Efficient Tracking of Moving Objects with Precision Guarantees

Alminas Čivilis, Christian S. Jensen, Jovita Nenortaitė, Stardas Pakalnis

February 21, 2004

TR-5

A DB Technical Report

Title	Efficient Tracking of Moving Objects with Precision Guarantees
	Copyright © 2004 Alminas Čivilis, Christian S. Jensen, Jovita Nenortaitė, Stardas Pakalnis. All rights reserved.
Author(s)	Alminas Čivilis, Christian S. Jensen, Jovita Nenortaitė, Stardas Pakalnis
Publication History	February 2004. A DB Technical Report

Abstract

We are witnessing continued improvements in wireless communications and geo-positioning. In addition, the performance/price ratio for consumer electronics continues to improve. These developments pave the way to a kind of location-based service that relies on the tracking of the continuously changing positions of the entire population of service users. This type of service is characterized by large volumes of updates, giving prominence to techniques for location representation and update.

In this paper, we present several representations, along with associated update techniques, that predict the future positions of moving objects. For all representations, the predicted position of a moving object is updated whenever the deviation between it and the actual position of the object exceeds a given threshold. For the case where the road network, in which the object is moving, is known, we propose a so-called segment-based policy that represents and predicts an object's movement according to the road's shape. Map matching is used for determining the road on which an object is moving. Empirical performance studies and comparisons of the proposed techniques based on a real road network and GPS logs from cars are reported.

1 Introduction

The growing use of online, mobile devices and the increasing availability of positioning technology combine to enable Location–Based Services (LBSs). An LBS is capable of providing location-dependent information to its user. The basic idea is to identify the location of each user and, depending also on other input from each user, provide personalized services to the users As the service users are capable of continuous movement, we term them "moving objects" (MOs).

Many kinds of LBSs may be envisioned. For example, "yellow maps," which generalize yellow pages, provide information about various points of interest. Route guidance services help their users reach points of interest. Services that track delivery trucks, service employees, hazardous materials, tourists, or senior citizens suffering from dementia improve cost effectiveness or safety and security.

Studies of potential LBSs and their architectures can be found in the literature (e.g., [7, 8, 9]).

This paper's focus is on providing basic support for the class of tracking services, where there is a need for continuously monitoring of the current positions of a population of moving objects. Ideally, at each time unit when an object moves, a new position should be sent from the moving object to the central database. A large population of moving objects entails a very large volume of such updates. Updates carry the following costs:

1. *Communication costs*. Subjecting a wireless network to a high volume of updates may overload the network and degrade performance. Additionally, the use of third–party mobile carrier for data transmission with a high volume of updates would increases the cost of the tracking services.

2. *Server side update costs*. Database management systems typically do not support very large volumes of updates well. This is particularly true if spatial indexing of the moving objects is employed [5]. Powerful, and expensive, hardware is needed if very large volumes of updates are to be supported.

3. *Client side costs*. Communication carries a significant computational overhead on a mobile client. This leads to a shorter operational time for battery powered, handheld devices.

The goal of the work reported here is to design and evaluate tracking techniques that reduces the number of updates. This involves server as well as client-side algorithms.

We assume that a client is able to perform certain computations, to determine its location, and has a wireless bi-directional data connection to the server. A typical example of such a client is a modern mobile phone, e.g., a Nokia 3650, equipped with a Bluetooth Global Positioning System (GPS) unit, e.g., a Navman

GPS 4400, and data connection, e.g., a GPRS subscription to a GSM network. We take into account that a client may have memory restrictions and thus may be unable to store the map of an entire region in its memory.

The techniques we investigate are based on movement prediction, and it is an underlying assumption that precise locations of moving objects are not needed for most tracking LBSs. Rather, we assume that the LBSs to be supported require tracking with a certain minimum accuracy. Location-based games may need high accuracy, while a localized weather information service needs only low accuracy.

Location prediction is done on the clients *and* on the server. On the server, a moving object's current location is determined using the location received with the last update from the moving object and a prediction algorithm. A client also predicts its own position using the same data and algorithm as the server. It continuously compares the predicted location with its actual position, and it sends an update to the server if the difference between the two locations is about to exceed the threshold implied by the predefined accuracy.

We propose three update policies: a point policy, a vector policy, and a segment-based policy. The point policy uses constant position prediction, the vector policy uses the direction and speed of a moving object for position prediction, and the segment-based policy predicts a moving object's position according to its speed and the shape of the road on which the object is traveling.

Because the segment-based policy "knows" the road on which the MO is moving, it enables a wider range of services, which makes this policy important. Specifically, it enables services that rely on information attached to the road network, e.g., speed limits, real-time traffic status information. For this reason, we analyze this policy in most detail.

Each update policy is evaluated using real data obtained from an intelligent speed adaptation project. [10]. The data was generated by cars equipped with GPS receivers and computing devices, which were driving in and around the city of Aalborg, Denmark.

The three policies have different properties, and situations exist where each policy is superior to the other two. The point policy yields the least performance improvements on the data used. The point policy can be used for tracking of moving objects that move unpredictably, e.g., pedestrians. The vector-based policy is suitable in cases when movement is more "directed" (e.g., car and airplane movement), but where no information about an underlying transportation network is available. Finally, the segment-based policy is suitable in cases where movement is restricted by a known transportation infrastructure.

The paper makes the following key contributions:

- Design and implementation of three database update policies for the tracking of moving objects. The update policies use different means of predicting the future positions of moving objects.

- Empirical studies of the efficiency of the presented update policies using real data.

As a minor contribution, the paper uses a variant of map matching that is suitable for tracking policies that position moving objects in transportation networks based on streams of position samples. Although map matching is necessary for the paper's proposal to work, other map matching techniques could be used in place of the paper's proposal.

Several researchers have addressed different aspects of the problem of tracking moving objects. A detailed presentation of related work is presented in Section 9. While techniques similar to the point-based policy have been proposed in related work, we believe that the vector and segment-based policies are novel. We also believe that this study is the first to report on experimental studies of the three tracking techniques with real data.

The remainder of the paper is organized as follows. Section 2 describes the general update scenario and updating policies. Section 3 defines the notion of moving object, describes the architecture and database of the system that manages the updates. The map matching strategy is covered in Section 4, and the detailed algorithms used for map matching are covered in Section 5. Section 6 presents the main algorithms that runs on the server and client sides. Algorithms of the update policies and empirical studies of their performances, are given in Sections 7 and 8. Section 9 offers an overview of related work. Finally, we summarize, conclude, and offer suggestions for future research.

2 Tracking Scenario and Update Policies

This section presents the general scenario for tracking of moving objects, and it introduces the tracking policies proposed in the paper.

2.1 Update Scenario

To be specific, we assume that moving objects are equipped with GPS receivers and that data is transmitted between the clients and the server using a GSM network. We assume that disconnects between client and server are dealt with by other mechanism in the network than the policies we propose. When a disconnect occurs, these mechanisms notify the server, which may then take appropriate action.

A moving object always knows its position: its GPS receiver reports a position every second, with an imprecision of less than 30 meters. We assume that a client knows its position with higher precision than the precision required at the server. For brevity, we say that the client's GPS position is precise.

In contrast, the required precision on the server is determined by the needs of the services to be supported. In the experiments reported in Section 8, this position ranges from 40 to 1000 meters.

The server receives updates from each client—a client reports its current position and possibly some additional movement data. According to this, the server is able to predict the client's movement and to calculate where the client will be at later points in time. The update frequency depends on the required precision. It is the clients that determine when to issue position updates to the server.

A client uses the data it has sent most recently to the server and the same prediction algorithm as the server for calculating where the server "thinks" the client is currently located. This predicted position is compared to the precise position as obtained from the GPS unit. If the allowed precision threshold is exceeded, an update is issued; otherwise, no update is issued. Using this scenario, the client ensures that the server will always know its position with the agreed-upon precision.

2.2 Update Policies

The *point policy* uses constant position prediction. It thus assumes that the object to be tracked is located at the position given by the most recent updates (with the given precision).

The object being tracked actually moves and when its position deviates by more than the allowed threshold, a new update is generated. Using this policy, the movement of an object is represented as a "jumping point." This policy is the most primitive among the presented policies, but it may well be suitable for movement that is erratic with respect to the threshold used. An example is the tracking with a threshold of 200 meters of a child who is playing soccer.

Next, the *vector policy* uses the object's position as well as the object's speed and direction of movement (velocity vector). It assumes that the object moves linearly and with the constant speed received from the GPS device in the most recent update.

Using this policy, the movement of an object is represented as a "jumping vector." This policy may be useful for the tracking of "directional" movement, where the tracked object moves towards a destination.

Finally, the *segment-based policy* assumes that an object's movement is restricted to a known road network, meaning that an object moves along a known road segment and, like in the vector policy, moves at constant speed. When the object reaches the end of the road segment on which it is traveling, the policy assumes that the object stops. After some time, the actual position of the object will then deviate by more than the threshold from the predicted position, and an update occurs. (Due to the client memory restrictions, the entire road network cannot be stored on each client. Therefore, a client is not able itself to find the new road segment on which it is located.)

When the server receives an end-of-segment update, it applies a map-matching procedure to locate the new segment of the moving object. A geometrical representation of this segment, a polyline, is then sent to the client. In case an object cannot be placed on a new road segment, the system switches to the vector policy. The next update is then treated as an update caused by an end of segment event, and map matching will be attempted.

Using this policy, the movement of an object is represented as set of road segments with positions on them, and as jumping vectors in case map matching fails.

The remainder of the paper considers these policies in detail.

3 Data and Database Structures

We first describe the GPS and map data used, then present definitions of the data structures that we will be using. Then follows descriptions of the stream of positions samples produced by a GPS device. Finally, descriptions of the tracking system's architecture and the server-side database are then given.

3.1 GPS and Map Data

Real data is essential for obtaining useful, reliable feedback on the performance of the different update policies.

We use real GPS logs from 5 out of some two dozen cars that took part in an Intelligent Speed Adaptation project [6, 10] at Aalborg University. In that project, the driver of a car was notified every time the car exceeded the speed limit.

All cars were owned by individuals living in Aalborg municipality. Each car was equipped with a GPS receiver and computer. When a car was moving, the computer logged GPS readings every second. Each reading, or record, contains the car's position, the time when the position was measured, the car's speed, and some additional information related to the speed adaptation project. No data were recorded when the cars were parked. Cars were logged for 6 weeks, and each log consists of about 100,000 records.

Next, a digital road network covering Aalborg Municipality is used. The map is organized as a set of segments, where each segment corresponds to the part of a road in-between neighboring crossroads or dead ends. Each segment's geometry is given as a polyline.

3.2 Representation of Moving Object Positions

We proceed to define data structures that are fundamental to the update policies. We present two different representations of moving object positions. The first representation describes the position of a moving object by an initial position and a velocity vector. The second represents moving object positions in terms of the road network.

Definition 3.1 (Domain) The underlying domain is two-dimensional Euclidean space, where a point is defined as a pair of coordinates $p = (x, y)$. The set of all the points is denoted by $\mathcal{P}$. Time is defined on $\mathbb{Q}_+$.

Definition 3.2 (Moving Object) The representation of a moving object *mo* is a three-tuple $mo = (p, \overline{v}, t)$, where

1. $p \in \mathcal{P}$ represents the initial position of the moving object

2. $\overline{v} = (dx, dy) \in \mathcal{R}^2$ is a velocity vector that captures the the movement direction and speed ($|\overline{v}|$) of the moving object

3. $t \in \mathbb{Q}_+$ is the time when p and $\overline{v}$ were measured

All three of these values can be obtained directly from a GPS receiver. More detail on GPS data is presented in Section 3.3. Before introducing the representation of a moving object's position in terms of a road network, we define the data structures required for modeling a road network.

Definition 3.3 (Polyline) A polyline $pl = (p_0, p_1, \ldots, p_n)$, where $n \geq 1$, is an n-tuple of points $p_i \in \mathcal{P}$. We term p_0 the start point and p_n the end point of the polyline. (The OpenGIS Consortium uses the term "line string" for this geometrical feature [4].) The *direction* of a polyline is from the start point to the end point. The set of all the polylines is denoted by $\mathcal{PL}$.

A polyline, exemplified in Figure 1, can be described as a sequence of connected line segments.

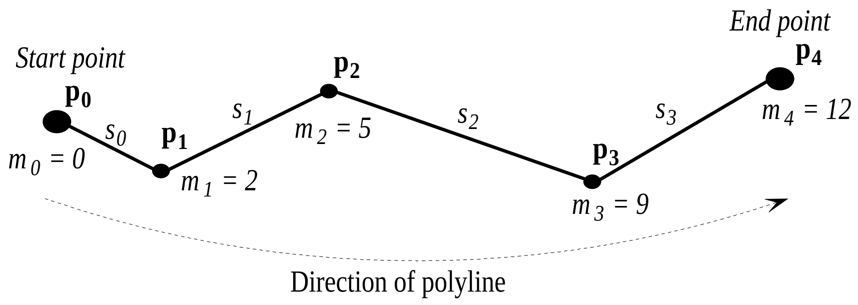

Figure 1: Polyline

Definition 3.4 (Line Segment) A line segment is a polyline with precisely two points. A line segment $s_i = (p_i, p_{i+1})$ belongs to polyline pl, written $s_i \in pl$, iff $p_i, p_{i+1} \in pl$. A point p belongs to line segment $s = (p_0, p_1)$ iff $\exists w \in [0,1](p = p_0 + w(p_1 - p_0))$.

Definition 3.5 (Measure) The measure (m) of a point p located on a polyline is the linear distance, measured along the polyline, from the start point to point p; see Figure 1.

Function $\mathcal{M}$ calculates the measure of any point p located on a polyline pl. Let function d compute Euclidean distance. Function $\mathcal{M} : \mathcal{PL} \times \mathcal{P} \mapsto \mathbb{R}$ is defined as follows:

$$\mathcal{M}(pl, p) = \begin{cases} 0 & \text{if } p = p_0 \\ \mathcal{M}(pl, p_i) + d(p_i, p) & \text{if } \exists i\big(i = \min(\{j \,|\, p \text{ belongsto } s_j \wedge s_j \in pl\})\big) \\ \mathit{undefined} & \text{otherwise} \end{cases}$$

The measure of a point is equal to the measure of the starting point p_i of segment s_i on which the point p is located, plus the Euclidean distance $d(p_i, p)$. This function is undefined if the point p is not on the polyline.

Function $\mathcal{M}^{-1}$ calculates a point with a given measure on a given polyline. Function $\mathcal{M}^{-1} : \mathcal{PL} \times \mathbb{R} \mapsto \mathcal{P}$ where $w = (m - \mathcal{M}(pl, p_{i-1}))/d(p_{i-1}, p_i)$ is defined as follows:

$$\mathcal{M}^{-1}(pl, m) = \begin{cases} p_{i-1}(1 - w) + p_i w & \text{if } \exists i\big(i = \min(\{j \,|\, \mathcal{M}(pl, p_j) \geq m \wedge p_{j-1}, p_j \in pl\})\big) \\ \mathit{undefined} & \text{otherwise} \end{cases}$$

At first, function $\mathcal{M}^{-1}$ is looking for two adjacent points p_{i-1} and p_i on the polyline that have measures in-between which the given measure m is located. Then the difference between measure m and the measure of point p_{i-1} is calculated. According to the ratio between this distance and the distance between points p_i and p_{i-1}, a parameterization of searched point is done. If two points between which the point with measure m is located cannot be found, the result is undefined.

Definition 3.6 (Road Network) A road network is a set of polylines RN, $RN \subset \mathcal{PL}$.

With the preceding definitions available, we can define the representation of a moving object's location in terms of a road network.

Definition 3.7 (Moving Object on a Polyline) The representation of a moving object on a polyline (MOP) is a four-tuple $mop = (pl, m, plspd, t)$, where:

1. $pl \in \mathcal{PL}$ is the polyline on which the moving object is located
2. $m \in \mathcal{R}$ is the measure of the moving object's location on pl
3. $plspd \in \mathcal{R}$ indicates the moving object's speed along the polyline (a positive value indicates movement in the direction of the polyline, while a negative values indicates the opposite direction)
4. $t \in \mathbb{Q}_+$ is the time when the preceding values are valid

This representation assumes that the object moves along a known polyline. The position of a moving object in mop is expressed as a measure m on a polyline pl; its velocity is expressed as a signed speed, where the sign indicates the direction along the polyline.

Definition 3.8 (Course) The course of a moving object is the angle between True North and the direction in which the object is moving. True North is the direction of Earth's geographic North Pole. The direction of a moving object is given by the direction of the line segment on which the object is moving.

We say that the course of a point located on a polyline is the course of the line segment on which that point is located. If the point is located on a connection point between two segments in a polyline, the course of the point is the course of the line segment that has the smaller index.

Definition 3.9 (Server Object) The information sent by the sever to a client is a server object, SO, which is a three-tuple $(policy, ob, thr)$, where:

1. $policy \in \{$*'point'*, *'vector'*, *'segment'*$\}$ identifies the update policy used
2. ob is the representation of the moving object's position on the server ($policy = 'segment'$ implies an mop; otherwise it is an mo)
3. $thr \in \mathcal{R}$ is a threshold

3.3 GPS Output

Communication with a GPS receiver is based on the NMEA (National Marine Electronics Association) standard, which aims to enable data communication between electronic marine instruments, navigation equipment, and communications equipment. Under the NMEA-0183 standard, the data is transmitted in the form of "sentences." A device typically sends a group of sentences at intervals determined by the unit's update rate. Generally, the sending of data is made at most once per second. Different GPS receivers send different amounts and types of information, i.e., different groups of sentences. For our purposes, we are interested in RMC (recommended minimum specific GPS/Transit data) sentences. This type of sentence contains all the information we need (position, ground speed, course, and time of sample), and it can be obtained from all GPS devices that supporting the NMEA standard.

An example RMC sentence is described next:

```
"GPRMC,142958,A,5540.83,N,01233.71,E,020.9,304.5,271102,001.1,E*74"

       142958   Time of sample, 14:29:58 UTC
            A   Validity of data,  A: valid; V: invalid
  5540.8342,N   Latitude, 55 deg. 50.83 min North
   01233.71,E   Longitude, 12 deg. 33.71 min East
        020.9   Speed over ground in knots
        304.5   Course in degrees
       271102   Date of sample, 27 November 2002
     001.1,E    Magnetic variation, 1.1 deg East
          *74   Checksum
```

For the representation of moving objects, we are using the fields with latitude and longitude (position); date and time (time); and speed and course (velocity).

3.4 System Architecture

The system has a client/server architecture—see Figure 2. A mobile client is equipped with a GPS receiver

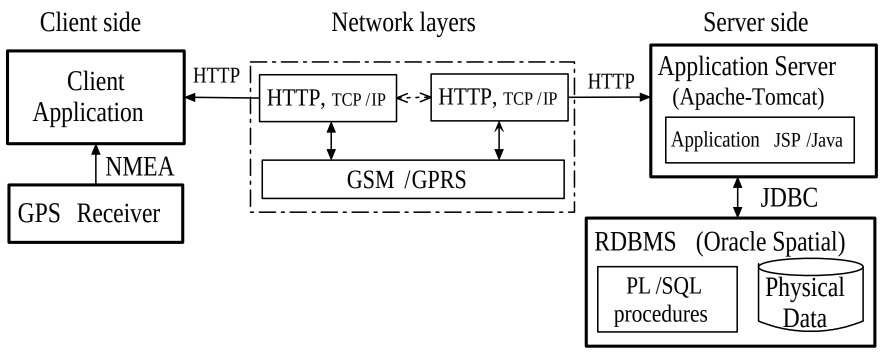

Figure 2: System Architecture

and a GSM device. The GPS receiver obtains the client's location and outputs this as a stream of sentences described by the NMEA standard. Using the GSM device, the client connects wirelessly to the server. Communication is done via the HTTP protocol, where data is transmitted using a GPRS data connection.

On the server side, applications are divided into tiers. An application server uses services of a database system to process client requests. Data management functions are located in the database layer as stored procedures. We use the Oracle DBMS with the Oracle Spatial package.

3.5 Database Schema

The data required by the different update policies is stored in a database. Logically, this is divided into (1) data about each moving object and (2) a geo-referenced representation of the road network.

For each moving object, the database stores the object's location in table *Object_position*, shown in Figure 3.

The fields $policy$ and field ob from the so structure (Definition 3.9) are stored here. Depending on the policy used, field ob is an mo or a mop. The pair of attributes *xcoord* and *ycoord*, and *dx* and *dy* applies to field p and field $\overline{v}$ of an mo, respectively. Attribute *measure* and *speed* applies to field m and $plspd$ of an mop. Attribute *segid* references a segment corresponding to field pl of an mop. Both representations include a time, which is stored in attribute *gpstime*.

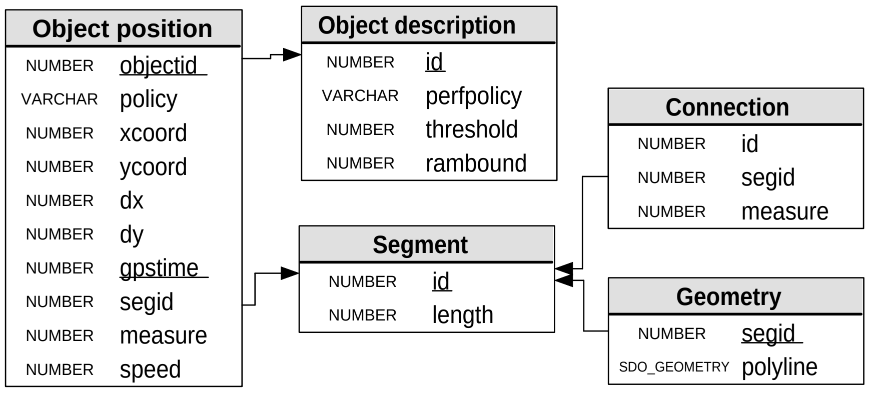

Figure 3: Database Schema

The value of *gpstime* is obtained from the GPS receiver on the client side. It is important for the prediction procedure that the server's clock is synchronized with the GPS clock used by all clients, as elapsed time is taken into account. Precise synchronization can be done by introducing a GPS receiver on the server side.

An object's preferences are stored in table *Object_description*, where attribute *threshold* stores the required accuracy, *prefpolicy* stored the preferred update policy, and *rambound* records the available memory. Both tables have an attribute that stores the update policy. The presence of attribute *policy* enables a temporary switch to another policy than *prefpolicy*.

A road network is modeled as a collection of road segments that intersect at connection points. The concept of a segment is thus central,and any content attached to a road network references the segments [3]. A tuple in table *Segment* identifies a segment and records its length. The geographical extent of a road segment is modeled by a polyline. A tuple in table *Geometry* stores a polyline for each segment.

We use the Oracle DBMS with its Spatial Data Option, where polylines are stored as linear spatial objects. In a DBMS without spatial capabilities, table *Geometry* may store sequence of (x, y) coordinates instead of linear spatial objects.

Finally, table *Connection* stores points along segments where there are intersections and exchange of traffic thus may occur. This table allows us to distinguish between real intersections and situations where road segments geometrically intersect, but where there is no access from one road to other, e.g., an underpass. A tuple indicates that an intersection is located on segment *segid* at *measure* distance units after the start of the segment. Attribute *id* is an identifier.

4 Map Matching Strategy

We proceed to describe the overall map matching strategy; a detailed coverage of map matching is given in Section 5.

Ideally, when an object traveling on a road reports its GPS position to the server, that position should intersect with the digital representation of the road. However, in practice this is typically not the case, the two most important reasons being the following.

- GPS positions without corrections are often inaccurate.
- The digital representation's positioning of the road network deviates from the real road network's positioning. This may be caused by the representation being imprecise, by road constructions not having been reflected in the digital representation, and by errors.

For these reasons, a non-trivial procedure must exist that map matches GPS positions onto segment in the representation of the road network.

Figure 4 shows a real example of the problem to be solved. Here, both the GPS positions and the digital road network are imprecise. Lines represent the road network representation, small circles represent reported GPS positions, and small squares on the road network represent map matched positions. The map

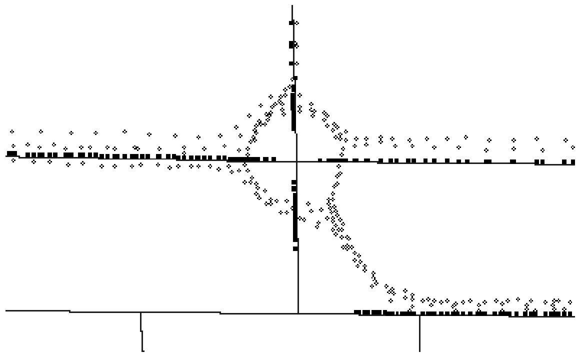

Figure 4: Correspondence Between Road Network and GPS Positions

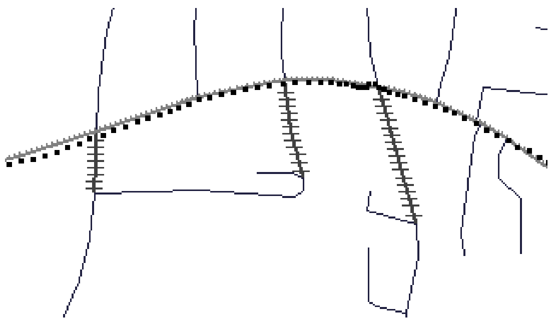

Figure 5: Redundant Road Segments

matching proceeds in two steps. First, all road segments located no further than some distance away from the reported GPS position are found. Second, the road segment on which the position should most probably be located on is identified.

In the second step, simply selecting the closest road segment may produce an incorrect result. This procedure can select a road perpendicular to the direction of the object's movement. See Figure 5, where small squares represent positions of moving objects and 3 dashed segments represent wrongly identified road segments. This problem is solved by eliminating the roads that are perpendicular to an object's movement direction. Also segments what are not connected according to the *Connection* table are eliminated.

Specifically, the movement direction of an object is taken into account by using the GPS position that preceded the GPS position to be map matched and by creating a third position that is the anticipated next GPS position of the moving object, assuming linear movement. The predecessor and successor positions are projected onto the same candidate road segment among several such candidates. The road segment for which the sum of the two distances between the positions and their projected positions is the lowest is selected.

An example of distance calculations is given in Figure 6. Here, it is assumed that the x and y axes

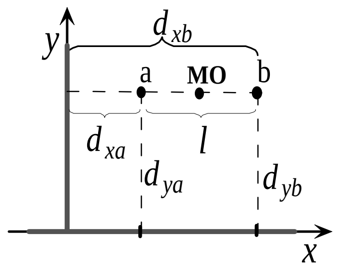

Figure 6: Distance Calculation

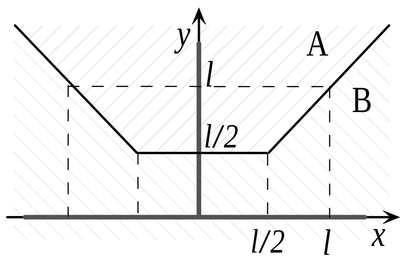

Figure 7: Map Matching Regions

represent intersecting road segments. The predecessor position is "a," the position to be map matched is "MO," and the anticipated successor position is "b." The object thus moves in parallel with the x axis, from the left to the right. We compare the sums $d_{xa} + d_{xb}$ and $d_{ya} + d_{yb}$ to determine to which road segment the GPS position should be matched.

The distance l between a and b influences the selection of a road segment, as shown in Figure 7. Again, the object moves in parallel with the x axis. If the GPS position is in region A, it is map matched to the y axis. If it is in region B, the x axis is selected.

Notice that l increases with the speed of the moving object. As l increases, the curve separating the two regions moves upwards, and we tend to chose the x axis. This is as desired, as the likelihood that an object makes a turn decreases with increasing speed.

However, when l is small the procedure just described is not effective. Instead, we make explicit use of direction, by comparing the direction of the object's movement to the direction of the road segments. The direction of a road segment is computed by first projecting two consecutive GPS positions for the object onto the polyline representing the segment. Then the direction of the line segment described by the projections of the points is the road segment's direction.

As a result, we have two values for each road segment, a sum of distances and a direction difference. These two are normalized, as direction and distance are expressed in different units of measure, upon which they are combined into a single value. We then select the road segment with the lowest value.

5 Map Matching Algorithms

This section presents the actual algorithms used for map matching. Initially, a number of auxiliary algorithms are presented.

5.1 Determining Moving Direction on a Polyline

We proceed to give the algorithms for determining an object's movement direction on a polyline. This direction is expressed as value $+1$ (with the polyline's direction) or -1 (against the polyline's direction).

5.1.1 Get Vectors Course

The direction of a moving object can be expressed as a velocity vector or as a course. Using a course makes it easier to compare two directions as differences between directions can be calculated directly by subtraction. Algorithm **GVC** transforms a direction represented as a velocity vector into a direction represented as a course.

Algorithm 5.1 $\mathbf{GVC}(\overline{v})$

```
(1)  $angle \leftarrow arcTan(\overline{v}.dx, \overline{v}.dy)$
(2)  **if** $angle < 0$ **then**
(3)      $angle \leftarrow 360^0 + angle$
(4)  **end if**
(5)  **return** $angle$
```

The transformation is done by calculating the arctangent of the vector's components. Function *arcTan* calculates the arctangent of dx/dy and resolves case where $dy = 0$. If the resulting angle is smaller than 0^0, 360^0 are added to avoid negative angles. A 0^0 course corresponds to North and 90^0 corresponds to East, etc.

5.1.2 Get Polyline Direction

The next algorithm, $\mathbf{GPD}$, calculates the direction of the polyline on which a given point is located. The point location is given by a polyline and a measure. The resulting direction is expressed as a vector.

Algorithm 5.2 $\mathbf{GPD}(pl, m)$

```
(1) if $m = 0$ then
(2)     $\overline{v} \leftarrow (p_1.x - p_0.x, p_1.y - p_0.y)$
(3) else
(4)     $i \leftarrow \max(\{i \mid \mathcal{M}(pl, p_i) < m\})$
(5)     $\overline{v} \leftarrow (p_{i+1}.x - p_i.x, p_{i+1}.y - p_i.y)$
(6) end if
(7) return $\overline{v}$
```

If $m = 0$, the direction of the point with measure $m = 0$ is equal to the direction of the first line segment. For all other cases, the algorithm looks for the polyline point p_i with the maximum index and with a measure smaller than the given measure (line 4). The direction vector of polyline segment s_i (on which the argument point is located) is calculated by subtracting its start point from its end point.

5.1.3 Get Point Direction on Polyline

Algorithm $\mathbf{GPDP}$ calculates the direction on a polyline of a moving object. The direction is determined by comparing movement direction of the object with the direction of the polyline (Figure 8).

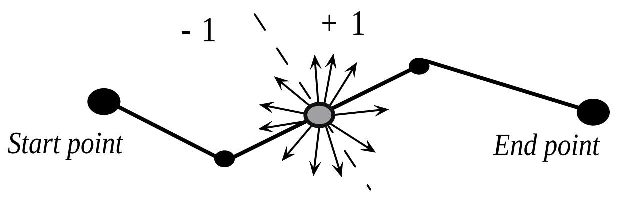

Figure 8: A Point's Direction on a Polyline

Algorithm 5.3 $\mathbf{GPDP}(pl, p, \overline{mov})$

```
(1) $\overline{plv} \leftarrow \mathbf{GVC}(\mathbf{GPD}(pl, \mathcal{M}(pl, p)))$
(2) $difangle \leftarrow (360 + \overline{plv} - \mathbf{GVC}(\overline{mov})) \mod 360$
(3) if $difangle < 90$ or $difangle > 270$ then
(4)     return $1$
(5) else
(6)     return $-1$
```

The polyline's direction is calculated using algorithm **GPD**, and algorithm **GVC** transforms vectors into courses. The difference between the courses is computed in line 2. Lines 3–5 then use this difference to determine moving objects direction on the polyline. If the angle between the directions is less than 90^0 or more than 270^0, the moving object's direction coincides with the polyline's direction; otherwise, the two are opposite.

5.2 Determining a Point's Location on a Polyline

This section defines algorithms that project points onto segments and polylines.

5.2.1 Projecting a Point onto a Line Segment

Algorithm $\mathbf{PPLS}$ projects a point onto a line segment, thus returning a point. Figure 9 illustrates two possible cases while projecting. The first occurs when the point's projection lies on the segment (point p).

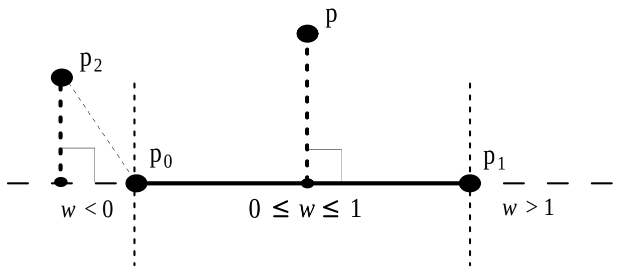

Figure 9: Projecting a Point onto a Line Segment

The second occurs when the point's projection lies on the extension of the segment (point p_2). In this latter case, the algorithm returns the line segment's end point that is closest to the point's projection (point p_0).

Algorithm 5.4 $\mathbf{PPLS}(p, s)$

```
(1)  $w \leftarrow \big((p.x - s.p_0.x)(s.p_1.x - s.p_0.x) + (p.y - s.p_0.y)(s.p_1.y - s.p_0.y)\big)/d(s.p_0, s.p_1)^2$
(2)  if $w < 0$ then $result \leftarrow s.p_0$
(3)  if $0 \leq w \leq 1$ then $result \leftarrow \big(s.p_0.x + w(s.p_1.x - s.p_0.x), s.p_0.y + w(s.p_1.y - s.p_0.y)\big)$
(4)  if $w > 1$ then $result \leftarrow s.p_1$
(5)  return $result$
```

The algortihm projects argument point p onto a line that is an extension of the argument line segment s. The location of the point's projection onto the line is expressed in terms of a variable w (line 1). A detailed description of the equation used can be found in Appendix A.

If w satisfies the condition $0 \leq w \leq 1$, the projection lies on the line segment. In this case (3), the projection is derived by using w. If $w < 0$ or $w > 1$, the projection lies on the extension of the line segment, and points p_0 and p_1 are returned.

5.2.2 Projecting a Point onto a Polyline

Algorithm $\mathbf{PPP}$ projects an argument point onto an argument polyline by using algorithm $\mathbf{PPLS}$.

Algorithm 5.5 $\mathbf{PPP}(p, pl)$

```
(1)  $i \leftarrow \min(\{i | s_i \in pl \wedge \forall j \neq i \; (d(\mathbf{PPLS}(p, s_i), p) \leq d(\mathbf{PPLS}(p, s_j), p))\})$
(2)  return $\mathbf{PPLS}(p, s_i)$
```

The algorithm finds the line segment that is the nearest to a projected point p. The projection onto that line segment is the projection onto the polyline. If more than one nearest line segment exists, the line segment with lower index is chosen.

5.3 Map Matching

Algorithm $\mathbf{MM}$ finds the polyline in the road network on which a given point (e.g., a GPS location) is located. The input to the algorithm is a position, a velocity vector, a maximum allowed distance r between the position and its projection onto a polyline, and a road network. The algorithm returns a polyline and a projection onto that polyline.

Algorithm 5.6 $\mathbf{MM}(p, \overline{v}, r, RN)$

```
(1)  $cand \leftarrow$all polylines $pl \in RN$ that are within
             distance $r$ of point $p$
(2)  **if** $cand = \emptyset$ **then return** $(undefined, p)$
(3)  **else**
(4)     $i \leftarrow \min(\{i | pl_i \in Candidates \wedge \forall j \neq i\ (\mathbf{MMW}(pl_i, p, \overline{v}, r) \leq \mathbf{MMW}(pl_j, p, \overline{v}, r))\})$
(5)     **return** $(pl_i, \mathbf{PPP}(p, pl_i))$
(6)  **end if**
```

In the first step, all candidate polylines are selected. If there are no candidate polylines, the algorithm returns the input point and an undefined polyline. Otherwise, the algorithm looks for the polyline with the smallest weight among the candidate polylines (line 4). Weights are calculated using function $\mathbf{MMW}$, which is presented below. Finally, the algorithm returns the selected polyline and the result of calling algorithm $\mathbf{PPP}$ on the argument point and selected polyline.

Function $\mathbf{MMW}$ that calculates map matching weights is defined next.

Algorithm 5.7 $\mathbf{MMW}(pl, p, \overline{v}, r)$

```
(1)  $p_r1, p_r2 \leftarrow$ project points $p - \overline{v}, p + \overline{v}$ onto $pl$
(2)  $dist \leftarrow d(p_r1, p - \overline{v}) + d(p_r2, p + \overline{v})$
(3)  **if** $p_r1 = p_r2$ **then**
(4)     $p_r \leftarrow$ project point $p$ onto $pl$
(5)     $difangle \leftarrow |\mathbf{GVC}(\overline{v}) \mod 180 - \mathbf{GVC}(\mathbf{GPD}(pl, p_r)) \mod 180|$
(6)  **else**
(7)     $difangle \leftarrow |(\mathbf{GVC}(\overline{v}) \mod 180 - \mathbf{GVC}(p_r1 - p_r2) \mod 180|$
(8)  **end if**
(9)  **if** $difangle > 90$ **then**
(10)    $difangle \leftarrow 180 - difangle$
(11) **end if**
(12) $weight \leftarrow \alpha \cdot difangle/90 + \beta \cdot dist/2r$
(13) **return** $weight$
```

Weights are calculated as described in Section 4. In the first line, two points are projected onto the given polyline. Distances between the points and their projections are calculated in line 2. If the projected points are not equal (line 3), the direction of the polyline is the direction of line segment from p_r1 to p_r2. The angular difference is calculated by subtracting the polyline's course from the velocity vector's course.

The course of the velocity vector is calculated using algorithm $\mathbf{GVC}$ followed by division with modulus 180, as we analyze the sharp angle ($< 90^0$) between the polyline and the object's movement direction. In addition, if the difference is greater than 90^0, we subtract it from 180 (lines 9 and 10).

The distance and angle are normalized with their maximum possible values, and the weight is calculated in line 12. Constants α and β are introduced in order be able to balance the influence of distance and angle on the resulting weight. We use value 1 for both constants.

6 Server and Client Side Cycles

This section presents the client and server side cycles that implement the update policies.

6.1 Server Side Cycle

The main server-side task is to update the database when a new update arrives. After each update, the server returns a confirmation message with information specific to the update policy in use.

Algorithm 6.1 Server Cycle

```
(1)  receive an update $mo$ from an object
(2)  if $prefpolicy$ = 'segment' then
(3)     $(mpl, mp) \leftarrow \mathbf{MM}(mo.p, mo.\overline{v}, r, RN)$
(4)     if $mpl = undefined$ then
(5)        $so \leftarrow$ ('vector', $mo$, $threshold$)
(6)     else
(7)        $mop.plspd \leftarrow \mathbf{GPDP}(mpl, mp, mo.\overline{v}) \cdot |mo.\overline{v}|$
(8)        $mop.pl \leftarrow$ cut $mpl$ according to direction and the client's memory limitations
(9)        $mop.m \leftarrow \mathcal{M}(mop.pl, mp)$
(10)       $so \leftarrow$ ('segment', $mop$, $threshold$)
(11)    end if
(12) else
(13)    $so \leftarrow (prefpolicy, mo, threshold)$
(14) end if
(15) store $so$ in the database and send it to the client
(16) goto 1
```

Having received data from a client, the server identifies the client and retrieves its preferred update policy $prefpolicy$ and threshold $threshold$ from the database. If the segment-based policy is preferred, map matching is performed. If map matching fails, the server switches to the vector policy (line 4) and updates the client's $policy$ attribute in table *Object_position*. The preferred policy remains unchanged; on the next update, the server will again perform map matching. After a successful map matching, mop object is created. Finally, the server packs all information into an so object, stores it in the database, and sends it to the client. After that, the server waits for a new request from a client.

6.2 Client-Side Cycle

The main task of a client is to control the precision of the predicted position on the server side. According to the required precision, the client decides whether or not to issue an update.

Algorithm 6.2 Client Cycle

```
(1)  receive position data $mo$ from the GPS receiver
(2)  send the current position $mo$ to the server
(3)  receive server object $so$ from the server
(4)  while not finished do
(5)     repeat
(6)        read position data $mo$ from the GPS receiver
(7)        if $so.policy$ = 'point' then
```

```
(8)             $p_{pred} \leftarrow \mathbf{PPP}\ (so.mo)$
(9)           **elsif** $so.policy$ = *'vector'* **then**
(10)            $p_{pred} \leftarrow \mathbf{PPV}(so.mo, mo.t)$
(11)          **elsif** $so.policy$ = *'segment'* **then**
(12)            $p_{pred} \leftarrow \mathbf{PPS}\ (so.mop, mo.t)$
(13)          **end if**
(14)        **until** $d(p_{pred}, mo.p) > so.threshold$
(15)        send the current position $mo$ to the server
(16)        receive server object $so$ from the server
(17)    **end while**
```

Lines 1–3 perform initialization. The client obtains position data from the GPS receiver and sends this to the server, which in response returns a server object that specifies a threshold and an update policy. The server object also includes an mo or mop, depending on the policy. With this information, the client is able to calculate the position predicted by the server. For the segment-based policy, the sever object contains an mop, which includes a map matched polyline and a position on the polyline.

The main cycle in line 4–17 continues until the client is switched off. These lines repeatedly read the object's GPS position, compute the predicted position, and the distance between these two positions. An update is issued if this distance exceeds the threshold. Notice that the client supports all update policies (covered in the next section). The server can change the policy and the threshold.

7 Prediction Policies

The prediction algorithms are used on the client side to determine if an update should be sent to the server. On the server side, these algorithms are used when location-based services query the positions of the moving objects.

The algorithm for the point policy, $\mathbf{PPP}$ (Predict Point with Point policy), is simple.

Algorithm 7.1 $\mathbf{PPP}(mo)$

```
(1)  **return** $mo.p$
```

As the prediction is constant, the predicted position is the same as the input position.

Next, the vector policy assumes that object moves in a linearly and with constant speed. Algorithm $\mathbf{PPV}$ (Predict Point with Vector policy) predicts the location of the given object mo at given a time t_{cur}. The result of the algorithm is the location of mo at time t_{cur}.

Algorithm 7.2 $\mathbf{PPV}(mo, t_{cur})$

```
(1)  $p_{pred} \leftarrow mo.p + mo.\overline{v}(t_{cur} - mo.t)$
(2)  **return** $p_{pred}$
```

The predicted location is calculated by adding the time-dependent traveled distance $(t_{cur} - mo.t)$ to the starting point in the direction of vector $\overline{v}$.

The segment-based policy takes into account the shape of the road on which an object is moving—an object thus moves according the shape of the road.

Algorithm $\mathbf{PPS}$ (Predict Position with Segment policy) is defined as follows.

Algorithm 7.3 $\mathbf{PPS}(mop, t_{cur})$

```
(1) $m_{pred} \leftarrow mop.m + mop.plspd\ (t_{cur} - mop.t)$
(2) **if** $m_{pred} >= \mathcal{M}\ (mop.pl, p_n)$ **then**
(3)     **return** $mop.pl.p_n$
(4) **elsif** $m_{pred} <= 0$ **then**
(5)     **return** $mop.pl.p_0$
(6) **else**
(7)     **return** $\mathcal{M}^{-1}(mop.pl, m_{pred})$
(8) **end if**
```

Here, an object's position is given by a polyline and a measure. The predicted location is given as a new measure, which is equal to the old measure (line 1) plus the distance traveled since the last update $t_{cur} - mop.t$. The traveled distance is negative if the object moves against the direction of the polyline. If the new measure is outside the polyline (lines 2 and 4), the result is one of end points of the polyline (lines 3 and 5). This way, the position prediction stops at a boundary point of the polyline.

8 Experimental Evaluation

We evaluate the three update policies using the data described in Section 3.1. The performance of each update policy is expressed as the number of updates required by a policy to ensure a specified precision. Precisions in the range 40–1000 meters are considered.

Experimental results are presented in Figure 10. Here, precision threshold values in meters are on x axis.

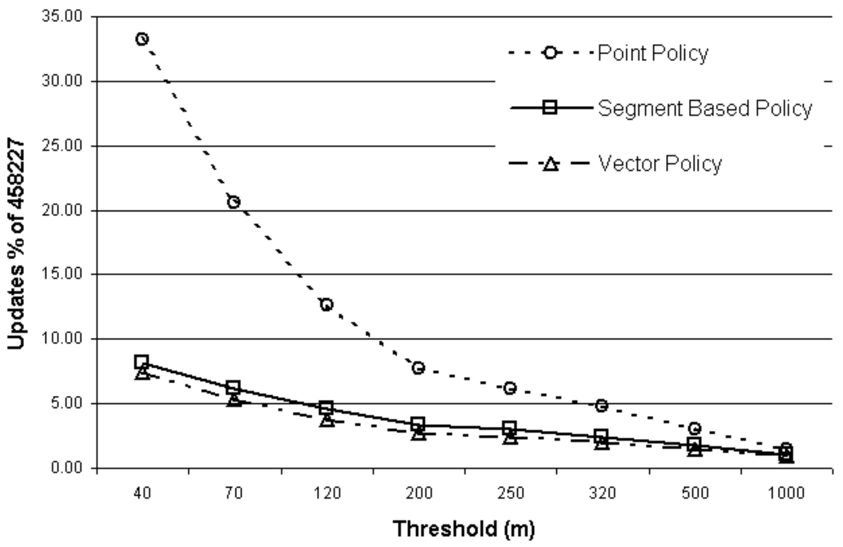

Figure 10: Comparison of Update Policies

The y axis presents the percentage of positions received by the client from its GPS receiver that triggered an update. For example, the meaning of 100% is that every position received from the GPS receiver leads to an update to be sent to the server in order to maintain the required precision.

The numbers of updates decrease as the precision threshold increases, i.e., as the required precision decreases. The point policy shows the worst performance in comparison to the two other policies. Notice that the largest performance improvement of the segment-based and vector policies over the point policy is for smaller thresholds, while for larger thresholds the improvement is smaller. For thresholds below 200 meters, the segment-based and vector policies are more than two times better than the point policy.

The segment-based and vector policies exhibit similar performance, with the vector policy being slightly better. Figure 11 offers a more detailed analysis of the update performance of the segment-based policy. Here, the solid line presents the same results as in Figure 10. The number of updates required for some

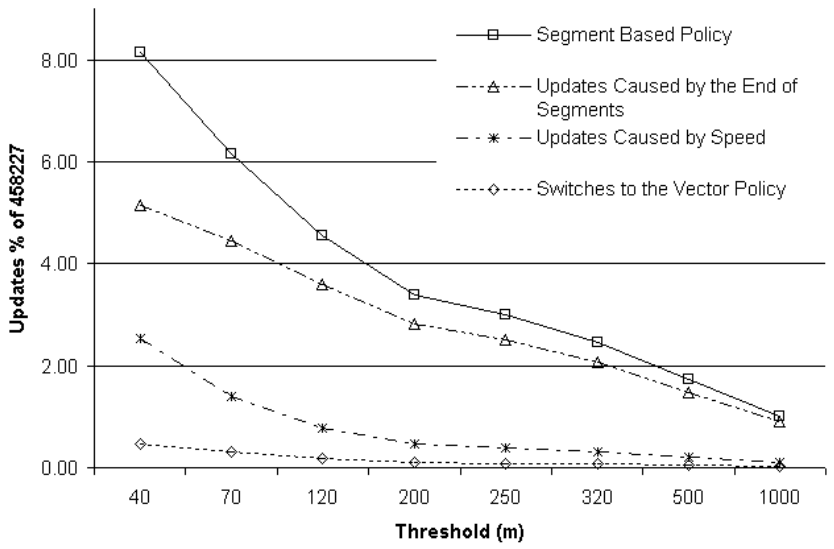

Figure 11: Detailed Analysis of the Segment-Based Policy

threshold in the segment-based policy is the sum of updates caused by segment changes, speed variations, and updates made due to switches to the vector policy. Updates caused by segment change are updates that occur when a new polyline is identified for position prediction. Updates caused by speed changes occur when a moving object sends an update of its position that does not result in the use of a new polyline. For example, this occurs when an object stops. Finally, some updates occur because map matching does not identify a segment, causing a switch to the vector policy. These update components are presented in Figure 11 by separate dashed lines.

Updates caused by segment changes exert the biggest influence on the total number of updates. Notice that the experiment was run on the same data set, only with different threshold values. So while the cars went though the same road segments, the number of segments used for prediction decreases as the precision threshold increases. This is because larger threshold values allow the policy to “avoid” some segments that are shorter than the threshold. Updates caused by speed has less influence to a total number of updates. And only few updates where done while switched to the vector policy. This illustrates the segment-based policy’s dependence on the representation of the road network.

9 Related Work

A few existing contributions related to those of this paper.

Wolfson et al. [14] propose dead-reckoning update policies for spatial databases. Like the policies presented in this paper, these policies use a deviation threshold. Specifically, three dead-reckoning policies are proposed, one of which allows recalculation of the threshold after each update. However, while we assume that thresholds are imposed by applications, the focus of Wolfson et al. is on determining threshold values according to so-called update, deviation, and uncertainty costs. The specifics of our policies differ from those of Wolfson et al. For example, they do not utilize the geometry of the underlying road network and thus do not consider issues such as map matching.

In other work, Wolfson et al. [13] propose immediate linear and delayed linear update policies. These are not dead reckoning policies, as a moving object does not update its location when the deviation reaches some threshold. When to update is dependent upon the overall behavior of the deviation since the last update. Deviation is estimated by a linear function of time. Experiments on simulated data [14] show that the policies mentioned in the previous paragraph are superior to the two policies described here.

Both works mentioned above assume that objects move on predefined routes. They also assume that clients have information about the routes. If an object changes its route, it sends a position update that includes a new route id. Our work differs by not making assumptions about predefined routes or, for that matter, the existence of a road network, and we do not require storage of routes on the mobile clients.

Gowrisankar and Nittel [2] propose a dead-reckoning policy that uses angular and linear deviations together, and that sends an update when one of these deviations exceeds a predefined threshold. They assume that the moving objects travel on predefined routes, and they do not cover aspects such as polyline representations of road segments and map matching, which underlie the segment-based tracking policy.

Lan et al. [15] propose an adaptive monitoring method. This method is based on dead-reckoning approach and takes into consideration not only update, deviation, and uncertainty costs, but also the cost of providing incorrect query results to mobile clients. The moving objects are divided into two groups. Objects that fall into a query region need close monitoring, so a small update threshold is used for them. Objects outside the query region can have larger threshold values. Our policies allow a different threshold for each object and allow these to be changed dynamically.

When the precision of a moving object's location is of low importance, so-called cellular techniques can be used for tracking [1, 11, 12]. Here, a cellular mobile network tracks the locations of the mobile clients in real time in order to be able to deliver messages or calls to the clients. With these techniques, update is handled by the mobile network. For large threshold values, on the order of several kilometers, cellular network accuracy is enough. We concentrate on handset-based positioning and significantly higher precision in the positioning. As a result, our proposals differ substantially from these.

We have found no experimental studies of update policies for moving objects that use real data.

10 Summary, Conclusions and Future Work

The overall setting is one where a central server tracks the positions of a population of mobile clients equipped with GPS receivers. We propose to have the server share a representation, or prediction, of a client's near-future position with the client. The client then monitors the deviation between its GPS position and its predicted position, and it send an update when needed to meet the precision guarantee.

The paper proposes tracking techniques that offer precision guarantees while attempting to reduce the volume of updates sent from the clients to the server. This is important, as a high volume of updates imposes significant communication costs as well as client and server side costs.

Specifically, three position prediction policies are proposed. The point policy uses constant position predictions, the vector policy approximates a moving object's trajectory by a linear movement at constant speed, and the segment-based policy approximates a moving object's trajectory by the shape of the road on which it travels while assuming constant speed.

The paper's proposal uses map matching to find the road on which an object is moving.

Several conclusions can be drawn from the empirical evaluation of the update policies reported in this paper:

1. The prediction of a moving object's future position allows a substantial reduction of the volume of updates.

2. For GPS logs from cars moving in a semi-urban environment, the point policy results in a relatively high number of updates.

3. The segment-based policy is more advanced than the other two update policies, as the road network is used for position prediction. The positioning of moving objects w.r.t. a road network enables services that utilize content that is also positioned w.r.t. the road network (which is commonplace and for which linear referencing is used).

4. The segment-based policy is sensitive to the number of visited segments.

We find that all three tracking policies introduced here are quite fundamental. While the point policy was not competitive for the data used, there is data and thresholds for which it is competitive. And while the segment-based policy has the greatest potential because it positions objects in the underlying transportation network, this policy applies only to network-constrained movement, assumes that a representation of the road network is available, and works only when map matching is successful. For other movement and to ensure robustness of the segment-based policy, the vector policy is attractive.

Several extensions to the work presented here are possible.

It would be worth evaluating the costs of data transmission. This evaluation may lead to more precise estimates of the actual cost savings for services that use the different update policies.

It would also be of interest to modify the road network, to offer a segmentation that is better suited for the segment-based policy. This may be done by simply trying to create longer segment, or it may be possible to modify the road network according to the movement data collected from all users. Using such data, smaller segments could be merged into longer ones according to their usage. This may well provide longer segments for the majority of users and thus increase performance of the segment-based policy.

Finally, it would be of interest to investigate the update load on the database for the general tracking scenario.

Acknowledgments

This work was supported in part by a grant from the Electronics and Telecommunications Research Institute, South Korea, and through contract number IST-2001-32645 from the European Commission.

References

[1] I. F. Akyildiz and J. S. M. Ho. A Mobile User Location Update and Paging Mechanism Under Delay Constraints. *ACM-Baltzer Journal of Wireless Networks*, 1:244–255, 1995.

[2] H. Gowrisankar and S. Nittel. Reducing Uncertainty In Location Prediction Of Moving Objects In Road Networks. *Conference on Geographic Information Science*, 2002. http://www.spatial.maine.edu/˜nittel/publications/giscience02_hari.pdf

[3] C. Hage, C. S. Jensen,T. B. Pedersen, L. Speičys, and I. Timko. Integrated Data Management for Mobile Services in the Real World. *VLDB Conference*, pp. 1019–1030, 2003.

[4] J. R. Herring (ed.). The OpenGIS Abstract Specification. Topic 1: Feature Geometry (ISO 19107 Spatial Schema). Open GIS Consortium, 2001.

[5] C. S. Jensen. Research Challenges in Location-Enabled M-Services. *Third International Conference on Mobile Data Management*, pp. 3–7, 2002.

[6] C. S. Jensen, H. Lahrmann, S. Pakalnis, and J. Runge. The INFATI Data. *Aalborg University, A DB Technical Report*, 2003. http://www.cs.auc.dk/TimeCenter

[7] P. Jonathan, P. Munson, and V. K. Gupta. Location-based notification as a general-purpose service. *The 2nd International Workshop on Mobile Commerce*, pp. 40–44, 2002.

[8] R. Jose, A. Moreira, H. Rodrigues, and N. Davies. The AROUND Architecture for Dynamic Location-Based Services. *Mobile Networks and Applications*, 8(4):377–387, 2003.

[9] E. Kaasinen. User Needs for Location-Aware Mobile Services. *Personal and Ubiquitous Computing*, 7(1):70–79, 2003.

[10] H. Lahrman. Intelligent Farttilpasning-forsøg med unge førere i Nordjyllands amt. Techical note. *Aalborg University*, 2002.

[11] G. Li, K. Lam, and T. Kuo. Location Update Generation in Cellular Mobile Computing Systems. *International Parallel & Distributed Processing Symposium*, p. 96, 2001.

[12] Z. Naor and H. Levy. Minimizing the Wireless Cost of Tracking Mobile Users: An Adaptive Threshold Scheme. *IEEE INFOCOM*, pp. 720–727, 1998.

[13] O. Wolfson, S. Chamberlain, S. Dao, L. Jiang, and G. Mendez. Cost and Imprecision in Modeling the Position of Moving Objects. *ICDE Conference*, pp. 588–596, 1998.

[14] O. Wolfson, A. Prasad Sistla, S. Camberlain, and Y. Yesha. Updating and Querying Databases that Track Mobile Units. *Distributed and Parallel Databases*, 7(3):257–387, 1999.

[15] K. Yiu Lam, O. Ulusoy, T. S. H. Lee, E. Chan, and G. Li. An Efficient Method for Generating Location Updates for Processing of Location-Dependent Continuous Queries. *Database Systems for Advanced Applications*, pp. 218–225, 2001.

A Projection of a Point onto a Line Segment

We introduce vectors $\overline{a} = p - m, \overline{b} = s.p_1 - s.p_0, \overline{c} = m - s.p_0$ (see Figure 12), and a variable w for which the following equation holds:

$$\overline{c} = w\overline{b} \qquad (1)$$

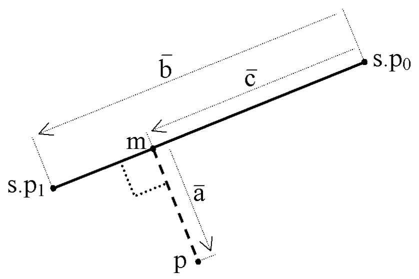

Figure 12: Distance from Point to Segment

Expression $\overline{c} = m - s.p_0$ inserted into Equation 1 produces:

$$w\overline{b} = m - s.p_0 \tag{2}$$

$$m = w\overline{b} + s.p_0 \tag{3}$$

Then vector $\overline{a}$ can be expressed in the following way:

$$\overline{a} = p - m = p - w\overline{b} - s.p_0 \tag{4}$$

The scalar product of vectors $\overline{a}$ and $\overline{b}$ is equal to zero, as the angle between them is 90^0.

$$\overline{a} \cdot \overline{b} = 0 \tag{5}$$

After inserting the expression for vector $\overline{a}$ from Equation 4 and the expression for vector $\overline{b}$ into Equation 5, we get:

$$\begin{aligned}\overline{a} \cdot \overline{b} = (p.x - w(s.p_1.x - s.p_0.x) - s.p_0.x) \cdot \\ (s.p_1.x - s.p_0.x) + (p.y - w(s.p_1.y - s.p_0.y) - \\ s.p_0.y)(s.p_1.y - s.p_0.y)\end{aligned} \tag{6}$$

$$\begin{aligned}(p.x - s.p_0.x)(s.p_1.x - s.p_0.x) - w(s.p_1.x - \\ s.p_0.x)^2 + (p.y - s.p_0.y)((s.p_1.y - s.p_0.y) - \\ w(s.p_1.y - s.p_0.y)^2 = 0\end{aligned} \tag{7}$$

Now w can be expressed as:

$$\begin{aligned}w = \big((p.x - s.p_0.x)(s.p_1.x - s.p_0.x) + \\ (p.y - s.p_0.y)(s.p_1.y - s.p_0.y)\big)\big/|s.p_1 - s.p_0|^2\end{aligned} \tag{8}$$

Substitution of w into the equation of the line gives the coordinates of the point m:

$$m.x = s.p_0.x + w(s.p_1.x - s.p_0.x) \tag{9}$$

$$m.y = s.p_0.y + w(s.p_1.y - s.p_0.y) \tag{10}$$

When $0 \le w \le 1$, point m is on vector $\overline{b}$.

Algorithm $\mathbf{PPLS}(p, s)$ uses Equations 8, 9, and 10.