A Hybrid Algorithm for Online Location Update using Feature Point Detection for Portable Devices

Jibum Kim[1], Inbin Kim[1], Namgu Kwon[1], Heemin Park[2], and Jinseok Chae[1]
[1] Department of Computer Science and Engineering, Incheon National University
Incheon 406-772, South Korea
[e-mail: {jibumkim, dlsqlsdl, knkky11, jschae}@incheon.ac.kr]
[2] Department of Computer Software Engineering, Sangmyung University
Cheonan 330-720, South Korea
[e-mail:heemin@smu.ac.kr]
*Corresponding author: Jinseok Chae

Received May 27, 2014; accepted December 8, 2014; published February 28, 2015

Abstract

We propose a cost-efficient hybrid algorithm for online location updates that efficiently combines feature point detection with the online trajectory-based sampling algorithm. Our algorithm is designed to minimize the average trajectory error with the minimal number of sample points. The algorithm is composed of 3 steps. First, we choose corner points from the map as sample points because they will most likely cause fewer trajectory errors. By employing the online trajectory sampling algorithm as the second step, our algorithm detects several missing and important sample points to prevent unwanted trajectory errors. The final step improves cost efficiency by eliminating redundant sample points on straight paths. We evaluate the proposed algorithm with real GPS trajectory data for various bus routes and compare our algorithm with the existing one. Simulation results show that our algorithm decreases the average trajectory error 28% compared to the existing one. In terms of cost efficiency, simulation results show that our algorithm is 29% more cost efficient than the existing one with real GPS trajectory data.

Keywords: Location-Based Service, Corner Detection, Moving Object Tracking, Online Trajectory Sampling

This work was supported by the Incheon National University Research Grant in 2012.

http://dx.doi.org/10.3837/tiis.2015.02.007 **ISSN : 1976-7277**

1. Introduction

Due to the rapid growth of GPS technology and embedded systems, portable devices such as smart phones are able to easily obtain their locations while they are moving. Therefore, many useful applications based on location (location-based services (LBS)) are now emerging as promising technologies [1, 2, 3, 4, 5]. LBS applications need periodic location updates from their portable host devices to track their trajectories. Many location update services are designed to perform in a streaming fashion; however, too frequent location updates could result in large data storage costs for the server and possibly significant battery consumptions of the mobile devices. On the other hand, if location updates occur too seldom, it is hard to track the location of moving objects. Additionally, the server might not be able to interpolate and reconstruct the entire trajectories given minimal location information. Because of these trade-off relationships, setting location update strategies are considered as optimization problems [2, 6, 7]. There have been many optimization-based location update studies to find the minimal location information to describe the original trajectories. However, most algorithms employ an offline approach in which the server collects the entire trajectory of the moving object and optimizes the location information (sample points). However, we consider more practical environments in which the mobile device itself decides whether to update its online location. Here, online means that the moving object decides whether to update its location to the server while it collects location information in real time.

In this paper, we focus on an online location update algorithm that tracks the trajectory of a moving object in real time. The goal of our online location-update algorithm is to decrease the average trajectory error while minimizing the number of location update points to decease the communication overhead. Here, the trajectory error means the difference between the actual mobile device trajectory and the trajectory reconstructed from the sample points. To achieve this goal, we efficiently combine feature point detection with our previously proposed online trajectory-based sampling algorithm.

2. Related Work

Many location update methods for moving objects have been proposed in the literature [2, 6, 8, 9, 10, 11]. However, most previous work focused on off-line-based location update methods, which assumes that the LBS server (simply, server) first collects the entire trajectory of the mobile device and optimizes the location update points to save storage [2, 6, 11]. We are interested in online-based location update methods in which the mobile device itself decides whether to update the current location in real time. This online-based location update approach is challenging because the mobile device and server have no information on the future trajectory but must decide on the fly whether to accept the current location. Among many existing online location update methods, one of the most straight-forward is a time-based trajectory sampling (location update) method. This method periodically (regularly in time) updates the locations of moving objects. As expected, the time-based location update method can provide an accurate trajectory of a moving object to the server, but this method often results in too frequent communication and thus often is burdened with increased communication and data storage costs [8, 12, 13, 14]. Velocity-based location update methods improve the time-based location update method by adaptively updating the location of the moving object according to the velocity of the moving object [2]. It turns out that the

velocity-based location update method decreases the communication cost compared with the time-based approach. Among online location update methods, one of the most popular is a distance-based method. This method initially predicts a threshold distance from the previous location update point within which the portable device stays. Portable devices update their locations when they move out of this prediction distance. It was reported that the distance-based location update method decreases the frequency of redundant location updates because the device does not update its location if it stays within this prediction distance. However, one of the difficulties of using this algorithm is to find the optimal prediction distance for various situations. Additionally, if the trajectory of the mobile device is simple or constantly moves straight, redundant location updates occur often. To improve the existing location update algorithms, Park et al. recently proposed a trajectory error-based (TB) location update algorithm that can decrease the number of location update (sample) points [8]. This method updates the location of a moving object only if the current trajectory exceeds a predefined trajectory error. Here, trajectory error means the difference between the actual trajectory and the trajectory reconstructed from location update sample points. This trajectory error could be considered as a predefined Quality of Service (QoS) level of the mobile device. This algorithm decreases the number of sample points compared with the distance-based approach, and therefore significantly decreases the trajectory data and communication cost between the server and the portable devices. However, the TB algorithm chooses sample points that are not located at roadway corners or intersections. Therefore, while the TB algorithm minimizes the number of sample points, their locations are not optimal and produce unnecessary trajectory errors.

3. Problem Formulation

3.1 System Model

We assume that the mobile device updates its locations to the server and the server stores the updated location information. A trajectory of the mobile device is a sequence of GPS location points (sample points) in ascending timestamp order of its positions on the map and denoted as $T=\{s_0, s_1, s_2, \ldots, s_n\}$. Here, n is the number of sample points in the trajectory T.

Location updates of the mobile device occur in a discretized manner and the server does not know intermediate locations between the last and current sample points. Therefore, the server must interpolate the trajectory between the last updated sample point and the current location. There exist many interpolation methods, but we assume that the server employs a *piecewise linear interpolation* method to reconstruct the entire trajectory, given locations of these discrete sample points from the mobile device. We use the piecewise linear interpolation method because it has good performance in terms of reconstructing trajectories and ease of implementation. In most cases, the *actual trajectory* where a mobile device moves and the *reconstructed trajectory* using the interpolation method are not same because the location update (sampling) occurs in a discretized manner. We define trajectory errors as follows: the set of distances between the actual trajectory of mobile devices and the trajectory reconstructed with the sample points. The *trajectory error* between two sample points is computed as average distances between the actual location on the actual trajectory and the estimated location on the reconstructed trajectory. In a similar way, the *average trajectory error* is computed as the average of trajectory errors for all sample points up to the current time.

The estimated location is computed by *linear interpolation* between the current sample point and the last sample point. Similar to [8], we use the great circle distance method to compute the geological distance between the actual location and the estimated location. Details of computing the trajectory error and the great circle distance between two geological locations are well described in [15]. **Fig. 1** shows an example of trajectory error when two sample points are given.

Obviously, the trajectory error decreases as the number of sample points increases because the

reconstructed trajectory becomes close to the original trajectory. However, if the number of sample points increase, the networking and the database storage cost at the server and battery consumption of the mobile device will also increase. Therefore, it is desired to minimize the number of sample (location update) points, while minimizing the *average trajectory error*. The most straightforward trajectory sampling method so far is a time-based trajectory sampling method. This method updates sample points to the server regularly in time (e.g., every 2 seconds). However, it was reported that many time-based trajectory sampling methods suffer from redundant sample points and therefore result in unnecessary networking and storage costs at the server. Our goal is to choose, in real-time, optimal sample point locations while minimizing the average trajectory error at the server.

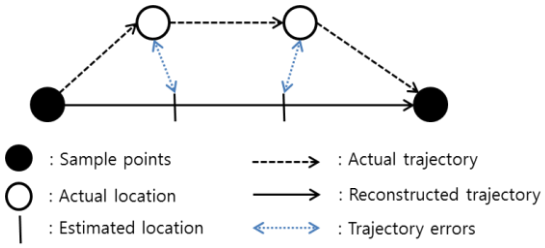


Fig. 1. Actual trajectory of the mobile device and trajectory reconstructed from sample points.

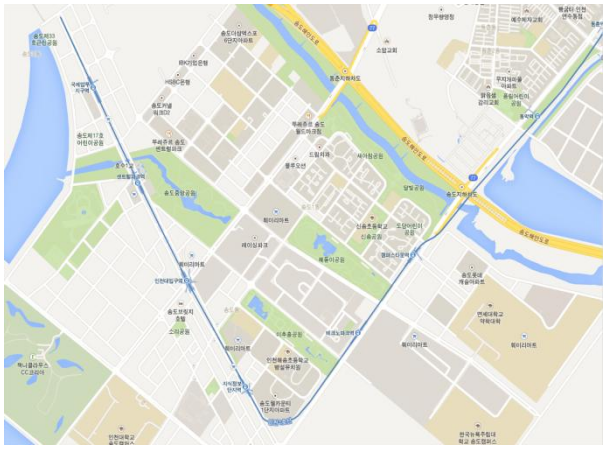
3.2 Feature (Corner) Point Detection

We assume that the server possesses an entire map (e.g., google map) for the region of interest where the mobile device is travelling. Given a map image file, the server finds feature points by exploiting a corner detection algorithm developed in the computer vision community. The motivation for using corner points as feature points is as follows. We noted that trajectory error mainly occurs when the mobiles change their direction suddenly (e.g., turning at intersections.) Our idea is to use these corner points as sample points to decrease the *trajectory error*. To the best of our knowledge, none of the previous work on optimal online (real-time) trajectory sampling methods has considered using these corner points from the map image as sample points to minimize trajectory errors.

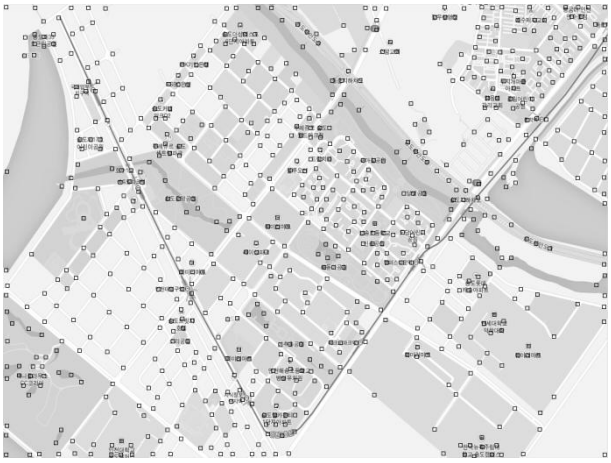
The server detects feature points by computing corner points on the map using the Harris corner detection algorithm offline [16]. The Harris corner detection algorithm is a robust corner detection algorithm popular in computer vision communities. We employ the algorithm because it is easy to implement and fast, but any efficient corner detection algorithm can be used if it can provide accurate corner information from the given map images.

Given a map image file (e.g., a jpg file), corners are defined as pixels in the image at which large changes in appearance occur. That is, corners are feature points where significant changes occur in all directions. Because we are interested in trajectory of mobile devices, corners correspond to curves, corners, or intersections on the road in the map image. We believe that these corner points should be strong candidates for sample points to update because large direction changes occur around them. Additionally, if we skip these corner points as sample points, trajectory error would increase. These significant changes in the map image file are computed by using image gradients. The server finds as many corner points as possible and only extracts corner points that are located on roads; it also eliminates duplicate corner points. **Fig. 2** shows an example of finding corner points using the Harris corner detection algorithm. The map of Songdo (**Fig. 2(a)**) in Incheon, South Korea, was acquired from Google maps [17]. Corner points detected, using the Harris corner detection algorithm, are marked as small squares in **Fig. 2(b)**. Extracted corner points (blue points) located on a road are shown in **Fig. 2(c)**.

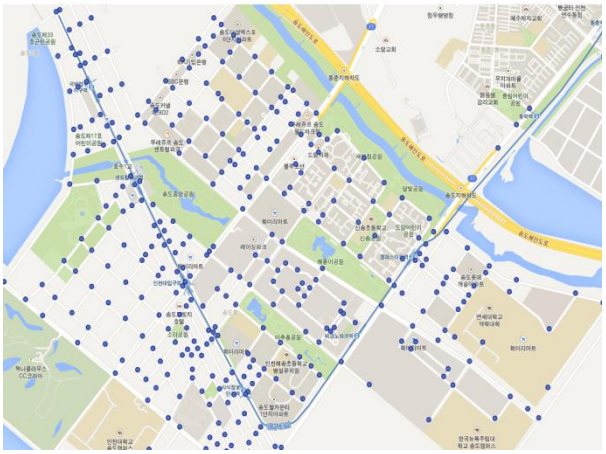
In practice, these detected corner points can be stored at the mobile device; alternatively, the server can report the points to the mobile device in real time when the mobile device approaches them.



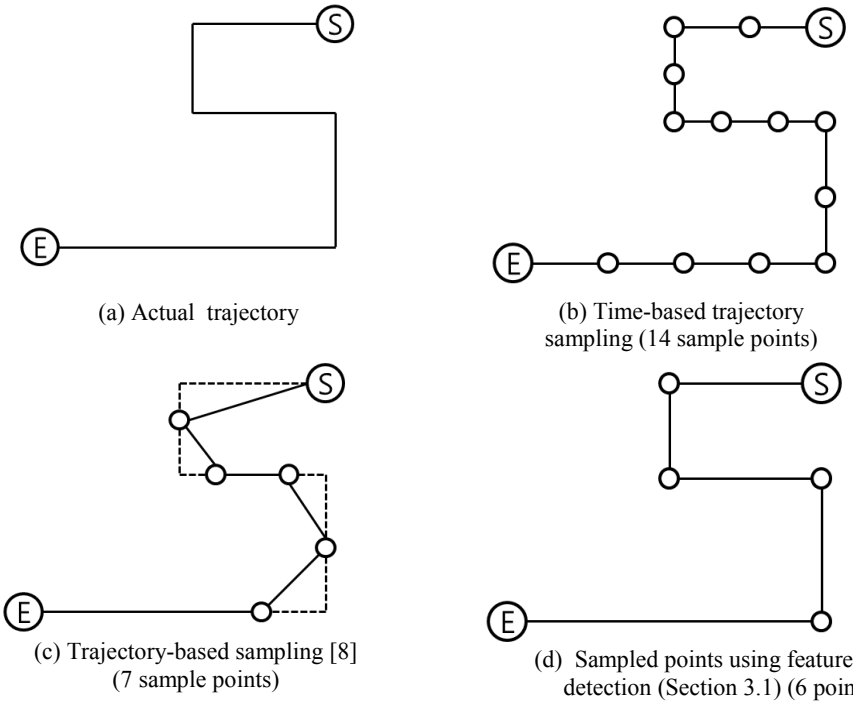
points)(a) Songdo map in Incheon, South Korea from Google map



Corner detection using Harris corner detection algorithm



(c) Extract corner points that are located on roads



(a) Actual trajectory

(b) Time-based trajectory sampling (14 sample points)

(c) Trajectory-based sampling [8] (7 sample points)

(d) Sampled points using feature point detection (Section 3.1) (6 points)

Fig. 3. Example of trajectory tracking and sampling. Trajectory-based sampling algorithm results in a large interpolation error as shown in (c).

3.3 Online Trajectory-based Sampling

There are many online (real-time) trajectory sampling algorithms, but one of the most efficient so far is called the online trajectory-based sampling (TB) algorithm, proposed by Park et al. [8]. The idea of the TB algorithm is to choose sample points at which the trajectory error exceeds a predefined error threshold. Here, the predefined error threshold is considered a Quality of Service (QoS) value and is agreed upon between the server and the mobile device. The TB algorithm is motivated by the fact that existing online trajectory sampling algorithms produce many redundant sample points, especially when the mobile device travels along straight roads. Park et al. noted that many distance-based online trajectory sampling algorithms suffer from these redundant sample points. Additionally, the distance-based algorithm has large trajectory errors if the speed of the mobile device varies. To solve these problems, the TB algorithm does not report sample points as long as the trajectory error is smaller than some threshold value. If a mobile device keeps moving straight, the number of sample points becomes small compared with distance-based algorithms.

The problem with the TB algorithm arises when the mobile device changes its direction frequently (e.g., when the mobile device encounters intersections or makes turns). Because the TB algorithm is not designed to preferably choose corner points as sample points, it always waits and chooses sample points where the trajectory error is greater than the threshold value. Therefore, in most cases the TB algorithm does not choose corner points as the sample points. For this reason, the TB algorithm produces unnecessary trajectory errors. **Fig. 3** highlights the problem of the TB algorithm. Figure 3(a) shows the actual trajectory on which the mobile device actually moves from a start point (S) to an end point (E). Time-based sampling (**Fig. 3(b)**), which updates sample points regularly in time, has 12 sampled points with no trajectory error. The reconstructed trajectory coincides with the actual trajectory when piecewise linear interpolation is used. The TB algorithm uses 5 sample points, which prevent the trajectory error from exceeding the predefined threshold value, δ, as shown in **Fig. 3(c)**. We observe that no sample points are located on corners because the TB algorithm does not update sample points until

the trajectory error exceeds the error threshold value. These cases occur when the mobile device has already passed an intersection. If we use corner points as sample points as shown in **Fig. 3(d)**, only 4 sample points are required and there is no trajectory error for this case.

3.4 Hybrid Online Location Update Algorithm

Our hybrid online location update algorithm is composed of 3 steps: feature (corner) point detection, trajectory-based sampling with a threshold value δ, and an additional trajectory-based sampling with the threshold value $\tau = \delta/2$. The first step (step 1 in **Fig. 4(a)**) is detection of feature points using the Harris corner detection algorithm as described in Section 3.2. The server finds all feature (corner) points on the given map image files and reports the corner points to the mobile device in real time when the mobile device approaches them. We expect that these corner points are strong candidates to become sample points because they contribute to decreasing trajectory errors.

For simple trajectories such as that shown in **Fig. 3(a)**, the output sample points after step 1 do not produce any trajectory errors or might have small trajectory errors. However, for real-life situations, it is possible that the corner detection algorithm might not detect corner (feature) points because of limitations of the corner detection algorithm or incorrect map information. For these cases, the output sample points after step 1 might have unexpected trajectory errors. As a second step (step 2 in **Fig. 4**), we employ our previously proposed TB algorithm [8] explained in Section 3.3 to prevent these unexpected trajectory errors. The reason we adopted the TB algorithm as a second step is to suppress the maximum trajectory error below some threshold value (δ). This δ value could be considered as a Quality of Service (QoS) value of the online location update algorithm. The TB algorithm can always guarantee that the maximum trajectory error is less than or equal to given threshold value. The maximum trajectory error is always bounded because the algorithm adds sample points whenever the current trajectory error exceeds the threshold value. We expect that this step adds several sample points that our first step fails to detect.

As a final step (step 3 in **Fig. 4**), we employ the second TB algorithm with a smaller threshold value ($\boldsymbol{\tau} = \delta/2$) than in step 2. The real map could have many road intersections; these intersections are considered feature points in our step 1. However, as long as the mobile device moves straight and does not turn, we do not need to consider all the corner points in intersections as sample points. These points at intersections are meaningful and should be final sample points only if the mobile device changes its direction there (e.g., turns). Otherwise, the corner points are redundant sample points, and need to be eliminated. For this reason, we employ the TB algorithm at the last stage with a smaller threshold value ($\boldsymbol{\tau} = \delta/2$) to eliminate these redundant sample points mostly on straight paths. The TB algorithm does not update sample points as long as the trajectory error is below the threshold value. Therefore, all redundant sample points on a straight road can be eliminated. Our proposed hybrid online location update algorithm is summarized in **Fig. 5**.

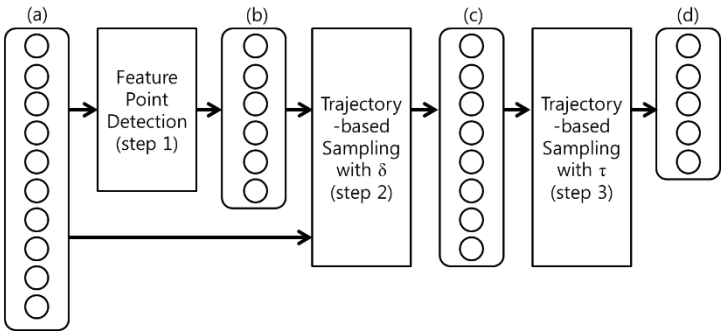


Fig. 4. System architecture of the proposed online sampling algorithm. A circle indicates a sampled location point. (a) Initially sampled points. (b) Output points after employing feature point detection (Section 3.2). (c) Output points after employing the online trajectory-based sampling algorithm with δ (Section 3.3). (d) Output points after employing the second online trajectory-based sampling with τ.

```
SelectSamples begin
     p_0 ← start location
     t_0 ←  start time
     p_tmp ← ∅ // temp location
     arr ← ∅ // temp array
     e_th ← threshold error
     s ← 0 // index of lastly sent location
     u ← 0 // index of lastly updated location in arr
     k ← 0 // index of current location
     whenever (new location information is available) begin
          k ← k + 1
          p_k ← current location
          t_k ← current time
          // Step 01. Feature point detection
          p_tmp ← GetCornerPoint(p_k, p_c) call
          if (p_tmp is not null) begin
         add p_tmp in arr
          u ← k
          end
          // Step 02. Trajectory-based sampling with δ
          p_tmp ← GetTrajectoryPoint (p_k, t_k, u, k, δ) call
          if (p_tmp is not null) begin
         add p_tmp in arr
          u ← k − 1
          end
          // Step 03. Trajectory-based sampling with τ
          if (arr is updated) begin
              arr_{p_u} ← lastly updated location in arr
              arr_{t_u} ← lastly updated time in arr
              p_tmp ← GetTrajectoryPoint (arr_{p_u}, arr_{t_u}, s, u, τ) call
              if (p_tmp is not null) begin
             send p_tmp to the server
                       s ← u − 1
         end
          end
     end
end
GetCornerPoint(p_k, p_c) begin
          p_k ← current location(argument)
          p_c ← all corner location
          s ← 0 // start index of corner location
          l ← last index of corner location
          For all intermediate count of corner points
                    d_{k→c} ← |p_k − p_{c_{s→l}}|
          if (d_{k→c} ≤ 30) begin
               return p_k
      end
end
GetTrajectoryPoint(p_k, t_k, s, k, e) begin
          p_k ← current location(argument)
          t_k ← current time(argument)
          s ← index of lastly saved location(argument)
          k ← index of current location(argument)
          For all intermediate points i, s < i < k
               p̂_i ← ((t_i − t_s)/(t_k − t_s))(p_s + p_k)
          e^{max}_{T_{s→k}} ← max_{s<i<k} |p̂_i − p_i|
          if (e^{max}_{T_{s→k}} > e) begin
               return p_{k−1}
          end
end
```

Fig. 5. Proposed hybrid online trajectory sampling algorithm

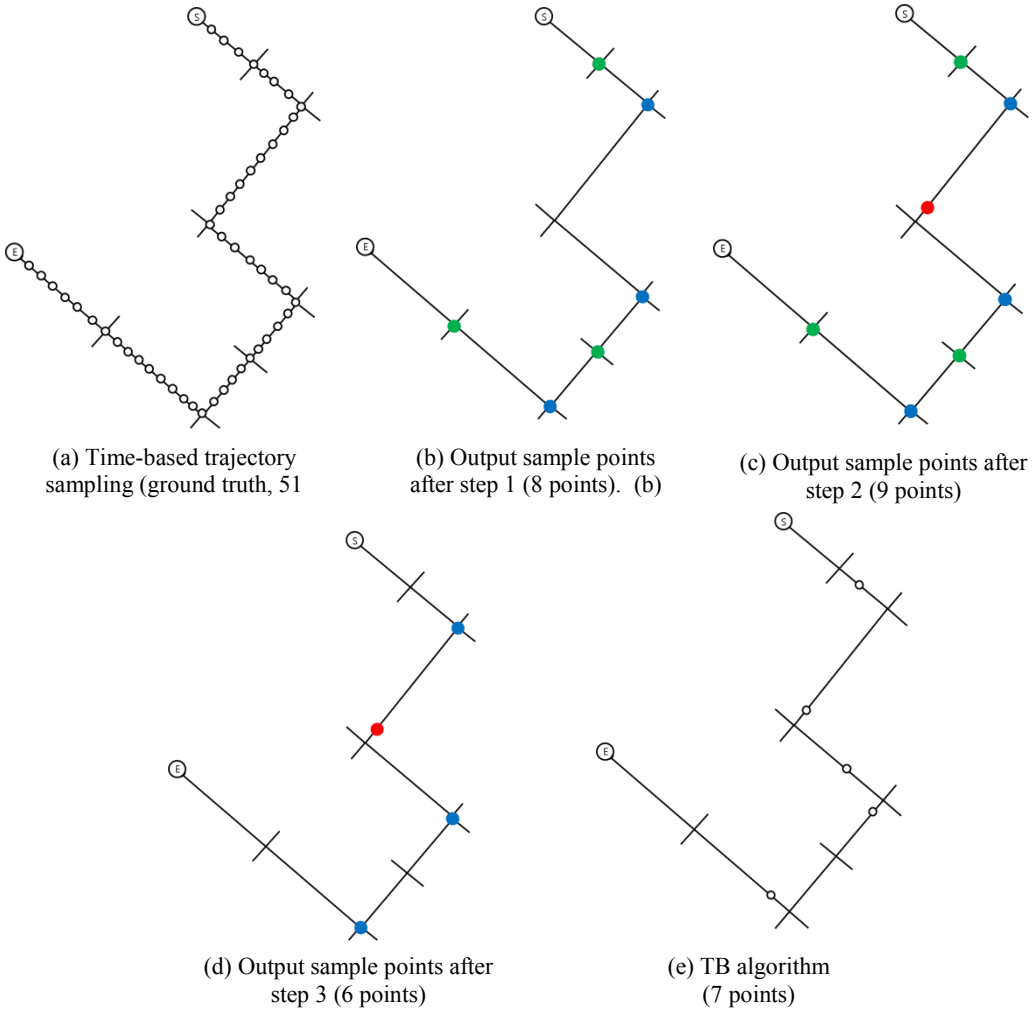
Fig. 6 shows the result of running our algorithm on an example trajectory. The time-based trajectory method samples 51 points as shown in **Fig. 6(a)**. We consider these initially sampled points to be close enough to the actual trajectory to consider as ground truth data. Our step 1 choses 8 sample points located at corners (here, intersections of the road). The green sampled points were redundant and needed to be eliminated. Our feature point detection algorithm found all corner points except that in the middle. We assume that the corner detection algorithm failed to detect this point due to a limitation of the corner detection algorithm itself. Here, redundant points (i.e., green sample points) were produced when the mobile made no turns at intersections (see **Fig. 6(b)**). Our step 2 compared the output sample points after step 1 with the ground truth and added several sample points if the trajectory error was greater than δ. Here, a red sample point was added because the trajectory error was greater than δ (see **Fig. 6(c)**). Finally, our step 3 removed redundant sample points (green) by employing the trajectory-based sampling algorithm again with ($\tau = \delta/2$) (see **Fig. 6(d)**). We observe that our step 3 was able to successfully remove redundant sample points (green) and finally just 6 sample points remained after step3. **Fig. 6(e)** shows the output sample points by running the only TB algorithm with δ. This figure shows that the TB algorithm does not choose corner points as sample points and therefore results in trajectory errors that could have been avoided.

4. Experimental Results

4.1 Experimental Setup

We used real GPS trajectory data to investigate the effectiveness of the proposed algorithm. We developed the TestGPS smart phone application to collect the GPS trajectory data. **Fig. 7 (a)** shows a screenshot of the TestGPS application running. The application has four menu buttons: Start, Stop, sendLog, and delLog. The first, Start, starts GPS data collection and the second, Stop, stops it. The 'sendLog' button sends the stored GPS information to the server and the 'delLoG' button deletes the stored GPS information. When we press the 'Start' button, the app starts storing the current location of the user; it stores Latitude, Longitude, Altitude, and the current time as an array. **Fig. 7 (b)** shows a screenshot of sample GPS trajectory data.

We collected 30 different bus trajectories (routes) in Incheon, South Korea, with the application. The bus trajectories were carefully chosen to avoid overlapping trajectories. Four sample bus trajectories are shown in **Fig. 8**. Here, initially sampled points (blue dots) are acquired every 3 seconds using the TestGPS app. We assume that these initially sampled points are dense enough to be a actual trajectory. Therefore, we assume that these initially sampled points, based on time-based trajectory sampling in **Fig. 8**, are actual trajectories (ground truth data). We used Matlab (R2013a) to implement our algorithm and the TB algorithm. Initially, we set the threshold value δ in step 2 as 60 meters and τ in step 3 as 30 meters (δ/2). The machine employed for this study was an Intel I7-2600 CPU 3.40 GHz with 8 GB main memory.



(a) Time-based trajectory sampling (ground truth, 51

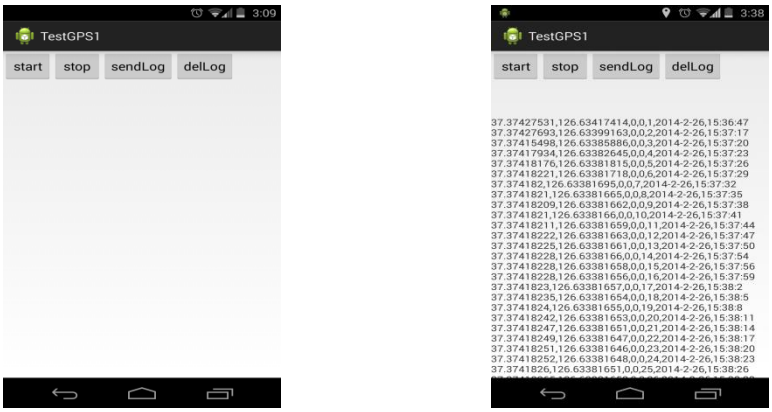
(b) Output sample points after step 1 (8 points). (b)

(c) Output sample points after step 2 (9 points)

(d) Output sample points after step 3 (6 points)

(e) TB algorithm (7 points)

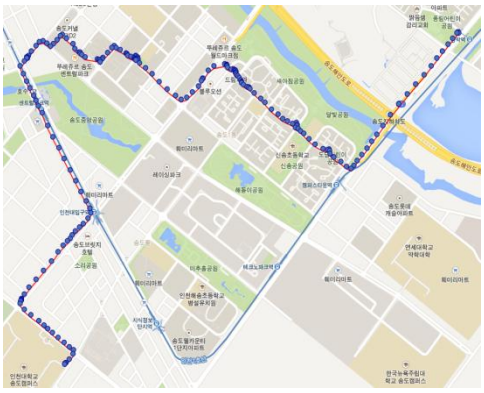
Fig. 6. Output sample points after each step. 'S' and 'E' are start and end points, respectively. (a) Time-based trajectory sampling results in 51 sample points. (b) Both green and blue sample points were detected as feature points after step 1. (c) Red sample point was added after step 2 because the trajectory error was greater than δ. (d) Green sample points, which are redundant, were eliminated after step 3 (e) Output sample points (black circles) of the TB algorithm



(a) TestGPS smartphone application

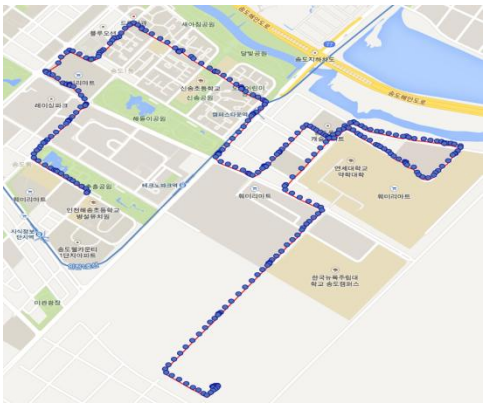
(b) GPS trajectory data of various bus routes collected by TestGPS application

Fig. 7. Screenshots of the TestGPS application on the Galaxy S3 smartphone.

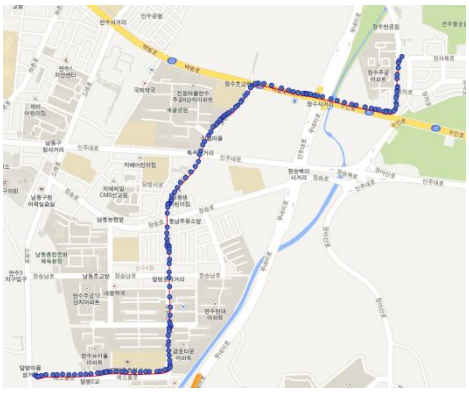


(a) Sample trajectory of bus 908 in Incheon, South Korea (196 points)

(b) Sample trajectory of bus 532 in Incheon, South Korea (277 points)



(c) Sample trajectory of bus 91 in Incheon, South Korea (372 points)



(d) Sample trajectory of bus 536 in Incheon, South Korea (139 points)

Fig. 8. Examples of bus trajectories in Incheon, South Korea, using the TestGPS application. Sample points (blue dot) were stored every 3 seconds. We assume that these collected bus trajectories are actual trajectories (ground truth data).

4.2 Experimental Results

In general, the number of sample points and the average trajectory error have a trade-off relationship. Our goal is to provide cost-effective sample sets to decrease average trajectory errors while minimizing the number of sample points. Therefore, the reconstructed trajectory using location update points (sample points) should be close to the actual trajectory with the minimal number of sample points. The average trajectory error (e) is defined as

$$e = \frac{total\ trajectory\ error}{\#\ of\ sample\ points}$$

Here, the average trajectory error is measured in meters (m). Our previously proposed online trajectory based sampling algorithm (simply, TB algorithm) significantly outperforms existing online location update algorithms (e.g., distance based algorithms) in terms of the trajectory error and cost efficiency [8]. Therefore, we focus on comparing our proposed algorithm with the TB algorithm in terms of average trajectory error and cost efficiency.

4.2.1 Average Trajectory Error

We first evaluate whether our proposed online trajectory sampling algorithm decreases the average trajectory error compared with the existing online location update algorithm, namely, the TB algorithm. **Figs. 9** and **10** show the average trajectory error of the TB and the proposed algorithm for various bus routes. We observe that our algorithm decreases the overall average trajectory error approximately 28 % compared with the TB algorithm. In most cases, our proposed algorithm needs few more sample points, but it significantly decreases the average trajectory errors compared with the TB algorithm.

Fig. 11 highlights the strength of using our proposed algorithm. This route is for bus No. 512 in Incheon, South Korea. We chose this route because it has many intersections and the bus changes direction often. The figure shows that the TB algorithm skips the corner points and updates incorrect sample points. Therefore, the TB algorithm increases the average trajectory error (e=29.29 m) and the reconstructed trajectory (**Fig. 11(a)**) is not close to the actual trajectory (**Fig. 11(c)**). Our proposed algorithm significantly decreases the average trajectory error (e=17.76 m) by choosing corner points as feature points (see **Fig. 11(b)**). Next, we modify the threshold value (δ) of our step 2 and investigate the robustness of our proposed algorithm with respect to various δ values. **Fig. 12** shows the average trajectory errors with respect to various δ values. We observe that as δ value increases, the gap between the proposed algorithm and the TB algorithm also increases. We observe that the TB algorithm is more sensitive to the δ value than our proposed algorithm because the TB algorithm skips corners as sample points and chooses incorrect sample points

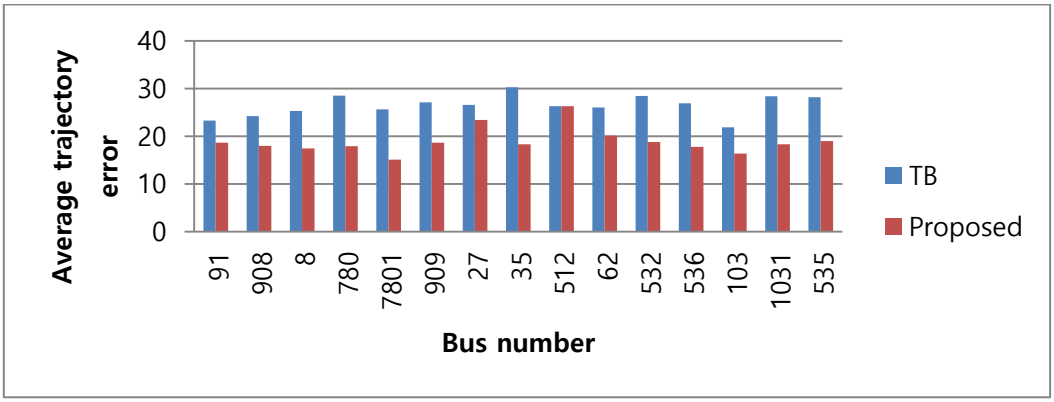


Fig. 9. Comparison of the average trajectory error (meters) between the TB algorithm and the proposed algorithm for the first set bust routes.

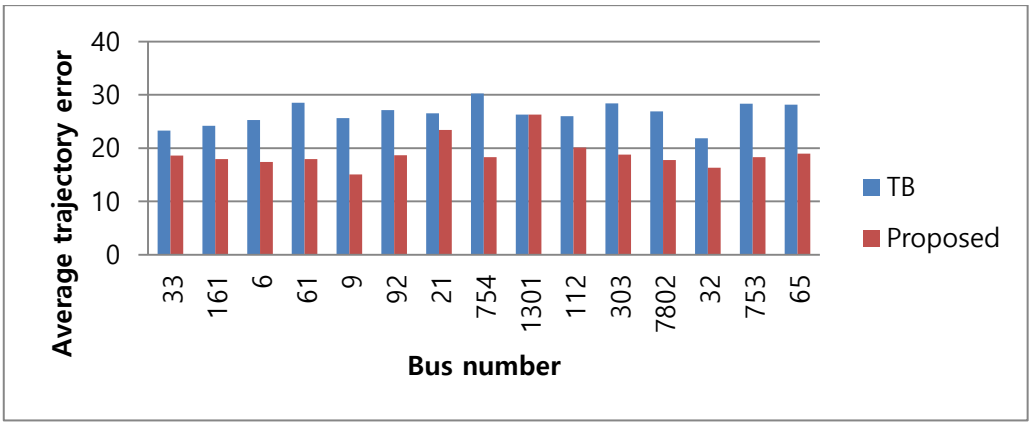


Fig. 10. Comparison of the average trajectory error (meters) between the TB algorithm and the proposed algorithm for the second set of bus routes.

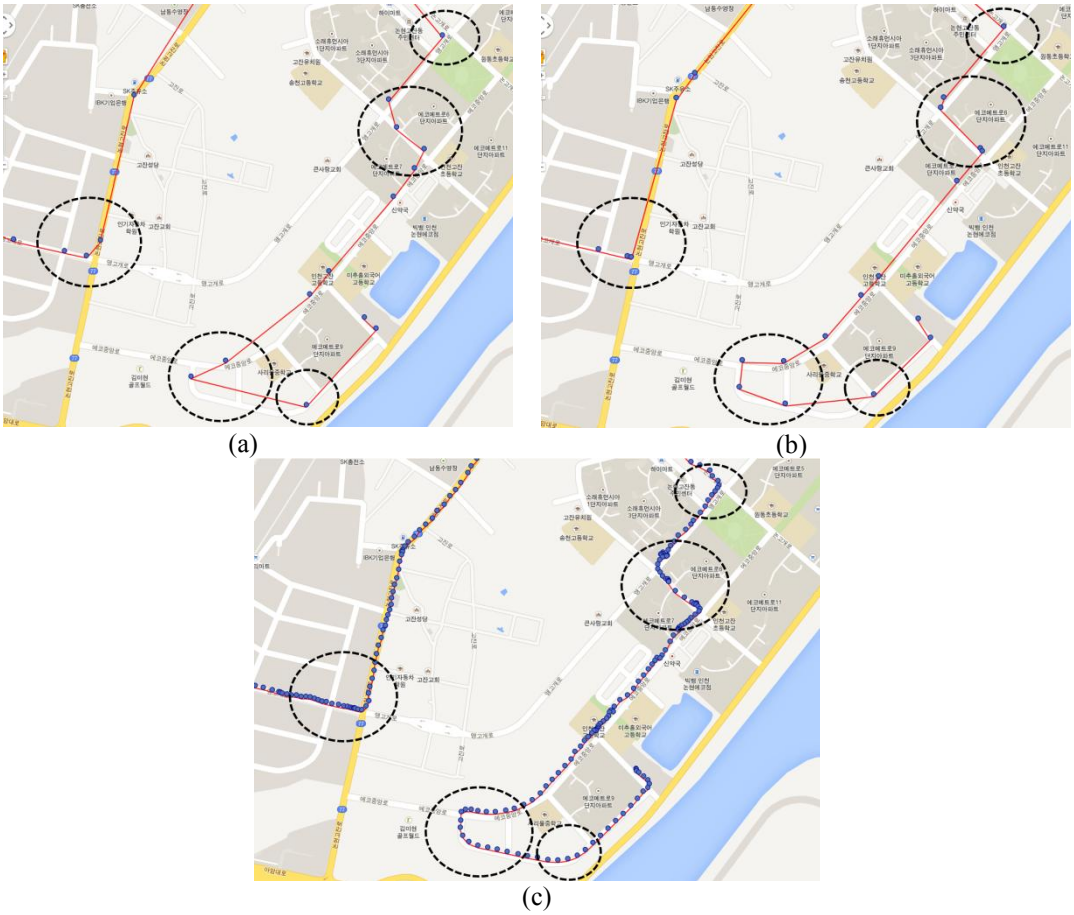


Fig. 11. Examples of employing (a) the TB algorithm and (b) the proposed algorithm for the 512 bus route in Incheon, South Korea. The blue circles and the red lines in (a) and (b) indicate the sample points and the reconstructed trajectory, respectively. The black circles highlight the strength of the proposed algorithm. The trajectory in (c) is the actual trajectory using time-based sampling.

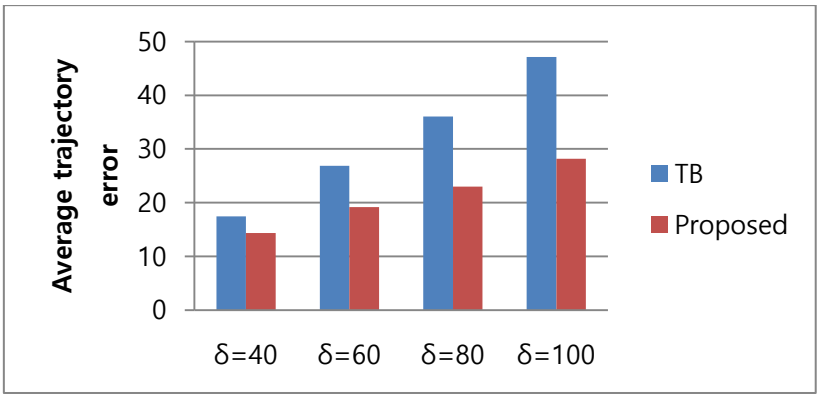


Fig. 12. Average trajectory error (meters) with respect to various threshold values (δ). Here, the average trajectory error is computed over all bus routes.

4.2.2 Cost Efficiency Ratio (Utilization)

The average trajectory error is inversely proportional to the number of sample points. We investigate how cost efficient our algorithm is compared with the TB algorithm. Let e_{TB} and e_{our} be the average trajectory errors of the TB algorithm and our proposed algorithm, respectively. Similarly, let N_{TB} and N_{our} be the number of sample points of the TB algorithm and our proposed algorithm, respectively. We define the cost efficiency ratio (μ) as following:

$$\mu = \frac{e_{TB}}{e_{our}} \cdot \frac{N_{TB}}{N_{our}}$$

If $\mu > 1$, then our algorithm efficiently decreases the average error using a minimal addition of sample points compared with the TB algorithm. Otherwise, we conclude that the TB algorithm is more cost efficient.

Table 1 shows the average error, the number of sample points, and μ values for various bus trajectories (routes) in Incheon, South Korea. For all bus trajectories, μ values are greater than 1; therefore, our algorithm has a better cost-efficiency ratio than the TB algorithm. These results indicate that our algorithm efficiently decreases the average trajectory error with minimal additional sample points. When δ value is 60m, our algorithm is approximately 18% more cost efficient than the TB algorithm. **Fig. 13** shows the μ values with respect to the various threshold (δ) values in our step 1. Similarly to **Fig. 11**, the cost efficiency increases as the δ value increases and our algorithm is more cost efficient than the TB algorithm for all δ values. For the route of bus No. 536 with δ=100, our algorithm is approximately 92% more cost efficient than the TB algorithm. The e_{TB} and e_{our} values for this bus route are 48.50 and 17.48, respectively, while the N_{TB} and N_{our} are 9 and 13, respectively.

4.2.3 Sensitivity of the Second Threshold Value, τ, in Step 3

Our algorithm sets the second threshold value, τ, as δ/2 because the second threshold value should be smaller than the first threshold value. To show the robustness of our proposed algorithm with respect to various τ values, we fix the first threshold value (δ) as 60 m and modify the τ value between 20 m and 40 m and investigate the robustness. **Fig. 14** shows the average trajectory error with respect to various τ values. For all τ values, our algorithm results in smaller trajectory errors than the TB algorithm. When τ=20 m, our algorithm has 34% smaller trajectory errors than the TB algorithm. **Fig. 15** shows the μ values for various τ values. For all τ values, our algorithm has better cost efficiency than the TB algorithm. When τ=100 m, our algorithm is approximately 29% more efficient than the TB algorithm.

Table 1. Summary of the average trajectory error, number of sample points, and cost efficiency ratio for various bus numbers in Incheon, South Korea. Here, δ=60 and τ=30.

Bus Number		91	908	8	780	7801	909	27	35
TB Algorithm	e_{TB}	26.41	27.37	26.35	25.84	24.66	28.35	27.63	29.27
	N_{TB}	47	25	21	31	28	27	16	20
Proposed Algorithm	e_{our}	24.07	18.32	20.90	20.16	18.91	24.83	19.80	21.83
	N_{our}	48	33	24	34	32	25	19	23
μ		**1.07**	**1.13**	**1.10**	**1.16**	**1.14**	**1.23**	**1.17**	**1.16**

Bus Number		512	62	532	536	103	1031	535	33
TB Algorithm	e_{TB}	29.29	26.13	27.82	25.16	28.50	29.22	27.58	23.30
	N_{TB}	36	11	22	13	10	8	13	10
Proposed Algorithm	e_{our}	17.76	17.34	16.25	14.27	17.37	15.83	23.68	18.62
	N_{our}	40	14	30	15	14	12	15	10
μ		**1.48**	**1.18**	**1.25**	**1.52**	**1.17**	**1.23**	**1.01**	**1.25**

Bus Number		161	6	61	9	92	21	754	1301
TB Algorithm	e_{TB}	24.18	25.29	28.53	25.61	27.10	26.53	30.23	26.77
	N_{TB}	8	26	28	13	34	9	6	19
Proposed Algorithm	e_{our}	17.97	17.43	17.92	15.08	18.67	23.43	18.32	26.27
	N_{our}	9	33	37	15	43	10	9	19
μ		**1.19**	**1.14**	**1.20**	**1.47**	**1.20**	**1.47**	**1.14**	**1.02**

Bus Number		112	303	7802	32	753	65
TB Algorithm	e_{TB}	26.00	28.41	26.90	21.84	28.36	28.17
	N_{TB}	7	15	18	12	18	28
Proposed Algorithm	e_{our}	20.11	18.76	17.77	16.34	18.33	18.97
	N_{our}	8	20	22	16	24	31
μ		**1.13**	**1.13**	**1.23**	**1.01**	**1.16**	**1.34**

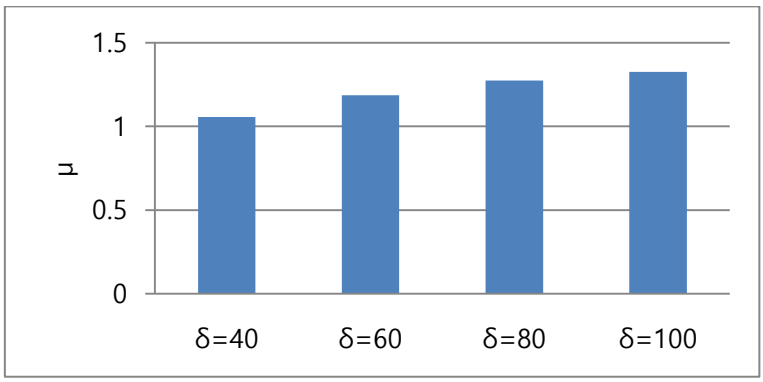


Fig. 13. Cost efficiency ratio with respect to various δ values in Step 1. Here, **μ** is the average **μ** value for all bus routes.

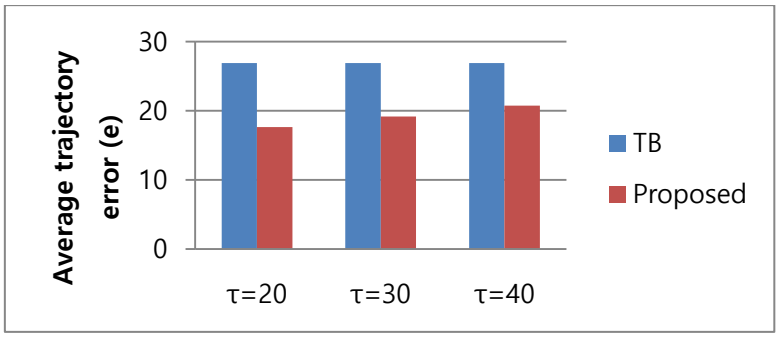


Fig. 14. Average trajectory error with respect to the second threshold value τ. Here, the average trajectory error is computed as the average *e* value for all bus routes.

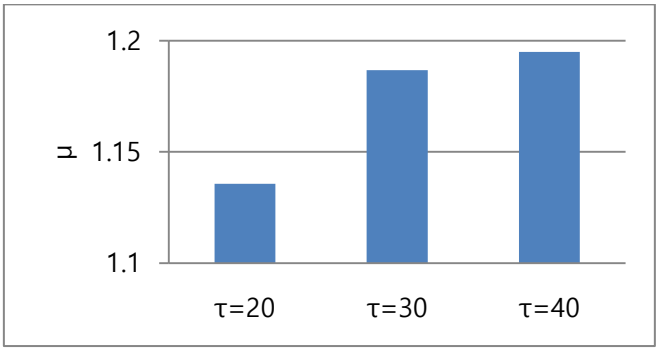


Fig. 15. Cost efficiency ratio with respect to the second threshold value τ. Here, **μ** is the **μ** for all bus routes.

4.2.5 Comparison with a Linear Dead-Reckoning Method

One of most popular online trajectory method in calculating moving object's position is Linear Dead-Reckoning (LDR) method [18, 19]. It is currently used in many navigation systems and location estimation systems due to its simplicity and ease of implementation. The LDR algorithm calculates and estimates moving object's current position by using both previously determined position and linear prediction of moving direction. The moving direction is obtained from the most recent update points. **Fig. 16** shows both the LDR method and its error model. The reconstructed trajectory by employing the LDR method consists of the sampled location points and moving directions. As shown in **Fig. 16**, the LDR method has a drawback in that it would yield discontinuities in the reconstructed trajectories.

We have compared our proposed method with the LDR method for the 30 bus routes we tested. Note that the definition of the trajectory error used for the LDR method and the proposed method is not same. The trajectory error of the LDR method is a distance between the moving vector and the location points in the original trajectory. On the while, the trajectory error of the proposed method is a distance between the location points in the original trajectory and interpolated location points from the reconstructed trajectory.

Since the error models for the LDR and the proposed methods are not same, we use these two different trajectory error models to compare the proposed method with the LDR method. When the trajectory error model employed in this paper is used, the proposed method shows better performance by 3.81 times than the LDR method in terms of the cost efficiency. For the 29 bus routes out of 30 bus routes, the proposed method outperformed the LDR method. If we use the error model used in the LDR for calculating cost efficiency, the proposed method is 1.49 times more cost efficient than the LDR method. The proposed method showed better results for the 17 bus routes out of 30 routes. The LDR method showed comparable results when its own error model is used.

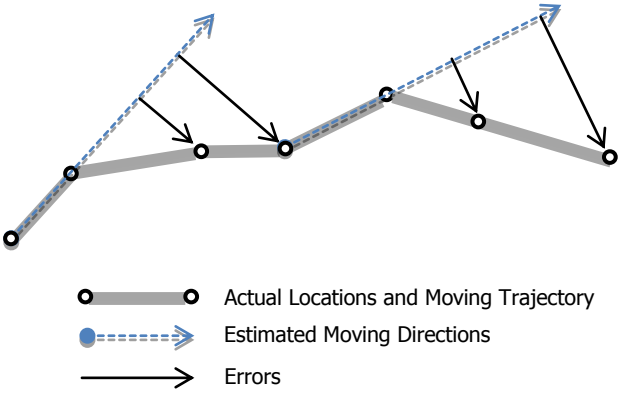


Fig. 16. Location estimation and the error model for the Linear Dead-Reckoning (LDR) Method.

However, the LDR method has several potential problems. First, the error model employed for the LDR method is not practical in that it could yield discontinuous reconstructed trajectories as shown in **Fig. 16**. Second, we observed that the LDR method results in poor performance when the moving object encounters corners or intersection. We observe similar problems for the TB method when the moving object encounters corners or intersections as shown in **Fig. 11**. This is mainly because these two existing online location update methods (i.e., TB and LDR) do not consider corners or intersections when they choose sample points. Also, the LDR method only cares about the trajectory shapes of the reconstructed trajectories, but does not consider timestamps when it predicts moving directions. In this case, users may have incorrect location information results when users query with some specific timestamps. Our proposed method chooses the error model where locations of missing points in the sampled trajectories are interpolated with queries' timestamps. Therefore, the proposed method is able to provide more accurate estimates of missing locations with timestamps from the sampled trajectories.

4.2.6 Transmission Delay Overhead

When the proposed online location update method is used, the server should report the corner points to the mobile device when the mobile device approaches to the corner points. The additional delay between the server and the mobile device could be generated when these pre-calculated corner points are transmitted. We have measured these additional delays for the tested 30 bus routes using the LG Optimus 4G cell phone. For the tested bus routes, we observe that the transmission delay is a dominant factor, which contributes to the actual delay and other delays including propagation delays are negligible compared with the transmission delay. The average transmission delay we measured between the server and the mobile device is approximately 0.21 sec for the tested bus routes.

We calculate the lower and upper bound of the (transmission) delay overhead when the proposed method is employed. Let the number of sample points for the TB algorithm and the number of corner points be N_{TB} and N_{corner}, respectively. For the worst case, the proposed algorithm utilizes both N_{TB} and N_{corner}, to choose its sample points, since this location information can be used in our step 1 and 2 of the proposed algorithm. Therefore, for the worst case, the delay overhead of the proposed algorithm compared with the TB algorithm is $(N_{TB} + N_{corner})/N_{TB}$. For the best case, the delay overhead becomes N_{corner}/N_{TB} when our first step of the proposed algorithm is good enough to detect all sample points with a bounded trajectory error. Let N_{LDR} be the number of sample points for the LDR method. Similarly, the delay overhead of the proposed algorithm compared with the LDR algorithm is $(N_{TB} + N_{corner})/N_{LDR}$ for the worst case and N_{corner}/N_{LDR} for the best case. For the 908 bus route, $N_{TB} = 25$, $N_{corner} = 37$, and $N_{LDR} = 58$.

In practice, the mobile device does not need to request the location of corner points in real time, since it can download all corner information when it installs our TestGPS application.

4.2.7 Summary of Experimental Results

Table 2 and 3 summarize the comparison results between the proposed method with two existing methods (TB and LDR) for the bus datasets in terms of the total number of sample points, average number of sample points, average trajectory error, total and the cost efficiency. Here, the cost efficiency is defined as

$$\mu = \frac{e_{existing}}{e_{our}} \cdot \frac{N_{existing}}{N_{our}}$$

Here, e and N are defined as the average trajectory and the number of sample points, respectively. Also, 'existing' means TB algorithm for the Table 2 and LDR algorithm for the Table 3. As mentioned earlier, LDR employed its own error model to compute the trajectory error [18, 19]. For a fair comparison, we use the LDR's error model [18, 19] when we compare our proposed algorithm with the LDR algorithm, since the LDR algorithm shows good performance when it uses its own error model. We employ δ=100m for the initial threshold value, since the previous experimental results show that the increase in a δ value results in better cost efficiency. Table 2 and 3 show comparison results between the proposed algorithm with two existing online location update algorithms (i.e., TB and LDR). Table 2 and 3 show that the proposed online trajectory algorithm is 29% more cost efficient than the TB algorithm and 27% more cost efficient than the LDR algorithm. These results indicate that the proposed algorithm is able to achieve same average trajectory with 29% and 27% fewer number of sample points compared with the TB and LDR algorithms, respectively.

Table 2. Comparision between the TB algorithm and the proposed algorithm for 30 bus routes

	Trajectory based algorithm [8]	Proposed algorithm
Total # of sample points	414	537
Average # of sample points	13.8	17.9
Average trajectory error (e)	47.1	28.2
Cost efficiency (μ)		1.29

Table 3. Comparison between the LDR algorithm and the proposed algorithm for 30 bus routes

	Linear dead-reckoning algorithm [18, 19]	Proposed algorithm
Total # of sample points	739	537
Average # of sample points	24.6	17.9
Average trajectory error (e)	26.0	28.2
Cost efficiency (μ)		1.27

5. Conclusion

We have proposed a hybrid algorithm for online location update that efficiently combines feature point detection and the online-trajectory-based sampling algorithm. The first step significantly decreases trajectory errors by choosing corners as sample points. This is motivated by the observation that trajectory errors mostly occur when the mobile device suddenly changes its direction. The second step is designed to detect several missing sample points in case our corner-detection algorithm fails to detect corner points in the map image. We observe that the trajectory error decreases further after employing our second step. Finally, the third step eliminates redundant sample points on straight paths and improves cost efficiency.

We tested our algorithm on real GPS trajectory data for various bus routes in Incheon, South Korea. Experimental results show that our algorithm has 28% smaller trajectory errors than the existing algorithms. In terms of the cost efficiency, our algorithm is approximately 29% and 27% more cost efficient than the TB and LDR algorithms, respectively. Experimental results show that our algorithm needs a few more sample points than the TB algorithm, but it is able to significantly decrease the

average trajectory error by choosing optimal sample point locations. The proposed algorithm requires 27% less number of sample points with similar number of average trajectory error. We also observe that our proposed algorithm is robust to the choice of various threshold values. Because of its portability, we use the Harris corner detection algorithm to find corner points in the map image. In future work, we plan to enhance the first step (to find feature points) in our algorithm by adopting more robust corner detection algorithms from the computer vision community.

References

[1] A. Dey, J. Hightower, E. de Lara, N. Davies, "Location-based services," *IEEE Pervasive Computing*, vol. 9, no. 1, pp. 11-12, 2010. Article (CrossRef Link)
[2] Y. Lee, M. Kim, J. Chae, W. Yoo, J. Lee, "An intelligent control scheme of moving objects based on smart phones," *Database Research*, vol. 28, no. 2, pp. 19-36, 2012.
[3] Y. Zheng, L. Zhang, X. Xie, W. Ma, "Mining interesting locations and travel sequences from GPS trajectories," in *Proc. of International Conference on World Wild Web*, pp. 791-800, April 20-24, 2009.
[4] K. Lee, S. Yang, "Three effective top-down clustering algorithms for location database systems," *Journal of Computing Science and Engineering*, vol. 4, no. 2, pp. 173-187, 2010. Article (CrossRef Link)
[5] S. Orlando, R. Orsini, A. Raffaeta, A. Roncato, C. Silvestri, "Trajectory data warehouses: Design issues and use cases," *Journal of Computing Science and Engineering*, vol. 1, no. 2, pp. 240-261, 2007. Article (CrossRef Link)
[6] Zheng, Q. Li, Y. Chen, X. Xie, W. Ma, "Understanding mobility based on GPS data," in *Proc. of International Conference on Ubiquitous Computing*, pp. 312-321, September 21-24, 2008.
[7] J. Zhou, V. Leong, Q. Lu, K. Lee, "Optimizing update threshold for distance-based location tracking strategies in moving object environments," in *Proc. of International Conference on World of Wireless, Mobile and Multimedia Networks*, pp. 1-8, June 18-21, 2007.
[8] H. Park, Y. Lee, J. Chae, W. Choi, "Online approach for spatio-temporal trajectory data reduction for portable devices," *Journal of Computer Science and Technology*, vol. 28, no. 4, pp. 597-604, 2013. Article (CrossRef Link)
[9] R. Lange, T. Farrell, F. Durr, K. Rothermel, "Remote real-time trajectory simplification," in *Proc. of IEEE International Conference on Pervasive Computing and Communications*, pp. 1-10, March 9-13, 2009.
[10] R. Lange, F. Durr, K. Rothermel, "Online trajectory data reduction using connection-preserving dead reckoning," in *Proc. of International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*, July 21-25, 2008.
[11] E. Pitoura, G. Samaras, "Locating objects in mobile computing," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 4, pp. 571- 592, 2001. Article (CrossRef Link)
[12] M. Potamias, K. Patroumpas, T. Sellis, "Sampling trajectory streams with spatiotemporal criteria," in *Proc. of International Conference on Scientific and Statistical Database Management*, pp. 275-284, July 3-5, 2006.
[13] N. Meratnia, R. de By, "Spatiotemporal compression techniques for moving point objects," in *Proc. of International Conference on Extending Database Technology*, pp. 765-782, March 14-18, 2004.
[14] H. Cao, O. Wolfson, G. Trajcevski, "Spatio-temporal data reduction with deterministic error bounds," in *Proc. of Joint Workshop on Foundations of Mobile Computing*, pp. 33-42, September 19, 2003.
[15] Calculation of great circle distance, website: http://en.wikipedia.org/wiki/Greatcircle_distance.
[16] C. Harris, M. Stephens, "A combined corner and edge detector," in *Proc. of Alvey Vision Conference*, pp. 147-151, 1988.
[17] Google Maps, website: https://www.google.com/maps/preview.
[18] A. Leonhardi, K. Rothermel, "A Comparison of Protocols for Updating Location Information," *Cluster Computing*, Vol. 4, No. 4, pp. 355-367, October 2001. Article (CrossRef Link)

[19] O. Wolfson, A. P. Sistla, S. Chamberlain, Y. Yesha, "Updating and Querying Databases that Track Mobile Units," *Distributed and Parallel Databases*, Vol. 7, No. 3, pp. 257-387, July 1999. Article (CrossRef Link)

Jibum Kim received B.S. and M.S. degrees in electrical engineering from Yonsei University, Seoul, Korea in 2003 and 2005, respectively and Ph.D. degree in computer science and engineering from the Pennsylvania State University in 2012. He was a postdoctoral fellow at the Los Alamos National Laboratory (Group T-5) in 2013. He is currently an assistant professor in the Department of Computer Science and Engineering at Incheon National University, Korea. His research interests include mobile computing, computational science and high performance computing.

Inbin Kim received B.S. degree in computer science and engineering from Incheon National University, Korea in 2014. He is working toward the M.S. degree in Graduate School of Computer Science and Engineering at Incheon National University. His research interests include internet software and mobile computing.

Namgu Kwon received B.S. degree in computer science and engineering from Incheon National University, Korea in 2013. He is working toward the M.S. degree in Graduate School of Computer Science and Engineering at Incheon National University. His research interests include internet software and mobile computing.

Heemin Park received his B.S. and M.S. degrees in computer science from Sogang University, Korea in 1993 and 1995, respectively, and Ph.D. degree in electrical engineering from the University of California, Los Angeles, in 2006. He is currently an assistant professor in the Department of Computer Software Engineering at Sangmyung University, Cheonan, Korea. Prior to joining Sangmyung University, he worked at Yonsei University, Sookmyung Women's University and Samsung Electronics in Korea. His research interests include web-based information systems, networked and embedded computing systems, cyber physical systems, multimedia applications, and entertainment computing.

Jinseok Chae received his B.S., M.S., and Ph.D. degrees in computer engineering from Seoul National University, Korea in 1990, 1992, and 1998, respectively. He is currently a professor in the Department of Computer Science and Engineering at Incheon National University, Korea. Prior to joining Incheon National University, he worked in the Engineering Laboratory at Seoul National University and the Korea Research Information Center, Korea. He was a visiting scholar in the department of computer science at California State University San Bernardino, USA in 2006. He served as an editor-in-chief of Journal of KIISE: Computing Practices and Letters from 2011 to 2014. His research interests include internet software, web technology and mobile computing.