Modelling Wireless Robots for Urban Search and Rescue in Artificial Rubble

A thesis
submitted in fulfilment
of the requirements for the degree
of
Master of Science in
Electronic and Computer System Engineering
at the
Victoria University of Wellington

By

Tik Wa Charles Tsui

2010

Abstract

Using robots to assist rescue personnel in USAR (Urban Search and Rescue) missions is an active area of research. Researchers are developing robots to penetrate into rubble to gather information about the environment and to search for victims. The School of Engineering and Computer Science of Victoria University of Wellington is developing a team of robots, the "robot family" to help at disasters. The robot family is a three-tier system. The first tier is "the grandmother" which carries second tier "mother robots" to the rubble. The mother robot each launches a group of the third tier "daughter robots" that will penetrate the rubble surface. The daughter robots will burrow deep into the disaster site. They will be equipped with sensors to search for and locate trapped persons. They are designed to be small, battery operated, low cost and disposable. The team of robots is hierarchically structured and to be remotely monitored by rescue personnel at a safe distance from the rubble via a wireless communication link.

This thesis describes the successful implementation of a wireless communication platform for the team of robots. This was verified using a simulated rubble site. A suitable ZigBee wireless module was selected by comparing a list of target brands to form the wireless network. A group of simulated wireless daughter robot models were developed by attaching wireless modules to microcontrollers. An automatic routing wireless network was implemented between the robots. They were deployed into artificial rubble and the communication system was characterised. Proof of concept experiments were carried out and demonstrated that rescue personnel using a computer at a safe distance outside the rubble could successfully establish reliable communication to monitor or control all robots inside the artificial rubble environment.

Acknowledgments

It is a pleasure to thank those who made this thesis possible. First of all, I am grateful to have Professor Dale Carnegie as my primary supervisor, who provided continuous guidance and supportive comments through these years.

I would like to say special thanks to Dr. Len Jennings who led me into the project for urban search and rescue missions by setting up the collaboration between the Victoria University of Wellington and the Manukau Institute of Technology.

The person that I must mention is Dr. Qing Wei Pan. He has taken up the second supervisor position without hesitation after Jennings' left. Without Pan, many of my practical experiments would not have been successful.

I am in debt to Mr. Sunny Yeung who has provided almost instant technical support whenever required.

Finally, I would like to acknowledge Mr. Neel Pandey, Head of School, School of Electrical Engineering and Trades of the Manukau Institute of Technology. Mr. Pandey provided the resources necessary and support to allow staff time-off that is vital to facilitate the completion of this thesis.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 The "robot family" for USAR missions

New Zealand is subject to a variety of natural disasters and non-natural emergencies that may give rise to structural collapses, which could trap people. Examples of such incidents are listed as follows [1].

- Earthquakes, land slips and subsidence
- Hurricanes, typhoons, storms, tornadoes and floods
- Technological and construction accidents
- Terrorist activities

USAR responses are required for such incidents. USAR missions often place rescue personnel at risk [2]. Robots are used to assist in such missions by operating in dangerous rubble scenarios in order to search for trapped victims. A system of robots, "the Robot Family" is proposed as shown in Figure 1.1. The system comprises 'grandmother, mother and daughter' components, each with distinct responsibilities [3][4].

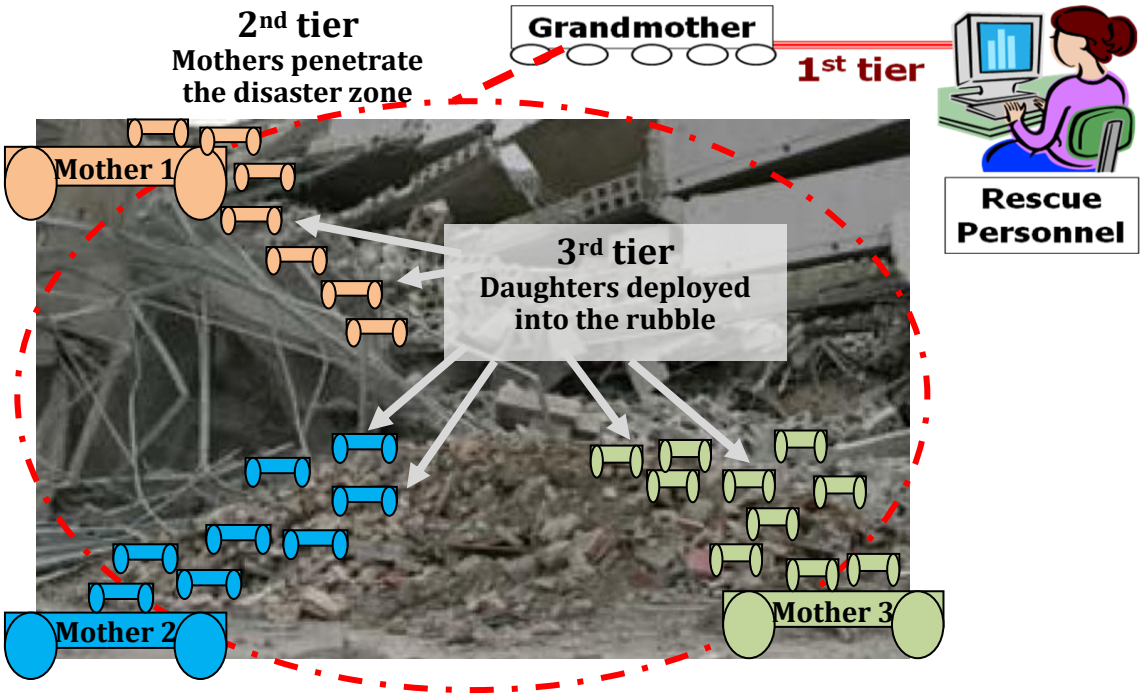

Figure 1.1: The three tier robot family system

The grandmother robot helps to carry mother robots close to the rubble vicinity. It maintains communication with rescue personnel via a long range wireless link, such as a mobile phone network or other radio link which potentially could operate over a distance of several kilometres. A proof of concept grandmother robot (Figure 1.2) is developed by Cordes in 2004 [5]. The mother robots (Figure 1.3) developed by Williamson [6], each carry a group of daughter robots to the perimeter of the rubble while keeping communication with the grandmother robot by a shorter-range wireless link, typically over a distance of a few hundred metres. The groups of daughter robots will be deployed by the mother robots when the mother detects openings that would allow the daughters to penetrate underneath the rubble.

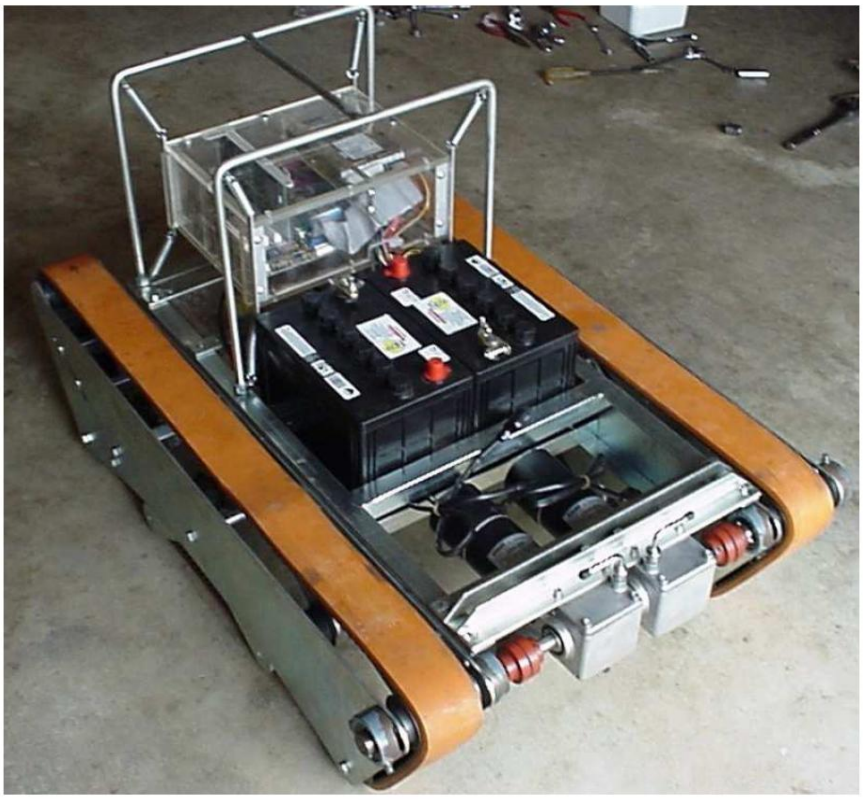

Figure 1.2: Grandmother robot, by Jason Craig Cordes

Figure 1.3: Mother Robot, by David Williamson

1.2 Projects and objective

The daughter robots are designed to be small and disposable. They are battery operated and consume low power. As mentioned, their purpose is to penetrate beneath the rubble surface in order to search for trapped victims. A low power consumption wireless communication network is required to connect the team of robots to facilitate remote monitoring or control of the robots by rescue personnel who are situated at a safe distance from the rubble.

The objective of this thesis is to design and characterise a wireless network for the team of robots and verify that appropriate communication can be established between the robots and rescue personnel outside the rubble.

Studies, literature review and comparisons were carried out on wireless networking technologies. ZigBee was chosen as the appropriate one to implement the wireless network between the mother robots and the daughter robots. A list of ZigBee wireless modules from several brands were acquired and analyzed. A

suitable module that provides the required networking mechanism was selected to form the wireless network. Investigations were carried out on several microcontroller development kit-sets. The most suitable microcontroller was chosen and wireless communication modules were attached. This provided a working simulation of a daughter robot.

The rubble environment was simulated by connecting holes and trenches that were dug in 50 cm deep soil. The simulated robots were placed in the bottom of these holes. The holes and trenches were then covered up by various building materials and soil to simulate an actual rubble environment.

An automatic routing wireless network was implemented on the wireless modules to provide communication between the robots. Proof of concept experiments were carried out and demonstrated that a monitoring computer placed 10 metres outside the rubble successfully established reliable communication with all robots inside the artificial rubble environment.

Chapter 2 Selection of Wireless Link

2.1 Wireless networking architecture

Recent research reports have proposed sparse MANET (Mobile Ad-hoc Networks) as a suitable communication network for robots and other computing devices for search and rescue operations [7] [8]. The sparse MANET concept can be implemented into the three-tier robot system. Each tier will have a separate sub-network. Then all the tiers are interconnected to form a larger MANET [9].

As described in chapter 1, the three tier robot system can be divided into two short-range networks with Grandmother and Mothers connected to one, and the Mothers and Daughters connected to another. The Grandmother can be connected to Rescue Personnel by a longer range link, say GPRS [10] or UMTS [11] on mobile phone or wirelessMAN [12].

Technically, there should be no difficulty in building the MANETs between the rescue personnel, grandmother and mothers. The main concern is maintaining real-time communication with the daughters while they are searching in the rubble. Signal attenuation is the first factor to be considered and the other factor is how to communicate with hundreds of robots at the same time.

2.2 2.4 GHz short-range network for daughter robots

At the start of a mission, an ad-hoc wireless network is established in-situ between the robots. Due to varying situations in the rubble, robots are often disconnected from the network and reconnection is required at any time. Attenuations to the radio signal in the rubble may not allow a direct link between daughter robots in the rubble to the mother outside. The network is required to pass information from one robot to another then to a mother robot; this can be achieved by routing of packetized data. Three short range wireless technologies that can fulfil such requirements are ZigBee, Bluetooth and Wi-Fi. The following summarised the three technologies from Bluetooth SIG [13].

Bluetooth Wireless Technology

- Operate Frequency - 2.4 GHz spectrum
- Operate distance - 10 to 100 metres
- Data Rate - 3 Mbps
- Cost of Bluetooth chips - under US$3

Wi-Fi (IEEE 802.11)

- Operate Frequency - 2.4 GHz or 5 GHz spectrum
- Operate distance - indoor usages, can be extended to outdoor
- Data rate - from 10 Mbps to 100 Mbps (proposals are seeking upwards of 500 Mbps)
- Bluetooth technology costs a third of Wi-Fi to implement
- Bluetooth technology uses a fifth of the power of Wi-Fi

ZigBee (IEEE 802.15.4)

- Operating Frequency - 2.4 GHz, 915 MHz and 868 MHz
- Operating Distance - 10-100 metres
- Date Rate - 20 Kbps to 250 Kbps
- ZigBee and Bluetooth chips are both low cost

Comparing the specifications, ZigBee and Bluetooth are much cheaper to implement than Wi-Fi. Kinney Consulting LLC summarised Bluetooth as best suited for connecting cell phone to PDA, hands-free audio and PDA to printer; whereas ZigBee is better for controls, sensors, lots of devices, low duty cycle small data packets and for projects where long battery life is critical [14].

To differentiate, Wi-Fi has the highest data rate and ZigBee has the lowest. For the daughter robots in the rubble, the messages that they will send includes temperature and gas data and whether a victim has been found. Such messages should have a length at the most of several bytes. The largest messages could be sending a kilo-bytes size photo, but that should not be very often except when

confirmation of a victim's situation is required. Thus, the data rate provided by ZigBee is adequate.

Summarising, ZigBee is chosen as the communication technology for the daughter robot. Investigations by proof of concept experiments and discussions in later section supported this selection.

2.3 Radio frequency penetrations in collapsed buildings

According to Akl, Tummala and Li [15], indoor path loss for wireless data link at 2.4 GHz in various room conditions at a distance of 10 m to 30 m could be in the range from 30 dB to 40 dB.

Estimation of penetration loss through walls and partitions can refer to Osama's theoretical models [16]. A 13 inch thick concrete external wall with insulation and metal enforcement could introduce 15 dB attenuation and a 4.5 inch internal dry wall could impose 5 dB penetration losses. This estimation also matched with the XBee-PRO manufacturer's test report [17].

As a total estimation, there could be 60 dB losses for a daughter robot inside a normal building to communication with a mother robot outside. During USAR mission in a collapsed building, the signal loss will be much higher than 60 dB. Most ZigBee modules would not have enough power to get through this rubble to make a direct link with the outside.

The solution requires network routing between the daughter robots, such that the robot that is nearest to the outside of the building can communicate with the mother robot with a direct link; other robots that are deeper inside the rubble establish a network through to the outermost robot.

2.4 ZigBee network for thousands of robots

ZigBee (IEEE 802.15.4) networking protocol supports cluster tree topology that is suitable for the network routing scenario described above [18]. A ZigBee device can be configured as an FFD (Full Function Device) or RFD (Reduced Function Device). Each device has a 64 bit IEEE address. 16 bit short addresses can be used to increase message passing efficiency. This allows a maximum of 65535 devices on the network.

The XBee-PRO modules on the daughter robots will be programmed by the v8x17 version firmware provided by MaxStream to work as FFD, which can function as either a router or an end-node in a cluster tree network and carry out message routing between devices [18]. This firmware works on top of the ZigBee network layout and implements the cluster tree network topology illustrated in Figure 2.1. The firmware supports 5 level routing. It allows 6 routers and 14 end-nodes in each level. Thus there can be maximum 31100 daughter robots supported with one coordinator in one network.

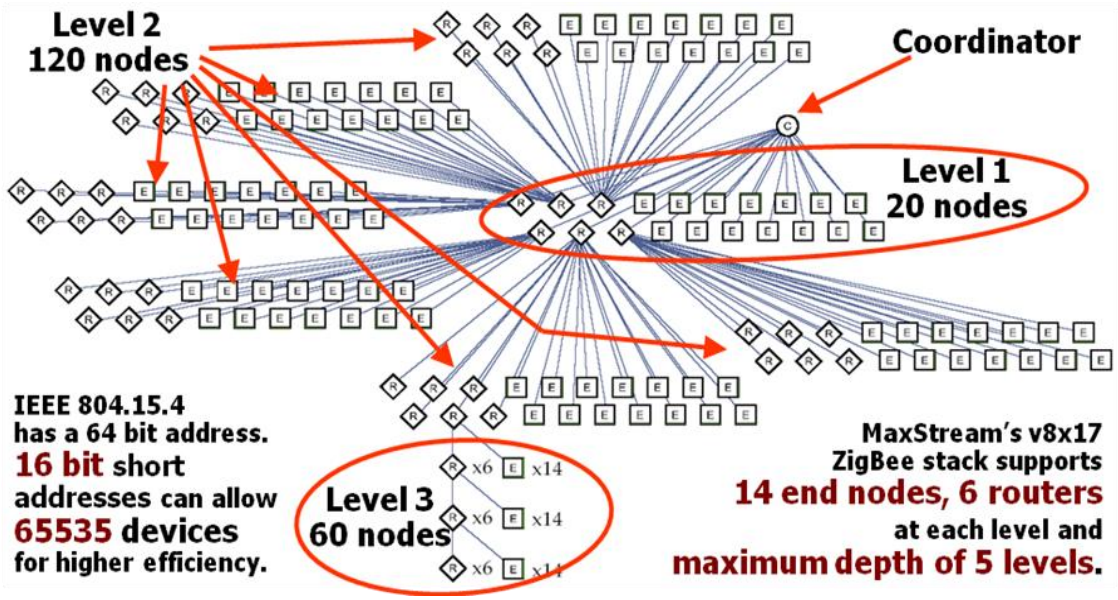

Figure 2.1: Cluster Tree ZigBee Network

In order to avoid prolonged delay in routing between levels and in practice there will not be 100 to 200 daughter robots in a USAR mission, a two level network which allows 140 robots or a three level network which allows 860 robots could be used.

2.5 ZigBee Modules Comparison

Four development kit sets or modules were sourced and analysed for their suitability for the projects. Significant specifications and features of each are listed below.

2.5.1 Freescale 13193EVB-BDM Development Kit

- Cost
  NZ$545 (ex. GST) from Arrow Electronics as of 2006 September, for a pair of evaluation boards
- Firmware available
  Full ZigBee stack source codes with application examples for pairs of coordinator and end-node
- Development tool
  Freescale's CodeWarrior for re-programming the embedded GT60 microcontroller and 13193EVK software for loading and re-configuration of application examples on to modules
- Interfaces
  RS232, USB 1.1, 4 push buttons, 4 LEDs
- Output power
  0 dBm (1 mW) typical, 3 dBm (2 mW) maximum
- Input power
  5 V – 9 V DC

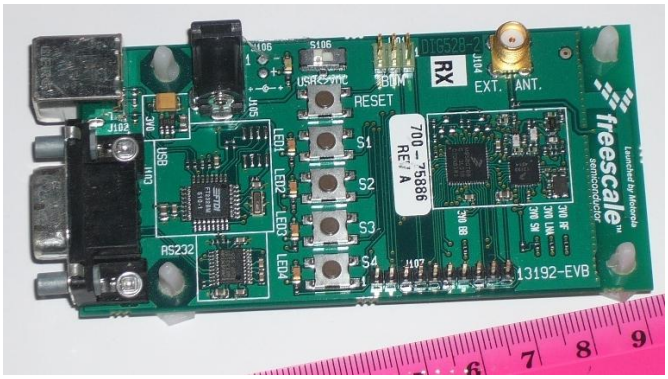

Figure 2.2: Freescale 13193EVB-BDM Development Kit

2.5.2 Panasonic PAN802154HAR00 Module [19]

- Cost
  US$28.01 (ex. GST) from Arrow Electronics as of 2007 March
- Firmware available
  One coordinator with three end-nodes and full ZigBee stack source code from Freescale
- Development tool
  Embedded GT60 microcontroller, can be re-programmed by Freescale's CodeWarrior
- Interfaces
  RS232, 2 analogue input, 8 digital I/O
- Output power
  0 dBm (1 mW) typical, 3 dBm (2 mW) maximum
- Input power
  3.0 V - 3.4 V DC (for using with RS232)

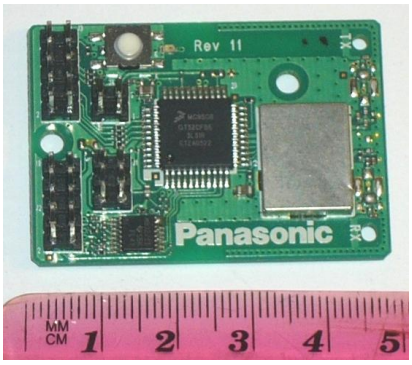

Figure 2.3: Panasonic PAN802154HAR00 Module

2.5.3 Microchip Technology PICDEM Z 2.4GHz Demo Kit

- Cost
  US$199.99 (ex. GST) from Microchip as of 2006 July, for a pair of development boards
- Firmware available
  Full ZigBee stack source codes with application examples for pairs of coordinator and end-node
- Development tool
  Microchips's MPLAB C18 can be used to develop the full ZigBee stack source code and re-programming of the embedded PIC18LF4620 microcontroller

- Interfaces
  RS232, 2 LEDs, 1 temperature sensor, analogue and digital I/O on the microcontroller

- Output power
  0 dBm (1 mW), typical and maximum
- Input power
  5 V – 9 V DC

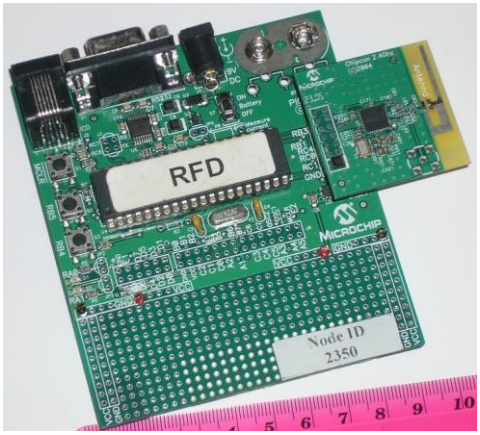

Figure 2.4: Microchip Technology PICDEM Z 2.4GHz Demo Kit

2.5.4 XBee-PRO OEM RF Module [18]

- Cost
  NZ$47.05 (ex. GST) from TCS (NZ) LTD as of 2007 October
- Firmware available
  Several versions of firmware that supports mesh and cluster tree network of one coordinator with numerous routes and end-nodes, but ZigBee Stack source code not provided
- Development tool
  Firmware programming and re-configuration software provided
- Interfaces
  RS232 (TTL level), 8 I/O pins reconfigurable as 5 analogue input or 8 digital I/O
- Output power
  18 dBm (60 mW) typical, 20 dBm (100 mW) maximum
- Input power
  2.8 V - 3.4 V DC

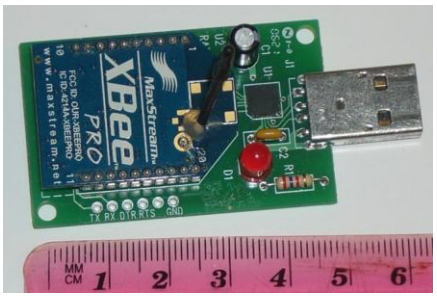

Figure 2.5: XBee-PRO OEM RF Module on USB adaptor

After loading and running demonstration examples on the four ZigBee modules, it was found that the time taken to become familiarised with the Freescale 13193EVB-BDM Development Kit and Microchip Technology PICDEM Z 2.4GHz Demo Kit was not justified within the limited project time frame.

Summarizing the specifications and features, and based on the project criteria of low cost, small size and easy integration with the robots, the PAN802154 module by Panasonic and the XBee-PRO module by MaxStream were selected for developing the prototype robots. Technical specifications for the modules are listed in Table 2.1 below.

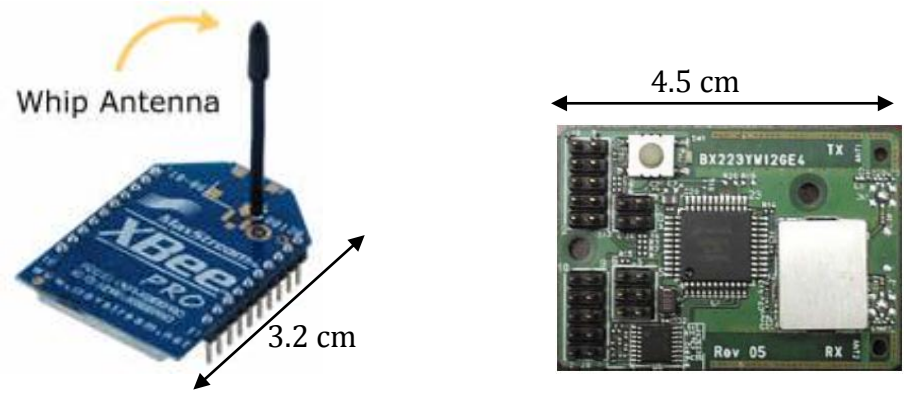

Figure 2.6: XBee-PRO (left) and PAN802154 (right) Modules

Table 2.1: ZigBee module specifications

	XBee-PRO	**PAN802154**
RF Output Power	18 dBm (60 mW) typical 20 dBm (100 mW) max	0 dBm (1 mW) typical 3 dBm (2 mW) max
Receive Sensitivity	-100 dBm typical	-92 dBm typical
Power Consumption	Tx: 270 mA Rx: 55 mA	Tx: 60 mA Rx: 35 mA
Antenna	¼ monopole integrated whip antenna	Printed onboard

Chapter 3 Prototype Development

3.1 Rapid Prototype Development

Due to the limited time allowed for the project, the technique of rapid prototyping is applied to the development of prototype robots. The process started from the functional design along with mechanical and electrical design to meet the required functionalities. The next step is hardware sourcing and combining parts to assemble the robot in parallel with programming of the robot. Tests and corrections were carried out and then looped back to modification on the design, with re-programming and re-assembly if required, as shown in Figure 3.1 below.

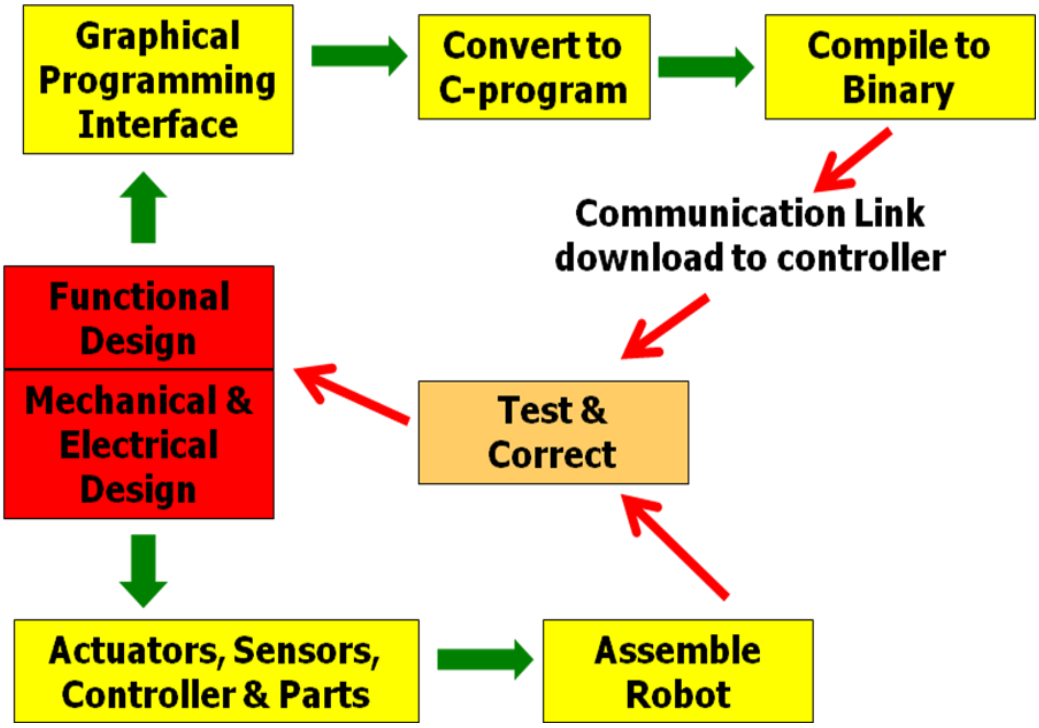

Figure 3.1: Rapid Prototype Development

Based on this rapid development concept, several prototype robots were developed and illustrated in the following sections.

3.2 1st Prototype - Microcontroller with ZigBee

The design aim of the 1st prototype is a 'quick start tester' that allows testing of message transfer on the communication link. The prototype does not need to be a mobile robot but need to be built in the shortest time. A PIC16F877 target board which provides a programmable serial port and interfaces was acquired and connected with the XBee-PRO module to form the first prototype. The block diagram below illustrates the design.

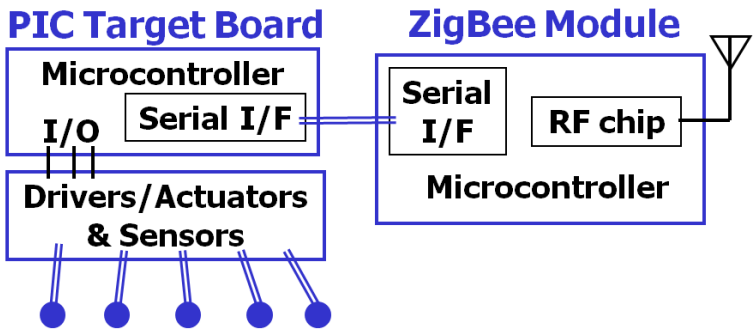

Figure 3.2: Block Diagram of 1st Prototype

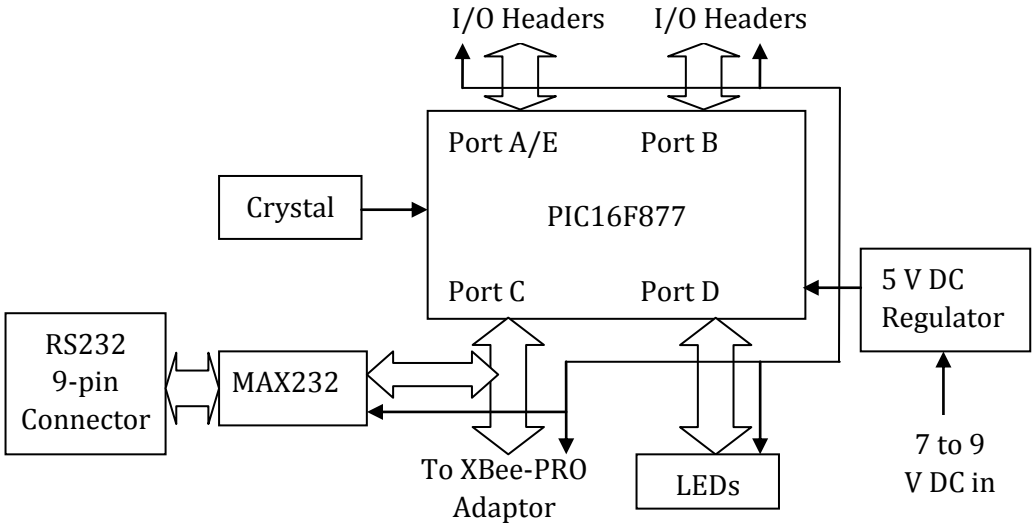

Figure 3.3: PIC16F877 target board block diagram

The above block diagram illustrates the PIC16F877 target board. A 5 V DC regulator provides power to the microcontroller and all the ports. The onboard MAX232 level shifts the data output to the required level for the RS232 connector which can be used for connecting to a computer. The data output can be shared for connecting to the XBee-PRO wireless module.

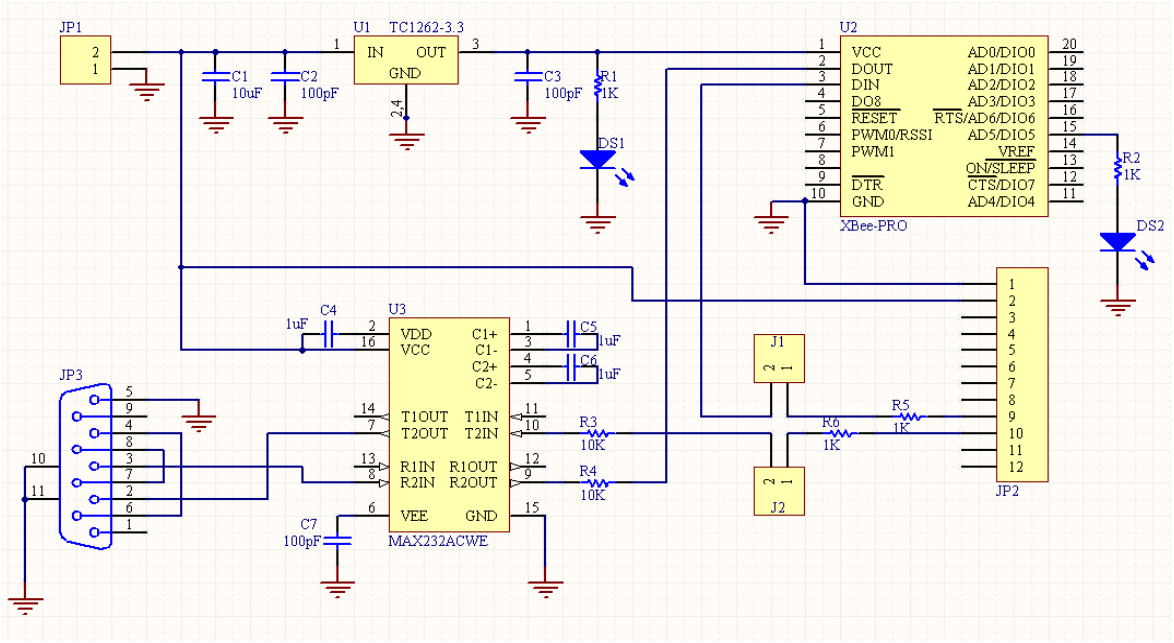

Figure 3.4: XBee-PRO adaptor board

Since the XBee-PRO is an OEM module, an adaptor board is designed to provide the necessary voltages and connections. The above schematic diagram shows the adaptor board. U1 (TC1262-3.3) provides regulated 3.3 V DC to the circuit. Data lines DOUT and DIN from U2 (the XBee-PRO module) are level shifted by U3 (the MAX232) to provide the required signal level to the 9-pin RS232 connector JP3 which can be used for connecting to a computer. When jumpers are connected onto J1 and J2, the header JP2 can be used to directly connect the data line to Port C of the PIC16F877 target board on Figure 3.3. The following figure illustrates the assembled prototype.

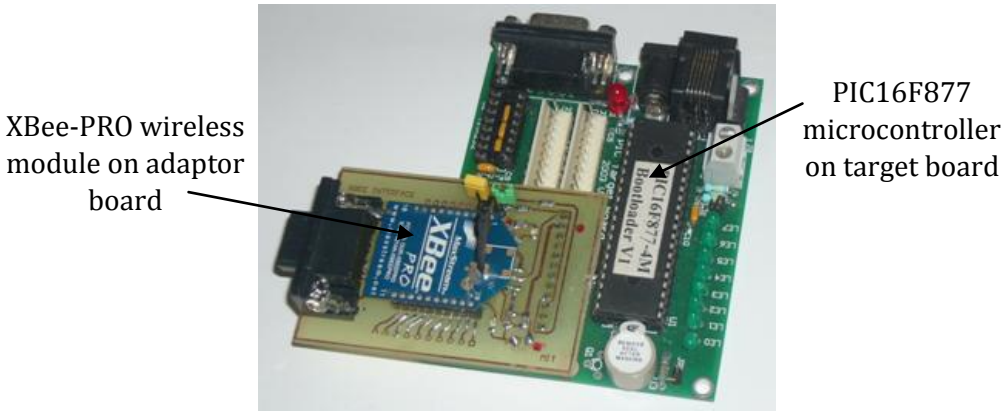

Figure 3.5: 1st Prototype - Xbee-PRO on PIC877 target board

3.3 2nd Prototype - The beetle with ZigBee

The second wireless module candidate is the Panasonic PAN802154. Figure 3.6 illustrates the block diagram of the module. It has an onboard micro-processing unit (MC9S08GT60) which implements the ZigBee stack and a RS232 level shifter for direct connection to any serial port.

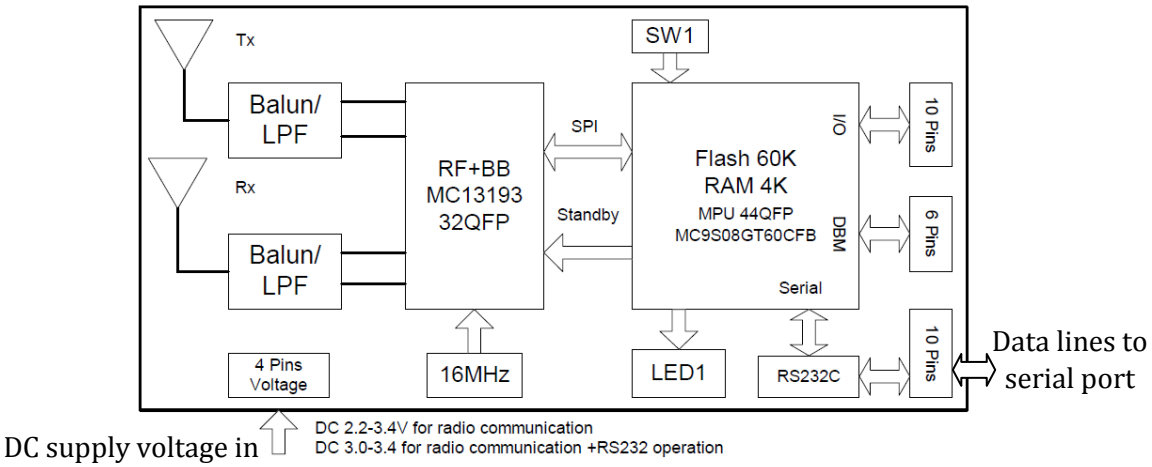

Figure 3.6: PAN802154 wireless module block diagram

The beetle was developed by modifying a low cost (NZ$30) toy car. The original 27 MHz radio remote control circuit was taken out and replaced by the PIC16F877 target board as illustrated above. The 2nd prototype robot is created by connecting the Panasonic PAN802154 module to the serial port of the target board inside the Beetle as illustrated in the photograph below.

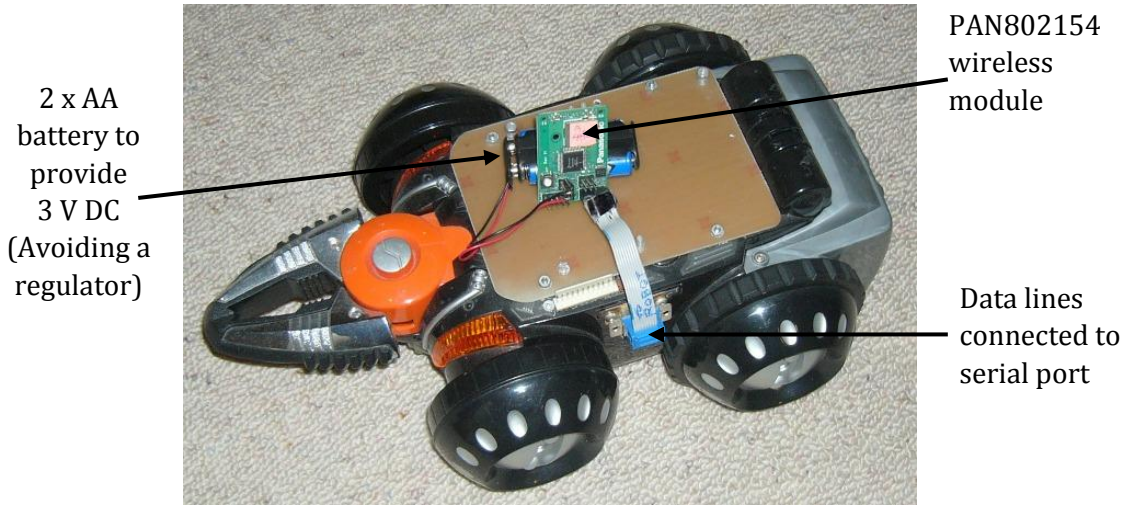

Figure 3.7: 2nd Prototype - The beetle with ZigBee

3.4 3rd Prototype - Two-motor robot with ZigBee

The size of the PIC16F877 target board and the Beetle is large. A new microcontroller board based on the PIC18F4550 with a base adaptor board which contains motor driving circuit and connectors for the Panasonic PAN802154 module was developed. The two boards were assembled on a modified two-motor body from the Beetle to form the 3rd prototype illustrated in the photographs below.

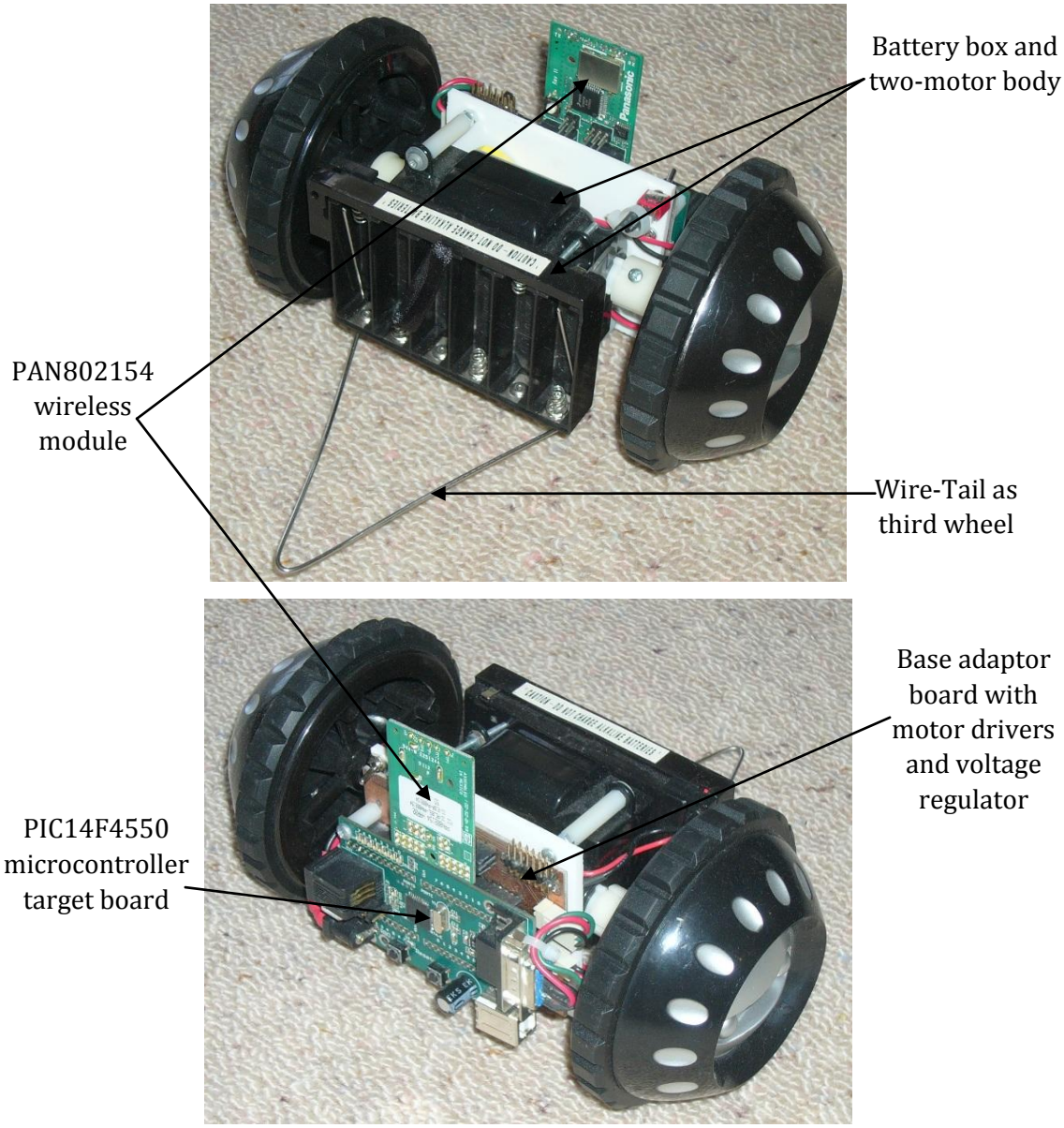

Figure 3.8: 3rd Prototype - Two-motor robot with ZigBee

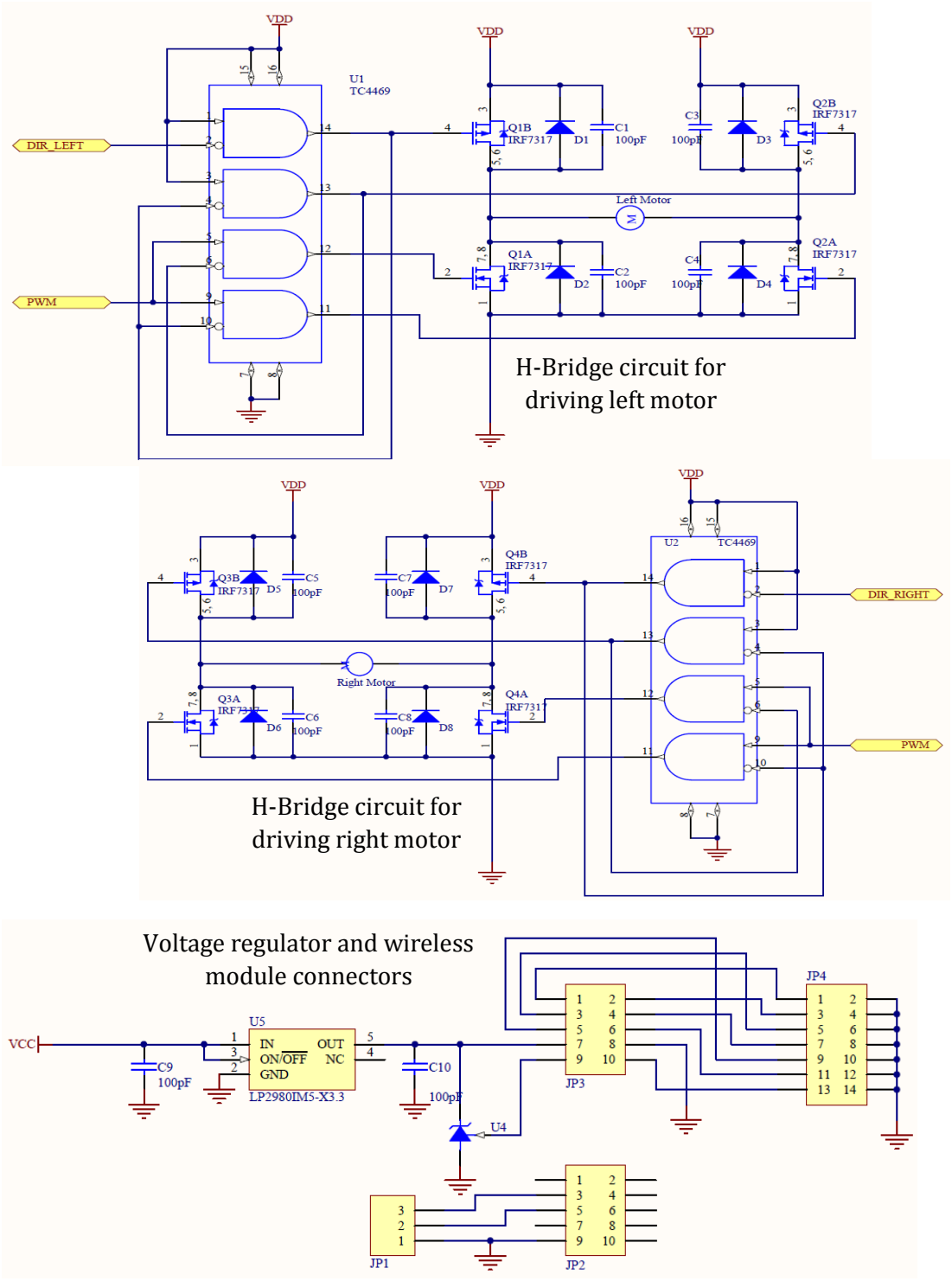

Figure 3.9: Two-motor robot - base board schematic diagram

Figure 3.9 illustrates the schematic diagram of the base adaptor board on the two-motor robot. It consists of three main parts. They are two identical H-Bridge circuits formed by four MOSFET (Q1A-B & Q2A-B or Q3A-B & Q4A-B) driving the left and right motors.

Each H-Bridge is driven by NAND-Gate Drivers (U1 or U2) that are themselves driven by the DIR_LEFT, DIR_RIGHT and PWM signals from the PIC16F4550 target board. The PWM signal controls the speed of the motor. The left and right signals control which motor to run.

The third part of the schematic is the 3.3 V voltage regulators (U5) that provides the required power. The connectors, J2, J3 and J4 work as plug-in sockets for the PAN802154 wireless module. JP1 is for connecting the RS232 lines between the wireless module and the target board.

3.5 4th prototype - The SRV-1 Surveyor

The two-motor robot (the 3rd prototype) demonstrated a scaled down version of the beetle (the 2nd prototype). However, using the wire-tail to act as the third wheel cannot provide proper movement on uneven surface. A better mechanical structure is required.

While searching for a better mechanical structure, a tank like robot (the SRV-1 Surveyor) was found. This 4th prototype robot is a completed robot bought from Surveyor Corporation [20]. The Surveyor contains a Phillips microcontroller with a plug-in XBee-PRO module, which is the chosen ZigBee module for the projects described in earlier sections. The photographs below illustrate the Surveyor robot and the XBee-PRO on a USB adaptor board.

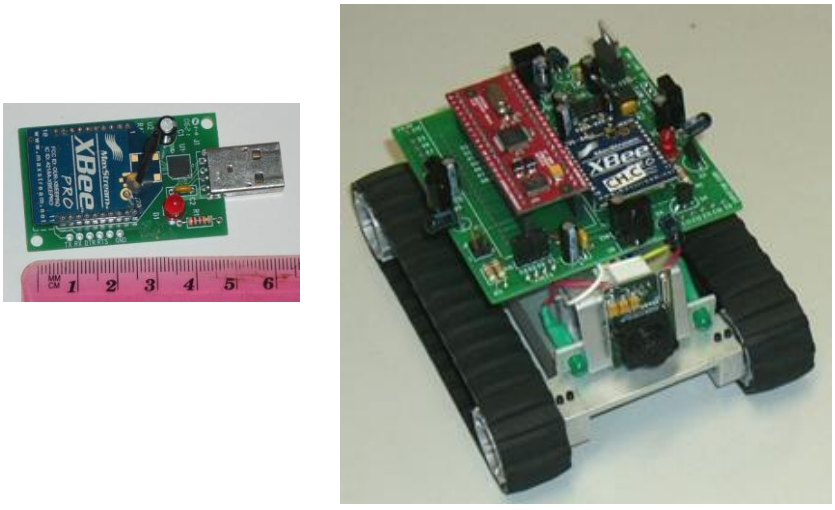

Figure 3.10: 4th Prototype - The Surveyor

3.6 Final Prototype - RoboExp with Sensors

The final prototype is made by attaching a ZigBee module onto the customizable robot kit-set, RoboExp robot purchased from JoinMax [21]. The RoboExp robot comes with collision detectors and an ultrasonic detector. A PIR (passive infrared) detector and a temperature sensor were designed and added on to the robot. Figure 3.11 illustrates the prototype robot.

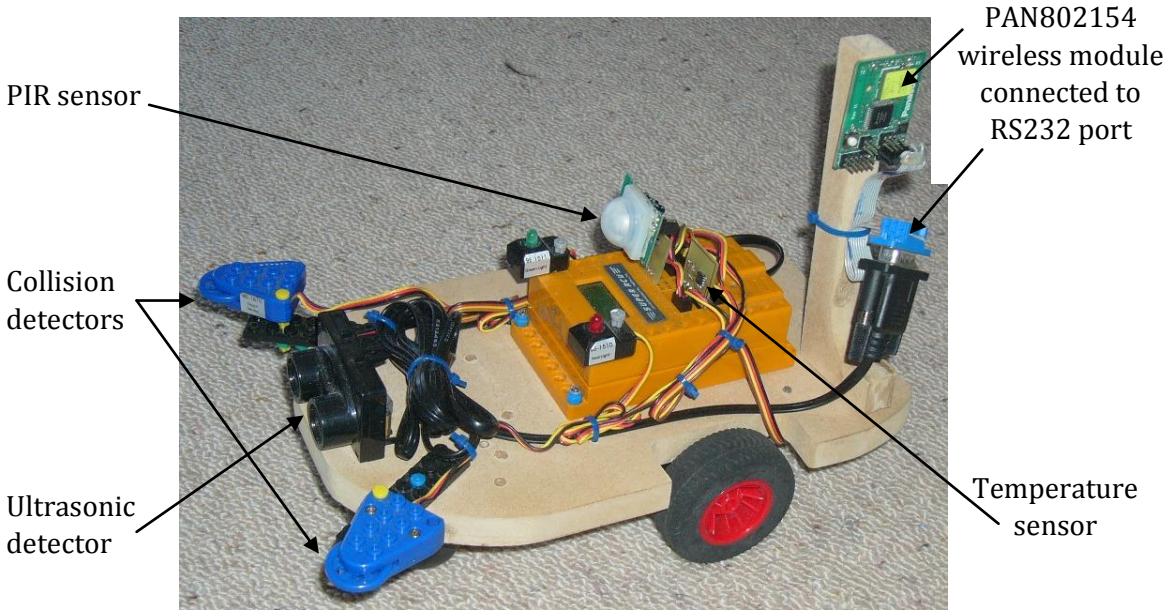

Figure 3.11: The Final Prototype - RoboExp Robot with sensors

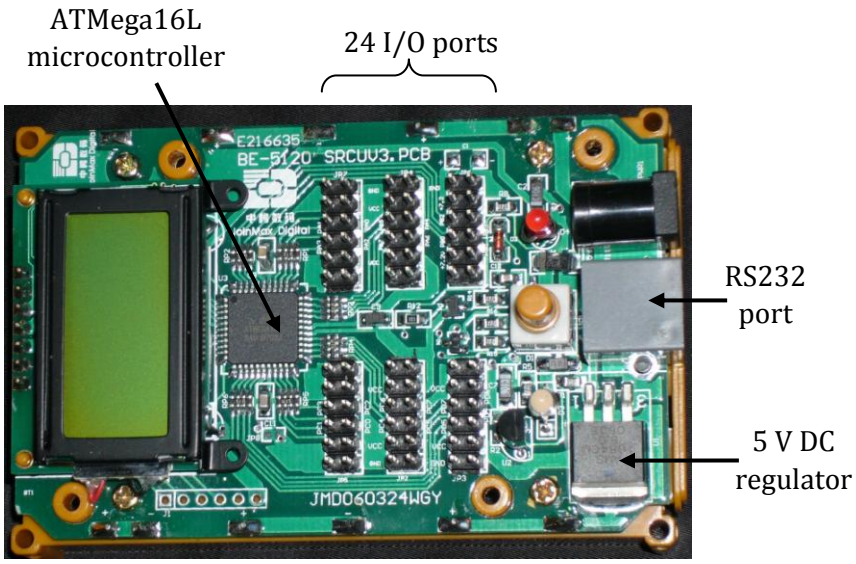

Figure 3.12: ATMega16L microcontroller of the RoboExp Robot

3.6.1 RoboExp Robot with ATMega16L microcontroller

Figure 3.12 illustrates the controller on the robot. It is based on an ATMega16L microcontroller providing 24 I/O ports which can be programmed as motor output, analogue input, I²C interface, or digital I/O. These ports facilitate easy interfacing of the RoboExp robot's sensors to the microcontroller. As illustrated in Figure 3.11, the PIR sensor is interfaced using a digital input port and the temperature sensor is connected onto the I²C interface.

An added advantage of the RoboExp robot to rapid development is that the software that comes with the controller provides a drag-drop-and-connect icon style programming interface and also supports C++ programming in a text editor. This allows a quick start-up of an application in minutes with detail and complex algorithm to be developed in a later stage.

3.6.2 Adding I²C temperature sensor to the RoboExp controller

To demonstrate easy interfacing of sensors, a digital temperature sensor board with based on Microchip's MCP9803 I²C was designed and constructed. Figure 3.13 shows the temperature sensor schematic and the finished PCB that was added onto the controller. The 6-pin port takes the 5 V DC at pin 2 to power the circuit and connects the I²C bus to the controller.

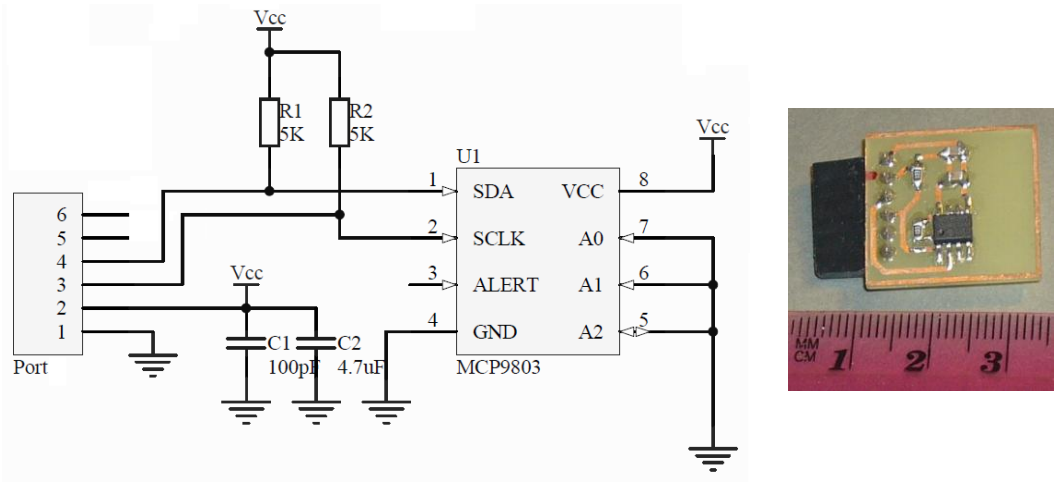

Figure 3.13: Schematic and photo of the I²C temperature sensor

The following C code subroutines was written and added to the source files to allow the controller to set up and read temperature from the sensor.

Line	SetI2CTemperature subroutine C code
1	`unsigned int SetI2CTemperature(_TEMPERATURE_ which){`
2	`      // Initialise I2C temperature module MCP9803`
3	`    struct select *information=&which;`
4	`    unsigned char ddr1=portarray[1][information->group1];`
5	`    unsigned char ddr2=portarray[1][information->group2];`
6	`    unsigned char port1=portarray[2][information->group1];`
7	`    unsigned char port2=portarray[2][information->group2];`
8	`    unsigned char pin1=portarray[0][information->group1];`
9	`    unsigned char pin2=portarray[0][information->group2];`
10	`    unsigned char bit1=information->bit1;`
11	`    unsigned char bit2=information->bit2;`
12	`    SDA_DDR=ddr2;`
13	`    SDA_PORT=port2;`
14	`    SDA_PIN=pin2;`
15	`    SDA_BIT=bit2;`
16	`    SCL_DDR=ddr1;`
17	`    SCL_PORT=port1;`
18	`    SCL_BIT=bit1;`
19	`    unsigned int errorCode=0;`
20	`    Soft_I2C_Start();`
21	`    Soft_I2C_Write(0x90);`
22	`        // write address byte - R/W bit should be 0`
23	`    Soft_I2C_Write(0x01);`
24	`        // point to configuration register`
25	`    Soft_I2C_Start();`
26	`    Soft_I2C_Write(0x40);`
27	`        // set to 11bit resolution`
28	`    Soft_I2C_Start();`
29	`    Soft_I2C_Write(0x90);`
30	`        // write address byte - R/W bit should be 0`
31	`    Soft_I2C_Write(0x0);`
32	`        // point to temperature register`
33	`    Soft_I2C_Stop();`
34	`    return errorCode;`
35	`  }`

Figure 3.14: Subroutine to setup I²C temperature sensor

The SetI2CTemperature subroutine sets up the required port address and bits for the I²C port. To fulfil the requirements to set up the MCP9803, lines 20 to 27 call the Soft_I2C_Start and Soft_I2C_Write subroutines to write the resolution parameter (0x40) into the configuration register (0x01) at the I²C address 0x90. Lines 28 to 33 set the sensor to point to the temperature register (0x0) and then stop to get ready and wait for commands to read temperature values.

Line	GetI2CTemperature subroutine C code
1	`unsigned int GetI2CTemperature(_TEMPERATURE_ which){`
2	`   // Read from I2C temperature module MCP9803`
3	`    struct select *information=&which;`
4	`    unsigned char ddr1=portarray[1][information->group1];`
5	`    unsigned char ddr2=portarray[1][information->group2];`
6	`    unsigned char port1=portarray[2][information->group1];`
7	`    unsigned char port2=portarray[2][information->group2];`
8	`    unsigned char pin1=portarray[0][information->group1];`
9	`    unsigned char pin2=portarray[0][information->group2];`
10	`    unsigned char bit1=information->bit1;`
11	`    unsigned char bit2=information->bit2;`
12	`    SDA_DDR=ddr2;`
13	`    SDA_PORT=port2;`
14	`    SDA_PIN=pin2;`
15	`    SDA_BIT=bit2;`
16	`    SCL_DDR=ddr1;`
17	`    SCL_PORT=port1;`
18	`    SCL_BIT=bit1;`
19	`    unsigned char byteTwo;`
20	`    unsigned char byteOne;`
21	`    unsigned int temp;`
22	`    Soft_I2C_Start();`
23	`    Soft_I2C_Write(0x91);`
24	`         // write address byte for reading temperature`
25	`    byteOne=Soft_I2C_Read(0);`
26	`    byteTwo=Soft_I2C_Read(1);`
27	`    Soft_I2C_Stop();`
28	`    temp = (byteTwo << 8) | byteOne;`
29	`         // combine two bytes to form the reading`
30	`    return temp;`
31	`  }`

Figure 3.15: Subroutine to read I²C temperature reading

The GetI2CTemperature subroutine has the same starting part as the set up subroutine, using the same port address and bits for the I²C port. Lines 22 and 23 call subroutines to put the address 0x91 on the bus for reading the MCP9803 module. Lines 25 to 27 read the two bytes of the temperature value and then stop the operation. Lines 28 and 29 combine the two bytes to form the temperature reading and return it.

The Soft_I2C Start, Read, Write and Stop subroutines are provided by the RoboExp development software.

3.7 Summary of Prototype Development

The aim of developing all the prototypes is to source and select hardware to build a platform that is suitable for further project experiments. The platform must fulfil the concept of "rapid prototyping", such that it can be easily re-configured, re-programmed and be adapted with sensors and motors for constructing simulated robots.

The wireless modules are programmed as pairs of point-to-point wireless modems and messages were sent between a computer and the prototypes. The following table summarises and comments on these test results.

Table 3.1: Experiments and comments on prototypes

Proto-type	Experiments & Comments
1st	• A pair of XBee-PRO modules at 4800 baud • 999 bytes message was received properly at the computer by sending the one-byte, 'H' request message • The baud rate is low and need to test motor driving to check noise immunity
2nd	• A pair of PAN802154 modules at 4800 baud • The Beetle (contains the same PIC16F877 as 1st prototype) • Motor driving tested and 999 bytes message tests without problem
3rd	• A new microcontroller board, PIC18F4550 was developed • Body of the 2nd prototype was modified to form a two-motor body • A pair of PAN802154 modules at 38400 baud • Same 999 bytes message and motor driving tests without problem
4th	• A pair of XBee-PRO modules at 115200 baud • Using the robot's Java based remote control program, but intermittent loss of photos was found. A new Java program was written to test at other baud rates, photos can be captured, but losses still happened. • Re-programming of the robot for the same test as the other prototypes was not carried out due to time limitation, but expected the robot can achieve the same as other prototypes, since photos can be captured.
Final	• RoboExp with the ATMega16L microcontroller was programmed to be tested the same as the 3rd prototype • Both XBee-PRO and PAN802154 have been used • 999 bytes message and driving tests were done without problem on various baud rates from 9600 to 115200

Summarising the above discussions, building a model of daughter robot can start from one of the last two prototypes. The Surveyor features an attractive camera that allows users to see "what is going on in there", but some lighting is required. The RoboExp is easily reconfigurable and with its I^2C ports cater for new sensors and actuators. However, both of them need "shape-up" to keep them from getting stuck in adverse situations.

The two selected wireless modules are well justified to continue with the next stage of experiments as both can be reconfigured with various baud rates and be direct connected to a standard RS232 interface on the microcontroller boards. Using the structure as the 1st prototype, both of the modules are attached to a microcontroller and used for the "feasibility tests" described in next chapter.

3.8 Prototype Robots Cost Analysis

The final prototype of the daughter robot as described in section 3.6 is based on the educational kit set RoboEXP robot [21]. The cost of a kit set is US$159.30. The kit set contains parts, such as two motors, two tyres, a front wheel, two collision sensors, one sound sensor and two light sensors, for building the robot framework. Extra sensors such as an ultrasonic distance detector (US$30), a PIR sensor (US$10) and an I^2C temperature sensor (US$20) were added for function enhancement. Adding an XBee-PRO module (NZ$47.05) completed a wireless prototype daughter robot at about US$250.

The 4th prototype, the SRV-1 Surveyor robot costs US$525 from Surveyor Corporation. The SRV-1 is completed with a miniaturized camera and a pair of laser lights for range finding. It has a pair of mini-size DC motors built into to an aluminium chassis, which drives a pair of rubber belts to form a self-laying track configuration [20].

The RoboExp robot is cheaper, but the SRV-1 has a better chassis and comes with a second XBee-PRO module on a USB adaptor for connecting to the computer. For the additional cost of a hundred dollars, the RoboExp can be built onto a similar aluminium chassis with motors to make it more or less the same cost and same structure.

The cost of these prototypes being of the order of several hundred dollars is acceptable and within the budget allocated to this project. No claim is made here what the final cost of a daughter robot might be once factors such as mass production and reliability/robustness of the device are taken into account.

If only signal routing is required at locations inside the rubble, further cost cutting can be done by using the mini-router describe in later section 4.3.2. This consists of two AA-batteries and an XBee-PRO module that can be made below US$30.

Chapter 4 RF Signal Tests

The next stage of the project is to run experiments to verify that the selected ZigBee modules can establish a reliable communication link for the daughter robots in USAR missions.

4.1 Wireless Link Test

4.1.1 Experiment setup

The 1st prototype illustrated in section 3.2 was used to carry out the experiments. The ZigBee modules are used in pairs and configured as wireless modems connecting a computer on one side and to the microcontroller on the other side. A single byte message was sent from the computer to the microcontroller, which was programmed to respond with a 300 byte text message. The communication was tested with the 300 byte response message verified for two scenarios described below.

4.1.2 Scenario 1: A normal office building

Searching for victims in collapsed buildings will be the normal mission that daughter robots designed for. It is reasonable to expect that in such a collapse, there will be substantial cavities (necessary to contain a surviving victim) as well as gaps in the rubble (one of which will admit the daughter search robot). A four level office building (Figure 4.1) was chosen to run the first set of GO/NO GO tests. Although this building has obviously not collapsed, it will serve to model the actual disaster environment given the commonality of building materials and structural layout.

The main structure of the building is supported by twenty-five 0.6 m by 0.6 m concrete columns. The floor of each level is a 0.3 m thick concrete slab. The floor area of each level is 30 m by 30 m and 3 m height. Each floor is divided into offices, class rooms, laboratories and computer rooms. The main stairway, toilets and the lift are grouped at the middle of the building and a back door stairway is on the north.

Figure 4.1: Office building for GO/NO GO tests

The robot was placed on a desk inside one of the rooms at the north side in second floor. The communication test was carried out by moving the computer to various locations inside and outside the building. Figure 4.2 illustrates the location of the ZigBee and micro-controller module on the second floor.

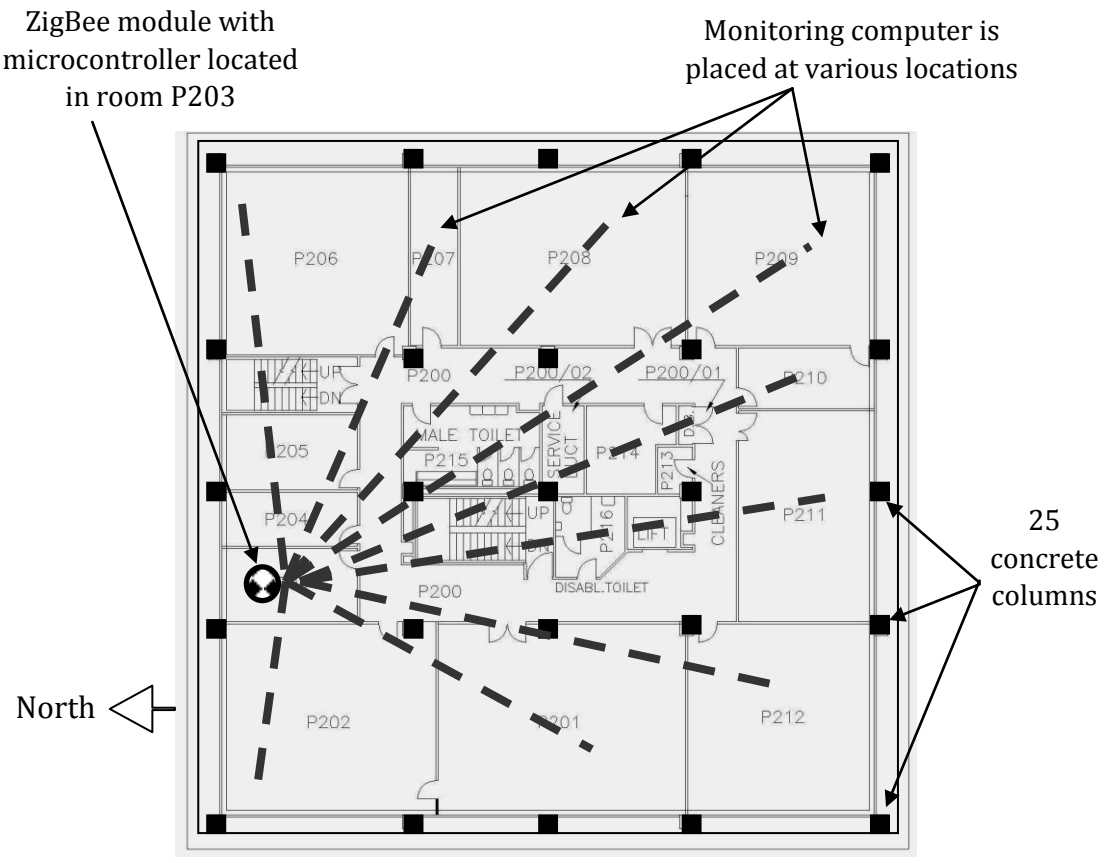

Figure 4.2: Floor plan for communication test

Results on reception of respond message for each floor are summarised in table 4.1 below.

Table 4.1: Office building communication test results

	Using XBee-PRO with microcontroller	**Using PAN802154 with microcontroller**
Floor 1	All bytes received in all rooms, corridors, stairways and foyer; except in rooms at south-west corner of building.	All bytes received in corridors, stairway, and foyer; but no respond message in rooms.
Floor 2	All bytes received in all rooms on second floor, corridors, and stairways.	All bytes received in rooms at north, corridors and stairways; but rooms at middle & south end only get respond message near doorways.
Floor 3	All bytes received in rooms at north, and corridors; but rooms at middle & south end only get respond message near doorways.	All bytes received in north half of the corridors, and stairways only.
Floor 4	All bytes received in corridors and stairways only.	All bytes received at stairways only.
Outside	All bytes received within 3m at north, north-east corner, and outside main entrance (even with glass door closed).	All bytes received only when very close to the room of the ZigBee modules at north of the building.

From the table, the range of the XBee-PRO module covers a lot more area than the PAN802154 module, including areas outside the building. This is mainly due to the XBee-PRO's maximum output power of 100 mW which is much higher than that of the PAN802154 at 2 mW. Note that for the actual experiments both devices were configured to transmit at their highest output power.

4.1.3 Scenario 2: Metallic effects

Reinforced concrete is the major building material used for the building. Another commonly found material is metal from electric appliances, such as computers. The following two GO/NO GO tests addressed this.

A. Aluminium shield

A 1.25 mm thick aluminium dome was used to shield the RF signal transmitting from the ZigBee module. A spectrum analyser with a dipole antenna was used to measure the received signal strength. Due to the shield was not grounded with respect to the RF transmitter; it became a re-radiator that produces secondary radiation of the signal [22]. An attenuation of about 15 dB was measured. The arrangement is illustrated in Figure 4.3.

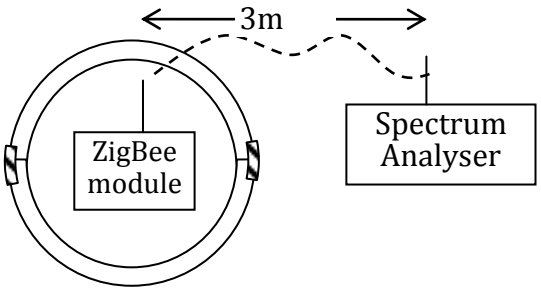

Figure 4.3: Aluminium Shield (Re-radiator) Test

Taking the PAN802154 module as an example, its output RF power is 0dBm. If a receiving module is placed at a 30 m distance, the estimated path loss is 30 dB [15]. Adding the 15 dB loss due to the aluminium shield, the received signal strength should be about -45 dBm, which is well above the module's sensitivity of -92 dBm. As a result, this amount of attenuation should not block the RF link, and the modules should be able to maintain proper communication at a range of more than 30 m. The XBee-PRO module should perform better, as it has a higher output power and better sensitivity.

B. Mild steel computer boxes

A second simulated environment was built by old computers. The computer boxes have air vents and openings, and they are not perfectly square. Twelve computers (cases with components fully installed) were arranged to form a "castle" (see photo and illustrations in Figure 4.4) such that the ZigBee and micro-controller modules were enclosed all around (top, left, right, front and rear) by two layers of computers on the concrete floor. A double layer structure was used to cover the openings, but there are small gaps (about 1 cm) between the computers.

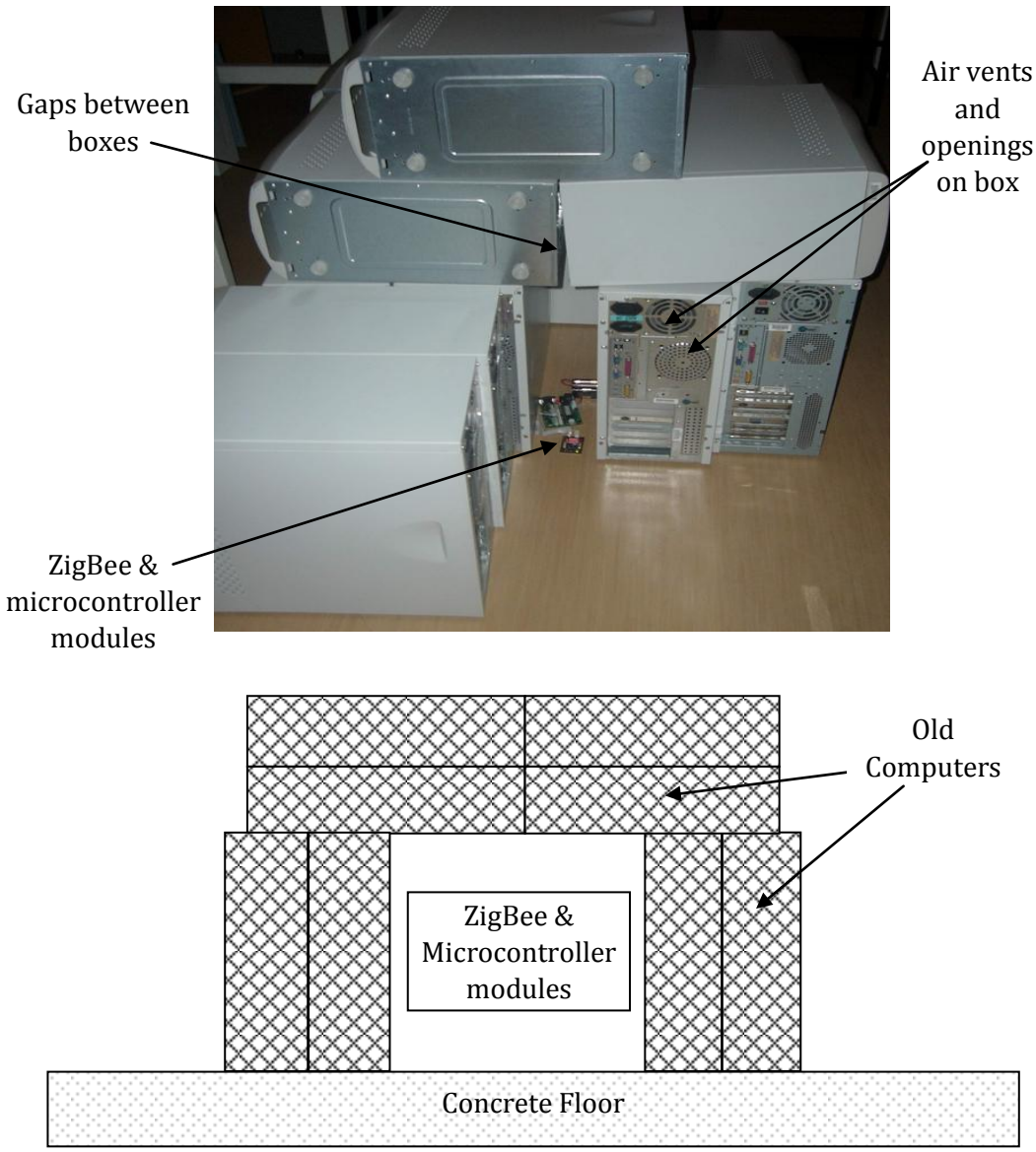

Figure 4.4: Mild steel computer boxes enclosing modules

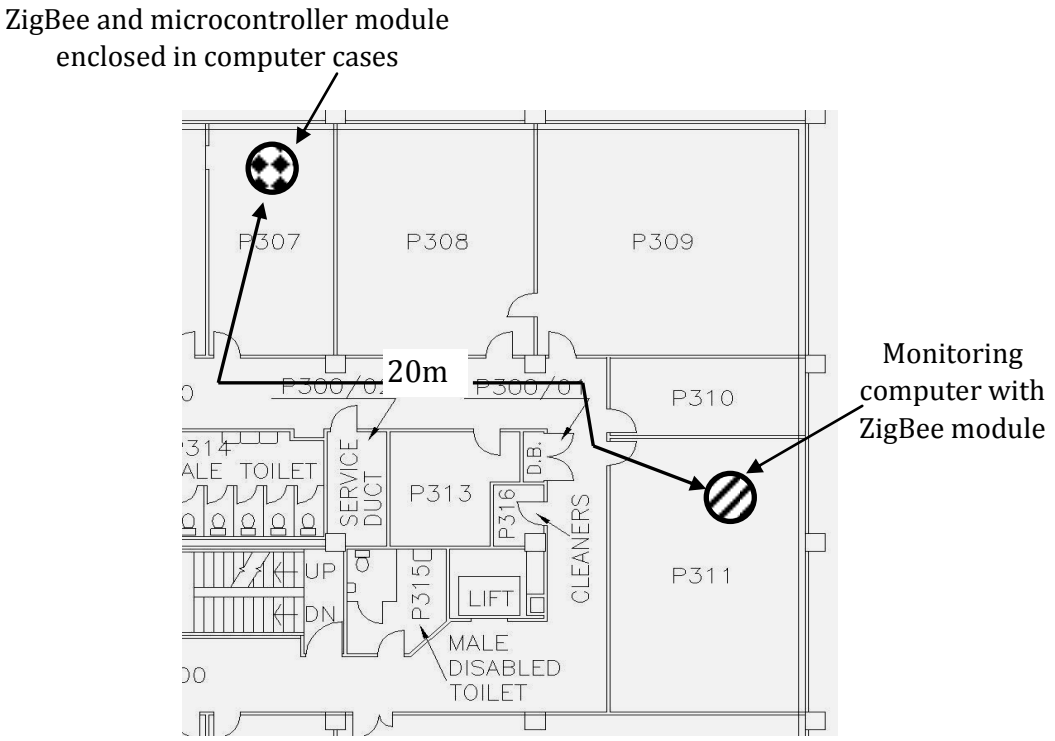

Figure 4.5: Test by enclosing modules in mild steel computer boxes

The monitoring computer with a ZigBee module was located in another room (as per the floor plan illustrated in Figure 4.5) at 20 m distance, and message passing was tested.

The test results showed that the communication is good with the XBee-PRO module, but the PAN802154 module gave intermittent results. When the monitoring computer is moved into the same room with the enclosed modules, both models of ZigBee module can establish proper communication. However, if more computer boxes were used to overlap all gaps, communication cannot be established.

4.2 Attenuation of RF Signal in Rubble

High frequency communication, such as the ZigBee technology in the range of MHz to GHz, is subject to significant attenuations in a rubble environment. The National Institute of Standards and Technology (NIST) have carried out experiments on RF

signals before, during, and after the implosion of three large building structures [23]. Their measurements showed a 20 to 80 dB of attenuation for RF signals in the frequency range 50 MHz to 1.8 GHz after the collapse, depending on the building type and location of the transmitter. This high attenuation is a major impediment to using these RF signals for direct point-to-point communication between devices inside the rubble and rescue persons outside.

The wireless link tests described above confirmed the same performance demonstrated by the XBee-PRO modules. A direct communication link is not always possible; and a routing wireless network is required to overcome the problem. Another often used wireless technology in buildings is Wi-Fi which also provides routing of data between devices. The next section compares the two technologies.

4.3 ZigBee versus Wi-Fi

Section 2.2 compared wireless technologies Wi-Fi and ZigBee. Both technologies support indoor and outdoor communication and data routing between device nodes. The Network-Centric Applied Research (NCAR) Team of Ryerson University in Canada has demonstrated communication range extension using Wi-Fi repeaters in an artificial rubble environment [24]. The NCAR team used the D-Link DWL-2100AP Wi-Fi access point for data routing [25]. This section investigates in more detail the suitability of the two technologies on the daughter robots by comparing the DWL-2100AP with the XBee-PRO.

4.3.1 Link Margins

The link margins for a ZigBee network and a Wi-Fi network can be estimated using the 80 dB attenuation measured by NIST as follows:

A. Link Margin for ZigBee device

A ZigBee device, XBee-PRO with a typical output power of +18 dBm and receiver sensitivity of -100 dBm will have a link margin of 38 dB.

B. Link Margin for Wi-Fi access point

A Wi-Fi access point, DWL-2100AP with a typical output power of +15dBm and receiver sensitivity of -89 dBm will have a link margin of 24 dB.

The link margin seems good for both ZigBee and Wi-Fi cases. However, the experiments done by NIST were up to 1.8 GHz but ZigBee and Wi-Fi is at a higher frequency of 2.4 GHz that will experience greater attenuation. This high level of attenuation coupled with multipath fading environmental noise indicates that a direct link for continuous data transfer will not be viable for both technologies. Thus multiple ZigBee devices or Wi-Fi access points are required to build an ad-hoc digital network which can route packets of data from one node to another until they arrive at the receiving end.

4.3.2 Cost, size and power

As described in section 2.4 the manufacturer of XBee-PRO provides firmware which allows configuration of the module as a coordinator, router or end-node. A cluster-tree wireless network can be formed by one coordinator and multiple routers and multiple end-nodes. The module can be programmed to work alone as a router and powered directly by two AA-size batteries to form a mini-router as shown in Figure 4.6 below. This makes a very low cost and small size routing node, such that it can be dropped into any opening or carried by a robot to anywhere in the rubble. Of course the size can be even further reduced if a more compact power supply is utilised.

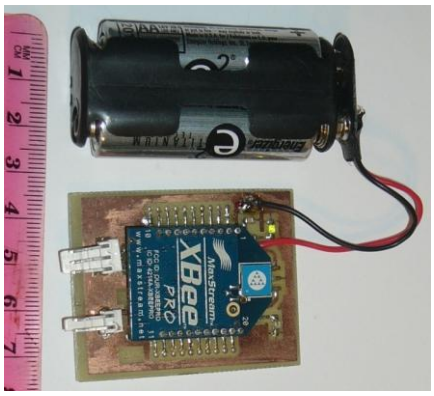

Figure 4.6: Mini-router - XBee-PRO with 2 AA-batteries

Compared with this mini-router, the access point is double the size, costs three times more and requires ten times more power. It is required for the group of robots to form an ad-hoc routing network. Due to the size and power requirements, it is not viable to have a robot carry a Wi-Fi access point and burrow into the rubble.

4.4 Summarizing the Wireless Link Test

The link test experiments showed that XBee-PRO performed significantly better than the PAN802154 module. It can establish a proper communication link when buried under commonly found materials in an office building. In most situations it can send messages between rooms, corridors and stairways; and in some cases it can get through walls and windows to outside the building. As compared with the Wi-Fi access point, the XBee-PRO module provides the same networking mechanism, but at a much smaller size and lower power consumption.

If we can assume that there is not much change on the material properties after a building is collapsed, and if there are still cracks (just like the cracks between computer boxes) between materials to allow wireless signals to get through, it is possible for the XBee-PRO modules to establish a usable communication link in the rubble by either a direct link or by passing a message from one robot to another until it gets outside the rubble. This supports the selection of this module for developing the daughter robots.

The next stage of the project is to implement the wireless network using the XBee-PRO modules, and then simulate a real rubble environment and carry out experiments to verify communication between robots inside and outside the rubble.

Chapter 5 Wireless Network Implementation

MaxStream produces several wireless modules that based on the ZigBee standard. XBee-PRO, because of its higher output power, is selected as one of the candidate modules for this project. After the analysis, prototyping and tests described in chapters 2, 3 and 4, it was chosen as the final module for network implementation. Several versions of firmware are available for the module. The version v8x17 is selected because it supports the cluster tree network topology as described in section 2.4. It also supports API (application programming interface) operation for controls by high-level host applications. This chapter describes the development of the automatic routing wireless network using the features provided by this firmware. High-level program is developed for communication test and analysis.

5.1 X-CTU software for configuring XBee-PRO modules

MaxStream provides the X-CTU software for configuring and programming firmware into the modules. Figure 5.1 illustrates the PC Settings tab of X-CTU [26].

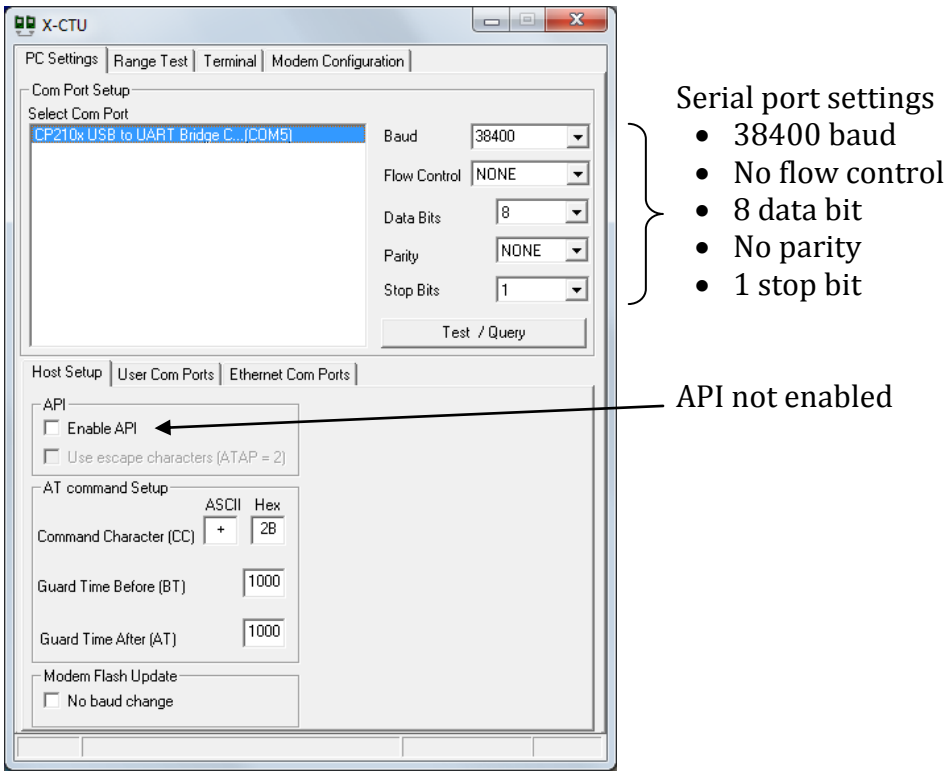

Figure 5.1: X-CTU port settings

5.2 XBee-PRO version v8x17 firmware

There are five sub-versions of the v8x17 firmware. The following describe the versions used for this project.

5.2.1 Coordinator Firmware - Version v8117

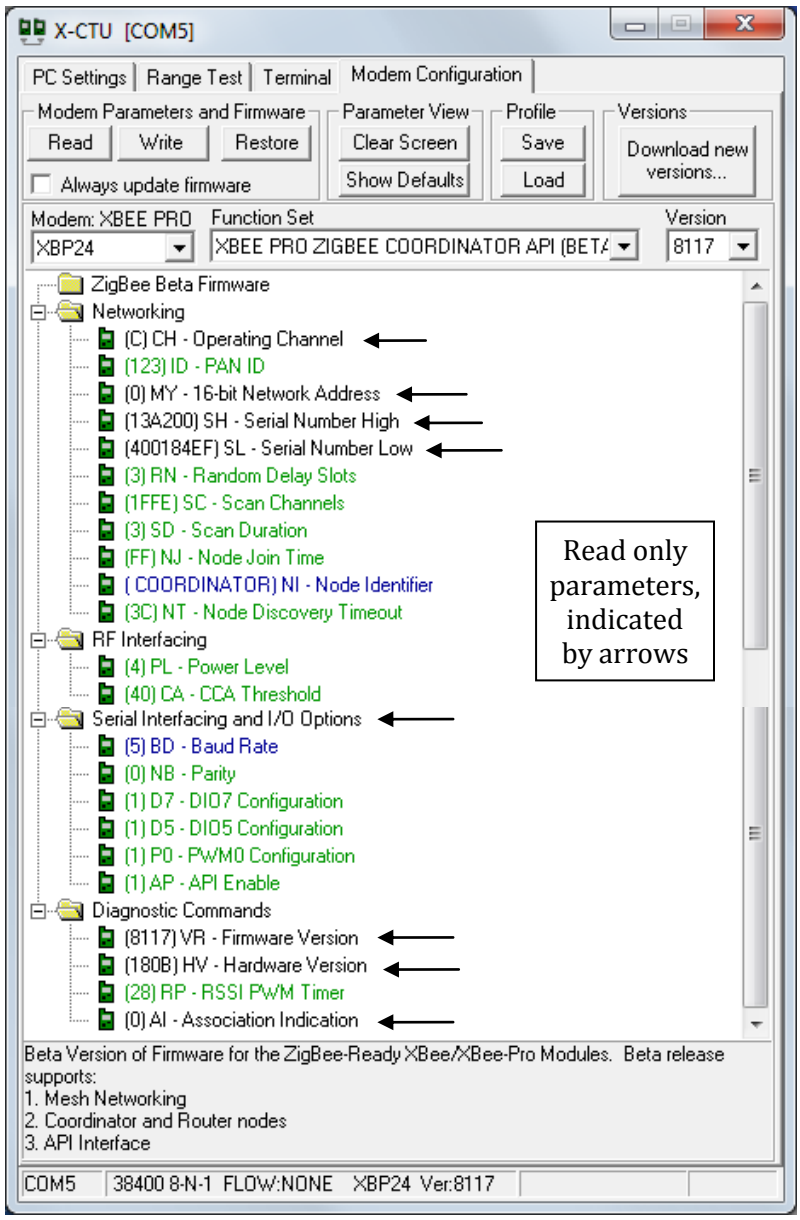

Figure 5.2: XBee-PRO coordinator firmware parameters

Figure 5.2 illustrates the Modem Configuration tab in X-CTU for the version v8117 firmware used on the XBee-PRO module to configure it as a coordinator on the network. The firmware is divided into four parts. Parameters in black colour are

read only; others are configurable. The following table describes usage of the parameters during coordinator operations.

Table 5.1: Firmware parameters for XBee-PRO coordinator operation

Networking
CH, ID, SC, SD & NJ – at power up a coordinator will issue an Active Scan for an unused channel and PAN ID. SC and SD determine the channels and time for scanning. If ID is set to 0xFFFF it will use a random PAN ID. Once a free channel and PAN ID is found, they will be written into the parameters, and the coordinator will allow nodes to join it for a time period based on the NJ parameter. If enabled, the Associate LED (connect to DIO5) will blink once per second.
MY – the network address for the coordinator is always 0.
SH & SL – Serial number is hard coded in the module.
RN – defines the back-off exponent in the CSMA-CA algorithm for collision avoidance. 0 to disable.
BH – defines the maximum number of hops for broadcast transmission. 0 will use the maximum number of hops.
NI – the name of the node in string format.
NT – defines the amount of time a node spends on discovering other nodes when a ND (node discover) command or a DN (destination node) command is received.
RF Interfacing
PL – defines transmitting power; five choices in dBm 10, 12, 14, 16, 18
CA – defines the CCA (Clear Channel Assessment) threshold level in dBm before transmitting a packet. If the detected level on the channel is above the CCA, the packet is not transmitted.
Serial Interfacing and I/O
BD – defines the baud rate; available choices from 1200 to 115200
NB – number of bits
RO – defines the number of inter-character silence that the module will wait, before packetizing data to be transmitted. If set to 0, data will be sent when they arrive without buffering.
D7 – configure the DIO7 pin of the module. 1 to use it as CTS flow control, 0 to disable.
D5 – configure the DIO5 pin of the module. 1 to use it as associated indicator to flash an LED (1 once per second at power up, twice per second when associated to a coordinator), 0 to disable.
P0 – 1 to enable RSSI (received signal strength indication) by PWM (pulse width modulation); 0 to disable.
AP – 1 to use API (Application Programming Interface) mode, 2 for API with escape character control. (For AT mode use firmware v8017)
Diagnostic Commands
VR & HV – read only firmware and hardware version of the module
RP – define the time (x100 ms) that the RSSI signal (see P0 above) will be output after last transmission; 0xFF to set output always on.
AI – stores information regarding last node join request.

5.2.2 Router Firmware - Version v8217

Figure 5.3 illustrates the configuration tab for the version v8217 firmware used on the XBee-PRO module to configure it as a router on the network.

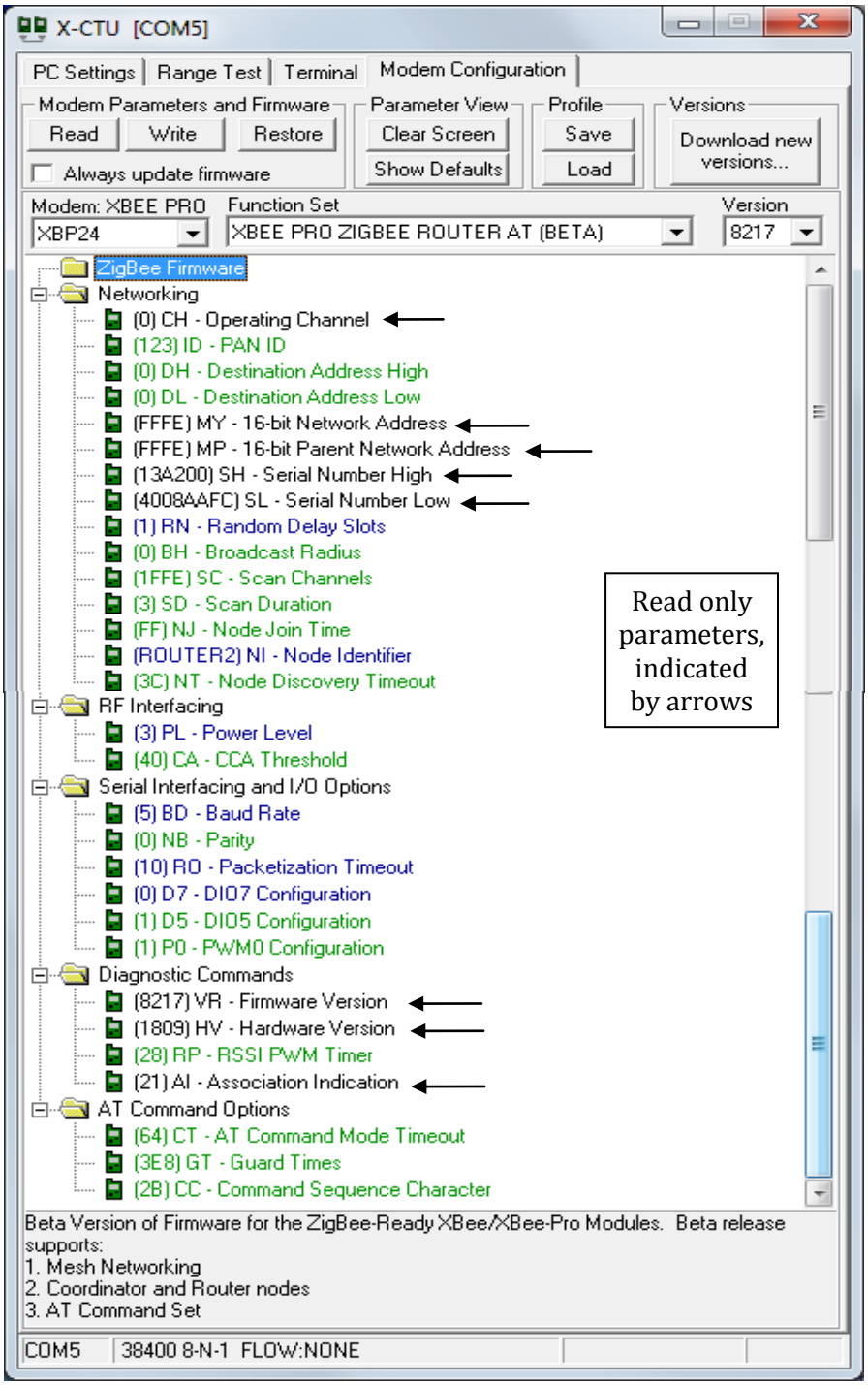

Figure 5.3: XBee-PRO router firmware parameters

The configuration parameters are divided into five parts. The following table describes the purpose of the parameters during router operations.

Table 5.2: Firmware parameters for XBee-PRO router operation

Networking CH, ID, SC, SD & NJ – at power up a router will scan for a coordinator or another router to allow it to join a PAN. SC and SD determine the channels and time for scanning. If ID is set to 0xFFFF it will join any available PAN. Once successfully joined a PAN the router will allow nodes to join it for a time period based on the NJ parameter. If enabled, the Associated LED (connected to DIO5) will blink twice a second to indicate successful connection to a PAN. CH will store the channel of the PAN. DH & DL – stores the 64-bit address of the destination node. This is set to zero for the router to send packets to the coordinator. Commands can be sent to the module to change this address. MY – stores the 16-bit network address is allocated by the coordinator. MP – stores the network address of the parent (the router or the coordinator that allow joining of the PAN). SH & SL – unique serial number hard coded in the module. RN – defines the back-off exponent in the CSMA-CA algorithm for collision avoidance. 0 to disable. BH – defines the maximum number of hops for broadcast transmission. 0 will use the maximum number of hops. NI – the name of the node in string format. NT – defines the amount of time a node spends on discovering other nodes when a ND (node discover) command or a DN (destination node) command is received.
RF Interfacing – same as for coordinator, see entries in Table 5.1
Serial Interfacing and I/O – same as for coordinator, see entries in Table 5.1
Diagnostic Commands – same as for coordinator, see entries in Table 5.1
AT Command Options This firmware provides operation of the module as a transparent wireless modem to the host (a microcontroller to simulate a robot in this project). Commands can be sent from the host to control the module (at AT Command mode) based on the following parameters. CT – stores the timeout (x100 ms) after which the module exits AT Command Mode and return to idle mode. GT – the Guard Times (x1 ms) is the silence period before and after the CC (Command Sequence Character) to prevent inadvertent entrance into AT Command Mode. CC – the character ('+' by default) to be used between the GT to set the module into AT Command Mode.

5.2.3 Profile files for modules to form a network

The X-CTU software allows saving and loading firmware parameters from profile files in text format. The following table displays the contents of profile files for the Coordinator, Router 1 and Router 3.

Table 5.3: Profile files of XBee-PRO modules

Coordinator.pro	Router1.pro	Router3.pro
XBP24_ZigBee_8117.mxi FE 0 241 8117 0 [A]ID=123 [A]RN=3 [A]SC=1FFE [A]SD=3 [A]NJ=FF [A]NI= COORDINATOR [A]NT=3C [A]PL=4 [A]CA=40 [A]BD=6 [A]NB=0 [A]D7=1 [A]D5=1 [A]P0=1 [A]AP=1 [A]RP=28	XBP24_ZigBee_8217.mxi FE 0 241 8217 0 [A]ID=123 [A]DH=0 [A]DL=0 [A]RN=3 [A]BH=0 [A]SC=1FFE [A]SD=3 [A]NJ=FF [A]NI=ROUTER1 [A]NT=3C [A]PL=4 [A]CA=40 [A]BD=6 [A]NB=0 [A]RO=10 [A]D7=0 [A]D5=1 [A]P0=1 [A]RP=28 [A]CT=64 [A]GT=3E8 [A]CC=2B	XBP24_ZigBee_8217.mxi FE 0 241 8217 0 [A]ID=123 [A]DH=0 [A]DL=0 [A]RN=3 [A]BH=0 [A]SC=1FFE [A]SD=3 [A]NJ=FF [A]NI=ROUTER3 [A]NT=3C [A]PL=0 [A]CA=40 [A]BD=6 [A]NB=0 [A]RO=3 [A]D7=0 [A]D5=1 [A]P0=1 [A]RP=28 [A]CT=64 [A]GT=3E8 [A]CC=2B

Four modules, a coordinator and three routers will be used to form the network for the experiments. Router 2 will have the same configuration as Router 1 but with NI changed to Router 2. Router 3 will be used as the data transmitter which programmed to the lowest transmitting power (PL=0, 10 dBm). This will maximise the effect of attenuation by the rubble and increase the chance of requiring message routing that will better test the performance of the network. In actual missions, all modules should be programmed to the maximum power.

5.2.4 AT mode versus API mode

Referring to the firmware parameters in Table 5.2 and Table 5.3, The XBee-PRO routers will work in AT mode (no AP parameter in the firmware) as transparent wireless modems. During experiments the simulated daughter robots will send data directly to the coordinator (address DH=0 and DL=0) through the serial port. The PAN ID will be fixed at 123 and no change is required on the router modules at real-time.

The coordinator is set to API mode to allow high-level programs to control the module at real-time operation. API operation facilitates a frame structure communication between the host and the module. Network information can be extracted from the coordinator by sending commands to it through API frames. To simply the design, it was chosen to use API mode without escaped characters. The simulated daughter robots and the simulated mother robot (the monitoring computer) will send messages in ASCII characters to avoid conflict with other bytes in the structured frames.

5.3 API Programming on the Simulated Mother Robot

The coordinator will be attached to a computer simulates the mother robot. A monitoring and communication analysis program is written to implement the API frame structure required for communicating with the coordinator. Borland Delphi (version 5), because of its well structured base language (Pascal) is chosen for developing the program. Several API frame structures from the XBee-PRO firmware are chosen for developing the program. The following summarises the chosen API frame structures from the firmware manual.

5.3.1 AT Commands in API frame structure

Two AT commands are used, NI to check the node identifier to make sure the coordinator is working and ND for node discovery to find all nodes connected to the coordinator. The frame structure for sending AT commands is shown in Figure 5.4.

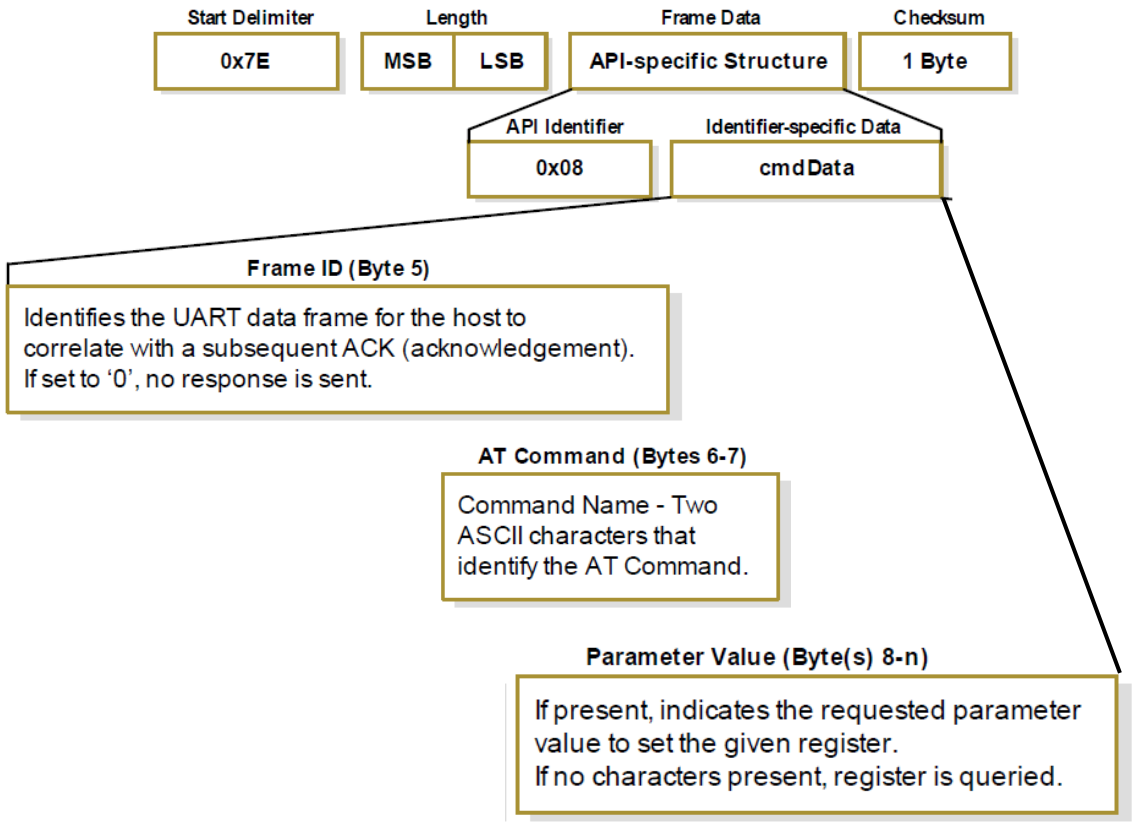

Figure 5.4: AT Command frame structure

The high-level structure is the same for all API frames: Start Delimiter + Length + Frame Data + Checksum. For example, to send the NI command, the following sequence of bytes is required to form the API frame.

Table 5.4: Byte sequence for NI command in API frame structure

Bytes	Values	Description
1	0x7E	All API frames use this start delimiter
2-3	0x00 0x04	Length = API Identifier + Frame Id + AT Command
4	0x08	API Identifier 0x08 for AT command
5	0x52	Frame ID an non-zero value, arbitrary chosen as 'R'
6-7	0x4E 0x49	AT Command in ASCII code for 'N' and 'I'
8	0x0E	Checksum for bytes 1 to 7

A similar byte sequence can be constructed for the ND command by replacing byte 7 with 0x64 (ASCII code for 'D') and byte 8 with the corresponding checksum.

5.3.2 API frame for transmit request

The ZigBee Transmit Request frame is used for sending data to devices on the network. The high-level structure is the same, Figure 5.4, but it has 0x10 as the API Identifier and a different block for Identifier-specific Data. The detail of the Identifier-specific Data block is shown in the table below.

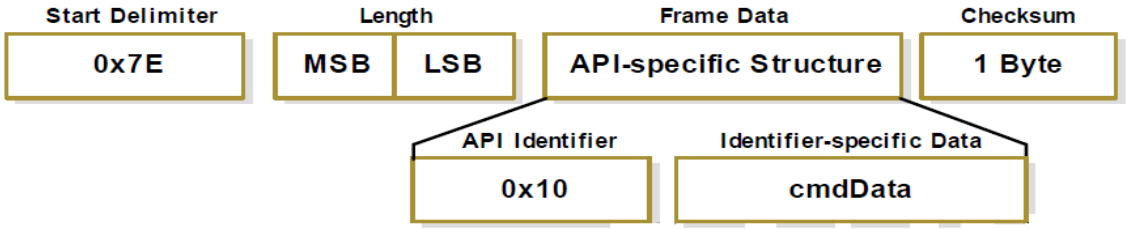

Figure 5.5: API frame for ZigBee Transmit Request

Table 5.5: Identifier-specific Data block for ZigBee Transmit Request

Bytes	Description
5	Identifies the data frame of the host to correlate with a subsequent ACK, using 0 will disable response frame
6 - 13	64-bit Destination Address (Broadcast = 0x000000000000FFFF)
14 - 15	16-bit Destination Network Address (0xFFFE for Broadcast, or when network address is unknown)
16	Set maximum hops for broadcast, 0 for using maximum network hops value of 10
17	0x01 = Disable ACK, 0x02 – Disable Network Address Discovery AT Command in ASCII code for 'N' and 'I'
18 - n	Data to send, up to 72 bytes per packet

5.3.3 API frame in respond to AT Command

In response to an AT Command, a module will send a Frame Data with 0x88 as the API Identifier and the Identifier-specific Data block illustrated in Table 5.6.

Table 5.6: Identifier-specific Data in respond to AT Command

Bytes	Description
5	Identifies the UART data frame being reported, 0 to indicate in AT Command mode and no response will be given.
6 - 7	ASCII characters of the command responded to
8	Status byte, 0=OK, 1=ERROR
9 - n	HEX value of the requested register

5.3.4 API frame of ZigBee Received Data Packet

When an API enabled module receives a data packet (not a command packet) from the RF link, it will send to the host an API frame with 0x90 as the API Identifier. Table 5.7 illustrates the Identifier-specific Data block within the API frame for the data packet. The actual message starts at byte 16 to the end of the block are the received data.

Table 5.7: Identifier-specific Data block of ZigBee Received Data Packet

Bytes	Description
5 - 12	64-bit Address of the sender, MSB first, LSB last
13 - 14	16-bit Address of the sender, MSB first, LSB last
15	Options – 0=Packet Acknowledged, 1=Broadcast Address, bits 2-7 [reserved by manufacturer]
16 - n	Data received, up to 72 bytes per packet

5.3.5 ZigBee Tester program on monitoring computer

The following figure illustrates the user interface of the tester program on the monitoring computer.

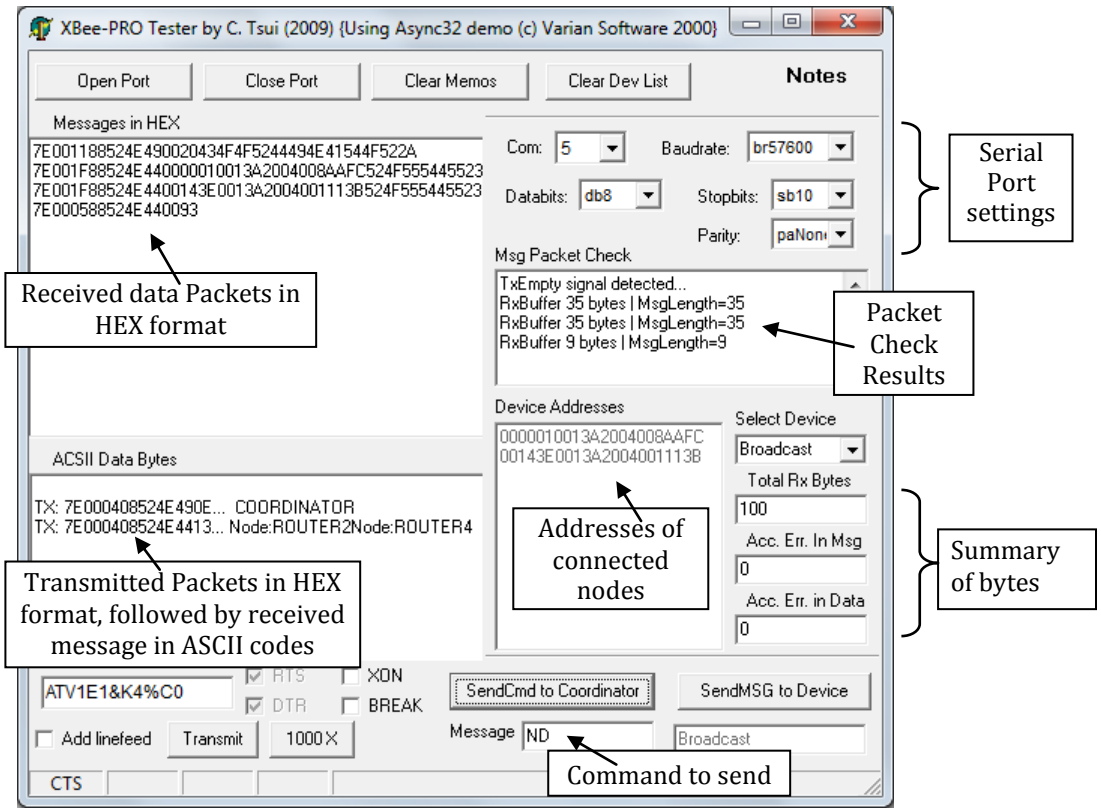

Figure 5.6: ZigBee tester and analysis program

The tester program is based on the Asyn32 Serial Port demo program by TMS Software [27]. It comes with a VaComm component which provides procedures for controlling and handling serial port events. Standard Borland Delphi components were used. Several functions were written to build the user interface (Figure 5.6) and to implement required communication tests. They are described below.

Functions for constructing AT Commands:

Line	Pascal code
1	`function findChkSum(cmd: String): char;`
2	`var`
3	`  I: Integer;`
4	`  Sum : Integer;`
5	`begin`
6	`  I := 1; Sum := 0;`
7	`  while I <= Length(cmd) do`
8	`  begin`
9	`    Sum := Sum + Integer(cmd[I]);`
10	`    I := I+1;`
11	`  end;`
12	`  I := 255 - Sum;`
13	`  Result := char(I);`
14	`end;`

Figure 5.7: Pascal function for finding Checksum

Line	Pascal code
1	`function getCommand(cmd: String): String;`
2	`var`
3	`  cmdMid: String;`
4	`begin`
5	`  cmdMid :='';`
6	`  if (Length(cmd)>$FF) then //for long commands`
7	`    cmdMid := char(Length(cmd)-$FF) + char(Length(cmd))`
8	`  else  //for short commands, e.g. ND & NI`
9	`    cmdMid := char(00) + char(Length(cmd));`
10	`  Result := APIdelimiter + cmdMid + cmd + findChkSum(cmd);`
11	`end;`

NOTE APIdelimiter: char = char($7E); //const defined elsewhere
Call findChkSum (Table 5.8) to get checksum of the command.

Figure 5.8: Pascal function for constructing the AT Command

Button for sending AT Command to coordinator:

When the "SendCmd to Coordinator" button was clicked, the procedure in Figure 5.10 will be triggered to call the above functions to construct the AT Command using the string in "Message" text box.

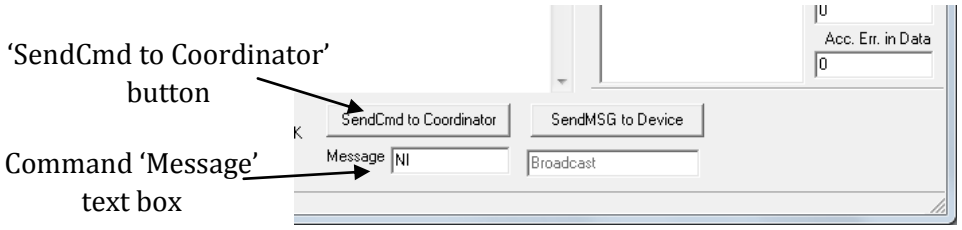

Figure 5.9: Button for sending AT command

Line	Pascal code
1	`procedure TfrmMain.btnSendCmdClick(Sender: TObject);`
2	`var`
3	`  S, Hs: String;`
4	`  I: Integer;`
5	`begin`
6	`  S := getCommand(char($08) + 'R' + edtCmd.Text);`
7	`  Vacomm1.writeText(S);`
8	`  I := 1;`
9	`  Hs:='';`
10	`  while I <= Length(S) do`
11	`    begin`
12	`       Hs := Hs+ IntToHex(Integer(S[I]),2);`
13	`       I := I + 1;`
14	`    end;`
15	`  Memo2.Lines.Text :=`
16	`      Memo2.Lines.Text+ char($0D)  +'TX: ' + Hs +'... ';`
17	`  if (UpperCase(edtCmd.Text) = 'ND') then`
18	`  begin`
19	`    Memo4.Lines.Clear;`
20	`    ComboBox1.Items.Clear;`
21	`    ComboBox1.Items.Add('Broadcast');`
22	`  end;`
23	`end;`

Figure 5.10: Pascal function for sending command to coordinator

Line 6 calls the getCommand function (Figure 5.8) to create the command string from the message box, edtCmd.Text. Line 7 calls the serial port component to send the command. Lines 8 to 16 update the display with the command information. Lines 17 to 21 check if a network discovery 'ND' command is sent, if so, the list of device addresses Memo4 will be cleared to wait for new list of devices. ComoBox1

for selection of devices is also cleared and set to 'Broadcast' as the default. This is to prepare the interface for sending messages to a device.

ZigBee message received:

When a message is received by the ZigBee module and passed to the serial port, the OnData event from the VaComm1 component will trigger the VaComm1Data procedure. A full listing of the program source code is provided in the Appendix. The following pseudo code illustrates the operation of the procedure, where RxMsg stores all characters of the received message.

Pseudo code for handing data received:

1. Read character to C, until no more
   If C='s', it is start of the 300 bytes, CountOneMsg=0, StartOneMessage=true
   If StartOneMessage
   If C is a number between '0' to '9', inc(CountOneMsg)
   If C='E', end of message
   StartOneMsg=false
   If CountOneMsg <>300
   Calculate and display number of errors
   If C is a linefeed
   Update accumulated error bytes
   Reset error bytes count
   If C equals 7E this indicates start of a new message
   If length(RxMsg)>0 indicates a second message arrived
   Call gotMessage(true) to handle second message
   Buffer C into RxMsg, and hex value of C into Hs
2. Display received hex characters, Hs on Memo3
3. If length(RxMsg)>0 and first character of RxMsg is 7E
   It indicates a single completed message
   Call gotMessage(false) to handle the message

Pseudo code for gotMessage(twoMessage):

1. Call getCompleteMsg to check any error in RxMsg
2. If no error – indicates a completed message in RxMsg
   - Update Memo1 & Memo2 to display information
   - If the RxMsg is a response to a Node Discovery command
     - Update Memo4 with received device address
     - Update ComboBox1 to display received device name
   - Clear RxMsg
3. Else (errors)
   - Update Memo1 to display errors
   - Update error bytes on display
   - If twoMessage=true, indicates first message is incomplete
     - Discard the first message by clearing RxMsg

Two functions, getCompleteMsg and chkATreponse, were written to support the data handling procedure. Figure 5.11 listed the getCompleteMsg function.

Line	Pascal code
1	`function getCompleteMsg(var Msg: String ): Integer;`
2	`var`
3	`  L: Integer;`
4	`begin`
5	`  L := Integer(Msg[2])*256 + Integer(Msg[3]);`
6	`  if (Length(Msg) = (L+4)) then`
7	`        //completely received one message`
8	`  begin`
9	`    if (Msg[4] = char($90)) then   //it is a ZigBee Message`
10	`       Msg := Copy(Msg,16,Length(Msg)-16)`
11	`                    //Extract the content`
12	`else if (Msg[4] = char($88)) then`
13	`                 //it is an AT command response`
14	`       Msg := chkATresponse(Msg)  // handle AT response`
15	`    else`
16	`       Msg :='Not a proper message!';`
17	`    Result := 0;`
18	`  end`
19	`  else  //keep RxMsg, but err bytes returned`
20	`  begin`
21	`     Result := Length(Msg)- (L+4)`
22	`  end;`
23	`end;`

Figure 5.11: Pascal function to check a complete message

The getCompleteMsg function checks if the length of the data packet is correct; if not, it will discard the message. Then if the response is a ZigBee data packet indicated by $90 in byte 4 (section 5.3.4), extract the data. If the response is for AT Command indicated by $88 (section 5.3.3), call chkATresponse to handle the message.

The chkATresponse function, listed in Figure 5.12, takes bytes 6 and 7 and checks the corresponding ASCII code. If it is 'NI', the response is the identifier of the coordinator; line 6 and 7 will extract the identifier. If it is 'ND', the response contains the device address and name. Lines 12 to 20 extract and store the address in the variable Raddress and return the node name.

Line	Pascal code
1	`function chkATresponse(Msg: String): String;`
2	`var`
3	`  I: Integer;`
4	`  S: String;`
5	`begin`
6	`  if (Copy(Msg,6,2) = 'NI') then`
7	`    Result := Copy(Msg,9,Length(Msg)-9)`
8	`  else if (Copy(Msg,6,2) = 'ND') then`
9	`          // each node will response with one message`
10	`    begin  // extract node name`
11	`       I:= 7;`
12	`       if (Length(Msg) > 19) then`
13	`       begin`
14	`          repeat`
15	`             I := I+1;`
16	`             S := S+IntToHex(Integer(Msg[I]),2);`
17	`          until (I=18);`
18	`          Raddress:= S;`
19	`          Rname := Copy(Msg,19,Length(Msg)-9);`
20	`          Result := 'Node:' + Rname  +':';`
21	`       end`
22	`       else`
23	`          Result := '';`
24	`    end`
25	`  else`
26	`    Result := 'Not identified';`
27	`end;`

Figure 5.12: Pascal function to handle AT Command response

Button to send message to devices:

The tester program allows sending messages to ZigBee devices on the network. In Figure 5.13, the 'Select Device' (ComboBox1) is showing 'Broadcast' and 'ND' is typed into the Message text box. If the 'SendMSG to Device' button is clicked, a Broadcast command of 'ND' for node discovery will be send to all devices connected on the network.

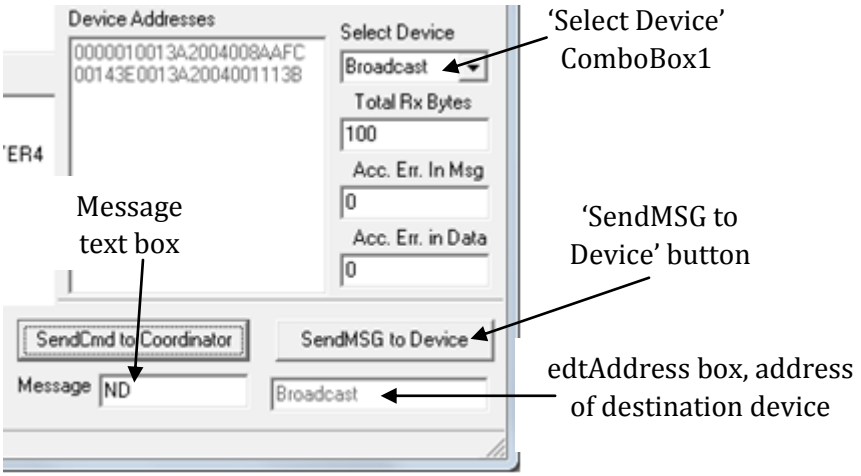

Figure 5.13: Send message to devices

If a user wants to send a message to one of the devices, the user has to choose a device by clicking on the 'Select Device' (ComboBox1) its OnClick event will trigger the ComboBox1Click procedure to put the select device address ComboBox1 onto the text box, edtAddress below the 'SendMSG to Device button'.

When the 'SendMSG to Device button' is clicked, the procedure btnSendMSGClick (Figure 5.14) will be triggered to send the message. Lines 10 to 21 call the getCommand function to construct the command for broadcast, if no device is selected. If a device is selected, using its name in the edtAddress box the getAddress function will provide the hard coded address of the device. Line 35 then calls the VaComm1 component to send the command. The rest of the procedure updates the display of information.

Line	Pascal code
1	`procedure TfrmMain.btnSendMSGClick(Sender: TObject);`
2	`var`
3	`  S, Hs, Addr: String;`
4	`  I: Integer;`
5	`begin`
6	
7	`  Addr := Memo4.Lines[0];`
8	`  if (edtAddress.Text='Broadcast') then`
9	`  begin`
10	`    S := getCommand(`
11	`       char($10)                //ZigBee Transmit Request`
12	`       + char($00)              //no ack`
13	`       + char($00)+ char($00)   //64 bit destination address`
14	`       + char($00)+ char($00)   //   0x0000 0000 0000 FFFF`
15	`       + char($00)+ char($00)   //   for broadcast`
16	`       + char($FF)+ char($FF)`
17	`       + char($FF)+ char($FE)   //16 bit Network address`
18	`                                //  0xFFFE broadcast/unknown`
19	`       + char($00)+ char($00)   // Max Hop + No options`
20	`       + edtCmd.Text            // the message`
21	`       );`
22	`  end`
23	`  else`
24	`  begin`
25	`     S := getCommand(`
26	`       char($10)                //ZigBee Transmit Request`
27	`       + char($00)              // no ack`
28	`       + getAddress(edtAddress.Text) //  edtAddress.Text`
29	`       + char($FF)+ char($FE)   // 16 bit Network address`
30	`                                //  0xFFFE Broadcast/Unknown`
31	`       + char($00)+ char($00)   // Max Hop + No options`
32	`       + edtCmd.Text            // the message`
33	`       );`
34	`   end;`
35	`  VaComm1.writeText(S);`
36	`  I := 1;`
37	`  Hs:='';`
38	`  while I <= Length(S) do`
39	`    begin`
40	`       Hs := Hs+ IntToHex(Integer(S[I]),2);`
41	`       I := I + 1;`
42	`    end;`
43	`  Memo3.Lines.Text := Memo3.Lines.Text`
44	`                + char($0D)+'TX: ' + Hs +'... ';`
45	`end;`

Figure 5.14: Pascal procedure to send message to devices

Port control and other buttons

The 'Open Port' and 'Close Port' buttons call the corresponding procedures of the VaComm1 component to control the port. A procedure was written to clear all display text and reset variables. The 'Clear Dev List' button calls a procedure to clear ComboBox1, reset its index and reset the address box to 'Broadcast'. The buttons 'Transmit' and '1000x' came with the demo program and were not used for the tests.

A full listing of this tester program is provided in the Appendix section.

5.3.6 Program in data transmitter and monitoring computer

The ZigBee tester program described in section 5.3.5 was used on the laptop to record data received by the coordinator. The program displays raw data packets in hex format for visual inspection. It also analyses data packets and displays statistics of error bytes.

The data transmitter is programmed to continuously send blocks of 500-byte data with 100 ms breaks between blocks. Within each of the 500-byte data block, an interleave pause of 20 ms is placed between 72 bytes of data. Due to the packetization mechanism in the XBee-PRO firmware, this pause will trigger the XBee-Pro module to send a packet and allow enough time for the ZigBee stack to process the packet, transfer it to the wireless link and allow the coordinator to send the data packet as an API frame to the monitoring program. Also, the pause includes the time for the monitoring program to process the packet and display information.

The main loop of the source code of the data transmitter program is shown in Figure 5.15. The program will first check whether the external switch is pressed, if so, it sets up the program to run 1000 loops (longRun=1000). Otherwise, it will run once only. Within the longRun loops, it will firstly transmit the name of the transmitter ('RT3-') by lines 12 to 16. The 20 ms pause will trigger the module to send the name in a packet as the starting strings before the 500 bytes data.

The for-loop (line 18) controlled by FOR1, repeats the blocks five times. The 500 bytes is broken down to 50 times of 10 numbers (0 to 9) in ASCII code. The loop controlled by FOR10 will send each numbers (line 25) 10 times. Then the loop controlled by FOR0, repeats the number loop 50 times.

Line	Data Transmitter main while-loop C code
1	`while ( 1 )`
2	`{  SetCentiS(50); //wait 500ms for external switch`
3	`   pSwitch = GetTouch(_TOUCH_Sw_);  //read switch status`
4	`   if (pSwitch > 0 )`
5	`      longRun = 1000;`
6	`   else`
7	`      longRun = 1;`
8	`  while ( longRun != 0 ) // longRun set by external switch`
9	`  {                      // to 1 or 1000`
10	`   for (int FOR3_= 0; _FOR3_< longRun; FOR3++ )`
11	`   {             //repeat many times`
12	`     uartsend(7,13);  //call the uartsend subroutine`
13	`     uartsend(7,'R'); //    to send the characters`
14	`     uartsend(7,'T');`
15	`     uartsend(7,'3');`
16	`     uartsend(7,'-');`
17	`     SetCentiS(2);  //pause 20ms to send start chars`
18	`     for (int FOR1 = 0; FOR1 < j; FOR1++ )`
19	`     {             //repeat 5 times`
20	`        uartsend(7,'s'); k = k+1;`
21	`        for (int FOR0 = 0; FOR0_< bCount/10; FOR0++ )`
22	`        {          //round count 50 times`
23	`           for (int FOR10 = 0; FOR10_<= 9; _FOR10++ )`
24	`           {       // 10 bytes each round`
25	`               uartsend(7,i+48);`
26	`               k = k + 1;`
27	`               if (k > 71) { SetCentiS(2); k =0;}`
28	`                   //pause 20ms to send packet of 72 bytes`
29	`            }`
30	`           i=i+1;`
31	`           if (i>9) i=0;`
32	`         }`
33	`        uartsend(7,'E');`
34	`        uartsend(7,_FOR1_+97); //display a, b, c, d, e`
35	`        uartsend(7,'!');`
36	`        SetCentiS(2);  //pause 20ms to send end chars`
37	`        SetLCD3Char(1, _FOR1); // display counts on LCD`
38	`      }`
39	`     SetCentiS(pauseT);  //pause 100 ms after 500bytes block`
40	`    }`
41	`   longRun = 0;   //long run completed, stop it`
42	`   SetLCD3Char(9, 0);`
43	`   SetLCD3Char(13, 0); //display 0 0 on LCD for loop end`
44	`  }`
45	`}`

Figure 5.15: Main loop of data transmitter program

Lines 26 and 27 will pause the program for 20 ms after 72 bytes are sent. This pause triggers the ZigBee module to send a packet. Lines 30 and 31 control the

increment of the number from 0 to 9. Lines 33 to 37 send the ending string and display counts on the transmitter's LCD screen. Line 39 pauses to allow enough time for the tester program on the computer to process all the 500 bytes and display information on screen.

Several subroutines (SetCentiS to pause, GetTouch to read switch, uartsend to send a byte, and SetLCD3char to display a character on the LCD) used in the main loop are provided by the RoboExp development software.

5.4 Summarising Network Implementation

The data transmitter program was debugged, complied and programmed into a microcontroller taken from a RoboExp robot, and then attached with the XBee-PRO module that has its firmware programmed as Router3. The tester program was debugged, complied and test run on the monitoring computer with the coordinator attached to the USB port.

Two XBee-PRO modules were programmed with the Router1 and Router 2 firmware. All the devices and the computer were set up and test run in open space, to make sure the programs are executed properly with all the date bytes received, before deploying them into the rubble.

Chapter 6 Experiments in Artificial Rubble

This chapter describes the experiments to verify the functioning of the wireless network implemented by the firmware and software described in Chapter 5. The wireless network was formed using XBee-PRO modules and each module was attached to an ATMega16L microcontroller from the RoboExp robot in order to model a daughter robot.

Proof of concept experiments were carried out by deploying the networked robots in artificial rubble. The rubble was simulated by connecting holes and trenches that were dug in 50 cm deep soil. The simulated robots were placed in the bottom of the holes. The holes and trenches were then covered up by various building materials and soil to simulate a real rubble environment. Experiments were carried out to verify that a monitoring computer placed 10 metres outside the rubble can establish proper communication with all robots inside the artificial rubble environment.

6.1 Prototype models for experiments

A data transmitter was made by attaching an XBee-PRO module to the serial port of an ATMega16L microcontroller (Figure 6.1). The micro-controller was programmed to continuously send out blocks of 500-byte data with 100 ms breaks between blocks.

Figure 6.1: Data Transmitter

Figure 6.2: XBee-PRO router with 4 AA-batteries

Two data routers (Figure 6.2) were made by XBee-PRO modules with a 3 V regulator adaptor board, powered by four AA-size batteries to allow prolonged tests.

An XBee-PRO module on a USB adaptor board was configured as a coordinator (Figure 6.3) and attached to the monitoring laptop computer.

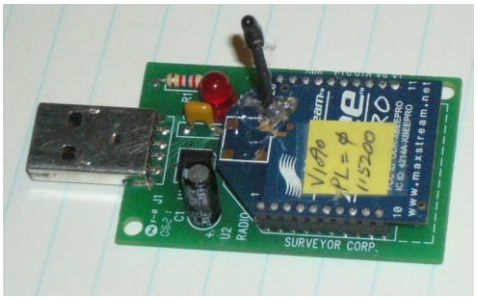

Figure 6.3: XBee-PRO coordinator on USB adaptor

6.2 Soil environment at rubble site

The test setup was arranged in the backyard of a residence in the eastern suburb of Auckland, New Zealand. The geology of the residential area is “Alternating SANDSTONE and MUDSTONE of the Waitemata Group” [28]. It has a bottom layer of about 10 m deep of clay with a layer of silt on top. The backyard was further filled up with a layer of organic soil to make an even surface, on which tough lawn was grown.

Soil can act as a lossy wave guide when its moisture level is above 25% [29]. That is the reason for choosing ground soil as the base foundation to build the artificial environment for the experiments. The bottom layer of clay forms a good layer for retaining water in the top soil layer.

The soil around the artificial rubble will be kept moisturised by plenty of water. Measurements will be taken during the experiments to verify that the RF signal will propagate through under soil trench, space, cracks and openings; instead of directly going above ground.

6.3 Measurement of materials for simulated rubble

In order to establish an accurate understanding of the effects on the 2.4 GHz RF signals introduced by various materials in the soil environment, a series of experiments were carried out.

6.3.1 Measurement equipment

Spectrum Analyzer, Wi-Spy 2.4x device

The Wi-Spy 2.4x device from MetaGeek LLC is a low cost (NZ$470 as of August 2008) and portable spectrum analyzer which attaches to a USB port of a computer [30]. With its Chanalyzer software, RF signal spectrums in the designed range can be captured in real-time and recorded for off-line analysis.

Table 6.1: Wi-Spy 2.4x Technical Specifications

Bandwidth:	2400 to 2495 MHz
Frequency Resolution:	328 KHz
Antenna:	External RP-SMA
Amplitude Range:	-110 dBm to -6.5 dBm
Amplitude Resolution:	0.5 dBm
Sweep Time:	165 millisecond

The above table shows the technical specifications of the Wi-Spy 2.4x device which covers the XBee-PRO's sensitivity of -100 dBm with reasonable frequency and amplitude resolution.

The device has a long sweep time of 165 ms which may impose a sampling problem. The data transmitter is sending packets of data interleaved with 20 ms pauses. Each data packet will appear in the RF link as a burst of less than a millisecond followed by 20 ms of no signal. The measuring device must take the sample from the RF link at the burst instant. For every 4 sweeps (taking a total of 660 ms) of the measuring device, there will be a round number of 33 packets transmitted (if the burst time is neglected). That would allow a sample to be picked up by the measuring device. From this calculation, 90 samples will be recorded in 59.4 seconds.

In order to display a spectrum with reasonable number of samples on the recorded graph, it is expected to take records for minutes for each measurement. The transmitter will be continuously sending data packets while the analyzer software is picking up samples.

Special Tripod for multipath signals

A tripod with a wooden support was built to hold the Wi-Spy 2.4x device and antenna to facilitate measurement of received signal strength (Figure 6.4).

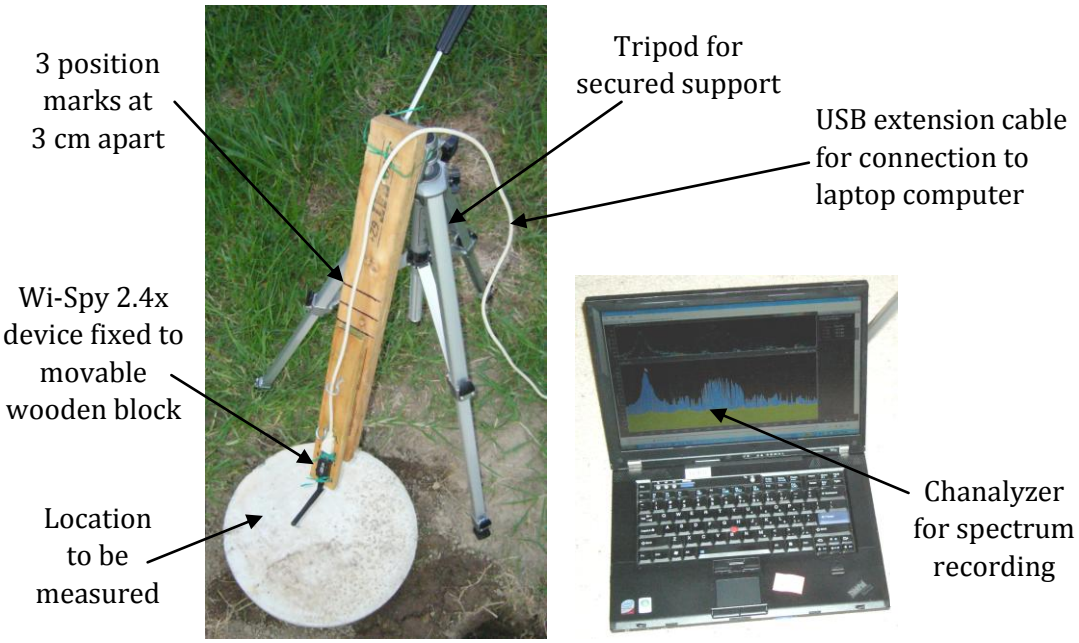

Figure 6.4: Wi-Spy 2.4x device and spectrum analyser on laptop

To get reliable results and to even out the effect of multipath signals, the measuring device was fixed onto the wooden support which allows three position settings. Each position is separated by 3 cm (about one quarter wavelength). The wooden support was then fixed onto a tripod, which can be positioned securely to any location for measurement.

RF signal spectrum recording on laptop

The Wi-Spy 2.4x device was connected to a laptop computer (Figure 6.4) with spectrum analyser software that will be used for measuring the received RF signal power at various locations in the artificial rubble. It was positioned at the centre and 20 cm on top of the measuring spot. For each set up, three readings were taken by shifting the measuring device to the three position settings on the wooden support. The average of the three readings was recorded as the final measured result for that setting.

6.3.2 Measurement setup

A 50 cm diameter by 50 cm deep hole was dug in the soil, at a location such that there was no underground piping or cable, or any other structure within 2 metres, except soil with lawn on top. Figure 6.5 illustrates the measurement setup.

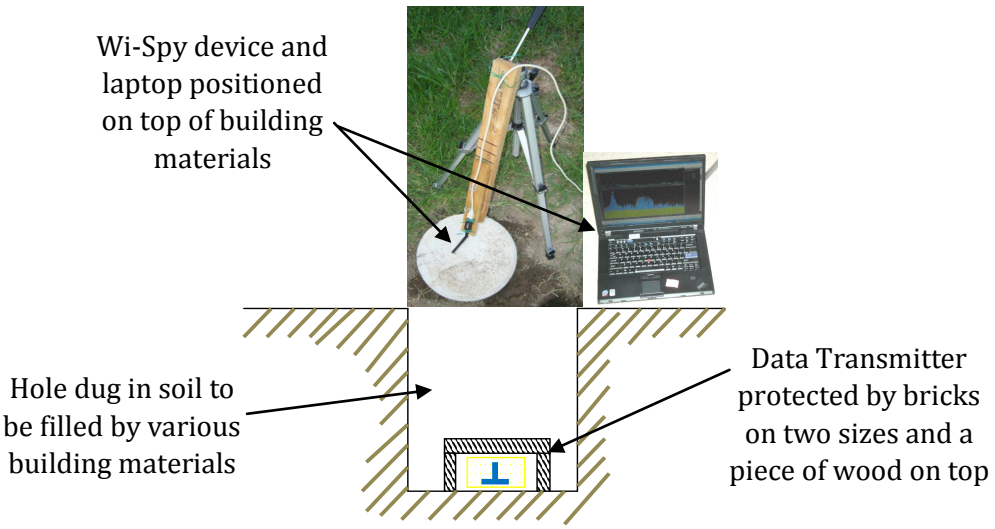

Figure 6.5: Measurement of material attenuations

The data transmitter (Figure 6.1) was placed in the bottom of the hole and then the hole was filled up with various building materials; any cracks and openings were filled by the soil that was dug out from the hole. Attenuation was found by comparing measured results before and after the hole was filled by various building materials.

6.3.3 RF signal background measurement

The first measurement to be taken is the RF signals at the background of the selected site. After the hole was dug, the measurement device was placed at the centre above the hole while it is empty. The following background spectrum (Figure 6.6) was recorded.

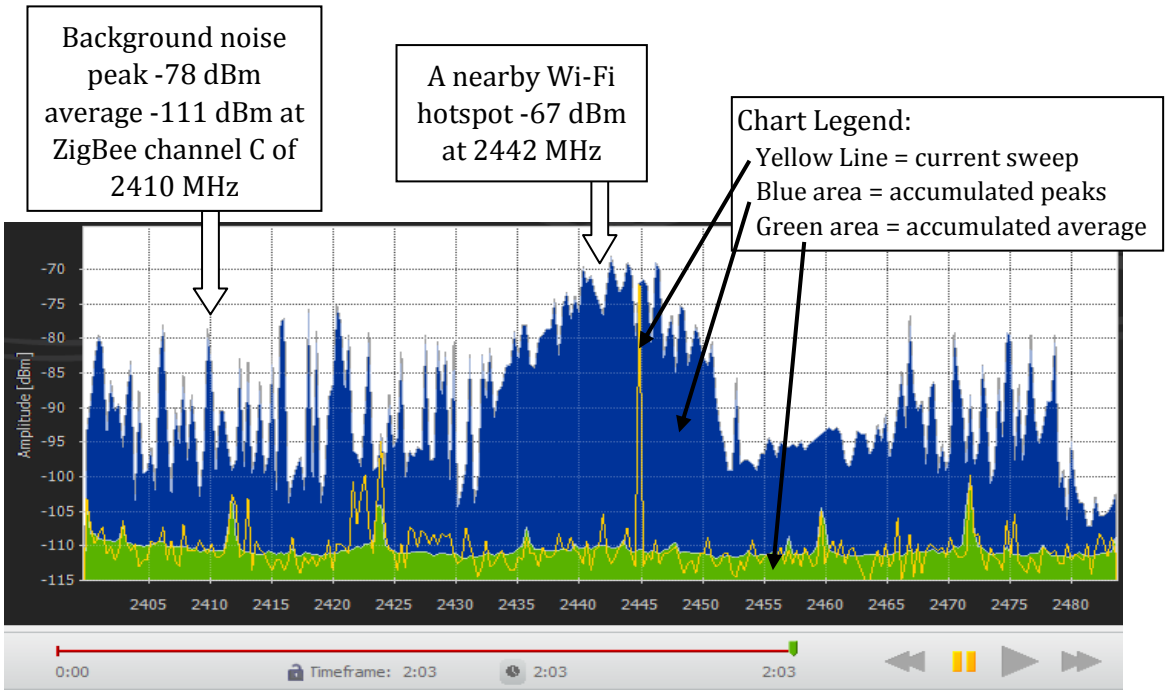

Figure 6.6: RF signal background spectrum

The above spectrum is a record of 2 minutes and 3 seconds. It shows a Wi-Fi hotspot at the commonly used channel 7 of 2442 MHz. The Wi-Fi signal is received at accumulated peaks (blue area) of about -67 dBm, showing that it is at a nearby distance. The accumulated average noise floor is about -110 dBm (green area). There are several narrow peaks on accumulated average at 2412 MHz, 2424 MHz, 2460 MHz and 2472 MHz with levels from -105 dBm to -100 dBm. At the ZigBee channel C of 2410 MHz the background noise has peaks at about -79 dBm.

From this spectrum result, as long as the XBee-PRO modules are working at the ZigBee channel C and having received signals above -79 dBm there should be no interruption of communication. If the received signals drop to between -80 dBm and -100 dBm, there will be intermittent loss due to signal interference or RF packet collision. Experiments and tests will verify this.

6.3.4 Measurement of soil attenuation

Soil is the main material for building the artificial rubble. The soil dug out from the hole is preserved for all the experiments and tests. Thus the first material to measure is the soil at the site. The data transmitter is protected by a thin plastic box and placed between two bricks in the bottom of the hole, Figure 6.7, and then covered by a thin piece of wood.

Figure 6.7: Data Transmitter for tests

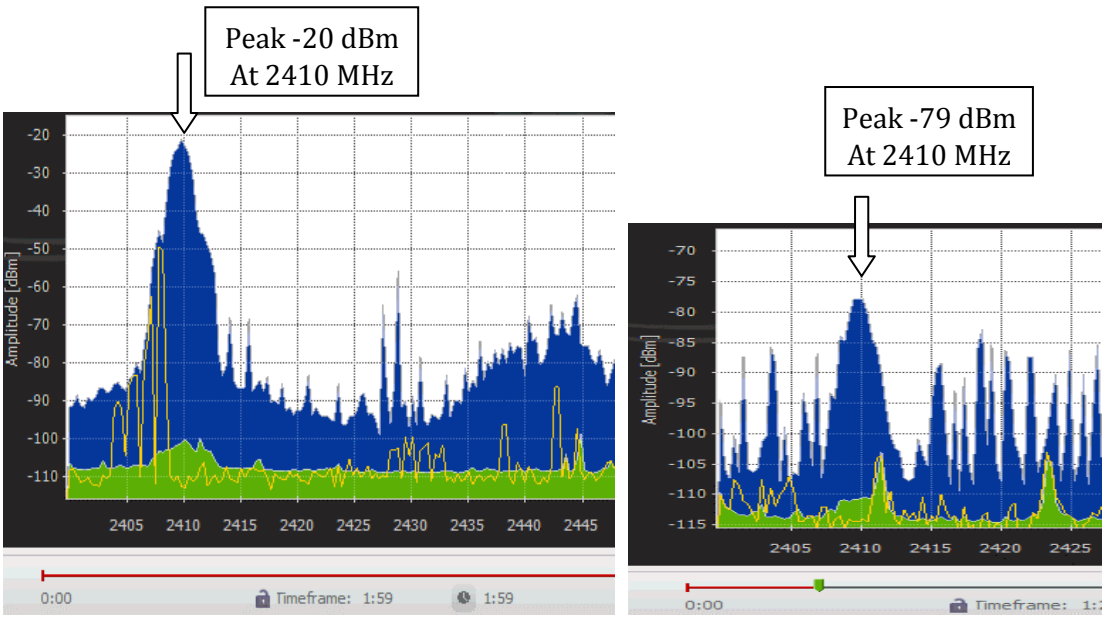

Figure 6.8: Spectrums for soil: before (left), after 30 cm soil (right)

With the transmitter turned on and before the hole was filled with soil the spectrum was recorded for 1 minutes and 59 seconds. The result on left side of Figure 6.8 shows a ZigBee spectrum of typical signature centred at 2410 MHz (channel C) with a peak of -20 dBm. Then soil was used to fill up the hole. The right side of Figure 6.7 shows the spectrum recorded for 1 minute 23 seconds after soil was filled to depth of 30 cm. Results were taken at three different depths of soil, 10 cm, 20 cm and 30 cm. At each depth, three readings were taken by moving the wooden support on the tripod (Figure 6.4 and 6.5). The average of these readings was recorded as the final result. The following table summarises the measurement of soil attenuations.

Table 6.2: Soil attenuations

Soil Depth	Average Measured Peak	Calculated Attenuations
0 cm	-20.0 dBm	0 dB
10 cm	-44.5 dBm	24.5 dB
20 cm	-72.0 dBm	52.0 dB
30 cm	-79.0 dBm	59.0 dB

From the results in Table 6.2, there is not much increase of attenuation when soil depth increased from 20 cm to 30 cm. This can be explained as when the soil depth is 30 cm it is at the same top surface level as the surrounding area. The RF signal can scatter though the surface layer of the surrounding area around the hole, and those areas are growing with lawn and will have roots and parts of the lawn that allow easier path (lower attenuation) for the RF signal. The following figure illustrates this scenario.

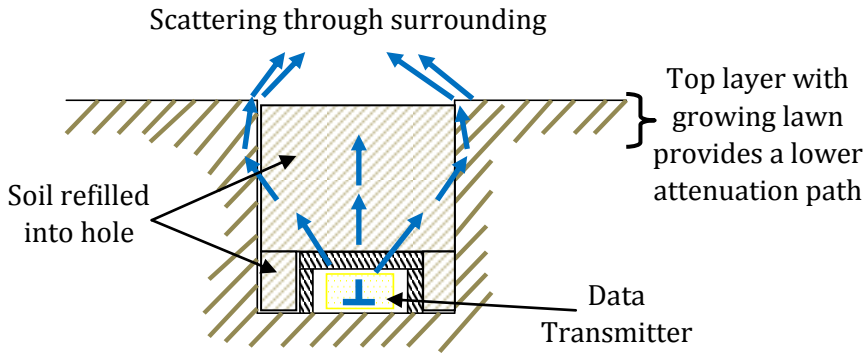

Figure 6.9: RF signal scattering through surrounding

6.3.5 Attenuation of various building materials

The same measurement experiment was carried out using a variety of standard building materials. The following photos illustrate the filling of some of the selected materials into the hole on top of the data transmitter during measurement of their attenuations.

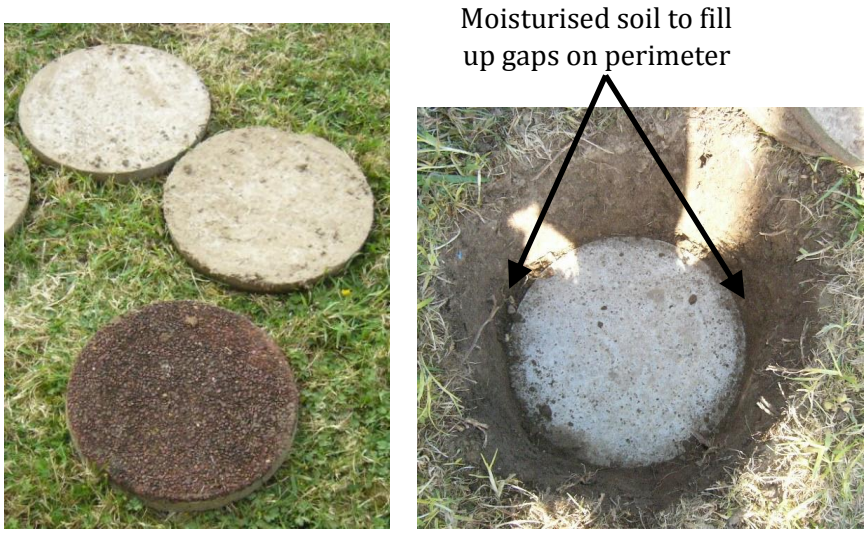

Figure 6.10: Concrete slabs

Figure 6.11: Concrete slabs with wire-mesh

To simulate reinforced concrete, concrete slabs that were interleaved with wire mesh were deployed. On alternate layers, wire mesh is placed alternatively at 90 degree and 45 degree orientations to maximise its attenuation effect on the RF signal. Gaps and openings on the perimeter of the concrete slabs are filled with moisturized soil.

When measuring the attenuation of bricks, each layer of bricks is placed into the hole in alternate 90 degree and 0 degree directions, and with its internal holes in horizontal orientation with the bottom of the hole, to maximise its attenuation effect on the RF signals.

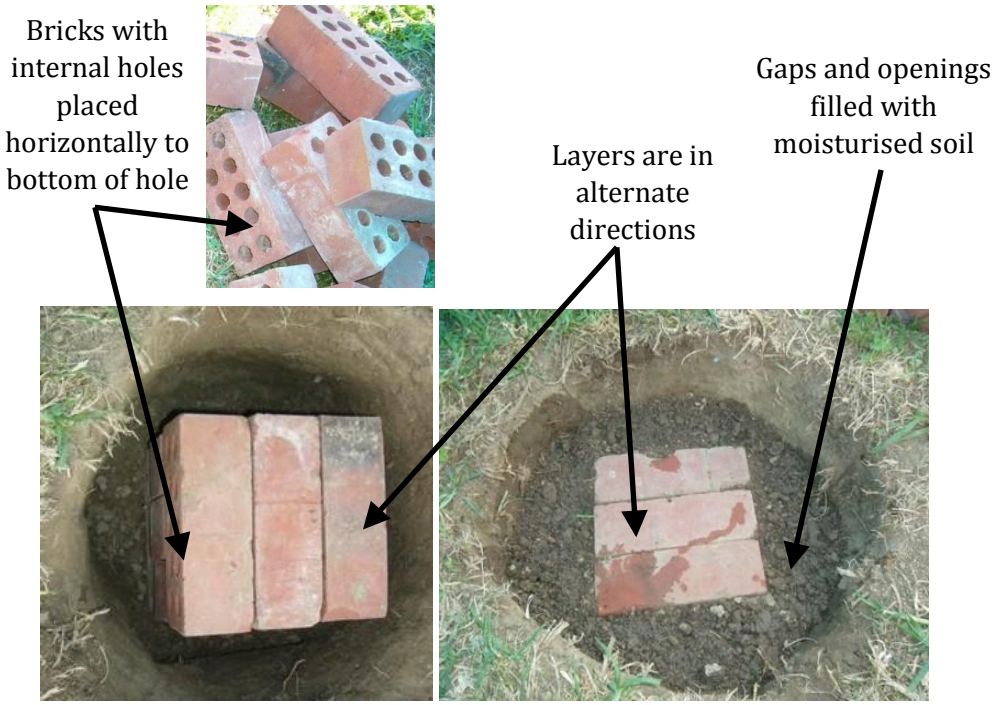

Figure 6.12: Placing bricks for attenuation measurement

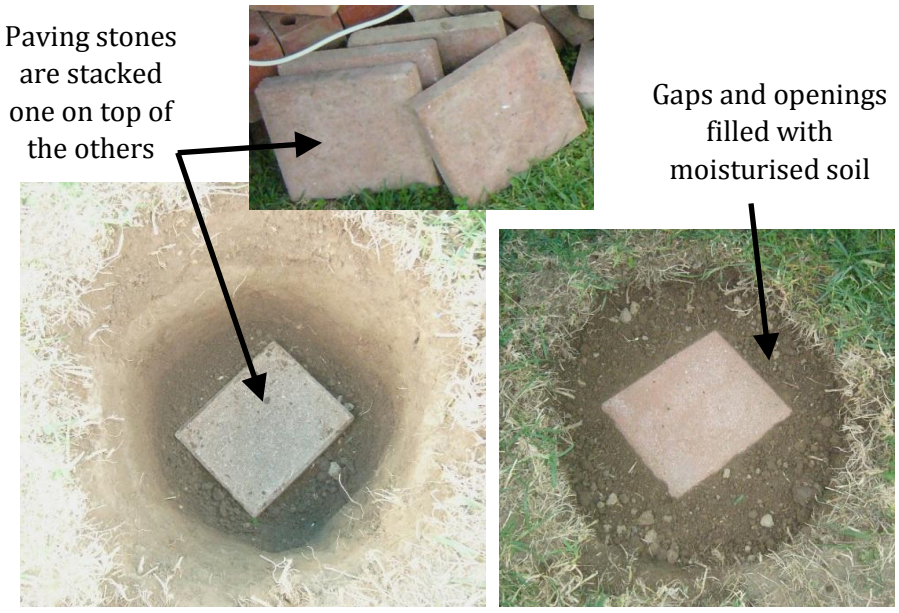

Figure 6.13: Placing paving stones for attenuation measurement

Table 6.3: Attenuations of various building materials

Soil	Thickness (cm)	Attenuation (dB)
	10.0	24.5
	20.0	52.0
	25.0	54.5
	30.0	59.0

Bricks	Thickness (cm)	Attenuation (dB)
1 Layer	7.5	9.0
2 Layer	15.0	18.5
3 Layer	22.5	22.5
4 Layer	30.0	23.5

Concrete	Thickness (cm)	Attenuation (dB)
1 Layer	3.5	8.5
2 Layer	7.0	15.0
3 Layer	10.5	17.5
4 Layer	14.0	22.0
5 Layer	17.5	23.0
6 Layer	21.0	23.5
7 Layer	24.5	28.5
8 Layer	28.0	36.0
9 Layer	31.5	39.5

Paving Stones	Thickness (cm)	Attenuation (dB)
1 Layer	4.0	3.3
2 Layer	8.0	8.0
3 Layer	12.0	15.8
4 Layer	16.0	20.0
5 Layer	20.0	22.0
6 Layer	24.0	24.5
7 Layer	28.0	25.0
8 Layer	32.0	26.0

Concrete + Wire	Thickness (cm)	Attenuation (dB)
1 Layer	4.5	9.5
2 Layer	9.0	19.5
3 Layer	13.5	29.5
4 Layer	18.0	36.5
5 Layer	22.5	54.5
6 Layer	27.0	56.5

The attenuations measured for the various materials are listed in Table 6.3. For easy comparison, the results are graphed in Figure 6.14.

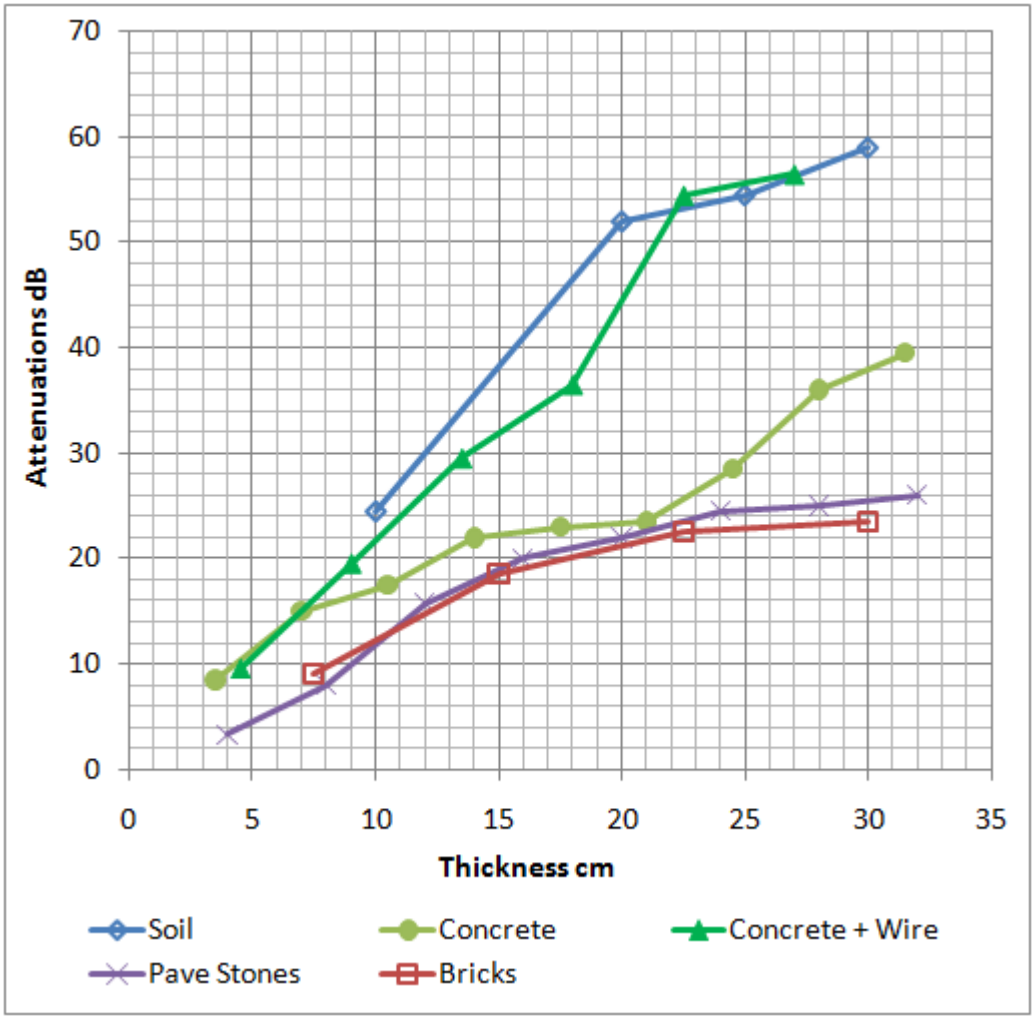

Figure 6.14: Graphical results of attenuation measurement

These results show both moisturised soil and concrete slabs with wire-mesh produce the highest attenuations of about 55 dB at a depth of 25 cm compared to the other materials. This aligns with the XBee-PRO manufacturer's test report that reinforced concrete has the highest attenuation compared to other building materials [17]. The results for soil match with the discussion by C.L. Holloway et al, who reported that moisturised soils becomes a very good waveguide material, that is it will produce high attenuation [29].

As moisturised soil is much easier to manage and reshape than concrete, it will be used as the main material for providing attenuation when building the artificial rubble.

6.4 Design of experiment setup in rubble

Recall that the main function of the wireless network is to provide a communication link for the daughter robots to send information to the mother robots situated on top of the rubble. As illustrated by the link tests and analysis in chapter 4, a direct link will not generally be possible between the mother robot and a daugther robot that is deep inside the rubble. The following experiment serves to verify that when a direct link is not possible, information can be sent from one robot to another until it arrives at the mother robot.

Four XBee-PRO modules and a laptop computer were used. Figure 6.15 shows the sectional view of the rubble experiment setup. Various materials and soil were used to fill up the holes and trench to simulate the rubble environment. At start of the experiment, a data transmitter was placed in the bottom of Hole No. 1, to simulate a daughter robot situation at the inner most area of the rubble. At later tests, two data routers were used for data routing. Router 1 was placed in the bottom of Hole No. 2. Router 2 was placed beside Hole No. 2 above ground, to simulate two other daughter robots at different locations of the rubble. The XBee-PRO coordinator is attached to the laptop, to simulate a mother robot outside at a distance from the rubble.

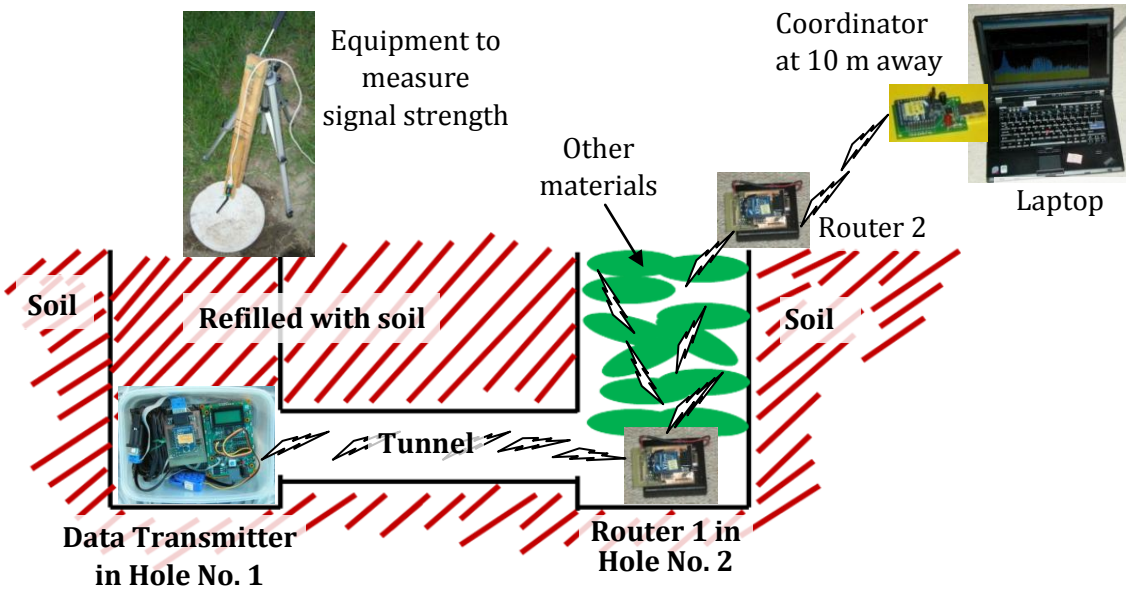

Figure 6.15: Sectional view of rubble experiment setup

The following figure illustrates the plan view of the rubble experiment setup.

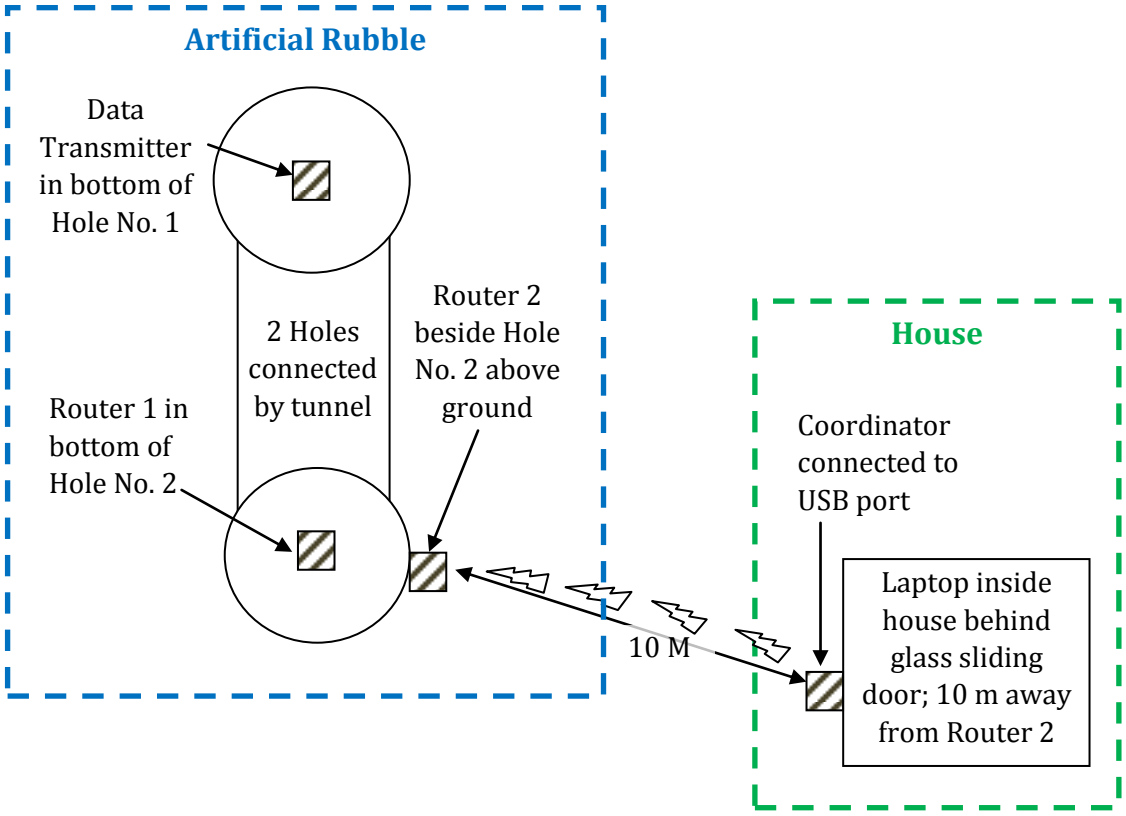

Figure 6.16: Plan view of rubble experiment setup

Referring back to the attenuation results from Table 6.3 and Figure 6.14, 30 cm soil will produce an attenuation of about 59 dB. Repeating the analysis in section 4.3.1, using XBee-PRO with output power of +18 dBm; the coordinator in the house is at a 10 metre distance which gives a free space path loss of about 60 dB at 2.4 GHz. The coordinator will receive the signal at -101 dBm which is just below the sensitivity of -100 dBm; that means the coordinator may just be able to receive some useful packets by a direct link.

Further signal reduction will be introduced by the tunnel and the materials in Hole No. 2. The XBee-PRO on the data transmitter will be programmed to a lower transmitting level of +10 dBm; this will further lower the received signal power such that a direct link will not be possible. This will achieve the aims of the experiment and forces the network to execute message routing mechanism.

6.5 Construction of artificial Rubble

Following the design in the previous section, the artificial rubble was built by digging a trench which linked two holes in the soil (Figure 6.17 and 6.18).

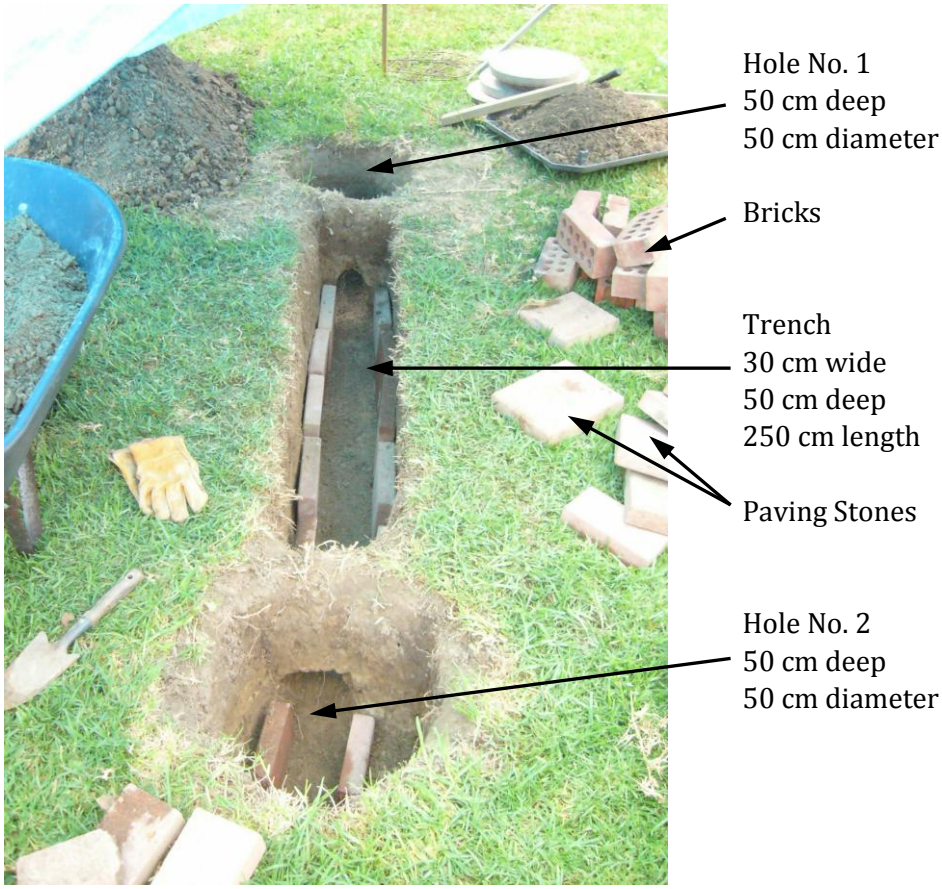

Figure 6.17: Construction of artificial rubble

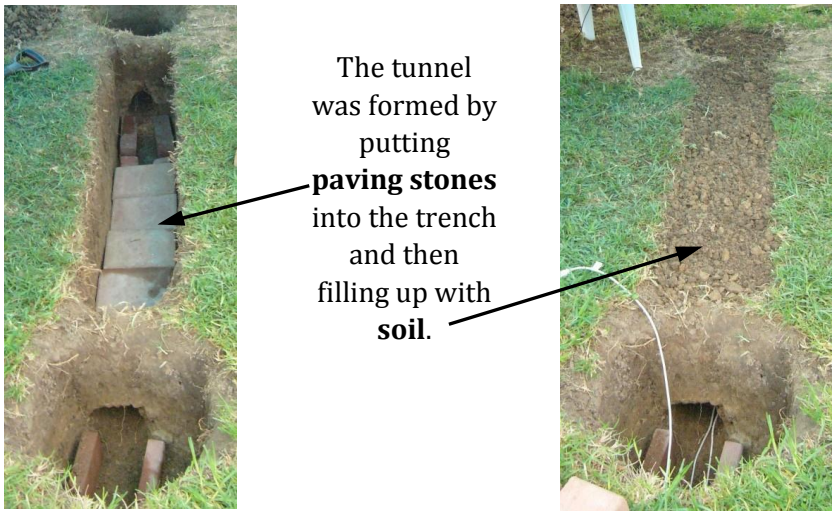

Figure 6.18: Building a tunnel in the rubble

6.6 Data Routing Experiments

When construction of the artificial rubble has come to the stage illustrated as Figure 6.17, an open trench connecting two holes was constructed, and tests on message passing between the simulated robots were initiated. These are described in the following sections.

6.6.1 Experiment Description and Results

Table 6.4 listed the sequence of experiment steps carried out to verify functioning of the wireless network.

Table 6.4: Routing experiment steps

	Experiment Description
1	• Data Transmitter (firmware as Router3 on Table 5.3, section 5.2.3) placed in bottom of Hole 1 • Hole 1 filled up with moisturised soil • Trench and hole 2 not filled • Data received by coordinator without error
2	• A tunnel of 19 cm height by 13 cm width was built in the trench by paving stones • The trench was covered with moisturised soil (Figure 6.18 right side) • Received signal strength measured at bottom of hole 2 was -50 dBm; received spectrum shown on Figure 6.20 • Both routers were switched off, laptop cannot receive any data
3	• Signal strength received above ground at nine spots as illustrated in Figure 6.19 were measured with values between -72.3 dBm to -83.3 dBm; graphed results shown on Figure 6.21
4	• Router 1 placed in bottom of hole 2 and switched on • Data received by laptop without error
5	• 8 pieces of concrete slab placed on top of router 1 • Intermittently, bytes missing in data received by coordinator
6	• Router 2 placed beside hole 2 above ground and switch on • All data received by coordinator without error

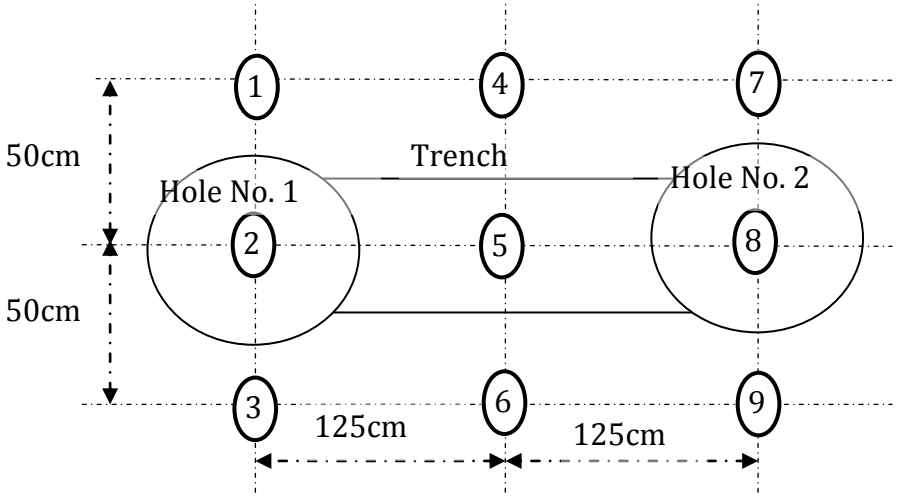

Figure 6.19: 9 locations for measuring signal strength above ground

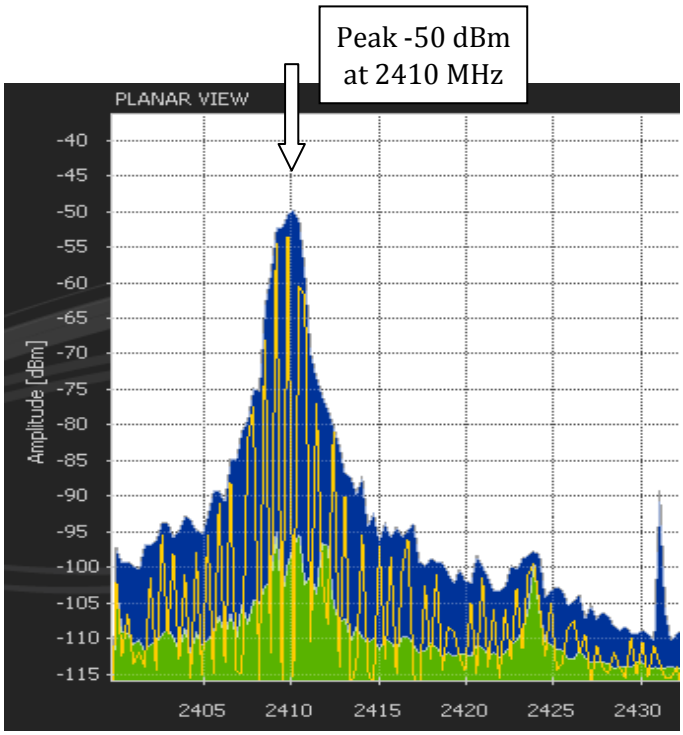

Figure 6.20: Signal Spectrum for experiment step 2

Figure 6.20 shows the received signal spectrum in bottom of Hole No. 2 with the transmitter placed at the bottom of Hole No.1. The peak signal power is at -50 dBm at the ZigBee channel used by the transmitter. As the transmitter is programmed as Router3 on Table 5.3, its transmitting power was set to +10 dBm; this means the tunnel introduced an attenuation of 60 dB.

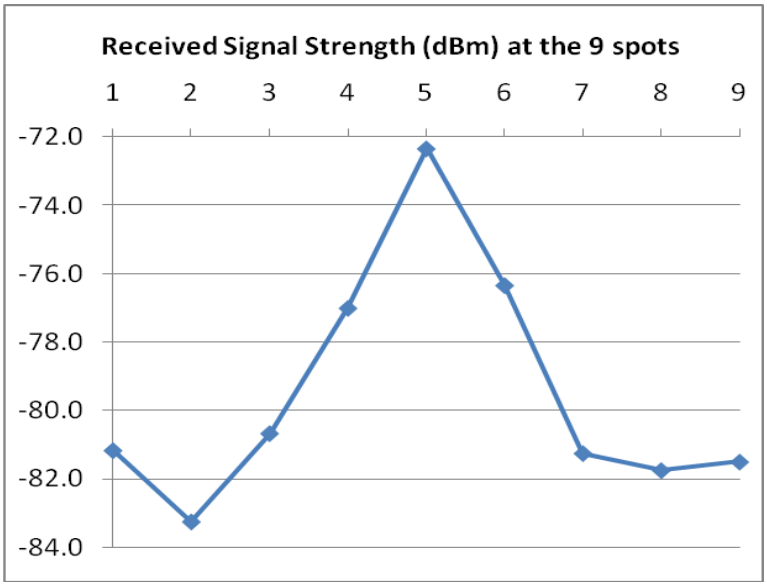

Figure 6.21: Received signal strength for experiment step 3

To ensure the RF signal does not escape above the tunnel, the signal strength is measured at nine spot in step 3 of the experiment. Results are illustrated in Figure 6.21. These results show the attenuation introduced by the rubble was 82.3 dB to 93.3 dB which is higher than the 80 dB measured by NIST for the collapsed buildings. With a 10 metre path loss about 60 dB in free space, the coordinator will receive the signal at -132.3 dBm to -143.3 dBm which is well below the receiver sensitivity of -100 dBm. This explains why a direct link cannot be established between the transmitter and the coordinator as shown by experiment step 2 in Table 6.4.

6.6.2 Routing Reconnection Tests

Further tests were carried out by switching Router 2 off for 5 minutes and then back to on again. The monitoring program shows no data at the instant of switching off. Then after about 1 minute, bytes of data reappeared but missing bytes were reported. This is back to step 5 in Table 6.4. About 1 minute after Router 2 was switched back on; all 500 bytes of data were displayed. This is back to step 6 in the table.

The tests demonstrate that the routers can re-establish the network automatically. Several runs of this reconnection test were carried out. Results showed that the

reconnection time varied between 40 seconds to 90 seconds. This variation depends on the power on and reconnection mechanism built-in on the XBee-PRO firmware provided by the manufacturer.

6.7 Summary

The experiment demonstrated that data routing is successfully implemented. By switching on and off the routers, it was verified that the routers can automatically reconnect with the network and re-establish the communication link from the simulated robot deep inside the rubble through the other simulated robots to the simulated mother robot at a safe distance from the rubble.

The attenuation results of the various building materials provided the foundation for selecting appropriate materials to build the artificial rubble. It also forms a set of useful references for future projects. Measurements showed that the artificial rubble produces an environment with attenuations that are comparable to those of the collapsed buildings reported by NIST [23].

The equipment developed provides a consistent method for ZigBee communication tests and RF signal measurements. It sets a low cost and effective example as compared to using a RF anechoic chamber with expensive devices. The artificial rubble is a side-product of the project that shows that a manageable, small scale, but still very useful disaster site can be built for USAR robots testing, as compared to that constructed by the NCAR team in a size of hundred square metres, using tons of steel reinforced sewer pipes [24].

Chapter 7 Conclusions

This chapter describes the conclusions of this thesis, lists the thesis contributions and provides suggestions for future research and improvement.

7.1 Conclusions

The aim of this thesis is to implement a wireless network for the team of robots during USAR missions, such that all the daughter robots deep inside the rubble can report their findings by sending messages through the wireless network to the mother robots outside the rubble. This aim is successfully achieved by the following parts of the project.

Selection of wireless modules

Studies on wireless networking technologies and a literature review on RF signals in collapsed buildings were carried out. Several possible candidates of ZigBee wireless modules from different brands were sourced. Comparisons on their specifications, costs, sizes and development support tools were done. Two models were selected for further experimentation.

Development of prototypes

A list of prototypes was sourced and developed by attaching wireless modules to microcontrollers. Sensors were added to the prototypes. They form the development platform for building the team of daughter robots. Cost analysis was done on the prototypes and concluded that they are low cost and fit within the overall project budget guidelines.

RF Signal Tests

Simulated robots were built using the two selected wireless modules. Wireless link tests were carried out in an office building. Artificial scenarios simulate the effect

of rubble and the performance of the wireless modules was investigated. Analysis of the test results combined with the literature review on wireless technologies in rubble environment resulted in the selection of the XBee-PRO as the preferred wireless ZigBee module.

Wireless network implementation

With the selected XBee-PRO module, firmware is chosen for implementing the wireless network. Networking features and configuration of the firmware are studied. The cluster-tree network model is set up by configuring profile files for the firmware. The wireless modules are programmed with profile files that were specifically built for the required wireless network structure, a coordinator with three routers.

The routers are programmed to AT mode which allows operation as transparent modems for the simulated robots. The coordinator is programmed to API mode which attaches to the monitor computer to simulate a mother robot. A ZigBee tester program is written to run on the monitoring computer which extracts network communication information by sending structured API frames to the coordinator. One of the routers is attached to a microcontroller which is programmed as a data transmitter, sending data at required blocks with specific timings for communication tests.

Communication test in artificial rubble

One of the major tasks for this project is to test the wireless network in a real rubble environment. A site was selected and a simulation of a real artificial rubble environment was built. A set of measurement experiments were constructed. Various building materials were tested and their attenuations to the RF signal were measured. After analysing the measurement results, a rubble scenario was designed and constructed at the selected site.

Moisturized soil is chosen as the base material to form the foundation of the rubble. Paving stones and concrete slabs were used together with moisturised soil to build the scenarios for stages of the experiment. The simulated daughter robots and the mother robot were deployed into the artificial rubble, and a series of experiments were carried out.

Results from the experiment were compared with results from other research and verified that the artificial rubble can produce attenuations that are very close to those from real examples of collapsed buildings. The experiments verified that the wireless network provides the required function, allowing a data transmitter that simulated a daughter robot at the inner most location of the rubble to send information by routing through other robots in the rubble to the simulated mother robot located at 10 metres distance from the rubble. Switching on and off the routers simulated daughter robots getting out and coming back at any time to rejoin the network and re-establish the wireless network. The experiment in the artificial rubble verified that the wireless network is fully functional and could be useful for the team of USAR robots.

7.2 Future work to minimise data loss

During development of the data transmitter program and the tester program on the monitoring computer, occasional data loss is discovered when testing the wireless network in open spaces where all modules are in close proximities and where attenuations in open space should not be a factor in data loss.

7.2.1 Data lost and measures taken

Two data lost scenarios were observed and measures were taken as follows:

1. Pauses of 20 ms after each 72 bytes of data transmission at the transmitter. This allows time for the wireless modules to process and send the data to the network. It also allows time for the monitoring computer to receive and process the data for display.

2. Displays on the monitoring computer program need to be cleared every 2 rounds of 5 blocks of 500 bytes of data; that is 5000 bytes. This is to avoid the monitoring program being slowed down when there are too many bytes on display. This slow-down affects the taking up of data from the serial port. Such data is held up on the port and pushed back to the wireless link. Eventually data loss occurs due to buffer overflow.

The following describes other issues that may need to be considered for any future projects using a similar set of hardware, firmware and software.

7.2.2 Hardware handshaking on XBee-PRO modules

A simplified approach was taken when designing the adaptor boards for connecting the XBee-PRO module to a RS232 serial port. Figure 7.1 illustrates the "quick connection" scheme.

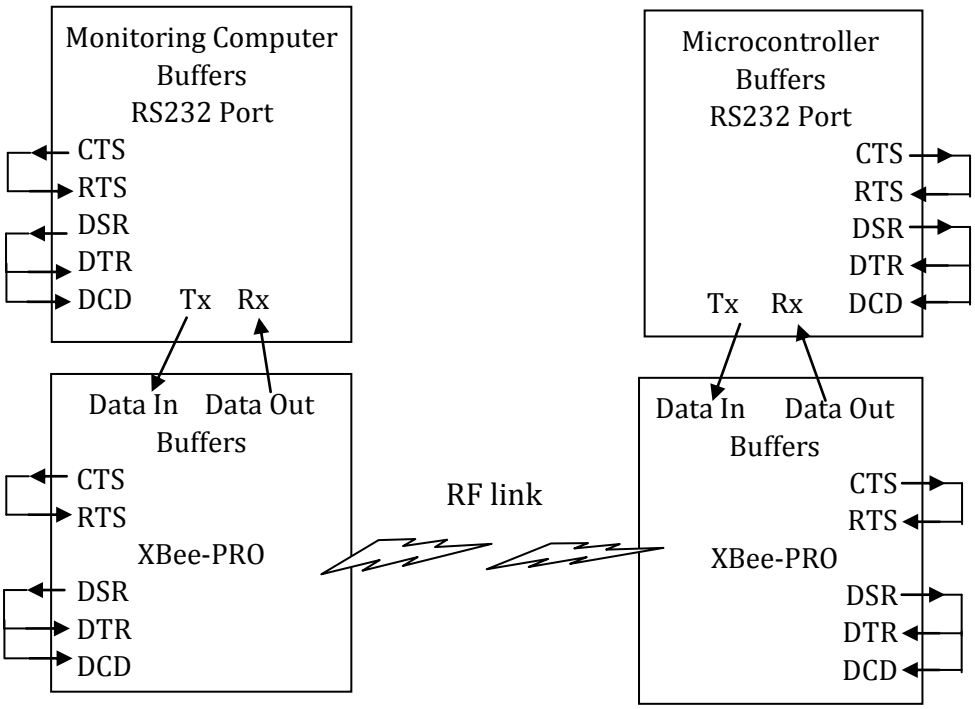

Figure 7.1: XBee-PRO RS232 port connection diagram

This connection scheme does not implement any handshaking between the RS232 ports. The control lines, CTS, RTS, DSR, DTR, and DCD are looped back to each

other without connecting to the other port. Proper communications relied on extraction of bit synchronisation signal from the data lines, Rx and Tx.

To avoid data loss, the following full connection scheme is recommend by the XBee-PRO manufacturer.

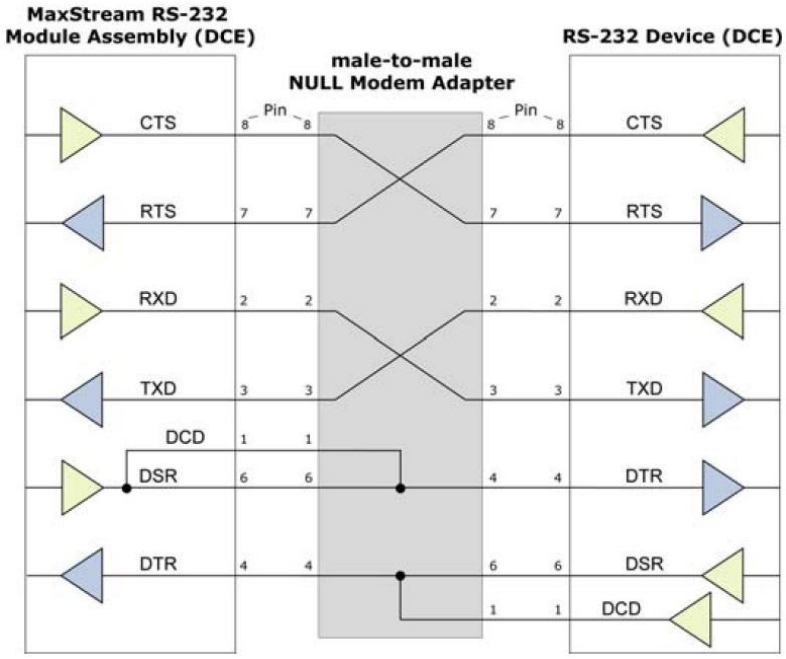

Figure 7.2: Full connection scheme for XBee-PRO to RS232 port

7.2.3 Useful data rate

Implementing packetization

As mentioned in section 7.2.1 point 1, a pause after 72 bytes of data is added to avoid data loss. It is a requirement for the XBee-PRO firmware to implement routing between nodes on the network. During data transfer, according to the firmware manual, data is received into the DI (Data In) buffer until one of the following happens, data will be sent out to the RF link.

1. No further data is received for the amount of time determined by the RO (Packetization Timeout) parameter. RO sets the number of characters (bytes) to wait for timeout.
2. Maximum number of data (72 bytes) for one packet has been received.

Based on the concept of packetization, several experiments were carried out on various baud rates (from 19200 to 115200). The microcontroller was programmed to send out blocks of 500 bytes repeatedly with 100 ms pause between blocks. An extra pause of 20 ms is added after sending 72 bytes to the serial port. This extra pause exceeds the RO parameter which is usually set between 3 to 10 bytes. Results of the experiments showed that data loss will occur if the extra pause is taken out or applied after too many bytes were sent. Adding the pauses makes the actual data rate about 28800 bits per second. Due to the limitation on the microcontroller program, it cannot be set to a pause value between 10 ms and 20 ms; otherwise, it may be able to work at a higher bit rate.

Buffering and processing delays

Further tests showed that data loss will also occur if the 100 ms pause between the blocks is decreased to below 30 ms at a baud rate setting of 115200. This cannot be explained by the packetization timeout mechanism and does not relate to the clearing up of the serial port buffer on the monitoring computer (section 7.2.1 point 2).

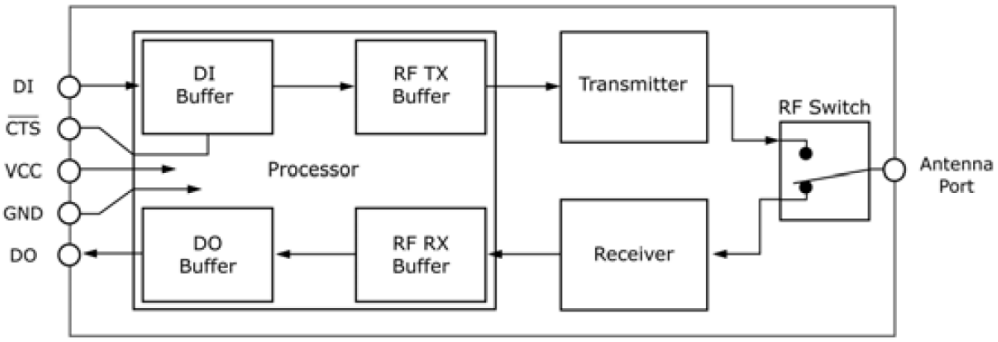

Figure 7.3: XBee-PRO internal data flow diagram

Referring to the Internal Data Flow Diagram (Figure 7.3) provided in the firmware manual, the RF data coming into the module has to get through the RF RX buffer then transfer to the DO (Data Out) buffer. Data loss will occur when more data is passed from the RF RX buffer into the DO buffer than it can handle. The next possible time delay to consider is processing in the XBee-PRO module. The ZigBee

stack is implemented by the processor in the module. It takes time to process networking policies and to implement the routing mechanism. To avoid data loss, extra transmission pauses are required between large blocks of data.

With reference to the analysis by Benoit et al [31], a maximum throughput of 163 Kbps can be achieved using no address (one transmitter and one receiver on one network) and no acknowledgement. The worst case scenario is 49.8% bandwidth efficiency (125 Kbps) when having long address with acknowledgement at packet size of 122 bytes. The networking layer implemented by the XBee-PRO firmware (Figures 5.2 and 5.3) to provide routing in a cluster-tree network topology that uses 64 bit addresses (32 bits for destination address, 16 bits for network address and 16 bits for parent network address).

From the firmware manual, a passive acknowledge scheme is implemented when using broadcast commands. A network level acknowledgement scheme is also implemented in the firmware when a device goes into transmit mode. An acknowledgment packet will be sent by the receiver and routed back to the source device. If a network acknowledgement is not received, the source node will re-transmit the data.

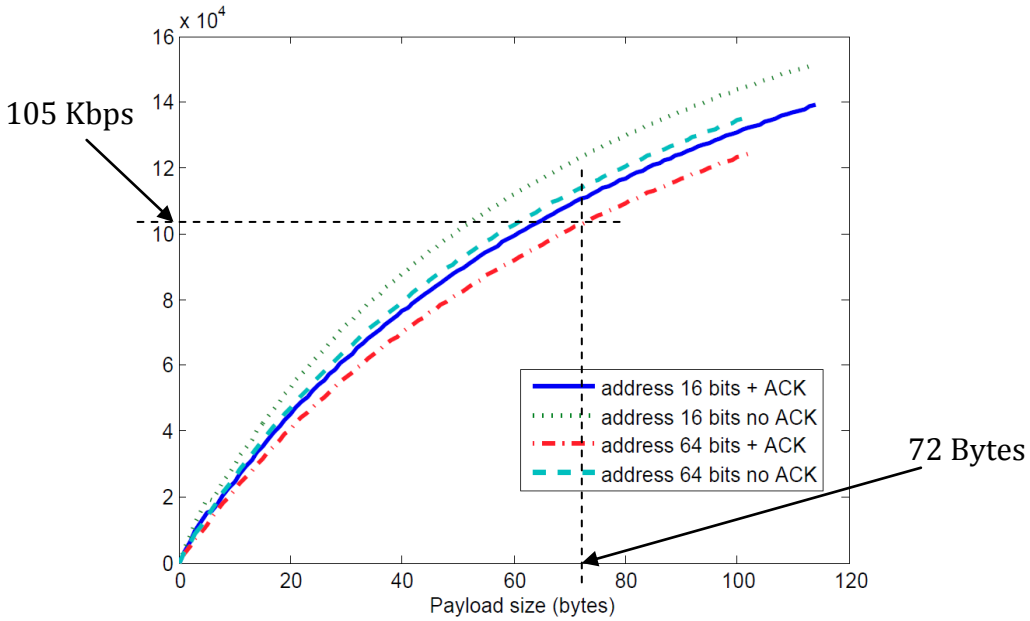

Figure 7.4: Useful bitrate graph by Benoit et al [31]

Using the "Useful bitrate graph" (Figure 7.4) produced by Benoit et al, the useful data rate is at 105 Kbps at packet size of 72 bytes with ACK and 64 bits address. To avoid data loss, an application has to be designed to operate the XBee-PRO wireless network at or below this useful data rate.

7.2.4 High-level mechanism to avoid data loss

Combining all of the above analyses, to establish a reliable data link for the daughter robots in the rubble, the high-level application software running on the monitoring computer and the microcontroller must be designed for a lower actual data rate with proper handshaking and acknowledgement mechanisms.

The root cause of data loss is the trade-off on simplified hardware design that does not implement hardware handshake between the RS232 interfaces. The above sections explained how to avoid data loss by placing pauses between transmissions. This is not an efficient approach. Proper handshaking mechanisms can be designed into the transmission and receiving software programs. The following software mechanisms are suggested but due to limitation on project time and budget allocation they were not implemented on the system.

"ACK before next transmit"

A common technique is to wait for an ACK (acknowledgement) from the receiver of the previous data block before sending the next data block. For short messages, such as a two bytes temperature value, an ACK may not be necessary.

"Silence" and "Wake up"

Since the ZigBee channel has a limited bandwidth of 250 Kbps maximum, when there are multiple devices trying to send large amounts of data, the channel can easily suffer from too many data packet collisions and eventually none of the receivers will get a complete message. A "silence" broadcast command can be sent from the coordinator to tell all devices on the same network to go silent. Then a

request command, such as capturing a photo and sending it back, can be addressed to a particular robot. The robot sends back the photo, which is usually thousands of bytes, in blocks of hundreds of bytes using the "ACK before next transmit" mechanism to the coordinator. If no ACK is received for any block, a re-transmission can be done. After the coordinator has received all the blocks and verified a completed photo, a "Wake up" command can be broadcasted to all devices again to tell them get back to normal.

7.3 Contributions of this thesis

This thesis has contributed the following regarding the use of wireless USAR robots.

- A prototype platform for developing wireless daughter robots is developed and can be used for any future projects.
- The process of configuring the wireless modules and developing the test software forms a useful structure for similar projects.
- The results of material attenuations and the construction of artificial rubble can be a good example and provide useful guidelines of building a close to real test-bed for wireless robots in rubble.
- Analysis of results and performance provides measures to get around the limitation of actual data rate for developing useful wireless application.

These results have been published and presented at two conferences during the development of this thesis [32] [33].

7.4 Summary

The project described by this thesis has successfully implemented a wireless network for the team of robots in rubble environment. Multiple sensors and actuators can be added to the prototype robot without difficulty for any further functional enhancement. It was for this reason (to facilitate developments) that a mobile platform was constructed to accommodate the communication device.

Attenuations of soil and building materials on 2.4 GHz ZigBee RF signals were measured. An artificial close to real rubble environment was designed and constructed. A group of simulated USAR robots were developed and deployed into the artificial rubble. It was demonstrated that the artificial rubble in the soil environment is suitable for testing and verifying the wireless network for USAR missions.

The experiments indicated that ZigBee technology implemented by the XBee-PRO modules can form a useful mesh wireless network for USAR robots. The modules can automatically reconnect after network interruption. Experiments have verified that a simulated mother robot at 10 metres away from the rubble can communicate with all the robots inside the rubble.

References

[1] NZ USAR, "Why USAR is needed?" visited on 29th February, 2010, http://www.usar.govt.nz/usarwebsite.nsf/wpg_URL/About-USAR-Why-USAR-is-Needed-Index?OpenDocument

[2] D. McGuigan, 2002, "Urban Search and Rescue and the Role of the Engineer", retrieved on 1st September 2007, from http://www.usar.govt.nz /usarwebsite.nsf/Files/McGuiganUSARMasters/$file/McGuiganUSARMasters.pdf

[3] School of Chemical and Physical Science, VUW, "Search and Rescue (SAR)", retrieved on 30th August 2007, from http://www.vuw.ac.nz/scps /research/mechatronics/sar.aspx

[4] D. Carnegie, "A Thee-Tier Hierarchical Robotic System for Urban Serach and rescue Applications", IEEE International workshop on Safety, Security and Rescue Robotics (New York, IEEE, 2007), pp. 1-6.

[5] D.A. Carnegie and J. C. Cordes, "The mechanical design and construction of a mobile outdoor multi-terrain mechatron," Proceedings of the Institution of Mechanical Engineersm Part B, Engineering Manufacture, Vol 218, pp 1563-1575, Nov 2004

[6] D. A. Williamson, "The development of a "Mother" agent for a hierarchical multi-robot urban search and rescue system: a thesis submitted to the Victoria University of Wellington", 2007

[7] The IETF, "Mobile ad-hoc network (MANET) working group", visited on 30th August 2007, from http://www.ietf.org/html.charters/manet-charter.html

[8] S. Das, H. Pucha and Y. Hu, "MicroRouting: A Scallable and Robus Communication Paradigm for Spase Ad Hoc Networks", in proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, 2005

[9] Sanderson et al, "Developing Mobile Middleware – An Analysis of Rescue and Emergency Operations", (2007), University of Oslo

[10] Third Generation Partnership Program (3GPP), "General Packet Radio Service (GPRS); Service description", visited 30th August 2007, http://www.3gpp.org/ftp/Specs/html-info/23060.htm

[11] Third Generation Partnership Program (3GPP), "Work programme for the standarization of Universal Mobile Telecommunications System (UMTS)", visited 30th August 2007, http://www.3gpp.org/ftp/Specs/html-info/0001U.htm

[12] IEEE, "IEEE 802.16 WirelessMAN® Standard for Wireless Metropolitan Area Networks", visited 30th August 2007, from http://ieee802.org/16/

[13] Bluetooth SIG, "Compare with other technologies", last accessed on 5th May 2010, visited http://www.bluetooth.com/English/Technology/Works /Pages/Compare.aspx

[14] Kinney Consulting LLC, "ZigBee Technology: Wireless Control that Simply Works", retrieved on 4th September 2007, http://www.zigbee.org/ imwp/idms/popups/pop_download.asp?contentID=5162

[15] R. Akl and X. Li, "Indoor propagation modelling at 2.4 GHz for IEEE 802.11 network", in proceedings of the Sixth IASTED International Muti-Conference on Wireless and Optical Communications, Wireless Networks and Emerging Technologies, 2006

[16] O. W. Ata, "Grade of Service Signal Density Enhancement – Modeling In-building penetration loss in various morphologies", in proceeding of the IEEE Tropical Conference on Wireless Communication Technology, 2003

[17] MaxStream Inc., "Indoor Path Loss", retrieved on 8th December 2008, from http://ftp1.digi.com/support/images/XST-AN005a-Indoor.pdf %20(334.50KB).pdf

[18] MaxStream Inc., "Product Manual v8x17 Beta – ZigBee Protocol", retrieved on 13th October 2008, from http://www.digi.com/support/kbase/ kbaseresultdetl.jsp?id=2182

[19] Panasonic Corporation of North America, "PAN802154HAR00 ZigBee Brochure", retrieved on 4th August 2007, from http://www.panasonic.com /industrial/components/pdf/zigbee_brochure.pdf

[20] Surveyor Corporation, "SRV-1 Surveyor robot", visited on 20th March 2010, http://surveyor-corporation.stores.yahoo.net/srrowestkit.html

[21] JoinMax Digital, "JoinMax RoboEXP Learning Kit V4.0", visited on 15th March 2010, http://www.roboexp.com/products/list.asp?id=81

[22] Industry Canada, "GL-01 — Guidelines for the Measurement of Radio Frequency Fields at Frequencies from 3 KHz to 300 GHz", visited on 30th April 2010, from http://www.icce.ca/eic/site/smt-gst.nsf/eng/sf08511.html

[23] C. L. Holloway, G. Koepke, D. Camell, K. A. Remley, "Radio Propagation Measurements Before, During, and After the Collapse of Three Large Building Structures", in proceedings of the 2008 General Assembly of the International Union of Radio Science (Union Radio Scientifique Internationale-URSI)

[24] A. Ferworn, N. Tran, J. Tran, G. Zarnett, F. Sharifi, "WiFi repeater deployment for improved communication in confined-space urban disaster search", IEEE International Conference on System of Systems Engineering, pp. 1 – 5, 16-18 April 2007

[25] D-Link Corporation, “DWL-2100AP, High Speed 2.4GHz (802.11g) Wireless 108Mbps1 Access Point”, retrieved on 15 January 2009 from http://www.dlink.com/products/resource.asp?pid=292&rid=912&sec=0

[26] MaxStream Inc. & Digi International Inc., “X-CTU firmware programming software”, retrieved on 25th October 2007, from http://www.digi.com/support/kbase/kbaseresultdetl.jsp?kb=125

[27] TMS Software, “Asyn32 Serial Port demo program”, visited on 15th April 2010, from http://www.tmssoftware.com/site/async32.asp

[28] GHD Ltd, Part 1 of “2001 (March) Pakuranga Creek Catchments Comprehensive Catchment Discharge Consent Application - Comprehensive Catchment Study and Management Plan Options”, retrieved on 26 Jan 2009 from http://www.manukau.govt.nz/tec/catchment/pakuranga_pages/pdf/pakuranga_creek_ccdc1_low.pdf

[29] C.L. Holloway, D.A. Hill, R.A. Dalke, G.A. Hufford, “Radio wave propagation characteristics in lossy circular waveguidessuch as tunnels, mine shafts, and boreholes”, IEEE Transactions on Antennas and Propagation, Volume: 48, Issue: 9, pp. 1354-1366, Sep 2000

[30] MetaGeek LLC, “The Wi-Spy 2.4x device”, last visited on 15th April 2010, from http://www.metageek.net/products/wi-spy-24x

[31] Benoit et al, “Throughput and delay analysis of unslotted IEEE 802.15.4”, retrieved on 14th October 2009, from https://www.academypublisher.com/~academz3/jnw/vol01/no01/jnw01012028.pdf

[32] C. Tsui, L. Jennings and D. Carnegie, "Is ZigBee a suitable communication link for the 'Robot Family' at disasters?" in proceedings of ENZCON, November 2007

[33] C. Tsui, D. Carnegie and Q. W. Pan, “USAR Robot Communication Using ZigBee Technology”, CCIS 44, p.380 ff, 2009

Appendix A: CD Contents

The attached CD (in the back cover of the printed copy) contains the following:

1. Soft copy of this thesis in PDF format
2. Prototype PCB Altium files
3. RoboExp Program
    a. Source Files for Data Transmitter
    b. Source files for adding I^2C Temperature Sensor
    c. RoboExp development software installer
4. ZigBee tester program source files (Borland Delphi v5)
5. Photos
    a. Rubble construction
    b. Attenuation measurement settings
6. RF signal test records
    a. Samples of measured spectrums
    b. Chanalyzer software installer
7. XBee-PRO Firmware
    a. Manual
    b. Configuration files
    c. X-CTU software installer

Appendix B: ZigBee Tester Program Source Code

The following files are the full source code listings for ZigBee tester program on the monitoring computer.

B1. Borland Delphi project file

```
program APIbyAddress_V3;

uses
  Forms,
  formMain in 'formMain.pas' {frmMain},
  APIcommands in 'APIcommands.pas';

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TfrmMain, frmMain);
  Application.Run;
end.
```

B2. User interface main form file

```
unit formMain;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ComCtrls, VaConst, VaTypes, VaClasses, VaComm,
  ExtCtrls, APIcommands;

type
  TfrmMain = class(TForm)
    VaComm1: TVaComm;
    StatusBar1: TStatusBar;
    Panel1: TPanel;
    EditTransmit: TEdit;
    CheckBoxAddLinefeed: TCheckBox;
    ButtonTransmit: TButton;
    btnTransmit1000: TButton;
    Panel2: TPanel;
    Panel5: TPanel;
    ButtonOpen: TButton;
    ButtonClose: TButton;
    CheckBoxRTS: TCheckBox;
    CheckBoxDTR: TCheckBox;
    CheckBoxBREAK: TCheckBox;
    CheckBoxXON: TCheckBox;
    Panel6: TPanel;
```

```
    LabelParity: TLabel;
    ComboParity: TComboBox;
    ComboStopbits: TComboBox;
    LabelStopbits: TLabel;
    LabelDataBits: TLabel;
    ComboDatabits: TComboBox;
    ComboBaudrate: TComboBox;
    LabelBaudrate: TLabel;
    Bevel1: TBevel;
    ButtonReset: TButton;
    Bevel2: TBevel;
    Label1: TLabel;
    ComboPortNum: TComboBox;
    Memo2: TMemo;
    btnSendCmd: TButton;
    edtCmd: TEdit;
    btnSendMSG: TButton;
    Memo1: TMemo;
    Memo3: TMemo;
    Memo4: TMemo;
    ComboBox1: TComboBox;
    Label2: TLabel;
    Label3: TLabel;
    Label5: TLabel;
    Label4: TLabel;
    edtAddress: TEdit;
    Label6: TLabel;
    Label7: TLabel;
    EdtTotalRxBytes: TEdit;
    EdtAccErrBytes: TEdit;
    BtnClearDeviceList: TButton;
    Label8: TLabel;
    EdtCountErrMsg: TEdit;
    Label9: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure ButtonOpenClick(Sender: TObject);
    procedure ButtonCloseClick(Sender: TObject);
    procedure ButtonResetClick(Sender: TObject);
    procedure ButtonTransmitClick(Sender: TObject);
    procedure Comm1TxEmpty(Sender: TObject);
    procedure Comm1Break(Sender: TObject);
    procedure Comm1Cts(Sender: TObject);
    procedure Comm1Dsr(Sender: TObject);
    procedure Comm1Error(Sender: TObject; Errors: Integer);
    procedure Comm1Ring(Sender: TObject);
    procedure Comm1Rlsd(Sender: TObject);
    procedure ComboBaudrateChange(Sender: TObject);
    procedure ComboDatabitsChange(Sender: TObject);
    procedure ComboStopbitsChange(Sender: TObject);
    procedure ComboParityChange(Sender: TObject);
    procedure btnTransmit1000Click(Sender: TObject);
    procedure CheckBoxRTSClick(Sender: TObject);
    procedure CheckBoxDTRClick(Sender: TObject);
    procedure CheckBoxBREAKClick(Sender: TObject);
    procedure CheckBoxXONClick(Sender: TObject);
```

```
    procedure VaComm1Data(Sender: TObject; Count: Integer);
    procedure VaComm1Event(Sender: TObject);
    procedure VaComm1Open(Sender: TObject);
    procedure VaComm1Close(Sender: TObject);
    procedure ComboPortNumChange(Sender: TObject);
    procedure btnSendCmdClick(Sender: TObject);
    procedure btnSendMSGClick(Sender: TObject);
    procedure ComboBox1Click(Sender: TObject);
    procedure Label5DblClick(Sender: TObject);
    procedure BtnClearDeviceListClick(Sender: TObject);
  private
    procedure HandleException(Sender: TObject; E: Exception);
  public
    { Public declarations }
  end;

var
  frmMain: TfrmMain;
  RxMsg: String ='';
  Hs1: String ='';
  countMsg: Integer =0;
  NewMsg : boolean = false;
  CompleteMsg : boolean = false;
  CountBytes: Integer = 0;  //total byes received by serial port
  ErrBytes: integer = 0;
  countOneMsg: Integer = 0; //counter for counting no. of '0'..'9'
  StartOneMsg: boolean = false; // for counting no. of '0'..'9'
  DispCount: Integer =0;
  Count5: Integer =0;

implementation

{$R *.DFM}

procedure TfrmMain.FormCreate(Sender: TObject);
begin
  Application.OnException := HandleException;

  with ComboPortNum do
    ItemIndex := Items.IndexOf('9');        //('3');
  with ComboBaudrate do
    ItemIndex := Items.IndexOf('br57600');  //('br38400');
  with ComboDataBits do
    ItemIndex := Items.IndexOf('db8');
  with ComboParity do
    ItemIndex := Items.IndexOf('paNone');
  with ComboStopbits do
    ItemIndex := Items.IndexOf('sb10');

  VaComm1.BaudRate := TVaBaudrate(ComboBaudrate.ItemIndex);
  VaComm1.Databits := TVaDataBits(ComboDatabits.ItemIndex);
  VaComm1.Parity := TVaParity(ComboParity.ItemIndex);
  VaComm1.StopBits := TVaStopBits(ComboStopbits.ItemIndex);
end;
```

```
procedure TfrmMain.HandleException(Sender: TObject; E: Exception);
begin
  if E is EVaCommError then
    with E as EVaCommError do
      ShowMessage(Message);
end;

procedure TfrmMain.ButtonOpenClick(Sender: TObject);
begin
  VaComm1.Open;
  Comm1Cts(VaComm1);
  Comm1Dsr(VaComm1);
  Comm1Ring(VaComm1);
  Comm1Rlsd(VaComm1);
end;

procedure TfrmMain.ButtonCloseClick(Sender: TObject);
begin
  VaComm1.Close;
  Comm1Cts(VaComm1);
  Comm1Dsr(VaComm1);
  Comm1Ring(VaComm1);
  Comm1Rlsd(VaComm1);
end;

procedure TfrmMain.ButtonResetClick(Sender: TObject);
begin
  Memo1.Lines.Clear;
  Memo2.Lines.Clear;
  Memo3.Lines.Clear;

  CountBytes := 0;  // total number of byes received by serial port
  ErrBytes := 0;
  CountOneMsg := 0;
  EdtTotalRxBytes.Text := '0';
  EdtAccErrBytes.Text := '0';
  EdtCountErrMsg.Text := '0';
end;

procedure TfrmMain.BtnClearDeviceListClick(Sender: TObject);
begin
  Memo4.Lines.Clear;

  ComboBox1.Items.Clear;
  ComboBox1.Items.Add('Broadcast');
  ComboBox1.ItemIndex :=0;
  edtAddress.Text := 'Broadcast';
end;

procedure TfrmMain.ButtonTransmitClick(Sender: TObject);
var
  S: string;
  Ok: Boolean;
begin
  S := EditTransmit.Text;
```

```
  if CheckBoxAddLinefeed.Checked then
    S := S + #13#10;
  Ok := VaComm1.WriteText(S);
  if not Ok then
    Memo1.Lines.add('Error writing to: '
          + Format('Port %d', [VaComm1.PortNum]))
  else
  begin
    Memo1.Lines.add(Format('Writing %d characters', [Length(S)]));
  end;
end;

procedure TfrmMain.btnTransmit1000Click(Sender: TObject);
var
  I: Integer;
  S: string;
begin
  if MessageDlg('This will sent the input a thousand times,
continue?',
    mtConfirmation, [mbOk, mbCancel], 0) <> mrOk then exit;
  S := EditTransmit.Text;
  if CheckBoxAddLinefeed.Checked then
    S := S + crlf;
  for I := 0 to 1000 do
  begin
    VaComm1.WriteText(S);
    Application.ProcessMessages;
  end;
end;

procedure TfrmMain.Comm1TxEmpty(Sender: TObject);
begin
  Memo1.Lines.add('TxEmpty signal detected...');
end;

procedure TfrmMain.Comm1Break(Sender: TObject);
begin
  Memo1.Lines.add('Break signal detected...');
end;

procedure TfrmMain.Comm1Cts(Sender: TObject);
begin
  if VaComm1.CTS then
    StatusBar1.Panels[0].Text := 'CTS'
  else StatusBar1.Panels[0].Text := '';
end;

procedure TfrmMain.Comm1Dsr(Sender: TObject);
begin
  if VaComm1.DSR then
    StatusBar1.Panels[1].Text := 'DSR'
  else StatusBar1.Panels[1].Text := '';
end;
```

```
procedure TfrmMain.Comm1Ring(Sender: TObject);
begin
  if VaComm1.Ring then
    StatusBar1.Panels[2].Text := 'RING'
  else StatusBar1.Panels[2].Text := '';
end;

procedure TfrmMain.Comm1Rlsd(Sender: TObject);
begin
  if VaComm1.Rlsd then
    StatusBar1.Panels[3].Text := 'RLSD'
  else StatusBar1.Panels[3].Text := '';
end;

procedure TfrmMain.Comm1Error(Sender: TObject; Errors: Integer);
begin
  if (Errors and CE_BREAK > 0) then Memo1.Lines.add(sCE_BREAK);
  if (Errors and CE_DNS > 0) then Memo1.Lines.add(sCE_DNS);
  if (Errors and CE_FRAME > 0) then Memo1.Lines.add(sCE_FRAME);
  if (Errors and CE_IOE > 0) then Memo1.Lines.add(sCE_IOE);
  if (Errors and CE_MODE > 0) then Memo1.Lines.add(sCE_MODE);
  if (Errors and CE_OOP > 0) then Memo1.Lines.add(sCE_OOP);
  if (Errors and CE_OVERRUN > 0) then Memo1.Lines.add(sCE_OVERRUN);
  if (Errors and CE_PTO > 0) then Memo1.Lines.add(sCE_PTO);
  if (Errors and CE_RXOVER > 0) then Memo1.Lines.add(sCE_RXOVER);
  if (Errors and CE_RXPARITY > 0) then Memo1.Lines.add(sCE_RXPARITY);
  if (Errors and CE_TXFULL > 0) then Memo1.Lines.add(sCE_TXFULL);
end;

procedure TfrmMain.ComboPortNumChange(Sender: TObject);
begin
  try
    VaComm1.PortNum := ComboPortNum.ItemIndex + 1;
  except
    ComboPortNum.ItemIndex := VaComm1.PortNum - 1;
    raise;
  end;
end;

procedure TfrmMain.ComboBaudrateChange(Sender: TObject);
begin
  VaComm1.BaudRate := TVaBaudrate(ComboBaudrate.ItemIndex);
  Memo1.Lines.add('Baudrate: ' + ComboBaudrate.Text);
end;

procedure TfrmMain.ComboDatabitsChange(Sender: TObject);
begin
  VaComm1.Databits := TVaDataBits(ComboDatabits.ItemIndex);
  Memo1.Lines.add('Databits: ' + ComboDatabits.Text);
end;

procedure TfrmMain.ComboStopbitsChange(Sender: TObject);
begin
  VaComm1.StopBits := TVaStopBits(ComboStopbits.ItemIndex);
  Memo1.Lines.add('StopBits: ' + ComboStopbits.Text);
```

```
end;

procedure TfrmMain.ComboParityChange(Sender: TObject);
begin
  VaComm1.Parity := TVaParity(ComboParity.ItemIndex);
  Memo1.Lines.add('Parity: ' + ComboParity.Text);
end;

procedure TfrmMain.CheckBoxRTSClick(Sender: TObject);
begin
  VaComm1.SetRTSState(CheckBoxRTS.Checked);
end;

procedure TfrmMain.CheckBoxDTRClick(Sender: TObject);
begin
  VaComm1.SetDTRState(CheckBoxDTR.Checked);
end;

procedure TfrmMain.CheckBoxBREAKClick(Sender: TObject);
begin
  VaComm1.SetBREAKState(CheckBoxBREAK.Checked);
end;

procedure TfrmMain.CheckBoxXONClick(Sender: TObject);
begin
  VaComm1.SetXONState(CheckBoxXON.Checked);
end;

procedure TfrmMain.VaComm1Data(Sender: TObject; Count: Integer);
var

  GetChar: boolean;
  C: char;
  CountErr: integer;
  Hs: string; //hex string

  procedure gotMessage(twoMsg: boolean);
  var
     mCount: Integer;
  begin
      // Count is bytes in buffer, from Vacomm event
       mCount := Length(RxMsg);

       CountErr := getCompleteMsg(RxMsg);
       if (CountErr = 0)then
       begin     // if last part of message received and it is valid
          if (DispCount >0) then
          Memo1.Lines.add('RxBuffer ' + IntToStr(Count) + ' bytes | '
               + 'MsgLength=' + IntToStr(mCount));

          Memo2.Lines.Text := Memo2.Lines.Text + RxMsg;
          if (Length(Raddress)>0) then
          begin    // if it's a reponse to ND  (node discovery)
             // Raddress and Rname will store the node information
             Memo4.Lines.add(Raddress);
```

```
            Raddress :='';
            ComboBox1.Items.Add(Rname);
            Rname :='';
         end;
         RxMsg := ''; // clear the message after processed
      end
      else  // an error in the message
      begin
         Memo1.Lines.add('RxBuffer ' + IntToStr(Count) + ' bytes | '
              + 'MsgCount=' + IntToStr(mCount) + ' Err='
              + IntToStr(CountErr));
         // If CountErr < 0, may be need to wait for some more bytes
         //  RxMsg will keep as is and continue to receive next char
         ErrBytes := ErrBytes + CountErr;
         EdtCountErrMsg.Text := IntToStr(ErrBytes);
         if ((CountErr > 0) or (twoMsg)) then //discard this message
           RxMsg:='';
         if (DispCount = 0) then
            DispCount := 2; // display 3 message when error occurs
      end;
  end;

begin
    CountBytes:= CountBytes + count;
            // total bytes received by serial port
    EdtTotalRxBytes.Text := IntToStr(CountBytes);
  repeat
    GetChar := Vacomm1.ReadChar(C);
    if (GetChar) then
       begin
       // check byte and count number of bytes if 0..9
            if (C = 's') then
            begin  // start counting for one message
                StartOneMsg := true;
                CountOneMsg :=0;
                inc(Count5);
            end;
            if StartOneMsg then
               case C of
                  '0'..'9': CountOneMsg := CountOneMsg + 1;
                  'E': begin // end of message
                        CountErr := StrToInt(EdtAccErrBytes.Text);
                        StartOneMsg := false;
                        if (CountOneMsg <> 300) then
                          begin
                            CountErr := CountErr
                                        + abs(CountOneMsg - 300);
                            EdtAccErrBytes.Text :=IntToStr(CountErr);
                          end;
                       end;
               end;

       if (C = char($0D)) then
       //Clear Memo2 if all bytes received without error
          begin
```

```
              CountErr := StrToInt(EdtAccErrBytes.Text);
              Count5 := 0;
                 if (CountErr = 0) then
                   Memo2.Lines.Clear;
               // If memos not clear, will slow down serial
                // port and stuff receiving buffer.
          end;

       if (C = char($7E)) then
          if (Length(RxMsg)>0) then // already got one message
          begin
             gotMessage(true); //second message = true
          end;
          RxMsg := RxMsg + C;  // buffer the received message
          Hs := Hs+ IntToHex(Integer(C),2);
       end;
  Until (not GetChar);
      // All char in buffer is received
      // display all buffered data in HEX code
        Memo3.Lines.Text := Memo3.Lines.Text + Hs +#13;

    // after all characters received from the serial port
    if ((Length(RxMsg)>0) and (RxMsg[1]=char($7E)))
    then gotMessage(false); //false for one message only

end;

procedure TfrmMain.VaComm1Event(Sender: TObject);
begin
  Memo1.Lines.add('Event signal detected...');
end;

procedure TfrmMain.VaComm1Open(Sender: TObject);
begin
  Memo1.Lines.add('Port open');
end;

procedure TfrmMain.VaComm1Close(Sender: TObject);
begin
  Memo1.Lines.Add('Port closed');
end;


procedure TfrmMain.btnSendCmdClick(Sender: TObject);
var
  S, Hs: String;
  I: Integer;
begin
  S := getCommand(char($08) + 'R' + edtCmd.Text);
  Vacomm1.writeText(S);
  I := 1;
  Hs:='';
  while I <= Length(S) do
    begin
       Hs := Hs+ IntToHex(Integer(S[I]),2);
```

```
      I := I + 1;
    end;
  Memo2.Lines.Text := Memo2.Lines.Text+ char($0D)+'TX: ' + Hs +'... ';
  if (UpperCase(edtCmd.Text) = 'ND') then
  begin
    Memo4.Lines.Clear;
    ComboBox1.Items.Clear;
    ComboBox1.Items.Add('Broadcast');
  end;
end;

procedure TfrmMain.btnSendMSGClick(Sender: TObject);
var
  S, Hs, Addr: String;
  I: Integer;
begin

  Addr := Memo4.Lines[0];
  if (edtAddress.Text='Broadcast') then
  begin
    S := getCommand(
       char($10)                //ZigBee Transmit Request
       + char($00)              // no ack
       + char($00)+ char($00)   // 64 bit destination address
       + char($00)+ char($00)   //     0x0000 0000 0000 FFFF
       + char($00)+ char($00)   //      for broadcast
       + char($FF)+ char($FF)
       + char($FF)+ char($FE)   // 16 bit Destination Network address
                                //    0xFFFE for Broadcast or Unknown
       + char($00)+ char($00)   // Max Hop + No options
       + edtCmd.Text            // the message (Max 72 bytes)
       );
  end
  else
  begin
     S := getCommand(
       char($10)                //ZigBee Transmit Request
       + char($00)              // no ack
       +  getAddress(edtAddress.Text) //  edtAddress.Text //
       + char($FF)+ char($FE)   // 16 bit Destination Network address
                                //    0xFFFE for Broadcast or Unknown
       + char($00)+ char($00)   // Max Hop + No options
       + edtCmd.Text            // the message (Max 72 bytes)
       );
   end;
  Vacomm1.writeText(S);
  I := 1;
  Hs:='';
  while I <= Length(S) do
    begin
       Hs := Hs+ IntToHex(Integer(S[I]),2);
       I := I + 1;
    end;
  Memo3.Lines.Text := Memo3.Lines.Text+ char($0D)+'TX: ' + Hs +'... ';
end;
```

```
procedure TfrmMain.ComboBox1Click(Sender: TObject);
begin
        edtAddress.Text := ComboBox1.Items[ComboBox1.ItemIndex];
end;

procedure TfrmMain.Label5DblClick(Sender: TObject);
begin
        ShowMessage(HelpMessage);
end;

end.   // end of form main unit
```

B3. Unit file of API functions

```
unit APIcommands;

interface

uses
  SysUtils;

function findChkSum(cmd: String): char;
function getCommand(cmd: String): String;
function getCompleteMsg(var Msg: String): Integer;
function getAddress(NI: String): String;

const

NJcmd: String = char($08)+ 'RNJ';
NDcmd: String = char($08)+ 'RND';
APIdelimiter: char = char($7E);
HelpMessage = 'The comport source codes of this program is based on'
      + #13 + 'Async32demo (c) Varian Software Services nl 1996-2000.'
      + #13 + 'ZigBee firmware is MaxStream v8x17 Beta';

var
  Raddress: String ='';
  Rname: String ='';
implementation

function getAddress(NI: String): String;
begin
if (NI='ROUTER4') then
Result:= (char($00) + char($13)  + char($A2)
        + char($00) + char($40)  + char($01) +char($84) + char($EF) )
        //address of ROUTER4
else if (NI='ROUTER1') then
 Result:= (char($00) + char($13)  + char($A2)
        + char($00)  + char($40)  + char($01) +char($84) + char($F2) )
        //address of ROUTER1
```

```
else if (NI='ROUTER3') then
 Result:= (char($00) + char($13)  + char($A2)
        + char($00)  + char($40)  + char($06) +char($05) + char($79) )
        //address of ROUTER3
else if (NI='ROUTER2') then
 Result:= (char($00) + char($13)  + char($A2)
        + char($00)  + char($40)  + char($08) +char($AA) + char($FC) )
        //address of ROUTER3
    else Result:= 'NIL';
end;

function findChkSum(cmd: String): char;
var
  I: Integer;
  Sum : Integer;
begin
  I := 1; Sum := 0;
  while I <= Length(cmd) do
  begin
    Sum := Sum + Integer(cmd[I]);
    I := I+1;
  end;
  I := 255 - Sum;
  Result := char(I);

end;

function getCommand(cmd: String): String;
var
  cmdMid: String;
begin
  cmdMid :='';
  if (Length(cmd)>$FF) then
    cmdMid := char(Length(cmd)-$FF) + char(Length(cmd))
  else
    cmdMid := char(00) + char(Length(cmd));
  Result := APIdelimiter + cmdMid + cmd + findChkSum(cmd);
end;

function chkATresponse(Msg: String): String;
var
  I: Integer;
  S: String;
begin
  if (Copy(Msg,6,2) = 'NI') then
    Result := Copy(Msg,9,Length(Msg)-9)
  else if (Copy(Msg,6,2) = 'ND') then
          // each node will response with one message
    begin  // extract node name
       I:= 7;
       if (Length(Msg) > 19) then
```

```
       begin
          repeat
             I := I+1;
             S := S+IntToHex(Integer(Msg[I]),2);
          until (I=18);
          Raddress:= S;
          Rname := Copy(Msg,19,Length(Msg)-9);
          Result := 'Node:' + Rname  +':';
       end
       else
          Result := '';
    end
  else
    Result := 'Not identified';
end;

function getCompleteMsg(var Msg: String ): Integer;
var
  L: Integer;
begin
  L := Integer(Msg[2])*256 + Integer(Msg[3]);
  if (Length(Msg) = (L+4)) then
         // completely received & one message only
  begin
    if (Msg[4] = char($90)) then        // ZigBee Message
       Msg := Copy(Msg,16,Length(Msg)-16)   // Extract the content
    Else if (Msg[4] = char($88)) then   // AT command response
       Msg := chkATresponse(Msg)       //Copy(Msg,9,Length(Msg)-9)
    Else
       Msg :='Not a proper message!';
    Result := 0;
  end
  else  // RxMsg does not got alter, but err bytes returned
  begin
     Result := Length(Msg)- (L+4)
  end;
end;

end.  // end of APIcommands unit
```

********* End of Thesis *********