Web Computing

Geocaching

m Jenkin + h Roumani

1. Introduction

Mobile computing devices differ in one fundamental way from traditional computing infrastructure: they are mobile. That is, the computation can take place anywhere, anytime. Furthermore, these devices often have a very good idea about 'where they are' in the environment. In an earlier lab you utilized the orientation sensor of the hand held device in order to control the display. Here, you will utilize the GPS receiver in the device to determine where you (well where your Android device is), update a 2D map to reflect this, and then use this map to represent geocached events in the world.

Of course, many large corporations provide similar map-based software — Google and Apple come to mind. Here you build a much simpler version of the complex software infrastructure that is available commercially. That being said, much of the basic concepts introduced in this lab are quite similar to the basic software structures used in these more complex software systems.

Although there are many tasks for which geolocation provides useful and perhaps critical information, one task that is perhaps unique in this range is that of geocaching. Geocaching is a treasure-hunt like game in which one party places hidden items at different GPS-defined locations and other parties seek to find them using the GPS coordinates. This is not as easy as it sounds as there is some error in any GPS signals and knowing where a particular item is does not necessarily make that item easy to find. The code you develop in this lab will provide an infrastructure that supports geocaching. The actual task of hiding objects at different GPS-defined locations is left as an exercise for the student.

One reality of coding for the actual mobile device is that much debugging will not be possible except when you are running on the Android device itself. Much of what your application will do is tied to actually reading the value of the GPS sensor on the device. Your laptop, or the department's one, lacks such a sensor so testing/debugging on a laptop will have only limited functionality.

This lab has an associated set of 'pre lab' assignments that must be completed prior to attending the lab. You will not be permitted to take part in the lab or receive a

grade for this lab unless you have completed the pre lab exercises. Exercises in the lab are to be documented in your ePortfolio lab book.

2. Background

This lab involves building an interface with HTML, formatting it using CSS, and implementing the logic through Javascript to monitor the position of the Android device relative to a global coordinate frame. In this section we highlight the key features needed from each of these technologies.

You may want to skim through the background material in your first review of this lab. After wards, go to the actual exercises associated with the lab and then refer back to the material presented here as needed.

2.1 GPS

The term GPS refers to a satellite-based navigation system established by the US military. A similar system (GLONASS) was established by the Russian military. Although both systems were originally restricted to military use only, both are now available to civilian users — albeit with reduced precision. Most modern cell phones and tablet devices can obtain localization information from both satellite systems and the term GPS has been broadened to refer to localization by satellite systems in general.

The basic approach in satellite-based localization is to have a group of satellites in known orbits transmit a common clock signal. As the satellites are at different positions relative to the receiver, the receiver obtains different clock times from different satellites. By knowing where each satellite is at any time, and comparing the clock times received at the receiver, it is possible to compute where the receiver must be. From the geometry, as soon as the receiver can obtain a signal from four or more satellites then the receiver can obtain three dimensional positional information along with time information. More satellites provide a more accurate positional fix. Satellites may not be 'visible' to a receiver as their signal may be blocked by trees, buildings and the like. Furthermore, GPS satellite coverage is not uniform over the earth, and certain remote areas have particularly poor GPS coverage.

Representing positions on the earth is typically given in terms of latitude and longitude. Latitude is the angular distance in degrees of a point north or south of the equator. So points on the equator have a latitude of zero. Points on the north pole have a latitude of 90 N, points on the south pole have a latitude of 90S. Longitude is the angular distance in degrees of a point east or west of Greenwich, England. This is known as the Prime Meridian. Points east or west of Greenwich have a longitude that is either east or west. Points east of Greenwich have positive longitude, while points west have a negative longitude. There is a line 180 degrees east or west of the Prime Meridian, known as the 180th Meridian where Longitude goes from east to west or from +180 degrees to -180 degrees. There are few land masses on the 180th Meridian, although Vanua Levu, is pretty close. Fractional degrees in latitude or longitude can either be given in terms of decimal degrees or in terms of minutes and seconds. Toronto is approximately 43.7 degrees north of the equator and 79.4 degrees west of Greenwich. Historically, GPS coordinates where given with the labels N, S and E, W. It is now common to have N and E considered as positive, with coordinates W and S considered as negative. So Toronto can be described as Latitude 43.7, Longitude -79.4. Also historically GPS measurements were given in degrees, minutes and seconds. It is more common today to use decimal degrees, and decimal degrees will be used exclusively here.

Given two GPS coordinates we can begin to ask questions such as ‘how far apart are two points’ or ‘how many meters are there in a degree of latitude or longitude. These are actually very difficult questions to answer in general. These are points on a sphere, and observe that the distance between two points with the same longitude goes from zero at the poles to roughly 111 km at the equator. Near Toronto, a degree of latitude is approximately 111km and a degree of longitude is approximately 80.6km. We can use these numbers to do approximate computations from differences in latitude and longitude (near Toronto) to motion east-west and north-south.

2.2 Geolocation in JavaScript

The Geolocation api in HTML5 is 'agnostic' in terms of the underlying technology. The system will use cell-tower signals, GPS and network-based location services depending on what is available to the underlying hardware. Furthermore, Geolocation is an 'opt-in' service. When a web page first requests to use Geolocation the user will be prompted to allow the software to have access to this information. One side-benefit of this is that you can test geolocation-based code in a regular browser on a laptop, although the accuracy may be poor.

In newer versions of the Android OS (e.g., from Nougat on) it is necessary to authorize an application to use localization services. So if you are using a later version of the Android OS — your own hardware, for example — you will need to go to the Settings 'app' and enable the 'Setting' permission.

JavaScript provides a number of different mechanisms to retrieve geolocation information. The JavaScript code snippet show below, for example,

```
var gps = document.getElementById("gps");
function getLocation() {
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(showPosition);
    } else {
        gps.innerHTML = "Geolocation is not supported";
    }
}
function showPosition(position) {
    gps.innerHTML = "Latitude: " + position.coords.latitude +
    " Longitude: " + position.coords.longitude;
}
```

will obtain a single measurement from the location system. If you run this JavaScript in your browser, your browser will prompt you for permission to return your location to the code. If you want your code to be called every time the estimate of your position changes, you can use the watchPosition method of the geolocation system.

This code snippet, for example

```
var gps = document.getElementById("gps");
function getLocation() {
    if (navigator.geolocation) {
        navigator.geolocation.watchPosition(showPosition);
    } else {
        gps.innerHTML = "Geolocation is not supported";
    }
}
function showPosition(position) {
    gps.innerHTML = "Latitude: " + position.coords.latitude +
    “ Longitude: " + position.coords.longitude;
}
```

will continually update the display as you move about the world. Unlike the earlier code snippet, the browser will only prompt the user **once** for position to access the geolocation system. Note: the html2apk Android program automatically grants geolocation permission to the web page executing within it, although see the observation about Nougat above.

The navigator.geolocation.watchPosition method actually returns an id value that refers to the watch process. When you are done with watching, you can terminate the process by calling navigator.geolocation.clearWatch(id) (where id is the value obtained from the navigator.geolocation.watchPosition call). This is important as the callbacks will continue while the application is running and the cost of keeping the geolocation system active can be quite expensive in terms of power consumption on the device.

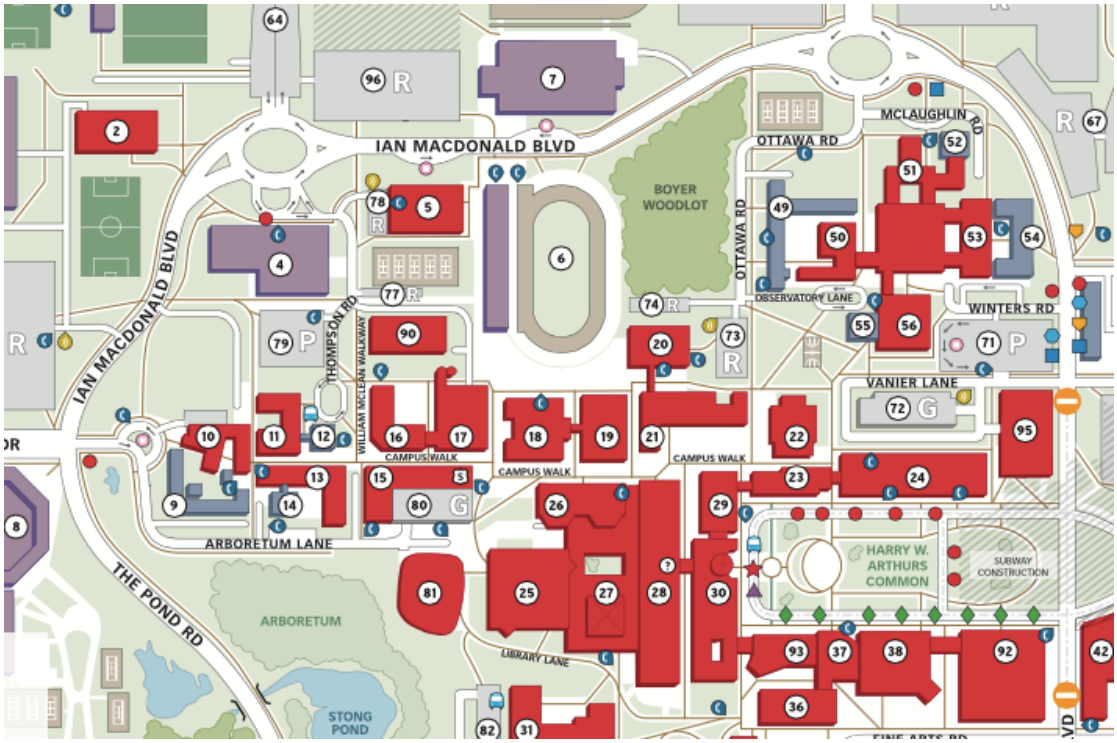
2.3 Creating a map

Maps can be a wonderful resource. They can tell you about things that are known, and things that are not known. Commercial maps are typically not just large images, rather they are represented as very high resolution ‘vector’ displays so that one can zoom in or out to almost arbitrary amounts, and different layers can be displayed or hidden depending on the application. A much more simple approach — and the approach taken in this lab — is to just treat the map as an image. If we follow this approach then we need to adopt some sort of standard for how to view

this map. Is the horizontal direction in the east-west direction or some other direction? How many pixels per meter in each direction? And so on.

So there is no "correct" answer here. But to provide a common answer to all of the students in the class, in this lab we make the following assumptions. The map will be represented as an image. The x direction in the image will correspond to east-west. The y direction of the image corresponds to north-south. The bottom left corner of the image will have some known GPS coordinate associated with it and that there is some known 'meters per pixel' which is equal in both the x and y direction. The upshot of this : if you have a gps coordinate (longitude, latitude) this corresponds to a particular pixel in the image and vice versa.

So how do we go about getting a map of our environment? Well, there are lots of ways (that include paying money to someone). Google provides maps (although there is a licensing agreement). Another option would be to edit your own. After all they are just images. There are many tools for creating and editing images. The CENTOS image provided in this course includes a copy of the 'gimp' image editing tool. You can use gimp to create images of almost anything (although it will not make an artist out of someone who has little artistic ability.). Gimp is free, and available for a wide range of platforms. By default, gimp does not save images in gif format, but it will import and export images in gif format. Full details on gimp can be found on line. But we can shorten the process a bit. Suppose we are interested in doing localization on campus. Fortunately the university has already built some rather nice maps of the campus that meet our needs and we can start with one of these. For example, the map shown below might be easily used as a first pass at a map of the university campus.



In order to simplify slightly the scope of this lab you will want a map that is rendered at its native resolution on your Android device. So in obtaining an image map of that portion of the world that is of interest to you, you will want an image that fits nicely on your Android device. For the devices in the lab, that will be something around 1000 pixels wide (max) by about 500 pixels high (max). It would be relatively straightforward to relax this constraint, and that is left as an optional assignment at the end of the lab.

2.4 Representing structures in JSON

It is often necessary for a program to represent complex information that is either static (never changes) or is initialized to some static value. One approach to addressing this problem is to develop a standard language for representing such structures. In JavaScript the JSON (JavaScript Object Notation) language often turns out to be a good choice for such structures. JSON is yet another language for representing things much as XML, HTML and CSS represent things. Each such representation has its own plusses and minuses. One particular advantage of JSON is that it is straightforward to take a JSON representation of information and

convert it into JavaScript and then to access that representation. An example will perhaps make this more clear.

Suppose that you wanted to represent a collection of named objects with latitude and longitude. Each object might be represented as the dictionary

```
{latitude : 123.0, longitude : 456.0, description : "Hot dog cart"}
```

Given an instance of an object

```
var v = {x : 2, y : 3, z : "ha ha"};
```

individual elements can be accessed as `v.x`, `v.y`, and `v.z`

A collection (an array) of such objects can also be constructed in JavaScript. For example,

```
var a = new Array();
a[0] = {x : 2, y : 3, z : "ha ha"};
a[1] = {x : 3, y : 5, z: "blue"};
```

allows one to talk about the x field, of the it's element of a as `a[i].x`.

2.5 Geocaching

Geocaching is a recreational pastime in which people secrete objects at different locations in the world identified by their GPS coordinates and others try to go and find them. There are large numbers of such cache's worldwide, and many in the Toronto area.

3. Exercises

As with other labs in this course, lab exercises are broken down into three sections, A, B and C. Exercises in section A are Pre-Lab exercises. All Pre-lab exercises **must** be completed prior to attending your lab. You will not be allowed to participate in the lab if you have not completed these exercised prior to attending the lab. Furthermore, you will not receive course credit for a lab for which the Part A lab was not completed. You will also get much more out of each laboratory if you spend some time going through the B and C exercises for each laboratory before attending your laboratory session.

A. Pre-lab

1. Download eecs1012gps.zip from the jr web page and unzip it. This is a simple web page that monitors the orientation of the device and updates a text display based on this. Examine the html and JavaScript provided. If you have access to an Android device, download and install the application on your Android device. Note: leaving the GPS sensor turned on has a significant effect on battery life. Review the code, what does pushing the button on the web page do the first time you push it? What does it do the second?
2. Complete the pre-lab quiz on the course moodle page.

B. In-lab

1. Obtain a lab kit from the lab monitor. Power up the laptop including the virtual CENTOS device. Power up the Android device. Ensure both are connected to the same network and both are plugged in for power. During the lab you will be walking around campus with the Android device running. Having a fully charged device will make things somewhat easier. Note: in this lab it may be easier to have your tablet locked to either portrait or landscape mode as you move it around.

2. In this lab you are going to write software to allow you to play a geocaching-like game on the university campus. (The code should actually work anywhere, but given that we are on the campus, that is what will be assumed. If you are working on the lab off-campus, feel free to change the location as appropriate.) The first thing that you are going to need is an image of a map. Capture an image of a map of a portion of the campus. In particular, you need to ensure that the portion of the campus that you will use includes the area around the William Small centre, and in particular on the south side of the William Small building. Starting with the map on the university web site would certainly work as would any other map that you can find. Or you can drawn your own. The image should be no larger than about 1000 pixels wide and around 500 pixels high. This map should represent at least a few hundred meters.
3. Download eecs1012gps.zip from the jr web site. this application contains a very simple GPS program written in HTML and JavaScript. Test it in the CENTOS virtual environment (this will be very boring as the CENTOS does not have particularly good localization capabilities). Test it in your host environment as well. Most host environments will provide some localization information through the known location of the wired ethernet or wifi network. Finally test it on the Android platform. Take the Android device outdoors and walk around with it. (Remember to unplug it from power before doing this.) Wander around to observe the gps location changes as you move. You need to obtain the GPS coordinates for two easily identifiable points on the map (ideally some distance apart). That is you need to obtain (gps_longitude, gps_latitude) and (u,v) coordinates for two points on the map. Try to make these points as far apart in terms of both longitude and latitude as is possible given your map and the realities of York campus. Given that Toronto is in the northern hemisphere, good GPS coverage is more likely on the south sides of buildings that have good views of the southern sky. You will not get very good GPS coverage to the north of William Small. These two locations will serve as the basis of the calibration of your map. Once you have recorded these two locations, record three more locations (longitude, latitude, u, v) for the location of three geocaches that you will also represent in your application. Given these cache's names so you can remember these locations. Record the process you followed

here so that you can record this information in your ePortfolio. Where were the best and worst places that you found for GPS reception?

4. Now head back to the lab. You are going to modify the application that you downloaded in the eecs1012gps.zip archive earlier to build your geocaching application. Copy these files into the GPS directory under server within the virtual box. Also download your map file (obtained during step 2) to the GPS directory in the virtual box. Modify the application so that above the information already being displayed in the application there is a element with the id of "map". Within this there should be an <img> element that refers to your map image the image is shown above the button and display provided in the code given in the eecs1012gps.zip. Give the image element in the web page the id "map". Also within the "map" element create two new elements with id's "target" and "me". Each of "target" and "me" should contain an <img> tag whose source references the symbol that will be used for the target (the next geocache to search for), and for the user.

5. Use gimp or some other editing tool to create two icons to represent the location of the user on the map and the location of the target on the map. Use a gif file format so you can represent semi-transparent objects. These files should be around 32 pixels square (or smaller). Move these files into the GPS directory and change the HTML code so that they are referred to by the <img> tags within the "target" and "me" divisions.

6. Create an external CSS file (gps.css) for this application and have the HTML code refer to it. Within the CSS file, create a CSS rule for the id "map" that makes "position : relative" and CSS rules for target and me that makes "position : absolute" and that gives values of 0px to "top" and "left". View the resulting web page. Where are the target and me symbols displayed now?

7. You now need to write code that given a GPS coordinate determines the corresponding location in pixels on the map. To do this observe that if you assume that the number of pixels were degree of longitude and latitude are constant over the campus — and that's probably a good approximation — then the u coordinate in the image, given a gps longitude (or latitude) its position on your map is given by u_1 + (u_2 - u_1) * (gps_i -gps_1) / (gps_2 - gps_1), where

gps_1 appears at u_1 on the map and gps_2 appears at u2 on the map. Note that the resulting u value is linearly interpolated between u_1 ag gps_1 and u_2 at gps_2. Write a function interpolate(gps1, gps2, u1, u2, gps) that performs this interpolation. Use the values you measured above for the values of gps1, gps2, u1 and u2 so that your code 'works' with your map. The function should return the u value that corresponds to gps. Call this function twice, once for each gps. Modify the html display so that it has a element with id "debug", and output the interpolated (u,v) values for the latitude and longitude. Take your Android device back outside and make sure that the (u,v) values that you obtain are correct for the positions you used to calibrate the map. Were they. Why might there be some discrepancy in the position>

8. In the pitch and tilt lab you updated the position of a ball on the screen based on the pitch and tilt of the Android device. Use a similar approach to change the position of 'me' on the screen so that it moves with changes in your GPS coordinates.

9. Record the operation of your GPS sensor as you walk around the campus. Show that it updates your position as you move and that it more or less shows your current location. You may find it useful to make short videos of this for your ePortfolio.

10. Now you are going to enhance your application so that it also displays the current geocache on the map. Create (at least) three global JavaScript objects that correspond to the geocache locations. Call these loc1, loc2, and loc3. Give each object elements 'lat', 'lon', and 'desc' corresponding to the latitude, longitude and description of each object.

11. Create a global JavaScript Array object called caches. Put loc1 in caches[0], loc2 in caches[1] and loc3 in caches[2]. Create a global variable currentCache and initialize it to zero.

12. Create a function updateCache() that takes no arguments. This function should add one to currentCache that adds one to currentCache modulo the number of cache locations you put in the caches array (three in the example above). After you update the value of currentCache, invoke the function showCache() which you will write in a moment.

13. Write a function showCache(). This function access the currentCache element in the array caches to find the cache that is to be drawn now. You should be able to write this function based on the function you used to draw 'me' on the screen.
14. Add a button to the HTML (call it next geocache) that when pushed invokes updateCache(). Now whenever you push the button, the next geocache will be selected and the screen updated.
15. Go out and test your application in the real world.
16. Record your experiences in your ePortfolio. Include your code along with videos of the system working. In meters, how accurate do you think the Android GPS system that you have built is? Is it comparable to the location obtained with the Google Map application? Why or why not?

C. Advanced

1. Add a on the page so that you can display the description of the current geocache whenever it changes. Update the showCache() function so that it updates this
2. Refine your interpolation code so that it works if the image is scaled. Note: you can access the real dimensions of an image through naturalWidth and naturalHeight. Once this is working, zoom the map so that it is the width of the Android device.
3. Add ten (or so) geocaches to the display

4. Further Reading

- There are many on-line resources concerning GPS including this.
- A comparison of GLONASS and GPS can be found here.
- Examples of geolocation in JavaScript can be found here.

5. Credits

The image on the front cover is from Kathmandu, Nepal.