

---

**The University of New South Wales**

**School of Electrical &  
Telecommunications Engineering**

**TELE4121  
Thesis B Report**

***“Haze Watch project:  
Communication between sensors  
and mobile phone”***

**Zhang Hailian**

**3313360**

**Supervisor: A/Prof. Vijay Sivaraman**

**20<sup>th</sup> October 2011**



## Thesis Pointers

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

14, 30, 33	Problem Statement
7	Objective

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

33	Two-way communication
18	Map-matching
33	EUASRT module
29	Protocol

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

33-34	Two-way communication circuit and programming
31-32	Protocol version 2
23-26	Non-linear interpolation
20-21	Linear-interpolation
39	Upload Buffer

### Contributions (most important first)

33-35	Realized two-way communication in both hardware and software
23-26	Implemented non-linear interpolation
32-32	Implemented new transmission protocol
35-36	Implemented Displacement-based sampling mode
20	Implemented linear interpolation
29	Improved upload buffer size added timeout function

### My work

11,16,18, 24-27,32,39	System block diagrams/algorithms/equations solved
21-22,27-28,34-35,37-38	Description of procedure (e.g. for experiments)

### Results

21-22,27-28,34-35,37-38	Succinct presentation of results
21-22,27-28,34-35,37-38	Analysis
22,27, 35-36	Significance of results

### Conclusion

42	Statement of whether the outcomes met the objectives
40	Suggestions for future research

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

12	[9] Microchip
12, 31	[10] Adeunis R.F.
19	[14] LUKIANTO .C & STERNBERG H
18,19	[13] Dimitris s. & Nathan W. et al.
7,10,30	[5] N. Youdale

## Abstract

Haze watch system is a system that is able to monitoring air pollution. It consists of a mobile sensor unit, a database server and a client application. Sensor unit is used to measure air pollution and upload to server. Database server provides an interface for client application to access measured data. Currently, the project has been divided into several parts. This report will focus on communication within mobile sensor unit.

# Acknowledgement

I would like to acknowledge my supervisor, A/prof. Vijay Sivaraman who has provided me strong inspiration, excellent guidance and support for my thesis. I would also like to thank my group partner, Junjie Jiang, with whom I have cooperated to complete all the work involved in this thesis. He has made great contribution to our work. I would like to further thank other project members and former fellows, Dawei Lu, Kunxuan Bi, James Carrapetta, and Nikolaus Youdale for their help, support, and valuable hints.

# Table of Contents

1. Introduction.....	7
2. Background.....	9
2.1 Reasons for mobile air pollution monitor .....	9
2.2 Similar project .....	10
3. Overview of the system .....	11
3.1 Sensor unit.....	12
3.2 Database server .....	13
3.3 Client application .....	13
4. Analysis and solutions .....	14
4.1 Positioning signal lost .....	14
4.1.1 Location request method .....	14
4.1.2 Location signal lost in tunnel.....	14
4.2 Solutions for Positioning signal lost .....	15
4.2.1 Filtering method .....	16
4.2.2 Related solutions for indoor positioning.....	18
4.2.3 Linear interpolation.....	20
4.2.4 Linear interpolation test result .....	21
4.2.5 Non-linear interpolation.....	23
4.2.6 Non-linear interpolation test result .....	27
4.2.7 Integrated interpolation.....	29
4.3 Transmission protocol.....	29
4.3.1 Overview of protocol version 1 .....	29
4.3.2 Transmission drawbacks .....	30
4.3.3 Version 2 protocol.....	31
4.4 Two-way communication .....	33
4.4.1 Circuit build for two-way communication.....	33
4.4.2 New functions .....	35
4.4.3 Test results .....	37
4.5 Upload Method .....	39

4.5.1	Buffer size .....	39
4.5.2	Timeout method .....	39
5.	Future work.....	40
5.1	Interpolation.....	40
5.2	Two-way communication .....	41
5.2.1	Reception protocol.....	41
5.2.2	New functions .....	41
6.	Conclusion .....	42
	References .....	43
	Appendix.....	45

# 1. Introduction

Air pollution has become a worldwide issue that has been frequently discussed in recent years. Pollutants in the air may severely affect the environment and raise the risk of getting disease for humans. However, most of the current air pollution monitoring systems fixed stations hosting by government. As a result, individuals have less chance to monitoring the pollutants level around them.

A new type of air monitoring system, Haze Watch, can monitor air pollutions in mobile way. It consists of three parts, including a mobile sensor unit, database server and client applications. Mobile sensor unit need to be carried by users to collect pollution data, stamp data with time and location, and then upload to server. Server will record the data and provide an interface to client application. Client application can access data via the interface and visualize pollutants level on a map. The infrastructure of this system has already been build up by James Carrapetta [4], Nikolaus Youdale [5], and Amanda Chow who had worked on this project last year. They indeed make a great contribution to the foundation of this project, and their aim was to set up a basic version of Haze watch system implemented for further experimentation and improvement.

In this year, the project has been divided into several parts for further refinement. This report is focused on the communication within the mobile sensor unit, which is actually between sensor device and Android mobile phone. All the works included in this report is completed together with my partner Junjie, Jiang. The mainly objectives of our thesis are:

- To estimate device location when positioning signal lost in tunnel
- To improve communication method between device and mobile phone.
  - Improve transmission protocol
  - Realization of two-way communication



- To improve upload process

Two programming languages used in our thesis are c for microcontroller programming and Java for Android application on the phone. In the following chapters, overview of the whole project will be introduced. Some problems existed in basic version will be proposed based on analysis of previous system performance. Corresponding solutions of the problems will also be explained in detail, and related test result of these solutions will be demonstrated in this thesis. Lastly, future works need to be done for further development will be suggested.

## 2. Background

### 2.1 Reasons for mobile air pollution monitor

Breath is the most common action that humans have to take every second. A scientific research indicates that adults will breathe about 20000 times per day on average, which approximately 20000 liters air [1]. Therefore, air quality has a close relation with human health. However, air pollution becomes more and more serious in recent decades as a result of society industrialization and explosion of using vehicles. There are two main sources of air pollution in urban area, which is automobiles emission and fuel combustion [2]. These two kinds of pollution source produce a plenty of harmful chemicals in the air, which mainly includes Sulfur dioxide ( $\text{SO}_2$ ), Carbon monoxide (CO), Nitrogen dioxide ( $\text{NO}_2$ ), and Ozone ( $\text{O}_3$ ). Long-time inhaling those air pollutants will cause some fatal diseases, such as asthma, lung cancer, and heart disease. Every year, 3 million people deaths are attributes to air pollution [3]. These diseases are preventable if people be aware of actual air quality around them and change their personal routines.

Moreover, a lot of medical researches have been conducted by professionals to figure out detailed influence of harmful chemicals in the air on human health, which is a critical part for curing those diseases. However, instruments used to monitoring air pollution are usually quite expensive and not convenient for volunteers to carry. Thus, a portable and accurate monitoring system that is capable of monitoring pollution level of particular people will be a valuable data source for their study.

In addition, the air pollution monitoring systems are always hosted by government. Normally, these systems are fixed monitoring stations located around city. The Department of Environment, Climate Change and Water (DECCW) is in charge of monitoring the air pollution in NSW. DECCW is currently running only 14 static

monitoring stations in Sydney as shown in Figure.1. The major pollutants source in Sydney is the extensive use of automobile to make transportation requirements [6]. However, some of these monitoring stations are not located near the highway or main traffic road. Thus, data collected at those stations may lead to an unreliable prediction of the actual pollutants concentration. Furthermore, these 14 static monitoring stations can only monitoring pollutants level in a limited region, and they are situated apart away from each other. Measuring the pollutants level in the region between stations will be a challenged work for them.



Figure .1 Location of monitoring sites in Sydney taken from DECCW website

## 2.2 Similar project

Mobile air pollution monitoring system will be a great progress. Many projects all over the world work on this. For example, Mobile Air Quality Monitoring Network (MAQUMON), which is undertaken by Networked Embedded Systems Lab at ISIS, has assembled many hardware modules, such as several air pollutants sensors, microcontroller, GPS module, USB port, and Bluetooth module, into a single wireless sensor device. Prototype of the device is shown in Figure.2. The on-board GPS module labels collected data with time and location. Then data is



Figure.2 Device prototype of MAQUMON[7]

periodically uploaded to server via Internet by connecting to a laptop or PDA and displayed on Microsoft SensorMap[7].

### 3. Overview of the system

Haze Watch system is a similar system to MAQUMON that is able to monitor air pollution by collecting data in mobile way, which is open to utilize by various group of people, such as individuals, medical research center, and environment department. It includes a portable sensor device that can measure pollutants wherever the user is, and users may view the pollution results either from an android phone interface or some client applications via internet. Figure.2 shows the infrastructure of the system, which had been built up by James Carrapetta [4], Nikolaus Youdale [5], and Amanda Chow who had worked on this project last year.

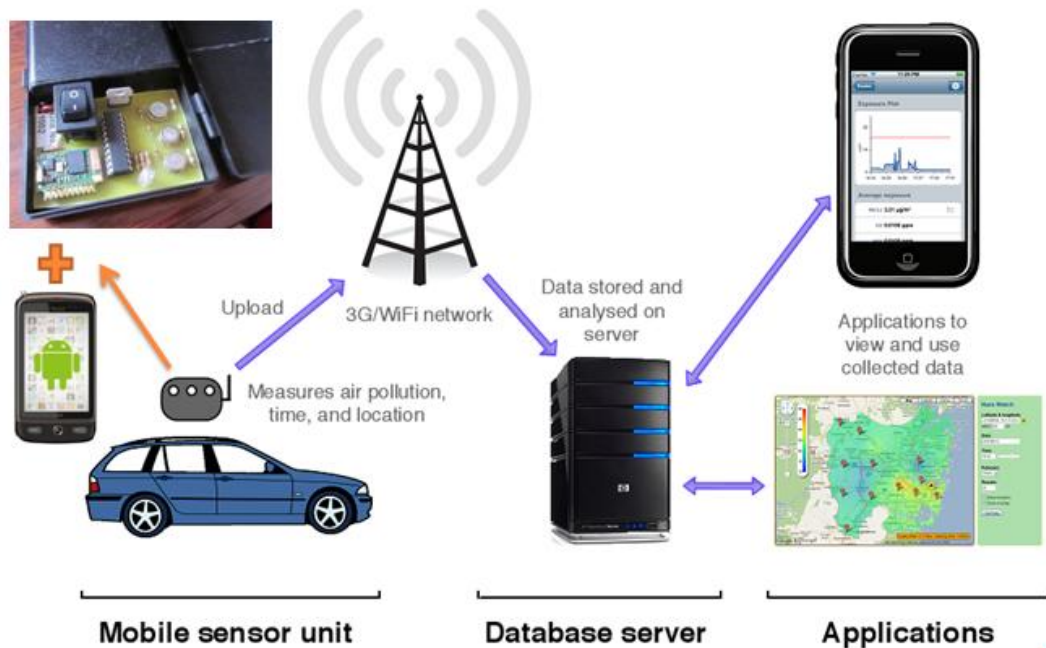


Figure.3 General overview of Haze Watch system  
taken from Nikolaus Youdale thesis[5]

As shown in Figure.3, the Haze Watch System mainly consists of mobile sensor unit, database server, and client application.

### 3.1 Sensor unit

Sensor unit includes a wireless sensor device and an Android phone, as shown in below figures.

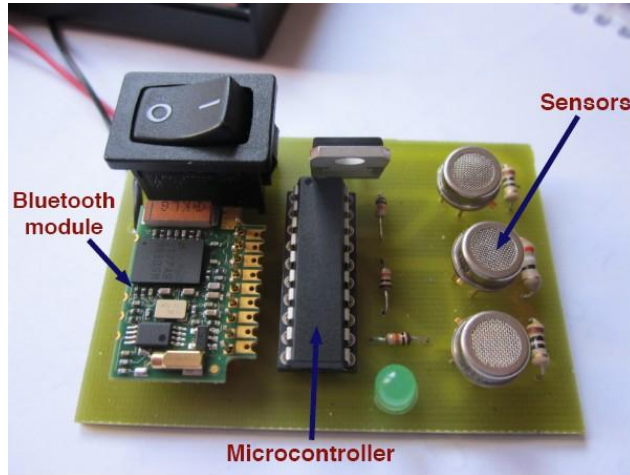


Figure.4 Wireless sensor device [4]

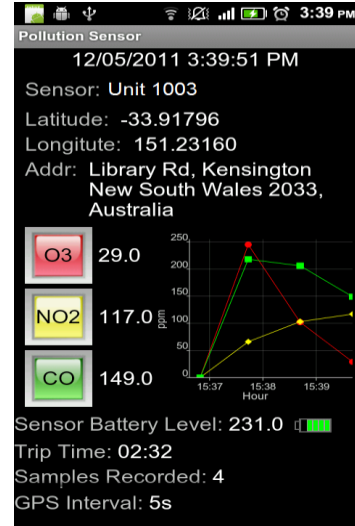


Figure.5 Android application user interface[8]

The sensor device is lower costly than that of MAQUMON because it was not assembled all the required hardware on the sensor board but relied on existing functions of an Android phone. There are three sensors on the device, which can measure carbon monoxide CO, Nitrogen dioxide NO, ozone O<sub>3</sub>, respectively. The microcontroller in the middle is the heart of the board, which is responsible for controlling all the operation manners of hardware components on the device. Specifically, the microcontroller PIC16f690 has 12 Analogue to Digital converts [9] that is able to communicate with sensors. Actually, the data collected from these sensors need to converts to digital values first and assembled in a message. Send it to Bluetooth by writing into the transmitting register of Enhanced Universal Synchronous Asynchronous Receiver and Transmitter (EUSART) module in PIC16f690 [9]. Bluetooth then forward the message to Android phone by following the Radio Frequency Communication (RFCOMM) protocol [10].

Android phone operates a pollution monitoring application installed in it. The application is able to display the collect data on a figure that indicates the current

pollutants level versus time, which is a new user interface that was being developed by Dawei, Lu and Kunxuan Bi who undertaken Android interface of this project in this year. Then Android phone labels data with time and location and upload to server. Android application is written in Java Android ADK.

### 3.2 Database server

Server records all the data in its database and provides an interface for client application accessing recorded data.

### 3.3 Client application

In current stage, the client applications contain a pollution map and personal exposure monitor as seen in Figure.6.



Figure.6 Personal exposure

The pollution map application shows all the data collected points on a map and represent different pollutants levels in terms of different colors, which will give people a directly visualized view of air pollution levels. The personal exposure monitor is an iPhone application. It can track user's location and get pollutants level from server. Based on the obtained information, it is able to calculate level of user exposure to various pollutants versus time and display in a graph.

## 4. Analysis and solutions

### 4.1 Positioning signal lost

As mentioned before, Haze watch system is just built in its basic version in last year, so many areas still remained to be improved. The first major problem suffered by Haze watch system regarding communication area is that the mobile phone fails to get accurate locations if the user carries sensor unit driving through tunnels.

#### 4.1.1 Location request method

In Haze Watch system, Android application running on the mobile phone utilizes two different ways to required locations, which is from GPS and Android's Network Location Provider. The former method depends on GPS satellites, and it provides more accurate location but more likely fail to work indoors because of signal interference result from walls. Satellites signals are too weak to go through thick walls of tunnels. Also, this method will consume more power and return location slowly. Network Location Provider specifies user's location via a cell tower and Wi-Fi signal, which is more quickly and less power expense to determine users' locations [12]. By combining these two request method, Android application will take use of better estimated location according to time and accuracy.

#### 4.1.2 Location signal lost in tunnel

The last year thesis group has conducted an experiment to check the working performance of the whole Haze watch system. This experiment involved driving with the sensor unit to collect air pollutants samples along a loop in Sydney. Experiment route is shown in Figure.7. Every label in the map indicates a recorded point. As described in Figure.7, there are gaps appeared in Sydney Harbour Tunnel, Eastern Distributor, and M5 east, which is because the two location request approaches fail to



work in the tunnel. GPS signal is too weak in tunnels, and Android's Network Location Provider is unable to give an accurate location in tunnels so that the inaccurate locations are filtered by accuracy detection function in Android application. As a result, location value was not updating during this period. The entire readings collect in the tunnel will be assigned to a same location which is the last known point before positioning signal lost.

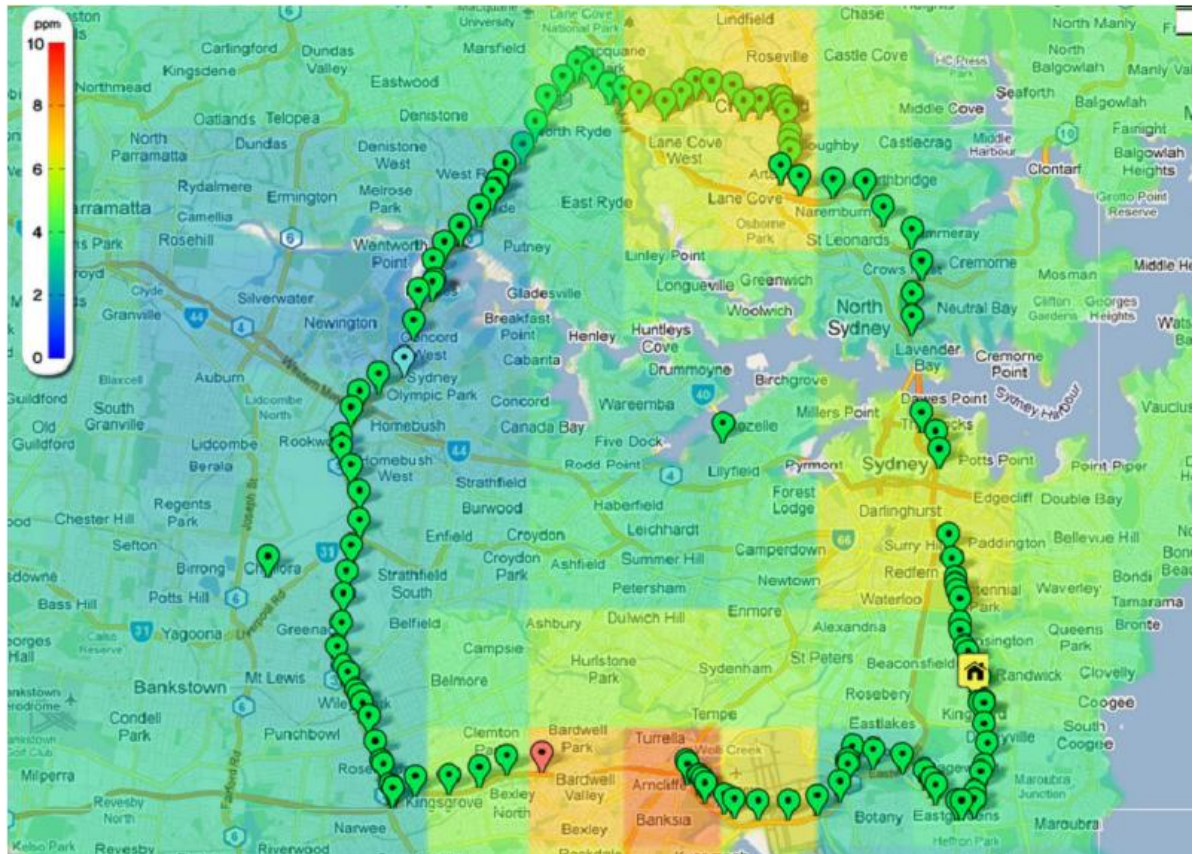


Figure.7 Experiment route taken from James Carrapetta thesisB[4]

## 4.2 Solutions for Positioning signal lost

The general procedures of solving this problem are concluded in the flow chart shown in Figure.8. First, Android application must be capable of knowing when the positioning signal lost so that it can stop using this location to avoid mismatching collected data with an inaccurate location. Then pollution data sampled during this period should be saved until recovering positioning signal. Once find new accurate location, mobile phone are expected to estimate previous sampling locations and



match them with corresponding sampled data. Finally, the application is supposed to upload these bind data group to server as same as commonly dose.

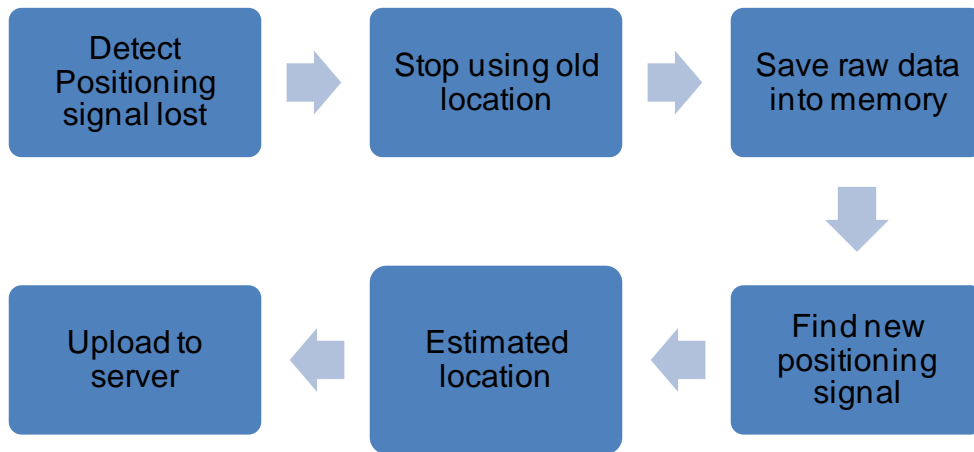


Figure.8 Flow chart of interpolation

#### 4.2.1 Filtering method

An accurate location filter is very important to find out the lost sampling points in tunnels because it determines start point for employing location estimation and time to stops using imprecise locations. In the old version application, a location filter published on Android developer website [12] for maintaining the current best locations is used, which functions as below.

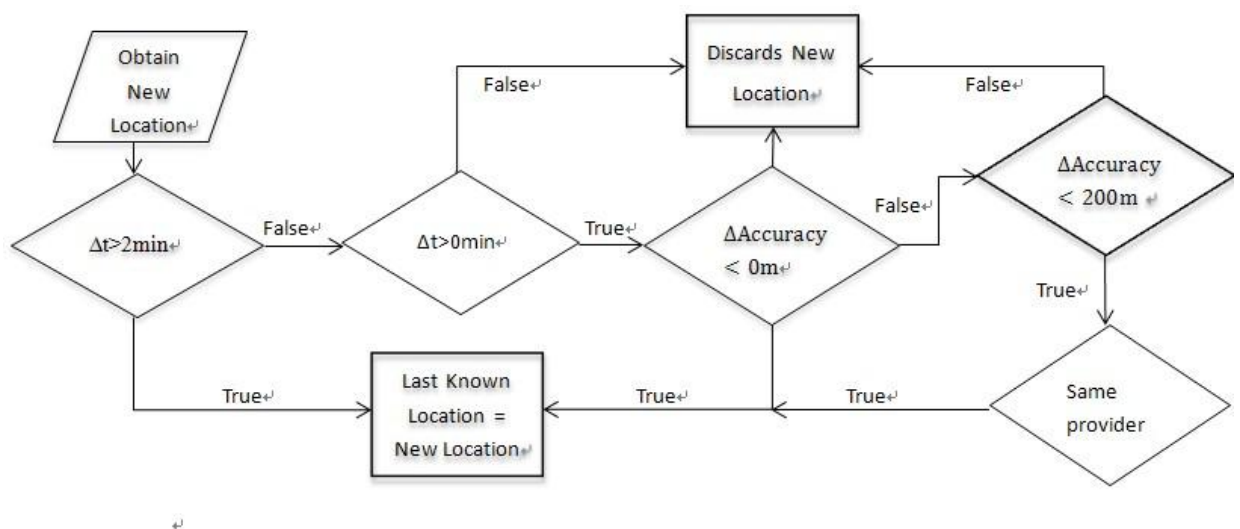


Figure.9 Location filter

When mobile phone obtains a new location, it will first check the time interval between last location and new location whether it is greater than 2 minutes. If it is, the application will adopt the new location since the user is very likely to move. If it is not, the application will check its accuracy difference between new location and previous location. Accuracy is useful value returned together with location to indicate the reliability of returned location. If it is less than 0 meters or less than 200 meters but provide by same location provider, the application will use new location. Otherwise, the new location will be discarded.

Our designed location filter is based on the idea of this one but some modifications. An efficient location filter must be capable of fulfilling two functions: detect the time positioning signal lost and stop using inaccurate location immediately. Android application detects when positioning signal lost by checking content of last known location. If it is null, the application will consider positioning signal lost. Two situations may cause that happening. First, when mobile phone obtains a new location, Android application will check its accuracy. If it is smaller than 100m, new location will be employed. Otherwise, new location will be discarded. The reason for setting accuracy threshold to 100m is that it will not exceed 100 meters inside tunnel. Air quality will not change so much within 100m, and no distance standard indicates that air pollution will distinctly change after how many meters. At the same time, we need make sure located sampling points on map are not too far away from actual points. Normally, the location accuracy can within 12m at outdoor circumstance.

In second situation, when last know location is being requested to use, it will check the time interval between last known location recorded time and current requested time. If it is smaller than 10s, it will be assigned to incoming data. Otherwise, it will be dropped. The normal speed in tunnel is above 40km/h, which means user may move 120m or more after 10s. Any positioning error exceed that is believed unacceptable. The small time threshold is more sensitive to positioning signal lost which can realize that type error faster. Algorithm of new filter is shown in Figure.11.

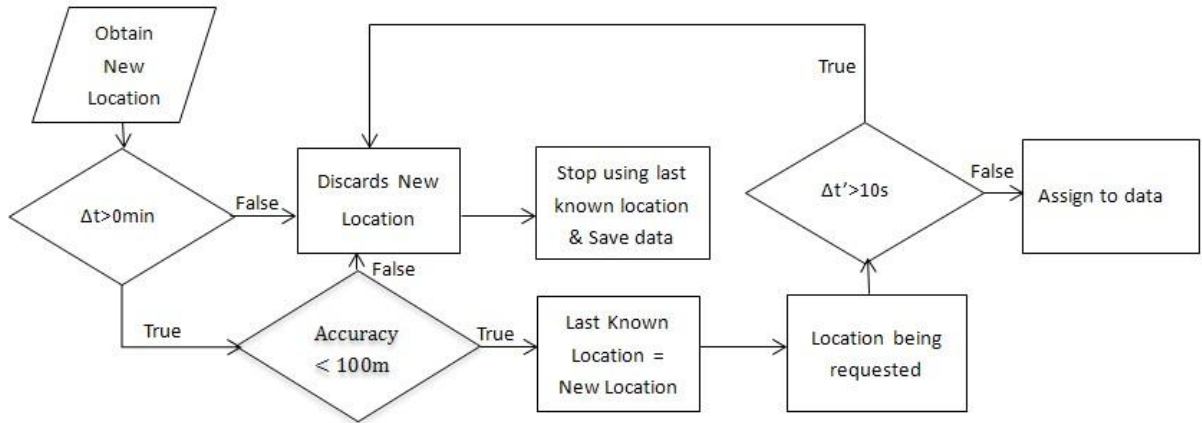


Figure.10 New location filter

Compare to the old version, new filter has more strict limitation in update time interval and accuracy. In addition, it will stop using last known location immediately when positioning signal has lost to avoid assigning entire data sampled inside tunnel to a same incorrect location.

#### 4.2.2 Related solutions for indoor positioning

Indoor positioning is not a new idea, and there are many researches all over the world concentrated on studying this topic. Almost all the GPS based products suffer from this kind of problem when there is insufficient satellite signal coverage. The current available indoor positioning techniques involve inertial navigation, sound-based navigation, electromagnetic wave-based techniques, and map matching technique.

Inertial navigation is a technique that estimates the devices current location in terms of acceleration, velocity, direction and last known location. This kind of techniques normally requires an accelerometer to measure motion, a gyroscope to measure direction. Thus, velocity can be calculated by integrating acceleration over time. [13] However, this method suffers from accumulated error caused in each measurement of acceleration and direction, which will contribute to unacceptable inaccuracy over time. Therefore, inertial navigation needs high quality sensors to give a more reliable acceleration and direction. Moreover, it has to be assisted by other positioning method, such as Wi-Fi, to do a long term calibration.

Sound-based navigation and electromagnetic wave-based techniques both need extra receivers to position one's location. Sound-based navigation measure the distance between transmitter and receiver using ultrasound and determine the position by trilateration [14]. The accuracy of electromagnetic wave-based techniques highly depends on the number of receivers. The more receivers install the more resolution it has.

Map matching technique is initially used to eliminate the errors occurred in the location acquired from positioning system. Map matching is very useful when the future locations are expected on a certain path. Figure.12 describes the block diagram of map matching process. The inputs include positioning data and a digital map. The digital map is not a traditional visualized map but a list of polylines. Map matching algorithm can correct a position to it expected path. [13]



Figure.11 Block diagram of map matching process [13]

Map matching process consists of three phase, including nomination, selection and calculation. In nomination phase, algorithm will find out all the possible polylines depending on the distance from the input positioning point to polylines. In selection phase, algorithm should specify the best polyline based on further information, such as future positioning point. Lastly, an estimated point on the polyline is given instead of input one. [13]

Considering specific problem in Haze watch system, it not actually required to direct one's location but estimate locations where data sampled. Inertial navigation seems not suitable because of its error accumulation nature. Acceleration sensor and

direction sensor in Android mobile phone varies from user to user, so sensor quality cannot be guaranteed. Sound-based navigation and electromagnetic wave-based techniques are also unable to be implemented in tunnel. Map matching technique is the only feasible way in tunnel because it is independent with any extra infrastructure. However, the purpose of map matching is to correct inaccuracy point back to actual path, which is not exactly same as what is expected to be solved in our case. Therefore, non-linear interpolation that will be introduced in section 4.2.4 is inspired by this technique but some modification.

### 4.2.3 Linear interpolation

Linear interpolation is a very simple method that we designed, which can partially solve positioning signal lost problem. The reason for using linear interpolation is because of its simple nature. Linear interpolation considers the route of user moving during the period of positioning signal lost as a straight line. The algorithm will calculate estimated points and insert them between last known location and current location. Suppose the device is moving at constant speed in the tunnel, and pollutants sampling frequency is fixed at 5s in tunnel. As a result, estimated location will equal space distributed for grouping with a data sample. This type of interpolation may works in the short tunnels which can be regarded as a straight line.

When the location filter has detected positioning signal lost, the application will stop using stored old location and save all the incoming data into memory together with a time. Until the next time finding a positioning signal, the program will make a subtraction on the location coordinates of those two points, namely  $LD$ . Since the estimated location are apart from each other at equal distance, the application can divided  $LD$  with total number of data sampled during this period and then get  $\Delta LD$ . Estimated location can be calculated by multiplying  $\Delta LD$  with data index of which data will be integrated with that location. Finally, the entire estimated locations can be obtained for each group of samples. The calculation formula is shown below Related

code refers to Appendix.

$$\frac{LD(\text{longitude difference, latitude difference})}{\text{data NO.}}$$

$$= \Delta LD(\Delta \text{longitude, difference, } \Delta \text{latitude difference})$$

Estimated location

$$= \text{Last known point (longitude, latitude)} + \Delta LD \times \text{data index}$$

#### 4.2.4 Linear interpolation test result

In order to test the performance of linear interpolation function, we have conducted an experiment in General Holmes Drive Tunnel and M5 East. These two tunnels are chosen as their length is significantly different, and they are close to each other. The objective of this experiment is to test how the linear interpolation function works in both short and long tunnels. Figure.12 shows one of test result obtained from pollution website when passing through tunnels.

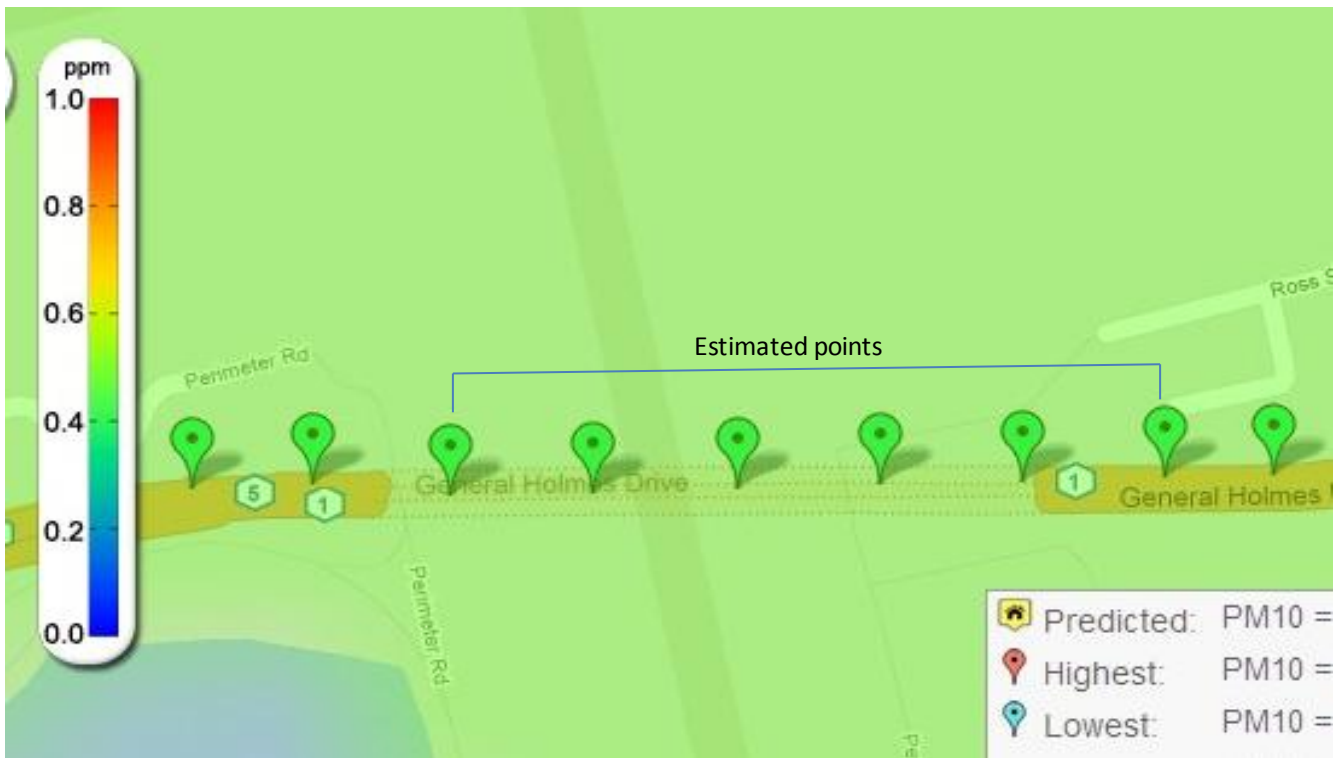


Figure.12 Linear interpolation test result in General Holmes Drive Tunnel



We can see from the above figure, linear interpolation performs well in this short tunnel. The estimated points are distributed along the expected path. Perhaps they are not very exactly consistent with the location where data actual sampled due to assumption of constant moving velocity, but small errors are acceptable because air quality inside such a short tunnel will not vary significantly.

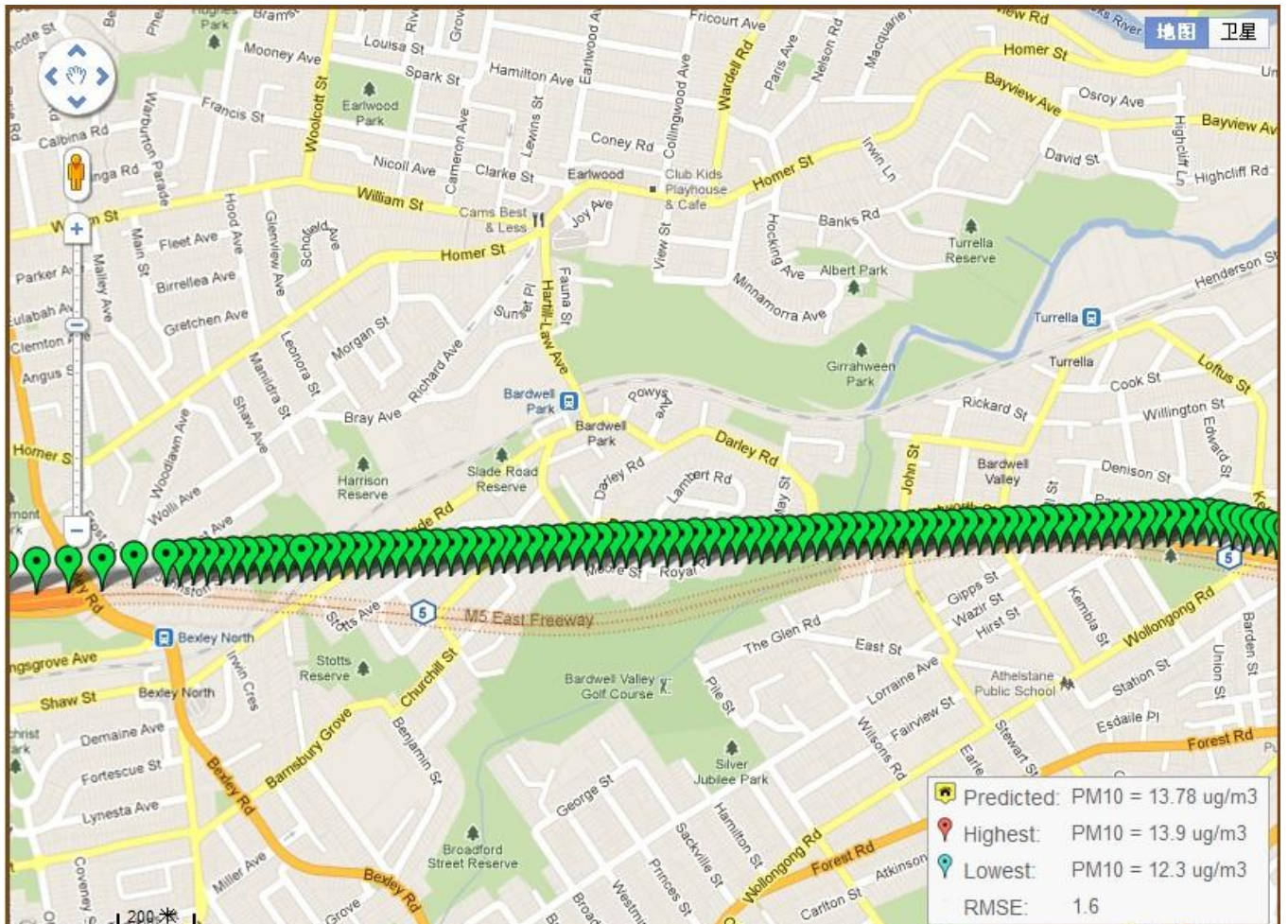


Figure.13 Linear interpolation test result in M5 East

Figure.13 shows another test result of linear interpolation in M5 East. The dash line on the map indicates the tunnel. Currently, M5 East is the longest tunnel in Sydney which is approximately 4000m [15]. We can see that linear interpolation fail to function inside M5 East Tunnel because this tunnel is too long to consider it as a linear path. Therefore, linear interpolation works only when the being interpolated route is sufficient short.

### 4.2.5 Non-linear interpolation

To solve the problem that is subjected by linear interpolation, non-linear interpolation is designed based on the map matching techniques introduced in section 4.2.2. A critical character of map matching is that it is very suitable for estimated point being expected on a certain path, which quite satisfies of what interpolation is required.

#### Digital map

Actually, non-linear interpolation is a transformation of map matching technique. Since the nonlinear interpolation processes after new location founded, the inputs of non-linear interpolation algorithm are last known location, current location, and digital map. The digital map consists of five long tunnels in Sydney, including M5 East, Cross City Tunnel, Eastern Distributor, Sydney Harbour Tunnel, and Lane Cove Tunnel. Let's take M5 East as an example to see how the digital map is constructed.

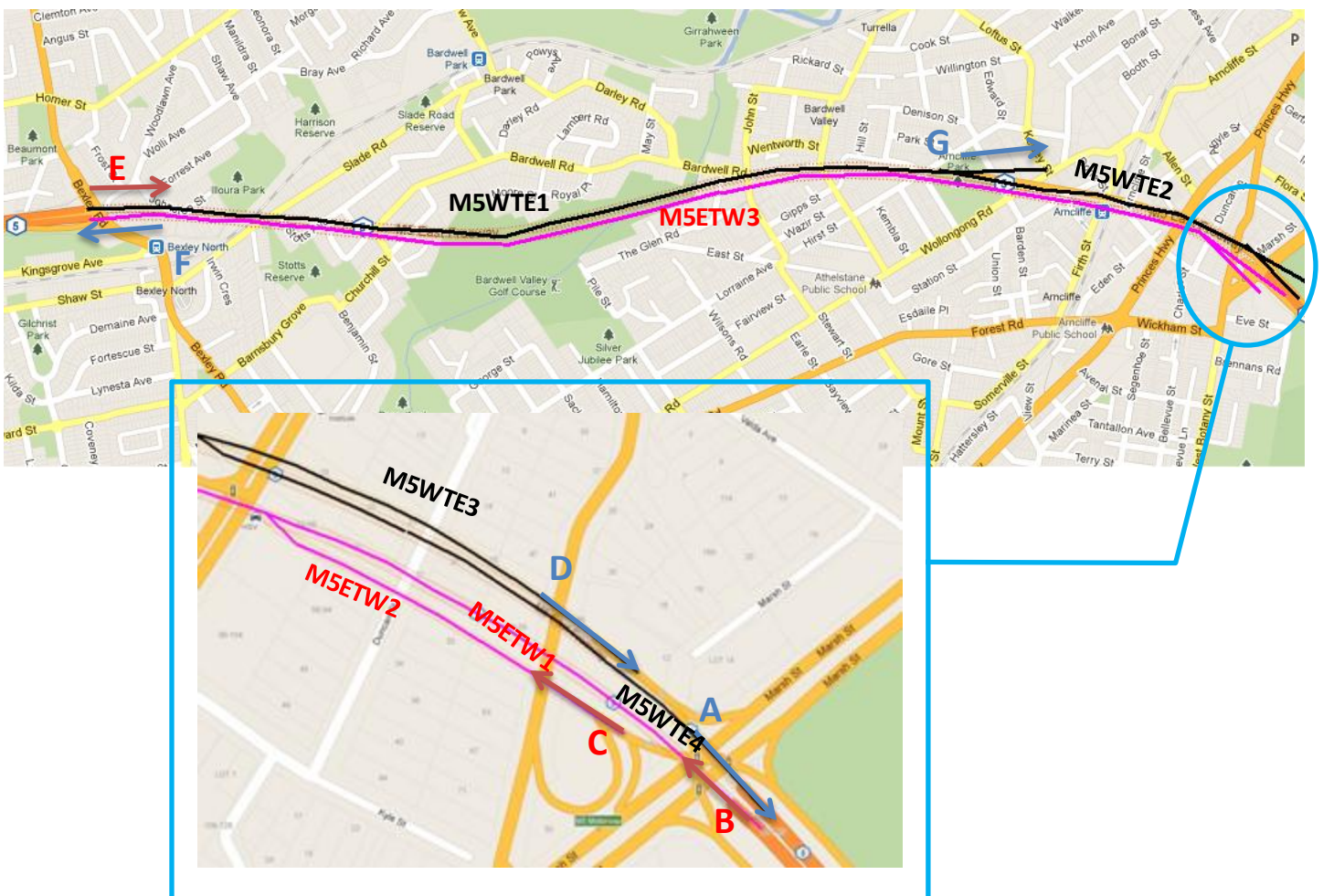


Figure.14 M5 East



We first find out all the entrance locations of M5, namely E, B, and C, which is labeled in Figure.14 together direction arrows. Next, all the potential exit locations should be found for each entrance. Every possible path is divided into small parts. The criteria for dividing is wherever there is a branch(es). For example, the black path in Figure.14 is divided into 4 parts, including M5WTE1, M5WTE2, M5WTE3, and M5WTE4. The table below shows the entrances and possible exit of M5 East. The last column shows the user need to go through which parts from an entrance to an exit. Therefore, the possible routes can be obtained by application according to combination of any route sections shown in Table.1. For example, driving from entrance E to exit A will go through section M5WTE1, M5WTE2, andM5WTE4.

Entrance Number	Entrances	Exit	Exit Number	Routes
0	E	G	0	M5WTE1
		D	1	M5WTE1+M5WTE2+M5WTE3
		A	2	M5WTE1+M5WTE2+M5WTE4
1	B	F	/	M5ETW1+M5ETW3
2	C	F		M5ETW1+M5ETW3

Table.1 M5 East digital map part 1

After that, the mobile phone should be able to know the shape and driving direction of each part so that it can assign location to data along with an expected route. We describe each section by marking location points every 40 meters in terms of their location coordinates in each section. Table.2 below shows sample of route section description.

Section name	M5WTE2
1	-33.934953,151.142411
2	-33.935006,151.142837
3	-33.935064,151.143264
4	-33.935138,151.14369
5	-33.935189,151.144122
6	-33.935251,151.144554
7	-33.935318,151.14498

8	-33.935382,151.145407
9	-33.935449,151.145841
10	-33.935512,151.146273
11	-33.935567,151.1467
12	-33.935629,151.147126
13	-33.935698,151.147553
14	-33.935765,151.147982
15	-33.93587,151.148398
16	-33.93597,151.148821
17	-33.93603,151.149248
18	-33.936063,151.149648

Table.2 Sample of route section description

According to the procedures described above, the digital map of five main tunnels can be obtained as summarized in Table.3. Summary of marking points inside each section can be referred to Java code in Appendix.

Entrance Number	Entrances	Exit	Exit Number	Routes	Tunnel name
0	E	G	0	M5WTE1	M5
		D	1	M5WTE1+M5WTE2+M5WTE3	
		A	2	M5WTE1+M5WTE2+M5WTE4	
1	B	F		M5ETW1+M5ETW3	Cross City & Eastern Distributor
2	C	F		M5ETW1+M5ETW3	
3	H	Q		H to Q =HQ1 +Q2	
4	I	Q		I to Q =IQ1 + Q2	
5	J	R	0	J to R = JX1 +JX2+NR0+NR2+NR3	
		X	1	J to X =JX1+JX2+JX3	
6	K	R	0	K to R=KX1+JX2+NR0+NR2+NR3	
		X	1	K to R=KX1++JX2+JX3	
7	L	U	0	L to U=L1+LU2	
		O	1	L to O=L1+LU2+UO1+UO2	
		P	2	L to P=L1+LU2+UO1+UO2+OP	
		W	3	L to W=L1+LW2+LW3	
8	M	W		M to W=MW1+LW3	
9	N	R		N to R=NR1+NR2+NR3	
10	Y	O	0	Y to O=Y1+YO2+UO2	
		P	1	Y to P=Y1+YO2+UO2+OP	
		V	2	Y to V=Y1+YV2	
11	Z	R	0	Z to R=Z1+Z3+NR3	
		S	1	Z to S=Z1+Z2+ZS3	
		T	2	Z to T= Z1+Z2+ZT3	

12	AA	DD		AADD	
13	BB	CC		BBCC	
14	EE	II		EEII	Sydney Harbour
15	FF	HH		FFHH1+FFHH2	
16	GG	HH		GG1+FFHH2	
17	JJ	OO	0	JJNN1+OO2	Lane Cove
		NN	1	JJNN1+JJNN2	
18	KK	M		KKMM1+KKMM2	
		M			
19	LL	M		LL1+KKMM2	
		M			

Table.3 Summary of digital map of five tunnels

The digital maps will be stored in Android mobile phone as a database to find out correct location when positioning signal had lost.

### Non-linear interpolation algorithm

After completing digital map, non-linear interpolation process can be implemented, which also includes three phrases, involving nomination, selection and calculation. In nomination phase, the Android application compares the last known location to all the entrances of five tunnels and finds the closest two entrances depending on the distance. Some entrances are close to each other, so choosing two closest entrances is to in case incorrect detection. Since the last know location and entrance locations are both represented in Longitude and latitude, so it must be convert to distance for comparison. Haversine formula is applied to do the conversion, which is a important equation for calculating the distance between two pints on the sphere but represented in longitude and latitude.

In selection phase, the application will first compare distance between their possible exit locations with current known location and choose a best suitable path based on the minimum sum distance of entrance offset and exit offset. Once the application chooses the optimal route, it will move to calculation phase in which each data collected in tunnel will be assigned a pre-stored location on that path. Suppose that the tunnel length is 1, n is the number of collected data. K is the number of pre-stored

marking points of sections involved in the optimal path, and  $m$  is the data sequence number. Pre-stored point number  $p$  that should be assigned to data  $m$  is calculated by following equation.

$$p \approx \frac{\frac{1}{n+1} \times m}{\frac{1}{K+1}},$$

#### 4.2.6 Non-linear interpolation test result

Non-linear interpolation have been tested at Eastern Distributor Tunnel as it crosses with Cross City Tunnel and many entrances and exits will be involved during testing, which forms the most complex digital map among the five tunnels. The first purpose of this experiment is to test whether this algorithm can select the actual moving route among a plenty of paths. The second purpose is to test whether the pre-stored marking points can assign to collected data in tunnel and display on pollution map. In this experiment, we carried sensor unit entering the tunnel from entrance L and going out at exit O (Table.3), and we also use a laptop to monitoring operating process. We have seen from the monitoring screen that non-linear interpolation selected correct entrance L but exit P. As these two exits are very close to each other (Figure.15), and positioning signal cannot be found immediately out from the tunnel, so the program may make mistake in selecting optimal path. Fortunately, most of the points are located exactly at the actual route but the exit point, which may be acceptable in current stage. Every the data collected in tunnel is allocated a location as the total number of sampled data matched the number of points inside tunnel.

Another test result of non-linear interpolation is shown in Figure.16. This experiment route has three tunnels involved, including Eastern Distributor Tunnel, Harbour Tunnel, and Lane Cove Tunnel. In this experiment, all the expected paths are correctly selected.

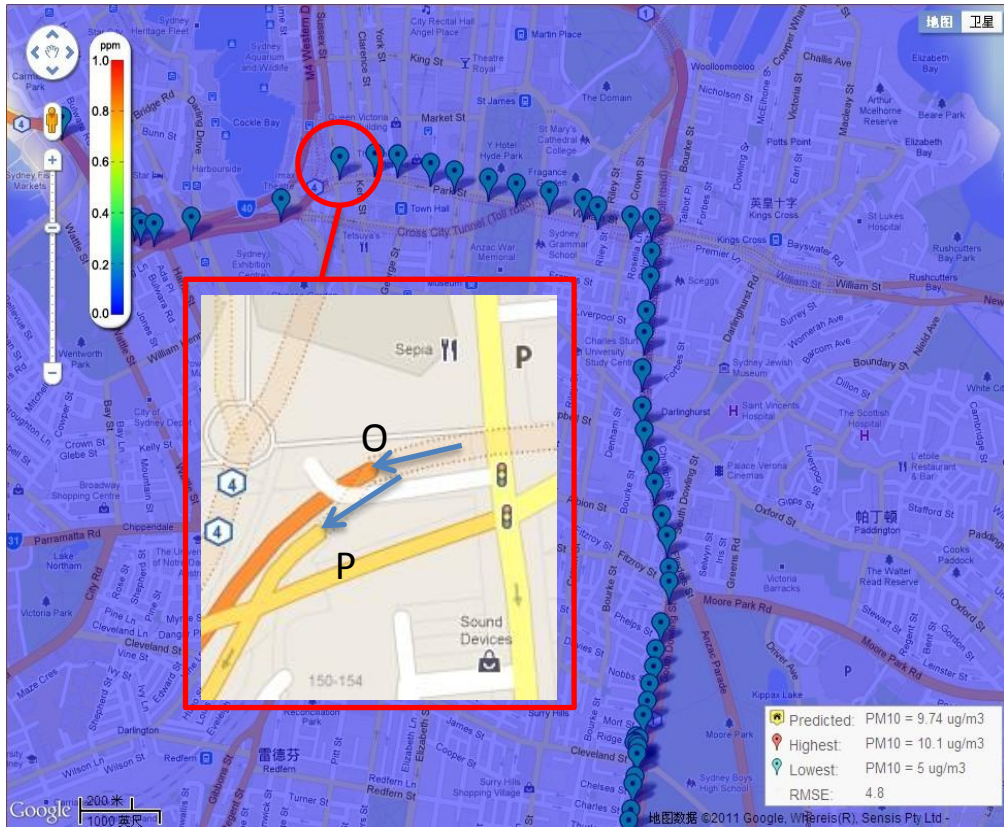


Figure.15 Non-linear interpolation test result

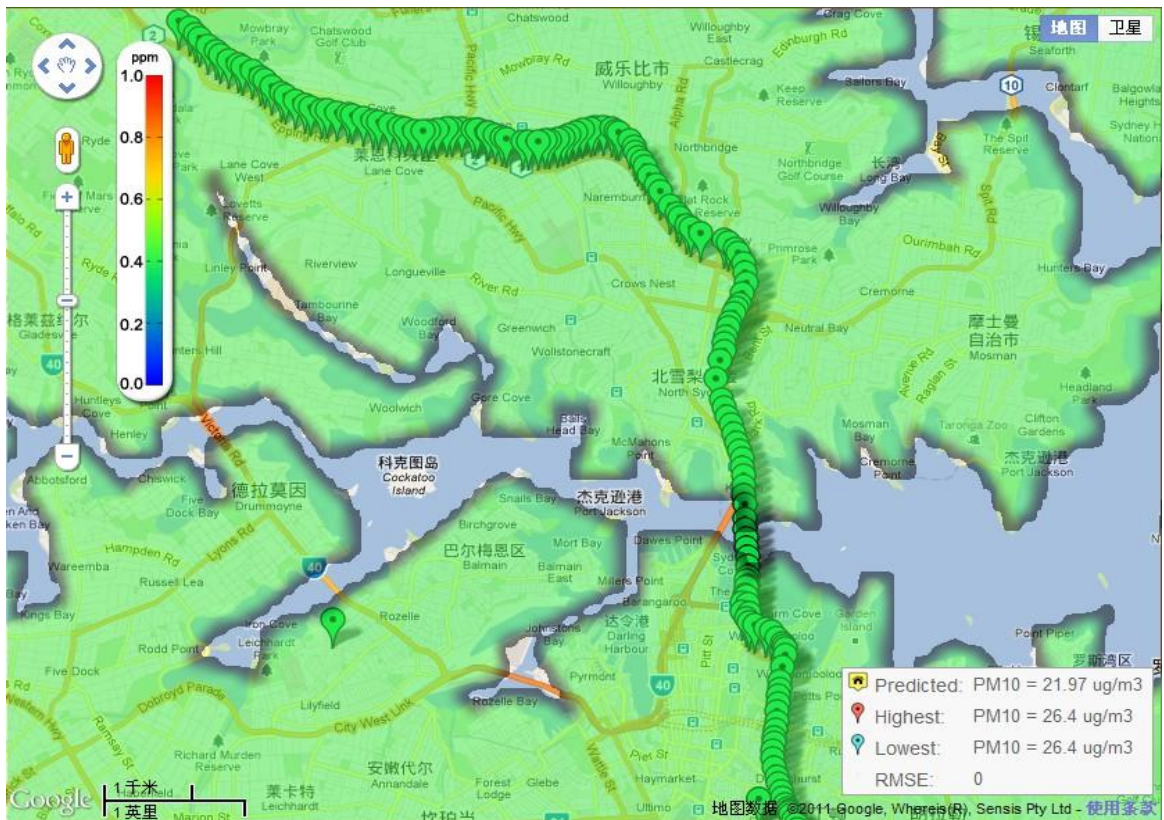


Figure.16 Non-linear interpolation test result

## 4.2.7 Integrated interpolation

The two types of interpolation both have advantages and disadvantages, but they are just complementary to each other. Therefore, we decided to combine these two functions together. Linear-interpolation is suitable for short tunnels, while non-linear interpolation must depend on digital maps. Linear interpolation is very limited, which can only apply when the tunnel is short and straight. Otherwise, the estimated points cannot be located exactly on expected path. Although non-linear interpolation can guarantee the output location assigned to each data exactly on the actual route, we only construct digital maps for five long tunnels, which means other short tunnels have to employ linear interpolation.

## 4.3 Transmission protocol

A consistent protocol is a significant prerequisite to guarantee a successful communication between device and Android phone. Therefore, the structure of the protocol format is necessary to be carefully considered.

### 4.3.1 Overview of protocol version 1

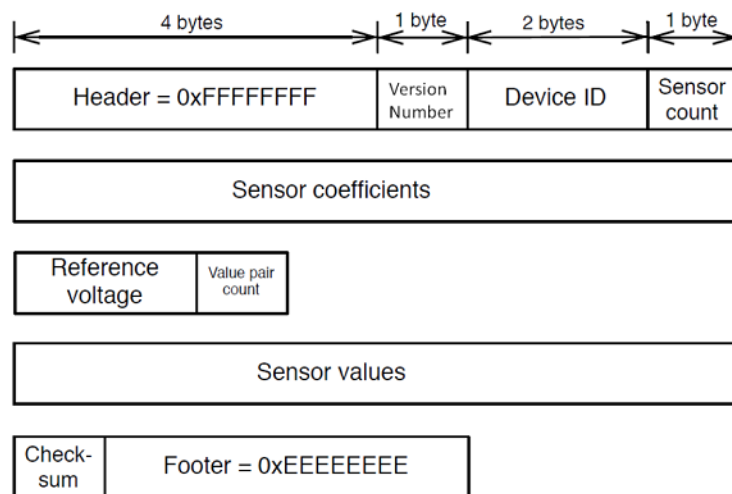


Figure .17 Transmission protocol between device and Android phone  
taken from James Carrapetta thesisB[4]



Figure.17 describes an agreed protocol that designed by last group people. This message format includes all the necessary information that is needed to send from device to Android phone.

Header field and footer field each contains a 4 bytes constant that is used to inform Android phone beginning and end of the message. Version number field is a 1byte field that used to distinguish different version of the format, which means Android phone can understand what format that device used as the format may be developed in someday. The version number is 1 for this format. Device ID field describe which device the phone talks to. Sensor count indicates the number of sensors on the board, which is 3 in current stage. Sensor coefficients is coefficients of a function that used to convert a voltage level to a ppm (part per million) value which is a standard unit to express pollutants level. Sensor coefficients vary from sensor to sensor, so every sensor need to be calibrated independently. Reference voltage is a base number that is used by Android phone to calculate the actual voltage values. Since sensor readings send by device are non-understandable numbers that lie in 0 to 255 which is 8 bits number read from ADC, so these numbers have to be converted to a reasonable voltage in terms of reference voltage (3.3V). The Sensor values contain 30 groups of reading including a sensor ID and sensor value, as shown in Figure.18, which means 10 readings from each sensor.

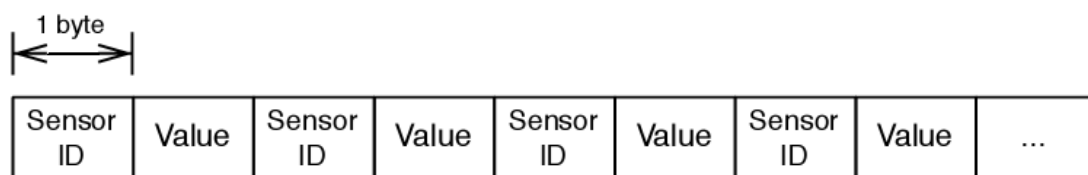


Figure.18 Sensor values format taken from Nikolaus Youdale thesisB[5]

### 4.3.2 Transmission drawbacks

After collecting readings from each sensor, device assembles all the fields by following the order of message format and sends them out byte by byte, because the transmitting register of the microcontroller is only 8 bits. Microcontroller sends data

to Bluetooth in Asynchronous mode, which means transmitter and receiver are unnecessary to synchronize their clock. Bluetooth will start to send data when it received bytes stream from microcontroller.

However, whenever the sensor readings send out, device have to send other control fields too by following version 1 protocol. Apart from sensor values, the rest of fields will be constant as long as Android phone talks to same device. Repeatedly transmitting those redundant fields lead to a waste of both power and time. It will takes microcontroller roughly 15s to transmit the entire message out. The Bluetooth module used on device is ARF32 who required at least 40mA current when transmitting data [10]. Obviously, a plenty of power consumed on transmitting control fields.

On the receiver, Android phone update some information first according to those control fields and extract 30 sensor readings. These 30 readings consist of 10 readings from each sensor. Android phone will take an average on those 10 readings, respectively. Then the group of three average values is tagged with time and location. However, both time and location are not accurate for most of points because they are actually collected at different locations if device is moving as well as different time. Message sending period which is the time between two consecutive messages sending out is approximately 30s for version 1 protocol. The maximum location error, which indicates the distance between the first sampled point and tagged location, will be 500m if device were carried by a car moving at 50km/h. Moreover, it is unreasonable to take average on those ten readings that are actually collected at different locations. A valuable average is supposed to be taken on the readings that collected at same location. Different location average value is meaningless in practice.

### **4.3.3 Version 2 protocol**

Version 2 protocol is the new protocol we have changed to avoid repeatedly sending



control fields. The structure of transmitting message in version 1 protocol has been separated into two messages, and we call them control message and data message, which are shown in Figure.19. and Figure.20.

### Control message

<b>Header</b> 4 bytes	<b>Version</b> 1 bytes	<b>Type</b> 1byte	<b>Device</b> ID 2 bytes	<b>Sensor</b> Count 1 byte	<b>Last</b> calibration date 3 bytes	<b>Next</b> calibration date 3 bytes	<b>Reference</b> voltage 2 bytes
<b>Sensor ID</b> 3 bytes		<b>Sensor coefficients</b> 18 bytes		<b>Checksum</b> 1 byte	<b>Footer</b> 4 bytes		

Figure.19 Version 2 protocol control message format

### Data message

<b>Header</b> 4 bytes	<b>Version</b> 1 byte	<b>Type</b> 1 byte	<b>Battery</b> level 2 bytes	<b>Sensor</b> values 9 bytes	<b>Checksum</b> 1 byte	<b>Footer</b> 4 byte
--------------------------	--------------------------	-----------------------	------------------------------------	------------------------------------	---------------------------	-------------------------

Figure.20 Version 2 protocol data message format

Control message contains all the information that will not alter when Android phone connects to same device, while data message including all valuable data that is constantly changing in every message.

Besides separating the message, some new fields (blue bold) are also added. Type field in both two messages informs Android phone which message it receives. Last calibration date and next calibration date contained in control message will display on Android phone to remind users doing sensor calibration since sensors have to be calibrated periodically, normally every 6 months. In the data message, battery level fields used to notice user battery status of device.

An important change in sensor values field is that it contains one group of readings from each sensor rather than ten groups. This change consideration is to reduce the message sending period; thereby diminishing location error. As a result, every group of samples will be labeled with time and location instead of doing average. Another correction in this field is to improve the precision of readings from ADC in microcontroller. In old version protocol, the microcontroller only read higher 8 bit of ADC, but the actual length of ADC is 10 bits. As a result, the precision is limited to 8 bits, which is not a wise decision. Currently, total 10 bits are applied, and thus output digital levels vary from 0-1024 rather than 0-255, which means the reference voltage is divided into 1024 precise.

## 4.4 Two-way communication

Only separating the transmitting message in two independent parts cannot completely avoid repeatedly sending control message, while it still has to be sent periodically rather than being sent once at the beginning of the connection in that case. The reason is that communication between device and Android phone is one way, so device is impossible to confirm whether the connection sets up successfully or not. To completely avoid redundant transmission, two-way communication between device and Android phone has to be realized. Apart from previous purpose, two-way communication can also increase connection guarantee between device and mobile phone, and moreover mobile phone can control device to fulfill some functions, such as sampling data based on moved distance.

### 4.4.1 Circuit build for two-way communication

Since the old version sensor board can only support one-way communication, the circuit must be modified. Circuit schematics refer to Appendix, which is not modified too much but added a receiver channel between Bluetooth and Microcontroller. PIC16F690 chip receives data also through its EUSART module, which is same as transmitting process. The receiving channel is built up by connecting Bluetooth's

transmitter pin Uart\_tx, to microcontroller receiver pin RX/DT.

After completing physical connection, microcontroller is required to be programmed by c language (Refer to Appendix) to complete the reception process. As EUSART receiver is set to operate in asynchronous mode, so there is no need to synchronize the transmitter and receiver. Bluetooth will automatically send received data from wireless interface to microcontroller's RX/DT pin through its transmitter pin. RX/DT pin is actually connected to a Receive Shift Register (RSR). When the 8 bits character has been shifted in, RSR will automatically save it to a two-character capacity FIFO buffer for receiver register to serve. Once there is a character inside the buffer, a flag bit of one of the status register will set to 1, and thus we programed the microcontroller to read receiver register by detecting that flag bit. Figure.21 shows the real circuit we have built on breadboard for experimental purpose.

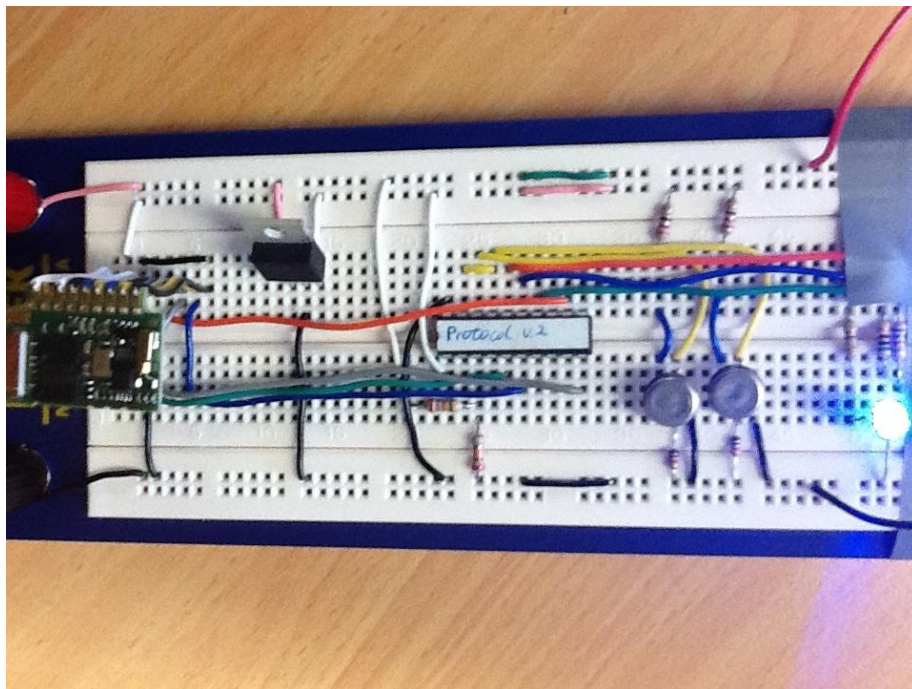


Figure.21 Real circuit for two way communication [17]

After many times experiments, we have successfully received data from Bluetooth transmitting pin. Figure.22 shows the first time received data from receiving pin of microcontroller. It is a test word represented 11001011, which is observed on

oscilloscope. We can see that receiver pin receive least significant bit (LSB) first. Time scale of oscilloscope is 500us, while received 10 bits data (including a start bit and a stop bit) account for about 2 grids, which is 1ms. Consequently, received data is consistent with the pre-set baud rate 9600 bps.

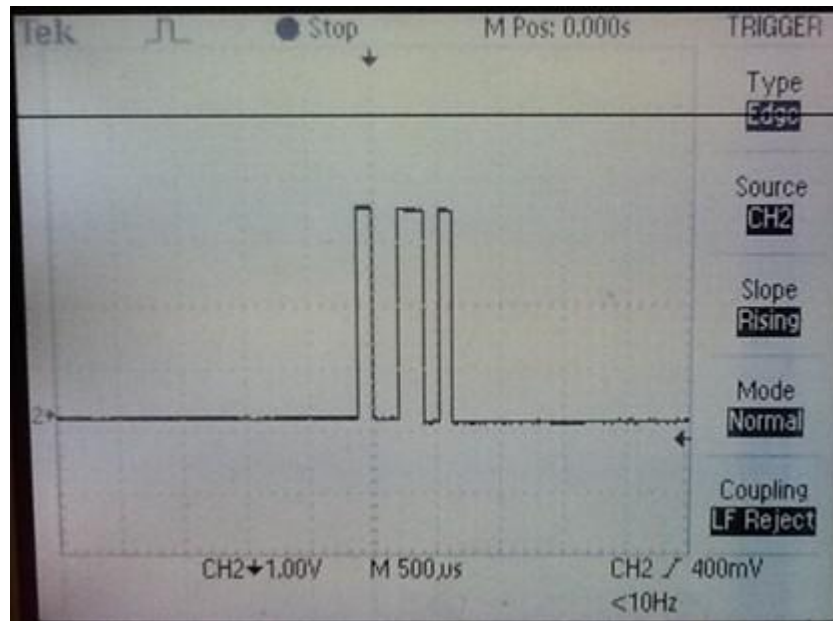


Figure.22 Received data sequence from receiving pin

#### 4.4.2 New functions

The success of two-way communication is a great improvement of communication between device and mobile phone as it makes device smarter. In other words, sensor board can follow the mobile phone's instruction to work rather than doing all the things blindly and mechanically. Two new functions have been achieved based on two-way communication.

##### Connection oriented communication

As mentioned in section 4.3.2, sensor device will blindly send sampled data out when it turns on regardless of connection, which is very power consuming. We have improved communication method between device and mobile phone to connection-oriented, which means device send message out based on request from

mobile phone. There are five modes defined for device in the two-way communication process.

- **Standby mode:** Bluetooth has connection to mobile phone, but mobile phone has not sent any request message. Therefore, there is no information exchange in this mode. This mode is indicated by red light of triple LED on breadboard.
- **Sleep mode:** Special type of standby mode but without Bluetooth connection with mobile phone.
- **Control message mode:** Sending control message out, which is indicated by green light.
- **Data message mode:** Sending data message out, this is indicated by blue light.
- **Cool down mode:** 3 second cool down time after every data message, which is indicated by an off light.

The new designed communication procedures will be implemented as follows.

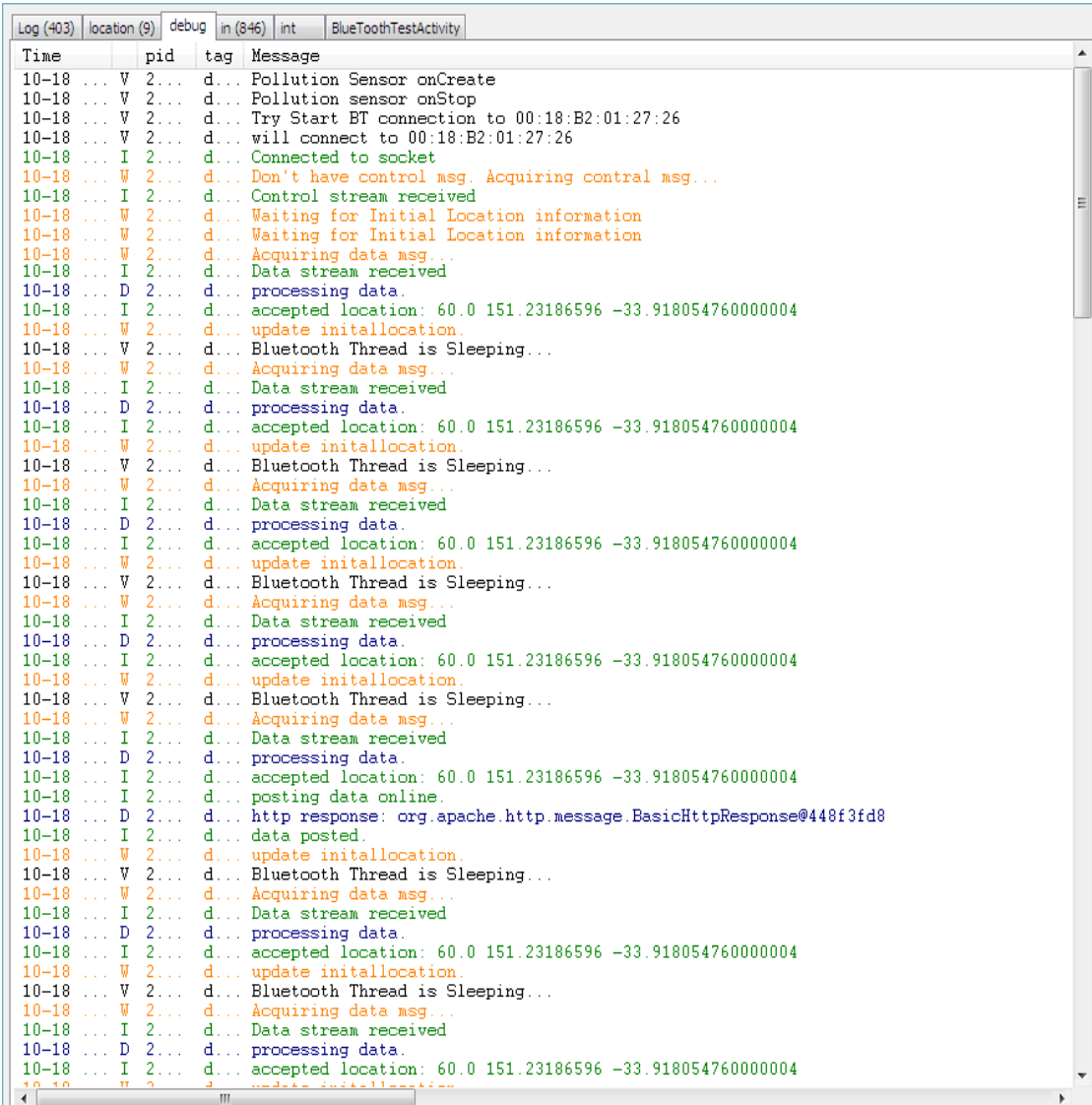
- Device will stay at sleep mode first when it turns on.
- Once it has set up connection with mobile phone, sensor board will move to standby mode.
- Mobile phone will automatically send control request to device by sending a control word 1.
- After receiving control message, mobile phone will only request data message by sending data request. Data request message is described by control word 4. The reason for choosing number 4 is that connection between two Bluetooth devices follows its own protocol in which header and footer are 2 and 3. To avoid misleading device, we use 4 to request data message.
- Three seconds cool down time after every data message is to prevent device from frequently sampling air pollution. As we have already significantly reduced device's transmitting delay by modifying protocol and decreasing operating delay time, device only needs 200ms from sampling air pollution to send message out, whereas it takes old version device about 15s to fulfill the same thing. This can be proved by observing blue light lasting time on two different version devices since blue light indicates data message sending.

## Displacement-based sampling mode

Displacement-based sampling mode is another realized new function which benefits from two-way communication. Under this mode, mobile phone can request location every equal distance and thereby sending data message request to device whenever there is a location change. Therefore, device will collect air pollution data at equally spaced points. The major benefit of this mode is that it can avoid that so many points concentrated in one location or small area when user moves at low speed or static, like waiting traffic lights.

### 4.4.3 Test results

#### Connection oriented communication



```
Log (403) location (9) debug in (846) int BluetoothTestActivity
Time pid tag Message
10-18 ... V 2... d... Pollution Sensor onCreate
10-18 ... V 2... d... Pollution sensor onStop
10-18 ... V 2... d... Try Start BT connection to 00:18:B2:01:27:26
10-18 ... V 2... d... will connect to 00:18:B2:01:27:26
10-18 ... I 2... d... Connected to socket
10-18 ... W 2... d... Don't have control msg. Acquiring control msg...
10-18 ... I 2... d... Control stream received
10-18 ... W 2... d... Waiting for Initial Location information
10-18 ... W 2... d... Waiting for Initial Location information
10-18 ... W 2... d... Acquiring data msg...
10-18 ... I 2... d... Data stream received
10-18 ... D 2... d... processing data.
10-18 ... I 2... d... accepted location: 60.0 151.23186596 -33.918054760000004
10-18 ... W 2... d... update initiallocation.
10-18 ... V 2... d... Bluetooth Thread is Sleeping...
10-18 ... W 2... d... Acquiring data msg...
10-18 ... I 2... d... Data stream received
10-18 ... D 2... d... processing data.
10-18 ... I 2... d... accepted location: 60.0 151.23186596 -33.918054760000004
10-18 ... W 2... d... update initiallocation.
10-18 ... V 2... d... Bluetooth Thread is Sleeping...
10-18 ... W 2... d... Acquiring data msg...
10-18 ... I 2... d... Data stream received
10-18 ... D 2... d... processing data.
10-18 ... I 2... d... accepted location: 60.0 151.23186596 -33.918054760000004
10-18 ... W 2... d... update initiallocation.
10-18 ... V 2... d... Bluetooth Thread is Sleeping...
10-18 ... W 2... d... Acquiring data msg...
10-18 ... I 2... d... Data stream received
10-18 ... D 2... d... processing data.
10-18 ... I 2... d... accepted location: 60.0 151.23186596 -33.918054760000004
10-18 ... W 2... d... update initiallocation.
10-18 ... V 2... d... Bluetooth Thread is Sleeping...
10-18 ... W 2... d... Acquiring data msg...
10-18 ... I 2... d... Data stream received
10-18 ... D 2... d... processing data.
10-18 ... I 2... d... accepted location: 60.0 151.23186596 -33.918054760000004
10-18 ... I 2... d... posting data online.
10-18 ... D 2... d... http response: org.apache.http.message.BasicHttpResponse@448f3fd8
10-18 ... I 2... d... data posted.
10-18 ... W 2... d... update initiallocation.
10-18 ... V 2... d... Bluetooth Thread is Sleeping...
10-18 ... W 2... d... Acquiring data msg...
10-18 ... I 2... d... Data stream received
10-18 ... D 2... d... processing data.
10-18 ... I 2... d... accepted location: 60.0 151.23186596 -33.918054760000004
10-18 ... W 2... d... update initiallocation.
10-18 ... V 2... d... Bluetooth Thread is Sleeping...
10-18 ... W 2... d... Acquiring data msg...
10-18 ... I 2... d... Data stream received
10-18 ... D 2... d... processing data.
10-18 ... I 2... d... accepted location: 60.0 151.23186596 -33.918054760000004
10-18 ... W 2... d... update initiallocation.
```

Figure.23 Monitoring result of connection-oriented communication [17]

For testing two-way communication procedure, we connected Android phone with our laptop and ran application in the phone, using eclipse to monitor message exchange process. Monitoring result is shown in Figure.23, which is consistent with the same procedures described in section 4.4.2. Mobile phone request control message first as shown in the first red line of Figure.23. After that control message received, it only request data message. Bluetooth sleeping line represents 3 second “cool down” time.

### Displacement-based sampling mode

The test result regarding displacement-based sampling mode is shown in Figure.24.

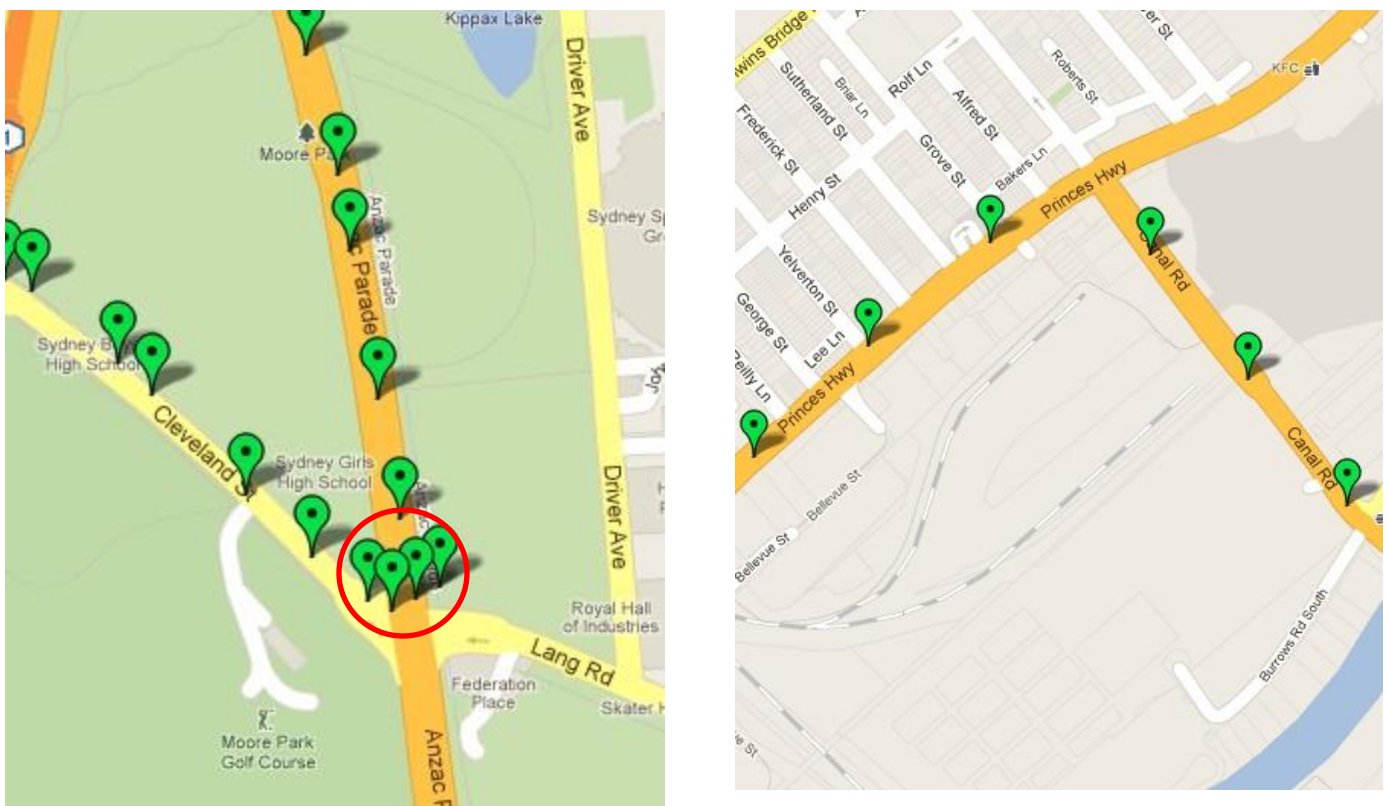


Figure.24 Comparison between normal sampling mode and Distance-based sampling mode

The distance between two location requests can be chosen as any value, and we choose 200m in this experiment just for test purpose. The left graph in Figure.24 shows distribution of sampling locations under normal mode, while the right one shows that in displacement-based mode. We have waited a traffic light at intersection shown in both maps. Device keep sampling air pollution data when we waited at intersection in left graph, which results in those highly concentrated points.



Displacement-based sampling mode can completely avoid that situation happened because it samples data according to location change.

## 4.5 Upload Method

### 4.5.1 Buffer size

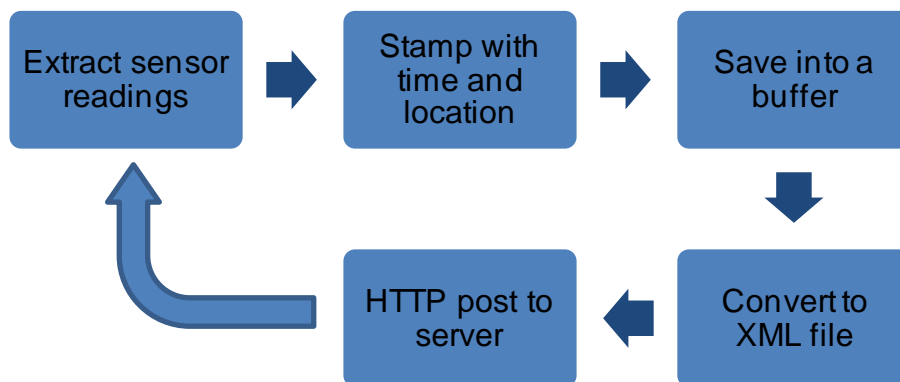


Figure.25 Message process procedure

Message process procedure designed by Nikolaus Y. is shown in Figure.25. As we have modified the transmission protocol, the average function was removed. From the flow chart, Android application will upload a record containing time, location, and three readings immediately to server whenever it writes into buffer. Since the new protocol is applied, samples sending period increased to 5s/message, which means Android phone has to upload one XML file every 5s. Such a frequent uploading is unnecessary and power consuming. Therefore, we have increased the buffer size to contain 20 records, and then the phone will upload to server once after 20 records saved. Each record occupies approximately 120 bytes, including pollutants name (18 bytes), sensor values (24 bytes), date (24 byte), and location (more than 44 bytes).

### 4.5.2 Timeout method

All data processing procedures involved in Android application were written in one thread, so the whole program will be blocked if the phone cannot connect to server,



which will affect receiving data from device. For that consideration, we have set a timeout value (5s) for HTTP post connection. Moreover, the buffer size will increase 5 record capacities for 5 more data message to receive if Android phone fail to connect to server once, and then try to upload again. This loop will repeat up to 15 times. After that saved data during this period will be deleted, and buffer size will reduce to 20 records again. This design consideration is to guarantee stability of the whole system because 3G network signal is very easily to lose for multiple reasons, such as moving at high speed on freeway. Commonly, it will recover within several minutes. Fortunately, short-term period of network lose will not affect function of the whole system benefited from the above consideration.

## 5. Future work

Haze watch system is still in its developing stage, so there are a lot of things need to be researched, improved, modified, and even corrected. In particular, I will summarize future works expected to be done focusing on our topic in this section.

### 5.1 Interpolation

Currently, linear interpolation function has been combined with non-linear interpolation. Tunnels below 500m will be applied linear interpolation, and non-linear interpolation is only responsible for five long tunnels mentioned in section 4.2.5. Tunnels above 500m are not made into consideration. Since tunnels applied linear interpolation must be very straight, so tunnels below 500m but not straight may also fail to apply linear-interpolation. Non-linear interpolation performs much better than linear one, but it has to depend on pre-stored digital map. Therefore, tunnels which are not sufficiently straight are recommended to apply non-linear interpolation, which means digital map of those tunnels need to be constructed.

## 5.2 Two-way communication

### 5.2.1 Reception protocol

In current stage, request messages only consist of a single number to distinguish the control request and data request. In the future, types of request message may not only constrain in these two types, and also the device needs to receive instructions from mobile phone. The most basic requirement is that the message should be added a header and footer or at least a start bit and stop bit, which helps to prevent errors. However, the message format cannot be designed too complicated because microcontroller cannot operate as fast as mobile phone.

### 5.2.2 New functions

In our thesis, we only realized two new function of two-way communication. Actually, two-way communication has large potential capability remaining developing. For example, disable a particular sensor, which is used to save more power on the device if a particular sensor is not much concerned in some area. Since microcontroller has 14 I/O ports, and most of them currently idle, so we can connect one of them to a certain sensor's power pin. When the I/O port outputs 0, sensor connected to that pin will be grounded.

## 6. Conclusion

Haze watch system is aim to design a mobile air pollution monitoring system with high resolution and low-cost for individuals as well as environment department. Mobile pollution monitoring will be a great achievement as it popularizes air pollution monitoring to normal people, and also supplement the weak point of fixed monitoring station.

The work we have done in this thesis helps improving the sensor unit part of the system. Specifically, we have successfully estimated locations in tunnel when positioning signal lost by interpolation. Also, we have improved the transmission protocol to avoid repeatedly sending control field, reduce transmission delay and location error, as well as save more power. What's more, we have realized two-way communication between device and Android phone, which enhanced the connection guarantee. Based on two-way communication, displacement-based sampling mode is developed that can avoid keeping sampling data within a very small area. Lastly, we also improve some parts regarding upload process. The achievements have been carefully tested in the related experiment.

## References

- [1] Air pollution.(n.d.) [Online]. Retrieved May 10,2010, from [http://library.thinkquest.org/26026/Environmental\\_Problems/air\\_pollution.html](http://library.thinkquest.org/26026/Environmental_Problems/air_pollution.html)
- [2] Tom Socha.(2007,Jan.) Air Pollution Causes and Effects. [Online]. Retrieved May 10, 2010 [http://healthandenergy.com/air\\_pollution\\_causes.htm](http://healthandenergy.com/air_pollution_causes.htm)
- [3] Cornell University (2007, August 14). Pollution Causes 40 Percent Of Deaths Worldwide, Study Finds. *ScienceDaily*. [Online]. Retrieved May 10, 2011, from <http://www.sciencedaily.com/releases/2007/08/070813162438.htm>
- [4] J. Carrapetta (2010, Oct), "Haze Watch: Design of a wireless sensor board for measuring air pollution". School of Electrical and Telecommunications Engineering, University of New South Wales, Sydney.
- [5] N. Youdale (2010, Oct), "Haze Watch: Database Server and Mobile Applications for Measuring and Evaluating
- [6] Air Pollution Exposure, University of New South Wales Undergraduate Thesis, 2010.
- [7] Vanderbilt University. MAQUMON. [Online]. Retrieved May 10, 2011, from <http://www.isis.vanderbilt.edu/projects/maqumon>
- [8] Lu D. (2011). "Android interface for pollution monitoring system". Thesis B Report. School of Electrical and Telecommunications Engineering, University of New South Wales, Sydney.
- [9] Microchip (2008). PIC16F631/677/685/687/689/690 Datasheet. [Datasheet].
- [10] Adeunis R.F. (2007, Aug). *ARF32 Bluetooth Modules User Guide*. [Online]. Retrieved May 11,2011, from <http://www.adeunis-rf.com>
- [11] National Environment Protection Measure for Ambient Air Quality: Air Monitoring Plan for NSW (2001, Jun). [Online]. Retrieved May 10, 2011, from <http://www.environment.nsw.gov.au/air/nepm/index.htm>
- [12] Obtaining User Location (n.d.). [Online]. Retrieved May 10, 2011, from <http://developer.android.com/guide/topics/location/obtaining-user-location.html>
- [13] Dimitris s. & Nathan W. et al. (2009, Oct). "Indoor Navigation System for

Handheld Devices”. faculty of the Worcester Polytechnic Institute, Worcester, Massachusetts, USA. [Online] Retrieved from [http://www.wpi.edu/Pubs/E-project/Available/E-project-102209-164024/unrestricted/Indoor\\_Navigation\\_System\\_for\\_Handheld\\_Devices.pdf](http://www.wpi.edu/Pubs/E-project/Available/E-project-102209-164024/unrestricted/Indoor_Navigation_System_for_Handheld_Devices.pdf)

[14] LUKIANTO .C & STERNBERG H. (2011, May). “Overview and Evaluation of Current Indoor Navigation Techniques and Implementation Studies”. Marrakech, Morocco. [Online]. Retrieved from [http://www.fig.net/pub/fig2011/papers/ts09a/ts09a\\_lukianto\\_sternberg\\_5102.pdf](http://www.fig.net/pub/fig2011/papers/ts09a/ts09a_lukianto_sternberg_5102.pdf)

[15] Australia Tunneling Society (n.d.). “M5 East Tunnel”, Sdney. [Online]. Retrieved May 15, 2011, from [http://www.ats.org.au/index.php?option=com\\_content&task=view&id=156&Itemid=4](http://www.ats.org.au/index.php?option=com_content&task=view&id=156&Itemid=4)

[16] Wikipedia (ed. 2011). “Haversine formula”. [Online]. Retrieved from [http://en.wikipedia.org/wiki/Haversine\\_formula](http://en.wikipedia.org/wiki/Haversine_formula)

[17] Jiang J. (2011). “Communication Between Sensors and Mobile Phone”. Thesis B Report. School of Electrical and Telecommunications Engineering, University of New South Wales, Sydney.

# Appendix

## PollutionSensor.java

```
package com.unsw.pollutionsensor;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.content.Context;
import android.content.Intent;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.provider.Settings;
import android.util.Log;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.TextView;

public class PollutionSensor extends Activity {
    private PollutionLocationListener locationListener = null;
    private LocationManager locationManager = null;
    BluetoothThread bluetoothThread = null;
    private static final int CHOOSE_DEVICE_REQUEST = 10;
    private String deviceMACAddress = null;
    TextView logView = null;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.pollution_sensor);
        Log.v("debug", "Pollution Sensor onCreate");
        logView = (TextView) findViewById(R.id.logTextView);
        UILog.setLogView(logView);
        UILog.println(UILog.LOG_TYPE_INFO, "App startup");

        // RadioButton config
        RadioGroup myRG =
(RadioGroup) this.findViewById(R.id.radioGroup1);
        final RadioButton push =
(RadioButton) this.findViewById(R.id.radio0);
        final RadioButton aquire =
(RadioButton) this.findViewById(R.id.radio1);
```

```

        myRG.setOnCheckedChangeListener(new
RadioGroup.OnCheckedChangeListener() {

        @Override
        public void onCheckedChanged(RadioGroup group, int checkedId)
{

        if (bluetoothThread!=null) {
            // TODO Auto-generated method stub
            if (checkedId == push.getId()) {
                bluetoothThread.settingPush(true);
            } else if (checkedId == aquire.getId()) {
                bluetoothThread.settingPush(false);
            }
        }
    }

});

if (deviceMACAddress == null) {
    // First spawn the device chooser
    Intent intent = new Intent (this, DeviceChooser.class);
    startActivityForResult(intent, CHOOSE_DEVICE_REQUEST);
} else {
    startBluetoothConnection(deviceMACAddress);
}
}

protected void onActivityResult(int requestCode, int resultCode,
Intent data) {
    if (requestCode == CHOOSE_DEVICE_REQUEST) {
        if (resultCode == RESULT_OK) {
            // A contact was picked. Here we will just display it
            // to the user.
            deviceMACAddress = data.getStringExtra("MACAddress");
            Log.v("debug","Try Start BT connection to " +
deviceMACAddress);
            startBluetoothConnection(deviceMACAddress);
        }
    }
}

private void startBluetoothConnection (String address) {
    if (address == null || address.length() == 0) {
        Log.e("debug","Failed to get MAC address from intent, using

```



```

default instead");
    address = "00:18:B2:01:1E:D2";
}

UILog.clear();
UILog.println(UILog.LOG_TYPE_INFO, "Connecting to "+address);
Log.v("debug", "will connect to "+address);

BluetoothAdapter bluetoothAdapter =
BluetoothAdapter.getDefaultAdapter();

    if (!bluetoothAdapter.isEnabled()) {
        Intent enableBtIntent = new Intent
(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivityForResult(enableBtIntent, 500); // 500 is an
arbitrary number (could be anything)
    }

    // setup location updates
    locationManager = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);
    locationListener = new PollutionLocationListener();

    if(!locationManager.isProviderEnabled(android.location.LocationManag
er.GPS_PROVIDER )) {
        Intent myIntent = new
Intent( Settings.ACTION_LOCATION_SOURCE_SETTINGS );
        startActivity(myIntent);
    }

    //locationManager.requestLocationUpdates(LocationManager.GPS_PROV
IDER, 0,200, locationListener);

    //locationManager.requestLocationUpdates(LocationManager.NETWORK_
PROVIDER, 0, 200, locationListener);

    locationManager.requestLocationUpdates(LocationManager.GPS_PROVID
ER, 5000,0, locationListener);

    locationManager.requestLocationUpdates(LocationManager.NETWORK_PR
OVIDER, 5000, 0, locationListener);

```

```

        bluetoothThread = new BluetoothThread();
        bluetoothThread.setDeviceMACAddress(address);
        bluetoothThread.setLocationListener(locationListener);
        bluetoothThread.start();
    }

    protected void onStop () {
        super.onStop();
        Log.v("debug", "Pollution sensor onStop");
        if (bluetoothThread != null) {
            bluetoothThread.cancel();
        }
        //off GPS update
        if (locationManager != null && locationListener != null){
            locationManager.removeUpdates(locationListener);
        }
    }

    protected void onDestroy(){
        super.onDestroy();
        Log.v("debug", "Pollution sensor on destroy");
        System.exit(0);
    }

    public class PollutionLocationListener implements LocationListener
    {
        private Location lastLocation = null;
        private Location initialLocation = null;
        public boolean newLocation = false;
        //testing
        private int counter = 0;

        @Override
        public void onLocationChanged(Location location) {
            Log.i("location", "in Location changed:
"+location.toString()+" counter = "+counter);

            if(initialLocation == null && location.getAccuracy()<200){
                Log.d("location", "iniLocation:
"+location.getAccuracy()+" "+location.getLongitude()
                +" "+location.getLatitude());
                initialLocation = location;
            }
        }
    }

```

```

}

/**
 * Two Way -- Location based data collect */

if(location!=null){
    if (lastLocation == null){
        if(location.getAccuracy()<100){
            Log.d("location", "newLocation:
"+location.getAccuracy()+" "+location.getLongitude()
            +" "+location.getLatitude());
            lastLocation = location;
            newLocation = true;
        }
    }else{
        long timeDelta = location.getTime() -
lastLocation.getTime();
        if(timeDelta>0 && location.getAccuracy()<100){
            Log.d("location", "newLocation:
"+location.getAccuracy()+" "+location.getLongitude()
            +" "+location.getLatitude());
            lastLocation = location;
            newLocation = true;
        }else if(timeDelta>0 && location.getAccuracy()>=100){
            Log.d("location", "inAccLocation:
"+location.getAccuracy()+" "+location.getLongitude()
            +" "+location.getLatitude());
            lastLocation = null;
            newLocation = false;
        }else{
            newLocation = false;
        }
    }
}

/**
 * Two Way -- Location based data collect finish */

/**
 * Old filter */

```

```

        /*if (isBetterLocation(location, lastLocation)) {
            lastLocation = location;
        }*/

        /**
         * Old filter end*/
    }

    public Location getInitialLocation () {
        return initialLocation;
    }

    public Location getLastLocation () {
        if(lastLocation!=null){
            long dt = System.currentTimeMillis() -
lastLocation.getTime();
            if(dt>100001){
                lastLocation = null;
                Log.e("location","Location timeout.");
            }else{
                Log.i("location",lastLocation.toString());
            }
        }
        return lastLocation;
    }

    @Override
    public void onProviderDisabled(String provider) {
        // TODO Auto-generated method stub
        Log.v("location","Location provider disable.");
    }

    @Override
    public void onProviderEnabled(String provider) {
        // TODO Auto-generated method stub
        Log.v("debug","Location provider enable.");
    }

    @Override
    public void onStatusChanged(String provider, int status, Bundle
extras) {
        // TODO Auto-generated method stub
    }

```

```

// Some example code from Android docs
private static final int TWO_MINUTES = 1000 * 60 * 2;

/** Determines whether one Location reading is better than the
current Location fix
 * @param location The new Location that you want to evaluate
 * @param currentBestLocation The current Location fix, to which
you want to compare the new one
 */
protected boolean isBetterLocation(Location location, Location
currentBestLocation) {
    if (currentBestLocation == null) {
        // A new location is always better than no location
        return true;
    }

    // Check whether the new location fix is newer or older
    long timeDelta = location.getTime() -
currentBestLocation.getTime();
    boolean isSignificantlyNewer = timeDelta > TWO_MINUTES;
    boolean isSignificantlyOlder = timeDelta < -TWO_MINUTES;
    boolean isNewer = timeDelta > 0;

    // If it's been more than two minutes since the current location,
use the new location
    // because the user has likely moved
    if (isSignificantlyNewer) {
        return true;
    }
    // If the new location is more than two minutes older, it must
be worse
    } else if (isSignificantlyOlder) {
        return false;
    }

    // Check whether the new location fix is more or less accurate
    int accuracyDelta = (int) (location.getAccuracy() -
currentBestLocation.getAccuracy());
    boolean isLessAccurate = accuracyDelta > 0;
    boolean isMoreAccurate = accuracyDelta < 0;
    boolean isSignificantlyLessAccurate = accuracyDelta > 200;

    // Check if the old and new location are from the same provider
    boolean isFromSameProvider =
isSameProvider(location.getProvider(),

```

```

        currentBestLocation.getProvider());

        // Determine location quality using a combination of timeliness
and accuracy
        if (isMoreAccurate) {
            return true;
        } else if (isNewer && !isLessAccurate) {
            return true;
        } else if (isNewer && !isSignificantlyLessAccurate &&
isFromSameProvider) {
            return true;
        }
        return false;
    }

    /** Checks whether two providers are the same */
    private boolean isSameProvider(String provider1, String provider2)
    {
        if (provider1 == null) {
            return provider2 == null;
        }
        return provider1.equals(provider2);
    }
}
}

```

### BluetoothStreamReader.java

```

package com.unsw.pollutionsensor;

import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import android.util.Log;

public class BluetoothStreamReader {
    private boolean cancelled = false;

    private static enum State {
        RECEIVE_HEADER,
        RECEIVE_PROTOCOL_VERSION,
        RECEIVE_DEVICE_ID,
        RECEIVE_COEFFICIENT_SENSOR_ID,
    }
}

```

```

    RECEIVE_COEFFICIENT_VALUES,
    RECEIVE_REFERENCE_VOLTAGE,
    RECEIVE_BATTERY_LEVEL,
    RECEIVE_SENSOR_VALUE_PAIR_COUNT,
    RECEIVE_SENSOR_INDEX,
    RECEIVE_SENSOR_VALUE,
    RECEIVE_CHECKSUM,
    RECERIVE_MSG_TYPE,
    RECERIVE_LAST_CALIBRATION,
    RECERIVE_NEXT_CALIBRATION,
    RECERIVE_FOOTER;
};

private static final int MSG_HEADER[] = {0xFF, 0xFF, 0xFF, 0xFF};
private static final int MSG_FOOTER[] = {0xEE, 0xEE, 0xEE, 0xEE};
private static final int CURRENT_PROTOCOL_VERSION = 2;
private static final int CONTROL_MSG = 1;
private static final int DATA_MSG = 2;
private boolean messageComplete = false;

public void cancel() {
    cancelled = true;
}

public DeviceMessage receiveSensorMessage (InputStream inStream,
DeviceMessage dMessage, int MsgType) {
    messageComplete = false;
    State state = State.RECEIVE_HEADER;
    int d, data_index = 0;
    ArrayList<Short> msgBytes = new ArrayList<Short>();

    try {
        while (!cancelled && !messageComplete) {
            d = inStream.read(); // read byte from stream
            Log.d("in", Integer.toString(d));
            //msgBytes.add((short)d);
            //debugBytesPrint(msgBytes);

            switch (state) {
                case RECEIVE_HEADER:
                    if (d == MSG_HEADER[data_index]) {
                        data_index++;
                        if (data_index == 4) {
                            state = State.RECEIVE_PROTOCOL_VERSION;

```



```

        data_index = 0;
    }
} else {
    data_index = 0;
}
break;
case RECEIVE_PROTOCOL_VERSION:
    if (d == CURRENT_PROTOCOL_VERSION) {
        msgBytes.add((short)d);
        readRestOfMessage_v2(dMessage, inStream, state,
msgBytes, MsgType);
        return dMessage;
    } else if (d > CURRENT_PROTOCOL_VERSION) {
        System.out.println("WARNING: received message of
newer unknown protocol version ("+d+"). Results are undefined");
        return dMessage;
    } else if (d == 1){
        System.out.println("WARNING: received message
protocol version ("+d+") cannot process");
        return dMessage;
    }
}
break;
}
}
inStream.close();
} catch (IOException e) {
    Log.e("debug", "Protocol version reading error:
"+e.toString());
}

return dMessage;
}

```

```

private DeviceMessage readRestOfMessage_v2 (DeviceMessage dMessage,
InputStream inStream, State state, ArrayList<Short> msgBytes, int MsgType)
{

```

```

    int d;
    /**
     * Check message type 1 for control message, 2 for data message
     * */
    state = State.RECERIVE_MSG_TYPE;
    try{
        while (!cancelled && !messageComplete){

```

```

        d = inStream.read(); // read byte from stream
        Log.d("in", Integer.toString(d));
        msgBytes.add((short)d);
        //debugBytesPrint(msgBytes);
        Log.w("in", "read msg type.");
        if (d==CONTROL_MSG && d==MsgType){
            readControlMsg_v2(dMessage, inStream, state,
msgBytes);

            return dMessage;
        }else if(d==DATA_MSG && d==MsgType){
            readDataMsg_v2(dMessage, inStream, state, msgBytes);
            return dMessage;
        }else{
            dMessage.dataReceived = false;
            return dMessage;
        }
    }
} catch (IOException e) {
    Log.e("debug", "Msg type reading error: "+e.toString());
}

return dMessage;
}

```

```

private DeviceMessage readControlMsg_v2 (DeviceMessage dMessage,
InputStream inStream, State state, ArrayList<Short> msgBytes) {

```

```

    Log.i("in","read control msg.");
    int d, data_index = 0,i = 0;
    short byteBuf[] = new short[16];

    // initial state
    state = State.RECEIVE_DEVICE_ID;
    try {
        while (!cancelled && !messageComplete) {
            d = inStream.read(); // read byte from stream
            Log.d("in", Integer.toString(d));
            //msgBytes.add((short)d);
            //debugBytesPrint(msgBytes);

            switch (state) {
            case RECEIVE_DEVICE_ID:
                msgBytes.add((short)d);
                byteBuf[data_index++] = (short)d;

```

```

        if (data_index >= 2) {
            dMessage.deviceIdentifier =
(byteBuf[0]<<8) |byteBuf[1];
            data_index = 0;
            state = State.RECERIVE_LAST_CALIBRATION;
        }
        break;
case RECERIVE_LAST_CALIBRATION:
    msgBytes.add((short)d);
    byteBuf[data_index++] = (short)d;
    if (data_index >= 3){
        dMessage.LastClibDate[0] = byteBuf[0];
        dMessage.LastClibDate[1] = byteBuf[1];
        dMessage.LastClibDate[2] = byteBuf[2];
        data_index = 0;
        state = State.RECERIVE_NEXT_CALIBRATION;
    }
    break;
case RECERIVE_NEXT_CALIBRATION:
    msgBytes.add((short)d);
    byteBuf[data_index++] = (short)d;
    if (data_index >= 3){
        dMessage.NextCLibDate[0] = byteBuf[0];
        dMessage.NextCLibDate[1] = byteBuf[1];
        dMessage.NextCLibDate[2] = byteBuf[2];
        data_index = 0;
        state = State.RECEIVE_REFERENCE_VOLTAGE;
    }
    break;
case RECEIVE_REFERENCE_VOLTAGE:
    msgBytes.add((short)d);
    byteBuf[data_index++] = (short)d;
    if (data_index >= 2) {
        dMessage.rawReferenceVoltage =
(byteBuf[0]<<8) |byteBuf[1];
        data_index = 0;
        state = State.RECEIVE_COEFFICIENT_SENSOR_ID;
    }
    break;
case RECEIVE_COEFFICIENT_SENSOR_ID:
    msgBytes.add((short)d);
    dMessage.sensorIdentifier[data_index]=(int) d;

    state = State.RECEIVE_COEFFICIENT_VALUES;

```

```

        break;
    case RECEIVE_COEFFICIENT_VALUES:
        msgBytes.add((short)d);
        byteBuf[i++] = (short)d;
        if (i >= 6) {
            dMessage.rawCoefficients[0] =
(byteBuf[0]<<8)|byteBuf[1];
            dMessage.rawCoefficients[1] =
(byteBuf[2]<<8)|byteBuf[3];
            dMessage.rawCoefficients[2] =
(byteBuf[4]<<8)|byteBuf[5];

            dMessage.coefficientMap.put(dMessage.sensorIdentifier[data_index],
dMessage.getSensorCoes());
            data_index++;
            i = 0;
            if (data_index < 3) {
                state = State.RECEIVE_COEFFICIENT_SENSOR_ID;
            } else {
                state = State.RECEIVE_CHECKSUM;
                data_index = 0;
            }
        }
        break;
    case RECEIVE_CHECKSUM:
        int sum = 0;
        for(short temp : msgBytes){
            sum += temp;
        }
        Log.i("in", Integer.toBinaryString(sum));
        Log.i("in", Integer.toBinaryString(d));

        int L = Integer.toBinaryString(d).split("").length-1;

        String[] checkSum =
Integer.toBinaryString(sum^d).split("");
        Log.i("in", Integer.toBinaryString(sum^d)+"
"+checkSum.length);
        boolean check = true;
        for(int
idx=checkSum.length-1;idx>=(checkSum.length-L)&&idx>0;idx--){

            if(Integer.parseInt(checkSum[idx])!=0) check=false;
        }

```

```

        //if(((double) (sum^d))!=Math.pow(2,
(checkSum.length-2))){
        if(!check){
            msgBytes.clear();
            messageComplete = true;
            Log.i("debug", "check sum wrong!");
        }else{
            dMessage.receivedChecksum = (short)d;
            state = State.RECERIVE_FOOTER;
        }
        break;
    case RECERIVE_FOOTER:
        if (d == MSG_FOOTER[data_index]) {
            data_index++;
            if (data_index == 4) {
                data_index = 0;
                Log.i("debug", "Control stream received");
                dMessage.controlReceived = true;
                messageComplete = true;
                msgBytes.clear();
            }
        } else {
            msgBytes.clear();
            System.out.println("Error: message format
error.");
            messageComplete = true;
        }
        break;
    default:
        break;
    }
}
} catch (IOException e) {
    Log.e("debug", "Control reading error: "+e.toString());
}
return dMessage;
}

```

```

private DeviceMessage readDataMsg_v2 (DeviceMessage dMessage,
InputStream inStream, State state, ArrayList<Short> msgBytes) {
    Log.i("in", "read data msg.");
    SensorData tempSensor = new SensorData();

    int d, data_index = 0;

```

```

int i = 0;
short byteBuf[] = new short[16];
short s = 0;
// initial state
// state = State.RECEIVE_DEVICE_ID;
state = State.RECEIVE_BATTERY_LEVEL;
try {
    while (!cancelled && !messageComplete) {
        d = inStream.read(); // read byte from stream
        Log.d("in", Integer.toString(d));
        //msgBytes.add((short)d);
        //debugBytesPrint(msgBytes);

        switch (state) {
        case RECEIVE_BATTERY_LEVEL:
            msgBytes.add((short)d);
            byteBuf[i++] = (short)d;

            if (i>1) {
                s = (short) (byteBuf[0]*256+byteBuf[1]);
                tempSensor.batteryLevel = s;
                state = State.RECEIVE_SENSOR_INDEX;
                s = 0;
                i = 0;
            }
            break;
        case RECEIVE_SENSOR_INDEX:
            msgBytes.add((short)d);
            tempSensor.sensorIdentifier[data_index] = (int)d;
            state = State.RECEIVE_SENSOR_VALUE;
            break;
        case RECEIVE_SENSOR_VALUE:
            msgBytes.add((short)d);
            byteBuf[i++] = (short)d;
            if(i>1){
                s = (short) (byteBuf[0]*256+byteBuf[1]);
                tempSensor.sensorData[data_index] = s;

                tempSensor.sensorDataMap.put(tempSensor.sensorIdentifier[data_index], tempSensor.sensorData[data_index]);
                data_index++;
                i = 0;
                if (data_index >= 3){
                    state = State.RECEIVE_CHECKSUM;

```

```

        data_index = 0;
        s = 0;
        i = 0;
    }else{
        state = State.RECEIVE_SENSOR_INDEX;
    }
}
break;
case RECEIVE_CHECKSUM:
    int sum = 0;
    for(short temp : msgBytes){
        sum += temp;
    }
    Log.i("in", Integer.toBinaryString(sum));
    Log.i("in", Integer.toBinaryString(d));

    int L =
Integer.toBinaryString(d).split("").length-1;

    String[] checkSum =
Integer.toBinaryString(sum^d).split("");
    Log.i("in", Integer.toBinaryString(sum^d)+"
"+checkSum.length);
    boolean check = true;
    for(int
idx=checkSum.length-1;idx>=(checkSum.length-L)&&idx>0;idx--){

        if(Integer.parseInt(checkSum[idx])!=0) check=false;
    }
    //if(((double)(sum^d))!=Math.pow(2,
(checkSum.length-2))){
        if(!check){
            msgBytes.clear();
            messageComplete = true;
            Log.i("debug", "check sum wrong!");
        }else{
            dMessage.receivedChecksum = (short)d;
            state = State.RECERIVE_FOOTER;
        }
    }
    break;
case RECERIVE_FOOTER:
    if (d == MSG_FOOTER[data_index]) {
        data_index++;
        if (data_index == 4) {

```



```

        data_index = 0;
        messageComplete = true;
        dMessage.sensorData = tempSensor;
        Log.i("debug", "Data stream received");
        msgBytes.clear();
        dMessage.dataReceived = true;
    }
} else {
    System.out.println("Error: message format
error.");

    msgBytes.clear();
    messageComplete = true;
}
break;
default:
    break;
}
}
} catch (IOException e) {
    Log.e("debug", "Data reading error: "+e.toString());
}

return dMessage;
}
}

```

### BluetoothThread.java

```

package com.unsw.pollutionsensor;

import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.StringWriter;
import java.io.UnsupportedEncodingException;
import java.text.DecimalFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;

```

```

import java.util.List;
import java.util.Map;
import java.util.UUID;

import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.ParseException;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.HttpClient;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.params.BasicHttpParams;
import org.apache.http.params.HttpConnectionParams;
import org.apache.http.params.HttpParams;
import org.apache.http.util.EntityUtils;
import org.xmlpull.v1.XmlSerializer;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.location.Location;
import android.util.Log;
import android.util.Xml;
import
com.unsw.pollutionsensor.PollutionSensor.PollutionLocationListener;

public class BluetoothThread extends Thread {
    private String deviceMACAddress = null;
    private static final UUID RFCOMM_UUID =
UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");
    private boolean receiveEnabled = true;
    private PollutionLocationListener locationListener = null;
    private ArrayList<DataWrapper> GPSBuffer = new
ArrayList<DataWrapper>();
    private ArrayList<DataWrapper> dataBuffer = new
ArrayList<DataWrapper>();
    private BluetoothStreamReader streamReader = null;

    private Location initialLocation = null;
    private int count = 0;
    private Date initialDate = null;

```

```

private static final int SENSOR_ID_NO2 = 2;
private static final int SENSOR_ID_O3 = 1;
private static final int SENSOR_ID_CO = 3;

//Max No. of data per-bundle
private int MaxStandbyData = 5;

/**
 * */
private boolean dataPush = true;

public void settingPush(boolean temp) {
    dataPush = temp;
}
/**
 * */

public void setDeviceMACAddress (String address) {
    deviceMACAddress = address;
}

public void setLocationListener (PollutionLocationListener listener)
{
    locationListener = listener;
}

private BluetoothSocket getBluetoothSocket() {
    BluetoothAdapter bluetoothAdapter =
BluetoothAdapter.getDefaultAdapter();
    BluetoothDevice device =
bluetoothAdapter.getRemoteDevice(deviceMACAddress);
    BluetoothSocket btSocket = null;

    try {
        btSocket =
device.createRfcommSocketToServiceRecord(RFCOMM_UUID);
    } catch (IOException e) {
        Log.e("debug", "Error: failed to create socket");
    }

    // Discovery is quite intensive. Just in case it's running, stop
it

```

```

        bluetoothAdapter.cancelDiscovery();

        return btSocket;
    }

    public void cancel () {
        if (streamReader != null) {
            streamReader.cancel();
        }
        receiveEnabled = false;
    }

    public void run () {

        BluetoothSocket socket = null;
        InputStream inStream = null;
        OutputStream outStream = null;
        DataOutputStream dataOutputStream = null;
        receiveEnabled = true;
        DeviceMessage msg = new DeviceMessage();

        /**
         * Get bluetooth connection */

        socket = getBluetoothSocket();
        try {
            socket.connect();
        } catch (IOException e2) {
            // TODO Auto-generated catch block
            Log.e("debug", "Failed to connect to socket");
            UILog.println(UILog.LOG_TYPE_INFO, "Failed to connect");
        }

        try {
            inStream = socket.getInputStream();
        } catch (Exception e1) {
            // TODO Auto-generated catch block
            Log.e("debug", "Error: failed to connect to inputstream");
        }

        try {
            outStream = socket.getOutputStream();
        } catch (Exception e1) {
            // TODO Auto-generated catch block
            Log.e("debug", "Error: failed to connect to outputstream");
        }
    }
}

```

```

}
dataOutputStream = new DataOutputStream(outStream);

Log.i("debug", "Connected to socket");
UILog.println(UILog.LOG_TYPE_INFO, "Connected!");

/**
 * Receive data*/

while (receiveEnabled) {

    /**
     * Acquiring control msg
     * */

    while (!msg.controlReceived){
        Log.w("debug", "Don't have control msg. Acquiring
contral msg...");
        try {
            dataOutputStream.write(1);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            Log.e("debug", "Error: failed to send data");
        }
        msg = receiveSensorValues(inStream, msg, 1);
    }

    /**wait for the initial known location*/

    while (initialLocation == null) {
        Log.w("debug", "Waiting for Initial Location
information");
        initialLocation =
locationListener.getInitialLocation();
        initialDate = new Date();
        try {
            Thread.sleep(500);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}

```

```

    /**Wait for location changes*/
    if(!dataPush){
        Log.w("debug", "Waiting for Location change");
        while(!locationListener.newLocation);
        locationListener.newLocation = false;
    }

    /**Acquiring Data */

    while (!msg.dataReceived){

        try {
            Thread.sleep(8000);
        } catch (InterruptedException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }

        Log.w("debug", "Acquiring data msg...");
        try {
            dataOutputStream.write(4);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            Log.e("debug", "Error: failed to send data");
        }
        msg = receiveSensorValues(inStream, msg, 2);
    }

    /**Data Processing */

    processSensorValues(msg);
    msg.dataReceived = false;

    // Wait a bit
    try {
        Log.v("debug", "Bluetooth Thread is Sleeping...");
        Thread.sleep(1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

```

```

    try {
        inStream.close();
        outputStream.close();
    } catch (IOException e) {
        e.printStackTrace();
    }

    try {
        if (socket != null) {
            socket.close();
            Log.i("debug", "socket closed");
        }
    } catch (IOException e) {
        Log.e("debug", "Error: failed to close socket");
    }

    Log.v("debug", "Bluetooth Thread finish");
}

private DeviceMessage receiveSensorValues (InputStream inStream,
DeviceMessage dMessage, int MsgType) {
    streamReader = new BluetoothStreamReader();
    DeviceMessage msg =
streamReader.receiveSensorMessage (inStream, dMessage, MsgType);
    streamReader = null;

    return msg;
}

private void processSensorValues (DeviceMessage msg) {
    Log.d("debug", "processing data.");

    // Pollutant values
    DataWrapper wrapper = new DataWrapper();
    wrapper.pollutantValues = convertToPollutantValues(msg);

    wrapper.date = new Date();
    Location location = locationListener.getLastLocation();

    if(location != null){
        Log.i("debug", "accepted location:
"+location.getAccuracy()+" "+location.getLongitude()

```



```

        +" "+location.getLatitude());
wrapper.location = location;

dataupload(wrapper);

if (count == 0){
    Log.w("debug", "update initiallocation.");
    initialLocation = location;
    initialDate = new Date();
} else{
    Log.w("debug", "counter = "+count+" try
interpolation.");
    //Interpolation(location, wrapper.date);
    linearInterpolation(location, wrapper.date);

    initialLocation = location;
    initialDate = new Date();
    count = 0;
    if(dataBuffer.size() != 0){
        if(postData(dataBuffer)){
            dataBuffer.clear();
            Log.w("debug", "!!!Done interpolation!!!");
        }
    } else{
        GPSBuffer.clear();
    }
}

}
//if don't have location information
else if (location == null){

    if(dataBuffer.size() != 0){
        if (postData(dataBuffer)) {
            dataBuffer.clear();
        }
    }

    count += 1;
    GPSBuffer.add(wrapper);
    Log.e("debug", "inacc location. "+count);
}
}

```

```

private String pollutantStringForSensor (int sensorIdentifier) {
    switch (sensorIdentifier) {
        case SENSOR_ID_NO2:
            return "no2";
        case SENSOR_ID_O3:
            return "o3";
        case SENSOR_ID_CO:
            return "co";
    }
    return "unknown";
}

private boolean isSensorEnabled (int sensorIdentifier) {
    switch (sensorIdentifier) {
        case SENSOR_ID_NO2:
            return true;
        case SENSOR_ID_O3:
            return true; // don't have calibration values for this
        case SENSOR_ID_CO:
            return true;
    }
    return false;
}

private double pollutantValueForSensor (short data, double refVoltage,
double[] sensorCoefficient) {
    double v = ((double)data)/255.0*refVoltage;

    double c0 = sensorCoefficient[0];
    double c1 = sensorCoefficient[1];
    double c2 = sensorCoefficient[2];

    // Apply quadratic calibration function
    double calibrated_value = c0*v*v + c1*v + c2;
    System.out.println("calibrated value = "+calibrated_value);
    if (calibrated_value < 0) calibrated_value = 0;
    return calibrated_value;
}

private Map<String, Double> convertToPollutantValues (DeviceMessage
msg) {
    Map<String, Double> map = new HashMap<String, Double>();

```

```

        for (int sensorID : msg.coefficientMap.keySet()) {
            if (isSensorEnabled(sensorID)) {
                map.put (pollutantStringForSensor (sensorID),

pollutantValueForSensor (msg.sensorData.sensorDataMap.get (sensorID
), msg.getReferenceVoltage(), msg.coefficientMap.get (sensorID)));
            }
        }
        return map;
    }

    private String xmlForPollutantValues (ArrayList<DataWrapper>
dataList) {
        XmlSerializer serializer = Xml.newSerializer();
        StringWriter writer = new StringWriter();
        String xmlString = null;

        try {
            serializer.setOutput (writer);
            serializer.startDocument ("UTF-8", true);
            serializer.startTag ("", "data");
            serializer.startTag ("", "info");
            serializer.startTag ("", "user_name");
            serializer.text ("debug");
            serializer.endTag ("", "user_name");
            serializer.startTag ("", "comment");
            serializer.text ("Testing");
            serializer.endTag ("", "comment");
            serializer.startTag ("", "group");
            serializer.text ("3608");
            serializer.endTag ("", "group");
            serializer.endTag ("", "info");
            serializer.startTag ("", "samples");

            SimpleDateFormat dateFormat = new
SimpleDateFormat ("yyyy-MM-dd HH:mm:ss");

            // print samples
            for (DataWrapper d : dataList) {
                serializer.startTag ("", "sample");
                serializer.startTag ("", "datetime");
                serializer.text (dateFormat.format (d.date));
                serializer.endTag ("", "datetime");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

```

        serializer.startTag("", "latitude");

serializer.text(String.valueOf(d.location.getLatitude()));
        serializer.endTag("", "latitude");
        serializer.startTag("", "longitude");

serializer.text(String.valueOf(d.location.getLongitude()));
        serializer.endTag("", "longitude");

        for (Map.Entry<String, Double> entry :
d.pollutantValues.entrySet()) {
            serializer.startTag("", entry.getKey());
            serializer.text(String.valueOf(entry.getValue()));
            serializer.endTag("", entry.getKey());
        }

        serializer.endTag("", "sample");
    }

    serializer.endTag("", "samples");
    serializer.endTag("", "data");
    serializer.endDocument();
    xmlString = writer.toString();
} catch (Exception e) {
    System.out.println("Error with xml");
}

Log.e("int",xmlString);
return xmlString;
}

private boolean postData (ArrayList<DataWrapper> dataList) {
    String responseBody = null;

    if (dataList.size() > 0) {
        String xml = xmlForPollutantValues(dataList);
        UILog.clear();
        UILog.printf(UILog.LOG_TYPE_INFO, "Most recent sample\n\n");
        UILog.println(UILog.LOG_TYPE_INFO,
dataList.get(dataList.size()-1).toString());

        HttpParams httpParameters = new BasicHttpParams();
        HttpConnectionParams.setConnectionTimeout(httpParameters,
5000);

```

```

        HttpClient client = new DefaultHttpClient(httpParameters);
        HttpPost httpPost = new
HttpPost("http://www.pollution.ee.unsw.edu.au/post-data.php");

        List<NameValuePair> nameValuePairs = new
ArrayList<NameValuePair>(2);
        nameValuePairs.add(new BasicNameValuePair("content", xml));

        if (client != null) {
            try {
                httpPost.setEntity(new
UrlEncodedFormEntity(nameValuePairs));
                HttpResponse response = client.execute(httpPost);
                if (response != null) {
                    responseBody =
EntityUtils.toString(response.getEntity());
                    Log.d("debug", "http response: "+response);
                    MaxStandbyData = 5;
                }
            } catch (UnsupportedEncodingException e) {
                e.printStackTrace();
            } catch (ClientProtocolException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
                MaxStandbyData+=5;
            } catch (ParseException e) {
                e.printStackTrace();
            }
        }
    }

    return (responseBody != null);
}

private class DataWrapper {
    public Date date = null;
    public Map<String, Double> pollutantValues = new HashMap<String,
Double>();
    public Location location = null;
    String s = "";

    public String toString() {

```

```

        return "{"+date+", "+pollutantValues+", "+location+"}";
    }

    public String toLogString() {
        SimpleDateFormat dateFormat = new
SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        //check whether location is null
        if (location != null){
            s = "Date: "+dateFormat.format(date)+"\nLocation:
"+location.getLatitude()+", "+location.getLongitude()+"\n\n";
        }else{
            s = "Date: "+dateFormat.format(date)+"\nLocation:
"+"null+", "+null+"\n\n";
        }

        DecimalFormat decimalFormat = new DecimalFormat("0.0000");
        for (Map.Entry<String, Double> entry :
pollutantValues.entrySet()) {
            s += entry.getKey()+" =
"+decimalFormat.format(entry.getValue())+" ppm\n";
        }
        return s;
    }
}

private void linearInterpolation(Location location, Date time){
    Location tempLocation = null;
    DataWrapper wrapper =null;
    double longitude = initialLocation.getLongitude();
    double latitude = initialLocation.getLatitude();
    double templongitude = 0;
    double templatitude = 0;

    double deltalongitude = (location.getLongitude() -
longitude)/(count+1);
    double deltalatitude = (location.getLatitude() -
latitude)/(count+1);

    for (int i=count;i>=1;i--){
        wrapper = new DataWrapper();
        tempLocation = new Location("gps");
        wrapper = GPSBuffer.get(i-1);
        //Date date = wrapper.date;
        templongitude = longitude + deltalongitude * i;
    }
}

```

```

        templatitude = latitude + deltalatitude* i;
        tempLocation.setLatitude(templatitude);
        tempLocation.setLongitude(templongitude);
        wrapper.location = tempLocation;

        dataBuffer.add(wrapper);
    }
}

private void Interpolation(Location location, Date time){
    Log.e("debug", "interpolation...");

    Tunnels tunnels = new Tunnels();
    final int dt_ShortTunnel = 4;

    double[][] route = null;
    if(count > dt_ShortTunnel){
        // the longitude and latitude for each entry of the tunnel
        // make sure the index is same
        double[][] entries = tunnels.entries;

        int n_tunnels = entries.length;
        double[] distanceOfEntry = new double[n_tunnels];
        double[] minEntry = {999999d, 999999d};
        int[] idxMinEntry = {0,0};

        for(int i=0;i<n_tunnels;i++){
            distanceOfEntry[i] =
calculateDistance(initialLocation,entries[i]);
            if(distanceOfEntry[i]<=minEntry[0]){
                minEntry[1] = minEntry[0];
                minEntry[0] = distanceOfEntry[i];
                idxMinEntry[1] = idxMinEntry[0];
                idxMinEntry[0] = i;
            }
        }

        Log.i("int", "distances: "+minEntry[0]+" "+minEntry[1]);

        boolean firstOutOfRange = minEntry[0]>600;
        boolean secondOutOfRange = minEntry[1]>600;

        // find possible exit

```

```

        if(firstOutOfRange){
            double[] loc =
{location.getLatitude(),location.getLongitude()};
            double distance =
calculateDistance(initialLocation,loc);

/*
            long lastTime = initialDate.getTime();
            double deltaT =(double) ((time.getTime() -
lastTime)/1000);// seconds
*/
            if(distance<1000){
                linearInterpolation(location,time);
            }
        }else if(secondOutOfRange){

            double[][] exits1 = tunnels.getExits(idxMinEntry[0]);

            double distanceOfExit = 0;
            int i = 0;
            int idxMinExit = 0;
            double minExit = 999999d;

            for(double[] loc : exits1){
                distanceOfExit = calculateDistance(location,loc);
                if(distanceOfExit<=minExit) {
                    minExit = distanceOfExit;
                    idxMinExit = i;
                }
                i++;
            }

            if (minExit<400) {
                route = tunnels.getTunnel(idxMinEntry[0],
idxMinExit);
                Log.d("int", "1: EN: "+idxMinEntry[0]+" EX:
"+idxMinExit);
                for(double[]temp1:route){
                    Log.d("int", "1: "+temp1[1]+","+temp1[0]);
                }
            }
        }else{
            double[] loc =
{location.getLatitude(),location.getLongitude()};
            double distance =

```



```

calculateDistance(initialLocation, loc);

/*          long lastTime = initialDate.getTime();
           double deltaT =(double) ((time.getTime() -
lastTime)/1000); // seconds
*/
           if(distance<1000){
               linearInterpolation(location,time);
           }
       }
else{

           double[][] exits1 = tunnels.getExits(idxMinEntry[0]);
           double[][] exits2 = tunnels.getExits(idxMinEntry[1]);

           double distanceOfExit = 0;

           int i = 0;
           int[] idxMinExit = {0,0};
           double[] minExit = {999999d, 999999d};

           for(double[] loc : exits1){
               distanceOfExit = calculateDistance(location,loc);
               if(distanceOfExit<=minExit[0]) {
                   minExit[0] = distanceOfExit;
                   idxMinExit[0] = i;
               }
               i++;
           }

           i = 0;
           for(double[] loc : exits2){
               distanceOfExit = calculateDistance(location,loc);
               if(distanceOfExit<=minExit[0]) {
                   minExit[1] = distanceOfExit;
                   idxMinExit[1] = i;
               }
               i++;
           }

           boolean betterRoute =
(minEntry[0]+minExit[0])<(minEntry[1]+minExit[1]);

```

```

        if(betterRoute){
            if (minExit[0]<400) {
                route =
tunnels.getTunnel(idxMinEntry[0],idxMinExit[0]);
                Log.d("int", "2: EN: "+idxMinEntry[0]+" EX:
"+idxMinExit[0]);

            }else{
                double[] loc =
{location.getLatitude(),location.getLongitude()};
                double distance =
calculateDistance(initialLocation,loc);
                /* long lastTime = initialDate.getTime();
                double deltaT =(double) ((time.getTime() -
lastTime)/1000);// seconds
*/
                if(distance<1000){
                    linearInterpolation(location,time);
                }
            }

        }else{
            if (minExit[1]<400) {
                route =
tunnels.getTunnel(idxMinEntry[1],idxMinExit[1]);
                Log.d("int", "3: EN: "+idxMinEntry[1]+" EX:
"+idxMinExit[1]);

            }else{
                double[] loc =
{location.getLatitude(),location.getLongitude()};
                double distance =
calculateDistance(initialLocation,loc);

                /*
                long lastTime = initialDate.getTime();
                double deltaT =(double) ((time.getTime() -
lastTime)/1000);// seconds
*/
                if(distance<1000){
                    linearInterpolation(location,time);
                }
            }
        }
    }

    if(route!=null) assignLocation(route);

```

```

    }else{
        linearInterpolation(location,time);
    }

}

private double calculateDistance(Location location, double[] tunnel) {
    double d_Long = location.getLongitude()-tunnel[1];
    double d_Lat = location.getLatitude()-tunnel[0];
    final double R = 6371004d;
    final double pi = Math.PI;
    final double R_sydney = R * 0.830265d;// longitude sydney R = R
* cos(-33.874)
    double distanceOfExit = Math.sqrt(Math.pow((pi*R/180*d_Lat),
2)+Math.pow(pi*R_sydney/180*d_Long, 2));
    return distanceOfExit;
}

private void assignLocation(double[][] route){
    Log.e("debug", "assign location...");

    Location location = null;

    int n = count;
    int N = route.length;

    DataWrapper wrapper = null;
    for (int k = 1;k<=n;k++){
        int index = ((N+1)*k/(n+1))-1;
        if(index < 0){
            index = 0;
        }else if(index>=route.length){
            index = route.length - 1;
        }
        wrapper = new DataWrapper();
        location = new Location("gps");
        Log.i("int","idx: "+index);

        wrapper = GPSBuffer.get(k-1);
        location.setLongitude(route[index][1]);
        location.setLatitude(route[index][0]);
        wrapper.location = location;
    }
}

```



```

import android.widget.RadioButton;
import android.widget.TextView;
import android.widget.Toast;

public class DeviceChooser extends Activity {
    //private ArrayAdapter<DeviceWrapper> arrayAdapter = null;
    private HashSet<BluetoothDevice> discoveredDevices = new
HashSet<BluetoothDevice>();
    private ArrayAdapter<DeviceWrapper> discoveryArrayAdapter = null;
    private ArrayAdapter<DeviceWrapper> builtInArrayAdapter = null;
    private int selectedListIndex = 1; // 'built in devices' list

    private class DeviceWrapper {
        private BluetoothDevice device;
        private String address;
        private String name;

        public DeviceWrapper (BluetoothDevice device) {
            this.device = device;
        }

        public DeviceWrapper (String address, String name) {
            this.address = address;
            this.name = name;
        }

        public String getAddress() {
            if (device != null) {
                return device.getAddress();
            } else {
                return address;
            }
        }

        public String getName() {
            if (device != null) {
                return device.getName();
            } else {
                return name;
            }
        }

        public String toString () {
            if (getName() != null && getName().length() > 0) {

```

```

        return getAddress()+" - "+getName();
    } else {
        return getAddress();
    }
}
}

private class ItemClickListener implements OnItemClickListener {
    public void onItemClick (AdapterView<?> arg0, View arg1, int arg2,
long arg3) {
        DeviceWrapper dw = getCurrentArrayAdapter().getItem(arg2);
        EditText MACField =
(EditText)findViewById(R.id.discoveryMACAddressField);
        MACField.setText(dw.getAddress());
    }
}

private void changeListContent(int listIndex) {
    if (listIndex != selectedListIndex) {
        selectedListIndex = listIndex;

        ListView listView = (ListView)findViewById(R.id.list_view);
        switch (selectedListIndex) {
            case 0:
                listView.setAdapter(discoveryArrayAdapter);
                break;
            case 1:
                listView.setAdapter(builtInArrayAdapter);
                break;
            default:
                break;
        }
    }
}

ArrayAdapter<DeviceWrapper> getCurrentArrayAdapter() {
    switch (selectedListIndex) {
        case 0:
            return discoveryArrayAdapter;
        case 1:
            return builtInArrayAdapter;
        default:
            break;
    }
}

```

```

        return null;
    }

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.device_chooser);
        System.out.println("Deivice chooser onCreate");
        // testing/debugging
        //PollutionSensor.doSomething();

//getWindow().setSoftInputMode(WindowManager.LayoutParams.SOFT_INPUT_
STATE_ALWAYS_HIDDEN);

        // Setup the list view
        ListView listView = (ListView)findViewById(R.id.list_view);
        //arrayAdapter = new ArrayAdapter<DeviceWrapper>
(getApplicationContext(), R.layout.list_row, R.id.textview);
        discoveryArrayAdapter = new ArrayAdapter<DeviceWrapper>
(getApplicationContext(), R.layout.list_row, R.id.textview);
        builtInArrayAdapter = new ArrayAdapter<DeviceWrapper>
(getApplicationContext(), R.layout.list_row, R.id.textview);
        listView.setAdapter(builtInArrayAdapter);
        listView.setOnItemClickListener(new ItemClickListener());

        // Setup built in devices array adaptor
        builtInArrayAdapter.add(new DeviceWrapper("00:18:B2:01:27:26",
"Two Way"));
        builtInArrayAdapter.add(new DeviceWrapper("00:18:B2:01:1E:D2",
"Unit 1001"));
        builtInArrayAdapter.add(new DeviceWrapper("00:18:B2:01:25:7B",
"Unit 1002"));
        builtInArrayAdapter.add(new DeviceWrapper("00:18:B2:01:1E:D7",
"Unit 1003"));
        builtInArrayAdapter.add(new DeviceWrapper("00:18:B2:01:1F:56",
"Unit 1004"));
        builtInArrayAdapter.add(new DeviceWrapper("00:18:B2:01:1F:69",
"Unit 1005"));

        // Setup the radio buttons

```

```

        RadioButton discoveryButton =
(RadioButton) findViewById(R.id.RadioButtonDiscovery);
        RadioButton builtInButton =
(RadioButton) findViewById(R.id.RadioButtonBuiltIn);
        discoveryButton.setOnClickListener(new OnClickListener () {
    public void onClick (View v) {
        changeListContent(0);
        startBluetoothDiscovery();
    }
});
        builtInButton.setOnClickListener(new OnClickListener () {
    public void onClick (View v) {
        stopBluetoothDiscovery();
        changeListContent(1);
    }
});
        stopBluetoothDiscovery(); // to put the status message there
        changeListContent(1); // change to built-in by default

// Setup the connect button
        Button button =
(Button) findViewById(R.id.discoveryConnectButton);
        button.setOnClickListener(new OnClickListener() {
    public void onClick (View v) {
        EditText MACField =
(EditText) findViewById(R.id.discoveryMACAddressField);
        String address = MACField.getText().toString();

        if (isValidMACAddress(address)) {
            Intent intent = new Intent();
            intent.putExtra("MACAddress", address.toUpperCase());
            setResult(RESULT_OK, intent);
            finish();
        } else {
            Toast.makeText(getApplicationContext(), "Invalid MAC
address", Toast.LENGTH_SHORT).show();
        }
    }
});

private boolean isValidMACAddress(String address) {
    for (int i = 0; i < address.length(); i++){
        char c = address.charAt(i);
        if ((i+1) % 3 == 0) {

```



```

        if (c != ':') return false;
    } else if (!(c >= '0' && c <= '9' || c >= 'A' && c <= 'F' || c >= 'a'
&& c <= 'f')) {
        return false;
    }
    return true;
}
});
}

```

```

private void startBluetoothDiscovery() {
    // Start bluetooth discovery
    BluetoothAdapter bluetoothAdapter =
BluetoothAdapter.getDefaultAdapter();
    int REQUEST_ENABLE_BT = 500;

    if (bluetoothAdapter != null) {
        if (!bluetoothAdapter.isEnabled()) {
            Intent enableBtIntent = new Intent
(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(enableBtIntent,
REQUEST_ENABLE_BT);
        }

        BroadcastReceiver receiver = new BroadcastReceiver() {
            public void onReceive (Context ctx, Intent intent) {
                if
(intent.getAction().equals(BluetoothAdapter.ACTION_DISCOVERY_FINISHED
)) {

                    System.out.println("Discovery finished!");
                    stopBluetoothDiscovery();
                } else {
                    BluetoothDevice device =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                    if (device != null) {
                        boolean changed = false;
                        if (!discoveredDevices.contains(device)) {
                            discoveredDevices.add(device);
                            DeviceWrapper dw = new
DeviceWrapper(device);
                            getCurrentArrayAdapter().add(dw);

```

```

        System.out.println("Found new device:
"+device);

        changed = true;
    }
    if
(BluetoothDevice.ACTION_NAME_CHANGED.equals(intent.getAction())) {
        changed = true;
    }
    if (changed) {
        getCurrentArrayAdapter().sort(new
Comparator<DeviceWrapper>() {
            public int compare(DeviceWrapper a,
DeviceWrapper b) {
                return
a.toString().compareTo(b.toString());
            }
        });

        getCurrentArrayAdapter().notifyDataSetChanged();
    }
}
};

IntentFilter filter = new
IntentFilter(BluetoothDevice.ACTION_FOUND);
filter.addAction(BluetoothDevice.ACTION_NAME_CHANGED);

filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
registerReceiver(receiver, filter);
bluetoothAdapter.startDiscovery();

TextView statusView =
(TextView) findViewById(R.id.discoveryStatusTextView);
statusView.setText("Searching...");
}

private void stopBluetoothDiscovery () {
    BluetoothAdapter bluetoothAdapter =
BluetoothAdapter.getDefaultAdapter();
    if (bluetoothAdapter != null && bluetoothAdapter.isDiscovering()) {
        bluetoothAdapter.cancelDiscovery();
    }
}

```

```

    }

    TextView statusView =
    (TextView) findViewById(R.id.discoveryStatusTextView);
    statusView.setText (getCurrentArrayAdapter().getCount()+" devices
    found.");
    }

    protected void onStop () {
        super.onStop();
        System.out.println("Deivice chooser onStop");
        BluetoothAdapter bluetoothAdapter =
        BluetoothAdapter.getDefaultAdapter();
        if (bluetoothAdapter != null) {
            bluetoothAdapter.cancelDiscovery();
        }
    }
}

```

## Helpers.java

```

package com.unsw.pollutionsensor;

public class Helpers {
    // converts our custom 16 bit floating point number to a double
    // 12 bit significand, 4 bit exponent
    public static double rawFloat16ToDouble (int rawValue) {
        short significand = (short) ((short) (rawValue & 0xffff) >> 4); //
this will also sign extend
        byte exponent = (byte) ((byte) ((rawValue & 0xf) << 4) >> 4); // shift
left/right by 3 to sign extend
        double v = significand*Math.pow(10, exponent);
        return v;
    }
}

```

## DeviceMessage.java

```

package com.unsw.pollutionsensor;

import java.util.Date;
import java.util.HashMap;
import java.util.Map;

```

```

import android.location.Location;

public class DeviceMessage {
    public boolean controlReceived = false;
    public boolean dataReceived = false;
    public short receivedChecksum;
    public short computedChecksum;
    public int rawReferenceVoltage;
    public int deviceId=0;
    public int valuePairCount = 0;
    public int rawCoefficients[] = new int[3];
    public SensorData sensorData;
    public double sensorCoefficients[] = new double[3];
    public int sensorIdentifier[] = new int[3];
    public int LastCLibDate[] = new int[3]; //DD/MM/YY
    public int NextCLibDate[] = new int[3]; //DD/MM/YY

    public Location location = null;
    public Date timeStamp = null;

    public Map<Integer, double[]>coefficientMap = new
HashMap<Integer, double[]>();

    public double[] getSensorCoes(){
        int index = 0;
        for(int i:rawCoefficients){
            sensorCoefficients[index]=Helpers.rawFloat16ToDouble(i);
            index ++;
        }
        return sensorCoefficients;
    }

    public double getReferenceVoltage () {
        return Helpers.rawFloat16ToDouble (rawReferenceVoltage);
    }

    public double getCoefficient (int index) {
        return Helpers.rawFloat16ToDouble(rawCoefficients[index]);
    }

    //public ArrayList<SensorData> sensors = new ArrayList<SensorData>
    ();
}

```

## SensorData.java

```
package com.unsw.pollutionsensor;

import java.util.HashMap;
import java.util.Map;

public class SensorData {
    public int sensorIdentifier[] = new int[3];
    public short sensorData[] = new short[3];
    public short batteryLevel = 255;
    public int deviceIdentifier=0;
    public Map<Integer, Short> sensorDataMap = new HashMap<Integer,
Short>();
}
```

## UILog.java

```
package com.unsw.pollutionsensor;

import java.util.Formatter;

import android.app.Activity;
import android.widget.TextView;

public class UILog {

    private static TextView logView = null;

    public static final int LOG_TYPE_READINGS = 1 << 0;
    public static final int LOG_TYPE_RAW_DATA = 1 << 1;
    public static final int LOG_TYPE_INFO = 1 << 2;

    public static void printf (int logType, String format, Object... args)
{
    Formatter f = new Formatter();
    String str = f.format(format, args).toString();
    UILog.print(logType, str);
}

    public static void println (int logType, String message) {
        UILog.print(logType, message+"\n");
    }
}
```

```

    public static void print (int logType, String message) {
        Activity activity = (Activity)logView.getContext();
        MainThreadRunner runner = new MainThreadRunner(logView, message,
MainThreadRunner.APPEND_MODE);
        activity.runOnUiThread(runner);
    }

    public static void clear () {
        Activity activity = (Activity)logView.getContext();
        MainThreadRunner runner = new MainThreadRunner(logView, "",
MainThreadRunner.SET_MODE);
        activity.runOnUiThread(runner);
    }

    public static void setLogView (TextView view) {
        logView = view;
        logView.setText("");
    }

    protected static class MainThreadRunner implements Runnable {
        private TextView _logView = null;
        private String _logText;
        private int _mode = APPEND_MODE;

        public static final int APPEND_MODE = 1;
        public static final int SET_MODE = 2;

        public MainThreadRunner (TextView logView, String logText, int
mode) {
            _logView = logView;
            _logText = logText;
            _mode = mode;
        }

        public void run() {
            if (_logView != null && _logText != null) {
                if (_mode == SET_MODE) {
                    _logView.setText(_logText);
                } else {
                    _logView.append(_logText);
                }
            }
        }
    }
}

```

```
}
```

## Tunnels.java

```
package com.unsw.pollutionsensor;

public class Tunnels {
    //entrances E B C H I J K L M N Y Z AA BB EE FF GG JJ KK LL

    public double[][] entries = {
        {-33.935928,151.11119},
        {-33.937861,151.153091},
        {-33.93743,151.152244},
        {-33.871442,151.203166},
        {-33.870498,151.2033},
        {-33.874048,151.202533},
        {-33.87476,151.20331},
        {-33.886796,151.217945},
        {-33.885669,151.218245},
        {-33.874106,151.21755},
        {-33.876464,151.224924},
        {-33.872003,151.217985},
        {-33.876379,151.224945},
        {-33.874979,151.220482},
        {-33.866062,151.213894},
        {-33.841608,151.210719},
        {-33.862972,151.213661},
        {-33.801242,151.144651},
        {-33.812843,151.181483},
        {-33.813346,151.176215}
    };
    //Exits
    public double[][] getExits (int idxEntry){
        switch(idxEntry){
            case 0:
                double[][] Eto = {{-33.934922,151.142153},
{-33.937049,151.152236}, {-33.93777,151.15314}};
                return Eto;
            case 1:
                double[][] Bto = {{-33.936175,151.111364}};
                return Bto;
            case 2:
                double[][] Cto = {{-33.93743,151.152244}};

```

```

        return Cto;
    case 3:
        double[][] Hto = {{-33.873304,151.202994}};
        return Hto;
    case 4:
        double[][] Ito = {{-33.873304,151.202994}};
        return Ito;
    case 5:
        double[][] Jto = {{-33.87613,151.224961},
{-33.887328,151.217974}};
        return Jto;
    case 6:
        double[][] Kto = {{-33.87613,151.224961},
{-33.887328,151.217974}};
        return Kto;
    case 7:
        double[][] Lto = {
            {-33.875983,151.217218}, {-33.872903,151.203686},
            {-33.873057,151.20352}, {-33.872344,151.217727}};
        return Lto;
    case 8:
        double[][] Mto = {{-33.872344,151.217727}};
        return Mto;
    case 9:
        double[][] Nto = {{-33.887328,151.217974}};
        return Nto;
    case 10:
        double[][] Yto = {{-33.872903,151.203686},
{-33.873057,151.20352}, {-33.870667,151.217271}};
        return Yto;
    case 11:
        double[][] Zto = {{-33.887328,151.217974},
{-33.887573,151.219444}, {-33.886838,151.221558}};
        return Zto;
    case 12:
        double[][] AAtto = {{-33.875077,151.220517}};
        return AAtto;
    case 13:
        double[][] BBto = {{-33.876241,151.224956}};
        return BBto;
    case 14:
        double[][] EEto = {{-33.841591,151.210319}};
        return EEto;
    case 15:

```



```

        double[][] FFto = {{-33.865946,151.213991}};
        return FFto;
    case 16:
        double[][] GGto = {{-33.865946,151.213991}};
        return GGto;
    case 17:
        double[][] JJto = {{-33.812132,151.176177},
{-33.812682,151.181206}};
        return JJto;
    case 18:
        double[][] KKto = {{-33.801383,151.144685}};
        return KKto;
    case 19:
        double[][] LLto = {{-33.801383,151.144685}};
        return LLto;
    default:
        return null;
    }
}

public double[][] getTunnel(int idxEntry, int idxExit){
    double[][] route = null;
    switch(idxEntry){
    case 0:
        if(idxExit == 0){
            route = M5WTE1;
        }else if(idxExit == 1){
            route =
computeRoute(computeRoute(M5WTE1,M5WTE2),M5WTE3);
        }else if(idxExit == 2){
            route =
computeRoute(computeRoute(M5WTE1,M5WTE2),M5WTE4);
        }
        return route;
    case 1:
        route = computeRoute(M5ETW1,M5ETW3);
        return route;
    case 2:
        route = computeRoute(M5ETW2,M5ETW3);
        return route;
    case 3:
        route = computeRoute(HQ1,Q2);
        return route;
    case 4:

```

```

    route = computeRoute(IQ1,Q2);
    return route;
case 5:
    if(idxExit == 0){
        route =
computeRoute (computeRoute (computeRoute (computeRoute (JX1, JX2) , NR0) , NR2
),NR3);
    }else if(idxExit == 1){
        route = computeRoute (computeRoute (JX1, JX2) , JX3);
    }
    return route;
case 6:
    if(idxExit == 0){
        route =
computeRoute (computeRoute (computeRoute (computeRoute (KX1, JX2) , NR0) , NR2
),NR3);
    }else if(idxExit == 1){
        route = computeRoute (computeRoute (KX1, JX2) , JX3);
    }
    return route;
case 7:

    if(idxExit == 0){
        route = computeRoute (L1, LU2);
    }else if(idxExit == 1){
        route =
computeRoute (computeRoute (computeRoute (L1, LU2) , UO1) , UO2);
    }else if(idxExit == 2){
        route =
computeRoute (computeRoute (computeRoute (computeRoute (L1, LU2) , UO1) , UO2)
,OP);
    }else if(idxExit == 3){
        route = computeRoute (computeRoute (L1, LW2) , LW3);
    }
    return route;
case 8:
    route = computeRoute (MW1, LW3);
    return route;
case 9:
    route = computeRoute (computeRoute (NR1, NR2) , NR3);
    return route;
case 10:
    if(idxExit == 0){
        route = computeRoute (computeRoute (Y1, YO2) , UO2);

```

```

    }else if(idxExit == 1){
        route =
computeRoute (computeRoute (computeRoute (Y1, YO2), UO2), OP);
    }else if(idxExit == 2){
        route = computeRoute (Y1, YV2);
    }
    return route;
case 11:
    if(idxExit == 0){
        route = computeRoute (computeRoute (Z1, Z3), NR3);
    }else if(idxExit == 1){
        route = computeRoute (computeRoute (Z1, Z2), ZS3);
    }else if(idxExit == 2){
        route = computeRoute (computeRoute (Z1, Z2), ZT3);
    }
    return route;
case 12:
    route = AADD;
    return route;
case 13:
    route = BBCC;
    return route;
case 14:
    route = EEII;
    return route;
case 15:
    route = computeRoute (FFHH1, FFHH2);
    return route;
case 16:
    route = computeRoute (GG1, FFHH2);
    return route;
case 17:
    if(idxExit == 0){
        route = computeRoute (JJNN1, OO2);
    }else if(idxExit == 1){
        route = computeRoute (JJNN1, JJNN2);
    }
    return route;
case 18:
    route = computeRoute (KKMM1, KKMM2);
    return route;
case 19:
    route = computeRoute (LL1, KKMM2);
    return route;

```

```

    default:
        return null;
    }
}

private double[][] computeRoute(double[][] R1, double[][] R2){
    double[][] route = new double[R1.length+R2.length][];
    int i = 0;
    for(double[] temp : R1){
        route[i] = temp;
        i++;
    }
    for(double[] temp : R2){
        route[i] = temp;
        i++;
    }
    return route;
}

//M5
private double[][] M5WTE1 = {
    {-33.935928,151.11119},
    {-33.935917,151.111633},
    {-33.935903,151.11207},
    {-33.935897,151.112512},
    {-33.935897,151.112512},
    {-33.93589,151.112952},
    {-33.935892,151.113392},
    {-33.935899,151.113835},
    {-33.935919,151.114272},
    {-33.935941,151.114714},
    {-33.935963,151.115149},
    {-33.936028,151.116031},
    {-33.936059,151.116461},
    {-33.936095,151.11689},
    {-33.936119,151.117322},
    {-33.936155,151.117748},
    {-33.936188,151.118183},
    {-33.936224,151.118609},
    {-33.936264,151.119038},
    {-33.936295,151.119465},
    {-33.93633,151.119886},
    {-33.936362,151.12031},
    {-33.936388,151.120728},

```

{-33.936419,151.121165},  
{-33.936455,151.121589},  
{-33.936486,151.122018},  
{-33.936517,151.122439},  
{-33.936549,151.122863},  
{-33.936582,151.123295},  
{-33.936604,151.123721},  
{-33.93662,151.124148},  
{-33.936624,151.124574},  
{-33.936624,151.125006},  
{-33.936606,151.125435},  
{-33.936591,151.125864},  
{-33.936564,151.126288},  
{-33.936517,151.126717},  
{-33.936468,151.127141},  
{-33.936404,151.127562},  
{-33.936322,151.127981},  
{-33.936228,151.128394},  
{-33.936137,151.128812},  
{-33.936046,151.129225},  
{-33.935952,151.129646},  
{-33.93587,151.130062},  
{-33.935792,151.13048},  
{-33.935703,151.130894},  
{-33.935612,151.131312},  
{-33.935523,151.131725},  
{-33.935434,151.132143},  
{-33.93534,151.132559},  
{-33.935242,151.132978},  
{-33.935155,151.133399},  
{-33.935075,151.133817},  
{-33.935004,151.134244},  
{-33.934924,151.134665},  
{-33.934864,151.135089},  
{-33.934817,151.135515},  
{-33.93477,151.135941},  
{-33.934735,151.136373},  
{-33.934708,151.136802},  
{-33.93469,151.137232},  
{-33.93467,151.138967},  
{-33.934697,151.139401},  
{-33.934722,151.139831},  
{-33.934746,151.140265},  
{-33.934777,151.1407},

```
        {-33.934826,151.141129},
        {-33.934864,151.141553},
        {-33.934904,151.141984},
        {-33.934922,151.142153},
        {-33.93492,151.142749}
};
```

```
private double[][] M5WTE2 = {
    {-33.934953,151.142411},
    {-33.935006,151.142837},
    {-33.935064,151.143264},
    {-33.935138,151.14369},
    {-33.935189,151.144122},
    {-33.935251,151.144554},
    {-33.935318,151.14498},
    {-33.935382,151.145407},
    {-33.935449,151.145841},
    {-33.935512,151.146273},
    {-33.935567,151.1467},
    {-33.935629,151.147126},
    {-33.935698,151.147553},
    {-33.935765,151.147982},
    {-33.93587,151.148398},
    {-33.93597,151.148821},
    {-33.93603,151.149248},
    {-33.936063,151.149648}
};
```

```
private double[][] M5WTE3 = {
    {-33.936112,151.150053},
    {-33.936233,151.150466},
    {-33.936366,151.150868},
    {-33.936524,151.15126},
    {-33.936707,151.151627},
    {-33.936891,151.151997},
    {-33.937049,151.152236},
    {-33.937301,151.152617}
};
```

```
private double[][] M5WTE4 = {
    {-33.936175,151.150053},
    {-33.93631,151.150452},
    {-33.936444,151.15086},
    {-33.9366,151.151254},
```

```

        {-33.93678,151.151638},
        {-33.936967,151.152005},
        {-33.937176,151.152351},
        {-33.937403,151.152695},
        {-33.93765,151.153014},
        {-33.93777,151.15314},
        {-33.9381,151.153534}
};

```

```

private double[][] M5ETW1 = {
        {-33.93786100, 151.15309100},
        {-33.93773000, 151.15289600},
        {-33.93749400, 151.15257100},
        {-33.93728100, 151.15222000},
        {-33.93709400, 151.15184700},
        {-33.93690900, 151.15147100},
        {-33.93674700, 151.15108300},
        {-33.93659100, 151.15069400},
        {-33.93646800, 151.15028300}
};

```

```

private double[][] M5ETW2 = {
        {-33.93743000, 151.15224400},
        {-33.93737600, 151.15214700},
        {-33.93719400, 151.15177200},
        {-33.93701100, 151.15140200},
        {-33.93683300, 151.15102600},
        {-33.93667800, 151.15063500},
        {-33.93651300, 151.15024800}
};

```

```

private double[][] M5ETW3 = {
        {-33.93634800, 151.14987600},
        {-33.93624800, 151.14946500},
        {-33.93615900, 151.14904900},
        {-33.93607200, 151.14863100},
        {-33.93599400, 151.14820500},
        {-33.93593700, 151.14777800},
        {-33.93587400, 151.14734600},
        {-33.93581200, 151.14692000},
        {-33.93575000, 151.14649100},
        {-33.93569400, 151.14606700},
};

```

{-33.93563800, 151.14563500},  
{-33.93557800, 151.14520800},  
{-33.93551400, 151.14478700},  
{-33.93545100, 151.14435600},  
{-33.93539100, 151.14393200},  
{-33.93533100, 151.14350500},  
{-33.93527300, 151.14307600},  
{-33.93521800, 151.14264400},  
{-33.93516900, 151.14221200},  
{-33.93511500, 151.14178600},  
{-33.93506200, 151.14135100},  
{-33.93501300, 151.14092000},  
{-33.93498900, 151.14048500},  
{-33.93491300, 151.13873900},  
{-33.93491100, 151.13830700},  
{-33.93490800, 151.13787000},  
{-33.93492400, 151.13743000},  
{-33.93495100, 151.13699800},  
{-33.93497700, 151.13656400},  
{-33.93501100, 151.13613200},  
{-33.93505300, 151.13570500},  
{-33.93511800, 151.13526800},  
{-33.93517300, 151.13483900},  
{-33.93524400, 151.13442100},  
{-33.93532200, 151.13400200},  
{-33.93540000, 151.13357600},  
{-33.93548000, 151.13315500},  
{-33.93557600, 151.13273400},  
{-33.93566500, 151.13231500},  
{-33.93576700, 151.13189700},  
{-33.93585200, 151.13147800},  
{-33.93595000, 151.13105700},  
{-33.93603900, 151.13063300},  
{-33.93613500, 151.13021800},  
{-33.93622100, 151.12979100},  
{-33.93630800, 151.12936500},  
{-33.93639900, 151.12894600},  
{-33.93649100, 151.12852800},  
{-33.93658200, 151.12810700},  
{-33.93664600, 151.12767800},  
{-33.93669500, 151.12724300},  
{-33.93674700, 151.12680900},  
{-33.93679600, 151.12638200},  
{-33.93682200, 151.12594500},



```
{-33.93685600, 151.12551000},
{-33.93685800, 151.12507800},
{-33.93686500, 151.12464400},
{-33.93685600, 151.12421500},
{-33.93684500, 151.12378300},
{-33.93681800, 151.12334000},
{-33.93678700, 151.12290900},
{-33.93675100, 151.12247700},
{-33.93672200, 151.12204200},
{-33.93668400, 151.12160800},
{-33.93664600, 151.12117600},
{-33.93661800, 151.12074700},
{-33.93658200, 151.12031800},
{-33.93655300, 151.11988300},
{-33.93651700, 151.11945400},
{-33.93647700, 151.11902500},
{-33.93643700, 151.11859300},
{-33.93640400, 151.11815600},
{-33.93637100, 151.11772400},
{-33.93633300, 151.11728900},
{-33.93629900, 151.11685800},
{-33.93625900, 151.11642600},
{-33.93622800, 151.11599400},
{-33.93620800, 151.11556200},
{-33.93618100, 151.11513000},
{-33.93615500, 151.11469800},
{-33.93613200, 151.11396100},
{-33.93613200, 151.11396100},
{-33.93612100, 151.11353200},
{-33.93612600, 151.11310200},
{-33.93613700, 151.11266800},
{-33.93614400, 151.11223600},
{-33.93615200, 151.11179900},
{-33.93617500, 151.11136400},
{-33.936215,151.110753}
```

```
};
```

```
//cross city & easter distributer
```

```
private double[][] L1 = {
    {-33.88679600, 151.21794500},
    {-33.88644200, 151.21797700},
    {-33.88607900, 151.21798200},
    {-33.88569600, 151.21796300},
```

```

        {-33.88537300, 151.21788600},
        {-33.88501700, 151.21776500},
        {-33.88469000, 151.21765800},
        {-33.88435300, 151.21754000},
        {-33.88398600, 151.21743000},
        {-33.88362100, 151.21736300},
        {-33.88327800, 151.21729000},
        {-33.88285700, 151.21721000}
    };

    private double[][] LU2 = {
        {-33.88257900, 151.21707600},
        {-33.88219800, 151.21699800},
        {-33.88188000, 151.21695000},
        {-33.88152100, 151.21689600},
        {-33.88111800, 151.21686400},
        {-33.88067500, 151.21683200},
        {-33.88028700, 151.21684200},
        {-33.87986200, 151.21686100},
        {-33.87948100, 151.21687200},
        {-33.87907600, 151.21690100},
        {-33.87864000, 151.21695200},
        {-33.87825000, 151.21699500},
        {-33.87792700, 151.21703500},
        {-33.87753300, 151.21709200},
        {-33.87715700, 151.21715600},
        {-33.87680000, 151.21719600},
        {-33.87636400, 151.21723400},
        {-33.87598300, 151.21721800},
        {-33.875464, 151.217148}
    };

    private double[][] UO1 = {
        {-33.87555800, 151.21733300},
        {-33.87512800, 151.21745100},
        {-33.87485800, 151.21722300},
        {-33.87481400, 151.21677200},
        {-33.87478900, 151.21636500},
        {-33.87471600, 151.21586600},
        {-33.87461800, 151.21541500},
        {-33.87446600, 151.21499700},
        {-33.87433100, 151.21490000}
    };

    private double[][] UO2 = {
        {-33.87424400, 151.21439300},

```

```

    {-33.87416100, 151.21393500},
    {-33.87407200, 151.21348700},
    {-33.87397900, 151.21300400},
    {-33.87389400, 151.21256400},
    {-33.87380500, 151.21206300},
    {-33.87371600, 151.21159900},
    {-33.87363800, 151.21103300},
    {-33.87354400, 151.21046100},
    {-33.87346900, 151.20997300},
    {-33.87339100, 151.20949300},
    {-33.87330200, 151.20902400},
    {-33.87318600, 151.20854300},
    {-33.87305000, 151.20803400},
    {-33.87293200, 151.20758900},
    {-33.87285600, 151.20713300},
    {-33.87278500, 151.20665800},
    {-33.87273400, 151.20615900},
    {-33.87272500, 151.20571400},
    {-33.87275400, 151.20520400},
    {-33.87278300, 151.20476700},
    {-33.87282300, 151.20427600},
    {-33.87290300, 151.20368600},
    {-33.873184,151.203262}
};

private double[][] OP = {
    {-33.873057,151.20352},
    {-33.873391,151.203214}
};

private double[][] LW2 = {
    {-33.882735,151.21718}
};

private double[][] LW3 = {
    {-33.88238900, 151.21710000},
    {-33.88201300, 151.21703800},
    {-33.88166600, 151.21699500},
    {-33.88129800, 151.21696000},
    {-33.88093800, 151.21693100},
    {-33.88057700, 151.21691200},
    {-33.88016300, 151.21690700},
    {-33.87973500, 151.21691700},
    {-33.87932100, 151.21700000},
    {-33.87899600, 151.21703500},
    {-33.87864900, 151.21707600},
    {-33.87823000, 151.21713200},

```

```

        {-33.87783100, 151.21718600},
        {-33.87744400, 151.21723100},
        {-33.87703000, 151.21728500},
        {-33.87662700, 151.21732500},
        {-33.87623500, 151.21737300},
        {-33.87579800, 151.21743200},
        {-33.87537900, 151.21748100},
        {-33.87499000, 151.21752400},
        {-33.87456900, 151.21757200},
        {-33.87413500, 151.21762000},
        {-33.87374900, 151.21765000},
        {-33.87335100, 151.21766600},
        {-33.87291600, 151.21768200},
        {-33.87257100, 151.21770600},
        {-33.87234400, 151.21772700},
        {-33.871856, 151.217824}
    };

    private double[][] HQ1 = {
        {-33.87144200, 151.20316600},
        {-33.87172900, 151.20316600},
        {-33.87205500, 151.20317600},
        {-33.87238900, 151.20319500},
        {-33.87258900, 151.20320100}
    };

    private double[][] IQ1 = {
        {-33.87049800, 151.20330000},
        {-33.87078300, 151.20330200},
        {-33.87105500, 151.20331900},
        {-33.87133800, 151.20334300},
        {-33.87161100, 151.20339900},
        {-33.87192100, 151.20348800},
        {-33.87221100, 151.20353600},
        {-33.87250200, 151.20346100}
    };

    };

    private double[][] Q2 = {
        {-33.87283400, 151.20326000},
        {-33.87303900, 151.20318700},
        {-33.87330400, 151.20299400},
        {-33.873607, 151.202739}
    };

    };

    private double[][] JX1 = {
        {-33.87404800, 151.20253300},
        {-33.87407900, 151.20300200},

```

```

        {-33.87411200, 151.20352000}
    };

    private double[][] KX1 = {
        {-33.87476000, 151.20331000},
        {-33.87453300, 151.20302900},
        {-33.87431500, 151.20270400},
        {-33.87402600, 151.20257800},
        {-33.87391600, 151.20296700},
        {-33.87399200, 151.20344500}

    };

    private double[][] JX2 = {
        {-33.87414600, 151.20395200},
        {-33.87422400, 151.20432700},
        {-33.87429700, 151.20475900},
        {-33.87436400, 151.20514800},
        {-33.87442000, 151.20563600},
        {-33.87447500, 151.20610300},
        {-33.87452700, 151.20654500},
        {-33.87454900, 151.20707400},
        {-33.87456700, 151.20749700},
        {-33.87457800, 151.20792900},
        {-33.87454900, 151.20841700},
        {-33.87452400, 151.20884100},
        {-33.87447500, 151.20923300},
        {-33.87436400, 151.20968100},
        {-33.87422800, 151.21016400},
        {-33.87407200, 151.21063600},
        {-33.87393600, 151.21107300},
        {-33.87380900, 151.21153100},
        {-33.87362700, 151.21196600},
        {-33.87346700, 151.21230900},
        {-33.87352900, 151.21272500},
        {-33.87359300, 151.21317600},
        {-33.87366700, 151.21367200},
        {-33.87374700, 151.21414900},
        {-33.87382500, 151.21464300},
        {-33.87388100, 151.21509100},
        {-33.87396100, 151.21560000},
        {-33.87403200, 151.21605900},
        {-33.87410800, 151.21650400},
        {-33.87418100, 151.21697600},
        {-33.87425900, 151.21740300},
        {-33.87433500, 151.21787200}
    };

```

```

};
private double[][] JX3 = {
    {-33.87441700, 151.21831500},
    {-33.87451300, 151.21884300},
    {-33.87460900, 151.21934500},
    {-33.87470700, 151.21986200},
    {-33.87479600, 151.22034800},
    {-33.87490500, 151.22087100},
    {-33.87500300, 151.22131300},
    {-33.87515700, 151.22175300},
    {-33.87534200, 151.22213700},
    {-33.87551800, 151.22245100},
    {-33.87563800, 151.22283200},
    {-33.87573600, 151.22328000},
    {-33.87580000, 151.22376000},
    {-33.87588500, 151.22420000},
    {-33.87602800, 151.22456200},
    {-33.87613000, 151.22496100},
    {-33.876219, 151.225519}
};

private double[][] MW1 = {
    {-33.88566900, 151.21824500},
    {-33.88529100, 151.21811400},
    {-33.88488100, 151.21797900},
    {-33.88458900, 151.21787500},
    {-33.88418200, 151.21771900},
    {-33.88372300, 151.21757700},
    {-33.88335800, 151.21742400},
    {-33.88302400, 151.21729300}

};

private double[][] Y1 = {
    {-33.87646400, 151.22492400},
    {-33.87641300, 151.22437900},
    {-33.87635700, 151.22378900},
    {-33.87629500, 151.22322100},
    {-33.87617700, 151.22264700},
    {-33.87602800, 151.22207800},
    {-33.87589800, 151.22166200},
    {-33.87570200, 151.22120900},
    {-33.87549500, 151.22075300},
    {-33.87525300, 151.22027300},
    {-33.87510300, 151.21979000},
    {-33.87500800, 151.21933400},

```

```

        {-33.87493000, 151.21880800}
    };
    private double[][] YV2 = {
        {-33.87499000, 151.21822400},
        {-33.87506100, 151.21775200},
        {-33.87505000, 151.21733300},
        {-33.87503000, 151.21689900},
        {-33.87498500, 151.21639700},
        {-33.87494300, 151.21593600},
        {-33.87477400, 151.21563800},
        {-33.87445500, 151.21551200},
        {-33.87410300, 151.21553100},
        {-33.87374500, 151.21561100},
        {-33.87334900, 151.21570000},
        {-33.87294800, 151.21579100},
        {-33.87254700, 151.21588500},
        {-33.87216200, 151.21598900},
        {-33.87179000, 151.21610700},
        {-33.87148500, 151.21641600},
        {-33.87120200, 151.21674000},
        {-33.87089000, 151.21704900},
        {-33.87066700, 151.21727100},
        {-33.870331, 151.217679},
    };
    private double[][] YO2 = {
        {-33.87484500, 151.21824200},
        {-33.87469100, 151.21724200},
        {-33.87462200, 151.21679700},
        {-33.87454200, 151.21627600},
        {-33.87446900, 151.21578800},
        {-33.87438800, 151.21527000},
        {-33.87433700, 151.21494600}
    };
    private double[][] Z1 = {
        {-33.87200300, 151.21798500},
        {-33.87234400, 151.21792600},
        {-33.87276300, 151.21785100},
        {-33.87320800, 151.21780800},
        {-33.87352700, 151.21778400},
        {-33.87392500, 151.21774600},
        {-33.87434400, 151.21769800},
        {-33.87469600, 151.21765800},
        {-33.87503400, 151.21762000},
        {-33.87545300, 151.21756900},
    };

```

```

    {-33.87586000, 151.21751800},
    {-33.87630100, 151.21746200},
    {-33.87666200, 151.21741900},
    {-33.87707600, 151.21736500},
    {-33.87747700, 151.21731200},
    {-33.87784700, 151.21726300},
    {-33.87821200, 151.21721000},
    {-33.87862000, 151.21716100},
    {-33.87901100, 151.21710200},
    {-33.87931400, 151.21705400},
    {-33.87969500, 151.21703300}

};

private double[][] Z2 = {
    {-33.87995100, 151.21708900},
    {-33.88027600, 151.21708900},
    {-33.88065300, 151.21709700},
    {-33.88102700, 151.21711000},
    {-33.88137200, 151.21718800},
    {-33.88180400, 151.21725500},
    {-33.88216500, 151.21731700},
    {-33.88257400, 151.21740000},
    {-33.88298000, 151.21749700},
    {-33.88336900, 151.21761700},
    {-33.88377000, 151.21777300},
    {-33.88412200, 151.21791200},
    {-33.88450700, 151.21804400},
    {-33.88489200, 151.21806500}

};

private double[][] ZT3 = {
    {-33.88523100, 151.21824200},
    {-33.88558300, 151.21840300},
    {-33.88591700, 151.21855600},
    {-33.88622800, 151.21869800},
    {-33.88648900, 151.21891300},
    {-33.88661100, 151.21935500},
    {-33.88668300, 151.21983300},
    {-33.88671100, 151.22031300},
    {-33.88674500, 151.22078500},
    {-33.88678300, 151.22117100},
    {-33.88683800, 151.22155800},
    {-33.886932, 151.222199}

};

```



```

private double[][] ZS3 = {
    {-33.88521500, 151.21812200},
    {-33.88559800, 151.21826400},
    {-33.88591200, 151.21840100},
    {-33.88622800, 151.21858800},
    {-33.88650700, 151.21883200},
    {-33.88685800, 151.21907400},
    {-33.88719900, 151.21924800},
    {-33.88757300, 151.21944400},
    {-33.888005,151.219645}

};

private double[][] Z3 = {
    {-33.880256,151.21703}
};

private double[][] NR0 = {
    {-33.87433500, 151.21788000},
    {-33.87405400, 151.21793700},
    {-33.87403000, 151.21748300},
    {-33.87436400, 151.21734100}

};

private double[][] NR1 = {
    {-33.87410600, 151.21755000},
    {-33.87440900, 151.21740300}

};

private double[][] NR2 = {
    {-33.87466500, 151.21729000},
    {-33.87506500, 151.21725300},
    {-33.87541500, 151.21725000},
    {-33.87579600, 151.21730900},
    {-33.87612800, 151.21730100},
    {-33.87652900, 151.21727400},
    {-33.87694300, 151.21722800},
    {-33.87727000, 151.21718000},
    {-33.87758600, 151.21712900},
    {-33.87786200, 151.21708900},
    {-33.87820300, 151.21705700},
    {-33.87853500, 151.21701100},
    {-33.87887300, 151.21697900},
    {-33.87919400, 151.21695800},
    {-33.87949700, 151.21696600},
    {-33.87982000, 151.21698200},

```

```

        {-33.88012900, 151.21699500}

};

private double[][] NR3 = {
    {-33.88054600, 151.21703000},
    {-33.88087800, 151.21705400},
    {-33.88126500, 151.21707300},
    {-33.88161700, 151.21712100},
    {-33.88196200, 151.21718600},
    {-33.88228000, 151.21725000},
    {-33.88262600, 151.21732000},
    {-33.88298800, 151.21737100},
    {-33.88334300, 151.21738900},
    {-33.88368800, 151.21742200},
    {-33.88401700, 151.21749400},
    {-33.88434000, 151.21760900},
    {-33.88464700, 151.21771700},
    {-33.88498800, 151.21783500},
    {-33.88530400, 151.21793900},
    {-33.88563200, 151.21801400},
    {-33.88599700, 151.21804100},
    {-33.88632600, 151.21803800},
    {-33.88669400, 151.21803000},
    {-33.88699200, 151.21800400},
    {-33.88732800, 151.21797400},
    {-33.887802, 151.217902}
};

private double[][] AADD = {
    {-33.87637900, 151.22494500},
    {-33.87631300, 151.22435500},
    {-33.87625200, 151.22386200},
    {-33.87617900, 151.22326100},
    {-33.87604100, 151.22271600},
    {-33.87583800, 151.22225000},
    {-33.87561800, 151.22189000},
    {-33.87539100, 151.22144000},
    {-33.87521900, 151.22096500},
    {-33.87507700, 151.22051700},
    {-33.874932, 151.219865}
};

private double[][] BBCC = {
    {-33.87497900, 151.22048200},
    {-33.87510600, 151.22098900},

```

```

    {-33.87526100, 151.22141800},
    {-33.87548400, 151.22182600},
    {-33.87568000, 151.22220700},
    {-33.87587600, 151.22268700},
    {-33.87599900, 151.22310500},
    {-33.87605900, 151.22362600},
    {-33.87612500, 151.22416500},
    {-33.87624100, 151.22495600},
    {-33.876306, 151.225538}
};
//harbour tunnel
private double[][] FFHH1 = {
    {-33.84160800, 151.21071900},
    {-33.84203400, 151.21083100},
    {-33.84246800, 151.21093100},
    {-33.84287800, 151.21104300},
    {-33.84334200, 151.21122300},
    {-33.84381200, 151.21146400},
    {-33.84426800, 151.21164100},
    {-33.84473200, 151.21184000},
    {-33.84518800, 151.21205200},
    {-33.84566100, 151.21226900},
    {-33.84607100, 151.21245700},
    {-33.84648300, 151.21259600},
    {-33.84681000, 151.21270600},
    {-33.84719300, 151.21285100},
    {-33.84756800, 151.21300100},
    {-33.84796200, 151.21317000},
    {-33.84835800, 151.21329600},
    {-33.84877500, 151.21336300},
    {-33.84920500, 151.21341700},
    {-33.84958600, 151.21344700},
    {-33.84996700, 151.21349200},
    {-33.85036800, 151.21349800},
    {-33.85071100, 151.21350600},
    {-33.85107200, 151.21350600},
    {-33.85152100, 151.21351100},
    {-33.85189800, 151.21351400},
    {-33.85228100, 151.21351900},
    {-33.85264600, 151.21352400},
    {-33.85306500, 151.21352700},
    {-33.85345100, 151.21353200},
    {-33.85383800, 151.21354800},
    {-33.85425700, 151.21357300},

```

```

    {-33.85470900, 151.21358900},
    {-33.85514600, 151.21360700},
    {-33.85553500, 151.21362900},
    {-33.85592100, 151.21364500},
    {-33.85627900, 151.21365600},
    {-33.85670300, 151.21367700},
    {-33.85708300, 151.21369600},
    {-33.85742200, 151.21370700},
    {-33.85785400, 151.21372300},
    {-33.85824200, 151.21374400},
    {-33.85866300, 151.21379500},
    {-33.85904800, 151.21384900},
    {-33.85939800, 151.21387300},
    {-33.85980300, 151.21389400},
    {-33.86016600, 151.21389200},
    {-33.86053800, 151.21387300},
    {-33.86091200, 151.21385200},
    {-33.86131100, 151.21381900},
    {-33.86172100, 151.21379000},
    {-33.86207900, 151.21380600},
    {-33.86245600, 151.21378200},
    {-33.86292800, 151.21360700},
    {-33.86329300, 151.21343900}

};

private double[][] GG1 = {
    {-33.86297200, 151.21366100},
    {-33.86332000, 151.21349200}
};

private double[][] FFHH2 = {
    {-33.86373400, 151.21327200},
    {-33.86411100, 151.21313000},
    {-33.86452000, 151.21308200},
    {-33.86491000, 151.21315700},
    {-33.86526900, 151.21334200},
    {-33.86560900, 151.21358900},
    {-33.86594600, 151.21399100},
    {-33.866249, 151.214557}

};

private double[][] EEII = {
    {-33.86606200, 151.21389400},
    {-33.86576100, 151.21350800},
    {-33.86536000, 151.21321100},

```

{-33.86493500, 151.21300400},  
{-33.86442200, 151.21296100},  
{-33.86396800, 151.21306600},  
{-33.86355400, 151.21324300},  
{-33.86309900, 151.21346800},  
{-33.86260500, 151.21366400},  
{-33.86210400, 151.21371700},  
{-33.86162900, 151.21371200},  
{-33.86112400, 151.21371700},  
{-33.86057800, 151.21372600},  
{-33.86001700, 151.21372600},  
{-33.85950700, 151.21372000},  
{-33.85902100, 151.21369100},  
{-33.85851600, 151.21362600},  
{-33.85800100, 151.21357800},  
{-33.85746700, 151.21354600},  
{-33.85697900, 151.21351600},  
{-33.85644900, 151.21347600},  
{-33.85595900, 151.21346300},  
{-33.85545700, 151.21344400},  
{-33.85490100, 151.21342500},  
{-33.85437700, 151.21340400},  
{-33.85386700, 151.21338000},  
{-33.85335700, 151.21336300},  
{-33.85282500, 151.21334700},  
{-33.85232100, 151.21332600},  
{-33.85176000, 151.21329600},  
{-33.85126300, 151.21328000},  
{-33.85073700, 151.21325100},  
{-33.85023800, 151.21322900},  
{-33.84977300, 151.21320500},  
{-33.84927800, 151.21315700},  
{-33.84879500, 151.21310600},  
{-33.84831200, 151.21302800},  
{-33.84785700, 151.21289700},  
{-33.84745400, 151.21270900},  
{-33.84700400, 151.21254800},  
{-33.84657400, 151.21239800},  
{-33.84608800, 151.21222600},  
{-33.84559200, 151.21205400},  
{-33.84509000, 151.21186900},  
{-33.84466900, 151.21166000},  
{-33.84421700, 151.21144000},  
{-33.84373100, 151.21121800},

```

    {-33.84322600, 151.21103500},
    {-33.84272000, 151.21085300},
    {-33.84227200, 151.21067000},
    {-33.84189400, 151.21046700},
    {-33.84159100, 151.21031900},
    {-33.841107, 151.210185}

};
//lane cove
private double[][] JJNN1 = {
    {-33.80124200, 151.14465100},
    {-33.80156600, 151.14510400},
    {-33.80185800, 151.14550600},
    {-33.80215200, 151.14586000},
    {-33.80248600, 151.14626300},
    {-33.80281200, 151.14665700},
    {-33.80315300, 151.14709400},
    {-33.80344000, 151.14749400},
    {-33.80374300, 151.14795000},
    {-33.80404000, 151.14843800},
    {-33.80434500, 151.14900400},
    {-33.80459500, 151.14953800},
    {-33.80482600, 151.15005800},
    {-33.80504300, 151.15058400},
    {-33.80525900, 151.15110700},
    {-33.80546200, 151.15166500},
    {-33.80566000, 151.15215500},
    {-33.80590300, 151.15264400},
    {-33.80617300, 151.15308300},
    {-33.80647100, 151.15352100},
    {-33.80679400, 151.15398500},
    {-33.80708000, 151.15435200},
    {-33.80739600, 151.15477900},
    {-33.80766600, 151.15516800},
    {-33.80793500, 151.15555900},
    {-33.80819800, 151.15599600},
    {-33.80847000, 151.15647900},
    {-33.80870000, 151.15694000},
    {-33.80893600, 151.15744200},
    {-33.80916300, 151.15797000},
    {-33.80937500, 151.15853400},
    {-33.80958200, 151.15915600},
    {-33.80973200, 151.15968400},

```

```

    {-33.80988100, 151.16028500},
    {-33.81000800, 151.16086700},
    {-33.81011900, 151.16140900},
    {-33.81024000, 151.16205000},
    {-33.81035300, 151.16262700},
    {-33.81045600, 151.16325700},
    {-33.81054500, 151.16383400},
    {-33.81063000, 151.16438400},
    {-33.81071700, 151.16499000},
    {-33.81079900, 151.16551000},
    {-33.81089700, 151.16603900},
    {-33.81102700, 151.16662600},
    {-33.81114000, 151.16715400},
    {-33.81130300, 151.16765900},
    {-33.81148600, 151.16827300},
    {-33.81166600, 151.16884100},
    {-33.81182000, 151.16932200}
};

private double[][] OO2 = {
    {-33.81192900, 151.16991400},
    {-33.81203400, 151.17047200},
    {-33.81214100, 151.17101700},
    {-33.81218100, 151.17159900},
    {-33.81220100, 151.17214600},
    {-33.81223000, 151.17277900},
    {-33.81225000, 151.17338500},
    {-33.81227200, 151.17399900},
    {-33.81229500, 151.17461400},
    {-33.81224100, 151.17519300},
    {-33.81218300, 151.17574000},
    {-33.81213200, 151.17617700}
};

private double[][] JJNN2 = {
    {-33.81201400, 151.17004000},
    {-33.81211600, 151.17059800},
    {-33.81219900, 151.17125000},
    {-33.81224100, 151.17182700},
    {-33.81227900, 151.17252400},
    {-33.81233000, 151.17319500},
    {-33.81237000, 151.17375800},
    {-33.81237700, 151.17431300},
    {-33.81235900, 151.17495400},

```

```

    {-33.81228600, 151.17553600},
    {-33.81223400, 151.17609900},
    {-33.81219000, 151.17667900},
    {-33.81215900, 151.17729600},
    {-33.81216500, 151.17794200},
    {-33.81219200, 151.17848900},
    {-33.81225200, 151.17908700},
    {-33.81233500, 151.17962900},
    {-33.81244800, 151.18020300},
    {-33.81256600, 151.18070500},
    {-33.81268200, 151.18120600},
    {-33.812845, 151.181845}
};

private double[][] KKMM1 = {
    {-33.81284300, 151.18148300},
    {-33.81271100, 151.18086600},
    {-33.81258900, 151.18025400},
    {-33.81249300, 151.17969400},
    {-33.81241700, 151.17912000},
    {-33.81233000, 151.17848100},
    {-33.81233500, 151.17786200},
    {-33.81235000, 151.17723100},
    {-33.81241300, 151.17649900},
    {-33.81248200, 151.17592800},
    {-33.81257100, 151.17533000},
    {-33.81262400, 151.17480400},
    {-33.81265300, 151.17424100},
    {-33.81266200, 151.17363200},
    {-33.81265100, 151.17300700},
    {-33.81260700, 151.17249500},
    {-33.81252000, 151.17189600}
};

private double[][] LL1 = {
    {-33.81334600, 151.17621500},
    {-33.81293200, 151.17595700},
    {-33.81256900, 151.17624400},
    {-33.81240800, 151.17678300},
    {-33.81267100, 151.17715100},
    {-33.81306300, 151.17705200},
    {-33.81331700, 151.17673000},
    {-33.81345600, 151.17622000},
    {-33.81339800, 151.17565700},
    {-33.81320800, 151.17509900},

```



```

        {-33.81299900, 151.17443400},
        {-33.81288100, 151.17387100},
        {-33.81281800, 151.17327200},
        {-33.81275600, 151.17268200},
        {-33.81263600, 151.17202800}
    };

    private double[][] KKMM2 = {
        {-33.81241500, 151.17129600},
        {-33.81230800, 151.17070300},
        {-33.81218300, 151.17009900},
        {-33.81201600, 151.16949900},
        {-33.81184200, 151.16888700},
        {-33.81164200, 151.16816300},
        {-33.81147400, 151.16743900},
        {-33.81138800, 151.16682400},
        {-33.81124900, 151.16613500},
        {-33.81112000, 151.16551500},
        {-33.81099300, 151.16489100},
        {-33.81085500, 151.16423900},
        {-33.81075200, 151.16359200},
        {-33.81065700, 151.16295100},
        {-33.81055900, 151.16236700},
        {-33.81044900, 151.16171700},
        {-33.81034500, 151.16107400},
        {-33.81022400, 151.16041900},
        {-33.81006800, 151.15973500},
        {-33.80984300, 151.15906700},
        {-33.80955300, 151.15842100},
        {-33.80933100, 151.15781200},
        {-33.80908300, 151.15721400},
        {-33.80880200, 151.15661900},
        {-33.80850800, 151.15605500},
        {-33.80819400, 151.15550800},
        {-33.80782800, 151.15495300},
        {-33.80751400, 151.15446500},
        {-33.80715300, 151.15395200},
        {-33.80679000, 151.15347200},
        {-33.80645100, 151.15296000},
        {-33.80615700, 151.15243700},
        {-33.80586700, 151.15186000},
        {-33.80562000, 151.15127800},
        {-33.80537700, 151.15067500},
        {-33.80514700, 151.15009500},
        {-33.80490400, 151.14954600},
    };

```

```
{-33.80462600, 151.14899600},  
{-33.80433600, 151.14845700},  
{-33.80403700, 151.14794200},  
{-33.80370100, 151.14743700},  
{-33.80336400, 151.14695500},  
{-33.80295600, 151.14645800},  
{-33.80253100, 151.14597600},  
{-33.80210500, 151.14550900},  
{-33.80173900, 151.14510700},  
{-33.80138300, 151.14468500},  
{-33.801024, 151.144165}
```

```
};
```

```
}
```