Sampad Bhusan Mohanty
Under supervision of –
Prof. Paresh G. Kale
Prof. Debiprasad P. Acharya

Vehicle Monitoring for Open Cast Mines under Unreliable Cellular Network using ZigBee

Department of Electrical Engineering
NATIONAL INSTITUTE OF TECHNOLOGY ROURKELA

Vehicle Monitoring for Open Cast Mines under Unreliable Cellular Network using ZigBee

Thesis submitted to the

National Institute of Technology Rourkela

in partial fulfilment of the requirements for the degree of

Bachelor and Master of Technology

(Dual Degree)

in

Electrical Engineering
(Specialization in Control and Automation)

by

Sampad Bhusan Mohanty
(Roll Number: 711EE3097)

under the supervision of

Prof. Paresh Govind Kale
(Assistant Professor, Dept. of EE)

and

Prof. Debiprasad Priyabrata Acharya
(Associate Professor, Dept. of ECE)

May 2016

Department of Electrical Engineering
National Institute of Technology Rourkela

Department of Electrical Engineering
National Institute of Technology Rourkela

May 2016

Certificate of Examination

Roll Number: 711EE3097
Name: Sampad Bhusan Mohanty

Title of Thesis: Vehicle Monitoring for Open Cast Mines under Unreliable Cellular Network using ZigBee

We the below signed, after checking the thesis mentioned above and the official record book (s) of the student, hereby state our approval of the thesis submitted in partial fulfillment of the requirements for the degree of Bachelor and Master of Technology (Dual Degree) in Electrical Engineering at National Institute of Technology Rourkela. We are satisfied with the volume, quality, correctness, and originality of the work.

Prof. Debiprasad P. Acharya
Co-Supervisor

Prof. Paresh G. Kale
Supervisor

Department of Electrical Engineering
National Institute of Technology Rourkela

May 2016

Supervisor's Certificate

This is to certify that the work presented in this thesis entitled "Vehicle Monitoring for Open Cast Mines under Unreliable Cellular Network using ZigBee“ by Sampad Bhusan Mohanty, Roll Number 711EE3097, is a record of original research carried out by him under our supervision and guidance in partial fulfillment of the requirements for the degree of Bachelor and Master of Technology (Dual Degree) in Electrical Engineering. Neither this thesis nor any part of it has been submitted for any degree or diploma to any institute or university in India or abroad.

Prof. Debiprasad P. Acharya
Co-Supervisor

Prof. Paresh G. Kale
Supervisor

Dedicated to Lord Shri Krishna

Dedicated to Lord Jagganatha

Declaration of Originality

I, Sampad Bhusan Mohanty, Roll Number- 711EE3097 hereby declare that this thesis entitled "Vehicle Monitoring for Open Cast Mines under Unreliable Cellular Network using ZigBee" represents my original work carried out as an undergraduate and postgraduate student of NIT Rourkela and, to the best of my knowledge, it contains no material previously published or written by another person, nor any material presented for the award of any other degree or diploma of NIT Rourkela or any other institution. Any contribution made to this research by others, with whom I have worked at NIT Rourkela or elsewhere, is explicitly acknowledged in the thesis. Works of other authors cited in this thesis have been duly acknowledged under the section “References”. I have also submitted my original research records to the scrutiny committee for evaluation of my thesis.

I am fully aware that in the case of any non-compliance detected in future, the Senate of NIT Rourkela may withdraw the degree awarded to me on the basis of the present thesis.

May, 2016

NIT Rourkela *Sampad Bhusan Mohanty*

Acknowledgement

Behind everything, there are three causes – the immediate cause, the remote cause, and the ultimate cause. The ultimate cause is the Supreme Godhead – Krishna. He is the cause of all causes. I thank Him, in my small capacity, for all the abilities and opportunities He has blessed me with in my life. The remote causes are my parents and teachers who have dedicated a significant amount of their lives towards my upbringing and education. I would like to thank my teachers Mrs. Mary Martin and (late) Mrs. Tripti Sanyal (primary school) Mrs. Pratibha Sahoo, Mrs. Pushpanjali Singh, Shantilata Panda (secondary school), tutors Ratnakar sir, Sanatan sir, Raghunath sir and Niharika ma'am (secondary education), Dinesh Das sir, Chhatrapati sir and Karunakar Patra Sir (higher secondary education) for their contributions to my education, support and guidance at different phases of my life. I would also like to thank the people who have contributed to this project directly through their hard work and are the immediate cause of it. I am grateful to Prof. Paresh Govind Kale for allowing me to pursue my interests and letting me work on this project. I am thankful to Prof. Debiprasad Priyabrata Acharya for always motivating and keeping faith in me at the hardest of times. I would like to thank my friend Himansu Pradhan, senior Santosh Yerme and juniors Sagnik Basu, Sharat Kandregula and Abhijit Behera for their help in developing the hardware of the system.

I am thankful to NIT Rourkela for providing me an excellent undergraduate education and opportunities to discover my strengths and weaknesses. The scientific/philosophical thoughts and quotes written beside the roads all over the campus have always inspired me, encouraged me, and helped me reflect on the truth and reality of nature and life during the brightest and the darkest moment of my stay here. I am grateful to CYBORG Robotics Society and my seniors there for introducing me to the world of robotics. I am grateful for my time here during which I have made many 'to be' lifelong friends. I am grateful to my friends Swayan Jeet Mishra, Kishan Patel, Chinmay Garanayak and Siddhartha Tripathy for their help, support and motivation. My time here at NIT Rourkela wouldn't have been such a great fun and learning experience without these

people. I am thankful to my senior Amiya Kumar Samantray for his help, support and advice at all points of time.

I would like to thank the Akshaya Patra Foundation for providing us such delicious food (prasadam) every Sunday that relieved us from the academic stress every weekend. I would also like to thank my spiritual mentors Pancharatna Prabhu, Jagannath Prabhu, Vivekananda Prabhu and Dhananjaya Prabhu for enlightening my friends and me with the spiritual and Vedic knowledge. I would also like to thank my senior Balamiki Kumar for his encouragement, advice, love and affection.

At last but not the least, I would like to thank my family and relatives for their love, affection and everything else which cannot be put into words. Hare Krishna.

Abstract

Implementation of a vehicle monitoring system in regions without proper cellular network strength to establish a GPRS connection, like in an open cast mine, is presented in this thesis. Conventional vehicle monitoring systems push the acquired GPS coordinates over a GPRS connection to a Vehicle Monitoring Server (VMS). To accommodate for the lack of a reliable GPRS connection, a ZigBee Wireless Network is deployed at the location of interest, i.e. the open cast mine, to act as a gateway to the internet. ZigBee nodes, functioning as routers, are interfaced to Vehicle Tracking Equipment (VTE) which are installed on each of the vehicles to be tracked. VTE also contain a GPS receiver and a GSM/GPRS modem. A ZigBee node functioning as the ZigBee Network Coordinator is interfaced to an SBC-Single Board Computer (Raspberry Pi). This assembly is called Coordinating Gateway Equipment (CGE) and is placed at a suitable location where it has access to the internet either via a wired or wireless infrastructures like Ethernet or Wi-Fi or via a Cellular Network (GPRS). Though it is not strictly required, for the best performance, the location of CGE is chosen such that most of the region that are devoid of cellular network falls within one hop range of the ZigBee Coordinator interfaced to it. Whenever a VTE fails to deliver GPS coordinates over GPRS, it sends the same via the ZigBee network to the CGE which temporarily stores the received data in a database and uploads it to the VMS over the internet available to it. The VMS logs the data received, and plots the location of each vehicle on a map using Google Maps Service. Network Watchdog Timers of ZigBee nodes present in VTE are configured such as to allow automatic re-joining when vehicles leave and enter the ZigBee Network Range. The VTE, CGE, and VMS are all implemented using open source hardware and software platforms/libraries such as Arduino, Raspberry-Pi, Python, and Debian operating system.

The project, funded by Jindal Steel and Power Limited, was finally installed at Jindal Iron Mines - Tensa, Sundergarh Odisha, India for testing and has been successfully operating since April 2016.

Keywords: Vehicle Tracking/Monitoring, ZigBee, Internet of Things, Embedded System

Contents

Certificate of Examination i

Supervisor's Certificate ii

Declaration of Originality iv

Acknowledgement v

Abstract vii

Chapter 1 - Introduction 1

1.1 Introduction to the Problem 1

1.2 Motivation 2

1.3 Towards a solution 3

1.4 Design goals for the system 4

1.5 Thesis Overview 5

Chapter 2 - Architecture 6

2.1 Requirements 6

2.2 Wireless PAN selection 7

2.3 System Components 9

2.4 Hardware and Software 10

2.5 Summary 12

Chapter 3 - Hardware 13

3.1 Overview 13

3.2 Components 14

3.2.1 GSM/GPRS Modem 14

3.2.2 GPS Receiver 15

3.2.3 ZigBee Module 17

3.2.4 16 x 2 Alphanumeric LCD 18

3.2.5 The Microcontroller 19
3.2.6 The Single Board Computer 20
3.3 Interconnections 21
3.3.1 Vehicle Tracking Equipment 21
3.3.2 Coordinating Gateway Equipment 22
3.4 Assembled Hardware 24
3.5 Summary 26
Chapter 4 - Software 27
4.1 Overview 27
4.2 Component Details 28
4.2.1 GSM/GPRS Modem 28
4.2.2 GPS Receiver 31
4.2.3 XBee Module 31
4.2.4 Database for the Raspberry Pi 37
4.2.5 Helper Module for CGE 39
4.2.6 Web Service for VMS 40
4.3 Integration 42
4.4 Summary 43
Chapter 5 - Results and Conclusion 44
5.1 ZigBee Range Test 44
5.2 Installation 45
5.3 VTE and CGE Performance 46
5.4 Results 49
5.5 Conclusion 51
5.6 Scope of Future Work 51
References 52

A. Appendix 54
Visits to Jindal Iron Mines, Tensa, Sundergarh Odisha India 54
Python SIM900 modem Library (ISAsim900.py) 62
Arduino C++ SIM900 modem Library 66
Arduino C++ Library for XBee (xbeeapi.h) 70
Database Library for Raspberry Pi (VdbModel.py) 75
Web.py service cum Application 77

List of Figures

Figure 1.1: Cross section of an Open Cast Mine 2
Figure 1.2: Jindal Iron Mines, Tensa Sundergarh Odisha India 3
Figure 2.1: Overall System Architecture 7
Figure 2.2: Google Map of the Open Cast Mine 9
Figure 3.1: GSM/GPRS Modem employed in VTE and CGE 14
Figure 3.2: GPS Receiver and external antenna used in VTE 16
Figure 3.3: NMEA 0183 Sample Data from a GPS Receiver 16
Figure 3.4: XBee Pro Series 2B ZigBee Module 17
Figure 3.5: 16x2 Alphanumeric LCD 18
Figure 3.6: Arduino Mega 2560 19
Figure 3.7: Raspberry Pi 2 SBC 20
Figure 3.8: VTE Hardware Interconnection 21
Figure 3.9: CGE Hardware Interconnection 23
Figure 3.10: VTE Circuit Board 24
Figure 3.11: VTE Assembly 25
Figure 3.12: CGE Assembly 26
Figure 4.1: Modem Interface Abstraction Model 30
Figure 4.2: XBee API Generic Frame 33
Figure 4.3: XBee Receive Data Frames 33

Figure 4.4: XBee Transmit Data Frames 34
Figure 4.5: XBee API Library Model (xbeeapi.h) 36
Figure 4.6: Database Model of CGE 38
Figure 4.7: VMS Database Model 40
Figure 4.8: VTE software 42
Figure 4.9: CGE Software 42
Figure 5.1: XBee Range Test 44
Figure 5.2: CGE installed on a Light Post 46
Figure 5.3: VTE installed on a Vehicle 47
Figure 5.4: Live Monitoring 49
Figure 5.5: Vehicle speed data 49
Figure 5.6: Location Data Received at VMS 50
Figure 5.7: Status messages received from CGE at VMS 50
Figure A.1: Installation of the CGE 54
Figure A.2: CGE installed on a Light Post 55
Figure A.3: VTE installed inside a dumper's cockpit 56
Figure A.4: Vehicle Maintenance Shop 57
Figure A.5: Debugging a VTE 58
Figure A.6: Testing the CGE 59
Figure A.7: Debugging the CGE 60
Figure A.8: A picture of the mine at night 61

Chapter 1 - Introduction

1.1 Introduction to the Problem

With rapid advancement in mobile communication, cell phones and cellular networks have become ubiquitous. With cellular network companies advertising the diversity of their network coverage ranging over remote locations like forests and mountains, it might be difficult to think of a region without a network coverage. Despite all this, there are certain corner cases where the cellular network is either completely unavailable or the signal strength is insufficient for the required use. One such example is an open cast mine. Open cast mines have depths ranging in hundreds of metres. As the earth is constantly changing due to excavation, the area and depth of mine increases over time. Therefore, it is not suitable to install base transceiver stations (BTS) close to an open pit mine. Open pit mines contain mine benches which have heights typically ranging from 10 to 30 metres. These benches shade the cellular network signal (frequency ranging from 900MHz to 2500MHz corresponding to wavelengths 0.33 metres to 0.12 metres) causing rapid decay of the signal strength as we move down the mine benches. In such a case, the performance of a conventional vehicle monitoring system which relies on the cellular network is unsatisfactory.

A conventional vehicle monitoring system uses a vehicle tracking equipment (VTE) which is installed on each vehicle to be monitored. Each VTE has three primary components – a GPS receiver, a microcontroller and a modem to make a data connection to the internet over the cellular network. Although there are many profiles for making a data connection over a cellular network, the one which is most commonly used is General Packet Radio Service (GPRS) which is a service on top of Global System for Mobile Communication (GSM). First, the GPS receiver calculates its location by the help of the GPS satellites visible to it. The microcontroller acquires the position data from the GPS receiver and sends it to the vehicle monitoring server (VMS) after initiating a GPRS connection using the modem. The VMS logs the position data received in a database and then, on request of a user, fetches the stored positions from database and plots the position in a meaningful interface like a map for the user. The above conventional tracking system

operates well when cellular network is available but fails in the absence of it. Hence, the above method is not suitable for vehicle monitoring in open cast mines suffering from the poor network.

1.2 Motivation

Open cast mines generate raw materials worth crores of rupees every day. Even a delay as little as fifteen minutes in the transport chain can cost a company a lot of money. With dozens of vehicles operating in an open cast mine like earth movers, dumpers and trucks, it would certainly be a herculean task for a team to monitor each vehicle manually without the help of technology and automation. A vehicle monitoring system can help the team to monitor all the vehicles simultaneously and from anywhere using the internet. Also, the vehicle position and speed data captured from the monitoring system can be post-processed to find out the flaws in the transport chain to further optimise the transportation routes and can increase the production of the mine significantly.

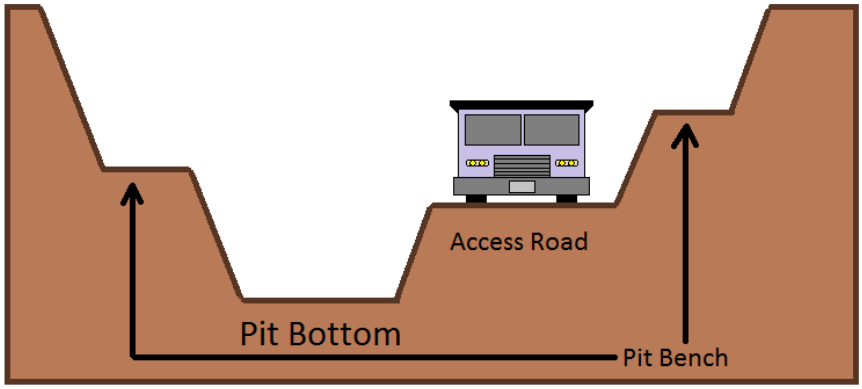

Figure 1.1: Cross section of an Open Cast Mine [1]

The issue of poor cellular service is general to all open cast mines. The problem is different from that of a conventional vehicle monitoring system and hence is a fascinating engineering challenge which motivated us to bring the solution to life. The project work

was funded by Jindal Steel and Power Limited (JSPL) located at Tensa Sundergarh Odisha, India. So we had an opportunity to deploy our system and test its performance in the field.

Figure 1.2: Jindal Iron Mines, Tensa Sundergarh Odisha India

1.3 Towards a solution

To address the above problem, a system to monitor vehicles in an unreliable cellular network is needed. This system has to function as a conventional vehicle monitoring system in the availability of a cellular network. In the case of unavailability of a cellular network, the system has to implement another way to push the GPS data to the vehicle monitoring server. This other way to acquire the positional data from the vehicles is addressed by creating a wireless Personal Area Network (PAN) whose coverage includes the open pit mine and all of its benches. In the situation when the location data cannot be sent over GPRS connection, it would be forwarded using this wireless PAN. This wireless PAN, which consists of many nodes, would have at least one (or more) nodes acting as a gateway (gateways) to the internet by connecting to a wired or wireless infrastructures like LAN (Ethernet) or Wi-Fi. If a LAN or Wi-Fi is unavailable, then a GPRS connection can be used if any of the gateways fall under cellular network coverage. Thus, the position data can be

routed via this wireless PAN to the internet to reach the vehicle monitoring server. The rest of the thesis is based on the work done towards implementing a stable and working system which incorporates this solution. In our solution, we have only one gateway which also happens to be the coordinator for the wireless PAN, and hence, it is called as the Coordinating Gateway Equipment (CGE).

1.4 Design goals for the system

The primary goals for the system are

- Must function as conventional vehicle monitoring in the availability of the cellular network.
- Must fall-back to wireless PAN when cellular network not available or signal is weak.
- Intelligent selection between the cellular network and the wireless PAN for low latency of position data and near-realtime operation.
- Robust and stable wireless PAN coverage over all benches of the open pit mine.
- The vehicle tracking equipment(VTE) must be energy efficient as they will work on vehicle battery.
- The mechanical design of VTE must specifically account for the vibrations occurring in the heavy vehicles like earth movers and the dumpers which operate on mine benches. This problem does not exist in a conventional VTE as there are no vibrations of this scale in vehicles operating on maintained roads.
- Two factors influence the location of Coordinating Gateway Equipment –
  I) Ability of the Wireless PAN (WPAN) coordinator node to cover as large region of the mine benches as possible under its one hop range for the best performance
  II) The availability of a connection to the internet (LAN, Wi-Fi or GPRS)
  A location that satisfies the above two constraints might not have a readily available power supply, and the CGE might have to run off solar energy and backups. Hence, the CGE needs to be energy efficient.

1.5 Thesis Overview

An introduction to the problem is given in the first chapter, formalising the problem statement and citing the motivation behind the work presented in this thesis. It also discusses the solution to the problem and how it can be implemented by pointing out the design goals of the system. The second chapter lays out a general architecture for the system after discussing the requirements to be fulfilled and the constraints to be satisfied by it. The details of the hardware modules used to build Vehicle Tracking Equipment and the Coordinating Gateway Equipment provided in chapter three. It also details on the interconnection between the various hardware modules, their circuit diagrams, and working. The fourth chapter includes the detail about the development of the system software and the different programming languages and software tools used in the process.

Chapter 2 - Architecture

2.1 Requirements

From earlier discussions we concluded that the Coordinating Gateway Equipment(CGE) must be positioned at a location such that it has access to the internet (via LAN, Wi-Fi or Cellular i.e. GPRS). The location must also be such that the WPAN coordinator node present in it has coverage over as large area as possible covering the mine benches in a single hop wireless radio range. Now if there are more than one options available like LAN, Wi-Fi, and GPRS, then we might have some flexibility in positioning the CGE. The availability of a power source nearby is also a factor determining the location of the CGE. The most favourable case would be to have a Wi-Fi access point near the CGE so that we have the flexibility of wirelessly connecting to a reliable source of internet. The next favourable case would be to have a LAN (Ethernet) connection to the CGE. Ethernet, being a wired connection, would limit the choice of possible positions of installing the CGE. However, it is more favoured than a cellular data connection like GPRS as Ethernet would provide a stable and guaranteed internet connection, unlike GPRS which is a best effort service. At the same time, it is unlikely for an open cast mine to have Wi-Fi or Ethernet connectivity at mine benches as wirings are very limited in a region of constantly moving earth for extraction of ores and movement of large and heavy machinery and vehicles. However, it is very much possible to have cellular coverage at the first few benches. It happens to be just the case at Jindal Iron Mines (Tensa Sundergarh Odisha, India), the place where the system has to be deployed. With cellular signal available at the top benches, these high benches are favourable to position the CGE as the WPAN node in the CGE, being placed higher above all other benches, will also cover a maximum of the mine benches in a single hop RF range.

The overall system can be broken down into three main components –

1. The Vehicle Tracking Equipment (VTE)
2. The Coordinating Gateway Equipment (CGE)
3. The Vehicle Monitoring Server (VMS)

The VTS is attached to each vehicle that has to be monitored. The CGE is fixed at a suitable location where it has access to the cellular network for a data connection (GPRS). Its location is also so chosen as to provide coverage to a maximum number of the mine benches in one hop radio range for the best system performance. The last component, VMS is a computer in the cloud, programmed to run a service that continuously listens for incoming position data from the VTE or the CGE over the internet. Figure 2.1 shows the position of the CGE and how it has been placed to lie in the coverage of cellular network while providing WPAN coverage to the mine benches. It also shows the communication between the CGE and the VTEs present on the different vehicles. The vehicles can communicate their location data directly to the VMS via GPRS if under cellular coverage or can forward it to the CGE which then communicates the same to the VMS over GPRS.

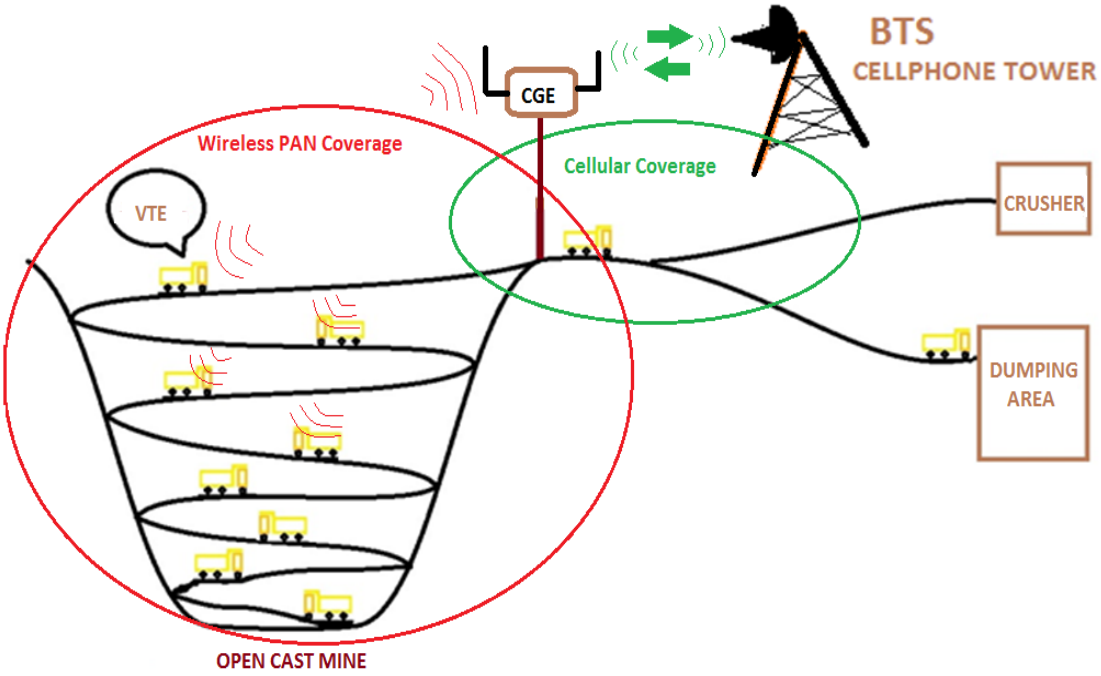

Figure 2.1: Overall System Architecture

2.2 Wireless PAN selection

With the basic architecture of the system already laid out, now we have to choose the right Wireless Personal Area Network (WPAN) that meets our requirements and, more importantly, that satisfies our constraints. Open cast mines have different shapes, and the

largest diameter of the area might range in hundreds of metres to more than a kilometre. Figure 2.2 shows the map of the iron ore mines where the system is to be deployed.

The WPAN should be selected such as to bring the whole open cast mine under coverage using the least number of hops. Another constraint on the WPAN is energy consumption. The WPAN should not be power hungry as the nodes will be running off from battery of vehicles or solar power or a backup battery. The nodes must also provide a simple method to interface with a microcontroller. Cost per node is another major factor influencing the choice of the WPAN. A large number of online resources, users and community support means we can get quick help on the problems faced during implementation of the WPAN and hence also influence the WPAN selection. Hence, the important factors that will affect our choice of WPAN are –

1. Single hop wireless range of the nodes must be in the range of hundreds of metres.
2. The nodes must be energy efficient.
3. The simplicity of interfacing to a resource constrained microcontroller.
4. Cost per node.
5. Learning resources available for the WPAN and community size and support.

Having discussed the requirements and constraints, we list the most common WPAN options available to us –

1. Bluetooth
2. Wi-Fi
3. ZigBee

Bluetooth has a limited range of few tens of metre. Wi-Fi, on the other hand, may have a range up to a few hundreds of metres but is very energy hungry and is optimised not for energy but speed. On the other side, ZigBee is a protocol that runs atop IEEE 802.15.4 which was designed to be an energy-efficient MAC layer protocol. Also, ZigBee offers range varying from hundreds of metres to well above a kilometre. Digi International Inc. is a company that offers ZigBee nodes under its ‘XBee’ trademark which use a UART interface to communicate with a microcontroller. At 63mW RF power, these nodes offer a single hop range of about a kilometre. Digi’s XBee RF modules have a large user community and an extensive number of online learning resources and tutorials. It is,

therefore, an excellent choice for our application. Hence, we chose Digi International's XBee Pro Series 2B nodes running ZigBee protocol to serve as our wireless Personal Area Network.

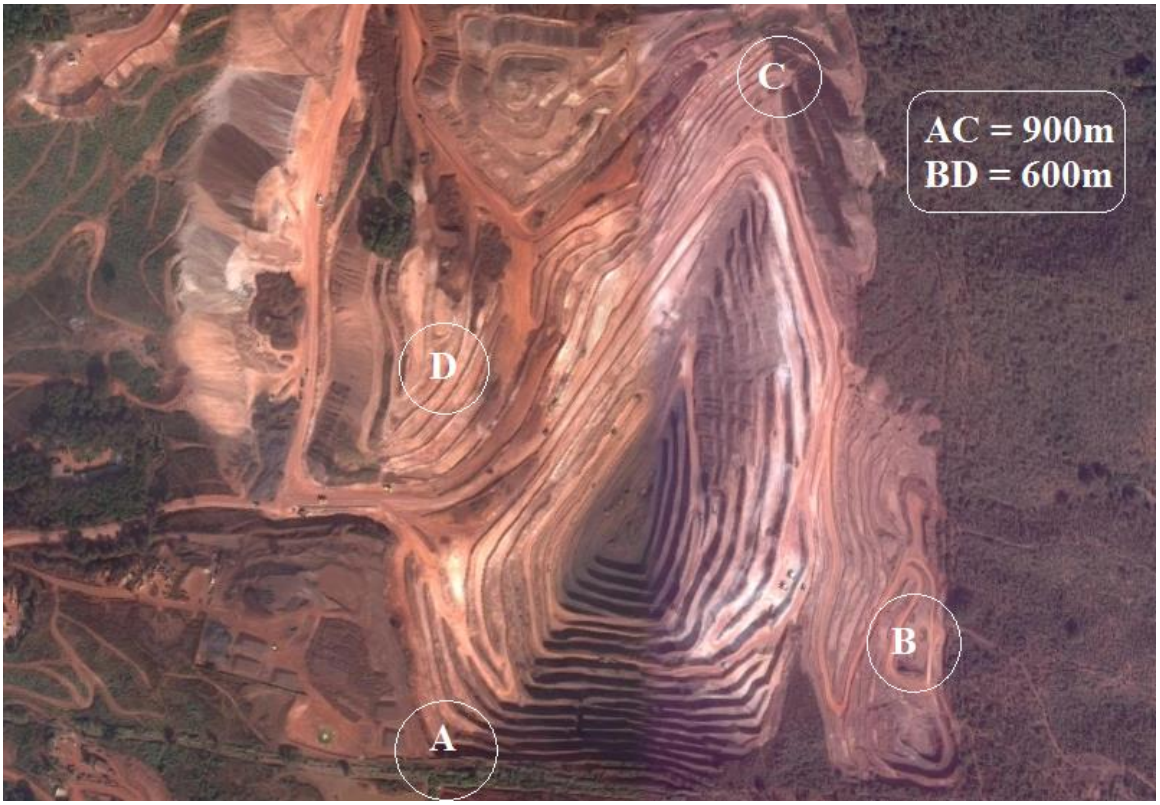

Figure 2.2: Google Map of the Open Cast Mine

2.3 System Components

The three most important elements of the system we mentioned earlier are –

1. The Vehicle Tracking Equipment (VTE)
2. The Coordinating Gateway Equipment (CGE)
3. The Vehicle Monitoring Server (VMS)

The vehicle tracking equipment, as already discussed, needs to function as a conventional tracking device in the availability of cellular coverage, i.e., it would send the position data over GPRS to the vehicle monitoring server. Only in the case of poor or unavailable cellular network will it behave differently by sending the same positional data

over the ZigBee WPAN. Also, it has to figure out intelligently when to switch back and forth between the cellular and the ZigBee PAN for sending the position data.

The Coordinating Gateway Equipment (CGE) has a more complex task to do. It has to listen over the ZigBee WPAN for any incoming position data from any of the VTEs, log it into a temporary database, to send it to the VMS over GPRS and finally to remove the corresponding entries from the database. It, therefore, must need a more powerful computational device than a simple microcontroller we would use for the VTEs.

Finally, the Vehicle Monitoring System is a Web Service (and a Web Application too, as it also provides a browser-based interface) running either on a cloud server or a physical computer that has a dedicated internet connection and a public IP address. In addition to a dedicated public IP address, a domain name is also mapped to the server's public IP for convenience in accessing it. This server runs a web service which takes care of the following tasks –

4. Logging position data received either from a VTE or the CGE into a database.
5. Rendering a map and plotting the positions of the vehicle(s) after retrieving the same from the database on request of an authorised user.
6. Running postprocessing algorithms on the acquired data, finding structure and pattern from it, to make a report of the delays caused in the transportation chain.

The VTE and the CGE required both hardware and software development. The VMS does not need a hardware as long as we are using a cloud server to implement it. Development for the VMS is completely software based requiring a fair knowledge of computer networking concepts.

2.4 Hardware and Software

For Vehicle Tracking Equipment and the Coordinating Gateway Equipment, both hardware and software were developed whereas the Vehicle Monitoring Server was completely

developed in software. The Vehicle Monitoring System is a web application which runs on a cloud computer or a computer with a dedicated internet connectivity.

The necessary hardware blocks of the VTE are –

1. Microcontroller Unit
2. GPS receiver
3. GSM/GPRS modem
4. ZigBee WPAN node (acting as Router)
5. Power circuit
6. Body/ Casing

Moreover, that of the CGE are –

1. Microprocessor Unit – Single Board Computer
2. GSM/GPRS modem
3. ZigBee WPAN node (acting as Coordinator)
4. Power Circuit
5. Solar panel and backup battery
6. Body / Casing

ZigBee nodes can be programmed to operate in one of the three following modes –

- Coordinator
- Router
- End Device

Every ZigBee WPAN has exactly one Coordinator and one or more Routers and End Devices. The microcontroller unit chosen for the VTE is an Arduino Mega 2560 board, and the microprocessor unit that was selected for the CGE is a Raspberry Pi 2 Single Board Computer. The software for the VTE is developed entirely in C++ using Arduino [2] IDE. The software for the CGE is primarily written in Python with a few Shell scripts written to automate initial setup and configuration. The Vehicle Monitoring Server was developed entirely in Python. All the software developed are built using open source software libraries and modules.

The necessary software components developed for the Vehicle Tracking Equipment are –

1. GPS receiver interface
2. GSM/GPRS modem interface
3. ZigBee node (XBee) interface
4. LCD interface

   Moreover, for the Coordinating Gateway Equipment are –

1. GSM/GPRS interface
2. ZigBee node (XBee) interface
3. Database for Vehicles and their Logs
4. Maintenance Scripts

The crucial software components for the Vehicle Monitoring Server are –

A. A python based web application
   a. Web-based frontend for user interaction
   b. Application backend for data logging and authentication of the user.
B. A production quality front facing web server (to serve static files and act as a reverse proxy server)

2.5 Summary

- The requirements of the system and the constraints imposed on it were discussed.
- The system comprises of three main components – VTE, CGE, and VMS
- The hardware and software of the VTE, CGE and VMS are briefed.
- Each ZigBee node operates in one of the three modes – Coordinator, Router and End Devices
- VTE contains a GPS receiver, a GPRS modem and a ZigBee node acting as Routers and a microcontroller unit
- The CGE contains a GPRS modem, a ZigBee node acting as Coordinator and a Single Board Computer

Chapter 3 - Hardware

3.1 Overview

The last chapter discussed the architecture of the overall system. The three most important elements of the system are the Vehicle Tracking Equipment, Coordinating Gateway Equipment, and the Vehicle Monitoring Server. The Vehicle Monitoring System is a web service and is developed entirely in software. The Vehicle Tracking Equipment and the Coordinating Gateway Equipment have hardware and software components to them. The hardware modules for the VTE and the CGE were discussed in the last chapter. The VTE and CGE, having many similar functions as connecting to the internet over GPRS and talking over the ZigBee PAN, are made up of hardware modules that are common to both. The hardware components that are common to both the VTE and the CGE are –

- GSM/GPRS modem
- ZigBee node
- Power circuit

The two differ from each other on the type of computational device used –

- A microcontroller (used in VTE)
- A Single Board Computer (used in CGE)

Apart from these similarities and differences, the other components used are –

- GPS Receiver (VTE)
- TTL to RS232 Level Shifter (VTE)
- USB to UART-TTL Converter
- 16 x 2 Alphanumeric LCD
- 5V to 3.3V Linear Voltage Regulator

The GSM/GPRS modem, used in both the VTE and the CGE, are based on the SIM900A wireless module from SIMCom Wireless Solutions Ltd, which supports dual-

band GSM at frequencies 900MHz and 1800MHz.The ZigBee node is implemented using a Digi International XBee Pro Series 2B module. The GPS receiver is a Skylab SKG13C module which uses a MediaTek 3329 chip. The microcontroller used is an Arduino Mega 2560 development board which implements ATmega 2560 which is an 8-bit microcontroller from Atmel Corporation. The Single Board Computer is a Raspberry Pi 2nd Generation computer from raspberrypi.org. The power circuit consists of one or more buck converter to step down voltage from 12V to 5V or 7V as required. Next section discusses the details of the individual hardware components used for building the Vehicle Tracking Equipment and the Coordinating Gateway Equipment.

3.2 Components

3.2.1 GSM/GPRS Modem

The GSM/GPRS modem used in both VTE and CGE is made by 'rhydoLABZ India' based on SIM900A SoC wireless module from SIMCom Wireless Solutions Ltd. The SIM900A supports Dual Band GSM/GPRS and supports frequencies at 900MHz and 1800 MHz. It can be interfaced to a microcontroller via a UART interface at selectable TTL level of 3.3V and 5V. It supports baud rate from 9600 – 115200. The default setting is again 9600/8-N-1 and this what we have used in both VTE and CGE.

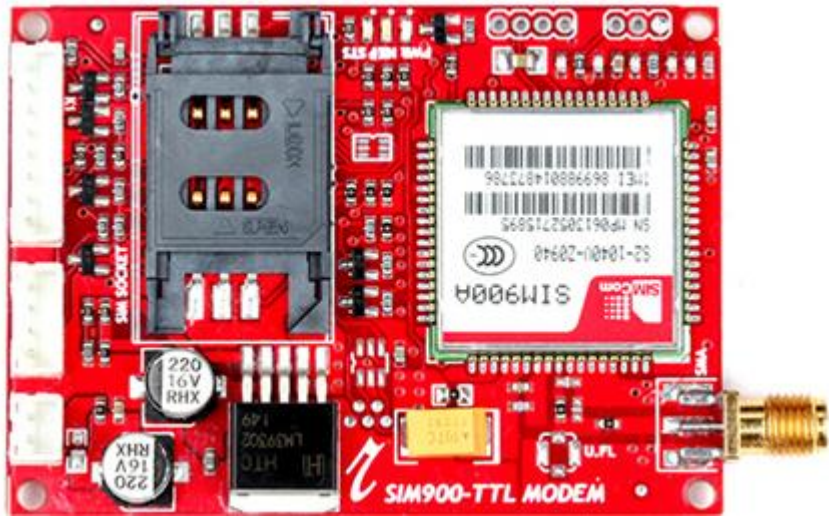

Figure 3.1: GSM/GPRS Modem employed in VTE and CGE

Some features of this GSM/GPRS modem are –

- Input Voltage: 5V-12V DC
- Normal operation temperature: -20 °C to +55 °C
- GPRS mobile station class B
- GPRS Class 10: max. 85.6 kbps (downlink)
- Control via AT commands (GSM 07.07, 07.05 and SIMCom enhanced AT Commands)
- Compliant to GSM phase 2/2+
    - Class 4 (2 W @850/ 900 MHz)
    - Class 1 (1 W @ 1800/1900MHz)
- Inbuilt PPP-stack
- Implements UDP and TCP stack
- UART interface with selectable logic levels 3.3V and 5V
- Certifications: CE, FCC, RoHS

3.2.2 GPS Receiver

The GPS Receiver used is made by Skylab M&C Technologies with part name SKG13C. It implements a MediaTek MT3339 GPS System on Chip (SoC). Features of this SoC are–

- Ultra-High sensitivity of -165dBm
- Extremely fast TTFT (Time To First Fix)
- low power consumption of 45mW@3.3V
- A wide operating temperature of -40° C to 85° C
- Supports NMEA-0183 protocol over UART at 3.3V Logic Level.

Figure 3.2: GPS Receiver and external antenna used in VTE

```
$GPGGA,092750.000,5321.6802,N,00630.3372,W,1,8,1.03,61.7,M,55.2,M,,*76
$GPGSA,A,3,10,07,05,02,29,04,08,13,,,,,1.72,1.03,1.38*0A
$GPGSV,3,1,11,10,63,137,17,07,61,098,15,05,59,290,20,08,54,157,30*70
$GPGSV,3,2,11,02,39,223,19,13,28,070,17,26,23,252,,04,14,186,14*79
$GPGSV,3,3,11,29,09,301,24,16,09,020,,36,,,*76
$GPRMC,092750.000,A,5321.6802,N,00630.3372,W,0.02,31.66,280511,,,A*43
$GPGGA,092751.000,5321.6802,N,00630.3371,W,1,8,1.03,61.7,M,55.3,M,,*75
$GPGSA,A,3,10,07,05,02,29,04,08,13,,,,,1.72,1.03,1.38*0A
$GPGSV,3,1,11,10,63,137,17,07,61,098,15,05,59,290,20,08,54,157,30*70
$GPGSV,3,2,11,02,39,223,16,13,28,070,17,26,23,252,,04,14,186,15*77
$GPGSV,3,3,11,29,09,301,24,16,09,020,,36,,,*76
$GPRMC,092751.000,A,5321.6802,N,00630.3371,W,0.06,31.66,280511,,,A*45
```

Figure 3.3: NMEA 0183 Sample Data from a GPS Receiver

The NMEA-0183 is a specification for communication between marine electronics standardised by the National Marine Electronics Association. It uses simple ASCII characters over a serial communication protocol (UART in our case). The GPS receiver emits NMEA-0183 sentences over UART in 8-N-1(8 data bits, no parity, 1 stop bit) data format and a configurable baud rate. The default baud rate is 9600, and this is the configuration we use in the VTE, which is commonly denoted by 9600/8-N-1.Sample

NMEA-0183 output from a GPS receiver is shown in figure 3.3 (Source: wikipedia.org/wiki/ NMEA_0183).

3.2.3 ZigBee Module

The ZigBee nodes used in both VTE and CGE are implemented using XBee Pro Series 2B modules (Part No. XBP24CZ7SIT-004) from Digi International Inc. The XBee modules implement the ZigBee protocol stack and can be interfaced to host microcontroller via UART at Logic level of 3.3V. These modules have a single hop distance of around one kilometre. The modules can be flashed with corresponding firmware to operate as ZigBee Coordinator, Routers and End Devices. The modules have rated RF power of 63mW and have RPSMA connectors for attaching the antenna.

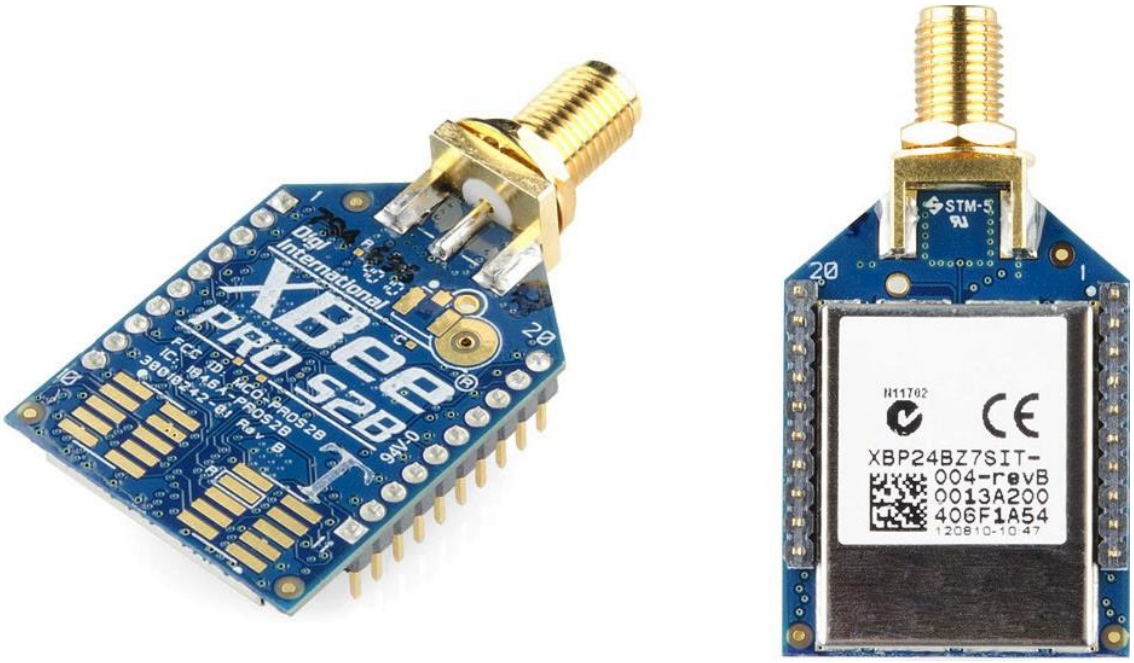

Figure 3.4: XBee Pro Series 2B ZigBee Module

Some of the specifications of the XBee Pro Series 2B (Part No. XBP24CZ7SIT-004) are –

- Operating Voltage: 2.7 V ~ 3.6 V
- Protocol: ZigBee®
- Frequency: 2.4GHz
- Raw Data Rate: 250kbps

- Power – Output: 18dBm
- Sensitivity: -102dBm
- Current – Receiving: 47mA ~ 62mA
- Current – Transmitting: 205mA ~ 220mA
- Operating Temperature: -40°C ~ 85°C
- Antenna Type: RP-SMA
- Serial Interfaces: SPI, UART
- Package / Case: Module

3.2.4 16 x 2 Alphanumeric LCD

The 16 x 2 alphanumeric LCD has an inbuilt controller which takes care of the actual low-level task of displaying pixels on the LCD. It provides an interface via a selectable 8-bit or 4-bit wide parallel data and command bus. We have selected 4-bit wide data bus to reduce the number of pins and wirings needed to interface with the Arduino. In this configuration, only seven pins are used (Data-4 through Data-7 and the control pins Enable, Read/Write, and Register Select).

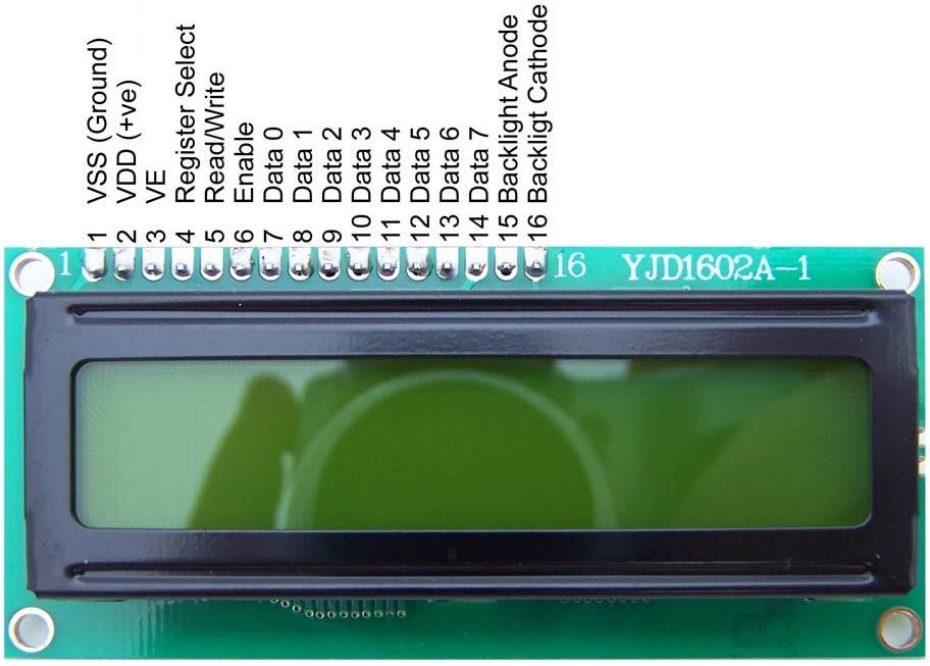

Figure 3.5: 16x2 Alphanumeric LCD

3.2.5 The Microcontroller

The microcontroller used in the VTE is an Arduino Mega 2560 development board from www.arduino.cc. It receives the position data from the GPS receiver, formats it properly to send to VMS, and then send it to the VMS over GPRS using the GSM/GPRS modem. Some key specifications of the Arduino Mega 2560 are –

- Microcontroller ATmega2560
- Operating Voltage 5V
- Input Voltage (recommended) 7-12V
- Input Voltage (limit) 6-20V
- Digital I/O Pins 54 (of which 15 provide PWM output)
- Analog Input Pins 16
- DC Current per I/O Pin 20 mA
- Flash Memory 256 KB of which 8 KB used by bootloader
- SRAM 8 KB
- EEPROM 4 KB
- Clock Speed 16 MHz

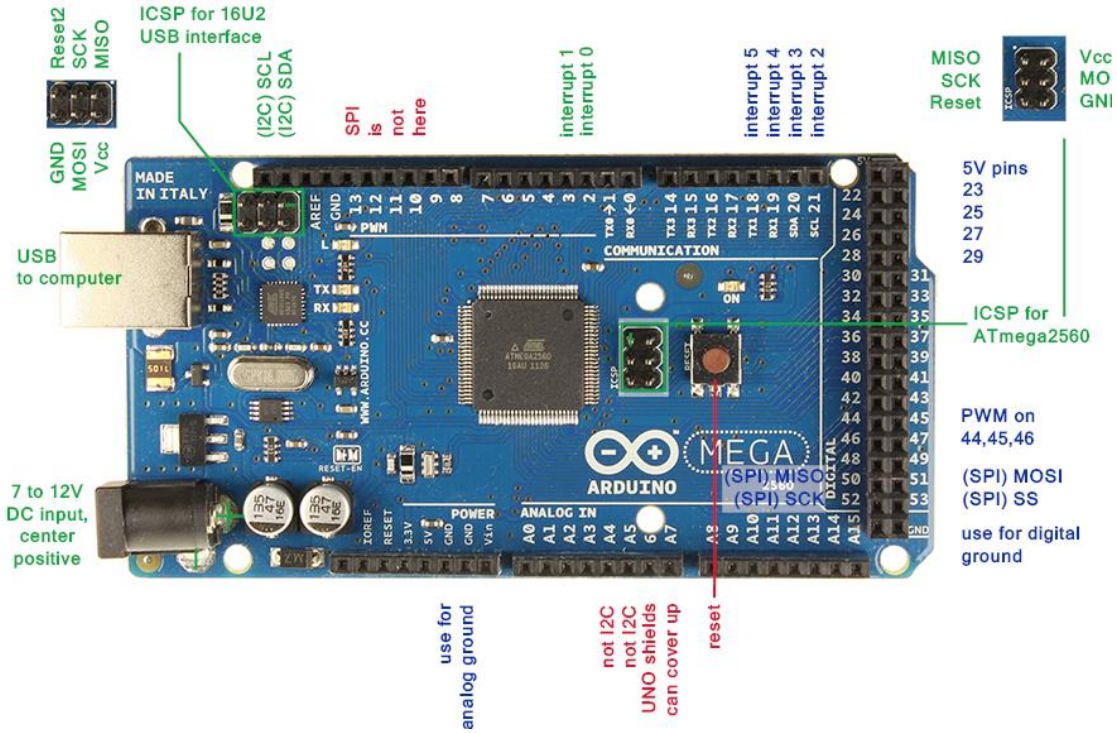

Figure 3.6: Arduino Mega 2560

3.2.6 The Single Board Computer

The CGE handles more complex tasks than VTEs, thus, requiring more computational power and resource than that of a VTE. The Raspberry Pi 2 is a single board computer (SBC) [3] that consumes a small amount of energy and runs a specially designed Debian operating system [4] and is a perfect candidate for our application.

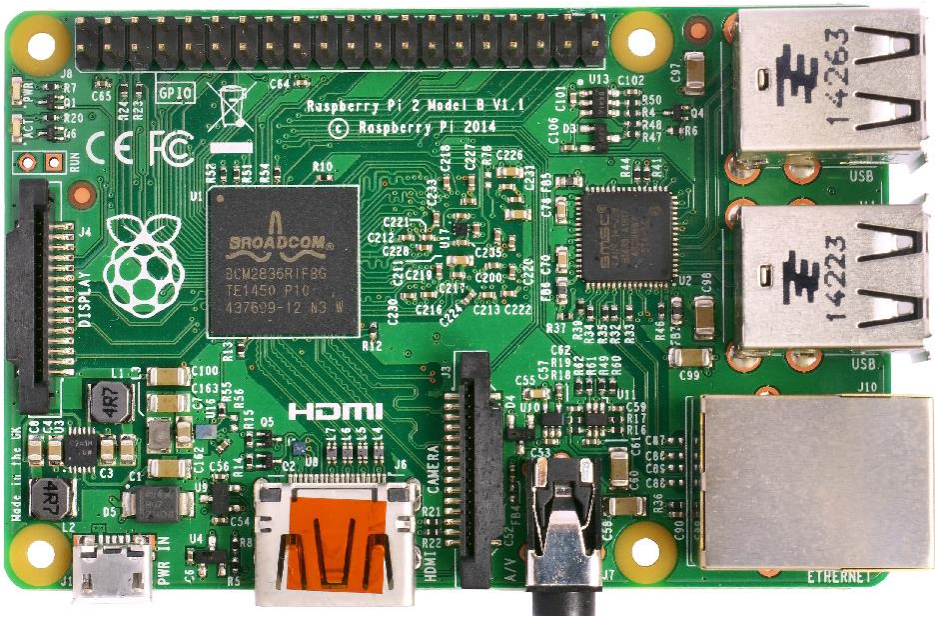

Figure 3.7: Raspberry Pi 2 SBC

The Raspberry Pi 2 has the following specifications –

- A 900MHz quad-core ARM Cortex-A7 CPU
- 1GB RAM
- SD Card as Secondary Memory (Support up to 32GB)
- 4 USB ports
- 40 GPIO pins
- Full HDMI port
- Ethernet port
- Camera interface (CSI)
- Display interface (DSI)

- Micro SD card slot
- Voltage: 5V
- Current: 500mA – 1500mA

3.3 Interconnections

3.3.1 Vehicle Tracking Equipment

The Arduino Mega 2560 has four UART serial ports, namely Serial-0 through Serial-3. The GSM/GPRS modem, GPS receiver, and the XBee module use the serial ports Serial-1 Serial-2 and Serial-3 respectively taking up three serial ports. The Serial-0 is used for uploading the code(firmware) developed into the Arduino board and for debugging the code running on it using a serial console. All the serial ports are configured for 9600/8-N-1 baud-rate and data-format. RS-232 level shifters are used because the XBee module is mounted a little higher than the vehicle's height at a distance of 3 to 5 metres from the rest of the circuit. RS-232 signalling is suitable for communicating over longer distances than TTL because of increasing capacitance due to increasing wire length. The interconnection between the different hardware components is shown in figure 3.6.

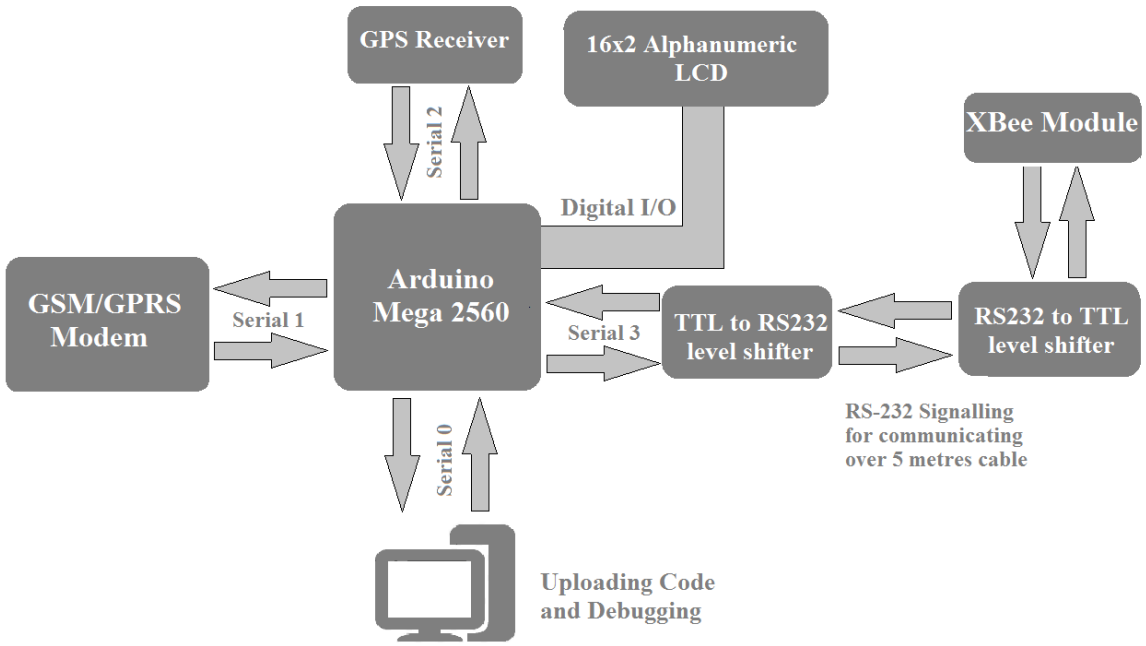

Figure 3.8: VTE Hardware Interconnection

The power circuit and supply are omitted from the illustration for simplicity. The power circuit consists of a buck converter converting 12V to 7V to feed the GSM/GPRS modem and the Arduino board. The XBee module and the TTL to RS-232 level shifters draw their power from the Arduino board's regulated 5V supply. The GPS receiver is supplied 3.3V from the output of a 5V to 3.3V linear voltage regulator which draws its supply from the regulated 5V of the Arduino board.

Each VTE consists of the following hardware components –

- GSM/GPRS Modem (1)
- GPS Receiver (1)
- XBee Pro Series 2B ZigBee Module (1)
- Arduino Mega 2560 Microcontroller board (1)
- TTL – RS232 level shifter (2)
- 5V – 3.3V Linear Voltage Regulator (1)
- 16 x 2 Alphanumeric LCD (1)
- Buck Converter (1)

3.3.2 Coordinating Gateway Equipment

Two USB to UART converters are required to interface the Raspberry Pi 2 to the GPRS modem and the XBee module, thus, taking up two USB ports of the Raspberry Pi 2. The two converters are from a different manufacturer so that software running on Raspberry Pi can automatically detect the two devices i.e. GPRS modem and the XBee module. It is certainly possible to distinguish between the two by querying the devices from software commands and check the responses when the two converters are the same. However, to keep things simple, we do not use the query-response method and instead, use the manufacturer name of the converter to identify and distinguish between the two.

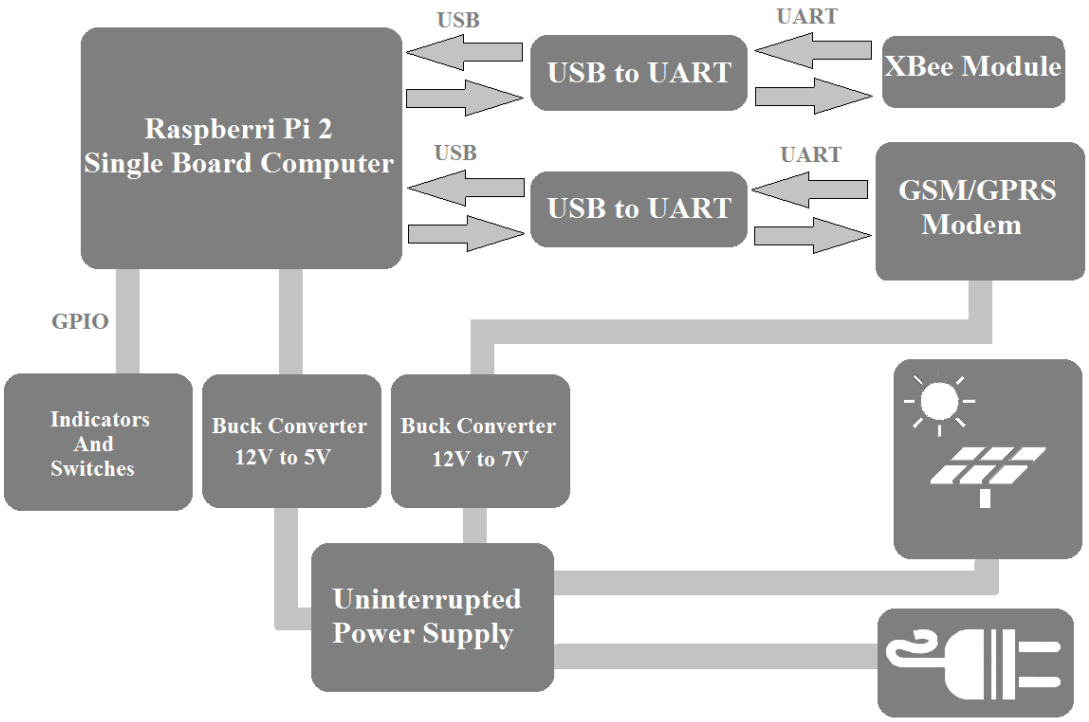

Figure 3.9: CGE Hardware Interconnection

The CGE consists of the following hardware components

- GSM/GPRS Modem (1)
- XBee Pro Series 2 ZigBee Module (1)
- Raspberry Pi 2 Single Board Computer
- USB to UART-TTL Serial Converter (2)
    - One from Future Technology Devices International Ltd – Part name FT-232
    - Another from Silicon Labs – Part name CP2103
- Buck Converter

Because CGE is the only gateway to the internet from ZigBee PAN network, its continuous operation is very crucial for the performance of the system in near real-time. An Uninterrupted Power Supply (UPS) system is used to keep CGE running during power outages during the night and cloudy days. The UPS outputs 12V DC and takes 12V DC input from the solar cells and also from 220V AC mains supply. The UPS can back up the CGE for nearly six hours continuously.

3.4 Assembled Hardware

CadSoft Eagle PCB Design Software was used to design the VTE assembly circuit board.

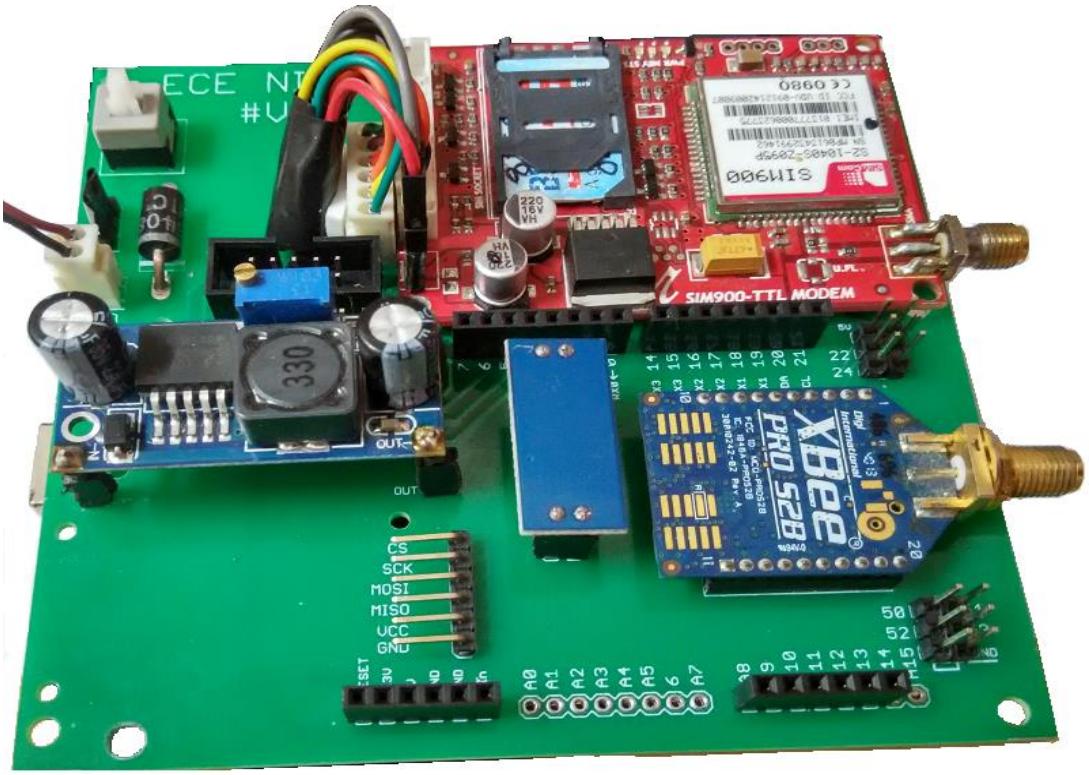

Figure 3.10: VTE Circuit Board

The casing for both VTE and CGE has been developed using SolidWorks. The VTE casing and circuit have to handle vibrations of the heavy vehicles. The casing for VTE and CGE were made using acrylic sheets after precision laser cutting. Casings for the XBee Module were 3D printed. After installing VTE in 35-tonne dumpers for initial testing, the components fell out of circuit as the casing failed to sustain the vibrations. The acrylic case was redesigned to make the model more compact and sturdy. Provision for mounting springs also was added to dampen the vibrations in the revision design.

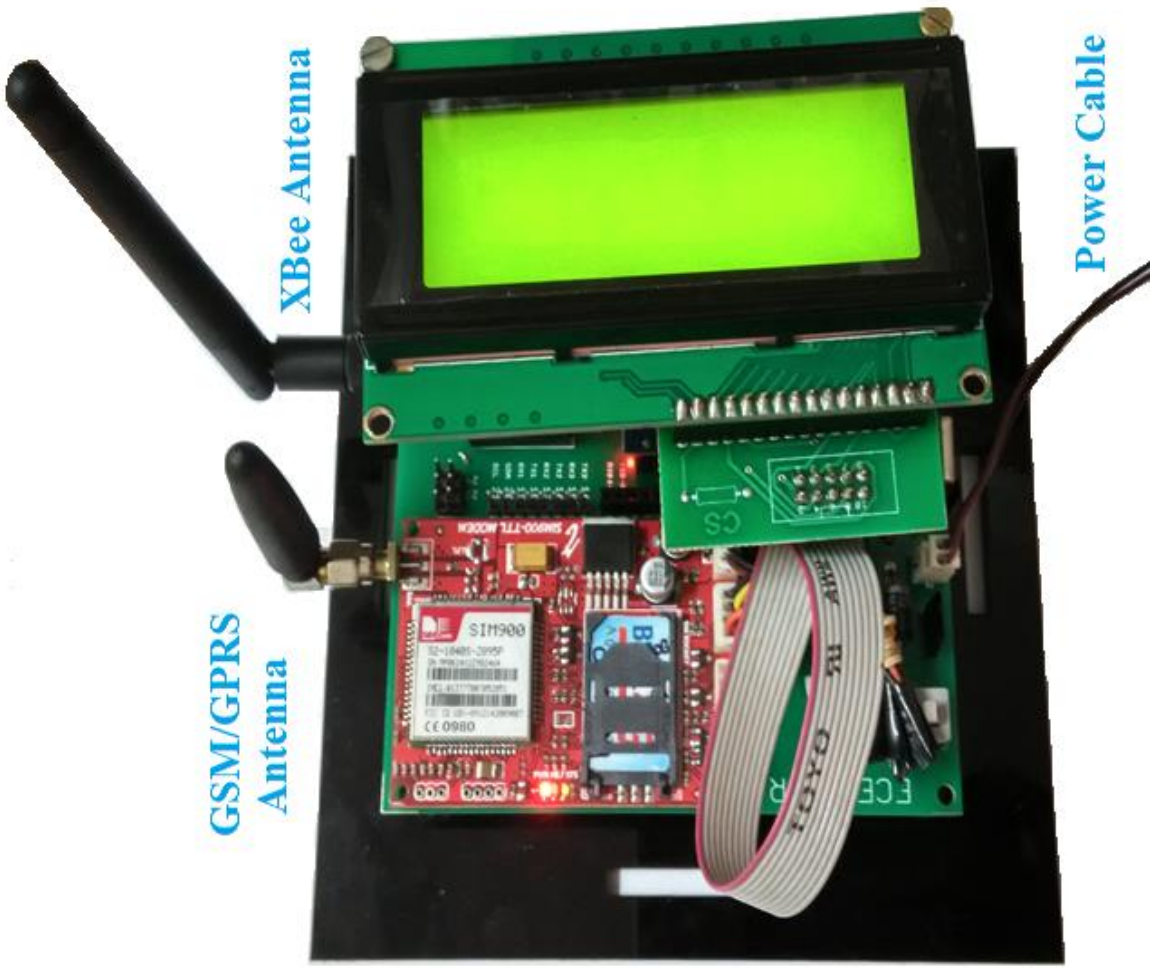

Figure 3.11: VTE Assembly

For the CGE, only the power circuits and the indicator and buttons circuit were designed. Unlike the VTE, which needs to be mounted on vehicles, the CGE will be installed on a stationary tower. Therefore, it does not have a central circuit board to assemble all the different components to keep parts from falling out of vibrations. Care has been taken to make the CGE casing water resistant and to prevent water formation due to moisture. The XBee module is contained in a 3D printed casing, and it connects to the CGE using a one metre USB cable. The XBee module casing is waterproof and is fixed at a suitable position and direction on the tower so as to provide radio line of sight to the mine benches.

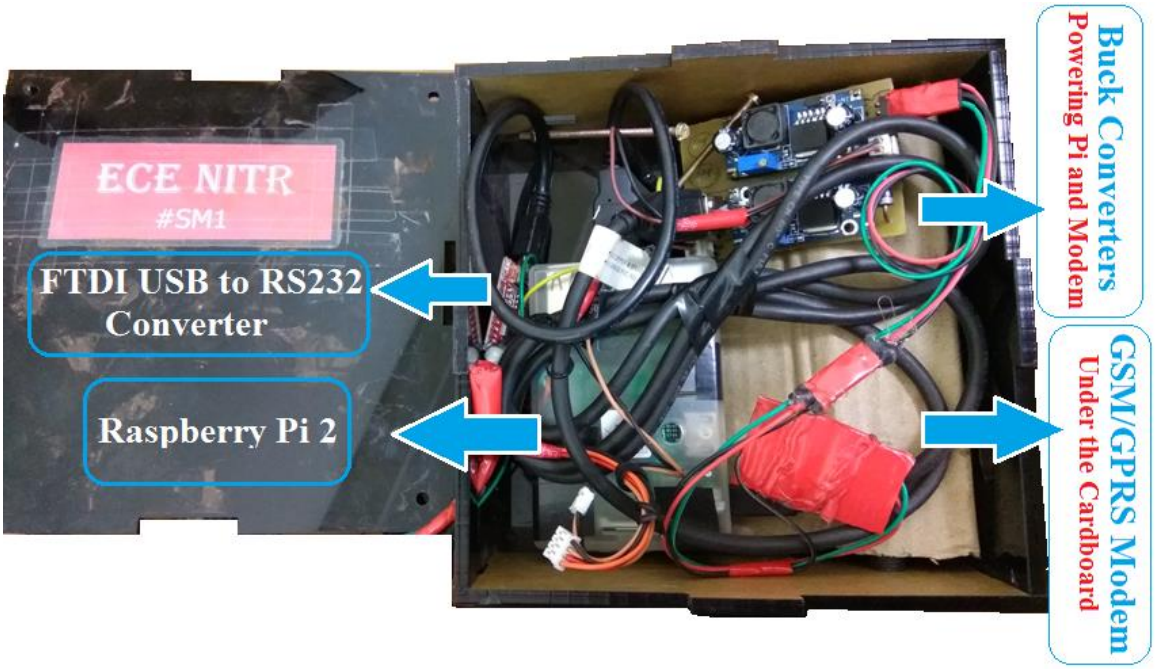

Figure 3.12: CGE Assembly

3.5 Summary

The hardware consists of Vehicle Tracking Equipment and the Coordinating Gateway Equipment. Many of the hardware components and modules are common to VTE and CGE. Arduino is used as the computational unit for the VTE whereas the CGE uses a more powerful unit – the Raspberry Pi 2 Single Board Computer. The Arduino is interfaced to the GSM/GPRS modem, GPS receiver and the XBee module using three serial UART ports. The Raspberry Pi connects to the GSM/GPRS modem and XBee module using two of its USB ports over a USB to UART converters. The CGE is attached to a UPS which provides backup for CGE during power failures as CGE must operate without fail to provide near real-time tracking of the vehicles. The casing of the VTE and CGE are designed to prevent dust and water from entering. The VTE casing is specially designed to sustain heavy vibrations produced from the heavy vehicles operating in the open cast mine.

Chapter 4 - Software

4.1 Overview

In this section, we discuss the software architecture of the system. Different programming languages have been used to code the various components of the system –

- VTE software running on Arduino Mega is developed entirely in C++.
- CGE software running on Raspberry Pi is developed using Python and Shell Script
- VMS software is developed using Python, HTML, and JavaScript

Python was preferred over PHP as it is more abstract and compact than the latter. It was also favoured to make maintenance of the software easier as it was also used for developing CGE software, thus, reducing the number of languages a developer has to be fluent with to maintain the system.

Essential elements of the Vehicle Tracking Equipment software are –

- GSM/GPRS modem interface
- GPS receiver interface
- XBee module interface
- LCD interface

Moreover, that of the Coordinating Gateway Equipment are –

- GSM/GPRS modem interface
- XBee modem interface
- Database structure and access interface
- Indicator LEDs and input switches
- Automatic maintenance and configuration scripts

Key elements of the Vehicle Monitoring Server are –

- The Web Application
- Database structure and interface
- Frontend browser-based user interface

All of the software was developed taking care of the software design principles. Modularisation, decomposition, and systematic reuse have been implemented wherever possible. Our primary concern was to ensure that the software developed for the system to stable enough for long periods of time. The VTE and CGE, once installed are supposed to run without human intervention almost indefinitely. Thus, the VTE and the CGE codes have been developed keeping stability before everything else. The VTE using an Arduino is not a big concern as every line of code running on it is known. GCE, on the other hand, uses Raspberry Pi 2 and runs our application on top of a Debian operating system [4]. Hence, most of the processes running on the operating system are developed by other developers. Thus, it is not possible to guarantee memory leaks over extended periods of time in the order of days and months. CRON jobs are scheduled to reboot the CGE at certain intervals and/or timing of the day. Shell scripts are written to run during boot to reconfigure things and handle some basic clean up tasks like emptying the database logs. At the time of writing, the CGE is configured to reboot every 4 hours for automatic maintenance. The CGE is the only gateway from the ZigBee PAN to the internet and hence is the backbone of the whole system. Hence, it must operate without interruption.

4.2 Component Details

4.2.1 GSM/GPRS Modem

The GSM/GPRS modem used in VTE and CGE are SIM900A wireless modules from SIMCom Wireless Solutions Ltd. The details and features of SIM900A are already mentioned in the third chapter - System Hardware. The SIM900A supports UART interface to communicate with a host device like Arduino or Raspberry Pi. SIM900A supports AT command set for GSM Mobile Equipment, version ‘GSM 07.07’ standardised by The European Telecommunications Standards Institute (ETSI). Figure 4.1 shows the structure and format of different AT commands.

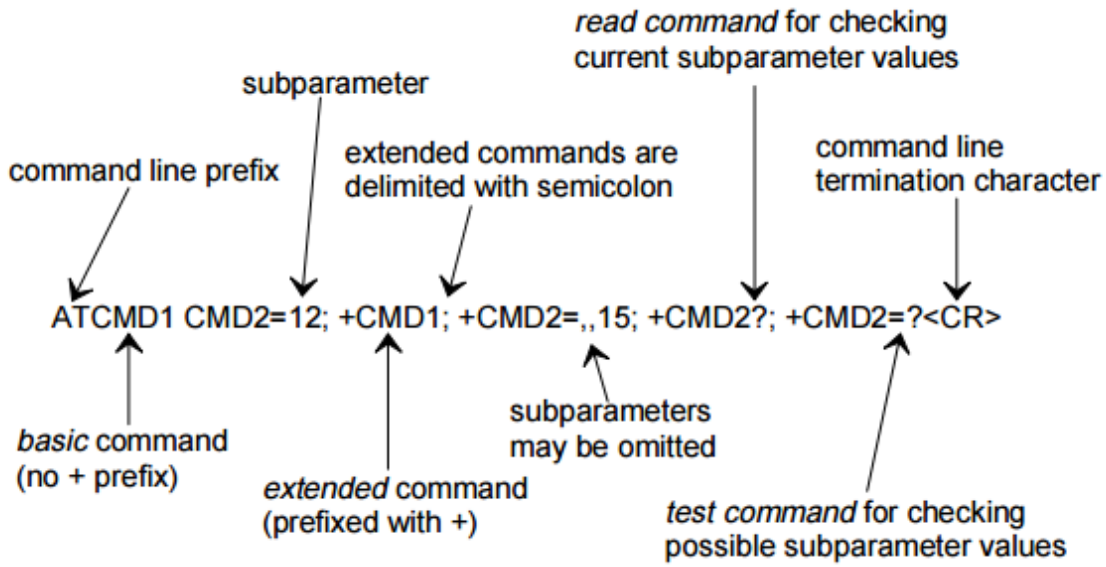

Figure 4.1: AT command structure from ETSI - GSM 07.07 Specification

The ETSI specification refers the host device as a Terminal Equipment (TE) and the modem as a Mobile Equipment (ME). According to the standard, each AT command addressed to the ME must be followed by a Carriage Return (ASCII 13) and a Line Feed (ASCII 10) i.e. <CR><LF>. Similarly, each response from the ME must follow the format <CR><LF>…response…<CR><LF>.

Two libraries were developed to interface the GPRS modem to the Arduino and the Raspberry Pi – one in C++ (named ISA_GSM.h) and the other in Python (named ISAsim900.py). Both of the libraries are abstractions built over the AT commands and their responses. Moreover, they also contain methods to configure and control the interface of the modem like opening the UART serial port, closing the serial port, setting/changing the baud rate and resetting the modem. The functions/methods present in both of the libraries perform similar operations. The libraries expose higher level abstract methods to perform various operations and return the parsed results by issuing command or query to the SIM900A GSM/GPRS modem. Critical operations handled by the functions provided in the library are –

- **Low-Level Methods**
    - Open serial port
    - Close serial port

- Send AT command over serial port
- Read AT Response from serial port
- Parse AT Response
- Reset Modem (Hardware reset and software reset)
- Bypassing the Host (Arduino or Raspberry Pi) to issue commands manually by a user for debugging purposes.

- **Abstract Methods**
  - Check if modem registered on cellular network
  - Open a GPRS connection
  - Close a GPRS connection
  - Make HTTP GET requests to the server (GET and POST)
  - Terminate HTTP Request/Response
  - Read Parse HTTP responses from the server

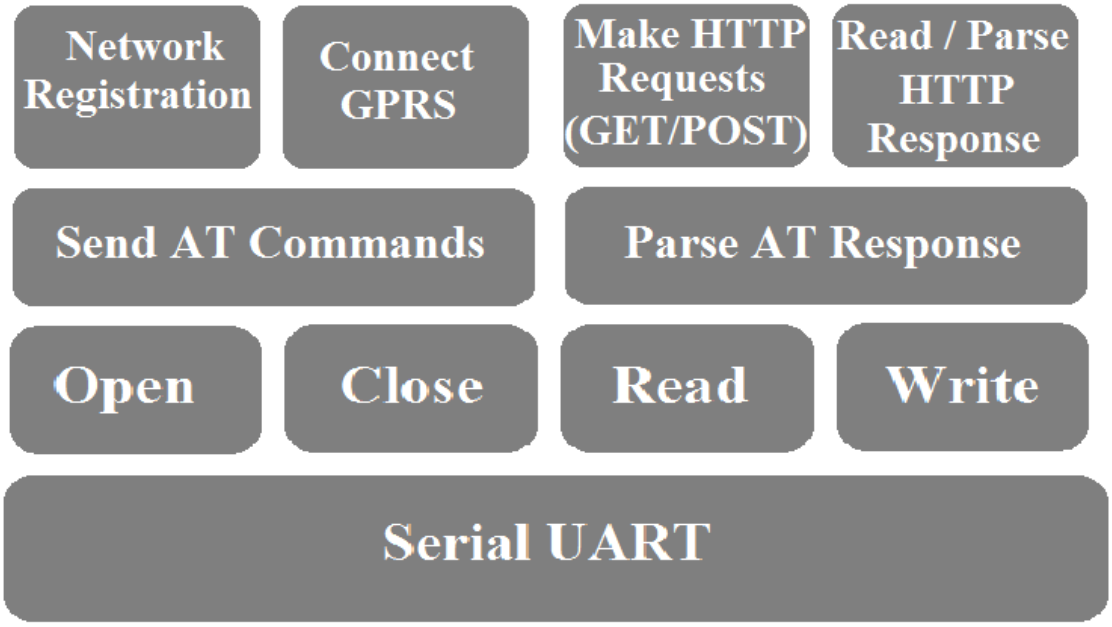

Figure 4.1: Modem Interface Abstraction Model

Apart from the above functions, some very low-level methods/features have been implemented in the C++ library for the Arduino. These low-level methods are not written for Raspberry Pi as Python, being a high-level language, comes with similar methods

already built in with its standard library. The following low-level functions were implemented for the Arduino –

- Read N characters from the serial port within timeout T.
- Read N lines from the serial port within timeout T.

N and T are provided as parameters in no. of characters/lines to read and time in seconds.

The Python library uses a few hardware GPIO pins of the Raspberry Pi to reset the SIM900A via its reset pin. The "platform" module shipped within the standard Python library was used to check if the underlying hardware is a Raspberry Pi board. If the hardware is not a Raspberry Pi board, then the library uses a software reset on the SIM900A modem using the AT commands instead of hardware reset using the GPIO pins of Raspberry Pi. This makes the library hardware independent.

4.2.2 GPS Receiver

The GPS receiver outputs NMEA-0183 sentences which contain position and speed data. An open source library named "TinyGPS" was used to parse the NMEA-0183 sentences and extract the latitude, longitude and speed data. The library is available at http://arduiniana.org/libraries/tinygps/ and https://github.com/mikalhart/TinyGPS.

4.2.3 XBee Module

The modules used to implement the ZigBee wireless PAN are from Digi International named as XBee Pro Series 2B 63mW RPSMA. The XBee modules are interfaced to Arduino and Raspberry Pi over serial UART interface running in the 9600/8-N-1 configuration. A host device (Arduino or Raspberry Pi) can communicate with an XBee module in two modes –

- Transparent Mode
- Application Programming Interface (API) Mode

In the transparent mode, XBee modules dump the payload received by them over the ZigBee network to the UART. Also, anything sent to an XBee module over the UART is made the payload and sent to another XBee module present in the network. In transparent mode, we can enter the command session to change the parameters and configurations of the XBee module by sending the special set of characters "+++**".** The destination address, ZigBee PAN ID, node's MAC address or network address can all be accessed or set using AT commands. Almost all of the ZigBee configurations can be done in the transparent mode as well as the API mode. However, the transparent mode is meant to be human readable and text oriented. It is meant for use with simple ZigBee networks not containing more than two XBee nodes. If more than two XBee nodes, say A, B and C are present, and A receives data from two both B and C, A will output the payloads from both B and C over UART, and it might not be possible to differentiate the payloads from the two. If more than two XBee nodes have to operate in this mode, we have to implement our protocol on top of transparent mode to take care of all the overlapping of payloads ourselves. The API mode implements the protocols needed to solve the problem of overlapping data received from more than two XBee nodes. The API mode outputs frames containing all the details about the source of the received payload, received signal strength of the packet and many others including a checksum field to detect any frame error occurred during transfer over the UART. Also, API mode permits for on the fly and per frame packet configurations. Thus the communication between the host device and the XBee module becomes robust, error free and fast. So even though API mode is more complex than the transparent mode, the API mode was used for communicating with the XBee from the host devices Arduino and Raspberry Pi.

XBee modules need to be programmed with the API version of the firmware to be able to use the API mode. XCTU, a software tool [5] provided by Digi International, was used to flash the appropriate firmware into the XBee modules. Only one XBee was flashed with the ZigBee Coordinator API version of the firmware to be employed in the CGE. The rest were flashed with ZigBee Router API version of the firmware. An XBee in API mode can communicate with another XBee in transparent mode. This is possible because the API mode is a protocol specification used for the XBee to host (Arduino or

Raspberry Pi in our case) interface. Wireless communication between all XBee modules, independent of the version of firmware, happens using the ZigBee wireless protocol.

Start Delimiter	Length		Frame Data							Checksum
			Frame type	Data						
1	2	3	4	5	6	7	8	9	...	n	n + 1
0x7E	MSB	LSB	API frame type	Frame-type-specific data						Single byte

Figure 4.2: XBee API Generic Frame

In API mode, interaction with XBee modules is done using API frames [6]. An API frame is a structured format for data that contains the command/payload along with helpful information about the command/payload called metadata in the headers and footers of the frame. The first byte of an API frame contains a special character called the start byte or the start delimiter. The next two bytes form a 16-bit word that indicates the number of bytes present in what is known as the frame data which starts from the 4th byte to the last byte called checksum but not including it. The checksum is used to check the integrity of the received frame.

API ID	Frame name	Description
0x88	AT Command Response	Displays the response to previous AT command frame
0x8A	Modem Status	Displays event notifications such as reset, association, disassociation, etc.
0x8B	Transmit Status	Indicates data transmission success or failure
0x90	Receive Packet	Sends wirelessly received data out the serial interface (AO = 0)
0x91	Explicit Rx Indicator	Sends wirelessly received data out the serial interface when explicit mode is enabled (AO ≠ 0)
0x92	IO Data Sample Rx Indicator	Sends wirelessly received IO data out the serial interface
0x94	XBee Sensor Read Indicator	Sends wirelessly received sensor sample (from a Digi 1-wire sensor adapter) out the serial interface
0x95	Node Identification Indicator	Displays received node identification message when explicit mode is disabled (AO = 0)
0x97	Remote AT Command Response	Displays the response to previous remote AT command requests

Figure 4.3: XBee Receive Data Frames

The structure of a frame data varies with the type of the frame. The kind of the frame is determined by the frame type field in the packet called the **Frame Type** or **API ID** [6] [7] [8]. The API frames can be classified into two categories based on the direction

of the frame transfer. Frames that are sent to an XBee module from a host device are called Transmit Data Frames, and frames that are received by a host device from an XBee module are referred to as Receive Data Frames.

API ID	Frame name	Description
0x08	AT Command	Queries or sets parameters on the local XBee
0x09	AT Command Queue Parameter Value	Queries or sets parameters on the local XBee without applying changes
0x10	Transmit Request	Transmits wireless data to the specified destination
0x11	Explicit Addressing Command Frame	Allows ZigBee application layer fields (endpoint and cluster ID) to be specified for a data transmission
0x17	Remote AT Command Request	Queries or sets parameters on the specified remote XBee
0x21	Create Source Route	Creates a source route in the module
0x24	Register Joining Device	Registers a module with the Trust Center

Figure 4.4: XBee Transmit Data Frames

The XBee API mode was used to interface the XBee nodes with the Arduino as well as the Raspberry Pi. For Arduino, a C++ library was written to interface the XBee in API mode. For the Raspberry Pi, an open source Python library, named XBee [9] (version 2.3.3), available at the Python Package Index was used. It can be downloaded from https://pypi.python.org/pypi/XBee, and its GitHub page is hosted at https://github.com/nioinnovation/python-xbee.The library did not handle all the API frame types like Route Record Indicator(frame type/API ID – 0xA1) and had to be modified slightly to behave as required by our system. The library is covered under The MIT License, which permits us to use and change the library as per our requirements. The primary tasks accomplished by the XBee python library are –

- Receive API frames from the local XBee module
- Send API frames to the local XBee module

By saying local XBee, we mean the XBee module which is connected to the Raspberry Pi over the UART interface.

The C++ library for the Arduino was, however, developed entirely as no open source Arduino library was found that was suitable for our implementation. The library is named ‘xbeeapi.h’ and it does not support all the API frame types in favour of reducing the size of the Arduino code as well as the RAM required to run the library properly.

The supported frame types are –

Frame Type Supported	Frame Type (API ID)
- AT Command	0x08
- AT Command Response	0x88
- Transmit Request	0x10
- Transmit Status	0x8B
- Receive Packet	0x90
- Modem Status	0x8A

The XBee C++ library for the Arduino accomplishes the same two tasks that are accomplished by the XBee Python library, which are –

- Receive API frames from the local XBee module (using the function – listen)
- Send API frames to the local XBee module (using the function – speak)

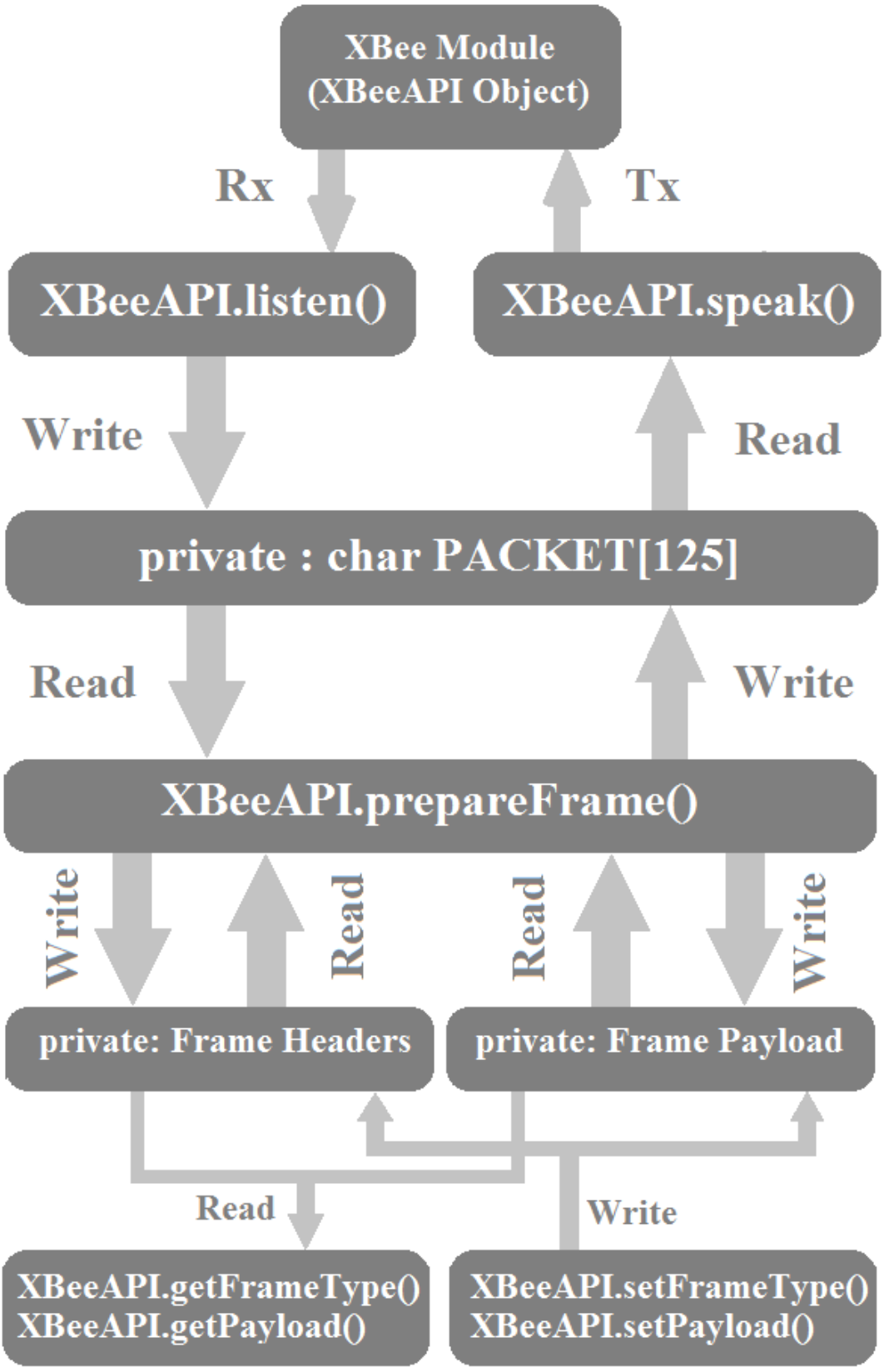

Figure 4.5: XBee API Library Model (xbeeapi.h)

4.2.4 Database for the Raspberry Pi

When the CGE receives a position log over ZigBee WPAN from a VTE, it must be forwarded to the VMS as soon as possible for near real-time monitoring. Multiple logs from different vehicles might be received in fast successions. It is also possible that an active GPRS connection is not available to the modem at the instant when position logs are received from the vehicles. Hence, the received position logs need to be buffered into memory. Although Raspberry Pi has enough RAM, we cannot buffer the logs in RAM as it is volatile and a power failure or restart would cause us to lose the logs. Also, if the CGE is unable to connect to GPRS for a long time, the incoming position logs might consume all of the RAM. So the incoming logs must be buffered in a secondary memory as it is larger in size than RAM and non-volatile. A simple way to implement this is by using a database. Hence, an SQLite database was implemented which is stored as a file on the SD card of the Raspberry Pi.

It is intuitive to upload the latest position log received to the VMS and then the stale ones so as to achieve a near real-time monitoring. The stale logs are already late and hence are of less priority than the newer position logs. The process of logging the incoming position logs over ZigBee network and the process of uploading the logs to VMS over GPRS must not block each other. So they need to run in parallel. Thus the database needs to be shared among two processes. To implement this, Last In First Out (LIFO) or stack abstract data type is the most suitable candidate. So the database is used as a LIFO, one process adds new logs to the database table, and the other reads the logs from the table.

A Python library named ‘peewee’ was used to interact with the database. The database contains two tables, one to contain the list of vehicles named as ‘Vehicles’, and the other to contain the positional data logs received over the ZigBee network by the CGE, named as ‘VLog’ as an abbreviation for Vehicle Logs.

For the table ‘Vehicles’, the ‘id’ is the primary key of the table. ‘MAC64’ is the 64-bit MAC address of the XBee module attached to the VTE of a vehicle. ‘NIname’ is the Node Identifier field of the XBee node. ‘MY16’ is the 16-bit network address of the XBee

node. Hereafter, by referring to the MAC address of a vehicle, we mean the MAC address of the XBee module contained in the VTE attached to that vehicle.

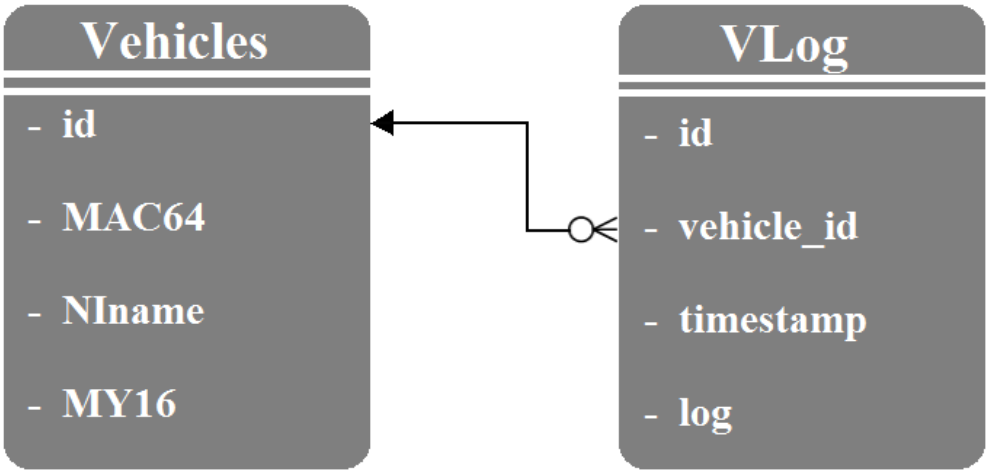

Figure 4.6: Database Model of CGE

For the 'Vlogs' table, the 'id' is the primary key of the table. 'vehicle_id' is a foreign key to the Vehicles table. 'timestamp' is a field for the storing the time at which the positional data was received by the CGE and 'log' contains the actual payload containing the position sent by a VTE over the ZigBee network. The payload sent by VTE are JSON (JavaScript Object Notation) string containing a Vehicle ID (different from vehicle_id), Latitude and Longitude. The Vehicle ID of a VTE is same as the Node Identifier of the XBee module attached to it. This is done so that, if need be, the Node Identifier of an XBee module, hence the Vehicle ID of the vehicle, can be updated via a remote XBee using remote AT commands.

A library was written in Python to abstract the interaction with the database. This library builds on top of the 'Peewee' [10] library mentioned earlier. The tasks accomplished by the functions provided by this library, named VdbModel.py, are –

- Create the tables and the database file.
- Inserts new vehicles or update existing vehicles in the Vehicles table.
- Insert position logs received from vehicles over ZigBee WPAN into the VLog table.
- Fetch position logs from the VLog table (to be sent to VMS).

- Delete logs from VLog table (after the logs are uploaded to the VMS).
- Delete the VLog table entirely (during a scheduled maintenance reboot)

4.2.5 Helper Module for CGE

When a VTE sends a location data over the ZigBee WPAN to the CGE, it is immediately logged into the database. Meanwhile, an active GPRS connection is maintained, and the latest position log from the database is uploaded to the VMS as fast as possible for near real-time monitoring. The above two processes must not block each other. The main thread and a daemon thread running in parallel handle the above two operations.

The open source Python XBee library supports an asynchronous mode in which a daemon thread is created, and whenever a packet arrives at the Raspberry Pi from the XBee module, a call-back function is executed. This frees the main thread of the Python script from repeatedly polling the XBee serial port to check if a packet has arrived. The main thread starts the XBee daemon thread once at the beginning of the program and proceeds to maintain an active GPRS connection and upload latest logs from the database to VMS.

- Logging data received from ZigBee WPAN to database require the libraries XBee and VdbModel.py
- Maintaining an active GPRS and uploading data to VMS requires the library ISAsim900.py

A helper module/library, named Helper.py, was developed on top of the three libraries XBee, VdbModel.py and ISAsim900.py to provide a level of abstraction to the above two processes.

The functions provided in Helper module accomplish the following abstract tasks –

- Receive location data from vehicles asynchronously over ZigBee WPAN and log them into the database.
- Fetch the latest logs from the database and convert them to a JSON string.
- Maintain an active GPRS connection.

- Upload a JSON string to the VMS.
- Remove logs from the database when successfully uploaded to VMS.

It also implements functions to search vehicles present in the ZigBee network and fill the Vehicles table of the database with the discovered vehicles.

4.2.6 Web Service for VMS

A web service is a service that allows machine to machine (VTE/CGE to VMS) interaction over a network. A web application is a web service that also includes a web browser based user interface. Most of the web applications use HTTP protocol while web services, which are meant for communication between machines, use a vast number of protocols depending on the type of machines, type of communication and the content/payload communicated.

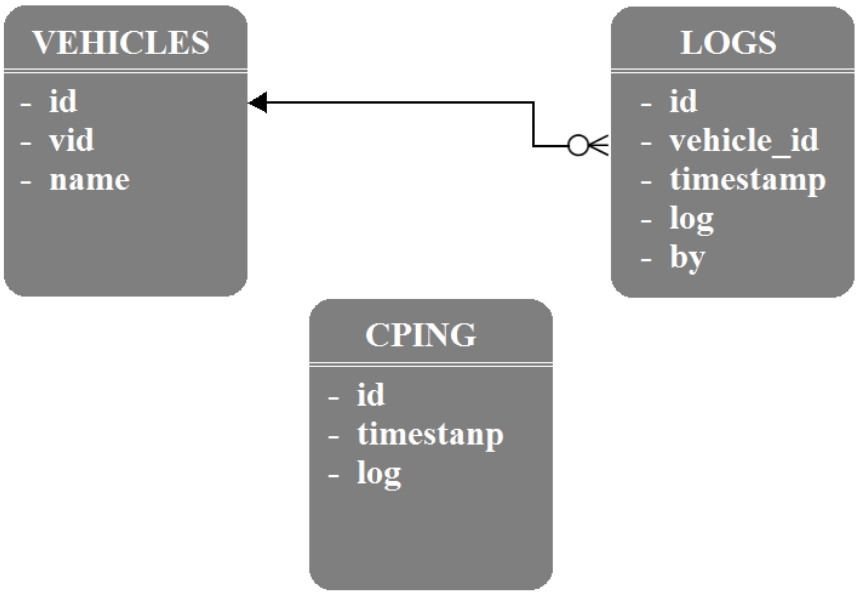

Figure 4.7: VMS Database Model

The VMS runs a web service cum application as it communicates with VTE and CGE which are machines and also displays position of vehicles on a map in a browser. However, in our case, the web application and the web service, both use HTTP protocol.

The Web service cum application running on the VMS was developed using an open source Python-based web framework called Web.py.

Web.py follows the model-view-controller architecture. The model is concerned with the database transactions. The view is the browser-based interface. The controller is the middleman who links the model and view with some logic. This web service provides a web interface for the VTE and CGE to log the position data to the server's database. The web application provides a browser-based interface for viewing the positions of vehicles on a map.

There are three components of our web service application –

- A database to store position data of vehicles and user information (MODEL)
- HTML templates to provide a browser-based web interface (VIEW)
- A program to control the logic of mapping different URLs to invoke specific actions (CONTROL). These specific actions might trigger a user interface on a browser (web application) or log position data sent by VTE/CGE into the database (web service). So every URL is related to either a web service or a web application.

  The services/application exposed by the VMS are –

- Log position data sent by VTE/CGE into database (service)
- Log ping (status) messages sent by CGE into database (service)
- Display details of vehicles existing in database (application)
- Display position of vehicle(s) on map (application)
- Display ping (status) messages from database (application)

The browser base user interface was developed using HTML and JavaScript. Google Maps API [11] and Highcharts [12] JavaScript plotting API have been used to display vehicles on map and plot their speed. The database is implemented using SQLite. The open source library 'peewee' is used to interact with it. The database consists of three tables. "VEHICLES" table is used to store the details of the registered vehicles, and "LOG" table for storing the position logs received from VTE (by = 1) or the CGE (by=0). The "CPING" table stores the status messages sent by the CGE.

4.3 Integration

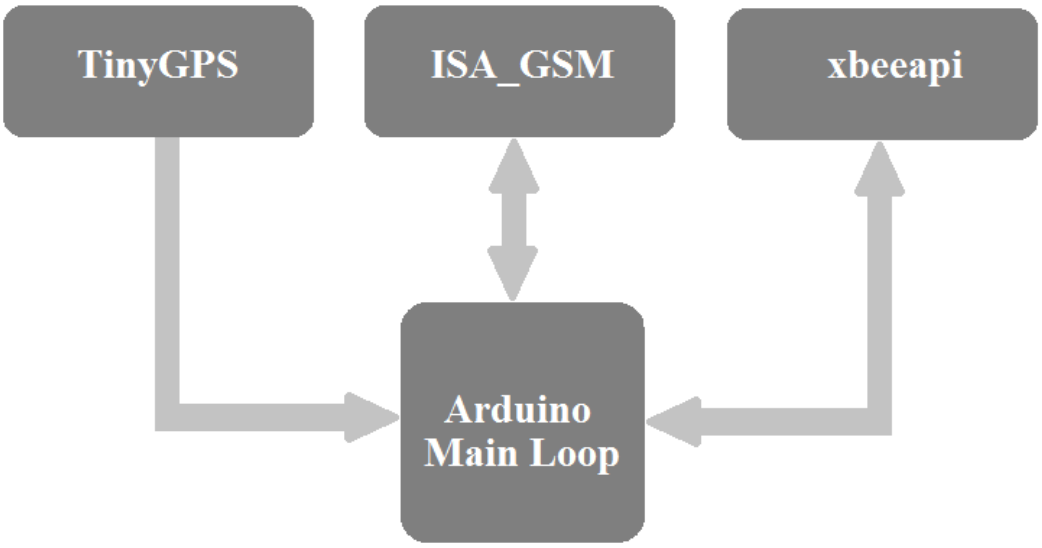

Figure 4.8: VTE software

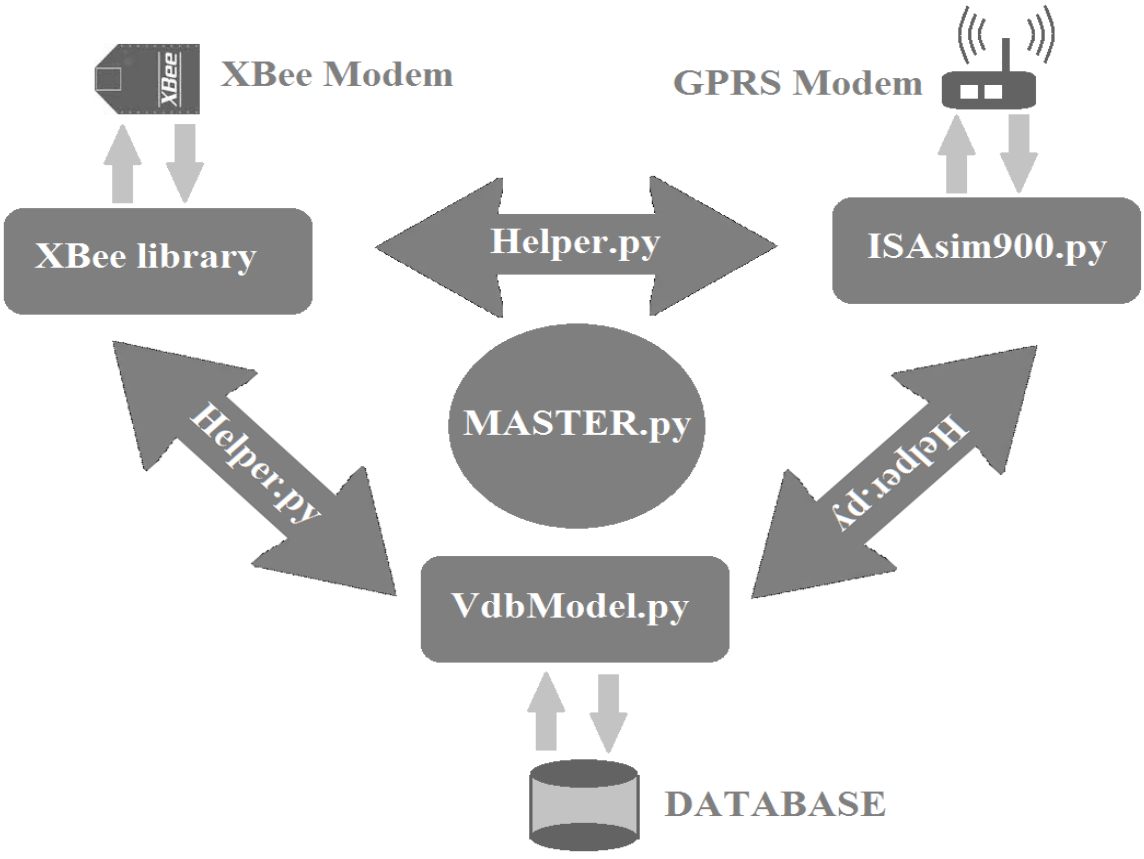

Figure 4.9: CGE Software

4.4 Summary

The software for VTE was developed using C++, and that for CGE and VMS were developed in Python. The libraries to interface XBee and GPRS modem with the Arduino were developed in C++. An open source library named “TinyGPS” is used to interface the GPS. An open source library named ‘XBee’ was used for interfacing the XBee modem with Raspberry Pi. The library for interfacing GPRS modem with Raspberry Pi was developed in Python. The web service for the VMS was developed using the Web.py web framework. The database for the VMS and CGE were implemented using SQLite. The open source Python library named as peewee is used to interact with the SQLite databases.

Chapter 5 - Results and Conclusion

5.1 ZigBee Range Test

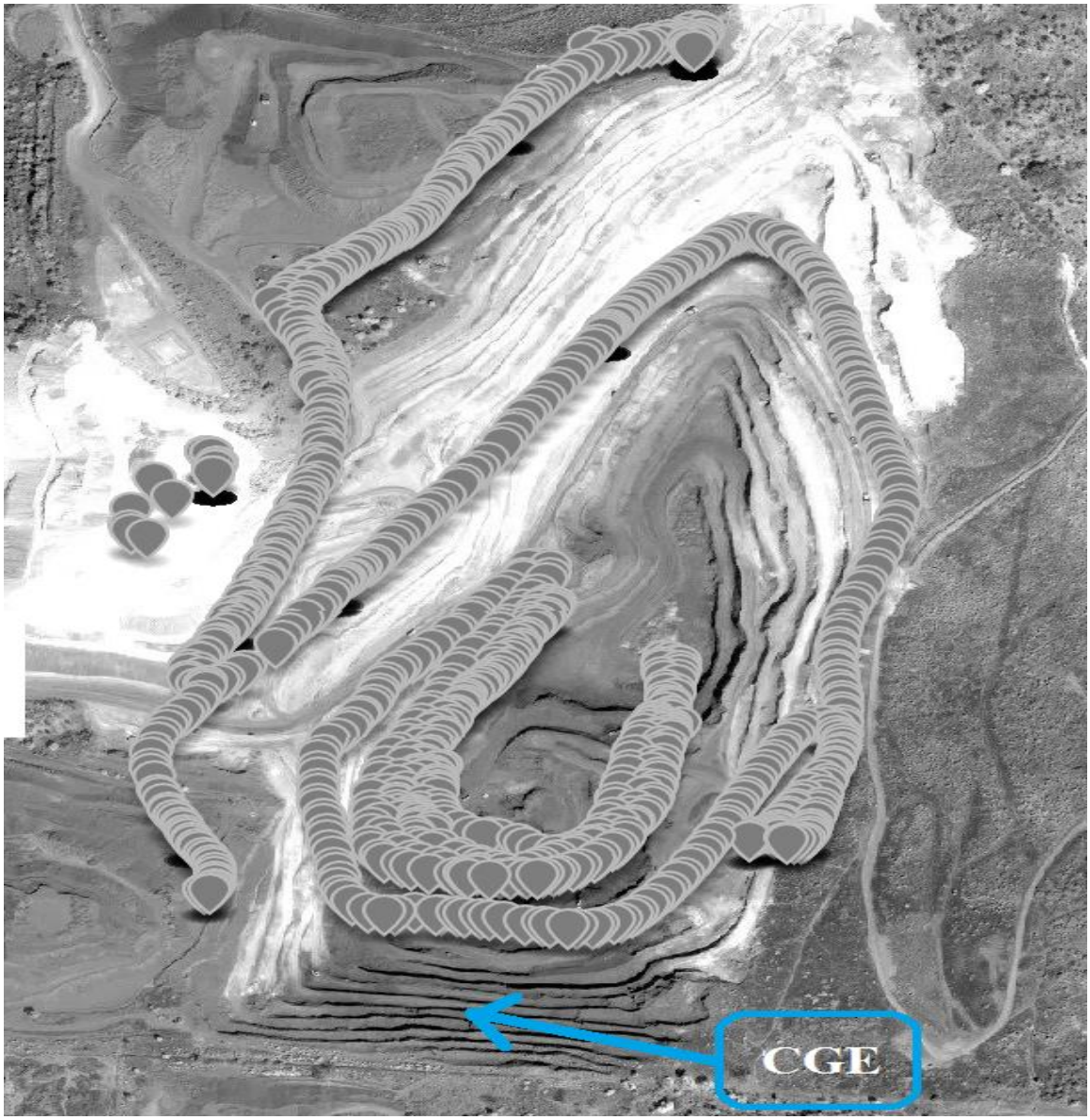

Figure 5.1: XBee Range Test

For successful working of the system, the ZigBee Pan implemented using XBee Pro Series 2B modules must provide sufficient range to cover most of the benches in single radio hop. The single-hop line of sight range as mentioned in the XBee Pro Series 2B data sheet is 1

mile or about 1.6 Km. However, the practical range may vary widely due to the non-line of sight, terrain topology and other reflecting and obstructing materials present.

On a test inside our institute campus, two XBee module communicated perfectly between Student Activity Centre and the Basket Ball court which are more than 500 metres apart. During the test, the XBee modules were at a human height above the ground. As the terrain is different in an open cast mine and is more open, we expected a slightly higher range than our test inside institute premises and was confirmed during our field testing in the mines.

To check the XBee module's range in the mines, we sent a vehicle to travel throughout the pit benches with an XBee transmitting its GPS coordinates every second. We observed that XBee could cover almost all the line of sight benches. The only places we did not get the GPS coordinates were the non-line of sight benches. The map in figure 5.1 shows all the GPS coordinates that we received over the ZigBee PAN network using the XBee Pro modules. Hence, the one-hop line of sight range we found for the XBee modules was as expected at around 800 metres.

5.2 Installation

The CGE was installed near the location from where we had originally conducted our XBee Range Test. It was installed at a height of 15 feet above the mine bench at a pre-existing light post which has electricity supply almost all of the time. A box made up of iron sheets and transparent fibreglass supports as well as protects the CGE from sun and rain. The XBee module is extended outside using a USB cable of about a metre so as to provide a better line of sight from the various mine benches.

VTE were installed inside the cockpit of the dumpers, and the XBee module was extended beyond the dumper's height to have a line of sight to the CGE. Category 6 (Cat 6) cable was used to carry the RS232 signal from the VTE in the cockpit to the XBee modules extended above the height of the dumpers. The supply to VTE were given from 12V batteries of the dumpers which are rated at 65 Ampere-Hour. At a peak current draw of one Ampere, the VTE can last for more than a day without requiring a recharge.

However, other instruments present inside the dumper also draw current from the battery and hence with a margin of hundred percent, the VTE can last for about 12 hours without draining the battery completely.

Figure 5.2: CGE installed on a Light Post

5.3 VTE and CGE Performance

As stability was a priority during development, the VTE software performed up to expectations. In a test at our lab where there is cellular network available all the time, the VTE operated as expected for over a week. The frequency of position data logged at the VMS varied from 1/7 Hz to 1/15 Hz with an average of around 1/11 Hz when the VTE is under cellular network. A VTE was installed for initial testing in February 2016 in a 35-

tonne dumper without the ZigBee PAN and CGE. It logged position data to VMS at positions under cellular coverage before going completely dead

The stability of CGE mattered more than VTE as it was running a general purpose Debian operating system meaning that it would be running applications that have been written by others alongside our application. At our lab in the Institute, we made a setup of two VTE modules and the CGE module thus creating a test environment simulating two vehicles, the ZigBee PAN, and the CGE. The system was operated for more than four days without human intervention.

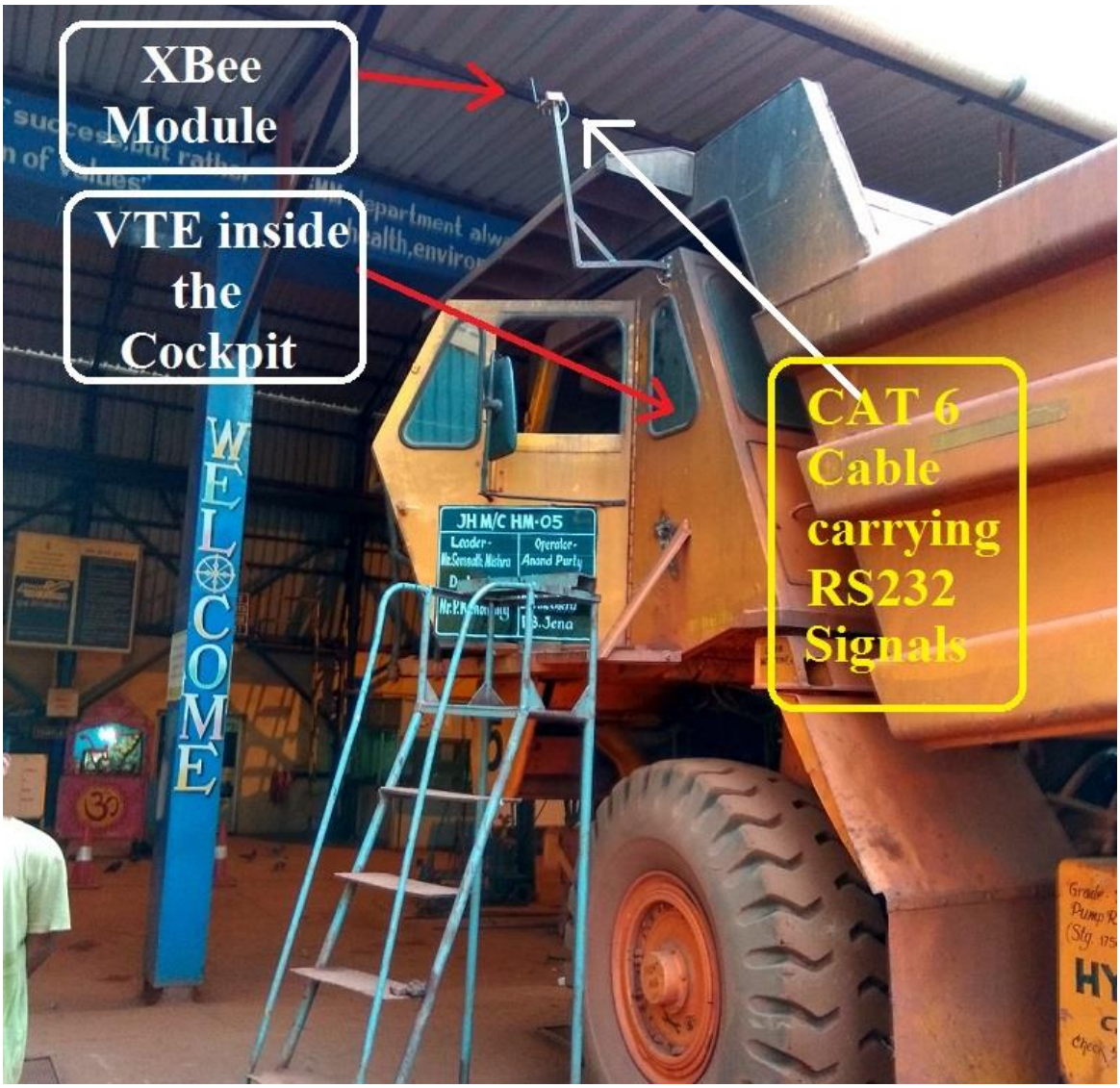

Figure 5.3: VTE installed on a Vehicle

The CGE and two VTE were installed at Jindal Iron Mines, Tensa on 1st of March, 2016 for an initial field test. The system operated without problems for around twenty days before the CGE had stopped logging data to the server. The CGE was restarted manually, and it operated for a week before going down again. On getting the CGE back from the field at our lab, we observed that the CGE repeatedly restarted during the boot process. We detected two problems –

- Raspberry Pi's CPU was overheating to around 80°C even under normal computational load.
- The Linear Voltage Regulator LM7805, which powered the Raspberry Pi, was also overheating and triggering a thermal shutdown of the IC and hence disgraceful shutdown of Raspberry Pi.

The overheating CPU might have developed either because of over voltage (due to AC mains voltage surge) or due to short-circuits on the Raspberry Pi PCB resulting from moisture in the air which is very high at the site of deployment because of its high altitude above sea level. Although the CPU was overheating, it was not the immediate cause of its restarts. The voltage regulator LM7805 is rated for 1A output current. Even though Raspberry Pi is rated from 500mA to 1500mA, the Raspberry Pi only consumed about 700mA in our lab tests. However, the overheating CPU caused the Raspberry Pi to draw more current than LM7805 can supply leading to the thermal shutdown of the IC.

The LM7805 regulator was replaced with LT1083, which is rated at 3A output current. The Raspberry Pi was also replaced with a new one. The ping script was modified to log the Raspberry Pi CPU temperature along with CGE status data every 2 minutes to the Vehicle Monitoring Server. The system was reinstalled on 3rd of April in the open cast mine. The system has been online and working as expected as of this writing (24th May 2016).

5.4 Results

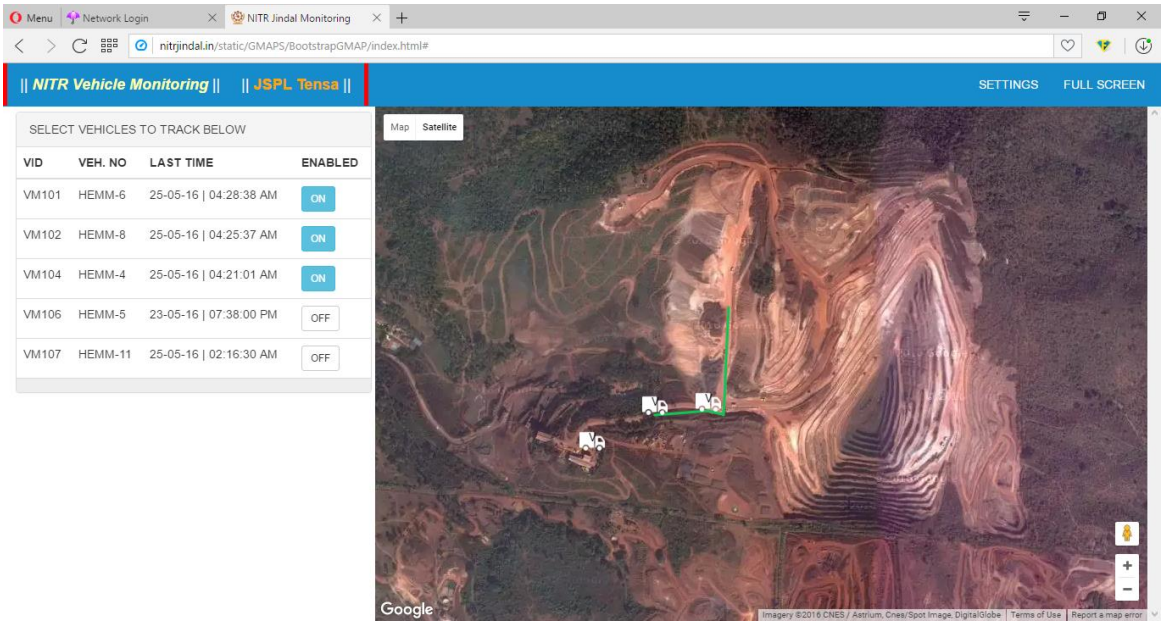

Figure 5.4: Live Monitoring

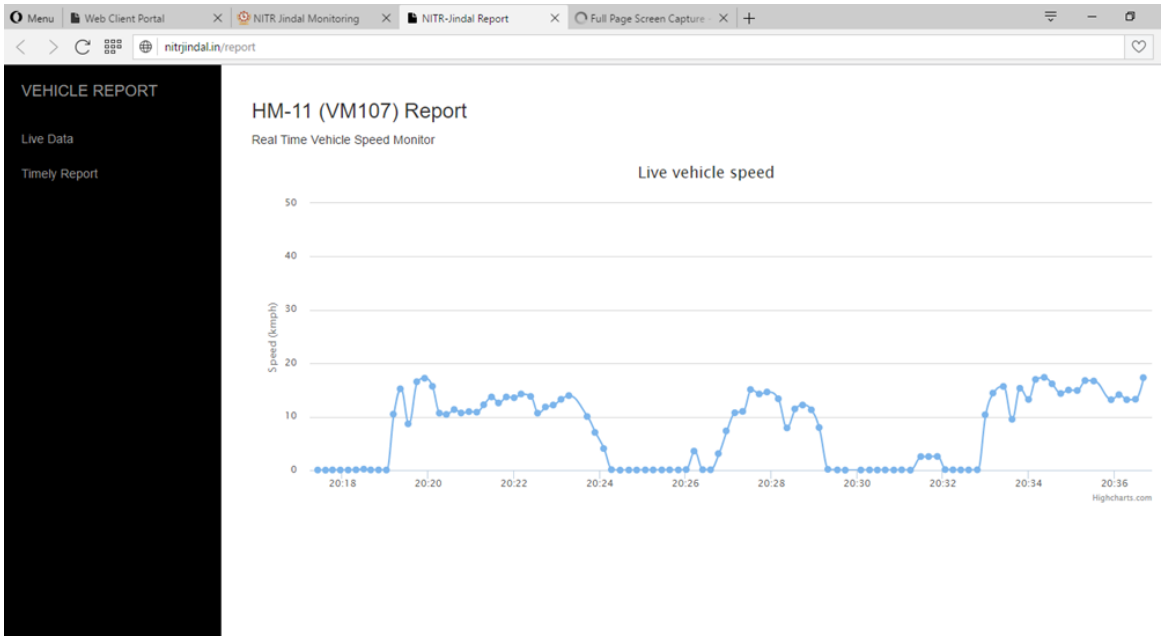

Figure 5.5: Vehicle speed data

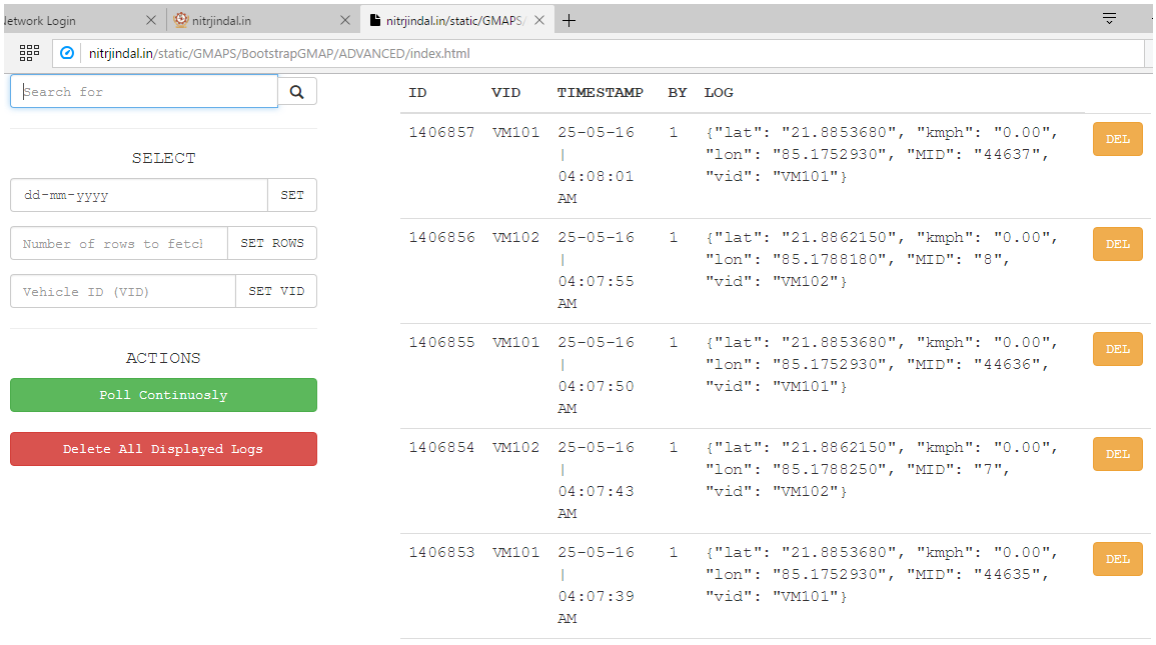

Figure 5.6: Location Data Received at VMS

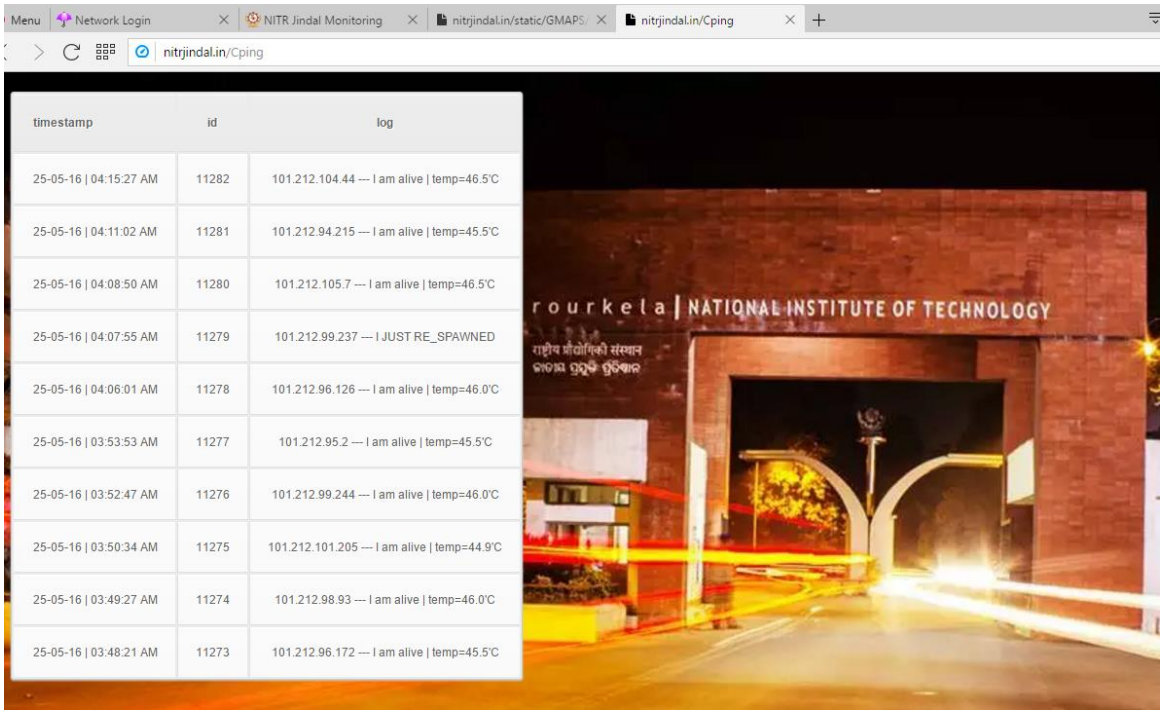

Figure 5.7: Status messages received from CGE at VMS

5.5 Conclusion

The system performs as envisioned and the idea has been materialized. The system captures location and speed data of vehicles at the VMS at an average of once every ten seconds when VTE is under cellular network. In the absence of a cellular network, the average time increases to thirty seconds, which can still be considered near real-time in the context of vehicle monitoring. However, the thirty seconds average is not a limitation of the system. It is limited by the simple timeout mechanism used in the VTE to decide when to give up on GPRS and forward the packet to the CGE over ZigBee WPAN. The speed data collected can be plotted on a graph to find the duration and location of the halt of a vehicle. The software is stable enough, and the system has been running without intervention for two months.

5.6 Scope of Future Work

- The acquired position and speed data of vehicles can be processed to find patterns of vehicle inactivity to optimise the transportation chain.
- Instead of a simple mechanism, an adaptive timeout mechanism which takes the GSM signal strength and ZigBee Network parameters into account can be used in VTE for switching between GPRS and ZigBee more intelligently.
- Static XBee nodes can be installed at suitable locations to act as routers to improve the range of the ZigBee network.
- More CGEs can be installed to act as gateways to share the ZigBee traffic and hence improve the time performance of the system.
- More information regarding the vehicle like the status of its engine, emission, oil level, etc. can also be acquired by integrating On Board Diagnostics (OBD) with the system.
- The system can be integrated with a weighing bridge for automation of the weighing process.

References

[1] “Open Pit Mine,” www.mine-engineer.com, [Online]. Available: http://www.mine-engineer.com/mining/OpnPit_Dw.jpg.

[2] “Arduino,” [Online]. Available: https://www.arduino.cc. [Accessed May 2016].

[3] “RASPBERRY PI 2,” [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/. [Accessed May 2016].

[4] “Raspbian,” [Online]. Available: https://www.raspberrypi.org/downloads/raspbian/. [Accessed May 2016].

[5] “XCTU,” [Online]. Available: http://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu. [Accessed May 2016].

[6] “XBee Frame Structure,” [Online]. Available: https://docs.digi.com/display/XBeeZigBeeMeshKit/Frame+structure. [Accessed May 2016].

[7] R. Faludi, Building Wireless Sensor Networks, O'Reilly Media.

[8] D. Gislason, Zigbee Wireless Networking, Elsevier, Newnes.

[9] S. J. E. Jr., “XBee API Mode Tutorial Using Python and Arduino,” May 2016. [Online]. Available: http://serdmanczyk.github.io/XBeeAPI-PythonArduino-Tutorial/.

[10] “Peewee Quickstart,” [Online]. Available: http://docs.peewee-orm.com/en/latest/peewee/quickstart.html. [Accessed May 2016].

[11] “Google Maps,” Google, [Online]. Available: https://developers.google.com/maps/. [Accessed May 2016].

[12] “Highcharts,” www.highcharts.com, [Online]. Available: http://www.highcharts.com. [Accessed May 2016].

[13] P. P. W. C. Paolo Baronti, “Wireless sensor networks: A survey on the state of the art and the 802.15.4 and ZigBee standards,” *Computer Communications,* 2007.

A. Appendix

Visits to Jindal Iron Mines, Tensa, Sundergarh Odisha India

Figure A.1: Installation of the CGE

Figure A.2: CGE installed on a Light Post

Figure A.3: VTE installed inside a dumper's cockpit

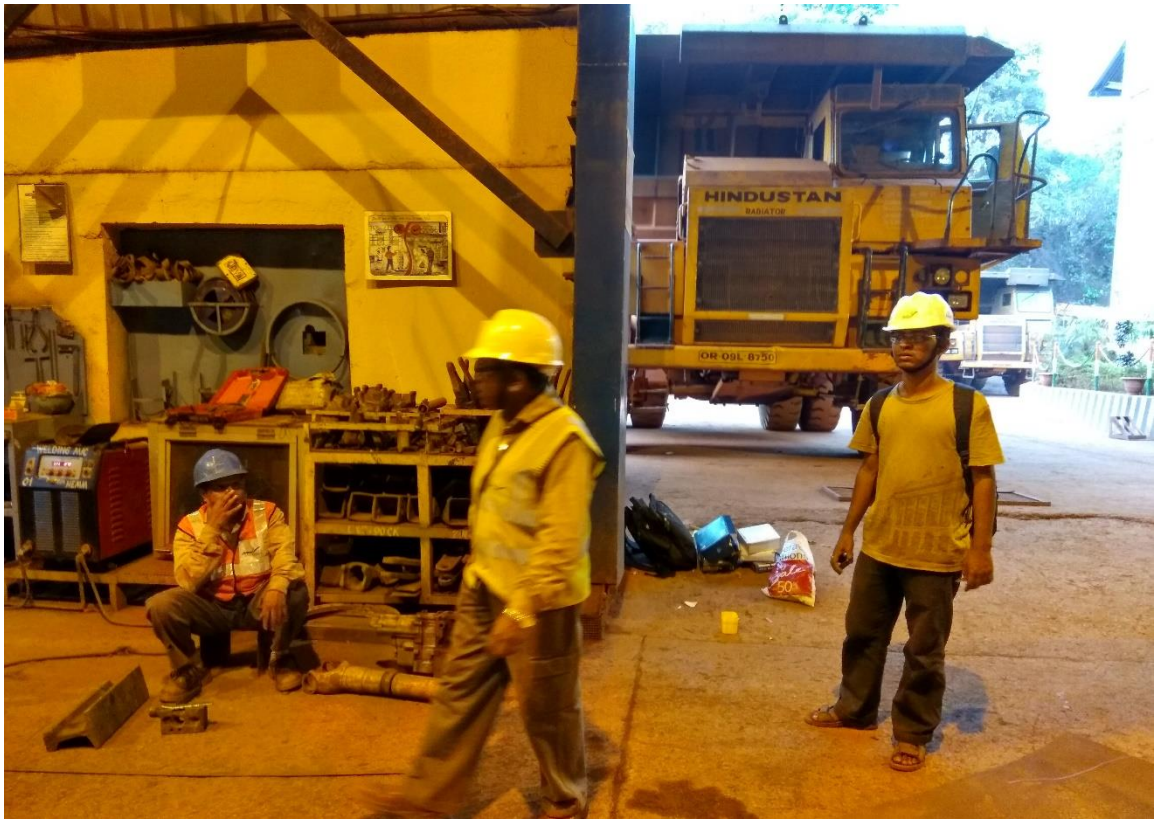

Figure A.4: Vehicle Maintenance Shop

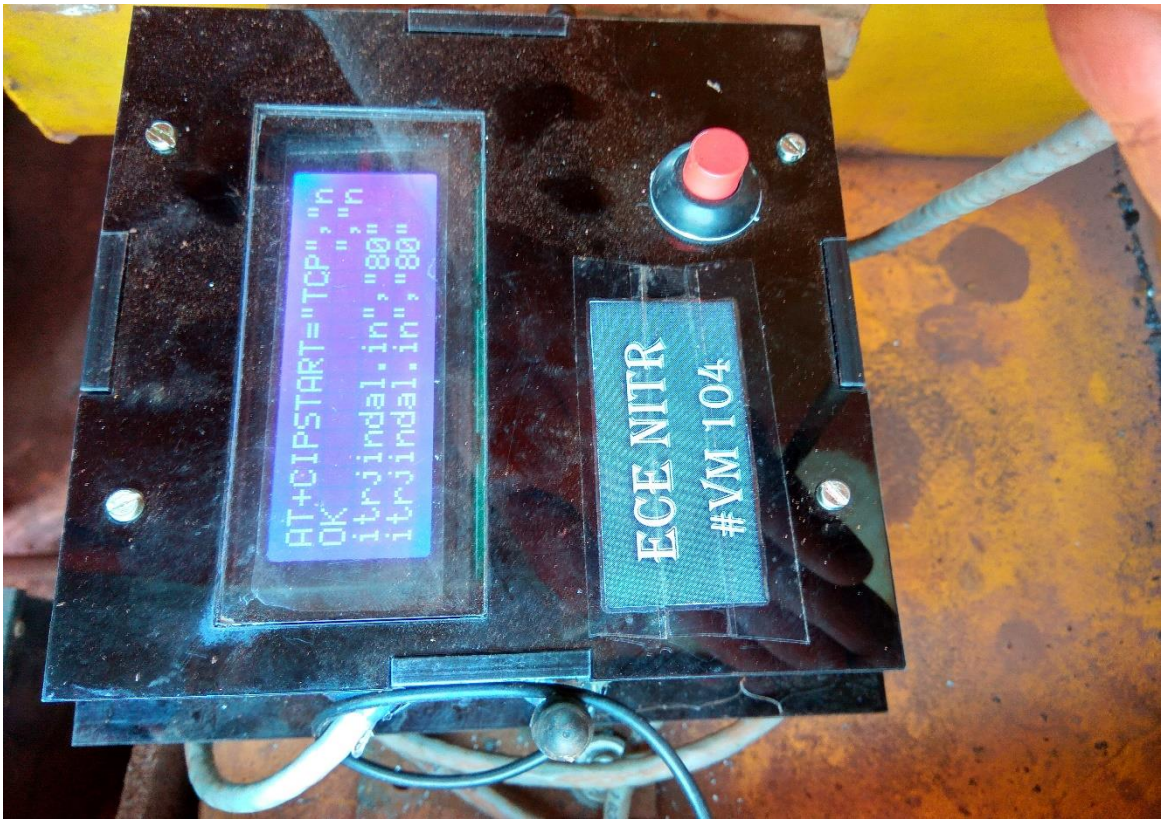

Figure A.5: Debugging a VTE

Figure A.6: Testing the CGE

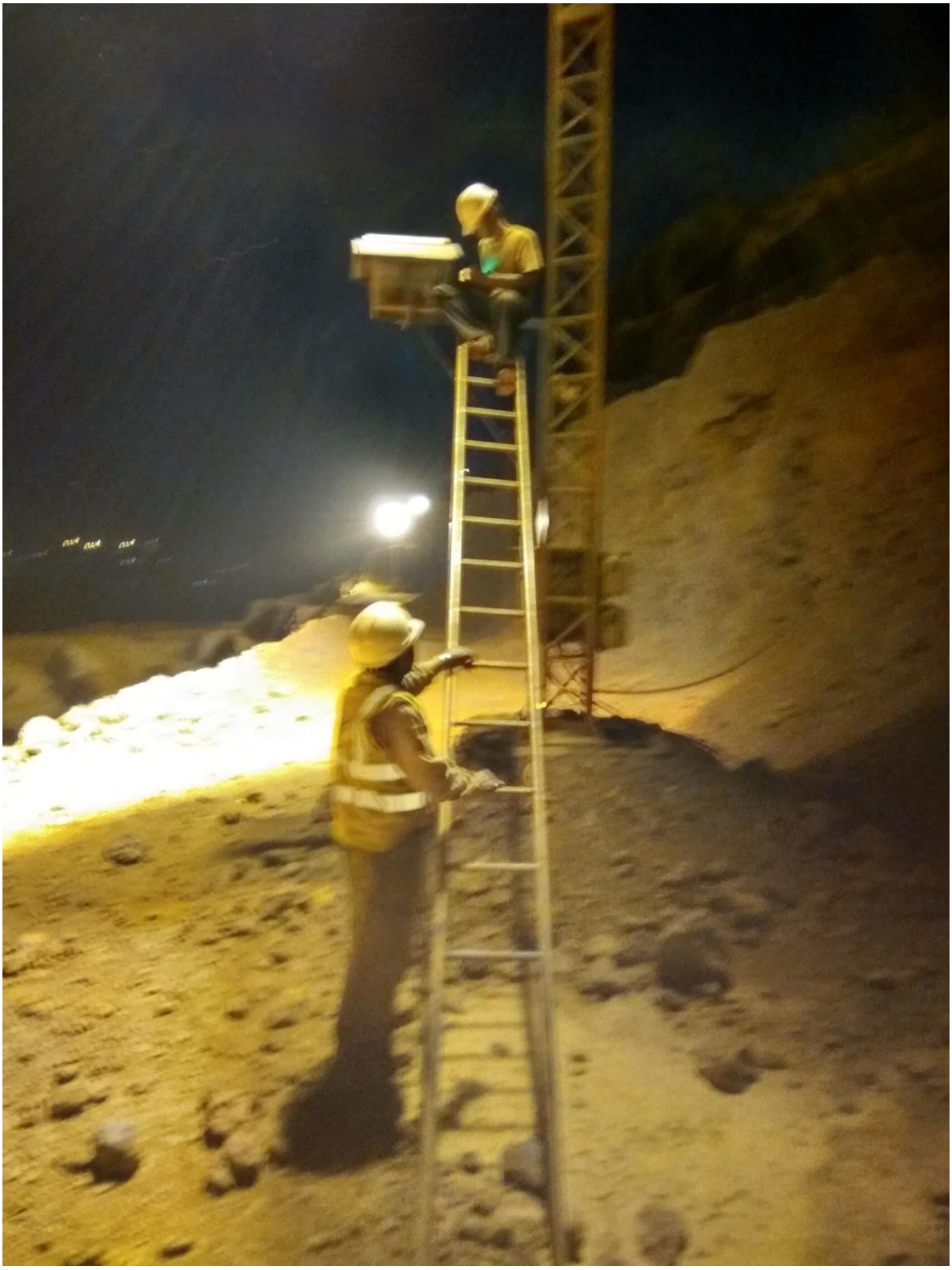

Figure A.7: Debugging the CGE

Figure A.8: A picture of the mine at night

Python SIM900 modem Library (ISAsim900.py)

The specific method/function names, their argument, return values, and operations are listed below –

isRPi

- Type – Function
- Arguments – None
- Returns – True if platform is Raspberry Pi, else False

ISAsim900

- Type –
  - Class
- Constructor Arguments –
  - Serial port name
  - Baud rate (default 9600)
  - Access Point Name
  - GPIO pin to reset modem (default BCM12)
- Returns –
  - ISAsim900 Object

The methods provided by the ISAsim900 class are listed below with their function, arguments and return values –

SoftReset

Function –

Resets the modem using AT the commands AT+CFUN=0 and AT+CFUN=1.

HardReset

Function –

Resets modem by sending a pulse via a GPIO pin connected to modem's reset pin.

openPort

Function –

Checks if the serial port is open, if not, opens the serial port.

closePort

Function –

Checks if the serial port is closed, if not, closes the serial port.

Pipe

Function –

It takes no arguments and starts the modem debug terminal. It connects the standard input and output to the modem bypassing the Raspberry Pi. This allows the user to type AT commands at the terminal and get responses from modem directly to terminal for debugging purposes.

sendAT

Function –

Writes AT command to the modem and parses the AT response received from it.

Arguments –

ATcmd	type = string	default = “AT”
success_phrase	type = string	default = “”
error_phrase	type = string	default = “”
timeout	type = float	default = 5 seconds
success_safeexit	type = Boolean	default = False
error_safeexit	type = Boolean	default = False

Returns –

If success_safeexit = False, returns True immediately after receiving a line (of AT Response) matching the success_phrase without processing the subsequent lines of the AT response.

If success_safeexit = True, after receiving a line (of AT Response) matching the success_phrase, waits for one more second to receive subsequent lines of the AT response before returning True.

If error_safeexit = False, returns False immediately after finding a line (of AT Response) matching the error_phrase without processing the subsequent lines of the AT response.

If error_safeexit = True, after receiving a line (of AT Response) matching the error_phrase, waits for one more second to receive subsequent lines of the AT response before returning False.

NetworkRegistration

Function –

Checks if the modem is registered on GSM network and if GPRS service is available.

Arguments –

strict_checking type = Boolean default = False

Returns –

If strict_checking = False, issue the AT commands to check if modem registered on the network and GPRS service is available and return True without parsing the AT responses of the commands.

If strict_checking = True, issue the AT commands to check if modem registered on the network and if GPRS service is available and parse the received AT responses to confirm the registration. If registered, return true, else print error codes extracted from the AT responses.

Connect2Internet

Function –

Checks if already connected to GPRS; if not, establishes a new connection.

Arguments –

forced type = Boolean default = False

Returns –

If forced = False, issues AT commands to check if already connected to GPRS and fetch the IP address assigned to the modem. If already connected to GPRS returns True immediately and prints the IP address of the existing connection. If not connected to GPRS already, issues AT commands to attempt a new GPRS connection and returns True and prints the IP address assigned. If attempt for a new GPRS connection fails, returns False.

If force = True, regardless of the status of the existing GPRS connection, connected or disconnected, proceeds to issue AT commands to attempt a new GPRS connection. If a GPRS connection is established successfully, prints the IP address freshly assigned and returns True, else return False.

GET

Function –

Make an HTTP GET request to given URL and pass the GET parameters supplied.

Arguments –

url	type = string	default = "httpbin.org/get"
params	type = dictionary	default = {}
timeout	type = float	default = 50 seconds
Listen2Server	type = Boolean	default = True

'params' is a dictionary containing the GET parameters to send with the URL as Query String

Returns –

True if the HTTP GET request was successful. Else returns False. If Listen2Server = True, parses the HTTP response from the server and prints them.

POST

Function –

Make an HTTP POST request and pass the POST parameters supplied.

Arguments –

url	type = string	default = "httpbin.org/get"
params	type = dictionary	default = {}
timeout	type = float	default = 50 seconds
Listen2Server	type = Boolean	default = True
percentEncode	type = Boolean	default = True

'params' is a dictionary containing the parameters to send in the POST body as Query String.

If percentEncoding = True, the Query String is encoded from ASCII to percent encoding. If set to false, the Query String is passed in pure ASCII, which may cause a problem if the Query String contains special characters like space, comma, etc. On the other hand, using percent encoding increases the size of the POST payload, hence, if it is clear that no special characters may appear in the POST parameters, then we may disable percentEncode which is by default always enabled.

Returns –

True if the HTTP POST request was successful. Else returns False. If Listen2Server = True, parses the HTTP response from the server and prints them.

HTTPread

Function –

Extract HTTP Response from the AT Response of modem and parse the HTTP response from the server

Arguments –

timeout type = float default = 30 seconds

Returns –

An array of lines received from HTTP response, False if a timeout occurs while waiting for HTTP response from the server.

Arduino C++ SIM900 modem Library

For the Arduino board, the library was developed entirely in C++. The important methods/ functions/variables/constant names, their argument, return values and operations are listed below for the C++ library (ISA_GSM.h) –

MACROS

- **ISAmodemResetPin** – Arduino I/O pin connected to modem's reset. (default = 13).
- **ISASerial** – Arduino Serial UART used for debugging
- **gsmSerial** – Arduino Serial UART connected to GSM/GPRS modem
- **GSM_DEBUG** – Enable verbose output to gsmSerial for debugging.

- **GSM_DEBUG_LCD** – LCD object to print debug information to.
- **ISA_GSM_NETWORK_APN** – GPRS Access Point Name
- **ISA_GSM_SERVER** – Server to make GPRS connection to.
  Example - "httpbin.org" or an IP address of the form "w.x.y.z"
- **ISA_GSM_PORT** – TCP port of the server. The default value is "80".
- **ISA_GSM_HTTP_METHOD** – HTTP verb to be used to make the request
  Example - "GET", "POST", "HEAD"
- **ISA_GSM_RELATIVE_URL** – URL to make HTTP request relative to server
  Example – to access the URL "http://httpbin.org/get", ISA_GSM_SERVER is set to "httpbin.org" and ISA_GSM_RELATIVE_URL to "/get".

FUNCTIONS

- void **ISAresetModem** (void)
  - Equivalent to HardReset method from the Python library.
  - Sends a pulse on ISAmodemResetPin Arduino pin to reset the modem.

- void **ISAgsmPIPE** (void)
  - Starts debugging session by mapping ISASerial to gsmSerial bypassing the Arduino board. Similar in function as PIPE function in ISAsim900.py

- int **find_text** (String **needle**, String **haystack**)
  - Searches for the keyword needle in a given string called haystack.
  - Returns 0 if found, else returns -1

- void **gsmSerialInputBufferFlush** (void)
  - Flushes input buffer of gsmSerial

- void **ISAgsmSend** (String **cmd**)
  - Writes the AT command passed as cmd to the modem.

- void **ISAgsmSendRaw** (char ***data**, int **data_length**)
  - Writes data_length bytes of binary data from the character array passed as data.

- String **ISAgsmReadRaw (**char **start,** char **end,** int **lengthToRead,** char **timeout)**
  - Reads characters from gsmSerial receive stream.
  - On encountering the first instance of the character **start**, begins recording the following characters into a response string till any one of the following occurs -
    - the character **end** is received from the stream, OR
    - number of characters recorded reaches lengthToRead, OR

- timeout (in milliseconds) amount of time passes before any of the above two conditions occur.
- Returns the response string.

- String **ISAgsmReadline** (char **timeout**)
  - Reads characters from the gsmSerial receive stream and starts recording them into a response string till any one of these occurs –
    - A <CR>, <LF>, or a <CR><LF> appears in the receive stream, OR
    - timeout (in milliseconds) amount of time passes before the above condition occurs.
  - Returns the response string.

- char **ISAgsmListen** (char **timeout**,
  char **lineCount**,
  String **expectedResponse** = "OK" ,
  String **errorResponse** = "ERROR")
  - Listens to the incoming AT response from gsmSerial and parses the response into separate lines.
  - lineCount is the expected number of lines to be parsed from the AT Response.
  - If any line of the AT response matches/contains the errorResponse**,** then returns 0 immediately.
  - If a timeout occurs before receiving a lineCount number of lines, returns 0.
  - If the number of lines received equals lineCount and the last line received matches/contains the expectedResponse**,** then returns 1.

- char **ISAgsmInit** (void)
  - Equivalent of NetworkRegistration method from the Python library.
  - Issues AT commands to check if SIM card is ready, the modem is registered on the network and if GPRS service is available and parses the AT responses.
  - Returns 1 if all of the above conditions turn out positive, else returns 0.
- char **ISAgsmConnect** (void)
  - Similar in function to the Connect2Internet method from the Python library although using different AT commands.
  - Attempts a new GPRS connection and fetches the assigned IP address.
  - Returns 1 if GPRS connection established successfully, else returns 0.

- char **ISAhttpSend** (String **data**, char ***data_raw**=NULL, int **data_raw_len**=0)
  - Similar in function to the GET or POST method from the Python library although the underlying AT commands used are different.

- Opens a TCP connection to the server ISA_GSM_SERVER at port ISA_GSM_PORT and sends the ASCII string **data** (if data_raw_len = 0) or the binary data present in character array data_raw (if data_raw_len != 0)
- **data** (or data_raw) contains the HTTP headers and body to be sent over the TCP port
- Returns 1 if data over TCP is successfully sent, else returns 0.

- char **ISAgsmATsapbr** (String **APN** = ISA_GSM_NETWORK_APN,
  char **forced** = 0)
  - Equivalent of Connect2Internet method from the Python library
  - Checks if already connected to the internet and if not connected or forced=1, issues AT commands to establish a new GPRS connection.
  - APN is the GPRS access point name, and it defaults to ISA_GSM_NETWORK_APN.
  - Returns 1 if GPRS connection is successfully established, else returns 0.

- char **ISAgsmHTTPRequest** (char **HTTPmethod** = 0,
  String **url** = "",
  String **url_encoded_payload** = "",
  char **listen** = 1)
  - Equivalent to the GET or POST method from the Python library**.**
  - HTTPmethod is 0 for GET, and 1 for POST requests
  - **url** is the URL to make the HTTP request
  - url_encoded_payload is the query string to be passed into the request header (for GET) or the request body (for POST). The query string must be URL encoded before passing it here so as to escape special characters like space, commas, etc. appearing in it. Except the ASCII space character which is replaced by a '+' character, no other special characters have been handled internally.
  - If listen=0 and HTTP request is successful, returns 1 immediately without listening for HTTP reply from the server.
  - If listen=1 and HTTP request is successful, returns 1 after retrieving the HTTP reply from the server and printing the parsed HTTP response.
  - Returns 0 if HTTP request unsuccessful.

Arduino C++ Library for XBee (xbeeapi.h)

The contents of the XBee library (xbeeapi.h) along with their purpose/features are –

MACROS

- XBEE_API_START_BYTE 0x7E

- FRAME_ATCommand 0x08
- FRAME_ATResponse 0x88
- FRAME_TXRequest 0x10
- FRAME_TXStatus 0x8B
- FRAME_RXReceived 0x90
- FRAME_ModemStatus 0x8A

- XBEE_PACKET_SIZE 128
- XBEE_PAYLOAD_SIZE 115

- XBEESerial –

  The Arduino serial UART connected to XBee. It must be one of Serial, Serial1, Serial2, or Serial3.

- XBEEAPI_DEBUG –

  Enable verbose output for debugging purposes.

All the above macros are self-explanatory. The XBEE_PACKET_SIZE is the maximum size of an API frame packet we expect to receive and hence prepare a buffer of this many numbers of bytes. XBEE_PAYLOAD_SIZE is the maximum size of the payload we expect to get after removing the start delimiter, length field, checksum and the frame ID and other frame headers.

CLASSES

- XBeeAPI
  - Constructor Arguments
    - unsigned int baudrate
  - Variables
    - unsigned char packet [XBEE_PACKET_SIZE]

        - unsigned char payload [XBEE_PAYLOAD_SIZE]
        - unsigned char calculated_checksum = 0
        - unsigned char received_checksum;
        - unsigned char frameValid = 1
        - unsigned char frameType = 0
        - unsigned int frameLength=0
        - unsigned char frameID =7
        - unsigned long destMAC64_MSB=0
        - unsigned long destMAC64_LSB=0
        - unsigned long srcMAC64_MSB=0
        - unsigned long srcMAC64_LSB=0;

    - Methods
        - setDestAddr
        - getSrcAddr
        - setFrameType
        - getFrameType
        - setPayload
        - getPayload
        - prepareFrame
        - speak
        - listen

Many variables contained in the XBeeAPi class are self-explanatory. A few who might need some explanation are –

- **packet** is a character buffer to store the incoming or outgoing API frame packet. "packet" is a private variable and is accessible only by the methods provided in the XBeeAPI class. No direct manipulation of this buffer is needed.

- **payload** is a character buffer to store the payload received or to be transmitted. This, too, is a private variable, and no direct manipulation of this buffer is needed.
- **frameValid** – This is a flag which is set to 1 if the received checksum matches the calculated checksum.

- destMAC64_LSB and destMAC64_MSB must be loaded with the 64-bit MAC address of the destination XBee node before making a transmission request using the method speak. By default, the values are set to zero which is a special MAC address of the ZigBee Coordinator. So by default, all transmissions will be intended to the Coordinator. These two variables are accessed by the method prepareFrame

to create the final contents of the packet buffer before the method speak is called to make the transmission.

- srcMAC64_LSB and srcMAC64_MSB are loaded with the 64-bit MAC address of the source XBee node by the method prepareFrame after a packet is received.

The methods provided by the XBeeAPI class are explained in more detail –

setDestAddr

Function –

Sets the variables destMAC64_LSB and destMAC64_MAC to the MAC address of the XBee node to which the next transmission is intended.

Arguments –

unsigned long	MAC_MSB	default=0
unsigned long	MAC_LSB	default=0

Returns –

None

getSrcAddr

Function –

Get the MAC address of the source XBee from which the last packet was received by loading the address into the variables whose pointers are passed as the arguments.

Arguments –

unsigned long*	MAC_MSB
unsigned long*	MAC_LSB

Returns –

None

setFrameType

Function –

Set the frame type of the packet to be transmitted next by loading the variable frameType with the value passed as an argument.

Arguments –

unsigned char	FT

Returns –

None

getFrameType

Function –

Return the frame type of the last received packet.

Arguments –

None

Returns –

unsigned char - Frame type of last received packet.

setPayload

Function –

Set the payload of the packet to be transmitted next by loading the payload buffer with the bytes the supplied character array.

Arguments –

char *	payload_data
int	payload_length

Returns –

None

getPayload

Function –

Fetch the payload from the last received packet by copying the data from the payload buffer into the memory/array supplied in the argument.

Arguments –

char* buffer

Returns –

int - Number of bytes present in buffer payload.

prepareFrame

Function –

Assembles the packet to be transmitted from the payload buffer, frameType, frameLength, destMAC64_LSB, destMAC64_MSB and other such variables and copies the packet to the packet buffer.

Parses/disassembles the packet received and stores the different parts in the payload buffer, frameType, frameLength, srcMAC64_LSB, srcMAC64_MSB and other such variables.

Arguments –

None

Returns –

None

speak

Function –

Send the packet prepared using prepareFrame to the XBee Module.

Arguments –

None

Returns –

None

listen

Function –

Listen to incoming packets from the XBee

Arguments –

unsigned int timeout default = 10000 milliseconds

Returns –

True, if a valid frame is received. Else if an invalid frame is received or a timeout occurs, then returns 0.

Database Library for Raspberry Pi (VdbModel.py)

The methods/functions and their purpose provided by this library are listed below –

CreateDatabase

Function –

Generates the tables in a new SQLite database file

Arguments –

None

Returns –

None

UpdateVehiclesTable

Function –

Inserts or updates a row in the Vehicle table

Arguments –

An array of Vehicle objects where each vehicle object must have a key named ‘MAC64’ containing the 64-bit MAC of the XBee node of the VTE attached to that vehicle.

Returns –

None

InsertVehicleLog

Function –

Inserts a new entry in the Vlog table

Arguments –

addr64, the MAC address of the vehicle

addr16, 16-bit ZigBee network address of the vehicle

data2log, the JSON string sent by the vehicle's VTE

Returns –

None

VehicleLogsFromEachVehicle

Function –

Fetch log from each vehicle contained in the VLog table

Arguments-

maxLogsPerVehicle – Maximum number of logs to fetch from each vehicle

Returns –

Array of log

DeleteInstances

Function –

Delete rows from a table

Arguments –

An array of peewee row objects

Returns –

None

ClearVLog

Function –

Delete all the entries in the VLog table

Arguments –

None

Returns –

None

Web.py service cum Application

Assuming that our VMS lives at the domain “example.com” , the URLs exposed by our Web service cum application are –

- example.com/logpos
    - Type – web service
    - Mapped action - Storing position data sent by VTE/CGE in database

- example.com/Cping
    - Type – web service
    - Mapped action – Record ping (status) messages from CGE into database

- example.com/getpos
    - Type – web application
    - Mapped action – Fetch position logs of vehicles stored in database

- example.com/getAllvehicles
    - Type – web application
    - Mapped action – Fetch details of all vehicles from database

- example.com/getAllLogs
    - Type – Wen Application
    - Mapped action – Get all the position logs present in the database
- example.com/delLog
    - Type – web application
    - Mapped action – Delete one or more logs from database
- example.com/addVehicle
    - Type – web application
    - Mapped action – Add/ Register a new vehicle to the database
- example.com/modVehicle
    - Type – web application
    - Mapped action – Modify/Update an existing vehicle in the database
- example.com/delVehicle

    - Type – web application
    - Mapped action – Delete an existing vehicle in database
- example.com/about
    - Type – web application
    - Mapped action – Display the website’s info/about page in browser
- example.com/report
    - Type – web application
    - Mapped action – Display the vehicle’s position data and speed report page
- example.com/index
    - Type – web service
    - Mapped action – Display the index (landing) page of the website