Route Tracking of Moving Vehicles for Collision Avoidance Using Android Smartphones

Ehab Ahmed Ibrahim, Said El Noubi, and Moustafa H. Aly

Abstract

Android Smartphones are widely used nowadays. They have a lot of capabilities that made our life easier and more comfort. They opened the area of Android programming used to develop applications suitable for these smartphones. Our target, in this paper, is to propose an application that helps in collision avoidance between moving vehicles (especially trains) by tracking their routes. This application is a modified version of Route Tracker Application [Harvey and Deitel (Android for programmers an app-driven approach, 2011)], we use the Route Tracker Application on each smartphone that installed in the moving vehicle, to track its route. Then we save all trains points in a database, construct a map shows these points. In this map, a different color is assigned to each vehicle to distinguish it. This map is shown on a smartphone found in a control room. Using points in this map, the distance between vehicles is calculated (distance between points of vehicles). When two vehicles, for example, become too closer to each other, the inspector, in the control (monitoring) room, must take an action, he generates an alarm to warn the drivers of both vehicles so they can adjust their speed to avoid collision.

Keywords

ADT • AVD • DDMS • KML • GPX • JSON

Introduction

The Android operating system [1–4] was developed by Android, Inc., which was acquired by Google in July 2005. In November 2007, the Open Handset Alliance™—a 34-company consortium initially and 81 now was formed to develop Android, driving innovation in mobile technology and improving the user experience while reducing cost. Android is used in numerous smartphones, e-reader devices and tablet computers.

One benefit of developing Android applications is that it is open source and free operating system. This allows to view Android source code and see how its features are implemented. One can also contribute to Android by reporting bugs or by participating in the open source project discussion groups.

Android applications are developed with Java [5, 6]; the world's most widely used programming language. Java was a logical choice for the Android platform, because it is powerful, free and open source. Java is used to develop large-scale enterprise applications to enhance the functionality of web servers and to provide applications for consumer devices (e.g., cell phones, pagers and personal digital assistants) and for many other purposes.

It enables to develop applications that will run on a variety of devices without any platform-specific code. Experienced

E.A. Ibrahim (✉)
Arab Academy for Science, Technology and Maritime Transport, Alexandria, Egypt
e-mail: ehab_ahmed@aast.edu

S. El Noubi
Faculty of Engineering, University of Alexandria, Alexandria, Egypt
e-mail: saidelnoubi@yahoo.com

M.H. Aly
College of Engineering and Technology, Arab Academy for Science, Technology and Maritime Transport, Alexandria, Egypt
e-mail: mosaly@aast.edu

K. Elleithy and T. Sobh (eds.), *New Trends in Networking, Computing, E-learning, Systems Sciences, and Engineering*, Lecture Notes in Electrical Engineering 312,
DOI 10.1007/978-3-319-06764-3_81,

Java programmers can quickly dive into Android development, using the Android (Application Programming Interfaces) APIs and others available from third parties.

Java is object oriented and has access to powerful class libraries that help in developing applications quickly. One can write applications that respond to various user-initiated events such as screen touches and keystrokes. In addition to directly programming portions of applications, one may also use Eclipse to conveniently drag and drop predefined objects such as buttons and textboxes into place on your screen, and label and resize them. Using Eclipse [7] with the Android Development Tools (ADT) Plugin, one can create, run, test and debug Android applications quickly and conveniently, and one can visually design his user interfaces.

Application Model

Our developed application depends mainly on Google Maps and Route Tracker Applications (discussed in next sections). We install a smartphone in each moving vehicles, on the phone we displaying a map that shows the vehicles moving points. Another map is used to show points of all vehicles. We used Google Maps Application to perform the following tasks:

1. Show a Google Map on each smartphones.
2. Locate the vehicles points on the map.
3. Add markers to these points (one marker for each point with the ability to use different markers colors for different points).
4. Draw a line that connects all these points together (route path).

At this stage, we completed an easy but very important task. Now, we become familiar with Google Maps and can deal, easily, with it. Then we move to the Route Tracker Application. This application targets to track only one route (for one moving vehicle).

To avoid collisions, we need to track routes for more than one vehicle (at least two), so we need to modify the Route Track Application to fit in our situation. We install the Route Tracker application for each smartphone, then collect the points of each vehicle (these points are the Route Tracker output) and save them on a database.

Finally, we show these points on a map that can be accessed by the vehicle driver, thus all drivers have an overview of the routes of other vehicles. If any sudden change occurs in any vehicle path, it updated automatically to all vehicles, so drivers can take a fast action towards this change.

Google Maps Application

Google Maps is one of the many applications bundled with the Android platform. In addition to simply using the Maps application, one can also embed it into his own applications and make it do some very cool things. This Android Google Maps application [8] shows how to use Google Maps in Android applications and how to programmatically perform the following:

- Change the views of Google Maps (street and satellite view).
- Display a Particular Location on the map.
- Display the Zoom Controls.
- Obtain the latitude and longitude of locations in Google Maps.
- Perform geocoding and reverse geocoding.
- Add multiple markers to Google Maps.

Obtaining a Maps API Key

One needs to apply for a free Google Maps API key before he can integrate Google Maps into his Android application. API version 2 is already in use today but for application in this paper we use API version 1.

Displaying the Map

To display the Google Maps in Android application, modify the main.xml file. One shall use the <com.google.android.maps.MapView> element to display the Google Maps and modify main class to extend from the MapActivity class, instead of the normal Activity (Fig. 1).

Displaying a Particular Location

By default, the Google Maps displays the map of the United States when it is first loaded. However, one can also set the Google Maps to display a particular location.

Getting the Location That Was Touched (Geocoding)

After using Google Maps for a while, one may wish to know the latitude and longitude of a location corresponding to the position on the screen that one has just touched. Knowing this information is very useful as one can find out the address of a location, a process known as Geocoding.

Adding Multiple Markers to Google Map

The objective is to add multiple markers to Google Map Application for Android. One may take a scenario where having a list of coordinates of different points and wants to display a marker on each point.

Fig. 1 Displaying Google maps Android Application

One must add a class that extends ItemizedOverlay which consists of a list of OverlayItems. This handles sorting north-to-south for drawing, creating span bounds, drawing a marker for each point, and maintaining a focused item. It also matches screen-taps to items, and dispatches focus-change events to an optional listener.

Drawing a Path or Line Between Several Locations

The objective is to draw a line between two points on Google Maps let's say point "A" and point "B". Repeat the process to draw a line between point "B" and point "C".

Route Tracking of Moving Vehicles

First, to run this Route Tracker application or to create your own application using the Google Maps API, you'll need to obtain a unique API key from Google. Applications must be signed with a digital certificate before they can be installed on a device. When you are building and testing applications, the ADT Plugin handles this automatically by creating a debug certificate and using it to sign your applications.

An Android device (such as smartphone) must be installed on the moving vehicle that one wants to track its route. This smartphone must have internet access to receive the map images, to acquire a GPS signal, the smartphone must have line-of-sight with the GPS satellite (the signal can take several minutes).

Once the Route Tracker application is running on the smartphone, and a GPS signal is received, a Toast appears on the screen saying that the GPS signal has been acquired. At this point, touch "Start Tracking" button.

As the vehicle moves, its route is marked with a red line. Open the application's menu and touch the Satellite item to display a satellite image rather than a standard street map.

One can switch back to a street map by selecting the menu's Map item. When the vehicle is stopped (have finished its route), touch "Stop Tracking" button. Touching "Start Tracking" again erases your route from the map and starts tracking a new one (Figs. 2, 3, and 4).

Then, one can extend the process of route tracking to more than one moving vehicle (two or more). For each vehicle, a smartphone must be installed and connected to internet and GPS Satellite.

We can use this approach in railway stations for tracking the routes of trains. For example, take a scenario of two trains tracking. A map is displayed on a smartphone on both trains. The first train positions are drawn on the map as blue balloons. The map is updated periodically with each point the train reaches. For the second train, points are drawn as a red balloons. The map on each train is showing its route. When the other train reaches a station (new point), the map is updated to show both train locations on the same map. So each train driver has an information about the current state. He can take a decision if any unusual event occurs. For example, if second train becomes closer, he must slow down and send alarm to this train. Also if one train is stopped suddenly, the train driver must send an alarm to all other trains to pay attention. Another situation is when a train gets close to a station or a crossing, the driver must be aware of that to slow down its speed.

A third map is displayed on a smartphone on a control room showing trains routes. The inspector on the control room also has the ability to take action such as open or close station barrier, send a signal to all other trains to slow down

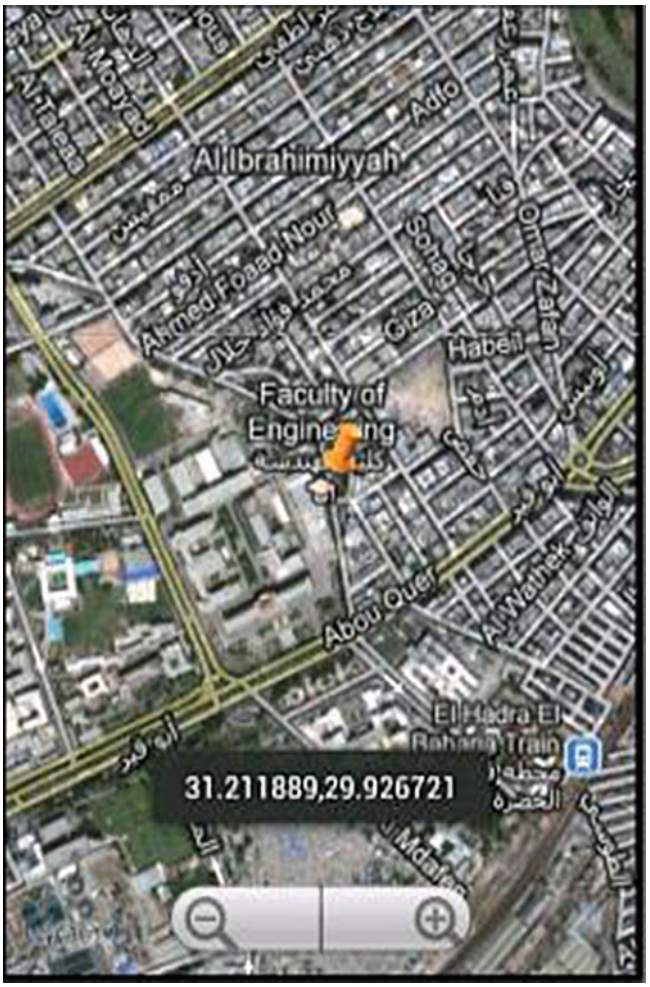

Fig. 2 Adding markers and geocoding

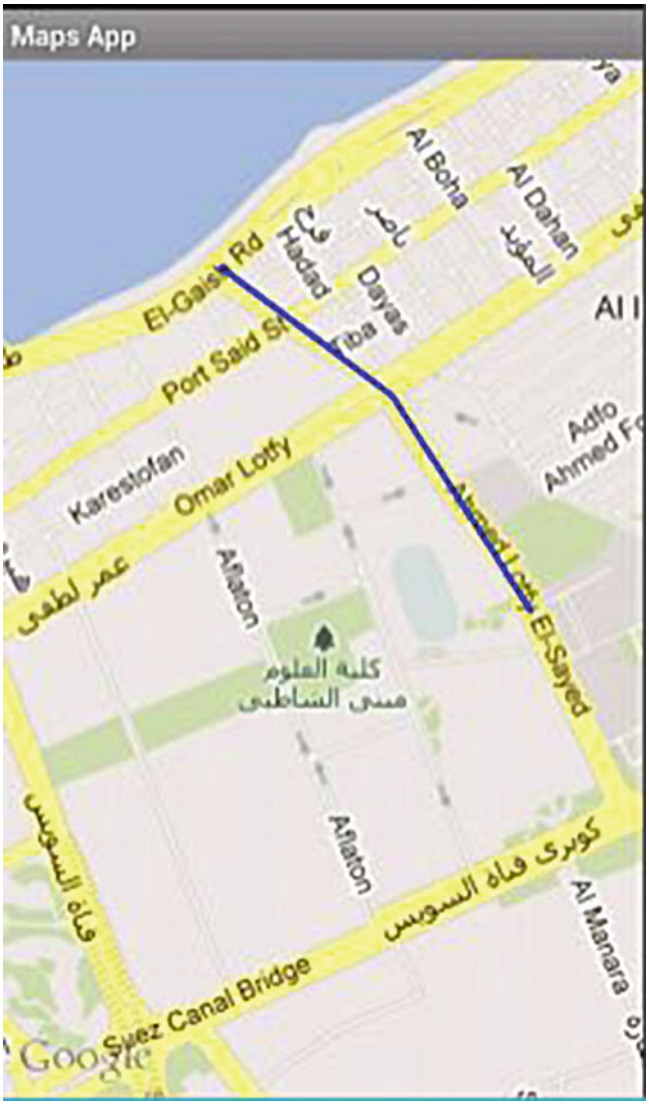

Fig. 3 Drawing a route path between three points

or completely stop until a crossing situation is perfectly handled. By this procedure, one can avoid (or at least decrease) collisions between trains and improve the safety factor of the train railway network.

Simulation Procedure

This simulation is carried out in an **A**ndroid **V**irtual **D**evice (**AVD**) [9] which is configured to use the Google APIs.

Sending GPS Data to an AVD

The Android emulator enables to send GPS data to an AVD. So, one can test this application without an actual Android smartphone. One can send a single point (latitude and longitude) manually Using **D**alvik **D**ebug **M**onitor **S**erver (**DDMS**) [10]. Actually, sending only one point is not suitable for tracking moving vehicle. So, one needs a series of points to really simulate a GPS data.

One can use a (**K**eyhole **M**arkup **L**anguage) [11] file (file with kml extension) that can be generated by simply drawing the desired routes on Google Earth software, save it in kml format and then convert this Google Earth routes to a format suitable for Eclipse Android ADT plugin using google earth to android ADT tool (http://ge2adt.applicationspot.com/).

This method has a drawback that the kml file contains only points but without their times (points are located on the Map all at once). So, one needs another way to take the factor of time into consideration so as to perfectly simulate GPS data.

To do so, one shall use a file containing GPS data in GPS Exchange Format. Such files typically end with the gpx extension and are called GPX files that one can load and "play" from the ADT Plugin DDMS perspective.

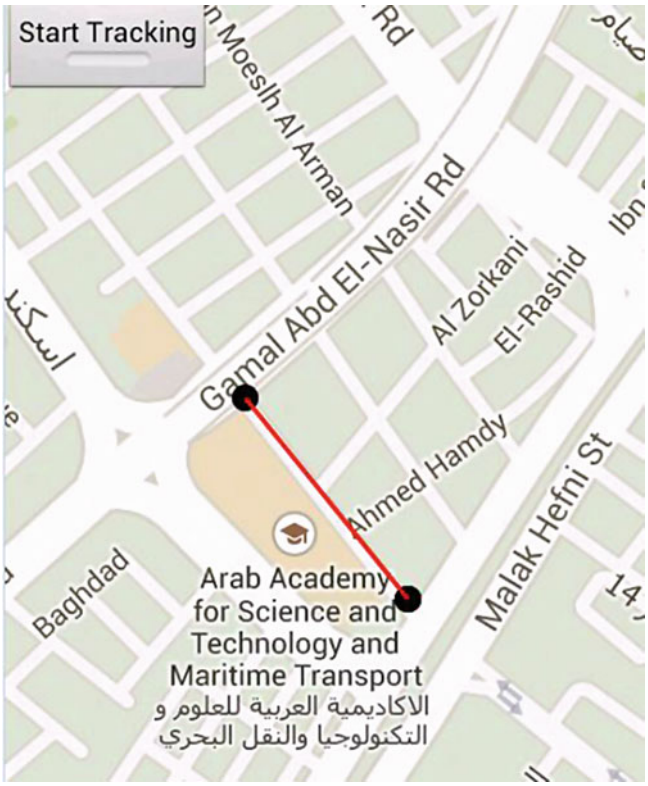

Fig. 4 Route path of one vehicle (street view)

Fig. 5 Android—MySQL communication process

To generate these files, one can do the same steps of creating kml file, draw the desired routes on Google earth, save it in kml format and then convert this kml file format to gpx file using gpsbabel tool (www.gpsbabel.org). Note that, Android emulator uses GPX version 1.1 format data (Version 1.0 is not supported).

For our case of tracking two (or more) trains, we will have more than gpx file (one for each train) and need to add a point for each train according to its gpx file. To work with all these files we need to organize them and save their points on a database. This can be performed using a WAMP server.

WAMP is acronym for Windows, Apache, MySQL and PHP [12, 13]. WAMP software is one click installer which creates an environment for developing PHP, MySQL web application. By installing this software you will be installing **Apache**, **MySQL** and **PHP**.

The Android application and a remote MySQL database cannot communicate with each other directly without interpreter. This interpreter will be PHP. The process in a visual form would look like this (Fig. 5):

To accomplish our developed Route Tracking application we must go through these steps:

1. Creating MySQL database containing one tables with fields (point id, point latitude, point longitude, train number, and point time).
2. Connecting to MySQL database using PHP. Using PHP statements to export the table data into an XML format that map can retrieve through asynchronous JavaScript calls.
3. Adding a row in MySQL database corresponding to a new point, storing its latitude, longitude, train number and time of point generation.
4. Reading all rows from MySQL database to display all points on the map.
5. Creating the Android application that connect all these components together.

Simulation Results

In this paper, we simulated a crossing scenario of two trains. A hypothetic route is chosen for each train (a route of four points or stations). The present Android application is a modified version of Route Tracker application discussed in section "Google Maps Application". In each train we run the ordinary Route Tracker application, save points to MySQL database. The map shows the route path until other train adds another point in the database. At this point, the map changes (using intent) to another view showing points of both trains. After a sufficient period of time (permits the driver to understand other train situation) the map returns again to its original view (original train route path).

Another map is shown on the central room. It is a webview activity, its source is a PHP file that reads all points from database and locates them on a Google Map distinguishing between train one and train two points by using different markers for each train. It always shows both trains routes.

The distance between trains is calculated using information about the trains' points (latitude and longitude) which are read from database. We use the Haversine form to calculate the distance from points coordinates (Figs. 6, 7, 8, and 9).

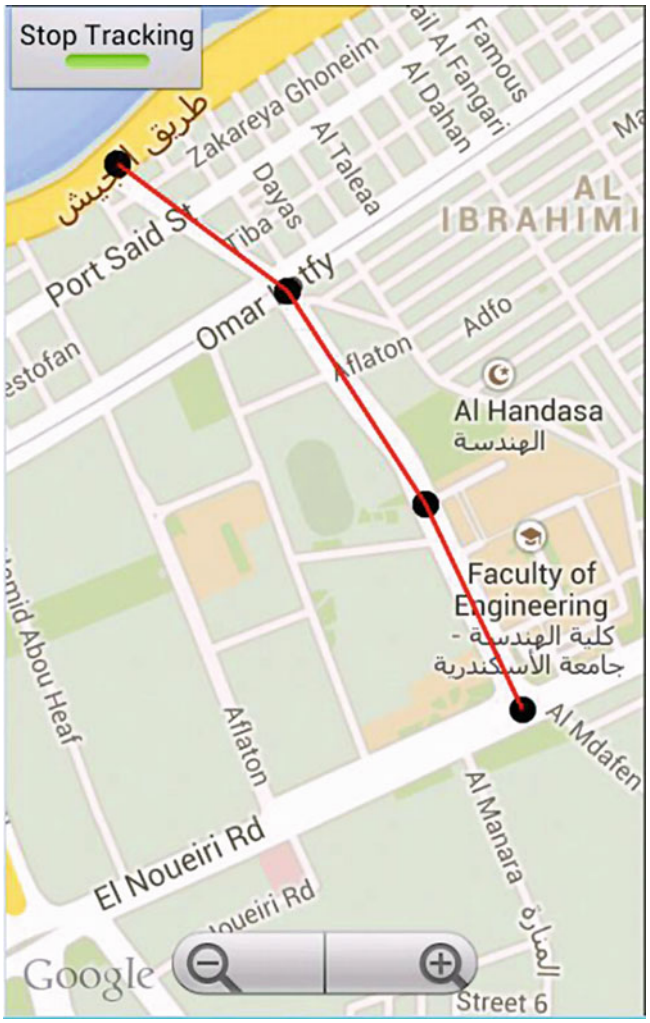

Fig. 6 Route path of train 1 (Map 1 view)

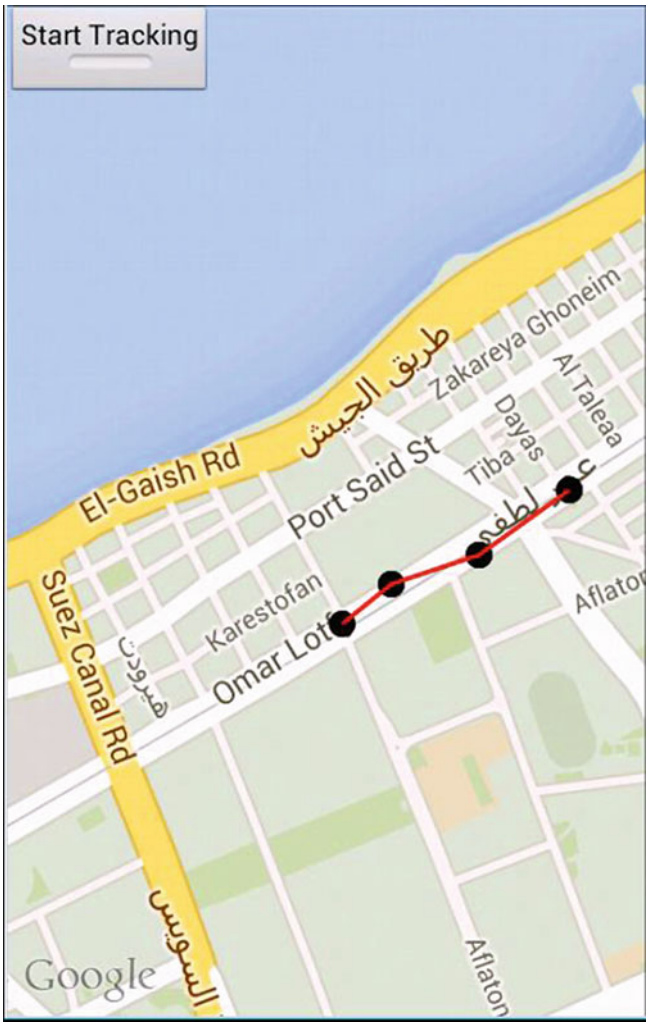

Fig. 7 Route path of train 2 (Map 2 view)

We put a distance of 5 km as a threshold. If the distance between the two trains becomes less than 5 km, an alert is generated (as shown in Fig. 8). The distance between trains is calculated periodically with each point update, and the check of 5 km is also repeated until all points are read from database.

Conclusion

In this paper, an application is developed that aims to participating in avoiding or decreasing the rate of collision between moving vehicles (in our simulation we worked on trains because the number of collision between trains is very large indeed). To achieve our goal, we start with Google Maps Application that, simply, displays a google map on Android phone, then, zoom controls are added to the map.

After that, we add markers for particular locations, and draw a line connecting the points drawing on the map. Then we move to Route Tracker Application that is already developed. We Display a map showing the route path of each train on a smartphone located in the train. This map is updated with points of the other train periodically. As Route Tracker Application is developed for tracking path for only one train, we need to modify this application to be suitable for tracking two or more trains.

To do so, we save the points of each train route in MySQL Database to construct a map showing routes of all trains (by locating points of all trains on this map). This map is displayed on a central (control) room.

If the two trains become close to each other (we took a distance of 5 km as a threshold), the smartphone generates an alarm to warn the drivers. The inspector on the control room must take an action based on this alarm. He must inform both drivers to adjust their speed to avoid collision.

Using this technique, we can decrease the collision rate between trains and improve its safety factor.

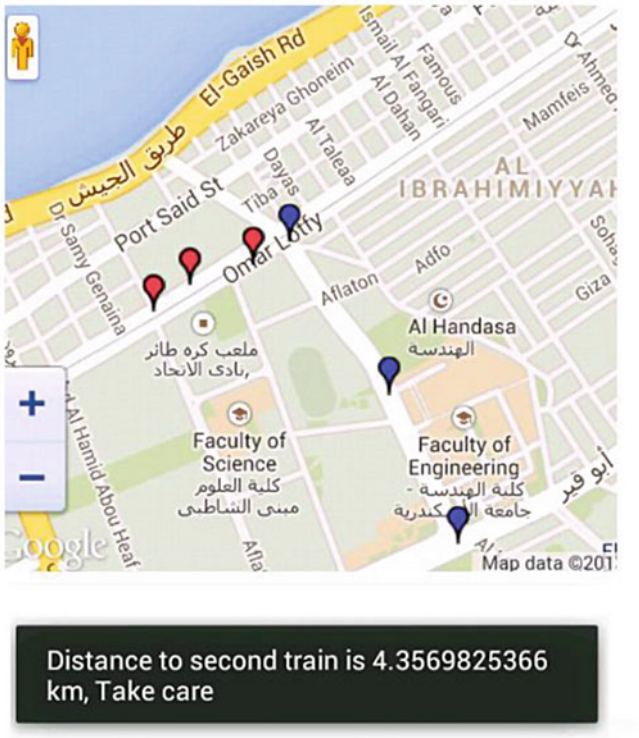

Fig. 8 Alarm generated when distance is less than 5 km, action is required (Control Room Map)

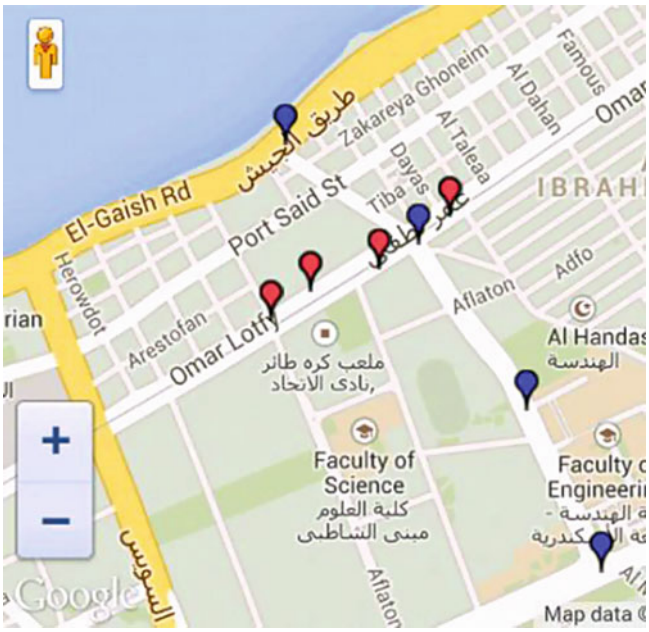

Fig. 9 Action is taken, each train completes its path (Control Room Map)

References

1. Paul, Harvey and Abbey Deitel, "Android for programmers an app-driven approach," 2011, pp. 291-319.
2. http://chrisrisner.com/31-Days-of-Android, accessed 28-10-2013.
3. Wei-Meng Lee, "Beginning Android 4 Application Development," Wrox, 2012, pp. 251-292, 351-392.
4. Mark L. Murphy, "Android Programming Tutorials," CommonsWare, 2011, pp. 1-27.
5. http://www.ntu.edu.sg/home/ehchua/programming/java/J2_Basics.html, accessed 8-11-2013.
6. "Sams Teach Yourself Java in 24 Hours (Covering Java 7 and Android)", 6th ed., Sams Publishing, 2012, pp. 4-102, 343-371.
7. http://developer.android.com/sdk/installing/bundle.html, accessed 20-10-2013.
8. http://mirnauman.wordpress.com/category/android, accessed 25-10-2013.
9. http://codebutler.com/2012/10/10/configuring-a-usable-android-emulator, accessed 1-11-2013.
10. http://developer.android.com/tools/debugging/ddms.html, accessed 5-11-2013.
11. http://en.wikipedia.org/wiki/Keyhole_Markup_Language, accessed 5-11-2013.
12. http://www.androidhive.info/2012/05/how-to-connect-android-with-php-mysql, accessed 8-11-2013.
13. http://www.mybringback.com/tutorial-series/12924/android-tutorial-using-remote-databases-php-and-mysql-part-1, accessed 9-11-2013.