Location prediction in a context-aware environment

Wim van Ravesteijn

Report of a bachelor assignment in Telematics from the University of Twente, Enschede, The Netherlands

Supervisors:
Dr. L. Ferreira Pires
P. Dockhorn Costa, MSc

Enschede, The Netherlands
December, 2004
Abstract

Nowadays, handheld devices such as mobile phones and PDAs are getting increasingly powerful in terms of their processing and storage capabilities. Communication networks like WLAN, GPRS and UMTS are becoming faster and widespread. These technologies bring a multitude of opportunities that have not been possible before, like the introduction of highly personalised and context-aware applications. These applications make use of information on the user’s context (e.g., location, temperature, handheld device capabilities, etc.) to provide tailored services. Context-aware platforms are being developed to support context-aware applications, by offering generic and reusable context-aware functionality. In this bachelor project we have identified and evaluated a couple of methods for location prediction that could be incorporated in context-aware platforms. In this report we will introduce these different methods for location predication and present our conclusions.
Table of contents

1 INTRODUCTION ..........................................................................................................1
  1.1 BACKGROUND 1
  1.2 OBJECTIVES 2
  1.3 APPROACH 2
  1.4 STRUCTURE 2

2 BACKGROUND.............................................................................................................3
  2.1 WEB SERVICES 3
  2.2 MOBILE ENVIRONMENT 4
  2.3 CONTEXT AWARENESS 4
  2.4 PERSONALIZATION 4

3 WASP PLATFORM.......................................................................................................6
  3.1 OVERVIEW 6
  3.2 PERSONALIZATION 7
    3.2.1 UserProfiler 8
    3.2.2 PredictInterests 8
  3.3 MONITORING 8
  3.4 REGISTRY 9
  3.5 USER CONTEXT MANAGER 9
    3.5.1 UserLocationHistoryService 9
    3.5.2 UserServicesService 9
    3.5.3 UserLocationSimulatorService 9
    3.5.4 UserLocationService 10
    3.5.5 UserLocationUpdateService 10
    3.5.6 BuddyListenerInterface 10
    3.5.7 userTriggerNotificationInterface 10
  3.6 PICTURE STORE 10
  3.7 FWA SERVER 11
  3.8 MAP SERVER 11
  3.9 POI SEARCH SERVICE 11

4 LOCATION PREDICTION............................................................................................12
  4.1 SAME SPEED 12
  4.2 SAME CURVE 14
  4.3 NEIGHBOURING CELLS 16
  4.4 PAST MOVEMENTS 17
  4.5 FOLLOWING ROADS 18
  4.6 AGENDA 19
  4.7 COMBINATIONS 19
  4.8 CONCLUSION 19

5 IMPLEMENTATION ...................................................................................................21
  5.1 OVERVIEW 21
  5.2 GENERAL CODE 22
  5.3 SAME SPEED 23
    5.3.1 Code 23
    5.3.2 Results 24
    5.3.3 Possible improvements 24
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4 SAME CURVE</td>
<td>24</td>
</tr>
<tr>
<td>5.4.1 Code</td>
<td>24</td>
</tr>
<tr>
<td>5.4.2 Results</td>
<td>25</td>
</tr>
<tr>
<td>5.4.3 Possible improvements</td>
<td>26</td>
</tr>
<tr>
<td>5.5 FOLLOWING ROADS</td>
<td>26</td>
</tr>
<tr>
<td>5.5.1 Code</td>
<td>26</td>
</tr>
<tr>
<td>5.5.2 Results</td>
<td>28</td>
</tr>
<tr>
<td>5.5.3 Possible improvements</td>
<td>28</td>
</tr>
<tr>
<td>6 CONCLUSIONS AND FUTURE WORK</td>
<td>30</td>
</tr>
<tr>
<td>6.1 CONCLUSIONS</td>
<td>30</td>
</tr>
<tr>
<td>6.2 FUTURE WORK</td>
<td>30</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>32</td>
</tr>
<tr>
<td>APPENDIX A: JAVA CODE</td>
<td>33</td>
</tr>
</tbody>
</table>
1 Introduction

In this chapter we introduce to the contents and goals of this report. We start giving background information, followed by the objectives and approach. We finish with the structure of the whole report.

1.1 Background

Nowadays, handheld devices such as mobile phones and Personal Digital Assistants (PDAs) are getting increasingly powerful in terms of their processing and storage capabilities. Communication networks like Wireless Local Area Network (WLAN), General Packet Radio Service (GPRS) and Universal Mobile Telephone System (UMTS) are becoming faster and widespread. These technologies bring a multitude of opportunities that have not been possible before, like the introduction of highly personalised and context-aware applications. These applications make use of information on the user’s context (e.g., location, temperature, handheld device capabilities, etc.) to provide tailored services. The WASP project [ebb04] has developed a platform for context-aware applications.

The WASP platform is a system of Web Services that gives support to context-aware applications. The platform is able to record locations of users and their interests. There are tools available to show the movements of users. Currently the platform is being further developed in the Freeband Awareness project [weg04]. The Awareness project combines results of a couple of projects that have developed context-awareness tools. The ideas described in this paper are therefore applicable to the AWARENESS project, in so far as it extends the WASP platform.

A lot of extensions to the platform are possible. Currently there is no function to calculate the distance between two points and no support for prediction functionality. We intend in this assignment to extend the platform by providing functions that are able to predict the location of the user.
1.2 Objectives

The objective of this project is to identify and compare prediction methods that can be used in context-aware applications. By knowing future positions of a user, context-aware applications adapt itself to the upcoming situation. For example, when the user is about to move out of the currently loaded map, we can load a new map. Another possible application is when a user wants to meet a friend when he gets close to this person, it is possible that the user has to leave the highway some time before reaching his friend, because his friend is on a road next to the highway. Predicting their locations, the system will determine that they will come close to each other, and instruct the user to leave the highway in the right time. We compare different prediction methods and determine which methods work best in which situation.

1.3 Approach

Before finding ways to perform prediction, we start with studying the different tools we have to use for creating the prototype. These tools include a programming environment and Web Services. For the programming environment we use Borland JBuilder. Since the WASP platform is a Web Service, we have to learn how to include Web Services in Java.

We have studied the different methods that can be used for prediction. We got these methods from searching in literature that handles prediction already. We have implemented prototypes based on these methods to find the best method for prediction.

1.4 Structure

- Chapter 2 gives background information about technologies we use in the rest of this report. These are web services, mobile environment, context-awareness and personalization.

- Chapter 3 gives an overview of the WASP platform, which is a tool we have used extensively in our project.

- Chapter 4 discusses different methods that can be used for prediction. It describes the methods in detail, and mentions their benefits and drawbacks. At the end we decide upon the methods that could be used for a prototype for prediction.

- Chapter 5 discusses our implementations of the prediction methods. It discusses how the theory has been translated to implementations in the WASP platform.

- In chapter 6 we finally draw up conclusions on the prediction methods.
2 Background

In this chapter we describe the different technologies and concepts we have studied to perform this assignment. We start with web services, followed by mobile environment, context awareness and personalization.

2.1 Web Services

Component based development and web are combined into Web Services [haa05]. From both the best parts are taken. Web Services are services that are offered on the web to clients. Figure 2.1 shows the relationship between the different parts. The UDDI registry is used to register offered web services. This is done by services in WSDL. A client can query the UDDI registry for a certain service. It will get back the WSDL document of the service, in which it is mentioned which services are offered, and how to invoke them. With this WSDL document the client is able to use the services.

DCOM, CORBA and RMI provide generally speaking same kind of service, but this is much less flexible than web services. This makes it easy to reuse
components. Because of the use of different existing internet protocols and Simple Object Access Protocol (SOAP) messages [gud03] using eXtensible Markup Language (XML) services are very flexible. Web services are used to distribute programs. Instead of running the whole program on your local computer, only a small part runs on your own computer, and the rest is running at different places on the internet. This makes it possible to have thin clients with still powerful programs.

2.2 Mobile environment

Because following a fixed device is not useful, we are all the time working with mobile devices, which are often small in size, but more important, limited in processor power, storage and bandwidth. Bandwidth might be, next to being limited, also expensive. This has to be taken into account when designing mobile applications. Costs and response time should be calculated.

An example of a mobile environment is the WASP platform. With your mobile device you can walk through the city, and follow on your screen where you are, and where there are interesting points. Since a mobile device might not have the space to save all map data, it can be loaded on the fly from a remote server. Via this application you can also request extra data, like hotels with prices and photos of buildings around you.

2.3 Context awareness

Context awareness means being aware of the context (for example, location, temperature, light intensity, humidity, air pressure). Sensors measure context. This can be different sensors, or existing tools used to detect context (for example, telephone receivers). By using the context of a user, the system can personalize the system for the user. For example, when a user is in a car, and drives into a city, the system can inform the user about parking possibilities. A user can also be informed about the movies playing in the cinema he is passing.

Although context is a broad range of data, in the WASP platform, we only use the location, direction and speed information. The WASP platform does also contain altitude information, but this is not relevant for our project.

2.4 Personalization

Personalization takes care of making the system aware of options of a user. This includes characteristics, preferences and interests. Characteristics are completely objective, such as date of birth, sex, name and address. This information is provided by the user, and cannot be taken from observing behaviour. Preferences are subjective settings made by the user. This can be sort order, colours and layout. Interests describe how much a user is interested in certain things. This can be both set explicitly by the user, as well be derived from his behaviour. For example, a user can set that he is interested in horror movies, but the system can also notice the user goes often to opera. From both the system can predict the interest in certain things.
As the name already says, personalization is personal. This means it has to be stored separate for every person. The data is also private, and thus should be protected from unauthorized access.
This chapter discusses the WASP platform. It gives an overview of all the functions it provides. The description of these functions facilitates the understanding of the following chapters of this report. The platform services are divided into three major functional modules: basic functionality, authentication and 3rd party-services. We describe these modules in detail in this chapter.

3.1 Overview

Figure 3.1: Overview of the WASP platform

The WASP platform consists of several web services, each performing a specific function. Figure 3.1 shows the connections between these parts. A set of 3rd-party web services is used by the platform. All these services except for the Parlay X
services (the services that provide location information) are directly accessible for the applications. These services can also be used without the platform. These are the Map server (to provide the maps in the applications) and the Point of Interest (POI) search service [ebb04].

All web services in the platform are directly accessible for WASP applications, except for the Parlay X services of the 3G Network providers. The Parlay X services are offered through the WASP platform, which hides the complexity of using Parlay X services. Within WASP, 3G Networks provide connectivity and location information.

![Figure 3.2: An overview of the WASP platform](image)

The WASP platform components are depicted in figure 3.2. WASP applications communicate with the WASP platform by means of web services. They are able to access all components, except the 3G Network providers. All services inside the platform, as well as the 3rd party services can directly be approached by the applications. The interactions between platform and 3G Network providers go beyond the scope of this report.

The 3rd party services are reachable through the platform. The platform takes care of the connection to this 3rd party services. This 3rd party services are rather important for the platform’s functioning.

### 3.2 Personalization

In order to support personalization all kinds of data about the user are saved by the platform. This data can be split in three categories:

- **Characteristics**: represent objective aspects of a user. Examples are sex, birthday, name, address, etc.
• Preferences: represent personal subjective settings defining how the user wants the application to behave. Examples are background colour, number of search results to return, etc.

• Interests: describe how much a user is interested in something. This can be explicitly set by the user, but also derived from his/her behaviour.

The personalization data is stored in the profile of the user.

3.2.1 UserProfiler

This service provides methods to set and request the data from the different personalization categories, and also to set and request the weight and certainty of an interest. A user can fill in his profile, which can be used for prediction of interest.

For all operations a password needs to be provided. The password is set when a new user profile is created.

3.2.2 PredictInterests

One can request the platform to predict an interest of a certain user. This prediction is based on preferences set by a user, interests he has shown in the past and explicit feedback about a visited Point of Interest (POI). In this way the platform can check if a user is interested in a POI when he is passing by, and see if it has to send out a notice.

When requesting the prediction of an interest, the following data is returned:

• Adjusted prediction: this is based on the normal prediction value for this user, but is lowered after a recent visit to a similar POI. For example, if a user just leaves a restaurant, it is not useful to inform him about any restaurant he is passing afterwards;

• The reason why the adjusted prediction differs from the normal prediction;

• The certainty about a prediction;

• An explanation why the system thinks a prediction is correct;

• Has the user given explicit feedback about this POI?

• The normal prediction value, which does not change because of recent visits to similar POI.

3.3 Monitoring

Administrators can use the monitoring function to monitor the platform. The monitoring function is based on the visualization of events. The WASP platform generates an event for every service that is invoked, every service that sends a response, another service that is invoked from a service and when receiving back a response from this other service. Events are not stored, but only forwarded to
people and applications subscribed to these events. To log the behaviour of the platform, you’ll have to attach your own logger that puts all events it receives in a file.

3.4 Registry

The registry is used by applications to register their services to the platform. This allows users to find services in the platform. The interface exchanges Resource Description Framework (RDF) messages [mil04] [mcb02]. Users can search in the registry with the RDF Data Query Language (RDQL) [sea03] [sea02] to find the service they need. This means that the interested user does not have to know the name of a certain service, but can search for a certain service type.

3.5 User Context manager

The User Context Manager manages the context information of the user. Currently, support to context information is limited to location information.

The user context manager consists of two parts: services and interfaces. The services are functions that can be called by applications. The interface is implemented by the applications, and is called by the platform in case a certain event happens.

3.5.1 UserLocationHistoryService

The platform saves the past locations of the users. It is possible to request this information from the platform by providing the minimal time interval between the points in the history. Furthermore, one can select a certain area and a time frame from which one wants to get the user position in the past.

3.5.2 UserServicesService

This service allows its users to query the technology through which a user can receive messages, and send messages to this user. Currently SMS, MMS and phone calls are supported.

3.5.3 UserLocationSimulatorService

This service allows its users to request the location information for a user, and also set up a simulated route for the user. The following information can be given by this service:

- The location in latitude and longitude;
- The accuracy of this location;
- The date and time this location was obtained;
- The user’s speed in km/h;
- The user’s direction in degrees;
- The source from which the location information was obtained;
- The user’s altitude;
- The accuracy of the altitude;
- If the GPS receiver of the user has determined a location;
- The number of GPS satellites used to determine the location;
- The user ID;
- The route of the user in latitude/longitude combinations.

This service allows one to request the data of a user, get a list of users, list users whose data have been changed since the last query, remove users and set a simulated route.

**3.5.4 UserLocationService**

This service provides the current location of a user, can request new buddies to be added and removed, and set triggers on moves of buddies and the user him/herself. Buddies are people known to the user, who have granted access to user’s location information.

**3.5.5 UserLocationUpdateService**

This service allows its users to set a new location for a user.

**3.5.6 BuddyListenerInterface**

This callback interface is called when the buddy list of a user changes. A list with the new buddies is given to the function.

**3.5.7 userTriggerNotificationInterface**

This interface notifies about a previously set trigger being fired. The UserLocationService sets this trigger.

**3.6 Picture store**

The picture store is used to store pictures made by the users of the WASP platform. The picture store stores the address and location where the picture was taken, by whom it was taken, a description of what is shown on the picture and the date the picture was taken. A picture can be publicly available or only available for certain users. In this way users can share pictures of their trips with friends, or one can preview the locations he will visit.

This service component allows one to search pictures in a certain area and request the size of a certain picture.
3.7 FWA server

The Federated Web Authentication (FWA) server takes care of authentication requests by applications built on top of the WASP platform. The application has to take care of authenticating the user. The FWA server only checks if the user account is still valid and if so, returns the interface through which the service can retrieve location information for this user.

3.8 Map server

The map server is a 3rd party service, and is provided by Microsoft MapPoint [mmp05]. Maps can be provided on both latitude/longitude coordinates and the Dutch ‘Rijksdriehoek’ (RD) grid. A zoom factor has to be given to the service. The map server returns the height and width of the part shown in kilometres. The image returned is in base64 encoding. Two kinds of maps can be requested: aerial pictures and cadastral data from the year 1832, both currently only for the municipality of Enschede.

3.9 POI Search Service

The Point Of Interest search service allows one to search for POIs in different databases. Currently POIs are only described in RDF. To be able to search for a POI in a certain area, the location has to be registered first in the POI Search Service. You can search in the following databases:

- RealEstateSearchService: this database contains houses that are on sale. A price is also included in the results;
- GoudenGidsSearchService: this database contains the yellow pages, mostly about shops and shopping malls;
- RouteSearchService: this database contains POIs that are situated close to predefined routes;
- VVVSearchService: this database contains POI information provided by the Tourist Information Office, which consists mostly of restaurants and all forms of accommodation (for example, hotels, bed & breakfast). Results are mostly very detailed. For example, the number of rooms in a hotel, average price of a meal and information about house rules;
- MappointSearchService: this database contains POIs taken from the Mappoint.net web service [mmp05]. Results can be public transport, parking places, restaurants, governmental buildings, etc.
- MonumentSearchService: this database contains historical POIs such as architecturally important buildings and monuments.
4 Location prediction

We have identified several aspects in which the WASP platform could be improved, ranging from fixing mistakes in the documentation to adding totally new functionality.

The platform lacks a function to calculate the distance between two points. This is rather complicated to implement, since locations in the platform are given as latitude-longitude, which are locations on a bulb. There are formulas available on the Internet in order to calculate the distance, but the problem is that they need the radius of the earth, which is not constant for all locations on earth. An improvement could include compensation for this so that programmers do not have to take care of this separately when expanding the platform to a whole continent.

The platform is able to keep track of the current and past positions of users, but the platform is not capable of predicting their future position. If a user passes close to a Point of Interest, but has to change his route to really pass by, it might be useful if the platform could already warn the user when there is still the possibility to change the route. For this we need prediction, functionality which is described in this chapter.

In this chapter we discuss different techniques for performing location prediction, including their benefits and drawbacks.

4.1 Same speed

When people move, they normally go from one point to another, via the shortest route. This means that on average they move in a straight line, and when possible with a constant speed. To do the prediction in this case, only the current direction and speed have to be known. Due to the small amount of data needed to predict location, a system based on this approach can predict locations for many users at the same time.

Prediction is done in the following way: the current speed and direction are taken to calculate the distance a person is expected to move in the prediction time (for
example, 5 seconds from now). The predicted location is then the calculated distance from the current position in the current direction of the user.

![Figure 4.1: prediction with Same Speed method](image)

Figure 4.1 shows that as long as a person goes ahead, prediction is accurate. But, as soon as the person starts making a curve, prediction is inaccurate, until the person moves again in a straight line.

To calculate the position, we need to know the current and past position: $x_n$, $y_n$, $x_{n-1}$, and $y_{n-1}$. We consider only the prediction of $\Delta t$ in future, where $\Delta t = t_n - t_{n-1}$.

With this we can calculate the predicted location: $x_{n+1}$ and $y_{n+1}$, using the following formulas:

$$\phi = \arctan\left(\frac{y_n - y_{n-1}}{x_n - x_{n-1}}\right)$$

$$s = \sqrt{(x_n + x_{n-1})^2 + (y_n + y_{n-1})^2}$$

$$x_{n+1} = x_n + s \cdot \cos(\phi)$$

$$y_{n+1} = y_n + s \cdot \sin(\phi)$$

Figure 4.2 shows all the variables that we use in the formula above.
Figure 4.2: Prediction with Same Speed method –(detailed)

The benefits of this method are that there is no need to store huge data sets, since you only need the current point, one previous point and the interval between the measurements of this points. Another benefit is that computation of the predicted point is easy.

The drawbacks of this method are that on smaller roads there are more curves, which makes prediction on these kinds of roads unreliable. Another drawback is that the predicted location could be in the water, or on a house. This is not likely when we are following a user.

### 4.2 Same curve

The *Same Curve* method is comparable with the *Same Speed* method, in the sense that we consider constant speed in both methods. In the *Same Curve*, though, we also try to calculate the direction in which the user will move. We add the difference between the current and previous direction to the current direction, so that the prediction also follows the curve the person is making. For this to work we need to know the current speed, current direction, and the previous direction. It should be no problem to keep this information for a big amount of users.

Prediction is done in the following way: similarly to the *Same Curve* method, the current speed and direction are taken to calculate the distance a person is expected to move in the prediction time (for example, 5 seconds from now). In order to predict the future direction, we take the current direction plus the difference between the current direction and the previous direction.
Figure 4.3 shows that as long as a person goes straight, prediction is accurate. But, as soon as the person starts making a curve, prediction is inaccurate, only on the first prediction try, and in the curve prediction is again accurate. When the person stops making the curve, the first prediction is inaccurate again.

To calculate the position, we need to know the current and two past position: $x_n, y_n, x_{n-1}, y_{n-1}$. We consider only the prediction of $\Delta t$ in future, where $\Delta t = t_n - t_{n-1}$. With this we can calculate the predicted location: $x_{n+1}$ and $y_{n+1}$.

$$\varphi_n = \arctan \left( \frac{y_n - y_{n-1}}{x_n - x_{n-1}} \right)$$

$$\varphi_{n-1} = \arctan \left( \frac{y_{n-1} - y_{n-2}}{x_{n-1} - x_{n-2}} \right)$$

$$s = \sqrt{(x_n - x_{n-1})^2 + (y_n - y_{n-1})^2}$$

$$x_{n+1} = x_n + s \cdot \cos(2\varphi_n - \varphi_{n-1})$$

$$y_{n+1} = y_n + s \cdot \sin(2\varphi_n - \varphi_{n-1})$$

Figure 4.4 shows all the variables that we use in the formula above.
The benefits of this method are that this method works fine with slow curves. In this case, this method performs better than the *Same Speed* method. Another benefit is that we only have to keep a small amount of data per user, that is, current position, and two past positions with the interval.

The drawback of this method is that this method does not work properly with sudden strong curves.

### 4.3 Neighbouring cells

[bpj97] describes a mobile network with cells where movements of persons is recorded as a function of time to predict their future location. The receivers for the mobile phones define the cells. This gives certain likelihood that at a certain moment a person is in a cell. To trace the person, the cells are polled in their order of likelihood.

[bpj97] assumes that people make the same movements at the same time of the day. Due to this reason prediction should be accurate, although less precise, because the predicted position will be in a cell, which is rather big. To reduce the amount of saved data, we can record movements of people in general, but this makes precision less accurate, because people do not always move in the same direction.

To implement this method we have to create a database that keeps track of the cell a person is at a certain moment. After some time we can start predicting for this person based upon his past movements. When a person is in a certain cell, we look up to which cell he moves most often at that time of day, and use that next cell as the predicted location. Due to the rather small amount of choices, chances are big that we predict the next cell accurate. On the other hand, the prediction is a rather big area, so this method is not really good to find a person in a crowded place.
The benefits of this method are that you don’t need an external device like GPS to locate a person. Just having your mobile phone switched on is enough. The location is taken from the mobile receiver.

The drawbacks of this method are the low precision. Because of the use of cellular phone cells, the location is in a rather big area. To store all the movement patterns, a lot of storing space is needed. Another drawback is the need to first record movement patterns for a certain time before prediction can be done.

4.4 Past movements

This method is related to the Neighbouring Cells method. But instead of defining the cells by the GSM receivers, we make a grid with smaller cells, where the size is dependant on the precision needs. In this case, we need an extra device to trace a person, since mobile phone cells are currently not precise enough. With this we get smaller areas in which a person can be found. This greatly improves the precision, but also means a huge amount of data to be stored for all the different persons.

Implementation can be done as described above, with the exception that for location tracking an external device like a GPS has to be used.

The benefit of this method is that it has better precision than the neighbouring cells method.

The drawback of this method is that you need to store very huge sets of patterns. This is even much more than in the neighbouring cells method due to the smaller grid size.
4.5 Following roads

[wol03] describes how the use of maps with roads can greatly improve prediction by assuming a person moves on the same road with the current speed. GPS is used to locate a user. As soon as the end of the road has been reached, prediction stops until the next update of the real location is made. If roads also include their average speed, the current speed of the person does not have to be known to the system, but the average road speed can be taken into account.

Location snapping is needed to be able to put a person on a road, since GPS might not indicate the location exactly on a road. Snapping means finding the road the person is most likely on, based on the position given by the GPS. The direction on the road can be found out by looking at the previous location.

This method works precisely, since most persons do not go off the road, except when they walk. In this case, people might move, for example, inside buildings and alleys.

To implement this model we need to have a map with the roads. Because we only need the current and previous location (for the direction) to do the prediction, the system can support a lot of people moving around (no need to store big amounts of data).

Figure 4.6: Prediction with the Following Roads method

Figure 4.6 shows a possible prediction by the following roads method. As long as there are no side roads, prediction is accurate. As soon as a crossing is reached, and thus there are several possible directions a person can go, prediction will be the crossing itself. Only after the person moves into one of the streets, prediction works again.

In figure 4.6 you can see that when the first crossing (left) is reached, the predicted location is on the crossing, which is a smaller distance that is predicted compared to previous predictions. The next prediction is performed when the person already moved past the crossing, so this time the distance that is predicted seems to be bigger. From these predictions you can see that the person is moving from left to right.

The benefits of this method are that the predicted locations are always on a road, which is the most likely position a person will move over. When there are only a few crossings, this method can predict far in future.
The drawbacks of this method are that the prediction stops as soon as a crossing has been reached. Within cities, where there are many crossings, this method will only predict small distances.

### 4.6 Agenda

Many people keep a digital agenda nowadays. By looking in their agenda, and interpreting their appointments, prediction of the movements of this users can be attained, since one can use a route planner to find the way to a location, and predict far in future with fair precision.

To implement this model we need access to the user’s agenda. A problem here might be that people do not write clear destinations, but, for example, only the name of the persons they meet. This problem can be solved by self-learning, whenever another person is visited in a location not clearly determined, the location the user visits is saved.

The benefit of this method is that prediction is easy when you know the destination of the person, and can be done with high accuracy.

The drawback of this method is that it can be hard to interpret appointments in an agenda.

### 4.7 Combinations

By combining the above mentioned methods we can do even more precise prediction than by using of the methods alone. Especially the Agenda method looks promising. If we can combine it with the Neighbouring cells method, for example, and see the roads as cells, we can predict after the end of the road with certain likelihood. If we combine this method with the same speed method we can assume that at the end of the road the person will move as much in the same direction as possible, since they normally follow the shortest way.

Another combination would be to consider that a person moves over the fastest roads, like highways, to a certain destination. As long as the person is not on a highway we can consider he moves in the direction of a highway entrance. The problem is the moment the person moves off the highway, in which the method does not work. We can also record final destinations of the person. This is still a rather limited number of points compared to other point recording methods. By looking at the direction the user is moving and comparing that with routes created by road planners to known destinations of the person, we may be able to predict the whole route of the person.

### 4.8 Conclusion

While the same speed and same curve method are easy to implement and require only few resources on the predicting host, the neighbouring cells and past movements methods need a lot of storage for all the users of the system, which is not realistic for huge usage. The following roads method needs to be aware of the roads in the area where prediction is done, but is much more precise for vehicles, since they normally stay on the roads.
To evaluate the different methods, we have built a prototype of Same Speed, Same Curve and Following Roads prediction. These methods look the most promising, and are suitable for the WASP platform.
5 Implementation

In order to compare the different methods, we implemented prototypes of some of the methods discussed in chapter 4, namely Same Speed, Same Curve and Following Roads. The methods Neighbouring cells and Past movements involve the storage of certain amount of complex data, which goes beyond the scope of this report. Furthermore, we also believe that these methods are less realistic in terms of available technology and costs.

In this chapter we describe the implementation of the methods Same Speed, Same Curve and Following Roads.

5.1 Overview

For testing purpose we have implemented the PredictGUI class. This class contains the user interface for the prediction classes. To draw results on the screen, PredictGUI uses the class FollowUser, which is able to draw locations and movements on the screen. This class uses Line to store past movements, to be able to redraw the screen when it was overwritten.

In figure 5.1 you can find an overview of the different classes, including their relationships.

PredictGUI is the user interface, which includes a map and some buttons. From here the other functions are called. FollowUser takes care of drawing moving positions on the map in PredictGUI. The class Line is used to store lines that are drawn by FollowUser.

The class Predict is the super class of the different prediction classes. This class takes care of the links to the different WASP platform functions. All prediction classes are an extension of Thread. This means they can run in parallel.
Figure 5.1: class relationships

5.2 General code

The earth has a shape similar to a bulb. For calculating the distance between two points on a bulb, there exist formulas, but they all need the radius of the bulb. Since the earth is a flattened bulb, we have to adjust the radius \( r \) in the formula to the location on earth:

\[
\begin{align*}
\text{dis} \tan ce &= \arccos \left( \cos(a1) \cos(b1) \cos(a2) \cos(b2) + \right. \\
&+ \left. \sin(a1) \sin(b1) \sin(a2) \sin(b2) + \right) / 360 * 2\pi \\
&+ \arcsin \left( \sin(a1) \sin(b1) \sin(a2) \sin(b2) + \right. \\
&+ \left. \cos(a1) \cos(b1) \cos(a2) \cos(b2) \right)
\end{align*}
\]

In this formula we calculate the distance between point \((a1, b1)\) and \((a2, b2)\), where all values are the latitude and longitude converted from degrees to radians.
After comparing distance calculations of the application with distance calculations in online route planners [mic04], we found out that with a radius $r$ of $6378,14$ [cog03] we have to use a correction factor of $1,3$ for distance calculations in The Netherlands. We calculated this by setting up a route in the WASP platform and calculating the distance of it via the distance function above. After this, the same route has been put in an online route planner [mic04]. When comparing these values, we found the correction factor of 1,3. In the rest of this chapter, $r$ is considered to be $6378,14*1,3=8291,582$

5.3 Same speed

This method is based on the principle that a person moves with the same speed in a straight line. Due to this consideration, the method to calculate the prediction is simple.

5.3.1 Code

For this method, we need to know the current location, speed and direction of the user. This information is provided by the WASP platform, so the formula mentioned in section 4.1.1 can be simplified.

From the WASP platform we use the UserLocationSimulator from which we can request the data of the user related to location. This is stored in the UserData class. The UserData class provides all location information of the user. We only use speed, direction and location. The location is stored in the LatLong class. From the LatLong class we can request the latitude and longitude of the current position.

The problem we have to face is the way in which locations are stored: latitude/longitude combinations. These cannot be easily used for calculating the predicted location, because they are locations on a bulb. For this we have to put the formula in section 5.2 in reverse order.

\[
\begin{align*}
    d &= \frac{\text{WASP.Speed}()}{3600} \times \text{time} \\
    \varphi &= \frac{\text{WASP.Direction}()}{360} \times 2\pi \\
    x_{n+1} &= x_n + \sin(\varphi) \times \frac{d}{r} \\
    y_{n+1} &= y_n - \cos(\varphi) \times \arccos \left( \frac{\cos \left( \frac{d}{r} \right) - \sin^2(x_n)}{\cos^2(x_n)} \right)
\end{align*}
\]

$d$ is the distance we move in the time we predict (for example, 5 seconds). Since the WASP platform returns the speed in km/h, it has to be divided by 3600 to make it seconds (time is in seconds as well). For $\varphi$ we have to translate the
direction from degrees to radians, because the \( \sin \) and \( \cos \) functions in the Java Math library use radians.

5.3.2 Results

![Figure 5.2: prediction results of Same speed](image)

Figure 5.2 shows prediction results of a test case. As you can see, the prediction is correct as long as straight movements are made, but as soon as curves start, prediction is inaccurate. Due to this we cannot predict far in future, since most roads have curves. Only in wide areas, like deserts, this method might work further in future.

5.3.3 Possible improvements

The predicted time is currently fixed. It might be interesting to use feedback to see how good prediction is. When prediction is good, the time prediction is done in future (\( \Delta T \)) can be increased, and when prediction is bad, the time (\( \Delta T \)) can be decreased.

In case there is no possibility for feedback, one can also think about research for the relation between speed and going straight. If there is a relation, the time we predict in future can be related to the current speed. In this way we can predict as far as possible, with still keeping a certain level of correctness.

5.4 Same curve

This method is based on the principle that people move at a constant speed, and follow the curves they are in. We need a small history of movements, but this is limited to one point close in the past (for example, 3 seconds ago).

5.4.1 Code

For this method, we need to know the current location, speed and direction, and the direction in the past. Since the WASP platform knows already about speed and direction, we don’t have to calculate them ourselves with the formula in section 4.1.2.

The current location, speed and direction can be taken from the WASP platform as described in section 5.3.1, the direction in the past can be taken from the
UserLocationHistory class in the WASP platform. This history is stored in the LocationInfo class. This one is comparable to UserData, with the difference that it also stores the time of the data. One should keep in mind that the latest point in time stored in the history is actually the current point.

We cannot request the latest point in history directly from the WASP platform. We can only select a time range from which we want the points. From this we’ll have to select the points we need.

The direction we follow is calculated with the formula in section 4.1.2, the rest of the calculations are the same as in section 5.3.1.

5.4.2 Results

The road being followed in figure 5.3 is the same as in figure 5.2. It is a real road, namely the Varvik Singel and Hogeland Singel between Gronausestraat and Zuiderval in Enschede. As one can see when comparing this result to figure 5.2, one can see that this prediction is actually worse than the same speed method. This is rather surprising if you look at the slow curves in the road. Corrections in prediction are going too far, resulting in spikes.

When looking at figure 5.4, you get even more strange results. Instead of nicely following the half circle, prediction is jumping all around, in strange figures. Especially after a turn prediction is inaccurate.
5.4.3 Possible improvements

In general the method does not seem to be working in a good way. When a person makes a sudden turn of 90°, prediction in the current method will think the person will go on with this curve, meaning the person turns back according to prediction. Since this is unlikely, a way should be found to prevent this kind of predictions, and only use feedback in a small curve.

Furthermore, we should have a look at the strange spikes in the prediction. With the current results it looks more like impossible to use. We have to check with the correctness of the direction given by the WASP platform if inaccurate values are returned.

5.5 Following roads

For the following roads method, we need to have roads. We use virtual roads for the testing. In real life, real roads from map services should be used.

5.5.1 Code

This prediction method makes use of several other classes. You can see the overview in figure 5.5.
A road segment is defined as a straight line between two points. Although these points are called start and end, a road segment does not have a direction in our implementation. We also do not define roads, since roads do not make sense for us. All road segments are put together in RoadMesh. RoadDistance is an extension of RoadSegment. It defines some extra variables to only have to do calculations once.

One of the problems we have to solve is finding the road the user is really on. With the other prediction methods the system does not know about roads, but here we have to snap the user onto a road. In the solution we implemented, we take the road closest to the point the user is said to be (by, for example, a GPS). In section 5.2 we already defined a formula to calculate the distance between two points, so we only have to find the point on the road that is closest to the point we have to compare to:

\[
 t = \frac{\left( p_x - r_{1x} \right) \left( r_{2x} - r_{1x} \right) + \left( p_y - r_{1y} \right) \left( r_{2y} - r_{1y} \right)}{\left( r_{2x} - r_{1x} \right)^2 + \left( r_{2y} - r_{1y} \right)^2}
\]

In this formula, \( p \) is the point we compare the road segment to, and \( r1 \) and \( r2 \) are the begin and end of the road segment. \( t \) now defines the point on the road segment that is closest to point \( p \). In case \( t \) is smaller or equal to 0, it means the begin of the road is closest to point \( p \). In case \( t \) is bigger or equal to 1, it means the end of the road is closest to point \( p \). When \( t \) is between 0 and 1, we can calculate the point in the following way:

\[
\begin{align*}
p_{2x} &= r_{1x} + t \left( r_{2x} - r_{1x} \right) \\
p_{2y} &= r_{1y} + t \left( r_{2y} - r_{1y} \right)
\end{align*}
\]

\( p2 \) defines here the point on the road segment that is closest to point \( p \). One should keep in mind that \( t \) is calculated for the line defined by point \( r1 \) and \( r2 \). A value smaller than 0 or bigger than 1 means that the point on the line that is closest to point \( p \) is not between \( r1 \) and \( r2 \). In this case we use the begin or end point of the
road segment as closest point. With the formula in section 5.2 we can now calculate the distance between a road (point $p_2$) and a point $p$.

When doing the prediction for a user, we can define two cases: the whole distance we predict (for example 50 meters) is still possible on the current road segment and that the whole distance we predict does not fit on the current road segment. Prediction in the first case is easy, but the second case we have to check also what happens at the end. If we search for road segments that have a begin or end at the same point as the end of the road segment the user is on, we know what to do based on the number of road segments that is returned. In case only one road is returned, it means it is a dead road end, which means prediction will stop there. In case two road segments are returned, it means that there is no choice, and most probably the two returned road segments are part of one road. In case more than two road segments are returned, it means it is a crossing, and prediction will stop at this crossing, since we do not make choices at crossings in this prediction method.

5.5.2 Results

![Diagram of prediction results](image)

*Figure 5.6: prediction results of same road*

First of all, we can see that the prediction is always exactly on a road, even if the external device used to see our location is telling we are a bit off the road. In the top right corner, prediction went further than the curve. Since prediction is only points, which are connected by lines only for clearer overview, prediction seems to shortcut the curve.

On the square just below this curve that is shortcut, there is some other strange effect. This happens because the closest road to the measured point at a certain moment is not the road we are moving on. When this is corrected, you get the lines through the middle of this square.

5.5.3 Possible improvements

It cannot be seen clearly in this static picture, but as soon as a crossing with several possibilities is reached, prediction waits at that crossing until the person passes by and moves into one of the streets. As long as there are few crossings...
only, this is not a problem, but in a city with many crossings, prediction will only be done on short distances. A combination with the other methods could be used to overcome this problem.

Another problem is the snapping of a person to the closest road. As can be seen in the figure 5.6, this is not always correct. With smarter snapping and also looking at directions a person is moving and previous points visited, this problem could be solved.
6 Conclusions and future work

This chapter draws our conclusions based on the research questions as formulated in the first chapter: identify and compare prediction methods that can be used in context-aware applications. It will also mention points for future work.

6.1 Conclusions

In this report we have proposed 6 different methods for predicting locations, namely same speed, same curve, neighbouring cells, past movements, following roads and agenda. From these 6 methods, we have made an implementation of 3 of these methods, namely same speed, same curve and following roads.

From the tests with the implementations we made we can conclude the following that same speed works rather good for predicting the future location of a user. Same curve was expected to work better than same speed, but in practice it is much worse. We could improve this by looking at the prediction being realistic. This means that after a sudden 90 degrees turn, we do not predict another 90 degrees, but just keep it at this 90 degrees. On the other hand, when there is only a turn of 5 degrees, we predict that this turn will be followed for a longer time.

Following roads is close to perfect, but stops at crossings because we did not choose a method to go on from there. Because following roads works perfect as long as there are no crossings, this method seems to be the best on roads without crossings. To also catch the crossings, a combination with the same speed can be made, since that is the other best solution from the tests we performed.

6.2 Future work

Because the results we gathered in this report are not perfect yet, further investigation should be performed on the different techniques, and combinations of them. Especially solutions that use following roads as a starting point seem to be interesting for future investigation. With a combination with the following roads method the past movements method can also be investigated. Although we
will still need to store a huge amount of data here, it is already much less because we can limit the stored points to roads. Also the *agenda* method could help deciding in the direction when reaching a crossing.
References

[bej97] Jens Biesterfeld, Elyes Ennigrou and Klaus Jobmann; Neural Networks for Location Prediction in Mobile Networks
Available at: http://www.ant.uni-hannover.de/Forschung/Public/Kn/1997/BEJ1997.pdf

[cog03] Simon Coggins; The Virtual Solar System Datastore
Available at: http://www.solarsystem.org.uk/datatext.html#Earth

[ebb04] Peter Ebben and Martin Wibbels; WASP Platform documentation - Documentation of the services of the WASP platform
Available at: https://doc.telin.nl/dscegi/ds.py/View/Collection-5650

[gud03] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau and Henrik Frystyk Nielsen; SOAP Specifications
Available at: http://www.w3.org/TR/soap12-part1/

[haa05] Hugo Haas; Web Services
Available at: http://www.w3.org/2002/ws/

[mcb02] Brian McBride; An Introduction to RDF and the Jena RDF API

[mic04] Via Michelin
Available at: http://www.viamichelin.com/

[mil04] Eric Miller, Ralph Swick and Dan Brickley; Resource Description Framework (RDF)
Available at: http://www.w3.org/RDF/

[mmp05] Microsoft MapPoint
Available at: http://www.microsoft.com/mappoint/net/

[sea02] Andy Seaborne; A Programmer’s Introduction to RDQL
Available at: http://www.hpl.hp.com/semweb/doc/tutorial/RDQL/index.html

[sea03] Andy Seaborne; RDQL - A Query Language for RDF
Available at: http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/

[weg04] Dr. Maarten Wegdam; AWARENESS project
Available at: http://www.freeband.nl/project.cfm?language=en&id=494

[wol03] Ouri Wolfson and Huabei Yin; Accuracy and Resource Consumption in Tracking and Location Prediction
Available at: http://www.cs.uic.edu/~wolfson/mobile_ps/sstd03.pdf
FollowUser.javapackage predict;

import nl.freeband.wasp.usercontextmanager.UserLocation;
import nl.freeband.wasp.usercontextmanager.UserLocationServiceLocator;
import javax.swing.JLabel;
import java.awt."
import nl.freeband.wasp.common.LatLong;
import javax.swing.JTextArea;

/* Class to draw locations on a map. Input can be either a variable containing
 * this position, or a user in the WASP platform.
 */

class FollowUser extends Thread {
    UserLocation ul;
    JLabel map;  // The map on which we draw the user
    LatLong topleft;  // Topleft position on map (latitude/longitude)
    LatLong bottomright;  // Bottomright position on map (latitude/longitude)
    JTextArea logger;  // Logger window
    String user;  // User for which we predict
    LatLong predict;  // The variable that contains the predicted user position
    Color colour;  // The colour in which we draw the prediction
    boolean abort = false;
    int delay = 2; // in secs

    public FollowUser(JLabel map, LatLong topleft, LatLong bottomright, JTextArea logger, String
        user, Color colour) {
        /* Constructor
         * + input:
         *   - map: the jlabel on which the prediction has to be drawn
         *   - topleft: the topleft position that is shown on map in latitude/longitude
         *   - bottomright: the bottomright position that is shown on map in latitude/longitude
         *   - logger: logger window
         *   - user: the user for who we have to do prediction
         *   - colour: the colour which has to be used to draw on the map
         * + output:
         *   - [none]
         * The constructor sets the global variables, and initiilizes the web services
         */
        this.map = map;
        this.topleft = topleft;
        this.bottomright = bottomright;
        this.logger = logger;
        this.user = user;
        this.predict = null;
        this.colour = colour;
        this.abort = false;
        try {
            ul = new UserLocationServiceLocator().getUserLocationService();
        }
        catch( Exception ex ) {
            ex.printStackTrace();
        }
    }
}

public FollowUser(JLabel map, LatLong topleft, LatLong bottomright, JTextArea logger, String
        user, Color colour) {
        /* Constructor
         * + input:
         *   - map: the jlabel on which the prediction has to be drawn
         *   - topleft: the topleft position that is shown on map in latitude/longitude
         *   - bottomright: the bottomright position that is shown on map in latitude/longitude
         *   - logger: logger window
         *   - predict: the variable that contains the predicted location
         *   - colour: the colour which has to be used to draw on the map
         * + output:
         *   - [none]
         * The constructor sets the global variables, and initiilizes the web services
         */
    }
/**
 * this.map = map;
 * this.topleft = topleft;
 * this.bottomright = bottomright;
 * this.logger = logger;
 * this.user = null;
 * this.predict = predict;
 * this.colour = colour;
 * this.abort = false;
 * try {
 * ul = new UserLocationServiceLocator().getUserLocationService();
 * } catch( Exception ex ) {
 * ex.printStackTrace();
 * }
 */

public void doStop() {
    /* Stop when convenient
    * + input:
    *   - [none]
    * + output:
    *   - [none]
    */
    this.abort = true;
}

public void run() {
    LatLong prevLoc = new LatLong();
    LatLong loc;  // Variable used to store the location we have to draw with (can be from user or
    // directly given)
    Line[] hist;  // History of points to redraw screen when needed
    int histItem;  // Last item in hist[] that is filled
    double topleftX=0;
    double topleftY=0;
    double facX=0;  // factor to translate from X (pixels in map) to longitude and back
    double facY=0;  // factor to translate from Y (pixels in map) to latitude and back

    if( topleft==null | bottomright==null | logger==null | map==null | (user==null & predict==null))
    {
        // We have too less data to do prediction
        if( logger!=null )
            logger.append("Unable to start following user, either user or map missing\n");
        abort=true;
    }else {
        prevLoc = null;
        topleftX = topleft.getLongitude();
        topleftY = topleft.getLatitude();
        facX = ( (double) map.getWidth()) /
               (bottomright.getLongitude() - topleft.getLongitude());
        facY = ( (double) map.getHeight()) /
               (bottomright.getLatitude() - topleft.getLatitude());

        // Initialize hist[]
        hist = new Line[30];
        for (int i = 0; i < hist.length; i++)
            hist[i] = new Line();
        histItem = -1;

        try {
            while( !abort ) {
                if( user!=null )
                    // We have to draw current position of a user
                    loc = ul.getUserLocation(user);
                else
                    // We have to draw the position stored in predict
                    loc = predict;

                if( prevLoc!=null )
                {
                    // Only go on if previous location is available. This we need because
                    // we draw a line from this previous point to the current point
                    if( isOnMap(loc) & isOnMap(prevLoc) ) {
                        // If location is outside map, do not draw anything
                        if( histItem+hist.length-1 )
                            // ...
histItem++;
else
histItem=0;

// Save the position for redrawing purpose
hist[histItem].x1 = (int) ( (loc.getLongitude() - topleftX) * facX);
hist[histItem].y1 = (int) ( (loc.getLatitude() - topleftY) * facY);
hist[histItem].x2 = (int) ( (prevLoc.getLongitude() - topleftX) * facX);
hist[histItem].y2 = (int) ( (prevLoc.getLatitude() - topleftY) * facY);

// Draw history plus new point on the map
Graphics graph = map.getGraphics();
graph.setPaintMode();
graph.setColor(colour);
for( int i=0; i<hist.length; i++ ) {
  if( user==null ) {
    // Draw a mark at the predicted locations
    graph.fillOval(hist[i].x1-2, hist[i].y1-2, 5, 5);
  }
  // Connect previous with current point
  graph.drawLine(hist[i].x1, hist[i].y1, hist[i].x2, hist[i].y2);
}
map.update(graph);
else {
  logger.append("FollowUser: user ran out of screen: (" + (int)((loc.getLongitude()-topleftX)*facX) + "; " + (int)((loc.getLatitude()-topleftY)*facY) + ")\n");
}
// Save the previous location
prevLoc.setLatitude(loc.getLatitude());
prevLoc.setLongitude(loc.getLongitude());
// Wait a bit
Thread.sleep((int)this.delay * 1000);
logger.append("FollowUser stopped\n");
catch( Exception ex ) {
  ex.printStackTrace();
}

private boolean isOnMap(LatLong loc) {
  /* Check if a given location is on the map
   * - input:
   *   - loc: the location that have to be checked if it is on the map
   * - output:
   *   - true if loc is on the shown part of the map, false in other cases
   */
  if ( (topleft.getLatitude() >= loc.getLatitude() &
       loc.getLatitude() >= bottomright.getLatitude()) &
       (topleft.getLongitude() <= loc.getLongitude() &
       loc.getLongitude() <= bottomright.getLongitude()))
    return true;
  else
    return false;
}

class Line {
  // Used for storing history points for redrawing the screen
  public int x1=0;
  public int y1=0;
  public int x2=0;
  public int y2=0;
}

public Line() {
  // Empty constructor, variables are initialized by default
}
Predict.java

package predict;

import nl.freeband.wasp.usercontextmanager.UserLocationSimulator;
import nl.freeband.wasp.usercontextmanager.UserLocationSimulatorServiceLocator;
import nl.freeband.wasp.usercontextmanager.UserLocationHistory;
import nl.freeband.wasp.usercontextmanager.UserLocationHistoryServiceLocator;
import nl.freeband.wasp.usercontextmanager.UserData;
import nl.freeband.wasp.common.LatLong;
import java.util.Date;
import java.util.Calendar;

/* Predict is the superclass for different prediction methods. It takes care of *
 * global vars, as well as initializing the web services needed for prediction *
 */

class Predict extends Thread{
    UserLocationSimulator uls; // WASP platform
    UserLocationHistory ulh; // WASP platform
    LatLong predictLoc; // Predicted location of a user
    String user; // The user we predict for
    int time; // The time in future we predict (in seconds)
    boolean abort=false; // variable that is set to true as soon as we want to stop prediction
    int delay = 2; // delay between the different predictions in seconds (how often to predict)
    UserData ud; // Current userdata for a user given by WASP platform
    LatLong userLoc; // Current user location as given by WASP platform
    LatLong nextLoc; // Next user location (predicted)
    double dist; // Distance to move for prediction

    public Predict(String user, LatLong predictLoc, int time) {
        this.user = user;
        this.predictLoc = predictLoc;
        this.time = time;
        try {
            uls = new UserLocationSimulatorServiceLocator().getUserLocationSimulatorService();
            ulh = new UserLocationHistoryServiceLocator().getUserLocationHistoryService();
        } catch( Exception ex ) {
            ex.printStackTrace();
        }
    }

    public void run() {
        /* start doing the prediction. */
    }

    public void doStop() {
        /* stop doing the prediction. The function might still continue for a short *
         * time, until it finds the right moment to quit. */
        this.abort = true;
    }
}
package predict;

import nl.freeband.wasp.common.LatLong;
import nl.freeband.wasp.common.LocationInfo;
import java.util.Date;
import java.util.Calendar;

/* Predict the future location of a user by letting him follow roads */

public class PredictFollowRoad extends Predict {

  RoadMesh roads; // class that keeps all the roads, and provides some functions for them

  public PredictFollowRoad(String user, LatLong predictLoc, int time, RoadMesh roads) {
    /* Constructor */
    super(user, predictLoc, time);
    this.roads = roads;
  }

  public void run() {
    LatLong curLoc = new LatLong(); // Current location
    RoadSegment curRoad = new RoadSegment(); // Keep the current road we are on
    RoadDistance[] rw;
    RoadDistance prevrw;
    LatLong prevLoc; // most often same as current location
    LatLong prev2Loc; // real previous location
    LatLong nextCrossing;
    Date now;
    LocationInfo[] locinfo;
    Calendar bestDateTime;
    double relative = 0; // Relative place on a road (start=0, end=1)
    double d;

    try {
      while (!abort) {
        // get the current location of the user
        ud = uls.getUserData(user);
        userLoc = ud.getLocation();
        // calculate the distance we’ll move in this run
        dist = ud.getSpeed() / 3600 * time;
        // ==> Snap the location of the user onto the closest road <==
        // Find all roads within 20 meters from the current position
        rw = roads.getRoadDistance(userLoc, 20);
        d = 999999; // Big number
        // Find the closest of them
        for(int i=0; i<rw.length; i++) {
          if( rw[i].getDistance()<d ) {
            d = rw[i].getDistance();
            curLoc.setLatitude( rw[i].getClosest().getLatitude() );
            curLoc.setLongitude( rw[i].getClosest().getLongitude() );
            relative = rw[i].getRelative();
            curRoad.setStart(rw[i].getStart());
            curRoad.setEnd(rw[i].getEnd());
          }
        }
        if( curRoad==null ) {
          // We were not able to find the road the user is on. He might be off-road
        }else {
          // ==> Find the previous location from history <==
          prevLoc = null; // most often same as current location
          prev2Loc = null; // real previous location
          now = new Date();
          locinfo = ulh.getHistory(user, 5 * 1000, now.getTime() - (60 * 1000), now.getTime());
          bestDateTime = locinfo[0].getDateTime();
          prevLoc = locinfo[0].getLocation();
      }
    }
  }
}
prev2Loc = prevLoc;
// Find the one but newest and newest in the list of returned positions
for (int i = 1; i < locinfo.length; i++) {
    if (bestDateTime.before(locinfo[i].getDateTime())) {
        bestDateTime = locinfo[i].getDateTime();
        prev2Loc = prevLoc;
        prevLoc = locinfo[i].getLocation();
    }
}

// Map the previous location to the road the user is on now.
prevrw = roads.getRoadRoad2Point(curRoad, prev2Loc);
// Next crossing marks the end of the road in which the user is currently going
nextCrossing = null;
// Direction points the direction we are walking on a road:
// - +1: from start to end
// - -1: from end to start
int direction = 0;
if( curRoad.equals(prevrw) ) {
    // current and previous location are on same road
    if (relative < prevrw.getRelative()) {
        // User moving to beginning of road
        nextCrossing = curRoad.getStart();
        direction = -1;
    }else {
        // User moving to end of the road
        nextCrossing = curRoad.getEnd();
        direction = 1;
    }
}else {
    // previous location is on a different road
    if( curRoad.getStart().equals(prevrw.getStart()) ) {
        nextCrossing = curRoad.getEnd();
        direction = 1;
    }else if( curRoad.getStart().equals(prevrw.getEnd()) ) {
        nextCrossing = curRoad.getEnd();
        direction = 1;
    }else if( curRoad.getEnd().equals(prevrw.getStart()) ) {
        nextCrossing = curRoad.getStart();
        direction = -1;
    }else if( curRoad.getEnd().equals(prevrw.getEnd()) ) {
        nextCrossing = curRoad.getStart();
        direction = -1;
    }else {
        System.out.println("Error, this is impossible in Predict5 (139)");
    }
}

// Calculate the next position of the user
nextLoc = getNextLoc(curLoc, curRoad, direction, nextCrossing, dist);
// Pass the values by value in the loc var, otherwise class FollowUser will not be able to follow
predictLoc.setLatitude(nextLoc.getLatitude());
predictLoc.setLongitude(nextLoc.getLongitude());
// Wait some time before doing the next update
Thread.sleep( (int)this.delay * 1000);
}
}

catch( Exception ex ) {
    ex.printStackTrace();
    System.out.println("Predict 'Follow road' stopped");
}

private LatLong getNextLoc(LatLong curLoc, RoadSegment curRoad, int direction, LatLong nextCrossing, double dist) {
    /* Calculate the next location of a person
     * - LatLong curLoc: location from where we have to do the prediction
     * - Road curRoad: current road we are on
     * - int direction: which direction are we moving on the road (1 is from start to end, -1 is from end to start)
     * - LatLong nextCrossing: what is the endpoint of the road in the direction we are moving
     * - double dist: what is the distance in km we have to move
     * The function will call itself when moving over the end of a road with new variables
    */

    // Calculate the next location of a person
    nextLoc = new LatLong(curLoc.getLatitude() + dist * Math.cos(curLoc.getLongitude()),
                           curLoc.getLongitude() + dist * Math.sin(curLoc.getLongitude()));
    // Correct the longitude if it is out of bounds
    if (nextLoc.getLongitude() > 180) nextLoc.setLongitude(nextLoc.getLongitude() - 360);
    if (nextLoc.getLongitude() < -180) nextLoc.setLongitude(nextLoc.getLongitude() + 360);
    // Calculate the road we are going to next
    nextRoad = roads.getRoadRoad2Point(curRoad, nextLoc);
    // Calculate the next crossing point on the road
    nextCrossing = nextRoad.getStart();
/*
   LatLong nextLoc;
   RoadDistance[] rw;
   RoadSegment nextRoad;
   double d;

   nextLoc = new LatLong(); // Return variable
   // Calculate the distance to the next crossing.
   d = roads.getDistance(curLoc, nextCrossing);

   if (d<0.0001) {
      // Hm, we are already at the end of the road or we got an error. Stop, otherwise we might
      loop
      nextLoc.setLatitude(curLoc.getLatitude());
      nextLoc.setLongitude(curLoc.getLongitude());
   } else {
      if (d <= dist) {
         // User will in this timeframe move off this road
         rw = roads.getRoadDistance(nextCrossing, 1);
         switch (rw.length) {
            case 0:
               // Ehm, aren't we on a road at the moment?
               break;
            case 2:
               // Road will go on here, only one possibility
               dist -= d; // distance to move on next road
               // Find out which of the results will be the next road.
               if (rw[0].equals(curRoad)) {
                  nextRoad = rw[1];
               } else {
                  nextRoad = rw[0];
               }
               // Set the current location to the beginning of the next road
               curLoc = nextCrossing;
               // Set the nextCrossing to the end of the next road, and set the direction we walk the
               road
               if (nextRoad.getStart().equals(curLoc)) {
                  nextCrossing = nextRoad.getEnd();
                  direction = 1;
               } else {
                  nextCrossing = nextRoad.getStart();
                  direction = -1;
               }
               nextLoc = getNextLoc(curLoc, nextRoad, direction, nextCrossing, dist);
               break;
            case 1:
               // Ehm, dead road end?
               default:
                  // User can choose next road, sorry, I cannot predict which
                  // We will move user only to the end of the road
                  nextLoc.setLatitude(nextCrossing.getLatitude());
                  nextLoc.setLongitude(nextCrossing.getLongitude());
                  break;
               }
         } else {
            // User will stay on this road in the timeframe
            nextLoc.setLatitude(curLoc.getLatitude() +
                  (nextCrossing.getLatitude() - curLoc.getLatitude()) *
                  (dist / d));
            nextLoc.setLongitude(curLoc.getLongitude() +
                  (nextCrossing.getLongitude() - curLoc.getLongitude()) *
                  (dist / d));
         }
      }
      return nextLoc;
   }
*/
```java
package predict;

import javax.swing.*;
import com.borland.jbcl.layout.*;
import java.awt.*;
import java.awt.event.*;
import java.lang.Math;
import java.math.*;
import nl.freeband.wasp.usercontextmanager.UserLocation;
import nl.freeband.wasp.usercontextmanager.UserLocationServiceLocator;
import nl.freeband.wasp.usercontextmanager.UserLocationSimulator;
import nl.freeband.wasp.usercontextmanager.UserLocationSimulatorServiceLocator;
import nl.freeband.wasp.usercontextmanager.UserData;
import nl.freeband.wasp.common.LatLong;
import nl_impl.freeband.wasp.mapserver.MapPointAirialPicture;
import nl_impl.freeband.wasp.mapserver.MapPointAirialPictureServiceLocator;
import nl.freeband.wasp.mapserver.LatLongMap;
import java.awt.image.ImageObserver;

public class PredictGUI extends JFrame {
    UserLocationSimulator uls;
    UserLocation ul;
    MapPointAirialPicture maps;
    FollowUser followuser;
    LatLong topleft;
    LatLong bottomright;

    RoadMesh roads;

    FollowUser followPredictSameSpeed;
    PredictSameSpeed predictSameSpeed;
    LatLong predictSameSpeedLoc;
    FollowUser followPredictSameCurve;
    PredictSameCurve predictSameCurve;
    LatLong predictSameCurveLoc;
    FollowUser followPredictFollowRoad;
    PredictFollowRoad predictFollowRoad;
    LatLong predictFollowRoadLoc;

    XYLayout xYLayout1 = new XYLayout();
    JLabel Map = new JLabel();
    JLabel MapDraw = new JLabel();
    JButton GetMap = new JButton();
    JLabel jLabel1 = new JLabel();
    JLabel jLabel2 = new JLabel();
    JTextField Zoom = new JTextField();
    JScrollPane LoggerFrame = new JScrollPane();
    JLabel jLabel3 = new JLabel();
    JLabel jLabel4 = new JLabel();
    JButton PredictSameSpeedStart = new JButton();
    JButton PredictSameSpeedStop = new JButton();
    JLabel jLabel5 = new JLabel();
    JButton PredictSameCurveStart = new JButton();
    JButton PredictSameCurveStop = new JButton();
    JLabel jLabel6 = new JLabel();
    JButton PredictFollowRoadStart = new JButton();
    JButton PredictFollowRoadStop = new JButton();
    JButton DrawRoads = new JButton();

    public PredictGUI() {
        roads = null;
        try {
            ul = new UserLocationServiceLocator().getUserLocationService();
            uls = new UserLocationSimulatorServiceLocator().getUserLocationSimulatorService();
            maps = new MapPointAirialPictureServiceLocator().getMapPointAirialPictureService();
            jbInit();
```
// Fill user list
String[] users = uls.listUserIDs();
for( int i=0; i<users.length; i++ )
    User.addItem(users[i]);
}
catch(Exception e) {
e.printStackTrace();
}
}

public static void main(String[] args) {
    PredictGUI predictGUI = new PredictGUI();
}

private void jbInit() throws Exception {
    this.getContentPane().setLayout(xYLayout1);
    this.getContentPane().setBackground(SystemColor.control);
    this.setEnabled(true);
    this.setResizable(true);
    this.addComponentListener(new MapFrame_this_componentAdapter(this));
    Map.setBorder(BorderFactory.createLoweredBevelBorder());
    Map.setText("");
    Map.addMouseListener(new MapFrame_Map_mouseAdapter(this));
    MapDraw.setText("");
    xYLayout1.setWidth(670);
    xYLayout1.setHeight(740);
    GetMap.setText("Get map");
    GetMap.addMouseListener(new MapFrame_GetMap_mouseAdapter(this));
    jLabel1.setText("User:");
    jLabel2.setText("Zoom:");
    Zoom.setText("");
    Logger.setEditable(false);
    Logger.setLineWrap(false);
    FollowStart.setText("Start");
    FollowStart.addMouseListener(new MapFrame_DrawStart_mouseAdapter(this));
    FollowStop.setText("Stop");
    FollowStop.addMouseListener(new MapFrame_DrawStop_mouseAdapter(this));
    jLabel3.setFont(new java.awt.Font("Dialog", 1, 11));
    jLabel3.setForeground(Color.red);
    jLabel3.setOpaque(false);
    jLabel3.setText("Follow user:");
    jLabel4.setFont(new java.awt.Font("Dialog", 1, 11));
    jLabel4.setForeground(Color.blue);
    jLabel4.setText("Same speed:");
    PredictSameSpeedStart.setText("Start");
    PredictSameSpeedStop.setText("Stop");
    PredictSameSpeedStart.addMouseListener(new MapFrame_PredictSameSpeedStart_mouseAdapter(this));
    PredictSameSpeedStop.addMouseListener(new MapFrame_PredictSameSpeedStop_mouseAdapter(this));
    jLabel5.setFont(new java.awt.Font("Dialog", 1, 11));
    jLabel5.setForeground(Color.green);
    jLabel5.setText("Same curve:");
    PredictSameCurveStart.setText("Start");
    PredictSameCurveStop.setText("Stop");
    PredictSameCurveStart.addMouseListener(new MapFrame_PredictSameCurveStart_mouseAdapter(this));
    PredictSameCurveStop.addMouseListener(new MapFrame_PredictSameCurveStop_mouseAdapter(this));
    jLabel6.setFont(new java.awt.Font("Dialog", 1, 11));
    jLabel6.setForeground(Color.magenta);
    jLabel6.setText("Follow road:");
    PredictFollowRoadStart.setText("Start");
    PredictFollowRoadStop.setText("Stop");
    DrawRoads.setText("Draw");
    DrawRoads.addMouseListener(new MapFrame_Draw Roads_mouseAdapter(this));
    this.getContentPane().add(Zoom, new XYConstraints(26, 532, 96, -1));
    this.getContentPane().add(Logger, new XYConstraints(26, 501, 36, 24));
    this.getContentPane().add(Map, new XYConstraints(20, 21, 636, 461));
    this.getContentPane().add(MapDraw, new XYConstraints(20, 21, 636, 461));
}
void GetMap_mouseClicked(MouseEvent e) {
String user = User.getText();
String user = String.valueOf(User.getSelectedItem());
double scale;
try {
    scale = Double.parseDouble(Zoom.getText()) * 693;
} catch( NumberFormatException nfex) {
    scale = 1;
    Zoom.setText(String.valueOf(scale));
}
int height = Map.getHeight();
int width = Map.getWidth();
Logger.append("Loading map for " + user + "\n");

/*try {
    LatLong loc = ul.getUserLocation(user);
    Logger.append(" Location: (" + loc.getLatitude() + ";" + loc.getLongitude() + ")\n");
    Logger.append(" Size: " + width + "\" width + "\" height + "\"\n");
    LatLongMap map = maps.getMap(loc, width, height, scale);
    if( map!=null ) {
        topleft = map.getTopLeft();
        bottomright = map.getBottomRight();
        Logger.append(" Vissible: (" + topleft.getLatitude() + "\";" + topleft.getLongitude() + ") - (" +
                        bottomright.getLatitude() + "\";" + bottomright.getLongitude() + ")\n");
        byte[] rawdata = map.getImageData();
        ImageIcon mapdata = new ImageIcon(rawdata);
        Map.setIcon(mapdata);
        Map.setText("\n");
    } else {
        Logger.append(" No map data available\n");
    }
}*/
roads=null;
/*
} catch(Exception ex) {
    ex.printStackTrace();
    Logger.append(" Error while loading map: " + ex.getMessage() + "\n");
    topleft = new LatLong();
    bottomright = new LatLong();
    topleft.setLatitude(52.218308);
    topleft.setLongitude(6.886051);
    bottomright.setLatitude(52.207540);
    bottomright.setLongitude(6.914719);
    Map.setText("No map loaded, but bounds set\n");
}*/
}

void DrawStart_mouseClicked(MouseEvent e) {
if( followuser==null || !followuser.isAlive() ) {
    /*
     * DrawStart = new JButton("Draw Start");
     * DrawStart.addMouseListener(new MouseAdapter() {
     *     public void mousePressed(MouseEvent e) {
     *         if( followuser==null || !followuser.isAlive() ) {
     *             return;
     *         }
     *         try {
     *             LatLong loc = ul.getUserLocation(user);
     *             Logger.append("Location: (" + loc.getLatitude() + ";" + loc.getLongitude() + ")\n");
     *             Logger.append("Size: " + width + ";" + height + "\n");
     *             LatLongMap map = maps.getMap(loc, width, height, scale);
     *             if( map!=null ) {
     *                 topleft = map.getTopLeft();
     *                 bottomright = map.getBottomRight();
     *                 Logger.append("Visible: (" + topleft.getLatitude() + ";" + topleft.getLongitude() + ") - (" +
     *                              bottomright.getLatitude() + ";" + bottomright.getLongitude() + ")\n");
     *                 byte[] rawdata = map.getImageData();
     *                 ImageIcon mapdata = new ImageIcon(rawdata);
     *                 Map.setIcon(mapdata);
     *                 Map.setText("\n");
     *             } else {
     *                 Logger.append("No map data available\n");
     *             }
     *         } catch(Exception ex) {
     *             ex.printStackTrace();
     *             Logger.append("Error while loading map: " + ex.getMessage() + "\n");
     *             topleft = new LatLong();
     *             bottomright = new LatLong();
     *             topleft.setLatitude(52.218308);
     *             topleft.setLongitude(6.886051);
     *             bottomright.setLatitude(52.207540);
     *             bottomright.setLongitude(6.914719);
     *             Map.setText("No map loaded, but bounds set\n");
     *         }*/
     */
}
String user = String.valueOf(User.getSelectedItem());
//String user = User.getText();
if (user != null) {
    Logger.append("Following user from now on\n");
    followuser = new FollowUser(MapDraw, topleft, bottomright, Logger, user, Color.RED);
    followuser.start();
}

void DrawStop_mouseClicked(MouseEvent e) {
    if (followuser != null) {
        if (followuser.isAlive()) {
            Logger.append("Not following user any more\n");
            followuser.doStop();
            //followuser = null;
        }
    }
}

void PredictSameSpeedStart_mouseClicked(MouseEvent e) {
    if (followPredictSameSpeed == null || !followPredictSameSpeed.isAlive()) {
        String user = String.valueOf(User.getSelectedItem());
        if (user != null) {
            try {
                predictSameSpeedLoc = new LatLong();
                LatLong loc = ul.getUserLocation(user);
                predictSameSpeedLoc.setLatitude(loc.getLatitude());
                predictSameSpeedLoc.setLongitude(loc.getLongitude());
                predictSameSpeed = new PredictSameSpeed(user, predictSameSpeedLoc, 20);
                predictSameSpeed.start();
                Logger.append("PredictSameSpeed running from now on\n");
                followPredictSameSpeed = new FollowUser(MapDraw, topleft, bottomright, Logger,
                    predictSameSpeedLoc, Color.BLUE);
                followPredictSameSpeed.start();
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
    }
}

void PredictSameSpeedStop_mouseClicked(MouseEvent e) {
    if (followPredictSameSpeed != null) {
        if (followPredictSameSpeed.isAlive()) {
            Logger.append("Not predicting same speed any more\n");
            followPredictSameSpeed.doStop();
            predictSameSpeed.doStop();
        }
    }
}

void PredictSameCurveStart_mouseClicked(MouseEvent e) {
    if (followPredictSameCurve == null || !followPredictSameCurve.isAlive()) {
        String user = String.valueOf(User.getSelectedItem());
        if (user != null) {
            try {
                predictSameCurveLoc = new LatLong();
                LatLong loc = ul.getUserLocation(user);
                predictSameCurveLoc.setLatitude(loc.getLatitude());
                predictSameCurveLoc.setLongitude(loc.getLongitude());
                predictSameCurve = new PredictSameCurve(user, predictSameCurveLoc, 20);
                predictSameCurve.start();
                Logger.append("PredictSameCurve running from now on\n");
                followPredictSameCurve = new FollowUser(MapDraw, topleft, bottomright, Logger,
                    predictSameCurveLoc, Color.GREEN);
                followPredictSameCurve.start();
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
    }
}
void PredictSameCurveStop_mouseClicked(MouseEvent e) {
    if (followPredictSameCurve!=null) {
        if (followPredictSameCurve.isAlive()) {
            Logger.append("Not predicting2 any more\n");
            followPredictSameCurve.doStop();
            predictSameCurve.doStop();
        }
    }
}

void PredictFollowRoadStart_mouseClicked(MouseEvent e) {
    if (followPredictFollowRoad==null || !followPredictFollowRoad.isAlive()) {
        String user = String.valueOf(User.getSelectedItem());
        if (user != null) {
            try {
                predictFollowRoadLoc = new LatLong();
                LatLong loc = ul.getUserLocation(user);
                predictFollowRoadLoc.setLatitude(loc.getLatitude());
                predictFollowRoadLoc.setLongitude(loc.getLongitude());
                predictFollowRoad = new PredictFollowRoad(user, predictFollowRoadLoc, 20, roads);
                predictFollowRoad.start();
                Logger.append("PredictFollowRoad running from now on\n");
                followPredictFollowRoad = new FollowUser(MapDraw, topleft, bottomright, Logger,
                predictFollowRoadLoc, Color.MAGENTA);
                followPredictFollowRoad.start();
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
    }
}

void PredictFollowRoadStop_mouseClicked(MouseEvent e) {
    if (followPredictFollowRoad!=null) {
        if (followPredictFollowRoad.isAlive()) {
            Logger.append("Not predicting5 any more\n");
            followPredictFollowRoad.doStop();
            predictFollowRoad.doStop();
        }
    }
}

void DrawRoads_mouseClicked(MouseEvent e) {
    if (roads==null) roads = new RoadMesh(MapDraw, topleft, bottomright);
    roads.draw(Color.PINK);
}

void this_componentResized(ComponentEvent e) {
    int width = this.getWidth();
    int height = this.getHeight();
    if (width<280) width=280;
    if (height<350) height=350;
    if (width!=this.getWidth() | height!=this.getHeight()) {
        this.setBounds(this.getX(), this.getY(), width, height);
    }
    //System.out.println("Frame: \" + this.getX() + \"; \" + this.getY() + \"; \" + this.getWidth() + \"; \" +
    this.getHeight());
    //System.out.println("Map: \" + Map.getX() + \"; \" + Map.getY() + \"; \" + Map.getWidth() + \"; \" +
    Map.getHeight());
    //System.out.println("[Label1: \" + JLabel1.getX() + \"; \" + JLabel1.getY() + \"; \" +
    JLabel1.getWidth() + \"; \" + JLabel1.getHeight());
    //System.out.println("[Label2: \" + JLabel2.getX() + \"; \" + JLabel2.getY() + \"; \" +
    JLabel2.getWidth() + \"; \" + JLabel2.getHeight());
    //System.out.println("User: \" + User.getX() + \"; \" + User.getY() + \"; \" + User.getWidth() + \"; \" +
    User.getHeight());
    //System.out.println("Zoom: \" + Zoom.getX() + \"; \" + Zoom.getY() + \"; \" + Zoom.getWidth() + \";
    \" + Zoom.getHeight());
    //System.out.println("GetMap: \" + GetMap.getX() + \"; \" + GetMap.getY() + \"; \" +
    GetMap.getWidth() + \"; \" + GetMap.getHeight());
}
//System.out.println("LoggerFrame: "+LoggerFrame.getX()+"; "+LoggerFrame.getY()+"; "+LoggerFrame.getWidth()+"; "+LoggerFrame.getHeight());
//System.out.println("Logger: "+Logger.getX()+"; "+Logger.getY()+"; "+Logger.getWidth()+"; "+Logger.getHeight());
Map.setBounds(20,20,width-40,height-270);
MapDraw.setBounds(20,20,width-40,height-270);
jLabel1.setBounds(20,height-230,40,20);
jLabel2.setBounds(20,height-195,40,20);
User.setBounds(65,height-230,100,25);
Zoom.setBounds(65,height-195,100,20);
GetMap.setBounds(10,height-165,85,30);
DrawRoads.setBounds(105,height-165,65,30);
jLabel3.setBounds(185,height-230,80,20);
FollowStart.setBounds(185,height-205,80,30);
FollowStop.setBounds(185,height-165,80,30);
jLabel4.setBounds(185,height-230,80,20);
PredictSameSpeedStart.setBounds(185,height-205,80,30);
PredictSameSpeedStop.setBounds(185,height-165,80,30);
jLabel5.setBounds(285,height-230,80,20);
PredictSameCurveStart.setBounds(285,height-205,80,30);
PredictSameCurveStop.setBounds(285,height-165,80,30);
jLabel6.setBounds(385,height-230,80,20);
PredictFollowRoadStart.setBounds(385,height-205,80,30);
PredictFollowRoadStop.setBounds(385,height-165,80,30);
LoggerFrame.setBounds(0,height-110,width-10,83);
Logger.setBounds(0,0,width-13,80);
}

void Map_mousePressed(MouseEvent e) {
if (e.getButton()==e.BUTTON3) {
    LatLong point = new LatLong();
    double leftX = topleft.getLongitude();
    double bottomY = bottomright.getLatitude();
    double facX = ((double) MapDraw.getWidth()) /
    (bottomright.getLongitude() - topleft.getLongitude());
    double facY = ((double) MapDraw.getHeight()) /
    (bottomright.getLatitude() - topleft.getLatitude());
    point.setLongitude(leftX + e.getX()/facX);
    point.setLatitude(bottomY -(MapDraw.getHeight()-e.getY())/facY);
    Logger.append("Button pressed at (" + String.valueOf(e.getX()) + "; "+
    String.valueOf(e.getY())) + "; "+
    String.valueOf(point.getLongitude()) + "; "+
    String.valueOf(point.getLatitude()) + ");
    if (roads==null) roads = new RoadMesh(MapDraw, topleft, bottomright);
    RoadSegment[] r = roads.getRoad(point, 20);
    Logger.append("There are " + String.valueOf(r.length) + " roads within 20m of this point.
    ");
    //Logger.append("Button pressed at (" + String.valueOf(e.getX()) + "; "+
    String.valueOf(MapDraw.getHeight()-e.getY()) + ");
    }
}

class MapFrame_GetMap_mouseAdapter extends java.awt.event.MouseAdapter {
    PredictGUI adaptee;
    MapFrame_GetMap_mouseAdapter(PredictGUI adaptee) {
        this.adaptee = adaptee;
    }
    public void mouseClicked(MouseEvent e) {
        adaptee.getMapMouseClicked(e);
    }
}

class MapFrame_this_componentAdapter extends java.awt.event.ComponentAdapter {
    PredictGUI adaptee;
    MapFrame_this_componentAdapter(PredictGUI adaptee) {

}}
this.adaptee = adaptee;
}
public void componentResized(ComponentEvent e) {
    adaptee.this_componentResized(e);
}
}

class MapFrame_DrawStart_mouseAdapter extends java.awt.event.MouseAdapter {
    PredictGUI adaptee;
    MapFrame_DrawStart_mouseAdapter(PredictGUI adaptee) {
        this.adaptee = adaptee;
    }
    public void mouseClicked(MouseEvent e) {
        adaptee.DrawStart_mouseClicked(e);
    }
}

class MapFrame_DrawStop_mouseAdapter extends java.awt.event.MouseAdapter {
    PredictGUI adaptee;
    MapFrame_DrawStop_mouseAdapter(PredictGUI adaptee) {
        this.adaptee = adaptee;
    }
    public void mouseClicked(MouseEvent e) {
        adaptee.DrawStop_mouseClicked(e);
    }
}

class MapFrame_PredictSameSpeedStart_mouseAdapter extends java.awt.event.MouseAdapter {
    PredictGUI adaptee;
    MapFrame_PredictSameSpeedStart_mouseAdapter(PredictGUI adaptee) {
        this.adaptee = adaptee;
    }
    public void mouseClicked(MouseEvent e) {
        adaptee.PredictSameSpeedStart_mouseClicked(e);
    }
}

class MapFrame_PredictSameSpeedStop_mouseAdapter extends java.awt.event.MouseAdapter {
    PredictGUI adaptee;
    MapFrame_PredictSameSpeedStop_mouseAdapter(PredictGUI adaptee) {
        this.adaptee = adaptee;
    }
    public void mouseClicked(MouseEvent e) {
        adaptee.PredictSameSpeedStop_mouseClicked(e);
    }
}

class MapFrame_PredictSameCurveStart_mouseAdapter extends java.awt.event.MouseAdapter {
    PredictGUI adaptee;
    MapFrame_PredictSameCurveStart_mouseAdapter(PredictGUI adaptee) {
        this.adaptee = adaptee;
    }
    public void mouseClicked(MouseEvent e) {
        adaptee.PredictSameCurveStart_mouseClicked(e);
    }
}

class MapFrame_PredictSameCurveStop_mouseAdapter extends java.awt.event.MouseAdapter {
    PredictGUI adaptee;
    MapFrame_PredictSameCurveStop_mouseAdapter(PredictGUI adaptee) {
        this.adaptee = adaptee;
    }
    public void mouseClicked(MouseEvent e) {
        adaptee.PredictSameCurveStop_mouseClicked(e);
    }
}
class MapFrame_PredictFollowRoadStart_mouseAdapter extends java.awt.event.MouseAdapter {
    PredictGUI adaptee;
    MapFrame_PredictFollowRoadStart_mouseAdapter(PredictGUI adaptee) {
        this.adaptee = adaptee;
    }
    public void mouseClicked(MouseEvent e) {
        adaptee.PredictFollowRoadStart_mouseClicked(e);
    }
}

class MapFrame_PredictFollowRoadStop_mouseAdapter extends java.awt.event.MouseAdapter {
    PredictGUI adaptee;
    MapFrame_PredictFollowRoadStop_mouseAdapter(PredictGUI adaptee) {
        this.adaptee = adaptee;
    }
    public void mouseClicked(MouseEvent e) {
        adaptee.PredictFollowRoadStop_mouseClicked(e);
    }
}

class MapFrame_DrawRoads_mouseAdapter extends java.awt.event.MouseAdapter {
    PredictGUI adaptee;
    MapFrame_DrawRoads_mouseAdapter(PredictGUI adaptee) {
        this.adaptee = adaptee;
    }
    public void mouseClicked(MouseEvent e) {
        adaptee.DrawRoads_mouseClicked(e);
    }
}

class MapFrame_Map_mouseAdapter extends java.awt.event.MouseAdapter {
    PredictGUI adaptee;
    MapFrame_Map_mouseAdapter(PredictGUI adaptee) {
        this.adaptee = adaptee;
    }
    public void mousePressed(MouseEvent e) {
        adaptee.Map_mousePressed(e);
    }
}
PredictSameCurve.java

package predict;

import nl.freeband.wasp.common.LatLong;
import nl.freeband.wasp.common.LocationInfo;
import java.util.Date;
import java.util.Calendar;

// Class that takes care of same curve prediction
public class PredictSameCurve extends Predict {

    public PredictSameCurve(String user, LatLong predictLoc, int time) {
        /* Constructor
         * use super class to construct
         */
        super(user, predictLoc, time);
    }

    public void run() {
        // Read the current values from the wasp platform
        Date now;
        Calendar bestDateTime; // time of the prevDirection
        Calendar bestDateTime2; // time of the prev2Direction
        LocationInfo[] locinfo;
        double curDirection;
        double prevDirection; // last direction
        double prev2Direction; // direction before last direction
        double timediff; // delta t for bestDateTime and bestDateTime2
        double direction; // direction used to do the prediction
        double r, x, y, a, b;
        try {
            while( !abort ) {
                // Read the current values from the wasp platform
                ud = uls.getUserData(user);
                userLoc = ud.getLocation();
                dist = ud.getSpeed() / 3600 * time;
                curDirection = ud.getDirection() / 360 * 2 * Math.PI;

                // Search for the previous directions
                prevDirection=0;
                prev2Direction=0;
                now = new Date();
                locinfo = ulh.getHistory(user, 5*1000, now.getTime()-(60*1000), now.getTime());
                bestDateTime = locinfo[0].getDateTime();
                bestDateTime2 = bestDateTime;
                prevDirection = locinfo[0].getDirection();
                prev2Direction = prevDirection;
                for( int i=1; i<locinfo.length; i++ ) {
                    if( bestDateTime.before(locinfo[i].getDateTime()) ) {
                        bestDateTime2 = bestDateTime;
                        bestDateTime = locinfo[i].getDateTime();
                        prev2Direction = prevDirection;
                        prevDirection = locinfo[i].getDirection();
                    }
                }
                prevDirection = prevDirection / 360 * 2 * Math.PI;
                prev2Direction = prev2Direction / 360 * 2 * Math.PI;

                // calculate the time between the 2 previous directions
                timediff = (bestDateTime.getTimeInMillis() - bestDateTime2.getTimeInMillis())/1000; // In seconds
                r = 6378.14 * 1.3 / 360 * 2 * Math.PI;
                x = userLoc.getLongitude() / 360 * 2 * Math.PI;
                y = userLoc.getLatitude() / 360 * 2 * Math.PI;
                a = dist / r;
                b = -Math.acos( (Math.cos(dist / (r * 1.3)) - Math.sin(x) * Math.sin(x)) / (Math.cos(x) * Math.sin(x)) );

                // calculate the direction we'll use for prediction
                direction = curDirection + (prevDirection - prev2Direction) * (time / timediff);
                // Calculate the values that will help us in prediction
                r = 6378.14 * 1.3 / 360 * 2 * Math.PI;
                x = userLoc.getLongitude() / 360 * 2 * Math.PI;
                y = userLoc.getLatitude() / 360 * 2 * Math.PI;
                a = dist / r;
                b = -Math.acos( (Math.cos(dist / (r * 1.3)) - Math.sin(x) * Math.sin(x)) / (Math.cos(x) * Math.sin(x)) );
            }
        }
    }
}
// Set the predicted location
predictLoc.setLongitude(userLoc.getLongitude() + Math.sin(direction) * a);
predictLoc.setLatitude(userLoc.getLatitude() - Math.cos(direction) * b);
//System.out.println("a: " + a + "; b: " + b);
    // Wait before we do a new prediction
    Thread.sleep((int)this.delay * 1000);
}
}
    catch( Exception ex ) {
        ex.printStackTrace();
    }
    System.out.println("Predict 'Same curve' stopped");
}
}
package predict;

import nl.freeband.wasp.common.LatLong;

// Class that takes care of same speed prediction
public class PredictSameSpeed extends Predict {

    public PredictSameSpeed(String user, LatLong predictLoc, int time) {
        /* Constructor */
        // use super class to construct */
        super(user, predictLoc, time);
    }

    public void run() {
        double direction; // direction in radians
        double r, x, y, a, b;

        try {
            while( !abort ) {
                // Get the current location and direction
                ud = uls.getUserData(user);
                userLoc = ud.getLocation();
                dist = ud.getSpeed() / 3600 * time;
                direction = ud.getDirection() / 360 * 2 * Math.PI;

                // Calculate vars that are needed to calculate the predicted location
                r = 6378.14 * 1.3 / 360 * 2 * Math.PI;
                x = userLoc.getLongitude() / 360 * 2 * Math.PI;
                y = userLoc.getLatitude() / 360 * 2 * Math.PI;
                a = dist / r;
                b = -Math.acos((Math.cos(dist / (r * 1.3)) - Math.sin(x) * Math.sin(x)) / (Math.cos(x) * Math.cos(x)));

                // Set the predicted location
                predictLoc.setLongitude(userLoc.getLongitude() + Math.sin(direction) * a);
                predictLoc.setLatitude(userLoc.getLatitude() - Math.cos(direction) * b);

                // Wait before we do a new prediction
                Thread.sleep((int)this.delay * 1000);
            }
        }

        catch( Exception ex ) {
            ex.printStackTrace();
        }

        System.out.println("Predict 'Same speed' stopped");
    }
}
package predict;

import nl.freeband.wasp.common.LatLong;

/* Class that is an extention on RoadSegment and also stores the closest point
* on a road to the point, the distance to this point and the relative position
* of this point (0=start of road, 1=end of road)
*/

public class RoadDistance extends RoadSegment{
    private LatLong closest;
    private double distance;
    private double relative;

    public RoadDistance() {
        /* Constructor
         * + input:
         *   - [none]
         * + output:
         *   - [none]
         */
        super();
        closest = new LatLong();
        distance = 0;
        relative = 0;
    }

    public RoadDistance(LatLong start, LatLong end, LatLong closest, double distance, double relative) {
        /* Constructor
         * + input:
         *   - start: beginning point of the road
         *   - end: ending point of the road
         *   - closest: the point on the road segment that is closest to a given point
         *   - distance: the distance between the road segment and a given point
         *   - relative: the relative position on the road that is closest to a
         *     given point (0=start of road, 1=end of road)
         * + output:
         *   - [none]
         */
        super(start, end);
        this.setClosest(closest);
        this.setDistance(distance);
        this.setRelative(relative);
    }

    public void setClosest(LatLong closest) {
        /* Set the closest position
         * + input:
         *   - closest
         * + output:
         *   - [none]
         */
        this.closest = closest;
    }

    public LatLong getClosest() {
        /* Get the closest position
         * + input:
         *   - [none]
         * + output:
         *   - closest position
         */
        return this.closest;
    }

    public void setDistance(double distance) {
        /* Set the distance
         * + input:
         *   - distance
         * + output:
         *   - [none]
         */
    }
}
public double getDistance() {
    /* Get the distance
     * + input:
     *   - [none]
     * + output:
     *   - distance
     */
    return this.distance;
}

public void setRelative(double relative) {
    /* Set the relative position
     * + input:
     *   - relative (between 0 and 1)
     * + output:
     *   - [none]
     */
    if( relative<0 ) relative = 0;
    if( relative>1 ) relative = 1;
    this.relative = relative;
}

public double getRelative() {
    /* Get the relative position
     * + input:
     *   - [none]
     * + output:
     *   - relative position
     */
    return this.relative;
}
RoadMesh.java

package predict;

import nl.freeband.wasp.common.LatLng;
import java.awt.Color;
import javax.swing.JLabel;
import java.awt.Graphics;
import javax.swing.JTextArea;

/* Class keeping track of all roadsegments known to the program. The class also
 * provides some functions to do calculations on the roads */

public class RoadMesh {
    private RoadSegment[] allroads;  // Contatins all road segments
    private JLabel map;  // contains the box in which we can draw
    private LatLng topleft;  // topleft position visible on the screen (map)
    private LatLng bottomright;  // bottomright position visible on the screen (map)

    public RoadMesh(JLabel map, LatLng topleft, LatLng bottomright) {
        /* Initialize the class, and set a testset of roads in allroads[]. */
        this.map = map;
        this.topleft = topleft;
        this.bottomright = bottomright;

        LatLng a = new LatLng();
        LatLng b = new LatLng();
        allroads = new RoadSegment[25];
        a.setLatitude(getLat(350)); a.setLongitude(getLong(350));
        b.setLatitude(getLat(350)); b.setLongitude(getLong(300));
        allroads[0] = new RoadSegment(a, b);
        a = new LatLng();
        b.setLatitude(getLat(300)); b.setLongitude(getLong(300));
        allroads[1] = new RoadSegment(b, a);
        a = new LatLng();
        b.setLatitude(getLat(300)); b.setLongitude(getLong(300));
        allroads[2] = new RoadSegment(a, b);
        a = new LatLng();
        b.setLatitude(getLat(300)); b.setLongitude(getLong(350));
        allroads[3] = new RoadSegment(b, a);
        a = new LatLng();
        b.setLatitude(getLat(350)); b.setLongitude(getLong(350));
        allroads[4] = new RoadSegment(b, a);
        a = new LatLng();
        b.setLatitude(getLat(350)); b.setLongitude(getLong(400));
        allroads[5] = new RoadSegment(b, a);
        a = new LatLng();
        b.setLatitude(getLat(350)); b.setLongitude(getLong(400));
        allroads[6] = new RoadSegment(b, a);
        a = new LatLng();
        b.setLatitude(getLat(350)); b.setLongitude(getLong(450));
        allroads[7] = new RoadSegment(b, a);
        a = new LatLng();
        b.setLatitude(getLat(450)); b.setLongitude(getLong(450));
        allroads[8] = new RoadSegment(b, a);
        a = new LatLng();
        b.setLatitude(getLat(450)); b.setLongitude(getLong(500));
        allroads[9] = new RoadSegment(b, a);
        a = new LatLng();
        b.setLatitude(getLat(500)); b.setLongitude(getLong(500));
        allroads[10] = new RoadSegment(b, a);
        a = new LatLng();
        b.setLatitude(getLat(500)); b.setLongitude(getLong(500));
        allroads[11] = new RoadSegment(b, a);
        a = new LatLng();
        b.setLatitude(getLat(500)); b.setLongitude(getLong(450));
        allroads[12] = new RoadSegment(b, a);
        a = new LatLng();
        b.setLatitude(getLat(450)); b.setLongitude(getLong(450));
        allroads[13] = new RoadSegment(b, a);
        a = new LatLng();
        b.setLatitude(getLat(450)); b.setLongitude(getLong(450));
        allroads[14] = new RoadSegment(b, a);
    }
}
```java
public RoadSegment[] getRoad(LatLng loc, double radius) {
    /*
    * Return an array of maximum 20 road that are within radius radius (in
    * kilometer) from location loc. In case there are more than 20 roads within
    * this radius, 20 roads will be chosen randomly.
    * + input:
    *   - loc: the position for which we want to know the roads that are near
    *   - radius: the radius of the circle with middle point loc where the road
    *     has to fall in (in kilometer)
    * + output:
    *   - array of maximum 20 RoadSegments that are within radius from loc
    */
    int a = 0;
    RoadSegment[] roads = new RoadSegment[20];
    RoadSegment[] rroads;

    // Get the roads which are within range, max 20
    for (int i = 0; i < allroads.length; i++) {
        if (getDistRoad2Point(allroads[i], loc) < (radius / 1000)) {
            if (a < roads.length) roads[a++] = allroads[i];
        }
    }

    // Put the roads from above in a new array that is exactly the size of the
    // number of roads to return
    if (a > roads.length) a = roads.length;
    rroads = new RoadSegment[a];
    for (int i = 0; i < a; i++) {
        rroads[i] = roads[i];
    }
    return rroads;
}
```
public RoadDistance[] getRoadDistance(LatLong loc, double radius) {
    /*
    * Return an array of maximum 20 RoadWeight that are within radius radius
    * (in kilometer) from location loc. In case there are more than 20 roads
    * within this radius, 20 roads will be chosen randomly.
    * + input:
    *   - loc: the position for which we want to know the roads that are near
    *   - radius: the radius of the circle with middle point loc where the road
    *     has to fall in (in kilometer)
    * + output:
    *   - array of maximum 20 RoadSegments including distance to loc that are
    *     within radius from loc
    */
    int a = 0;
    RoadDistance[] roads;
    RoadDistance[] rroads;
    RoadDistance rw;

    roads = new RoadDistance[20];
    for( int i=0; i<allroads.length; i++ ) {
        rw = getRoadRoad2Point(allroads[i], loc);
        if( rw!=null ) {
            if (rw.getDistance() < (radius / 1000)) {
                if (a < roads.length) {
                    roads[a] = rw;
                }
                a++;
            }
        }
    }

    rroads = new RoadDistance[a];
    for( int i=0; i<a; i++ ) {
        rroads[i]=roads[i];
    }

    return rroads;
}

public void draw(Color colour) {
    /*
    * Draw the roads defined in allroads[] on the screen
    * + input:
    *   - colour: the colour in which the roads have to be drawn on the screen
    * + output:
    *   - [none]
    */
    LatLong prevLoc=new LatLong();
    double topleftX=0;
    double topleftY=0;
    double facX=0;
    double facY=0;

    if( topleft==null | bottomright==null | map==null ) {
        // error
    } else {
        // Split topleft in x and y, and calculate a correction factor to
        // translate from LatLong to x,y.
        LatLong prevLoc = null;
        topleftX = topleft.getLongitude();
        topleftY = topleft.getLatitude();
        facX = ( (double) map.getWidth()) / 
        (bottomright.getLongitude() - topleft.getLongitude());
        facY = ( (double) map.getHeight()) / 
        (bottomright.getLatitude() - topleft.getLatitude());
    }
}
try {
    //Draw the roads
    Graphics graph = map.getGraphics();
    graph.setPaintMode();
    graph.setColor(colour);

    for (int i=0; i<allroads.length; i++) {
        int x1 = (int) ((allroads[i].getStart().getLongitude() - topleftX) * facX);
        int y1 = (int) ((allroads[i].getStart().getLatitude() - topleftY) * facY);
        int x2 = (int) ((allroads[i].getEnd().getLongitude() - topleftX) * facX);
        int y2 = (int) ((allroads[i].getEnd().getLatitude() - topleftY) * facY);
        graph.drawLine(x1, y1, x2, y2);
    }
    map.update(graph);
}
} catch (Exception ex) {
    ex.printStackTrace();
}

private double getLat(double y) {
    /*
    * Return the Latitude position on earth calculated from a Y position on
    * the screen
    * + input:
    *   - y: the y position on the screen
    * + output:
    *   - latitude on earth position
    */
    return (y*(topleft.getLatitude()-bottomright.getLatitude())/map.getHeight())+bottomright.getLatitude();
}

private double getLong(double x) {
    /*
    * Return the Longitude position on earth calculated from a X position on
    * the screen
    * + input:
    *   - x: the x position on the screen
    * + output:
    *   - longitude on earth position
    */
    return (x*(bottomright.getLongitude()-topleft.getLongitude())/map.getWidth())+topleft.getLongitude();
}

private int getY(double lat) {
    /*
    * Return the Y position on the screen of a Latitude position on earth.
    * + input:
    *   - lat: latitude on earth position
    * + output:
    *   - the y position on the screen
    */
    return (int) Math.round((lat-bottomright.getLatitude())/((topleft.getLatitude()-bottomright.getLatitude())/map.getHeight()));
}

private int getX(double lon) {
    /*
    * Return the X position on the screen of a Longitude position on earth.
    * + input:
    *   - lon: longitude on earth position
    * + output:
    *   - the x position on the screen
    */
    return (int) Math.round((lon-topleft.getLongitude())/((bottomright.getLongitude()-topleft.getLongitude())/map.getWidth()));
}

public RoadDistance getRoadRoad2Point(RoadSegment r, LatLong p) {
    /*
    * Return the RoadDistance of the comparison between a road and a point
    * + input:
    *   - r: the road segment
    */
    // Code
* + p: the point
* + output:
* - the road, the distance and the point on the road that is closest to p
*/
double dx;
double dy;
double t = 0;
LatLong p2 = null; // The point on r that is closest to p

if( r==null | p==null ) return null;

//get the length of the road in x and y
dx = (r.getEnd().getLongitude() - r.getStart().getLongitude());
dy = (r.getEnd().getLatitude() - r.getStart().getLatitude());

if( dx==0 & dy==0 ) {
    // Line is a point, not line
    p2 = r.getStart();
} else {
    t = ( ( ( p.getLongitude() - r.getStart().getLongitude() ) * dx ) +
          ( ( p.getLatitude() - r.getStart().getLatitude() ) * dy ) ) /
          ( dx*dx + dy*dy );
    if (t < 0) {
        // closest point on line is before starting point
        p2 = r.getStart();
    } else if (t > 1) {
        // closest point on line is after ending point
        p2 = r.getEnd();
    } else {
        // closest point on line is between beginning and ending point
        p2 = new LatLong();
p2.setLongitude(r.getStart().getLongitude() + t * dx);
p2.setLatitude(r.getStart().getLatitude() + t * dy);
    }
}

// Now the problem is downgraded to the distance between point p and p2
double s = getDistance(p, p2);

return new RoadDistance(r.getStart(), r.getEnd(), p2, s, t);

private double getDistRoad2Point(RoadSegment r, LatLong p) {
    /*
     * Return the distance in kilometers between a Road r and LatLong p (point)
     * + input:
     *   - r: the road segment
     *   - p: the point
     * + output:
     *   - the shortest distance between r and p
     */
    return getRoad2Point(r, p).getDistance();
}

public double getDistance(LatLong point1, LatLong point2) {
    /*
     * Return the distance in kilometers between point1 and point2
     * + input:
     *   - point1: the point which we have to calculate the distance to
     *   - point2: the point which we have to calculate the distance to
     * + output:
     *   - the distance between point1 and point2
     */
    double a1 = point1.getLatitude() ;// 180*Math.PI;
    double b1 = point1.getLongitude() ;// 180*Math.PI;
    double a2 = point2.getLatitude() ;// 180*Math.PI;
    double b2 = point2.getLongitude() ;// 180*Math.PI;
    Sin[a2]]/360 * 2Pi = r
    return Math.acos( Math.cos(a1) * Math.cos(b1) * Math.cos(a2) * Math.cos(b2) +
                    Math.cos(a1) * Math.sin(b1) * Math.cos(a2) * Math.sin(b2) +
                    Math.sin(a1) * Math.sin(a2) ) / 360 * 2 * Math.PI * 6378.14 * 1.3;
}
public class RoadSegment {
    private LatLong start;
    private LatLong end;

    public RoadSegment() {
        /* Constructor */
        start = new LatLong();
        end = new LatLong();
    }

    public RoadSegment(LatLong start, LatLong end) {
        /* Constructor */
        this.setStart(start);
        this.setEnd(end);
    }

    public boolean equals(RoadSegment r) {
        /* Compare if r is the same as this */
        if (this.getStart().getLatitude()==r.getStart().getLatitude() &
            this.getStart().getLongitude()==r.getStart().getLongitude() &
            this.getEnd().getLatitude()==r.getEnd().getLatitude() &
            this.getEnd().getLongitude()==r.getEnd().getLongitude() )
            return true;
        else
            return false;
    }

    public void setStart(LatLong start) {
        /* Set the beginning point of the roadsegment */
        this.start = start;
    }

    public LatLong getStart() {
        /* Get the beginning point of the roadsegment */
        return this.start;
    }

    public void setEnd(LatLong end) {
        /* Set the ending point of the roadsegment */
        this.end = end;
    }

    public LatLong getEnd() {
        /* Get the ending point of the roadsegment */
        return this.end;
    }

    public void setEnd(LatLong end) {
        /* Set the ending point of the roadsegment */
        this.end = end;
    }
}
/** Set the ending point of the roadsegment
   * + input:
   *  - start: the ending of the roadsegment
   * + output:
   *  - [none]
   */
this.end = end;
}

public LatLong getEnd() {
    /* Get the ending point of the roadsegment
     * + input:
     *  - [none]
     * + output:
     *  - ending point of the roadsegment
     */
    return this.end;
}