SCHOOL OF ELECTRICAL ENGINEERING AND TELECOMMUNICATIONS

Communication between Sensors and Mobile Phone

Thesis B Report

Supervisor: A/Prof. Vijay Sivaraman

Author: Jiang Junjie (3297513)
Bachelor of Telecommunication Engineering

2011-10-20

UNSW
THE UNIVERSITY OF NEW SOUTH WALES

Thesis title: Communication between Sensors and Mobile Phone **Topic number:** VR36

Student Name: Junjie Jiang **Student ID:** 3297513

A. **Problem statement**

Medical researches have indicated some relationships between air pollutions and diseases including respiratory allergies, emphysema and asthma. Current air pollution monitory systems are stationary and only low-resolution pollution map is provided. The Haze Watch system is aiming to measure air pollutants in Sydney while provide accurate high-resolution pollution data with mobile sensor units. The prototype system has been developed and operated, but it still needs improvements on problems including, no pollution readings when GPS lost, inefficient protocol and one-way communication between device and mobile phone.

B. **Objective**

Estimate the user's routes when GPS signal lost
Improve the protocol architecture
Establish the duplex communication between wireless sensor board and mobile phone
Enhance the data uploading method

C. **My solution**

Develop linear and nonlinear interpolation method for tunnels
Separate constant fields and varying fields in protocol version 1
Connect RX/DT pin (pin 12) on microcontroller and UART_tx pin (pin 5) on BT module
Agreed configuration characters transmitted for data acquisition
Buffer and timeout mechanism when data uploading

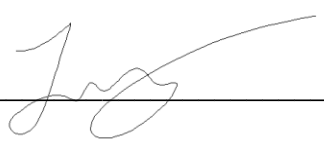
D. **Contributions** (at most one per line, most important first)

Achieved the duplex communication between wireless sensor board and mobile phone
Developed displacement-based data acquisition method
Location interpolation method
Enhancements on data uploading method
Field tests
Provide software assistance for sensor calibration

E. **Suggestions for future work**

Efficient power management
Monitoring modules on board by microcontroller
New protocol based on two-way communication

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

Signature:  Date: 20/ 10/ 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!

7	Problem Statement
8	Objective

Theory (up to 5 most relevant ideas)

18-25	Location interpolation
26	Protocol
31,32	USART interface
38	Buffer
40	TCP connection timeout

Method of solution (up to 5 most relevant points)

18-25	Linear and nonlinear interpolation
21	Heversine formula
28-31	Separate constant fields and varying fields in protocol
35	Displacement-based data acquisition
40	Buffer resize

Contributions (most important first)

30-34	Achieved the duplex communication between device and mobile phone
35	Developed displacement-based data acquisition method
17-22	Location interpolation method
38-40	Enhancements on data uploading method
22-25,35-37	Field tests

My work

19-21,29-34, 39-40	System block diagrams/algorithms/equations solved
19-21,29-34, 39-40	Description of procedure (e.g. for experiments)

Results

23-25,35-37	Succinct presentation of results
23-25,35-37	Analysis
23-25,35-37	Significance of results

Conclusion

43	Statement of whether the outcomes met the objectives
41-42	Suggestions for future research

Literature: (up to 5 most important references)

11,12,13,14,16,26,35	[1] N. Youdale
27,32	[19] Microchip Technology Inc
28,33,41	[20] Adeunis RF.
7,12,13,17	[2] J. Carrapetta
9	[5] World Health Organization

Abstract

The Haze Watch project aims to build a mobile air pollution monitoring system which produces a high-resolution image of the air quality in a large region as well as the accurate pollution exposure for individuals. The system consists of wireless sensor device, Android Smartphone, database server and clients applications. Based on the cooperation between device and Smartphone the air pollution are measured and uploaded to database server for computing a visualized pollution level.

Currently, the whole project is divided and under improvement from different parts. In this article, the enhancements on communication between wireless sensor device and Android Smartphone will be exhibited.

Acknowledgements

I would like to acknowledge my supervisor, A/prof. Vijay Sivaraman who has provided me with excellent guidance, direction and support throughout my thesis. I would also like to thank my group partner, Hailian Zhang, who has completed all the works in this thesis with me. I would like to further thank other project members, Dawei Lu, Kunxuan Bi, James Carrapetta, and Nikolaus Youdale for their contributions on this project.

Table of Contents

1 Introduction

Air pollution in the progress of urbanization is increasing with burgeoning population and increased traffic and industry. Simply consider the stale or moldy smell in the air or the black exhausts of automobiles. The public health has been threatened but unknown for major publics. Medical researches have indicated some relationship between air pollution and diseases including respiratory allergies in some extents, but still making best efforts for seeking conclusive evidences. Consequently, a robust air quality monitoring system with abilities to provide accurate high-resolution pollution data at anywhere within urban area is advised.

Haze Watch system is a mobile air pollution monitoring system designed and developed by James Carrapetta [1], Nikolaus Youdale [2], and Amanda Chow [3] in the University of New South Wales. This project is aiming to measure the major types of air pollutants in Sydney by wireless mobile sensor units and display the pollution level as well as the accurate pollution exposure for individuals via several applications processing on various platforms. The prototype system has been developed and continuously operated for the last one year, but it still needs improvements on various problems including, no pollution readings when GPS lost, inefficient protocol, one-way communication between device and mobile phone, unfriendly user interface, power consumption of Smartphone and uncelebrated sensor readings.

Our Role

Aiming at providing a robust, accurate and flexible pollution measurement system, in this year, 2011, this project has been divided several parts for further developments. These parts are including:

1) User interface for Android application
2) Calibration for pollution sensors
3) Communication between wireless sensor device and Android phone
4) Improvement on iPhone client application
5) Database server maintains

6) New sensors for wireless sensor board

My partner Hailian Zhang and I are involving in improve the communication method between sensors and Android phone in order to obtain reliable and accurate pollution readings.

Our Objective

In general, our thesis aims to improve the system by:

- Estimate the user's routes when GPS signal lost
- Improve the protocol architecture
- Establish the duplex communication between wireless sensor board and mobile phone
- Realize the distance based pollution sampling method
- Enhance the data uploading method

In the rest of this article, the background information regard to air pollutions as well as its jeopardies concerning to the public health will be presented initially. Right following the background information some relative pollution monitory projects and our Haze Watch will be introduced. In the main sections, the circumstance of GPS lost and new location estimation method for dealing with the pollution readings within that circumstance will be introduced first. Then, the communication protocol between device and mobile phone followed by two-way communication achievements will be exhibited detailedly. Subsequently, an enhancement on data uploading method will be demonstrated and this report will end with a conclusion regarding to the works we have done in the thesis A and B.

2 Background

2.1 Air Pollution

Air pollution as one of the most significant pollution concerns constantly affects the major human beings, but only few people realized how serious this problem is. A medicinal research for ThinkQuest [4] specifies that it is about 20000 times with approximately 20000 liters air on average that an adult will breathe every day. Thus, 'how your good the air is', has an inextricable relationship with one's health.

The major pollutants affecting public health includes ozone (O_3), carbon monoxide (CO), nitrogen dioxide (NO_2), sulfur dioxide (SO_2) and particulate matters (PM) [5]. According to large amount researches, it has shown that the long-term effects of those air pollutants on health are including diseases emphysema, asthma, lung and heart diseases, and respiratory allergies [6]. The WHO (World Health Organization) [5] estimated that approximately 2 million premature deaths worldwide are caused by pollutants in the air annually and by reducing the particulate matter in the atmosphere the death related to the air could be reduced by 15%. The medical studies are still focusing efforts on searching for conclusive relationships between exposure to air pollution and various medical conditions. Therefore, an air pollution monitoring system aiming at satisfied the increasing requirements on personal healthy quality is extremely helpful and important for professionals, governments and individuals.

In Sydney, the DECCW (Department of Environment, Climate Change and Water) [7] takes the responsibility for monitoring air pollution in NSW. Currently, they are running 14 fixed observer stations around Sydney and the citizens can visit their homepage to view AQI (Air Quality Index) map and search the air quality database which have been demonstrated in figure1.

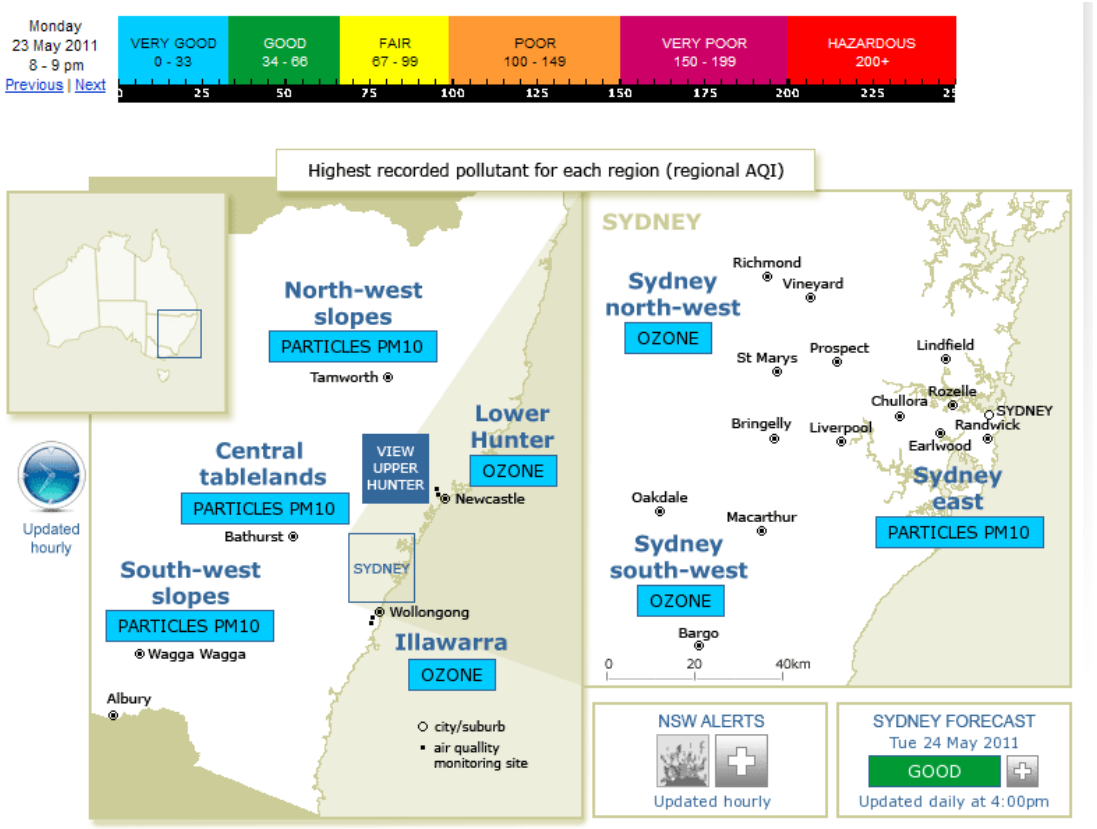


Figure 1 The DECCW Air Quality Index map [7]

2.2 Correlative Projects

However, stationary monitoring systems may hardly provide a high-resolution image of AQI. This has been aware years ago and with the new developed technologies, like 3G network, engineers around world have started to invent and develop mobile air pollution monitoring systems in order to provide more accurate, robust, reliable air pollution readings.

2.2.1 MDAU

S. Dhanalkshmi [8] and her group members from Idhaya Engineering College for Women have designed a Mobile Data-Acquisition Unit (MDAU) which could record air pollution data and request location information through a GPS module. These data are subsequently uploaded to server through a GPRS-Modem (General Packet Radio Service Modem).

2.2.2 MAQUMON

The MAQUMON (Mobile Air Quality Monitoring Network) is a project that

undertaken in 2007 by the Networked Embedded Systems Lab at ISIS, Vanderbilt University [9]. It has built a prototype device packing several modules into a small package. This prototype in figure 2 consists of metal oxide semiconductor sensors, GPS module accelerometer, LCD, USB port, flash memory and a Bluetooth interface. The collected pollution readings are able to periodically upload to a server through the Internet via a laptop or a PDA and displayed on Microsoft SensorMap.

Figure 2 MAQUMON Prototype

2.2.3 Discussion

Several other projects have been presented in [1], for instance, iSniff, Sensaris, MESSAGE project. By closely study at their concepts, it is not hard to find that these projects have a common idea which is integrating pollution measurement module with GPS module and/or mobile network module. This design model, in deed, successfully achieved mobile monitoring, but it also boosts the budget for each single device. The budget given in MDAU is estimated as around US$650, it is far more than an individual can offer for this kind of device.

2.3 System Overview

Based on essentially same concept but different method to achieve, the Haze Watch system separates the GPS module and network module from the sensor board by utilize as much existing hardware as possible present in Smartphones as substitutions rather than duplicate them [1]. The general architecture of overall system is shown as figure 3.

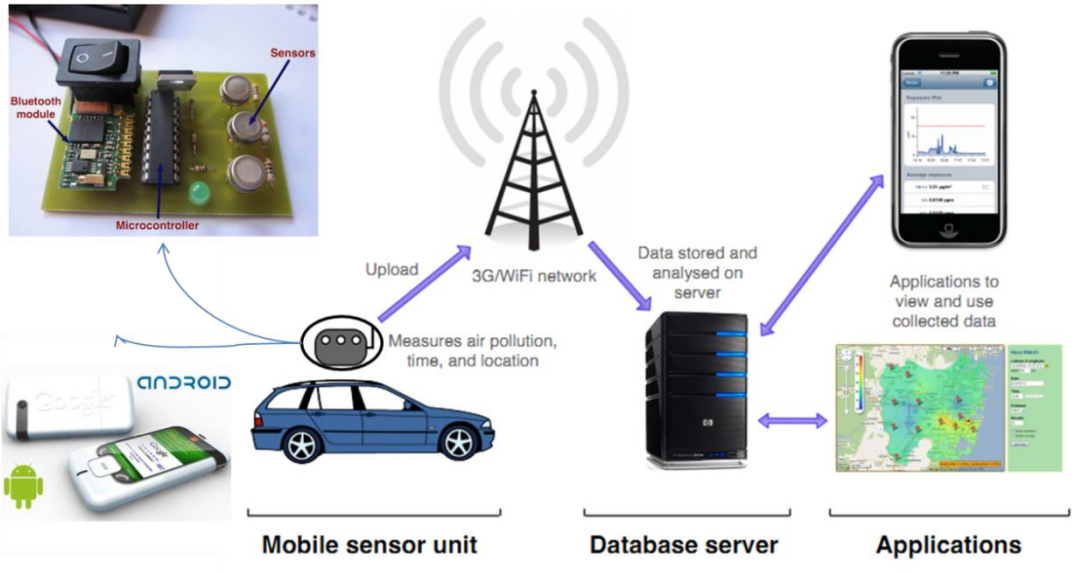


Figure 3 Architecture of Haze Watch system [1]

2.3.1 Wireless Sensor Board

The wireless sensor board designed by J. Carrapetta [2] is the device which takes responsibility of measuring the air pollutants in the atmosphere. The manufactured board is presented in figure 4.

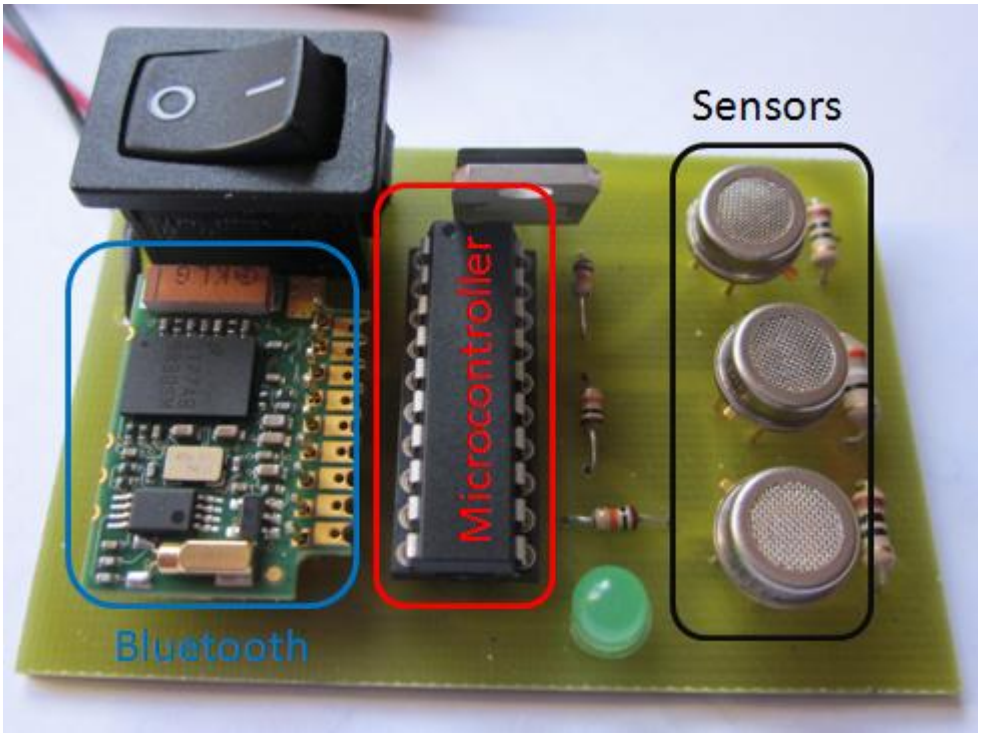


Figure 4 Wireless Sensor Board [1]

This wireless sensor board integrated 3 metal oxide semiconductor sensors measuring O_3, NO_2, and CO, a PIC16F690 microcontroller (MCU), an ARF32 Bluetooth (BT), a

triple-LED as a state indicator and a power source supplied by 4 AA batteries. Manufacturing prize for this board is estimated around AU$100~120 [2].

Once the device is powered up, the sensors on board start converting the pollutants level into voltage changes and reflecting these changes onto the analog input pins of the MCU. The microcontroller subsequently converts these voltages into a 10-bits digital value through built-in ADC (analog to digital converter) module. The device continuously collects samples from each sensor and stores them into memory. After data collection phase, microcontroller enables the Bluetooth to send all collected data as well as other reference coefficients and sleeps a while before next period start.

2.3.2 Smart Phone

The android application programmed by N. Youdale [1] connects with device though Bluetooth under RFCOMM protocol [10]. It takes responsibility that tags each received pollution readings with accurate location and time marks as well as uploading tagged readings to the database server.

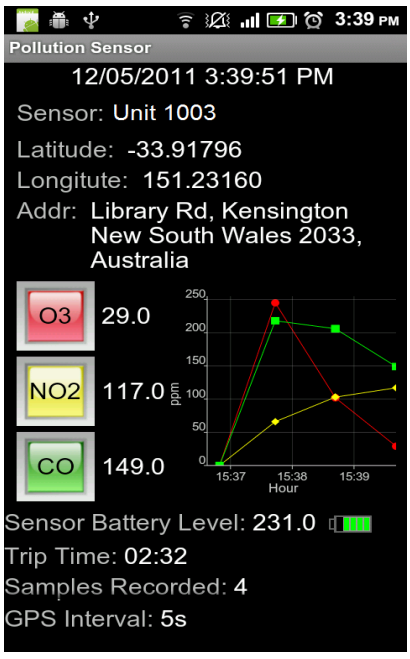


Figure 5 User Interface of Android Application [11, 12]

After the BT connection has established, the Smartphone has been programmed to constantly read data from BT input stream and convert the digital sensor voltage values into standard air pollution readings in terms of ppm by substituting them into calibration equations. Subsequently, these values are stamped with date and geographical location obtained from built-in functions of the Smartphone, before they are uploaded to serveWr through 3G network.

The Android phones are selected in current phase because of its open source code for Bluetooth module and wide and abundant customer groups.

2.3.3 Database Server

The server is also designed by Youdale [1] and it is an extremely important part in the whole system. It runs a database to store all the pollution readings sent from Android phones while provides an interface to client application to querying data from database. It also used for host the project website and source codes for applications.

2.3.4 User Applications

Currently, the client applications include a website containing a pollution map designed by A. Chow [3] and an iPhone application designed by Youdale [1].

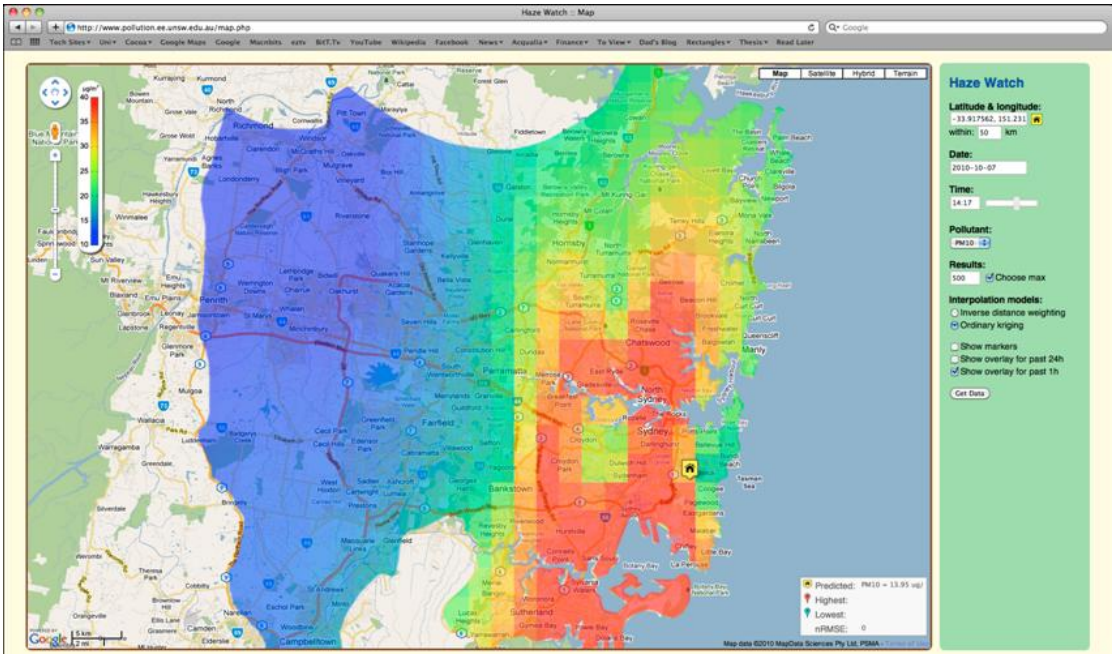


Figure 6 Screenshot of the pollution maps web application [1]

The pollution map (figure 6) provides any accessed user with a graphical picture of the pollution levels in Sydney, overlaid on Google map. The pollution levels are represented in term of colors with blue means low and red means high. Points at where data has been collected including readings for each pollutant may also be viewed on this map.

Figure 7 Screenshot of personal pollution exposure application running on an iPhone [1]

The iPhone application (figure 7) aims to compute the personal exposure of clients. It can track user's geographical location and request pollution data from the server to graphically display the pollutant level around user over time.

3 GPS Lost and Location Estimation

3.1 GPS Lost

Like almost all the other GPS based positioning and navigation system, the Haze Watch system is suffered from the weak GPS signal strength and incorrect location information inside a tunnel. The GPS lost means that the GPS module in the Smartphone stops functioning and no longer providing any accurate locations. However, the pollution data inside a tunnel is extremely important for certain researchers, for example, an environmental engineer may monitor the pollution level to figure out whether their fans are working or not. Additionally, a chunk of blank appears in a tracked trip would give some negative impressions on this system. Consequently, how to estimate the positions inside tunnels is an extremely important task in this thesis.

3.2 GPS Lost Detection

To avoid the absent of data inside tunnels, a precise GPS lost detector is required basically. A location filter mechanism provided on [13] has been employed by Nikolaus [1]. This filter works following the procedure shown in figure 8, where the black lines represent false in conditional checks while the red means true.

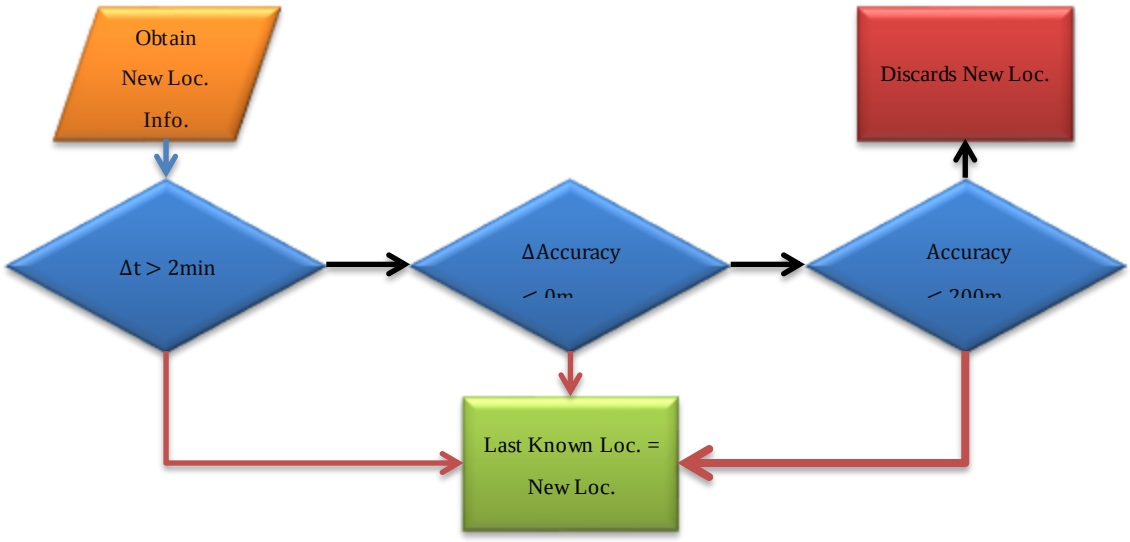


Figure 8 Flow Chat of Android provied GPS Filter

This kind of location filter mainly used to maintain a relatively good location for users by comparing the time and accuracy of newly obtained location information with the stored one. This mechanism in some extent can detect the GPS signal lost but it cannot satisfy the situations in tunnels, because it works based on the newly incoming location information which might not be updated for minutes under that circumstance. Therefore, the field test done by James [2] has the result shown in figure 9.

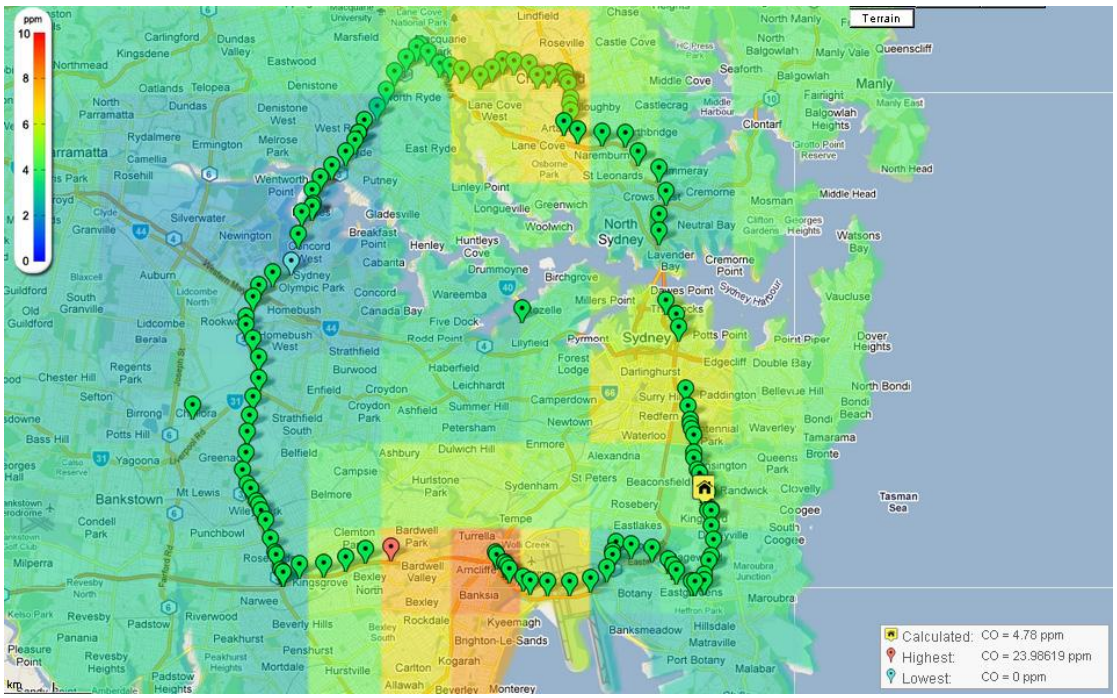


Figure 9 Carbon Monoxide Readings Plotted on a Visualisation Map [2]

A modified filter is developed based on the original location filtering mechanism. This filter works not only when new positioning information is available but also when a set of new pollution data acquiring current location (data reception and GPS update are not synchronized in the traditional data pushing mode, the it always regard last known location as current location. This will be improved and presented in section 4.3.3). The last known location will be cleared out when any newly received location is failed in Accuracy check. Similarly, it will be emptied when the last known location is found not updated within 10s (depends on the minimum time interval settings of the GPS) as the pollution data requiring locations. Therefore, as soon as the application received a null location, the application will regard the mobile phone is entered GPS lost phase

and start applying location estimation method.

3.3 Location Estimation Method

Overview of current indoor positioning and navigation technologies [14, 15, 16], they could be classified into follow categories:

- *Map based positioning system*. The location information of a target is based on premeasured and built map in a database.
- *Electromagnetic wave based positioning technology*. The location of an objective is determined through applying analyses and measurements on the features like frequency, detection and strength of electromagnetic waves received by the system. Commonly, these technologies require some base stations or transmitters with fixed and known positions.
- *Cooperative positioning technology*. These technologies determine the locations of a group of objects by measuring the relative location and exchanging the information among each other.
- *Inertial Navigation System*. It is a system determines target's position by integrating the speed and orientation information from accelerometer and gyroscope assisted with several other sensors and algorithms. With only the accelerometer and gyroscope, the positioning error of this system will increase dramatically.

Since it is almost impossible to preinstall fix base stations inside tunnels or built up a cooperative positioning network, the INS and map based system have been consider. Comparing these two technologies, the INS seems more flexible for diverse situations, but it drains out the batteries extremely quickly because the large power consumption when continuous sensing data. Therefore, INS has been abandoned at last and only the simplest method has been selected.

3.3.1 Linear Interpolation

Linear interpolation is the simplest approach to estimate the route by simply regard the path during the GPS lost period is a straight line. When the GPS lost is detected, the

application will enable a counter to count the number of pollution data has received during this period and store these data into memory. As long as a newly accurate location is detected, the location for each cached data will be calculated applying the formula:

$$\Delta Location = \begin{pmatrix} \Delta Longitude \\ \Delta Latitude \end{pmatrix} = \frac{\begin{pmatrix} Current\ Longitude - Last\ Known\ Longitude \\ Current\ Latitude - Last\ Known\ Latitude \end{pmatrix}}{No.of\ data}$$

$$Location = Last\ Known\ Location + \Delta Location \times index\ of\ data$$

Since the time interval of each received data is equal and fixed (explained in section 4) and the speed of the car could deem as constant, the position for each data is equally distributed on the straight path between the start and end point of the route.

3.3.2 Nonlinear Interpolation

The nonlinear interpolation is based on premeasured tunnel map stored in the hard disc. Five major tunnels in Sydney including, Eastern Distributer, Cross City Tunnel, Sydney Harbor Tunnel, Lane Cove Tunnel and M5 East Tunnel, are premeasured in this application with each adjacent two sample has approximately equal distance interval at 40 meters, and all the possible entrances and corresponding exits are clearly marked. These tunnels then are divided into small parts at each fork of the road and the all possible routes are indicated with names and indexes as demonstrated in table 1 [17].

Entrance Index	Entrances	Exit	Exit Index	Routes
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

Table 1 Tunnel Matching Table

The corresponding routes are shown in figure 10.

Figure 10 Possible Route From Entrance L

After all the tunnels are mapped, the next step is to select correct tunnel when GPS lost. The flow chat shown in figure 11 is the steps to estimate the appropriate tunnel from all.

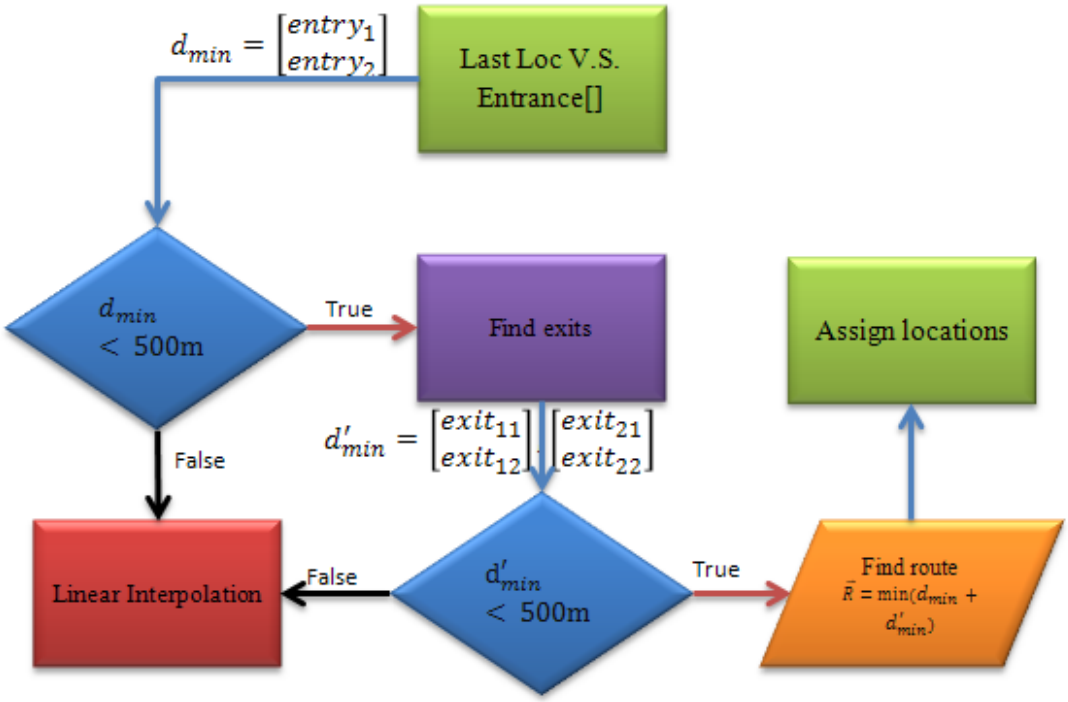


Figure 11

The application first compares the last accepted location with all entrances' coordinates to find the closest two using the Haversine formula (an equation gives circle distances between two points on a sphere from their longitudes and latitudes [18]):

$$d = 2R \cdot atan2(\sqrt{a}, \sqrt{1-a}),$$

$$where, a = sin^2\left(\frac{\Delta Lat}{2}\right) + cos(Lat1)\, cos(Lat2)\, sin^2\left(\frac{\Delta Lng}{2}\right),$$

$$\Delta Lat = \frac{2\pi}{360} \times (Latitude_2 - Latitude_1),$$

$$\Delta Lng = \frac{2\pi}{360} \times (Longitude_2 - Longitude_1)$$

$$R = radius\ of\ the\ earth$$

The reason why to choose the closest two entrances is that several entrances are very close to each other as well as the GPS may not return a location at the precise entrance of the tunnel. In order to avoid mismatching the route, the closest two entrances with distances smaller then 500m are selected for route estimation. For the same reason, the closest two exits comparing to the current location are selected. Then by comparing the summation of the errors the route will be estimated as the combination which gives the shortest distance offset over all.

Based on the assumption, constant speed, the approximated location in the tunnel could be calculated through,

$$index\ of\ estimated\ location \approx \frac{\frac{1}{n+1} \times k}{\frac{1}{N+1}},$$

$$n\ is\ total\ No.of\ data\ collected\ in\ tunnel$$

$$N\ is\ total\ No.of\ points\ premeasured$$

Finally, the pollution data will wrapped with coordinate stored in the corresponding index.

3.3.3 Hybrid System

Each single interpolation method above has its advantages and drawbacks and cannot fully satisfy the positioning when GPS lost. The linear interpolation is flexible for any

location and no premeasured coordinates are required while the nonlinear interpolation is more accurate in the prebuilt areas. Additionally, location information for tunnels requires large memory space in hard disc and RAM (Random Access Memory) when operating route estimation. Hence, a hybrid system is realized for these interpolations to complement each other. The system will operate linear interpolation in conditions:

1. Number of pollution data collected during GPS lost is smaller than 5, which means when users drives at speed 60km/s the start location is about 420m away from the end location.
2. Cannot find an entrance within 500m around the start location and the distance between two locations is not excess 500m.
3. Cannot find a possible exit within 500m around the end location when entrances are given and the distance between two locations is not excess 500m.

The first concern for choosing 5 set of pollution data as the threshold to distinguish the two interpolation is that the displacement during receiving 5 pollution data is estimated as 420 in regular situations. It is a distance that can cover almost all the short tunnels in Sydney. Additionally, the threshold is checked before any tunnel information becomes instances for distance calculation, thus the application may faster the responses.

3.4 Field Test Result

Several field tests for demonstrate the performance of the interpolations has been taken.

The linear interpolation method has been test in two tunnels. One is a short tunnel, the airport tunnel, with result shown in figure 12.

Figure 12 Linear Interpolation at Airport Tunnel

In this test the points that have been circled, are the samples applied linear interpolation. The first point is the last accepted location while the last point is the new location. It is obvious that the linear interpolation performs well with all the interpolated points have dropped on to the tunnel.

The M5 East tunnel has also been test for linear interpolation with result shown in figure 13.

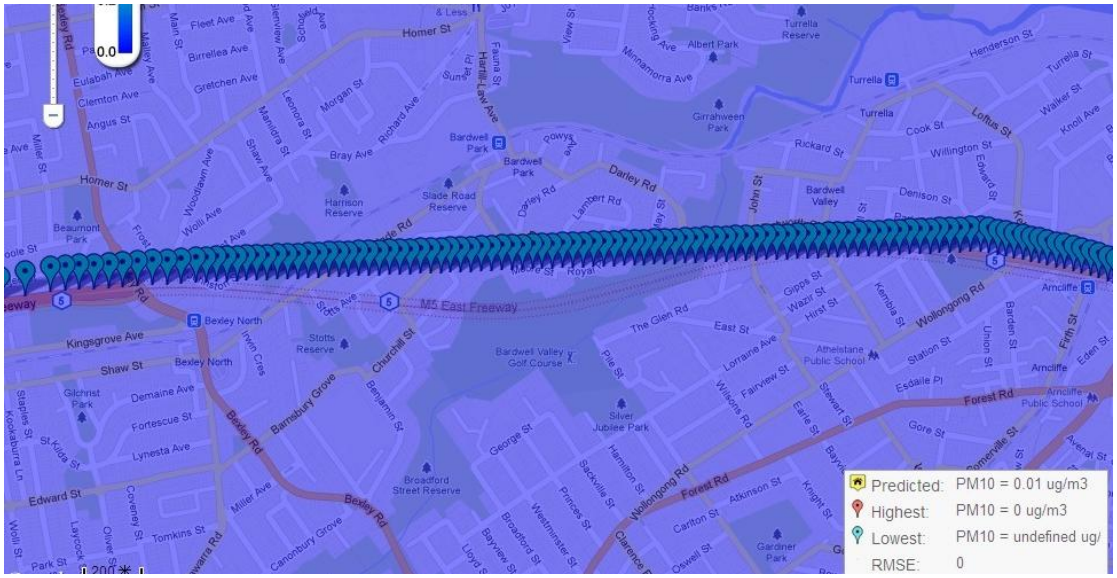


Figure 13 Linear Interpolation at M5 East Tunnel

In this test, almost all the results drift off the road with significant errors which can be at most 200m away from the tunnel. Therefore, the linear interpolation is not suitable for these long tunnels.

The nonlinear interpolation method is also tested. The experiment chose the route start from our school and goes into the Eastern Distributor from the South Dowling St. entrance. Then, from the Eastern Distributor, we turned left entering the cross city tunnel and exited from M4 Western Distributor Freeway. It was chosen for demonstrating the performance of the route chose method of nonlinear interpolation because of its complexity. The test result is demonstrated in figure 14.

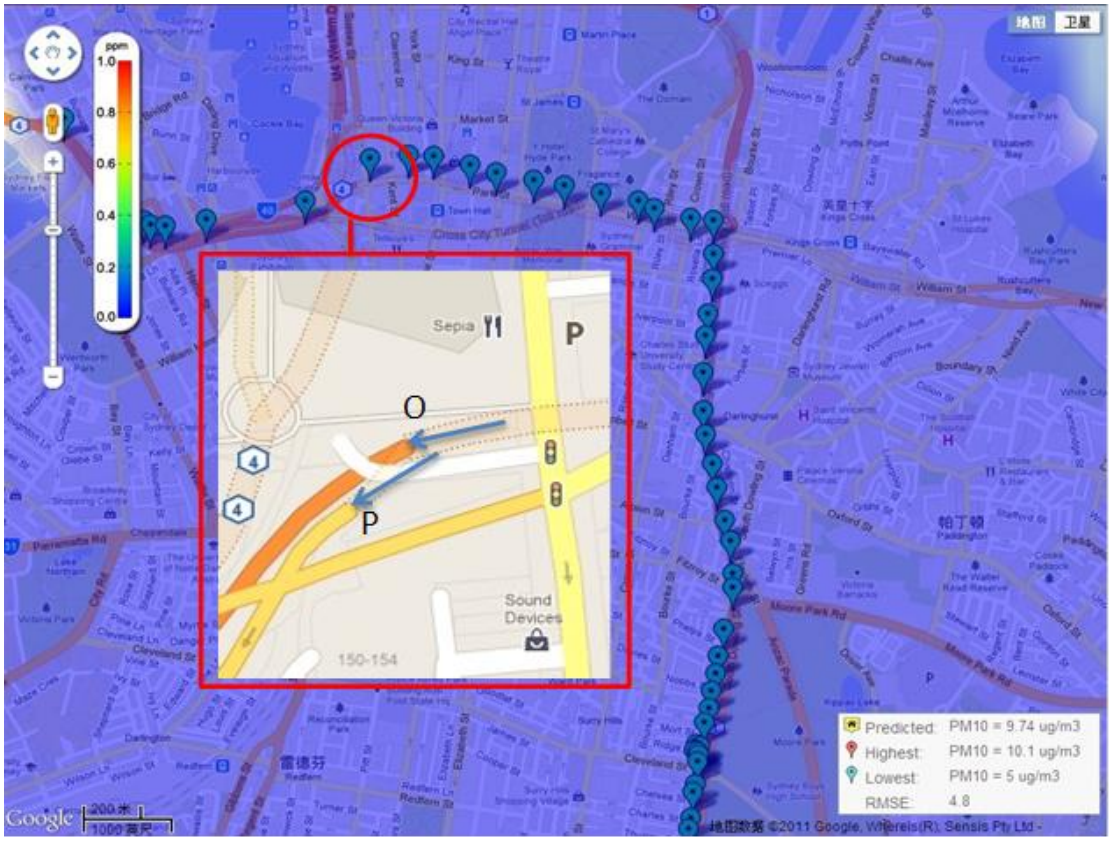


Figure 14 Nonlinear Interpolation at Eastern Distributor and Cross City Tunnel [17]

Basically, the test result is encouraging and satisfactory. The data collected inside the tunnel was assigned locations stick on the actual path and the route estimated by the application does not have any significant error. The only concern regard to the experiment is the exit chosen between O and P. These two exit are very close to each

other, even it cannot identified from the figure result, the route was estimated based on exit P, read from outputted log, instead of actual exit O. But fortunately it is acceptable for this drift, since it did not create other errors.

4 Communication Between device and Mobile Phone

For reliable and high-performance network communication between wireless device and mobile phone, a protocol formatting rules that specify how data is packaged into messages sent and received was designed, approved and employed in this project.

4.1 Protocol Version 1

Protocol version 1 is the first prototype message format introduced in [1, 2] for data exchanging between device and mobile phone. The based concept of this protocol is to wrap all data with clear defined and agreed header and footer and associate with a checksum for confidential concern. Also, the data included in the message has to have ability that can distinguish different device and identify the features of the connected board. Therefore, the protocol is designed as the format shown in figure 15.

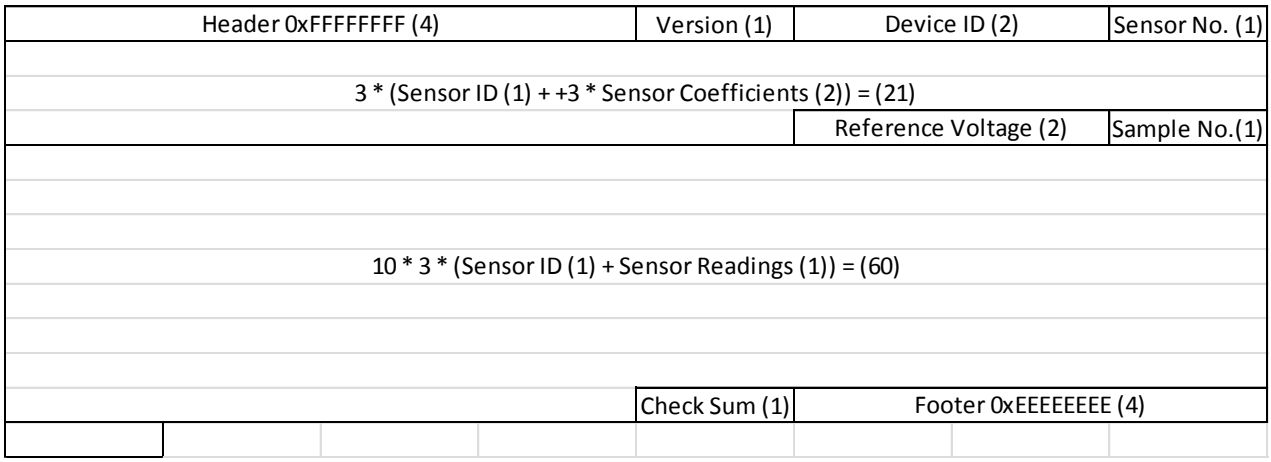


Figure 15 Protocol Version 1

The message format in the figure 15 is one-dimensional and starts from top left to bottom right. Each field encompassed in the protocol is putted in black frame with number of bytes in the brackets. The information contained in each single message includes,

- **Protocol version:** marks the current protocol version and indicates the sequence of following fields.
- **Device ID**: identify the processing device
- **Number of sensors**: number of sensors working on the current device. For

the manufactured devices, it has fixed number of 3 sensors built on board.

- **Reference voltage**: a parameter for Android phone to convert the digital data to analog voltage values
- **Sensor IDs**: identify each sensor's coefficients or reading
- **Sensor coefficients**: provide all three coefficients in the calibrated formula which used for calculating the pollution level in terms of ppm (parts per million) from sensor voltage. Each coefficient is represented in floating-point with first 12 bits significant signed integer and 4bits exponent signed integer and the actual value$= sig \times 10^{exp}$. The value could be ranged between $-2048 \times 10^{8} \sim 2048 \times 10^{8}$ and the precision could be as high as1×10^{-8}.
- **Number of samples**: number of data sampled on each sensor. It is 10 samples for protocol version 1.
- **Sensor data**: the raw digital value that directly converted from analog input, the voltage of a sensor, through ADC module of the microcontroller. This value is determined with following steps [19]:
  - Divide the reference voltage, the V_{dd} operated on microcontroller in this case, into 1024 (2^{10}) levels and each level covers a small range of voltage.
  - Sample the voltage of the sensor and allocate it into a corresponding level
  - Convert this level into 10-bits binary

The reasons for each field was selected are clearly introduced in [1]. Once the connection between device and mobile phone is established, the phone will start reading data from an arbitrary location mostly not aligned with the beginning of a message. Then the header of message is listened as the starting mark of a completed message stream. The 4-bytes hex number FFFFFFF is chosen because of this specific sequence would appear nowhere in both handshake protocol of BT connection and designed message format.

The microcontroller operated on board is a RISC (reduced instruction set) 8-bit microcontroller, PIC16F690. Like most other microcontrollers at the same level can

only handle 8-bit integer math in hardware and has no floating-point math hardware. Such computations are opted to execute on the Smartphones, where math libraries and floating-point operations are far more perfect. So instead of transmitting a computed pollution value, several fields including Reference voltage, Sensor coefficients and Sensor data are sent over BT connection.

This protocol has been phenomenal successful in transmitting data from device to Android phone. The Haze Watch system has successfully collected pollution data around Sydney, but it is not robust enough comparing with other similar projects.

The numerous information contained in a single message, is transmitted regularly and repeatedly whenever the device is switched on. Among it, parts of the information will stay constant and unnecessarily to be sent as long as the connection between the devices and mobile phone remains open. It is extraordinarily power-consuming as it required 40mA (60mA peak) transmission current while only 15mA for maintain the connection and 5mA for listening connection requires [20].

The multiple samples provided in one message stream will be averaged after converted in to pollution levels, which is unreasonable because these data acquisitions happened in different time and locations. The whole procedures to collect 10 data samples running based on protocol 1 takes approximately 15s. It takes about another 15s for transmit the bulky message over the BT connect (the reasons for this relatively long transmission period will be presented in section 4.3.2), which means, when sensors travelling at 60km/h the practical location of the first sample is 500~650m away from the location stamp as well as 30s earlier than the time stamp given by the Android phone.

Consequently, new version of protocol and an improved procedure for data acquisition is designed, developed and implemented.

4.2 Protocol Version 2

The main objective of protocol version 2 is to avoid the constant information

contained transmitted repeatedly by via separate them from varying valuables. The message format consisted with constant fields is called control message while the other is data message.

4.2.1 Control Message

The control message comprises information fields exhibited in figure 16.

Header 0xFFFFFFFF (4)	Version (1)	Msg. type (1)	Device ID (2)
Last Calibration Date (3)	Next Calibration Date (3)		Reference Voltage (2)
3 * (Sensor ID (1) + +3 * Sensor Coefficients (2)) = (21)			
		Check Sum (1)	Footer 0xEEEEEEEE (4)

Figure 16 Protocol Version 2 Control Message

- **Version**: marks the current protocol version and indicates the sequence of following fields.
- ***Message Type***: new field added in protocol version 2. It is used to distinguish the control message and data message as well as specifies the sequence of following data. Control message is denoted by integer number '*1*' in this field.
- **Device ID**: identify the processing device
- ***Last calibration date*** and ***Next calibration date***: new fields added in protocol version 2. They indicate the last and next calibration dates for each different device. The sensor used on the wireless boards need to be calibrated every half year and these fields are used to alarm users.
- **Reference voltage**: a parameter for Android phone to convert the digital data to analog voltage values
- **Sensor ID**: identify coefficients for each sensor
- **Sensor coefficients**: provide all three coefficients in the voltage-ppm transfer functions.

In order to preserve the integrity, this protocol also employs header and footer to wrap the information contained. Also, two new fields are integrated while the sensor count field is discarded. This is because the wireless sensor board already has standard schematic with fix sensor count that equals 3.

4.2.2 Data Message

The data message is transmitted following the data sequence demonstrated in figure 17.

Header 0xFFFFFFFF (4)			Version (1)	Msg. type (1)	Bettery Level (2)
3 * (Sensor ID (1) + Sensor Readings (2)) = (9)					
	Check Sum (1)	Footer 0xEEEEEEEE (4)			

Figure 17 Protocol Version 2 Data Message

- **Version**: marks the current protocol version and indicates the sequence of following fields.
- ***Message Type***: new field added in protocol version 2. It is used to distinguish the control message and data message as well as specifies the sequence of following data. Data message is denoted by integer number '*2*' in this field.
- ***Battery Level***: new field added in protocol version 2. It is a continuously decreasing digital value representing the voltage of device's batteries and it will be converted into percentage of electricity remained inside the batteries.
- **Sensor ID**: identify coefficients for each sensor
- **Sensor coefficients**: provide all three coefficients in the voltage-ppm transfer functions.

Same as control message, the data message also started with header, version and message type flag and end with footer. Likewise, battery level field is added to satisfy the corresponding function that alarm user to recharge batteries packed on device [11,12]. Furthermore, misused readings from ADC in microcontroller are corrected. As mentioned in section 4.1, the output of ADC is a 10-bit digital reading. However, in protocol version 1, all the pollution readings are only used the 8 most significant bits and the reference voltage is divided into 256 (2^8) different levels instead of 1024. Consequently, the resolution and accuracy of these readings are reduced. As a correction, all the pollution readings are extended to 2 bytes, 8-bits for storing the 10-bits ADC outputs.

4.3 Two-way Communication

The two-way communication is designed to improve the data acquisition method and

enhance the interaction between device and Smartphone. Associating new protocol on data exchanging and achieving distance-based data acquisition method are the two main purposes and achievements in this thesis.

4.3.1 Schematic

In order to realize duplex communication, the original schematic of the wireless sensor board is modified into the new one shown in figure 18.

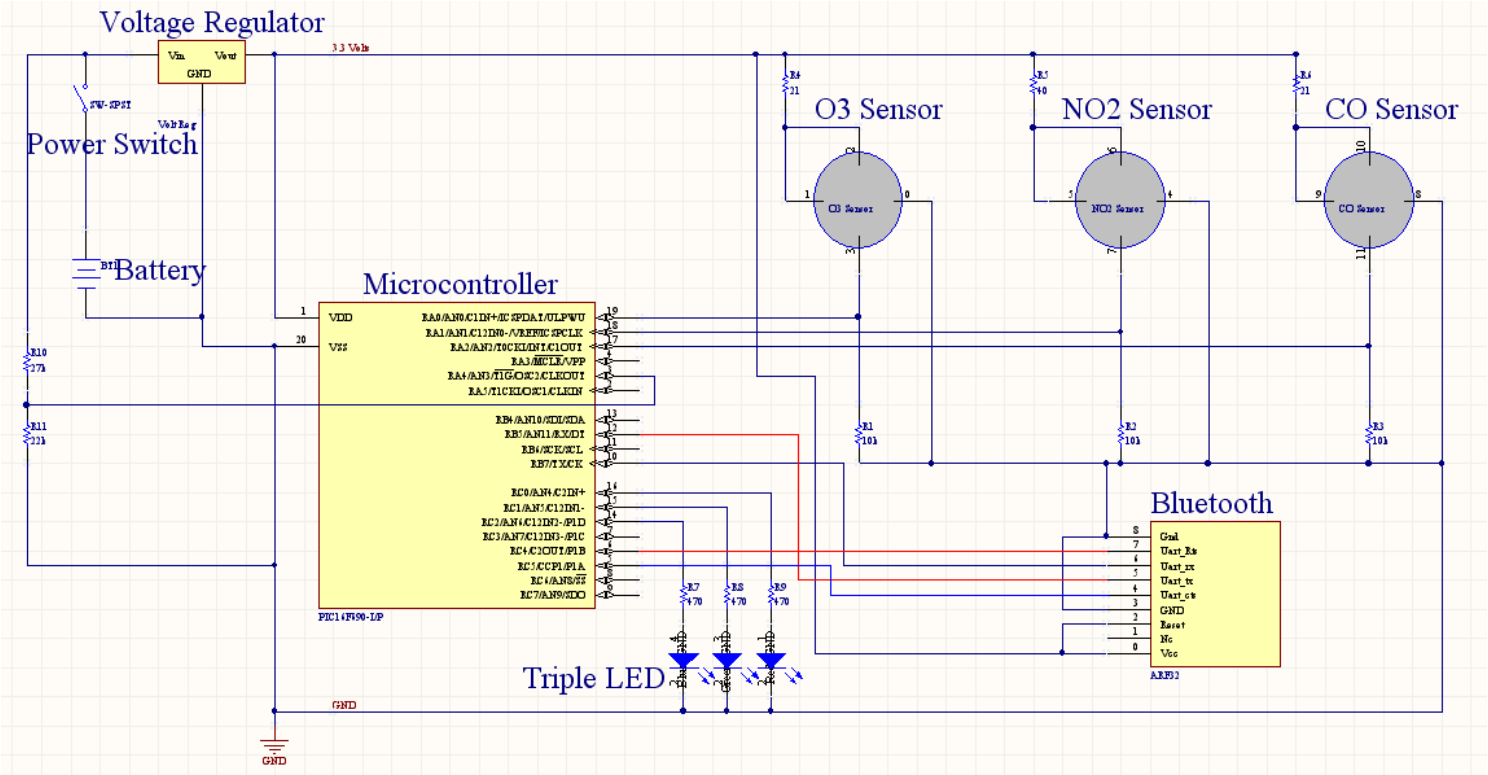


Figure 18 Schematic for Two-way Communication

The microcontroller interfaces with the BT module using the USART interface. The simple connection between RX/DT pin (pin 12) on microcontroller and UART_tx pin (pin 5) on BT module will make the hardware available for duplex communication. The Enhanced Universal Synchronous Asynchronous Receiver Transmitter (EUSART) built in PIC16F690 is a serial I/O communications peripheral which transmits and receives data using the standard non-return-to-zero (NRZ) format, so the controller has ability to detect and initiates character reception on the falling edge of the first bit for any incoming data signal sequences [19]. But the connection between I/O port (pin 5) on controller and UART_rts (pin 7) on BT module is still preserved for further use. The new schematic has been built on breadboard as posed in figure19 and already be used in field tests presented in section 3.4 and 4.4.

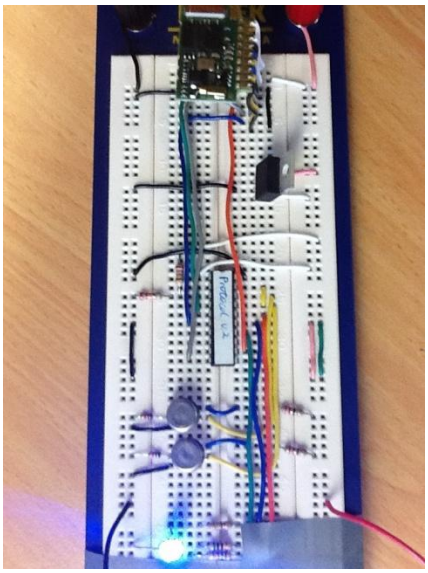


Figure 19 Board for Two-way Communication

4.3.2 Procedures

Once the hardware is ready for two-way communication, the whole procedures for data exchanging between sensor board and Android phone will change dramatically with the assistance of protocol version 2.

In the previous version the device can only blindly and continuously push data once it has been powered. For now, when the device is powered up, it goes to standby mode and wait for any incoming message. The RCIF interrupt flag bit will be set when there is an unread character in the RCREG, receiver register where the received byte is stored [19]. At same time the device will jump out form standby mode, read data from RCREG then check the meaning of the data. It is single byte control with '1' represents "request to send control message" while '4' signifies data message is requested. These two characters are chosen to avoid mismatching these configure bytes with the start delimiter and end delimiter of BT connection's handshake text

which are ‘2’ and ‘3’ [20]. Any byte that neither ‘1’ nor ‘4’ will be ignored while the corresponding messages for ‘1’ and ‘4’ will be transmitted immediately after they received.

Furthermore, the overestimated transmission delay between two characters causes the abnormal long period in transmission phases. The speed of the physical USART interface is 9600bits/s with data format 1start bit, 8 data bits and 1stop bit, 10 bits in total [19, 20]. Thus, it requires only approximately 104us to transmit 1byte from microcontroller to BT module. And for BT module, it fully complies with the V2.0 Bluetooth® standard and data rate goes up to 723kbps [20], which could broadcast data even faster. Therefore, the original transmission delay is reduced form 200ms to 10ms (can be even shorter) and the transmission period of a data message is reduced to around 200ms. Additionally, a 4s ‘cool down’ between each data message is added since there is no necessity for sampling air pollution too frequently. So the sampling period will fix at about 5s/sample when the data is acquired uninterruptedly. A sample byte received on RX/DT pin of the controller is captured on oscilloscope shown in figure 20 with 1 start bit (the first square peak) followed by ‘11001011’ as data and 1 stop bit.

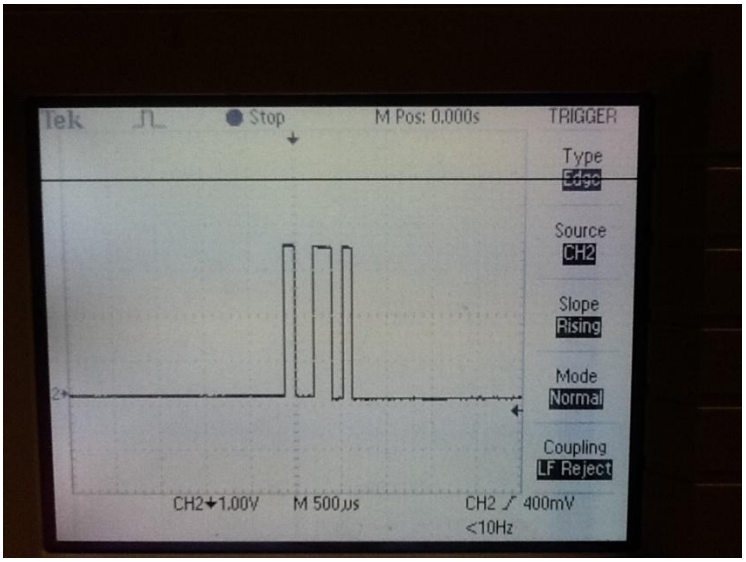


Figure 20 Oscilloscope Captured Received Signal

At the application aspect, the Android phone will first require control message to be send as soon as the BT connection is ready. The control message is used to construct a framework for further incoming data. Once the framework is established, it will not be requested again and only data message is transmitted in the rest of time. The procedures are presented in a visualized way as figure 21, the screen shot of outputted Log.

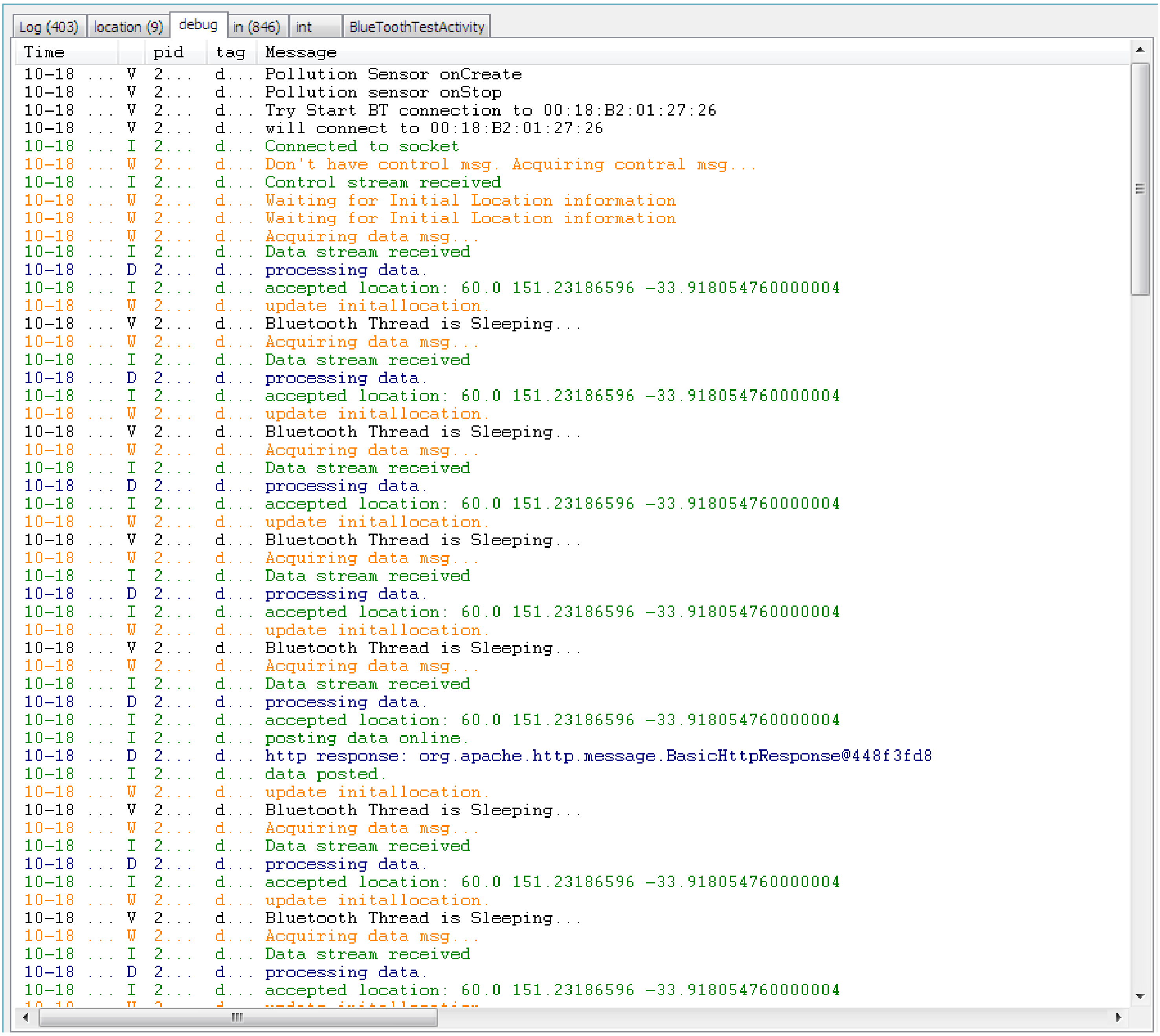


Figure 21 Screen Shot of The Log for The Application

4.3.3 Displacement-Based Data Acquisition

Since the location information can only be obtained by the Smartphone, the duplex data switching capability makes it possible that the device samples the air pollutants equidistantly.

In the displacement-based model, the GPS updates the user location equidistantly. When this update detected by application, the acquisition for pollution data will be send to device and it start sampling data at that critical position. This kind of method avoids the location drift caused by transmission delay and the accuracy is only depended on GPS module of the Smartphone. Besides, too much data collected in a small area when user is stationary, is unnecessary and will waste energy and also heavies the server load [1].

4.4 Test Results

The two-way communication assisted new protocol has been tested and functioned in several field tests including those demonstrated in section 3.4. And the nonlinear interpolation tests had also taken as a part of a power consumption experiment. Two devices mounted two different microcontrollers with old and new protocol burned, were supplied with fully charged Energizer AA batteries. These two devices were switched on at same time at 8 o'clock in the morning and the old device failed to connect with mobile phone after 15h and 10m while the new one failed after 28h and 24m. This significant improvement is mainly because the device goes to sleep mode when there is no connection between device and phone established.

The displacement-based data acquisition method also has been tested and the result is shown in figure 22

Figure 22 Displacement-based Data Acquisition Method

Obviously, the pollution samples are perfectly taken at every 200m displacement from each other. This function improved the pollution collection manner significantly especially when users waiting for a traffic light. The red circles in figure 22 are marking the traffic lights while the figure 23 is showing the data collection manner in the normal data push mode.

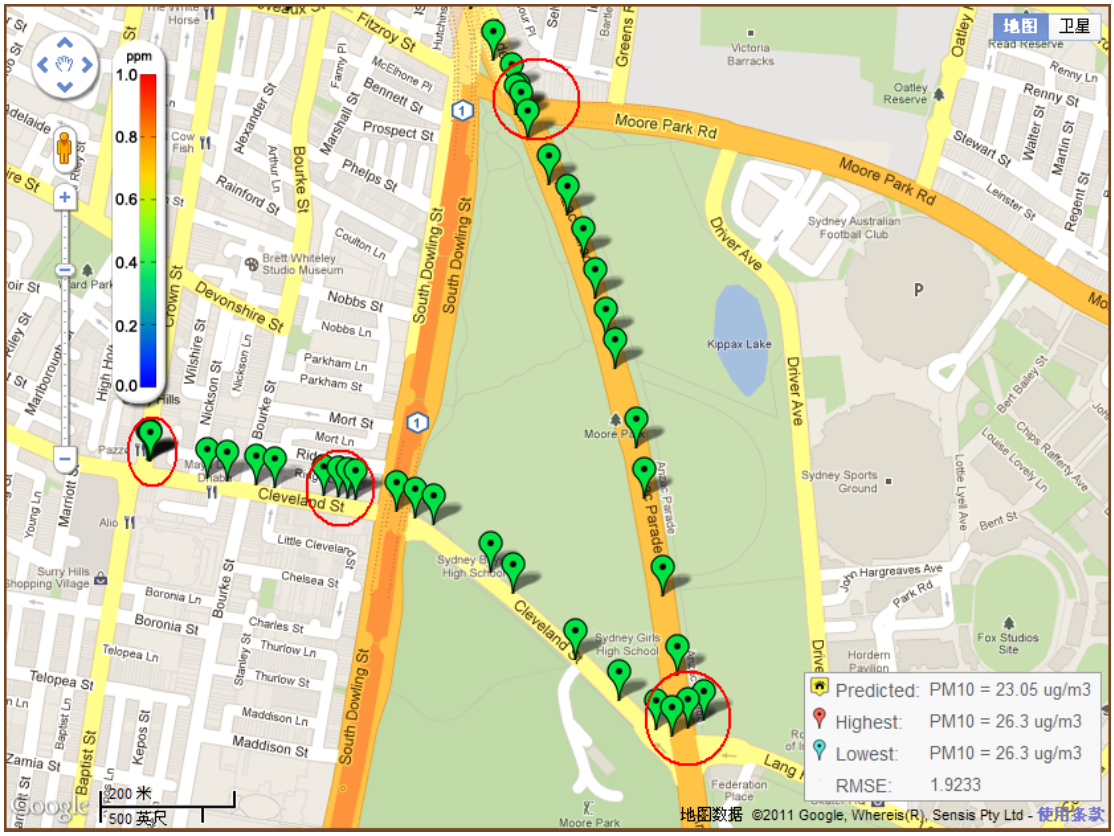


Figure 23 Normal Data Push Mode

It is very obviously that in figure 23, air pollution is over-measured in a short period at a small area and some of them are even overlaid by later measurements, while this situation can be totally eliminated by displacement-based data acquisition method.

5 Data Uploading

The application running on the Android phone is consist of two threads. One is the main thread which takes responsibility to manage application and update user interface. The other one is the service thread which consists of maintaining BT connection, processing pollution data and uploading pollution readings.

5.1 Constrains

In the previous Android application, the average readings of each sensor will be computed and wrapped with current time and location. Subsequently, it will convert this packet to an xml file then try to upload this packet through http post and returning back to data receiving phase as soon as there is a response from the server.

The old procedures might no longer suitable for the new data acquisition methods. The updated protocol running between device and Android phone reduces the pollution sampling period into 5s, which means Android phone has to try TCP connection and upload one XML file every 5s. Such a high frequent uploading rate will significantly increase the battery consumption as well as the 3G network traffics since each connection required handshakes based on TCP protocol. Besides that, the single service thread has to have unblocked and free accessed features for a good preference. In the old procedures there are no time limitations for expecting the server's response. If a disconnection happens between Android phone and server, the whole application will be stacked until there is any response from server.

5.2 Uploading Method

Some modifications have been worked on the data uploading phase of the application.

5.2.1 Buffer

A data buffer is employed for caching any completely processed data (tagged with correct location and time) before uploading. The size each well readied data can be estimated as follow:

- Data: readings of each sensor mapped with pollutants' name
  - Names of pollutants: 1 String (3char * 2bytes * 3sensors = 18bytes)
  - Values of sensors: 1 Double (1double * 8bytes * 3sensors = 24bytes)
- Date: including year, month, date, hour, minute, second
  - 1 Integer value for each parts (6int * 4bytes = 24bytes)
- Location: consist of a huge amount of information [13]
  - Longitude: 1 Double (8bytes)
  - Latitude: 1 Double (8bytes)
  - Altitude: 1 Double (8bytes)
  - Accuracy: 1 Float (4bytes)
  - Speed: 1 Float (4bytes)
  - Bearing: 1 Float (4bytes)
  - Time: 1 Long(8bytes)
  - Provider: 1 String (varied length)

Consequently, we may estimate the data packet size is 120bytes and set the initial buffer size to 20 packets with the upper limit of the buffer size is 100 packets, which means it only takes at most 12kilobytes of RAM. The typical size of RAM for a Smartphone is ranging from 256MegaBytes to 4 GigaBytes, for our case it is more than sufficient for our needs. The sample XML file uploaded corresponding to multiple data uploading is demonstrated in figure 24.

```
<?xml version="1.0" encoding="UTF-8" ?>
<data>
  <info></info>
  <samples>
    <sample>
      <datetime>2010-05-24 18:00:00</datetime>
      <latitude>-33.7972222222222</latitude>
      <longitude>150.765833333333</longitude>
      <no2>2.0</no2>
      <o3>0.2</o3>
      <pm10>17.6</pm10>
    </sample>
    <sample>
      <datetime>2010-05-24 18:00:00</datetime>
      <latitude>-33.9177777777778</latitude>
      <longitude>151.134722222222</longitude>
      <no2>2.2</no2>
      <o3>0.1</o3>
      <pm10>22.2</pm10>
    </sample>
  </samples>
</data>
```

Figure 24 Uploaded XML File [1]

5.2.2 Timeout

A timeout mechanism has been utilized for unblocking the service thread. It processes obeying the flow chat presented in figure 25.

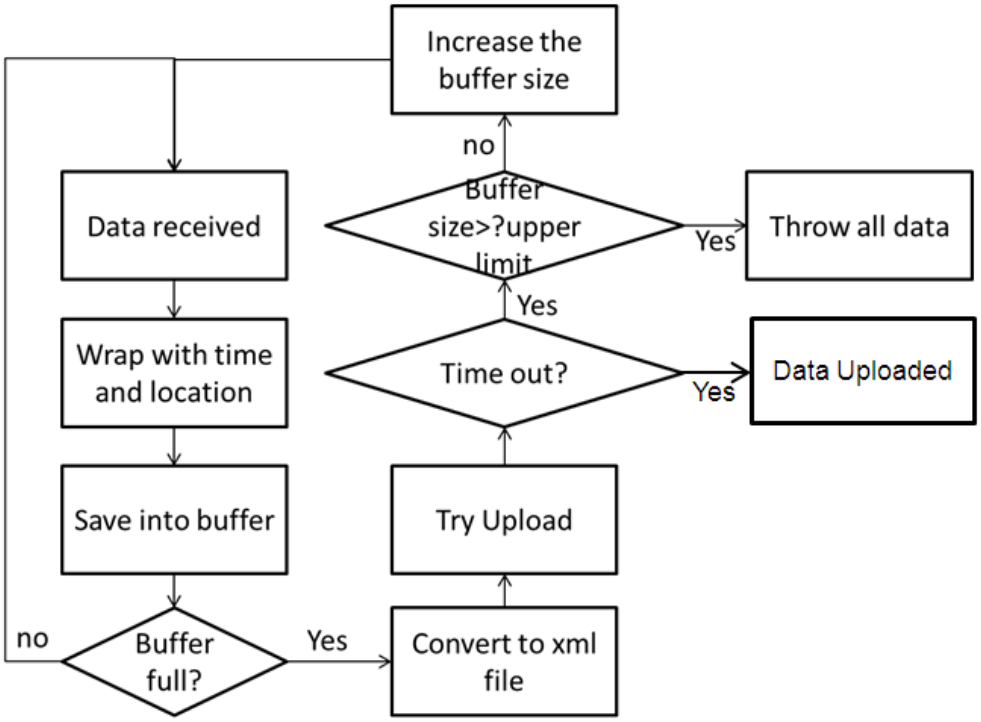


Figure 25 Timeout Procedures

Once the application tries to setup a connection between Android phone and the server, it will only wait 5s for server's reply. If the request dose not receives any acknowledgement from server with this time limit, the application will be interrupted and jumped out from listening server's response. The unsuccessful uploaded data will keep caching in buffer, but the buffer size will be enlarged by 20 for more data. This loop will repeat at most 5time until the buffer size reaches the upper limit, 100. If still no response for the last try, all data cached in buffer will be clear and the buffer size will be reset to 20.

This design is considering the stability of the whole system in some short term disconnection situations occurred like 3G network signal lost while not overload the RAM because of long term disconnection, for instance, the server is down.

6 Future Work

If we compare the Haze Watch project to a human being, it is just entering the adolescence form infancy. The possibilities for this project are almost unlimited. According to my personal knowledge and ability, several recommendations for future works have been presented in this section.

6.1 Hardware

6.1.1 Batteries

The power system of the device is supplied by rechargeable batteries. When the batteries run out of electricity, it is very unfriendly for user to take batteries out of device and recharge them for 11hours. If a rechargeable power supply with vehicle mounted charger adapter is available for device, it would be much helpful.

6.1.2 Power Managements

Based on the power consumption experiment and some researches, the 3.3V voltage regulator on-board consumes a great amount power and turns them into heat. The current power supply output 4.8V DC voltage, but the Bluetooth module can only work with voltage not excessed 3.6V [20]. The 3.3V voltage regulator is only one solution, other ways to divide and stable the supply voltage would have to be undertaken on increasing the battery performance.

6.1.3 Modules Monitory

Currently, all the modules on wireless sensor board are powered up and enabled once the device is switched on. Since the two-way communication has achieved as well as the standby model of the device, the various modules could be designed as monitory by microcontroller. For example, the Vss pin of the sensors could be connected to the digital I/O ports of the controller and output logic high to enable the sensor while output low to disable them.

6.2 Smartphone Application

6.2.1 Intelligent Application

Currently, the application running on the smartphone requires client manually open the application and connect to the device. Although some auto connection mechanism has been developed by other group, this automation method is power consuming because it require a continuously speed detection based on GPS. A robust automation method on connection between device and phone or other aspects would be helpful for a friendly user experience.

6.2.2 New Protocol

Since the application has achieved duplex communication with device, new functions depending on message exchanging will become more and more complex. Therefore, a new protocol for the two-way communication would be needed and it has to carefully choose the format to distinguish the standard handshake messages.

7 Conclusion

The old stationary air pollution monitory systems are going be obsoleted and replaced in future development. Haze Watch system with potential capacities on providing a more accurate, robust, reliable air pollution monitory system and higher-resolution image of the air quality, has been developed and operated.

The works taken within this thesis have solved the difficulty of data collection in tunnels in Sydney. The two-way communication established between the device and Smartphone has laid foundation for other more complicated functions to be realized in further developments. Based on the practical experiments and their corresponding results, I believe we have achieved objectives set in thesis A. But the possibilities for Haze Watch system are far more then these, significant improvements are expected in future.

References

[1] J. Carrapetta, "Haze Watch: Design of a wireless sensor board for measuring air pollution", University of New South Wales, Sydney, Thesis 2010.

[2] N. Youdale, "Haze Watch: Database Server and Mobile Applications for Measuring and Evaluating Air Pollution Exposure", University of New South Wales, Sydney, Thesis 2010.

[3] A. Chow, "Haze Watch: Data Modeling and Visualization using Google Maps", University of New South Wales, Sydney, Thesis 2010.

[4] Caroline *et al.*,(n.d.) *Air Pollution* [Online, accessed 10 May 2011]. <http://library.thinkquest.org/26026/Environmental_Problems/air_pollution.html>.

[5] World Health Organization. (2011). *Air pollution*. [Online, accessed 14 May 2011]. <http://www.who.int/topics/air_pollution/en/>.

[6] Wikipedia. (21 May 2011 at 13:21). *Air pollution*. [Online, accessed 22 May 2011]. <http://en.wikipedia.org/wiki/Air_pollution>.

[7] Office of Environment & Heritage. (02 March 2011) *Air quality index (AQI) values – updated hourly*. [Online, accessed 23 May 2011]. <http://www.environment.nsw.gov.au/AQMS/aboutaqi.htm>

[8] S. Dhanalkshmi, *et al.* (2011). *A Mobile Air Pollution Monitoring System Using GPRS*. [Online, accessed 20 May 2011]. <http://www.aksheyaa.com/ece/A%20Mobile%20Air.pdf>

[9] Institute for Software Integrated Systems. (Jan 2008) *Mobile Air Quality Monitoring Network*. [Online, accessed 22 May 2011]. <http://www.isis.vanderbilt.edu/projects/maqumon>.

[10] Android developers. (20 May 2011). *Bluetooth*. [Online, accessed 18 May 2011]. <http://developer.android.com/guide/topics/wireless/bluetooth.html>

[11] D. Lu. (October 2011). *Android interface for pollution monitoring system*. University of New South Wales, Sydney, Thesis 2011.

[12] K. Bi. (October 2011). *Android interface for pollution monitoring system*. University of New South Wales, Sydney, Thesis 2011.

[13] Android Developers. (20 May 2011). *Location*. [Online, accessed 18 May 2011]. <http://developer.android.com/reference/android/location/Location.html>

[14] J. Rantakokko et al., "*Accurate and Reliable Soldier and First Responder Inder Positioning: Multisensor System and Cooperative Localization*" IEEE Wireless Communications, vol. 18, issue 2, April 2011, pp. 10-18.

[15] S. Draganov, M. Harlacher, L. Haas, M. Wenske and C. Schneider, "*Synthetic Aperture Navigation in Multipath Envi-ronments*" IEEE Wireless Communica-tions, vol. 18, issue 2, April 2011, pp. 52-58.

[16] L. M. Ni, D. Zhang and M. R. Souryal, "*RFID-Based Localization and Tracking Technologies*," IEEE Wireless Communications, vol. 18, issue 2, April 2011, pp. 45-51.

[17] H. Zhang. (October 2011). *Communication between Sensors and Mobile Phone*. University of New South Wales, Sydney, Thesis 2011.

[18] Wikipedia. (3 October 2011 at 10:56). *Haversine formula*. [Online, accessed 18 August 2011]. <http://en.wikipedia.org/wiki/Haversine_formula>.

[19] Microchip Technology Inc.(2008) *PIC16F631/677/685/687/689/690 Data Sheet*.

[20] Adeunis RF. (Aug 2007) ARF32 Bluetooth Modules - User Guide. Datasheet.

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 defualt
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.LocationManage
r.GPS_PROVIDER )) {
        Intent myIntent = new
Intent( Settings.ACTION_LOCATION_SOURCE_SETTINGS );
        startActivity(myIntent);
        }

    //locationManager.requestLocationUpdates(LocationManager.GPS_PROVI
DER, 0,200, locationListener);

    //locationManager.requestLocationUpdates(LocationManager.NETWORK_P
ROVIDER, 0, 200, locationListener);


    locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDE
R, 5000,0, locationListener);

    locationManager.requestLocationUpdates(LocationManager.NETWORK_PRO
VIDER, 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>10000l){
                    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 massage type 1 for control massage, 2 for data massage
     * */
    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.rawCoefficents[0] =
(byteBuf[0]<<8)|byteBuf[1];
                        dMessage.rawCoefficents[1] =
(byteBuf[2]<<8)|byteBuf[3];
                        dMessage.rawCoefficents[2] =
(byteBuf[4]<<8)|byteBuf[5];

    dMessage.coefficentMap.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: massage 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_inde
x], 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: massage 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 dataOutStream = 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");
    }
    dataOutStream = 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 {
                        dataOutStream.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 {
                dataOutStream.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();
            outStream.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 initallocation.");
                    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 sensorIdentifer) {
        switch (sensorIdentifer) {
        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.coefficentMap.keySet()){
```

```
            if (isSensorEnabled(sensorID)) {
                map.put(pollutantStringForSensor(sensorID),

   pollutantValueForSensor(msg.sensorData.sensorDataMap.get(sensorID),
msg.getReferenceVoltage(), msg.coefficentMap.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");
                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).toLogString());

            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 firstOutRange = minEntry[0]>600;
            boolean secondOutRange = minEntry[1]>600;

            // find possible exit

            if(firstOutRange){
                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(secondOutRange){

                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)asignLocation(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;// longtitude 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 asignLocation(double[][] route){
        Log.e("debug", "asign 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;
            //wrapper.setLocation(location);

            Log.w("int", "Long: "+location.getLongitude()+" Lat:
```

```
"+location.getLatitude());

            dataBuffer.add(wrapper);
        }
    }

    private void dataupload(DataWrapper wrapper){
        dataBuffer.add(wrapper);
        if (dataBuffer.size() >= MaxStandbyData) {
            Log.i("debug", "posting data online.");
            if (postData(dataBuffer)) {
                Log.i("debug", "data posted.");
                dataBuffer.clear();
            }
        }
    }

}
```

DeviceChooser.java

```
package com.unsw.pollutionsensor;

import java.util.Comparator;
import java.util.HashSet;
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.AdapterView.OnItemClickListener;
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 deviceIdentifier=0;
    public int valuePairCount = 0;
    public int rawCoefficents[] = new int[3];
    public SensorData sensorData;
    public double sensorCoefficents[] = 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[]>coefficentMap = new
HashMap<Integer,double[]>();

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

    public double getReferenceVoltage () {
        return Helpers.rawFloat16ToDouble (rawReferenceVoltage);
    }
    public double getCoefficient (int index) {
        return Helpers.rawFloat16ToDouble(rawCoefficents[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[][] AAto = {{-33.875077,151.220517}};
            return AAto;
        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}

};

}
```