CHAPTER 1

Architecture

Since the advent of a standardised Internet of Things (IoT) protocol stack discussed in Section **??** on page ??, the decision making process for protocol architecture has been simplified immensely. As a key part of an effective Thing is interoperability, it is clear that adopting the standardised protocol stack is the way forward. As such, the proposed protocol architecture described in Table **??** on page ?? will form the stack used by the "WPAN" network shown in Figure **??** on page ??.

Moving from a protocol perspective to a device perspective, when one considers the energy efficiency and cost constrains of this project, it is clear that a system in which low-powered and cheap embedded systems, such as Arduinos, are the best choice for each of the sensing nodes. This recognises the fact that these nodes have computationally complex tasks, and are merely responsible for the transmission of the collected data.

As a natural consequence of choosing simple sensing nodes, a more powerful processing node must be added to the system to collect the unprocessed data produced by the sensing nodes and interpret it into the high-level occupancy answers this project wishes to provide. As such a node does not need to be in a particular location (provided it is in range of the sensor WPAN), it does not need to be as considerate of low power requirements. A primary hardware candidate for this node is the Raspberry Pi. Advantages include it still being quite low powered, built-in support for WPAN networking expansion cards, and traditional built-in LAN networking. These characteristics also allow it to act as the "smart gateway" between the sensors and the broader IoT.

1.1 Prototype System Architecture

Due to limited time available, parts of the above ideal system architecture have been deemed outside of the scope of the project. To help achieve appreciable results in the time available, the use of wireless mesh networking and the support of a one-to-many "smart gateway" to sensor has not been explored. However, as discussed below, the prototype architecture selected as been designed such that a clear path to the idea system architecture is available.

1.1.1 Hardware

Due to low cost and ease of use, the Arduino platform was selected as the host for the low-level I^2C interface for communication to the Melexis MLX90620 (*Melexis*). Initially, this presented some challenges, as the *Melexis* recommends a power and communication voltage of 2.6V, while the Arduino is only able to output 3.3V and 5V as power, and 5V as communication. Due to this, it was not possible to directly connect the Arduino to the *Melexis*, and similarly due to the two-way nature of the I^2C 2-wire communication protocol, it was also not possible to simply lower the Arduino voltage using simple electrical techniques, as such techniques would interfere with two-way communication.

A solution was found in the form of a I^2C level-shifter, the Adafruit "4-channel I2C-safe Bi-directional Logic Level Converter" [1], which provided a cheap method to bi-directionally communicate between the two devices at their own preferred voltages. The layout of the circuit necessary to link the Arduino and the *Melexis* using this converter can be seen in Figure 1.1 on the next page.

Additionally, as used in the Thermosense paper, a Passive Infrared Sensor (PIR) motion sensor [2] was also connected to the Arduino . This sensor, operating at 5V natively, did not require any complex circuitry to interface with the Arduino . It is connected to digital pin 2 on the Arduino , where it provides a rising signal in the event that motion is detected, which can be configured to cause an interrupt on the Arduino . In the configuration used in this project, the sensor's sensitivity was set to the highest value (TODO: check) and the time-out for re-triggering was set to the lowest value (approximately 2.5 seconds). Additionally, the continuous re-triggering feature (whereby the sensor produces continuous rising and falling signals for the duration of motion) was disabled using the provided jumpers.

1.1.2 Software

To calculate the final temperature values that the *Melexis* offers, a complex initialisation and computational process must be followed, which is specified in the sensor's datasheet [9]. This process involves initialising the sensor with values attained from a separate on-board I^2C EEPROM, then retrieving a variety of

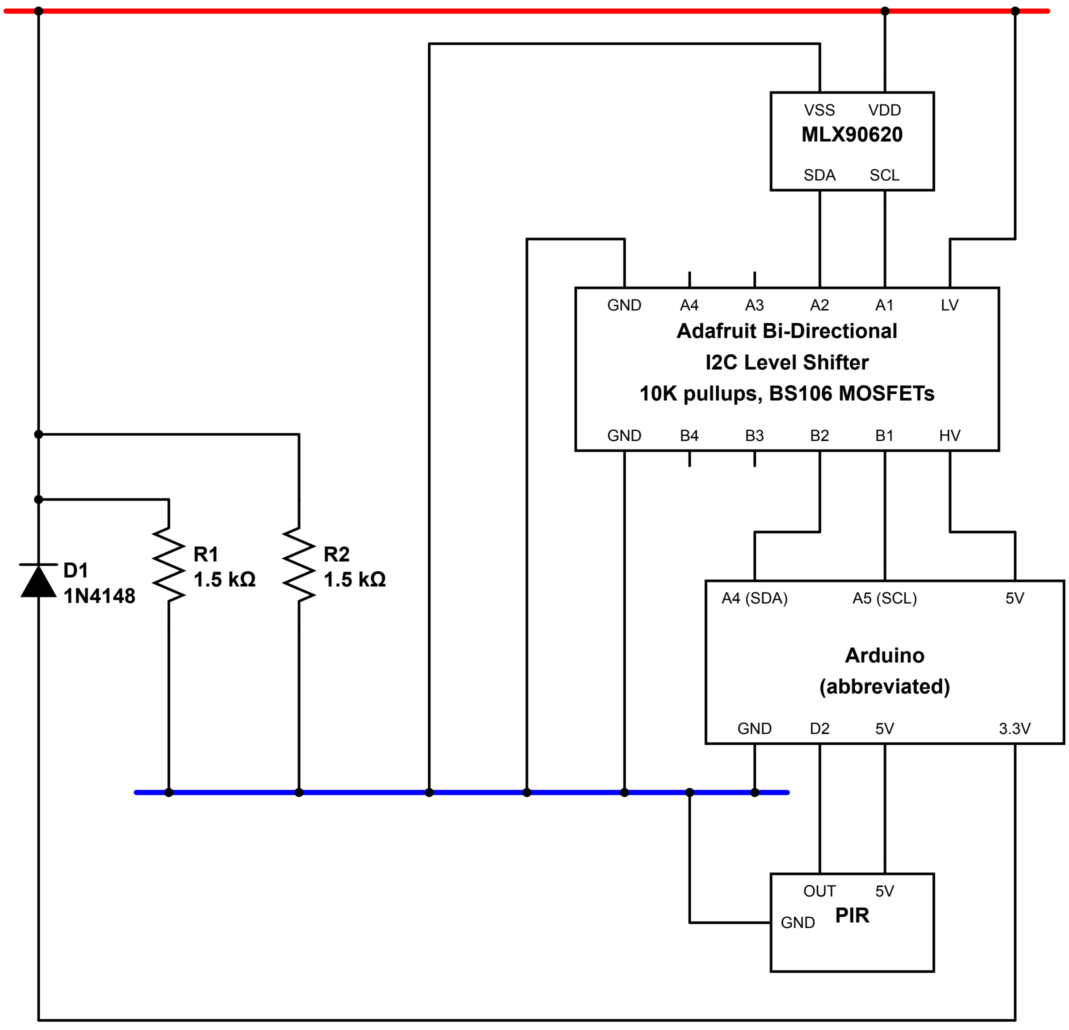


Figure 1.1: MLX90620, PIR and Arduino integration circuit

```
INIT 0
INFO START
DRIVER MLX90620
BUILD Feb  1 2015 00:00:00
IRHZ 1
INFO STOP
ACTIVE 33
```

Figure 1.2: Initialisation sequence

normalisation and adjustment values, along with the raw sensor data, to compute the final temperature result.

The basic algorithm to perform this normalisation was based upon code by users “maxbot”, “IIBaboomba”, “nseidle” and others on the Arduino Forums [3] and was modified to operate with the newer Arduino “Wire” I^2C libraries released since the authors’ posts. In pursuit of the project’s aims to create a more approachable thermal sensor, the code was also restructured and rewritten to be both more readable, and to introduce a set of features to make the management of the sensor data easier for the user, and for the information to be more human readable.

The first of the features introduced was the human-readable format for serial transmission. This allows the user to both easily write code that can parse the serial to acquire the serial data, as well as examine the serial data directly with ease. When the Arduino first boots running the software, the output in Figure 1.2 is output. This specifies several things that are useful to the user; the attached sensor (“DRIVER”), the build of the software (“BUILD”) and the refresh rate of the sensor (“IRHZ”). Several different headers, such as “ACTIVE” and “INIT” specify the current millisecond time of the processor, thus indicating how long the execution of the initialisation process took (33 milliseconds).

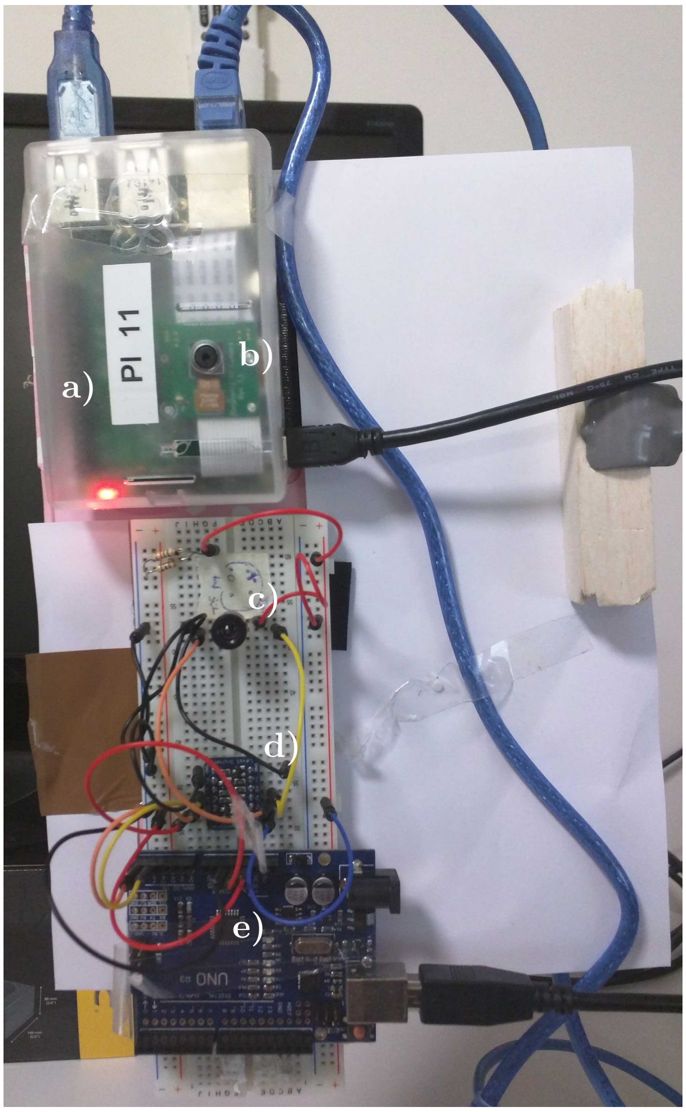
Once booted, the user is able to send several one-character commands to the sensor to configure operation, which are described in Table 1.1 on the next page. Depending on the sensor configuration, IR data may be periodically output automatically, or otherwise manually triggered. This IR data is produced in the packet format described in Figure 1.3 on the following page. This is a simple, human readable format that includes the millisecond time of the processor at the start and end of the calculation, if the PIR has seen any motion for the duration of the calculation, and the 16x4 grid of calculated temperature values.

`R`	Flush buffers and reset Arduino
`I`	Print INFO again
`T`	Activate timers for periodic IR data output
`O`	Deactivate timers for periodic IR data output
`P`	Manually trigger capture and output of IR data
`F`x	Set sensor refresh frequency to x and reboot

Table 1.1: Commands

```
START 34
MOVEMENT 0
1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
STOP 97
```

Figure 1.3: Thermal data packet



a) Raspberry Pi

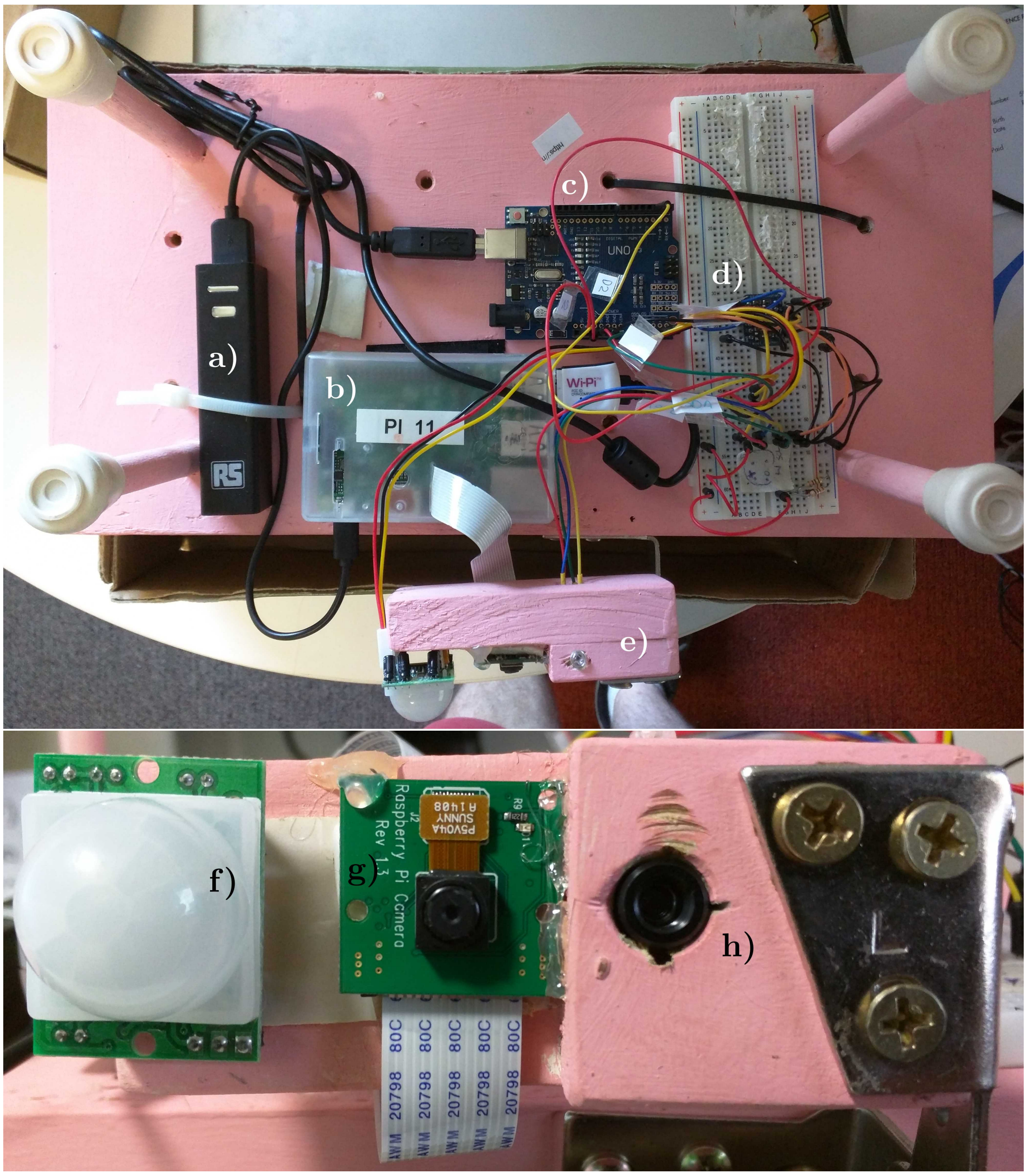
b) Camera

c) *Melexis*

d) Level-shifting circuitry

e) Arduino

Figure 1.4: Prototype A



a) Battery pack
b) Raspberry Pi
c) Arduino
d) Level-shifting circuitry
e) Movable sensor mount
f) PIR
g) Camera
h) *Melexis*

Figure 1.5: Prototype B

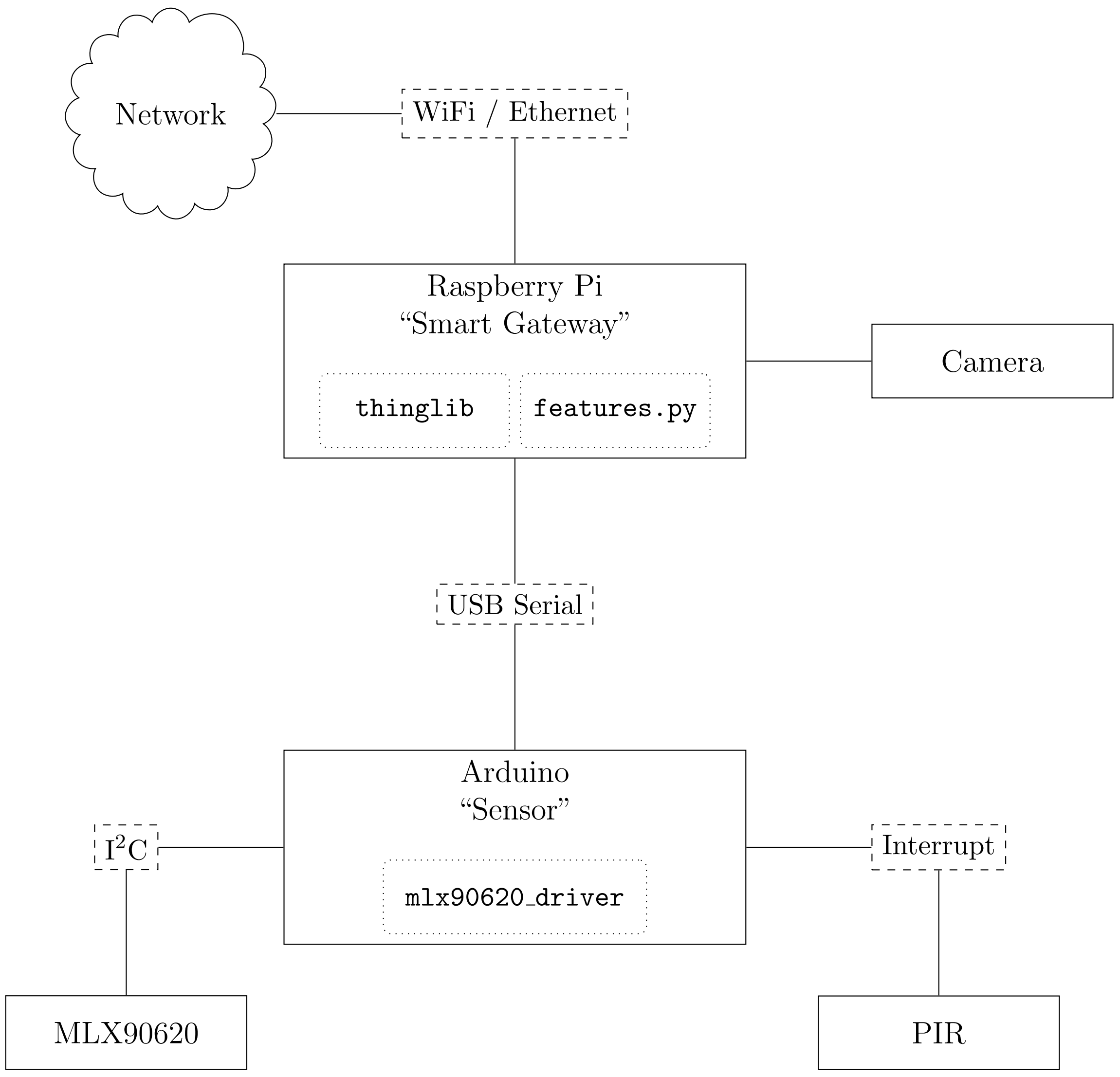


Figure 1.6: Prototype B system architecture

Bibliography

[1] Adafruit. 4-channel I2C-safe bi-directional logic level converter - BSS138 (product ID 757). `http://www.adafruit.com/product/757`. Accessed: 2015-01-07.

[2] Adafruit. PIR (motion) sensor (product ID 189). `http://www.adafruit.com/product/189`. Accessed: 2015-02-08.

[3] Arduino Forums. Arduino and MLX90620 16X4 pixel IR thermal array. `http://forum.arduino.cc/index.php/topic,126244.0.html`, 2012. Accessed: 2015-01-07.

[4] Atzori, L., Iera, A., and Morabito, G. The internet of things: A survey. *Computer networks 54*, 15 (2010), 2787–2805.

[5] Beltran, A., Erickson, V. L., and Cerpa, A. E. ThermoSense: Occupancy thermal based sensing for HVAC control. In *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings* (2013), ACM, pp. 1–8.

[6] Guinard, D., Ion, I., and Mayer, S. In search of an internet of things service architecture: REST or WS-*? a developers perspective. In *Mobile and Ubiquitous Systems: Computing, Networking, and Services*. Springer, 2012, pp. 326–337.

[7] Guinard, D., Trifa, V., Mattern, F., and Wilde, E. From the internet of things to the web of things: Resource-oriented architecture and best practices. In *Architecting the Internet of Things*. Springer, 2011, pp. 97–129.

[8] Kovatsch, M. CoAP for the web of things: from tiny resource-constrained devices to the web browser. In *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication* (2013), ACM, pp. 1495–1504.

[9] Melexis. Datasheet IR thermometer 16X4 sensor array MLX90620. `http://www.melexis.com/Asset/Datasheet-IR-thermometer-16X4-sensor-array-MLX90620-DownloadLink-6099.aspx`, 2012. Accessed: 2015-01-07.

[10] Palattella, M. R., Accettura, N., Vilajosana, X., Watteyne, T., Grieco, L. A., Boggia, G., and Dohler, M. Standardized protocol stack for the internet of (important) things. *Communications Surveys & Tutorials, IEEE 15*, 3 (2013), 1389–1406.

[11] Shelby, Z., and Bormann, C. *6LoWPAN: The wireless embedded Internet*, vol. 43. John Wiley & Sons, 2011.

[12] Winter, T., Thubert, P., Cisco Systems, Brandt, A., et al. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550, Internet Engineering Task Force, March 2012. Retrieved October 6, 2014 from `http://tools.ietf.org/html/rfc6550`.