Süleyman Demirel Üniversitesi
Fen Bilimleri Enstitüsü Dergisi
Cilt 22, Sayı 2, 416-428, 2018

Süleyman Demirel University
Journal of Natural and Applied Sciences
Volume 22, Issue 2, 416-428, 2018

DOI: 10.19113/sdufbed.12712

A Navigation Framework with Map Support for Developing Desktop Applications

Özgün YILMAZ[*1]

[1]Ege University, Faculty of Engineering, Department of Computer Engineering, 35100, İzmir

(Alınış / Received: 17.12.2017, Kabul / Accepted: 27.03.2018, Online Yayınlanma / Published Online: 11.05.2018)

Keywords
Navigation framework,
Mobile computing,
Location-awareness,
Map support

Abstract: With the widespread use of mobile devices, location-awareness in software applications has become an increasingly important concept. One important aspect of location-aware applications is the ability to provide map and navigation support for the user. Basically, providing maps and navigational information to the user complements and empowers the notion of location-awareness. Although map and navigation support are widely used in mobile applications, there are circumstances, where they are also useful for desktop applications. There are applications which run without any connection to the Internet and/or demand high computing power, where a smart phone is not sufficient. There are many frameworks for providing map and navigation support in the mobile and web domain. For the desktop domain, map and navigation support is limited. There are no frameworks which provide map and navigation support in an integrated manner. In this paper, an easy to use open source navigation framework for the Java programming language is presented. By using this specific navigation framework, software developers will be able to add map, navigation and path finding support to their applications. In order to minimize network costs for downloading map images over the Internet, a caching strategy is employed. The software developers will be able to integrate navigation support and maps easily to their applications. Finally, demonstration applications implemented using our framework are described to demonstrate the capabilities of our framework.

Masaüstü Uygulamaları Geliştirilmesi için Harita Desteği Sunan Navigasyon Çerçevesi

Anahtar Kelimeler
Navigasyon çerçevesi,
Mobil bilgi işlem,
Konum farkındalık,
Harita desteği

Özet: Günümüzde mobil cihazların yaygınlaşmasıyla birlikte yazılım uygulamalarında konum farkındalık gittikçe önem kazanmıştır. Konum farkında uygulamaların önemli özelliklerinden birisi kullanıcıya harita ve navigasyon desteği sunmaktır. Temel olarak kullanıcıya harita ve navigasyon bilgisi sağlamak, konum farkındalık kavramını tamamlar ve güçlendirir. Harita ve navigasyon desteğinin, mobil uygulamalarda daha yaygın olarak kullanılmasına rağmen, masaüstü uygulamalarında da kullanılmasını gerektiren durumlar bulunmaktadır. İnternet'e bağlı olmadan çalışan ve/veya bir akıllı telefonun yeterli olamayacağı yüksek bilgi işlem gücü gerektiren uygulamalar bulunmaktadır. Mobil ve veb alanında harita ve navigasyon desteği sunan çok sayıda yazılım çerçevesi bulunmaktayken, bu destek masaüstü uygulamalar için sınırlıdır. Harita ve navigasyon desteğini bütünleşik bir şekilde sunan bir yazılım çerçevesi yoktur. Bu çalışmada, Java programlama dilinde kullanımı kolay açık kaynak kodlu bir navigasyon çerçevesi geliştirilmesi amaçlanmaktadır. Geliştiriciler bu çerçeveyi kullanarak; uygulamalarına harita, navigasyon ve yol bulma desteği ekleyebileceklerdir. Haritaların elde edilmesinde, ağ iletişim maliyetlerini azaltmak için bir ön bellekleme stratejisi kullanılmaktadır. Bu sayede, geliştiriciler navigasyon ve harita ile ilgili işlevleri kolayca geliştirdikleri uygulamalarına entegre edebileceklerdir. Son olarak, geliştirilen çerçevenin yeteneklerini göstermek için çerçeve kullanılarak geliştirilen örnek uygulamalar anlatılmaktadır.

*Corresponding author: ozgun.yilmaz@ege.edu.tr

1. Introduction

With the extensive use of mobile devices, computing becomes increasingly mobile and ubiquitous nowadays [1, 2]. Location is an important source of context in mobile and ubiquitous computing [3]. As a result, location-awareness and location based services has gained some importance in the telecommunications industry [4, 5, 6]. One important aspect of location-aware applications is the ability to provide map and navigation support for the user. Basically, providing maps and navigational information to the user complements and empowers the notion of location-awareness.

According to performed literature review, currently there are no open source software frameworks that provide navigation and map support for the Java desktop applications. Such a framework is needed when there is no internet connection and/or high computing power is needed. Although map and navigation support are widely used in mobile applications, there are circumstances, where they are also useful for desktop applications. There are applications which run without any connection to the Internet and/or demand high computing power where a smart phone is not sufficient. This might be the case in an environmental science setting. Environmental scientists sometimes work in remote and isolated places without any internet access to gather or analyze location-aware environmental data on map. The software used by environmental scientists might demand high computing power where a smart phone would be insufficient. There are many frameworks for providing map and navigation support in the mobile and web domain. However, for the desktop domain, map and navigation support is limited. There are no frameworks which provide map and navigation support in an integrated manner. In addition to this, in some situations the level of detail provided by common map providers might be insufficient. National mapping agencies such as Ordnance Survey in Great Britain provide much more detailed maps. The developers might want to use these detailed maps. In this case, a proprietary navigation framework should be used.

For sensing location, a location sensing device, such as a GPS (Global Positioning System) receiver is needed. There are many GPS receivers on the market which can be connected to a computer over the bluetooth or the USB port.

There are products, which provide map support, such as MapXtreme Java [7], but these products lack navigation support and also they are not free. Google Maps [8] and Yahoo Maps [9] provide maps and path finding through Javascript. These services can be used in web applications, but cannot be used directly in desktop applications since third party components are needed to use Javascript from Java. These third party components are expensive. They, acting as a middleware requiring new technologies to be mastered, complicate the problem. In order to simplify this problem, an integrated solution, which provides maps, navigation and path finding in a single framework, is needed.

By using EgeNav framework proposed in this paper, this need can be satisfied. EgeNav is an open source framework where the source codes are available through Git repository from [10]. In EgeNav, the aim is to implement a simple navigation framework for Java software developers. It basically supports many functionalities in an integrated manner and provides an abstraction level for the developers. By using EgeNav, developers will be able to add map, navigation and path finding support to their applications. Textual, visual and audial navigation and direction information and directives can also be presented to the users. External URL-based map and direction services are used to get maps together with directions, making it possible for software developers to integrate maps, navigation and direction support to their applications. As a result, EgeNav provides an abstraction layer for the developers.

In order to get maps and directions using HTTP requests, an upper class structure is defined. These classes are thereafter extended to support Google Maps [11] and Google Directions API [12] directly. By extending these upper classes, other URL-based map and direction services can be used. In addition to getting path finding support from external services, EgeNav also supports raster image-based path finding. A modified flood fill algorithm, which is basically a breadth-first search, is used for this purpose. Turning points, road junctions and textual direction instructions are extracted by processing the image.

Downloading map images over the Internet comes with a cost and reducing this cost is very important. EgeNav uses a caching strategy for reducing network costs. As the user moves, adjacent map tiles are downloaded. Then custom maps can be obtained from the pre-cached map tiles.

EgeNav includes GUI (Graphical User Interface) components. The map panel is one of these components and it provides functionalities such as showing maps, presenting audio, visual and textual directives, recording navigation history, setting map object, type of the map and zoom level and dragging map by mouse to explore other regions. Another GUI component is the navigation information panel which can be used in tandem with the map panel, which continuously updates it, as the navigation information changes.

The subsequent sections of this paper are organized as follows: In Section 2, related research is surveyed

and EgeNav is evaluated according to related work. In Section 3, EgeNav is described in detail. The system architecture and map provider, path finding and map caching components are explained. In Section 4, in order to show the capabilities of EgeNav, a demonstration application is described. Finally, in Section 5, conclusion is discussed.

2. Related Work

As mentioned previously, there isn't any open source navigation framework which supports both desktop and web applications for the Java programming language. MapXtreme Java [7] is a commercial product which provides online maps for developers. Google Maps [8], Yahoo Maps [9] and Bing Maps [13] provide online maps service and direction support in web applications by using Javascript. To use Javascript from Java, third party components are needed. These third party components are expensive. Also they require new technologies to be mastered and this complicates the problem.

Another effort worth mentioning is the OpenStreetMap [14] project. OpenStreetMap is an open source collaborative project, which aims to create a map of the world and this information is collected by volunteers over the world. It is an alternative to the before mentioned proprietary road networks in commercial business geomatics software. OpenStreetMap provides access to spatial data without any costs or fees [15, 16] and it is updated on a daily basis [17]. OpenStreetMap is used for many applications [18], especially in urban studies [19]. Recent research [15, 20-25] shows that OpenStreetMap can compete with commercial products in many countries around the world and it is continuously getting better. Road information of OpenStreetMap data shows good accuracy and completeness as a result of the convergence of volunteered GPS traces [20, 26].

As mentioned earlier Google Maps, Yahoo Maps and Bing Maps require Javascript. Third party paid products are needed to run Javascript from Java. One of these products is JxBrowser [27] which provides browser and Javascript support for Java applications. Today, Java 7 platform supports Javascript through the JavaFX [28] package, but again this complicates the problem, because this approach requires web programming and Javascript knowledge. To simplify matters, we need a framework which integrates getting maps, navigation and getting directions in a single solution. EgeNav framework is intended to fulfill this role.

There are similar works related to EgeNav. A comparison of these related work is shown in Table 1. In the table, related work are evaluated according to following criteria:

- **Map support:** This feature denotes if the related work provides showing of maps. It also includes basic functionalities such as browsing the map, zooming in and out, changing map type and etc. All of the related work uses an external map service.
- **Navigation support:** This feature denotes whether the related work has ready-to-use infrastructure for managing and showing navigation information. Navigation information consists of speed, average speed, heading direction, time elapsed, traveled distance, and etc.
- **Path finding:** Path finding refers to the assistance by providing information and instructions for going from one location to a destination location. It is used to help the users find their way. It can include visual, textual and audial assistance.
- **Custom server support:** The related work, listed in the table, all use an external map service. Custom server support denotes whether the related work supports different map servers.
- **Caching of maps:** To increase the performance and minimize the costs, downloaded maps are stored in a permanent storage. When a map is requested, cache is searched if it is possible to construct the map from the cached maps without downloading.
- **Java desktop support:** To support developing visual Java desktop applications, a component should be compatible with a Java GUI (Graphical User Interface) widget kit. An example of such a widget kit is Java Swing.
- **Open source:** One important aspect of this work is that it is open source and free of any charge. So related work are also evaluated according to this criterion.

MapPanel [29] is a free visual map viewer component intended to be used in user interfaces. It uses OpenStreetMap as the map server. By using MapPanel, developers can integrate mapping into their applications. Basic mapping functionalities such as browsing the map, zooming in and out are supported, but this component provides only mapping support [30]. It lacks navigation, path finding and directions support. There is also no custom server support. It is open source and can be used in developing Java desktop applications since it supports Java Swing.

The other one of the related works is JXMapViewer2 [31]. Developed by Swing Labs, it is a visual map viewer component similar to MapPanel. It can use its own map server or OpenStreetMap server. Similar to MapPanel, browsing the map, zooming in and out is supported whereas navigation, path nding and directions services are not supported. It is open source and supports developing Java desktop applications.

Table 1. Comparison of EgeNav with other related work

	Map Support	Navigation Support	Path Finding	Custom Server Support	Caching of Maps	Java Desktop Support	Open Source
MapPanel	Yes	No	No	No	Yes	Yes	Yes
JXMapViewer2	Yes	No	No	Yes	Yes	Yes	Yes
osmdroid	Yes	No	No	Yes	Yes	No	Yes
skobbler	Yes	Yes	Yes	No	Yes	No	No
EgeNav	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Osmdroid [32] is an open source replacement for Android's MapView class. Basically, it is a view which displays a map. It supports basic operations such as browsing the map, zooming in and out, etc. It is designed for use in Android applications and does not support Java desktop applications. As a map provider it uses OpenStreetMap, but can be extended to support another map server. Osmdroid lacks navigation, path finding and directions support.

Skobbler [33] provides a SDK (Software Development Kit) for developing software with map and navigation capabilities based on OpenStreetMap. Skobbler supports showing maps, navigation, path finding and directions support features. It only supports OpenStreetMap and does not support other map servers. It is aimed at developing Android and web applications, but it does not support Java desktop applications. It is a commercial product and is not open source.

In EgeNav, getting maps, navigation support, finding paths and getting direction support are integrated in a single framework. Google Static Maps [11] is used for providing maps and Google Directions API [11] is used for path finding. Custom map and direction servers are supported by inheritance. Downloaded map images are optionally cached for later use similar to the all of the other related work. EgeNav also supports different map servers. It is an open source project and its usage is free of any charge or fee. As a result, EgeNav is novel in terms of these mentioned features over the previous related work.

3. EgeNav

At the present time, a navigation framework should be able to provide map and navigation support to its users. The user should be able to zoom in and out, change map type and drag map to explore new regions. By navigation support, navigation information such as speed, average speed, heading direction, time elapsed, traveled distance, and etc. is implied. It is best these information are calculated and displayed to the user visually over a map as the user moves. This can enhance the comprehension of the current situation as the user's location changes continuously.

One of the alternatives for sensing location is Global Positioning System (GPS). It is a very accurate and successful outdoor positioning system. GPS includes 24 satellites plus three redundant backups orbiting around the earth. Inexpensive GPS receivers can calculate its position utilizing these satellites with an accuracy of 10 meters for approximately 95 percent of measurements. More expensive units can reach an accuracy of 1 to 3 meters 99 percent of the time. GPS can be used anywhere in the world without any fees for free [34, 35].

Path finding services, for providing assistance going from one location to a destination location, should also be included in a navigation framework. There are many GPS navigation devices in the commercial market. Today, most of the smart phones come with a built-in GPS receiver [36]. These devices are used widely and they provide path finding and routing services. They also provide turn-by-turn navigation directions to a human in charge of a vehicle via text or speech. Therefore, nowadays a navigation framework cannot be thought without path finding.

In a navigation framework, if the maps are to be downloaded over a network, then a caching strategy should be employed to minimize the communication costs. This strategy reduces costs by constructing new maps from the previously downloaded maps. This will reduce the network costs when the user passes from the same locations a second time. For example, a person, who goes to work from home every day and vice versa, would benefit from this.

EgeNav framework is developed with these previously mentioned requirements in mind and it conforms fully to these requirements. After implementation, the EgeNav framework is tested using JUnit and discovered errors, faults and bugs are corrected. In this section, EgeNav framework is described in detail. Subsequent subsections of this section will describe the system architecture, map provider, path finding and map caching components in detail.

3.1. System architecture

The infrastructure of EgeNav is depicted in Figure 1. EgeNav consists of map provider, path finding, map caching, navigation, visual (GUI), and text-to-speech components. Each component consists of many classes. Map provider component provides maps by downloading map images from external URL-based map servers. The map caching component is used by map provider component and it caches downloaded

maps for later use to minimize network costs. The use of map caching component is optional. Constructing new maps by partially combining previously cached maps are supported.

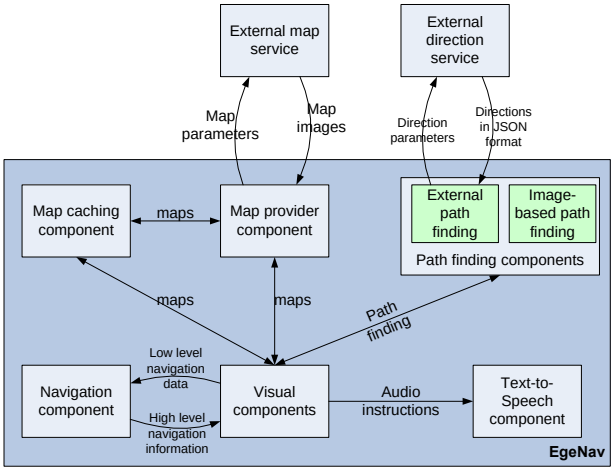


Figure 1. System architecture of EgeNav.

The path finding component provides information and instructions for going from one location to a destination location. It consists of external path finding and image-based path finding sub-components. Similar to the map provider component, external path finding subcomponent gets directions from URL-based external direction services. Image-based path finding subcomponent provides direction assistance by applying a modified flood fill algorithm on the map images. Turning points, road junctions and textual direction instructions are extracted. Navigation component is responsible for storing navigation history (traveled waypoints) and computing navigational information such as speed, average speed, heading direction, time elapsed, and traveled distance.

Visual components consist of the map panel and the navigation information panel. These panels are Java Swing components which are compatible with other Java Swing visual components. The map panel class provides functionalities such as showing maps, path finding, presenting audio, visual and textual instructions for path finding, storing navigation history, showing navigation information, and dragging map by mouse to explore other regions. To provide these functionalities, the map panel uses map provider, navigation, path finding, map caching and text-to-speech components. The navigation information panel can be used for showing navigation information such as speed, average speed, heading direction, time elapsed and total traveled distance. The navigation information panel is updated continuously by the map panel as the navigation information changes.

In order to convert text to speech, FreeTTS [37] library is utilized in the text-to-speech component. This component is used for speaking textual path finding instructions by a robotic voice.

3.2. Map provider component

In EgeNav, the map provider component supports the construction of map URLs and the downloading of map images. Once downloaded, maps can be shown to the user in map panel which is also part of the framework. Maps can be browsed by dragging the mouse, changing zoom level, and map type. Maps are obtained from map servers as image files through HTTP requests. EgeNav directly supports Google Static Maps (GSM) as the map service, and it can be extended to support any map server through inheritance, because it is object oriented. The developers construct URLs by just calling related methods of the *MapURL* class, and they don't need to worry about the URL string and syntax rules. If Google Static Maps is used, then there will be no need to build a map server. In Figure 2, an example GSM map URL composed of several parts is shown.

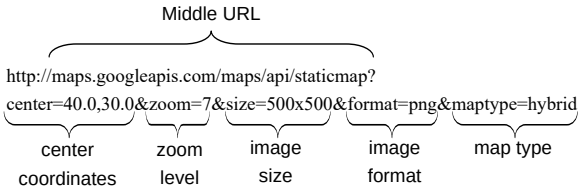


Figure 2. A map URL.

3.3. Path finding component

The path finding component is used to assist users with direction information for going from one location to a destination location. The user can be assisted either visually by marking the path on the map or by providing turn-by-turn navigation directions via text or speech.

For path finding and getting directions support, two different approaches are used in EgeNav. These approaches are compatible by inheritance, and they share the same super class. One of these approaches is getting path finding service from an external directions server, and it is similar to getting maps from map servers. By default, Google Directions service is used for getting external directions support, but EgeNav can be extended to support other path finding services. When Google Directions service is used, the results are retrieved in JSON (JavaScript Object Notation) [38] format, and in order to process the results, an open source Java library, Gson API [39] is used.

The other approach to path finding is using image processing techniques. For this approach, the term “image-based path finding” is used throughout the text. The map image is processed to find a route between two given points on a map image, and no

external service is used for this purpose. One restriction is that the paths on the map image must be visible and without text labels.

A modified flood fill algorithm, which is basically a breadth-first search, is used for image-based path finding. Flood fill algorithm is mainly used for painting an entire bounded area with a color. Flood fill is used in implementation of the bucket paint tool in some graphical editing software. Flood fill algorithm starts with a pixel, and paints it to the target color. Then painted pixel's adjacent pixels (north, west, south and east directions) are visited, and if necessary they are painted. This operation continues recursively until all of the pixels in the area are painted to the target color.

The main idea behind our modified flood fill algorithm used in EgeNav is recording the coordinates of the visited points. When the history is recorded in a flood fill algorithm, we get a path finding algorithm. Pseudocode for this path finding algorithm is shown in Figure 3.

Flood fill algorithm can be implemented using recursion as well as using a stack. Since recursion

Input: image to be processed (img), initial location (init), destination location (dest), and map color model (mcm)
Output: A path consisting of points for going from initial location to destination location

```
FUNCTION findPath(img, init, dest, mcm)
   x = dest.x;
   y = dest.y;
   target_color=img.getRGB(x , y)

   IF (!mcm.isOnTrack(target_color)) THEN //destination is not on a path
      return null;
   END IF

   pencil=mcm.getABorderColor()  //get a non-track (border) color
   path.add(init)
   paths.add(path)
   WHILE (NOT paths.isEmpty())
      temp=paths.firstElement()
      x2 = temp.getLastX()
      y2 = temp.getLastY()
      img.setRGB(x2, y2, pencil);

      IF ((x2 == x) AND (y2 == y)) THEN
         return temp  //A path is found
      ELSE
         IF (x2 > 0 AND mcm.isOnTrack(img.getRGB(x2 - 1, y2))) THEN
            img.setRGB(x2 - 1, y2, pencil)
            Path p = temp.clone()
            p.add(new Point(x2 - 1, y2))
            paths.add(p)
         END IF

         IF (y2 > 0 AND mcm.isOnTrack(img.getRGB(x2, y2 - 1))) THEN
            img.setRGB(x2, y2 - 1, pencil)
            Path p = temp.clone()
            p.add(new Point(x2, y2 - 1))
            paths.add(p)
         END IF

         IF (x2 < img.getWidth() - 1 AND mcm.isOnTrack(img.getRGB(x2 + 1, y2))) THEN
            img.setRGB(x2 + 1, y2, pencil)
            Path p = temp.clone()
            p.add(new Point(x2 + 1, y2))
            paths.add(p)
         END IF

         IF (y2 < img.getHeight() - 1 AND mcm.isOnTrack(img.getRGB(x2, y2 + 1))) THEN
            img.setRGB(x2, y2 + 1, pencil)
            Path p = temp.clone()
            p.add(new Point(x2, y2 + 1))
            paths.add(p)
         END IF

      END IF
      paths.remove(0)
   END WHILE
   return null        //Path not found
END FUNCTION
```

Figure 3. Pseudocode for the modified flood fill algorithm.

might cause stack overflow error in large areas, instead of recursion, a queue data structure is used in this implementation to overcome possible stack overflow errors. If there is any, the path that will be returned by this algorithm is a shortest path, because the algorithm starts from a point (center) and orderly visits adjacent points in the west, east, north and south directions, thus expands from center to outwards in a rhombic pattern.

The application of the path finding algorithm given in Figure 3 to a map image, is shown in Figure 4, where a route between two selected points is found and painted in red.

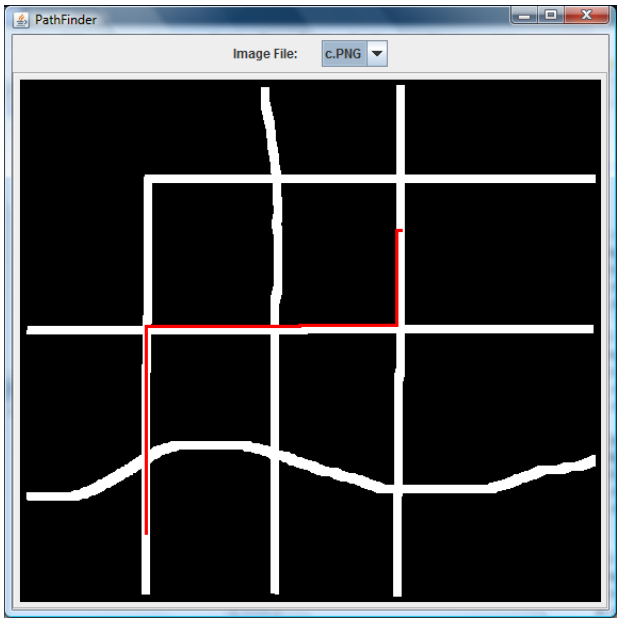


Figure 4. The application of the path finding algorithm to a map image.

In addition to path finding, to make a detailed description of the situation, road junctions and turning points are also extracted by processing map images. Then textual instructions such as "after 200 m. turn right" are generated from this data. The steps to extract junctions and turning points are illustrated in Figure 5. The first step is finding a path between two points with the modified flood fill algorithm. This path consists of a set of adjacent points. In normalization step, this path is converted to a path consisting of lines. In this conversion, the idea is to remove unnecessary points by checking whether it is possible to travel to a reference point from that point. The points between these two points are unnecessary and are removed. Now we have converted our path to a line path. Then the line path is centered using vertical and horizontal threshold values. These threshold values are in pixels and can be detected automatically by EgeNav framework.

After the centering process, junctions are extracted by walking on the line path pixel by pixel and applying half circle search. In half circle search, while looking at the heading direction, 180 degrees of field of view is examined to see if there is any obstacle closer than a predetermined threshold value. If there is no obstacle between two angles' values, then average of these two angles denote a possible direction. If a point has only one possible direction, then this point is not a junction. To be marked as a junction, a point should have at least two possible directions.

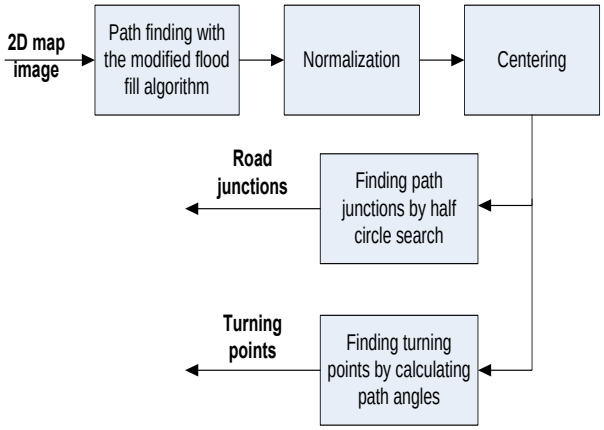


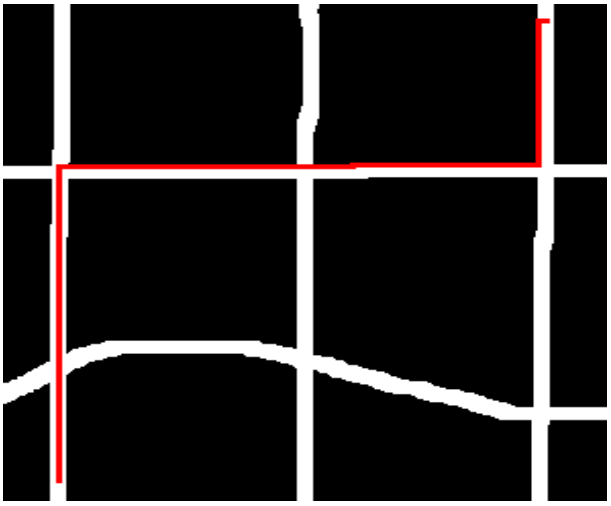
Figure 5. Process for extraction of road junctions and turning points.

In this approach, while walking on the line pixel by pixel for a junction point, several consequent points might be marked as junction points, because of the width of the road. In this case, middle one of the repeating points is stored, and the others are eliminated to get only one junction point.

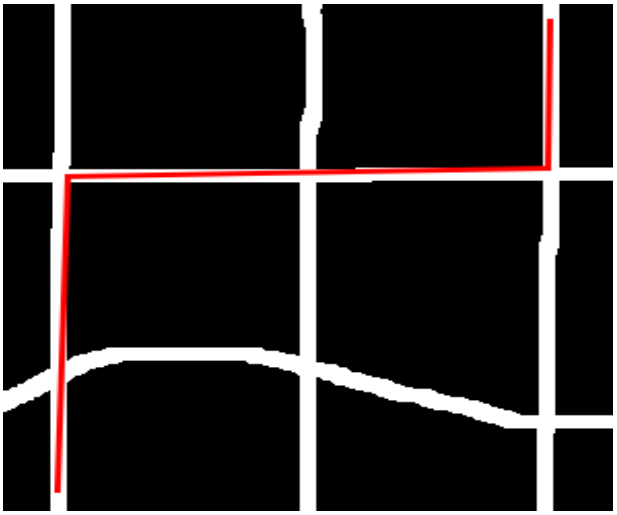
Turning points are extracted by examining lines in the path and computing angles between the lines. Turning points are also used for generating basic text direction instructions such as "turn right after 200 meters" for the user.

The application of path finding, normalization, and centering steps of the extraction process shown in Figure 5 to a map image is shown in Figure 6. The path is shown in red color. The map image is the same image as the image in Figure 4, but its unnecessary parts are cropped.

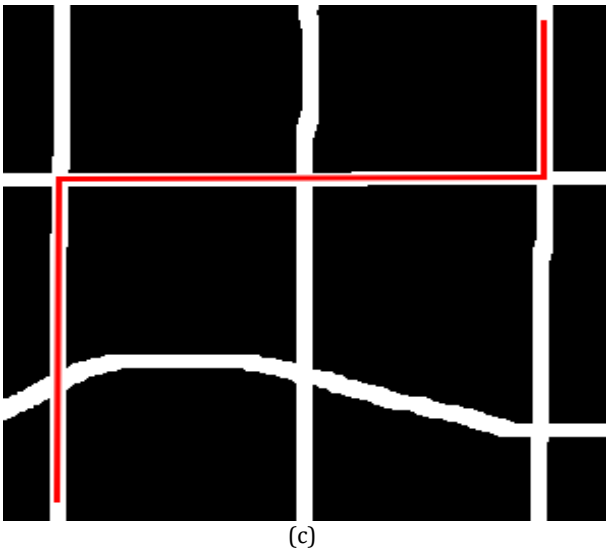
To demonstrate EgeNav's image-based path finding capabilities, a demonstration application is implemented using EgeNav framework. This demonstration application is available as an executable jar file (Ege-Nav_Demo2.zip) from https://drive.google.com/drive/folders/0B0jxyO3H3yKHX19GTTJQVEdkbm8. A screen capture of this demonstration application is shown in Figure 7. The right panel shows the map image. The image file can be changed from the combo box above the image. All of the image files in the application directory are listed in the combo box. The application finds a path between origin and destination points as well as



(a)



(b)



(c)

Figure 6. Path **a)** after path finding step **b)** after normalization step **c)** after centering step.

junction points and turning points on this path. The origin and destination points are selected by clicking on the white area of the map. The left panel shows information about the found path. Origin, destination, and auto-detected horizontal and vertical threshold values are shown to the user. Also the road junctions and turning points on the path are computed and listed on separate tables. The junctions and turning points can be viewed on the map by just clicking on a row of the table. The junction and turning points are indicated by transparent circles painted in blue and green respectively. The first junction and the first turning point are selected and shown in this figure.

3.4. Map caching

EgeNav provides caching support for map images as mentioned earlier. The aim is to store downloaded maps in a permanent storage and to retrieve, as needed, from permanent memory without downloading it over the Internet again. By doing this, network costs are reduced, especially if the user passes from same locations (for example going to work from home and vice versa) often.

When map caching is used, map cache object is asked for maps. If the requested map is found in the cache, it is acquired from local storage and returned without downloading it over the Internet. If the map is not found, then entire map image or missing parts are downloaded and stored in the cache. A later request to the same map is replied by reading the local image file(s).

When map caching is used in EgeNav, maps are downloaded as tiles to form a whole, and as a result, they improve performance. Also, new maps can be obtained by combining cached maps. For example: if a request comes for a map that is partly contained in the cache, then only the adjacent missing tiles are to be downloaded. Then the requested map is constructed by composing all of the necessary tiles. To illustrate this feature, suppose the situation in Figure 8. The user is at point P1, so map tile 1 is downloaded and is currently being shown to the user. The user moves to P2, and we need to get the map denoted by the red dotted lines in the figure. In this situation, tiles adjacent to tile 1, namely 2, 3, and 4 are downloaded. New map is constructed by using tiles 1 - 4. From now on, any map spanning these four tiles can be composed without downloading anything.

Caching is performed according to a strategy. If the downloaded maps are never deleted, then as the time passes, size of the stored data will continuously increase. In EgeNav, limits for map cache can be specified. This limit might be a number limit, a time limit, or both. By setting a number limit, the number of stored maps can be limited. The time limit is intended to limit the maximum storage time of an image file, which is required by some map servers. For example, GSM time limit is 30 days [40]. If a map is downloaded and stored; after 30 days it should be deleted.

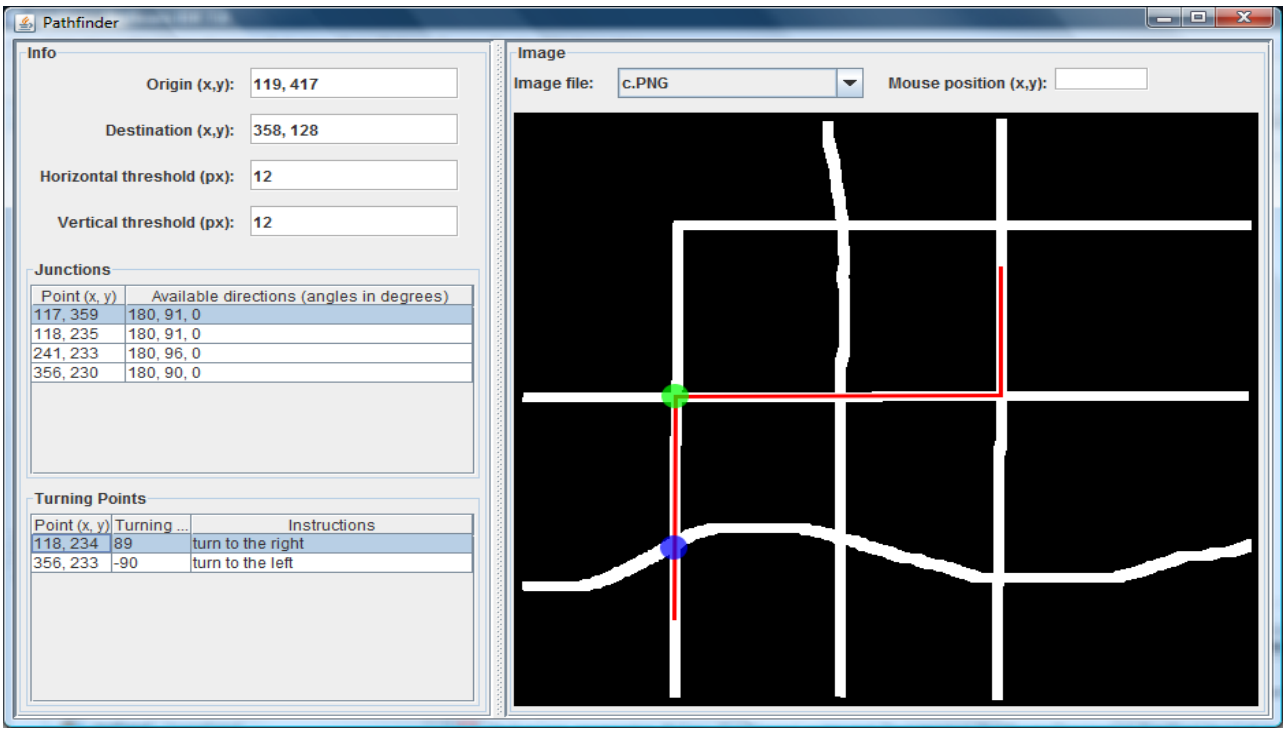


Figure 7. Screen capture of the demonstration application showing EgeNav's image-based path finding capabilities.

Every map is assigned a usage count which is incremented when that map is requested. If there is a number limit and this limit is exceeded, then map with the least usage count is deleted. If there is more than one map with same usage counts, then the oldest map is deleted. Time limit is also checked and if the time limit is exceeded, then one or more map images are deleted to obey the time limits.

Two types of approaches used for storing cache information are memory stored map cache and database stored map cache. By default, these two approaches are supported by EgeNav, allowing developer to decide which approach to use according to his/her needs.

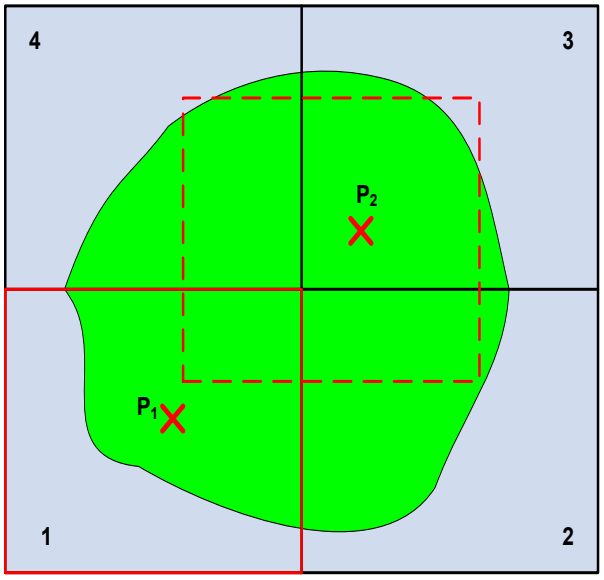


Figure 8. Map caching in EgeNav.

In memory stored cache, cache information is read from a file in the hard disk and transferred to main memory when the program starts. In main memory, information is stored in a dynamic list. During program execution, this dynamic list is used and manipulated. When the program ends, it is saved back to the file in the hard disk.

In database stored cache, cache information is stored in a database file. Modifications are reflected on the fly to the database file. For the database engine, SQLite [41] library is used, because it is serverless, file-based, and lightweight.

4. Demonstration Application

In this section, another demonstration application is introduced to show most of the features of EgeNav. The demonstration application is available as an executable jar file (EgeNav_Demo1.zip) from https://drive.google.com/drive/folders/0B0jxyO3H3yKHX19GTTJQVEdkbm8. A screen capture of this application is shown in Figure 9. The demonstration application provides changing map type and zoom level, moving from one location to another location, getting navigation support (path finding) for going from one location to another location and simulating a journey passing by several locations. Google Maps and Google Directions services are used through EgeNav framework in this application.

The user can change map type and zoom level from the map properties panel which is located in the upper left corner. The user can move the map to view other parts of the map by dragging the map image.

Right to the map properties panel is the navigate panel where the user can enter latitude and longitude values of a location in decimal format. When the user presses the Go button, the user's current location is updated, and this new location is added to navigated points list as well as the computed navigation information being shown to the user on the navigation information panel. The map is also updated to reflect the new location, such as the navigated path is shown in blue and the heading direction is shown by a blue arrow, as can be seen in Figure 9. If the user presses the Reset button, then the navigation history is cleared.

The user can get direction support from his/her current location to a destination location by using the navigation support panel at the upper right corner of the frame. In order to get direction support, the user should enter destination location coordinates or an address and then press Get support button. Note that the precondition necessary here is that there should be at least one added navigation point by using the navigate panel. If a path is found, then this path is shown in red color on the map to the user as shown in Figure 9. If the user is following the red path as in Figure 9, then textual direction instructions are shown in navigation information panel and are also read out loud by a robotic voice.

The demonstration application also enables making simulations, and the goal is to simulate getting location coordinates from a GPS receiver. Latitude and longitude values of the passed waypoints and time difference in milliseconds between this point and the previous point are defined in a text file as comma separated values. To start a simulation, the name of the simulation file should be entered, and then the Start simulation button should be pressed. The demonstration application comes with a ready to use simulation file, and also when the application starts, the coordinates pre-entered into the *Navigate* and the *Navigation support* panels are compatible with this simulation file to demonstrate full capabilities of the demonstration application. To see this full demonstration with the pre-entered data, the user first moves to the specific location by pressing the Go button in the navigate panel. Next the user gets direction support by pressing the Get support button in the navigation support panel and finally presses the Start simulation button to start the simulation. The result of these actions is shown in Figure 9. The red line on the map shows the path that should be followed to arrive at the destination location. The blue line shows the followed path by the user, and it is formed by connecting the navigation points defined in the simulation file. The blue arrow shows the heading direction.

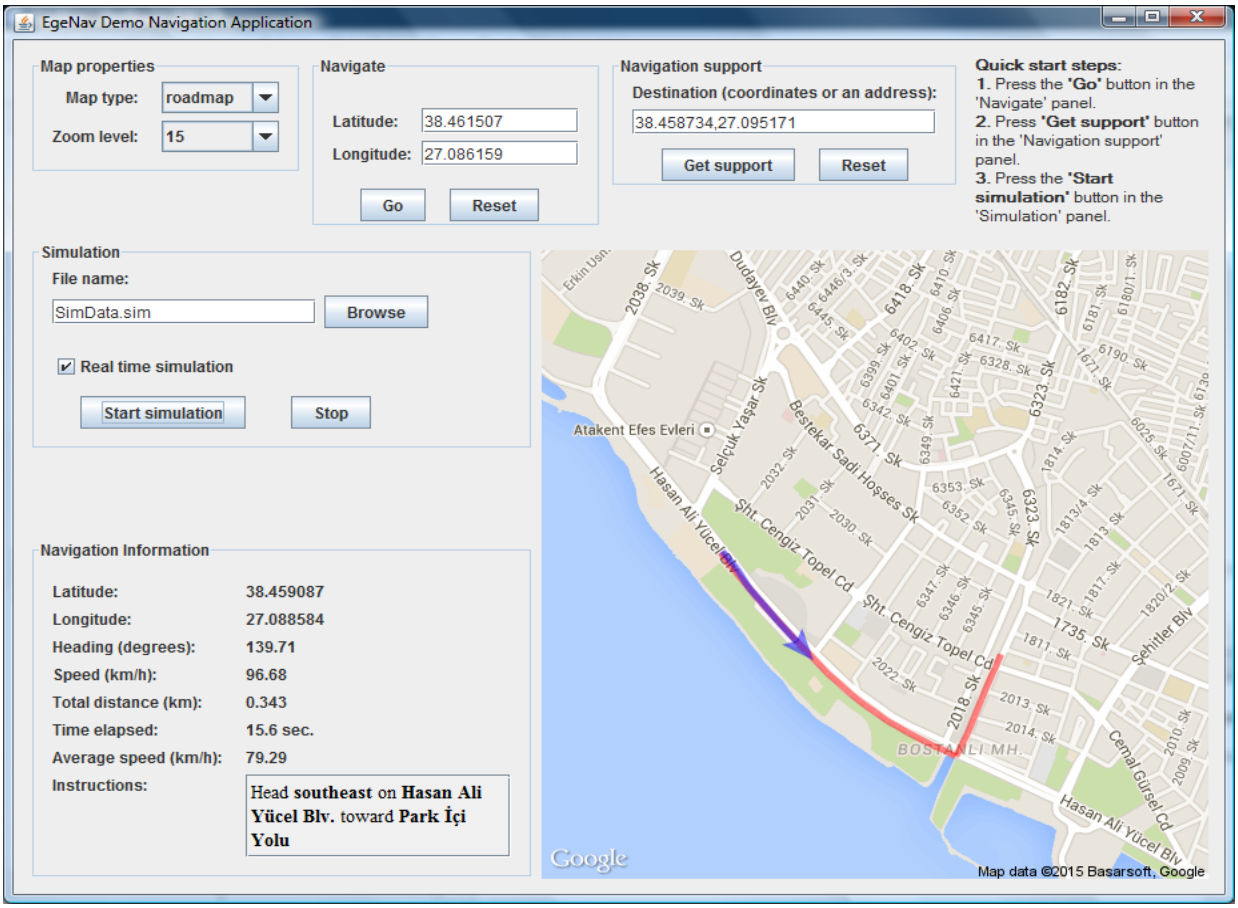


Figure 9. Screen capture of the demonstration navigation application.

Figure 9 shows the data for the moment 15.6 seconds after the simulation is started. Navigation information is shown on the navigation information panel located on the left side of the map panel. Latitude, longitude, heading direction, speed, total distance, elapsed time, average speed, and instructions for path finding are shown to the user in the navigation information panel. Instructions for path finding are also read out loud by a robotic voice.

Simulation can be a real time simulation or a non-real time simulation, depending upon checking or unchecking the Real time simulation check box. In real time simulation, time differences between location points are taken into consideration, while in non-real time simulation, these differences are ignored. Non-real time simulation is useful if the journey is too long, because the user does not have to wait for a long time. Feeding location data to EgeNav's navigation component is accomplished by a thread which reads the data from the simulation file and forwards latitude and longitude values to the EgeNav's navigation component. If the simulation is real time, then the thread considers the time difference and waits for the time difference before feeding the location data.

5. Discussion and Conclusion

Location-awareness in software applications has become an important matter with the widespread use of mobile devices. The ability to provide map and navigation support is an important aspect of location-awareness. Providing maps and navigational information to the user can enrich the notion of location-awareness.

In this paper, EgeNav, which is an easy to use navigation framework, is presented. Until now, there have been no open source software frameworks that have provided navigation and map support for the Java programming language in an integrated manner. By EgeNav, this shortfall can be overcome. By using EgeNav, software developers will be able to add map, direction, and navigation support to their software applications.

In the scope of this study, a generic map provider infrastructure to download maps, as image files from map servers according to some given properties such as location, size, zoom level, map type, etc., is developed. Then, to enable the use of Google maps, this generic component is extended to support Google Static Maps.

To minimize network costs for downloading map images over the Internet, a caching strategy is used. This strategy is implemented in the map caching component of EgeNav. Composition of new maps by combining previously downloaded maps are also possible. Maps are downloaded as adjacent tiles in order not to download maps with overlapping parts over and over again and to increase efficiency. If a map is needed which spans multiple tiles, it can be composed from related tiles without downloading anything.

For path finding and getting direction support, a generic path finding infrastructure is implemented. This generic infrastructure is extended to support two types of path finding. One of them is URL-based path finding, which uses an external path finding service. The other type is image-based path finding where the map image is processed to find a route between two points. URL-based path finding component is further extended to support Google Directions service. Path finding components are integrated with the map provider component, and by using EgeNav's path finding capabilities, it is possible to find a route between two points, show this route on map and provide textual and audio descriptive instructions.

EgeNav also provides navigation services. Navigation history can be recorded, and waypoints can be shown on maps. Speed, average speed, heading direction, elapsed time, and traveled distance are computed. EgeNav framework also includes ready to use GUI components for the Java programming language. These components are derived from Java Swing components and are fully compatible with the Swing components. These components include map panel and navigation information panel. Map panel works in collaboration with the map provider, navigation, path finding and map caching components and shows map images. Navigation information such as passed waypoints, heading direction and direction support can be shown on the maps. Navigation information panel shows navigation information composed of speed, average speed, headed direction, elapsed time and traveled distance. Navigation information panel is updated automatically by the map panel, as the navigation information changes.

To demonstrate the capabilities of EgeNav framework, two demonstration applications are implemented by using EgeNav framework and these applications are described in detail. The first application described in Section 3.3 shows raster image-based path finding capabilities of EgeNav. By scanning the map image, a valid route is found and road junction points, turning points and textual instructions for path finding are extracted from the map image. Second application shows getting maps and direction support from external services while it facilitates simulating a journey passing from several waypoints.

As a future work, it is planned to prepare detailed documentation of EgeNav for software developers.

Acknowledgment

This project was partially supported by Ege University's Scientific Research Project program under grant number 12-MUH-001.

References

[1] Chen, G., Kotz, D. 2000. A Survey of Context-Aware Mobile Computing Research (Report No. TR2000-381). Dartmouth College, USA.

[2] Dearman, D., Inkpen, K., Truong, K. 2010. Mobile Map Interactions during a Rendezvous: Exploring the Implications of Automation. Personal and Ubiquitous Computing, 14(1), 1-13.

[3] Becker, C., Dürr, F. 2005. On Location Models for Ubiquitous Computing. Personal and Ubiquitous Computing, 9(1), 20-31.

[4] Rao, B., Minakakis, L. 2003. Evolution of Mobile Location-Based Services. Communications of the ACM, 46(12), 61-65.

[5] Ficco, M., Pietrantuono, R., Russo, S. 2010. Supporting Ubiquitous Location Information in Interworking 3G and Wireless Networks. Communications of the ACM, 53(11), 116-123.

[6] Ficco, M., Palmieri, F., Castiglione, A. 2014. Hybrid Indoor and Outdoor Location Services for New Generation Mobile Terminals. Personal and Ubiquitous Computing, 18(2), 271-285.

[7] Pitney Bowes Software. MapInfo MapXtreme Java Edition Datasheet. http://www.pbinsight.com/files/resource-library/resource-files/mapxtremejava-data-sheet.pdf (Date Accessed: 15.02.2018).

[8] Google Inc. Google Maps JavaScript API v3. http://developers.google.com/maps/documentation/javascript (Date Accessed: 15.02.2018).

[9] Yahoo Inc. Yahoo! Maps Web Services. http://developer.yahoo.com/maps/ (Date Accessed: 15.02.2018).

[10] Yılmaz O. EgeNav - A Simple Navigation Framework. https://github.com/ozgunyilmaz/EgeNav (Date Accessed: 15.02.2018).

[11] Google Inc. Google Static Maps API V2 Developer Guide. https://developers.google.com/maps/documentation/staticmaps/ (Date Accessed: 15.02.2018).

[12] Google Inc. Google Maps Directions API. https://developers.google.com/maps/documentation/directions/ (Date Accessed: 15.02.2018).

[13] Microsoft Corporation. Bing Maps Developer Resources. http://www.microsoft.com/maps/ (Date Accessed: 15.02.2018).

[14] OpenStreetMap Foundation. OpenStreetMap. http://www.openstreetmap.org/ (Date Accessed: 15.02.2018).

[15] Zielstra, D., Zipf, A. 2010. A Comparative Study of Proprietary Geodata and Volunteered Geographic Information for Germany". 13th AGILE International Conference on Geographic Information Science, Guimarães, Portugal.

[16] Haklay, M., Weber, P. 2008. OpenStreetMap: User-Generated Street Maps. IEEE Pervasive Computing, 7(4), 12-18.

[17] Zhou, Q. 2018. Exploring the relationship between density and completeness of urban building data in OpenStreetMap for quality estimation. International Journal of Geographical Information Science, 32(2), 257-281.

[18] Rickles, P., Ellul, C., Haklay, M. 2017. A suggested framework and guidelines for learning GIS in interdisciplinary research. Geo: Geography and Environment, 4(2), e00046.

[19] Mobasheri, A. 2017. A rule-based spatial reasoning approach for OpenStreetMap data quality enrichment; case study of routing and navigation. Sensors,17(11), 2498.

[20] Haklay, M. 2010. How Good Is Volunteered Geographical Information? A Comparative Study of OpenStreetMap and Ordnance Survey Datasets. Environment and Planning B: Planning and Design, 37(4), 682-703.

[21] Ludwig, I., Voss, A., Krause-Traudes, M. A. Comparison of the Street Networks of Navteq and OSM in Germany. Pp. 65-84. Geertman, S., Reinhardt, W., Toppen, F., ed. 2011. Advancing Geoinformation Science for a Changing World, Springer, Berlin, Heidelberg.

[22] Neis, P., Zielstra, D., Zipf, A. 2011. The Street Network Evolution of Crowdsourced Maps: OpenStreetMap in Germany 2007-2011. Future Internet, 4(1), 1-21.

[23] Hayakawa, T., Imi, Y., Ito, T. 2012. Analysis of Quality of Data in OpenStreetMap. 2012 IEEE 14th International Conference on Commerce and Enterprise Computing. September 9-11, Hangzhou, China.

[24] Wang, M., Li, Q., Hu, Q., Zhou, M. 2013. Quality Analysis of Open Street Map Data. 8th International Symposium on Spatial Data Quality, May 30 - June 1, Hong Kong.

[25] Sehra, S. S., Singh, J., Rai, H. S. 2014. A Systematic Study of OpenStreetMap Data Quality Assessment. 2014 11th International Conference on Information Technology: New Generations, April 7-9, Las Vegas, USA.

[26] Wan, T., Lu, H., Lu, Q., Luo, N. 2017. Classification of High-Resolution Remote-Sensing Image Using OpenStreetMap Information. IEEE Geoscience and Remote Sensing Letters, 14(12), 2305-2309.

[27] TeamDev.JxBrowser.http://www.teamdev.com/jxbrowser (Date Accessed: 15.02.2018).

[28] Oracle. JavaFX - The Rich Client Platform. http://www.oracle.com/technetwork/java/javase/overview/javafx-overview-2158620.html (Date Accessed: 15.02.2018).

[29] Rutz S. Java Swing MapViewer. http://mappanel.sourceforge.net/(Date Accessed: 15.02.2018).

[30] OpenStreetMap Foundation. OpenStreetMap About. http://www.openstreetmap.org/about (Date Accessed: 15.02.2018).

[31] Steiger M. JXMapViewer2. https://github.com/msteiger/jxmapviewer2 (Date Accessed: 15.02.2018).

[32] Osmdroid. OpenStreetMap-Tools for Android. https://github.com/osmdroid/osmdroid (Date Accessed: 15.02.2018).

[33] Skobbler. Smart mobile technology based on OpenStreetMap.http://developer.skobbler.com/ (Date Accessed: 15.02.2018).

[34] Hightower, J., Borriello, G. 2001. Location Systems for Ubiquitous Computing. Computer, 34(8), 57-66.

[35] Hardegger, M., Roggen, D., Tröster, G. 2015. 3D ActionSLAM: Wearable Person Tracking in Multi-Floor Environments. *Personal and Ubiquitous Computing*, 19(1), 123-141.

[36] Barkhuus, L., Polichar, V. 2011. Empowerment through Seamfulness: Smart Phones in Everyday Life. Personal and Ubiquitous Computing, 15(6), 629-639.

[37] FreeTTS. FreeTTS 1.2.3 - A speech synthesizer written entirely in the Java programming language. http://freetts.sourceforge.net/ (Date Accessed: 15.02.2018).

[38] JSON. Introducing JSON. http://www.json.org/ (Date Accessed: 15.02.2018).

[39] Google. A Java serialization/deserialization library to convert Java Objects into JSON and back. https://github.com/google/gson (Date Accessed: 15.02.2018).

[40] Google. Google Maps/Google Earth APIs Terms of Service. https://developers.google.com/maps/terms (Date Accessed: 15.02.2018).

[41] SQLite. About SQLite. http://www.sqlite.org/about.html (Date Accessed: 15.02.2018).