Efficient Tracking of Moving Objects with Precision Guarantees

Alminas Čivilis[1] Christian S. Jensen[2] Jovita Nenortaitė[3] Stardas Pakalnis[2]

[1]Department of Computer Science II, Vilnius University, Lithuania
[2]Department Computer Science, Aalborg University, Denmark
[3]Department of Computer Science, Kaunas Faculty of Humanities, Vilnius University, Lithuania

Abstract

Sustained advances in wireless communications, geo-positioning, and consumer electronics pave the way to a kind of location-based service that relies on the tracking of the continuously changing positions of an 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.

This paper presents several representations, along with associated update techniques, that predict the present and future positions of moving objects. An update occurs when the deviation between the predicted and the actual position of an object exceeds a given threshold. For the case where the road network, in which an object is moving, is known, we propose a so-called segment-based policy that 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 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 technologies 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.

Several studies of potential LBSs and their architectures can be found in the literature (e.g., [9, 10, 11]).

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. The costs associated with updates include the following:

1. *Communication costs*. A high volume of updates may overload and degrade the performance of a wireless network. Additionally, when using a third–party mobile carrier for data transmission, a high volume of updates increases the cost of the tracking.
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 [7]. 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 and real-time traffic status information.

Each update policy is evaluated using real data, 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 present and 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.

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. Section 8 offers a detailed presentation of related work.

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. Section 5 presents the main algorithms that run on the server and client sides. Algorithms of the update policies and empirical studies of their performances, are given in Sections 6 and 7. Section 8 covers 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 7, 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. 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 [8] 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.

Definition 3.1 (Moving Object) The position of a moving object is defined as a point $p = (x, y)$ in two-dimensional Euclidean space $\mathcal{R}^2$. (The set of all points is denoted by $\mathcal{P}$.) The representation of a moving object *mo* is a three-tuple $(p, \overline{v}, t)$, where $p \in \mathcal{P}$ is the initial position of the moving object, $\overline{v} = (dx, dy) \in \mathcal{R}^2$ is the velocity vector of the moving object, and $t \in \mathbb{Q}_+$ is the time when p and $\overline{v}$ were measured.

This three-tuple can be obtained directly from a GPS receiver. We also use a representation of a moving object's position in terms of a road network. To define this, we need data structures for modeling a road network.

Definition 3.2 (Polyline) A polyline $pl = (p_0, p_1, \ldots, p_n)$, where $n \geq 1$, is an $(n+1)$-tuple of coordinates points $p_i \in \mathcal{P}$. We term p_0 the start point and p_n the end point of the polyline. The *direction* of a polyline is from its start point to its 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.

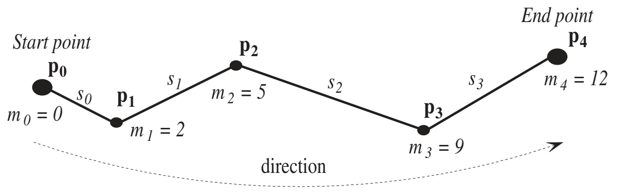


Figure 1. Polyline

Definition 3.3 (Line Segment) A line segment is a polyline with precisely two points. 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 3.4 (Measure) The measure (m) of a point p located on a polyline is the 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 $\mathcal{M} : \mathcal{PL} \times \mathcal{P} \mapsto \mathcal{R}$ is defined as follows:

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

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 $d(p_i, p)$.

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

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

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 3.5 (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 3.6 (Moving Object on a Polyline) The representation of a moving object on a polyline (MOP) is a four-tuple $mop = (pl, m, plspd, t)$. Here, $pl \in \mathcal{PL}$ is the polyline on which the moving object is located; $m \in \mathcal{R}$ is the measure giving the moving object's location on pl; $plspd \in \mathcal{R}$ is the moving object's signed speed along the polyline; and $t \in \mathbb{Q}_+$ is the time when the preceding values are valid.

This representation assumes that the object moves along a known polyline.

Definition 3.7 (Server Object) The information sent by the sever to a client is a server object, so, which is a three-tuple $(policy, ob, thr)$. Here, $policy \in \{$*'point'*, *'vector'*, *'segment'*$\}$ identifies the update policy used; ob is the representation of the moving object's position on the server ($policy$ = *'segment'* implies an mop; otherwise, it is an mo); $thr \in \mathcal{R}$ is a threshold.

3.3. System Architecture

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

3.4. Database Schema

The data required by the different update policies is stored in a database. Logically, this is divided into (1) data

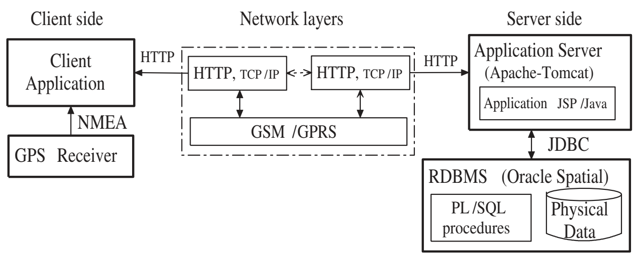


Figure 2. System Architecture

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.

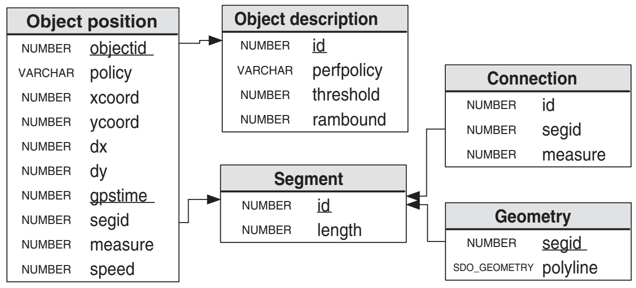


Figure 3. Database Schema

The fields $policy$ and field ob from the so structure (Definition 3.7) 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 $plsp\ d$ 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*.

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 [6]. 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. 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

Ideally, when an object traveling along a road reports its GPS position to the server, that position should intersect with the digital representation of the road. However, in practice this might not be the case. First, GPS positions are imprecise and may well deviate by up to about 30 meters from the true position. Second, the digital representation of a road network's geometry may deviate from the road network's real geometry. For these reasons, a non-trivial procedure must exist that map matches GPS positions onto segments 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 "+"'s represent reported GPS positions, and small squares on the road network represent map matched positions.

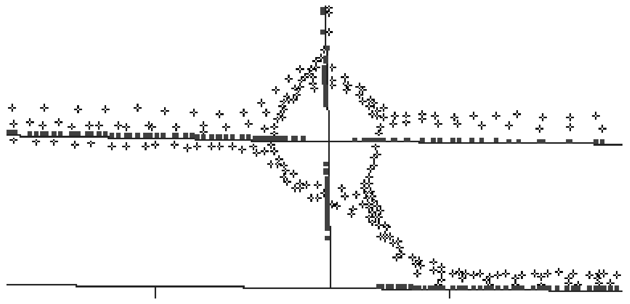


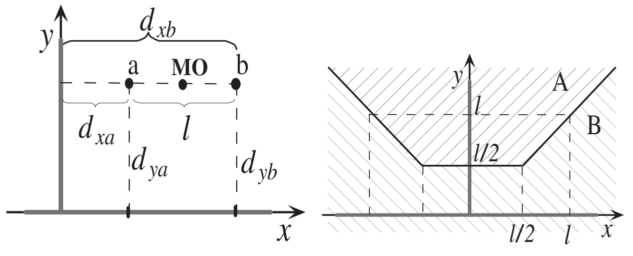
Figure 4. Road-Network Representation and GPS Positions

Map 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 segment among these on which the position should most probably be located on is identified.

In the second step, simply selecting the closest road segment may result in the selection of a road segment that is perpendicular to the movement direction of the object. Therefore, such road segments are eliminated, as are segments that are not accessible according to the *Connection* table.

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 5(a). Here, it is assumed that the x and y axes 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.



(a) Distance Calculation (b) Map Matching Regions

Figure 5. Map Matching

The distance l between a and b influences the selection of a road segment, as shown in Figure 5(b). 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 (they 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 [2]. The experimental evaluation covered in Section 7 suggests that the map matching described in this section is effective.

5. Server and Client Side Cycles

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

5.1. Server-Side Auxiliary Algorithms

We initially describe two server-side algorithms.

First, algorithm $\mathbf{GPDP}(pl, p, \overline{mov})$ ("Get Point Direction on Polyline") determines the direction on a polyline of a moving object. The direction is expressed as a signed number, where 1 means that object moves from the start of the polyline to the end, while -1 indicates the opposite direction (Figure 6).

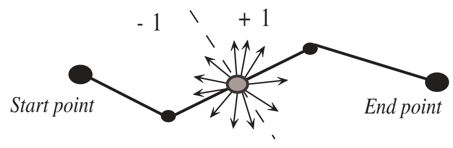


Figure 6. A Point's Direction on a Polyline

The direction of the object with a location given by a point p on a polyline pl is determined by comparing the object's velocity vector $\overline{mov}$ with the direction of the line segment on which object p is located. The direction of a line segment is defined by the vector from the start point to the end point of the segment.

Next, algorithm $\mathbf{MM}(p, \overline{v}, r, RN)$ ("Map Match") 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 p, a velocity vector $\overline{v}$, a maximum allowed distance r between position p and its projection onto a polyline, and a road network RN.

For each polyline reachable within distance r of point p, the algorithm calculates a weight according to the strategy described in Section 4. If no polylines are within distance r, the result is undefined. Otherwise, point p is projected onto the polyline with the least weight, and the result is the projection together with the selected polyline.

5.2. Server-Side Cycle

The main 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 5.1 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 $mpl = undefine$ dthen $so \leftarrow$ ('vector', $mo$, $threshold$)
(5)     else
(6)        $mop.plspd \leftarrow \mathbf{GPDP}(mpl, mp, mo.\overline{v})$
(7)        $mop.pl \leftarrow$ cut $mpl$ according to direction
                 and the client's memory limitations
(8)        $mop.m \leftarrow \mathcal{M}(mop.pl, mp)$
(9)        $so \leftarrow$ ('segment', $mop$, $threshold$)
(10)    end if
(11) else
(12)    $so \leftarrow (prefpolicy, mo, threshold)$
(13) end if
(14) store $so$ in the database and send it to the client
(15) 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, an *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 client request.

5.3. 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 5.2 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 $p_{pred} \leftarrow \mathbf{PPP}\ (so.mo)$
(8)        elsif $so.policy$ = 'vector' then $p_{pred} \leftarrow \mathbf{PPV}(so.mo, mo.t)$
(9)        elsif $so.policy$ = 'segment' then
(10)          $p_{pred} \leftarrow \mathbf{PPS}\ (so.mop, mo.t)$
(11)       end if
(12)    until $d(p_{pred}, mo.p) > so.threshold$
(13)    send the current position $mo$ to the server
(14)    receive server object $so$ from the server
(15) 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 lines 4–15 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.

6. 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 6.1 $\mathbf{PPP}(mo)$

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

Algorithm 6.2 $\mathbf{PPV}(mo, t_{cur})$

```
(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 to the start 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 6.3 $\mathbf{PPS}(mop, t_{cur})$

```
(1) $m_{pred} \leftarrow mop.m + mop.plspd\ (t_{cur} - mop.t)$
(2) if $m_{pred} >= \mathcal{M}\ (mop.pl, p_n)$ then return $mop.pl.p_n$
(3) elsif $m_{pred} <= 0$ then return $mop.pl.p_0$
(4) else
(5)    return $\mathcal{M}^{-1}(mop.pl, m_{pred})$
(6) 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 plus the distance traveled since the last update (line 1). The traveled distance is negative if the object moves against the direction of the polyline. If the new measure is outside the polyline, the result is one of end points of the polyline (lines 2 and 3). This way, the position prediction stops at a boundary point of the polyline.

7. 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 frequency 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 7. Here, precision threshold values in meters are on the x axis. The client receives a GPS position from the GPS device every second and performs a comparison between the GPS position and the predicted position. The y axis presents the average number of seconds in-between consecutive updates sent from the client to the server in order to maintain the required precision.

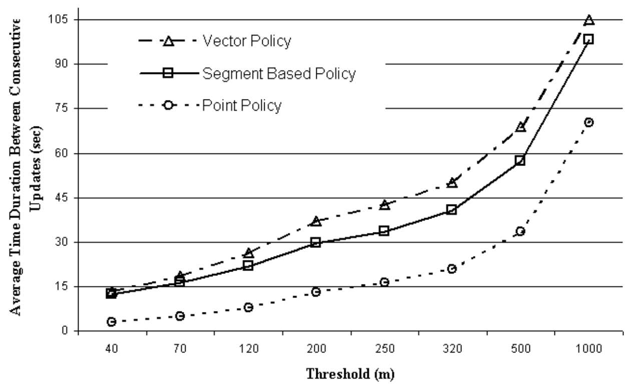


Figure 7. Comparison of Update Policies

The time between updates increase 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 8 offers results from a more detailed study of the update performance of the segment-based policy. Here, the y axis presents the percentage of positions received by the client from its GPS device 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.

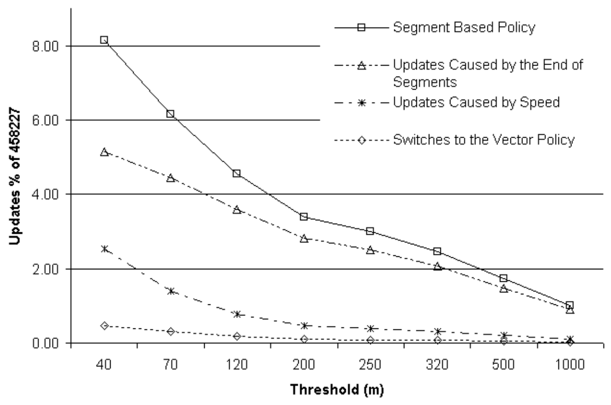


Figure 8. Analysis of Segment-Based Policy

The solid line presents the same results as in Figure 7. The number of updates required for some 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 8 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 have less influence on the total number of updates. And only few updates where done due to switches to the vector policy. This study illustrates the segment-based policy's dependence on the representation of the road network. In particular, it indicates that the policy is quite sensitive to the lengths of segments.

Next, the study indicates that the map matching technique used rarely fails to map a GPS position to a segment, for the map data and GPS logs considered. It should also be noted that our particular use scenario is map-matching friendly. The segment-based policy effectively switches to the point policy at ends of segments, which is typically at intersections. This switch has the effect that map matching is not attempted within the tracking threshold of intersections. And this is where correct map matching is difficult.

8. Related Work

Several existing contribution relate to the paper's contributions. Wolfson et al. [15] 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. [14] 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. Experiments on simulated data [15] show that the policies mentioned in the previous paragraph are superior to these two policies.

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 that routes are stored on the mobile clients.

Gowrisankar and Nittel [5] give an abstract overview of a dead-reckoning policy that uses angular and linear deviations together, and that sends an update when one of these deviations exceeds a threshold. They assume that objects travel on predefined routes, and they do not cover aspects, such as representations of road segments and map matching, that underlie the segment-based tracking policy.

Lan et al. [16] propose an adaptive monitoring method. 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 position accuracies on the order of several kilometers suffice, tracking can use cellular-network based positioning [1, 12, 13]. With this kind of positioning, updates are handled by the mobile network. Unlike for GPS, speeds and headings are not available. We concentrate on handset-based positioning and significantly higher precision in the position tracking.

Assuming a network of geo-stationary "presence" sensors, Goel and Imielinski [4] propose to use an MPEG-based prediction model in order to determine the current location of an object while using as little sensor battery power as possible.

Next, Fox et al. [3] explore the use of statistical methods, e.g., multiple hypothesis tracking, in a more abstract location estimation context than the one we consider. Integration of such methods into our setting may enable more detailed analysis of the proposed tracking techniques.

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

9. 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 to the server 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.

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 segment-based policy uses map matching to find the road on which an object is moving, and, to be robust, it switches to the vector policy when map matching does not succeed.

Five notable conclusions can be drawn from the empirical evaluation of the update policies reported in this paper. First, the prediction of a moving object's future position allows a substantial reduction of the volume of updates. Second, for GPS logs from cars moving in a semi-urban environment, the point policy results in a relatively high number of updates. Third, 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 common). Fourth, the segment-based policy is sensitive to the number of visited segments. Fifth, the map matching technique used by the segment-based policy is effective.

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. The paper currently uses the number of updates as a basis for comparison. While this is correlated with the costs of data transmission, it would be of interest to evaluate these costs more accurately. This evaluation may lead to more precise estimates of the actual cost savings for services that use the different update policies. A detailed study of the update load on the database for the general tracking scenario is also warranted.

It would 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 is of interest to attempt to integrate insights reported by Goel and Imielinski [4] and Fox et al. [3] into our setting.

Acknowledgments

The authors wish to thank the anonymous reviewers for their constructive comments.

This work was supported in part by a grant from ETRI, South Korea, and through contract number IST-2001-32645 from the European Commission. In addition to his primary affiliation with Aalborg University, C. S. Jensen is an adjunct professor in Department of Technology, Agder University College, Norway.

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] A. Čivilis, C. S. Jensen, J. Nenortaitė, and S. Pakalnis. Efficient Tracking of Moving Objects with Precision Guarantees. *Aalborg University, Department of Computer Science. DB Technical Report 5*, 23 pages. 2004. Available via: <http://www.cs.auc.dk/DBTR>.

[3] D. Fox, J. Hightower, L. Liao, D. Schultz, and G. Borriello. Bayesian Filters for Location Estimation. *IEEE pervasive Computing*, 2(3): 24–33, 2003.

[4] S. Goel and T. Imielinski. Prediction-Based Monitoring in Sensor Networks: Taking Lessons from MPEG. *ACM Computer Communication Review*, 31(5), 17 pages, 2001.

[5] H. Gowrisankar and S. Nittel. Reducing Uncertainty In Location Prediction Of Moving Objects In Road Networks. *Conference on Geographic Information Science*, 2002.

[6] 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.

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

[8] C. S. Jensen, H. Lahrmann, S. Pakalnis, and J. Runge. The INFATI Data. *Aalborg University, Department of Computer Science. DB Technical Report*, 10 pages, 2003.

[9] 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.

[10] 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.

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

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

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

[14] 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.

[15] 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.

[16] 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.