

The University of New South Wales  
School of Electrical & Telecommunications Engineering

# Haze Watch: Database Server and Mobile Applications for Measuring and Evaluating Air Pollution Exposure

Thesis B Report

As part of ELEC4121

**Author:** Nikolaus Youdale (3186690)

Bachelor of Engineering

**Supervisor:** Vijay Sivaraman

**Date:** October 2010

Haze Watch: Database Server and Mobile  
**Thesis title:** Applications for Measuring and Evaluating Air Pollution Exposure      **Topic number:** VR25

**Student Name:** Nikolaus Youdale

**Student ID:** 3186690

**A. Problem statement**

Exposure to air pollution increases the incidence of various health problems, such as respiratory infections, heart disease and lung cancer. Current systems in place to monitor air pollution levels typically have only a handful of fixed monitoring stations distributed over an entire city. Such sparse monitoring cannot effectively monitor variations seen around pollution 'hotspots'. Medical studies are still searching for conclusive links between exposure to air pollution and various medical conditions. A system capable of monitoring an individual's exposure to various pollutants would provide very valuable data to such studies.
---

**B. Objective**

To design and implement the server infrastructure to support the collection and retrieval of air pollution data. To interface with pollution sensor hardware to facilitate the upload of readings to the server, via a Smartphone. And to build an iPhone application to conveniently monitor and report a user's pollution exposure.
---

**C. My solution**

Implement server using industry standard software.
Use Bluetooth on an Android Smartphone to interface with sensor hardware.
Use existing hardware on iPhone to implement a personal pollution exposure monitor

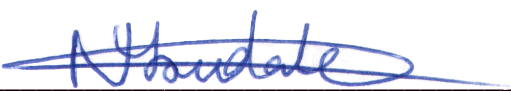
**D. Contributions** (at most one per line, most important first)

Bluetooth interface
Server architecture and implementation
iPhone application
Message format
Server optimizations
Proved functionality of above components

**E. Suggestions for future work**

User interface improvements, investigate issues with GPS signal loss, privacy concerns, extension to monitor other variables.

While I may have benefited from discussion with other people, I certify that this thesis is entirely my own work, except where appropriately documented acknowledgements are included.

Signature: 

Date: 20 / 10 / 2010

## Thesis Pointers

List relevant page numbers in the column on the left. Be precise and selective: Don't list all pages of your thesis!

6	Problem Statement
7-8	Objective

### Theory (up to 5 most relevant ideas)

17	Server
34	Message format
43	Exposure computation

### Method of solution (up to 5 most relevant points)

18	Server architecture
20	Data model
23	Data interface
43	Exposure computation
27	Server supporting software

### Contributions (most important first)

29-41	Bluetooth interface
17-28	Server architecture and implementation
42-48	iPhone application
33-38	Message format
26, 49-51	Server optimizations
49-64	Proved functionality of above components

### My work

13,18,20,34	System block diagrams/algorithms/equations solved
49-61	Description of assessment criteria used
49-61	Description of procedure (e.g. for experiments)

### Results

49-64	Succinct presentation of results
49-52,62-64	Analysis
62-64	Significance of results

### Conclusion

68	Statement of whether the outcomes met the objectives
65-67	Suggestions for future research

### Literature: (up to 5 most important references)

42	[22] WHO Air Quality Guidelines
43	[23] Human Exposure Assessment...
43	[25] EPHC Ambient air quality measure
6	[1] WHO Air quality and health fact sheet
13	[13] Participatory sensing

## Abstract

This project, dubbed Haze Watch, aims to build a system for recording air pollution levels at a higher spatial resolution than that currently available in existing systems, and hence to also build a Smartphone application to enable individuals to monitor their personal exposure based on readings retrieved via the internet from our system. Our system includes a specially built hardware sensor module, a database server connected to the Internet and web and mobile applications for accessing the data.

This report focuses on the key components of my work in this system: the server, database and pollution exposure application.



## Acknowledgements

I would like to acknowledge the other members of the Haze Watch project; James Carrapetta and Amanda Chow for their support and hard work throughout this project. I would also like to specially thank my supervisor, Dr Vijay Sivaraman who provided me with excellent guidance, direction and support throughout this project.

## Table of Contents

Abstract .....	2
Acknowledgements .....	3
<b>1. Introduction .....</b>	<b>6</b>
1.1. Project aims .....	7
1.2. My roles .....	7
<b>2. Background.....</b>	<b>9</b>
2.1. Related work .....	11
2.1.1. MAQUMON .....	11
2.1.2. Sensaris .....	11
2.1.3. iSniff.....	11
2.1.4. Common Sense .....	12
2.1.5. MESSAGE Project.....	12
<b>3. System overview.....</b>	<b>13</b>
3.1. Mobile sensor unit .....	14
3.2. Database server.....	15
3.3. Client Applications.....	15
3.3.1. Pollution maps.....	15
3.3.2. Personal exposure monitor .....	16
<b>4. Server.....</b>	<b>17</b>
4.1. Specifications.....	17
4.2. Architecture.....	18
4.2.1. Model .....	18
4.3. Database .....	20
4.3.1. Data model .....	20
4.3.2. Units .....	21
4.3.3. Capacity .....	22
4.4. Data Interface .....	22
4.4.1. Import.....	23
4.4.2. Export .....	23
4.4.3. Real time uploads.....	24
4.4.4. Manual Data Imports .....	25
4.4.5. Automatic Imports.....	25
4.5. Batch queries.....	26
4.6. Supporting software .....	27
<b>5. Mobile pollution sensor interface .....</b>	<b>29</b>
5.1. Hardware .....	29
5.1.1. Overview .....	29
5.1.2. Communication interface .....	30
5.1.3. Bluetooth module .....	31
5.2. Platform .....	31

<b>5.3. Protocol.....</b>	<b>32</b>
5.3.1. General idea .....	33
5.3.2. Requirements .....	33
5.3.3. Message format .....	33
<b>5.4. Mobile application .....</b>	<b>38</b>
5.4.1. Screens .....	38
5.4.2. Hardware connection.....	40
<b>6. Personal exposure estimation application.....</b>	<b>42</b>
<b>6.1. Pollutants of concern .....</b>	<b>42</b>
<b>6.2. Methods of exposure computation.....</b>	<b>43</b>
6.2.1. Initial development.....	43
<b>6.3. iPhone application .....</b>	<b>44</b>
6.3.1. Overview .....	45
6.3.2. Technologies used .....	47
<b>7. Evaluation.....</b>	<b>49</b>
<b>7.1. Server.....</b>	<b>49</b>
7.1.1. Batch query optimization.....	49
7.1.2. Database.....	52
<b>7.2. Hardware interface .....</b>	<b>52</b>
<b>7.3. Personal pollution exposure application.....</b>	<b>55</b>
7.3.1. Verification tests .....	55
7.3.2. Field test .....	57
7.3.3. Discussion .....	62
<b>8. Future work.....</b>	<b>65</b>
<b>8.1. Server and website .....</b>	<b>65</b>
8.1.1. Website user interface .....	65
8.1.2. Data export format .....	65
<b>8.2. Personal pollution exposure iPhone application .....</b>	<b>65</b>
8.2.1. User interface.....	65
8.2.2. Confidence levels .....	66
8.2.3. GPS accuracy .....	66
8.2.4. Path simplification .....	66
<b>8.3. Mobile pollution sensor Android application .....</b>	<b>67</b>
8.3.1. User interface.....	67
8.3.2. Uploading extra information.....	67
8.3.3. Privacy concerns .....	67
8.3.4. Tunnels.....	67
<b>9. Conclusion .....</b>	<b>68</b>
<b>References.....</b>	<b>69</b>

# 1. Introduction

The World Health Organization (WHO) estimates that air pollution causes 2 million premature deaths worldwide each year. It also estimates that around 1.4 billion urban residents worldwide are living in areas with air pollution above recommended levels [1]. Exposure to air pollution increases the incidence of various health problems, such as respiratory infections, heart disease and lung cancer, and the mortality rate of the general population. Exposure to particulate matter, for example, contributes to the risk of developing cardiovascular and respiratory diseases & lung cancer. The case for ozone (O<sub>3</sub>) is similar, with exposure attributing to breathing problems, asthma and lung diseases. Both ozone and particulate matter pollution levels are regularly above the target levels in developing and developed countries. Specific sources of air pollution vary by pollutant, however most emissions can be attributed to the transportation and power generation industries [2]. In the United Kingdom, for example, it is estimated that road transportation accounts for approximately 61% of carbon monoxide (CO), and 46% of nitrogen oxides (NO<sub>x</sub>) emissions.

Current systems in place to monitor air pollution levels, usually maintained by local governments, typically have only a handful of fixed monitoring stations distributed over an entire city. Such sparse monitoring cannot effectively monitor variations seen around pollution 'hotspots' such as roads and industrial areas. For instance, if samples are taken on the roofs of buildings, far away from roadways or in uninhabited areas, and averaged over the rest of the city, the sources of pollution, namely cars and industry will not be represented in these readings.

Medical studies are still searching for conclusive links between exposure to air pollution and various medical conditions. A system capable of monitoring an individual's exposure to various pollutants would provide very valuable data to such studies.

With only sparse monitoring available, an individual's personal pollution exposure and hence health impacts, cannot be accurately assessed without carrying a costly physical pollution-measuring device on their person. Many people, however, do carry

a very versatile device on their person almost all the time – a Smartphone<sup>1</sup>. The Haze Watch system exploits this fact by taking advantage of the built-in GPS hardware found in many such Smartphones. By tracking the location of an individual through time and utilizing a collection of spatially dense pollution samples, we can estimate that individual's personal exposure to various pollutants relatively well without the need to carry a physical pollution-measuring device.

## 1.1. Project aims

In general, this project aims to build a system which

- Collects air pollution readings for a variety of pollutants
- Takes these readings with high spatial resolution compared to sparse fixed-site systems currently available
- Can provide air pollution data on request via the internet, both visually and as raw data
- Can utilise this data to quantitatively estimate the pollution exposure of individuals and to yield results comparable to a dedicated pollution-monitoring device.

The system should fulfill these objectives elegantly, and cost effectively.

## 1.2. My roles

This project is in its infancy, having commenced at the beginning of this year. Fulfilling all the aims of this project is a substantial task, and it is likely the project will be ongoing for some time into the future. As it pertains to thesis work this year, the aims of the project were to get a basic version of this system implemented which could be used for experimentation, and further refined in the future.

A team of 3 is undertaking this project, at the present moment: James Carrapetta [3], Amanda Chow [4], and myself. My roles in particular for this thesis are to:

- Design and implement the server infrastructure to support data collection, retrieval and to host the maps software

---

<sup>1</sup> A 'Smartphone' is just a mobile phone that offers more advanced features than a regular mobile phone. They can be thought of as a hand-held computer with an integrated phone, rather than a mobile phone with extra tacked on features - a regular phone.

- Interface with the sensor hardware to facilitate the upload of readings to the server, via a Smartphone.
- Build an iPhone application to conveniently monitor and report a user's pollution exposure

## 2. Background

The idea of real-time air pollution monitoring is not new. Most developed countries have a pollution monitoring system of some description in place, in and around major centres of population. The major problem with these systems is the sparse nature of the monitoring, which leads to readings indicative only of the background levels of pollution. A secondary problem is that the information provided by these systems is sometimes not readily accessible in a user friendly way.

Take for example the DECCW's (Department of the Environment, Climate Change and Water, NSW) system for monitoring air pollution. Pictured in Figure 1, there are only 14 static monitoring stations covering the entire city of Sydney, plus an additional 6 covering the rest of the state.

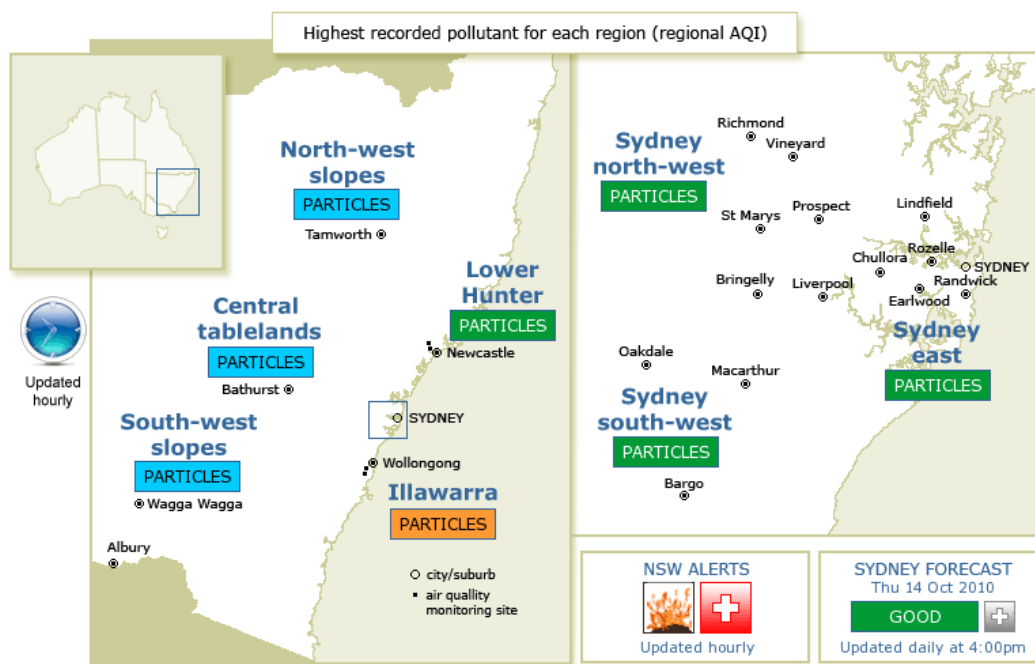


Figure 1: The DECCW Air Quality Index map. Air pollution monitoring sites are marked.

Actual pollutant level readings are accessed via a separate page and presented in tabular form with no relation, other than the site name, to the map.

Similar systems exist around the world in major cities. Perhaps one of the best of these is the GreatAir Manchester project [5]. Pictured in Figure 2, this system offers sparse monitoring of pollutants over several static sites around the city of Manchester, UK.

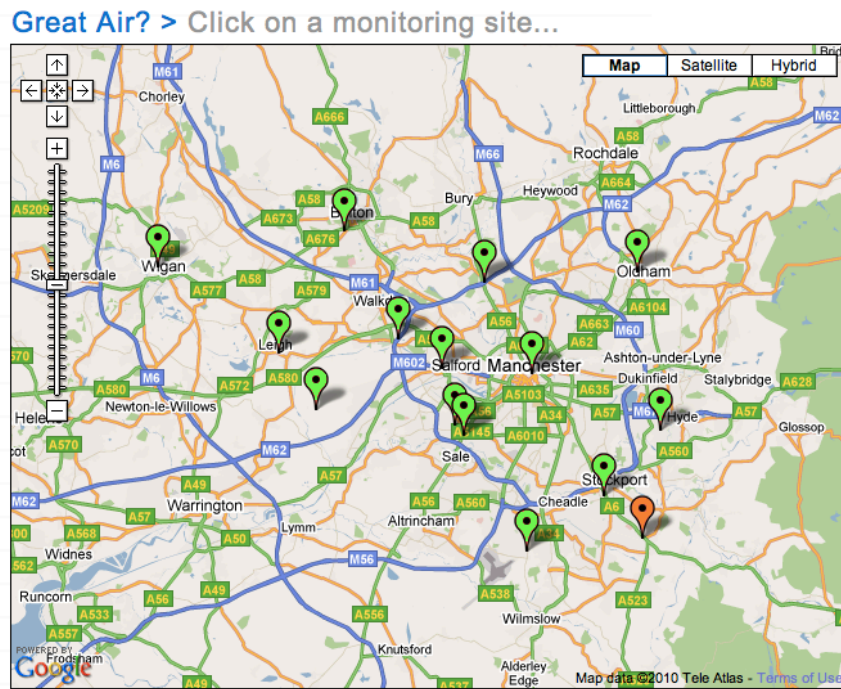


Figure 2: GreatAir Manchester (United Kingdom) map from site home page showing monitoring sites around the city. Each marker can be clicked to bring up readings for the site, historical data, and graphs.

The key difference between this and the DECCW’s website is the presentation of information and the user interface. No more information is available than the DECCW site, however the presentation is superior. Pollution readings can be accessed via a clickable map, in addition to historical readings and graphs. This interface makes much more sense than the one provided by the DECCW, and is thus more user-friendly.

Systems such as these give a good indication as to the background levels of air pollution in a given area. Research suggests, however, that pollutant levels in air show significant temporal and spatial variation. A study of the pollution levels on and near busy roads, for example, in Amsterdam over a 10 year period [6] showed that there can be an up to 2x difference in pollutants such as NO, NO<sub>2</sub>, and CO between samples taken in close proximity to the road and those further away. The systems outlined above are designed to measure ambient air pollution levels and thus cannot take into account hotspots such as heavy industry or roads due to the physical location of the sensors.

Given this and the fact that people generally spend a non-insignificant period of time on or near roadways, there is a need for higher density sampling and modelling to take these variations into account.



Research work currently underway concerned with tackling this problem is briefly outlined in the following section.

## 2.1. Related work

There are a number of other projects working on similar systems to measure air pollution. All projects found during our research are briefly outlined here.

### 2.1.1. MAQUMON

The MAQUMON (Mobile Air Quality Monitoring Network) project [7,8] has built a prototype device packing a lot of hardware into a small package. The prototype contains O<sub>3</sub>, NO<sub>2</sub>, CO, temperature and humidity sensors, a GPS chip, accelerometer, LCD, USB port, flash memory and a Bluetooth interface. The device is able to connect to the Internet via a laptop or a PDA and periodically uploads pollution measurements to a server. The project was undertaken in 2007 by the Networked Embedded Systems Lab at ISIS, Vanderbilt University, and has been funded by a grant from Microsoft Research.

### 2.1.2. Sensaris

Unlike the other projects discussed here, Sensaris is a commercial enterprise based in France [9]. They sell a number of sensor ‘pods’ (handheld devices) equipped with various sensors, including CO, CO<sub>2</sub>, NO and NO<sub>2</sub> sensors, motion sensors, and the option for a customized device with sensors suiting one’s particular needs. The sensors have wireless communication technologies built in that can upload data to Sensaris servers. The Sensaris product line appears to be targeted at organizations and companies concerned with monitoring various environmental parameters relative to them.

### 2.1.3. iSniff

The iSniff project is seeking to study the health impacts of air pollution, in particular black soot (particulate matter), on asthma in children. They have developed a pocket-sized sensor device, which children will carry on their person to record their exposure to pollution. The data will then be analysed by researchers with the hope of finding a link between exposure and the incidence of asthma. The project is being funded by the National Institute of Environmental Health Service (NIEHS) as part of the *Genes and Environment Initiative* – aimed at gaining an improved understanding of how genetic factors and environmental exposures influence human disease [10].

#### 2.1.4. Common Sense

The Common Sense project is developing mobile sensing technologies that help communities gather and analyse environmental data [11]. They have developed two sensor platforms for research, primarily focusing on measuring air pollution. One prototype has been deployed on street sweepers in San Francisco to collect air pollution data as they go about their work. They have also developed a handheld device with built in mobile communications hardware to upload data directly to a server. Both platforms record carbon monoxide, ozone, nitrogen oxide and nitrogen dioxide. They are focusing their efforts in demonstrating the utility of embedding environmental sensors in consumer devices such as mobile phones. The project's principle funding is sourced from Intel Labs Berkeley, California.

#### 2.1.5. MESSAGE Project

The MESSAGE (Mobile Environmental Sensing System Across Grid Environments) project [12] is another project with the aim of monitoring various environmental parameters. The project aims to develop a network of low cost vehicular sensors to monitor various urban parameters including traffic and environmental conditions. The project has been ongoing since 2006, and has the financial backing of several companies. Led by Imperial College London, the project is being conducted in collaboration between several universities in the United Kingdom. The system architecture is very general and allows for multiple types of data to be collected and analysed.

#### *Discussion*

The Haze Watch system shares common features with, and builds on the ideas of, most of these projects. Several of these projects are either trying to create a general sensor network for collecting and visualising many different types of data, or including a lot of functionality on each board. We are trying to focus as much as possible on air pollution sensing alone, and utilize as much existing hardware as possible present in Smartphones rather than duplicate it in our own sensor boards.

The major way in which our work differs from these research projects is that our system is focused around the collection of air pollution data in the pursuit of computing accurate pollution exposure for individuals, and to follow on from this, enabling investigations as to the impacts of such exposure.

### 3. System overview

The Haze Watch system consists of three essential components: a mobile sensor unit, the database server and the client applications. Figure 3 illustrates this concept.

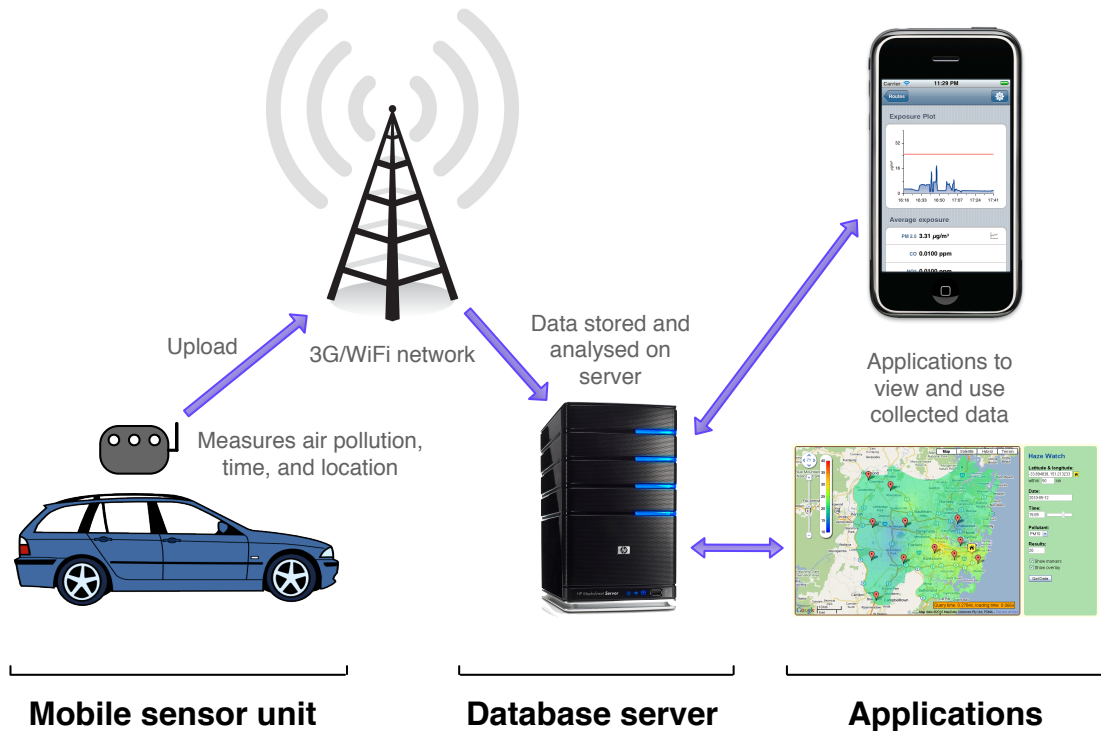


Figure 3: General overview of the Haze Watch system

This system is based on participatory sensing methodologies [13], where users of the system can contribute data to make the system as whole better. Our plan is to have a number of ‘core’ users equipped with a hardware device capable of measuring air pollution levels, and many more ‘non-core’ users who can use various client applications to visualise the data, without actually carrying a mobile sensing unit themselves.

Essentially, any vehicle that travels regularly with journeys spanning large areas of a city could constitute a valuable ‘core’ user. The ‘core’ users of the system could, for example, include buses, taxis, regular people, local government vehicles and trucks. Core users would each carry a mobile sensor unit, which would communicate with Smartphones carried by these users, through which the pollution readings are uploaded to our server over the Internet.

Regular ‘non-core’ users can benefit from the collected data through the client applications of our system. For example, one application we are developing for Smartphones enables users to assess their exposure to various pollutants without having their own physical pollution-monitoring device.

### 3.1. Mobile sensor unit

The mobile sensor unit consists of two important components: the sensor hardware, pictured in Figure 4, and an Android Smartphone application. The sensor hardware has been designed and built by James Carrapetta [3].

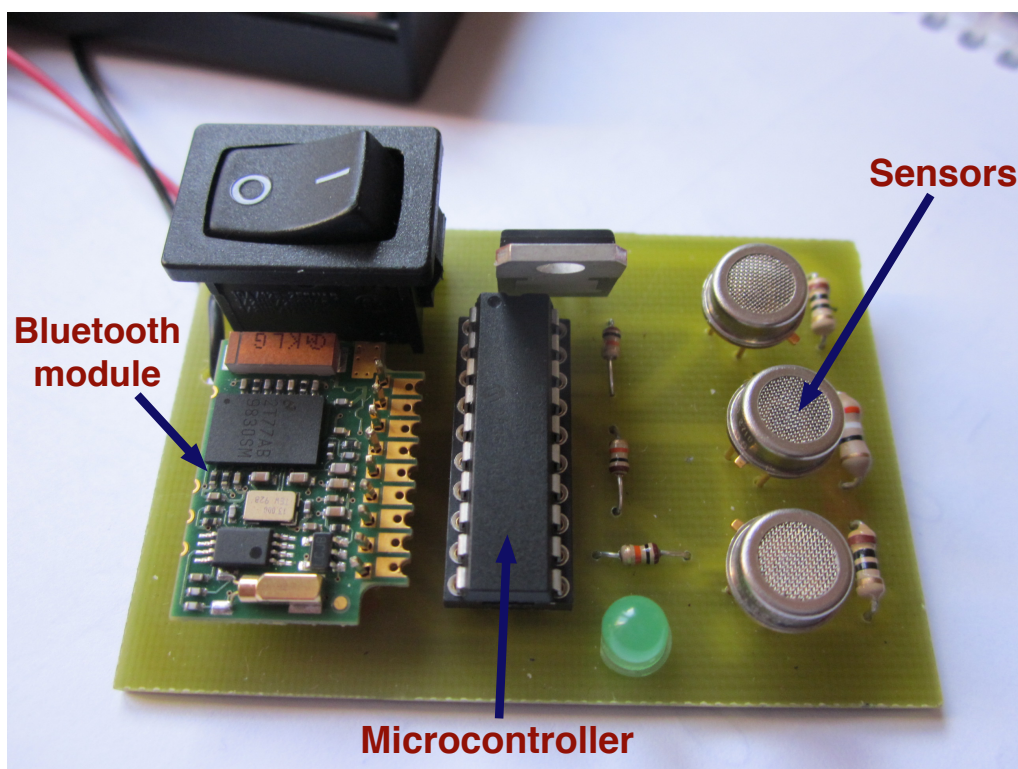


Figure 4: Mobile sensor unit PCB

It currently has 3 onboard pollution sensors for sensing:

- Carbon monoxide (CO)
- Nitrogen dioxide (NO<sub>2</sub>)
- Ozone (O<sub>3</sub>)

Data is retrieved from the device via a Bluetooth interface. A mobile application, designed for the Android platform, can connect to the mobile sensor unit via, read data and facilitate the upload of this data to our server. Where no network connection is available, the data may also be viewed in log files saved on the phone.

This device is designed to be as simple, and low-cost as possible, utilising as much existing hardware already present on Smartphones as possible.

## 3.2. Database server

The database server plays an important, yet fairly simple role in the system. Firstly, it runs a database to record the pollution samples. Secondly, it provides a data interface to client applications wishing to use air pollution data. And thirdly, it hosts the project website and selected client applications (currently just one – pollution maps).

## 3.3. Client Applications

The Haze Watch system currently includes two client applications. Each is briefly outlined here.

### 3.3.1. Pollution maps

Pictured in Figure 5, the pollution map is a web application run from the server.

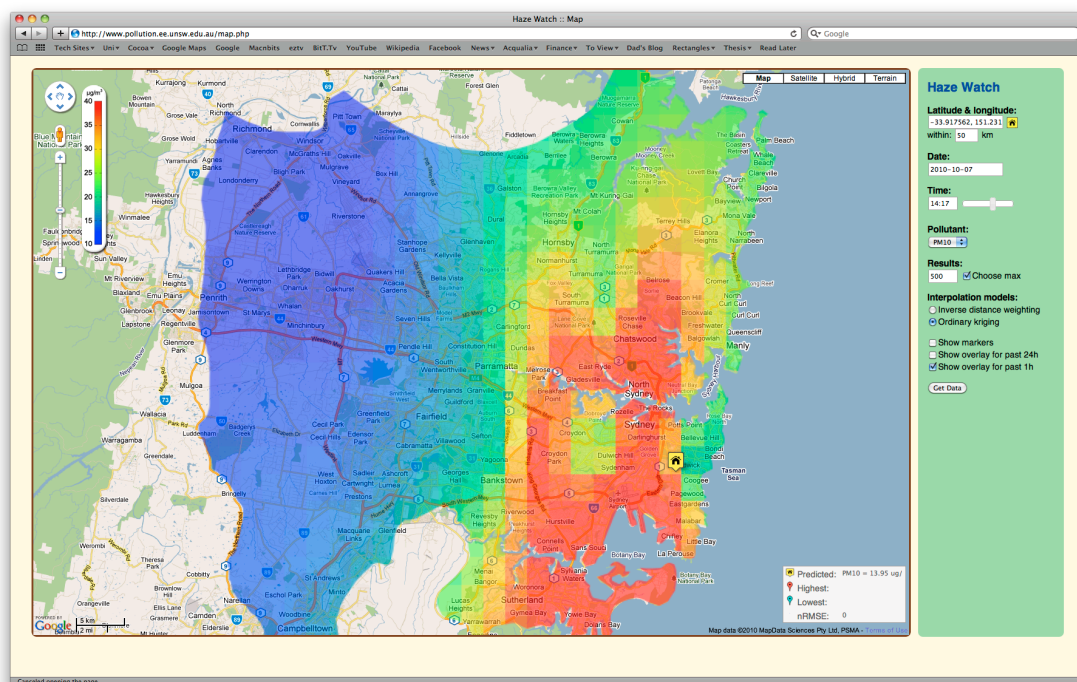


Figure 5: Screenshot of the pollution maps web application

It provides a user with a graphical picture of the pollution levels in a city, overlaid on a map. Points at which data has been collected may also be viewed on the map, including readings for each pollutant. Work on this application, including the

generation of the coloured map overlays and the underlying interpolation model, has been undertaken by Amanda Chow [4].

### 3.3.2. Personal exposure monitor

The personal pollution exposure monitor is an iPhone application, which tracks a user's geographical location through time. It then uses this data plus pollution level data from the server to graphically display and compute the users exposure to various pollutants over time. A sample screenshot of this application is included in Figure 6.

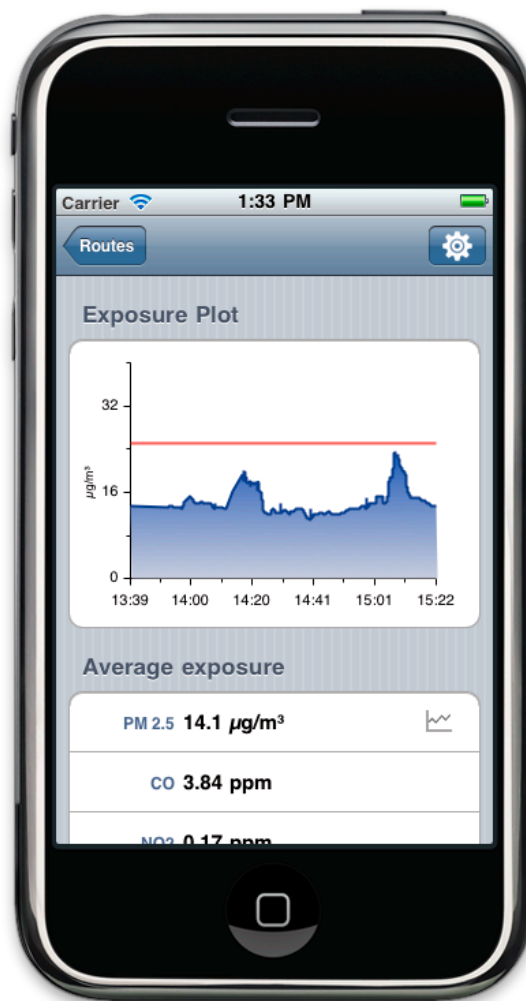


Figure 6: Screenshot of personal pollution exposure application running on an iPhone. Pictured is a graph of PM 2.5, with the guideline value (WHO) plotted in red

With sufficiently dense pollutant sampling, this application could potentially be as accurate as a physical pollution-monitoring device.

The following sections will discuss each of these components in detail.



## 4. Server

The server represents the backend and backbone of the system. It has the following general responsibilities:

- Store data on air pollution in a standard way
- Provide this air pollution data to clients
- Host project website
- Host and run software to interpolate sparse data samples
- Facilitate data import

### 4.1. Specifications

The specifications of the server are outlined in Table 1.

Computer:	Intel Core 2 Duo 2.83 GHz, 3 GB RAM, 220 GB HDD
Operating System:	Ubuntu Linux 9.10
HTTP Server:	Apache 2.2.12, PHP 5.2.10
Database:	MySQL 5.1.37

**Table 1: Server specifications**

Linux has been chosen as the operating system as it is open source (i.e. free), ideal for server applications and includes a Unix command line environment that is far more flexible than those offered by Windows, for example. ‘Ubuntu’ is the particular Linux operating system distribution used. It includes an easy to use graphical user interface, has all the server side software needed and is thoroughly documented and supported. In the event that a particular piece of software is not included by default, it can easily be installed via a built-in package manager.

For the server side software, we have opted to use standard and widely used software in the industry. For example, we are using the ‘Apache ‘ HTTP server, which has been the most popular HTTP server in use on the Internet since 1996 [14]. The other software (PHP, MySQL, Perl) was chosen for similar reasons.

## 4.2. Architecture

The general architecture of the server side software is depicted in Figure 7. It is a layered approach to implementing the requirements. Essentially there are 3 layers: the web server layer, model layer and the database layer.

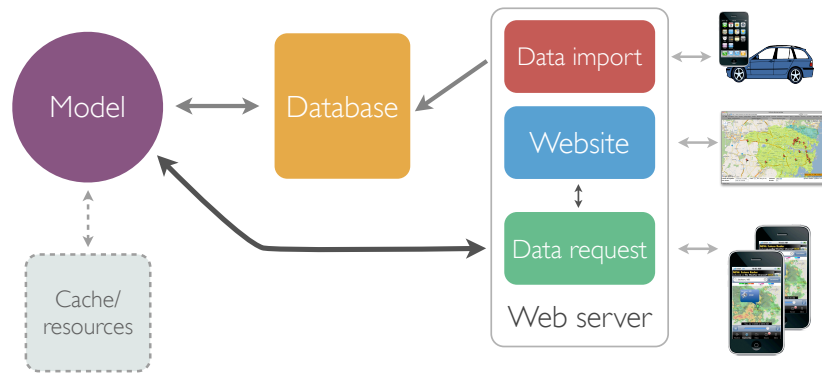


Figure 7: Architecture of server software

Note that there is no direct interaction between the outside world and the database, when requesting data. Only the data import module directly interfaces with the database. All requests are routed through the ‘model’ software which has its own connection to the database.

### 4.2.1. Model

The model software is needed to interpolate samples – i.e. return air pollution levels for any arbitrary point in location and time (data which the discrete database can not possibly entirely contain). The interface between the web server and the model is standard and fixed. This allows the model component to be easily upgraded without affecting the rest of the system.

The model software for interpolating pollution samples upon request has been designed and implemented by Amanda Chow [4], the architecture on the other hand is part of the server design. What follows is a brief outline of the structure of the model, and the interface between model and other server-side software; consult her work for a more detailed explanation of the implementation.

### Structure

On the server, the model is simply a folder of files. The only requirement for the folder to be considered a model is that it be placed in the ‘models’ directory, and that it has a



file named *'model.php'*. The *'model.php'* file serves as the interface between the model and other software. PHP was chosen as the file type for the interface as that is the language of the majority of other components of the server. The model, however, is not confined to the PHP language for implementation; it can be written in any language and simply be invoked via PHP. The model defines its own connection to the database, and coordinates with its own resources and caches.

## **Interface**

The interface (*model.php*) currently consists of the following functions:

### **model\_location\_and\_time\_array**

*(latitude, longitude, radius, datetime, numresults, pollutant, maxfit)*

This function returns an array of data points for the given location and time. *'numresults'* specifies the maximum number of points to return. If *'maxfit'* is true then as many results as possible are returned regardless of the value of *'numresults'*.

### **model\_pollutant\_grid**

*(latitude, longitude, radius, datetime, numresults, pollutant, modeltype, maxfit)*

This function returns the levels of pollution for a particular pollutant as an array representing a grid. The *'pollutant'* parameter may be any one of the following values: *'pm10'*, *'pm2.5'*, *'no2'*, *'so2'*, *'o3'*, *'co'*. The *'numresults'* and *'maxfit'* parameters serve the same purpose as in other functions.

### **model\_pollutant\_point**

*(latitude, longitude, radius, datetime, numresults, pollutant, modeltype, maxfit)*

This function returns the level of pollution for a particular pollutant, at a particular point. The *'radius'* parameter specifies the maximum distance from the given point that samples must be to be included in the calculation.

### **model\_pollutant\_points\_for\_locations\_and\_times**

*(latitudes, longitudes, datetimes, pollutant)*

This function returns the levels of pollution for a particular pollutant at multiple points. Arrays of latitudes, longitudes and times are passed in as parameters, as well as the particular pollutant desired. Use this function instead of *model\_pollutant\_point* where multiple values need to be computed for improved performance.

### **model\_pollutant\_rmse**

*(latitude, longitude, radius, datetime, numresults, pollutant, modeltype, maxfit)*

This function returns the standard deviation of all measured points within a query as well as the root mean square error (RMSE) and normalized root mean square error (NRMSE).

### 4.3. Database

The database forms the core of the system. It is where all readings are stored, and also provides a convenient interface for extracting and filtering readings. An SQL (structured query language) based database has been selected in favour of other data storage technologies, such as an xml or other flat file format, because these features are provided by default. SQL based databases offer easy searching and filtering of data, and are relatively efficient for random access over large sets of data. We chose to use MySQL as the database hosting software because of its status as the world’s most popular open source database in use [15].

#### 4.3.1. Data model

The tables in the database are modelled around the data model depicted in Figure 8.

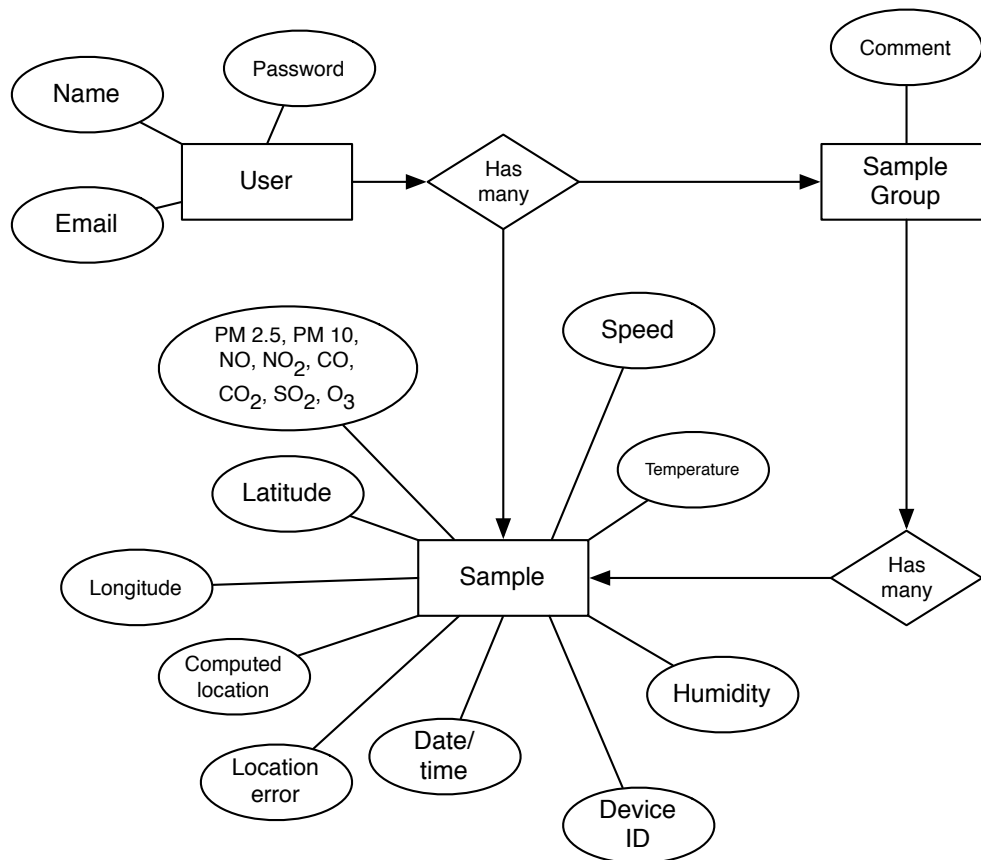


Figure 8: Database structure model

The database keeps track of several key pieces of information. These are:

- Pollutant samples
- Time and location information for each sample
- Sources of samples

The structure is quite simple, consisting of just 3 tables. The ‘Samples’ table is used to track pollution readings, or samples. Each row of data contains, amongst other fields, the date, time and location of the readings and readings for each of the pollutants we are tracking. When no value is available for a particular pollutant, NULL is inserted into the database instead of a zero value. Each sample can be tied back to which device uploaded the measurement through the device ID field, and also back to the user through the relationship.

The ‘SampleGroup’ table facilitates the grouping of several samples from the same user, and also the ability to tag those with a comment. The grouping feature is used to track bulk imports of data, and also to help separate debugging data from real data.

The ‘Users’ store information about users, and would be used more than anything for a future web interface where users could log in to track their readings online. Currently it is used only to tag uploaded samples, and to allow select individuals to login to upload data. Should a user not wish to have their uploads tracked by the server (due to privacy concerns), their readings, and those of any other anonymous users, could be tagged with the same ‘anonymous’ user identifier. Naturally this would only provide some form of anonymity when multiple users use this feature simultaneously.

#### 4.3.2. Units

The units used by the database for storing samples for each pollutant is outlined in Table 2.

Pollutant	Unit
PM 2.5, PM 10	µg/m <sup>3</sup>
Carbon monoxide (CO), Carbon dioxide (CO <sub>2</sub> )	ppm
Nitrogen monoxide (NO), Nitrogen dioxide (NO <sub>2</sub> )	ppm
Ozone (O <sub>3</sub> )	ppm
Sulphur dioxide (SO <sub>2</sub> )	ppm

Table 2: Units used for stored samples in the database

We chose to use parts-per-million notation (ppm) for most pollutants, primarily because they are the units used by several authorities on the matter (DECCW, NEPC/EPHC). In practice, all of these pollutants except particulate matter can be represented as either ppm or mass-per-volume quantities. The reason particulate matter (PM 2.5 and PM 10) may not be represented in ppm form is due to its composition. Particulate matter is composed of suspended particles in air including dust, soot, smoke and mist with an aerodynamic diameter of less than 10 microns (PM 10) or 2.5 microns (PM 2.5). The particles, therefore, vary dramatically in composition depending upon their origin. For this reason a molecular weight for particulate matter cannot be accurately determined, and hence particulate matter cannot be represented in ppm notation, only in mass-per-volume.

The typical and acceptable ranges of these pollutants is discussed in section 6.

#### **4.3.3. Capacity**

The capacity of a MySQL database is limited only by constraints imposed by the operating system [16]. For example, the FAT32 file system used by older versions of windows has a maximum file size limit of 4GB, limiting the size of a database to 4GB. Fortunately, the Linux operating system has few such constraints and practically speaking the capacity of a database is determined solely by the capacity of the hard disk, in our case 220 GB.

Considering only of the 'Samples' table (representing about 95% of our storage requirements), the server in it's current form has enough hard disk space to store 1.8 billion rows of data – more than sufficient for our needs at this stage.

## **4.4. Data Interface**

The server is required to import and export data. The method by which services interact with a server is known as an interface. The ultimate aim when defining an interface is to make it flexible and simple, such that all clients can use it to implement their features. This has been the guiding philosophy behind the data interface of the server.

#### 4.4.1. Import

The data import interface allows for the uploading of data files. Data is imported via an HTTP post to the server<sup>2</sup>. The data is encoded as XML – a sample of which is included in Figure 9. This format allows for the inclusion of one or multiple samples, and an arbitrary number of pollutants. XML is very flexible, human readable, and easily allows for new fields of data to be added later, such as new pollutants, without affecting the ability of the server to parse and extract the data.

This data interface at present provides no means of specifying the unit of each pollutant value. Instead the units presented earlier, in Table 2, are assumed.

```
<?xml version="1.0" encoding="UTF-8" ?>
<data>
  <info></info>
  <samples>
    <sample>
      <datetime>2010-05-24 18:00:00</datetime>
      <latitude>-33.797222222222</latitude>
      <longitude>150.765833333333</longitude>
      <no2>2.0</no2>
      <o3>0.2</o3>
      <pm10>17.6</pm10>
    </sample>
    <sample>
      <datetime>2010-05-24 18:00:00</datetime>
      <latitude>-33.917777777778</latitude>
      <longitude>151.134722222222</longitude>
      <no2>2.2</no2>
      <o3>0.1</o3>
      <pm10>22.2</pm10>
    </sample>
  </samples>
</data>
```

Figure 9: Sample XML data format for import

Using XML also has an advantage over other formats in that it is very logical in layout and readily human readable. It is, however, very verbose, incurring a lot of overhead in the tags. Despite its shortcoming, XML is a popular format supported by a wide variety of platforms and programming languages, making it both easy to generate and parse.

#### 4.4.2. Export

The original intention was to have the export interface in the same format as for the import interface, i.e. as XML. Practically, however, for the short term this seemed to

---

<sup>2</sup> The URL is currently: <http://www.pollution.ee.unsw.edu.au/post-data.php>

be a bit over the top given that the only application that uses the data export interface is the iPhone Personal Pollution Exposure application. Whilst XML is a very nice human readable format, its verbosity made it unsuitable for this particular application. The personal exposure application typically requests several hundred to a few thousand data points each request – data, which encoded in XML, would take a noticeable amount of time to download over a mobile network. For this reason, a simple, compact data format was chosen - CSV (Comma Separated Values), which compacts the same data by about 85%.

In a similar fashion to the data import interface, exporting is also achieved via an HTTP POST or GET request<sup>3</sup>. A request includes a list of latitudes, longitudes, times and the pollutants of interest. A sample request is shown in Figure 10.

```
get-data.php?latitudes=-33.825302,-  
33.823993&longitudes=151.117850,151.111237&datetimes=2010-03-24  
08:53:00, 2010-03-24 08:53:30&pollutants=pm10,no2
```

Figure 10: A sample request for pollution levels (PM 10 and NO<sub>2</sub>) at 2 locations.

A response from the server is simply a set of lists of values corresponding in index to the locations and times provided in the request. Each list of values is prefixed by the pollutant name. For example, a possible response for the request in Figure 10 is shown in Figure 11.

```
pm10=22.5,23.4  
no2=0.023,0.019
```

Figure 11: Sample response for the previous request

The resultant response is very easy for a program to interpret, and is very compact, containing only the bare minimum information, which was the main goal in the development of this interface.

#### 4.4.3. Real time uploads

The mobile pollution sensor uses the data import interface as described above. Periodically, the mobile pollution sensor application will send a batch of pollution readings time-stamped and tagged with location coordinates. Further details about the implementation and mechanism of the data uploads can be found in Section 5.4.

---

<sup>3</sup> The URL is currently: <http://www.pollution.ee.unsw.edu.au/get-data.php>

Essentially, the application uses the XML libraries built into the Android platform to bundle up the readings and uploads these to the server through an HTTP post.

#### 4.4.4. Manual Data Imports

Whilst the sensor hardware was still in development, an interface for importing data collected manually with a commercial particulate matter sensor was also developed. It is nothing more than a webpage, which allows for data files to be uploaded and passed onto the standard importing interface discussed above. Since the commercial sensor does not have a built in GPS hardware, a GPS TrackStick<sup>4</sup> was used to simultaneously record location information. Data collected manually in this fashion is logged into two separate data files: one with particulate matter readings & time, the other with latitude, longitude & time. The manual data import page requires both files to be uploaded.

#### *File merging*

In order to import the data, a location and time must be assigned to each pollution reading. Since each parameter is being measured on separate, independent devices, in most cases the time-stamps of the respective readings will not match up exactly. Hence, the server combines the data from both files by interpolating temporally closest GPS readings to the pollution sample. The '*gps-data-join.pl*' script on the server performs this function. Once the data has been merged, an XML file (as in Figure 9) is generated for import through the standard interface.

#### 4.4.5. Automatic Imports

The server is also configured to download data from the Department of the Environment, Climate Change and Water, NSW (DECCW) every hour. This data complements any data uploaded by the mobile pollution sensor, providing a good baseline for our system. It also serves as a bit of a sanity check for our manually collected readings.

The DECCW data was chosen simply because it covers the area in which we are situated (Sydney, NSW). In theory we could extend this component to import data from any such source, or multiple sources.

---

<sup>4</sup> A small device about the size of a USB thumbdrive that logs its location every couple of seconds <<http://www.trackstick.com>>.

Like the other data import tasks, the automatic imports also make use of the same interface. In this way, any incoming data to our system is funnelled through the same code, aiding in maintenance and reliability.

## 4.5. Batch queries

An issue identified whilst writing the personal pollution exposure application was that query times on the server were incredibly slow for batch queries. A typical request, for example, of about 400 pollution levels spread over approximately 900 square km, over a day took on average well over 2 minutes. This is far too long – most users are likely to give up waiting after a time period of the order of about 5 seconds.

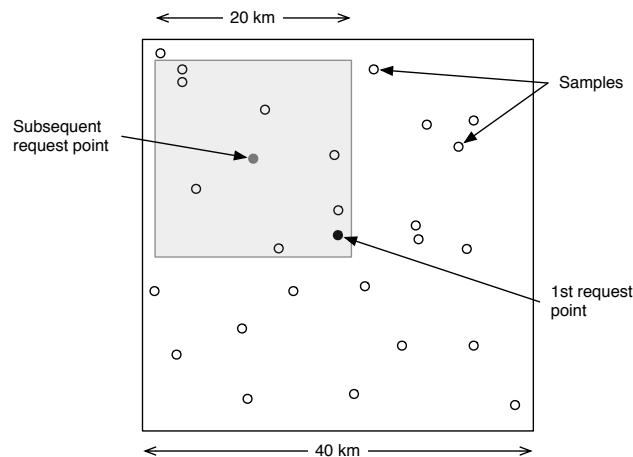
The essential problem with the query implementation was that to calculate the pollution level at each of the 400 points, the model would hit the database requesting a number of surrounding points (10-100) to then finally calculate the resultant pollutant level. Database queries are generally quite slow (on the order of several milliseconds per query) – it is easy to see how this quickly adds up for hundreds or thousands of queries.

By recognizing that the majority of requests are spatially and temporally dense, the points surrounding the requested point (used for calculation) can be cached and re-used for subsequent calculations. Exploiting this fact, I wrote a new function specifically for batch requests which works roughly as follows:

1. If there are no cached sample points, retrieve from the database up to 200 points in a 40km box surrounding the requested point.
2. Use the points to calculate the requested value.
3. If the following request is for a point within the 40km box (spatial window constraint) and has a timestamp within 15 minutes of the original request point (temporal window constraint), then use the cached points to calculate a new pollution level.

This technique is illustrated in Figure 12.





**Figure 12: Windowing technique for caching sample points used in the computation of multiple pollution levels**

The improvement in performance using this technique over the original is several orders of magnitude. Comparing the time taken to calculate the levels of pollution at a varying number of points using the new method versus the previous method has shown a 10-100x improvement – dependant on the number of points. A complete discussion of these results is presented in Section 7.1.1.

## 4.6. Supporting software

In addition to the database and HTTP server software, an integral part of the server is the collection of backend scripts performing handy little tasks. The key idea behind the architecture of this component of the software is to divide the work up into small manageable chunks, and assigning each of these tasks to a script. The scripts can then be run separately, or as a chain to produce output for as much, or as little of the complete task as desired. All scripts described here reside in the ‘*scripts*’ folder on the server, most are written in Perl.

### *DECC hourly data*

A script, named *DECC-hourly-data.pl*, written in Perl, runs every hour as a ‘cron’ job. This script is responsible for downloading the appropriate webpage from the DECCW website, parsing the data and outputting it in CSV format. If the format of the data on the webpage were to change, this script would no longer function as desired, however because it is such a small component of the system, performing a specific, simple task it would be no trouble to re-implement it.

### ***XML generator***

Since there is only a single xml-based data import interface, any data that is to be imported must first be converted into XML format. A script named *'gen-xml.pl'* takes as input CSV formatted values and converts that representation into the appropriate xml version. Figure 9 depicts a sample of this output.

### ***GPS data join***

This script takes, as input, a file with time stamped GPS coordinates and a file with time stamped pollution readings. It then merges these 2 sets of data on the time column, outputting the final data.

### ***Post***

Each script mentioned thus far performs a specific task in a chain of tasks culminating in the upload of pollution data. Each task is run in a Unix shell, allowing output to be piped from one program to the next. The final step in this chain is the HTTP post. A script called *'post-local.pl'* takes an XML file as input, and does an HTTP post on localhost (the server) to finally upload the data.

### ***Database backup***

A script named *'database-backup.pl'* runs daily and dumps the database to a file. Should something go wrong, the data can be easily re-imported into the database.

## 5. Mobile pollution sensor interface

The mobile pollution sensor hardware, which has been designed and built by James Carrapetta [3], has no built in mobile telephony hardware. Instead, we have opted to connect the mobile pollution sensor to the Internet via a Smartphone.

### 5.1. Hardware

#### 5.1.1. Overview

The mobile pollution sensor, pictured in Figure 13, features three sensors for measuring carbon monoxide, ozone and nitrogen dioxide.



Figure 13: Mobile pollution sensor hardware

The design is centred on a Microchip PIC16F690 microcontroller. Table 3 lists the key specifications of the device.

Specification	Value
Power supply	4x AA batteries (6V)
Battery life	20-22 hours continuous operation
Dimensions	65 x 45 mm (PCB), 120 x 70 mm (external case, including batteries)
Cost	\$168.65 (prototype)

Table 3: Technical specifications of the mobile sensing unit [3]

### 5.1.2. Communication interface

The technology we have chosen for communication between the mobile sensor unit and phone is Bluetooth. This section outlines the rationale for this decision.

#### *Requirements*

The connection should:

- Be reliable
- Be power efficient
- Be easy to set up
- Have a range of up to 5m

Additionally, data will need to be read from the sensors in an interval on the order of several seconds, and the communication method should require no hardware modifications to the user's phone (i.e. it must be built in).

#### *Technologies considered*

We have considered USB, Bluetooth, ZigBee (IEEE 802.15.4) and WiFi (IEEE 802.11) as potential candidates. WiFi and ZigBee were quickly eliminated:

- Although the ZigBee standard appears perfect for our application, most Smartphones are not equipped with compatible hardware, which immediately discounts it as a solution.
- WiFi is a nice communication medium to use, as it works on a much higher level than the other protocols. It would, however, require the presence of a wireless network – most phones are unable to act as a wireless base station, only as a client. Although there are likely to be plenty of wireless networks present on the average commute through an urban area, an increasing number of these are not open, and in a moving environment network conditions would change rapidly.

USB was the only wired technology considered, as most Smartphones only connect via USB. Being a wired technology, USB would allow our sensor module to run without a separate power supply – instead sourcing its power from the USB port. This is a tremendous advantage. Unfortunately, most Smartphones are configured as a USB slave, and cannot act as a host – i.e. they can be plugged into other devices, but you can't plug other devices into them.

Bluetooth meets all our requirements in that it is wireless, built in to most phones, provides reliable transfer of data and has a range of up to 10m (Class 2). The only potential disadvantage of using Bluetooth over a wired technology such as USB for instance is the power efficiency. All wireless circuits require significant power to transmit data (several milliwatts). This limitation has been overcome through careful power management on the sensor board.

### 5.1.3. Bluetooth module

The particular Bluetooth module that we have used in our design is the ARF32<sup>5</sup>, pictured in Figure 14. It supports the Bluetooth 2.0 protocol stack, and readily interfaces with standard microcontroller I/O ports via a standard serial interface (UART – Universal Asynchronous Receiver/Transmitter).

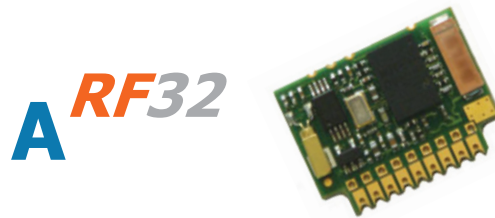


Figure 14: The Bluetooth module used in our sensor device

Of most importance to our design, it supports the RFCOMM (Radio Frequency Communication) protocol – providing a reliable serial data stream, similar to TCP.

## 5.2. Platform

As discussed earlier, a key part of our system is an Internet connected Smartphone. As an added advantage, as well as network connected hardware, Smartphones often have a built in GPS sensor, which eliminates the need to build one into our own hardware.

### Requirements

The principle requirements for a Smartphone platform for use in our system are:

- Popularity. The more users, the better (in a participatory sensing sense).
- Good APIs and developer support
- Bluetooth support

---

<sup>5</sup> Distributed by Adeunis RF <[www.adeunis-rf.com](http://www.adeunis-rf.com)>. Part number: ARF7044A

- Location awareness
- Internet connected

### *Platforms considered*

Looking at the 2010 and 2009 worldwide market share of the Smartphone industry [17,18] revealed that the top 5 mobile platforms in 2010 are (in order): Symbian, RIM, Android, Apple iOS and Windows Mobile. According to the statistics, Symbian and RIM (Blackberry) are clearly the most popular platforms. They are, however, also both losing market share at a rapid rate. Apple iOS and Google Android on the other hand are surging ahead, particularly in Australia [19]. This view is consistent with my own observations on the types of phones carried by people around me.

### *Platform chosen*

We have chosen Android as the platform for our mobile sensing unit as iOS does not expose any low level Bluetooth APIs (a requirement), whereas Android does. In most other areas, such as UI, location tracking and Internet connectivity, the iOS APIs are generally easier to use, but the lack of Bluetooth support means it cannot be used<sup>6</sup>. The major drawback to using Android as opposed to iOS is that the same application for measuring personal exposure, which I have been developing on the iPhone, could not also be used to connect to our hardware. Given more time, the reverse could certainly be possible – including personal pollution exposure estimation on the Android application.

## **5.3. Protocol**

Communication between the sensor device and the mobile application is achieved over a Bluetooth RFCOMM connection. The RFCOMM protocol establishes a reliable data stream between the device and the phone. The sensor hardware acts as a slave and accepts incoming connections from our mobile application. Data is then streamed to the phone periodically in messages, as long as the connection remains open. The message format is explained in 5.3.3.

---

<sup>6</sup> iOS does in fact have a Bluetooth API, however it is not publicly exposed. In order to use the API, Apple must specifically approve a developer. We tried several times to gain this approval, but our requests were consistently ignored.

### 5.3.1. General idea

The math capabilities of the microcontroller we are using are limited. It is a Microchip PIC16F690 – a RISC (reduced instruction set) 8-bit microcontroller, and as such can only handle 8-bit integer math in hardware. This particular microcontroller, like most others at the same level, has no floating-point math hardware. Higher precision and/or floating-point math can be simulated in software, but adds unnecessary complexity. In light of this we opted to do as much computation as possible, in software, on the Smartphone, where math libraries and floating-point operations are built in.

Consequently, the sensor hardware transmits raw 8 bit values (0-255), straight from the analogue-to-digital converter. The Smartphone, therefore, must convert these raw values into sensible pollution readings in software. Unfortunately, the low-cost sensors we are using each have a slightly different performance characteristic, meaning that the conversion equation will vary slightly in each sensor unit. There is a need, then, for the sensor unit to communicate not only the raw sensor values, but also information about the characteristics of the onboard sensors.

### 5.3.2. Requirements

The following data is required to be communicated from the sensor device to the mobile application:

- Sensor values
- Reference voltage
- Device ID
- Calibration parameters specific to the sensors on the board

For simplicity we opted to include the above information in all messages, even though the calibration parameters and reference voltage are likely to remain constant for extended periods of time.

### 5.3.3. Message format

The general format of each message is as indicated in Figure 15. It consists of a number of fixed-length fields, and also variable length fields for calibration coefficients and sensor readings. These variable length fields allow for sensor devices to have a variable number of sensors, and to transmit multiple readings in bulk. Consequently the length of any message length is a variable quantity. The specific

formats for the coefficient and values section is illustrated in Figure 16 and Figure 18, and explained thereafter.

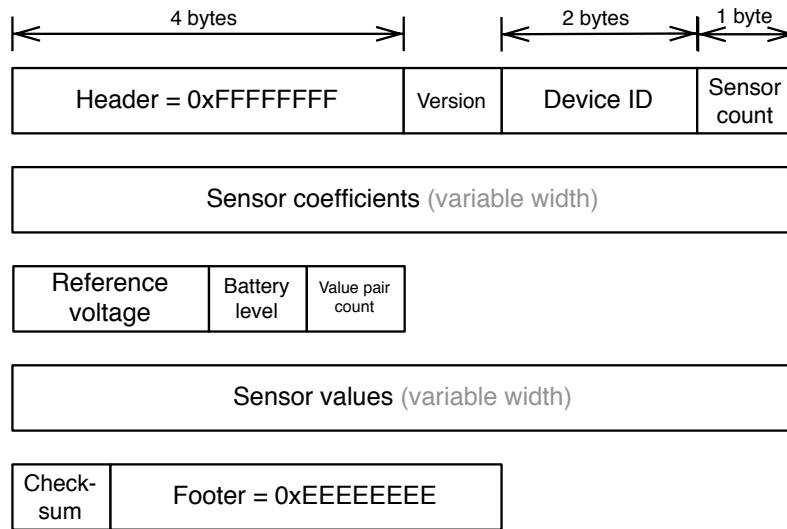


Figure 15: General format of message

Where any single quantity spans multiple bytes, little-endian byte ordering is used.

### Header and footer

The Bluetooth RFCOMM connection is essentially just a stream of data. Once a connection is established, reading of data begins from an arbitrary location almost certainly not aligned with the beginning of a message. For this reason a message header is included to clearly mark the beginning of a message. I chose the 4-byte sequence 0xFFFFFFFF as it is very unlikely that particular sequence would appear anywhere in any valid message. Although not strictly necessary, a message footer has also been included, of the same length as the header, to clearly mark the end of a message. This proved useful during debugging.

### Version

As time goes on and development of the protocol advances, the message format is likely to change. To help future proof our system on the software side, a 1-byte protocol version field follows the header. Software reading the message can then know in advance the version of the protocol used for encoding the message and parse it accordingly. During the development of the hardware interface application the message format changed several times. Before adding the version field it was very difficult to augment the message format as changes had to be implemented in hardware and software in sync for the system to function. Tagging each message with



the protocol version, allows the software to choose the decoding method, which leads to backwards compatibility with previous versions and means that the hardware logic does not necessarily need to be in sync with the software all the time. Since implementing the version field we have gone through 2 different versions of the protocol, maintaining complete backwards compatibility down to version 1. Version 2 of the message format (pictured) includes a battery level field, which was not present in version 1.

### Device identifier

In order to differentiate samples by device, we have included a device identifier field. At the present time this is a 2-byte unsigned integer, allowing for up to 65,536 unique devices. This field follows the protocol version field.

### Calibration coefficients

As previously mentioned, each sensor has slightly different electrical characteristics. Therefore to get an accurate reading from a sensor, its value must be calibrated against a known reference manually (during manufacture). James [3] worked out that a quadratic calibration function of the form  $c_0v^2 + c_1v + c_2$  works well (where  $v$  is the voltage level of the sensor). The 3 coefficients of this equation uniquely characterise a particular sensor. It is these values that need to be communicated from the sensor hardware to the Smartphone. Figure 16 illustrates the format of this section of the message.

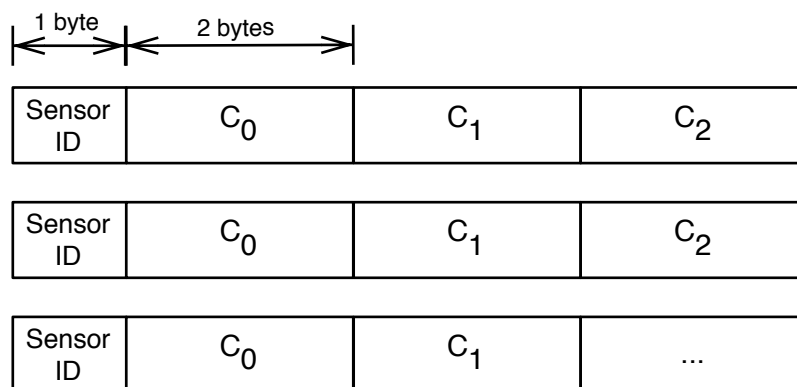


Figure 16: Coefficients format. Each sensor has 3 calibration coefficients

We opted to send the coefficients in every message, rather than having a separate message type just for sending out calibration information mainly to keep things simple. Given the time constraints we also did not want to complicate things further if

a simpler solution worked just as well. All things considered, the extra 21 bytes per message overhead do not make much of a difference overall given the high bandwidth connection offered by Bluetooth (up to about 90 KB/s). Our expected utilization of the available bandwidth is less than 1% during actual transmission. Each message, then, also contain a complete set of coefficients for each sensor. For example, if there are 3 sensors, 9 coefficients will be transmitted in each message. Each coefficient is a 2 byte signed floating-point real number, the exact format of which is discussed below. The coefficient groups are identified by a 1-byte sensor ID, as shown in Figure 16, which are just integer numbers we've assigned to each type of sensor. Table 4 outlines this mapping.

Identifier	Sensor Type
1	Ozone
2	Nitrogen Dioxide
3	Carbon Monoxide

Table 4: Mapping of sensor identifiers to sensor types

### Floating-point format

The calibration work undertaken by James [3] calculated that the calibration coefficients could range from about 0.1 up to about 10000 in magnitude, and need precision of at least 2 significant digits. Numbers could also be positive or negative. We considered using fixed point and floating point representations. A fixed point representation to just meet our requirements would require a 15 bit integer component and about 8 bits for the fractional part – a total of 23 bits. A floating-point representation on the other hand could represent the same numbers, to our specifications, with just 16 bits. We thus chose to use the 16 bit floating-point format shown in Figure 17.

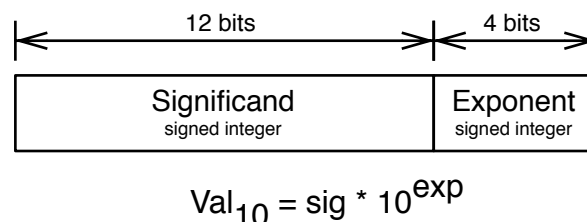


Figure 17: 16 bit floating point format

This format is conceptually similar to the familiar IEEE 754 standard [20] used on computers today, in the significand/exponent division of bits. The lengths of the significand and exponent bit fields vary, however, and also the representation of numbers as signed integers instead of in sign-magnitude or biased form as in IEEE 754. The rationale for using a representation differing from the “standard” representation is that it is simpler to represent both components in the same form (signed integer) and the particular bit lengths align very nicely with nibble boundaries and exactly meet our requirements. With this format we can effectively represent values from as low as  $10^{-8}$ , up to  $10^{10}$  with a precision of 3 significant digits, some numbers with 4 significant digits (those with a significand of magnitude 1000 to 2048). This is sufficient for our needs.

As an example, consider the number 3.358; this can be represented as  $336 \times 10^{-2}$  (to 3 significant figures). Recognizing that ‘336’ is 0x150 in hex, and ‘-2’ is 0xE, the number in our custom 16-bit floating-point representation is 0x150E.

### Reference voltage

All sensor readings are transmitted as raw ADC values relative to the reference voltage. Since we do the calculations on the Smartphone, we require a voltage level to refer these values to. This field is 2 bytes wide and has the same 16-bit floating-point format described above.

### Battery level

This 1-byte value is the approximate power level of the sensor’s battery, expressed as a percentage with a range 0-100. This field was added in version 2 of the protocol, and is not currently implemented by the hardware. The vision is to be able to use this field to notify the Smartphone software when the sensor board’s battery needs replacing/recharging.

### Sensor values

The format of the sensor values section is depicted in Figure 18.

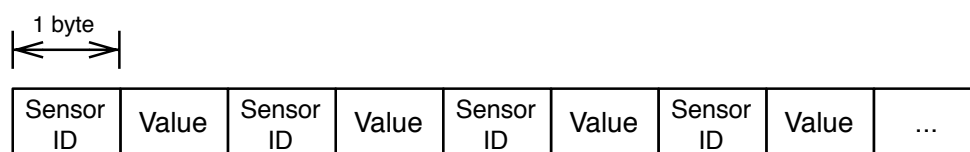


Figure 18: Sensor values format

The sensor values section of the message contains first a 1-byte count field (in Figure 15), and then up to 256 ID-value pairs. The sensor-IDs are used to determine the type of pollutant measured by the sensor. These fixed constants, discussed previously, are outlined in Table 4.

Each value, being an integer quantity in the range [0, 255] represents a percentage of the reference voltage when divided by 255. For example, a value of 130 represents 50.8% of the reference voltage - 1.68 volts if the reference is 3.3 volts. This is where the reference voltage field, outlined earlier, is useful.

### *Checksum*

Although Bluetooth provides reliable data transfer through the RFCOMM protocol, we have nevertheless included a checksum field. Although not technically needed if everything is set up perfectly, during testing we found good reason to included such a field. We found that for some reason the data stream sent through the Bluetooth module and received on the phone would intermittently become slightly scrambled. The scrambling was not so severe that the entire message structure was compromised; rather it was just a few bytes here and there with exchanged positions, which yielded incorrect received pollutant values. The reason for this turned out to be a wiring error in the hardware, and since we've wired it up correctly we've had no such issues. We decided, however, to keep the checksum in case any similar issues arose in the future.

Its implementation is quite simple. All bytes of the message are added, treated as unsigned integers. The resultant 1-byte value is the checksum.

## **5.4. Mobile application**

The application, written for Android based Smartphones is very simple, consisting solely of 2 basic screens to connect to, and monitor the sensor hardware. Its essential function is to package sensor readings into xml format and transmits those to the pollution database server. It is written entirely in Java using the Android SDK.

### **5.4.1. Screens**

#### *Device selection*

The first screen of the application is depicted in Figure 19. This screen allows the user to select a pollution sensor device to connect to.

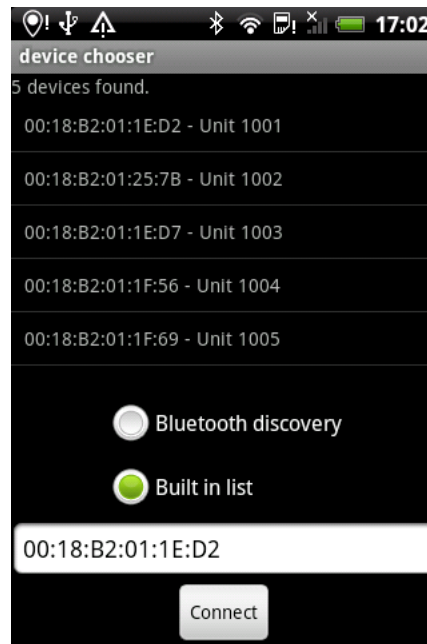


Figure 19: Device selection screen of hardware interface Android application

The application has a built-in list of devices, which we have been using for testing, as well as the ability to scan for any Bluetooth enabled hardware nearby.

We found during testing that the HTC Desire phone we had acquired had a Bluetooth bug that prevented our device (and a number of others going by user reports online) from appearing in the discovery list. For this reason we also included a text field where the user may manually type in a MAC address to connect to, should device discovery fail. Once connected, the application shows the main screen of the application.

It should be noted that the Bluetooth hardware on Android phones allows multiple simultaneous connections, such that using our application will not impact on the user's ability to use other Bluetooth hardware at the same time, such as Bluetooth headsets.

### ***Main display***

The main screen of the application, shown in Figure 20, displays the status of the connection to the sensor device as well as the latest pollution readings. In this state, a Bluetooth connection is established periodically to the sensor hardware. The stream of data from the microcontroller is read and decoded, and if valid, displayed on the screen. All readings collected in a particular session are stored in memory, and later dumped to a file, in CSV format, for inspection or analysis.

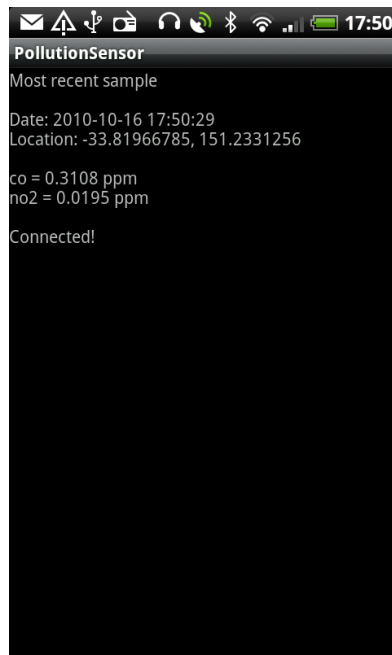


Figure 20: Main screen of the application whilst connected to the sensor hardware. Latest readings of carbon monoxide and nitrogen dioxide are shown, as well as the location and time of the sample.

As each sample is collected, it is also packaged as XML and uploaded to the server using the methods described in Section 4.4. For our purposes testing and debugging the hardware, each upload is also tagged as belonging to the '*debug*' user.

#### 5.4.2. Hardware connection

Because of the overhead introduced by each message, samples from the pollution sensors are cached in hardware before being sent to the Smartphone. In the current implementation, the sensor hardware will collect 10 samples before activating the Bluetooth module and transmitting a message. These 10 values are then averaged to compute the value for upload to the server.

To save battery power on both the phone and the sensor hardware, the application does not remain connected to the hardware all the time. Rather, it connects periodically to receive the latest message, before disconnecting, sleeping for a short while, and then reconnecting. In testing the hardware and the Smartphone application we have found that it will receive approximately 1.6 samples every minute. Practically, this means that the spacing between samples would be about 1km if the sensor were travelling at 100km/h, and 520m at 50km/h. These are reasonable values, particularly considering that the average speed of vehicles during peak periods in a major city such as Sydney ranges from about 20-65 km/h [21].

### *Pollutants monitored*

Although the hardware has sensors for carbon monoxide, nitrogen dioxide and ozone built in, currently we have not been able to calibrate the ozone sensor due to a lack of equipment. For this reason, only CO and NO<sub>2</sub> are currently being measured and recorded by the Android application. When we have calibration data for our ozone sensors, the application will have the capability to record these samples without the need to modify the protocol.

## 6. Personal exposure estimation application

As discussed earlier, this component of the system is a client to information from the server. The aim of this component is to be able to provide an estimation similar to that which could be provided by a physical pollution-measuring device.

### 6.1. Pollutants of concern

Several organizations publish guidelines on acceptable exposure levels. The World Health Organization (WHO) is one such body. Table 5 lists the pollutants of concern, and WHO guideline concentration levels. These guidelines, expressed in terms of concentration, have also been converted to parts per million (ppm) where applicable<sup>7</sup>. Particulate matter is not measured in ppm, and hence is expressed as  $\mu\text{g}/\text{m}^3$ .

Pollutant	Guideline concentration	Converted (ppm)
PM 2.5	25 $\mu\text{g}/\text{m}^3$ 24-hour mean	-
PM 10	50 $\mu\text{g}/\text{m}^3$ 24-hour mean	-
O <sub>3</sub>	100 $\mu\text{g}/\text{m}^3$ 8-hour mean	0.047 ppm
NO <sub>2</sub>	40 $\mu\text{g}/\text{m}^3$ annual mean	0.020 ppm
SO <sub>2</sub>	20 $\mu\text{g}/\text{m}^3$ 24-hour mean	0.007 ppm

Table 5: WHO guideline air pollution concentrations for the pollutants of concern [22]

For comparison, the guideline concentrations from The Environment Protection and Heritage Council Australia (EPHC) are shown in Table 6.

Pollutant	Guideline concentration	Converted ( $\mu\text{g}/\text{m}^3$ )
PM 2.5	25 $\mu\text{g}/\text{m}^3$ 24-hour mean	-
PM 10	50 $\mu\text{g}/\text{m}^3$ 24-hour mean	-
O <sub>3</sub>	0.08 ppm 4-hour mean	169 $\mu\text{g}/\text{m}^3$
NO <sub>2</sub>	0.030 ppm annual mean	61 $\mu\text{g}/\text{m}^3$
SO <sub>2</sub>	0.08 ppm 24-hour mean	226 $\mu\text{g}/\text{m}^3$
CO	9.0 ppm 8-hour mean	11,100 $\mu\text{g}/\text{m}^3$

Table 6: EPHC guideline concentrations of certain pollutants

---

<sup>7</sup> I used this conversion tool: <http://www.lenntech.com/calculators/ppm/converter-parts-per-million.htm>



## 6.2. Methods of exposure computation

Two alternative methods for computing exposure to air pollutants have been considered [23] – that of ‘dose’ and average concentration. It is important to understand the difference between these two measures, as the terminology can be confusing. *Dose* refers to the actual mass of pollutant absorbed by the body and is measured as such in  $\mu\text{g}$  (micrograms). The formula for dose is presented in Equation 1.

$$Dose = W_{person} \cdot \sum_{i=0}^N C_i B_i T_i \quad (\mu\text{g})$$

Equation 1: Dose of a pollutant (amount absorbed by the body)

*Exposure*, on the other hand, is a product of concentration and time. Its general formula is presented in Equation 2. Dividing this measure by time yields the average concentration exposed to over that period.

$$E = \sum_{i=0}^N C_i T_i \quad \left( \frac{\mu\text{g}}{\text{m}^3} \cdot \text{s} \right)$$

Equation 2: Exposure as a product of concentration and time. This product is then divided by the total time interval to yield an average value.

Measuring exposure as an average concentration has a number of advantages over the dose method. Firstly, fewer assumptions must be made. We eliminate the breathing rate,  $B$ , from the equation (which is most certainly time variant), and we get rid of the body weight term,  $W$ , leaving only the time and concentration terms,  $T$  and  $C$ . While these parameters could easily be included in an application, which prompts the user to enter them, calculations based on these approximated values would introduce more error into the results. Secondly, guideline values [22,24,25] are expressed as concentration averages. This makes the average concentration readily comparable with guideline values. For these reasons we have chosen to use average exposure as the measure of exposure in our system.

### 6.2.1. Initial development

In part A of this thesis, I had written a proof of concept application, which using the methods described above, computes the pollution exposure given a GPS trace file. A screenshot of this program is shown in Figure 21.

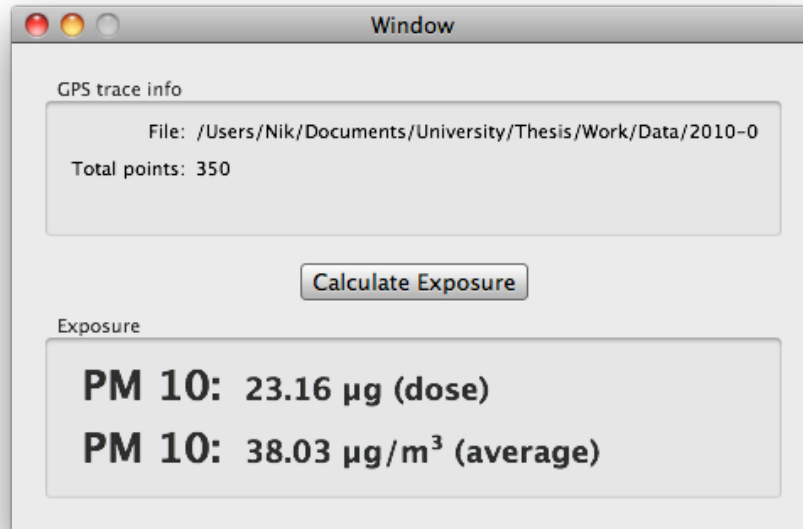


Figure 21: Screenshot of preliminary exposure calculation software

As input, the program takes a comma separated value (CSV) file of GPS coordinates and time. The program pulls pollutant concentration levels from the server for the locations and times contained within the GPS trace file, and from this data computes the exposure for that route using Equation 1 and Equation 2.

### 6.3. iPhone application

The objective of this application is to provide a user with a measure of their pollution exposure in a similar fashion to how a sensor device carried on their person would. The iPhone was decided on as the target platform due to its popularity and appeal in terms of user interface to a wide audience. It is also a platform I have a fair bit of experience with. Its core functionality is the same as the desktop Mac app previously developed, but with a couple of extra features not possible for a desktop application. The key features of this application are as follows:

- Records user location through time
- Calculates user's exposure to pollutants based on location trace
- Displays graphical representation of exposure over time
- Graphical view of recorded routes
- Allows user to check whether they have exceeded the guideline recommendation for pollution exposure

### 6.3.1. Overview

The application consists of 2 main screens. The first screen displays historical location traces made by this application in the past. Location trace files (same format as for the desktop variant) can also be uploaded to the application via iTunes and are included in this same list. A screenshot of the first screen is show in Figure 22.

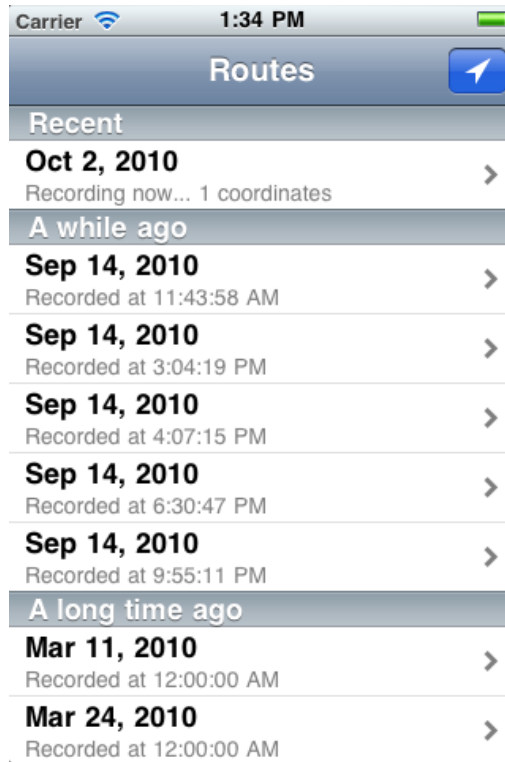


Figure 22: Screen capture of first screen of iPhone application

#### *Location tracking*

The blue button in the top right corner enables the user to toggle location tracking on and off. A blue highlight indicates tracking is on; the lack of this highlight indicating location tracking is turned off. This is a useful feature given that the application can run in the background (whilst using other applications), and the iPhone provides no means of quitting an application. Therefore, if the user wants to close the background and not use unnecessary battery power whilst the application is closed they can simply press that button before closing. Location traces are recorded as CSV files on the phone and can easily be retrieved and views on a personal computer.

Location traces are recorded in 'sessions'. Each session is intended to represent a journey from one point to another – i.e. from home to work. In case the user stops location tracking for some reason and then re-enables it, the application will

automatically resume the latest session if the time interval between stopping and re-starting location tracking is less than 30 minutes.

### Exposure plots

Selecting a row in the main table, i.e. a location trace, shows the second screen in the application. This screen, shown in Figure 23, shows a graph of the user's exposure over time and also computes the average concentration the user has been exposed to over their route. The red line plotted on the graph represents the recommended maximum exposure according to WHO or EPHC guidelines. The user can then easily see whether or not they have met these guidelines.

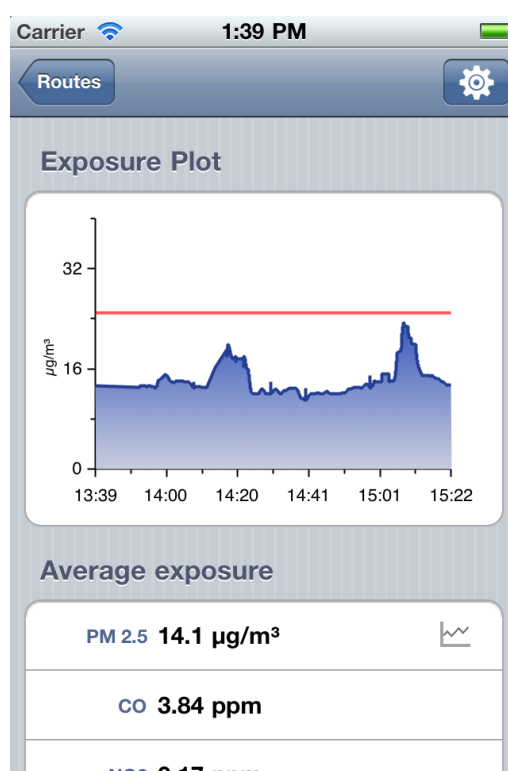


Figure 23: Second main screen of iPhone application. The view is too tall to fit entirely on screen, which is why the picture is slightly cut off. In the actual application, this view scrolls.

This screen currently displays information regarding the following pollutants:

- PM 2.5
- PM 10
- Carbon monoxide
- Nitrogen dioxide

The capability is there to display data for any pollutant in our database, however our database is currently only populated with significant amounts of data for the

pollutants listed above, and very little for others. Additionally our mobile sensing unit is current only capable of reporting concentrations for CO and NO<sub>2</sub>. Hence, display of other pollutants has been disabled in the current version of the application.

The average exposure experienced over the selected route is computed for each pollutant and displayed in the table. The pollutant graphed in the exposure plot may be selected by tapping on the corresponding row.

From this screen, the user may also view their location trace as recorded by the application. This screen is shown in Figure 24.

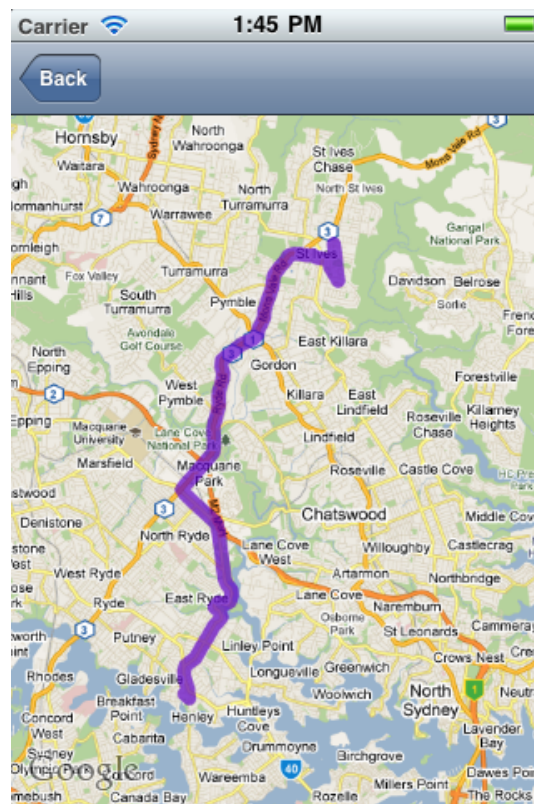


Figure 24: This view allows a user to see their route

Although this view is currently very basic, it could be extended in future to include an indication of the user's exposure over their route.

### 6.3.2. Technologies used

This application has been written in Objective-C using Apple iOS Cocoa APIs. These APIs provide a great deal of functionality built in – particularly with respect to the user interface and basic data structures. These APIs include styled sectioned tables, buttons and the Google Maps view shown in Figure 24. Although there is no built-in

abstraction for drawing graphs, a third party framework, by the name of CorePlot, has been used. Leveraging such an API has saved a lot of time compared to plotting the graphs with primitive drawing functions.

## 7. Evaluation

The system, as it stands today, functions as per the aims of this thesis. Since March 2010 we have been collecting and storing pollution readings from the DECCW as well as a number of our own readings in our database. Both the mobile pollution sensor unit and the associated mobile application function, as well as the database server and personal pollution exposure iPhone application.

Presented here are some results and an evaluation of each component of the system.

### 7.1. Server

#### 7.1.1. Batch query optimization

The results of a few comparison tests I have conducted are displayed in Figure 25, for the improvements introduced into the server software for generating multiple pollution levels in batches. The technique and implementation of these improvements was described in Section 4.5. The request time is still linear with the number of data points, as before, but about 10-100x faster than previously. Practically, this means a user, instead of waiting 1-2 minutes per request, would wait only about 0.5 seconds per request.

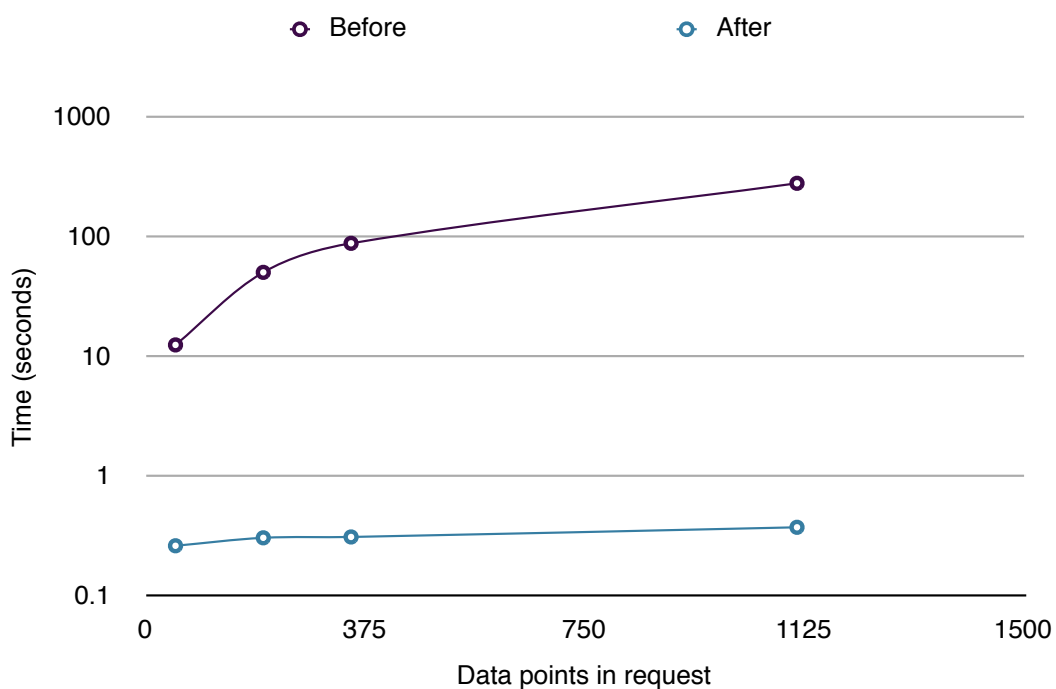
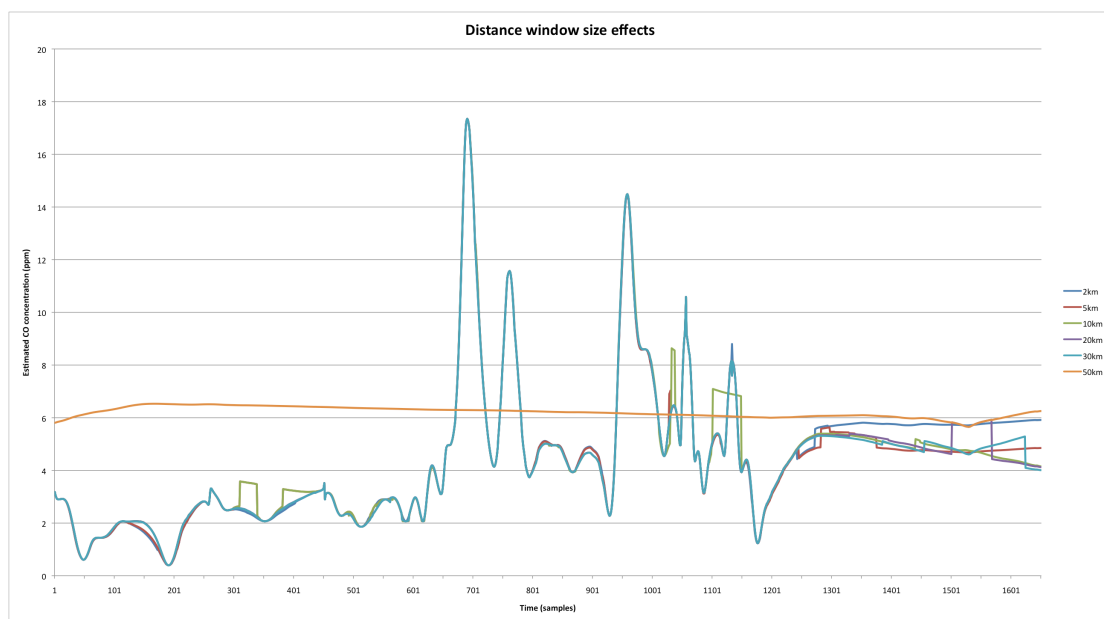


Figure 25: Comparing query times before, and after optimization

As with most performance improvements, there exists a delicate balance between the precision of the results and the computation time. In this optimization there are two parameters to vary which effect both the accuracy and computation time of the results. These are the location distance window size, and the time window size. Given that our main source of data at the present time is from the DECCW with an update frequency of an hour, our data does not change very rapidly. For this reason I have selected a 15 minute time window, a value small enough to account for the hourly refresh of data and to also take into consideration any other values which may be imported by our pollution measuring devices. With more real-time data in future, this value may need to be revised, but as it stands right now there is insufficient data to be able to choose a more appropriate value.

The critical parameter, then, is the distance window. Figure 26 shows the variation in estimated carbon monoxide concentrations (by the model) for a series of locations.



**Figure 26: Effects of distance window size on estimated carbon monoxide concentrations, by the model, using optimized queries for a request of about 1600 points**

The graph shows that for location windows of size 2km-30km, the reported concentrations are fairly similar. When the window size is adjusted to 50km, however, the model deteriorates drastically and all detail observed with smaller window sizes is lost. The square irregularities present in the above curves were characteristics of the model at the time of testing [4].



The distance window size also affects the computation time of the model. For a request of 3049 points, using the 15 minute time window specified earlier, the computation time as a function of the distance window has been included in Figure 27.

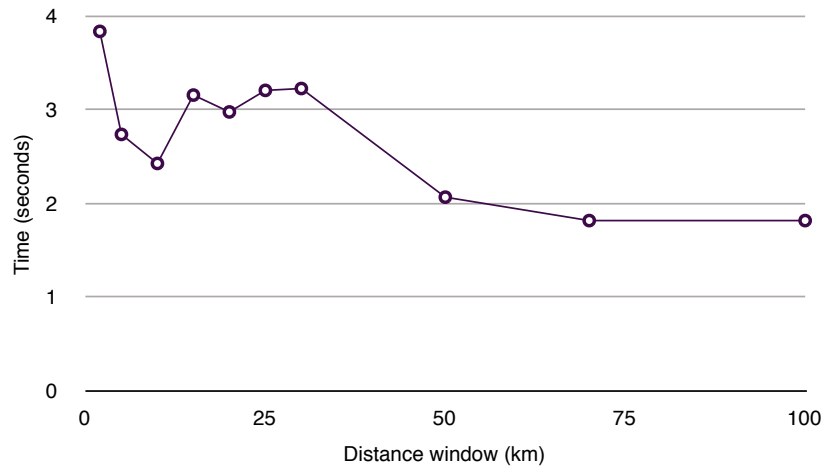


Figure 27: Model computation time as a function of distance window for a 3049-point request

It is interesting that the computation time actually increases around the 25km mark after having decreased from the initial best-case estimate. The most likely explanation for this is that as the size of the distance window increases, so too does the number of points included in the calculation. The initial high computation time arises from few data points, but frequent cache flushes. The next high point in computation time (around the 25km mark) arises from fewer cache flushes, but more data points. As the distance window increases, fewer cache flushes are required thus reducing computation time.

Taking the results from both Figure 26 and Figure 27 into account, the optimal distance window – balancing computation time and accuracy is about 20km. This is the reason for that choice.

It should be noted that these optimizations are dependent on the implementation of the model, which was not the focus of my thesis, rather that of Amanda Chow [4]. These results are true as of the time of writing, however may vary as the model development advances in the future. The optimization concepts presented here are still valid, however, the parameter values may require some adjustment for a different model implementation.

### 7.1.2. Database

The database began recording samples on 19 March 2010. Since then, data has been added at an average rate of 512 new rows per day.

In theory, we can store an infinite number of records in the database. Practically, however, we are limited by the storage constraints of our server and by the database software. Fortunately, MySQL has no limitation on the maximum size of a database [16]. The actual maximum size is determined by the available disk space and constraints imposed on file sizes by the file system. For example, the FAT32 file system used by older versions of windows has a maximum file size limit of 4GB. Since we chose the Linux operating system, this constraint is of no concern for our server.

The samples table, as of 1 October 2010, had a total 100,052 rows with an average size each of 116 bytes. The particular version of Linux in use by our server uses the *ext4* (fourth extended file system) file system, which has a maximum file size of 16 TB – well exceeding the current hard disk capacity of 220 GB. The maximum capacity of the database, then, is limited solely by the available hard disk space. Assuming 100 GB of space available for the storage of the samples table and an average row size of 116 bytes, we can store approximately 860 million rows of data before having to worry about upgrading our storage capacity. Even with 100,000 new samples added each day (200x the present level), the database will have sufficient storage capacity for over 20 years of continuous data.

## 7.2. Hardware interface

The Android mobile application, facilitating an interface to our sensor hardware, works as intended. We have successfully demonstrated that it can connect to our hardware and extract the latest readings. It can then package these readings in xml form, and upload them to the database server.

In addition to regular testing during development, we conducted a test, to test the functionality and stability of the mobile pollution sensor Bluetooth interface. We did this by driving for about 1 hour 30 minutes around Sydney (approx 60km).

### Setup

A single car departed UNSW at approx 1:45pm, driving along the route shown in Figure 28, and arriving back at UNSW at 3:30pm. Throughout the journey the

hardware sensor was switched on and the mobile application was running, connected to the device.

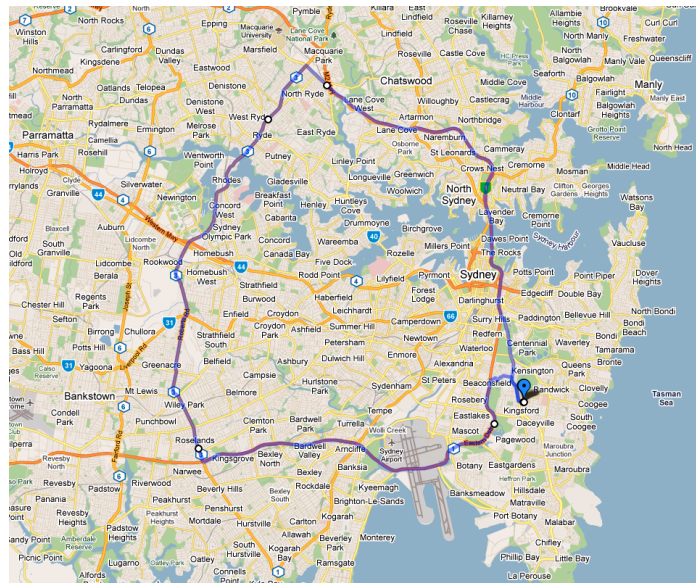


Figure 28: Map of route driven around Sydney

## Results

Over the 1 hour 28 minute journey, 143 samples were recorded and uploaded to the database. This represents a collection rate of 1.6 samples per minute. Figure 29 shows the points at which samples were collected along the journey.

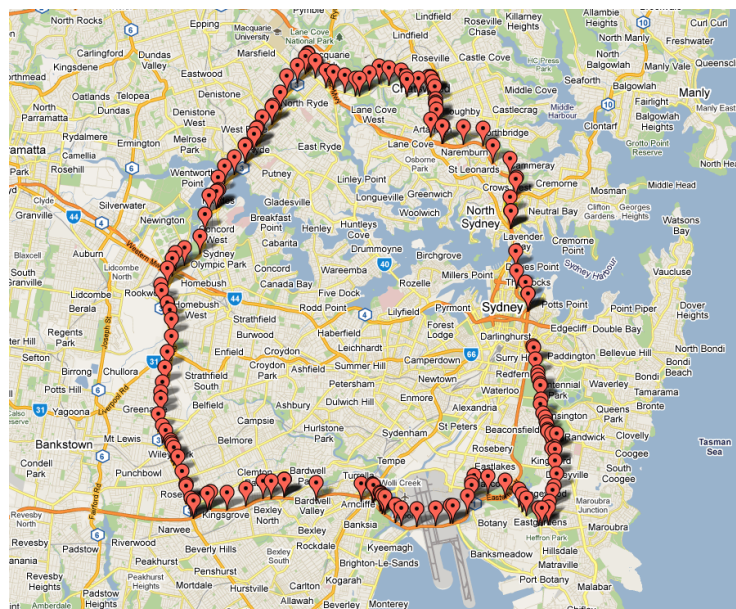


Figure 29: Locations of 143 data points collected and uploaded by our sensor device mobile application

During this experiment, samples of carbon monoxide and nitrogen dioxide were collected, 143 of each. Figure 30 displays a graph of the carbon monoxide readings collected during this experiment. The NO<sub>2</sub> readings collected displayed a steady upwards trend, indicative of a hardware and/or calibration fault. These results are not very meaningful and have thus not been included in this report.

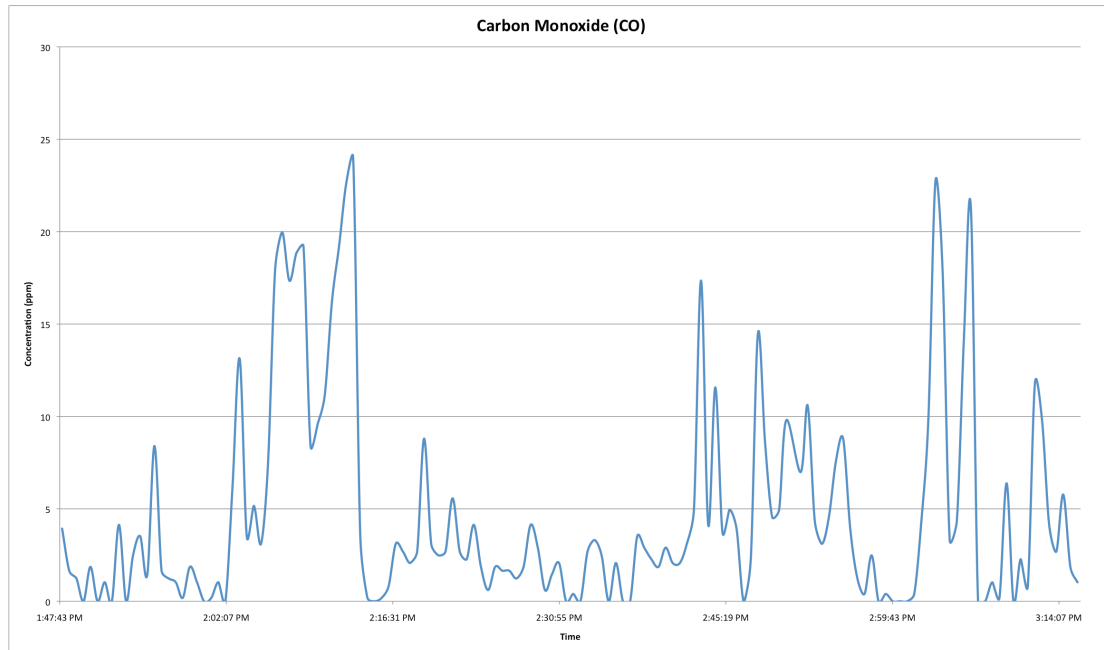


Figure 30: Carbon monoxide concentration readings collected and uploaded to the server during the journey

### Discussion

This journey also included 4 tunnels (airport tunnel, M5 East, Harbour tunnel and the Eastern Distributer tunnel). The GPS built into the Android Smartphone failed to function in the tunnels. Consequently any readings taken in the tunnels were reported at the last known location – i.e. at the entrances of the tunnels. This is the reason for the sparsely located data points and gaps around these locations. Upon closer inspection there were a number of stacked data points at the entrance of the tunnel.

The graph in Figure 30 shows several spikes of carbon monoxide concentration, which correspond to locations of tunnels and major roadway intersections. The first rather wide spike was recorded in the M5 east tunnel and displays much higher levels of CO than other locations. This was expected, and it is encouraging to see that our results reflect that.

These results show that both the sensor hardware and the mobile application function as intended for extended periods of time. Measurements were taken consistently, and the results look reasonable. Work will have to be done on the hardware and calibration to investigate the problems with NO<sub>2</sub> measurement.

## 7.3. Personal pollution exposure application

### 7.3.1. Verification tests

As part of the development process, the personal pollution monitor application was repeatedly tested for validity against known data. Here, I will outline the nature of some of the tests conducted in the final stages of development.

#### *Correct GPS data*

On various occasions in my travels to and from university, and around Sydney, I launched the personal pollution exposure application and tracked my location over that journey. Figure 31 displays one such journey as recorded by the application.

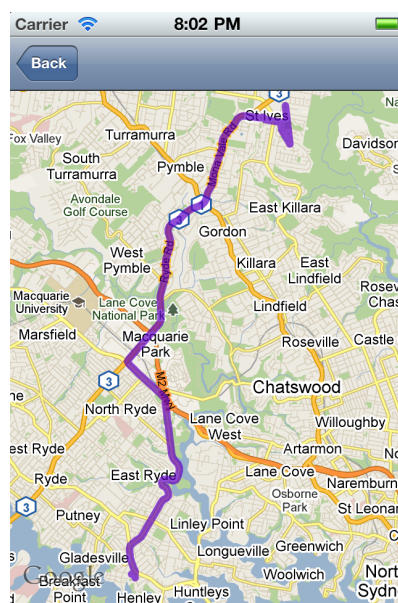


Figure 31: Recorded location trace for data verification test. The square kink in the route, near Macquarie Park, is an actual road.

#### *Correct pollution levels and exposure computations*

In addition, to confirm that the data displayed by the application was correct when compared to the data provided by our database server, I performed a comparison between the graph and average value shown on screen to that captured directly from the server's data export interface.

Figure 32 shows the exposure plot for NO<sub>2</sub> as displayed on the iPhone, while Figure 33 shows the plot of the pollution level data for the same pollutant downloaded directly from the database server, using the same location trace.

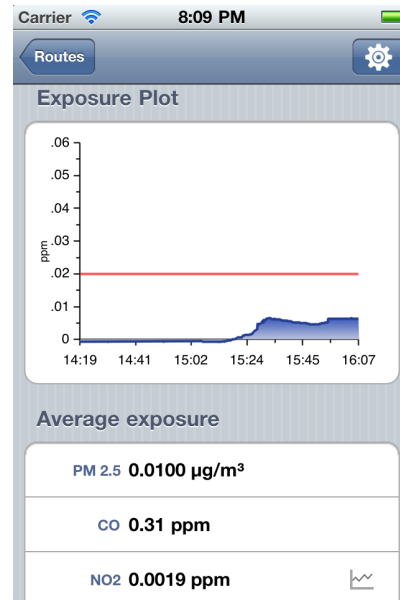


Figure 32: NO<sub>2</sub> exposure graph, as displayed in the iPhone application including a value for average exposure

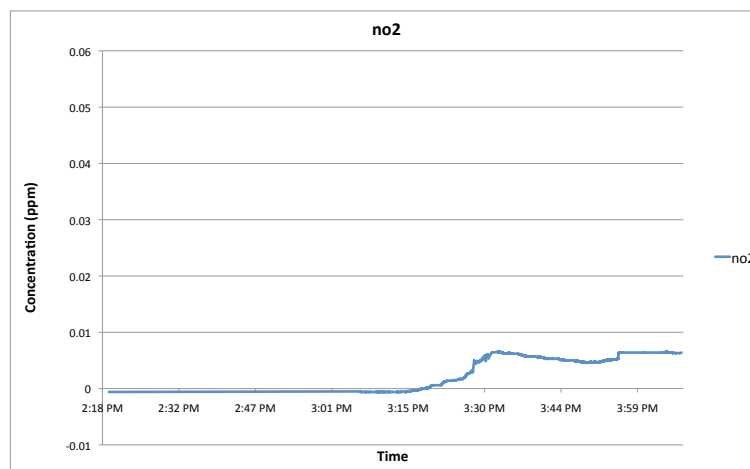


Figure 33: NO<sub>2</sub> exposure graph, drawn in Microsoft Excel using data directly downloaded from the database server

The average value for exposure as computed by Microsoft Excel using the methods outlined in Section 6.2 was 0.001860683.



### 7.3.2. Field test

This experiment was carried out as part of the experiment outlined in Section 7.2, and was intended to verify that the personal pollution exposure application functions as intended in a real-world usage scenario.

#### *Aim*

To test the server's and the iPhone application's stability and reliability for large sets of collected data, and also to record the actual exposure to NO<sub>2</sub> and CO, and to compare that to the estimates produced by the model.

#### *Setup*

A second car departed approximately 15 minutes after the first car fitted with the sensor hardware (as described in Section 7.2). This second car was equipped with a GasAlert Micro air pollution monitor, and an iPhone running the personal pollution exposure application. The GasAlert Micro is a commercial gaseous pollutant meter capable of measuring CO and NO<sub>2</sub> concentrations in air. This was suspended in the front passenger window of the car, and the window was left open for the entire trip as to allow the outside air to flow past the meter. This setup is shown in Figure 34.



Figure 34: GasAlert Micro setup in car

The route taken by the second car differed slightly to the first car with respect to the final destination. The route as recorded by the iPhone application is shown in Figure 35.

For this experiment, the first car also recorded concentrations of particulate matter: PM2.5 and PM10, using a commercial particulate matter sensor.

Samples recorded by the particulate matter sensor were recorded onto the device's internal memory and later uploaded to our server. Samples recorded by the GasAlert Micro, on the other hand, were not uploaded to the server; rather they were used to solely to compare the results of our model with actual real-world exposure.

### Results

The personal pollution monitor application recorded a total of 3049 locations during the 1 hour 20 minute journey. The application was able to run in the background without incident, and consumed only a small amount of the battery power (5-10%). Based on the recorded GPS route and associated requests to the server, and its responses, graphs for PM 2.5, PM 10, CO and NO<sub>2</sub> exposure for the journey were produced by the application. The estimates for CO are shown in Figure 36.

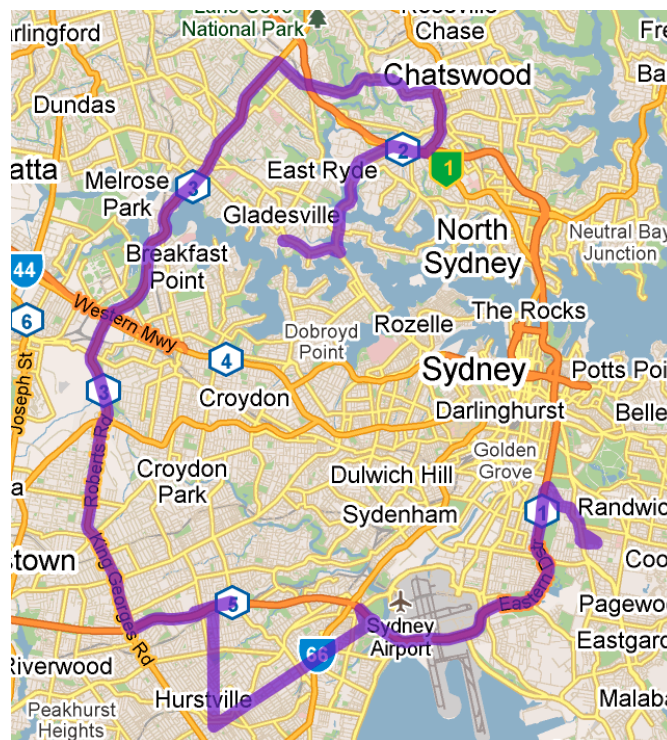


Figure 35: Route map of second car as recorded by the iPhone application.



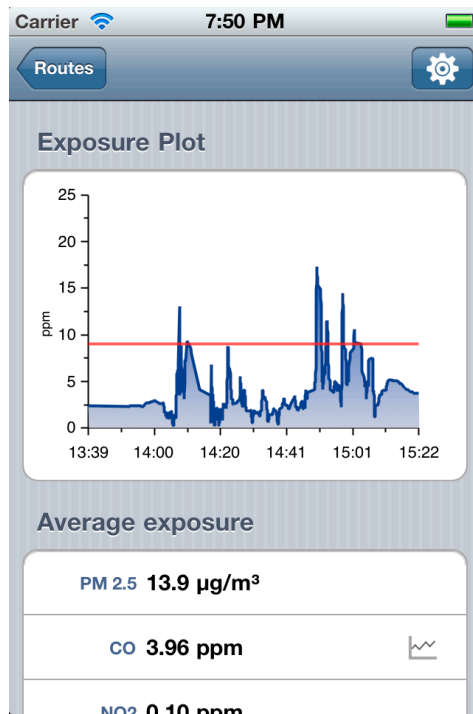


Figure 36: Carbon monoxide estimation based on location recordings taken by the iPhone, as displayed by the personal pollution exposure application

The concentration of CO recorded by the GasAlert Micro device is presented in Figure 37.

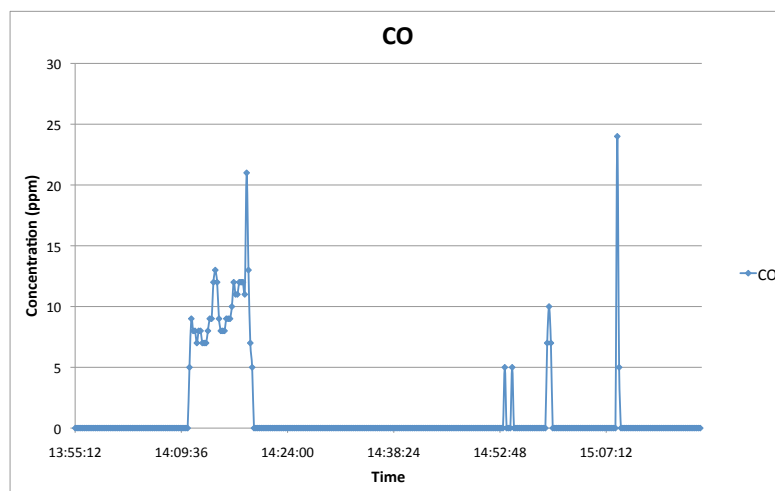


Figure 37: CO exposure as measured by the GasAlert Micro. This should represent the actual exposure.

The average value of these readings is 0.96 ppm.

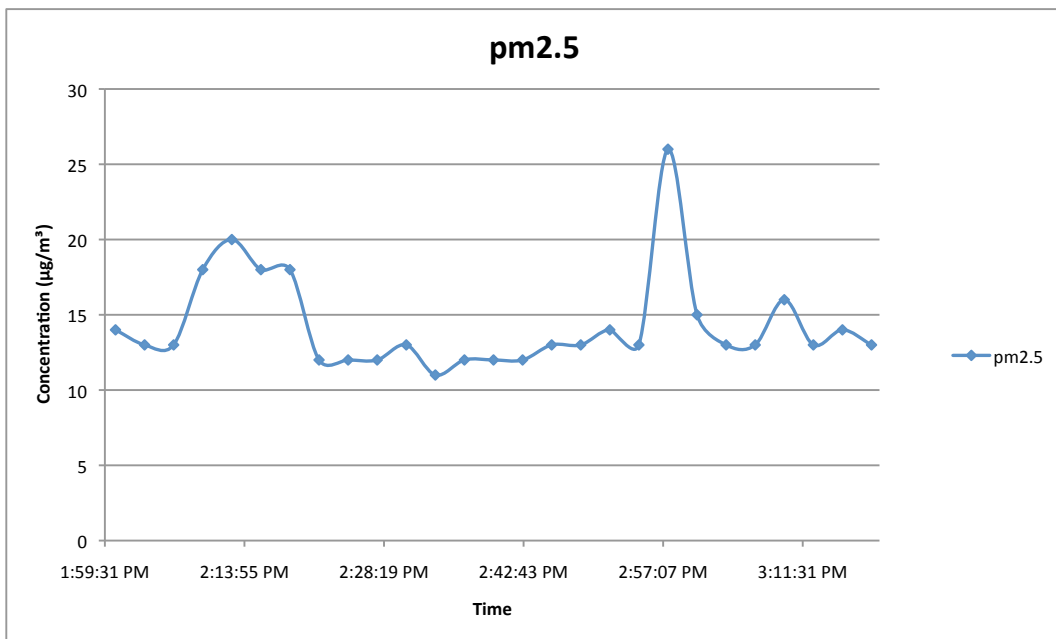
These measurements overlaid with the data plotted on the iPhone application, and the estimation of exposure sourced from the DECCW's data directly is shown in Figure 38.



**Figure 38: Graph of carbon monoxide concentration as i) estimated using DECCW+experimental data, ii) measured by the GasAlert Micro, iii) Estimated by the DECCW data alone**

The DECCW’s fixed CO measurement sites reported an average CO concentration of approximately 0.5 ppm around the testing circuit over the testing period.

The recorded concentrations of PM 2.5 and PM 10 for this journey as recorded by the particulate matter pollution meter (first car) are shown in Figure 39 and Figure 40, respectively.



**Figure 39: Concentration of PM 2.5 as recorded by the first car**

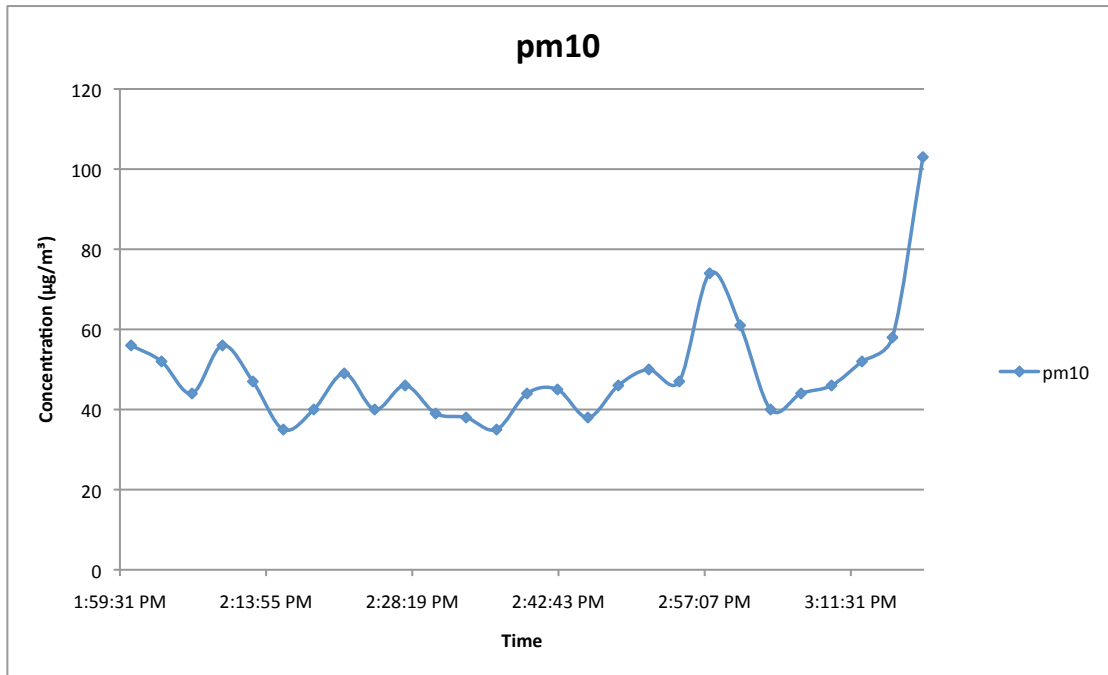


Figure 40: Concentration of PM 10 as recorded by the first car

The plots of PM 2.5 and PM 10 exposure as reported by the iPhone personal pollution exposure application are shown in Figure 41 and Figure 42.

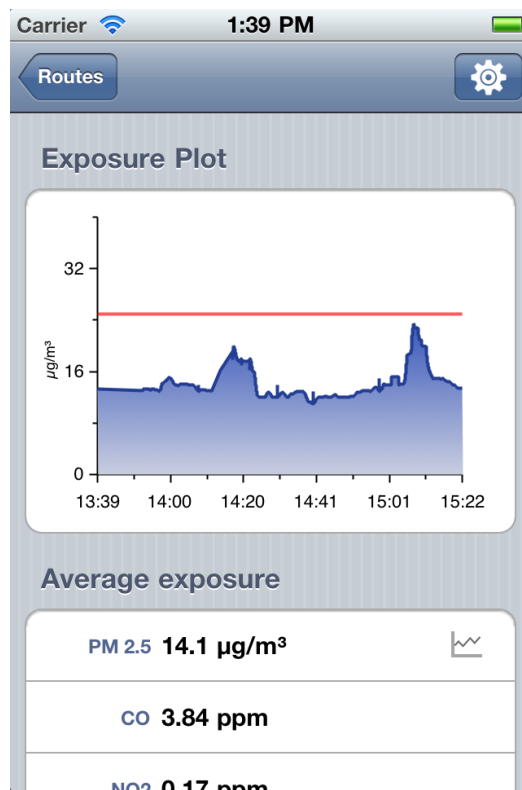


Figure 41: Plot of PM 2.5 exposure in the iPhone application

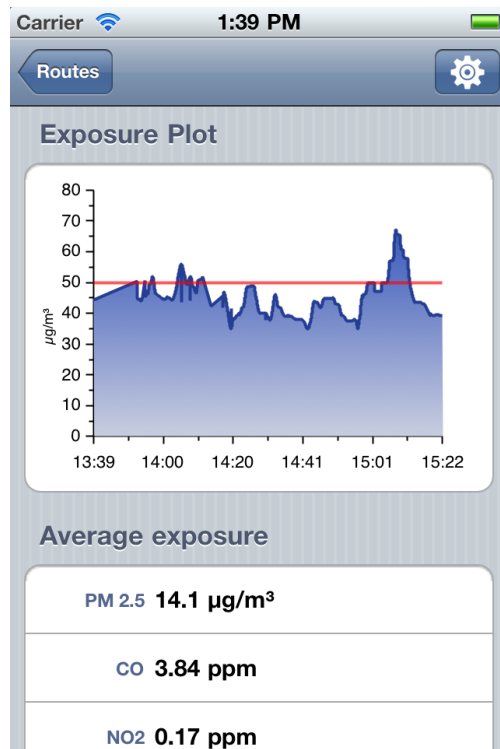


Figure 42: Plot of PM 10 exposure in the iPhone application

### 7.3.3. Discussion

#### *Verification tests*

The route map, shown in Figure 31, accurately reflects the route that I drove. There is a slight deviation at the north end of the route, which is a result of the iPhone losing GPS reception in a heavily forested valley, and resorting to less accurate means of computing its location.

The exposure plots in Figure 32 and Figure 33 do appear to be the same, which is the desired outcome. It is worth mentioning that these plots both depict a series of negative values over the first half of the plot. This is not an error with either the server or the iPhone application; rather it is a characteristic of the model software or the data itself<sup>8</sup>. Most importantly, the value for average exposure as computed by the iPhone application is the same as that computed using Excel and the methods outlined earlier.

These tests have shown that the iPhone personal pollution exposure application is able to record location data reliably, and to subsequently request pollution levels from the

---

<sup>8</sup> I have observed on a number of occasions that the DECCW publishes negative readings for NO<sub>2</sub> – this fact is the most likely source of the observed characteristics.

server based on recorded location traces. The graphs displayed in the application accurately reflect the data output by the interpolation model running on the server, and average exposure is computed correctly.

### *Field test*

This experiment produced the highest number of data points yet seen during development (3049). The fact that the personal exposure monitor application worked and was able to produce a plot is an indication that both the data export interface and the interpolation model on the server are also able to handle large sets of data and still produce results.

The results for the personal pollution exposure monitor were as expected – data from the model was successfully exported and displayed in the iPhone application. For example, the plot of carbon monoxide exposure on the iPhone in Figure 36 closely matches the plot of raw data in Figure 38.

The route map of Figure 35 shows a strange deviation from the route around the vicinity of the M5 east tunnel. This was again due to the iPhone losing GPS reception in the tunnel, and falling back to cell triangulation for location computation. Dealing with GPS issues such as this in tunnels is an area of research from which this project could benefit greatly.

Regarding the GasAlert Micro carbon monoxide readings, shown in Figure 37, these are quite poor and are not of the standard what we expected. It appears that the GasAlert Micro is insensitive to CO concentrations below 5ppm – well above ambient levels. Consequently, the results show only major spikes in CO concentration. The first, much wider such spike can be attributed to the conditions inside the M5 east tunnel, which is known for its highly polluted air [26], the other spikes corresponding to selected major intersections.

The results, as they stand, are insufficient to be able to draw a conclusion as to the accuracy of the exposure estimation of the model. In order to do this it is clear we need firstly a more sensitive meter to measure the real exposure, and secondly several more vehicles collecting data simultaneously in relatively close proximity.

What we can say, however, is that using the data of the DECCW alone is insufficient to calculate an approximate value for an individual's exposure. The intermittent data of the GasAlert Micro yields an average value of 0.96ppm, which is higher than the

approximated 0.5 ppm derived from the DECCW data alone. Given that any readings collected by the GasAlert Micro below 5ppm were recorded as 0, this average would be higher given a more sensitive meter. The estimated average of 3.96 ppm provided by our exposure estimation software, then, may not be too far off.

The graph in Figure 36 looks slightly similar to the graph in Figure 30 (CO concentration as recorded by the first car). This is expected, as the same route was taken with a 15-minute time delay. Furthermore, the plots of PM 2.5 in Figure 39 and Figure 41 are very similar in shape, and the key spikes in concentration exhibit a constant delay of about 10-15 minutes, corresponding roughly to the time separation between vehicles. The plots of PM 10 in Figure 40 and Figure 42 display similar traits, but the correlation between them is not as strong.

The fact we get a similar plot on the iPhone personal pollution exposure application as for the recorded data 15 minutes earlier suggest that the model is utilizing the newly uploaded data in its computations, and that the iPhone is able to access this data and faithfully display it to a user.

Together these results show that the foundations of the Haze Watch system are working. In particular, the components I have been focusing on, namely the database server, iPhone application and hardware interface, have demonstrated their utility in integrating the other major components of the system developed by my fellow team members.

## 8. Future work

The work we have undertaken this year has laid the foundations of the system set out in the aims. There are a number of improvements that could be made to this system in the future. In this section we will explore a few of these relating in particular to my roles in this thesis.

### 8.1. Server and website

#### 8.1.1. Website user interface

The project website currently features a map of pollution levels, for various pollutants for a given time window. A future improvement to the website would be a feature that users would use to login and track their exposure, much like the iPhone application, but online.

Users with the mobile sensing unit could also view the readings they uploaded via their devices, and statistics in relation to other users and the system as a whole. The potential is also here to provide users with some kind of motivation for uploading more data.

#### 8.1.2. Data export format

As originally intended, the server-side software could be further expanded to support the export of data in XML, plus optionally other formats. In addition, the capability to export the confidence levels of calculated data needs also to be included.

### 8.2. Personal pollution exposure iPhone application

#### 8.2.1. User interface

While the iPhone application is relatively user friendly in it's current state, it could do with some nice graphics and some layout changes. Currently the application is capable of overlaying the recorded routes over a map. In the future this functionality could be extended to also somehow overlay the pollution exposure over the route on the map, perhaps in a similar way to the popular running application '*Nike+GPS*', pictured in Figure 43, where the route is coloured according to your recorded speed.

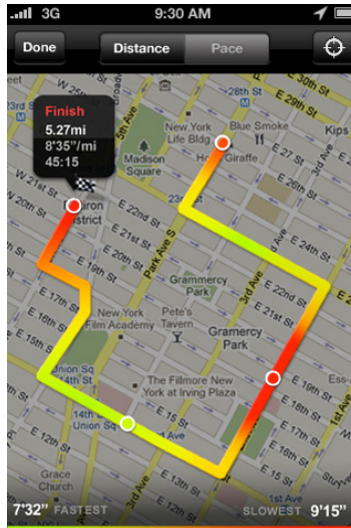


Figure 43: Screenshot of Nike+ iPhone application showing colour gradients over the route

### 8.2.2. Confidence levels

The iPhone application currently has the ability to compute an average value of exposure for a particular route based on information provided by the interpolation model. Naturally the pollution levels are estimated based on the set of available samples. The model has the ability to report the errors involved with each pollution level calculation. This feature was only recently implemented and has not yet made it to the iPhone application. An important feature for the future is the ability to include a level of confidence, or error for each exposure calculation, perhaps even the option to display error bars on the exposure plot.

### 8.2.3. GPS accuracy

As demonstrated by the results of tests we conducted, the accuracy of location information provided by the phone is occasionally fairly poor. For example in tunnels, and on roads with many overhanging trees, the iPhone fails to receive any GPS signal at all and instead reverts to using cell or WiFi triangulation to compute its location. A prudent improvement for the future would be to filter out such inaccurate locations, instead only recording locations 'good' accuracy.

### 8.2.4. Path simplification

In the tests we conducted, we have found that over a long period of time many location points are collected – approximately 30 every minute. It is nice to have this resolution on detailed journeys, but completely unnecessary on straight roads. If this data was simplified somehow, exposure estimates could be computed faster and server load would also decrease substantially.



## 8.3. Mobile pollution sensor Android application

### 8.3.1. User interface

The Android application for the interface between the sensor hardware and the server was designed only for testing purposes. The interface is quite unintuitive, and the application does not behave as a user of the platform would expect upon termination. For instance, some background location tracking is still active after the application exists. This deficiency was perhaps due to my lack of experience with the platform – I spent a lot of time figuring out *how* to do something, and very little on making it look good. Also the current method of selecting a device to connect to is very basic and could easily confuse a user. For example, in the device selection screen, *all* Bluetooth devices are shown, regardless of type. A simple improvement to this experience would be to show *only* mobile pollution sensor hardware.

### 8.3.2. Uploading extra information

From the collected GPS data, an approximate speed of the pollution sensor can be calculated. It may be beneficial to collect this data for use in model estimations. Additionally, this data could be quite valuable in determining the state of traffic across a city. This could not only be useful for the obvious purpose of avoiding traffic, but could also help the model to interpolate samples by considering that there is likely to be a higher pollution concentration amongst slow moving traffic.

### 8.3.3. Privacy concerns

Whilst collecting data, the almost-real-time location of that user is periodically uploaded to the server. This information could be used by an eavesdropper in nefarious ways, and is a breach of the user's privacy. To prevent this type of attack, the information could be encrypted before being sent such that the server is the only party able to decrypt the information. An RSA public/private key pair system could be one possible solution to this problem.

### 8.3.4. Tunnels

Similar to the problems identified with the iPhone application with respect to GPS accuracy, the Android application also suffers from the same problem, particularly in tunnels. Research into methods for accurately determining one's location inside a tunnel in the absence of GPS data could be of great benefit to this project.

## 9. Conclusion

Current systems in place to monitor air pollution levels typically have only a handful of fixed monitoring stations distributed over an entire city. Driven by a need for higher resolution, cost effective air pollution data to aid epidemiological studies and to better inform any interested members of the general public, the proposed Haze Watch system is being developed.

Using motor vehicles as a platform for our hardware, the Haze Watch system utilises existing internet-enabled Smartphones to act as a conduit to transmit air pollution readings to our server. These key elements in our design allow our system to collect data over a wide area for a lower cost, and to also monitor pollution at the source presenting better estimates of actual exposure.

The work undertaken in this stage of the project has laid the foundations of the Haze Watch system set out in the aims of this project. In particular, the components I have been assigned to work on, namely the database server, a mobile application interface for the sensor hardware, and the personal pollution exposure monitor iPhone application have all been implemented and function as intended. We have demonstrated that components of the system work, and that my work has successfully integrated these functioning components into a functioning system.

The results of the tests we have conducted so far have proven that the basic system we set out to implement in this phase is operational and is producing reasonable results. Our preliminary results are also promising with regard to the aims of the project overall – particularly with respect to the quality of estimated exposure compared to real exposure. Although there is still more testing to be done, as it stands, the Haze Watch project is now well underway, and is ready for such testing and expansion to more users.

## References

- [1] World Health Organization. (2008, August) *Air quality and health, Fact sheet no. 313*. [Online, accessed 24 May 2010]. <<http://www.who.int/mediacentre/factsheets/fs313/en/index.html>>
- [2] AEA Group, "Air Quality Pollutant Inventories for England, Scotland, Wales and Northern Ireland: 1990 – 2008," National Atmospheric Emissions Inventory, United Kingdom, 2010.
- [3] James Carrapetta, "Haze Watch: Design of a wireless sensor board for measuring air pollution," School of Electrical and Telecommunications Engineering, University of New South Wales, Sydney, Thesis 2010.
- [4] Amanda Chow, "Haze Watch: Data Modelling and Visualisation using Google Maps," University of New South Wales, Sydney, Thesis 2010.
- [5] *GreatAir Manchester*. [Online, accessed 13 October 2010]. <<http://www.greatairmanchester.org.uk/GreatAir/default.aspx>>
- [6] W H Roemer and JH Wijnen, "Daily mortality and air pollution along busy streets in Amsterdam," *Epidemiology*, vol. 12, pp. 649-653, 2001.
- [7] Vanderbilt University. *MAQUMON*. [Online, accessed 24 May 2010]. <<http://www.isis.vanderbilt.edu/projects/maqumon>>
- [8] W Hedgecock, P Volgyesi, A Ledeczi, and X Koutsoukos, "Dissemination and Presentation of High Resolution Air Pollution Data from Mobile Sensor Nodes," Institute for Software Integrated Systems, Vanderbilt University, Nashville, 2007.
- [9] *SenSaris*. [Online, accessed 24 May 2010]. <<http://www.sensaris.com/>>
- [10] Lynne Peebles. (2009) *iSniff: Pocket-Size Pollution Sensors Promise Big Improvement in Monitoring Personal Environment*. Scientific American. [Online, accessed 24 May 2010]. <<http://www.scientificamerican.com/article.cfm?id=air-pollution-monitoring-sensor-asthma-black-carbon>>
- [11] *Common Sense Project*. [Online, accessed 2010 October 2010]. <<http://www.communitysensing.org/>>
- [12] *MESSAGE Project*. [Online, accessed 6 October 2010]. <<http://bioinf.ncl.ac.uk/message/>>
- [13] J Burke et al., "Participatory Sensing," *ACM*, 2006.
- [14] NetCraft. (2010) *May 2010 Web Server Survey*. [Online, accessed 21 May 2010]. <[http://news.netcraft.com/archives/web\\_server\\_survey.html](http://news.netcraft.com/archives/web_server_survey.html)>
- [15] MySQL. *Market Share; Database Installations and Deployment Plans - 2008*. [Online, accessed 30 September 2010]. <<http://www.mysql.com/why-mysql/marketshare/>>
- [16] MySQL. (2010) *MySQL 5.0 Reference Manual - B.5.12 The table is full*. [Online, accessed 16 October 2010]. <<http://dev.mysql.com/doc/refman/5.0/en/full-table.html>>
- [17] Gartner. (2010, February) *Gartner Says Worldwide Mobile Phone Sales to End Users Grew 8 Per Cent in Fourth Quarter 2009; Market Remained Flat in 2009*. [Online, accessed 16 September 2010]. <<http://www.gartner.com/it/page.jsp?id=1306513>>

- [18] Gartner. (2010, August) *Gartner Says Worldwide Mobile Device Sales Grew 13.8 Percent in Second Quarter of 2010, But Competition Drove Prices Down*. [Online, accessed 16 September 2010]. <<http://www.gartner.com/it/page.jsp?id=1421013>>
- [19] Andrew Colley. (2010, June) *Surging iPhone hot on the heels of Nokia as Australia's No 1 smartphone*. [Online, accessed 16 September 2010]. <<http://www.theaustralian.com.au/australian-it/surging-iphone-hot-on-the-heels-of-nokia-as-australias-no-1-smartphone/story-e6frgakx-1225879621669>>
- [20] IEEE, 754-2008 *IEEE Standard for Floating-Point Arithmetic*, 2008.
- [21] Auditor General, NSW, *AUDITOR-GENERAL'S REPORT: FINANCIAL AUDITS*. Sydney, NSW, Australia, 2009, vol. 10.
- [22] World Health Organization, "Air Quality Guidelines: A Global Update 2005," 2006.
- [23] National Academy of Sciences, *Human Exposure Assessment for Airborne Pollutants: Advances and Opportunities*. Washington D.C.: National Academy Press, 1991, pp. 37-42.
- [24] Environment Protection and Heritage Council. (1998) *Ambient Air Quality NEPM*. [Online, accessed 24 May 2010]. <<http://www.ephc.gov.au/taxonomy/term/23>>
- [25] Environment Protection and Heritage Council, "National Environment Protection (Ambient Air Quality) Measure," 2003, p. 12.
- [26] SOUTH EASTERN SYDNEY PUBLIC HEALTH UNIT & NSW DEPARTMENT OF HEALTH, "M5 East Tunnels Air Quality Monitoring Project," Sydney, 2003.