2013 International Conference on Indoor Positioning and Indoor Navigation, 28-31th October 2013

Workshop: Location-based services for Indoor Smart Environments

Simple Indoor Routing on SVG Maps

Julian Ohrt
Hamburg University of Technology
Institute of Telematics
Hamburg, Germany
Email: julian.ohrt@tuhh.de

Volker Turau
Hamburg University of Technology
Institute of Telematics
Hamburg, Germany
Email: turau@tuhh.de

***Abstract*—Until now no general applicable technology for indoor localization has prevailed. Similarly, there are many approaches to model buildings for indoor navigation being developed. Most of them are very expensive to apply. This work presents a tool for graphically adding navigational data to existing maps. It further demonstrates how to use the created map for routing as pure web application which can be embedded in smartphone applications and used with any positioning technology. As SVG is used as format for the map, it analyses performance of rendering and manipulating SVGs on different devices. We show that even large maps can be displayed on current mobile Android devices sufficiently fast for usage as indoor navigation application.**

I. Introduction

Even though a research area for many years and many promising appearing results (e.g., [1]–[6]), there is still no general applicable technology available for reliably determining the position of handheld devices within buildings sufficiently detailed for navigation. Further, there is no general applicable technology for displaying indoor maps including current position or routing paths available for researchers or developers. Existing systems for outdoor maps can be used; however, they have some shortcomings: Most are not well suited for fine granular structures such as rooms within building. Further, data is usually stored on third party servers, making it difficult to enforce security restrictions and making updates troublesome or slow. Commercial products are available; however, their technical details are disclosed only vaguely.

This work assumes that in the foreseeable future a technology will be generally available that allows at least room-based localization of smartphones within buildings. Further, we assume that for large buildings, e.g., shopping centers or hospitals, floor plans are already available as vector graphics which can easily be converted into SVG files. That given, this work presents a method for enhancing existing floor plans to indoor maps which allow showing the user's position as well as navigation instructions. For creating such navigable maps a software application is presented which allows graphically adding routing information on top of an existing map. This way it is possible to create floor maps which - whenever a suitable localization technology becomes available - can be used for indoor navigation. The resulting software project SvgNaviMap is available under GNU General Public License.

II. Related work

Available systems for displaying mainly outdoor maps, e.g., Google Maps (GM) or Open Street Maps (OSM), provide worldwide map data and allow routing on streets. Since 2012 GM also starts to offer multi-level building plans without navigation functionality [7]. OSM, specifically IndoorOSM, also offers indoor maps which can be even used for routing [8], [9]. Even though there are tools available for creating indoor models, it is still a very time consuming task. Further, routing on indoor models is only available as part of some research projects for handpicked-buildings [10], [11].

What is more, using GM or OSM all map data is centralized and not under control of the owner of the building. Updates may be troublesome. Security issues may arise being not able to apply access restrictions to maps or parts of maps.

MapBiquitous [12] is a research project which aims to combine outdoor and indoor navigation. It is based on building models described in the Geographic Markup Language. Creating such models is similar costly as creating OSM models. However, no editor is provided.

To the best of our knowledge, there is no technology freely available which allows researchers to easily create and display simple, navigable, indoor maps which are stored in a decentralized way.

Alternatively, there are several commercial, closed-source systems available which provide indoor localization and/or routing [13]–[18]. However, their working principals are usually not disclosed and are thus not further considered.

III. System design

The presented system - SvgNaviMap - is intended to provide indoor navigation on smartphones. It bases on existing floor plans without modifying them. Thus, colors and outline of maps are the same on wall-mounted maps and the navigation application. This facilitates orientation for the user as recommended by [19], reduces workload for creating maps, and enables a uniform corporate identity through maps on different media. Further, emphasis is put on easy creation of navigation models. To the best of our the knowledge SvgNaviMap is the first open-source project which allows navigation on existing 2D maps across several floors.

The basic work flow for making existing maps navigable is shown in Figure 1. First, a floor plan of the building in question in the SVG format is required (a). This map is then loaded with the SvgNaviMap editor. Visually routing information is drawn on top on the map (b). This includes defining points-of-interest (POIs) and grouping them into categories. Further, the map has to be pinned to a global coordinate system (c). After the configuration phase is completed, the map including

978-1-4799-4043-1/13/$31.00

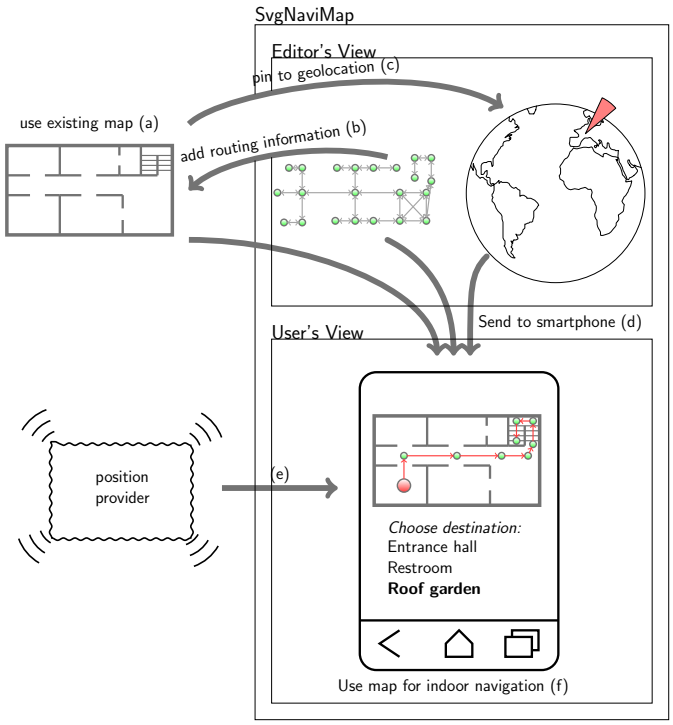


Fig. 1. Schematic overview of how to use SvgNaviMap

navigation information is supplied to a smartphone, e.g., via a web server (d). Assuming that a position provider is installed (e); the user can choose a destination and start navigation.

In the following we present our design choices for SvgNaviMap. Afterwards we explain what kind of routing information is required.

A. Design choices and implications

Assuming that each large building in which a navigation system would be useful also has a server, we decided to use a distributed approach for SvgNaviMap. This way, map data is under control of the owner or IT administrator of the building in question. However, it is also possible to use an external Internet server. Even though the system implements a server-client-architecture, the web server is merely necessary for providing data. No dynamic procedures are performed on the server reducing load on the server and assuring good scalability.

As there are many mobile clients in use, and web technologies are being unified in the HTML5 standard, we chose to rely on web technologies. In specific, we designed a web application that strongly relies on SVG and JavaScript. No browser plugins are required. Route calculation is done on the client-side using JavaScript. Browsers are powerful SVG viewers which supports most features[1]. Zooming and panning is enabled by default. Further, the SVG DOM tree can be accessed and manipulated using JavaScript. For Android devices, SVG in the default browser is supported since version 3 [20].

[1]Overview of officially supported SVG elements in Firefox and WebKit: https://developer.mozilla.org/en/docs/SVG_in_Firefox
http://www.webkit.org/projects/svg/status.xml

Further, it should be possible to use existing maps without modifying them. However, bitmap graphics need to be vectorized first in order to profit of the interactive features of SVG as well as lossless scaling. SvgNaviMap supports any number of levels and needs thus to support one separated SVG file per floor.

In order to be able to use any positioning technology which may become available in the future, global coordinates in the format of the World Geodetic System are being supported. Since SVG uses its own coordinate system, a mapping between the internal Cartesian SVG coordinates and global latitude and longitude is required. SvgNaviMap even supports maps which are not drawn to scale.

Navigational data is stored independently of the map data. We assume that a visitor's routing destination may be located on any floor. It is thus not practical to split routing information per level. It is not required, either, since routing information even for a large building is relatively small. Thus, before starting navigation, the client needs to download the complete routing information.

The actual web application is split into two independent parts. First is the Editor's View. It allows the developer to load an SVG map, then to add and configure navigational data during setup phase. The other part is the User's View which simply displays the map and shows the current position and routing directions to the end user.

B. Routing Information

Being able to show the route, SvgNaviMap needs to have navigational information as well as the map itself. The configuration XML file combines both parts. It is thus the starting point when loading a new SvgNaviMap project. It contains links to all required SVG files. Links may either be absolute URLs or local, relative paths. Further, it contains all information required for calculating and displaying routes which is detailed in the following.

Routing is performed using a directed graph which is drawn as overlay on top the SVG map. When the user enters a destination, his location is mapped to a close vertex. From this start position routing information is displayed as arrows along the edges toward the destination vertex.

Vertices are either helper points, solely used to calculate and show routes or they are Points of Interest (POIs). Latter are detailed by a description and can be chosen as routing destinations. They can also be grouped into self-defined categories, which allow e.g., to offer a list of all restaurants in a shopping mall. Group names are also stored within the configuration file.

The weight of each edge is by default equal to its length. Optionally, a weight factor can be manually added allowing to increase or decrease the total weight of the edge. This way e.g., stairs can be rated more expensive than an elevator; an auto walk cheaper than a normal corridor. Further, flags can be set indicating in which direction the edge is routable and whether it is wheelchair accessible. All routing information is stored in the XML format. This way more edge-specific information can easily be added.

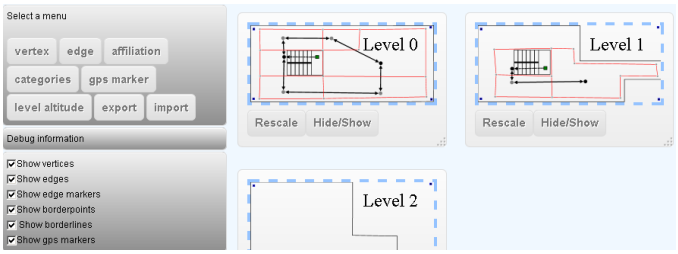


Fig. 2. Overview of Editor's View of SvgNaviMap

For multistory buildings, navigation maps should provide multiple floors. SvgNaviMap is able to span the routing tree over multiple floors. Each floor is assigned a lower and upper height (floor and ceiling) allowing to map a given altitude to a floor.

Finally, for mapping between the global, geographic coordinates system and the local SVG coordinates system, two concepts are integrated in SvgNaviMap and its configuration file. For one, we included GPS Markers which lock an SVG position to a real-world coordinate. As floor maps are often not true to scale any number of markers are supported. For correct rotating and scaling of the coordinates systems during mapping, at least three GPS markers are required. So called affiliation areas, on the other hand, allow mapping from any SVG coordinate to one routing vertex. Thus, each affiliation area is a self-defined area around its corresponding vertex. This concept is similar to cell in a Voronoi graph, however, note that due to obstacles like walls affiliation areas cannot be computed automatically but have to be defined manually by the editor. Care must be taken in order to not overlap affiliation areas to prevent ambiguous mappings. Further, complete mapping is only possible if they cover the whole map.

IV. Usage

A. Editor's View

Before using either view, SVG maps have to be integrated into SvgNaviMap via a configuration file. Next, routing information has to be created for the SVG using the Editor's View (Figure 2). Using a menu, all routing information elements described above can be created. If available, routing information can also be imported.

First, vertices need to be added. In general, in the center of each room a POI vertex should be placed. Additionally, a room label may be added as description. Helper vertices close to each door will assure that direction arrows will not pass through walls. Next, edges are added to connect vertices. For edges connecting different levels, stepmarkers are inserted automatically. Further, affiliation areas need to be assigned. All vertices need to be inside their corresponding areas, as shown in Figure 3.

GPS Markers are first added to the SVG map. Afterwards, their corresponding GPS position is set either by explicitly entering latitude and longitude or by choosing a position on a map as shown in Figure 4. Finally, lower and upper heights of all levels need to be set.

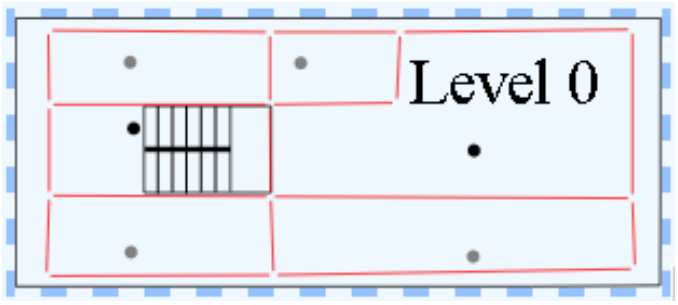


Fig. 3. SvgNaviMap floor map showing routing vertices and affiliation areas

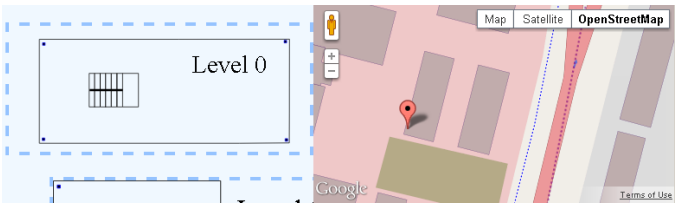


Fig. 4. SvgNaviMap floor map showing GPSmarkers on the left and an according anchor on an OSM map on the right.

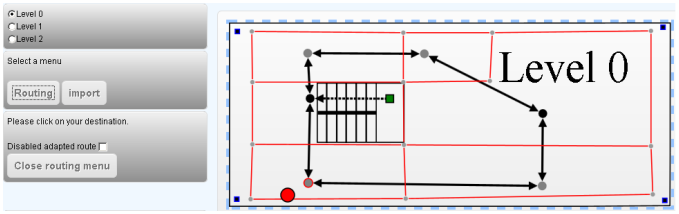


Fig. 5. User's View: Mapping from real position (red circle) to according vertex. On the left, an example for a control menu using SvgNaviMap JavaScript interface.

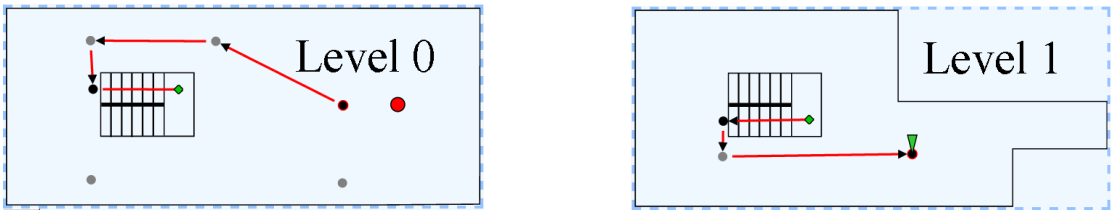
B. User's View

This view for the user is mainly a map of the current building which allows to display the own position, which is mapped to the appropriate vertex according to the defined affiliation areas (Figure 5).

After a destination is selected, the shortest route is calculated and displayed. Internally, an inverted Dijkstra algorithm is used which calculates routes from any source node to the destination. This way, the shown navigation directions can be updated whenever the own position changes without recalculating the route. In case a route contains floor changes, SvgNaviMap shows stepmakers. They indicate where floors are entered and exited as shown in Figure 6.

Generally, the User's View is included into a smartphone application inside a WebView element which by default allows panning and zooming of the map. By calling JavaScript functions inside the WebView the main application is able to control SvgNaviMap.

The JavaScript interface offered by SvgNaviMap allows to set the current position of the user which is displayed. It may be either provided as SVG coordinate or as geographic latitude and longitude. For a cell-based localization, it is also possible to set the user position to a POI vertex. Further, the interface offers methods to change the shown level, retrieve



(a) (b)

Fig. 6. Diamond-shaped stepmarker indicate floor change

TABLE I. DETAILS ABOUT DEMONSTRATION MAPS

	Small project	Large project
Number of levels	3	5
Rooms per level (averaged)	2	90
Number of vertices	9	825
Number of edges	9	797
SVG file size per level [kB] (gzip, level 6)	4 (1)	567 (365)
XML file size [kB] (gzip, level 6)	13 (2)	743 (70)

a list of all POIs and their categories, as well as calculating distances between POIs. Finally, it is possible to set a routing destination.

Using this interface the smartphone application can be customized. E.g., it can be used to create a single purpose application which leads the user to a predefined destination or to create a flexible navigation system which allows the user to chose any destination (e.g. from POI list or by tapping on the map).

V. PERFORMANCE EVALUATION

For usability in real-world applications it is essential that the User's View responds quickly. For evaluating response times we conducted two experiments, one using small maps, the other using large maps. Details about the example projects are listed in Table I. As the Editor's View is only used by the creator and maintainer of the map, performance issues about this part of SvgNaviMap are not considered. During development and testing an off-the-shelf PC was perfectly sufficient this purpose.

For our experiments we loaded the complete web application including project data into the client-sided HTML5 application cache. This way effects of different network connections are eliminated. Loading a project consists of three major tasks: 1) Parsing the XML file and building the corresponding XML DOM tree using the JavaScript XMLHttpRequest object; 2) Rendering the SVG map of all levels at once; 3) Drawing the routing graph as overlay on top of the map according to the XML DOM. The corresponding times are measured using JavaScript timing functions in 10 successive runs. Their mean values are shown in Figures 7 and 8.

As hardware platforms we used several Android devices and a desktop PC for comparison. For the desktop PC SvgNaviMap was tested using Chromium 29, Chrome 27, and Firefox 23. On the Android devices the default system browsers were used. For a comparison of CPU and RAM of the used devices as well as OS versions see Table II.

As the Android operating system allows other applications to run in the background the measured values deviate up to

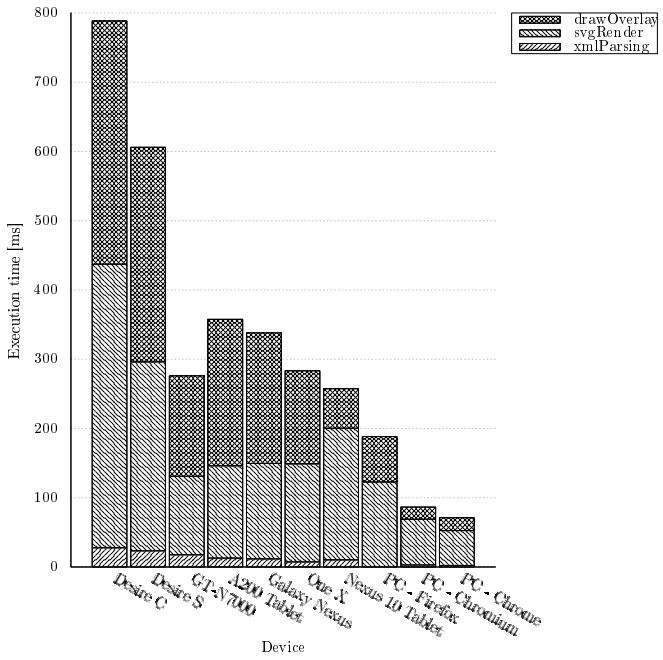


Fig. 7. Loading and parsing times of SVG and XML files of small example project

48% from corresponding mean value. However, as also on end users' smartphones background applications are running, we believe that the mean value is nonetheless an appropriate indicator for showing execution times. Furthermore, for the large example project ignoring the first three low-performance devices, the average deviation from the mean is less than 13%.

Considering Figure 7, a simple, but complete building can be rendered in less than one second even on devices with less than 1GB RAM. For much larger maps (Figure 8), however, the low-memory devices need 16 seconds and more. We further tested two older devices (Huawei Ideos X3, 256MB and HTC Desire, 576MB). Both needed more than 45 seconds for parsing and rendering. Several times the browsers even crashed during the experiment. Clearly, these devices were not meant and are not suited to display several large SVG files at a time.

Smartphones with at least 1GB of RAM, however, were able to show the complete building in 14 seconds or less. The Galaxy 10 tablet needs even less than 4 seconds on average. After the map is completely loaded into memory and displayed there are no further delays. Panning and zooming

TABLE II. COMPARISON OF CPU AND RAM OF TEST DEVICES

Device name	HTC Desire C	HTC Desire S	Samsung GT-N7000	Acer Iconia Tablet A200	Samsung Galaxy Nexus I9250	HTC One X	Samsung Nexus 10 Tablet	PC
OS	Android 4.0.3	Android 4.1.2 CyanogenMod-10	Android 4.1.2	Android 4.0.3	Android 4.2.2	Android 4.1.1	Android 4.2.2	Windows 7 32bit
CPU	Cortex A5 600 MHz	Qualcomm MSM8255 Snapdragon 1 GHz	ARM Cortex A9 Dual-core 1.4GHz	Nvidia Tegra 2 Dual-core 1GHz	TI OMAP 4460 Dual-core 1.2 GHz Cortex-A9	Nvidia Tegra 3 Quad-core 1.5 GHz	Exynos 5250 Dual-core 1.7 GHz Cortex-A15	Intel Core2 Duo 2.33GHz
RAM	512 MB	768 MB	1GB	1GB	1 GB	1 GB	2GB	2GB

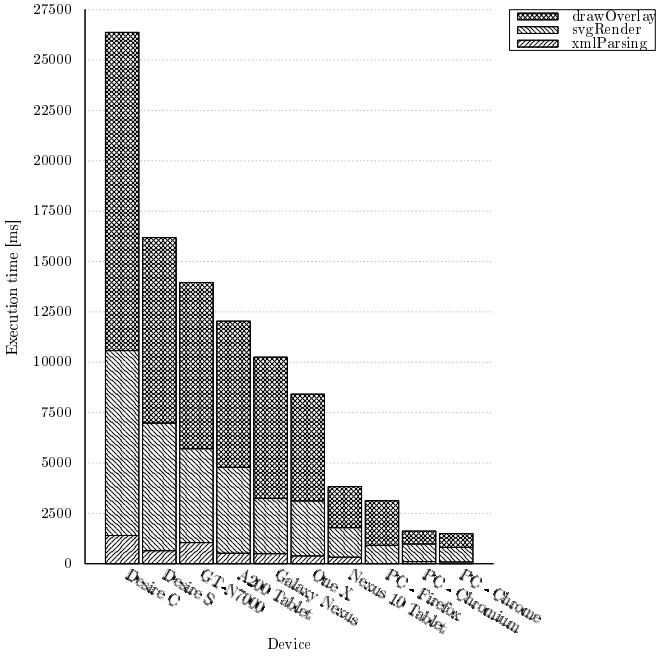


Fig. 8. Loading and parsing times of SVG and XML files of large example project

works smoothly on those devices. Even though we believe that these loading and rendering times can be shortened, they are already acceptable for research purposes.

The three tests with Firefox, Chrome and Chromium were conducted in order to compare the three integrated SVG engines: Gecko (Firefox), WebKit (Chrome), Blink (Chromium). The total loading time is similarly fast with WebKit and Blink, while Gecko is a little slower. Since Android's default browsers are based on WebKit and in the future on Blink, no performance loss is to be expected due to the choice of browser.

VI. FUTURE RESEARCH

Even though displaying routing instructions using arrows is common in outdoor navigation, it cannot be applied directly for stairs and elevators. Our proposed stepmarkers appear to be a well-suited solution for buildings with two or three levels, however, for more stories, especially in combination with elevators, a description tag indicating the destination level would be useful.

Displaying large and detailed SVG files requires a lot of memory and processing power. In order to reduce loading times shown in Figure 7 and Figure 8 we recommend loading and displaying only one map when starting User's View. Maps of all other levels should be loaded in the background and rendered on request only. This will roughly reduce the perceived starting time inversely proportionally to the number of levels.

After being able to load a single level at a time, another optimization approach consists of dividing a single floor in multiple SVG files. SvgNaviMap already supports different levels. Thus, considering different parts of a single level as pseudo-levels, splitting a single floor plan in different maps is possible without changing the Editor's View. Dynamically loading of needed SVG can provide a seamless user experience.

Currently, on loading a navigable map the User's View builds the complete overlay routing tree by inserting graphical SVG objects into the SVG DOM tree. This way, by toggling the visibility of these objects a route between any two vertices can very quickly be shown to the user. However, for most devices building the overlay takes even longer than loading and rendering of the SVG files themselves. To enhance responsiveness and reduce memory requirements, it should be evaluated whether it is feasible to build the graphical routing tree on demand only, i.e. when a route or a POI has to be shown to the user. This would decrease loading time almost by a factor of two.

For the near future we plan to facilitate usage of SvgNaviMap and to improve it such that new maps can be created more easily. Afterwards we will create a navigable model of a complete office building to perform usability tests and collect real-world user experiences.

VII. CONCLUSION

We have implemented and tested an approach to show indoor maps for several floors using the SVG format, which allows viewing and lossless scaling in modern browsers without any plugins. We connected the floor plans with routing information using an XML file allowing to show simple navigation instructions to the user. In order to create routing information, we developed an editor and made it publicly available as open-source project SvgNaviMap.

Experimentally we have shown that even without optimization rendering navigable maps of small buildings on low-end smartphones is possible in less than 1 second. For larger buildings optimization approaches were presented and need to be implemented and tested.

ACKNOWLEDGMENT

The German Federal Ministry of Education and Research, under funding code 03CL26B, supported the research described in this article.

REFERENCES

[1] M. Youssef and A. Agrawala, "On the optimality of wlan location determination systems," University of Maryland, College Park, Tech. Rep. CS-TR-4459, UMIACS-TR-2003-29, April 2003. [Online]. Available: http://hdl.handle.net/1903/1271

[2] ——, "The horus location determination system," *Wirel. Netw.*, vol. 14, no. 3, pp. 357–374, Jun. 2008. [Online]. Available: http://dx.doi.org/10.1007/s11276-006-0725-7

[3] V. V. Nguyen and J. W. Lee, "Self-positioning system for indoor navigation on mobile phones," in *Consumer Electronics (ICCE), 2012 IEEE International Conference on*, 2012, pp. 114–115.

[4] Y. Liu, Q. Wang, J. Liu, and T. Wark, "Mcmc-based indoor localization with a smart phone and sparse wifi access points," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*, 2012, pp. 247–252.

[5] V. Honkavirta, T. Perala, S. Ali-Loytty, and R. Piche, "A comparative survey of wlan location fingerprinting methods," in *Positioning, Navigation and Communication, 2009. WPNC 2009. 6th Workshop on*, 2009, pp. 243–251.

[6] E. Martin, O. Vinyals, G. Friedland, and R. Bajcsy, "Precise indoor localization using smart phones," in *Proceedings of the international conference on Multimedia*, ser. MM '10. New York, NY, USA: ACM, 2010, pp. 787–790. [Online]. Available: http://doi.acm.org/10.1145/1873951.1874078

[7] (2013, June) Google indoormap. [Online]. Available: http://maps.google.com/help/maps/indoormaps/

[8] J. Rocha and N. Alves, "Osm indoor: moving forward," in *OGRS2012 - Symposium proceedings*, 2012, pp. 261–167.

[9] (2013, June) Openstreetmap foundation: Indoorosm. [Online]. Available: http://wiki.openstreetmap.org/wiki/IndoorOSM

[10] (2013, June) Alpen adria universitt klagenfurt - campus-gis. [Online]. Available: http://campus-gis.aau.at/

[11] (2013, June) indoorosm - mapping the indoor world. [Online]. Available: http://indoorosm.uni-hd.de/

[12] T. Springer, "Mapbiquitous–an approach for integrated indoor/outdoor location-based services," in *Mobile Computing, Applications, and Services*. Springer, 2012, pp. 80–99.

[13] (2013, June) Senionlab homepage. [Online]. Available: http://www.senionlab.com/

[14] (2013, June) Loop21 homepage. [Online]. Available: 13. http://www.loop21.net/

[15] (2013, June) Gomogi homepage. [Online]. Available: http://www.gomogi.com/

[16] (2013, June) Lambda:4 homepage. [Online]. Available: http://www.lambda4.com/

[17] (2013, June) meridianapps - how it works. [Online]. Available: http://www.meridianapps.com/howitworks

[18] (2013, July) infsoft - produkt. [Online]. Available: http://www.infsoft.de/produkt

[19] A. Puikkonen, A.-H. Sarjanoja, M. Haveri, J. Huhtala, and J. Häkkilä, "Towards designing better maps for indoor navigation: experiences from a case study," in *Proceedings of the 8th International Conference on Mobile and Ubiquitous Multimedia*, ser. MUM '09. New York, NY, USA: ACM, 2009, pp. 16:1–16:4. [Online]. Available: http://doi.acm.org/10.1145/1658550.1658566

[20] (2013, June) Android honeycomb's browser supports svg. [Online]. Available: http://googlesystem.blogspot.de/2011/02/android-honeycombs-browser-supports-svg.html/