EECE 496

ENGINEERING PROJECT

Final Report

Nokia N810: Localization via Context

Date Submitted: April 8th, 2009

Name: Kattie Mun Mun Tay

Student Number: 91034041

Technical Supervisor: Dr. Sathish Gopalakrishnan

Google Code website: http://code.google.com/p/eece496-ktay/

ABSTRACT

This report describes the work that was completed on the “Nokia N810: Localization via Context” project. The “Nokia N810: Localization via Context” project involves the development of a mobile application for the Nokia N810 Internet Tablet that allows users to determine their location inside a particular building with the help of contextual information such as light intensity and wireless signal strength. This report will outline the work done at each stage of the application development process which includes setup of the development environment, design and architecture of the application and finally implementation and testing of the application. The report will also mention the technical challenges faced during the term of the project and recommendations of improvement on the application.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

GUI	Graphical User Interface
GPS	Global Positioning System
IDE	Integrated Development Environment
SSH	Secure Shell
SCP	Secure Copy Protocol
WLAN	Wireless Local Area Network
ESSID	Extended Service Set Identifier
MAC	Media Access Control
SQL	Structured Query Language
ER	Entity-Relationship
MSE	Mean Squared Error
API	Application Programming Interface
PHP	Hypertext PreProcessor

1.0 INTRODUCTION

This project involves the development of a mobile application for the Nokia N810 Internet Tablet whose purpose is to use contextual information from the surrounding environment to locate the user of that device in the absence of GPS signals. This report represents a documentation of the entire application development process and can be used as a reference by future students wishing to work on similar projects or who are interested in expanding on this project.

My objectives for this project are to design and develop an application which will allow users of the Nokia N810 device to accurately determine their location in a specific building based on the last saved GPS location, light intensity, and wireless signal strengths of Internet access points in the building. The application will interface with a database used for the storage and retrieval of contextual data. Users will also be able to easily add, modify, retrieve and delete data through the application's user interface.

Such an application is significant for users who wish to pinpoint their current location in a particular building but are unable to do so due to the lack of GPS signals. This is because GPS data are obtained via satellites and thus GPS signals tend to be weak or non-existent in an indoor environment. Also, GPS data only provides latitude and longitude values of a location but this is not sufficient to determine a user's specific position inside a particular building. The application can also be used to keep track of the location of users within a building.

As part of this project, I individually designed and implemented an application for the Nokia N810 device which retrieves the last saved GPS data on the device and uses it together with other information such as light intensity and wireless signal strength to estimate the location of the device user within a specified building. The application consists mainly of a GUI that allows for interaction between user and application and a disk-based database that runs on the back-end which allows for the storage and retrieval

of data. Testing was performed in parallel throughout the entire process to ensure that all functionalities of the application were working as desired.

The scope of this project is limited by the device used itself which is the Nokia N810 Internet Tablet that runs on a Linux-based Maemo platform. This is because the application development is restricted by the availability of software libraries and packages for mobile devices which are generally less in order to reduce the disk space required on such devices. Before starting on the project, some other factors such as development environment and development language to use were also considered. For this project, I have chosen to develop my application using the Python programming language within the Eclipse IDE on a Linux Virtual Machine.

In my report, I will be elaborating on the development environment used in this project, features, design, architecture, implementation and testing of my application, as well as challenges faced during the project and further improvements that can be done on the project.

2.0 DEVELOPMENT ENVIRONMENT

The choices made at the beginning of the project regarding the development environment and language is extremely crucial as any wrong choices could restrict or hinder the progress of the project. Having to switch languages or environment and start from scratch halfway through the project is definitely not ideal and should be avoided. Therefore, before starting actual development, I first did some research to determine exactly what I would need for the purpose of my application. Some factors taken into consideration were the availability of support libraries and online resources, ease of GUI development and familiarity with the programming language itself.

2.1 Overview

The Nokia N810 Internet Tablet runs on the Linux-based Maemo platform. In order to develop my application, I ran a Linux Virtual Machine using VMWare Player on my existing Windows XP system. The programming language that I chose to use was Python. Within VMWare Player, I used the Eclipse IDE along with the PyDev plug-in for the purpose of code development.

The reason I chose to use Python as the programming language for my application was because of the widely available and established support libraries for Python on Maemo. Although the C programming language is the official development language for Maemo, I found it to be more low-level and less flexible compared to Python. I also found Python to be much easier for developing GUI based applications. My ideal programming language would have been Java; however, the Jalimo project which provides support for Java on Maemo is relatively new and thus has a limited number of resources available.

To simplify the setup of the development environment, I used the Maemo SDK VMWare image which is readily available on the Maemo website. This image is run using

VMWare Player which can be downloaded for free and provides a complete environment for Maemo applications programming. The Maemo SDK image also contains Ubuntu, Scratchbox, Maemo root straps, ESBox, and other programming tools that are useful for the development of Maemo applications.

The Eclipse IDE was used mainly because of my familiarity and knowledge of it. In order to develop Python based files, the PyDev plug-in for Eclipse was also required. Besides that, the Subclipse plug-in which provides support for Subversion within Eclipse was used to synchronize my code with the Google Code project page that I had created for this project. With this feature, I could easily upload my code online in addition to keeping track of each change that was made throughout the project.

The Maemo SDK comes with Scratchbox, a cross-compilation toolkit that also contains an emulator for the Maemo platform which allowed applications to be developed and tested on the PC before being transferred to the device. However, my application required the use of the device's hardware such as the GPS and light sensor which meant that my testing had to be done on the device itself. For this purpose, I originally used Pluthon, an Eclipse plug-in which made use of an SSH connection to allow the application to be developed and run directly on the Nokia N810 device. Even though useful, I found it very slow and decided instead to write my code in Eclipse and then manually transfer my files from PC to device using the command line SCP program. I could then test my application by running the files directly using the terminal emulator, xterm, on the Nokia N810 device.

The instruction set for setting up the development environment described above can be found in Appendix A of this report.

3.0 APPLICATION OVERVIEW

The application that I have written for the Nokia N810 device is currently tentatively named as ***LocateME!***. As suggested by the name, the purpose of ***LocateME!*** is to allow users of the device to determine their current location in a specified building where GPS signals are not available. GPS signals are transmitted from satellites and are generally undetectable in an indoor environment. The application makes use of the last saved GPS data to find the last location of the user and with this information indicate the building that the user is mostly likely to be in.

The ***LocateME!*** application then performs a scan for Internet access points and light intensity data at the user's current position and compares it with existing data stored in the database in order to estimate the location of the user. Users are also allowed to add, edit, and delete location and scan point data.

3.1 Features and Functionalities

The ***LocateME!*** application currently contains the following features and functionalities:

1. Determination of device user's current location
    - The application identifies the building the user is located in and makes an estimation of the position of the user in the building
    - This estimation is made based on information such as last saved GPS data, wireless signal strength of Internet access points and light intensity

2. User-friendly graphical user interface
    - The application is packaged into an easy to use graphical user interface that emphasizes on usability and enhancing the user experience

3. On-disk database for internal storage and retrieval of data
    - The application comes equipped with an on-disk database that runs on the back end of the program
    - The database serves as a storage area for all collected contextual data that can be retrieved whenever the data is needed
    - The database will be empty the very first time the application is loaded and users will be able to customize the application and define locations according to their needs

4. Automatic connection of device to Internet at start up
    - The application checks for an Internet connection when it starts running and attempts to connect to the default saved connection if not connected
    - If there are no saved connections, the Connection Wizard window opens to allow the user to select a connection

5. Acquirement of last saved GPS data
    - The application is able to retrieve the latitude and longitude of the last known GPS location from the device's built-in GPS module

6. Collection of wireless data from Internet access points in range
    - The application scans for available Internet access points and extracts wireless signal strength information

7. Light intensity detection
    - The application makes use of the device's built-in ambient light sensor to detect the light intensity of the surrounding environment

8. Addition, modification and deletion of data by user
    - Users are able to easily insert, edit and delete contextual data through the application's GUI

9. Display of device user’s location on Google Static Maps
    - The application uses the latitude and longitude values obtained from the GPS module and maps the location on Google Static Maps

4.0 SYSTEM DESIGN AND ARCHITECTURE

The application comprises of four major components: the graphical user interface, on-disk database, algorithm to determine location of user, and data acquisition modules. The majority of the time spent on the design process was in thinking of how the different components would interface with each other and the interaction between the application and its user.

4.1 Overview

The figure below depicts how the different components of the application interface with each other. The following sub-sections will describe the design and architecture of each component in detail.

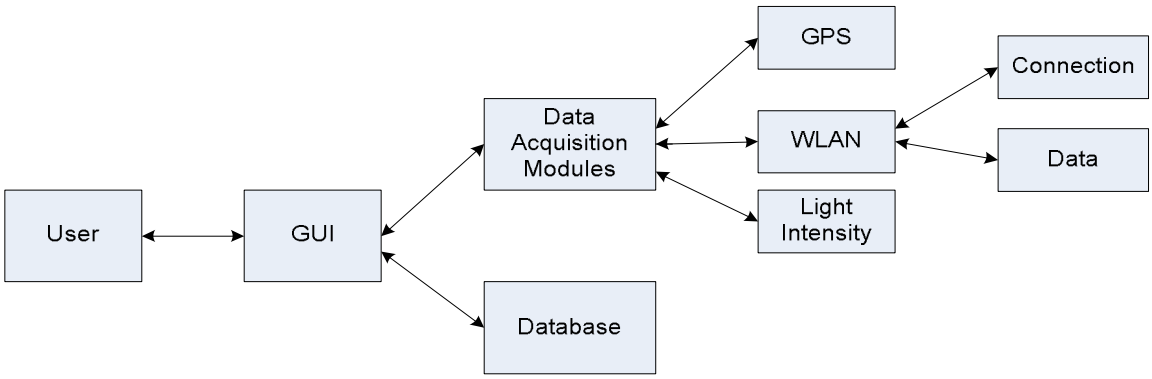

Figure 1 Overview of System Architecture

4.2 Graphical User Interface

The application's GUI layout was first designed by hand and then implemented using the Python packages, PyGTK and Pango. The main goal of the GUI design was to ensure that users of the application would be able to easily access and modify data as needed. The layout also had to be user-friendly, easily understandable and pleasant to the eye.

The GUI is divided into two windows. The first is the main window displayed at the start of the application. The second is a window showing results from the user's request to find his/her location.

4.2.1 *Main Screen* Window

The ***Main Screen*** window is displayed on start-up of the application and is shown in the figure below. At initialization of the application, the latitude and longitude values of the last GPS location are retrieved from the GPS module. The database is searched for these values and if a match is found, the corresponding location name is returned and displayed. If the values do not exist in the database, the location is displayed as ***Unknown***.

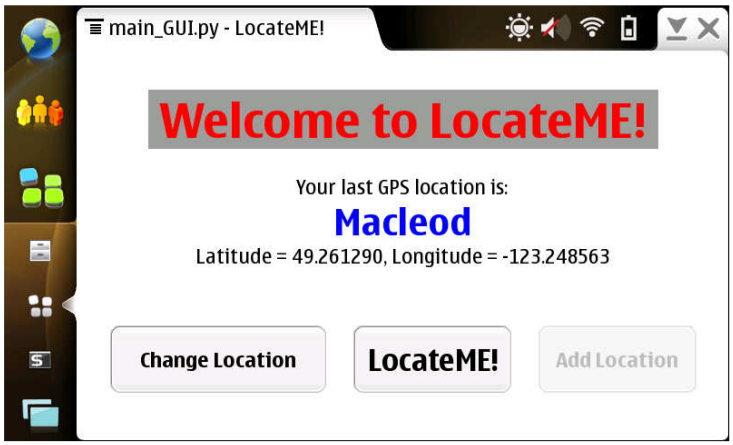

Figure 2 *Main Screen* Window

The following is a list of tasks that can be performed within the ***Main Screen*** window:

- Add New Location
- Change Location
- Edit Location Data
- Locate User

4.2.1.1 Add New Location

The ***Add New Location*** option allows users to add a new location to the application's database. Users can access this function by clicking on the ***Add Location*** button in the ***Main Screen*** window or from the application's menu as shown in Figure 3 below.

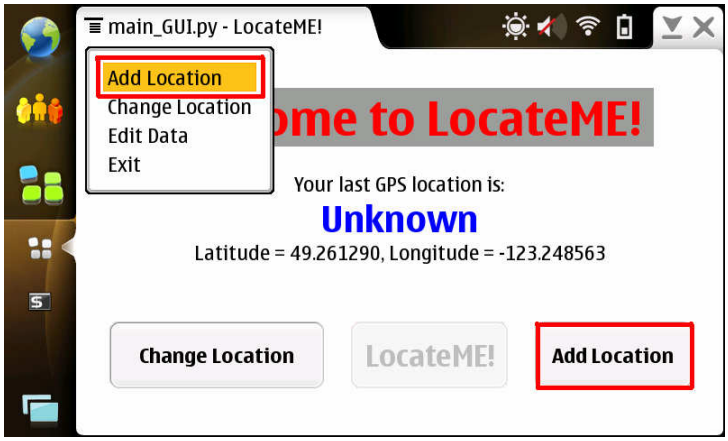

Figure 3 Methods of Accessing *Add New Location* Option

A check is first performed to ensure that the device is connected to the Internet. If no Internet connection was detected, a pop-up error message, shown in Figure 4, is displayed requesting the user to connect to the Internet.

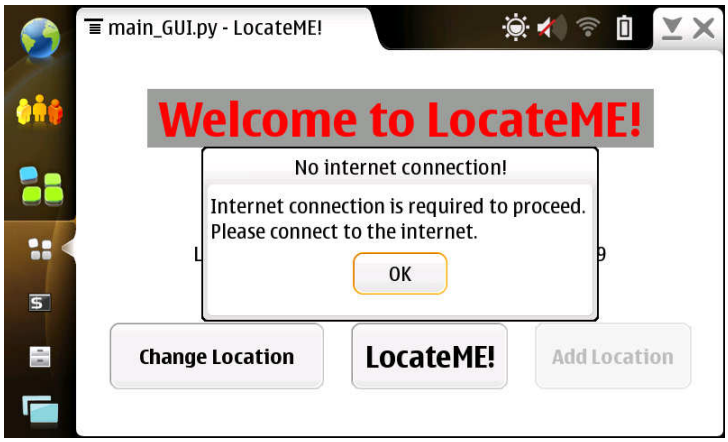

Figure 4 *No Internet Connection* Error Message

The ***Add New Location*** window is shown below and it displays the geocode of the unknown location and ESSID of the current WLAN connection. Users can choose to

enter a new location name or select from a list of existing locations by clicking on the ***Browse*** button. The concept behind this is that a building location could have multiple entrances, each with a different set of latitude and longitude values. By letting the user add more than one geocode pair to a location, the application would be able to detect the user's location regardless of which entrance the user last came in from. Clicking on the ***Add*** button will store the location together with the geocode and ESSID values in the database.

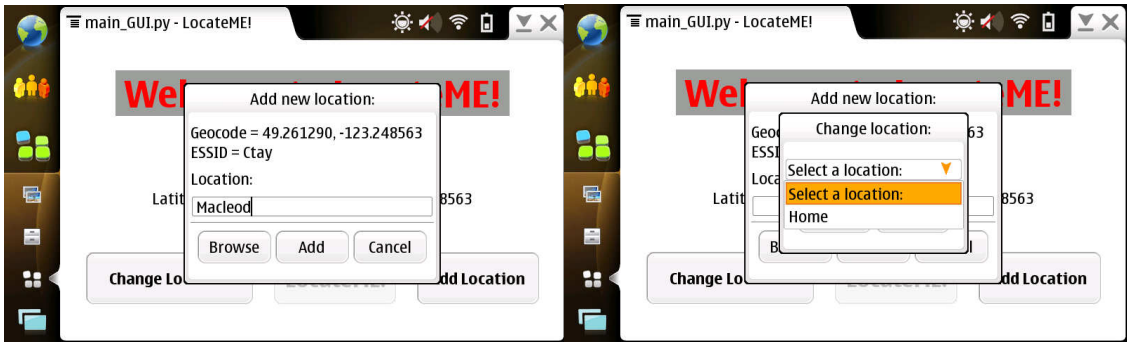

Figure 5 *Add New Location* Window

The ***Add New Location*** option is only activated when the last known GPS location on the device does not match any of the values in the database. If a match was found, the user will not be allowed to access the ***Add New Location*** function. The reason for this is to ensure that each location is tied to a geocode value that enables the application to easily determine the location's name based on this value in the future.

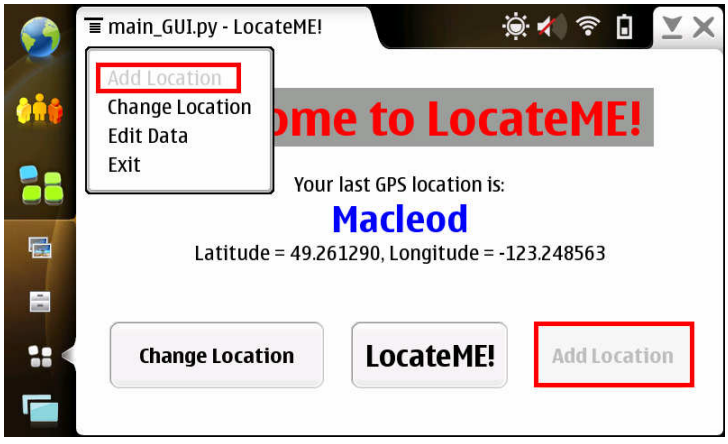

Figure 6 Deactivated *Add Location* Buttons

4.2.1.2 Change Location

The ***Change Location*** option gives users the choice of switching locations if the detected location is not the desired one. Users would then be allowed to select from a list of locations. This option can be accessed either from the menu or by clicking on the ***Change Location*** button in the ***Main Screen*** window as shown below.

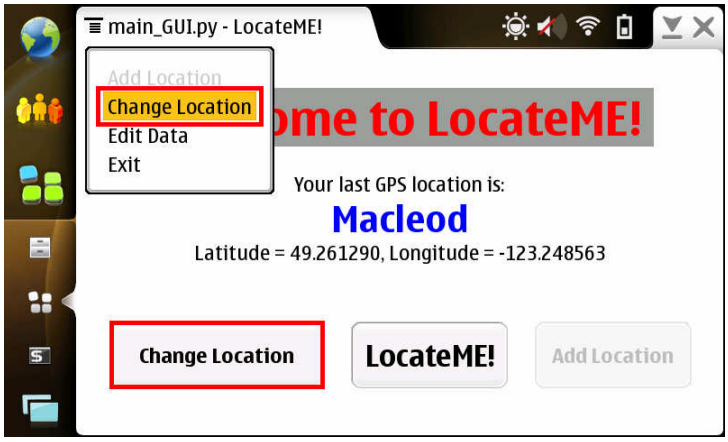

Figure 7 Methods of Accessing *Change Location* option

The ***Change Location*** window is shown in Figure 8 and consists of a drop-down combo box. The drop-down list contains a list of all existing locations that are currently stored in the database.

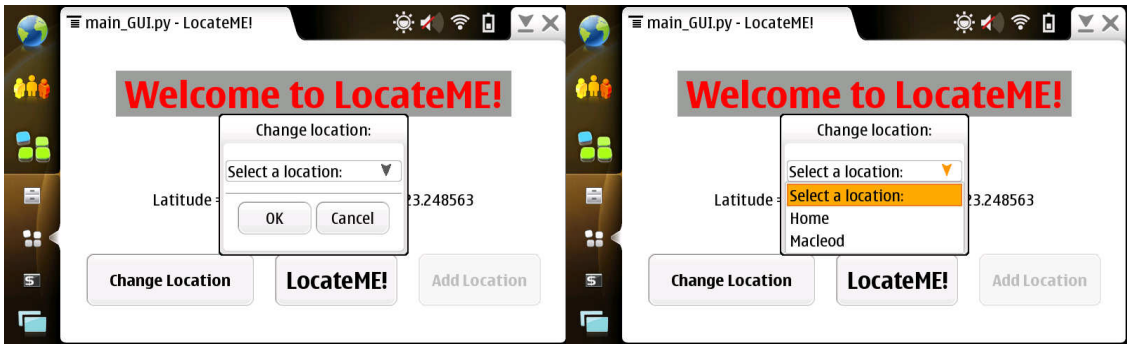

Figure 8 *Change Location* Window

Once a location has been chosen, the ***Main Screen*** window is updated to reflect the user's chosen location as seen in the following figure.

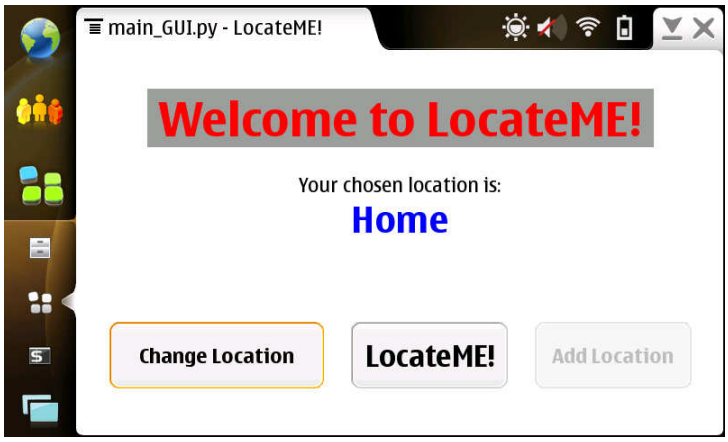

Figure 9 Updated *Main Screen* Window

An assumption made here is that the user should be "smart" enough to pick a location name that actually matches their current location. If an incorrect location is selected, the user will never be able to correctly determine their position in that location.

4.2.1.3 Edit Location Data

This option is accessed from the ***Edit Data*** item in the ***Main Screen*** window's menu as shown in Figure 10. Users can use this option to delete geocode values of a location or delete the location itself.

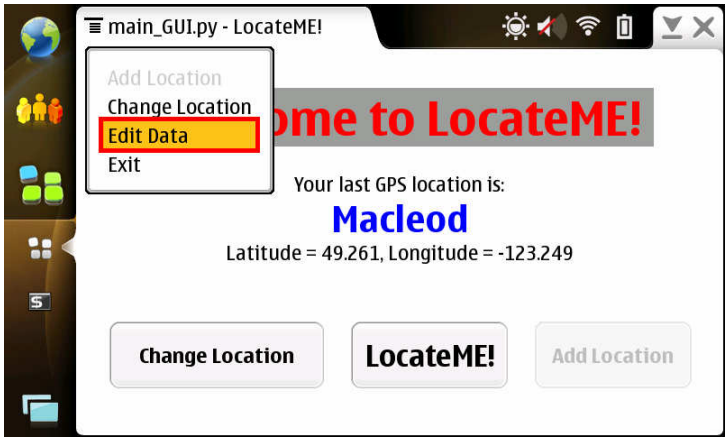

Figure 10 *Edit Data* button

The ***Locations*** window displays a list of all locations currently available in the database. Clicking the ***Edit*** button opens a separate window, seen below, which contains the list of geocode values that are associated with the selected location. The geocode values can then be deleted using the ***Delete*** button. These values will be removed from the database.

Figure 11 *Edit Data* Windows

Deleting a location removes all data associated with this location from the database. Before deletion, users will first be required to confirm their choice through the pop-up confirmation dialog box as shown in Figure 12.

Figure 12 *Delete Item* Confirmation Message

4.2.1.4 Locate User

The ***LocateME!*** button allows users to find their current position at the location previously determined or chosen in the ***Main Screen*** window. Clicking on the

LocateME! button will lead users to a new window displaying the result of the application's estimation of the user's current location along with other options. If the device is not connected to the Internet, a pop-up error message as seen in Figure 4 is displayed. The results window will be described in further detail in the next section.

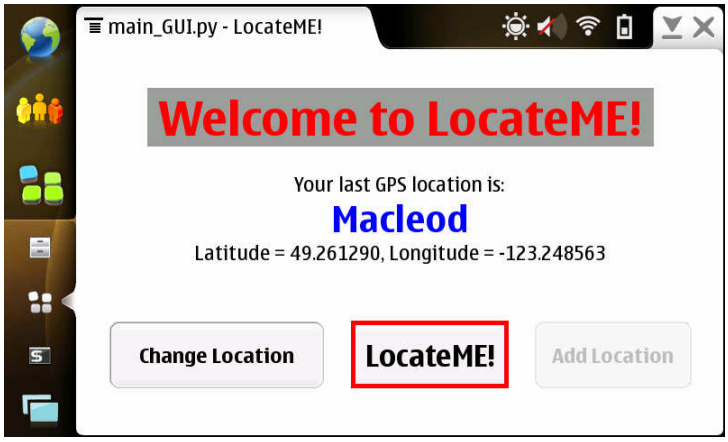

Figure 13 *LocateME!* Button

4.2.2 *LocateME! Results* Window

The ***LocateME! Results*** window is displayed upon selection of the ***LocateME!*** button in the ***Main Screen*** window. At initialization of this window, the application performs a scan for Internet access points that have the same ESSID associated with the pre-determined location and collects signal strength data from these points. The current light intensity is also recorded. The application then compares this information with those of other existing scan points associated with this location and makes an estimation of the user's current position relative to the available scan points.

If the current location does not have any scan points associated with it, this is indicated by the message ***No scan points available***" as shown in Figure 14 below. If there are no matching Internet Access Points, the message ***No matches found!*** is displayed, demonstrated in Figure 15 below. Otherwise, the nearest scan point to the user's current location is returned.

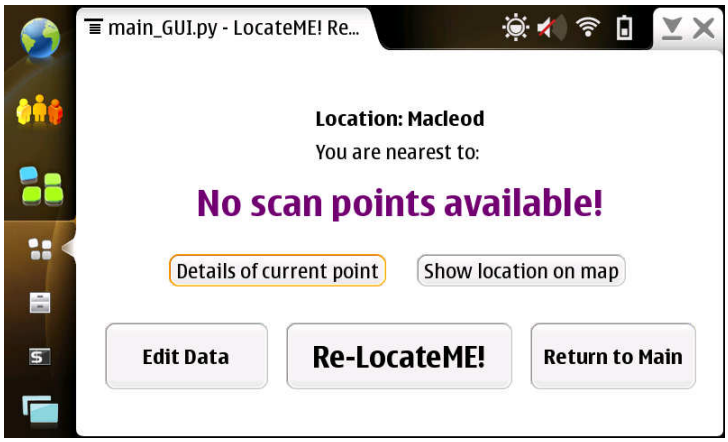

Figure 14 *No Scan Points* Message on *Results* Window

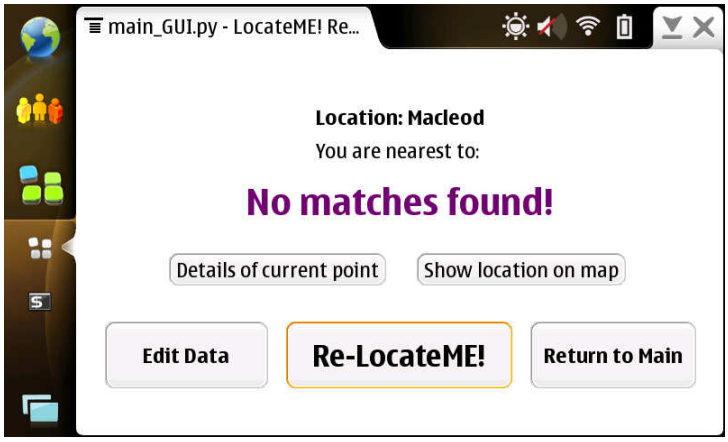

Figure 15 *No Matches Found* Message on *Results* Window

The following five tasks can be performed from within the ***LocateME! Results*** window:

- Get details of current position
- Show location on Google Maps
- Repeat scan for user's current position
- Edit scan points data
- Return to Main Screen window

4.2.2.1 Get Details of Current Position

By clicking on the ***Details of current point*** button in the ***LocateME! Results*** window, users will be able to view information about their current position. The button is indicated in the figure below. Details such as the ESSID, light intensity and list of internet access point addresses with corresponding signal strengths are displayed in a pop-up window.

Figure 16 ***Details of Current Point***

4.2.2.2 Show Location on Google Maps

The ***Show location on map*** button shown in the following figure allows users to view their current location on Google Maps. The application retrieves the geocode value associated with the location name and uses the Google Static Maps API to create a map. The map is then displayed in a new browser window and the current location is indicated with an orange marker. An example of the map is shown below. If no Internet connection is detected, a pop-up error message as shown in Figure 4 will be displayed.

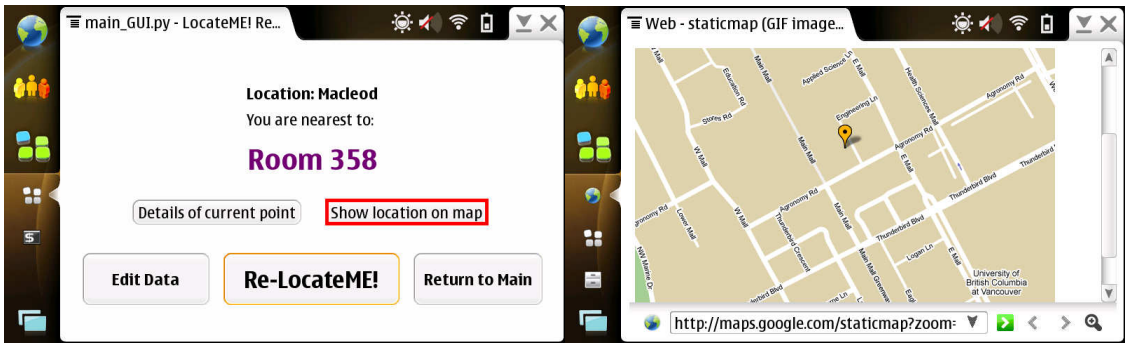

Figure 17 *Show Location On Map*

4.2.2.3 Repeat Scan for User's Current Position

This function can be accessed either from the ***Re-LocateME!*** menu item or by clicking on the ***Re-LocateME!*** button in the ***Results*** window as seen in the figure below. The purpose of this button is to allow users to repeat the action of finding their location multiple times without having to keep returning to the ***Main Screen*** window. If no Internet connection is detected by the device, a pop-up error message like the one in the Figure 4 is displayed.

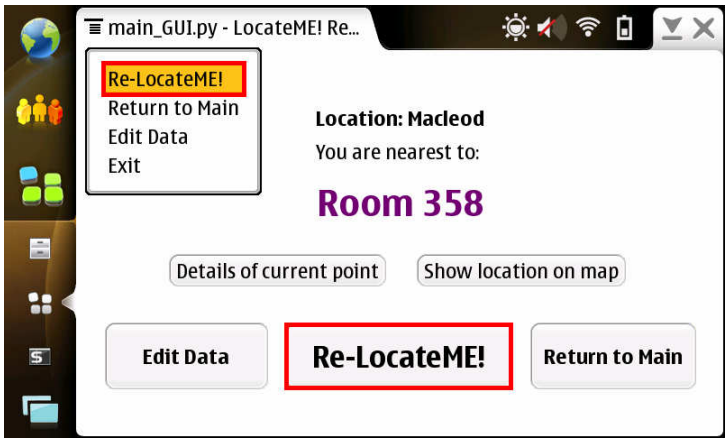

Figure 18 Methods of Accessing *Re-LocateME!* Option

4.2.2.4 Edit Scan Points Data

Users can access this window by clicking on the ***Edit Data*** button in the ***Results*** window or by selecting the ***Edit Data*** menu item in the ***Results*** window menu, as shown in the following figure.

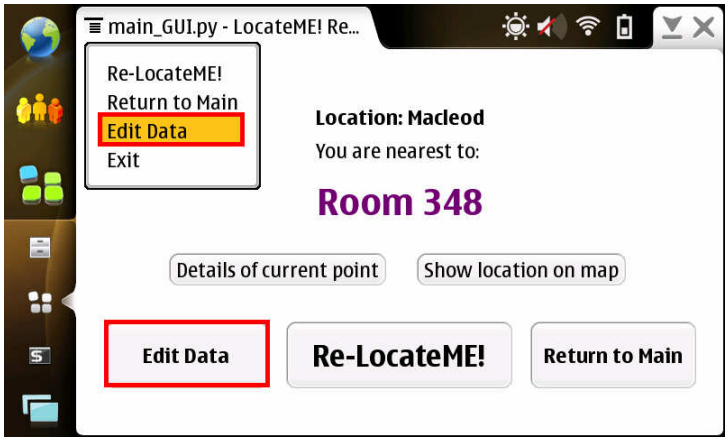

Figure 19 Methods of Accessing *Edit Data* Option

This brings up a window named ***List of Scan Points*** which displays a list of all scan points that are associated with the current location name. Users have three options in this window; that is to add, edit, or delete a scan point.

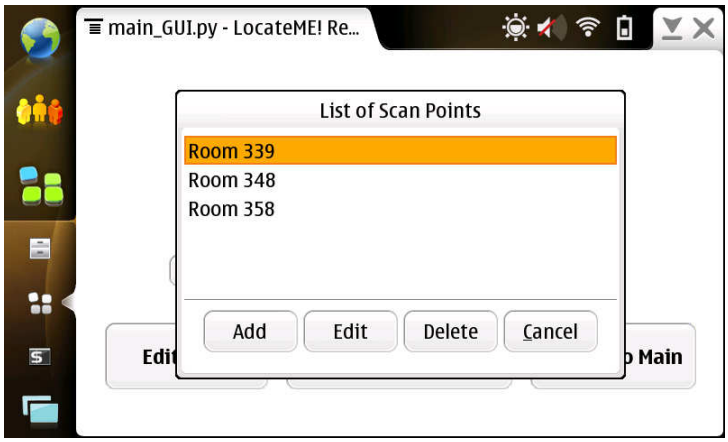

Figure 20 *List of Scan Points* Window

Clicking on the ***Add*** button opens a pop-up window named ***Add Data*** such as the one shown in Figure 21. This window contains a text box in which users can enter a

description for the scan point they wish to add. Users will then have to click on the ***Scan*** button in order to collect information such as light intensity and internet access point signal strengths for that particular scan point. The ***Save*** button is originally deactivated and only enabled after a scan has been initiated as illustrated in Figure 21. After the data is saved, the ***Save*** button is deactivated again until the next scan is performed. This is a workaround for preventing users from saving the same data in the database multiple times.

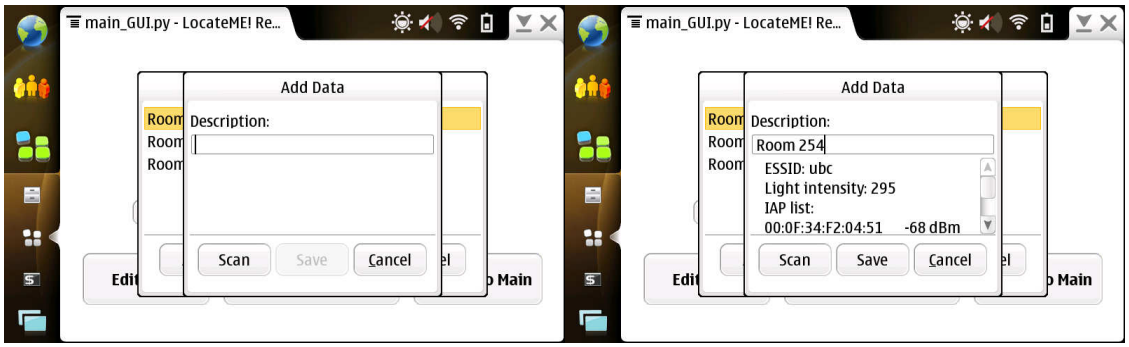

Figure 21 *Add Data* Window

If a user attempts to enter a description that already exists or save the scan point without a description, the error messages shown in Figure 22 will be displayed. If no Internet connection is detected at the time the ***Scan*** button is clicked, then a pop-up error message as seen in Figure 4 is displayed to the user.

Figure 22 *Description Exists* and *No Description* Error Messages

The ***Edit*** button in the ***List of Scan Points*** window lets users update the details for a particular scan point. This is useful for users who only wish to perform a re-scan for a scan point without having to enter the description for that point again. The ***Save*** button

here uses the same idea as in the ***Add Data*** window and is only activated when a scan has been performed. After the data is saved, the button is once again deactivated. Below is a figure of the ***Edit Data*** window that appears when the ***Edit*** button is selected.

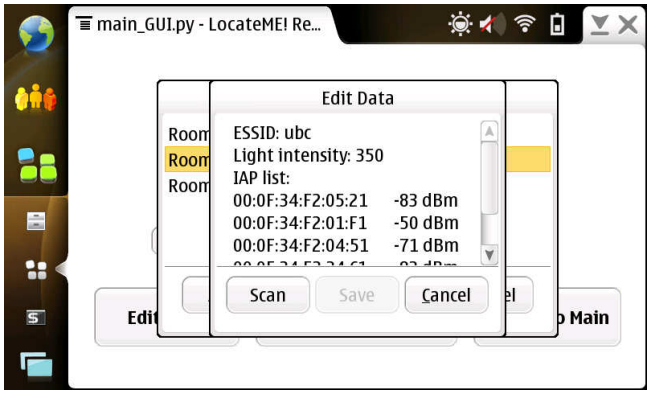

Figure 23 *Edit Data* Window

To delete a scan point, users can select the ***Delete*** button in the ***List of Scan Points*** window. A confirmation dialog box similar to the one in Figure 12 is displayed and requires users to confirm that they indeed wish to delete that particular scan point.

4.2.2.5 Return to Main Screen Window

Users can return to the ***Main Screen*** window through three ways; by clicking on the ***Return to Main*** button, by selecting the ***Return to Main*** menu item, and by clicking on the ***X*** at the top right corner of the ***Results*** window.

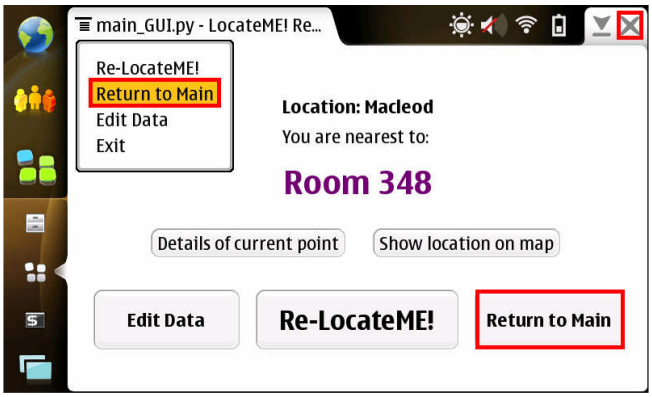

Figure 24 Methods of Accessing *Return to Main* Option

4.3 Database

This application makes use of a SQLite database to store and retrieve contextual data internally. SQLite is a software library that implements a self-contained, serverless, zero-configuration, and transactional SQL database engine [1]. The database is stored on the device itself and is created during the very first use of the application. It runs on the back-end of the application and is not visible to the user of the application.

In order to be able to access the database from within Python, the module ***sqlite3*** was used. This module provides an interface between Python and SQLite and allows the database to be accessed using a non-standard variant of the SQL query language.

4.3.1 Entity-Relationship Diagram

The figure below shows the ER diagram of the application's database. This diagram illustrates the relationships between the different entities in the database.

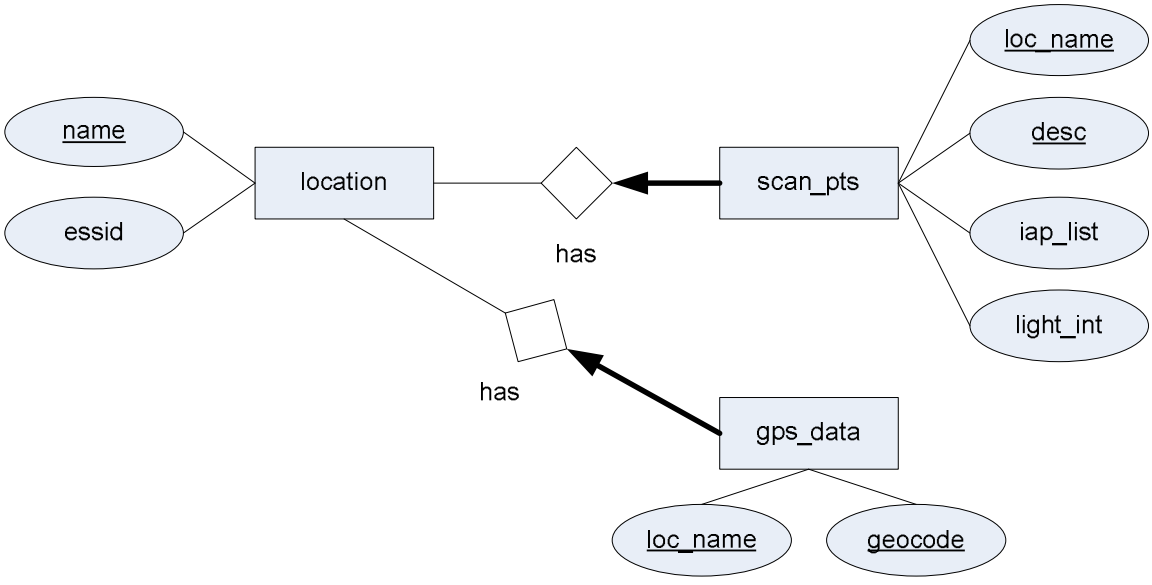

Figure 25 ER Diagram

4.3.2 Relation Schemas

The following tables were derived from the ER diagram shown in the previous sub-section. There are three tables in total. The first table is used to store information about the location. The second table contains the geocode values that correspond to each location and the third table contains information about the scan points corresponding to each location.

location (name, essid)
Primary Key: name

gps_data (loc_name, geocode)
Primary Key: loc_name, geocode
Foreign Key: loc_name references **location**

scan_pts (loc_name, desc, iap_list, light_int)
Primary Key: loc_name, desc
Foreign Key: loc_name references **location**

4.3.3 Triggers

It is a known issue that SQLite supports syntax for foreign key constraints but ignores them. Therefore, in order to enforce the foreign key constraints in the tables, triggers need to be created.

In my application's database, the ***gps_data*** and ***scan_pts*** tables both reference the ***location*** table. A trigger was added to enforce the ***Delete on Cascade*** constraint of both tables such that if a location was deleted from the ***location*** table, rows in the ***gps_data*** and ***scan_pts*** table that corresponded to that location would be deleted as well. The code to do this is shown below.

```
db_cur.execute("create trigger if not exists delete_loc \
                   before delete on location\
                   begin \
                      delete from gps_data where loc_name = old.name; \
                      delete from scan_pts where loc_name = old.name; \
                   end;")
```

4.4 Algorithm to Determine Location of User

The heart of my application lies in its ability to be able to correctly determine the current position of a user in a particular building. The method that is currently used in the application is the mean squared error. Mean squared error measures the average of the square of the error where the error is the amount by which the estimator differs from the quantity to be estimated [2]. The equation for calculating mean squared error is given below.

$$MSE(\hat{e}) = E((\hat{e} - e)^2)$$

In my algorithm, the wireless signal strength and light intensity of the user's current position is compared one-by-one with the list of saved scan points in the database and the mean squared error is calculated for each point in the list. The point in the list with the corresponding lowest mean squared error is then assumed to be the position that the user is closest to.

4.4.1 Accuracy Issues

The algorithm used in this application is admittedly not the most accurate one. Throughout the course of this project, the majority of the emphasis was placed on the other components in the system and less effort was spent on actually designing an algorithm that could perfectly find the location of a user within a building.

One problem with using my algorithm is that users need to have a very wide range of scan points for that particular building stored in its database before hand. This is because the end result is very dependent on the list of available scan points in the database. If the users try to locate themselves in an area which did not have pre-defined scan points, they would obtain an inaccurate result. Therefore, it is necessary for users to allow the application to "learn" about a building in advance before being able to correctly identify their location in that building.

Another issue that affects the accuracy of the result is the huge fluctuation in the wireless signal strength and light intensity measurements. Measurements of wireless signal strength made at the same point at different times show a fairly significant difference that potentially reduces the accuracy of the result. Also, the light intensity measurements are very sensitive and a slight change in the angle of the device produces a large change in the measurement.

4.5 Data Acquisition Modules

The data acquisition modules in this application consist of functions used to interface with the hardware of the Nokia N810 device in order to retrieve information from them. The hardware that my application interfaces with is the built-in GPS receiver, WLAN module, and the ambient light sensor. The data collected from these hardware components are used to determine the location of a user.

4.5.1 Retrieving GPS data

The Nokia N810 device contains a built-in GPS receiver. My application makes use of this unit to retrieve the latitude and longitude values of the last saved GPS location on the device and from there, deduce the building that the user is in. This information is only

obtained once, during the initialization of the application at start-up. The geocode values that are returned consist of floating point numbers of up to six decimal places. However, my application only records the values up to 3 decimal places. This is to reduce the errors that result from fluctuating values.

In order to be able to access the device's GPS receiver from within Python, I used a file named ***liblocation.py*** which is a Python wrapper for the official Maemo Liblocation API that was originally written in the C language. The Maemo Liblocation API is a library for the location framework of the device and is made up of functions for parsing GPSD outputs and controlling GPSD [3]. GPSD is a daemon that receives data from a GPS receiver and is used in Maemo to talk to GPS devices and report position data [3]. The ***liblocation.py*** file was written by Rob Brewer and can be found on his website [4]. A copy of this file can also be found in the Appendix section of this report.

4.5.2 Retrieving WLAN data

One of the critical parts of the application was to be able to retrieve WLAN data from the device. For this purpose, the ***wireless-tools*** package was used. This package is used to manipulate the Linux Wireless Extensions which is an interface that allows WLAN specific parameters to be set and retrieved [5]. The device does not come with this package installed, therefore to be able to use the application; this package will have to be installed on the device as well. The package ***sudser*** is also required to enable root access on the device. Instructions on how to install the two packages can be found in Appendix A.3 of this report.

In my application, the ***subprocess*** module is used to execute the following shell command ***sudo iwlist wlan0 scan.*** This command returns a list of access points that are in range along with detailed information such as ESSID, Address, Signal Strength, Frequency, Quality, etc. for each of them [6]. The only values that my application is interested in are the ESSID, Address and Signal Strength. The list is then parsed to

extract the ESSID, Address and Signal Strength of each available access point. The results from this are then searched for access points with ESSID values that match the saved ESSID value corresponding to that particular location.

As mentioned earlier in Section 4.2.1.1 of this report, when a new location is added to the database, the ESSID of the current WLAN connection is saved as well. From then on, this ESSID value will be used as the comparison factor for all wireless data that are retrieved. This is to ensure consistency among all scan points in the situation where the user does not necessarily connect to the same WLAN network every time.

The concept behind this is that when a user first adds a new location, the ESSID of the current connected WLAN network has to be a valid one. Thus, whenever a user performs a scan at that location, it will only return results that correspond to that ESSID value. Even if the user switches to a different WLAN network later, the application would still be able to locate the user since the results are based on the saved ESSID value and not the current ESSID value.

4.5.3 Ensuring WLAN connectivity

Internet connectivity is needed for my application to function properly. The ***Python-Conic*** package is used at initialization of my application to check for an Internet connection. If the device is not connected, it will try to connect to the default connection or display the connection selection dialog window.

The ***Python-Conic*** package contains bindings for the Maemo LibConiC API which is an Internet Connectivity API for applications to manage Internet connections on Maemo devices [3]. This package can be found on the Python for Maemo website [7] and is bundled together with the installation of the Python programming language support on Maemo, thus, no additional installation is required to use it.

Besides that, the command ***sudo iwlist wlan0 scan*** is used to check the status of the Internet connection. If the device is not connected, executing this command returns no results and this in turn will lead to a pop-up message dialog box requesting the user to connect to the Internet. This check is performed every time the user clicks on the ***LocateME!*** and ***Add Location*** buttons in the ***Main Screen*** window, and the ***Re-LocateME!*** and ***Show location on map*** buttons in the ***Results*** window.

4.5.4 Retrieving Light Intensity Data

The Nokia N810 device comes with a built-in ambient light sensor that is located on the top left corner of the device, right above the camera. The light intensity can be measured by reading the contents of the file ***/sys/bus/i2c/devices/0-0029/lux*** that is stored on the device [8]. The result returned is an integer whose value indicates the intensity of the light that is able to reach the light sensor on the device. The larger the returned value is, the higher the intensity of the light.

5.0 APPLICATION TESTING

Testing is an integral part of the development process because it ensures that the application is able to function correctly in the desired manner. The sections below describe briefly the types of testing that was performed on the system.

5.1 System testing

Most of the testing was conducted in parallel with the development of the application. Testing was mainly done on the GUI of the application through exploratory methods. Since the project development was done individually, it allowed the code to be written and tested immediately after. If any errors were found, they could be fixed right away.

Each component of the system was developed and tested individually to ensure that they were fully functioning as desired. The system was also tested as a whole after all the different components had been integrated together. The bugs found in the application were listed and fixed according to priority and risk level. After the bugs had been fixed, regression testing was performed to ensure that the fixes were working correctly and that no other parts of the system had been affected by the changes.

5.2 Accuracy Testing

The accuracy of the algorithm's result is crucial to the success of the application. In order to be able to quantify the accuracy of the algorithm, the following test was devised.

5.2.1 Description

This test was conducted inside the Macleod building located on the campus of UBC. The Macleod building consists of five floors, however, since the fifth floor of the building was not accessible, this test only covered the first four floors of the building. Each floor of the building was first divided into several sections as shown in the figure below.

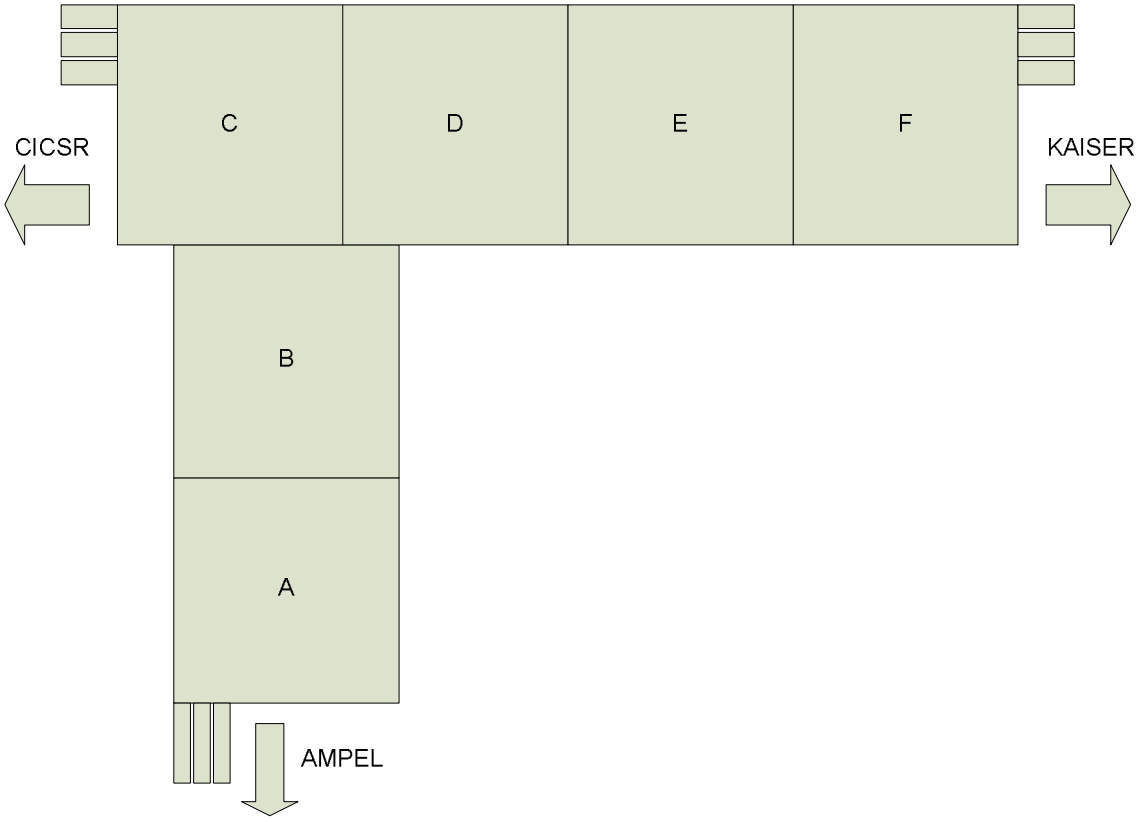

Figure 26 MacLeod Building Floor Plan

Using the application, a scan point for each section of every floor was obtained and saved. Then, to determine how accurately the application was able to locate the position of a user, each section on every floor was revisited and the location returned by the application was recorded. The measurements were repeated three times for each section on each floor. The results were then checked to determine the number of times the correct location was returned. Based on the results, the accuracy of the algorithm was found by calculating the number of times the correct location was returned out of the total number of measurements made.

5.2.2 Results

The results from the test described above are tabulated in the following table.

Location: Macleod building					
No.	**Expected location**	**Returned location**			**Number of matches (out of 3)**
		Attempt 1	**Attempt 2**	**Attempt 3**	
1	1A	1A	2C	1E	1
2	1B	3C	1A	1E	0
3	1C	1C	1C	1C	3
4	1D	1E	2F	1E	0
5	1E	1C	1E	1E	2
6	1F	1F	1F	2C	2
7	2A	1A	2F	1B	0
8	2B	2D	1A	2F	0
9	2C	1C	4E	1E	0
10	2D	2F	1E	4B	0
11	2E	1F	1E	2E	1
12	2F	2E	3F	3E	0
13	3A	4B	3A	3A	2
14	3B	2C	2F	4A	0
15	3C	1C	3C	1C	1
16	3D	2D	3D	4F	1
17	3E	1E	3E	3E	2
18	3F	3F	3F	3E	2
19	4A	1A	2C	4A	1
20	4B	4E	4E	4B	1
21	4C	2A	4C	4C	2
22	4D	3C	4C	3C	0
23	4E	1E	2C	4E	1
24	4F	4D	3F	4D	0

Table 1 Algorithm Accuracy Test Results

Total number of matches, $Total_{match} = 22$

Total number of measurements, $Total_{meas} = 72$

$$Accuracy = \frac{Total_{match}}{Total_{meas}} x100\% = \frac{22}{72} X100\% = 30.56\%$$

6.0 TECHNICAL CHALLENGES

While working on this project, I encountered a number of problems and obstacles that were potentially hazardous to the development and progress of my project. Although these problems set me back on my project and made me doubt my ability at times, in the end, I was able to resolve most of them or find a workaround to the problem.

The very first challenge that I faced when I first started on this project was the fact that I knew nothing about programming mobile applications. Trying to set up the development environment itself was tedious and much time was spent on researching for the best and simplest development setup. Choosing a programming language was another headache because although C, C++ and Python were all supported on the Maemo platform, I was not very good at any of them. Java, which was my preferred language was not well-supported and thus was not a viable option. In the end, I chose to use Python because I thought it would be easier to familiarize myself with it compared to C and C++.

After that was out of the way, the real work lay in figuring out how to obtain the contextual information I needed from the device such as GPS data, light intensity data and wireless signal strengths. Although the Maemo platform was well-established, its usage was still at most limited to small numbers, thus, online resources were rather difficult to find. At least half the term of the project was spent on this portion of the project, mainly on research. Some time was also spent on implementing and testing the various ways of obtaining the data until I was able to achieve what I desired. The hardest to implement was probably the retrieval of wireless signal strength data. In order to get that to work successfully, I tried at least three different methods but was still unable to make it function as I wanted. Finally, I was able to successfully implement one of the initial methods that I had used.

Once I was able to retrieve the various types of contextual data, formal work on the design and implementation of the application's GUI began. One of the challenges faced

here was to come up with a GUI design that was user-friendly and allowed the user to interact easily with the application. A lot of emphasis was placed on the usability of the application itself. Coding of the GUI was also time-consuming because of the many buttons and different functionalities in the GUI.

Another issue that arose was regarding the database. Initially, I had planned to use a remote MySQL database server to allow for a global database that everyone could connect to. However, I was unable to install a Python database interface driver on the device because it was not yet supported on the device. After much consideration, I settled for the SQLite database which ran on the device itself. Thus, users would have their own individual database rather than a shared one.

Overall, this project although filled with challenges, was a great learning opportunity and allowed me to enhance my software design and development skills.

7.0 SUGGESTIONS FOR IMPROVEMENT

This application can be said to be a very primitive version and contains many areas of improvement. Due to time constraints, these areas were set as lower priority tasks and were not implemented. The following is a list of my suggestions on improvements that can be made on the application.

1) Use a remote SQL database

   Ideally, a remote database server should be used. However, as mentioned in the previous section, I was unable to find a Python database interface driver that was supported on the device. One method that could be used instead would be to use PHP from within Python to upload the data to a web server.

2) Add SQL exceptions

   Currently, no error checking is done within the SQLite database itself. The application is entirely dependent on the GUI to perform checks for errors such as incorrect inputs, empty inputs, duplicate entries, and etc. This increases the vulnerability of the application since the GUI error messages could be by-passed by a malicious user.

3) Increase accuracy of algorithm

   The algorithm that is currently used incorporates the mean squared error method. This is not the most accurate algorithm and thus can be further improved for better accuracy.

4) Obtain additional contextual data

   Other contextual data such as the ambient temperature, snap shots and sound level of the location could be used together with the existing types of contextual data to improve the accuracy of the results.

5) Allow users to add locations without GPS data

   Currently, users can only add a new location when the last saved GPS data that was retrieved does not exist in the database. This can be a problem when users wish to add information about a new location but did not previously have their GPS receiver turned on. By adding this feature, users will have more flexibility in the usage of the application.

6) Add descriptions for geocode values

   Currently, geocode values that are added to the database do not come with a description. This can be confusing later when the user wishes to view the list of geocode values associated with each location. Adding a description would make it easier for users to identify which geocode value corresponded to which entrance of the location.

7) Allow location name and corresponding ESSID value to be modified

   Currently, location names and their corresponding ESSID value can only be deleted and not edited. This can be an inconvenience for users, thus, adding this feature would increase the usability of the application.

8.0 CONCLUSION

The ***LocateME!*** application that I created for this project, although still at a primitive stage, can be considered to have met most of its initial requirements. These requirements included the ability to obtain the device's last saved GPS data, wireless signal strength and light intensity, the ability to store and retrieve these data in a database, and finally, the ability to locate a user based on these data.

In this report, the different phases of the application's development life cycle were described in detail to provide readers of this document with a thorough understanding of the inner workings of the application. The development stages mentioned included setup of the development environment, system design and architecture, and implementation and testing of the application. The technical challenges faced during the term of the project and suggestions for improvement on the application were also outlined in the report.

All in all, this project has been a rewarding and useful learning experience. Through this project, I gained knowledge about mobile application development as well as refined my skills in software design and development. This project also gave me the chance to experience each stage of the application development life cycle from start to finish. Besides obtaining technical skills, I also learned the importance of time management and report writing skills.

9.0 REFERENCES

[1] SQLite. (2009, Mar.). Home Page. [Online]. Available: http://www.sqlite.org/

[2] Wikipedia. (2009, Mar.). Mean Squared Error. [Online]. Available: http://en.wikipedia.org/wiki/Mean_squared_error

[3] Nokia Corp. (2008, Dec.). Maemo Diablo Reference Manual for Maemo 4.1. [Online]. Available: http://maemo.org/maemo_release_documentation/maemo4.1.x/

[4] R. Brewer. (2008, Nov.). Nokia Internet Tablet Stuff. [Online]. Available: http://brewer123.home.comcast.net/~brewer123/projects/nokia/

[5] Nokia Corp. Maemo 4.1 SDK Tools: wireless-tools. [Online]. Available: http://maemo.org/development/tools/doc/diablo/wireless-tools/

[6] Nokia Corp. Maemo Man Pages: iwlist. [Online]. Available: http://maemo.org/development/documentation/man_pages/iwlist/

[7] Instituto Nokia de Technologia. Python for Maemo Home Page. [Online]. Available: http://pymaemo.garage.maemo.org/documentation.html

[8] Narius. (2009, Jan.). The Nokia N810: Hardware Interface Commands. [Online]. Available: http://thescienceofgreen.com/?m=20090120

APPENDIX A: DEVELOPMENT ENVIRONMENT SETUP

This section will briefly outline the steps that are needed in order to set up the development environment. The setup can be divided into three portions: one for the Host PC, a second one for setup within VMWare Player and another for the Nokia N810 device.

A.1 Host PC (Windows XP Pro)

1. Download and install VMWare Player
   http://www.vmware.com/products/player/

2. Download Maemo SDK VMWare Appliance, 0.8
   http://maemovmware.garage.maemo.org/

3. Download 7-zip
   http://www.7-zip.org/

4. Use 7-zip to extract the Maemo SDK VMWare Appliance 0.8 image

5. Run VMWare Player and open the Maemo SDK VMWare image

A.2 Within VMWare Player

1. The following are applications that need to be installed on the Linux Virtual Machine:
   - *Sun J2SE Runtime Environment (JRE) 5.0*
   - *Python 2.5*
   - *SSH client*

- *Eclipse v3.3 or higher*
- *PyDev plug-in for Eclipse*
- *Subclipse plug-in for Eclipse*

2. To install Sun J2SE Runtime Environment (JRE) 5.0
   - Add the following line to ***/etc/apt/sources.list***:

     (Gutsy) deb http://us.archive.ubuntu.com/ubuntu gutsy universe multiverse

     or

     (Hardy) deb http://us.archive.ubuntu.com/ubuntu hardy universe multiverse
   - In the terminal window, type:

     sudo apt-get update

     sudo apt-get install sun-java5-jre
   - Set the Sun Java virtual machine as default:

     sudo update-alternatives –config java

     and select ***/usr/lib/jvm/java-1.5.0-sun/jre/bin/java***

3. To install Python 2.5, in the terminal window, type:

   sudo apt-get install python2.5

4. To install SSH client, in the terminal window, type:

   sudo apt-get install openssh-client

5. To obtain Eclipse, download and extract the compressed folder. The executable file for Eclipse is located within the extracted folder.
   http://www.eclipse.org/downloads/

6. To install PyDev, use the Remote Update Site within Eclipse:
   - Select ***Help > Software Updates*** from the main menu bar
   - Under the ***Available Software*** tab, select ***Add Site***

- Enter the URL http://pydev.sourceforge.net/updates/ and click ***OK***
- Select the checkbox beside the URL entered above and click ***Install***

7. Configure the Python interpreter in Eclipse
   - Go to ***Window > Preferences > PyDev > Interpreter – Python***
   - Choose the interpreter installed on the PC. In my setup, this was ***/usr/bin/python2.5***

8. To install Subclipse in Eclipse, repeat Step 6 using the following URL: http://subclipse.tigris.org/update_1.6.x

9. To allow for synchronization between Eclipse and Google Code:
   - In Eclipse, go to ***Window > Show View > SVN Repositories > Add New Repository***
   - Enter the URL provided in the Google Code page. In my setup, this was https://eece496-ktay.googlecode.com/svn/trunk/
   - To share an existing project, right click on the project folder and select ***Team > Share Project***
   - Choose SVN as the repository type and select the repository added earlier

A.3 Nokia N810 Internet Tablet (Maemo 4.1.2 Diablo)

1. The packages listed below are required for the development of Maemo applications on the device:
   - *maemo-pc-connectivity*
   - *maemo-python-device-env*
   - *sudser*

2. The above mentioned packages can be installed using the Application Manager tool on the device:

- Go to Settings > Application Manager > Browse installable applications
- Select and install the packages

3. The wireless-tools package is needed specifically for this application and is used to manipulate the Linux Wireless Extensions. To install:
    - Add the following lines to ***/etc/apt/sources.list***:

      deb http://repository.maemo.org diablo/tools free non-free

      deb-src http://repository.maemo.org diablo/tools free non-free
    - In xterm, type:

      sudo apt-get update

      sudo apt-get install wireless-tools

APPENDIX B: SOURCE CODE

This section displays the contents of the source code files.

B.1 *main_GUI.py*

```
#!/usr/bin/env python2.5

#############################################################################
#
#    LocateME! - Finds location of user in a specific building using contextual
#    information
#    Copyright (C) <2009>  <Kattie Tay>
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#    the Free Software Foundation, either version 3 of the License, or
#    (at your option) any later version.
#
#    This program is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU General Public License for more details.
#
#    You should have received a copy of the GNU General Public License
#    along with this program.  If not, see <http://www.gnu.org/licenses/>
#
#############################################################################


# Import packages
import pygtk
pygtk.require('2.0')
import gtk
import hildon
import pango
import re
import gps_funcs
import wlan_funcs
import results_GUI
import db_funcs


# Initialize global variables
geocode = 0
lat_long_label_str = ""


class mainGUI(hildon.Program):

    # Initialize GUI
    def __init__(self):
```

```
        hildon.Program.__init__(self)

        # Retrieve last saved GPS data and store as global variable geocode
        # Only store 3 decimal places
        global geocode, lat_long_label_str
        latitude = gps_funcs.gps_get_latitude()
        longitude = gps_funcs.gps_get_longitude()
        geocode = "%.3f, %.3f" %(latitude, longitude)


        # --- Main Window ---
        # Create main window
        self.main_window = hildon.Window()
        self.main_window.connect("destroy", self.destroy_cb, None)
        self.main_window.set_border_width(10)
        self.main_window.set_title("LocateME!")

        # Create table layout
        table_layout = gtk.Table(10, 10, True)
        self.main_window.add(table_layout)


        # --- Welcome label ---
        # Create label to display welcome message
        welcome_label_str = " Welcome to LocateME! "
        welcome_label = gtk.Label(welcome_label_str)
        # Set label attributes
        welcome_label_attr = pango.AttrList()
        fg_color = pango.AttrForeground(65535, 0, 0, 0, len(welcome_label_str))
        bg_color = pango.AttrBackground(40000, 40000, 40000, 0,
len(welcome_label_str))
        size = pango.AttrSize(40000, 0, -1)
        weight = pango.AttrWeight(pango.WEIGHT_BOLD, 0, len(welcome_label_str))
        welcome_label_attr.insert(fg_color)
        welcome_label_attr.insert(bg_color)
        welcome_label_attr.insert(size)
        welcome_label_attr.insert(weight)
        welcome_label.set_attributes(welcome_label_attr)
        welcome_label.set_alignment(0.4, 1)
        # Set position of widget in window and show it
        table_layout.attach(welcome_label, 1, 9, 0, 3, gtk.EXPAND, gtk.EXPAND,
5, 5)
        welcome_label.show()


        # --- Last GPS location label ---
        # Create label to display last GPS location
        last_gpsloc_label_str = "Your last GPS location is:"
        self.last_gpsloc_label = gtk.Label(last_gpsloc_label_str)
        self.last_gpsloc_label.set_alignment(0.5, 0)
        # Set position of widget in window and show it
        table_layout.attach(self.last_gpsloc_label, 1, 9, 3, 4, gtk.FILL,
gtk.EXPAND, 5, 5)
        self.last_gpsloc_label.show()


        # --- Geocode values label ---
        # Create label to display latitude and longitude
        lat_long_label_str = "Latitude = %.3f, Longitude = %.3f" %(latitude,
longitude)
        self.lat_long_label = gtk.Label(lat_long_label_str)
        self.lat_long_label.set_alignment(0.5, 1)
```

```
        # Set position of widget in window and show it
        table_layout.attach(self.lat_long_label, 1, 9, 5, 6, gtk.FILL,
gtk.EXPAND, 5, 5)
        self.lat_long_label.show()


        # --- Location name label ---
        # Create label to display location name
        loc_name_label_str = db_funcs.db_get_loc_name(geocode)
        self.loc_name_label = gtk.Label(loc_name_label_str)
        # Set label attributes
        loc_name_label_attr = pango.AttrList()
        fg_color = pango.AttrForeground(0, 0, 65535, 0,
len(loc_name_label_str))
        size = pango.AttrSize(30000, 0, -1)
        weight = pango.AttrWeight(pango.WEIGHT_BOLD, 0,
len(loc_name_label_str))
        loc_name_label_attr.insert(fg_color)
        loc_name_label_attr.insert(size)
        loc_name_label_attr.insert(weight)
        self.loc_name_label.set_attributes(loc_name_label_attr)
        self.loc_name_label.set_alignment(0.5, 0.5)
        # Set position of widget in window and show it
        table_layout.attach(self.loc_name_label, 1, 9, 4, 5, gtk.FILL,
gtk.EXPAND, 5, 5)
        self.loc_name_label.show()


        # --- Change location button ---
        # Create button to change location
        change_loc_button_str = " Change Location "
        self.change_loc_button = gtk.Button(change_loc_button_str)
        self.change_loc_button.connect("clicked", self.change_loc_cb, None)
        self.change_loc_button.set_alignment(0.5, 0.5)
        # Set button label attributes
        change_loc_label = self.change_loc_button.child
        change_loc_label.modify_font(pango.FontDescription("Bold"))
        # Set position of widget in window and show it
        table_layout.attach(self.change_loc_button, 0, 4, 7, 10, gtk.FILL,
gtk.FILL, 15, 20)
        self.change_loc_button.show()


        # --- LocateMe button ---
        # Create button to locate user
        locate_user_button_str = " LocateME! "
        self.locate_user_button = gtk.Button(locate_user_button_str)
        self.locate_user_button.connect("clicked", self.locate_user_cb, None)
        self.locate_user_button.set_alignment(0.5, 0.5)
        # Set button label attributes
        locate_user_label = self.locate_user_button.child
        locate_user_label.modify_font(pango.FontDescription("Bold 24"))
        # Set position of widget in window and show it
        table_layout.attach(self.locate_user_button, 4, 7, 7, 10, gtk.FILL,
gtk.FILL, 15, 20)
        self.locate_user_button.show()


        # --- Add Location button
        # Create button to add location
        add_loc_button_str = " Add Location "
        self.add_loc_button = gtk.Button(add_loc_button_str)
        self.add_loc_button.connect("clicked", self.add_loc_cb, None)
```

```
        self.add_loc_button.set_alignment(0.5, 0.5)
        # Set button label attributes
        add_loc_label = self.add_loc_button.child
        add_loc_label.modify_font(pango.FontDescription("Bold"))
        # Set position of widget in window and show it
        table_layout.attach(self.add_loc_button, 7, 10, 7, 10, gtk.FILL,
gtk.FILL, 15, 20)
        self.add_loc_button.show()


        # --- Menu ---
        # Set up the menu bar
        self.options_menu = gtk.Menu()

        # Create menu items
        self.add_loc_menuitem = gtk.MenuItem("Add Location")
        self.change_loc_menuitem = gtk.MenuItem("Change Location")
        self.edit_data_menuitem = gtk.MenuItem("Edit Data")
        self.exit_menuitem = gtk.MenuItem("Exit")

        self.options_menu.append(self.add_loc_menuitem)
        self.options_menu.append(self.change_loc_menuitem)
        self.options_menu.append(self.edit_data_menuitem)
        self.options_menu.append(self.exit_menuitem)

        self.add_loc_menuitem.connect_object("activate", self.add_loc_cb, None)
        self.change_loc_menuitem.connect_object("activate", self.change_loc_cb,
None)
        self.edit_data_menuitem.connect_object("activate", self.edit_data_cb,
None)
        self.exit_menuitem.connect_object("activate", self.destroy_cb, None)

        self.add_loc_menuitem.show()
        self.change_loc_menuitem.show()
        self.edit_data_menuitem.show()
        self.exit_menuitem.show()

        self.main_window.set_menu(self.options_menu)


        # --- Call disable_buttons_check function ---
        # Check which buttons to enable or disable
        self.disable_buttons_check()


        # Show main window
        table_layout.show()
        self.main_window.show()


    # Get no value selected dialog box
    def get_none_sel_dialog(self):
        # Create no value selected error dialog window
        none_sel_dialog = gtk.Dialog("Error!", self.main_window,
gtk.DIALOG_MODAL, (gtk.STOCK_OK, gtk.RESPONSE_OK))
        none_sel_dialog_label = gtk.Label("No value selected!")
        none_sel_dialog.vbox.pack_start(none_sel_dialog_label, True, True, 0)
        none_sel_dialog.set_has_separator(False)
        none_sel_dialog.show_all()

        # Run no value selected error dialog window
        none_sel_dialog_result = none_sel_dialog.run()
        # Destroy no value selected erorr dialog window when OK is pressed
```

```
        if none_sel_dialog_result == gtk.RESPONSE_OK:
            none_sel_dialog.destroy()


    # Get no internet connection detected dialog box
    def get_no_conn_dialog(self):
        # Create no connection error dialog window
        self.no_conn_dialog = gtk.Dialog("No internet connection!",
self.main_window, gtk.DIALOG_MODAL, (gtk.STOCK_OK, gtk.RESPONSE_OK))
        no_conn_dialog_label = gtk.Label("Internet connection is required to
proceed. Please connect to the internet.")
        no_conn_dialog_label.set_line_wrap(True)
        self.no_conn_dialog.vbox.pack_start(no_conn_dialog_label, True, True,
0)
        no_conn_dialog_label.show()
        self.no_conn_dialog.set_has_separator(False)

        # Run no connection error dialog window
        noconn_result = self.no_conn_dialog.run()
        # Destroy no connection error dialog window when OK is pressed
        if noconn_result == gtk.RESPONSE_OK:
            self.no_conn_dialog.destroy()


    # Perform check to determine which buttons to enable or disable
    def disable_buttons_check(self):
        unknown = re.search('Unknown', self.loc_name_label.get_text())
        # If location name does not exist in db
        if unknown:
            # Disable LocateMe button
            self.locate_user_button.set_sensitive(False)
            # Enable Add Location button and menu item
            self.add_loc_button.set_sensitive(True)
            self.add_loc_menuitem.set_sensitive(True)
        else:
            # Disable Add Location button and menu item
            self.add_loc_button.set_sensitive(False)
            self.add_loc_menuitem.set_sensitive(False)
            # Enable LocateMe button
            self.locate_user_button.set_sensitive(True)


    # Get list of locations from database and allows user to select from them
    def get_loc_list_combobox(self):
        # Create select location dialog window
        self.loc_list_dialog = gtk.Dialog("Select location:", self.main_window,
gtk.DIALOG_MODAL, (gtk.STOCK_OK, gtk.RESPONSE_OK, gtk.STOCK_CANCEL,
gtk.RESPONSE_CANCEL))
        self.loc_list_dialog.set_size_request(250, 130)

        # Create combobox within select location dialog window
        self.loc_list_combobox = gtk.combo_box_new_text()
        # Fill combobox list with values from db
        loc_list = db_funcs.db_get_loc_list()
        self.loc_list_combobox.append_text('Select a location:')
        for i in range(0,len(loc_list)):
            self.loc_list_combobox.append_text(loc_list[i])
        self.loc_list_combobox.set_active(0)

        self.loc_list_dialog.vbox.pack_start(self.loc_list_combobox, True,
False, 10)
        self.loc_list_combobox.show()
```

```
        # Run select location dialog window
        loclist_result = self.loc_list_dialog.run()
        # Destroy select location dialog window when Cancel button is pressed
        if loclist_result == gtk.RESPONSE_CANCEL:
            self.loc_list_dialog.destroy()
        # Get selected value from combolist when OK is pressed
        elif loclist_result == gtk.RESPONSE_OK:
            # Get selected value
            model = self.loc_list_combobox.get_model()
            index = self.loc_list_combobox.get_active()
            # If a value was selected, return it and destroy select location
dialog window
            if index:
                self.loc_list_dialog.destroy()
                return model[index][0]
            # If no value selected, destroy select location dialog window
            else:
                self.loc_list_dialog.destroy()


    # Callback function for LocateMe button
    def locate_user_cb(self, widget, data=None):
        # Get connection status
        connected = wlan_funcs.wlan_check_conn_status()
        if not connected:
            # Display no connection error
            self.get_no_conn_dialog()
        else:
            # Get location name
            loc_name = self.loc_name_label.get_text()
            # Open results window and pass in loc_name as parameter
            open_results = results_GUI.show_window(loc_name)
            # Hide main window
            self.main_window.hide()


    # Callback function for Change Location button
    def change_loc_cb(self, widget, data=None):
        # Call function to display Change Location window
        loc_name = self.get_loc_list_combobox()
        # If valid value returned
        if loc_name:
            # Update main window with new values
            self.loc_name_label.set_text(loc_name)
            self.last_gpsloc_label.set_text("Your chosen location is:")
            self.lat_long_label.set_text("")
        # Call function to check which buttons to enable or disable
        self.disable_buttons_check()


    # Callback function for Add Location button
    def add_loc_cb(self, widget, data=None):
        global geocode
        # Get connection status
        connected = wlan_funcs.wlan_check_conn_status()
        if not connected:
            # Display no connection error
            self.get_no_conn_dialog()
        else:
            # Create add location dialog window
            self.add_loc_dialog = gtk.Dialog("Add new location:",
self.main_window, gtk.DIALOG_MODAL, ("Browse", gtk.RESPONSE_APPLY, "Add",
gtk.RESPONSE_ACCEPT, gtk.STOCK_CANCEL, gtk.RESPONSE_CANCEL))
```

```
            # Retrieve essid value of current connection
            essid = wlan_funcs.wlan_get_conn_essid()
            str = "Geocode = %s\nESSID = %s" %(geocode, essid)
            str = str.strip()
            add_loc_dialog_label = gtk.Label(str)
            self.add_loc_dialog.vbox.pack_start(add_loc_dialog_label, True,
True, 5)
            add_loc_dialog_label.set_alignment(0, 0.5)
            add_loc_dialog_label.show()

            # Create label
            loc_label = gtk.Label("Location:")
            self.add_loc_dialog.vbox.pack_start(loc_label, True, True, 5)
            loc_label.set_alignment(0, 0.5)
            loc_label.show()

            # Create text entry field
            loc_name_entry = gtk.Entry()
            self.add_loc_dialog.vbox.pack_start(loc_name_entry, True, True, 5)
            loc_name_entry.show()

            while True:
                # Run add location dialog box
                addloc_result = self.add_loc_dialog.run()
                # Exit infinite loop when Cancel is pressed
                if addloc_result == gtk.RESPONSE_CANCEL:
                    break
                # Display location list combobox when Browse is pressed
                elif addloc_result == gtk.RESPONSE_APPLY:
                    loc_name = self.get_loc_list_combobox()
                    # If valid value returned, set text entry field to value
returned
                    if loc_name:
                        loc_name_entry.set_text(loc_name)
                # Add location to db when Add is pressed
                elif addloc_result == gtk.RESPONSE_ACCEPT:
                    loc_name = loc_name_entry.get_text()
                    loc_name = loc_name.strip()
                    # If description field is empty, display error message
                    if loc_name == "":
                        # Create description empty error dialog window
                        self.empty_field_dialog = gtk.Dialog("Error!",
self.main_window, gtk.DIALOG_MODAL, (gtk.STOCK_OK, gtk.RESPONSE_OK))
                        empty_field_dialog_label = gtk.Label("Please enter or
select a location name.")
                        empty_field_dialog_label.set_line_wrap(True)

self.empty_field_dialog.vbox.pack_start(empty_field_dialog_label, True, True,
0)
                        self.empty_field_dialog.set_has_separator(False)
                        self.empty_field_dialog.show_all()

                        # Run description empty error dialog window
                        emptyfield_result = self.empty_field_dialog.run()
                        # Destroy description empty error dialog window when OK
is pressed
                        if emptyfield_result == gtk.RESPONSE_OK:
                            self.empty_field_dialog.destroy()

                    # If description field not empty
                    else:
                        # Check if description already exists in db
```

```
                        exists = 0
                        loc_list = db_funcs.db_get_loc_list()
                        for i in range(0, len(loc_list)):
                            match = re.match(loc_name, loc_list[i])
                            # If description exists
                            if match:
                                exists = 1
                                # Add geocode value to existing location
                                db_funcs.db_add_geocode_to_loc(loc_name,
geocode)
                                break
                        # If description does not exist
                        if not exists:
                            # Add location name, geocode value and essid to db
                            db_funcs.db_add_new_loc(loc_name, geocode, essid)
                        break

            # Update main window
            loc_name = db_funcs.db_get_loc_name(geocode)
            self.loc_name_label.set_text(loc_name)
            self.disable_buttons_check()
            # Destroy add location dialog window
            self.add_loc_dialog.destroy()


    # Callback function for Edit Data menu item
    def edit_data_cb(self, widget, data=None):
        global geocode, lat_long_label_str
        # Create edit location dialog window
        edit_data_dialog = gtk.Dialog("Locations", self.main_window,
gtk.DIALOG_MODAL, ("Edit", gtk.RESPONSE_OK, "Delete", gtk.RESPONSE_ACCEPT,
gtk.STOCK_CANCEL, gtk.RESPONSE_CANCEL))
        edit_data_dialog.set_size_request(400, 250)
        # Create list widget within edit location dialog window
        loc_liststore = gtk.ListStore(str)
        # Get list of all locations in db and display in list
        loc_list = db_funcs.db_get_loc_list()
        for i in range(0, len(loc_list)):
            loc_liststore.append(['%s' %loc_list[i]])
        loc_listview = gtk.TreeView(loc_liststore)
        loc_column = gtk.TreeViewColumn('Location')
        loc_listview.append_column(loc_column)
        loc_cell = gtk.CellRendererText()
        loc_column.pack_start(loc_cell, True)
        loc_column.add_attribute(loc_cell, 'text', 0)

        loc_listsel = loc_listview.get_selection()
        # Allow only one value to be selected at any given time
        loc_listsel.set_mode(gtk.SELECTION_SINGLE)

        # Create scrolled window for list
        scroll_window = gtk.ScrolledWindow()
        scroll_window.add(loc_listview)

        edit_data_dialog.vbox.pack_start(scroll_window, True, True, 0)
        scroll_window.set_policy(gtk.POLICY_AUTOMATIC, gtk.POLICY_AUTOMATIC)
        loc_listview.show()
        scroll_window.show()

        while True:
            # Run edit location dialog window
            loc_list_result = edit_data_dialog.run()
            # Exit infinite loop when Cancel is pressed
```

```
            if loc_list_result == gtk.RESPONSE_CANCEL:
                break
            # Display geocode values for selected location when Edit is pressed
            elif loc_list_result == gtk.RESPONSE_OK:
                # Get selection
                (model, iter) = loc_listsel.get_selected()
                # If none selected
                if iter == None:
                    # Display error message
                    self.get_none_sel_dialog()
                else:
                    # Get value of selection
                    loc = model.get_value(iter, 0)
                    # Create edit location geocode dialog window
                    edit_loc_geocode_dialog = gtk.Dialog("Geocodes-Lat/Long",
self.main_window, gtk.DIALOG_MODAL, ("Delete", gtk.RESPONSE_ACCEPT,
gtk.STOCK_CANCEL, gtk.RESPONSE_CANCEL))
                    edit_loc_geocode_dialog.set_size_request(300, 250)
                    # Create list widget within edit location geocode dialog
window
                    loc_geocode_liststore = gtk.ListStore(str)
                    # Get list of geocode values corresponding to selected
location and add to list widget
                    loc_geocode_list = db_funcs.db_get_loc_geocode_list(loc)
                    for i in range(0, len(loc_geocode_list)):
                        loc_geocode_liststore.append(['%s'
%loc_geocode_list[i]])
                    loc_geocode_listview = gtk.TreeView(loc_geocode_liststore)
                    loc_geocode_column = gtk.TreeViewColumn()
                    loc_geocode_listview.append_column(loc_geocode_column)
                    loc_geocode_cell = gtk.CellRendererText()
                    loc_geocode_column.pack_start(loc_geocode_cell, True)
                    loc_geocode_column.set_attributes(loc_geocode_cell, text=0)

                    loc_geocode_listsel = loc_geocode_listview.get_selection()
                    # Allow only one value to be selected at any given time
                    loc_geocode_listsel.set_mode(gtk.SELECTION_SINGLE)

                    # Create scrolled window for list
                    scroll_window = gtk.ScrolledWindow()
                    scroll_window.add(loc_geocode_listview)

                    edit_loc_geocode_dialog.vbox.pack_start(scroll_window,
True, True, 0)
                    scroll_window.set_policy(gtk.POLICY_AUTOMATIC,
gtk.POLICY_AUTOMATIC)
                    loc_geocode_listview.show()
                    scroll_window.show()

                    while True:
                        # Run edit location geocode dialog window
                        edit_geocode = edit_loc_geocode_dialog.run()
                        # Exit infinite loop when Cancel is pressed
                        if edit_geocode == gtk.RESPONSE_CANCEL:
                            break
                        # Delete selected value when Delete is pressed
                        elif edit_geocode == gtk.RESPONSE_ACCEPT:
                            # Get selection
                            (geocode_model, geocode_iter) =
loc_geocode_listsel.get_selected()
                            # If none selected
                            if geocode_iter == None:
                                # Display error message
```

```
                                self.get_none_sel_dialog()
                            else:
                                # Get value of selection
                                geocode = geocode_model.get_value(geocode_iter,
0)
                                # Create delete confirmation dialog window
                                confirm_del_dialog = gtk.Dialog("Delete?",
self.main_window, gtk.DIALOG_MODAL, (gtk.STOCK_YES, gtk.RESPONSE_YES,
gtk.STOCK_NO, gtk.RESPONSE_NO))
                                confirm_del_dialog_label = gtk.Label("Are you
sure you want to delete this?")

confirm_del_dialog.vbox.pack_start(confirm_del_dialog_label, True, True, 0)
                                confirm_del_dialog.set_has_separator(False)
                                confirm_del_dialog.show_all()

                                # Run delete confirmation dialog window
                                confirm_del_result = confirm_del_dialog.run()
                                # Remove value from db and list when Yes is
pressed
                                if confirm_del_result == gtk.RESPONSE_YES:
                                    del_loc = db_funcs.db_remove_geocode(loc,
geocode)
                                    geocode_model.remove(geocode_iter)
                                    confirm_del_dialog.destroy()
                                    if del_loc == True:
                                        model.remove(iter)
                                # Destroy delete confirmation dialog window
when No is pressed
                                elif confirm_del_result == gtk.RESPONSE_NO:
                                    confirm_del_dialog.destroy()
                    # Destroy edit location geocode dialog window
                    edit_loc_geocode_dialog.destroy()

            # Delete selected location from db and list when Delete is pressed
            elif loc_list_result == gtk.RESPONSE_ACCEPT:
                # Get selection
                (model, iter) = loc_listsel.get_selected()
                # If none selected
                if iter == None:
                    # Display error message
                    self.get_none_sel_dialog()
                else:
                    # Create delete confirmation dialog window
                    confirm_del_dialog = gtk.Dialog("Delete?",
self.main_window, gtk.DIALOG_MODAL, (gtk.STOCK_YES, gtk.RESPONSE_YES,
gtk.STOCK_NO, gtk.RESPONSE_NO))
                    confirm_del_dialog_label = gtk.Label("Are you sure you want
to delete this?")

confirm_del_dialog.vbox.pack_start(confirm_del_dialog_label, True, True, 0)
                    confirm_del_dialog.set_has_separator(False)
                    confirm_del_dialog.show_all()

                    # Run delete confirmation dialog window
                    confirm_del_result = confirm_del_dialog.run()
                     # Remove value from db and list when Yes is pressed
                    if confirm_del_result == gtk.RESPONSE_YES:
                        db_funcs.db_remove_loc(model.get_value(iter, 0))
                        model.remove(iter)
                        confirm_del_dialog.destroy()
                    # Destroy delete confirmation dialog window when No is
pressed
```

```
                    elif confirm_del_result == gtk.RESPONSE_NO:
                        confirm_del_dialog.destroy()

            # Update main window
            loc_name = db_funcs.db_get_loc_name(geocode)
            self.loc_name_label.set_text(loc_name)
            self.last_gpsloc_label.set_text("Your last GPS location is:")
            self.lat_long_label.set_text(lat_long_label_str)
            self.disable_buttons_check()
            # Destroy edit location dialog window
            edit_data_dialog.destroy()


        # Callback function for Exit menu item
        def destroy_cb(self, widget, data=None):
            # Call function to close db
            db_funcs.db_close()
            # Exit application
            gtk.main_quit()


        # Run application
        def run(self):
            gtk.main()


    # Initialization
    if __name__ == "__main__":
        db_funcs.db_init()
        gps_funcs.gps_init()
        wlan_funcs.wlan_init_connection()
        app = mainGUI()
        app.run()
```

B.2 *results_GUI.py*

```
#!/usr/bin/env python2.5

#############################################################################
#
#    LocateME! - Finds location of user in a specific building using contextual
#    information
#    Copyright (C) <2009>  <Kattie Tay>
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#    the Free Software Foundation, either version 3 of the License, or
#    (at your option) any later version.
#
#    This program is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU General Public License for more details.
#
#    You should have received a copy of the GNU General Public License
#    along with this program.  If not, see <http://www.gnu.org/licenses/>
#
#############################################################################


# Import packages
import pygtk
pygtk.require('2.0')
import gtk
import hildon
import pango
import webbrowser
import time
import re
import main_GUI
import gps_funcs
import wlan_funcs
import db_funcs


# Initialize global variables
rloc_name = ""
results_window = 0
loc_essid = ""
curr_iap_list = []
curr_light_int = ""


# Open results GUI window
def show_window(loc_name):
    global rloc_name
    # Set passed in value as global variable rloc_name
    rloc_name = loc_name
    # Call function to initialize GUI
    init_resultsGUI()


# Initialize results GUI window
def init_resultsGUI():
    global rloc_name, results_window, loc_essid, curr_iap_list, curr_light_int
```

```
    # --- Results window ---
    # Create results window
    results_window = hildon.Window()
    results_window.connect("destroy", return_to_main_cb, None)
    results_window.set_border_width(10)
    results_window.set_title("LocateME! Results")

    # Create table layout
    rtable_layout = gtk.Table(10, 10, True)
    results_window.add(rtable_layout)


    # --- Location name label ---
    # Create label to display location name
    rloc_name_label_str = "Location: %s" %rloc_name
    rloc_name_label = gtk.Label(rloc_name_label_str)
    rloc_name_label.modify_font(pango.FontDescription("Bold"))

    rtable_layout.attach(rloc_name_label, 1, 9, 1, 2, gtk.EXPAND, gtk.EXPAND,
5, 5)
    rloc_name_label.set_alignment(0, 0)
    rloc_name_label.show()


    # --- Scan results label ---
    # Create label to display scan results
    rscan_results_label1_str = "You are nearest to: "
    rscan_results_label1 = gtk.Label(rscan_results_label1_str)
    rtable_layout.attach(rscan_results_label1, 1, 9, 2, 3, gtk.EXPAND,
gtk.EXPAND, 5, 5)
    rscan_results_label1.set_alignment(0.5, 0.5)
    rscan_results_label1.show()

    # Call function to get scan results
    rscan_results_label2_str, loc_essid, curr_iap_list, curr_light_int =
get_scan_results()
    # Display results in window
    rscan_results_label2 = gtk.Label(rscan_results_label2_str)

    # Set label attributes
    rscan_results_label2_attr = pango.AttrList()
    fg_color = pango.AttrForeground(30000, 0, 30000, 0, 5000)
    size = pango.AttrSize(30000, 0, -1)
    weight = pango.AttrWeight(pango.WEIGHT_BOLD, 0, 5000)
    rscan_results_label2_attr.insert(fg_color)
    rscan_results_label2_attr.insert(size)
    rscan_results_label2_attr.insert(weight)
    rscan_results_label2.set_attributes(rscan_results_label2_attr)
    rscan_results_label2.set_alignment(0.5, 0.5)

    rtable_layout.attach(rscan_results_label2, 1, 9, 3, 5, gtk.FILL,
gtk.EXPAND, 5, 5)
    rscan_results_label2.show()


    # --- Details of current point button ---
    # Create button to show details of current location
    curr_loc_details_button_str = "Details of current point"
    curr_loc_details_button = gtk.Button(curr_loc_details_button_str)
    curr_loc_details_button.connect("clicked", curr_loc_details_cb, None)
    curr_loc_details_button.set_alignment(0.5, 0.5)
```

```
    rtable_layout.attach(curr_loc_details_button, 1, 5, 5, 7, gtk.EXPAND,
gtk.FILL, 5, 20)
    curr_loc_details_button.show()


    # --- Show location on map button ---
    # Create button to show location on map
    show_loc_on_map_button_str = "Show location on map"
    show_loc_on_map_button = gtk.Button(show_loc_on_map_button_str)
    show_loc_on_map_button.connect("clicked", show_loc_on_map_cb, None)
    show_loc_on_map_button.set_alignment(0.5, 0.5)

    rtable_layout.attach(show_loc_on_map_button, 5, 9, 5, 7, gtk.EXPAND,
gtk.FILL, 5, 20)
    show_loc_on_map_button.show()


    # --- Edit Data button ---
    # Create button to edit data
    redit_data_button_str = "Edit Data"
    redit_data_button = gtk.Button(redit_data_button_str)
    redit_data_button.connect("clicked", edit_data_cb, None)
    redit_data_button.set_alignment(0.5, 0.5)

    # Set button label attributes
    redit_data_label = redit_data_button.child
    redit_data_label.modify_font(pango.FontDescription("Bold"))

    rtable_layout.attach(redit_data_button, 0, 3, 7, 10, gtk.FILL, gtk.FILL,
10, 20)
    redit_data_button.show()


    # --- RelocateME button ---
    # Create button to re-run scan to find location of user
    relocateme_button_str = "Re-LocateME!"
    relocateme_button = gtk.Button(relocateme_button_str)
    relocateme_button.connect("clicked", relocateme_cb, rscan_results_label2)
    relocateme_button.set_alignment(0.5, 0.5)

    # Set button label attributes
    relocateme_label = relocateme_button.child
    relocateme_label.modify_font(pango.FontDescription("Bold 24"))

    rtable_layout.attach(relocateme_button, 3, 7, 7, 10, gtk.FILL, gtk.FILL,
10, 20)
    relocateme_button.show()


    # --- Return to Main button ---
    # Create button to return user to main screen
    return_to_main_button_str = "Return to Main"
    return_to_main_button = gtk.Button(return_to_main_button_str)
    return_to_main_button.connect("clicked", destroy_cb, None)
    return_to_main_button.set_alignment(0.5, 0.5)

    # Set button label attributes
    return_to_main_label = return_to_main_button.child
    return_to_main_label.modify_font(pango.FontDescription("Bold"))

    rtable_layout.attach(return_to_main_button, 7, 10, 7, 10, gtk.FILL,
gtk.FILL, 10, 20)
    return_to_main_button.show()
```

```
    # --- Menu ---
    # Set up the menu bar
    roptions_menu = gtk.Menu()
    # Create menu items
    relocateme_menuitem = gtk.MenuItem("Re-LocateME!")
    return_to_main_menuitem = gtk.MenuItem("Return to Main")
    edit_data_menuitem = gtk.MenuItem("Edit Data")
    exit_menuitem = gtk.MenuItem("Exit")

    roptions_menu.append(relocateme_menuitem)
    roptions_menu.append(return_to_main_menuitem)
    roptions_menu.append(edit_data_menuitem)
    roptions_menu.append(exit_menuitem)

    relocateme_menuitem.connect("activate", relocateme_cb,
rscan_results_label2)
    return_to_main_menuitem.connect_object("activate", destroy_cb, None)
    edit_data_menuitem.connect_object("activate", edit_data_cb, None)
    exit_menuitem.connect_object("activate", exit_cb, None)

    relocateme_menuitem.show()
    return_to_main_menuitem.show()
    edit_data_menuitem.show()
    exit_menuitem.show()

    results_window.set_menu(roptions_menu)


    # Show results window
    results_window.show_all()


# Callback function for Details button
def curr_loc_details_cb(widget, data=None):
    global results_window, loc_essid, curr_iap_list, curr_light_int
    # Create result details dialog window
    details_dialog = gtk.Dialog("Result Details", results_window,
gtk.DIALOG_MODAL, (gtk.STOCK_CANCEL,

gtk.RESPONSE_CANCEL))
    details_dialog.set_size_request(320, 250)
    # Display essid, light intensity and iap addr with sig strength in window
    details_dialog_label_str = "ESSID: %s\nLight intensity: %s\nIAP list:\n"
%(loc_essid, curr_light_int)

    for i in range(0, len(curr_iap_list)):
        details_dialog_label_str = details_dialog_label_str +
curr_iap_list[i][0] + "\t" + curr_iap_list[i][1] + " dBm\n"

    details_dialog_label = gtk.Label(details_dialog_label_str)
    details_dialog_label.set_line_wrap(True)
    scroll_window = gtk.ScrolledWindow()
    scroll_window.add_with_viewport(details_dialog_label)
    details_dialog.vbox.pack_start(scroll_window, True, True, 0)
    details_dialog_label.show()
    scroll_window.set_policy(gtk.POLICY_AUTOMATIC, gtk.POLICY_AUTOMATIC)
    scroll_window.show()

    # Run result details dialog window
    result = details_dialog.run()
    # Destroy result details dialog window when Cancel is pressed
```

```
    if result == gtk.RESPONSE_CANCEL:
        details_dialog.destroy()


# Callback function for Show Location On Map button
def show_loc_on_map_cb(widget, data=None):
    global rloc_name
    # Get connection status
    connected = wlan_funcs.wlan_check_conn_status()
    if not connected:
        # Display error message
        get_no_conn_dialog()
    else:
        # Get corresponding geocode value of location
        geocode = db_funcs.db_get_loc_geocode(rloc_name)
        # Call function to get url of map
        map_addr = gps_funcs.gps_get_map_addr(geocode)
        # Open url in new browser window
        webbrowser.open_new(map_addr)


# Callback function for ReLocateMe button and menu item
def relocateme_cb(widget, rscan_results_label2):
    global loc_essid, curr_iap_list, curr_light_int
    # Get connection status
    connected = wlan_funcs.wlan_check_conn_status()
    if not connected:
        # Display error message
        get_no_conn_dialog()
    else:
        # Call function to get scan results
        loc_desc, loc_essid, curr_iap_list, curr_light_int = get_scan_results()
        # Update results window with description of current position
        rscan_results_label2.set_text(loc_desc)


# Callback function for closing window and re-opening main window
def return_to_main_cb(widget, data=None):
    global results_window
    # Destroy current window
    results_window.destroy()
    # Re-open main window
    main_GUI.mainGUI()


# Callback function to destroy window
def destroy_cb(widget, data=None):
    global results_window
    # Destroy results window
    results_window.destroy()


# Callback function for Edit Data button and menu item
def edit_data_cb(widget, data=None):
    global results_window, rloc_name, loc_essid
    # Create edit scan pts dialog window
    edit_data_dialog = gtk.Dialog("List of Scan Points", results_window,
gtk.DIALOG_MODAL, ("Add", gtk.RESPONSE_APPLY, "Edit", gtk.RESPONSE_OK,
"Delete", gtk.RESPONSE_ACCEPT, gtk.STOCK_CANCEL, gtk.RESPONSE_CANCEL))
    edit_data_dialog.set_size_request(480, 250)
    # Create list within edit scan pts dialog window
    desc_liststore = gtk.ListStore(str)
    # Get list of all descriptions corresponding to rloc_name and display
```

```
    desc_list = db_funcs.db_get_loc_desc_list(rloc_name)
    for i in range(0, len(desc_list)):
        desc_liststore.append(['%s' %desc_list[i]])
    desc_listview = gtk.TreeView(desc_liststore)
    desc_column = gtk.TreeViewColumn('Description')
    desc_listview.append_column(desc_column)
    desc_cell = gtk.CellRendererText()
    desc_column.pack_start(desc_cell, True)
    desc_column.add_attribute(desc_cell, 'text', 0)

    desc_listsel = desc_listview.get_selection()
    # Allow only one value to be selected at any given time
    desc_listsel.set_mode(gtk.SELECTION_SINGLE)

    # Create scrolled window
    scroll_window = gtk.ScrolledWindow()
    scroll_window.add(desc_listview)

    edit_data_dialog.vbox.pack_start(scroll_window, True, True, 0)
    scroll_window.set_policy(gtk.POLICY_AUTOMATIC, gtk.POLICY_AUTOMATIC)
    desc_listview.show()
    scroll_window.show()

    while True:
        # Run edit scan pts dialog window
        desc_list_result = edit_data_dialog.run()
        # Exit infinite loop whee Cancel is pressed
        if desc_list_result == gtk.RESPONSE_CANCEL:
            break
        # Display add data window when Add Data is pressed
        elif desc_list_result == gtk.RESPONSE_APPLY:
            # Call function to display add data window
            add_new_desc = get_add_edit_dialog(None, "Add Data")
            # If valid value is returned, add desc to list
            if not add_new_desc == None:
                desc_liststore.append(['%s' %add_new_desc])
        # Display edit data window when Edit Data is pressed
        elif desc_list_result == gtk.RESPONSE_OK:
            # Get selection
            (model, iter) = desc_listsel.get_selected()
            # If none selected
            if iter == None:
                # Display error message
                get_none_sel_dialog()
            else:
                # Get selected value
                desc = model.get_value(iter, 0)
                # Call function to display edit data window
                get_add_edit_dialog(desc, "Edit Data")
        # Delete selected value when Delete is pressed
        elif desc_list_result == gtk.RESPONSE_ACCEPT:
            # Get selection
            (model, iter) = desc_listsel.get_selected()
            # If none selected
            if iter == None:
                # Display error message
                get_none_sel_dialog()
            else:
                # Create delete confirmation dialog window
                confirm_del_dialog = gtk.Dialog("Delete?", results_window,
gtk.DIALOG_MODAL, (gtk.STOCK_YES, gtk.RESPONSE_YES, gtk.STOCK_NO,
gtk.RESPONSE_NO))
```

```
                confirm_del_dialog_label = gtk.Label("Are you sure you want to
delete this?")
                confirm_del_dialog.vbox.pack_start(confirm_del_dialog_label,
True, True, 0)
                confirm_del_dialog.set_has_separator(False)
                confirm_del_dialog.show_all()

                # Run delete confirmation dialog window
                confirm_del_result = confirm_del_dialog.run()
                # Remove selected value from db and list when Yes is pressed
                if confirm_del_result == gtk.RESPONSE_YES:
                    # Remove selected value from db
                    db_funcs.db_remove_desc(model.get_value(iter, 0))
                    # Remove selected value from list
                    model.remove(iter)
                    # Destroy delete confirmation dialog window
                    confirm_del_dialog.destroy()
                # Destroy delete confirmation dialog window when No is pressed
                elif confirm_del_result == gtk.RESPONSE_NO:
                    confirm_del_dialog.destroy()

    # Destroy edit scan pts dialog window
    edit_data_dialog.destroy()


# Callback function for Exit menu item
def exit_cb(widget, data=None):
    # Exit application
    gtk.main_quit()


# Get add or edit dialog box
def get_add_edit_dialog(desc, type):
    global rloc_name, results_window, loc_essid
    # Create add or edit data dialog window
    add_edit_dialog = gtk.Dialog(type, results_window, gtk.DIALOG_MODAL,
("Scan", gtk.RESPONSE_ACCEPT, "Save", gtk.RESPONSE_OK, gtk.STOCK_CANCEL,
gtk.RESPONSE_CANCEL))
    add_edit_dialog.set_size_request(350, 250)

    if type == "Add Data":
        # Create an entry text field for users to enter description
        add_entry_label = gtk.Label("Description: ")
        add_edit_dialog.vbox.pack_start(add_entry_label, True, True, 5)
        add_entry_label.set_alignment(0, 0.5)
        add_entry_label.show()
        add_entry = gtk.Entry()
        add_edit_dialog.vbox.pack_start(add_entry, True, True, 5)
        add_entry.show()
        add_edit_dialog_label_str = ""
    elif type == "Edit Data":
        # Get details of scan pt corresponding to rloc_name and desc
        desc_iap_list, desc_light_int = db_funcs.db_get_desc_details(rloc_name,
desc)
        add_edit_dialog_label1_str = "ESSID: %s\nLight intensity: %s\nIAP
list:\n" %(loc_essid, desc_light_int)
        add_edit_dialog_label2_str = ""
        for i in range(0, len(desc_iap_list)):
            add_edit_dialog_label2_str = add_edit_dialog_label2_str +
desc_iap_list[i][0] + "\t" + desc_iap_list[i][1] + " dBm\n"

        add_edit_dialog_label_str = add_edit_dialog_label1_str +
add_edit_dialog_label2_str
```

```
    add_edit_dialog_label = gtk.Label(add_edit_dialog_label_str)
    add_edit_dialog_label.set_line_wrap(True)
    scroll_window = gtk.ScrolledWindow()
    scroll_window.add_with_viewport(add_edit_dialog_label)
    add_edit_dialog.vbox.pack_start(scroll_window, True, True, 0)
    add_edit_dialog_label.show()
    scroll_window.set_policy(gtk.POLICY_AUTOMATIC, gtk.POLICY_AUTOMATIC)
    scroll_window.show()
    # Disable Save button
    add_edit_dialog.set_response_sensitive(gtk.RESPONSE_OK, False)

    while True:
        # Run add or edit data dialog window
        add_edit_result = add_edit_dialog.run()
        # Exit infinite loop when Cancel is pressed
        if add_edit_result == gtk.RESPONSE_CANCEL:
            break
        # Scan for new data when Scan is pressed
        elif add_edit_result == gtk.RESPONSE_ACCEPT:
            # Get connection status
            connected = wlan_funcs.wlan_check_conn_status()
            if not connected:
                # Display error message
                get_no_conn_dialog()
            else:
                # Get new wlan data
                new_desc_iap_list = wlan_funcs.wlan_get_iap_stats(loc_essid)
                # Get new light intensity data
                new_desc_light_int = get_light_intensity()
                add_edit_dialog_label1_str = "ESSID: %s\nLight intensity:
%s\nIAP list:\n" %(loc_essid, new_desc_light_int)
                add_edit_dialog_label2_str = ""
                for i in range(0, len(new_desc_iap_list)):
                    add_edit_dialog_label2_str = add_edit_dialog_label2_str +
new_desc_iap_list[i][0] + "\t" + new_desc_iap_list[i][1] + " dBm\n"

                add_edit_dialog_label_str = add_edit_dialog_label1_str +
add_edit_dialog_label2_str
                add_edit_dialog_label.set_text(add_edit_dialog_label_str)
                # Enable Save button
                add_edit_dialog.set_response_sensitive(gtk.RESPONSE_OK, True)
        # Save values when Save is pressed
        elif add_edit_result == gtk.RESPONSE_OK:
            if type == "Add Data":
                # Get description entered in entry field
                add_new_desc = add_entry.get_text()
                add_new_desc = add_new_desc.strip()
                if add_new_desc == "":
                    # Create empty field dialog window
                    empty_field_dialog = gtk.Dialog("Error!", results_window,
gtk.DIALOG_MODAL, (gtk.STOCK_OK, gtk.RESPONSE_OK))
                    empty_field_dialog_label = gtk.Label("Please enter a
description for this scan point.")

empty_field_dialog.vbox.pack_start(empty_field_dialog_label, True, True, 0)
                    empty_field_dialog.set_has_separator(False)
                    empty_field_dialog.show_all()
                    # Run empty field dialog window
                    emptyfield_result = empty_field_dialog.run()
                    # Destroy empty field dialog window when OK is pressed
                    if emptyfield_result == gtk.RESPONSE_OK:
                        empty_field_dialog.destroy()
```

```
                else:
                    # Check if description exists
                    exists = 0
                    desc_list = db_funcs.db_get_loc_desc_list(rloc_name)
                    for i in range(0, len(desc_list)):
                        match = re.match(add_new_desc, desc_list[i])
                        # If description exists
                        if match:
                            exists = 1
                            # Create description exists dialog window
                            desc_exists_dialog = gtk.Dialog("Error!",
results_window, gtk.DIALOG_MODAL, (gtk.STOCK_OK, gtk.RESPONSE_OK))
                            desc_exists_dialog_label = gtk.Label("The
description entered already exists.\nPlease enter a new description.")
                            desc_exists_dialog_label.set_line_wrap(True)

desc_exists_dialog.vbox.pack_start(desc_exists_dialog_label, True, True, 0)
                            desc_exists_dialog.set_has_separator(False)
                            desc_exists_dialog.show_all()
                            # Run description exists dialog window
                            desc_exists_result = desc_exists_dialog.run()
                            # Destroy description exists dialog window when OK
is pressed
                            if desc_exists_result == gtk.RESPONSE_OK:
                                desc_exists_dialog.destroy()
                                # Exit for loop
                                break
                    if not exists:
                        # Add new scan pt to db
                        db_funcs.db_add_new_scanpt(rloc_name, add_new_desc,
new_desc_iap_list, new_desc_light_int)
                        # Destroy add or edit dialog window
                        add_edit_dialog.destroy()
                        # Return name of new desc
                        return add_new_desc
            elif type == "Edit Data":
                # Update scan pts data corresponding to rloc_name and desc
                db_funcs.db_edit_scanpt_details(rloc_name, desc,
new_desc_iap_list, new_desc_light_int)
                # Disable Save button
                add_edit_dialog.set_response_sensitive(gtk.RESPONSE_OK, False)
    # Destroy add or edit dialog window
    add_edit_dialog.destroy()


# Get no value selected dialog box
def get_none_sel_dialog():
    global results_window
    # Create no value selected dialog window
    none_sel_dialog = gtk.Dialog("Error!", results_window, gtk.DIALOG_MODAL,
(gtk.STOCK_OK, gtk.RESPONSE_OK))
    none_sel_dialog_label = gtk.Label("No value selected!")
    none_sel_dialog.vbox.pack_start(none_sel_dialog_label, True, True, 0)
    none_sel_dialog.set_has_separator(False)
    none_sel_dialog.show_all()

    # Run no value selected dialog window
    none_sel_dialog_result = none_sel_dialog.run()
    # Destroy no value selected dialog window when OK is pressed
    if none_sel_dialog_result == gtk.RESPONSE_OK:
        none_sel_dialog.destroy()
```

```
# Get no internet connection dialog box
def get_no_conn_dialog():
    global results_window
    # Create no connection dialog window
    no_conn_dialog = gtk.Dialog("No internet connection!", results_window,
gtk.DIALOG_MODAL, (gtk.STOCK_OK, gtk.RESPONSE_OK))
    no_conn_dialog_label = gtk.Label("Internet connection is required to
proceed. Please connect to the internet.")
    no_conn_dialog_label.set_line_wrap(True)
    no_conn_dialog.vbox.pack_start(no_conn_dialog_label, True, True, 0)
    no_conn_dialog_label.show()
    no_conn_dialog.set_has_separator(False)

    # Run no connection dialog window
    noconn_result = no_conn_dialog.run()
    # Destroy no connection dialog window when OK is pressed
    if noconn_result == gtk.RESPONSE_OK:
        no_conn_dialog.destroy()


# Get scan results
def get_scan_results():
    global rloc_name, loc_essid, curr_iap_list, curr_light_int
    # Get essid corresponding to rloc_name
    loc_essid = db_funcs.db_get_loc_essid(rloc_name)
    # Perform wlan scan at current point
    curr_iap_list = wlan_funcs.wlan_get_iap_stats(loc_essid)
    # Get light intensity at current point
    curr_light_int = get_light_intensity()

    # Get list of all scan pts data corresponding to rloc_name
    loc_all_iap_list, loc_light_int_list =
db_funcs.db_get_loc_iap_lightint_list(rloc_name)
    # Get list of all descriptions corresponding to rloc_name
    loc_all_desc_list = db_funcs.db_get_loc_desc_list(rloc_name)

    # Algorithm for finding location
    # If description list is empty
    if len(loc_all_desc_list) == 0:
        closest_desc = "No scan points available!"
    else:
        mse_list = []
        mse_idx = []
        # Loop through each row in loc_all_iap_list
        for i in range(0, len(loc_all_iap_list)):
            # Extract current row
            one_iap_list = loc_all_iap_list[i]
            mse_sigstrength = 0
            total_sq_err_sigstrength = 0
            count = 0

            # Loop through each iap in curr_iap_list
            for j in range(0, len(curr_iap_list)):
                # Compare it with each row in one_iap_list until match is found
                for k in range(0, len(one_iap_list)):
                    # If match
                    if curr_iap_list[j][0] in one_iap_list[k][0]:
                        # Get difference in signal strength
                        err_sigstrength = int(curr_iap_list[j][1]) -
int(one_iap_list[k][1])
                        # Square the error value
                        sq_err_sigstrength = err_sigstrength * err_sigstrength
```

```
                        # Add this squared error with the rest of the squared
errors
                        total_sq_err_sigstrength = total_sq_err_sigstrength +
sq_err_sigstrength
                        # Keeps track of number of matches
                        count = count + 1
                        # After a match is found, break out of this for loop
                        break

            # If there was at least 1 match
            if not count == 0:
                # Get difference in light int
                err_lightint = int(curr_light_int) - int(loc_light_int_list[i])
                # Square the error value
                sq_err_lightint = err_lightint * err_lightint
                # Get total mse
                total_mse = (total_sq_err_sigstrength + sq_err_lightint) /
(count+1)
                # Add this to mse_list
                mse_list.append(total_mse)
                # Use mse_idx to keep track of which mse belongs to which
description
                mse_idx.append(i)

        # If mse_list is empty
        if len(mse_list) == 0:
            closest_desc = "No matches found!"
        else:
            # Get the min value in the list
            min_mse = min(mse_list)
            # Get the index corresponding to the min value
            min_index = mse_list.index(min_mse)
            index = mse_idx[min_index]
            # Retrieve description which has lowest mse i.e. closest to current
point
            closest_desc = loc_all_desc_list[index]
    return closest_desc, loc_essid, curr_iap_list, curr_light_int

# Get light intensity
def get_light_intensity():
    # Open file containing light intensity data
    f = open('/sys/bus/i2c/devices/0-0029/lux', 'r')
    # If file exists
    if f:
        # Read light intensity value
        light_int = f.read()
        # Close file
        f.close()
    # Return light intensity value
    return light_int.strip()
```

B.3 *db_funcs.py*

```
#!/usr/bin/env python2.5

#############################################################################
#
#    LocateME! - Finds location of user in a specific building using contextual
#    information
#    Copyright (C) <2009>  <Kattie Tay>
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#    the Free Software Foundation, either version 3 of the License, or
#    (at your option) any later version.
#
#    This program is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU General Public License for more details.
#
#    You should have received a copy of the GNU General Public License
#    along with this program.  If not, see <http://www.gnu.org/licenses/>
#
#############################################################################


# Import packages
import sqlite3
import re


# Initialization of global variables
db_conn = None
db_cur = None


# Initialize database
def db_init():
    global db_conn, db_cur

    # Connect to the database
    # Create if it does not exist
    db_conn = sqlite3.connect('LocateMe_db.db')
    db_cur = db_conn.cursor()

    # Create the following tables if they do not exist
    # location table
    db_cur.execute("create table if not exists location(\
                    name text primary key, \
                    essid text)")

    # gps_data table
    db_cur.execute("create table if not exists gps_data(\
                    loc_name text, \
                    geocode text, \
                    primary key(loc_name,geocode), \
                    foreign key(loc_name) references location(loc_name) \
                    on delete cascade)")

    # scan_pts table
```

```
    db_cur.execute("create table if not exists scan_pts(\
                    loc_name text, \
                    desc text, \
                    iap_list text, \
                    light_int text, \
                    primary key(loc_name,desc), \
                    foreign key(loc_name) references location(loc_name) \
                    on delete cascade)")

    # Create trigger to enforce foreign key constraints
    db_cur.execute("create trigger if not exists delete_loc \
                    before delete on location\
                    begin \
                        delete from gps_data where loc_name = old.name; \
                        delete from scan_pts where loc_name = old.name; \
                    end;")


# Add new scan point to scan_pts table corresponding to specified location
def db_add_new_scanpt(loc_name, desc, iap_list, light_int):
    global db_conn, db_cur
    iap_list_str = db_convert_list_to_str(iap_list)
    db_cur.execute("insert into scan_pts values (?,?,?,?)", (loc_name, desc,
iap_list_str, light_int))
    db_conn.commit()


# Add new location name to location and gps_data tables
def db_add_new_loc(loc_name, geocode, essid):
    global db_conn, db_cur
    db_cur.execute("insert into location values (?,?)", (loc_name, essid))
    db_cur.execute("insert into gps_data values (?,?)", (loc_name, geocode))
    db_conn.commit()


# Add geocode values to specified location
def db_add_geocode_to_loc(loc_name, geocode):
    global db_conn, db_cur
    db_cur.execute("insert into gps_data values (?,?)", (loc_name, geocode))
    db_conn.commit()


# Edit data of specified location and description in scan_pts table
def db_edit_scanpt_details(loc_name, desc, iap_list, light_int):
    global db_conn, db_cur
    iap_list_str = db_convert_list_to_str(iap_list)
    db_cur.execute("update scan_pts set iap_list=?,light_int=? where loc_name=?
and desc=?", (iap_list_str, light_int, loc_name, desc))
    db_conn.commit()


# Remove specified geocode value from gps_data table
def db_remove_geocode(loc_name, geocode):
    global db_conn, db_cur
    del_loc = False
    db_cur.execute("delete from gps_data where geocode=?", (geocode,))
    # Check if all geocode values corresponding to specified location have been
deleted
    # If yes, delete specified location from location table and return True
    exists = db_cur.execute("select geocode from gps_data where loc_name=?",
(loc_name,)).fetchone()
    if exists == None:
        db_cur.execute("delete from location where name=?", (loc_name,))
```

```
            del_loc = True
        db_conn.commit()
        return del_loc


    # Remove specified location from location table
    def db_remove_loc(loc_name):
        global db_conn, db_cur
        db_cur.execute("delete from location where name=?", (loc_name,))
        db_conn.commit()


    # Remove specified scan point from scan_pts table
    def db_remove_desc(desc):
        global db_conn, db_cur
        db_cur.execute("delete from scan_pts where desc=?", (desc,))
        db_conn.commit()


    # Get scan point details of specified location and description
    def db_get_desc_details(loc_name, desc):
        global db_cur
        details = db_cur.execute("select iap_list,light_int from scan_pts where
    loc_name=? and desc=?", (loc_name, desc)).fetchone()
        if not details == None:
            iap_list = details[0]
            light_int = details[1]
            iap_list2 = db_convert_str_to_list(iap_list)
            return iap_list2, light_int


    # Get list of all iaps and light intensity value corresponding to specified
    location
    def db_get_loc_iap_lightint_list(loc_name):
        global db_cur
        all_iap_list = []
        all_iap_list2 = []
        light_int_list =[]
        db_cur.execute("select iap_list,light_int from scan_pts where loc_name=?",
    (loc_name,))
        for row in db_cur:
            all_iap_list.append(row[0])
            light_int_list.append(row[1])

        for i in range(0, len(all_iap_list)):
            all_iap_list2.append(db_convert_str_to_list(all_iap_list[i]))

        return all_iap_list2, light_int_list


    # Get list of all scan points corresponding to specified location
    def db_get_loc_desc_list(loc_name):
        global db_cur
        desc_list = []
        db_cur.execute("select desc from scan_pts where loc_name=?", (loc_name,))
        for row in db_cur:
            desc_list.append(row[0])
        return desc_list


    # Get list of all geocode values corresponding to specified location
    def db_get_loc_geocode_list(loc_name):
        global db_cur
```

```
    geocodes_list = []
    db_cur.execute("select geocode from gps_data where loc_name=?",
(loc_name,))
    for row in db_cur:
        geocodes_list.append(row[0])
    return geocodes_list


# Get geocode value corresponding to specified location (just one)
def db_get_loc_geocode(loc_name):
    global db_cur
    geocode = db_cur.execute("select geocode from gps_data where loc_name=?",
(loc_name,)).fetchone()
    if not geocode == None:
        return geocode[0]


# Get essid vale corresponding to specified location
def db_get_loc_essid(loc_name):
    global db_cur
    essid = db_cur.execute("select essid from location where name=?",
(loc_name,)).fetchone()
    if not essid == None:
        return essid[0]


# Get location corresponding to specified geocode
def db_get_loc_name(geocode):
    global db_cur
    loc_name = db_cur.execute("select loc_name from gps_data where geocode=?",
(geocode,)).fetchone()
    if loc_name == None:
        loc_name = "Unknown"
    else:
        loc_name = loc_name[0]
    return loc_name


# Get list of locations stored in location table
def db_get_loc_list():
    global db_cur
    loc_list = []
    db_cur.execute("select name from location")
    for row in db_cur:
        loc_list.append(row[0])
    return loc_list


# Close connection to database
def db_close():
    global db_conn
    db_conn.close()


# Convert format of data from list to string
def db_convert_list_to_str(list):
    str = ""
    for i in range(0, len(list)):
        str = str + "[\"" + list[i][0] + "," + list[i][1] + "\"]"
        if not i == len(list)-1:
            str = str + ";"
    return str
```

```
# Convert format of data from string to list
def db_convert_str_to_list(str):
    list = []
    str = str.split(";")
    print str
    for line in str:
        match = re.search('"(.*)(,)(.*)"', line)
        if match:
            list.append([match.group(1),match.group(3)])
    return list
```

B.4 *gps_funcs.py*

```
#!/usr/bin/env python2.5

#############################################################################
#
#    LocateME! - Finds location of user in a specific building using contextual
#    information
#    Copyright (C) <2009>  <Kattie Tay>
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#    the Free Software Foundation, either version 3 of the License, or
#    (at your option) any later version.
#
#    This program is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU General Public License for more details.
#
#    You should have received a copy of the GNU General Public License
#    along with this program.  If not, see <http://www.gnu.org/licenses/>
#
#############################################################################


# Import packages
import liblocation


# Initialize global variables
fix = 0
MAP_API_KEY = "ABQIAAAAuA3xWoVLqOF0jnHFSQL8ORTP1N2wIy6k-
xOQaacU0D_VGP3w4xTV7gHY3fLXeVugSTb-vUJDXQbQ_Q"


# Initialize GPS device
def gps_init():
    global fix
    gps = liblocation.gps_device_get_new()
    gps_struct = gps.struct()
    fix = gps_struct.fix


# Get latitude value of last saved GPS data
def gps_get_latitude():
    global fix
    if fix:
        latitude = fix.latitude
    return latitude


# Get longitude value of last saved GPS data
def gps_get_longitude():
    global fix
    if fix:
        longitude = fix.longitude
    return longitude
```

```
# Get map address for Google Static Maps
def gps_get_map_addr(geocode):
    global MAP_API_KEY
    # Set map parameters
    zoom = "zoom=16"
    marker = "markers=%s,orange" %geocode
    size = "size=600x600"
    maptype = "maptype=mobile"
    key = "key=%s" %MAP_API_KEY
    sensor = "sensor=true"
    map_addr = "http://maps.google.com/staticmap?%s&%s&%s&%s&%s&%s" %(zoom,
marker, size, maptype, key, sensor)
    return map_addr
```

B.5 *wlan_funcs.py*

```
#!/usr/bin/env python2.5

#############################################################################
#
#    LocateME! - Finds location of user in a specific building using contextual
#    information
#    Copyright (C) <2009>  <Kattie Tay>
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#    the Free Software Foundation, either version 3 of the License, or
#    (at your option) any later version.
#
#    This program is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU General Public License for more details.
#
#    You should have received a copy of the GNU General Public License
#    along with this program.  If not, see <http://www.gnu.org/licenses/>
#
#############################################################################


# Import packages
import subprocess
import re
import dbus
import gobject
import dbus.glib
import conic


# Initialize connection
def wlan_init_connection():
    bus = dbus.SystemBus(private=True)
    gobject.idle_add(wlan_connect)


# Create connection
# If device not connected, attempts to connect to default connection or
# display connection selection dialog box
def wlan_connect():
    # Creates the connection object and attach the handler.
    connection = conic.Connection()
    assert(connection.request_connection(conic.CONNECT_FLAG_NONE))


# Check if device is connected to Internet
# If yes, return True, else, return False
def wlan_check_conn_status():
    status = 1
    proc = subprocess.Popen('/sbin/iwlist wlan0 scan', shell=True,
stdout=subprocess.PIPE)
    stdout_str = proc.communicate()[0]
    if stdout_str == "":
        status = 0
    return status
```

```
# Get essid of current wlan connection
def wlan_get_conn_essid():
    proc = subprocess.Popen('sudo iwconfig wlan0', shell=True,
stdout=subprocess.PIPE)
    stdout_str = proc.communicate()[0]
    stdout_list = stdout_str.split('\n')
    for line in stdout_list:
        line = line.strip()
        match = re.search('ESSID:"(.*)"',line)
        if match:
            return match.group(1)


# Get wireless data corresponding to specified essid
def wlan_get_iap_stats(loc_essid):
    proc = subprocess.Popen('sudo iwlist wlan0 scan', shell=True,
stdout=subprocess.PIPE)
    stdout_str = proc.communicate()[0]
    stdout_list = stdout_str.split('\n')
    essid=[]
    address=[]
    sig_level=[]

    # Extract essid, address and signal strength of each access point in list
returned
    for line in stdout_list:
        line = line.strip()
        match = re.search('ESSID:"(.*)"', line)
        if match:
            essid.append(match.group(1))
        match = re.search('Address: (\S+)',line)
        if match:
            address.append(match.group(1))
        match = re.search('Signal level:(\S+) dBm',line)
        if match:
            sig_level.append(match.group(1))

    # Loop through list to find essid that matches specified essid
    # and extract corresponding address and signal strength
    results = []
    for i in range(0,len(essid)):
        if essid[i] == loc_essid:
            results.append([address[i], sig_level[i]])
    return results
```

B.6 *liblocation.py*

```
# Python wrapper to the Maemo 4.0 "Chinook" liblocation.
# Wrapper version 0.1.
#
# Copyright 2008 by Robert W. Brewer < rwb123 at gmail dot com >
# Licensed under GNU LGPL v3.
#
# This file is free software: you can redistribute it and/or modify
# it under the terms of the GNU Lesser General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This file is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU Lesser General Public License for more details.
#
# Please see <http://www.gnu.org/licenses/> for a copy of the
# GNU Lesser General Public License.

########################################
# For a documentation overview of liblocation please see:
# http://maemo.org/development/documentation/how-tos/4-
x/maemo_connectivity_guide.html#Location
########################################

import gobject
import ctypes as C
from types import MethodType

########################################
# constants
########################################

(STATUS_NO_FIX,
 STATUS_FIX,
 STATUS_DGPS_FIX) = range(3)

(MODE_NOT_SEEN,
 MODE_NO_FIX,
 MODE_2D,
 MODE_3D) = range(4)

NONE_SET     = 0
ALTITUDE_SET = 1<<0
SPEED_SET    = 1<<1
TRACK_SET    = 1<<2
CLIMB_SET    = 1<<3
LATLONG_SET  = 1<<4
TIME_SET     = 1<<5

########################################
# ctypes structure definitions
########################################

class GTypeInstance(C.Structure):
    _fields_ = [('g_class', C.c_ulong)]

class GObject(C.Structure):
```

```
    _fields_ = [('g_type_instance', GTypeInstance),
                ('ref_count', C.c_uint),
                ('qdata', C.c_void_p)]

class GPtrArray(C.Structure):
    _fields_ = [('pdata', C.c_void_p),
                ('len', C.c_uint)]

class LocationGPSDeviceSatellite(C.Structure):
    _fields_ = [('prn', C.c_int),
                ('elevation', C.c_int),
                ('azimuth', C.c_int),
                ('signal_strength', C.c_int),
                ('in_use', C.c_int)]

class LocationGPSDeviceFix(C.Structure):
    _fields_ = [('mode', C.c_int),
                ('fields', C.c_uint),
                ('time', C.c_double),
                ('ept', C.c_double),
                ('latitude', C.c_double),
                ('longitude', C.c_double),
                ('eph', C.c_double),
                ('altitude', C.c_double),
                ('epv', C.c_double),
                ('track', C.c_double),
                ('epd', C.c_double),
                ('speed', C.c_double),
                ('eps', C.c_double),
                ('climb', C.c_double),
                ('epc', C.c_double),
                # private, not used yet
                ('pitch', C.c_double),
                ('roll', C.c_double),
                ('dip', C.c_double)]

class CLocationGPSDevice(C.Structure):
    _fields_ = [('parent', GObject),
                ('online', C.c_int),
                ('status', C.c_int),
                ('Cfix', C.POINTER(LocationGPSDeviceFix)),
                ('satellites_in_view', C.c_int),
                ('satellites_in_use', C.c_int),
                ('Csatellites', C.POINTER(GPtrArray))]  # of
LocationGPSDeviceSatellite

    def sv_iter(self):
        if not self.Csatellites:
            return

        gar = self.Csatellites.contents
        sv_ptr_ptr = C.cast(gar.pdata,
                            C.POINTER(C.POINTER(LocationGPSDeviceSatellite)))

        for i in range(gar.len):
            yield sv_ptr_ptr[i].contents

    def __getattr__(self, name):
        try:
            return C.Structure.__getattr__(self)
        except AttributeError:
            if name == 'fix':
                if self.Cfix:
```

```
                    return self.Cfix.contents
                else:
                    return None
            if name == 'satellites':
                return self.sv_iter()
            raise AttributeError

class CLocationGPSDControl(C.Structure):
    _fields_ = [('parent', GObject),
                ('can_control', C.c_int)]

################################################
# gobject C->Python boilerplate from pygtk FAQ
################################################

# this boilerplate can convert a memory address
# into a proper python gobject.
class _PyGObject_Functions(C.Structure):
    _fields_ = [
        ('register_class',
         C.PYFUNCTYPE(C.c_void_p, C.c_char_p,
                           C.c_int, C.py_object,
                           C.py_object)),
        ('register_wrapper',
         C.PYFUNCTYPE(C.c_void_p, C.py_object)),
        ('register_sinkfunc',
         C.PYFUNCTYPE(C.py_object, C.c_void_p)),
        ('lookupclass',
         C.PYFUNCTYPE(C.py_object, C.c_int)),
        ('newgobj',
         C.PYFUNCTYPE(C.py_object, C.c_void_p)),
        ]

class PyGObjectCPAI(object):
    def __init__(self):
        addr = C.pythonapi.PyCObject_AsVoidPtr(
            C.py_object(gobject._PyGObject_API))
        self._api = _PyGObject_Functions.from_address(addr)

    def pygobject_new(self, addr):
        return self._api.newgobj(addr)

# call like this:
# Cgobject = PyGObjectCPAI()
# Cgobject.pygobject_new(memory_address)

# to get memory address from a gobject:
#  address = hash(obj)

###################################
# pythonized functions
###################################

def gps_device_get_type():
    return loc_gps_type()

def gps_device_get_new():

    def struct(self):
        ptr = C.cast(C.c_void_p(hash(self)),
                     C.POINTER(CLocationGPSDevice))
        return ptr.contents
```

```
    # create C gobject for gps device
    cgps_dev = gobj_new(gps_device_get_type(), None)

    # wrap in python gobject
    pyobj = Cgobject.pygobject_new(cgps_dev)

    # add a struct() method to hide the ctypes stuff.
    setattr(pyobj, 'struct', MethodType(struct, pyobj, pyobj.__class__))
    return pyobj

def gps_device_reset_last_known(gpsdevice):
    libloc.location_gps_device_reset_last_known(C.c_void_p(hash(gpsdevice)))

def gps_device_start(gpsdevice):
    libloc.location_gps_device_start(C.c_void_p(hash(gpsdevice)))

def gps_device_stop(gpsdevice):
    libloc.location_gps_device_stop(C.c_void_p(hash(gpsdevice)))

def gpsd_control_get_default():
    def struct(self):
        ptr = C.cast(C.c_void_p(hash(self)),
                     C.POINTER(CLocationGPSDControl))
        return ptr.contents

    gpsd_control_ptr = loc_gpsd_control()

    # wrap in python object
    pyobj = Cgobject.pygobject_new(gpsd_control_ptr)

    # add a struct() method to hide the ctypes stuff.
    setattr(pyobj, 'struct', MethodType(struct, pyobj, pyobj.__class__))
    return pyobj

def gpsd_control_start(gpsdcontrol):
    libloc.location_gpsd_control_start(C.c_void_p(hash(gpsdcontrol)))

def gpsd_control_stop(gpsdcontrol):
    libloc.location_gpsd_control_stop(C.c_void_p(hash(gpsdcontrol)))

def gpsd_control_request_status(gpsdcontrol):
    libloc.location_gpsd_control_request_status(C.c_void_p(hash(gpsdcontrol)))

########################################
# initialize library
########################################

# load C libraries
libloc = C.CDLL('liblocation.so.0')
libgobject = C.CDLL('libgobject-2.0.so.0')
Cgobject = PyGObjectCPAI()

# inform ctypes of necessary function prototype information

loc_gps_type = libloc.location_gps_device_get_type
loc_gps_type.restype = C.c_ulong

gobj_new = libgobject.g_object_new
gobj_new.restype = C.c_void_p

loc_gpsd_control = libloc.location_gpsd_control_get_default
loc_gpsd_control.restype = C.POINTER(CLocationGPSDControl)
```

```
libloc.location_distance_between.argtypes = [C.c_double,
                                             C.c_double,
                                             C.c_double,
                                             C.c_double]
libloc.location_distance_between.restype = C.c_double
```