EasyTracker: An Android application for capturing mobility behavior

Alexandros Doulamis
Dept. of Informatics
University of Piraeus
Piraeus, Greece
alex.doulamis@gmail.com

Nikos Pelekis
Dept. of Statistics & Insurance Sci.
University of Piraeus
Piraeus, Greece
npelekis@unipi.gr

Yannis Theodoridis
Dept. of Informatics
University of Piraeus
Piraeus, Greece
ytheod@unipi.gr

***Abstract* — This paper presents EasyTracker, a mobile application developed for the Android O/S that enable the storage, analysis and map visualization of routes of mobile users. Furthermore, it enable users to manually annotate part of their routes with labels describing their activity and behavior (e.g. "home having breakfast", "travelling by car to work", etc.). Of equal importance, the application encapsulates several state-of-the-art line simplification algorithms for compressing the trajectories drawn from collected GPS records, as well as segmenting trajectories into homogeneous parts in order to facilitate automatic auditing of the user's manual annotation.**

Keywords - EasyTracker; GPS data; trajectory; path; track; privacy; compression; segmentation; Android OS

I. INTRODUCTION

Nowadays, the number of users possessing and using smartphones equipped by a GPS receiver increases rapidly, while their interest for mobile applications that enable the storage, analysis and visualization of the collected space-time information is apparent.

There are already plenty of such general-purpose applications, including "My Tracks" [1], AndAndo" [2], "GPS Tracker" [3] and "EveryTrail" [4]. Apart from these, there are a great number of applications developed for specific groups of users. A few examples include "Runkeeper" [5], "Endomondo Sports Tracker" [6] and "Sports Tracker" [7] that support functionality mainly focusing on sports/fitness. All of these have as main function to store and display a route on a map (usually, Google Maps), also extracting simple statistics, such as average speed, distance travelled, etc.

In this paper, we present "EasyTracker" for Android O/S. The core of EasyTracker is threefold. At first, EasyTracker collects space-time points of user's movement using a GPS receiver, thus creating and visualizing a path (or trajectory) followed by the user. In contrast to similar applications, the collected path, to be stored in a server's database, is sanitized by filtering out locations that lie inside user-defined areas of interest – sensitive areas where the user does not allow to be tracked (e.g. around home, a hospital, etc.). This way, EasyTracker provides end users with personalized privacy functionality [8].

The second key functionality is that EasyTracker allows a user to annotate parts of her track with labels and therefore to describe her current activity (e.g. "stopped at café A", "driving towards office") [9]. Such feature can turn out to be very useful for automatic fill-in of surveys performed in transportation science or for researchers working on activity recognition who are usually restricted to use manually processed surveys.

Third, EasyTracker encapsulates state-of-the-art algorithms that process in an online fashion the received stream of GPS recordings and transforms it into meaningful tracks, ready to be stored into a trajectory database [10] for further analysis. Specifically, EasyTracker includes line simplification methods that compress the incoming stream of timestamped locations (thus reducing the storage cost), which are then partitioned into homogeneous portions according to some spatio-temporal criteria using a state-of-the-art segmentation method [11]. According to this segmentation, the track is split into portions (i.e. sub-tracks), which can be labeled by tags that describe the corresponding spatio-temporal behavior of the user (e.g. STOPPED, when the speed is very low). This is important as it facilitates the user (or some auditing algorithm) to compare her manual annotations with the classified sub-tracks as provided by the segmentation algorithm. The big picture that illustrates these novel features of EasyTracker is depicted in Fig. 1.

The rest of the paper is organized as follows. Section II presents related work. In section III, the various features of EasyTracker are explained in more detail. Subsequently, we provide a general overview of EasyTracker's architecture (section IV) and describe the challenges to develop for different devices (section V). Section VI concludes and proposes improvements for future work.

II. RELATED WORK

The number of existing applications that make use of the information captured by a GPS receiver is significant. To obtain a better overview, we present the most successful applications, by classifying them in different categories.

In the general-purpose category, which targets to people who simply like to track and visualize their routes, Google's "MyTracks" [1] is the most famous application for Android O/S. We have to note that this application is a relatively simple tracker with no advanced features, however it has more than 5 million downloads. Other similar applications in the same category include "AndAndo" [2], "EveryTrail" [4], which has more advanced features by allowing users to add pictures or points of interest and create guides, and "OruxMaps" [12], which allows utilizing various kinds of maps.

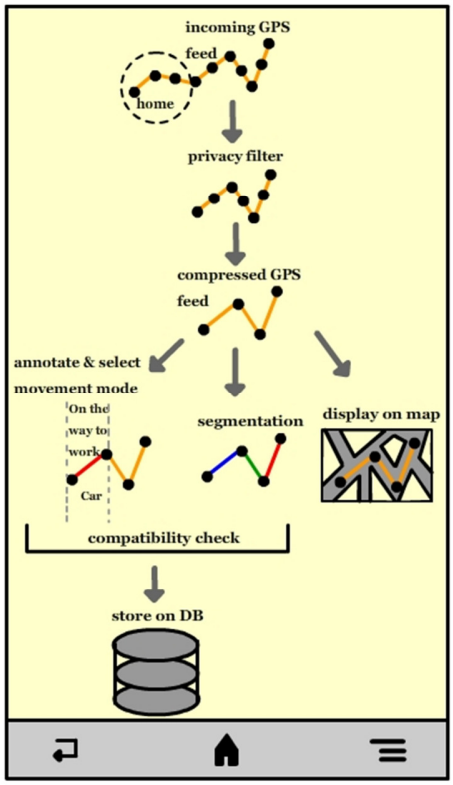

Figure 1: The big picture of EasyTracker

Another category is about applications focusing on sports/fitness. Here we find many specialized features like the computation of calories consumption or the measurement of heart rate utilizing internal or external sensors; "RunKeeper" [5], "Endomondo Sports Tracker" [6] and "runtastic" [13], to name a few.

Third, there exist applications designed to track lost devices or devices that belong to another person (assuming permission is granted). For instance, tracking a smartphone is feasible nowadays with the assistance of GPS and wireless networks. Applications of this type are, for example, "Wheres My Droid" [14], "GPS Tracking Pro" [15] and "GPS Tracker" [3].

We classify EasyTracker in the first category, since it does not focus to a specific target group or application. We also argue that the supported functionality is by far richer than the functionality provided by current commercial apps.

III. EASYTRACKER IN ACTION

A. *Basic functionality enhanced with privacy filters*

Common to all similar applications, the main functionality of EasyTracker is to record a track and visualize it in real time on a user-selected map. Thereby it stores the geographical coordinates on a local database and it is able to calculate useful statistics (total distance travelled, average speed, etc.). Using the device orientation sensor, the real heading can be calculated and displayed on a map and on a built-in compass. The user has also the options to take photos or set places (point of interests - POI) and associate them to geographical coordinates, create notations to each part of a track, view a stored track on a map or export a saved track in a geographical file following one of many well-known data formats like GPX, KML and CSV.

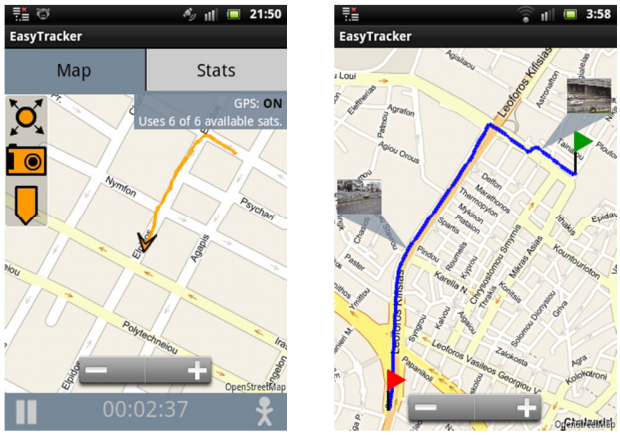

Figure 2: Screenshots of EasyTracker. Left: recording a new track, Right: displaying a recorded track

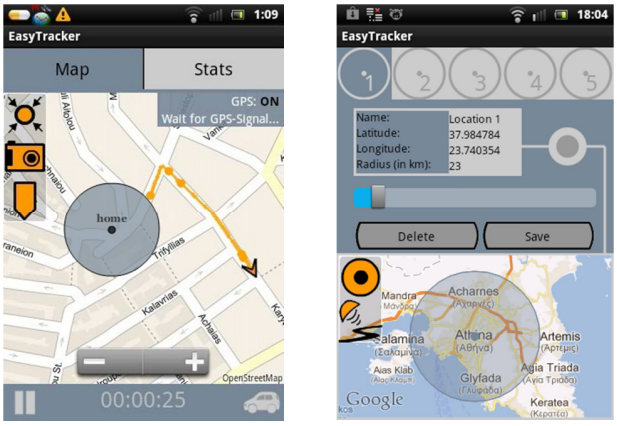

Figure 3: Privacy filter of EasyTracker.
Left: Showing privacy area while recording a track;
Right: Setting privacy regions and respective levels of privacy

An important feature of EasyTracker is its ability to choose among various maps (Google Maps, Open Street Map and Microsoft Bing Maps), also the possibility to use offline maps. The latter is very important when the user is currently in an area that has no network or a bad connection. Fig. 2 illustrates two screenshots of the application.

Departing from the features that are more or less available in many other similar applications, to protect users' privacy the application allows registering user-defined areas inside where recording is not permitted. Such an area is defined by a point (either set manually by providing exact coordinates or by touching on the map) along with a desired radius around this point and a characteristic name (e.g. home, work, mam's house). The user can specify up to five privacy locations using a radius which is limited from 15 meter till 300 kilometers. Fig. 3 depicts screenshots of this privacy filter.

B. Compressing incoming GPS feed

One of the differences between conventional GPS-Trackers and EasyTracker is the exploitation of several state-of-art algorithms which are responsible to decide which GPS recordings to be saved in the stored trajectory so as to minimize the mobile device's storage cost. EasyTracker incorporates four spatiotemporal compression techniques for this purpose: algorithms to save a track to a local data base: *Minimal Time / Minimal Distance Method*, which is a component part of the Android SDK [16], *Before Opening Window* (BOPW) [17], *Normal Opening Window* (NOPW) [17] and *Threshold Algorithm* [18]. (The detailed presentation of the above methods is omitted due to space limitations.)

To demonstrate the results of compression achieved by the above methods, we use the path illustrated in Fig. 4.

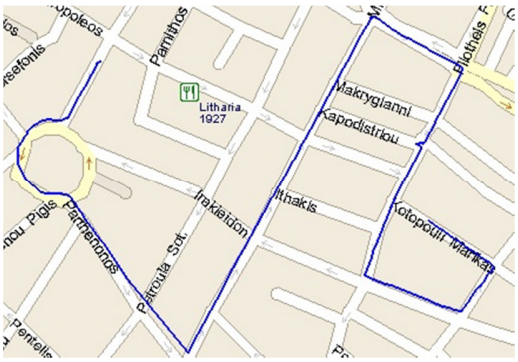

Figure 4: A test path for the comparison of compression techniques.

The *Minimal Time / Minimal Distance Method* [16] is the simplest method to save coordinates. The user sets two parameters: a value for the "minimal time" and a value for the "minimal distance". The first value determines when to check the second value. If the second value is exceeded, then the incoming coordinate will be saved. For example, if minimal time is set to 5 sec and minimal distance is set to 12 meters, the application checks every 5 sec whether the user is 12 meters away from the previously stored point. If yes, the point will be saved. Otherwise, it is omitted.

This method is very sensitive to the threshold values chosen by the user. To demonstrate this, we recorded the test path twice by varying the threshold values. For the first track it is 3 sec and 10 meters while for the second it is 10 sec and 80 meters. Fig. 5 illustrates the results. It is obvious that the larger the values the less the coordinates stored, which affects the quality of the stored track. On the other hand, if we choose small values to avoid this problem we store many (perhaps redundant) points. The 'optimal' values of course depend on the mode of movement (walking, running, driving, etc.) and may be difficult to be known in advance by the user.

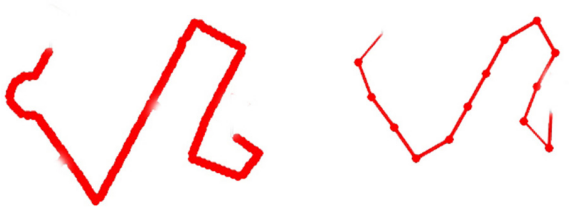

Figure 5: Compressed tracks with Minimal time/distance method. Left: Minimal Time = 3 sec / Minimal Distance = 10 m; Right: Minimal Time = 10 sec / Minimal Distance = 80 m

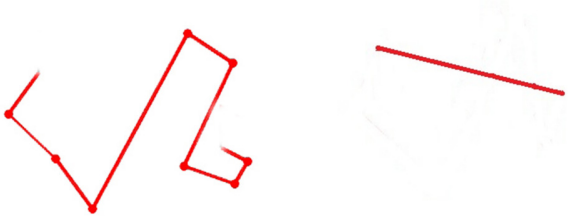

Figure 6: Recorded tracks with the BOPW-algorithm. Left: ED = 8 m; Right: ED = 15 m

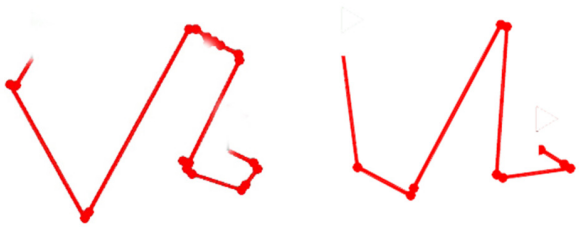

Figure 7: Recorded tracks with the Threshold-method. Left: threshold = 8 m; Right: threshold = 15 m

The two methods proposed in [17], *Before Opening Window (BOPW) and Normal Opening Window (NOPW)*, achieve a very good compression rate. The user gives as parameter a distance threshold (*Euclidean Distance* - ED) which is a bound on how much the original path can deviate from the compressed one. However, a high ED value may lead to loss of data which are crucial to represent well the

sketch of the movement, so the tradeoff between compression and quality is still under question. In Fig. 6 we record the test path using the BOPW algorithm with different ED values.

In the first path (ED = 8 meters) the tracker stored only a few geographical coordinates without losing any remarkable information. On the contrary, in the right path (ED = 15 meters) the path is actually destroyed since only two points (start and end point) are stored.

Different from the previous ones, the *Threshold Method* [18] relies on the speed of the user; to achieve good results the speed has to be constant. In Fig. 7 we recorded the sample track twice (in the first the threshold was set to 8 meters while in the second it was 15 meters) and in both cases the speed was constant at 30 km/h. As with the previous techniques, we notice that also this cannot avoid a loss in representativeness when the threshold is set to a high value.

C. *Manual annotation vs. automatic segmentation*

EasyTracker allows the user to annotate parts of the track with labels in order to describe her current activity (e.g. "stopped at Acropolis museum", "walking towards Parthenon") [9]. This way we allow the automatic annotation of user's activities, including her stops and movements. In detail, the user has the option to select an annotation tag/label which is attached to a specific part of the track, or to type a smart text of her choice (Fig 8).

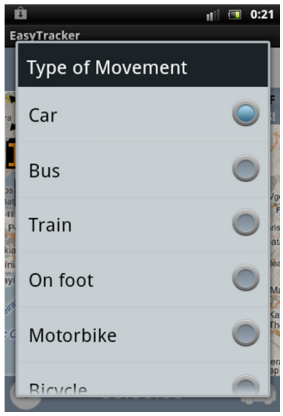

Figure 8: Selecting movement mode.

An important feature of EasyTracker is the usage of the *Trajectory Segmentation Algorithm* (TSA), proposed in [11]. TSA partitions tracks into homogeneous portions according to their speed pattern. This segmentation splits the track into portions (i.e. sub-tracks) where each of them can be labeled by a tag that describes the corresponding speed range (e.g. WALKING, assuming the speed pattern is in a predefined range). This is important as it facilitates the user to compare her manual annotations with the classified sub-tracks as given by the segmentation algorithm.

IV. THE ARCHITECTURE OF EASYTRACKER

As already mentioned, the main features of EasyTracker include the recording and storage of a track and the visualization of a stored track on a map.

A. *Recording a track*

Only a few steps are required to record a track. As it is illustrated in Fig. 9, the application at first verifies the GPS signal. If it is available, the tracker starts storing the coordinates. Whether a location is stored is the decision of the compression method enabled by the user. Each time a set of coordinates are stored into the database, the application uses the TSA algorithm to segment, if necessary, the track and calculates various useful statistics (total distance of the track, average speed, etc.).

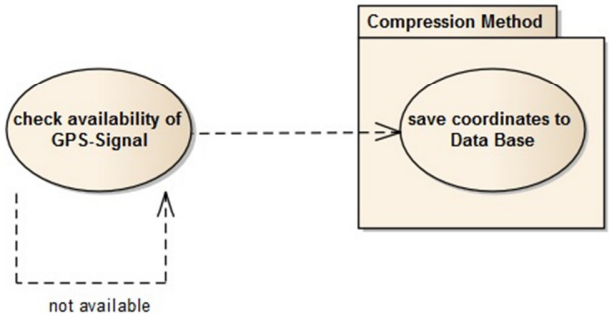

Figure 9: Recording and storing coordinates

B. *Storing track data*

We decided to store information in a local database in order to avoid requiring Internet connection. For this purpose, we make use of the Sqlite data base which is embedded on the Android SDK [19]. The database is a simple schema that contains the main information of a track, the GPS coordinates it derives from, as well as the statistics described above. Accompanying places (POIs) and pictures are also stored in the database.

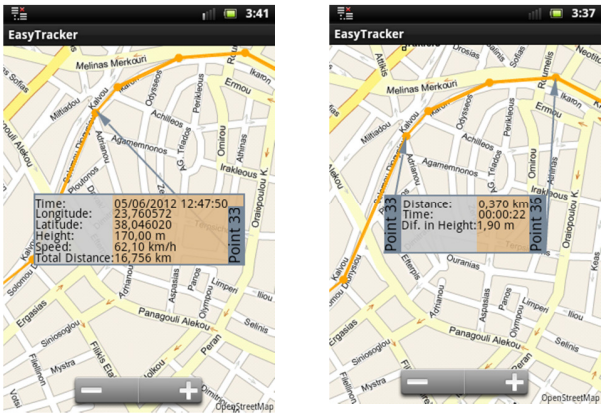

Figure 10: Visualizing information of a point (left), calculating distance of two points (right)

C. *Visualizing and exporting a track*

EasyTracker has the function to display a recorded track on a map, so that the user can analyze her stored track. In particular, the user has the option to study each stored point closer, calculate the distance between two different geographical points, get more information about a stored place or view a picture taken at this point (Fig. 10).

It is also feasible to export a stored track to a file format appropriate for further processing (e.g. the general purpose CSV, KML for visualizing the track in Google Earth, GPX for GPS use).

V. A DEVELOPER'S DISCUSSION

One of the major advantages of Android O/S is the compatibility to many different devices and screen resolutions. But it is exactly this that increases the degree of complexity for the software developer. On the following two sub-sections we discuss this challenge.

A. *Devices with different displays*

As already mentioned, there are several devices that support Android. Except the various kind of sensors which differs from device to device and the different hardware performances, one of the biggest variances has to do with the display screen, which is not only the size, but also the screen density. Therefore, Android uses dpi (dots per inch) instead of pixels [20]. Nevertheless, the developer should make extra effort, especialy if graphics are used (Fig. 11).

Figure 11: EasyTracker's main menu in different displays. Left: 800x480 (high-dpi); Right: 240x380 (low-dpi)

B. *Orientation-depentent devices*

To make the work as user-friendly as possible, Android has decided to support different device orientations. This means that the application adapts to the current orientation of the device, so the user can choose the desired orientation depending on the circumstances. Frequently, this change takes effect automatically by the O/S, but sometimes the developer has to do it manually. To solve this, several layouts for each orientation (mostly horizontal or vertical) should be developed, see e.g. Fig. 12.

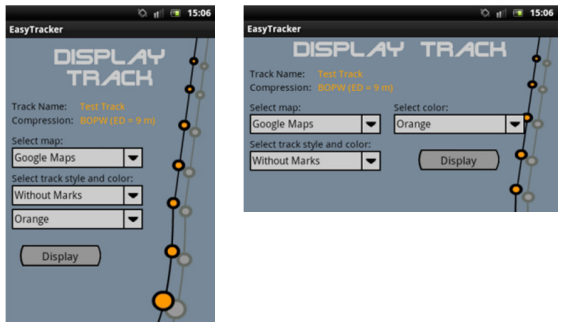

Figure 12: Display Track pre-menu in different orientations. Left: vertical (portrait), right: horizontal (landscape)

VI. CONCLUSION AND FUTURE WORK

In this paper, we presented EasyTracker, a mobile application developed for the Android O/S that enables the storage, analysis and map visualization of routes of mobile users. In comparison with related applications, EasyTracker provides novel functionality at three levels: (i) it enables users to manually annotate part of their routes with labels describing their activity and behavior, (ii) it encapsulates several state-of-the-art trajectory compression algorithms for a tradeoff between storage cost and quality of movement's representation, and (iii) it automatically segments tracks according to a state-of-the-art trajectory segmentation algorithm in order to facilitate automatic auditing of the user's manual annotation. As a fourth (preliminary) contribution, it enables users to protect their privacy by defining sensitive areas where recording is not permitted.

This type of applications can be used in several fields, from route planning and resources administration (e.g. carpooling) to entertainment and social networking (e.g. next generation location-based social networks).

Challenges for future work include the extension of TSA algorithm in order to recognize for each track segment the activities of the user in such a way as to enable the automatic validation and auditing of the user's annotation, an energy saving policy since our application is used in devices with limited energy resources, a possibility to detect user's position in non GPS-available areas (e.g. in indoors environments or in subsurface regions) without loosing in accuracy and a hybrid local – in the cloud storage schema that would make it ready for social networking applications.

ACKNOWLEDGMENT

This work was partially supported by the European FP7 Open-FET project DATASIM "Data Science for Simulating the Era of Electric Vehicles" (URL: http://www.datasim-fp7.eu/).

REFERENCES

[1] Google MyTrack. URL: http://www.google.com/mobile/mytracks/ (accessed: 1 Jun. 2012)

[2] Javi Pacheco, AndAndo. URL: https://play.google.com/store/apps/details?id=com.javielinux.andando (accessed: 1 Jun. 2012)

[3] InstaMapper LLC, GPS Tracker. URL: http://www.instamapper.com/ (accessed: 1 Jun. 2012)

[4] GlobalMotion Media, Inc, EveryTrail. URL: http://www.everytrail.com/ (accessed: 1 Jun. 2012)

[5] FitnessKeeper, Inc, RunKeeper. URL: http://runkeeper.com/ (accessed: 1 Jun. 2012)

[6] Endomondo, Sports Tracker. URL: http://www.endomondo.com (accessed: 1 Jun. 2012)

[7] Sports Tracking Technology Ltd, Sports Tracker. URL: http://www.sports-tracker.com/ (accessed: 1 Jun. 2012)

[8] N. Pelekis, A. Gkoulalas-Divanis, M. Vodas, D. Kopanaki, and Y. Theodoridis. Privacy-Aware Querying over Sensitive Trajectory Data, *Proceedings of the ACM Conference on Information and Knowledge Management* (CIKM), 2011.

[9] C. Parent, S. Spaccapietra, C. Renso, G. Andrienko, N. Andrienko, V. Bogorny, M. L. Damiani, A. Gkoulalas-Divanis, J. Macedo, N. Pelekis, Y. Theodoridis, and Z. Yan, "Semantic Trajectories Modeling and Analysis" *ACM Computing Surveys*, to appear.

[10] N. Pelekis, E. Frentzos, N. Giatrakos, and Y. Theodoridis. HERMES: Aggregative LBS via a Trajectory DB Engine, *Proceedings of the ACM SIGMOD Conference*, 2008.

[11] C. Panagiotakis, N. Pelekis, I. Kopanakis, E. Ramasso, and Y. Theodoridis. Segmentation and Sampling of Moving Object Trajectories based on Representativeness, *IEEE Transactions on Knowledge and Data Engineering*, 2011.

[12] Jose Vasquez, OruxMaps. URL: http://www.oruxmaps.com (accessed: 1 Jun. 2012)

[13] Runtastic. URL: http://www.runtastic.com/ (accessed: 1 Jun. 2012)

[14] Alienman Tech, Wheres My Droid. URL: http://wheresmydroid.com/ (accessed: 1 Jun. 2012)

[15] GPS Tracking Pro. URL: https://play.google.com/store/apps/details?id=com.fsp.android.c (accessed: 1 Jun. 2012)

[16] Android developers, Location Manager. URL: http://developer.android.com/reference/android/location/LocationManager.html (accessed: 1 Jun. 2012)

[17] N. Meratnia and R. A. de By. Spatiotemporal Compression Techniques for Moving Point Objects, *Proceedings of the Extending Database Technology Conference* (EDBT), 2004.

[18] M. Potamias, K. Patroumpas, and T. Sellis. Sampling Trajectory Streams with Spatiotemporal Criteria, *Proceedings of the Scientific and Statistical Database Management Conference* (SSDBM), 2006.

[19] Android developers, SQLite Database. URL: http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html (accessed: 1 Jun. 2012)

[20] Android developers, Supporting Multiple Screens. URL: http://developer.android.com/guide/practices/screens_support.html (accessed: 1 Jun. 2012)