N96 Localization via context

Abstract

Location- aware application such as MobiLocate utilises physical location of the user with variety of different social applications. This report details the design and development of the MobiLocate application that uses context-data from different sensors to pin-point the location of the user. The application focuses on using only contextual data from WLAN signal strength in one particular building to find the location of the user. Evaluation of the application suggests that the accuracy of the results depends on heavily on the positioning algorithm used as well as the number of different sensors used for context data. Further improvements are suggested to improve the application and also bring it to public domain.

Table of Contents

List of Illustrations

Glossary

API – Application Programming Interface	
GPS - Global Positioning System	
JSON – JavaScript Object Notation	
MAC – Media Access Control	
RSSI – Received Signal Strength Indicator	
SIS - Symbian Installation System	
SQL - Structured Query Language	
UI – User Interface	
OS – Operating System	
WLAN – Wireless Local Area Network	

1. INTRODUCTION

This project investigates the use of contextual information to locate a user with a mobile phone in the absence of Global Positioning System (GPS) signals.

My objectives during this project are to design a context-aware application for the Nokia N96 that collects context data such as the lighting level, noise level, wireless signal strength and accelerometer readings as well as the last-known GPS signal, calibrates and displays the possible location of the user on the device. This is useful in indoor locations where GPS signals are inaccessible.

Context aware applications employ localisation-based resources in a variety of social scenarios. Examples include Micro-Blog [1], AAMPL [2], SenSay [3] and SATIRE [4]. Each of these applications utilise only one particular context data to localise the user. This project is designed to be extensible and additional context information can be easily incorporated in the future.

This project involves designing, implementing, testing and documenting the application framework as well as the procedures needed to successfully build the application.

The main scope of this report is to establish a suitable application framework to estimate the position using the Received Signal Strength Indicator (RSSI) from the UBC WLAN (Wireless Local Area Network) Access points.

This report divides into the following primary sections including architecture and design, system implementation, results and limitations, and conclusion.

2. ARCHITECTURE AND DESIGN

MobiLocate uses client-server architecture as shown in Figure 1. The client is responsible for collecting data and also interacting with the user. The server is responsible for storing and manipulating the data. The following sections describe the architecture more in detail.

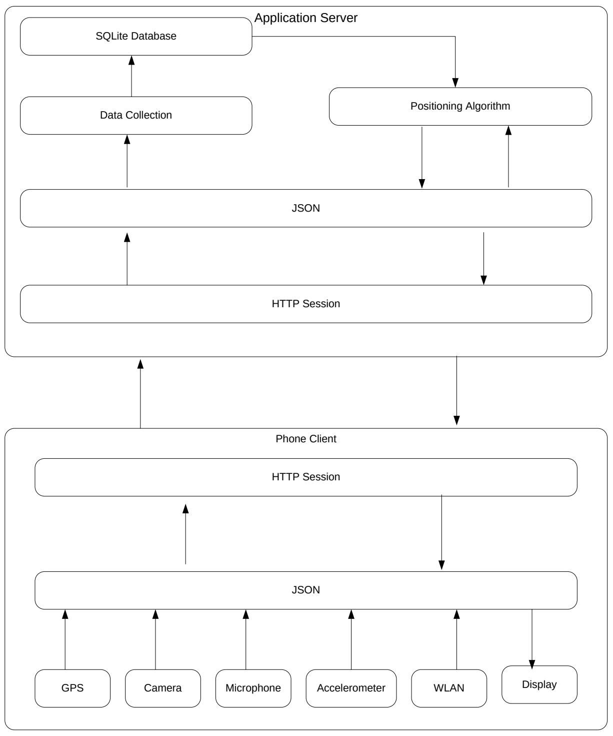

Figure 1 : Block diagram of the MobiLocate architecture

2.1. Client Server Architecture

The MobiLocate client runs on a Nokia N96 mobile phone and communicates with the server using a WLAN connection. The client collects and transmits the context data to the server application, and the server application approximates the location and sends this information back to the phone client to be displayed.

The phone client collects contextual data such as lighting level, noise level, wireless signal strength, accelerometer readings and the last-known GPS signal, from the camera, microphone, WLAN, accelerometer and GPS sensors respectively. The data is then serialized and uploaded to the server using HTTP POST request. JavaScript Object Notation (JSON), a lightweight data interchange format independent of programming languages, is used for this serialization. The location in JSON format received from the server is decoded to a format that is recognisable by the phone and is displayed to the user.

The MobiLocate application server consists of a web application to serve requests from the phone client and a SQL database server to store the samples. The web application consists of several server-side scripts, which perform the various tasks as follows. A static image map of the location is obtained from the Google maps API (Application Programming Interface) using the last known GPS coordinates and sent to the phone client; the received POST data in JSON format is first decoded to a format that is recognisable by the web application; for data collection, the decoded data is stored in the database; for position approximation, the decoded data is retrieved from the database, used in performing location calculations and result is encoded in JSON format and sent back to the phone client; the SQL database server contains tables corresponding to each context-data from the sensors; and the server-side scripts use SQL query commands to perform 'insert, update or select' operations on the database.

2.2. Data Collection Setup

This application relies on previously collected data to predict the current location information of the user. Hence, the accuracy is greatly dependent on the amount of data set available for processing. The error margin will be reduced if the data set is relatively high. The samples collected are categorized into ‘buildings, floors and sections’. For instance, each floor in a building is divided into several sections. A manual scan is initiated at each section using the MobiLocate phone client to obtain the context data from the different available sensors. The samples gathered are tagged with the unique ‘building, floor and section’ information and sent to the server. The MobiLocate application server then stores the gathered data in the database.

2.3. Position Approximation

There are a number of position approximation algorithms that can be used, depending on the different contextual information selected to predict the location, including Bayesian classifier [4], k-nearest classifier [4] and using a list of known WLAN access points [5].

This report concentrates on a modified version of the third algorithm. The following describes the four general steps needed for a position calculation: (1) WLAN scan; (2) identifying the WLAN access points of the building’s WLAN list; (3) counting and weighting the detected access points of the floors and sections; (4) identifying the most detected and highest weighted floor and section, which represents the calculated position. The weighting of the detected building sections and floors is based on the RSSI values [5].

3. SYSTEM IMPLEMENTATION

For this report, the application framework only focuses on calculating the position using WLAN signal strength. This can be easily extended to use context-data from other additional sensors. The following sections describe the phone client and the web application.

3.1. Phone Client

The MobiLocate phone client is implemented on a Nokia N96 mobile phone device, which is based on Series 60 (S60) 3rd Edition, Feature Pack 2 platform, with Symbian OS (Operating System) version 9.3. The application is written in Python for S60 version 1.4.5.

An additional open-source WLAN Scanning module for Python S60 is installed to scan the WLAN access points available in the vicinity to obtain their MAC address and the RSSI of the values. A JSON module for Python S60 is also installed to encode and decode JSON objects.

Ensymble developer utilities are used to convert the Python source code into Symbian Installation System (SIS) package. Open Signed online certification [6] tool provided by Symbian is used to sign the packages. This is necessary in order to access the system functions, such as WLAN and GPS in the device.

The source code for the phone client is contained in the Python module MobiLocate.py (see Appendix A). This module accesses the standard Python module as well as the S60 specific modules. It is divided into three different components, namely, application user interface (UI) component, sensor component and communication component.

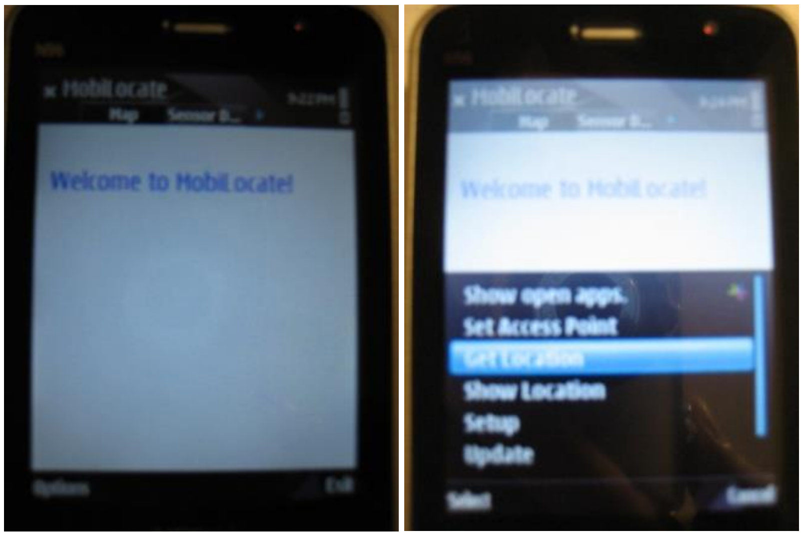

Figure 2 : Welcome Screen and Menu

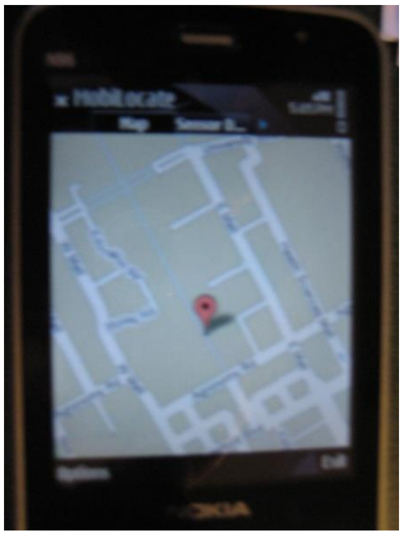

Figure 3: Google Static Map Image

The application UI consists of navigation tabs, menus and forms. The two navigation tabs are described as follows. The map tab displays the location map image of the building obtained from Google Maps API, and the sensor Data tab lists the MAC address of the access points together with its RSSI value. Additional tabs to display other contextual information can also be added. The six menu items are described as follows. The left softkey activates the application menu. The ‘Get Location’ menu item is used to find and display the current location information of the user. The ‘Setup’ menu item opens the setup data form. The ‘Update’ menu item opens the update data form. The ‘Show Location’ menu item displays the previously retrieved location information in a form. The three forms are described as follows. The setup data form and the update data form are editable forms that get the user input of the location information and pass the data to the communication component. The location information form just displays stored location information.

The sensor component uses the WLAN scanning module to extract and add to a list the MAC addresses and the corresponding RSSI values of the access points. The list is either displayed or sent to the server by the communication component.

The communication component is responsible for sending to and receiving data from the server application. It uses the JSON module to decode and encode the data and uses the HTTP module to connect to the server for uploading or downloading data.

3.2. Web Application

The web application is hosted on an Ubuntu Linux 8.10 running Apache Webserver 2.2.9 with PHP 5.2.6. A SQLite 3 embedded database is used to store the data. The server side scripts are implemented in PHP, a server-side scripting language. PHP has the built-in functions to access the SQLite database file and also to encode and decode JSON objects.

The server-side scripting consists of a database component, image map retrieval component, setup and update data component, and positioning component.

The database component is divided into SQL query and the tables. The server-script has functions with SQL query commands that create the tables, insert values into the tables, select values from the table and update values in the table. The database consists of three tables, specifically, MAC address (BSSID), RSSI and location information (AREA), where their values are classified with a unique identifier (ID) as shown in Figure 4. Additional tables can be added for other sensor data (see Appendix B).

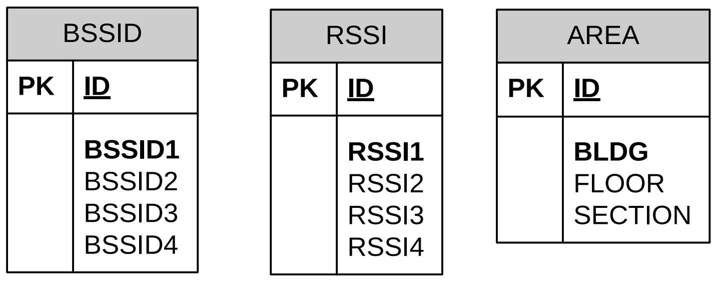

Figure 4 : Database Tables

The image map retrieval component uses the latitude and longitude values from the HTTP POST data and fetches the static image of the map illustrating the building location.

The setup and update data component is responsible for data collection. For this report, the data collection is mainly focused in Kasier and Macleod buildings in UBC. Both buildings have four floors and each floor is divided into five different sections. The MAC addresses and corresponding RSSI values of the UBC WLAN access points, with the maximum of four values, are manually tagged with their respective building, floor and section information.

The positioning component does the actual position calculation using the following algorithm specified. To find the current location, a scan is initiated and the scanned MAC address is compared with the MAC addresses stored in the database to obtain the list of the unique identifiers. The highest weighted unique identifier is selected from this list. The RSSI values with this unique identifier are checked against the RSSI values from the scan. If this is within a margin of ±10, the location information with this unique identifier is returned to the user (See Appendix C).

4. RESULTS AND LIMITATIONS

The project achieved the goal set forth that is building a suitable application framework for position calculation using different contextual data, while the application framework's integrity has not been fully tested with extensive data collection. Conversely, it has performed as well as expected.

This application's accuracy and prediction is limited by the number of the different sensor data available and also the amount of data set used in the calculation. Moreover, the device and the software API's functions limit the capabilities of the application framework.

The N96 device, based on Symbian OS 9.3, is relatively new and it uses a new sensor framework to access the accelerometer readings and also has upgraded security permissions. On the other hand, the Python S60 version 1.4.5 is built for Symbian OS 9.2 and hence many of the features of the N96 device are not very well supported, including the sensor framework. The MobiLocate phone client could be easily upgraded to run on the next version of Python S60 2.0 for Symbian OS 9.3. Though, the Python S60 2.0 supports the sensor framework and should be used to access the accelerometer readings, it is still under development.

The list of known WLAN access points that are installed in the buildings is not available and the WLAN scanning is very erratic. The number of access points available and the corresponding

RSSI values in one location varies considerably each time. Therefore, this makes it harder to calculate the position accurately and using only WLAN signal strength would not give an accurate result.

The positioning algorithm used in this algorithm is not reliable, and hence, further work and research has to be carried out to find a suitable algorithm for use with different contextual data.

Since this is a mobile application, energy consumption is one of the main concerns and sending data over the WLAN consumes a lot of resources. Therefore, a lightweight interchange format has to be used.

5. CHALLENGES AND FUTURE WORK

There were many challenges faced when building this application. Building applications for mobile platform presents its own unique constrains including certifying the application with proper permissions, learning and using the device specific APIs and debugging and testing the application on the device specifically. These constraints consumed more time than the actual development. The client and server applications are written in different programming languages therefore considerable amount of time is spent in choosing the right format to transfer data properly between the client and the server. A dedicated server also has to be setup with proper access permissions to serve the phone client. Although Symbian C++ offers more functionality, developing an application with it is time-consuming and requires in-depth knowledge of the underlying platform; therefore Python for S60 has to be chosen for reduced development time

With the convergence of mobile devices with rich set of features and the Internet, lot of possibilities to build social applications for mobile platform has emerged. MobiLocate application being one of them could be developed into a platform to support various social applications such as photo blogging directly from a phone, monitoring and locating criminals, children and seniors and posting information about public grievances and crimes. Also, privacy concerns have to be adequately managed for a successful implementation.

6. CONCLUSION

This report investigated the use of contextual information to locate a user with a mobile phone in the absence of Global Positioning System (GPS) signals by building a useful application framework using only contextual data from WLAN signal strength sensor without worrying about the algorithm used for the position calculation. This was executed by designing a context-aware application using contextual information such as the lighting level, noise level, wireless signal strength and accelerometer readings and the last-known GPS signal to locate a user with a mobile phone (Nokia N96) in the absence of GPS signals, particularly useful when indoors, by estimating the position using the Received Signal Strength Indicator (RSSI) from the UBC WLAN (Wireless Local Area Network) Access points.

Though the project had given a satisfactory performance, the result was found unable to produce accurate predictions of the location. This is mainly because the application framework's integrity had not been fully tested with extensive data collection and is limited by the number of the different sensor data available and also the amount of data set used in the calculation. In addition to software constraints, the unavailability of the list of known WLAN access points that are installed in the buildings and the erratic nature of the WLAN scanning had contributed to inaccurate results.

Hence, it is suggested that further work and research has to be carried out to find a suitable algorithm for use with different contextual data, and it is necessary to allow time for the adequate technology to be developed for practical usage of the feature, while there is also a recommendation for a lightweight interchange format to be used in order to be energy efficient as it is a major concern when sending data over the WLAN.

7. REFERENCES

[1] G.Shravan, J Li, R R Choudhury,L. Cox, and A. Schmidt, “Micro-blog: Sharing and querying content through mobile phones and social participation,” in ACM MobiSys, 2008.

[2]A.Ofstad, E.Nicholas, R.Szcodronski and R.R.Choudhury, “AAMPL: Accelerometer Augmented Mobile Phone Localization,” in ACM Mobile entity localization, 2008

[3] D. Siewiorek, et al., “SenSay: A Context-Aware Mobile Phone,” in IEEE International Symposium on Wearable Computers, 2003.

[4] R. Ganti, et al. “SATIRE: A Software Architecture for Smart AtTIRE,” in Proceedings of ACM Mobicom, 2003.

[5] M.Hermersdorf, “ Indoor Positioning with a WLAN Access Point List on a Mobile Device ,“ WSW SenSyS, 2006.

[6] Symbian Signed. <https://www.symbiansigned.com>.

APPENDIX A

MobiLocate.py

APPENDIX B

db_sql.php

APPENDIX C

loc_data.php

APPENDIX D

image_map.php