LittleBot

by

James McVay

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Master of Engineering
Victoria University of Wellington
2014

Abstract

Robots to assist in USAR (urban search and rescue) situations have been employed since 2001. Such robots are designed to provide video and sensor feedback to evaluate hazardous environments before human task-forces are sent in. This minimises the risks human personnel are exposed to, while increasing the effectiveness of USAR operations. However, the typically high cost of such robots and the reliance on trained operators puts them out of reach of most USAR teams. In New Zealand, there are no nationally available robots suitable for USAR purposes. This thesis explores the development of new affordable devices that can be deployed for USAR operations, known as LittleBots.

Three LittleBot variants are developed. Differing primarily in their locomotive capability, two mobile variants provide tether-less video reconnaissance and selectable gas level readings. The third, stationary variant, may be reconfigured with up to four selectable sensors, and is targeted at providing ongoing environmental monitoring at a disaster site. With all variants costing less than USD \$155 in components, LittleBots are sufficiently low cost to be considered disposable, greatly increasing the likelihood they will be employed en masse. The stationary Sentry variant demonstrates a minimum runtime of over 60 hours, while the mobile variants provision up to 6 hours of mobile video reconnaissance. For independent deployment of LittleBots, a compatible Controller device is developed. Through user testing, the Controller device demonstrates easy and intuitive use, with no training required.

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Professor Dale Carnegie, for his constructive criticism, enthusiastic encouragement, and unwavering patient guidance.

I would also like to extend my thanks to Tim Exley, Sean Anderson, and Jason Edwards, for their technical assistance and support throughout the year.

Finally, I wish to thank my family and friends for their support and encouragement throughout my study.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Objectives	3
1.3	Thesis Overview	4
2	Background	7
2.1	Search and Rescue Robots	7
2.2	Wireless Video and Networking	12
2.3	Sensor Platforms	15
3	LittleBot	19
3.1	Overview	19
3.1.1	Communications	19
3.1.2	Test Platform	21
3.1.3	Proof-of-Concept	22
3.1.4	Variants	23
3.2	Mechanical Design	25
3.2.1	Locomotion	26
3.2.2	Component Mounting	30
4	Electronics	43
4.1	Overview	43
4.2	Core	45

4.2.1	ATMega328 Microcontroller	45
4.2.2	XBee Pro 900 HP	47
4.3	Video Feedback	48
4.3.1	Camera	48
4.3.2	Transmitter	50
4.3.3	Constant Current Driver	53
4.4	Motor Drivers	54
4.5	Sensors	57
4.5.1	Gas	57
4.5.2	Temperature & Humidity	59
4.5.3	Motion	60
4.5.4	GPS	61
4.6	Power Supply	62
4.6.1	Required Battery Capacities	63
4.6.2	Lithium Polymer Cells	65
4.6.3	DC-DC Conversion	68
4.6.4	Battery Monitoring	71
4.6.5	Charging	73
5	Controller	75
5.1	Mechanical Construction	75
5.1.1	Frame & Construction Method	76
5.1.2	Video Display & PCB Mounting	79
5.1.3	OLED Display	81
5.1.4	Joystick	82
5.1.5	Buttons & Labels	83
5.2	Electronics	85
5.2.1	ATMega2560 Core & XBee	86
5.2.2	SD Card	87
5.2.3	Real Time Clock	88
5.2.4	User Input	89

5.2.5	OLED Display	92
5.2.6	Video Receiver & Display	92
5.2.7	Power Supply	96
5.3	Summary	98
6	XBee Communication	101
6.1	LittleBot's Protocol & Packets	101
6.1.1	Controller Association	102
6.1.2	Command, Control and Feedback	104
6.2	Firmware Configuration	108
6.3	API Implementation	110
7	LittleBot Software	113
7.1	Overview	113
7.2	Initialization & Association	115
7.3	Receiving Packets	116
7.4	Transmitting Packets	119
7.4.1	Battery Level	121
7.4.2	Temperature & Humidity	122
7.4.3	Gas Level	123
7.4.4	GPIO	124
7.4.5	GPS	125
8	Controller Software	127
8.1	Overview	127
8.2	Initialisation	128
8.3	Input Handling	130
8.4	Packet Receipt	133
8.5	Data Logging	135
8.6	LittleBot Control	136
8.7	Information Display	138

9	Evaluation and Analysis	143
9.1	Battery Performance	143
9.1.1	Runtime	143
9.1.2	Charging	149
9.2	Ingress Protection	150
9.3	Locomotion	153
9.3.1	Speed	153
9.3.2	Incline	155
9.3.3	Terrain	156
9.4	Wireless Range	157
9.4.1	Command and Control	157
9.4.2	Video	159
9.5	System Test	161
9.6	Ease of Use	162
9.7	Cost	164
9.8	Summary	166
10	Conclusions	167
10.1	Review	167
10.2	Future Work	170
10.3	Summary	171

List of Figures

2.1	MOTHERSHIP mounted in a test bed	9
2.2	MorpHex walking and rolling robot	10
2.3	ReconRobotics' Throwbot XT	11
3.1	2-Wheel test platform	22
3.2	3D printed proof-of-concept	23
3.3	Junior LittleBot	24
3.4	Standard LittleBot	24
3.5	Sentry LittleBot	25
3.6	DC motor and 100:1 gearbox	27
3.7	Exploded view of a mobile LittleBot's transaxle	27
3.8	Proposed wheel types	28
3.9	Junior LittleBot's tail	29
3.10	Standard LittleBot's battery mounting	30
3.11	Junior LittleBot's battery mounting	31
3.12	O-ring sealing the chassis cover	32
3.13	Covers sealing the camera and LEDs	33
3.14	Bracket securing the camera	33
3.15	Gas sensor mounted on the Standard LittleBot	34
3.16	O-Ring placed in the gas sensor mount	35
3.17	O-ring sealing around the PIR lens	36
3.18	DHT22 on a Standard LittleBot	37
3.19	Combination jack and power indicator	38

3.20	RP-SMA connectors on a Standard LittleBot	39
3.21	RP-SMA connector on a Sentry LittleBot	40
4.1	Standard LittleBot PCB	44
4.2	Sentry LittleBot PCB	44
4.3	Schematic of the implemented ATmega328	47
4.4	FH10C V9 PAL video camera	49
4.5	Camera power control using an N-type MOSFET	49
4.6	TX5823 module mounted on a LittleBot PCB	50
4.7	Schematic of the TX5823 implementation	52
4.8	Schematic of the ZXLD1350 implementation	53
4.9	Schematic of the DRV8835 for a Junior LittleBot	54
4.10	Schematic of the DRV8835s for a Standard LittleBot	55
4.11	Two DRV8835s in low profile sockets	56
4.12	Schematic of the gas sensor module	58
4.13	Parallax PIR sensor	60
4.14	Schematic of the Seiko S8232 implementation	66
4.15	Battery protection PCB	67
4.16	Schematic of the PTH08080 for 3.3 V	69
4.17	Schematic of the PTH08080 for 5 V	71
4.18	LittleBot's 1.2 A charger	73
5.1	Partially de-constructed controller frame	76
5.2	Combination jack and antenna connectors	77
5.3	SD card access	78
5.4	Components mounted to the front plate	78
5.5	Video LCD with mounting frame	79
5.6	Stacked PCB mounting	80
5.7	OLED mounted to the front plate	81
5.8	Thumbstick installed on its mounting PCB	82
5.9	Thumbstick behind the front plate	83
5.10	Control buttons with labels	84

5.11	Rear of the Controller PCB	85
5.12	DS3231 RTC implemented with battery backup	89
5.13	Buttons connectors with decoupling	91
5.14	OLED connections	92
5.15	KYV-N2 display driver PCB	93
5.16	Schematic of the video electronics	95
5.17	Controller power supply	97
5.18	Controller battery input and measurement	98
5.19	The LittleBot Controller	99
6.1	XBee UART to USB adaptor	109
6.2	X-CTU configuration utility	110
7.1	Overview of the LittleBot software execution	114
7.2	LittleBot packet reception process	117
7.3	LittleBot packet transmission process	120
8.1	Overview of the Controller software	127
8.2	Overview of <i>handleButtons</i>	131
8.3	Overview of <i>handleIncoming</i>	134
8.4	Overview of <i>controlUnit</i>	137
8.5	Overview of <i>updateDisplay</i>	139
8.6	Inverted title on the OLED display	140
8.7	OLED display formats for various statuses	141
9.1	Mobile LittleBot operating times while stationary	144
9.2	Mobile LittleBot operating times while driving	146
9.3	Sentry LittleBot operating times	147
9.4	Controller operating times	148
9.5	Charging times	149
9.6	Internal DHT22	151
9.7	DHT22 readings during submersion	152
9.8	Proposed wheel types	153

9.9	LittleBot speeds with full and empty batteries	154
9.10	Maximum climbable inclines	155
9.11	Noise on the 3.3 V supply	158
10.1	The LittleBot system	172

List of Tables

4.1	Mapping of TX5823 control pins to transmission frequency .	51
4.2	Power consumption of Standard LittleBot components . . .	63
4.3	Power consumption of Sentry LittleBot components	64
5.1	Power consumption of the LittleBot Controller components	96
9.1	User study results	163
9.2	LittleBot component cost breakdown	165

List of Structures

6.1	<i>assocRequest</i> packet	103
6.2	<i>standardResponse</i> packet	105
6.3	<i>sentryResponse</i> packet	106
6.4	<i>standardControl</i> packet	107
6.5	<i>sentryResponse</i> packet	108

Chapter 1

Introduction

Urban search and rescue, or USAR, involves finding and rescuing people who become trapped in an urban environment. Examples include victims trapped in a building collapse as a result of a major landslide or earthquake, or in a building that is on fire. While not urban in nature, the technology and services can also cover people trapped in mines or in mine-shafts. As part of the specialist equipment, robots specifically designed for USAR operations can be a valuable asset, as mobile reconnaissance devices can be deployed to evaluate unstable or potentially hazardous environments to assess a situation before people are sent in.

1.1 Motivation

In New Zealand, USAR primarily comes under the umbrella of the New Zealand Fire Service, with current USAR task forces including firefighters, search and rescue dogs and handlers, communications experts, engineers, paramedics, doctors, and other technical experts [1]. These task forces routinely call on external support and expertise as required, in areas such as construction, heavy machinery operation and specialist equipment. During the first response in an USAR situation, task forces must proceed slowly and with care so as to not put themselves at risk [2]. In

many situations voids and crevices may not be penetrable by dogs, making purpose-built small mobile robots that can get into these crevices an appealing alternative. Assuming these mobile robots are inexpensive and their possible destruction acceptable, they can be sent into dangerous situations where a human or dog would not normally be sent [3].

Nationally there are no robots that are designed specifically to assist in USAR operations. Of those capable of reconnaissance with video feedback, few incorporate moisture ingress protection into their design. Perhaps the most prominent example of the problem resulting from this unavailability was during the 2010 Pike River Coal mine disaster where an army bomb disposal robot was ineffectively deployed for USAR purposes [4]. When moisture ingress rendered the robot inoperable, specialist robots had to be flown in from Australia [5]. In such a situation, there is a substantial time delay for devices and expertise to be shipped in from another country. As the chances of finding trapped survivors decreases over time, this is a significant factor to consider for more lives to be saved.

The Mechatronics Group at Victoria University of Wellington has been developing a three-tier hierarchical system for USAR deployment [6]. The system consists of three levels of robots; the grandmother, mother, and daughter. The grandmother and mother are intended for command and control, while the daughter robots are to serve as sensor platforms for reconnaissance in small crevices, collecting and transmitting data. However, a viable low cost and potentially disposable daughter has yet to be successfully developed. Such daughter robots could not only provide reconnaissance and search, but also ongoing minimisation of risk to USAR task force personnel through monitoring of environmental conditions, such as air quality and temperature.

By reducing the time that it takes to find trapped victims, a new device that targets the objectives proposed in Section 1.2 could result in more efficient and successful rescue operations. Through prior reconnaissance and hazard identification using both video feedback and gas sensor de-

tection, such devices could reduce the risks USAR teams are exposed to. This project seeks to develop such a low cost, small robot, that can be either manually controlled by a human operator or form the lowest tier of an autonomous robotic USAR system.

1.2 Objectives

The outcome of this thesis is a new low cost robot prototype, LittleBot. This section details the specific objectives it must achieve in its design.

First, a LittleBot must be sufficiently inexpensive to be considered disposable. It should also be capable of rapid deployment over long distances and be easy to use by even an untrained operator. As a result the design must permit a low production cost, and tether-less operation through an independent Controller that is intuitive to use. Video feedback and gas sensors are incorporated to allow reconnaissance and hazard identification by LittleBot, facilitating safe path planning for the recovery of trapped victims and reducing the risks that human rescue task forces are exposed to.

To address the issues previously identified where robots have been unsuccessfully deployed due to debilitating moisture ingress, an IP55 ingress protection rating is sought. While this does not decree a completely water-tight device, it stipulates that the ingress of dust cannot enter in sufficient quantities to interfere with the operation of the device, and that a water jet against the external enclosure of the device has no harmful effect. Although not designed to be completely submerged in water for extended periods of time, a LittleBot should competently operate in damp environments and be unaffected by temporary submersion.

With LittleBot likely to be used in areas composed of rubble or within collapsed buildings, it should be of a compact size. This is to allow it to get into crevices and small, confined spaces that would otherwise be near impossible for current USAR robots to enter. Finally as a LittleBot will be

tether-less, a power source will be included as part of its design. In order for LittleBot to be of value to USAR operations for the reconnaissance and exploration purposes, this power source should be capable of providing at least 120 minutes of runtime. If LittleBot cannot do a complete reconnaissance in this time, then it is likely to have become stuck.

In summary, a LittleBot must:

- Be low cost to produce (<\$125 USD)
- Provide tether-less operation
- Provide video feedback
- Incorporate a variety of gas sensors
- Incorporate IP55 ingress protection
- Be compact in size
- Provide a 2 hour runtime
- Be intuitive to use

The first six of these objectives apply directly to the design of LittleBot discussed in Chapters 3 and 4. The final intuitive use criterion which applies to the design and operation of the Controller for independent operation is discussed in Chapters 5 and 8.

1.3 Thesis Overview

This chapter has introduced the motivations for the project, detailing specific objectives in Section 1.2. Chapter 2 explores and discusses related works both for their merits that may be included, and drawbacks that should be avoided in the design of LittleBot. Chapters 3 and 4 detail the

approach taken in designing and producing LittleBot to meet the objectives discussed in Section 1.2, while Chapter 5 presents the design and production of the intuitive Controller device enabling independent use of LittleBots for USAR purposes. Chapter 6 explains the configuration of the XBees to set up the DigiMesh network facilitating tether-less operation and details the LittleBot protocol. In addition, Chapters 7 and 8 detail the software and accompanying libraries written for the LittleBots and Controller for their desired operation. Chapter 9 evaluates the resultant system against the original objectives, and finally Chapter 10 concludes the successful work conducted and presents future avenues for LittleBot development.

Chapter 2

Background

Introduced in Chapter 1, the motivation of this project is to develop a low cost, sensor equipped mobile robot called LittleBot. LittleBot will be independently controllable through manual deployment, and can also serve as the lowest tier of an autonomous USAR hierarchy [6]. As wireless communication is essential for the tether-less operation objective, a novel approach to maximise operating range while adding robustness and reducing the cost is explored. Additionally the creation of a remote operating tether-less sensor platform also presents the opportunity for other applications of these devices as explored in Section 2.3. This chapter presents the background and prior works in these areas.

2.1 Search and Rescue Robots

Over the past two decades, various robots for reconnaissance [7, 8] and urban search and rescue have been developed in a variety of university environments [9, 10, 11, 12, 13]. However the first use of robots in an urban search and rescue situation did not occur until September 12, 2001 at Ground Zero, following the collapse of the WTC (World Trade Centre) [14]. During this event, five independent teams deployed a total of seven robots of varying designs and configurations, some tethered and

some tether-less. While no survivors were discovered using the robots, they did discover 10 victims. This accounted for more than 2% of all the victims found. Robin Murphy, a pioneer in the field of robots for urban search and rescue and who led the team at ground zero envisioned that “robots will soon become like search dogs in the minds of the rescue community: essential” [15]. However well over a decade since the first use of robots in a USAR operation, they have yet to become an integral part of any USAR team worldwide.

As the robots deployed following the WTC collapse were multi purpose devices, they were typically quite large and the smallest of these devices had a footprint of $6 \times 16 \times 31$ cm [16]. The use of the devices was typically difficult and unintuitive, requiring training to operate the devices at even the most basic of levels. Furthermore, the devices were expensive with the cheapest tether-less robot employed costing over USD \$33,000, to the most expensive exceeding USD \$100,000. Even if a typical USAR team could afford to possess one of the more expensive robots and train members to operate it safely, the high replacement cost of the device would heavily influence deployment decisions. This is due to the risk of it becoming lost or damaged in the unstable or hazardous environments in which it is likely to be employed.

In addition to independent robots, hierarchical robotic systems for reconnaissance and surveillance purposes have been explored since 2000 [17]. Constituting a two-tier system in this example, larger Ranger robots deployed the small Scout robots to carry out reconnaissance. Similarly, the MOTHERSHIP and CRAWLER system [18] employs a two tier system for the purposes of greater penetration through a better equipped MOTHERSHIP. Research groups have been exploiting this hierarchical concept so that expensive control units are kept safe, and cheaper units with varying capabilities deployed to penetrate a disaster zone. Mentioned previously, VUW’s Mechatronics Group pioneered a three-tier hierarchical USAR robotic system in 2007 [6], with a daughter for the system yet to

be realised.

The device developed as part of this project is intended for both independent use and incorporation into the hierarchical system. With a low cost design targeted, the method of locomotion must be considered. The MOTHERSHIP focuses on a system of locomotion enabling greater obstacle penetration into a disaster site. Following an initial infiltration into a disaster site, the MOTHERSHIP shown in Figure 2.1, deploys rubble-penetrating CRAWLER robots capable of tele-operated manual control with video feedback.

Figure 2.1: MOTHERSHIP mounted in a test bed

While this approach has the potential for greater penetration, the MOTHERSHIP's locomotive design may cause it to become stuck vertically when traversing a rubble field. Furthermore, tracked systems are more expensive than their wheel counter parts due to an increased component count. As a result, the complex combination of serpentine design and caterpillar tracks are likely to increase the cost of the MOTHERSHIP (although the author has been unable to determine their cost). In addition to the typical wheeled and tracked designs, walking and rolling robots have become increasingly prominent in recent years, varying in design from biped to spider-like with six or more legs, balls, or a combination of both such as the MorpHex [19] shown in Figure 2.2.

Figure 2.2: MorpHex walking and rolling robot

Although these designs may be advantageous in terms of their terrain climbing capabilities, they are more complex both in mechanical design and control. With the devices proposed as part of this project being small in size, legged locomotion would be difficult to implement at such a small scale. Furthermore the component count would again be increased, and combined with added complexity in achieving ingress protection within the legs, would be difficult to effectively implement at a low cost.

Reverting to simpler locomotion designs, the Scouts of the two-tier system [17] employ a two-wheel system with spring loaded tail on which the device rests on. In addition to low cost associated with the reduced component count of wheeled designs, ingress protection of the locomotion is easier to achieve in comparison to the legged robots due to the reduced number of pivot points that must be sealed. Similarly to the Scouts, the commercial Throwbots from ReconRobotics employs the same two-wheel with tail design, demonstrating the effectiveness of this design approach. Shown in Figure 2.3, the Throwbot XTs are water resistant to incidental immersion up to 30 cm for five minutes, making them ideal for a USAR team as part of their inventory. Such devices would unlikely succumb to the moisture ingress problems highlighted during a recent national disas-

ter discussed in Chapter 1. Moreover the Throwbot is controlled using a joystick with video feedback, making it easy to use.

Figure 2.3: ReconRobotics' Throwbot XT

Although there are cheaper Throwbot models such as the Recon Scout LE starting at USD \$4875 [20], the Throwbot XT is the cheapest model that incorporates the desirable ingress protection. However costing USD \$13,000 [21], Throwbot XTs are not sufficiently inexpensive to be considered disposable, and placed out of reach of USAR operations due to their unjustifiably high cost.

With the largest pitfall of existing systems still being their high cost that makes them unavailable to USAR teams, a significantly simplified design using off-the-shelf components is selected in order to reduce the cost of the devices to a point where they can be considered disposable. To reduce component count and hence cost, a two-wheel approach without a tail but instead self-balancing is initially proposed. To mitigate the identified issues encountered during recent national disasters, ingress protection is incorporated into the device as required, with sealing of a wheeled locomotion design considered relatively straightforward. By emphasising the combination of these concepts throughout design, a low cost, robotic search and rescue platform is presented in Chapter 3.

2.2 Wireless Video and Networking

One of the greatest complications in the development and deployment of the type of robotic devices discussed is communication. For the reasons discussed in Section 1.2 the devices must be tether-less, necessitating a form of wireless communication. Typically to achieve a reliable communications link over a reasonable distance, large antennas and high power transmitters are employed, which consumes a considerable amount of power. Furthermore to transmit video wireless in a digital fashion, a considerable amount of bandwidth is required.

Existing tether-less robotic platforms typically employ proprietary low frequency RF communications for command and control, and where fitted, video feedback transmitted separately [8]. While simple, this one to one approach limits the range between a controller or base node and the robotic platform. An alternative to this approach is to use a mesh network and leverage other nodes that may be present as part of the system. By allowing data to hop between nodes, mesh networking provides a novel way to increase the maximum range by routing data through intermediate nodes [22]. Furthermore, reliability is increased through self healing, that is the ability to create alternate paths when one node fails or a connection is lost [23]. Currently 802.15.4 is the standard commonly employed in creating wireless mesh networks.

Since 2005 Digi International have been manufacturing XBee modules for communication between electronics devices by implementing the 802.15.4 standard, with the modules successfully employed in a variety of situations from wireless energy monitoring systems [24] to heart rate monitors [25]. In addition to supporting the 802.15.4 standard used in these applications, XBees support an alternative proprietary protocol coined DigiMesh. While 802.15.4 is a recognised standard in wireless mesh networking, DigiMesh creates a more robust network as there are none of the parent-child dependencies that are present in 802.15.4 mesh networks,

and all nodes play an equal role in configuration and routing. Although XBee modules have been used to control a robotic platform [26], neither the 802.15.4 nor DigiMesh mesh functionalities were employed for command and control.

In 2012 Digi International released a new product called the XBee-PRO 900HP featuring a revised version of their DigiMesh protocol for communication between the modules on frequencies in the 900 MHz band. Capable of direct line-of-sight communication between any two modules at up to 28 km with a 200 kbps RF data rate, DigiMesh with additional intermediate nodes allows this range to be further extended by forming the self-healing self-discovering mesh network described. When powered, a DigiMesh enabled XBee-PRO 900HP module will attempt to join the network it is configured for, with all nodes in a DigiMesh network intelligently updating routes and paths between nodes for robustness. When a node is instructed to deliver a packet to another node, identified by its hardware MAC address, the sending node will query the network to discover a path to that node as long as it is within range of one other node in that network. Other nodes selected as part of the path to the target node will then forward the packet to this node using the self-discovered route, and a delivery acknowledgement forwarded back to the sending node. With command and control packets for a robot typically of a few kilobytes in size, the DigiMesh network provided by the XBee modules easily suffices in providing the required wireless control. Additionally, control of a device directly out of range of a controller is possible through automatic routing without added complication.

While a mesh network provided by the XBee modules is capable of facilitating command and control, video feedback is more complex. Fundamentally the data rate of uncompressed digital video can be found by $D = C \times V \times H \times F$ where D is the data rate in bits per second, C is the colour depth, V is the vertical resolution, H is the horizontal resolution and F is the refresh rate. A video feed similar to DVD with a resolution of

640 × 480 pixels at 25 frames per second with a colour depth of 24 bits per pixel would have a bit rate of $D = 24 \times 640 \times 480 \times 25 = 184320000$ bps or 184.3 Mbps. Therefore a wireless system transporting this video must provide an absolute minimum bandwidth of 184.3 Mbps, excluding any over-heads included as part of the system selected. Furthermore as real time video is required (to facilitate real time control of a mobile device), buffering of the video stream is not possible as this would introduce an undesirable delay.

The data rate calculated for the desirable quality of video not only exceeds the 200 kbps maximum provided by the XBee modules, but also the effective data rates of even the fastest of alternative wireless solutions such as 802.11n [27]. Compression using a standard such as H.264 could be used to sufficiently reduce the bit rate to be feasibly transmitted over a wireless medium using standards such as 802.11n, WiMax or 3G, with optimisations applied to reduce the delay inherently included with compression [28]. However such an approach would result in an overly complex and costly design for the target robotic platforms. As cost is of great concern, a digital implementation for the video feedback is therefore unappealing. Instead, a design similar to the works of [7] and [8] is more suitable, employing a supplementary high power analogue transmitter employed to achieve video feedback independent of the wireless system for control.

The proposed solution to the control and video feed back challenges on a mobile robotic device employs a combination of the XBee-PRO 900HP with DigiMesh and a high power analogue transmitter. The DigiMesh protocol forms an integral part of the extended range operation by providing a simplified mesh based network for command and control, as presented in Chapter 3. Complementing the command and control, analogue video transmission on a frequency substantially different to the XBee provides limited range video feedback. While analogue video does not guarantee the same level of quality as the digital equivalent, its lower implementa-

tion simplicity and cost make it preferable for this application.

2.3 Sensor Platforms

The main motivation behind this thesis is to develop an inexpensive, mobile robotic sensor platform to provide reconnaissance during USAR events. While the most valuable outcome anticipated by the proposed device is the location of a trapped human victim in some disaster scenario, other opportunities are apparent. By slightly extending the design parameters of the platform and making the sensors easily interchangeable, a far greater range of applicability may be realised. For example, the platform could be configured and deployed singularly to detect hazardous gas in contained environments, or potential dangers awaiting law enforcement officers as they investigate a suspected methamphetamine lab. Extending this, the inexpensive nature of the proposed devices already facilitates the application of a large number of these devices. While this has already been advocated as being extremely useful in USAR applications where a larger number of devices could increase the speed and likelihood of detection of a trapped human, it also provides the ability to cover a large area for wide-scale sensing and monitoring.

A large number of devices could be employed in security applications, distributed mapping applications, and even wildlife detection. Certainly non-mobile devices have been designed and deployed for applications such as measurement of water temperatures over a large area [29], monitoring of air temperature and gas levels [30] and have been advocated for deployment in the upper atmosphere in order to map wind currents. However, the proposed devices are dissimilar to those presented in these examples in that they are mobile, meaning that precise initial positioning of the devices is not essential. Similarly should the environment or tracking target move or be changed, the mobile devices are capable of repositioning without direct human intervention to continue their intended pur-

pose with minimal disruption.

The concept proposed here to extend the devices as sensor platforms for large-scale deployment is not entirely new. Certainly mobile robots with modular sensors exist [7], however there are no known devices that combine sensor reconfigurability with any form of ingress protection and very low production costs. Swarm robotics have envisioned large scale deployments for the purposes of spill finding [31, 32], exploration [33], and odour plume tracking [34] using mobile platforms. While these mobile swarms have been practically realised with varying degrees of success, they offer limited applications due to their design that results in only a single configuration for each and are again limited in terms of ingress protection and locomotion.

When used in hazardous environments such as active volcanoes, gaseous or unstable mines, landslides or earthquakes, the benefits of a disposable device are obvious. Previous attempts to create sensor platforms include the Arkas (applied for heavy snowfall observation) [35], sensor platforms for underwater monitoring [36], and the Dante and Dante II robots for exploring volcanoes. Designed to explore Mount Erabus [37], the Dante II project cost over USD \$1.7 million and was lost during an unsuccessful extraction attempt when the umbilical cord connecting the device to the extraction helicopter snapped [38]. Clearly this is a huge loss to sustain, both financially and in development time, making cheaper devices such as that proposed in this project an appealing alternative. By deploying a large number of these small devices as mobile sensors in a variety of configurations, the same tasks may be achieved with lower costs while making the loss of the devices inconsequential. The concept of smaller nodes for volcanic monitoring has been explored with limited success [39], but the networking method in this example resulted in a network uptime of only 61% and all node data was lost during down time.

With no known references to inexpensive mobile sensing platforms, that is one of the novel contributions that this project will endeavour to

create. Hence in addition to the original USAR motivated purposes of the platform developed in this project, the potential for a more generic form of mobile sensor platform is explored. Detailed in Chapter 3, this is attempted through a design that facilitates simple interchangeability of gas sensors and the ability to substitute other sensors as necessary.

Chapter 3

LittleBot

As outlined in Chapter 1, this thesis develops a new proof-of-concept robot for urban search and rescue called LittleBot. This chapter details the mechanical design considerations of the LittleBot devices developed.

3.1 Overview

Before building a platform to test the feasibility of the two-wheel self balancing concept proposed in Chapter 2, the skeleton for communication is first defined. Following this, a test platform and proof-of-concept are constructed. Testing of the proof-of-concept device leads to the decision to create three variants, differing primarily in the locomotion ability.

3.1.1 Communications

In addition to command and control with sensor feedback, video feedback from a LittleBot is required as part of the tether-less operation objective. For the reasons outlined in Section 2.2, an analogue approach to video feedback is taken. This is achieved by taking a camera with an analogue output encoded using the PAL or NTSC systems, and transmitting this signal frequency modulated at a high frequency. On the receiver side, the

signal is demodulated to regain the original video signal.

Transmitters and receivers for this purpose are common amongst hobbyists for FPV (first-person view) purposes, with 5.8 GHz transmitter and matching receiver modules commonly available. These typically have a +23 dBm transmission power and −85 dBm receiver sensitivity, and are normally coupled with 3 dBi gain dipole antennas. Here the transmission power and receiver sensitivity are expressed in dBm as they are measured in reference to one milliwatt, while the forward gain of the antennas are expressed in dBi and measured in reference to the hypothetical isotropic antenna representing a point source (which would have a 0 dBi gain). As the dipole antennas are omni-directional as opposed to a point source, they have at least a 2.15 dBi gain.

To calculate the theoretical transmission distance, the Friis transmission equation may be applied as $P_r = P_t + G_t + G_r + 20 \log_{10} \left(\frac{\lambda}{4\pi R} \right)$. Here P_t is the power delivered to the transmission antenna, P_r is the power available at the receive antenna, λ is the wavelength, R is the distance, and G_t and G_r are the respective transmit and receive antenna gains. Rearranging to solve for distance and substituting in the aforementioned values, a line-of-sight transmission of $R = \frac{10^{(23+3+3-(-85)) \div 20}}{41.88 \times 5800} = 2.1$ km is found when 3 dBi gain dipole antennas are used. Although the non-line-of-sight transmission distance is difficult to calculate as it depends on the environment, this set up is expected to be capable of transmitting at least 50 m in a non line of sight situation.

While an analogue design provides a means to obtain tether-less video feedback from a LittleBot, a way of controlling a LittleBot as well as allowing all the LittleBots to communicate with each other and transmit their sensor feedback is still required. The DigiMesh capabilities of the 900 MHz XBee-PRO 900HP modules discussed in Section 2.2 make them ideal for long range communications exceeding 10 km line of sight, and are therefore selected. With the modules configured to automatically form a DigiMesh network, they can be commanded over UART by a paired

microcontroller through API (application programming interface) commands as specified by Digi International [40].

Configuring the microcontroller to create packets with the desired data, the packets can be addressed and transmitted to a target recipient node in the DigiMesh network using the relevant XBee API commands. With the XBees handling the transmission of the packets transparently, packets received by the XBee are passed to the microcontroller for decapsulation and processing. This provisions command and control of a LittleBot (within video range of a Controller), as well as long-range sensor feedback using the automatic routing of the DigiMesh network (when the recipient node is out of direct range of the transmitting node).

In summary, communication between LittleBots and an operator is achieved through use of two radio frequency bands. Limited range analogue video feedback is provided using frequency modulation of the video feed in the 5.8 GHz spectrum, while a digital mesh network in the 900 MHz band provided by XBee-PRO 900 HP modules is used for command and control of the LittleBots from the base station.

3.1.2 Test Platform

A platform with a low centre of gravity is first constructed using off-the-shelf parts to test the feasibility of the two-wheel self-balancing design proposed in Chapter 2. Two gear-motors attached to 90 mm diameter wheels provide the basis for locomotion of the device, while an Arduino Uno provides control. A shield featuring two MC33926 motor driver ICs is attached to the Uno to drive the gear-motors with control over speed and direction. The motors and electronics are mounted on a laser cut acrylic platform, and a hobby style two-cell lithium polymer battery attached to the underside of the platform serves as the power source while contributing to the low centre of gravity. Finally a second shield with an XBee is stacked on top of the motor driver shield, configured to provide a trans-

parent wireless UART link to the Arduino for control of the platform.

Figure 3.1: 2-Wheel test platform

Shown on the left of Figure 3.1, the platform described is controlled using the remote on the right that consists of a second Arduino Uno, a matching wireless shield and an analogue joystick. Basic software mapping the joystick position to power and directions for each motor is implemented on this controller, which is sent to the platform over the wireless link provided by the XBee. The Arduino on the platform processes this to set the direction and duty cycles of the motors through control of the MC33926 ICs. With the test platform demonstrating the potential feasibility of the two-wheel design approach, a full proof-of-concept design is undertaken.

3.1.3 Proof-of-Concept

Following the test platform just described, a proof-of-concept device including sensors is designed and constructed with a 3D printed chassis. Shown in Figure 3.2, this proof-of-concept is equipped with an MQ9 gas sensor [41] capable of detecting carbon monoxide concentrations between 10 and 1000 ppm and flammable gas with concentrations between 100 and 10,000 ppm. A camera with a 720×576 pixel resolution provides video feedback, transmitted in analogue at 5.8 GHz. Finally a custom PCB is fabricated to provide the required functionality, while a 2-cell 2500 mAh lithium polymer battery powers the device.

Figure 3.2: 3D printed proof-of-concept

Experimental testing of this proof-of-concept successfully enabled piloting of the device using the video feedback with the operator unable to observe the device directly. However, the two-wheel design displays some drawbacks, with the device oscillating back and forth upon coming to a stop. Furthermore, accelerating too quickly causes the chassis to pivot on the axis of wheel rotation. While these issues could be addressed through careful control of the motor acceleration and deceleration to prevent rapid changes in speed, further testing of the device demonstrated a susceptibility to rolling down inclines. Within the SolidWorks CAD program, simulations demonstrated that this configuration would be unable to climb an incline greater than 7.5 degrees.

3.1.4 Variants

With the proof-of-concept demonstrating a viable device, avenues for variants of the device that could be deployed in varying circumstances were investigated. Three LittleBots are proposed, two mobile and one stationary. Varying in cost and configuration, a specific LittleBot variant may be selected to better suit the requirements of a particular situation.

Presented in Figure 3.3, the 2-wheel proof-of-concept results in the first mobile variant (labelled “Junior”) that targets the low cost disposable objective.

Figure 3.3: Junior LittleBot

Initial testing of the proof-of-concept demonstrated that relying on a low centre of gravity to keep the LittleBot level on two wheels was ineffective. As a result, the axis of rotation is repositioned to the front of the device, and the partially visible red tail in Figure 3.3 added for the device to rest on. While similar in design, the second mobile variant (labelled “Standard”) features four-wheel locomotion as shown in Figure 3.4.

Figure 3.4: Standard LittleBot

Essentially a sensor suite on a four-wheel independent drive, the Standard LittleBot is designed to traverse a greater range of terrain than the Junior. The third and final variant is the small, stationary Sentry LittleBot.

This LittleBot, shown in Figure 3.5, is designed for monitoring at a fixed location while reinforcing the DigiMesh network.

Figure 3.5: Sentry LittleBot

Equipped with extra sensors such as motion or additional gas sensors, a Sentry LittleBot can provide long-term monitoring of the conditions of interest (hazardous or otherwise). These variants forego the video feedback, and may be deployed manually by a human or larger robot [6].

All three variants communicate with each other and the Controller (discussed in Chapter 5) through the DigiMesh network provided by the XBee. If a LittleBot is out of direct range of the base station but within range of any other LittleBot, communication with that node will still be possible through the ad-hoc mesh networking system provided by the DigiMesh feature of the XBees.

3.2 Mechanical Design

Although injection moulded ABS is the target manufacturing technique for potentially cheaper mass production, Sentry prototypes are manufactured through 3D printing, and mobile variants through the milling of aluminium (as 3D printing is insufficient in resolution to meet the tolerances required for the transaxle designed later). Starting with a block of alu-

minium of sufficient size, the required features are milled from each side of the chassis to achieve the specified design. This technique imposes some limitations on what features can be created, as inside corners will have a radius and pockets cannot be created, both of which must be considered throughout the chassis design. When required for component attachment, aluminium screws of a softer alloy than the chassis are always used to minimise the likelihood of threads in the expensive prototype chassis being damaged.

For all variants of LittleBot, a two-piece design consisting of a chassis and cover is created. For each chassis, the electronics mount onto the roof and motors onto each side leaving 6 mm of clearance between the tallest of all components and the face of the chassis for the battery cells. Discussed further in 3.2.2, the battery cells are mounted into the cover, which is then bolted to the chassis with seals providing ingress protection.

3.2.1 Locomotion

As Standard and Junior LittleBots are mobile variants, a form of locomotion is required so that they may be repositioned remotely. A combination of geared DC motor, bearings, wheel hub and rubber wheels are selected and employed in a manner that provides ingress protection and shock absorption.

DC Motor & Gearbox

A micro metal DC motor coupled to a 100:1 gearbox, resulting in a low cost, compact yet powerful drive unit, provides the locomotion for a mobile LittleBot. The unit measures $12 \times 10 \times 25.5$ mm including the 18 mm long drive shaft, making it suitably small for application in both mobile variants of LittleBot. Illustrated in Figure 3.6, suppression capacitors are added before mounting to reduce the high frequency noise generated during operation.

Figure 3.6: DC motor and 100:1 gearbox

This set up provides a final output drive speed of 220 RPM at 6 V, the minimum that will be supplied by the power source discussed in Section 4.6. Driving the 80 mm diameter wheels through the wheel hubs discussed in the next section, a LittleBot is capable of a maximum speed of up to $\frac{220}{60} \times \pi \times \frac{80}{1000} = 0.92$ metres per second.

Wheel Hub & Bearings

The design of the transaxle must take ingress protection into account in order to maintain the IP55 rating required as one of the objectives. A radial bearing mounted in LittleBot's chassis in front of the gear-motor relieves lateral stress from the gearbox, while a larger sealed radial bearing press-fitted into the chassis wall of LittleBot provides the primary source of ingress protection in the transaxle.

Figure 3.7: Exploded view of a mobile LittleBot's transaxle

Illustrated in Figure 3.7, this larger bearing also serves as the main bearing for the wheel hub which is press-fit into the outer sealed bearing. The shaft of the gear-motor goes inside the wheel hub and a grub screw radially mates the wheel hub to the output shaft of the gear-motor. Although there are boss' on the wheel hub to keep each of the bearings in place, a retainer plate screwed to the chassis of LittleBot is added to minimise bearing movement and provide further mechanical robustness. Through use of a wheel hub, each wheel on LittleBot can be replaced through removal of three Philips screws. This provides the opportunity for purpose built wheels for different environments to be designed, which can easily be swapped out on-site by an untrained user without specialist tools.

Wheels & Tail

The wheel interchangeability allows a LittleBot to be equipped with varying wheel designs to suit particular terrains. To demonstrate this, two prototype wheel designs are manufactured through water jet cutting of 10 mm, 60 shore natural rubber.

Figure 3.8: Proposed wheel types

Both wheels pictured in Figure 3.8 are 80 mm in diameter which provides 27 mm of ground clearance for a Standard LittleBot, and 22 mm for a Junior. The T1 design (a) is proposed as the standard wheel supplied with

a LittleBot for a wide range of terrains, while the T2 design (b) is targeted at more rugged terrains. The selected combination of thickness and hardness of the rubber makes the wheels rigid enough to not collapse under the weight of the chassis, and provides some shock absorption. Although the manufacturing technique employed for these prototypes is expensive, mass production of wheels could be realised through use of a dye for each design once finalised.

A tail for the Junior variant to naturally rest on under its own weight is required, as its design employs only two wheels that are mounted towards the front of a chassis. The tail is designed to have the distance between its tips match the 80 mm diameter of the wheels so that the chassis is parallel with the ground. These tips are filleted in all directions to provide a round, smooth surface without sharp edges to minimise drag and catching on objects. Illustrated in blue attached to a Junior chassis in Figure 3.9, the prototypes are manufactured through 3D printing.

Figure 3.9: Junior LittleBot's tail

Two 3 mm bolts are used to attach each tail to the chassis. Although the prototype tails are manufactured through 3D printing, inexpensive mass production of this component could be achieved through injection-moulded ABS (in a similar manner as is proposed for each chassis).

3.2.2 Component Mounting

For each LittleBot, components need to be mounted on the chassis both externally and internally. In creating the features in the chassis to mount these components, the clearance between other components and the underlying goal of ingress protection must be taken into account.

Cover & Battery Cells

As outlined previously, each LittleBot chassis is a two piece design with the cover being the second piece to which the battery cells discussed in Section 4.6 are adhered. The footprint of a Standard LittleBot chassis is thinner but wider than the other variants, and allows the two battery cells to be mounted side by side as illustrated in Figure 3.10.

Figure 3.10: Standard LittleBot's battery mounting

The cover begins as a 3 mm piece of aluminium, with a 1 mm recess milled out. This provides a space to mount the cells accurately while also increasing clearance between the cells and electronics when the cover is fitted. Before the cells are adhered into the cover, non-conductive insulation tape is applied behind and around cell terminals to prevent accidental shorting. As the Junior LittleBot is shorter than the Standard, the cells cannot be mounted side by side. Instead, a thicker 8 mm piece of aluminium provides the basis for the cover, with a 6 mm recess milled out.

Figure 3.11: Junior LittleBot's battery mounting

Pictured in Figure 3.11, the cells are mounted within the recess on top of one another, as opposed to the side by side layout in the Standard LittleBot. This thicker plate with recess adequately spaces the cells away from components within the main chassis when the cover is mounted. Again insulation tape is applied to the cover around the cell terminals to prevent shorting, with the first cell adhered to the bottom of the recess and the second cell adhered on top of the first. In both the Standard and Junior LittleBot, the cells protrude from the face of each respective cover by 5 mm. With the cover bolted flush against the LittleBot chassis, the recessing into the covers provides adequate clearance between the cells and the tallest components within the chassis.

With the operating time goal of a Sentry LittleBot considerably longer than the mobile variants, higher capacity cells that are physically larger are required as discussed in Section 4.6. The height of the Sentry chassis is already increased in order to accommodate the PIR sensors discussed later, allowing a stacked cell-mounting configuration in a similar fashion to the Junior LittleBot. As there is ample space between the electronics and face of the chassis to which the cover mounts, a thicker cover with a recess for the cells is not required. Similarly to the approach taken with the Standard LittleBot, a 3 mm thick cover is used with a 1 mm recess to facilitate straightforward alignment of the battery cells during their adhesion.

With the battery cells mounted in their respective covers, a seal between this and the chassis when mounted is required in order to attain the IP55 ingress protection objective. O-rings are a common way to achieve a seal between two static components and are straightforward to implement. For each LittleBot, a 1.5 mm wide 1.2 mm deep groove around the inner edge of the face of the chassis that mounts with the cover is added.

Figure 3.12: O-ring sealing the chassis cover

Illustrated in Figure 3.12 a 1.5 mm diameter nitrile O-ring is mounted in this groove. With the groove only 1.2 mm deep, 0.3 mm of the O-ring protrudes from the chassis and makes contact with the cover when mounted to form a seal. Four mounting holes are added to the corners of the cover with matching threaded holes in the chassis through which the cover is secured. As the mounting screws for the cover are placed outside of the sealed area, they themselves do not need to be sealed.

Camera & LEDs

The camera and accompanying LED illumination discussed in Section 4.3 are mounted behind polycarbonate covers that are sealed with an O-ring. This provides the required ingress protection without obstructing the camera's visibility or the effectiveness of the LEDs. Features are designed in

the chassis to precisely mount each component, and a 1.2 mm deep groove is added around the outer edge of each component to house a 1.5 mm O-ring. The screws to mount the covers are placed outside the diameter of the O-ring so that they do not need to also be sealed.

Figure 3.13: Covers sealing the camera and LEDs

Illustrated in Figure 3.13 behind the sealed O-ring covers, the LED emitter plates are mounted directly into the circular features cut into the front of a LittleBot. Instead of metal screws, nylon screws secure the emitters to the chassis to avoid shorting of the emitter's exposed terminals. For power, wires to each emitter are run through channels in the chassis behind the cutout of each emitter. Finally the manual focus of the camera is empirically set before being mounted behind the centre O-ring sealed polycarbonate window.

Figure 3.14: Bracket securing the camera

To hold the camera in place in its mount, a locking bracket that also acts as a channel for routing the LED wires is designed and manufactured using 3D printing. Shown in Figure 3.14, the bracket is screwed down onto

the designated pillars housing the camera in the aluminium chassis. The bracket locks the camera in place while allowing its wires to pass under the bottom of the bracket for connection to the PCB. Countersunk screws are used to ensure the top is flush, maintaining the required clearance from the battery cells when the LittleBot is assembled.

Gas & PIR Sensor

All three LittleBot variants feature a user-exchangeable gas sensor module. Discussed later in Chapter 4, the gas sensors require supporting electronics, and are mounted on a custom PCB as a gas module. The sensor itself must be exposed to the environment to operate, however the module should be easy to exchange yet still form a seal with the chassis for ingress protection. This results in the PCB being designed to directly mount to the chassis, with a seal created between the PCB and chassis using an O-ring.

To maintain ingress protection, the PCB itself cannot contain any channels through which ingress can occur. The round PCB is 19.5 mm diameter, 3 mm larger than the diameter of the largest gas sensor to be supported. With a gas sensor mounted to the front of the PCB, the supporting electronics are surface mounted on the rear of the board. Once the sensor is soldered into place, no ingress susceptible holes remain in the PCB. M2 threaded holes are added to the PCB for mounting to a LittleBot using Philips head M2 screws. With this mounting system, a user can exchange a gas module for an alternative within a few minutes.

Figure 3.15: Gas sensor mounted on the Standard LittleBot

On the Junior LittleBot, the gas sensor is located at the rear of the device while on the Standard LittleBot, it is placed between two motors on one side of the device behind the wheels. In both cases it is mounted in a 20 mm diameter 5 mm deep circular hole to minimise the protrusion of the sensor. In the case of the Standard LittleBot, this also prevents interference with the wheels as demonstrated in Figure 3.15. In the remaining 2 mm of chassis wall, holes that align with the mounting holes of the gas module PCB are added for bolting from behind inside the LittleBot chassis. An additional cut out enables the PCB to sit flat and not short the pins of the gas sensor against the chassis, while providing a channel for the module's wiring.

Figure 3.16: O-Ring placed in the gas sensor mount

With the PCB sitting flat in the set of mounting features described, an O-ring can be used to provide a seal between the PCB and chassis. Shown in Figure 3.16 between the tails on a Junior, a 1 mm thick 19.5 mm outer diameter O-ring is placed in the gas sensor mounting socket before the PCB is installed. When the mounting screws for the module are tightened from behind, the PCB is pressed up against the O-ring, and the desired seal for ingress protection formed.

On the Sentry LittleBot, four universal mounting points allow for a greater range of sensor modules to be installed. In addition to the gas modules just described, PIR sensors are considered potentially useful for

motion detection on the Sentry. As the sensor component of the gas module is smaller in diameter than that of the PIR module, the mount is consequently designed to house and seal a PIR module. Again O-rings are used, with the holes in the Sentry chassis made to be 0.5 mm bigger in diameter than the base of the PIR module. Holes matching those on the PIR module are added on the inside of the chassis to which the module can be bolted, and a groove for an O-ring added around the edge of the mounting hole.

Figure 3.17: O-ring sealing around the PIR lens

Once the PIR sensor is mounted and bolted, the lens of the sensor is pressed against the O-ring and the O-ring in turn against the chassis, shown prior to the screws being tightened in Figure 3.17. This results in a seal between the PIR module's lens and the LittleBot chassis, preventing ingress. With the mount primarily designed to accommodate a PIR sensor, the previously described gas modules cannot be directly mounted without modification.

Rather than designing a second type of gas module that is specifically made for the Sentry, an adaptor plate is used to maintain interchangeability of the modules between mobile and Sentry LittleBots. The adaptor bolts into the universal mount with an O-ring to seal it against the chassis in the same manner as that applied for the PIR sensors. A gas module can then be installed into the adaptor plate the same way it would be installed into a Standard or Junior LittleBot, with an O-ring around the outer edge of the PCB and the wires for the module passing through the cut-out at the

back of the adaptor. As the adaptor plate is sealed with respect to the Sentry Chassis, and the gas module sealed with respect to the adaptor plate, it follows that the gas module is sealed with respect to the chassis.

Temperature & Humidity Sensor

On the Standard and Sentry LittleBot variants, the DHT22 sensor discussed in Section 4.5 provides temperature and humidity readings. The electronics of these devices are inherently sealed, with only the sensing elements exposed. Consequently the devices can be mounted so that they are exposed to the environment, however they must be wired back to the electronics inside the LittleBot chassis through a channel that must be sealed.

Figure 3.18: DHT22 on a Standard LittleBot

Shown in Figure 3.18, the sensor is mounted on a Standard LittleBot in a recess between the motor mounts that is deep enough to provide ample clearance from the wheels. The same approach is taken in mounting the DHT22 on the Sentry LittleBot, however in the absence of wheels the sensor can be mounted on any face. Attached using a bolt, the sensor's three wires pass through a channel created in the chassis to the PCB, with the channel sealed using silicone.

Charging Jack, Switch & Power Indicator

With the battery cells installed inside each LittleBot, a way of charging them externally is required. Additionally, a switch to turn a LittleBot on and off is desirable such that they can be charged in advance and stored without the battery discharging. Finally, a fail safe method of ensuring a LittleBot is off during charging is necessary in order for the external smart charger to operate correctly. Instead of having a separate charging jack and power switch, a three pin keyed charging port that also doubles as a power switch is used. One pin of the jack is a shared common ground, attached to both the ground of the battery protection PCB (discussed later in Section 4.6), and the ground of the LittleBot's main electronics PCB. The second pin is attached to the positive terminal of the battery protection PCB, and the third pin to the positive input jack on the LittleBot PCB.

With this configuration, a mating plug may be fitted to the charger giving it a direct connection to the battery protection PCB, while another plug called a power key is made that connects the positive terminals of the electronics and battery PCBs together. As the socket and plugs are keyed to allow mating in only one orientation, there is no risk of connecting the charger or power key incorrectly and damaging a LittleBot. As the power key must be removed in order to attach the charger, there is no risk of a user incorrectly charging a LittleBot while it is powered.

Figure 3.19: Combination jack and power indicator

SwitchCraft three circuit sockets and plugs are selected for the charging jack and switch, shown installed on the rear left of a Junior LittleBot in Figure 3.19. In addition to their IP67 rating, these series of connectors are selected due to their included locking ring feature. When inserted, a locking ring on the plug is slid towards the socket and turned to lock it on. The locking effectively reduces the likelihood of a LittleBot being accidentally switched off by falling debris or other impacts that could dislodge the power key from its socket. With all the mounting hardware and gaskets supplied, all that is required is to create the specified cut out in the LittleBot chassis to mount the socket. In addition to the power jack, the IP67 rated blue LED indicator (demonstrated mounted on the rear right of a Junior LittleBot in Figure 3.19) provides a visual indicator of a LittleBot's power status.

Antennas

All variants of LittleBot employ a U.FI to RP-SMA cable to enable the attachment of external antennas for the XBee, and where fitted, TX5823 video transmitters. On mobile LittleBots, these connectors are centred on top of the chassis towards the front, shown in Figure 3.20 on a Standard LittleBot.

Figure 3.20: RP-SMA connectors on a Standard LittleBot

This allows 3 dBi antennas with adjustable angle to be attached, and can be angled down to minimise their impact on the footprint of a mobile LittleBot. On the Sentry LittleBot, the connector is instead placed on the side of the chassis as illustrated in Figure 3.21 with an antenna attached. This allows it to be orientated to be parallel with the lid, reducing the overall height of the Sentry.

Figure 3.21: RP-SMA connector on a Sentry LittleBot

In all cases the RP-SMA connector itself is sealed, however the mounting point for the connector creates a potential point for ingress to occur. The base of the connector that butts up against the chassis when mounted is flat enough for a sealing device such as an O-ring or gasket to be effective. However given the small size of the connector, the round profile of an O-ring is not ideal to provide such a seal. Instead a flat gasket made from a material such as nitrile is more appropriate, providing a seal over a larger surface area between the two flat surfaces. Custom gaskets that fit between the base of the RP-SMA connector and respective LittleBot chassis are designed and laser cut from 0.2 mm thick nitrile sheeting. With the gasket fitted between the connector and chassis, tightening the mounting nut of the connector on the opposite side of the chassis places pressure against the gasket to create the desired seal.

PCB

To mount the PCBs that control the components of a LittleBot, round standoffs are added to the roofs of each chassis. The standoffs are made to be 3.5 mm tall, which prevents shorting of leaded components against the chassis while allowing components to be mounted on the rear of the PCB, With the battery cells protruding from the face of the covers by 5 mm, the maximum permissible height of the PCBs including components is limited to 12.5 mm from the stand offs. This design restriction is adhered to throughout the electronics design, further discussed in the next chapter.

Chapter 4

Electronics

In order to achieve the desired functionality for each LittleBot, purpose built electronics are required. This chapter discusses the three PCBs designed to provide the necessary supporting electronics, one for each LittleBot variant.

4.1 Overview

As discussed in Chapter 3, the design of the chassis for both mobile LittleBots provides a maximum of 12.5 mm of clearance between the PCB stand-offs and the battery cells when assembled. To maximise the amount of space available for components, 1.2 mm PCBs are fabricated instead of the typical 2.0 mm, leaving $12.5 - 1.2 = 11.3$ mm of clearance between the top of a PCB and the battery cells. To increase the available space on the front for components that are classically taller, components that are shorter than the 3.5 mm of clearance provided by the standoffs are placed on the rear of the PCBs.

Figure 4.1: Standard LittleBot PCB

The Standard LittleBot PCB, shown in Figure 4.1, features all of the required electronics to achieve the desired functionality. As the Standard and Junior LittleBots are very similar in functionality, so too are their PCBs which differ primarily in the physical layout. However the PCBs for a Sentry LittleBot are substantially different, omitting the electronics for video feedback and motor control.

Figure 4.2: Sentry LittleBot PCB

As shown in Figure 4.2, the Sentry PCB shares some common elements with Standard LittleBot PCB including an XBee socket and the same type of connectors. The DRV8835's are removed as they are redundant, and a second DC-DC converter is added given the higher 5 V power requirements discussed later. Due to space constraints, the ICSP header is relocated from the centre of the board to above the XBee as a single row six-pin header instead of a two-row header.

4.2 Core

The XBee Pro 900 HP modules provide a self-healing 900 MHz mesh network, with each node being addressable by their hardware address. Through an API mode discussed further in Chapter 6, the modules can be commanded to send a packet of up to 100 bytes to any node on the network. As the modules cannot be programmed to control devices such as the motor drivers or take sensor readings, a microcontroller that interfaces with the XBee is required. This also handles the composition and processing of packets, executing the required XBee API commands to send and receive these packets.

4.2.1 ATMega328 Microcontroller

In selecting the microcontroller to act as an interface between actuators, sensors, and XBee, simplicity, low cost, low power consumption and ease of programming were prioritised. Given the author's comprehensive knowledge of the Arduino IDE, its libraries, and advanced functionality, the selection of an AVR Mega compatible microcontroller is desirable (as the Arduino IDE is capable of compiling for this target platform). To satisfy the requirements of all three LittleBot variant peripherals the Atmel ATMega328 is selected, as it is inexpensive and possesses sufficient I/O (input/output) pins.

The ATMega328 can be powered with a VCC that is between 1.8 V and 5 V, however this VCC dictates the logic high input and output voltages. While all of the sensors can operate on and communicate using voltages between 3.3 V and 5 V, the ATMega328 is interfaced with the XBee which has an operating voltage range of between 2.1 V and 3.6 V. Like the AT-Mega328, the VCC applied to the XBee determines the logic high input and output voltages of the device. As a result it is desirable for the AT-Mega328 and XBee to share a common VCC in order to avoid the need for logic level conversion. With the ideal range of VCC limited to this range,

the TX5823 video transmitter module discussed later has a specified operating voltage of 3.3 V. Although there is no logic level conversion required in controlling this device, minimising the number of supply voltages on the board reduces component count and design complexity. This results in 3.3 V being selected as the VCC for both the ATmega328 and XBee.

In order for the ATmega328 to operate correctly, a clock is required. While the ATmega328 comes with an internal 8 MHz RC clock that is enabled by default, the inherent inaccuracy of this clock's RC design drastically reduces the maximum speed at which UART can be reliably utilised. The XBee and ATmega328 communicate via the UART, and all XBees in a network must have the same UART data rate. In turn the UART speed employed on the LittleBots dictates the speed of the XBee communications on the Controller. The speed at which the XBee communicates with the Controller's ATmega2560 discussed in the next chapter needs to be maximised for real time control of a mobile LittleBot, hence the UART speed on the LittleBot themselves must also be maximised. As a result, an external crystal that is superior in accuracy to the internal RC clock is employed and attached as the clock to the ATmega328. The maximum frequency at which the ATmega328 can be reliably clocked without error is dictated by the VCC supplied, limited to 12 MHz when with the 3.3 C selected [42].

Depicted in Figure 4.3 for a Standard LittleBot PCB, the ATmega328 is implemented with a 12 MHz crystal and the recommended 18 pF load capacitors, while input/output pins are assigned as required to each of the devices discussed later in this chapter. Power supply to the device is decoupled using the standard practice of capacitors between VCC and ground in close proximity to the device, while the voltage reference for the ADC is decoupled to ground as it will be configured to use the internally generated voltage reference instead. The SPI, reset and power lines used to program the device using an ATMel AVR ISP Mark II programmer are broken out to the header *P6*, with the reset line pulled high through a 10 k Ω resistor enabling normal operation of the device.

Figure 4.3: Schematic of the implemented ATmega328

4.2.2 XBee Pro 900 HP

As discussed, the XBee Pro 900 HP can operate at voltages between 2.1 V and 3.6 V, with 3.3 V selected to match the ATmega328 and avoid the use of logic level converters. In interfacing the XBee with the ATmega328, UART or SPI can be selected. However, in order to use DigiMesh, the modules must be commanded using the API mode that is only available over UART. The supporting electronics for the XBee Pro 900 HP module are minimal and in this application only the UART and power must be connected. As the modules are one of the more expensive components in a LittleBot, header sockets that are of sufficiently low profile for the module to not exceed the height restriction placed on the PCB are used instead of permanently soldering the modules. This allows the XBee to be easily salvaged from a LittleBot in the event of damage. With the modules requiring an initial configuration using the X-CTU utility under Windows™ discussed later in Chapter 6, socketing the modules allows them to be removed for reconfiguration if required. 3.3 V is supplied to the sockets with 47 μ F ceramic decoupling capacitors to prevent brown-outs during

transmission spikes, while the UART data lines are directly connected to the UART of the ATmega328. Using a U.FI to RP-SMA cable and an XBee with a U.FI connector selected, the RP-SMA connector allows an external antenna to be attached to the XBee, with the connector sealed as explained previously in Section 3.2.

4.3 Video Feedback

For remote piloting and reconnaissance, video feedback is required on the mobile LittleBots. As discussed in Chapters 2 and 3, an analogue approach is taken for the video transmission. An analogue output camera supplies the video signal, which is fed into a 5.8 GHz FM transmitter. This is then demodulated on the controller and fed into a display on the Controller discussed in Chapter 5. As it is unlikely that a LittleBot will always be operated in daylight, high power LEDs supplement the video feedback system for illuminating low light environments that may be encountered.

4.3.1 Camera

An FH10C V9 [43] camera measuring $10 \times 10 \times 18$ mm that is appropriately small for application in LittleBot provides a PAL encoded composite video signal. A sensitivity of 0.008 Lux/F1.2 and automatic exposure adjustment makes the camera suitable for use in dark environments, while a 120° fixed focus lens provides a wide viewing angle. Finally, with a resolution of 720×576 pixels at 25 frames per second, the camera is capable of providing a video feed comparable to that of a DVD and more than adequate for the purposes of LittleBot.

Figure 4.4: FH10C V9 PAL video camera

The camera, pictured in Figure 4.4, consumes 80 mA at 5 V, a considerable drain on LittleBot's limited power supply. To maximise runtime, a way of controlling the power to the camera when it is not required is sought. With an on resistance in the tens of milliohms, the use of MOSFETs for low frequency power switching has a negligible impact on battery life. As a result, a single N-channel enhancement mode MOSFET is used to control the camera's ground connection to switch it on and off.

Figure 4.5: Camera power control using an N-type MOSFET

This approach is illustrated in Figure 4.5, where component designated *P6* is the three pin connector to which the camera is connected on the LittleBot PCB. A JST PH series connector is used here and throughout all LittleBot electronics to provide removable connections. The power input of the camera is attached directly to the 5 V power supply, while the video output signal from the camera is fed into the TX5823 transmitter discussed in the next section. The ceramic 47 μF X7R capacitor *C13* provides localised power decoupling for the camera, which is common practice to minimise visible interference that may result from noise on the power supply. Finally the ground connection of the camera is attached to the drain of the

N Type MOSFET *Q11*, while the MOSFET's source is attached directly to ground. By applying 3.3 V to the gate of the MOSFET, the ATmega328 turns the MOSFET on as explained previously, in turn providing the camera with power. When the ATmega328 brings the gate to 0 V, the MOSFET's minimum gate-source threshold voltage of 0.4 V is no longer exceeded, turning the MOSFET off and cutting power to the camera.

4.3.2 Transmitter

To transmit the composite video output from the camera to the controller, a Boscam TX5823 5.8 GHz AV transmitter module is employed. This device is designed to take analogue audio and video signals, and transmit them using frequency modulation on a selectable frequency in the 5.8 GHz band at +23 dBm.

Figure 4.6: TX5823 module mounted on a LittleBot PCB

Shown mounted on the back of a Standard LittleBot PCB in Figure 4.6 and measuring $22 \times 19 \times 3$ mm, the module is sufficiently compact to be incorporated into the design of the LittleBot PCB as a daughter-board. As the module is slimmer than the 3.5 mm clearance between the chassis and PCB (provided by the stand-offs discussed in Section 3.2), the TX5823 is mounted on the rear of the LittleBot PCBs for efficient board space utilisation. For correct operation, all that must be attached externally is power, a video signal and an antenna. Optionally, grounding three control pins

in different combinations allows the exact transmission frequency of the module to be specified.

Table 4.1: Mapping of TX5823 control pins to transmission frequency

TX Sel 1	TX Sel 2	TX Sel 3	TX Frequency
Floating	Floating	Floating	5705 MHz
Floating	Floating	Ground	5685 MHz
Floating	Ground	Floating	5665 MHz
Floating	Ground	Ground	5645 MHz
Ground	Floating	Floating	5885 MHz
Ground	Floating	Ground	5905 MHz
Ground	Ground	Floating	5925 MHz
Ground	Ground	Ground	5945 MHz

As listed in Table 4.1, one of eight transmission frequencies in the 5.8 GHz band can be selected using these three control pins. Software control of the transmission frequency is desirable as it allows a new channel to be selected in a situation where excessive noise may be present on a particular frequency. However, the pins cannot be directly connected to the ATmega328, as the internal pull-ups within the TX5823 are found to be lower than the 3.3 V VCC of the ATmega328 at 1.8 V. If the ATmega328 were to erroneously drive the control pins high, the TX5823 may be damaged. Instead N type enhancement mode MOSFETs are again employed, with one for each channel control pin.

Illustrated in Figure 4.7, three PMV16UN's MOSFETs, Q8, Q9 and Q10, are employed to selectively ground each of the frequency selection pins. The gate of each MOSFET is connected to an I/O pin on the ATmega328, allowing selection of the transmission frequency through the ATmega328's software discussed in Chapter 7.

Figure 4.7: Schematic of the TX5823 implementation

As the TX5823 consumes 320 mA at 3.3 V, the fourth MOSFET Q3 is employed for the purposes of turning the TX5823 on and off to save power. With multiple ground connections to the TX5823, pins that would normally be attached to ground are instead attached to the net *TX5823GND*. This is connected to ground on demand through the power control MOSFET Q3, ensuring all grounds to the TX5823 are removed and current cannot erroneously flow through the channel selection pins when the device is meant to be off.

Of the remaining components, the aluminium electrolytic 470 μF capacitor *C7* provides bulk decoupling to the TX5823 and the 47 μF X7R ceramic capacitor minimises the impact of high frequency noise on the power lines. The 470 μF capacitor *C14* removes the DC bias present in the composite PAL signal from the camera before feeding it into the TX5823, while the FM output from the module is routed to the U.FI antenna connector *J1*. Similarly to the XBee, a U.FI to RP-SMA cable provides an external antenna connection to the TX5823.

4.3.3 Constant Current Driver

Illumination for video feedback is achieved through two Cree XPG R5 high-power LEDs pre-mounted on aluminium discs that provide heat dissipation. With a forward current of 350 mA applied, each LED has a rated output of 139 lumens and a typical forward voltage of 2.9 V. In order to supply the LED with the correct amount of current, a constant current driver is required. As the two LEDs are wired in series, this driver needs to supply 350 mA at a minimum of 5.8 V. Typical LED drivers only control the output current and not voltage, as the voltage drop is dictated by the LEDs. As a result the drivers typically have a wide input voltage range. To maximise efficiency and minimise the number of power converters employed in LittleBot, it is desirable for the driver to be attached directly to the raw power supply. Consequently it must be capable of operating on the 6.0 V to 8.4 V range from the power supply discussed later in Section 4.6. A Diodes Inc. ZXLD1350 30 V 350 mA LED Driver [44] satisfies all of these requirements, and is implemented as depicted in Figure 4.8.

Figure 4.8: Schematic of the ZXLD1350 implementation

Types and values for diode $D1$ and inductor $L1$ are chosen based on recommendations from the driver's datasheet, while setting the nominal average output current is achieved through the selection of resistor $R6$ using $I_{out} = \frac{0.1}{R6}$ [44]. For 350 mA, $R6 = \frac{0.1}{0.35} = 0.29 \Omega$. With 0.3Ω the closest

available resistor value, the LEDs will be driven with $I_{out} = \frac{0.1}{0.3} = 333$ mA. Like the camera and TX5823 module, the LEDs are relatively power hungry devices that do not need to be on all the time, and a way of controlling the power to the driver using the ATmega328 is desirable. As previously with the camera and TX5823, an N-channel enhancement mode MOSFET, Q7, is employed to control the ground connection of the ZXLD1350 driver, allowing it to be turned on and off by the ATmega328.

4.4 Motor Drivers

In order to power the gear-motors, a motor driver is required. Texas Instruments DRV8835 drivers [45] are employed for this purpose, with each IC capable of driving two DC motors independently with 1.2 A at 11 V continuously. An advantage of the DRV8835 over similar DC motor drivers is an increased number of drive modes, including coasting. In this mode the driver can internally disconnect the motors instead of grounding them, ultimately reducing wear and tear on the gearboxes that would normally result from the motor abruptly stopping due to driver-side braking. The DRV8835 also features over-current and over-temperature protection, shutting down if the IC gets too hot and automatically resuming operation once a safe temperature is reached. For the applications of LittleBot, these protections reduce the likelihood of the drivers being destroyed due to malfunction during LittleBot deployment.

Figure 4.9: Schematic of the DRV8835 for a Junior LittleBot

For the Junior LittleBot, a single DRV8835 is employed to drive both motors as illustrated in Figure 4.9. While the DRV8835 is capable of driving two motors independently, a Standard LittleBot is fitted with four motors. As each pair of motors on each side of the LittleBot will be driven identically, a single DRV8835 could be used in the Standard LittleBot by connecting each pair of motors in parallel to one output channel of the DRV8835. However this approach would limit the maximum amount of current that is available to each motor. As the Standard LittleBot is targeted at climbing more difficult terrain, the reduced current would lead to an undesirable reduction in motor performance. Instead two DRV8835s are used, with one driver for the left pair of motors and another for the right pair.

Figure 4.10: Schematic of the DRV8835s for a Standard LittleBot

For each DRV8835 on the Standard LittleBot, a single pair of PWM capable lines from the ATmega328 controls both pairs of motor channel control inputs as illustrated in Figure 4.10. By attaching each motor of the two motors on the same side of a LittleBot to an independent output channel on the same DRV8835, both motors will exhibit the same behaviour without sharing current and hence without a reduction in performance.

In electronics, FET based motor drivers are susceptible to damage which is typically mitigated by utilising over specified drivers. While this

is incorporated through the application of the DRV8835 in LittleBot, the IC is in a WSON “very very thin small outline no lead package” that is difficult to solder by hand. Should a DRV8835 become damaged in the field, an untrained user would be unable to replace it and that LittleBot would be rendered inoperable. As a result, the LittleBot would have to be sent to an electronics technician for the IC to be replaced, a timely and costly process. To allow an untrained user to replace the driver, the IC is instead placed on a carrier PCB. This is then installed into matching sockets on the LittleBot PCB, as shown in Figure 4.11 installed on a Standard LittleBot PCB.

Figure 4.11: Two DRV8835s in low profile sockets

In addition to the driver IC, ceramic capacitors for local decoupling are included on this carrier PCB to reduce the effects of LC spikes and further minimise the chance of damage to the driver. To avoid exceeding the component height restrictions placed on the LittleBot PCB with the driver installed, components of sufficiently low profile are employed. By supplying simple instructions and spare drivers with a LittleBot, drivers can be replaced by untrained users in the event they become damaged.

4.5 Sensors

The Standard and Junior variant PCBs feature a single four pin connector for a sensor, while the Sentry PCBs possess four, four-pin GPIO (general purpose input outputs) connectors. These ports supply 3.3 V, 5 V, ground, and a configurable bidirectional digital or analogue connection to the AT-Mega328. While gas and PIR motion sensors have been constructed for connection to these ports, other sensors could be easily designed and connected for future expansion.

In addition to video feedback, mobile LittleBots are equipped with a gas module by default that is selected for sensitivity to gases of interest for a particular application. The larger chassis of the Standard and Sentry LittleBot provides space to accommodate an additional sensor. A temperature and humidity sensor is fitted to these variants, and location information is provided through use of a GPS module on the Sentry. The following sections detail how the sensors and supporting electronics are integrated into a LittleBot.

4.5.1 Gas

MQ Series [41] and Figaro brand [46] gas sensors are selected for the purposes of gas detection as they share a common method of operation. With different models capable of detecting different gases, PCBs that implement a gas sensor with the required supporting circuitry are created resulting in user exchangeable gas modules. This allows an untrained user to select and outfit a LittleBot with a gas sensor based on the type of gases they wish to detect.

Within each sensor is a heater that must be powered, and a sensing element. The resistance of the sensing element varies depending on the target gas concentration, where a lower resistance represents a higher target gas concentration. In order for this change in resistance to be measured by the ADC of the AT-Mega328, a voltage divider is implemented. VCC is

applied to one side of the sensing element, with the other connected to ground through a load resistor (with the value depending on the specific gas sensor). As the resistance across the sensing element varies due to changes in the concentration of target gas present, the voltage at the point between the sensing element and load resistor varies. Using the ADC of the ATmega328, this variation can be detected.

Figure 4.12: Schematic of the gas sensor module

Shown in Figure 4.12 is the schematic of a gas module, where *P1* represents the wired four pin JST connector that plugs into the LittleBot PCB and *R1* serves as the load resistor. The heater for a gas sensor consumes 150 mA at 5 V, hence control of the heater power is desirable to save power when not in use. On a Sentry, gas power control is implemented through shutting down of the 5 V supply for all GPIO pins, discussed in Section 4.6. On the mobile LittleBots, the 5 V regulator supplying the heater cannot be shut down as it is also shared with the camera. As a result power control is only feasible by switching the gas module's ground connection through use of an N-channel MOSFET.

When the ground connection to the module is disconnected by the MOSFET, the output attached to the ADC of the ATmega328 will be pulled up by the 5 V attached to the heater through the load resistor of the module. Over time this condition may damage the ADC of the microcontroller as it exceeds the maximum input voltage specified as $VCC + 0.3 = 3.3 + 0.3 = 3.6$ V. To prevent this, the Schottky diode *D1* with a voltage drop of 0.2 V is placed between the output and 3.3 V connections on the module. This allows the heater to be safely disabled through dis-

connection of ground using the MOSFET, as the diode prevents the output voltage of the module being pulled up beyond 3.5 V.

4.5.2 Temperature & Humidity

As discussed in Section 3.1.4, the Standard and Sentry LittleBots are equipped with a sensor that allows them to detect the temperature and relative humidity of their environment. Instead of employing separate sensors for each, an Aosong DHT22 combination sensor is utilised to minimise component count and simplify the design. Through a single wire combination digital I/O pin, the DHT22 utilises a thermistor to provide temperature readings and a polymer capacitor to provide relative humidity readings [47]. As with all other devices on the LittleBot, the DHT22 is connected to the main LittleBot electronics through a JST PH connector to provide power, ground, and data. As the electronics inside the DHT22 that convert the raw analogue data from its internal sensors into the digital single wire output are sealed from the exposed section of the device, the DHT22 poses little difficulty in achieving ingress protection.

While the DHT22 can operate on a voltage between 3.3 V and 5 V, 3.3 V is selected to avoid the need for level shifting for communication with the ATmega328. As the DHT22 needs to be signalled by the ATmega328 when to take a reading by having its data pin driven low, it is assigned a pin on the ATmega328 that is capable of being used as an input and output. Once driven low in output mode, the pin needs to be reconfigured as an input to read the digital signal from the DHT22 and is achieved in the LittleBot software later discussed in Chapter 7. For correct operation, the DHT22 requires a 20 k Ω pull-up resistor to provide logic high signals for outputting its data. The ATmega328 features software configurable internal pull-up resistors suitable for this purpose, making an external resistor unnecessary. Finally, the power consumption of the DHT22 is 50 μ A at idle, and 2.5 mA at peak. As the low power idle mode can be entered simply by

not requesting a reading, no external control of the power to the device is required. Further the 2.5 mA peak consumption during measurement is so insignificant that the DHT22's impact on battery life is expected to be negligible, even with continuous sampling.

4.5.3 Motion

For motion sensing on the Sentry LittleBots, passive infrared, ultrasonic and microwave technologies are considered. Of these, passive infrared modules are the simplest to implement reliably, and allow straightforward ingress protection simply by sealing the lens on the face of the device. At its simplest, a PIR (passive infrared) sensor uses a pair of detectors to sense the amount of infrared radiation. Motion is detected when the amount of radiation detected by each sensor differs, with an appropriate threshold set to prevent false triggers. Rather than buying the discrete components and IC to construct a PIR sensor, the Parallax PIR sensor modules pictured in Figure 4.13 are utilised.

Figure 4.13: Parallax PIR sensor

When the module detects motion, the TTL digital output is pulled high for two seconds and can be sensed by the ATmega328. Capable of operating on 3 V to 6 V, the voltage selected to power the module dictates the logic high output voltage. Again to avoid the need for logic level conversion between the PIR module and the ATmega328, the 3.3 V supply from

the GPIO pin is used as the VCC instead of the 5 V. The modules consume 130 μ A when idle and 3 mA when active with no load on the TTL output. While the digital output can source up to 12 mA with a logic high output, the inputs of the ATmega328 are high impedance and will sink effectively zero current from the module when sampling its output. When the modules are active in the case of motion being detected, the 3 mA of current they consume is negligible in terms of impact on the runtime of a LittleBot and additional power control circuitry is not necessary.

4.5.4 GPS

To determine the location of a deployed LittleBot Sentry, a GPS is required. Commercially available GPS modules commonly utilise a low baud rate serial data interface to provide positional information typically encoded in the NMEA format. Although the lone UART interface of the ATmega328 is reserved for communication with the XBee leaving no hardware UART for a GPS, bit-banged serial communication up to 115200 bps is possible on any pin of the ATmega328 using a software library. Consequently a serial output GPS module can still be used as long as its data rate does not exceed this 115200 bps limitation.

The GP-2106 [48], a 48 channel GPS receiver utilising the SiRF IV chipset, is employed. With a default communication data rate of 4800 bps, the serial output of this GPS can be handled using the bit-banged software UART. As the GP-2106 can consume up to 150 mA at 1.8 V in its active state, it is desirable to minimise this power consumption when positional information is not required. The power could be disconnected on demand through use of a MOSFET controlled by the ATmega328, as has been done previously with other sensors. However this is an undesirable solution as the GPS would then always be performing a cold start, taking over a minute to reattain a GPS fix. Instead the GPS can be put into its hibernate mode through pulsing of a *shutdown* pin. In this state the GP-2106 con-

sumes less than 30 μA , resuming operation with a GPS lock in less than one second.

While most GPS modules provide a standard serial data output, they vary greatly in operating voltage and in turn their logic level voltages, as well as their physical connector. The GP-2106 requires a 1.8 V supply, and as a result, logic level shifting of the serial output and shutdown signal between the ATmega328 and GP-2106 is required. As the GP-2106 is the only device requiring a 1.8 V supply, an interface board is employed instead of implementing the electronics on the main LittleBot PCB. This allows a different GPS module and interface board to be substituted if so desired.

On the main LittleBot PCB, a four pin JST PH connector to attach this interface board is included, supplying it with 3.3 V, ground, and two connections from the ATmega328. The interface board converts the 3.3 V to 1.8 V using an LDO regulator, while the two lines from the ATmega328 are level shifted using MOSFETs and pull-up resistors. One of these is terminated to the serial output connection from the GP-2106, and the other to the hibernate input. Finally an FPC connector matching that of the GP-2106 is added to the interface board, so that it may be connected using an FPC cable of appropriate length.

4.6 Power Supply

With the devices being tether-less and a minimum 2 hour runtime specified as part of the objectives, a power supply of sufficient capacity is required. For a Sentry LittleBot, a longer runtime time of at least 24 hours is desired. The power supply of all three variants may be supplemented with solar cells, and although not included in the scope of this thesis, headers for solar cell attachment are included on the PCBs.

4.6.1 Required Battery Capacities

To achieve these target runtimes, the power consumption of all the electronics, actuators and sensors must be considered so that the battery capacity required may be determined. As the motors and LEDs on a mobile LittleBot can operate on a wide input voltage range, they are supplied directly from the raw output of the lithium polymer cells (discussed later) and their current at the 7.4 V nominal output of the cells used for consumption estimates. Taking pessimistic estimations of the power requirements for the devices in a Standard LittleBot yields the values in Table 4.2, where all sensors and actuators are at 100% duty cycle (where applicable).

Table 4.2: Power consumption of Standard LittleBot components

Device	Voltage	Current	Power
TX5823	3.3 V	320 mA	1056 mW
XBee (Transmitting)	3.3 V	215 mA	710 mW
Camera	5 V	85 mA	425 mW
Gas Sensor	5 V	150 mA	750 mW
Motors (4)	7.4 V	400 mA	2960 mW
ATMega328	3.3 V	20 mA	66 mW
LEDs	7.4 V	350 mA	2590 mW

For the XBee, the 215 mA at 3.310 V specified is its maximum consumption while transmitting with a 100% duty cycle [40]. While the XBee is unlikely to have a transmission duty cycle approaching 10% let alone 100%, the way in which the DigiMesh network is created and in turn how often a LittleBot is forwarding packets from other LittleBots varies. As a result the absolute worst-case scenario of constantly transmitting is taken in calculating the power consumption. From this, a Standard 4-wheel LittleBot can be expected to consume up to 8.56 W under normal operating conditions. In order to achieve the desired two hours, it follows that a bat-

tery capacity of at least $8.56 \times 2 \approx 17.12$ Wh or higher is required. As the power consumption of a Junior LittleBot only differs from that of the Standard in its two less motors, the power consumption of a Junior LittleBot is therefore $8.56 - \frac{2.96}{2} = 7.08$ Watts. As a result, a Junior LittleBot must be equipped with a battery of at least $7.08 \times 2 \approx 14.16$ Wh capacity.

For the Sentry LittleBot, a longer runtime is desired for long-term applications. Further, it is unlikely that the gas sensors need to be active constantly and their power requirement is based on intermittent use. As the gas sensor's heater must be powered for 30 seconds before readings stabilise, a 20% duty cycle representing an on time of one minute followed by an off time of four minutes is used as the duty cycle to calculate their power requirement. This provides 30 seconds of gas readings following the heat period, rather than a single reading. Similarly it is unlikely the GPS is required to be active constantly, and as a result is not included in the power requirements for the Sentry.

Table 4.3: Power consumption of Sentry LittleBot components

Device	Voltage	Current	Power
XBee (Transmitting)	3.3 V	215 mA	710 mW
Gas Sensors (4, 20% Duty)	5 V	150 mA	750 mW
ATMega328	3.3 V	20 mA	66 mW

Summing the power consumption of all the devices summarised in Table 4.3, a LittleBot Sentry can be expected to consume 1.52 Watts. Hence in order to achieve the minimum runtime of one day, a battery capacity of $1.52 \times 24 \approx 36.48$ Wh is required. Finally, power conversion efficiency is not taken into account in these calculations, and needs to be maximised as discussed later in this section.

4.6.2 Lithium Polymer Cells

At over 200 Wh/kg, lithium polymer cells have one of the highest energy densities of the various battery technologies. Costing around USD \$8 each, cells of sufficient capacity and compact size for use in LittleBot are relatively cheap, and are hence selected. Depending on the charge level, one lithium polymer cell has a voltage typically between 3.0 V and 4.2 V. This is insufficient for the voltage requirements of LittleBot as the LEDs and motors require a supply of at least 6 V, while the gas sensor and camera require 5 V. An inverter could be utilised to step up the voltage, however this is unnecessarily complex and impractical given the relatively high power consumption of the higher voltage devices. Instead two lithium polymer cells are connected in series for a power supply that varies between 6.0 V and 8.4 V which can easily and efficiently be stepped down to other voltages as necessary. On the mobile LittleBots, two 2.7 Ah cells are employed for a battery capacity of $3.7 \times 2.7 \times 2 \approx 20.0$ Wh, exceeding the minimum of 17.12 Wh calculated previously. For a Sentry LittleBot, two physically larger 5.2 Ah cells are employed to obtain a battery capacity of $3.7 \times 5.2 \times 2 \approx 38.5$ Wh, again exceeding the required 36.48 Wh minimum previously calculated to meet the runtime goal.

To use lithium polymer cells safely, adequate protection against overcharging, over-discharging, and short-circuiting of each cell should be implemented. Without protection, any one of these conditions can lead to self-combustion of the cell, causing a fire or explosion. Protection ICs for lithium polymer cells and battery packs provide an inexpensive method of achieving this. For LittleBot, the Seiko S8232 battery protection IC [49] in combination with a pair of MOSFETs is utilised to protect the cells. This IC is designed for two cells in series, protecting each from overcharging beyond 4.35 V, discharging below 2.30 V, and limits the maximum discharge current to 3.0 A. With all LittleBot electronics expected to have a maximum current draw no greater than 2 A, this 3 A limit will not inhibit normal operation. As the both the 2.7 Ah and 5.2 Ah cells are rated for a continuous

At a total cost of \$2.59 for all the components required in implementing the S8232 as shown in Figure 4.14 [49], this expense is justified. Protection against short circuits is only provided between the *EB+* and *EB-* terminals of the configuration, meaning short-circuits of the individual cells before their connection to the S8232 remain unprotected. As a result the battery protection circuitry is placed on a PCB separate from the rest of LittleBot electronics, allowing it to be placed in close proximity to the lithium polymer cells.

Shown in Figure 4.15 and measuring 40.5×6.9 mm, the battery protection PCB employs two JST PH connectors that have been used throughout the LittleBot electronics. The three-circuit connector attaches to the battery cells, and the two-circuit to the SwitchCraft combination jack. Through this the cells can be recharged using the external charger described, or the LittleBot electronics powered on by inserting the power key.

Figure 4.15: Battery protection PCB

Once wired to the JST connectors, the PCB is insulated with heat-shrink to prevent any accidental short-circuiting of components or connection points on the board. In addition to the over-current protection provided by this battery protection PCB, a 2 A PTC fuse is added to each LittleBot PCB directly after the connector to the battery PCB for increased protection.

Finally, it is desired for the cells to be recharged without being removed from the LittleBot, meaning they have to be charged in their series configuration. If the internal resistance of two lithium polymer cells in series is significantly different, it follows that they will charge and discharge at different rates. As the cells are to be charged using a smart charger that is only capable of detecting the series voltage and not the individual voltage of each cell, this difference in internal resistance could lead to the voltage difference between the cells increasing over time with each recharge. In such an event the capacity of the pack will eventually become diminished or unusable, as when one cell becomes fully charged (or even overcharged) before the other, the S8232 will prevent further charging of the two cells in series. While this cannot be avoided entirely as no two cells have an identical internal resistance, the effect can be delayed such that the lifetime of the cells is exceeded before the condition occurs. By pairing cells that have a near-identical internal resistance, they will charge and discharge at similar rates. The pairs of cells used in each LittleBot are selected so that the internal resistance between them is less than $0.1 \text{ m}\Omega$, minimising the likelihood of a LittleBot being affected by this condition and maximising the useful life of a battery pack.

4.6.3 DC-DC Conversion

Although the motors and LED drivers have a wide operating voltage range that allows them to be powered directly from the 6.0 V to 8.4 V provided by the lithium polymer cells, the XBee and TX5823 modules require 3.3 V, while the gas sensor's heater element and camera require 5 V for correct operation. To supply these from the higher variable output of the lithium polymer cells, power converters are required. As one of the key considerations in the design of LittleBot is operating runtime, power wastage needs to be kept to a minimum and efficiency of these conversions maximized.

Referring back to Table 4.2, up to 235 mA at 5 V and 555 mA at 3.3 V are required by the devices of a mobile LittleBot. Linear regulators are inefficient, as the energy wasted is effectively equal to the voltage difference between the input and output, multiplied by the current supplied. Worst-case scenario when the battery is full and all devices are active, $8.4 - 5 \times 0.235 = 0.8$ Watts of energy would be wasted by using a linear regulator to supply the 5 V devices, and $8.4 - 3.3 \times 0.555 = 2.8$ Watts wasted in supplying the 3.3 V devices.

For the 5 V devices an LDO (low-dropout) regulator is employed. Capable of supplying up to 300 mA, the regulator is sufficient in meeting the maximum 235 mA required by 5 V devices. Other than input and output decoupling capacitors for stability, the regulator requires no other external components and is suitable for the application in a mobile LittleBot. For the increased current requirements of the 3.3 V devices, a Texas Instruments PTH08080 adjustable switching regulator module is employed instead of a linear regulator. Capable of supplying up to 2.25 A with an efficiency of at least 90% over a 4.5 V to 18 V input voltage range, the PTH08080 is superior in efficiency in comparison to a linear regulator. Although a switching regulator could be employed to supply the 5 V device, in quantities of 500 an LDO regulator costs approximately USD \$0.20, and the PTH08080 USD \$6.20. As a result the peak 0.8 Watts of energy wasted

for the 5 V devices is considered an acceptable compromise in maintaining the low cost objective, with the higher efficiency PTH08080 an unjustified expense.

Figure 4.16: Schematic of the PTH08080 for 3.3 V

While the PTH08080 requires only three external components for correct operation, extra capacitance on the input and output is added as shown in Figure 4.16. This increases the stability of the regulator and reduces high frequency noise. The value for the resistor R_4 that sets the output voltage is determined using the equation $R_{set} = 10000 \times \frac{0.891}{V_{out}-0.9} - 1820$ [50], where V_{out} is the desired output voltage. As 3.3 V is desired, $R_{set} = 10000 \times \frac{0.891}{3.3-0.9} - 1820 = 1892.5 \Omega$. The closest value resistor available is 1.87 k Ω and as the precision of this resistor affects the output voltage achieved, a 1% resistor is used to minimise further deviation from the desired 3.3 V. Should any of the electronics malfunction and create a temporary short, the PTH08080 features non-latching over-current protection and will automatically reset. Similar to the over-temperature protection of the DRV8835 motor drivers, this increases the robustness of LittleBot and minimises the likelihood of a LittleBot experiencing electronic malfunction.

Referring back to Table 4.2, the efficiency of the power conversions in a worst case scenario when the batteries are full must be taken into account. The employment of the 5 V regulator increases the power consumption for the 5 V devices to a maximum of $8.4 \times 0.235 = 1974$ mW, and the consumption for the 3.3 V using the PTH08080 at 90% efficiency to $\frac{3.3 \times (320+215+20)}{0.9} = 2035$ mW. Thus the total power consumption for a Stan-

Standard LittleBot is $P_{3.3V} + P_{5V} + P_{Motors} + P_{LEDs} = 1974 + 2035 + 2960 + 2590 = 9559$ mW, and $1974 + 2035 + \frac{2960}{2} + 2590 = 8079$ mW for a Junior. To maintain the goal of a two hour runtime, the battery capacities required become $9.559 \times 2 = 19.1$ Wh and $8.079 \times 2 = 16.2$ Wh for the Standard and Junior LittleBots respectively. As these are both less than the 20.0 Wh combined capacity provided by the lithium polymer cells employed, the losses resulting from DC conversion are tolerable.

While the losses resulting from use of a linear regulator to supply the 5 V on a mobile LittleBot are acceptable, a Sentry LittleBot outfitted with four gas sensors on the GPIO ports results in up to 600 mA at 5 V being required. Using a linear regulator to supply this would result in up to $(8.4 - 5) \times 0.6 = 2.04$ Watts of power wastage, a significant amount. Instead, a second PTH08080 regulator is employed with the value for the voltage setting resistor in this case found to be $R_{set} = 10000 \times \frac{0.891}{5-0.9} - 18200 = 360 \Omega$, and again a 1% precision resistor is used to minimise output voltage error. Taking into account the efficiency of the PTH08080s on the Sentry, the power consumption is re-evaluated and found to be $\frac{3.3 \times (215+20)}{0.9} = 862$ mW and $\frac{5 \times 150 \times}{0.9} = 833$ mW for the 5 V and 3.3 V devices respectively. With the Sentry consuming a calculated 1.7 Watts including the conversion efficiencies, the 38.5 Wh battery capacity is just below the $1.7 \times 24 = 40.8$ Wh theoretically required, and the one day minimum runtime expected to be maintained.

As explained previously, control of power to the gas sensors on a Sentry is desired to allow their duty cycle to be controlled and power saved where possible. The design of the PTH08080 allows it to be shutdown through grounding of the inhibit pin that had previously been left unconnected in the 3.3 V applications. To minimise component count and simplify the design of the PCB, the 5 V power to all the GPIO ports and in turn any attached gas sensors is controlled through enabling and disabling of the PTH08080 using its inhibit pin instead of a MOSFET for each GPIO port.

Figure 4.17: Schematic of the PTH08080 for 5 V

Illustrated in Figure 4.17, the inhibit pin of the PTH08080 supplying 5 V is attached to an N type MOSFET. The gate of this MOSFET is controlled by the ATmega328, and as seen previously, effectively allows control of the connection to ground. Hence by driving the gate of MOSFET Q2 high, the inhibit pin of the PTH08080 will be grounded and the device shutdown. In this state the PTH08080 is effectively off, and consumes less than 1 mA.

4.6.4 Battery Monitoring

In order for an operator to determine the remaining runtime of a LittleBot, an indication of the battery level must be available to them. Typically a Coulomb counter is implemented with lithium polymer cells, integrating the current entering and leaving the battery cells to allow the amount of charge remaining to be calculated. However, this approach requires ongoing calibration, and given the intended application and disposable nature of LittleBot, is unnecessarily complex. A simpler approach is to measure the voltage of the battery, as the open-circuit voltage of a lithium polymer cell is non-linearly proportional to the charge remaining in the cell. This can be used to provide an estimation of the remaining charge and although less accurate than a Coulomb counter, suffices in providing a rough estimate of the battery level.

A typical LiPo cell measures approximately 4.2 V at full charge, and 3.0 V when depleted. It follows that LittleBot's battery pack of two LiPo cells in series measures approximately 8.4 V at full charge and 6.0 V when

depleted. As the maximum analogue reference voltage that can be used for the ATmega328's ADC is limited by design to the 3.3 V VCC powering the IC, the ADC cannot directly measure the much higher battery voltage and attempting to do so would destroy the microcontroller. In order to measure the voltage of the cells using the ADC of the ATmega328, a voltage divider is applied.

To select values for the voltage divider resistors, the current through the divider is first specified as 100 μ A to limit current draw. With a maximum input voltage of 8.4 V, the total resistance of the voltage divider is therefore $R_T = \frac{V_{in}}{I} = \frac{8.4}{100 \times 10^{-6}} = 84 \text{ k}\Omega$. While the ADC can measure voltages up to 3.3 V, a maximum output of 2.9 V from the voltage divider is selected in order to provide headroom in the case of unexpected events such as overcharging. As $V_{out} = \frac{R_2}{R_1 + R_2} \times V_{in}$ and $R_T = R_1 + R_2$, then $R_2 = \frac{2.9}{8.4} \times 84000 = 29 \text{ k}\Omega$ and $R_1 = R_T - R_2 = 84000 - 29000 = 55 \text{ k}\Omega$.

The closest available resistor values to those calculated for R_1 and R_2 are 28.7 k Ω and 54.9 k Ω respectively. With 1% tolerance and a 63 mW power dissipation rating in a 0603 SMD package to minimise board space utilisation, these resistor values yield a current of $I = \frac{8.4}{83600} = 100.4 \text{ }\mu\text{A}$. Although it is clear that the power dissipation ratings of the resistors will not be exceeded in this voltage divider setup, for completeness this is verified. From Joule's first law, the power P consumed by a resistor of resistance R ohms is calculated as $P = I^2 R$, and from Kirchhoff's laws, the current through both of the resistors is 100.4 μ A as they are in series. As such, the power dissipations are found to be

$$P_{dR_1} = (100.4 \times 10^{-6})^2 \times 28700 = 2.893 \times 10^{-4} = 0.2893 \text{ mW} \quad (4.1)$$

$$P_{dR_2} = (100.4 \times 10^{-6})^2 \times 54900 = 5.534 \times 10^{-4} = 0.5534 \text{ mW} \quad (4.2)$$

for R_1 and R_2 respectively, which are well below the 63 mW ratings of the resistors. Finally, the output voltage of the divider is calculated as

$$V_{out} = \frac{28700}{(28700 + 54900)} \times V_{in} = \frac{287}{836} \times V_{in} \quad (4.3)$$

Hence $V_{out} = \frac{287}{836} \times 8.4 = 2.88$ V when the battery is full, and $V_{out} = \frac{287}{836} \times 6.0 = 2.06$ V when the battery is discharged, both well within the reference voltage of the ATmega328's ADC.

4.6.5 Charging

The cells employed can be recharged at a rate of $0.5 C$, where C is the capacity of the cells in ampere-hours. For the mobile LittleBots this is $0.5 \times 2.7 = 1.35$ A while for the Sentry it is $0.5 \times 5.2 = 2.6$ A. As the cells are in series and each adequately protected against the fault conditions mentioned previously, the battery does not need to be removed from LittleBot for recharging. Instead it can be recharged using an 8.4 V smart charger through the external combination jack previously described. To maintain simplicity and prevent confusion to users, a single 1.2 A charger [51] is selected. The constant 1.2 A output is less than the maximum charge rate for the cells, and can therefore be safely used for the cells in both the mobile and Sentry variants.

Figure 4.18: LittleBot's 1.2 A charger

Illustrated in Figure 4.18 with the SwitchCraft three pin plug fitted, the charger operates in two stages. Initially the voltage applied to the cells is increased until a charge current of 1.2 A is reached. As the difference in voltages between the charger and cells directly correlates to the charge

current, the charger can adjust the voltage to maintain this constant current until the voltage reaches 8.4 V. Once the charge voltage reaches 8.4 V it is maintained until little current is flowing into the cells, indicating they are full. At this point the indicator on the charger changes colour, representing charging is complete. Discussed in the next chapter, this charger is also employed for the Controller.

Chapter 5

Controller

For independent piloting of mobile variants and visual display of all LittleBot sensor information to an operator, a control device is required. The Controller participates as another node in the LittleBot DigiMesh network, receiving response packets from all LittleBots, and transmitting control packets to a selected LittleBot for piloting and manipulation of peripheral power statuses. To display the video feedback from a mobile LittleBot that facilitates their piloting, the Controller must include a 5.8 GHz FM receiver and video display. Additionally a graphic display is required to present the Controller and LittleBot's statuses, buttons to select and toggle features on a selected LittleBot, and a joystick for piloting mobile variants. Finally, to store data reported by a LittleBot that isn't actively selected on the Controller, an SD card is required with an RTC (real time clock) for time and date stamping of the data. This chapter explains the design and construction of the LittleBot Controller.

5.1 Mechanical Construction

Similarly to the LittleBots designed in Chapters 3 and 4, the design of the Controller is divided into mechanical design and electronics. This section discusses the Controller chassis for housing and mounting the required

components, designed for manufacturing using a combination of 3D printing and laser cutting.

5.1.1 Frame & Construction Method

The 3D printing technology available throughout the course of this project is capable of manufacturing parts up to $12 \times 12 \times 12$ cm in size, while the laser cutter is capable of manufacturing parts from sheets of suitable material up to $600 \times 400 \times 10$ mm. It is infeasible to fit all the required Controller components into a chassis bound by the 3D printer dimensions in a functional manner. Instead of a single 3D printed chassis, a frame consisting of six 3D printed pieces that assemble together is designed with laser cut acrylic plates attached to the front and back of this frame. Slots and tabs between the 3D printed pieces are used for alignment, with laser cut acrylic gaskets added for aesthetic purposes.

Figure 5.1: Partially de-constructed controller frame

Demonstrated in Figure 5.1 with one of the corner pieces removed, the remaining five 3D printed pieces demonstrate how the frame of the Little-Bot controller is formed. Two holes in each of the pieces provide a point for front and back plates to be attached, and the 3D printed frame pieces designed so that these plates are recessed to be flush. The holes on the

front acrylic pieces are 3 mm in diameter, while the matching holes in the back piece are sufficiently smaller to allow them to be tapped out with a thread for 3 mm bolts. With the frame assembled, 40 mm long bolts are inserted into each of the holes on the front plate, through each hole in the frame, and secured into the corresponding thread tapped out of the back plate.

The exact dimensions of the Controller are ultimately dictated by the components mounted to the Controller, however an operator must be able to easily access control inputs which are placed on the front plate as described in the following sections. The most appropriate place for the XBee and video receiver antennas are at the top of the Controller. To allow antennas to be removed for compact transport of the controller, RP-SMA connectors are used for their attachment and are mounted into the centre top piece of the 3D printed frame.

Figure 5.2: Combination jack and antenna connectors

For charging and to turn the Controller on and off, the same combination jack employed on each of the LittleBots is used. Illustrated in Figure 5.2, the combination jack is mounted between the two RP-SMA connectors for antennas. Finally, with a socket on the Controller PCB for an SD card present, a way of inserting and removing this SD card from the PCB that is contained within the Controller enclosure is required. A cut out is designed in the middle bottom piece to align with the SD socket on the PCB when assembled, as pictured in Figure 5.3 with an SD card inserted.

Figure 5.3: SD card access

The alignment of the socket and PCB within the frame is designed to ensure an SD card is completely recessed within the frame of the Controller when inserted, reducing the risk of accidental removal of the card during use. To facilitate easier removal of the card, a circular sweep cut is added to this slot.

With all necessary features for the frame completed, the rest of the components are mounted to the front and back plates that secure the frame.

Figure 5.4: Components mounted to the front plate

Demonstrated in Figure 5.4, the electronics, inputs and displays are mounted behind the front plate of the Controller. Each component's mounting is described further in the subsequent sections of this chapter.

5.1.2 Video Display & PCB Mounting

Video feedback from each of the LittleBots is displayed on an Innolux 7" TFT LCD display, the details of which are discussed further in Section 5.2. With the front plate of the Controller manufactured from laser cut transparent acrylic, the most suitable method of mounting this TFT LCD display is behind the front plate. Two pieces equal in thickness to the display are designed and manufactured through 3D printing to form a mounting frame. With the LCD framed by these pieces centred behind the front plate of the Controller, an acrylic plate is bolted from behind the display into the front plate of the Controller through the frame. This effectively locks the LCD display and framing pieces in the centre of the Controller, visible from the front through the transparent acrylic plate.

Figure 5.5: Video LCD with mounting frame

Illustrated in Figure 5.5, the LCD display is secured by the frame and acrylic. With both pieces of acrylic being 3 mm thick and the frame for the LCD display 6.2 mm thick, screws no longer than $3 + 3 + 6.2 = 12.2$ mm long can be used to secure the assembly. Anything longer than this would protrude from the front of the Controller's front plate. Therefore 12 mm

long 0.5 pitch M2 screws are used, with the holes in the front Controller plate made to be 1.5 mm and tapped out with a suitable thread.

For the display to operate correctly, a display driver PCB is required. The flex cable pre-attached to the display for connection to the driver PCB is short and cannot easily be extended. To accommodate this the display driver PCB is mounted directly behind the display itself, with the back acrylic plate of the display mounting assembly serving as the display driver PCB's mounting point. Threaded holes that match the mounting holes of the driver PCB are added, along with an opening aligned with the LCD display's flex cable allowing the cable to pass through to the connector on the driver PCB. The driver PCB is spaced away from the acrylic back plate, as there are components on the back of the board preventing it from being mounted flush.

As the second PCB discussed later in Section 5.2 must also be mounted within the Controller, it is designed to have the same footprint and mounting holes as the display driver PCB allowing it to be mounted directly above. Additional spacers are placed between driver PCB and the main Controller PCB, and both are bolted into the backing acrylic of the display mount.

Figure 5.6: Stacked PCB mounting

With the display mounted behind the front plate of the Controller, both PCBs are stacked behind the display as demonstrated in Figure 5.6. Again screws of suitable length to not breach the other side of the mounting acrylic are used for this mounting.

5.1.3 OLED Display

To display the current status of the Controller and a selected LittleBot, a dot matrix OLED (organic light-emitting diode) display is employed. With the front plate of the Controller fashioned from transparent acrylic, a cut out is not needed for the OLED display to be visible allowing it to be mounted behind the front plate in the same manner as the video display. The OLED display selected comes mounted on a PCB with holes for mounting already available. Matching holes are added to the design of the front plate, which are tapped out after being laser cut to accommodate a 0.5 mm pitch M2 bolt. As the front of the OLED display itself is not flush with the PCB it is mounted on, spacers are used to space the display away from the front plate, preventing the glass front of the OLED display from being crushed against the front plate of the Controller.

Figure 5.7: OLED mounted to the front plate

Illustrated in Figure 5.7 with the initialisation graphic displayed, the OLED display is mounted behind the front plate of the Controller. As the PCB of the OLED display is 1.6 mm thick, the spacers 2 mm, and the front plate acrylic 3 mm, it follows that the screws used to secured the display to the front plate should be less than $1.6 + 2 + 3 = 6.6$ mm long so that they do not protrude from the face of the Controller. To to ensure the display is mounted securely, screws of 6 mm in length are used. Wires

are soldered to the appropriate connections on the OLED PCB, with a JST PH series connector crimped to the other end for attachment to the mating connector on the Controller PCB.

5.1.4 Joystick

To allow a mobile LittleBot to be piloted by an operator, an analogue thumbstick is included on the controller. Panel and chassis mount joysticks are prohibitively expensive therefore a cheap thumbstick that requires mounting on a custom PCB is selected. The custom mounting PCB is created with four mounting holes and a JST PH connector for attachment to the main electronics.

Figure 5.8: Thumbstick installed on its mounting PCB

Pictured in Figure 5.8 with a cable attached, the thumbstick is installed on the PCB and mounted to the front plate of the Controller from behind. As the majority of people are right handed (approximately 85%) the thumbstick is placed on the right hand side of the Controller above the OLED display such that an operator's hand can grip the Controller while manipulating the thumbstick with their thumb.

Figure 5.9: Thumbstick behind the front plate

To prevent the shaft of the thumbstick hitting the edge of the front plate during use, the PCB must be spaced away from behind the front plate so that the centre of the thumbstick shaft is aligned with the front plate. A custom spacer is empirically designed with a height that allows the thumbstick to move freely without interference with the front plate, and is manufactured through 3D printing. Shown in Figure 5.9, the thumbstick is successfully mounted with the spacer, again using the longest screws possible that do not breach the face of the Controller plate.

5.1.5 Buttons & Labels

Six buttons are present on the controller, allowing an operator to select which LittleBot they are operating, as well as to provide control over what features are enabled on the selected LittleBot. The buttons employed include an LED that can be illuminated to indicate the status of a particular feature, and are through-hole mounted in the front plate using a retaining nut.

Figure 5.10: Control buttons with labels

As shown in Figure 5.10, the buttons are mounted on the left side of the controller next to the display in a 2×3 layout. The top two buttons are assigned the functionality of selecting a LittleBot, and the remaining four correspond to the power control of peripherals on a LittleBot as previously described in Chapter 4. Labels are laser engraved above each of the buttons on the front plate, with the engravings painted post-manufacturing to make them more prominent.

5.2 Electronics

A microcontroller is required on the Controller to handle the inputs from and outputs to other devices, as well as to control an XBee module to facilitate communication in the LittleBot DigiMesh network. Additional components such as a power converter and an FM demodulator are required. This section discusses the Controller PCB design and implementation.

Figure 5.11: Rear of the Controller PCB

Displayed in Figure 5.11 is the rear of the resultant LittleBot Controller PCB. As previously discussed, the PCB is designed to have the same dimensions as the video display driver PCB so that both may be mounted in a stacked format behind the video display. Components of sufficiently low profile such as the ATMega2560 are mounted on the back of the PCB, while taller components such as connectors and the socketed XBee are mounted on the front where there is increased clearance.

5.2.1 ATmega2560 Core & XBee

The ATmega328 employed in each of the LittleBots features 2 KB of SRAM that is the temporary memory used for the storage of variables during program execution. While previously the only major occupier of this SRAM had been the 128 byte UART buffer that prevents packets received by the XBee being overwritten before they can be processed by the microcontroller, the additional peripherals present on the Controller discussed in Sections 5.2.2 and 5.2.5 require large amounts of SRAM. The buffer for the 128×64 pixel OLED display must be calculated and updated in one pass, which requires $128 \times 64 = 8192$ bits or 1024 bytes of SRAM on the driving microcontroller to compose this display buffer. Additionally, the SD card is addressed in 512 byte blocks, and these must be calculated in SRAM before being written. Finally the size of the UART buffer for the XBee should be maximised, as the higher volume of packets expected to be received by the Controller may result in packets being lost if the buffer becomes full.

Taking into account the increased memory requirements of the peripherals on the Controller, the 2 KB of SRAM provided by the previously utilised ATmega328 is insufficient. However, with the desire for all LittleBot software to be written in the Arduino IDE, another megaAVR microcontroller with increased SRAM is sought. The ATmega2560 features the same megaAVR core as the ATmega328, but with 8 KB of SRAM and 256 KB of flash memory. As it is difficult to ascertain the amount of flash memory required to store a program before it has been written, a test program utilising function calls from all libraries expected to be required for the Controller is compiled for the megaAVR platform. The compiled test program requires 48 kB of flash memory, less than a quarter of the flash memory available, making the ATmega2560 a suitable microcontroller for the requirements of the Controller PCB.

Low-speed and high-speed variants of the ATmega2560 are available, with the 16 MHz high-speed version requiring a VCC between 4.5 V and 5.5 V, and the low speed 8 MHz accepting a wider 1.8 V to 5.5 V supply.

As the ATmega2560 is interfacing with the XBee and SD card, a common VCC for all three is preferable to avoid the need for logic level converters. The XBee limits the VCC range to between 2.7 V and 3.6 V, while the SD can operate on either 1.8 or 3.3 V. As a result 3.3 V is selected as the VCC, and the low speed variant employed with an 8 MHz crystal for clocking.

Interfacing of the XBee and the ATmega2560 is performed in the same manner as previously on a LittleBot, with the XBee attached via a UART to allow the API mode required for DigiMesh to be used. However the ATmega2560 features three more hardware UART ports than the ATmega328 for a total of four. Some of these UART ports are on a limited number of pins that can be configured as external interrupts, which as discussed later, are all required for use on the button inputs. As a result the XBee is assigned to a UART on pins that are not capable of being configured as external interrupts. Finally power is supplied to the XBee with a 47 μ F ceramic capacitor in close proximity for decoupling, and the XBee mounted using sockets to allow it to be removed for external reconfiguration of its firmware.

In addition to pins being assigned to peripherals in the sections following, another UART without external interrupt capabilities is broken out to a connector as an auxiliary UART port. Not intended for use by an operator, this UART can be used for debugging during the software design stage through serial printouts during code execution. Such printouts enable the cause of runtime errors to be more easily traced, as well deducing what functions may require optimisation by printing out execution times for each. To this port a UART to USB adaptor can be connected, and the printouts displayed in a terminal on a computer.

5.2.2 SD Card

For long-term storage of the data provided by each of the LittleBot nodes in the network, an SD card is included on the Controller. SD cards have

become prominent in modern times as the memory card standard, and can be easily interfaced with the ATmega2560 through its SPI bus. Once formatted with the FAT standard, software libraries can be used to address the card to create files, and to read and write data from the card. Implementation of the SD card interface begins with a socket, as in order for the data to be retrieved and analysed, the card must be able to be removed from the controller. The socket is mounted at the bottom of the PCB, such that it is close enough to allow the card to be inserted and removed from outside the Controller chassis through the designated opening discussed in Section 5.1.

The relevant pins of the socket are wired to the appropriate SPI data lines of the ATmega2560, and 3.3 V, the required VCC for an SD card, attached with a ceramic capacitor included in close proximity for decoupling. Finally the SD card socket features two pins that allow detection of when a card is inserted and determine if that card has its write protection switch enabled. These pins are either floating or grounded depending on the card insertion status and are attached to pins on the ATmega2560 with internal pull-ups enabled in software. This facilitates detection of when these pins are grounded and hence the relevant insertion and write protection statuses of the SD card.

5.2.3 Real Time Clock

In order for data reported by LittleBots to the Controller to be stored with a timestamp on the SD card, a method of time keeping is required. The ATmega2560 keeps count of number of clock cycles since power up, and can be used to derive time based on the clock frequency and start up time. However this technique would require the user to continually set the time whenever power was lost, and this is an undesirable approach. Instead, an RTC (real time clock) with independent backup battery is employed. The Maxim DS3231 RTC is selected as it operates on 3.3 V and communi-

cates using the I2C protocol, making it compatible with the ATmega2560. As the internal software configurable pull-up resistors of the ATmega2560 satisfy the design requirements specified by the I2C protocol for correct operation, the SDA and SCL lines of the two ICs can be directly connected with no other supporting circuitry necessary.

Figure 5.12: DS3231 RTC implemented with battery backup

Shown attached to the DS3231 in Figure 5.12, the CR2032 3 V button cell *BAT1* is employed as the backup battery for the DS3231. When the 3.3 V VCC to the IC is lost (for example the Controller is turned off or the main batteries go flat), the CR2032 allows the DS3231 to continue to keep time for up to 5 years. Upon the Controller being turned on again, the software running on the ATmega2560 will communicate with the DS3231 and synchronize the system time. To reset the date and time of the DS3231, the ATmega2560 is programmed with software that listens for the current time and date encoded in serial data on the auxiliary UART header of the Controller PCB, which then sets the internal time of the DS3231 to match.

5.2.4 User Input

As discussed, user input is provided through six buttons and a thumbstick. The buttons allow a user to select a LittleBot and toggle the power status of peripherals, while the thumbstick is used to pilot a mobile LittleBot. For both the buttons and the thumbsticks, JST PH connectors are used for connection to the main electronics.

Thumbstick

Attached to the thumbstick are two 10 k Ω potentiometers and a switch, with one of the potentiometers attached to the X-axis of the stick, and the second to the Y-axis. The button is positioned beneath the thumbstick, and can be activated by pressing down vertically on the thumbstick in the Z-axis. As the thumbstick is spring loaded in all axes, it returns to centre when no pressure is applied and therefore the wipers of the attached potentiometers will be approximately centred.

To read the current position of the thumbstick, each potentiometer can be set up as a voltage divider. As the position of an axis varies due to displacement placed on the thumbstick, the voltage at the wiper of the corresponding potentiometer will vary between 0 V and the voltage applied to the potentiometer. Hence by applying 3.3 V, the ATmega2560's 10-bit ADC can sample this voltage and return digital values between 0 and 1023 from each potentiometer. This can then be used in software to calculate the duty cycles for each of the motors on a mobile LittleBot, allowing it to be piloted.

Buttons

The six buttons described previously feature a normally open switch, as well as an LED illuminated indicator ring. On the two buttons that let a user cycle through the units, this LED ring is not utilized. However, for the four buttons assigned to toggle the power to the gas sensor, camera, lights and motors or GPS on a LittleBot, the LED ring is used to indicate to a user whether that peripheral is enabled or not.

To detect whether a button has been pressed, one of its contacts is attached to ground and the other to a pin of the ATmega2560 that supports external interrupts. Within the software, the internal pull up on the relevant pin is enabled and an interrupt attached to detect a falling edge. As the button is normally open, the pin will be pulled high by the AT-

Mega2560's internal pull up. However when the button is pressed, the pin is connected to ground until the button is released and a falling edge detectable on that pin.

Figure 5.13: Buttons connectors with decoupling

To minimize the effects of switch bounce inherently present in this situation, a 1 µF capacitor is attached between the interrupt and ground pins of each button connector as shown in Figure 5.13. The capacitor takes time to be charged by the pull up resistor on the line, reducing the number of calls to the interrupt routine for a single press. As switch bounce cannot be removed completely in this manner, a single press will still result in multiple calls to the interrupt service routine. The software running on the Controller is designed to ignore these by setting a flag in the interrupt service routine instead of calling a function, with the flag handled as part of the main routine discussed later in Chapter 8.

To illuminate the button LED rings on demand, a controllable power supply to each is required. As the LED can share the ground connection of the button itself and contains an internal current limiting resistor, all that is required is to supply the required 3 V that will illuminate it. These LEDs require 6 mA of current, while each pin of the ATmega2560 can supply up to 20 mA. As LEDs are tolerant of less than ideal voltages, each LED ring is directly driven by a pin of the ATmega2560 with 3.3 V to avoid the need for a separate switch such as a MOSFET.

5.2.5 OLED Display

To allow an operator to select which LittleBot to control and view the reported sensor data, a digital display is required. A 128×96 pixel monochrome OLED display driven by an SSD1306 IC is selected, and can be controlled using either SPI or the I2C protocols. Instead of attaching the OLED Display's SSD1306 controller to the already utilized SPI or I2C buses, bit-banged software SPI control is selected. The libraries available for controlling the SSD1306 support this mode by default, allowing the OLED display to be attached to any of the ATmega2560's pins. As a result, the display can be manipulated without needing to worry about interfering with operations involving the SPI attached SD card or I2C attached RTC.

Figure 5.14: OLED connections

Depicted in Figure 5.14, the JST PH connector *P13* provides six data lines between the SSD1306 and ATmega2560, while *P16* provides a decoupled 3.3 V supply to power the display. This satisfies all the requirements to successfully manipulate the display over the selected software SPI using the ATmega2560, with software and libraries discussed in Chapter 8 manipulating the ATmega2560's data lines appropriately.

5.2.6 Video Receiver & Display

To receive the 5.8 GHz video transmitted by the TX5823 modules in the mobile LittleBot, a Boscam RX5808 5.8 GHz receiver is employed. Performing the opposite function to the TX5823, the RX5808 module demodulates the signal back to the PAL encoded signal of a LittleBot camera.

Similarly to the TX5823's transmission frequency selection, the RX5808's reception frequency is specified through selective grounding of three pins on the module. The mapping of the grounded pin combination to reception frequency is the same as the TX5823's for transmission frequency listed previously in Table 4.1. The ground connections to these pins are controlled using N Channel MOSFETs, with the gates of each MOSFET controlled by the ATmega2560. Ultimately this allows the reception frequency to be configured by the ATmega2560 so that it matches the transmission frequency of the currently selected LittleBot.

To display the LittleBot video feed to an operator, an Innolux 7" TFT LCD display is employed. As the display has a digital input requiring a LVDS (low voltage differential signalling) input, the PAL encoded composite output from the RX5808 cannot be directly supplied to the display. The display also features a CCFL (cold-cathode fluorescent lamp) backlight to illuminate the LCD, which requires a supply of the order of hundreds of volts to operate. Various driver PCBs are commonly available that act as an interface between various video sources and LVDS displays, and provide the required high voltage for the CCFL backlight. Instead of incorporating the relatively complex ICs for converting PAL to LVDS into the main electronics of the Controller, a compatible display driver board is instead selected.

Figure 5.15: KYV-N2 display driver PCB

Shown in Figure 5.15, the KYV-N2 display driver [52] features the necessary LVDS connector in the top right corner that is compatible with the flex cable of the display. This provides the correct video signal and the high voltage supply for the CCFL backlight to the display. With the display driver equipped with composite video inputs, it is capable of taking the PAL encoded video from the RX5808 and converting it to the digital signal required by the display.

Although the data sheet for the driver specifies a 12 V power input, reverse engineering of the board reveals a switching regulator that provides a constant output across a 4.5 V to 17 V input voltage. As a result, the display driver can be powered directly from the 6 V to 8.4 V variable supply of the lithium polymer cells, and no power conversion is required. On the Controller PCB, a connector for the display driver PCB is added. With the RX5808 included in the Controller PCB design, the demodulated video feed is fed from the RX5808 to one of the pins of this connector. A second pin of the connector provides decoupled raw power from the battery cells, while the two remaining pins of the connector provide a controllable ground through an N MOSFET. The gate of this MOSFET is driven by the ATmega2560, enabling control of the power to the display system so that power may be saved when video feedback is not required (or unavailable in the case of a Sentry LittleBot).

With control of the display power provided using an N MOSFET, power control of the RX5808 should also be implemented to save power when the display is disabled as the RX5808 would otherwise be needlessly consuming power. With the display driver PCB and RX5808 supplied with different voltages, they cannot share a common MOSFET for control of their ground connections. If a single MOSFET were used to disconnect both the display driver and RX5808 from ground, then their ground pins would be connected to one another when the MOSFET had its gate driven low to disconnect the power ground from both devices. As a result, the lower 5 V input to the RX5808 would provide a path for current

To avoid this issue, separate MOSFETs are used to control the RX5808 and display driver ground connections. Illustrated in Figure 5.16, MOSFETs $Q1$ and $Q2$ both have their gate attached to the *videoPower* net and are driven by the same pin of the ATmega2560. $Q1$ controls the *RX5808GND* net, in turn controlling all ground connections for the RX5808 and frequency selection MOSFETs, while $Q2$ controls the ground connection to the display driver PCB through connector $P9$. When the ATmega2560 drives the gates of the MOSFETs low, the grounds of the two devices are isolated from one another. As a result the power to these devices can be successfully controlled, and the battery life of the controller extended when video is not desired or unavailable on the currently selected LittleBot.

5.2.7 Power Supply

The power supply for the Controller follows the same design used for each LittleBot. Two 3200 mAh lithium polymer cells connected in series are selected as the power source that provide between 6 V and 8.4 V depending on the amount of charge remaining in the cells. With the cells adhered to inside of the back plate of the Controller for mounting, the same protection PCB used in a LittleBot is also used in the Controller to prevent the cells entering the previously discussed conditions that could cause them to combust. Similarly to the batteries on the LittleBots, the impedance of these cells is matched to be within 10 m Ω of one another. This prevents the previously described problems that may result from the cells charging and discharging at markedly different rates. With the selected cells, a battery capacity of $3.2 \times 3.7 \times 2 \approx 23.7$ Wh is realised.

Table 5.1: Power consumption of LittleBot Controller components

Device	Voltage	Current	Power
XBee (Transmitting)	3.3 V	215 mA	710 mW
ATMega2560	3.3 V	20 mA	66 mW
User I/O (Thumbstick/Buttons)	3.3 V	80 mA	264 mW
SD Card	3.3 V	100 mA	333 mW
OLED Display	3.3 V	30 mA	99 mW
RX5808	5 V	200 mA	1000 mW
Video Display & Driver	7.4 V	460 mA	3400 mW

Summing the peak consumption of all the Controller components listed in Table 5.1 yields a total maximum consumption of 5.54 Watts, resulting in the Controller being capable of at least 4 hours of runtime. While the display driver accepts an input voltage between 4.5 V and 18 V and hence can operate directly from the raw supply of the batteries, 3.3 V is required for the ATMega2560, XBee, SD Card, OLED Display and User

Input, and 5 V for the RX5808 5.8 GHz FM Receiver. Referring to Table 5.1 again, up to 445 mA at 3.3 V is required and 200 mA at 5 V. As it is desirable to minimise power wastage in order to maximise the runtime of the Controller, inefficient linear regulators are again dismissed in favour of two Texas Instruments PTH08080 switching regulators.

Figure 5.17: Controller power supply

Shown in Figure 5.17, the supporting circuitry used in the LittleBot Sentry for each regulator is repeated for the Controller, with decoupling capacitors and 1% tolerance resistors used to set the output voltage of each module accurately. The SwitchCraft socket acting as a charging port and power key on each of the LittleBots is employed on the Controller to serve the same purpose, enabling the chargers and power keys that are used for each of the LittleBots to also be used on the Controller. This increases the simplicity of the entire system by having a universal charger and power key for all devices, removing any possibility of confusion about which charger or key belongs to which device. Additionally as the combination jack features a pin that is directly attached to the power input to the Controller PCB, the runtime of the Controller can be further extended using an external battery, or an external power supply directly attached to the combination jack for an indefinite runtime.

Figure 5.18: Controller battery input and measurement

Depicted in Figure 5.18, the power attachment to the Controller electronics is through the JST PH connector *P12*, with the 2 A PTC fuse *F1* supplementing the battery protection PCB in providing protection against shorts. Finally the voltage divider formed by *R11* and *R12* allows the battery voltage to be measured by the ATmega2560, so that the charge level may be displayed to an operator.

5.3 Summary

Combining the hardware and electronics designed and constructed in the previous sections, the LittleBot Controller fulfils the independent control objective. Shown completed in Figure 5.19, an operator selects a LittleBot on the Controller using the two unit selection buttons highlighted in the top left. The selected LittleBot is shown on the OLED display in the bottom right, along with sensor information and the Controller status.

In addition to the antenna on the right for the XBee, the 5.8 GHz antenna on the left is included for the RX5808 to receive video from a selected Mobile Littlebot. When enabled on the Controller and LittleBot using the *Video* button, the video from the camera is fed to the Video display in the centre. Enabling the motors using the relevant button, an operator may use this video feed to pilot a Mobile LittleBot using the thumbstick. An operator may toggle the power of the lights or gas sensor on a LittleBot using the remaining two buttons, while the power key is used to turn the Controller on through the combination jack. By removing the power key the Controller may be recharged through the combination jack using the charger discussed previously in Chapter 4. Discussed in the next chapter, the XBees are configured and a protocol designed to provide the desired

command and control functionality, while Chapters 7 and 8 detail the required software written for the LittleBots and Controller.

Figure 5.19: The LittleBot Controller

Chapter 6

XBee Communication

With each of the XBee's paired with a microcontroller on a LittleBot or the Controller via UART, a protocol is devised for communication between LittleBot nodes in the same DigiMesh network. As part of the software written for the LittleBots and Controller discussed in Chapters 7 & 8, each of the microcontrollers are programmed to issue and handle XBee API requests and responses to and from the XBee over UART, implementing the LittleBot protocol over the DigiMesh network. This chapter details the design of the LittleBot protocol, configuration of the XBee modules, and how the API control of the XBee modules, mandatory for DigiMesh operation, is achieved on the microcontrollers of the LittleBots and Controller.

6.1 LittleBot's Protocol & Packets

Each of the XBee modules used has a unique, fixed hardware address to identify it. Once configured, the XBees automatically form a wireless mesh network allowing one node to send a packet to another simply by setting the destination address of the packet using API commands. As part of the DigiMesh protocol, other XBees may receive the packet and forward it on towards the intended recipient node. However, only the XBee of the recipient node with the matching hardware address will pass the packet

back through the UART to the paired microcontroller for processing. All other nodes utilised for forwarding the packet will not completely process the packet, and their respective paired microcontrollers will be unaware that forwarding even occurred.

6.1.1 Controller Association

As the number of LittleBots deployed is not constant and only limited to 16 by the Controller software discussed in Chapter 8, an association method is required to allow the Controller to discover all active LittleBots. The LittleBot protocol is designed such that when a LittleBot is first powered on, it is in a disassociated state. This means that it is not paired to a Controller, and will search for the Controller node in the LittleBot network. As a result, an *assocRequest* will be sent periodically until an *assocResponse* is received, being the first two types of packets created for the LittleBot protocol. The hardware address of the Controller XBee could be hard coded into the software of each LittleBot, giving a disassociated LittleBot a specific address to attempt to send an association request. However this means that if the XBee in the Controller or the Controller itself were to be replaced, all the LittleBots would have to have their software updated with the hardware address of the new XBee, as the existing Controller XBee address would no longer be valid. A simpler approach is for the *assocRequest* packets to be addressed as a broadcast to all nodes in the LittleBot DigiMesh network, and for only a Controller's microcontroller to process these and send a response.

As part of the XBee DigiMesh protocol, there is the reserved broadcast address `0x0000FFFFFFFFFFFF`. Any packet addressed with this as the destination will be delivered to all XBees as though it is intended for that node, and as a result that packet will be passed back through UART to all the paired microcontrollers. This address is subsequently used as the destination address for an *assocRequest* by a disassociated LittleBot, with

the Controller software configured to handle these requests and LittleBots configured to discard them. As part of the DigiMesh protocol, the hardware address of the sending node is automatically included in every packet and can be retrieved by the microcontroller from the extra headers as part of the packet delivery using the XBee API. Using this, the Controller can build a list of active LittleBots in the network as they are powered on and attempt to associate with the Controller. In order for the Controller to process subsequent data received from a LittleBot appropriately, the *assocRequest* packets must include the LittleBot type, and sensor configuration in the case of a Sentry LittleBot due to its increased number of configurable sensors.

```
struct assocRequest {  
    byte packetType;  
    byte unitType;  
    byte sensorConfiguration;  
};  
typedef struct assocRequest assocRequest;
```

Structure 6.1: *assocRequest* packet

The *assocRequest* packet displayed in Structure 6.1 is defined using structs. To simplify processing, all packets used in the LittleBot protocol have their first byte reserved to identify the type of packet, hence the *packetType* field. Additionally, the *assocRequest* includes a byte long field for the type of LittleBot, and a byte long field that represents the sensor configuration in the case of a Sentry LittleBot. When a LittleBot broadcasts its *assocRequest*, these fields contain the relevant information for that LittleBot. A receiving Controller processing the packet learns the type and configuration of that LittleBot from these fields, while the address of the LittleBot is extracted from the headers of the higher level DigiMesh packet it had been encapsulated in using API commands. The Controller then sends an *assocResponse* to the LittleBot using the address just extracted. Upon

receiving the *assocResponse* from the Controller, the LittleBot extracts and stores the XBee hardware address of the Controller from the *assocResponse* using API commands, before asserting an internal *associated* flag. Following a successful association, a Controller can command the LittleBot and correctly interpret received sensor data, while the LittleBot can address future packets directly to the Controller to which it is associated.

In addition to the *assocRequest* and *assocResponse* packets that facilitate association of a LittleBot with a Controller upon power up, a third type of association packet is required. In the event that LittleBots are associated with the Controller and the Controller is subsequently restarted, the LittleBots will still be in an associated state. While packets will still successfully be received by the Controller, the Controller will not know the type of LittleBot nor the configuration of the sensors as this information is included as part of the association request. As a result the packet cannot be processed correctly as that LittleBot variant is not known, while the correct type of command packet cannot be sent to that LittleBot resulting in loss of command. A third type of association packet, *assocInvalid*, is added to enable the Controller to inform a LittleBot that it has no record of it being associated. Upon receiving an *assocInvalid*, a LittleBot will reset its association status and repeat the association process with a Controller.

6.1.2 Command, Control and Feedback

Once a LittleBot is associated with a Controller, it periodically transmits sensor readings and battery levels. As the Standard and Junior LittleBots differ only in the lack of a DHT22 sensor on the Junior, the structure of the packet they transmit to the Controller is used for both and the DHT22 field left unpopulated in the case of the Junior. With the Controller mapping the address of a LittleBot to its variant during the association process just discussed, it can be configured to ignore the irrelevant DHT22 field in response packets received from Junior variants.

```
struct standardResponse {  
    byte packetType;  
    byte batteryLevel;  
    int gasLevel;  
    uint8_t DHT22[4];  
};  
typedef struct standardResponse standardResponse;
```

Structure 6.2: *standardResponse* packet

Shown in Structure 6.2, the *standardResponse* packet is used for feedback from Standard and Junior LittleBots. As previously mentioned, the first byte always defines the packet type, allowing the structure of the packet to be recognized and processed appropriately by a receiving microcontroller. The second byte is used to report the current battery status of the LittleBot, and as this will be displayed in 1% increments on the Controller, the $\frac{100}{2^8} = \frac{100}{256} = 0.39\%$ resolution provided by this byte sized field is adequate. As the gas sensors are sampled by the ADC of the microcontroller with a resolution of 10 bits, a byte sized field is inadequate for the gas sensor reading and an integer occupying two bytes used instead. Finally for the DHT22, the raw data from the sensor, excluding checksums, occupies four bytes. Arithmetic must be performed on this raw data in order to obtain the temperature reading in degrees Celsius and relative humidity as a percentage. However the output of the arithmetic would be of the *float* type and would occupy twice as much space as the raw data. Instead of performing the arithmetic on a LittleBot, the raw data is sent to the Controller and the arithmetic performed there so that only four bytes are required in the packet instead of eight.

As the Sentry can be outfitted with a greater number of sensors, a different packet structure is required resulting in the *sentryResponse* packet type shown in Structure 6.3. Both the battery level and DHT22 readings can be transmitted in the same manner used in the *standardResponse* structure. However as the Sentry can have up to four configurable sensors, the

four *integer* array *sensorReadings* is employed in place of the single field previously. Currently these sensors may be gas or PIR, however the range of supported sensors can be expanded through software additions.

```
struct sentryResponse {  
    byte packetType;  
    byte batteryLevel;  
    uint8_t DHT22[4];  
    int sensorReadings[4];  
    float gpsLat;  
    float gpsLon;  
};  
typedef struct sentryResponse sentryResponse;
```

Structure 6.3: *sentryResponse* packet

For a port fitted with a gas sensor, the raw ADC measurements can be directly assigned to the corresponding index in the *sensorReadings* array. Where a PIR sensor is fitted, the integers defined as part of the GPIO library (discussed later) representing the PIR's state are assigned instead. Finally float fields for the GPS latitude and longitude are present, matching the float output of the library handling the GPS data.

When an operator selects an active LittleBot, control packets must be sent to that LittleBot to allow the desired commands to be processed and executed. As previously, the Standard and Junior LittleBots can share the same control packet structure given their functional similarities, while the Sentry's greater variance in design warrants its own unique packet structure for Control. For the Standard and Junior LittleBots, Booleans to control the power to the motors, lights, video and gas sensor, and the direction of each motor must be communicated from the Controller to the LittleBot. Although the Boolean type exists in the Arduino IDE, it occupies a byte instead of a bit as would be expected. If this were to be implemented for each of the controls just described, six bytes would be required instead of six bits. As it is desirable to keep packet size and redundant data to a

minimum, a single byte *booleanFlags* is used, as shown in Structure 6.4.

```
struct standardControl {  
    byte packetType;  
    byte booleanFlags;  
    byte MOTLDuty;  
    byte MOTRDuty;  
    byte videoChannel;  
};  
typedef struct standardControl standardControl;
```

Structure 6.4: *standardControl* packet

Individual bits of the *booleanFlags* byte correspond to each of the power and motor direction controls just described, with the index of each bit globally defined to ensure the correct bits are being accessed. *bitWrite* operations are used to set each bit using the defined index, and the reverse performed on the receiving LittleBot using *bitRead* operations. As the duty cycle of LittleBot's motor drivers are controlled using the ATmega328's 8-bit PWM outputs, the bytes *MOTLDuty* and *MOTRDuty* satisfying this PWM resolution are included in the *standardControl* structure. Each byte corresponds to the duty cycle set for each motor or pair of motors in the case of the Standard LittleBot. Finally with the electronics designed to allow a video transmission on one of the eight selectable channels, a field specifying the video channel is required. Although only three bits corresponding to the three channel selection pins on the TX5823 are required, again the issue of a Boolean occupying an entire byte in the IDE results in a byte being added to specify the video channel in hexadecimal format. With the structure for a *standardControl* packet defined, the *sentryControl* packets are much simpler, as shown in Structure 6.5.

Similarly to the *standardControl*, the single byte *booleanFlags* is used for peripheral power control. Although a Sentry may be outfitted with up to four gas sensors, the control for their heater power is shared. The duty cy-

```
struct sentryControl {  
    byte packetType;  
    byte booleanFlags;  
};  
  
typedef struct sentryControl sentryControl;
```

Structure 6.5: *sentryResponse* packet

cle for the gas sensors may be specified in 20% increments by an operator, resulting in six options ranging from 0% to 100% being possible. As three bits allows for up $2^3 = 8$ gas duty cycles to be encoded, the three lowest bits of *booleanFlags* are reserved for the gas power control. With no cameras or motors on a Sentry, Booleans to control these are not required. The only other device with power control is the GPS, occupying a fourth bit in the *booleanFlags*. Again bit write operations are used on the field when the Controller is constructing a packet for transmission to a Sentry, and the corresponding bit read functions executed on the receiving LittleBot to ascertain the desired power status for the heaters and the GPS.

6.2 Firmware Configuration

As XBee modules are supplied with factory defaults programmed into them, they must be configured for use in LittleBots. While there are a plethora of configuration options, the bare minimum required for LittleBot purposes is to enable API mode 2 and set a fast UART baud rate. API mode 2 is required to allow the paired microcontroller to command the XBee over UART, while the UART baud rate should ideally be maximized to reduce the time taken for data transfer to occur between the devices. Although DigiMesh is enabled by default and the XBees can communicate with one another already, unique preamble, network, and cluster identifiers also need to be programmed into the modules to ensure rogue XBee modules that may be unknowingly present do not interfere with the Lit-

tleBot DigiMesh network. Each of the identifiers are essentially a level of checking as to whether an XBee is part of the same network, with preamble the lowest form and cluster the highest. To configure an XBee and set the parameters as desired, each is attached to a computer through a UART to USB adaptor that possesses a socket compatible with the footprint of the XBee, shown in Figure 6.1

Figure 6.1: XBee UART to USB adaptor

With the XBee placed in the adaptor and the host computer running the Windows™ operating system, the X-CTU utility from Digi International can be used to configure each XBee. Enabling API mode 2 is mandatory in order for the microcontrollers to be able to control the XBees in the desired manner. However, the selection of the baud rate is not as straightforward. As previously mentioned in Section 4.2, a crystal is used to clock the ATmega328. This clock directly affects the accuracy of the UART timing, and in turn the maximum baud rate that can be reliably realised. While the libraries for UART communication allow for its initialisation at up to 115200 bps, driving the ATmega2560 at 8 MHz results in an error of 8.5% in the serial timing at such a high baud rate [53]. As this is outside the 5% recommended maximum deviation specified by Digi International for reliable communication with an XBee, 57600 bps is instead selected. At this baud rate the timing error is only -3.5% with the 8 MHz clock on the Controller [53] and 2.1% with the 12 MHz used to clock the ATmega328 on each of the LittleBots [42], well within the recommended timing error tolerances.

Figure 6.2: X-CTU configuration utility

Shown in Figure 6.2, the X-CTU utility is used to read the configuration parameters from an XBee module. In this figure, the necessary changes from the default configuration have already been made. This is shown by the blue colour of the *AP* (API Mode) and *BD* (Baud Rate) parameters under *Serial Interfacing* parameters differing from the green colour representing the default of other parameters. Once all the desired changes have been made, *Write* is selected within the GUI causing the selected configuration to be uploaded to the connected XBee module and verified.

6.3 API Implementation

With the six packet types defined that facilitate the desired operation for the LittleBots in a DigiMesh based network, the API defined by Digi International to control each XBee module must be implemented to address and send packets to the desired nodes. Upon receipt of a packet, the API is also required to extract the sender address from a packet for identification of the sending LittleBot in the case of a Controller, and for a LittleBot to

store the Controller address from an *assocResponse* during association.

Instead of implementing the API from scratch and building a library to support the desired features provided by the DigiMesh protocol, a library for the Arduino IDE is sought and found, *xbee-arduino* [54]. As described by the developers, *xbee-arduino* is “an Arduino library for communicating with XBees in API mode, with support for both Series 1 (802.15.4) and Series 2 (ZB Pro/ZNet). This library includes support for the majority of packet types, including: TX/RX, AT Command, Remote AT, I/O Samples and Modem Status”. While the XBee-Pro 900 HP modules being used in LittleBot are Series 3 modules, comparison of the Series 2 and Series 3 API reveals that the API command structure for the transmission and receipt of packets is identical between the two, including extraction of a sender’s address from a received packet. As these are the only API features required for the LittleBot protocol, this library is utilised for control of the XBee by the microcontroller.

With the *xbee-arduino* library included and an *XBee* object declared, initialisation is achieved by calling *begin* on the *XBee* object with the desired 57600 UART baud rate. With the XBee library initialised, the serial buffer is inspected for packets when a *readPacket* is performed on the *XBee* to handle any API encoded data communicated to the microcontroller by the XBee over UART. Using commands provided by the library, received packets can then be extracted and cast to the correct type for processing by inspecting the first byte. Where required, the sending node’s address can be retrieved and stored using types included in the library, while the same types can be used to define the hardware address to transmit a packet to. The library can then be used to structure a complete transmission API request, passing it the address just defined, and the memory address and size of the completed LittleBot packet that is to be transmitted to the designated node. Further discussed in the following chapters, the API is implemented in this manner as part of the software written for each LittleBot and the Controller.

Chapter 7

LittleBot Software

For each of the LittleBots, software must be written to run on the AT-Mega328. Each board features an ICSP (in circuit system programming) header to which an ATMel AVRISP MKII programmer can be attached for programming with software written in the Arduino IDE. This chapter details the software written to run on each LittleBot.

7.1 Overview

Utilizing the *xbee-arduino* library to handle XBee API commands for communication, the ATMega328 is programmed to process and execute *standardControl* and *sentryControl* packets for command by the Controller. To provide sensor data back to the Controller, *standardResponse* and *sentryResponse* packets are constructed and transmitted. Similar to the packet design discussed in Chapter 6, the mobile LittleBots run the same software with *ifdef* statements inspecting the target variant during compilation to determine whether a DHT22 is fitted. A completely separate code base is written for the Sentry, as it is significantly different in sensor equipment to the mobile variants. However both code bases share the same initialisation and main loop structure illustrated in Figure 7.1, with variations reflecting the differences between mobile and Sentry LittleBots in terms of what can

be controlled and sensed on each.

Figure 7.1: Overview of the LittleBot software execution

To enable easy configuration of the software to reflect what sensors are fitted on the Sentry, a *UserConfig* header file is implemented that includes definitions that are intended to be configured by an operator. The main purpose of the header file is to allow an operator to specify what type of sensor is fitted to each of the four GPIO ports on a Sentry variant. However it is also used to define the variant of the LittleBot, in turn facilitating pin mappings between the ATmega328 and peripherals such as motor drivers and sensors automatically during compilation. Additionally, definitions for a *controlTimeout* and *TXPeriod* specified in milliseconds are included in this header.

The *controlTimeout* allows a user to specify the maximum length of time that may elapse between control packets before a mobile LittleBot powers down its motors and video system. This is necessary as in a situation where the Controller is piloting a LittleBot and that LittleBot stopped receiving control packets, it would otherwise continue moving at the current speed and direction uncontrolled. By default the timeout is set to 500 ms, however this may be user modified to a lower value to reduce the chance of collisions in the event that control is unexpectedly lost. The second definition, *TXPeriod*, specifies how often the LittleBot should transmit an *standardResponse* or *sentryResponse* packet to the Controller. When a LittleBot is not associated this also defines how often the LittleBot will broadcast an *assocRequest*. By default this is set to 1000 such that a LittleBot will report once per second, and again can be altered as desired. Caution must be taken here however, as the Controller software includes a correspond-

ing user selectable *RXTimeout* period. If the Controller does not receive a packet from the currently selected LittleBot within this period, the display is updated to inform the operator that communication has been lost. Until another response packet is received from that LittleBot, the Controller will stop sending the respective *standardControl* or *sentryControl* packets.

7.2 Initialization & Association

Once powered on, a LittleBot executes a *setup* routine to configure each of the ATmega328's pins in the appropriate mode and state for the corresponding peripheral that is attached. Although the pin mappings from the ATmega328 to each of the devices is not identical across the electronics for each of the LittleBot variants, conditional definitions based on the LittleBot variant that is defined in the *UserConfig* file are used to map the correct pin to each device during initialisation and to reduce the complexity of the code. This definition also enables the *unitType* field of an *assocRequest* to be filled correctly so that the Controller learns the correct variant of a LittleBot during the association process.

Where fitted, the DHT22 is initialised using its library, as is a gas sensor in the case of the mobile variants. On the Sentry however, the GPIO library discussed later is used to configure each of the sensors. An initialisation call is made for each of the GPIO ports, with the type of sensor attached to each port supplied from the *UserConfig* header as specified by the operator during configuration. Once complete, a *begin* call is made on the library that sets up each of the pins for the GPIO ports appropriately for the attached sensors. Finally the *SoftwareSerial* library is used to initialise the GPS at 4800 bps, and the internal variables that allow the software to determine the power status of the GPS (described later in Section 7.4) instantiated to their defaults.

With the *setup* routine complete and a LittleBot initialised, it is in a disassociated state by default. The main *loop* entered following *setup* com-

prises one routine that handles the receipt of packets, and another that constructs and transmits a packet when the *TXPeriod* lapses. Instead of constructing a separate routine dedicated to handling the association process called as part of *setup*, these packet receipt and transmission routines are constructed so that the LittleBot association is handled within these routines as part of the main loop. Further discussed in the subsequent sections, the routine that handles packet receipt includes the functionality to handle *assocResponse* and *assocInvalid* packets and update the LittleBot state and Controller address appropriately. Conversely the routine that transmits a packet inspects the *associated* flag indicating the association status, broadcasting an *assocRequest* if the LittleBot is not associated to a Controller.

7.3 Receiving Packets

With the LittleBot initialised, the main loop of the LittleBot software enters the packet receipt process illustrated in Figure 7.2 by checking whether any packets have been received by the XBee. Fundamentally by the design of the DigiMesh network, any packets that are received by and passed to the LittleBot's microcontroller for processing are either directly addressed to that XBee, or broadcast to the entire network in the case of an *assocRequest*. If a packet has been received, the first byte is inspected to determine its type. Only *assocResponse*, *assocInvalid*, and either *standardControl* or *sentryControl* packets are handled, depending on the variant. The only other type of packet that should be received by a LittleBot is the broadcasted *assocRequests* of other LittleBots. These, and any other erroneously received packets irrelevant to a LittleBot, are discarded.

If the first byte indicates the packet is an *assocResponse* (implying that an *assocRequest* previously broadcast by that LittleBot had been received by a Controller and the Controller has granted association), the address of the Controller is extracted from the higher layer XBee API packet, stored,

Figure 7.2: LittleBot packet reception process

and the LittleBot's *associated* flag asserted. Conversely if the byte indicates that the packet is an *assocInvalid* (that would have been sent by the Controller the LittleBot had been associated to and had unexpectedly restarted), the *associated* flag is de-asserted. As a result the LittleBot will revert to periodically broadcasting an *assocRequest* each time the *TXPeriod* lapses until it re-enters an associated state. In the case the first byte reveals that a *standardControl* or *sentryControl* packet is received, it is cast to the corresponding type so that the commands sent by the Controller can be

extracted from the packet and executed. For a Sentry, all that needs to be controlled using the packet is the 5 V power supplying the GPIO ports and the power of the GPS module. The power for the GPIO is set by calling the appropriate command on the GPIO library (discussed later) while for the GPS, an internal Boolean flag that the GPS function (executed as part of setting the outputs) inspects as part of its routine is updated.

The *standardControl* packets for the Standard and Junior LittleBots require more processing in comparison to the *sentryControl* packets, as there are a larger number of outputs that must be controlled. First the video system power must be configured by inspecting the relevant bit of the byte long *booleanFlags* field previously described, and the output to the pin driving the MOSFETs for the TX5823 and camera driven to the specified state. The same process is applied to set the power to the LED illumination for the video system, using the relevant bit of the *booleanFlags* field to drive the pin corresponding to the gate of the MOSFET controlling the LED driver power to the correct state. Using the *videoChannel* byte of the packet, the TX5823 transmission channel is configured by driving the gates of the MOSFETs corresponding to the TX5823 control lines to match the channel selected by the Controller. Finally the power control of the gas sensor is similar to the Sentry, but is instead made using a call to a function of the gas library instead of the GPIO library. Again the relevant bit of the *booleanFlags* field defined for control of the gas power is inspected and supplied as the argument to the function call on the gas library.

The last remaining outputs dictated by a *standardControl* packet are those for the motor drivers. The packet includes two bytes and two bits in *booleanFlags* corresponding to the duty cycle and direction for each motor or pair of motors on one side of a LittleBot, with a third bit in the *booleanFlags* controlling power to the motor drivers as selected by a button on the Controller. On each DRV8835 there are a pair of inputs that correspond to each of the two motor outputs on the driver, *IN1* and *IN2*. During the electronics design the inputs are configured to be in *IN/IN* mode, and a motor

can be driven forward at $x\%$ by driving *IN1* to logic high, and *IN2* with a PWM signal of $x\%$ duty cycle. To drive the motor in reverse at $x\%$, *IN2* must instead be driven to logic high and *IN1* supplied a PWM signal of $x\%$ duty cycle. Finally when a duty cycle of 0% is specified for a motor, the DRV8835 can be set to allow coasting by applying a logic low to both the *IN1* and *IN2* inputs. To implement the desired DRV8835 control, a routine is written that takes the three Booleans and two bytes representing motor power, directions, and duty cycles. This routine controls the outputs of the ATmega328 corresponding to each of the *IN1*, *IN2* and logic power lines of the DRV8835's to drive the motors in the correct direction with the desired duty cycle, configuring the PWM registers as appropriate.

Referring back to Figure 7.2, if a packet had been received then the *lastRXTime* is updated. Before continuing to check whether the LittleBot should transmit a packet, this *lastRXTime* is used to determine if the *controlTimeout* period has been exceeded. If a packet has just been received then clearly this will not be true. However if the period has been exceeded, inferring communications with the controller have been lost, a mobile LittleBot will power down its motors and video feed for the aforementioned reasons.

7.4 Transmitting Packets

With its process outlined in Figure 7.3, the second function of the main loop handling packet transmission begins by inspecting the time elapsed since the last packet was transmitted. If this period exceeds the *TXPeriod* as defined in *UserConfig*, a packet is constructed and transmitted to the Controller. If the LittleBot is not yet associated, an *assocRequest* is constructed with the LittleBot's type and broadcasted. If the LittleBot is a Sentry, a function from the GPIO library is used to populate the *sensorConfiguration* field of the *assocRequest* before transmitting. This allows the receiving Controller to determine what sensors the Sentry is equipped with, so that

it can interpret the data in subsequent *sentryResponse* packets correctly.

Figure 7.3: LittleBot packet transmission process

Once a LittleBot is associated and the *associated* flag asserted, it will transmit a *standardResponse* or *sentryResponse* packet addressed to the associated Controller each time the *TXPeriod* lapses. In order to populate a transmission packet with sensor data, as well as control power to peripherals such as the heater on a gas sensor, supporting code and libraries to facilitate the desired operation of the sensors are written, adapted or employed as appropriate. While some of the libraries and code written are hundreds of lines in length, the following gives an overview of what mandatory functionality is required and how it is achieved for each.

7.4.1 Battery Level

As previously described, the raw power supply is scaled down using a voltage divider for measurement by the 10-bit ADC present on the AT-Mega328 to determine an approximate battery level. As calculated in Section 4.6, the output voltage from the voltage divider will be 2.88 V when the battery is full, and 2.06 V when empty. As the reference voltage for the ADC is the 3.3 V VCC, the result of measurements by the ADC will be $\frac{2.88}{3.3} \times 2^{10} = 894$ when the battery is full, and $\frac{2.06}{3.3} \times 2^{10} = 639$ when the battery is empty.

While the battery level could be calculated as a percentage of full capacity on the LittleBot before being transmitted to the Controller, the byte provided by the packet design allows for a greater resolution. Although the configuration of the OLED display results in the battery level being truncated to 1% steps, data reported by the LittleBots is stored in its raw format on the SD card, with the greater resolution retained. In the case of the LittleBot Sentry where extended runtime is expected, this greater resolution may prove useful in analysis of the data stored on the SD card at a later date.

A byte sized field can store an unsigned integer value from 0 up to $2^8 - 1 = 255$. With an expected measurement range of $894 - 639 = 255$ across the range of battery level voltages, the measured battery level can be assigned to the field simply by off-setting the measurement by the lowest possible value that is expected to be measured, 639. Hence a 0 will be assigned when the battery is considered empty, and 255 when the battery considered full. The Controller can then convert this value to a percentage of full battery, as well as store the raw reading to the SD card. To prevent erroneous battery levels being reported in the case of the battery being over charged or over depleted, checks for measurements outside the expected range are applied and the measured value bounded to be between 639 and 894.

7.4.2 Temperature & Humidity

The DHT22 sensor employs a proprietary low baud serial-like protocol to communicate six bytes of data including a checksum that when decoded, provides a temperature and humidity reading. To initiate a reading, the ATmega328 must pull the data line of the DHT22 low for 20 ms, before pulling it high using its internal resistor. The DHT22 recognises this sequence of events and subsequently pulls the line low or lets the ATmega328 pull it high in a manner representing digital highs and lows. With the timing between bits given by the DHT22 datasheet, the software running on the ATmega328 can read the six bytes of data from the DHT22. Similarly to the XBee, open source libraries already exist for the DHT22 that handle its operation, however all of these libraries have limitations considered less than optimal in terms of the DHT22's application in LittleBot.

Although the raw data from the DHT22 that is used to provide the temperature and humidity readings is only four bytes long, the library returns the decoded data in the form of two floats occupying eight bytes. As previously discussed, the *standardResponse* and *sentryResponse* packet structures are written to accommodate the raw data instead of the decoded data to minimise packet size. In turn the Controller software performs the necessary arithmetic on the raw data to obtain the temperature and humidity readings. As a result the DHT22 library used is modified to return the four required bytes of data from the six provided by the DHT22. The second issue with the libraries is due to a limitation of the DHT22 itself, where it can only provide an update at most every two seconds. If a read is called on the library within two seconds of the previous call, the library will block and wait until two seconds have elapsed. As the read call will be made as part of the main loop to populate the packet that is transmitted to the Controller, and a packet is transmitted every second by default, the LittleBot will freeze for a second every two seconds in the main loop. As this main loop is also handling packet receipt and hence piloting of a LittleBot, such

an approach is not acceptable as it would result in loss of control during the blocking periods. As a result the library is modified to store the four bytes internally each time a reading is requested, with this stale data returned if a subsequent read is made before two seconds have elapsed since the last.

7.4.3 Gas Level

The gas sensors employed on LittleBot are set up in a voltage divider configuration, such that the change in voltage of the divider caused by a change in resistance across its sensing element can be measured by the ADC of the ATmega328. For the gas sensors to operate and an accurate measurement made, the heater element of the gas sensor must be on for at least 30 seconds prior to the measurement being taken [41]. As the power to the heater element can be switched on and off by an operator from the Controller, a way of determining if the 30 seconds has elapsed is necessary. Additionally if the heater is on but the 30 seconds have yet to elapse, a method of communicating that the heater is still warming up must be represented in a response packet.

To facilitate the initialisation of the gas sensor and manage timing of the heater power, the *Gas* library is written. This library allows a gas sensor to be instantiated, where the pins to read the sensor and control the heater are supplied as arguments during declaration. Three functions are included in the library, with *begin* called as part of the LittleBot *setup* routine that correctly initialises the pins relevant to the gas sensor. The second function, *powerOn*, must be supplied with a Boolean that determines whether the heater should be on. If the heater is desired to be on and was already on, no action is taken. However if the heater was not already on, the line controlling the MOSFET to turn the heater on is driven high, and the time at which this is done is recorded. Finally if the heater is desired to be off as specified by a *false* supplied to the function, the line to the gate

of the MOSFET is driven low and the heater switched off.

With the library keeping track of when the heater was switched on, the third function *read* first determines whether 30 seconds have elapsed since the heater has been switched on. If true, the ADC is used to take a reading from the gas sensor, returning a value that is between 0 and 1023 where a higher value indicates a higher gas concentration. If the 30 seconds have not elapsed, then the value returned should indicate that the gas sensor is still heating as the value returned by the function directly populates the field for the gas sensor in the response packet. Furthermore, a value representing that the gas sensor is off is also required for the same reason. As the integer field in the packet for the returned value allows for negative values, and the ADC always returns a positive value, negative values can be selected and used as flags to represent the heating and off states of the gas sensor. Consequently -1 and -2 are selected to represent the gas sensor in the off and heating states respectively.

7.4.4 GPIO

In addition to gas sensors, the Sentry can also be outfitted with PIR sensors. For simple configuration of a Sentry variant by an operator, the type of sensor fitted to each port is specified in the *UserConfig* header. To simplify programming and population of the *sensorReadings* field of the *sentryResponse* packet, a GPIO library is written to handle the sensors of a Sentry LittleBot. During the initialisation routine of a LittleBot Sentry, each port is set up by the GPIO library for the sensor attached as defined in the *UserConfig* header. When the LittleBot attempts to associate with a Controller, its sensor configuration is returned by this GPIO library as a bit mask to populate the *sensorConfiguration* byte of the *assocRequest* accordingly.

With the GPIO initialised, the sensor of each port can be read in an identical manner to populate the *sensorReadings* array, irrespective of the

type of sensor fitted. For a PIR sensor, one of four statuses are returned by the read function, where each status is defined by a unique integer to match the type of the *sensorReadings* array. If the LittleBot has been powered on for less than 30 seconds, the sensor is still in its calibration stage and a *PIRCALIBRATING* status returned. Once the Sentry has been powered for more than 30 seconds, the PIR sensor is active and will drive its output high for two seconds whenever motion is detected. When a logic high is read by the ATmega328, a *PIRTRIPPED* status is returned, while if a logic low is read and the last trip was less than 30 seconds ago, a *PIRRECENTLYTRIPPED* is returned. Finally if the line is low and the last trip detected was over 30 seconds ago, a *PIRIDLE* is returned.

The gas functionality of the GPIO library is similar to the standalone Gas library implemented for the mobile LittleBots, with the exception of one GPIO library function controlling the power to all the gas sensors fitted with control over the duty cycle. The *powerOn* function instead controls the inhibit pin of the PTH08080WAH, supplying 5 V to all or none of the GPIO ports. As part of the *powerOn* function, a duty cycle can be specified. Using this the GPIO library controls the power to the heater in a five-minute period to attain the selected duty cycle. When a read is performed on a gas sensor, the same negative integers representing the gas sensor being off or still heating will be returned in the relevant case. Otherwise a measurement is taken by the ADC and the value returned to populate that sensor's index in the *sensorReadings* array.

7.4.5 GPS

The final peripheral that must be handled is the GPS on the Sentry. As previously mentioned, the GPS has a soft power control that can be toggled by pulsing one of its digital lines, while the data from the GPS is provided to the microcontroller serially in the NMEA (National Marine Electronics Association) specified format. As this serial data must be handled using

bit banging, the *SoftwareSerial* library included as part of the Arduino IDE is employed, initialised with the corresponding pins and 4800 bps baud rate for the GPS during the *setup* routine.

With the *SoftwareSerial* library handling the bit banged serial data, the NMEA data received must be decoded. Another library, *tinyGPS*, is employed for this purpose, and is passed any serial data available from the *SoftwareSerial* library for *encoding* on each iteration of the main loop. Once a sufficient number of NMEA messages have been encoded by the *tinyGPS* library, the library will return the latitude and longitude determined by the GPS as floats. These are then placed in the relevant fields of the *sentryResponse* packet for transmission back to the Controller.

As the GPS does not have an output indicating its power status and the power status is toggled by pulsing its *shutdown* input pin, a method of deducing whether the GPS is on is required in order to determine whether the GPS's current power status matches what the operator has selected. In its hibernate state, the GPS will not transmit any serial data. However when active (regardless of reception), the GPS will transmit at least one NMEA encoded message every 200 ms. During the routine that handles the GPS, if serial data is available from the GPS module it can be deduced the GPS is not in hibernation. Conversely if no serial data is received within a 200 ms period, then the GPS must be in hibernation. Using this principle and the desired GPS power status specified by *sentryControl* packets received, the GPS routine toggles the *shutdown* pin to control the hibernation mode as required. With this the final library written for the LittleBot, software is written for the Controller in a similar manner as detailed in the next chapter.

Chapter 8

Controller Software

Software must be written and programmed into the Controller's AT-Mega2560 to perform the desired functionality, with XBee communications handled as discussed in Chapter 6 for association, control and feedback of LittleBots. Additionally the software must handle the button and thumbstick inputs for selection and control of a LittleBot, and button LED and OLED display outputs to visually represent its status. This chapter details the software written for the Controller.

8.1 Overview

While the three LittleBot variants share the same code structure differing only where necessary for the different sensors equipped to each, the Controller software is substantially different in functionality. As illustrated in Figure 8.1, the main loop executes four functions following initialisation.

Figure 8.1: Overview of the Controller software

handleButtons, as the name implies, takes care of buttons pressed or held by an operator and updates the state of the Controller appropriately. *handleIncoming* handles any packets received by the XBee in the Controller, generating responses where required and storing all received data to an SD card (if present). *updateDisplay* generates and updates the OLED display buffer as necessary, while *controlUnit* sends a control packet to the currently selected LittleBot, piloting it using input from the thumbstick if it is a mobile LittleBot.

Similarly to the LittleBot software, a *UserConfig* file is included to allow an operator to customise some of the software's parameters. Of particular importance is the *RXTimeout* definition which, as previously mentioned, specifies how long the Controller allows between packets received from the currently selected LittleBot before communication is considered lost. As a result *RXTimeout* must be greater than the *TXPeriod* programmed into the LittleBots, and by default is five times the length of the default *TXPeriod*.

8.2 Initialisation

As with the LittleBots, a *setup* routine is called during power up of the Controller to appropriately configure pins and interrupts, and initialise peripherals such as the SD card, OLED display and XBee. Similarly to the LittleBot software, the first step is to initialise the *xbec-arduino* for control of the XBee using the API that allows packets to be sent and received in the LittleBot DigiMesh network. While there is only a lone UART port on each LittleBot's ATmega328, the Controller's ATmega2560 features four UARTs, with the XBee attached to the third. As the library by default selects UART0, additional code in the software for the Controller makes a call on the *XBee* object to set the appropriate UART prior to calling *begin* specified with the 57600 bps baud rate.

The next stage of *setup* handles the RTC initialisation to enable time

and date stamps to be included during data logging to the SD card. A library specific to the DS3231 is employed to handle initialisation of both the RTC as well as the I2C bus it is attached to. A function of this library is specified as the *SyncProvider* to the Arduino IDE's built in *Time* library, and an initial synchronisation performed to obtain the date and time from the RTC. Following this initial synchronisation, the ATmega2560's 8 MHz clock is used to keep time but with periodic synchronization to the RTC to ensure accuracy. Once initialised, time stamps can be included during the data logging in the standard Unix time format using a globally available function call to the *Time* library.

In order to store data to an SD card, its file system must be mounted and a file for writing created. Before attempting to initialise an SD card, the Controller must deduce whether an SD card is inserted without write protection. As previously discussed in Section 5.2.2, the card detect and write protect lines of the socket can be used to determine the SD card status. To allow the ATmega2560 to determine if a line is grounded (which would correspond to a card being inserted or the write protection enabled), both are pulled high by enabling the ATmega2560's internal pull up resistors. When either of these conditions are detected, the initialisation routine will return an error to set the SD card status flag, later used during *updateDisplay* to indicate the SD status to an operator. In a situation where a card is detected without write protection, the *SD* libraries included with the Arduino IDE are used to attempt a mount on the file system and to create a text file with the current time and date. With a file opened successfully and the SD status flag set accordingly, the SD initialisation is complete and data can be written to the card using functions from the *SD* library.

With the switches inside the six buttons attached to external interrupt capable pins, the internal pull-ups of these pins must be enabled so that falling edges may be detected when the button is pressed resulting in the line being grounded. The pull-ups are enabled in software during the initialisation, with individual interrupt service routines attached to each but-

ton to allow each to be handled as discussed in the next section. As each button also possesses an LED ring that is used to visually display the status of a feature for some of the buttons, the pins assigned to driving the LEDs are placed in output mode so that it may be illuminated as required.

With the major peripherals configured, the *setup* routine concludes by appropriately configuring the miscellaneous remaining inputs and outputs. The analogue lines for the battery level monitoring and thumbstick are configured as high impedance inputs, while pins attached to the MOSFETs that control the power and channel of the video receiver are configured as outputs. Finally the OLED display is initialised using a single routine included in the Adafruit Libraries discussed later in Section 8.7, with the software SPI pin mappings from the ATmega2560 to the display supplied during declaration.

8.3 Input Handling

Once the Controller is initialised, the main routine is entered. The first function called is *handleButtons* and as just discussed, each of the buttons has an interrupt service routine attached to it. Instead of directly modifying program variables, the interrupt service routine for each button simply asserts a flag indicating that button has been pressed, and stores the time it was called in milliseconds. As switch bounce will be at higher frequencies than the execution of the main loop, this approach prevents switch bounce from having an effect on program execution while also avoiding the race conditions caused by modifying variables during an interrupt. When *handleButtons* is called, the flag for each button is inspected and any necessary action taken as shown in Figure 8.2. If the *Next* or *Prev* buttons which allow a LittleBot to be selected have been pressed, the variable representing the index of the currently selected LittleBot will be incremented or decremented accordingly. Bound checking with wrap around is implemented to prevent invalid indexes being selected, while another flag

unitChanged becomes asserted and the index of the previously selected LittleBot stored for use later in the *controlUnit* routine.

Figure 8.2: Overview of *handleButtons*

The remaining four buttons are used to control features on a selected LittleBot, with *Lights* used to control the light power of a mobile LittleBot. Should the interrupt flag be asserted indicating that the button has been pressed, the status of the lights is toggled and the LED ring for the button set to reflect the new status. If the lights are enabled, the LED is illuminated while conversely if they are off, the LED is not. The remaining three

buttons, *Motors/GPS*, *Video* and *Gas*, can be pressed in the same way as *Lights* to toggle the power status of the corresponding peripheral. However these buttons can also be held for additional functionality. Holding *Motors/GPS* button will toggle a *Turbo* status. Described further in Section 8.6, having turbo disabled essentially halves the speed of the motors to allow for finer control of a LittleBot. Although this button also doubles as the power control for the GPS in the case a Sentry is selected, the extra functionality that enables the button to be held does not interfere with the GPS power toggling function. For the *Video* button, a press will toggle the power status, while a hold will advance the currently selected video channel. Finally for *Gas*, a press will toggle the power status between an on and off, while a hold will increment the duty cycle for a Sentry in 20% increments, wrapping around when 100% is reached.

To achieve the hold functionality, *handleButtons* inspects if a button with its interrupt flag asserted is still held. If not, then the status is simply toggled and the relevant LED ring for the button updated accordingly. However if the button is still held, the time at which the button was pressed is inspected. Once a button has been held for half a second, the *Turbo* status is toggled in the case of the *Motor/GPS* button, the video transmission channel incremented in the case of *Video*, and the duty cycle incremented in the case of *Gas*. Until the button is released, the corresponding interrupt flag will not be cleared and the *handleButton* will continue to check how long the button has been held during each execution. As a result the functionality will continue and repeat so that the *Video* and *Gas* buttons can be held continuously to advance multiple channels or duty cycles without requiring subsequent press and holds. Again bound checking and wrap around is applied to ensure an invalid video channel or duty cycle cannot be selected, and the *displayStale* flag asserted such that the new motor turbo status, video channel, or duty cycle is displayed. Finally if the status of video system changes in any way, a function that sets the power status and channel for the video display is called to configure the

MOSFETs controlling the RX5808 and video display accordingly.

8.4 Packet Receipt

As previously described, the XBee is configured in API mode so that any broadcasted packets or packets addressed specifically to that XBee are passed to the handling microcontroller. As the ATmega2560 is by nature going to process more packets than the LittleBots and possesses four times more SRAM, the UART buffer is increased from 128 bytes to 1024 bytes to ensure packets are not lost due to delays in their processing. On the Controller, the *handleIncoming* routine illustrated in Figure 8.3 is executed as part of the main function to process received data.

Before checking for packets, the first part of *handleIncoming* checks the SD status, further discussed in the next section. The UART buffer is then inspected using the *xbec-arduino* library and any packets received are retrieved for storage on the SD card before being processed. As with the software on the LittleBot, the first byte of the packet is inspected to determine its type. Whenever an *assocRequest* is received, its XBee address, type, and in the case of the Sentry, *sensorConfiguration*, must be stored in a linked manner on the Controller, followed by an *assocResponse* being transmitted to that LittleBot. Arrays are used to keep track of each LittleBot that is associated with the Controller, with a LittleBot added when their *assocRequest* is received. To ensure that duplicates of the same LittleBot are not added in the case they have erroneously restarted, the array is checked when an *assocRequest* is received to see if it already contains an entry with the address of the requesting LittleBot. If so, the existing entry is overwritten instead of a new entry being added. In either case however, an *assocResponse* is sent to acknowledge that the LittleBot is correctly associated to the Controller. If the number of associated LittleBots has increased, the *displayStale* flag is asserted so that the correct count of LittleBots is shown on the OLED display when *updateDisplay* is executed next.

Figure 8.3: Overview of *handleIncoming*

When a *standardResponse* or *sentryResponse* is received, the table of associated LittleBots is first inspected using the sending LittleBot's address to ensure it is correctly associated with the Controller. If the address is not present in the table, an *assocInvalid* is constructed and sent to that LittleBot, causing it to re-associate. If the packet received is from the currently selected LittleBot (and hence whose information is displayed on the OLED display), the data is extracted from the packet, the *displayStale* flag asserted, and the *noDisplayData* flag cleared for use later in the *updateDisplay* routine. The *commsLost* flag is also cleared here, as it is evident com-

munications are not lost with the currently selected LittleBot as a packet has just been received from it.

Each time a packet is received from the selected LittleBot, the current time is recorded so that the amount of time since the last packet was received from the currently selected LittleBot may be determined. If this period exceeds *RXTimeout*, the *commsLost* and *displayStale* flags are asserted. When the *updateDisplay* method is later called, this combination of flags results in a message on the OLED display informing the operator that communications have been lost with the selected LittleBot. The *controlUnit* routine also utilises the *commsLost* flag, and will not attempt to send a control packet if communication is lost. Upon a subsequent packet being successfully received from the selected LittleBot, the *commsLost* flag is cleared and *displayStale* asserted, resulting in the OLED display being updated with the latest sensor information and allowing control of that LittleBot to continue as normal.

8.5 Data Logging

As previously discussed, all data reported by LittleBots present in the DigiMesh network is stored to an SD card with a time and date stamp, and the XBee address of the sending LittleBot. With handling of the SD card initialisation completed as previously discussed, an *if* statement is added to the just described *handleIncoming* routine that handles incoming packets from the LittleBot network. Each time a packet is received, this statement checks the SD status flag and if a file has been successfully opened, the raw data of the packet is written in hexadecimal format to the SD card. In addition to the data, the time the packet was received and the hardware address of the sending LittleBot are also stored. As the first byte of every packet specifies the type of packet, the stored data can ultimately be decoded for analysis. As the sensor configuration of a Sentry is provided only in the *assocRequest*, all packets are stored from start up in-

cluding association requests. This enables the GPIO sensor configuration to be determined and subsequent *sentryResponse* packets stored on the SD card to be decoded at a later date.

For completeness, and to allow the hot swapping of cards, the SD card buffer is flushed every time a packet is received, otherwise the entire 512 byte buffer must be filled with received data before it will be written to the card. In turn, data may be lost if the card were to be removed or the power to the Controller lost between the automatic flushing of the buffer that occurs only when it is full. Additionally, ongoing checking of the SD card status is undertaken, enabling a user to remove a card and insert a new one without requiring a restart of the Controller. Should the SD status subsequently change after the initial *setup* routine of the Controller, the same initialisation routine for the SD card is called and the *displayStale* flag asserted enabling the change in SD status to be reflected on the OLED display.

8.6 LittleBot Control

With *handleButtons* and *handleReceipt* handling button inputs and packet receipt implemented, the *controlUnit* routine of the main function handles control of a currently selected LittleBot. Illustrated in 8.4, the routine first checks whether the selected LittleBot has been changed by inspecting the *unitChanged* flag. If true and the previously selected LittleBot is a mobile variant, a *standardControl* packet is sent to that LittleBot to power down the video, lights and motors as it is no longer being piloted. This flag is then cleared, and in addition to the *displayStale* flag, the previously mentioned *noDisplayData* flag asserted in recognition of a different LittleBot being selected with a packet yet to be received. In this state the status of that LittleBot cannot yet be displayed, prompting the OLED display to present a “Waiting for Response” message until a packet is received from the newly selected LittleBot.

Figure 8.4: Overview of *controlUnit*

Before constructing and transmitting a packet to control the currently selected LittleBot, the `commsLost` flag previously discussed is inspected to ensure that the LittleBot is still responding. If communications have been lost, the routine does not attempt to send a packet, preventing the DigiMesh network being saturated with undeliverable packets. Although these would time out within a few seconds, undelivered packets can cause the network to slow down as the Controller's XBee would attempt to re-transmit the packets up to three times. If the currently selected LittleBot is a Sentry variant, a `sentryControl` packet is sent only if the heater or GPS power settings have changed since the last control packet was sent. Otherwise sending a `sentryControl` packet would be redundant, since the transmission of a new control packet would not change the state of the LittleBot. If the configuration selected on the Controller has changed since the last

packet transmitted, a *sentryControl* packet is populated with the current GPS and Gas configuration, and transmitted to the selected Sentry LittleBot.

For the Standard and Junior variants, *standardControl* packets ultimately manipulate the motors based on the thumb stick input from the user, making a high packet transmission rate desirable for responsive control. The linear design approach to the main *loop* of the Controller software inherently limits the rate *controlUnit* may be called at due to the time taken to execute each of the four main functions. However a 100 ms minimum period is imposed between *standardControl* packets being sent to prevent saturation of the XBee network.

To populate the packet, the bits of the *booleanFlags* field in the packet corresponding to each of the power buttons are asserted appropriately, reflecting the selected status of each. The *videoChannel* field is populated using the video channel selected by the operator, while the motor duty cycles and directions are calculated using an algorithm that maps the X & Y output from a joystick to the two outputs corresponding to motors on either side of a vehicle. Before the duty cycle fields of the *standardControl* packet are populated, the *Turbo* status previously mentioned is inspected. If asserted, the calculated duty cycles are halved through bit shifting to allow slower and finer piloting of a LittleBot. Finally the remaining bits of the *booleanFlags* field are asserted depending on the motor directions calculated and the packet transmitted to the selected LittleBot.

8.7 Information Display

The SSD1306 controller driving the OLED display requires its entire buffer to be updated in one pass by the microcontroller. This means that the AT-Mega2560 must generate the entire desired display buffer within SRAM, and write it to the SSD1306 using the previously selected bit-banged SPI interface. During the execution of the main loop of the Controller soft-

ware, each of the previously discussed routines called will assert the *displayStale* flag to indicate that there has been a change in the status of the Controller and the display should be updated to reflect this. This occurs not only when a packet is received from the currently selected LittleBot (and hence new data from its sensors is available to be displayed or more LittleBots are found in the network as previously discussed), but also when the status of the SD card changes.

Figure 8.5: Overview of *updateDisplay*

When the *updateDisplay* illustrated in Figure 8.5 is called as part of the main loop, the *displayStale* flag is first inspected. If asserted or the current time differs from that displayed, the display buffer is rebuilt with the

current status and variables of the Controller. Each bit within the buffer corresponds to a pixel on the display, with an asserted bit in the buffer representing a white pixel. Manipulation of each of the individual bits within the buffer to achieve the desired text layout on the display is an impractical approach to generating the display buffer, and instead existing open source libraries from Adafruit are employed [55]. Standard C print functions can be called to manipulate the buffer, placing text as desired with control over position and size. Small macros are written to support the libraries, allowing text to be centred and right aligned where appropriate within the buffer. Once the buffer has been calculated, a routine with the library is called that sends this to the SSD1306 through bit banded SPI, resulting in the OLED display being updated as desired. Along the top of the display, the current time, SD card status, and Controller battery level remaining are displayed, shown in Figure 8.6 in a situation where no LittleBots found by the Controller.

Figure 8.6: Inverted title on the OLED display

To make the title bar stand out from the LittleBot data, it is inverted so that the background is white and the text black. Originally to achieve this desired effect, the libraries were used to draw a white rectangle in the buffer of appropriate height before black text was added using print calls. Measurement of the routine that added the title revealed this process took 122 ms. Further investigation revealed that the routine to draw the white rectangle was taking 100 ms of this time. As the display may be updated while the Controller is piloting a mobile LittleBot, the time taken for each subroutine should be minimised to prevent the rate of commands

sent by the Controller being reduced to a point where control may seem unresponsive to an operator.

During each call of the display update method, the display buffer is cleared using the *memset* function, an extremely fast call that takes less than 1 ms to perform. Instead of drawing a white rectangle to get the bar across the top of the display, the routine that clears the buffer is modified to add the white bar. An additional *memset* is used to assert bits in the display buffer corresponding to the first six rows for the header, while the starting address and length arguments to the original *memset* are modified to clear the remainder of the buffer. This achieves the white title bar effect as part of the buffer clear routine in less than 1 ms as opposed to the original 100 ms. With the routine to build the common title bar complete, the rest of the *updateDisplay* function is executed conditionally, and print calls used to manipulate the buffer to display the currently selected LittleBot's data in an appropriate format.

Figure 8.7: OLED display formats for various statuses

Illustrated in Figure 8.7, the format of the display is dependant on the variant of the currently selected LittleBot and communication status. If no LittleBots are found, the operator is presented with the message shown in

(a). If the previously discussed *noDisplayData* flag is asserted, the “Waiting for Response” message shown in (e) is displayed to inform the user a packet has yet to be received since selecting the LittleBot. The other discussed flag, *commsLost*, results in the “Lost Communication” message shown in (f). When neither of these flags are asserted and a mobile LittleBot is selected, the latest sensor information is displayed along with the selected video channel and *Turbo* status as shown in (b) and (c). Finally if a Sentry LittleBot is selected, the display configuration shown in (d) is used.

The Sentry sensor configuration obtained by the Controller during association is decoded through binary manipulation to determine what sensors are fitted, so that the data encoded in *sensorReadings* is displayed correctly with the appropriate label. Where a DHT22 sensor is fitted (in the case of the Standard and Sentry variants), the four raw bytes transmitted are manipulated to decode the temperature and humidity for display on the OLED. For a gas sensor reading, the value returned is inspected to check for negative values reflecting a heating or off status and displayed accordingly, else the ADC value is displayed. Finally for each PIR sensor (where fitted on the Sentry), the value returned is used to determine the status of the sensor, and also displayed accordingly.

With the *updateDisplay* routine complete, additional improvements are made to the Adafruit libraries to increase the speed of the buffer manipulation. The unmodified libraries took 400 ms to draw the complete the display buffer, which in turn would slow down the execution speed of the main routine. By replacing inefficient division operations with bit shifts and removing unnecessary *if* statements within the library, a lower time of 120 ms to execute the *updateDisplay* is achieved.

Chapter 9

Evaluation and Analysis

With the LittleBots and Controller designed and constructed, they are evaluated against the original objectives presented in Section 1.2. The tether-less operation, video feedback, gas sensor equipment and compact size are achieved in principle through the design of LittleBots. However, the range of the tether-less operation must be evaluated, along with the overall cost, ingress protection, 2 hour runtime, and intuitive use objectives. These are presented and discussed in this chapter.

9.1 Battery Performance

With a 2 hour runtime stipulated as an objective, the run times for each LittleBot must be evaluated under the various possible power configurations.

9.1.1 Runtime

The power sources for the mobile LittleBots are designed to provide a minimum of 2 hours runtime, while the design for a Sentry equipped with four gas sensors running at a 20% duty cycle permits a runtime of at least 24 hours. For each variant, the maximum runtime under different periph-

eral power configurations and loadings is evaluated to test whether the minimum runtimes have been met. The run times for each type of power configuration are measured three times, and the means evaluated. For all runtime test configurations, the standard deviation of these measurements never exceeds 0.1 hours.

Mobile LittleBots

As previously discussed, the power consuming peripherals of a mobile LittleBot are the gas module, lights, video feedback and motors. The power to each of these can be controlled through selection on the Controller, with different combinations resulting in different runtimes. Testing is divided into driven and stationary, with Figure 9.1 presenting the stationary runtimes for the various power configurations.

Figure 9.1: Mobile LittleBot operating times while stationary

Indicated by the legend on the right, a stationary LittleBot with a power configuration of camera, lights and gas enabled achieves minimum mean runtimes of 3.9 hours for a Junior variant, and 3.8 hours for a Stan-

dard. Testing across the various power configurations, a runtime of up to 57.1 hours can be achieved with no peripherals enabled and the LittleBot only acting as a DigiMesh repeater. For each of the configurations, the Standard variant's mean runtime is always 0.1 hours lower than that of the Junior even though the electronics between the two should be identical in terms of power consumption. Although within the noise margin resulting from the 0.1 hour maximum standard deviation, this insignificant but consistent difference is likely due to a small variance in the capacities of the lithium polymer cells.

For driving tests where the motors are enabled, the amount of loading on the motors impacts the amount of current they draw. As driving a LittleBot continuously under constant conditions for numerous hours is invariably difficult to achieve, runtime measurements involving motors are completed with the motors unloaded. In addition to the time taken, the average current drawn is measured for each unloaded test configuration. Each test configuration is then repeated with the LittleBot driving on a level surface for 30 seconds, and the average current recorded. The difference in average current between unloaded and loaded test is finally used to adjust the unloaded runtimes, resulting in the runtimes presented below in Figure 9.2.

As piloting without the video enabled is considered normally impractical, driving test configurations without video feedback enabled are omitted. A Junior LittleBot in its most power hungry state achieves a runtime of 3.2 hours, while a Standard LittleBot achieves a lower runtime of 2.7 hours. In both cases the 2 hour objective is exceeded. The results also demonstrate that an operator may remotely pilot a LittleBot with just the video feed enabled continuously for up to 5.6 hours or 6 hours on a Junior or Standard LittleBot respectively. This outperforms the 1 hour runtime of the \$4,875 Throwbot LE [20] by up to six fold, and the 2 hour runtime of the \$13,000 Throwbot XT [21] by up to three fold. This is a considerable achievement for a relatively inexpensive device.

Figure 9.2: Mobile LittleBot operating times while driving

Sentry LittleBot

A Sentry LittleBot may be configured with a GPS and up to four gas or PIR sensors, with four gas sensors at 100% duty cycle the most power hungry and with no sensors operating the least. While the runtime objective is for four gas sensors at a 20% duty cycle, the runtime of the Sentry is evaluated across the various possible configurations.

As illustrated in Figure 9.3, the Sentry exceeds the 24 hour runtime objective for four gas sensors at a 20% duty cycle by over 29%, achieving 62.1 hours. A shorter runtime of 14.2 hours is achieved with the four gas sensors running continuously, as expected, while a runtime of 152.4 hours can be realised with no sensors at all. With four constantly powered PIR sensors fitted, the maximum runtime is only reduced by 0.1 hours. In addition to being within the 0.1 hour noise margin, at most a small reduction is expected since the PIR sensors are low power. Finally enabling the GPS with a line-of-sight to the sky and four PIR sensor fitted reduces the runtime by 4.9 hours to 147.4 hours.

Figure 9.3: Sentry LittleBot operating times

Controller

In testing the runtime of the Controller, three distinct states that impact the runtime exist. A mobile LittleBot selected with the video enabled, a mobile LittleBot selected with the video disabled, and a Sentry LittleBot selected. The video state and hence the video display being powered influences the runtime, while the type of unit selected dictates how often a packet is being transmitted for control.

Demonstrated in Figure 9.4, the Controller piloting a mobile LittleBot with the video on lasts 5.4 hours. With the video disabled but a mobile LittleBot still selected, the Controller lasts for 54.4 hours receiving data. Finally with the Controller in the same set up but with a Sentry selected, a runtime of 58.2 hours is achieved. The increased runtime between a mobile and Sentry being selected is due to the Controller constantly transmitting *standardControl* packets to a mobile when selected, but only transmitting a *sentryControl* to a selected Sentry if the power configuration selected by a user changes. As the power configuration does not change during testing, the XBee will only be transmitting to acknowledge *sentryResponse* packets received as part of its own DigiMesh protocol.

Figure 9.4: Controller operating times

While the lowest runtime for the Controller also exceeds the minimum 2 hour runtime objective, a LittleBot Junior demonstrates a runtime of up to 6 hours with just its video and motors enabled, and hence will last longer than the Controller commanding it. Ideally the runtime of the Controller should be extended to match this 6 hours, or even extended to 12 hours so that two Juniors may be consecutively driven. This could be achieved by using cells of higher capacity within the Controller, or attaching an external battery pack to the combination socket as previously mentioned in Section 5.2.7. In terms of purely receiving and storing data (and not piloting a LittleBot with video), the Controller lasts longer than a mobile LittleBot in any power configuration. However a Sentry will out-run the Controller if it is outfitted with no gas sensors, gas sensors operating intermittently, or supplemented by solar power as part of the future works discussed in Section 10.2. If such long-term data collection were required, the Controller could again be powered externally, with a mains supply power pack (if available) providing an indefinite runtime.

9.1.2 Charging

While not an objective, the amount of time taken to charge each device is measured. Similarly to the runtimes, each is completely discharged and the time taken to recharge measured three times with the mean taken. For all charging tests, the standard deviation of the measurements never exceeds 0.2 hours.

Figure 9.5: Charging times

Presented in Figure 9.5, the Junior and Standard LittleBots take an identical amount of time to charge. This is expected, as they have the same 2700 mAh two cell battery configuration. The higher capacity 5000 mAh cells of the Sentry take longer, while the 3200 mAh cells of the Controller is between the prior two combinations as expected. While a time of $\frac{2700}{1200} = 2.25$ hours to charge is anticipated for the 2700 mAh cells of the mobile LittleBots, this assumes a constant current during the entire charging process. As previously explained in Section 4.6, only the first stage of charging is constant current. Once the charger reaches constant voltage mode where the ceiling of 8.4 V is applied to the cells, the charge current begins to decrease as the capacity of the cells is reached.

9.2 Ingress Protection

Referring back to the objectives, an IP55 ingress protection rating is sought for each LittleBot. As specified by the International Electrotechnical Commission (IEC) in the 60529 standard [56], an IP rating comprises two components. The first number of the rating indicates the level of protection against dust ingress, and the second against water. For an IP55 rating, the first characteristic numeral 5 (IP5X) stipulates that “ingress of dust is not totally prevented, but dust shall not penetrate in a quantity to interfere with satisfactory operation of the apparatus or to impair safety”. Similarly, the second characteristic numeral 5 (IPX5) represents a level of protection where “water projected in jets against the enclosure from any direction shall have no harmful effects”

As a Standard LittleBot includes all ingress protection methods applied on the Sentry and Junior, it is selected for testing. Following the 14.2.5 test conditions for IPX5 [56], a hose is configured with a nozzle approximately 6 mm in diameter and the water flow rate adjusted to 12.5 litres per minute by timing how long it takes to fill a container of known volume. With the LittleBot powered, the water jet is applied to each surface from a distance of approximately 2.8 m, moving the LittleBot evenly through the jet to ensure all surfaces are sprayed over the course of 5 minutes. This is repeated three times, with the LittleBot disassembled after each test and inspected for water ingress.

Following three tests, no ingress is found. However as expected due to their exposed sensing elements, the gas module and DHT22 report abnormal readings. With the heater enabled, any water present in the gas module evaporates within 3 minutes and readings return to normal. As the DHT22 has no heater element to assist in evaporation, it takes 20 minutes to report the correct temperature and over an hour to report the correct relative humidity. Placed in close proximity to those mounted on the LittleBot chassis, control DHT22s and gas modules are used to verify the

readings. Aside from the temporary impairment of these sensors, the LittleBot continues to operate normally and an IPX5 water ingress protection rating demonstrated.

Specified by the 3.4 conditions to test for an IP5X dust ingress protection rating [56], a device is suspended in an enclosure with 2 kg of talcum powder per cubic metre of test chamber volume circulated for 2 hours using a powder circulation pump. With no readily achievable method of constructing the required environment (due to the lack of a powder circulation pump) and no suitable testing services available locally, an alternative is sought to evaluate the dust ingress protection of a LittleBot. As water is expected to enter a chassis more easily than talcum powder, submersion of a LittleBot can provide an indication as to whether the IP5X rating for dust has been met. This can also indicate whether the IPX5 water ingress rating has been exceeded, as it is expected that complete submersion is more taxing than the water jet testing performed. With the level of ingress protection implemented during the LittleBot design expected to withstand temporary submersion, this approach is selected as a satisfactory alternative for the IP5X test.

To allow the internal temperature and relative humidity of the LittleBot to be monitored during testing, a DHT22 is fitted inside the chassis as illustrated in Figure 9.6. Data is recorded for a minute, before the LittleBot is submerged in water for 30 minutes with the top of the LittleBot chassis 150 mm below the water line.

Figure 9.6: Internal DHT22

Immediately following testing, driving, video feedback and the LED illumination all operate as previously with no observable defects. However readings from the external DHT22 and gas sensor are again temporarily affected, with each reporting their maximum values. Similarly to the water jet testing, readings from the gas module return to normal within 3 minutes, while the temperature and relative humidity take 20 minutes and over an hour respectively.

Figure 9.7: DHT22 readings during submersion

From the internal DHT22 readings presented in Figure 9.7, the internal temperature decreases and relative humidity increases shortly following submersion of the LittleBot. With the LittleBot chassis being cooled by the water and in turn the air inside the LittleBot, the amount of water vapour the air can hold decreases. Hence as the amount of water vapour in the air inside the LittleBot stays constant, the relative humidity reported increases due to the reduction in capacity resulting from the temperature drop, explaining both results.

Finally removing the cover from the LittleBot, no visible signs of water ingress are found. Although this testing does not explicitly award the first characteristic numeral 5 (IP5X) rating due to deviations from the test conditions specified by the IEC60529 standard, a level of ingress protection exceeding IP55 is demonstrated on LittleBot through these tests.

9.3 Locomotion

Although not an objective, the locomotion of each LittleBot is evaluated to find the maximum speed and hence range, and compare the incline and terrain climbing capabilities between the two wheel designs proposed.

Figure 9.8: Proposed wheel types

As previously discussed and re-presented for clarification, the T1 wheel shown in (a) of Figure 9.8 is designed as a universal wheel, while T2 in (b) attempts to provide better climbing ability for more rugged terrain such as small rocks and gravel.

9.3.1 Speed

To evaluate the maximum speeds, each LittleBot is outfitted with the T1 wheels and driven in a straight line with turbo enabled for maximum performance. As the battery discharges, the voltage to the motors decreases

which affects their speed. Hence the motor speed tests are performed at 100% and 5% battery levels. Repeated five times and the means taken, the time to cover 3 m on a smooth level surface is recorded and used to find the speed of each at their minimum and maximum charges.

As shown in Figure 9.9, a Standard LittleBot is capable of covering 0.94 and 1.06 metres per second depending on the charge, and a Junior between 0.82 and 0.73 metres per second. As expected, the Junior LittleBot with two less motors but of similar weight is slowest overall. Finally taking the minimum speeds of each LittleBot and their respective 5.6 and 6 hour runtimes with just motors and video, the locomotive range for each device is calculated.

Figure 9.9: LittleBot speeds with full and empty batteries

Although unlikely to ever be realised in practice, a Junior LittleBot can be expected to cover $0.73 \times 60 \times 60 \times 6 = 15768$ metres of flat terrain, and a Standard $0.82 \times 60 \times 60 \times 6 = 16531$ metres. For both variants irrespective of charge, a LittleBot is at least twice as fast as the 0.3 metres per second a Throwbot is capable of. Based on the 2 hour runtime of a Throwbot, it can cover up $0.3 \times 60 \times 60 \times 2 = 2160$ metres of terrain on a single charge. At 14% of the capable range and less than half the speed of a Junior LittleBot, the

expensive Throwbot is clearly outperformed by LittleBots in both speed and the resulting locomotive range.

9.3.2 Incline

To evaluate the maximum slope a LittleBot may climb, each are tested on two surfaces of varying grip with different wheel configurations. Linoleum is used to represent a smooth, low friction surface and carpet a rough, high friction surface. The Junior is tested with both pairs of T1 and pairs of T2 wheels fitted, while a Standard is tested with four T1 wheels fitted, a pair of T1 wheels on the front and a pair of T2s on the rear, and finally T2s on the front and T1s on the rear. A test with all T2s on the Standard LittleBot is not performed due to only two T2 prototype wheels being available.

Figure 9.10: Maximum climbable inclines

With a LittleBot driving along the selected surface and turbo enabled for maximum power, the incline is increased until traction is lost and a LittleBot incapable of advancing any further. This is repeated five times, with the means for each situation presented in Figure 9.10 and a maximum

standard deviation of 1.1 degrees across all tests. Regardless of wheel type or situation, the Standard LittleBot always outperforms the Junior, with the tests of the Junior LittleBot suggesting that the T1 wheels provide more flat surface grip than the T2.

For comparison Baldwin street, the steepest street in New Zealand, has a maximum slope of 19 degrees. With asphalt expected to provide a grip similar to carpet, both LittleBots regardless of configuration would be capable of climbing such a steep incline. The Throwbot does not specify the maximum slope it can climb, however its lower speed and plastic wheels suggest it is not capable of climbing such a slope, and the LittleBot again is likely to outperform it.

9.3.3 Terrain

For completeness and to ensure LittleBot can be driven in outdoor environments, both mobile variants of LittleBot are test driven on gravel, dirt, short grass, long grass, carpet, and linoleum. In all but the long grass and gravel tests, both LittleBots perform similarly when fitted with T1 wheels. However for the long grass, the Junior LittleBot is far slower than a Standard, and struggles to move with increasing densities of grass.

Testing the LittleBots on gravel, again the Standard LittleBot performs without issue regardless of wheel configuration. However a Junior LittleBot fitted with T1 wheels occasionally struggles to gain traction, especially on inclines. Fitting the Junior LittleBot with T2 wheels improves traction in the same situation, however the Standard LittleBot is still clearly superior in terrain climbing capability. Referring back to Chapter 3 this is the expected performance, as the Standard is targeted at a greater range of terrain with four wheel drive and is a completely acceptable result. In terms of the wheel design, the T2 wheels provide better traction in the case of the Junior on gravel. However overall the T1 wheels provide the smoothest driving, and both variants are easier to control when fitted with T1 wheels.

9.4 Wireless Range

Although the tether-less objective is already met through the wireless systems employed as previously discussed, the range of these systems is evaluated to ascertain the operating range of a LittleBot, both in urban and line-of-sight situations.

9.4.1 Command and Control

With the command and control of a LittleBot provided by the DigiMesh network of the XBee-PRO modules, a range of a few kilometres is expected as Digi International specify a 10 km line-of-sight range with the 3 dBi antenna configuration used in LittleBot. With one LittleBot and the Controller in an open field, the distance between the two devices is increased until communications are lost as indicated by a message on the Controller display. This unexpectedly occurs at 82 m, over two orders of magnitude below what Digi International specify for the configuration used. Testing with all three variants of LittleBots exhibits the same result, differing by no more than a metre in range for each.

After updating the firmware and verifying the configuration of the XBee modules with no apparent solution for the abysmally short operating range, the power supply to each device is inspected for noise. This is done as excessive noise on the power supply could cause suboptimal operation, and is performed with the XBee removed to ensure the XBee itself does not generate any noise found.

Shown in Figure 9.11, there is approximately 200 mV peak to peak of noise on the 3.3 V supplied to the XBee. The noise observed is the result of using the PTH08080 switching regulator, and is to be expected. While this is considered an acceptable amount, it cannot be dismissed as a contributor to the short range observed. To check that the noise on the power supply is not the cause of the limited range, two MWEs (minimal working examples) are constructed using a linear regulator to supply the VCC to

all devices so that switching noise is not present.

Figure 9.11: Noise on the 3.3 V supply

Each MWE consists of a linear regulator, an ATmega328, an XBee attached to the same 3 dBi gain antenna, and an LED. Configured to light the LED for a second when a packet is received from the other node, the LittleBot software is loaded on one of the ATmega328's, while a version of the Controller software without the display libraries is small enough to be loaded on the second. In this configuration the MWE's execute the same procedure as a LittleBot and Controller, with the LittleBot programmed MWE associating with the Controller MWE. With the LED configured as explained to indicate a packet received, the MWE's can be deployed to test the DigiMesh range in an isolated way.

Repeating the same line-of-sight tests as previously with the MWE's, a similar range of 84 m is achieved. Attempting communication through walls in an urban environment, a range of 24 metres through a single plaster wall is achieved, and 15 m through two spaced 3 m apart. Again these are both orders of magnitude below what is expected, with Digi International specifying 610 m of urban range using a configuration identical to that applied in LittleBot. With the set up and configuration of each XBee verified, the abysmal range compared to the manufacturer's specifications

is inexplicable and far from what is expected. As the XBee Pro 900 HP is an extremely recent product, there is very little support through discussion and support forums. Digi International has been contacted for further support, however at the time of evaluation there has been no response and further follow-ups are underway.

Although the direct range is not what was expected, a short-range test of the mesh networking functionality provided by the XBees is undertaken. The Controller is placed out of range of a LittleBot, followed by a second LittleBot placed halfway between the first LittleBot and Controller. With this second LittleBot in place, self-healing is successfully demonstrated as the Controller resumes communication with the LittleBot that was previously out of range. Moving the Controller away from the middle LittleBot results in loss of communication with both at 82 m as expected. With the Controller placed 80 m away from the middle LittleBot, communication with the farther LittleBot is lost when placed more than 79 m from the middle one. Finally adding a third LittleBot to form a line with the previous two and the Controller, communication with this third node is possible at distances up to 80 m from the second. While the line-of-sight performance between nodes is unexpectedly reduced, the DigiMesh functionality is verified to be working as expected.

9.4.2 Video

Following the testing of the XBee's DigiMesh network, a similar testing approach is applied for the range of the video feedback beginning with line-of-sight range. As just discussed, the range of the DigiMesh network is currently limited to 82 m line-of-sight. The safety feature implemented into a LittleBot normally shuts down its video and motors when communications are lost to save power and prevent collisions. With the less than anticipated range of the DigiMesh network, this would limit the video range to that of the DigiMesh network during testing. As a result, the soft-

ware is modified for range testing so that the LittleBot's video system is not disabled when DigiMesh communication with the Controller is lost.

A Standard LittleBot is deployed in an open field, paired to the Controller with the video enabled. The distance between the Controller is then increased until visible static appears on the video feed, and the distance measured. At 96 m static begins to appear in the video feed, with video lost entirely at 102 m. While this exceeds the current DigiMesh network performance, again this is substantially lower than the anticipated performance.

Similarly to the XBee modules, MWE's are created using a TX5823 and camera for one, and an RX5808 with video display for the second. Again both are supplied using linear regulators to ensure power supply switching noise is not a contributing factor, and the line-of-sight test repeated. With a reduced range of 101 m demonstrated that is substantially less than the 2.1 km theoretical range calculated in Section 3.1.1, further investigation into the cause of the result is warranted.

Measuring the current consumed by the module, an average of 232 mA is drawn which is substantially less than the 320 mA specified. Removing the shielding cover of the TX5823, RichWave RTC6705 and RTC6671 IC's are found.

Referring to RichWave's listings for each device, the RTC6705 performs the frequency modulation with a power output of up to +13 dBm, and the RTC6671 is a power transmitter with a power of up to +18 dBm. Although it appears that the IC's can be cascaded to achieve a transmission power of $13 + 18 = 31$ dBm, obtaining the data sheets from RichWave reveals otherwise. The RTC6671 cannot handle an input power greater than +3 dBm, and the RTC6705 has a selectable output power of either +2 dBm or +13 dBm. Hence even when cascaded, a maximum transmission power of only $2 + 18 = 20$ dBm is possible with these two IC's. Further investigation implies that an RTC6679 capable of a +22 dBm output power is meant to be present in place of the RTC6671 inside the TX5823, and suggests that

the TX5823 modules supplied are counterfeit products.

With the poor video range attributed to counterfeit TX5823 modules and no response received from the supplier, it is expected a far greater range can be achieved with genuine modules. While time did not permit such modules to be obtained and fitted, the poor video range is an addressable issue and the tether-less video concept successfully demonstrated.

9.5 System Test

With each of the components tested individually, a system test is performed with three LittleBots in a room and an SD card inserted into the Controller for data collection. As each LittleBot is powered, they associate with the Controller as expected, and the Controller displays the correct number of LittleBots. Selection of an active LittleBot is successfully achieved using the designated selection buttons, with messages informing a user that the Controller is “Waiting for Response” in the interim between selection and a packet being received. With no issues in selecting and displaying data from a LittleBot, a test drive is performed using the video feed.

Toggling the video feed works as expected, while disabling the motors causes the joystick to become unresponsive and the LittleBots remain stationary. Each LittleBot is successfully piloted using the thumbstick, with the *Turbo* functionality halving the speed of the motors as expected. The gas sensors behave as expected, displaying a “Heat” message to the user during the first 30 seconds before displaying the ADC values while for a Sentry outfitted with PIR sensors, the OLED of the Controller displays the correct statuses for each. Using a mobile LittleBot in complete darkness with video feedback, the LEDs provide illumination of objects up to 20 m away and compares favourably to the Throwbot’s 8 m IR illumination. Finally, removing the SD card from the Controller and inserting it into a com-

puter, text files with the date of creation are present as expected. Opening these files, data received from LittleBots is stored as expected with packet receipt time, sender hardware address, and the decapsulated packet data in hex separated by tab characters.

9.6 Ease of Use

To evaluate the ease of use of LittleBots using the Controller, a user study involving 20 participants is commissioned with the required ethics approval obtained. The participants selected are of varying backgrounds and age, ranging from industrial design and architecture students, to sonic arts students and office administrators. No training is provided, other than to inform them that the Controller commands each of the LittleBots.

With a Standard and Junior LittleBot powered and associated to the Controller, each participant is supplied with the Controller and given the following list of tasks to carry out:

1. Locate the information display on the controller
2. Select the second LittleBot
3. Read the type of LittleBot (Standard, Sentry or Junior) from the information display
4. Turn the Camera on
5. Turn the Lights on
6. Turn the Motors on
7. Drive the LittleBot to the marked area
8. Turn the Gas sensor on
9. Read the gas level, temperature, humidity, and the battery level reported by the LittleBot

10. Select the first LittleBot
11. Deduce the type of LittleBot (Standard, Sentry or Junior)
12. Read the battery level of the LittleBot
13. Read the battery level of the Controller

Following completion of the tasks, each participant is given an evaluation form with a set of questions to rate aspects of the Controller and LittleBots pertaining to the tasks performed. Each is rated on a scale of 1 to 5, with a 1 representing very bad or difficult, and a 5 very good or easy.

Table 9.1: User study results

Question	Mean	Standard Deviation
How readable are the button labels?	4.3	0.8
How easy was it to select which LittleBot to pilot?	4.4	0.8
How easy was it to toggle the Camera/Lights/Motor power?	4.6	0.6
How easy was it to pilot the LittleBot to the marked area?	4.2	0.8
Is the size of the text on the information display easy to read?	3.7	1.1
Does the layout of information on the information display make it easy to understand?	4	1.1
How obvious was the Controller battery level?	3.5	1.2
How responsive did the selected LittleBot feel to joystick input?	4.4	0.9
How comfortable was the Controller to hold?	3.9	1.0
How easy do you feel the Controller was to use overall?	4.3	0.7

Summarised in Table 9.1, question responses with the lowest mean and highest standard deviations were those involving reading from the OLED information display. The lowest two infer the Controller battery level is not obvious and layout of the data on the display suboptimal, while the next implies that the size of the text is too small. All three of these could be corrected using a larger display with either larger pixels, or a higher number of pixels, and could be corrected on a production version of the Controller. Finally some users found the Controller not overly comfortable to hold, implying that the ergonomics could be improved. However an average of 4.3 for ease of use is awarded by the test group, indicating that the Controller is intuitive and easy to use, with the objective achieved.

In addition to these questions, users were provided the opportunity to give written feedback and suggestions for improvement. Four users commented that the button labels should be black to stand out more, while most comments related to the average readability of the information display already pointed out in the ratings. No other direct drawbacks were listed, however improvements including a neck strap for the Controller and engraving instructions on the rear were suggested. These are useful improvements, which could be implemented should the Controller be revised for production.

9.7 Cost

With a USD \$125 component cost targeted for a LittleBot, the final production cost in quantities of 500 for each LittleBot variant are summarised in Table 9.2. During design, each LittleBots chassis' components were manufactured through milling of aluminium and 3D printing for the prototypes. However as stated in Chapter 3 for mass production, these components could be manufactured out of ABS through injection moulding to reduce costs. Similarly for the wheels, the rubber could be cut using a stamp or die instead of water jet cutting.

Table 9.2: LittleBot component cost breakdown

Component	Junior	Standard	Sentry
Chassis, Cover & Retainers	\$10.82	\$12.96	\$14.13
Motors & Gearboxes	\$5.70	\$11.40	—
Motor Bearing Hubs	\$4.22	\$8.44	
Blank PCBs	\$1.09	\$1.09	\$1.09
Passive Components	\$9.87	\$9.87	\$6.13
Motor Drivers	\$3.25	\$6.50	—
Power Converters	\$5.49	\$5.49	\$10.16
Xbee	\$32.00	\$32.00	\$32.00
TX5823	\$8.22	\$8.22	—
Connectors & Cabling	\$8.59	\$11.51	\$4.39
Camera	\$8.45	\$8.45	—
LEDs	\$2.21	\$2.21	—
DHT22	—	\$2.72	\$2.72
GPS	—	—	\$29.95
Battery Cells & Protection	\$11.70	\$11.70	\$37.85
Bearings	\$3.96	\$7.92	—
Gaskets	\$1.27	\$1.37	\$2.17
Screws	\$5.38	\$7.04	\$3.23
Wheels	\$0.89	\$1.78	—
Total (USD)	\$122.22	\$148.89	\$143.82

Designed to be the cheapest device and costing only \$122.22 in components, the Junior LittleBot meets the \$125 objective. Reviewing the costs of the other variants, a Standard can be manufactured for 26% more, and a Sentry 16%. Further reductions in cost may be possible in greater quantities and further refined manufacturing techniques, however the current design satisfies the objectives set out in Section 1.2. As a result the Junior LittleBots can be considered disposable and may be deployed for USAR operations, with far less financial concern for their loss in comparison to current devices such as the \$13,000+ Throwbot XT.

9.8 Summary

Reviewing the evaluation of LittleBot, all objectives have been met or exceeded, with only the range of the tether-less operation objective less than anticipated. It is expected the video range can be increased through use of a genuine TX5823 as opposed to the counterfeit ones that were unfortunately acquired. The range of the DigiMesh network is yet to be explained, however as the XBees are a commercial product it is expected that a solution will be provided in the near future by the manufacturer. Comparing to the devices discussed in Chapter 2, the most similar device to LittleBot is the Throwbot XT. At a fraction of the cost, a Junior LittleBot outperforms the Throwbot in every aspect with the exception of being throwable. Although a robustness objective was not included, mobile LittleBots have sustained falls of up 2 m without damage.

Chapter 10

Conclusions

This chapter reviews the work completed, suggests future work, and concludes the project with reference to the original objectives listed in Section 1.2.

10.1 Review

Chapters 1 and 2 outlined the need for small and inexpensive robots for use in USAR (and potentially other) applications. This thesis presents the design and development of a solution to fill this void. Although a self-balancing design is initially proposed, the test platform and proof-of-concept constructed in Chapter 3 reveal stability issues with this design. Following lessons learned from this proof-of-concept design, three low cost device variants are investigated and developed, along with the Controller for easy, independent management of the LittleBots.

The locomotion of the four-wheel Standard and two-wheel Junior LittleBots is provided by 100:1 gear motors that are ingress protected using sealed bearings and a hub. Addressing the two-wheel design's stability issues, the axis of locomotion is positioned at the front of the Junior variant and a tail added for the device to rest on under its own weight. Wheels attached to hub are user exchangeable, providing the opportunity for ap-

plication specific wheels to be designed and easily swapped.

To achieve ingress protection, nitrile O-rings and gaskets are used throughout the design of all variants as required to seal most components. In addition to the camera and LEDs sealed behind polycarbonate windows on mobile LittleBots, Standard and Sentry variants are fitted with a DHT22 sensor to provide temperature and relative humidity readings. In this instance, the wires are sealed using silicon around the channel where the wires enter the LittleBot chassis. On all variants at least one gas module may be fitted in the form of an MQ series gas sensor mounted on a PCB with the supporting electronics. Secured with Philips head screws and sealed using an O-ring, these modules are user interchangeable, and a Sentry LittleBot may be configured with up to four gas or PIR sensors. While gas and PIR sensors are demonstrated, there is no limitation as to what type may be installed, as the software may be easily modified to accommodate these.

Coupled with an ATMega328 on each LittleBot for peripheral manipulation and sensor feedback, XBee Pro 900HP's provide a digital mesh network for command and control in an abstracted manner through DigiMesh. Video feedback is provided in analogue using TX5823 modules that frequency modulates the PAL composite output of a video camera. Using RP-SMA connectors for easy antenna replacement and reconfiguration, 3 dBi gain antennas are attached externally to both the XBee and TX5823 modules for increased range. For the high current LEDs and motors, ZXLD1350 and DRV8835 drivers are used respectively for supply and control. As motor drivers are inherently susceptible to damage due to the use of FETs, the DRV8835s are implemented on carrier PCBs that are installed into sockets on the LittleBot PCBs for easy replacement. Two lithium polymer batteries protected using a Seiko S8232 IC based protection PCB provide the power source, with remaining battery capacity estimated by measuring the cell voltage scaled through a voltage divider. Where high currents at a regulated voltage are required, switching sup-

plies are used over linear regulators for power conversion to minimise wastage and maximise runtime. Additionally, MOSFETs driven by the ATmega328 allows control of power to peripherals as selected by a user on the Controller.

For charging and master power control, a SwitchCraft combination jack is fitted to each LittleBot. To this jack, either the power key or a 1.2 A smart charger may be attached. The former turns the device on, and the latter charges both cells in series. Although the cells between mobile and Sentry LittleBots vary in capacity, the charger may be safely applied to all three. Similarly to the LittleBots, two lithium polymer cells are employed in the Controller for supply, with the same switching regulators providing voltage conversion where necessary. The Controller allows an operator to independently control LittleBots, using the same type of keys and charger.

The Controller is constructed using a combination of 3D printing and laser cut acrylic plates. Six buttons are included to allow a user to select a LittleBot and control the power status of each peripheral, as indicated by an LED integrated into the relevant button. To pilot a mobile LittleBot, a user may manipulate the thumbstick attached to analogue inputs on the ATmega2560. An ATmega2560 is employed instead of the LittleBot's ATmega328 due to the increased SRAM requirements of the SD card, OLED display, and larger serial buffer for the XBee. The SD card is user removable, and allows sensor data from all LittleBots to be stored with time and data stamping provided by an RTC with a battery backup. For information display to an operator including the currently selected unit and sensor readings, an OLED display controlled by the ATmega2560 is included. To display the video feed from a mobile LittleBot, an RX5808 5.8 GHz frequency demodulator with Innolux LCD display and driver are employed, with MOSFETs used to control the reception frequency.

With the LittleBots and Controller complete, the XBee modules are configured using the X-CTU utility, and the LittleBot protocol devised to achieve the communications required between LittleBots and the Con-

troller. With the XBee API necessary for packet transmission and reception handled in an existing library, the protocol is implemented as part of the software written for each of the robots. A simple association method allows the Controller to discover all active LittleBots and provisions each to learn the Controller's address for direct sensor feedback. Following association, the Controller may select any LittleBot for manipulation or piloting in the case of a mobile variant, while all sensor data reported by LittleBots is stored to an SD card if inserted.

10.2 Future Work

Demonstrated as part of the results in Chapter 9, there are some clear areas of improvement for the LittleBots and the Controller. The XBee's unexpectedly reduced range requires further investigation, however a solution is expected with support from Digi International. Similarly for the reduced video range, a solution is expected by using genuine TX5823 modules over the suspected counterfeit modules supplied.

While the cost objective has been met, further reductions in cost can certainly be realised through design refinements that target the selected manufacturing techniques. Although injection moulded ABS for the chassis and stamped rubber for the wheels contribute towards cheap mass manufacturing, further reduction in costs may be possible through simplification of the bearing system. As part of this simplification, impact tolerance of the chassis and locomotion could be investigated and further developed to make a LittleBot throwable.

Although the Controller demonstrates excellent intuitiveness and ease of use, improvements in the areas of ergonomics and readability of information are suggested. As mentioned by the majority of the user test group, the Controller could be more comfortable to hold, while some found the information display difficult to read. The ergonomics may be improved through a refined physical design in consultation with industrial design-

ers, while the readability of the display could be improved by employing a larger display with a revised information layout. As part of this refinement, the chassis could also be modified to better suit mass manufacturing techniques as opposed to the assembled frame approach suitable for rapid prototyping.

Finally, LittleBots could be incorporated into the existing three-tier hierarchical system at VUW by fitting the mother or grandmother devices with an XBee and a 5.8 GHz receiver. With these devices processing the video feed and controlling each LittleBot as required, autonomous operation may be achieved. Should longer-range video feed back be required, the electronics of each LittleBot could be further developed to act as video repeater. With headers for the attachment of solar cells on each LittleBot, further extended runtimes could also be achieved through the development of a solar charging and a more advanced power management system.

10.3 Summary

Referring back to the original objectives set out in Section 1.2, all have been met or exceeded as demonstrated during the evaluation performed in Chapter 9. With a cost of USD \$122.22, a Junior LittleBot is sufficiently inexpensive to be considered disposable. Tether-less operation, video feedback, and a compact size are achieved in principle, with DigiMesh demonstrated and the reduced operating range expected to be resolved in the near future. In addition to a variety of gas modules that are user exchangeable, a Sentry may also be outfitted with PIR sensors for motion detection, and there is no limitation on what other sensor modules may be designed for a LittleBot.

The IP55 objective is achieved, with a LittleBot withstanding complete submersion under 15 cm of water for half an hour. The 2 and 24 hour runtime objectives are exceeded, with minimum times of 2.7 and 62.1 hours

achieved respectively. Finally, the Controller provides an easy to use system to monitor and control the LittleBots, satisfying the intuitive use objective as demonstrated through user testing. This work has applications outside of USAR, for example as generic inexpensive mobile and reconfigurable sensors platforms, and a paper on this theme is to be presented at the 2014 IEEE International Instrumentation and Measurement Conference in May [57].

In summary, the LittleBot system illustrated in Figure 10.1 is an inexpensive and practical system to aid in USAR and environment sensing operations. All objectives other than wireless range have been met or exceeded, finally providing the Mechatronics Group of VUW with a viable “daughter” device.

Figure 10.1: The LittleBot system

Bibliography

- [1] NZFS. Urban search and rescue (usar). Accessed 19 February 2014. [Online]. Available: <http://www.fire.org.nz/About-Us/Our-Organisation/Pages/UrbanSearchandRescue.aspx>
- [2] R. Ventura and P. Lima, "Search and rescue robots: The civil protection teams of the future," in *Third International Conference on Emerging Security Technologies*, Sept 2012, pp. 12–19.
- [3] R. Murphy, J. Casper, J. Hyams, M. Micire, and B. Minten, "Mobility and sensing demands in usar," in *26th Annual Conference of the IEEE Industrial Electronics Society*, vol. 1, 2000, pp. 138–142.
- [4] J. Booker, "Robot broken in pike river mine," *The New Zealand Herald*, November 2010.
- [5] R. Macfie, *Tragedy at Pike River Mine: how and why 29 men died*. Awa Press, 2013.
- [6] D. A. Carnegie, "A three-tier hierarchical robotic system for urban search and rescue applications," in *IEEE International Workshop on Safety, Security and Rescue Robotics*, 2007, pp. 1–6.
- [7] B. Chemel, E. Mutschler, and H. Schempf, "Cyclops: miniature robotic reconnaissance system," in *IEEE International Conference on Robotics and Automation*, vol. 3, 1999, pp. 2298–2302.

- [8] J. Mullins, B. Horan, M. Fielding, and S. Nahavandi, "A haptically enabled low-cost reconnaissance platform for law enforcement," in *IEEE International Workshop on Safety, Security and Rescue Robotics*, Sept 2007, pp. 1–6.
- [9] J. G. Blitch and R. Maurer, "Knobsar: A knowledge based system prototype for robot assisted urban search and rescue," *Simulation*, vol. 66, no. 6, pp. 375–391, 1996.
- [10] M. Guarnieri, I. Takao, P. Debenest, K. Takita, E. Fukushima, and S. Hirose, "Helios ix tracked vehicle for urban search and rescue operations: Mechanical design and first tests," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept 2008, pp. 1612–1617.
- [11] R. Murphy, "Marsupial and shape-shifting robots for urban search and rescue," *IEEE Intelligent Systems and their Applications*, vol. 15, no. 2, pp. 14–19, Mar 2000.
- [12] K. Osuka, T. Doi, and Y. Yokokouji, "In-rubble robot system for use under debris," in *SICE-ICASE International Joint Conference*, Oct 2006, pp. 3439–3442.
- [13] G. Dissanayake, J. Paxman, J. Miro, O. Thane, and H.-T. Thi, "Robotics for urban search and rescue," in *First International Conference on Industrial and Information Systems*, Aug 2006, pp. 294–298.
- [14] R. Murphy, "Trial by fire [rescue robots]," *IEEE Robotics Automation Magazine*, vol. 11, no. 3, pp. 50–61, Sept 2004.
- [15] A. Davids, "Urban search and rescue robots: from tragedy to technology," *IEEE Intelligent Systems*, vol. 17, no. 2, pp. 81–83, March 2002.
- [16] J. A. Kramer and R. R. Murphy, "Endurance testing for safety, security and rescue robots," in *Performance Metrics for Intelligent Systems Workshop*, 2006, pp. 247–254.

- [17] D. Hougen, S. Benjaafar, J. Bonney, J. Budenske, M. Dvorak, M. Gini, H. French, D. Krantz, P. Li, F. Malver, B. Nelson, N. Papanikolopoulos, P. Rybski, S. Stoeter, R. Voyles, and K. Yesin, "A miniature robotic system for reconnaissance and surveillance," in *IEEE International Conference on Robotics and Automation*, vol. 1, Apr 2000, pp. 501–507.
- [18] J. Huff, S. Conyers, and R. Voyles, "MOTHERSHIP - a serpentine tread/limb hybrid marsupial robot for USAR," in *IEEE International Symposium on Safety, Security, and Rescue Robotics*, Nov 2012, pp. 1–7.
- [19] R. S. King, "State of the art in robotics and robotic actuation," in *BiLB IQ: A Biologically Inspired Robot with Walking and Rolling Locomotion*. Springer, 2013, vol. 2, pp. 29–47.
- [20] Recon Scout Throwbot LE specifications. Accessed 19 February 2014. [Online]. Available: http://www.recon-scout.com/pdfs/Recon_Scout_Throwbot_LE_microprice.pdf
- [21] Recon Throwbot XT. Accessed 19 February 2014. [Online]. Available: <http://www.pr-infrared.com/shop/recon-scout-throwbot/>
- [22] G. Aggelou, *Wireless Mesh Networking*. McGraw-Hill Professional, 2008.
- [23] J. Adams, "An introduction to IEEE STD 802.15.4," in *IEEE Aerospace Conference*, 2006, p. 8.
- [24] X. Liu, H. Chen, M. Wang, and S. Chen, "An XBee-Pro based energy monitoring system," in *Australasian Telecommunication Networks and Applications Conference*, Nov 2012, pp. 1–6.
- [25] N. S. A. Zulkifli, F. Harun, and N. S. Azahar, "XBee wireless sensor networks for heart rate monitoring in sport training," in *International Conference on Biomedical Engineering*, Feb 2012, pp. 441–444.

- [26] K. Jaskot and A. Babiarz, "Autonomous mobile platform II," in *International Carpathian Control Conference*, May 2011, pp. 167–171.
- [27] V. Visoottiviseth, T. Piroonsith, and S. Siwamogsatham, "An empirical study on achievable throughputs of IEEE 802.11n devices," in *7th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, June 2009, pp. 1–6.
- [28] O. Harmanci and A. Tekalp, "Optimization of H264 for low delay video communications over lossy channels," in *International Conference on Image Processing*, vol. 5, Oct 2004, pp. 3209–3212.
- [29] A. Skinner and M. Lambert, "Using smart sensor strings for continuous monitoring of temperature stratification in large water bodies," *IEEE Sensors Journal*, vol. 6, no. 6, pp. 1473–1481, Dec 2006.
- [30] B. Le Comte, G. Sen Gupta, and M.-T. Chew, "Distributed sensors for hazard detection in an urban search and rescue operation," in *IEEE International Instrumentation and Measurement Technology Conference*, May 2012, pp. 2385–2390.
- [31] D. J. Bruemmer, D. D. Dudenhoeffer, M. D. McKay, and M. O. Anderson, "A robotic swarm for spill finding and perimeter formation," DTIC Document, Tech. Rep., 2002.
- [32] G. Zhang, G. Fricke, and D. Garg, "Spill detection and perimeter surveillance via distributed swarming agents," *IEEE/ASME Transactions on Mechatronics*, vol. 18, no. 1, pp. 121–129, Feb 2013.
- [33] M. Masar, "A biologically inspired swarm robot coordination algorithm for exploration and surveillance," in *IEEE 17th International Conference on Intelligent Engineering Systems*, June 2013, pp. 271–275.
- [34] A. Marjovi and L. Marques, "Swarm robotic plume tracking for intermittent and time-variant odor dispersion," in *European Conference on Mobile Robots*, Sept 2013, pp. 379–384.

- [35] H. Yamamoto, S. Minomi, K. Nakamura, and K. Yamazaki, "Development of arkas sensor platform and application for heavy snowfall observation," in *12th International Symposium on Applications and the Internet*, July 2012, pp. 399–404.
- [36] R. Kastner, A. Lin, C. Schurgers, J. Jaffe, P. Franks, and B. Stewart, "Sensor platforms for multimodal underwater monitoring," in *International Green Computing Conference*, June 2012, pp. 1–7.
- [37] D. Wettergreen, C. Thorpe, and R. Whittaker, "Exploring mount erebus by walking robot," *Robotics and Autonomous Systems*, vol. 11, no. 3, pp. 171–185, 1993.
- [38] J. E. Bares and D. S. Wettergreen, "Dante ii: Technical description, results, and lessons learned," *The International Journal of Robotics Research*, vol. 18, no. 7, pp. 621–649, 1999.
- [39] G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Welsh, "Deploying a wireless sensor network on an active volcano," *IEEE Internet Computing*, vol. 10, no. 2, pp. 18–25, 2006.
- [40] *XBee Pro 900 HP datasheet*, Digi International Limited, September 2012. [Online]. Available: ftp://ftp1.digi.com/support/documentation/90002173_B.pdf
- [41] H. E. Co, *MQ9 datasheet*. [Online]. Available: <http://www.pololu.com/file/0J314/MQ9.pdf>
- [42] *ATMega328 datasheet*, ATMel Corporation, October 2009. [Online]. Available: <http://www.atmel.com/Images/doc8161.pdf>
- [43] FH10C V9 PAL Camera. Accessed 3 March 2014. [Online]. Available: http://old.fpvmodel.com/fh10c-v9-5v-pal-90-degree-fov-micro-camera-lens_p829.html

- [44] *ZXLD1350 datasheet*, Diodes Incorporated, March 2011. [Online]. Available: <http://www.diodes.com/datasheets/ZXLD1350.pdf>
- [45] *DRV8835 datasheet*, Texas Instruments, January 2014. [Online]. Available: <http://www.ti.com/lit/ds/symlink/drv8835.pdf>
- [46] Figaro gas sensors. Last Accessed 3 March 2014. [Online]. Available: <http://www.figarosensor.com>
- [47] *DHT22 (AM2302) datasheet*, Aosong Electronics Company Limited, January 2013. [Online]. Available: <http://meteobox.tk/files/AM2302.pdf>
- [48] *GP-2106 datasheet*, ADH Technology Company Limited, August 2010. [Online]. Available: [http://www.adh-tech.com.tw/files/GP-2106\[2\].pdf](http://www.adh-tech.com.tw/files/GP-2106[2].pdf)
- [49] *S8232 datasheet*, Seiko Instruments Incorporated, 2011. [Online]. Available: http://datasheet.sii-ic.com/en/battery_protection/S8232_E.pdf
- [50] *PTH08080W datasheet*, Texas Instruments, September 2013. [Online]. Available: <http://www.ti.com/lit/ds/slts235d/slts235d.pdf>
- [51] *3P10-L1004 datasheet*, Master Instruments, March 2007. [Online]. Available: <http://www.batteryspace.com/prod-specs/4208.pdf>
- [52] *KYV-N2 datasheet*, VS Display Technology Limited, April 2010. [Online]. Available: [http://www.vslcd.com/Specification/KYV-N2%20V1\(A104SN03\).pdf](http://www.vslcd.com/Specification/KYV-N2%20V1(A104SN03).pdf)
- [53] *ATMega2560 datasheet*, ATMel Corporation, October 2012. [Online]. Available: <http://www.atmel.com/images/doc2549.pdf>
- [54] xbee-arduino library. Accessed 19 February 2014. [Online]. Available: <https://code.google.com/p/xbee-arduino/>

- [55] Adafruit SSD1306 library. Adafruit Industries. Accessed 19 February 2014. [Online]. Available: https://github.com/adafruit/Adafruit_SSD1306
- [56] *IEC60529 standard*, 2nd ed., International Electrotechnical Commission, August 2013.
- [57] D. A. Carnegie and J. McVay, "A maneuverable, configurable, low cost sensor deployment platform," in *IEEE International Instrumentation and Measurement Technology Conference*, May 2014, to be presented.