# Gesture Input for GPS Route Search

Radu Mariescu-Istodor(✉) and Pasi Fränti

University of Eastern Finland, Joensuu, Finland
{radum, franti}@cs.uef.fi

**Abstract.** We present a simple and user-friendly tool for an efficient search from a spatial database containing GPS tracks. The input is a sketch of a route drawn by a user on a map by mouse, hand or other means. This type of interaction is useful when a user does not remember the date and time of a specific route, but remembers its shape approximately. We evaluate the efficiency of the retrieval when the shape given by the gesture is simple or complex, and when the area contains either a small or large number of routes. We use the Mopsi2014 route dataset to demonstrate that the search works in real time.

**Keywords:** GPS · Route · Gesture · Matching · Touchscreen · Draw

## 1 Introduction

GPS-enabled smartphones allow users to collect large amounts of location-based data such as geo-tagged notes, photos, videos and geographical trajectories hereafter referred to as *routes*. Mobile users track routes for reasons like: recording travel experiences, recommending a certain path and keeping track of personal statistics in sports such as hiking, running, cycling and skiing. A sample route collection is shown in Fig. 1. From a large collection like this, it is difficult to find a specific route unless user remembers the date when it was recorded. Otherwise the amount of data is overwhelming to perform systematic search from among all the records.

Many applications exist that allow users to track their movement; some of these are: *Sports Tracker*[1], *Endomondo*[2], *Strava*[3] and *Mopsi*[4]. Mopsi is a location-based social network created by the School of Computing at the University of Eastern Finland. Mopsi users can find out who or what is around. They can track their movements, share photos and chat with friends. Mopsi includes fast retrieval and visualization of routes [1] using a real-time route reduction technique [2]. Transport mode information is automatically inferred by analyzing the speed variance of the route [3]. Movement is classified as either: walking, running, cycling or car. Route similarity, novelty, inclusion and noteworthiness [4, 5] are computed by using cell representations of the routes created by a grid which covers the planet. Searching for spatially similar routes is done efficiently by indexing these cells.

[1] http://www.sports-tracker.com.
[2] https://www.endomondo.com.
[3] https://www.strava.com.
[4] http://cs.uef.fi/mopsi.


A. Robles-Kelly et al. (Eds.): S+SSPR 2016, LNCS 10029, pp. 439–449, 2016.
DOI: 10.1007/978-3-319-49055-7_39


**Fig. 1.** Route collection of user Pasi over the city of Joensuu, Finland is shown on left. The collection spans from 2008 to 2014. A circle-shaped route that we want to find is emphasized. Four attempts (all failed) to find this route by clicking the map are shown right.

We propose a real-time search for routes in the Mopsi collection by using gestures and pattern matching. The gesture is a hand-drawn input in the form of a free shape done on a map. The shape approximates the locations where the targeted route passes through. According to [6], this gesture can be classified as of the *symbolic* type, implying that it has no meaning when performed in other contexts (not using a map). Referring to the taxonomy in [7] the result of the gesture is to trigger a command: search for route(s) with given spatial characteristics. This search works by computing the similarity between the input gesture and every route in the database. The most similar route candidates are provided to the user.

Gestures have been used as a means to access menu items without the need to traverse large hierarchies. In [8], gestures are continuous pen traces on top of a stylus keyboard. This soft keyboard can be inconvenience as it wastes screen space unnecessarily. In our method, we use the underlying map as a canvas for drawing the gestures. On desktop computers, the gesture mode is explicitly activated by holding a hotkey while drawing the gesture by mouse. On touchscreens, we need to distinguish the gesture from normal map interaction (panning and zooming). In [9], it was discovered that it is possible to distinguish gesture from other touch events such as scrolling or tapping by buffering the touch events and analyzing the queue to determine if the sequence is a gesture or not. We use this method to activate the gesture mode, and neither designated area nor activation button are therefore needed.

Typically, symbolic gesture-based systems require the user to learn a set of symbols [6]. Our method is simpler as no learning of symbols is required. However, the user is expected to understand and be able to read maps because the roads, buildings and terrain elements such as forests, lakes and rivers are the key information used when giving the input. For example, user may draw the input by following a river front, road, or other landmarks visible on map. Users who have a large route collection benefit most from the gesture search. It is therefore fair to assume that these users have also the necessary skills to understand maps.

## 2 User Interface

### 2.1 System Overview

Let us assume that Mopsi user Pasi wants to review the statistics of a specific route from his collection but he does not recall the date. Pasi knows that the route is in Joensuu, Finland so he proceeds to move the map to this location. Figure 1 shows that Pasi has a large route collection in Joensuu. Let us further assume that he wishes to find the highlighted circular route. Exhaustive search among all the routes would not be reasonable so best change is to try to distinguish the route on map. In Mopsi, this is possible by clicking any individual route on the map. However, this is also difficult because the targeted route overlaps with many others.

Gesture search enables a user to search routes by drawing the sample shape of the desired route over the map. Figure 2 shows how Pasi's route is found by drawing a circular gesture on the map around the center of Joensuu. The search returns four possible candidates, including the one he was looking for.

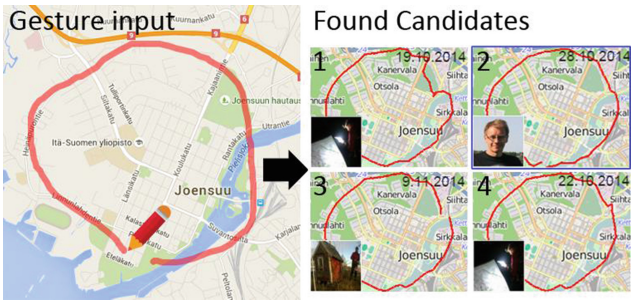


**Fig. 2.** A circle-shaped gesture surrounding the center of Joensuu reveals four circular route candidates. Pasi's route is number two in the list.

### 2.2 Map Handling

Mopsi uses Google Maps and OpenStreetMap. They both offer several built in functions for user interaction. A user can pan the map by clicking and dragging it in the desired direction. Zooming in can be done by double left-click and zooming out is done by double right-click. Zooming can be also done using the mouse wheel or by the pinch gesture.

To start the gesture search on a computer, user presses a hotkey (Ctrl). When pressed, the built-in map handling functions are disabled and the gesture input mode is enabled. In this mode, a user can draw on the map by clicking, holding and moving the mouse around while keeping the hotkey pressed. Releasing the hotkey causes two things to happen simultaneously: the input gesture is sent to the server for processing the query, and default map behavior is reactivated.

Majority of touchscreens nowadays do not have a keyboard and existing buttons serve for different purposes such as exiting applications, changing volume levels or enabling the camera. It is possible to implement a soft button on the screen to toggle the

gesture input mode however this wastes screen space which makes drawing more difficult, especially on small screens.

Instead, we activate the gesture first by a click (tap) and then, immediately, touch the screen again to draw the shape. We denote this event as *Tap-and-Draw*. The *Draw* event works similarly as panning the map, however, the preceding *Tap* event triggers gesture input mode. When the *Draw* gesture is complete, the input gesture is sent to the server and the search is initiated; default map behavior is reactivated.

### 2.3 Real-Time Route Search

The search returns the route(s) that are most similar to the shape of the gesture input. For the matching, we use the method in [5]. It computes the spatial similarity between routes by first representing them as cells in a grid and then using the Jaccard similarity coefficient:

$$J(C_A, C_B) = \frac{|C_A \cap C_B|}{|C_A \cup C_B|}, \tag{1}$$

where $C_A$ and $C_B$ are two sets of cells. However, because of the arbitrary division of the grid, route points may end up in different cells even though the points are close to each other. This problem is solved by applying morphological dilation with square structural element and using the additional cells as a buffer region when computing the similarity. The similarity is then formulated as:

$$S(C_A, C_B) = \frac{|C_A \cap C_B| + |C_A \cap C_B^d| + |C_B \cap C_A^d|}{|C_A| + |C_B| - |C_A \cap C_B|}, \tag{2}$$

where $C_A^d$ and $C_B^d$ are the dilated regions of routes $C_A$ and $C_B$ respectively. To make the search efficient we pre-compute the cell representation and use B-tree index [12] on the cell database. With this setup the search works real-time.

To perform the search, the input shape is converted into cells. The similarity between this cell set and all routes is then computed using (2). The result is similarity ranking which often contains a multitude of results with varying levels of similarity to the given shape. To the user we present only the most significant candidates.

### 2.4 Map Projection and the Grid

Most online maps (*Google Maps*, *OpenStreetMap*, *Yahoo! Maps*, *Bing Maps*) use a variant of the *Mercator* projection [10]. In Mopsi, we use Google Maps or OpenStreetMap as the map interface. *Mercator* is a cylindrical map projection which preserves the angles, however, the linear scale increases with latitude. The parallels and meridians are straight and perpendicular to each other. The meridians are equidistant, but the parallels become sparser as they further themselves from the equator.

Creating a grid by choosing a fixed cell size (in degrees) will cause the cells to appear vertically stretched when viewed on the Mercator projection. The amount cells stretch increases the farther away they are from the equator. In Joensuu, Finland the cells appear twice as tall as they are wide.

### 2.5 Multi-resolution Grids

The precision of drawing the gesture should be independent on the zoom level of the map. When the zoom level is decreased by one unit the content of the map becomes half of its previous size, and consequently, the regions on the map become twice as difficult to read. We create 10 grids with different resolutions and store the routes at each of these approximation levels (see Table 1).

The finest grid has a cell size of 25 × 25 meters. Finer grids are not needed because at this level, GPS error becomes already apparent and the route approximations become unreliable. The amount of cells needed increases exponentially when finer grids are produced. Therefore, we do not compute unnecessary levels in vain. Sparsest grid has cell length of 12.5 km. At lower levels (≥ 25 km) the cell size becomes so big that even the longest routes are represented by only a few cells.

**Table 1.** A mapping from zoom-level to the grid resolution. The statistics are for Mopsi2014 Route dataset using each of the grid resolutions.

| Zoom level | ≤ 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | ≥15 |
|---|---|---|---|---|---|---|---|---|---|---|
| Grid resolution | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Cell size (km) | 12,8 | 6,4 | 3,2 | 1,6 | 0,8 | 0,4 | 0,2 | 0,1 | 50 m | 25 m |
| Amount of cells | 7× $10^4$ | 9× $10^4$ | 1× $10^5$ | 2× $10^5$ | 4× $10^5$ | 7× $10^5$ | 1× $10^6$ | 3× $10^6$ | 5× $10^6$ | 1× $10^7$ |
| Memory (MB) | 3,5 | 4,5 | 6,5 | 9,5 | 16,5 | 30,6 | 59,6 | 118,6 | 238 | 486 |
| B-tree Index (MB) | 8,5 | 9,5 | 13,5 | 21,5 | 35,6 | 66,7 | 131,8 | 263,1 | 526 | 1,1 GB |

## 3 Route Search

We present next our algorithm for performing the gesture-based route search. The algorithm (*GSearch*) first extracts the cells that the input shape passes through using the *Find-Cells* function. This function chooses the correct grid resolution based on the zoom level using the mapping presented in Table 1. Every point is then mapped to the cell it resides in. At the Equator, one degree is roughly 111 km and the smallest cell length we support is 25 × 25 meters. We dilate the input route $C_A$ with 3 × 3 square structural element to obtain $C_A^d$.

**GSearch:** Searches for route candidates matching a given gesture.

**Input:** gesture shape G, zoom level z
**Output:** candidates list L
C, $C^d$ ← **Find-Cells** ( G, z )
ranking ← **RSR** ( C, $C^d$, z )
Top-Cluster ← **Cluster** ( ranking. similarities )
**for** i ← 1 to size ( ranking ) **do**
  **if** ranking [ i ] is in Top-Cluster **then**
    add ranking [ i ] to L. append ( ranking [ i ] )

**Find-Cells:** Obtains the cells that a given shape passes through at a specific zoom level.

```
Input: shape S, zoom level z
Output: cell set C, dilated region C^d
r               ← Get-Resolution ( z ) // according to Table 1
min-cell-size   ← 25 // meters
degree-size     ← 111 // km
dividing-factor ← min-cell-size * degree-size / pow( 2, r )
for i ← 1 to size ( S ) do
    x ← round ( S [ i ]. latitude * dividing-factor)
    y ← round ( S [ i ]. longitude * dividing-factor)
    add ( x, y ) to C
C^d             ← Dilate ( C )
```

*Route Similarity Ranking* (*RSR*) algorithm is then applied to find all similar routes in the database. *RSR* iterates through every cell in $C_A$ and $C_A^d$, and finds what other routes pass through the same cells. For each found route $C_B$, it checks if the cell belongs to $C_A \cap C_B$, $C_A \cap C_B^d$ or $C_A^d \cap C_B$. The algorithm maintains counters for each type and uses them for computing the similarity values using (2). Time complexity is $O((|C| + |C^d|)(\log(MQ)) + a(C) + a(C^d))$ where M is the number of routes in the database, Q is the average route size in cells and $a(C) = \sum \left(|C \cap C_i| + |C \cap C_i^d|\right)$, $i = \overline{1, M}$.

The similarity ranking usually results in a large number of routes, of which only few are relevant to the user. It might be possible to filter out routes below a given threshold, but then we might get no result in some cases; the other extreme is when searching for a very common route. Then there can be too many results above the threshold. Therefore, we limit the number of results using clustering as follows.

**RSR:** Computing the route similarity ranking.

```
Input: cells C, dilated part C^d, zoom level z
Output: ranking of routes according to similarity values
SC ← initialize SetCounter array; // structure defined below
// process input route
for i ← 1 to size ( C ) do
    R_i, R^d_i ← Get-Routes ( C [ i ] )
    for j ← 1 to size ( R_i ) do
        SC [ R_i [ j ] ]. A ++; SC [ R_i [ j ] ]. B ++; SC [ R_i [ j ] ]. AB ++;
    for j ← 1 to size ( R^d_i ) do
        SC [ R^d_i [ j ] ]. A ++; SC [ R^d_i [ j ] ]. B ++; SC [ R^d_i [ j ] ]. AB^d ++;
// process dilated part
for i ← 1 to size (C^d ) do
    R_i, R^d_i ← Get-Routes ( C^d [ i ] )
    for j ← 1 to size ( R_i ) do
        SC [ R_i [ j ] ]. B ++; SC [ R_i [ j ] ]. A^dB ++;
    for j ← 1 to size ( R^d_i ) do
        SC [ R^d_i [ j ] ]. B ++; SC [ R^d_i [ j ] ]. A^dB^d ++;
ranking ← new list;
for each r_id in SC do
    sim ← ( SC [ r_id ] . AB + SC [ r_id ] . A^dB + SC [ r_id ] . AB^d ) /
          ( SC [ r_id ] . A + SC [ r_id ] . B - SC [ r_id ] . AB)
    add ( r_id, sim ) to ranking
SetCounter {  A←0;  B←0;  AB←0;  A^dB←0;  AB^d←0;  }
```

We cluster the threshold values by *Random Swap* (*RS*) algorithm [11] with 10,000 iterations with 16 clusters. The algorithm alternates between K-Means and random relocation of centroids in order to avoid getting stuck in a local optimum. The algorithm converges to the final result in few hundreds of iterations, on average. However, since *Random Swap* is fast, we can afford to use 10,000 iterations to increase the probability of finding optimal partitioning.

From the clustering result, we take the cluster having the routes of highest similarities. The idea is that the clustering will find the size of this set automatically by fitting the clustering structure to the distribution of the similarities.

**Cluster:** Limits the ranking to the most likely candidates.

**Input:** similarities S
**Output:** cluster with highest similarities
T ← 10.000 // number of iterations
M ← 16 // number of clusters
P ← **RS** ( S, T, M ) // Random Swap clustering
Top-Partition ← 0; Top-Cluster ← empty set
**for** i ← 1 to size ( S ) **do**
  **if** S [ i ] = max ( S ) **then**
    Top-Partition ← P [ i ]
**for** i ← 1 to size ( S ) **do**
  **if** P [ i ] = Top-Partition **then**
    add S [ i ] to Top-Cluster

## 4 Experiments

We perform experiments using the Mopsi2014[5] dataset, which is a subset of all routes from the Mopsi database collected by the end of 2014. It contains 6,779 routes recorded by 51 users who have 10 or more routes. Routes consists of a wide range of activities including walking, cycling, hiking, jogging, orienteering, skiing, driving, traveling by bus, train or boat. Most routes are in Joensuu region, Finland, which creates a very dense area suitable for stressing the method. A summary of the dataset is shown in Table 2. All experiments were performed on Dell R920, 4 x E7-4860 (total 48 cores), 1 TB, 4 TB SAS HD.

**Table 2.** Statistics of Mopsi2014 route dataset.

| Routes | Points | Kilometers | Hours |
|---|---|---|---|
| 6,779 | 7,850,387 | 87,851 | 4,504 |

[5] http://cs.uef.fi/mopsi/routes/dataset.

### 4.1 Efficiency of the Search

The efficiency of the search is proportional to the size of the database, and to the resolution of the grid. The grid to be chosen depends on the zoom level required to view the targeted route on the map: small routes are best viewed using a higher zoom-level. The grid depends also on the size of the screen. To get a better understanding of this we computed the zoom-level for the best-view of each route in the Mopsi2014 dataset. We consider the best-view as the maximum zoom-level that shows the entire route on screen. The results in Fig. 3 show that lowest zoom levels are rarely used. Routes in such zoom levels should span across multiple countries or even continents, and thus, are rare in the dataset. The highest zoom levels (20–21) are also not often used because they cover only very short routes, usually non-movement records.

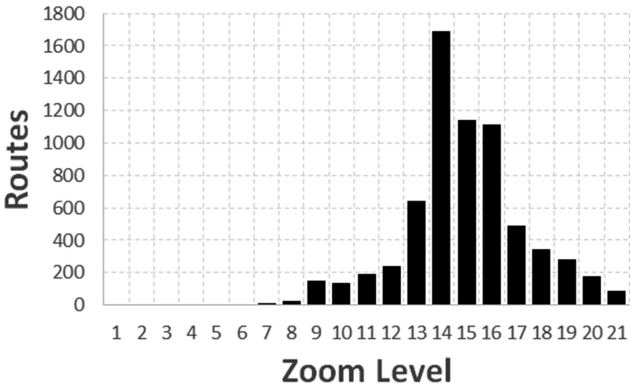


**Fig. 3.** Histogram showing what zoom-levels are used more often when viewing routes.

When computing the histogram from Fig. 3, we assumed a screen size of 1366 × 768, which, according to the free statistics provided by W3Counter[6], was the most used screen size during March 2016.

We next compute the efficiency of the G-Search algorithm by taking every route in Mopsi2014 as the target route. The best-viewed zoom level for them is first found. A perfect gesture is then simulated for the route by selecting the cells it travels through. Search is then performed using the default screen size of 1366 × 768. The results are summarized in Fig. 4. As expected, the time required is small (0.2–0.8 s) at small zoom levels. At the largest zoom levels the time is also small, but this is against expectations. The reason for the low execution times is the fact that for zoom level 15 and above, the same grid is used and, as a result, the number of cells required to represent each route is lower. Only the middle level routes can take slightly more than 1 s.

This experiment shows that, given a random target route, the expected search time is about 1 s or less, thus, it can be considered real-time. In practice, a smaller zoom level is used by the user than the best-fitting one is selected. Thus, < 1 s result happens

[6] https://www.w3counter.com/globalstats.php.

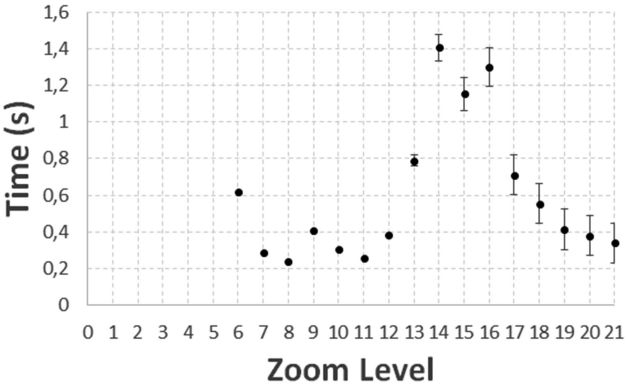


**Fig. 4.** Times required by G-Search when searching all routes in Mopsi2014 dataset. The results are grouped by the zoom-level and averaged. The average of all searches is 0.9 s.

more often. The reason is that often at zoom levels just below the best-fitting one it is easier to see the landmarks on the map. Furthermore, it is possible that at the best-fitting level the gesture implies drawing on the edges of the map, which are more difficult to target than the central area. Another reason is that the 1366 × 768 screen size is large, and using a smaller screen implies a finer grid will be used. Processing times with default screen size of 320 × 658 yields even smaller processing time of about 0.2 s.

The search time also depends on the density of the routes. In low density areas (< 200 routes), the search time is 0.14 s, on average. In very dense areas (> 1000 routes) the search time is 2.2 s, on average. There is also minor dependency on the size of the gesture. A gesture passing through 50 cells takes 0.7 s time on average, whereas as gesture passing through 200 cells takes 0.7 s, on average. The upper limit is the number of cells that can fit on the screen (3600 with the 1366 × 768 screen size).

### 4.2 Usability Evaluation

We study next the efficiency of the gesture search from usability point of view. We compare the average time user spends on searching a randomly chose route using the gesture search and using the previous (traditional) system. Eleven volunteers were asked to search randomly selected routes using a tool[7] built for this purpose as follows:

A target route was shown on map but no date, length or duration were shown. User can study and memorize the route as long as wanted.

When user pressed the *Start* button, user was (randomly) directed either to the traditional system or to the new Gesture search. Timer was started.

The task was to find the route and input its date and then press *Stop* button. If the date was correct the timer was stopped. If the user considered the task too difficult he was allowed to press *Give-up* button.

[7] http://cs.uef.fi/mopsi/routes/gestureSearch/qual.php.

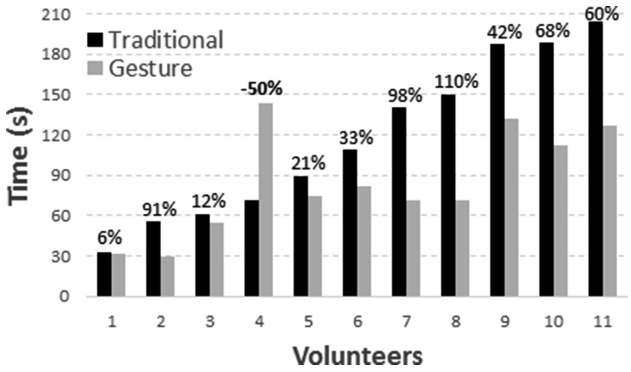


**Fig. 5.** Average search times and the relative difference between traditional and gesture search.

Each volunteer was asked to repeat the test at least 10 times, or as long as he/she found it fun to do.

In total, 106 routes were searched using the traditional system, and 98 using the gesture search. The searched routes were found 77 % of the time using traditional search compared to 91 % when using gestures. Gesture search was 41 % faster, on average. The individual performance differences are shown in Fig. 5. Traditional search is slower on average than gesture search for all except one user.

The search time is affected also by other factors such as complexity and length of the route, and density of the areas the route passes through. We next group the results by these three factors. The complexity is calculated as the number of points used by the polygonal approximation [2] to represent the route at its best-fit zoom level. Density is calculated as the proportion of cells that are overloaded by other routes; it is the opposite to the noteworthiness value in [5]. Results in Table 3 show that although shorter and less complex routes in low density areas are faster to find, the Gesture search outperforms the traditional approach in all cases.

The volunteers were also asked if they liked the Gesture search and which one they would prefer for such search task. They all rated Gesture search as good (10) or excellent (1). Most (9) preferred Gesture search, none (0) preferred the traditional search, and some (2) would not use either. Written comments included "*I really liked it*" and "*It was fun*".

**Table 3.** Average search times when grouped by different factors.

| | Length | | Complexity | | Density | |
|---|---|---|---|---|---|---|
| | Short 2.7 km | Long 12.7 km | Low 31 pts | High 128 pts | Low 12 % | High 75 % |
| Traditional | 90 s | 116 s | 87 s | 120 s | 90 s | 117 s |
| Gesture | 64 s | 78 s | 65 s | 77 s | 54 s | 88 s |
| Reduction | 30 % | 33 % | 25 % | 36 % | 30 % | 24 % |

## 5 Conclusion

We showed that gestures can be successfully used as input for searching routes from large data collections. We solved all the components of the search including user input, database optimization, pattern matching, and selecting threshold by clustering to show only the most significant results. The effectiveness of the method was demonstrated by run time analysis showing that it works real time, and by usability experiments showing that it outperforms traditional search.

## References

1. Waga, K., Tabarcea, A., Mariescu-Istodor, R., Fränti, P.: Real time access to multiple GPS tracks. In: International Conference on Web Information Systems and Technologies (WEBIST 2013), Aachen, Germany, pp. 293–299 (2013)
2. Chen, M., Xu, M., Fränti, P.: A fast multiresolution polygonal approximation algorithm for GPS trajectory simplification. IEEE Trans. Image Process. **21**(5), 2770–2785 (2012)
3. Waga, K., Tabarcea, A., Chen, M., and Fränti, P.: Detecting movement type by route segmentation and classification. In: IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2012), Pittsburgh, USA, pp. 508–513 (2012)
4. Mariescu-Istodor, R., Tabarcea, A., Saeidi, R., Fränti, P.: Low complexity spatial similarity measure of GPS trajectories. In: International Conference on Web Information Systems and Technologies (WEBIST 2014), Barcelona, Spain, pp. 62–69 (2014)
5. Mariescu-Istodor, R., Fränti, P.: Grid-based method for GPS route analysis and retrieval. Manuscript (2016, submitted)
6. Cirelli, M., Nakamura, R.: A survey on multi-touch gesture recognition and multi-touch frameworks. In: ACM Conference on Interactive Tabletops and Surfaces (ITS 2014), Dresden, Germany, pp. 35–44 (2014)
7. Karam, M., Schraefel, M.C.: A taxonomy of Gestures in Human Computer Interaction. ACM Transactions on Computer-Human Interactions (2015, in press)
8. Kristensson, P.O., Zhai, S.: Command strokes with and without preview: using pen gestures on keyboard for command selection. In: SIGCHI Conference on Human Factors in Computing Systems (CHI 2007), New York, USA, pp. 1137–1146 (2007)
9. Li, Y.: Gesture search: a tool for fast mobile data access. In: ACM Symposium on User Interface Software and Technology (UIST 2010), New York, USA, pp. 87–96 (2010)
10. Kennedy, M., Kopp, S.: Understanding Map Projections. ESRI Press, Redlands (2001)
11. Fränti, P., Kivijärvi, J.: Randomized local search algorithm for the clustering problem. Pattern Anal. Appl. **3**(4), 358–369 (2000)
12. Cormen, T.H.: Introduction to Algorithms. MIT press, Cambridge (2009)