust levels accordingly, we could adjust the values in the histogram by smaller increments when processing an older GPS trace than when processing a more recent one. So, when a road is closed, we would no longer receive GPS traces showing it in use. Eventually the histogram cells' value would fall below our binarization threshold, causing the road to disappear from the map. However, this implies that a certain latency would exist between when the road closes and when the map reflects the closure. We could reduce this delay if we can obtain more GPS traces from more vehicles.

Map regeneration

For the map to be suitable for navigation, we need to associate metadata with each edge in the directed graph. However, this task might be relatively expensive because it might involve manual effort. While metadata such as the speed limit could be inferred from the GPS data, metadata such as the road name could be determined only by visiting each road or, perhaps in the future, from active road signs. Because we want the application to generate up-to-date maps, we must execute the algorithm repetitively as new GPS traces come to light. However, we need to avoid the cost of reassociating the metadata from scratch every time we regenerate a map.

On the basis of the knowledge obtained from additional GPS traces, roads in the old version might change their shape, and junctions might shift position. Furthermore, roads—and thus junctions—might appear or disappear. This makes it difficult to determine which roads in the old version correspond to which roads in the present version, thus making it difficult to transfer the old roads' metadata to the new roads.

To estimate which roads in the old version correspond to which roads in the present version, and which roads exist only in one version, we can employ a *weighted bipartite graph*. A bipartite graph is a special type of undirected graph that splits vertices into two disjoint sets, with no edges between two vertices in the same set. A weighted bipartite graph has costs associated with its edges.

In this case, the two sets of vertices represent the roads in the old version and the roads in the present version. The edges have an associated cost relating to the similarity between a pair of roads from those sets. So, a low-cost edge between two vertices in the bipartite graph means that a road in the old version very closely corresponds to a road in the present version. The cost metric relates to the distance that the ends of the road network edges have moved between the two versions and to the similarity in the edges' shape.

Figures 2a and 2b show a map's old and new versions; figure 2c shows a bipartite graph constructed from the two versions.

A minimum-cost maximal matching on this bipartite graph will therefore indicate which edges in the old and present versions best correspond and will

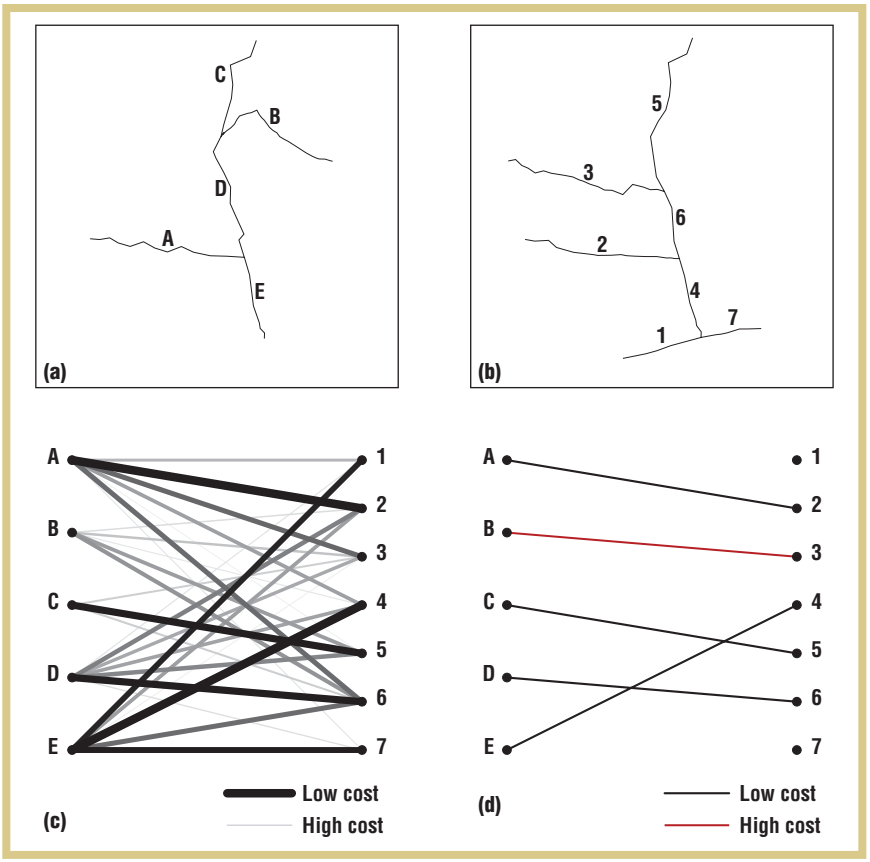

Figure 2. Estimating which roads in an old map correspond to which roads in a new map: (a) an old version of a map, (b) a new version of a map, (c) a weighted bipartite graph of the maps, and (d) a minimum-cost maximal matching on the bipartite graph.

contain high-cost edges between the remaining vertices. (A matching is a subset of the graph's edges with no vertices in common. A maximal matching employs as many edges as possible. A minimum-cost maximal matching minimizes the sum of its edges' costs.) The application can ignore these high-cost edges, which correspond to pairs of roads in one version of the map but not the other. For the remaining low-cost edges, the application can transfer metadata between the roads corresponding to the vertices. In figure 2d, the red edge represents a discarded high-cost edge. This matching indicates that road B has closed and roads 1, 3, and 7 are new.

Scalability

To process GPS traces from a vast number of vehicles covering a large geographical area, our algorithm must scale gracefully. Fortunately, it's highly parallelizable, by dividing up space into tessellating regions or *tiles*. (A tile comprises many cells and might cover several square kilometers.) Then, an individual processing node can use the GPS traces falling in a tile to produce a directed graph of the road network in that tile.

Once each processing node has produced its tile's graph, we need to stitch the results back together into one complete graph. However, we can't expect that roads spanning the tiles' edges will necessarily align and thus be contiguous when juxtaposed. This is because we can't be certain about the results produced near a tile's border when that tile has been processed in isolation.

To solve this problem, we process a set of tiles, each of which overlaps the adjacent tiles by the number of cells corresponding to the region of uncertainty. Then, to stitch together the resulting directed graphs from each tile, we simply clip the roads to the tile's central region. A road that originally traversed two adjacent tiles will now be contiguous, so we join into a single edge the edges that meet at the seam between their respective tiles. The ratio of the overlap region's area to that of the entire map corresponds to the parallelization overhead.

We've tested this technique successfully by partitioning the data used in figure 1 into four quadrants, mimicking separate processing nodes. Processing each quadrant individually and stitching the resulting graphs together produced the same output as processing all the data at once.

Possible system architectures

Many modern satellite navigation units are, in fact, small computers containing general-purpose processors and hard disks. Some (such as the TomTom GO) already use a GPRS (General Packet Radio Service) connection from an attached mobile telephone to download real-time traffic reports. Navigation units with Internet connectivity could support our mapping application by sharing GPS traces at regular intervals. Further in the future, vehicles will likely also carry IEEE 802.11 communications equipment, which could upload data when within range of a suitable access point.

A broad spectrum of system architectures could support map generation. We can view a network of vehicles and any infrastructure support as a set of nodes in a distributed-memory parallel computer. At one end of the spectrum is the fully centralized approach, where vehicles upload their raw GPS readings to a central server that generates new map data. Despite bringing benefits, particularly timely data delivery and homogeneous results, this approach will likely be impractical because of the communication bandwidth needed to transmit all the GPS readings to the server.

To distribute the communications

bandwidth, we can employ multiple servers, each processing data for a different geographical region. Commercial operators seeking to gather and process data cheaply might prefer this approach. Furthermore, to avoid requiring a costly backhaul network between regional servers, the vehicles themselves could distribute data across region boundaries.

> We can view a network of vehicles and any infrastructure support as a set of nodes in a distributed-memory parallel computer.

This is a topic of ongoing research in the CarTel project, which uses vehicles as high-bandwidth "data mules."[3]

An increasingly common solution is to execute some processing on the vehicles themselves. For example, in the Vehicle Data Stream Mining (VEDAS) project, vehicles process their own sensor data, uploading the results to a remote central server over a low-bandwidth wireless network.[4]

Another architecture is to provide public data caches that are connected to the Internet and store data but don't contain any processing facilities. In this scenario, the vehicles must acquire as many GPS traces as they can from the nearest public data cache (perhaps by IEEE 802.11 communications) and execute the map application locally. Consumers concerned about location privacy might prefer this approach; such a decentralized scheme can limit the transmission of personally identifiable data.

At the far end of the architecture spectrum is the fully peer-to-peer scenario in which vehicular ad hoc networks (VANETs) share GPS and map data. Although this solution won't have any ongoing service costs to customers, it's unlikely that it can gather sufficient data to produce up-to-date, reliable maps. VANETs will likely be more suited to small-scale cooperative driver assistance systems such as TrafficView[5] and those enabled by Network-on-Wheels.[6]

Performance limitations

Our application has several fundamental performance limitations.

GPS reading errors are typically modeled by a bivariate normal distribution. The standard deviation, σ, in the error of the position estimate for a modern GPS receiver has recently been estimated as 4.25 m, giving a 95 percent confidence interval of 8.5 m.[7] (Others have estimated the value to be 3.5 m.[8]) We believe that a reasonable minimum distance between two adjacent parallel roads should be 4σ. This means that no more than approximately 2.5 percent of the position estimates of vehicles at one road's edge can overlap with no more than approximately 2.5 percent of the position estimates of vehicles at the other road's nearest edge. If this limit isn't observed, the region between the roads might be filled with stray GPS fixes, causing no discernable gap between the roads after thresholding. For example, with $\sigma = 4$ m, the minimum road spacing tolerated is approximately 16 m.

The distribution of GPS readings for vehicles traveling in all lanes of a road won't be normal. However, we can approximate it using a multimodal distribution consisting of one normal distribution per lane. We assume that this distribution's underlying mean is on the road's centerline (roughly, that the traffic volume is evenly distributed about the centerline). By the central limit theorem, the error on the mean of the GPS readings, when compared with the road's actual centerline, will be normally distributed. Also, its standard deviation will decrease at a rate of $1/\sqrt{n}$ for an n-fold increase in the number of samples. So, with sufficient samples, GPS data becomes an accurate predictor of the real centerline.

More specifically, we can model the distribution of GPS samples collected from a two-lane road as a distribution $(N(\mu_1, \sigma^2) + N(\mu_2, \sigma^2))/2$, where μ_1 and μ_2 are the positions of the lanes' centerlines and the mean is the road's centerline. By the central limit theorem, with 73 samples, the estimate of the road's centerline with the lanes' centers 3 m apart will be within 1 m of the true position, 95 percent of the time.

High GPS sampling rates are desirable. Ideally, the sampling rate should be such that when an abrupt change of direction occurs, consecutive position fixes are no more than one cell width apart. This corresponds to a frequency of v/w for maximum cornering speed v and cell width w. For a cell width of a few meters, a 1 Hz sampling rate typically gives adequate performance. With lower frequencies, when the samples are linearly interpolated, the change of direction won't be as sharp as in reality. When a vehicle is traveling rapidly, the distance between consecutive fixes will be larger, but abrupt direction changes aren't possible owing to physical limits on deceleration. To decrease the volume of GPS data that a vehicle collects, we could adopt a strategy such as recording only the points where the direction changes substantially.

Roads with little traffic volume will require more time for changes to appear in the map. Furthermore, less popular roads might even fall under the threshold and thus not appear in the generated map because they aren't distinguishable from erroneous road segments.

When the algorithm receives a new GPS trace, the degree to which the trace

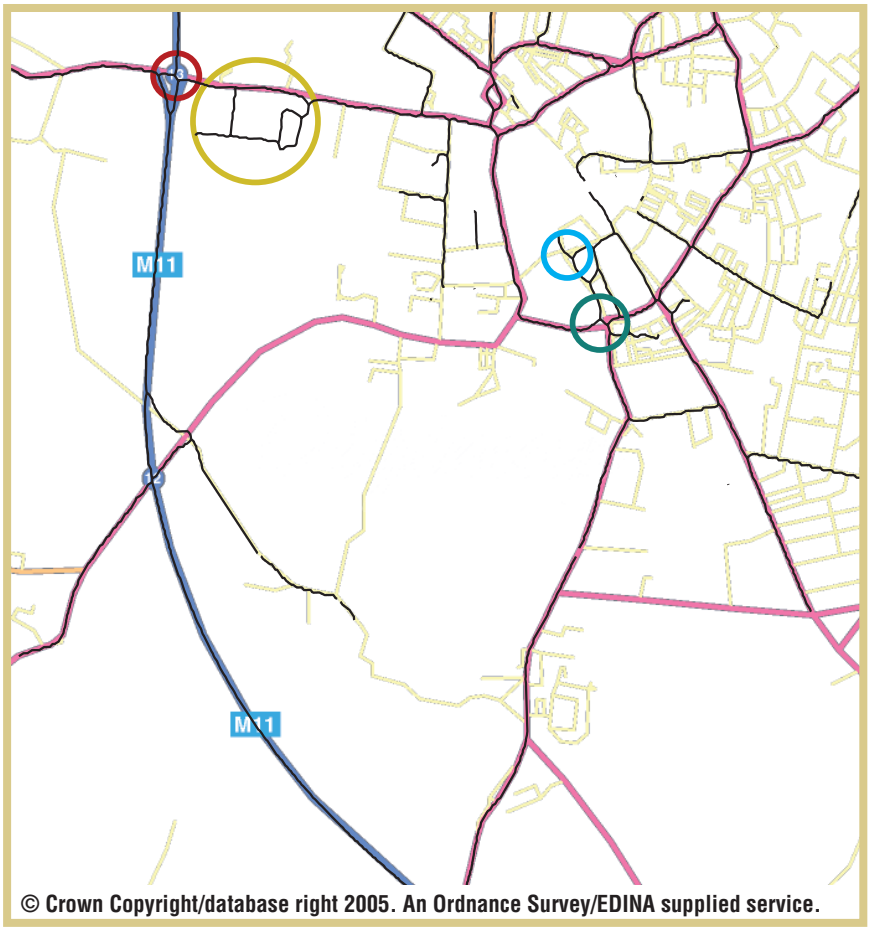

Figure 3. A UK Ordnance Survey map of the Cambridge area, with a generated road map overlaid in black. The yellow circle highlights new roads. The red circle highlights a bridge misinterpreted as a junction. The blue circle highlights a misaligned junction. The green circle highlights two junctions that have merged into one.

affects the generated map varies. If the trace uses roads that haven't been visited recently, the map won't likely be affected because the contributions to cells in the histogram won't rise above the threshold. On the other hand, if it uses roads that have been recently heavily visited, the map will be minimally affected because the trace will have little impact on the centerlines' positions. Between these two extremes, the trace will have a more significant effect. Because of the cost of regenerating the map, we can choose to regenerate it only when we have sufficient new data to significantly affect the output. We can further reduce the cost of executing the algorithm by regenerating only the map parts that have received new data.

Practical analysis

To investigate our algorithm's efficacy with real-world data, we performed a practical experiment. Our application generated the images in figures 1 and 3 from GPS traces constituting nearly one million position readings collected by a single vehicle that we drive in and around Cambridge. (For more on that vehicle, see the Works in Progress department in this issue. We've also generated road maps from other GPS data sources.)

We compared our application's output with a UK Ordnance Survey map of the same region (see figure 3). Our proof-of-concept implementation of the algorithm takes less than two minutes to complete on a standard desktop workstation, for a 40 km^2 region comprising approximately one million cells, with approximately one million GPS samples. In the histogram, 93 percent of the cells were empty, with 22 percent of the remainder falling below an empirically determined threshold.

According to the Cambridgeshire County Council, approximately 1,000 vehicles per hour use a typical city-center road.[9] If we need 73 samples at a particular point along a road's length to achieve acceptable accuracy, we should regenerate the map 14 times per hour—once every four minutes—with fresh data. Because the algorithm's time complexity is linear with area, our implementation could process an area approximately 80 km^2 every four minutes. So, to process the entire UK, we would need approximately 3,000 processing nodes. However, we believe that an optimized implementation could execute at least an order of magnitude more quickly.

On the roads that have a sufficient density of GPS readings, the generated road segments align well with the OS road segments. However, we noticed five main differences between our map and the OS map.

First, some road segments exist in the generated map but not in the OS map. These correspond to newly constructed roads that don't yet appear in the OS data, justifying our claim that you can use our algorithm to highlight the creation of new roads. The yellow circle in figure 3 indicates such new roads.

Second, the generated segments are much more jagged than the OS segments. This is because the algorithm extracts the segments from a Voronoi graph of jagged boundaries. A topic for further research is whether a line simplification algorithm such as the Douglas-Peucker algorithm[10] could smooth the segments.

Third, the generated map interprets road bridges as crossroads (see the red circle in figure 3). This is because we discard altitude data when initially forming the 2D histogram. This problem has two

potential solutions, which we plan to explore. By using a 3D histogram, extracting the 3D surface (using an algorithm such as Marching Cubes[11]), and producing a 3D Voronoi graph, we should find that the two roads no longer intersect. Alternatively, we could analyze each generated junction and determine which turns the vehicles can actually make. The algorithm would then interpret bridges as crossroads with no permissible turns.

Fourth, the generated map has some skewed junctions (see the blue circle in figure 3). This results from the errors inherent in the initial GPS fixes, causing the histogram to inaccurately reflect the true road layout. This problem is hard to fix but might be solved by having vehicles use accelerometers and sensor fusion techniques to improve the location fixes' accuracy.

Finally, some pairs of nearby junctions have merged (see the green circle in figure 3). This results from an excessively gross discretization resulting from too large a cell size or too heavy a blur. So, we can potentially fix this by adjusting these parameters.

Generating road maps in this way doesn't produce a perfect result, but neither do traditional mapmaking techniques. By using data from privately owned vehicles, we have the advantage of being able to discover changes to the road network. We hope that the degree of automation will increase as we solve the issues outlined in this section.

We plan to investigate making the histogram's cell sizes adaptive, so that our algorithm can more accurately inspect areas with many GPS readings. We also plan to analyze vehicles' direction of approach to and departure from junctions to determine the junctions' nature more precisely.

Future research on such a system's architectural design will simulate various architectures and investigate their applicability to applications with other data and processing requirements. Research is also necessary on social and security issues related to such participative applications, such as protecting vehicle owners' privacy and protecting the system from malicious users. P

ACKNOWLEDGMENTS

We thank Andrew Rice, David Cottingham, Robert Harle, and Ripduman Sohan for useful discussions; Richard Gibbens for assistance with the statistical analysis; Andrew Rice for the use of his contour follower implementation; and Keith Farkas for his helpful advice.

REFERENCES

1. S. Yokoi, J.-I. Toriwaki, and T. Fukumura, "An Analysis of Topological Properties of Digitized Binary Pictures Using Local Features," *Computer Graphics and Image Processing*, vol. 4, no. 1, 1975, pp. 63–73.

2. F. Aurenhammer, "Voronoi Diagrams—A Survey of a Fundamental Geometric Data Structure," *ACM Computing Surveys*, vol. 23, no. 3, Sept. 1991, pp. 345–405.

3. B. Hull et al., "CarTel: A Distributed Mobile Sensor Computing System," to be published in *Proc. 4th ACM Conf. Embedded Network Sensor Systems* (SenSys 06), ACM Press, 2006.

4. H. Kargupta et al., "VEDAS: A Mobile and Distributed Data Stream Mining System for Real-Time Vehicle Monitoring," *Proc. SIAM Int'l Data Mining Conf.*, Cambridge Univ. Press, 2004, pp. 300–311.

5. T. Nadeem et al., "TrafficView: Traffic Data Dissemination Using Car-to-Car Communication," *Mobile Computing and Comm. Rev.*, vol. 8, no. 3, 2004, pp. 6–19.

6. M. Torrent-Moreno, A. Festag, and H. Hartenstein, "System Design for Information Dissemination in VANETS," *Proc. 3rd Int'l Workshop Intelligent Transportation* (WIT), Technische Universität Hamburg-Harburg, 2006, pp. 27–33.

7. R. Prasad and M. Ruggieri, *Applied Satellite Navigation Using GPS, GALILEO, and Augmentation Systems*, Artech House, 2005.

8. K.D. McDonald and C. Hegarty, "Post-modernization GPS Performance Capabilities," *Proc. IAIN World Congress and the ION 56th Ann. Meeting*, Inst. of Navigation, 2000, pp. 242–249.

9. *2005 Traffic Monitoring Report*, Cambridgeshire County Council, 2006, www.cambridgeshire.gov.uk/transport/monitoring/network/traffic+monitoring+report.htm.

10. D.H. Douglas and T.K. Peucker, "Algorithms for the Reduction of the Number of Points Required to Represent a Line or Its Caricature," *The Canadian Cartographer*, vol. 10, no. 2, 1973, pp. 112–122.

11. W.E. Lorensen and H.E. Cline, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," *SIGGRAPH Computer Graphics*, vol. 21, no. 4, 1987, pp. 163–169.

the AUTHORS

Jonathan J. Davies is a PhD candidate in the University of Cambridge's Computer Laboratory. His research interests include intelligent transportation systems and sentient computing. He received his BA in computer science from the University of Cambridge. Contact him at the Computer Laboratory, 15 JJ Thomson Ave., Cambridge, CB3 0FD, UK; jjd27@cam.ac.uk; www.cl.cam.ac.uk/~jjd27.

Alastair R. Beresford is a research associate in the University of Cambridge's Computer Laboratory and a research fellow at Robinson College. His research interests include ubiquitous systems, computer security, and networking. He received his PhD in engineering from the University of Cambridge. Contact him at the Computer Laboratory, 15 JJ Thomson Ave., Cambridge, CB3 0FD, UK; arb33@cam.ac.uk; www.cl.cam.ac.uk/~arb33.

Andy Hopper is a professor of computer technology and the department head in the University of Cambridge's Computer Laboratory. His research interests include network design and sustainable and mobile computing. He received his PhD in computer science from the University of Cambridge. He's a fellow of the Royal Academy of Engineering and the Royal Society and is a trustee of the Institution of Engineering and Technology. Contact him at the Computer Laboratory, 15 JJ Thomson Ave., Cambridge, CB3 0FD, UK; ah12@cam.ac.uk; www.cl.cam.ac.uk/~ah12.