

Developing a robust system for occupancy detection in the household

Ash Tyndall

*This report is submitted as partial fulfilment
of the requirements for the Honours Programme of the
School of Computer Science and Software Engineering,
The University of Western Australia,
2015*

Abstract

This is the abstract.

Keywords: keyword, keyword

CR Categories: category, category



© 2014–15 Ashley Ben Tyndall

This document is released under a Creative Commons Attribution-ShareAlike 4.0 International License. A copy of this license can be found at <http://creativecommons.org/licenses/by-sa/4.0/>.

A digital copy of this document and supporting files can be found at <http://github.com/atyndall/honours>.

The following text can be used to satisfy attribution requirements:

“This work is based on the honours research project of Ash Tyndall, developed with the help of the School of Computer Science and Software Engineering at The University of Western Australia. A copy of this project can be found at <http://github.com/atyndall/honours>.”

Code and code excerpts included in this document are instead released under the GNU General Public License v3, and can be found in their entirety at <https://github.com/atyndall/thing>.

Acknowledgements

These are the acknowledgements.

Contents

Abstract	ii
Acknowledgements	iv
1 Introduction	1
2 Literature Review	3
2.1 Intrinsic traits	4
2.1.1 Static traits	4
2.1.2 Dynamic traits	6
2.2 Extrinsic traits	6
2.2.1 Instrumented traits	6
2.2.2 Correlative traits	8
2.3 Analysis	8
2.4 Thermal sensors	10
2.5 Research Gap	11
2.6 Conclusion	12
3 Architecture	13
3.1 Ideal System Architecture	13
3.1.1 Protocols	14
3.1.2 Devices	15
3.2 Prototype System Architecture	17
3.2.1 Hardware	17
3.2.2 Software	19
4 Sensor Properties	24

5	Methods	26
6	Results	27
7	Discussion and Conclusion	28
7.1	Future Directions	28
A	Original Honours Proposal	29
A.1	Background	29
A.2	Aim	30
A.3	Method	31
A.3.1	Hardware	31
A.3.2	Classification	31
A.3.3	Robustness / API	32
A.4	Timeline	32
A.5	Software and Hardware Requirements	33
A.6	Proposal References	34
B	Code Listings	36
B.1	ThingLib	36
B.1.1	cam.py	36
B.1.2	pxdisplay.py	49
B.1.3	features.py	51
B.2	Arduino Sketch	55
	Bibliography	72

List of Tables

2.1	Comparison of different sensors and project requirements	9
3.1	Proposed protocol stack	14
3.2	Commands	20
4.1	Mean and standard deviations for each pixel at rest	25

List of Figures

2.1	Taxonomy of occupancy sensors	4
3.1	Proposed system architecture	16
3.2	MLX90620, PIR and Arduino integration circuit	18
3.3	Initialisation sequence	20
3.4	Thermal data packet	20
3.5	Prototype A	21
3.6	Prototype B	22
3.7	Prototype B system architecture	23

CHAPTER 1

Introduction

The proportion of elderly and mobility-impaired people is predicted to grow dramatically over the next century, leaving a large proportion of the population unable to care for themselves, and also reducing the number of human carers available [8]. With this issue looming, investments are being made in technologies that can provide the support these groups need to live independent of human assistance.

With recent advance in low cost embedded computing, such as the Arduino and Raspberry Pi, the ability to provide a set of interconnected sensors, actuators and interfaces to enable a low-cost ‘smart home for the disabled’ that takes advantage of the Internet of Things (IoT) is becoming increasingly achievable.

Sensing techniques to determine occupancy, the detection of the presence and number of people in an area, are of particular use to the elderly and disabled. Detection can be used to inform various devices that change state depending on the user’s location, including the better regulation energy hungry devices to help reduce financial burden. Household climate control, which in some regions of Australia accounts for up to 40% of energy usage [5] is one area in which occupancy detection can reduce costs, as efficiency can be increased with annual energy savings of up to 25% found in some cases [7].

While many of the above solutions achieve excellent accuracies, in many cases they suffer from problems of installation logistics, difficult assembly, assumptions on user’s technology ownership and component cost. In a smart home for the disabled, accuracy is important, but accessibility is paramount.

The goal of this research project is to devise an occupancy detection system that forms part of a larger ‘smart home for the disabled’, and integrates into the IoT, that meets the following qualitative accessibility criteria;

- *Low Cost*: The set of components required should aim to minimise cost, as these devices are intended to be deployed in situations where the serviced user may be financially restricted.

- *Non-Invasive*: The sensors used in the system should gather as little information as necessary to achieve the detection goal; there are privacy concerns with the use of high-definition sensors.
- *Energy Efficient*: The system may be placed in a location where there is no access to mains power (e.g. roof), and the retrofitting of appropriate power can be difficult; the ability to survive for long periods on only battery power is advantageous.
- *Reliable*: The system should be able to operate without user intervention or frequent maintenance, and should be able to perform its occupancy detection goal with a high degree of accuracy.

To create a picture of what options there are in this sensing area, a literature review of the available sensor types and wireless sensor architectures is needed. From this list, proposed solutions will be compared against the aforementioned accessibility criteria to determine their suitability.

CHAPTER 2

Literature Review

To achieve the accessibility criteria, a wide variety of sensing approaches must be considered. It can be difficult to approach the board variety of sensor types in the field, so a structure must be developed through which to evaluate them. Teixeira, Dublon and Savvides [24] propose a 5-element human-sensing criteria which provides a structure through which we may define the broad quantitative requirements of different sensors.

These quantitative requirements can be used to exclude sensing options that clearly cannot meet the requirements before the more specific qualitative accessibility criteria will be considered for those remaining sensors.

The quantitative criteria elements are;

1. *Presence*: Is there any occupant present in the sensed area?
2. *Count*: How many occupants are there in the sensed area?
3. *Location*: Where are the occupants in the sensed area?
4. *Track*: Where do the occupants move in the sensed area? (local identification)
5. *Identity*: Who are the occupants in the sensed area? (global identification)

At a fundamental level, this research project requires a sensor system that provides both Presence and Count information. To assist with the reduction of privacy concerns, excluding systems that permit Identity will generally result in a less invasive system also. The presence of Location or Track are irrelevant to our project's goals, but overall, minimising these elements should in most cases help to maximise the energy efficiency of the system also.

Teixeira, Dublon and Savvides [24] also propose a measurable occupancy sensor taxonomy (see Figure 2.1 on the following page), which categorises different

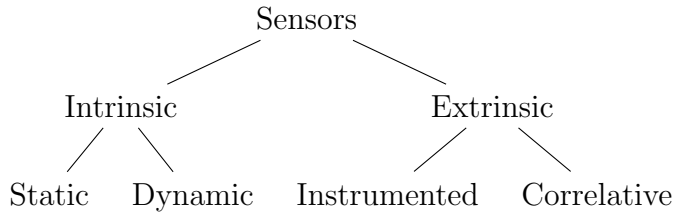


Figure 2.1: Taxonomy of occupancy sensors

sensing systems in terms of what information they use as a proxy for human-sensing. We use this taxonomy here as a structure through which we group and discuss different sensor types.

2.1 Intrinsic traits

Intrinsic traits are those which can be sensed that are a direct property of being a human occupant. Intrinsic traits are particularly useful, as in many situations they are guaranteed to be present if an occupant is present. However, they do have varying degrees of detectability and differentiation between occupants. Two main subcategories of these sensor types are static and dynamic traits.

2.1.1 Static traits

Static traits are physiologically derived, and are present with most (living) occupants. One key static trait that can be used for occupant sensing is that of thermal emissions. All human occupants emit distinctive thermal radiation in both resting and active states. The heat signatures of these emissions could potentially be measured with some apparatus, counted, and used to provide Presence and Count information to a sensor system, without providing Identity information.

Beltran, Erickson and Cerpa [7] propose Thermosense, a system that uses a type of thermal sensor known as an Infrared Array Sensor (IAR). This sensor is much like a camera, in that it has a field of view which is divided into “pixels”; in this case an 8×8 grid of detected temperatures. This sensor is mounted on an embedded device on the ceiling, along with a Passive Infrared Sensor (PIR), and uses a variety of classification algorithms to detect human heat signatures within the raw thermal and motion data it collects. Thermosense achieves Root Mean Squared Error ≈ 0.35 persons, meaning the standard deviation between Thermosense’s occupancy predictions and the actual occupancy number was \approx

0.35.

Another static trait is that of CO₂ emissions, which, like thermal emissions, are emitted by human occupants in both resting and active states. By measuring the buildup of CO₂ within a given area, one can use a variety of mathematical models of human CO₂ production to determine the likely number of occupants present. Hailemariam et al. [14] trialled this as part of a sensor fusion within the context of an office environment, achieving a $\approx 94\%$ accuracy. Such a sensing system could provide both the Presence and Count information, and exclude the Identity information as required. However, a CO₂ based detection mechanism has serious drawbacks, discussed by Fisk, Faulkner and Sullivan [10]: The CO₂ feedback mechanism is very slow, taking hours of continuous occupancy to correctly identify the presence of people. In a residential environment, occupants are more likely to be moving between rooms than an office, so the system may have a more difficult time detecting in that situation. Similarly, such systems can be interfered with by other elements that control the CO₂ buildup in a space, like air conditioners, open windows, etc. This is also much more of a concern in a residential environment compared to the studied office space, as the average residence can have numerous such confounding factors that cannot easily be controlled for.

Visual identification can be, achieved through the use of video or still-image cameras and advanced image processing algorithms. Video can be used in occupancy detection in several different ways, achieving different levels of accuracy and requiring different configurations. The first use of video, POEM, proposed by Erickson, Achleitner and Cerpa [9] is the use of video as a “optical turnstile”; the video system detects potential occupants and the direction they are moving in at each entrance and exit to an area, and uses that information to extrapolate the number of occupants within the turnstiled area; this system has up to a 94% accuracy. However, the main issue with such a system applied to a residential environment is the system assumes that there will be wide enough “turnstile areas”, corridors of a fairly large area that connect different sections of a building, to use as detection zones. While such corridors exist in office environments, they are less likely to exist in residential ones.

Another video sensor system is proposed by Serrano-Cuerda et al. [22], that uses ceiling-based cameras and advanced image processing algorithms to count the number of people in the captured area. This system achieves a specificity of $TP/(TP + FP) \approx 97\%$ and a sensitivity $TP/(TP + FN) \approx 96\%$ (TP = true positives, FP = false positives, FN = false negatives). Such a system could be successfully applied to the residential environment, as both it and the “optical turnstile” model provide Presence and Count information. However, these

systems also allow Identity to be determined, and thus are perceived as privacy-invasive. This perception leads to adoption and acceptance issues, which work against the ideal system’s goals.

2.1.2 Dynamic traits

Dynamic traits are usually products of human occupant activity, and thus can generally only be detected when a human occupant is physically active or in motion.

Ultrasonic systems, such as Doorjamb proposed by Hnat et al. [15], use clusters of such sensors above doorframes to detect the height and direction of potential occupants travelling between rooms. This acts as a turnstile based system, much like POEM [9], but augments this with an understanding of the model of the building to error correct for invalid and impossible movements brought about from sensing errors. This system provides an overall room-level tracking accuracy of 90%, however to achieve this accuracy, potential occupants are intended to be tracked using their heights, which has privacy implications. The system can also suffer from problems with error propagation, as there are possibilities of “phantom” occupants entering a room due to sensing errors.

Solely PIR based systems, like those used by Hailemariam et al. [14], involve the motion of the sensor being averaged over several different time intervals, and fed into a decision tree classifier. This PIR system alone produced a $\approx 98\%$ accuracy. However, such a system, due to only motion detection capabilities, can only provide Presence information, and is unable to provide Count information, nor detect motionless occupants.

2.2 Extrinsic traits

Extrinsic traits are those which are actually other environmental changes that are caused by or correlated with human occupant presence. These traits generally present a less accurate picture, or require the sensed occupants to be in some way “tagged”, but they are generally also easier to sense in of themselves. The sensors in this category have been divided into two subcategories.

2.2.1 Instrumented traits

One extrinsic trait category is instrumented approaches; these require that detectable occupants carry with them some device that is detected as a proxy for

the occupant themselves.

The most obvious of these approaches is a specially designed device. Li et al. [19] use RFID tags placed on building occupant’s persons and a set of transmitters to triangulate the tags and place them within different thermal zones for the use of the HVAC system. For stationary occupants, there was a detection accuracy of $\approx 88\%$, and for occupants who were mobile, the accuracy was $\approx 62\%$. Such a system could be re-purposed for the residence, however, these systems raise issues in a residential environment as it requires occupants to be constantly carrying their sensors, which is less likely in such an environment. Additionally, the accuracy for this system is not necessarily high enough for a residential environment, where much smaller rooms are used.

To make extrinsic detection more reliable, Li, Calis and Becerik-Gerber [16] leverage a common consumer device; wifi enabled smart phones. They propose the *homeset* algorithm, which uses the phones to scan the visible wifi networks, and from that information estimate if the occupants are at home or out and about by “triangulating” their position from the visible wifi networks. This solution does not provide the fine-grained Presence data that we need, as it is only able to triangulate the phone’s position very roughly with the wireless network detection information.

Balaji et al. [6] also leverage smart phones to determine occupancy, but in a more broad enterprise environment: Wireless association logs are analysed to determine which access points in a building a given occupant is connected to. If this access point falls within the radio range of their designated “personal space”, they are considered to be occupying that personal space. This technique cannot be applied to a residential environment, as there are usually not multiple wireless hotspots.

Finally, Gupta, Intille and Larson [13] use specifically the GPS functions of the smartphone to perform optimisation on heating and cooling systems by calculating the “travel-to-home” time of occupants at all times and ensuring at every distance the house is minimally heated such that if the potential occupant were to travel home, the house would be at the correct temperature when they arrived. While this system does achieve similar potential air-conditioning energy savings, it is not room-level modular, and also presupposes an occupant whose primary energy costs are from incorrect heating when away from home, which isn’t necessarily the case for this demographic.

2.2.2 Correlative traits

The second of these subcategories are correlative approaches. These approaches analyse data that is correlated with human occupant activity, but does not require a specific device to be present on each occupant that is tracked with the system.

The primary approach in this area is work done by Kleiminger et al. [17], which attempts to measure electricity consumption and use such data to determine Presence. Electricity data was measured at two different levels of granularity; the whole house level with a smart meter, and the consumption of specific appliances through smart plugs. This data was then processed by a variety of classifiers to achieve a classification accuracy of more than 80%. Such a system presents a low-cost solution to occupancy, however it is not sufficiently granular in either the detection of multiple occupants, or the detection of occupants in a specific room.

2.3 Analysis

From these various sensor options, there are a few candidates that provide the necessary quantitative criteria (Presence and Count); these are thermal, CO₂, Video, Ultrasonic, RFID and WiFi association and triangulation based methods. All sensing options are compared on Table 2.1 on the next page.

In the context of our four qualitative accessibility criteria, CO₂ sensing has several reliability drawbacks, the predominant ones being a large lag time to receive accurate occupancy information and interference from a variety of air conditioning sources which can modify the CO₂ concentration in the room in unexpected ways.

Video-based sensing methods suffer from invasiveness concerns, as they by design must have a constant video feed of all detected areas.

Ultrasonic methods suffer from reliability concerns when a user falls outside the prescribed height bounds of normal humans. Wheelchair bound occupants, a core demographic of our proposed sensing system, are not discussed in the Door-jamb paper. Their wheelchair may also interfere with height measurement results. Ultrasonic methods also provide weak Identity information through height detection.

RFID sensing also has several drawbacks; it is difficult value proposition to get residential occupants to carry RFID tags with them continuously. Another drawback is that the triangulation methods discussed are too unreliable to place occupants in specific rooms in many cases.

	Requires		Excludes	Irrelevant	
	Presence	Count	Identity	Location	Track
<u>Intrinsic</u>					
<i>Static</i>					
Thermal	✓	✓	✓	✓	
CO ₂	✓	✓	✓		
Video	✓	✓	✗	✓	✓
<i>Dynamic</i>					
Ultrasonic	✓	✓	✗		✓
PIR	✓	✗	✓		
<u>Extrinsic</u>					
<i>Instrumented</i>					
RFID	✓ ¹	✓	✓	✓	
WiFi assoc. ²	✓ ¹	✓	✗	✓	
WiFi triang. ²	✓ ¹	✓	✗		
GPS ²	✓ ¹	✗	✓	✓	
<i>Correlative</i>					
Electricity	✓ ¹	✗	✓		

¹Doesn't provide data at required level of accuracy for home use.

²Uses smartphone as detector.

Table 2.1: Comparison of different sensors and project requirements

WiFi association is not granular enough for residential use, as the original enterprise use case presupposed a much larger area, as well as multiple wireless access points, neither of which a typical residential environment have.

WiFi triangulation is a good candidate for residential use, as there are most likely neighbouring wireless networks that can be used as virtual landmarks. However, it suffers from the same granularity problems as WiFi association, as these signals are not specific enough to pinpoint an occupant to a specific room.

For approaches presupposing smartphones being present on each occupant, it is more difficult to ensure that occupants are carrying their smartphones with them at all times in a residential environment. Another issue with smart phones is that they represent an expense that the target markets of the elderly and the disabled may not be able to afford.

Finally, we have thermal sensing. It provides both Presence and Count information, as it uses occupants' thermal signatures to determine the presence of people in a room. It does not however provide Identity information, as thermal signatures are not sufficiently unique with the technologies used to distinguish between occupants. Such a sensor system is presented as low-cost and energy efficient within Thermosense [7], is non-invasive by design and can reliably detect occupants with a very low root mean squared error. For our specific accessibility criteria, thermal sensing appears to be the best option available.

2.4 Thermal sensors

Our analysis (Subsection 2.3 on page 8) concluded that thermal sensors are the best candidates for this project. In this section we discuss the thermal sensing field in more detail.

A primary static/dynamic sensor fusion system in this field is the Thermosense system [7], a Passive Infrared Sensor (PIR) and Infrared Array Sensor (IAR)¹ used to subdivide an area into an 8×8 grid of sections from which temperatures can be derived. This sensor system is attached to the roof on a small embedded controller which is responsible for collecting the data and transmitting it back to a larger computer via low powered wireless protocols.

The Thermosense system develops a thermal background map of the room using an Exponential Weighted Moving Average (EMWA) over a 15 minute time window (if no motion is detected). If the room remains occupied for a long period, a more complex scaling algorithm is used which considers the coldest points in

¹Phillips GridEYE; approx \$30

the room empty, and averages them against the new background, then performs EMWA with a lower weighting.

This background map is used as a baseline to calculate standard deviations of each grid area, which are then used to determine several characteristics to be used as feature vectors for a variety of classification approaches. The determination of the feature vectors was subject to experimentation, since the differences at each grid element too susceptible to individual room conditions to be used as feature vectors. Instead, a set of three different features was designed; the number of temperature anomalies in the space, the number of groups of temperature anomalies, and the size of the largest anomaly in the space. These feature vectors were compared against three classification approaches; K-Nearest Neighbors, Linear Regression and an a feed-forward Artificial Neural Network of one hidden layer and 5 perceptions. All three classifiers achieved a Root Mean Squared Error (RMSE) within 0.38 ± 0.04 . This final classification is subject to a final averaging process over a 4 minute window to remove the presence of independent errors from the raw classification data.

The Thermosense approach presents the state of the art in the field of sensing with IAR technology. Using a similar IAR system along with those types of classification algorithms should yield useful sensing results which can be then integrated into the broader sensor system.

2.5 Research Gap

Throughout this review of the area of wireless occupancy sensors within the Internet of Things (IoT) it can be seen that there is a clear research gap within the area of occupancy. No group could be found who has assembled an occupancy sensor that optimises these areas of Low Cost, Non-Invasiveness, Energy Efficiency and Reliability into a architected software and hardware package that can be integrated like any other Thing into the IoT.

This is a key research area, because, as we have previously mentioned, the true “disruptive level of innovation” [4] the IoT provides can only be realised once a novel idea has been properly packaged as a Thing, rather than as a research curiosity. Packaging something as a Thing requires careful consideration of the best sensing systems, the best hardware to run those systems on, the best protocols to allow these Things to communicate, and the best device architecture to enable that communication. The state of the art in all these areas have been discussed throughout this literature review.

2.6 Conclusion

Several criteria were identified through which the spectrum of occupancy sensing could be examined; a quantitative criteria by Teixeira, Dublon and Savvides [24] to examine the different functionality offerings of sensor systems and a qualitative criteria derived from the aims of the project to examine how those sensors fit within the project's parameters.

Occupancy research performed with different sensor types was examined methodically through a set of taxonomic categories also originally proposed by Teixeira, Dublon and Savvides [24], but modified to better suit the specifics of occupancy sensors. These sensor types included Thermal, CO₂, Video, Ultrasonic, Passive Infrared Sensor (PIR), RFID, various WiFi based methods, GPS and electricity consumption. Through an examination of these sensing systems quantitative and qualitative characteristics, it was determined that the Thermosense Infrared Array Sensor (IAR) system [7] was the most suitable to the project's aims.

A key part of enabling the "smart home for the disabled" is creating a set of Things that can improve quality of life for those people. We believe our proposed Thing has clearly demonstrated this potential.

CHAPTER 3

Architecture

Since the advent of a standardised Internet of Things (IoT) protocol stack discussed in Section 3.1, 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 3.1 on the following page will form the stack used by the “WPAN” network shown in Figure 3.1 on page 16.

Moving from a protocol perspective to a device perspective, when one considers the energy efficiency and cost constraints 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.

3.1 Ideal System Architecture

Beyond specific sensor design and occupancy detection algorithms, a core goal of this project is to create a system that is designed to operate as a useful Thing

REST	
Application	CoAP
Transport	UDP
IP / Routing	IETF RPL
Adaptation	IETF 6LoWPAN
Medium Access	IEEE 802.15.4e
Physical	IEEE 802.15.4-2006

Table 3.1: Proposed protocol stack

in a real-world Internet of Things (IoT) environment, as the key advantage of Things is the “disruptive level of innovation”[4] brought about by their ability to be combined in ways unforeseen (yet still enabled) by their creators. This architecture involves careful consideration of the embedded hardware that will drive the system, as well as the communications protocols utilised between the sensor and devices interested in the sensor’s information.

3.1.1 Protocols

In an ideal smart-home environment, the sensor systems used will communicate with each other wirelessly. As the complete sensor system has low power requirements to enable battery operation, it is important to prioritise those protocols and architectures that minimise power usage while still enabling the necessary wireless communication. The system will also ideally exist in a system with other identical sensors (one for each room in a residence), thus it is important to prioritise those protocols which allow multiple identical sensor systems to coexist on the same network without conflict, and to be uniquely addressable and identifiable. In recent years, many developments have been made in the IoT arena, with standards emerging specifically designed for low-power embedded devices to communicate between themselves and bigger systems that address these and other unique needs, across the entire protocol stack.

Palattella et al. [21] propose a protocol stack that aligns with the above requirements, with the key advantage being a wholly standardized implementation of the stack exists. This implementation is based on TCP/IP, uses the latest IEEE and IETF IoT standards, and is free from proprietary protocol restrictions (unlike ZigBee 1.0 devices, for instance). Table 3.1 shows the full stack proposed. The key components of this proposal are the introduction of CoAP at the application layer, RPL at the IP / Routing layer and 6LoWPAN at the Adaptation layer.

Above the application layer, Guinard et al. [11] propose the use of Representational state transfer (REST) over Web Services Descriptive Language / Simple Object Access Protocol (WS-*) as a method of exchanging information between sensor systems. Their data suggests that REST is easier to use than WS-*, and the key advantage of a WS-* based approach is its ability to represent much more complex data and abstractions, which are unnecessary in this project’s situation.

Constrained Application Protocol (CoAP) [18] is an application layer protocol designed to replace HTTP as a way of transmitting RESTful information between clients. The chief advantage of CoAP over HTTP is it compresses the broad-strokes of the HTTP feature set into a binary language that is much more suitable for transmission over low-bandwidth and low-power links, such as those discussed here.

IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) [25] is a routing protocol designed for low power environments, allowing low power nodes to create and maintain a mesh network between themselves, allowing, among other things, the routing of packets to a “root” node and back again. RPL is particularly suited to the routing situation of our proposed architecture, as individual sensors do not need to communicate with one another, but rather report back to a larger node (further discussed in Subsection 3.1.2).

IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) [23] is a compression and formatting specification to allow IPv6 packets to be sent over an 802.15.4 based network. Optimisations are found in the reduction of the size of 6LoWPAN packets, IPv6 addresses as well as redesigning core Internet Protocol algorithms so that they can run with low power consumption on participating devices.

3.1.2 Devices

In addition to the protocol stack used, how these nodes relate to each other is also an important consideration. Part of what will inform these decisions are the requisite processing power and internet connectivity required to successfully execute all elements of the sensing system. Kovatsch [18] provides a constructive classification system to consider this, by describing three classes of resource constrained devices that would benefit from CoAP, and each can provide different levels of security for an IP stack;

- *Class 0*: “not capable of running an RFC-compliant IP stack in a secure manner. They require application-level gateways to connect to the Internet.”

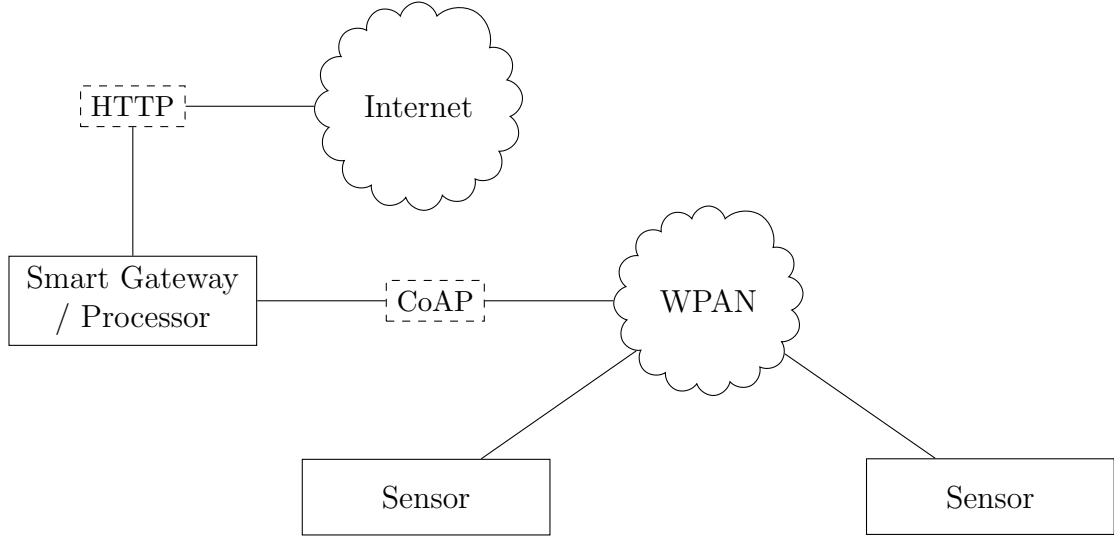


Figure 3.1: Proposed system architecture

- *Class 1*: Able to connect to the internet with some “integrated security mechanisms”. Are unable to employ full HTTP with TLS.
- *Class 2*: Normal Internet nodes, able to use the full HTTP stack with TLS.

The devices that we propose the sensors will connect to are the likes of the Arduino, which can be classified as class 0 or possibly class 1 devices. Due to their insecurity and difficulty running a fully fledged IP stack, Guinard et al. [12] propose the use of a “Smart Gateway” system to bridge the wider internet and these sensor systems. This gateway would be able to communicate with the sensor systems over CoAP and 802.15.4, as well as receive API requests via HTTP from a traditional TCP/IP network to forward on to these sensors.

The Thermosense paper [7] proposes several different algorithms to process the raw sensing data into the occupancy estimates (further discussed in Section 2.4 on page 10), all of which are fairly computationally expensive. Because of this, it would be non-trivial to implement these algorithms on the embedded sensing devices themselves. This problem is already resolved in our proposed system, as the aforementioned “Smart Gateway” can easily also take on the task of processing the raw sensor data into estimates which it can relay to interested parties over its HTTP-based API. A visualisation of this proposed system is shown in Figure 3.1.

3.2 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.

3.2.1 Hardware

Due to low cost and ease of use, the Arduino platform was selected as the host for the low-level I²C 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 I²C 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 I²C 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 3.2 on the following 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.

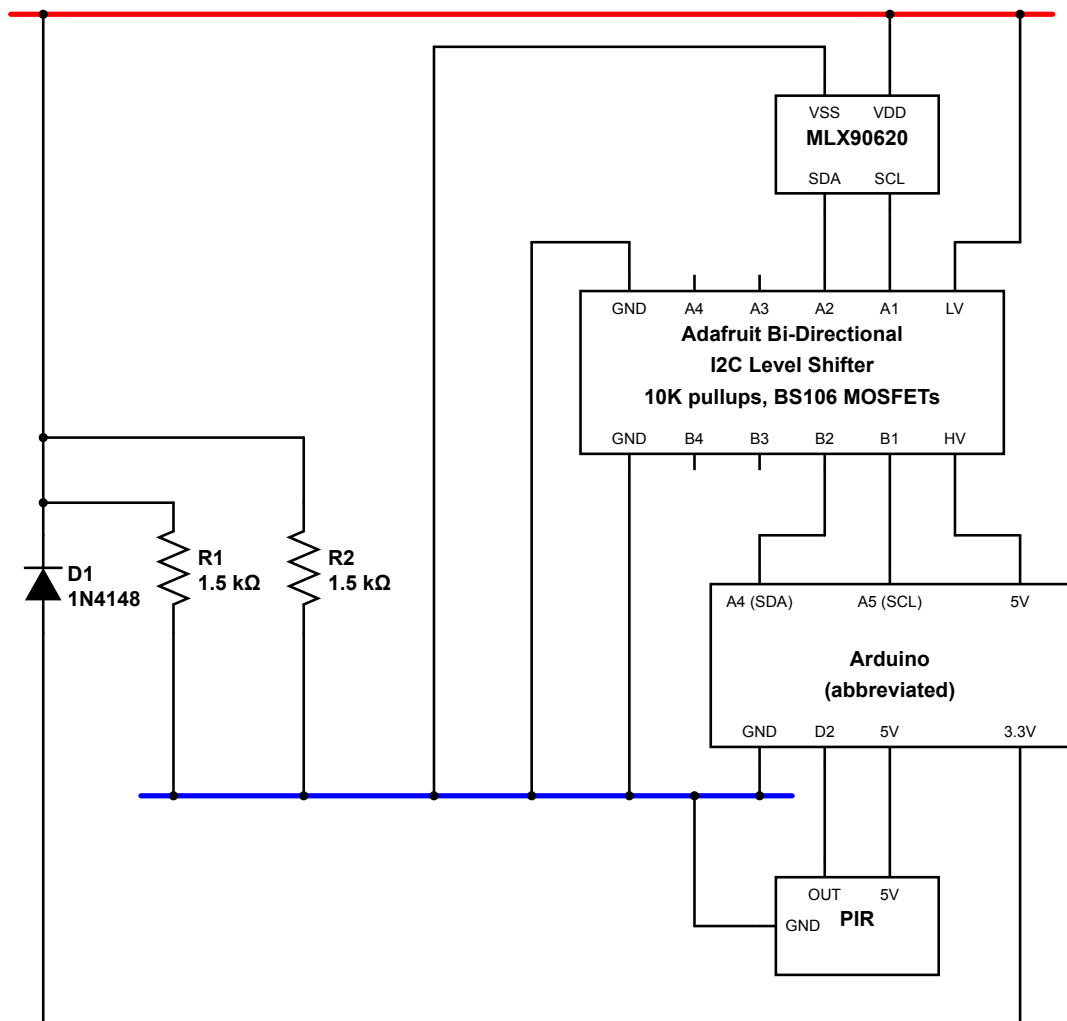


Figure 3.2: MLX90620, PIR and Arduino integration circuit

3.2.2 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 [20]. This process involves initialising the sensor with values attained from a separate on-board I²C EEPROM, then retrieving a variety of 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” I²C 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 3.3 on the next page 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 3.2 on the following 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 3.4 on the next 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.

```

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

```

Figure 3.3: Initialisation sequence

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
Fx	Set sensor refresh frequency to x and reboot

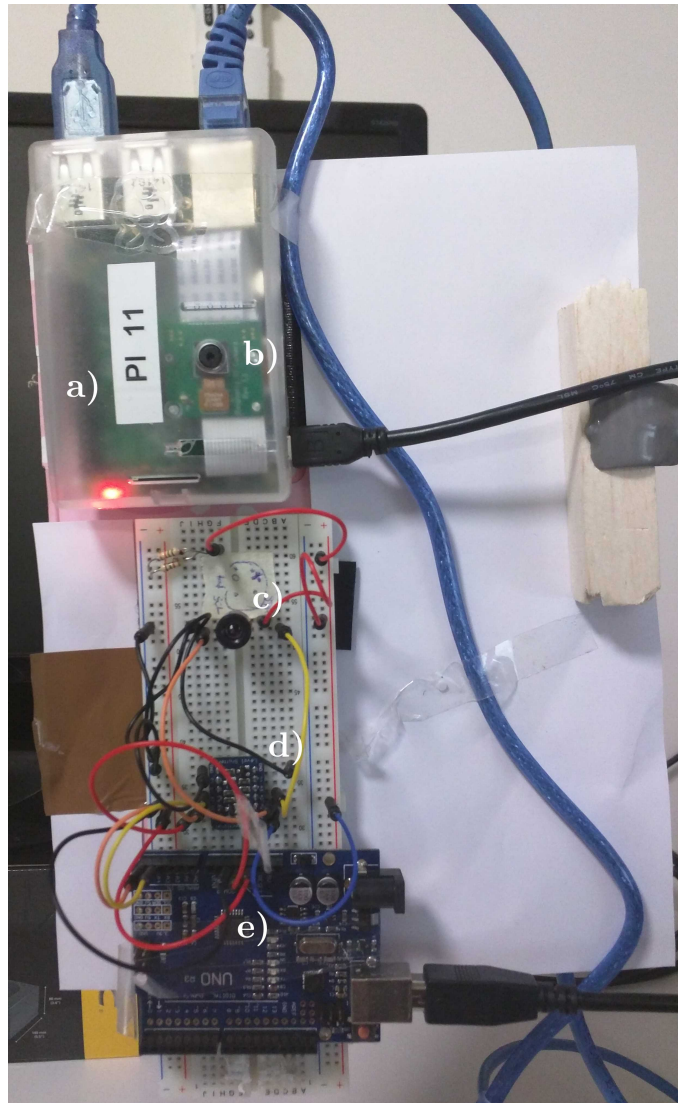
Table 3.2: Commands

```

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

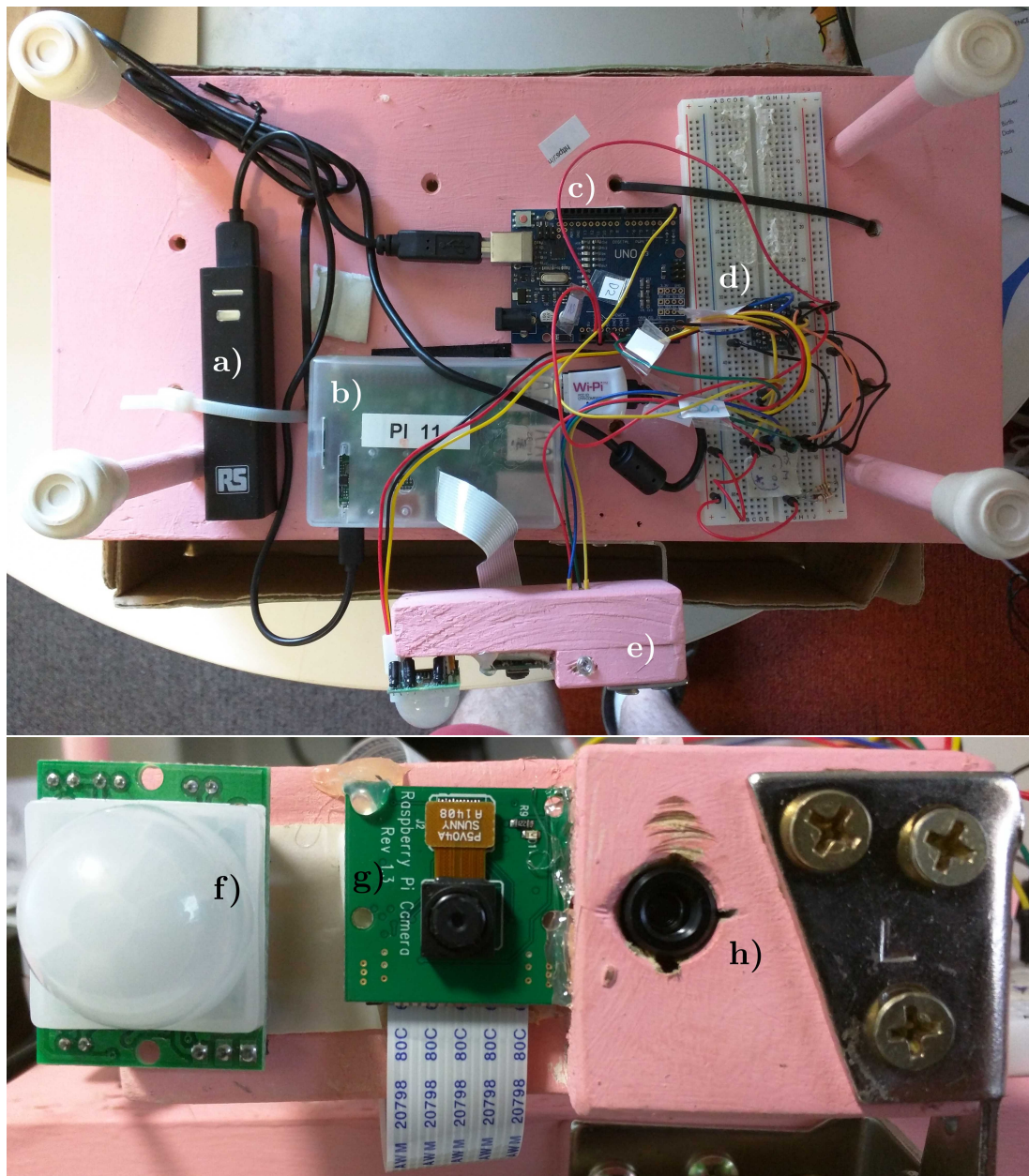
```

Figure 3.4: Thermal data packet



- a) Raspberry Pi
- b) Camera
- c) *Melexis*
- d) Level-shifting circuitry
- e) Arduino

Figure 3.5: Prototype A



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

Figure 3.6: Prototype B

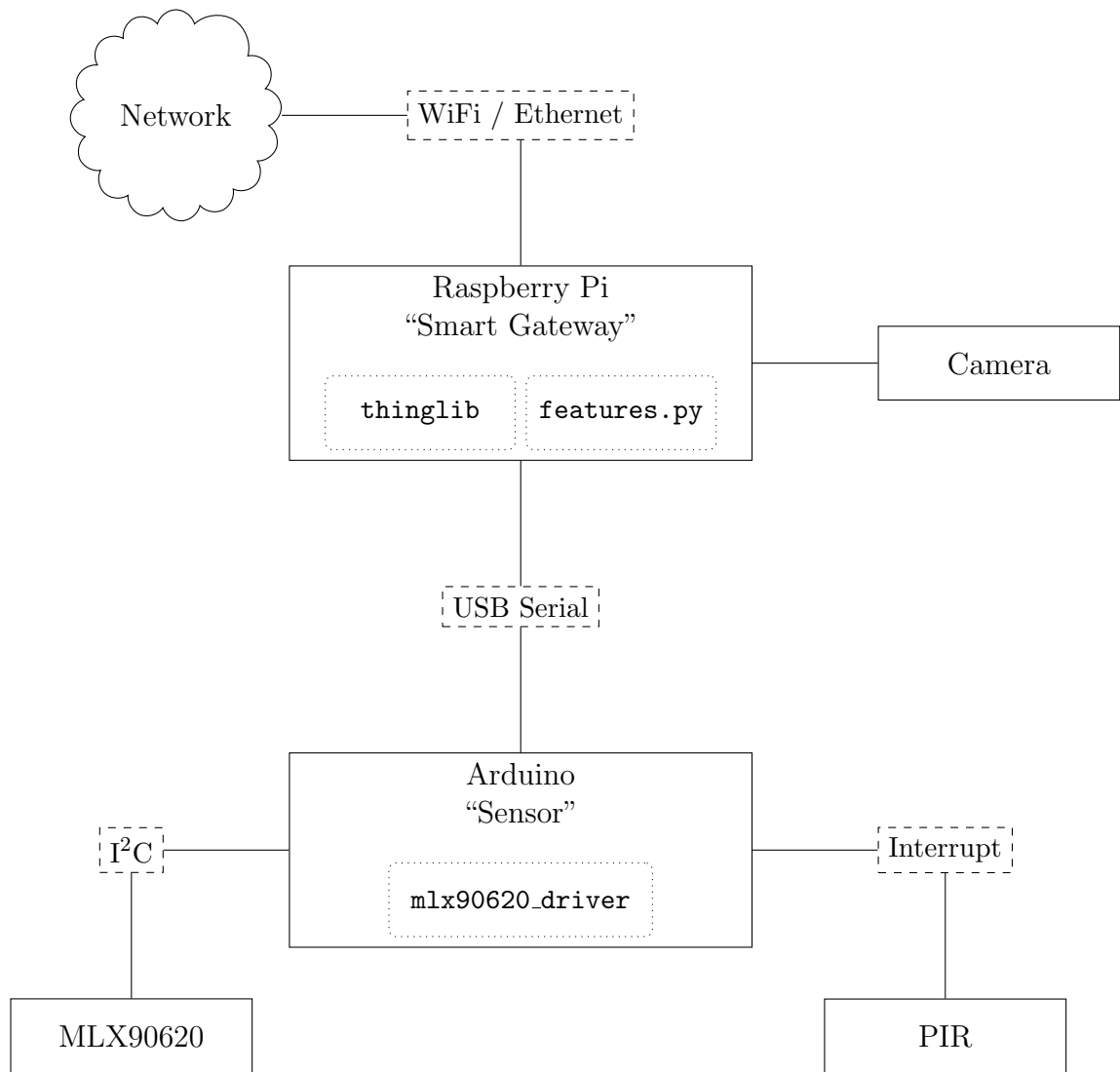


Figure 3.7: Prototype B system architecture

CHAPTER 4

Sensor Properties

In Table 4.1 on the following page the thermal sensor was exposed to the night sky at a capture rate of 1Hz for 4 minutes, with the sensing results combined to create a set of means and standard deviations to indicate the pixels at “rest”.

14.95 0.51	14.33 0.27	12.34 0.27	8.77 0.33	8.15 0.31	10.84 0.38	9.02 0.26	7.79 0.37	6.67 0.27	9.63 0.29	9.29 0.26	8.24 0.27	9.84 0.25	14.28 0.33	14.92 0.3	13.16 0.25
14.54 0.34	15.62 0.31	12.73 0.23	11.51 0.27	11.79 0.26	11.47 0.27	11.43 0.29	9.02 0.35	8.57 0.23	11.15 0.23	10.64 0.22	10.3 0.24	12.09 0.22	14.49 0.26	14.88 0.31	14.71 0.36
18.25 0.45	16.62 0.31	14.15 0.24	11.97 0.34	13.11 0.3	12.64 0.22	10.66 0.23	9.15 0.24	9.58 0.28	11.95 0.28	11.22 0.24	11.52 0.36	11.11 0.23	12.59 0.25	14.44 0.31	13.35 0.28
16.02 0.28	16.81 0.36	15.0 0.25	11.53 0.28	10.18 0.29	12.2 0.25	11.78 0.29	8.36 0.31	8.15 0.33	10.36 0.32	10.74 0.31	8.25 0.36	9.99 0.35	12.42 0.38	11.39 0.4	11.06 0.34

Table 4.1: Mean and standard deviations for each pixel at rest

CHAPTER 5

Methods

CHAPTER 6

Results

CHAPTER 7

Discussion and Conclusion

7.1 Future Directions

- Wireless mesh networking
- Convert into circuit board
- MLX90621
- Lenses
- Rotating the sensor to see wider FOV

APPENDIX A

Original Honours Proposal

Title: Developing a robust system for occupancy detection in the household
Author: Ash Tyndall
Supervisor: Professor Rachel Cardell-Oliver
Degree: BCompSci (24 point project)
Date: October 8, 2014

A.1 Background

The proportion of elderly and mobility-impaired people is predicted to grow dramatically over the next century, leaving a large proportion of the population unable to care for themselves, and consequently less people able care for these groups. [6] With this issue looming, investments are being made into a variety of technologies that can provide the support these groups need to live independent of human assistance.

With recent advancements in low cost embedded computing, such as the Arduino [1] and Raspberry Pi, [2] the ability to provide a set of interconnected sensors, actuators and interfaces to enable a low-cost ‘smart home for the disabled’ is becoming increasingly achievable.

Sensing techniques to determine occupancy, the detection of the presence and number of people in an area, are of particular use to the elderly and disabled. Detection can be used to inform various devices that change state depending on the user’s location, including the better regulation energy hungry devices to help reduce financial burden. Household climate control, which in some regions of Australia accounts for up to 40% of energy usage [3] is one particular area

in which occupancy detection can reduce costs, as efficiency can be increased dramatically with annual energy savings of up to 25% found in some cases. [8]

Significant research has been performed into the occupancy field, with a focus on improving the energy efficiency of both office buildings and households. This is achieved through a variety of sensing means, including thermal arrays, [5] ultrasonic sensors, [11] smart phone tracking, [12][4] electricity consumption, [13] network traffic analysis, [15] sound, [10] CO₂, [10] passive infrared, [10] video cameras, [7] and various fusions of the above. [16][15]

A.2 Aim

While many of the above solutions achieve excellent accuracies, in many cases they suffer from problems of installation logistics, difficult assembly, assumptions on user's technology ownership and component cost. In a smart home for the disabled, accuracy is important, but accessibility is paramount.

The goal of this research project is to devise an occupancy detection system that forms part of a larger 'smart home for the disabled' that meets the following accessibility criteria;

- *Low Cost*: The set of components required should aim to minimise cost, as these devices are intended to be deployed in situations where the serviced user may be financially restricted.
- *Non-Invasive*: The sensors used in the system should gather as little information as necessary to achieve the detection goal; there are privacy concerns with the use of high-definition sensors.
- *Energy Efficient*: The system may be placed in a location where there is no access to mains power (i.e. roof), and the retrofitting of appropriate power can be difficult; the ability to survive for long periods on only battery power is advantageous.
- *Reliable*: The system should be able to operate without user intervention or frequent maintenance, and should be able to perform its occupancy detection goal with a high degree of accuracy.

Success in this project would involve both

1. Devising a bill of materials that can be purchased off-the-shelf, assembled without difficulty, on which a software platform can be installed that performs analysis of the sensor data and provides a simple answer to the occupancy question, and
2. Using those materials and softwares to create a final demonstration prototype whose success can be tested in controlled and real-world conditions.

This system would be extensible, based on open standards such as REST or CoAP, [9][14] and could easily fit into a larger ‘smart home for the disabled’ or internet-of-things system.

A.3 Method

Achieving these aims involves performing research and development in several discrete phases.

A.3.1 Hardware

A list of possible sensor candidates will be developed, and these candidates will be ranked according to their adherence to the four accessibility criteria outlined above. Primarily the sensor ranking will consider the cost, invasiveness and reliability of detection, as the sensors themselves do not form a large part of the power requirement.

Similarly, a list of possible embedded boards to act as the sensor’s host and data analysis platform will be created. Primarily, they will be ranked on cost, energy efficiency and reliability of programming/system stability.

Low-powered wireless protocols will also be investigated, to determine which is most suitable for the device; providing enough range at low power consumption to allow easy and reliable communication with the hardware.

Once promising candidates have been identified, components will be purchased and analysed to determine how well they can integrate.

A.3.2 Classification

Depending on the final sensor choice, relevant experiments will be performed to determine the classification algorithm with the best occupancy determina-

tion accuracy. This will involve the deployment of a prototype to perform data gathering, as well as another device/person to assess ground truth.

A.3.3 Robustness / API

Once the classification algorithm and hardware are finalised, an easy to use API will be developed to allow the data the device collects to be integrated into a broader system.

The finalised product will be architected into a easy-to-install software solution that will allow someone without domain knowledge to use the software and corresponding hardware in their own environment.

A.4 Timeline

Date	Task
Fri 15 August	<i>Project proposal and project summary due to Coordinator</i>
August	Hardware shortlisting / testing
25–29 August	<i>Project proposal talk presented to research group</i>
September	Literature review
Fri 19 September	<i>Draft literature review due to supervisor(s)</i>
October - November	Core Hardware / Software development
Fri 24 October	<i>Literature Review and Revised Project Proposal due to Coordinator</i>
November - February	<i>End of year break</i>
February	Write dissertation
Thu 16 April	<i>Draft dissertation due to supervisor</i>
April - May	Improve robustness and API
Thu 30 April	<i>Draft dissertation available for collection from supervisor</i>
Fri 8 May	<i>Seminar title and abstract due to Coordinator</i>
Mon 25 May	<i>Final dissertation due to Coordinator</i>
25–29 May	<i>Seminar Presented to Seminar Marking Panel</i>
Thu 28 May	<i>Poster Due</i>
Mon 22 June	<i>Corrected Dissertation Due to Coordinator</i>

A.5 Software and Hardware Requirements

A large part of this research project is determining the specific hardware and software that best fit the accessibility criteria. Because of this, an exhaustive list of software and hardware requirements are not given in this proposal.

A budget of up to \$300 has been allocated by my supervisor for project purchases. Some technologies with promise that will be investigated include;

Raspberry Pi Model B+ Small form-factor Linux computer

Available from <http://arduino.cc/en/Guide/Introduction>; \$38

Arduino Uno Small form-factor microcontroller

Available from <http://arduino.cc/en/Main/arduinoBoardUno>; \$36

Panasonic Grid-EYE Infrared Array Sensor

Available from <http://www3.panasonic.biz/ac/e/control/sensor/infrared/grid-eye/index.jsp>; approx. \$33

Passive Infrared Sensor

Available from various places; \$10–\$20

A.6 Proposal References

- [1] Arduino. <http://arduino.cc/en/Guide/Introduction>. Accessed: 2014-08-09.
- [2] Raspberry pi. <http://www.raspberrypi.org/>. Accessed: 2014-08-09.
- [3] AUSTRALIAN BUREAU OF STATISTICS. 4602.2 - household water and energy use, victoria: Heating and cooling. Tech. rep., October 2011.
- [4] BALAJI, B., XU, J., NWOKAFOR, A., GUPTA, R., AND AGARWAL, Y. Sentinel: occupancy based hvac actuation using existing wifi infrastructure within commercial buildings. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems* (2013), ACM, p. 17.
- [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] CHAN, M., CAMPO, E., ESTÈVE, D., AND FOURNIOLS, J.-Y. Smart homescurrent features and future perspectives. *Maturitas* 64, 2 (2009), 90–97.
- [7] ERICKSON, V. L., ACHLEITNER, S., AND CERPA, A. E. Poem: Power-efficient occupancy-based energy management system. In *Proceedings of the 12th international conference on Information processing in sensor networks* (2013), ACM, pp. 203–216.
- [8] ERICKSON, V. L., BELTRAN, A., WINKLER, D. A., ESFAHANI, N. P., LUSBY, J. R., AND CERPA, A. E. Thermosense: thermal array sensor networks in building management. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems* (2013), ACM, p. 87.
- [9] 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.

- [10] HAILEMARIAM, E., GOLDSTEIN, R., ATTAR, R., AND KHAN, A. Real-time occupancy detection using decision trees with multiple sensor types. In *Proceedings of the 2011 Symposium on Simulation for Architecture and Urban Design* (2011), Society for Computer Simulation International, pp. 141–148.
- [11] HNAT, T. W., GRIFFITHS, E., DAWSON, R., AND WHITEHOUSE, K. Doorjamb: unobtrusive room-level tracking of people in homes using doorway sensors. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems* (2012), ACM, pp. 309–322.
- [12] KLEIMINGER, W., BECKEL, C., DEY, A., AND SANTINI, S. Using unlabeled wi-fi scan data to discover occupancy patterns of private households. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems* (2013), ACM, p. 47.
- [13] KLEIMINGER, W., BECKEL, C., STAAKE, T., AND SANTINI, S. Occupancy detection from electricity consumption data. In *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings* (2013), ACM, pp. 1–8.
- [14] 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.
- [15] TING, K., YU, R., AND SRIVASTAVA, M. Occupancy inferencing from non-intrusive data sources. In *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings* (2013), ACM, pp. 1–2.
- [16] YANG, Z., LI, N., BECERIK-GERBER, B., AND OROSZ, M. A multi-sensor based occupancy estimation model for supporting demand driven hvac operations. In *Proceedings of the 2012 Symposium on Simulation for Architecture and Urban Design* (2012), Society for Computer Simulation International, p. 2.

APPENDIX B

Code Listings

B.1 ThingLib

B.1.1 cam.py

```
1  from __future__ import division
2  from __future__ import print_function
3
4  import serial
5  import copy
6  import Queue as queue
7  import time
8  from collections import deque
9  import threading
10 import pygame
11 import colorsys
12 import datetime
13 from PIL import Image, ImageDraw, ImageFont
14 import subprocess
15 import tempfile
16 import os
17 import os.path
18 import fractions
19 import pxdisplay
20 import multiprocessing
21 import numpy as np
22 import io
23
24
25
26 class BaseManager(object):
27     driver = None
28     build = None
29     irhz = None
30
31     tty = None
```

```

32     baud = None
33
34     hflip = True
35     vflip = True
36
37     _temps = None
38     _serial_obj = None
39     _queues = []
40
41     def __init__(self, tty, hz=8, baud=115200):
42         self.tty = tty
43         self.baud = baud
44         self.irhz = hz
45
46         self._serial_obj = serial.Serial(port=self.tty, baudrate=self.baud,
47             ↪ rtscts=True, dsrdtr=True)
48
49     def __del__(self):
50         self.close()
51
52     def _reset_and_conf(self, timers=True):
53         self._serial_obj.write('r\n') # Reset the sensor
54         self._serial_obj.flush()
55
56         time.sleep(2)
57
58         if timers:
59             self._serial_obj.write('t\n') # Turn on timers
60         else:
61             self._serial_obj.write('o\n') # Turn on timers
62
63         self._serial_obj.flush()
64
65     def _decode_packet(self, packet):
66         decoded_packet = {}
67         ir = []
68
69         for line in packet:
70             parted = line.partition(" ")
71             cmd = parted[0]
72             val = parted[2]
73
74             try:
75                 if cmd == "START":
76                     decoded_packet['start_millis'] = long(val)
77                 elif cmd == "STOP":
78                     decoded_packet['stop_millis'] = long(val)
79                 elif cmd == "MOVEMENT":
80                     if val == "0":

```

```

80         decoded_packet['movement'] = False
81     elif val == "1":
82         decoded_packet['movement'] = True
83     else:
84         ir.append(tuple(float(x) for x in line.split("\t")))
85     except ValueError:
86         print(packet)
87         print("WARNING: Could not decode corrupted packet")
88         return {}
89
90     if self.hflip:
91         ir = map(tuple, np.fliplr(ir))
92
93     if self.vflip:
94         ir = map(tuple, np.flipud(ir))
95
96     decoded_packet['ir'] = tuple(ir)
97
98     return decoded_packet
99
100 def _decode_info(self, packet):
101     decoded_packet = {}
102     ir = []
103
104     for line in packet:
105         parted = line.partition(" ")
106         cmd = parted[0]
107         val = parted[2]
108
109         if cmd == "INFO":
110             pass
111         elif cmd == "DRIVER":
112             decoded_packet['driver'] = val
113         elif cmd == "BUILD":
114             decoded_packet['build'] = val
115         elif cmd == "IRHZ":
116             decoded_packet['irhz'] = int(val) if int(val) != 0 else 0.5
117
118     return decoded_packet
119
120 def _update_info(self):
121     ser = self._serial_obj
122
123     ser.write('i')
124     ser.flush()
125     imsg = []
126
127     line = ser.readline().decode("ascii", "ignore").strip()
128

```

```

129     # Capture a whole packet
130     while not line == "INFO START":
131         line = ser.readline().decode("ascii", "ignore").strip()
132
133     while not line == "INFO STOP":
134         imsg.append(line)
135         line = ser.readline().decode("ascii", "ignore").strip()
136
137     imsg.append(line)
138
139     packet = self._decode_info(imsg)
140
141     self.driver = packet['driver']
142     self.build = packet['build']
143
144     if packet['irhz'] != self.irhz:
145         ser.write('f{}'.format(self.irhz))
146         self._update_info()
147
148     def _wait_read_packet(self):
149         ser = self._serial_obj
150         line = ser.readline().decode("ascii", "ignore").strip()
151         msg = []
152
153         # Capture a whole packet
154         while not line.startswith("START"):
155             line = ser.readline().decode("ascii", "ignore").strip()
156
157         while not line.startswith("STOP"):
158             msg.append(line)
159             line = ser.readline().decode("ascii", "ignore").strip()
160
161         msg.append(line)
162
163         return msg
164
165     def close(self):
166         return
167
168     def get_temps(self):
169         if self._temps is None:
170             return False
171         else:
172             return copy.deepcopy(self._temps)
173
174     def subscribe(self):
175         q = queue.Queue()
176         self._queues.append(q)
177         return q

```

```

178
179     def subscribe_multiprocess(self):
180         q = multiprocessing.Queue()
181         self._queues.append(q)
182         return q
183
184     def subscribe_lifo(self):
185         q = queue.LifoQueue()
186         self._queues.append(q)
187         return q
188
189
190
191     class Manager(BaseManager):
192         _serial_thread = None
193         _serial_stop = False
194         _serial_ready = False
195
196         _decode_thread = None
197
198         _read_decode_queue = None
199
200     def __init__(self, tty, hz=8, baud=115200):
201         super(self.__class__, self).__init__(tty, hz, baud)
202
203         self._serial_thread = threading.Thread(group=None,
204             ↪ target=self._read_thread_run)
205         self._serial_thread.daemon = True
206
207         self._decode_thread = threading.Thread(group=None,
208             ↪ target=self._decode_thread_run)
209         self._decode_thread.daemon = True
210
211         self._reset_and_conf(timers=True)
212
213         self._read_decode_queue = queue.Queue()
214
215         self._decode_thread.start()
216         self._serial_thread.start()
217
218         while not self._serial_ready: # Wait until we've populated data before
219             ↪ continuing
220             pass
221
222     def close(self):
223         self._serial_stop = True
224
225         if self._serial_thread is not None:
226             while self._serial_thread.is_alive(): # Wait for thread to terminate

```



```

224         pass
225
226     def _read_thread_run(self):
227         ser = self._serial_obj
228         q = self._read_decode_queue
229         self._update_info()
230
231         while True:
232             msg = self._wait_read_packet()
233
234             q.put(msg)
235             self._serial_ready = True
236
237             if self._serial_stop:
238                 ser.close()
239                 return
240
241     def _decode_thread_run(self):
242         dq = self._read_decode_queue
243         while True:
244             msg = dq.get(block=True)
245
246             dpct = self._decode_packet(msg)
247
248             if 'ir' in dpct:
249                 self._temps = dpct
250
251             for q in self._queues:
252                 q.put(self.get_temps())
253
254             if self._serial_stop:
255                 return
256
257
258     class OnDemandManager(BaseManager):
259         def __init__(self, tty, hz=8, baud=115200):
260             super(self.__class__, self).__init__(tty, hz, baud)
261
262             self._reset_and_conf(timers=False)
263
264             self._update_info()
265
266         def close(self):
267             self._serial_obj.close()
268
269         def capture(self):
270             self._serial_obj.write('p') # Capture frame manually
271             self._serial_obj.flush()
272

```

```

273     msg = self._wait_read_packet()
274     dpct = self._decode_packet(msg)
275
276     if 'ir' in dpct:
277         self._temps = dpct
278
279         for q in self._queues:
280             q.put(self.get_temps())
281
282     return dpct
283
284
285
286 class ManagerPlaybackEmulator(BaseManager):
287     _playback_data = None
288
289     _pb_thread = None
290     _pb_stop = False
291     _pb_len = 0
292
293     _i = 0
294
295     def __init__(self, playback_data=None):
296         if playback_data is not None:
297             self.irhz, self._playback_data = playback_data
298             self._pb_len = len(self._playback_data)
299
300             self.driver = "Playback"
301             self.build = "1"
302
303     def set_playback_data(self, playback_data):
304         self.stop()
305         self.irhz, self._playback_data = playback_data
306         self._pb_len = len(self._playback_data)
307
308     def close(self):
309         return
310
311     def start(self):
312         if self._pb_thread is None:
313             self._pb_stop = False
314             self._pb_thread = threading.Thread(group=None,
315                 ↪ target=self._pb_thread_run)
316             self._pb_thread.daemon = True
317             self._pb_thread.start()
318
319     def pause(self):
320         self._pb_stop = True

```

```

321     while self._pb_thread is not None and self._pb_thread.is_alive():
322         pass
323
324     self._pb_thread = None
325
326     def stop(self):
327         self._pb_stop = True
328
329         while self._pb_thread is not None and self._pb_thread.is_alive():
330             pass
331
332         self._pb_thread = None
333         self._i = 0
334
335     def get_temps(self):
336         return self._playback_data[self._i]
337
338     def _pb_thread_run(self):
339         while True:
340             if self._pb_stop:
341                 return
342
343             for q in self._queues:
344                 q.put(self._playback_data[self._i])
345
346             time.sleep(1.0/float(self.irhz))
347
348             self._i += 1
349
350             if self._i >= self._pb_len:
351                 return
352
353
354
355     class Visualizer(object):
356         _display_thread = None
357         _display_stop = False
358         _tmin = None
359         _tmax = None
360         _limit = None
361         _dwidth = None
362
363         _tcam = None
364         _ffmpeg_loc = None
365
366         _camera = None
367
368     def __init__(self, tcam=None, camera=None, ffmpeg_loc="ffmpeg"):
369         self._tcam = tcam

```

```

370     self._ffmpeg_loc = ffmpeg_loc
371     self._camera = camera
372
373     def display(self, block=False, limit=0, width=100, tmin=15, tmax=45):
374         q = self._tcam.subscribe_multiprocess()
375         _, proc = pxdisplay.create(q, limit=limit, width=width, tmin=tmin,
376             ↪ tmax=tmax)
377
378         if block:
379             proc.join()
380
381     def playback(self, filen, tmin=15, tmax=45):
382         hz, playdata = self.file_to_capture(filen)
383
384         print(hz)
385
386         q, thread = pxdisplay.create(
387             limit=hz,
388             tmin=tmin,
389             tmax=tmax,
390             caption="Playing back '{}'.format(filen)
391         )
392
393         start = datetime.datetime.now()
394         offset = playdata[0]['start_millis']
395
396         for n, frame in enumerate(playdata):
397             frame['text'] = 'T+%.3f' % ((frame['start_millis'] - offset)/ 1000.0)
398             q.put(frame)
399
400     def display_close(self):
401         if self._display_thread is None:
402             return
403
404         self._display_stop = True
405         self._display_thread = None
406
407     def close(self):
408         self.display_close()
409
410     def capture_to_file(self, capture, hz, filen):
411         with open(filen + '_thermal.hcap', 'w') as f:
412             f.write(str(hz) + "\n")
413
414             for frame in capture:
415                 t = frame['start_millis']
416                 motion = frame['movement']
417                 arr = frame['ir']
418                 f.write(str(t) + "\n")

```

```

418         f.write(str(motion) + "\n")
419     for l in arr:
420         f.write('\t'.join([str(x) for x in l]) + "\n")
421     f.write("\n")
422
423 def capture_to_img_sequence(self, capture, directory, tmin=15, tmax=45,
    ↪ text=True):
424     hz, frames = capture
425     pxwidth = 120
426     print(directory)
427
428     for i, frame in enumerate(frames):
429         im = Image.new("RGB", (1920, 480))
430         draw = ImageDraw.Draw(im)
431         font = ImageFont.truetype("arial.ttf", 35)
432
433         for k, row in enumerate(frame['ir']):
434             for j, px in enumerate(row):
435                 rgb = pxdisplay.temp_to_rgb(px, tmin, tmax)
436
437                 x = k*pxwidth
438                 y = j*pxwidth
439
440                 coords = (y, x, y+pxwidth+1, x+pxwidth+1)
441
442                 draw.rectangle(coords, fill=rgb)
443
444                 if text:
445                     draw.text([y+20, x+(pxwidth/2-20)], str(px), fill=(255,255,255),
    ↪ font=font)
446
447         im.save(os.path.join(directory, '{:09d}.png'.format(i)))
448
449 def capture_to_movie(self, capture, filename, width=1920, height=480,
    ↪ tmin=15, tmax=45):
450     hz, frames = capture
451     tdir = tempfile.mkdtemp()
452
453     self.capture_to_img_sequence(capture, tdir, tmin=tmin, tmax=tmax)
454
455     args = [self._ffmpeg_loc,
456             "-y",
457             "-r", str(fractions.Fraction(hz)),
458             "-i", os.path.join(tdir, "%09d.png"),
459             "-s", "{}x{}".format(width, height),
460             "-sws_flags", "neighbor",
461             "-sws_dither", "none",
462             "-vcodec', 'qtrle', '-pix_fmt', 'rgb24',
463             filename + '_thermal.mov'

```

```

464     ]
465
466     subprocess.call(args)
467
468     def file_to_capture(self, filen):
469         capture = []
470         hz = None
471         with open(filen + '_thermal.hcap', 'r') as f:
472             frame = {'ir': []}
473
474             for i, line in enumerate(f):
475                 if i == 0:
476                     hz = float(line)
477                     continue
478
479                 j = (i-1) % 7
480                 if j == 0:
481                     frame['start_millis'] = int(line)
482                 elif j == 1:
483                     frame['movement'] = bool(line)
484                 elif 1 < j < 6:
485                     frame['ir'].append(tuple([float(x) for x in line.split("\t")]))
486                 elif j == 6:
487                     capture.append(frame)
488                     frame = {'ir': []}
489
490             return (hz, capture)
491
492     def capture(self, seconds, name=None, hcap=False, video=False):
493         buff = []
494         q = self._tcam.subscribe()
495         hz = self._tcam.irhz
496         tdir = tempfile.mkdtemp()
497
498         camera = None
499         visfile = name + '_visual.h264' #os.path.join(tdir, name + '_visual.h264')
500
501         if video and self._camera is not None:
502             self._camera.resolution = (1920, 1080)
503             self._camera.framerate = hz
504             self._camera.start_recording(visfile)
505
506         start = time.time()
507         elapsed = 0
508
509         while elapsed <= seconds:
510             elapsed = time.time() - start
511             buff.append( q.get() )
512

```

```

513     if video and self._camera is not None:
514         self._camera.stop_recording()
515
516         #args = [self._ffmpeg_loc,
517         #        # "-y",
518         #        # "-r", str(fractions.Fraction(hz)),
519         #        # "-i", visfile,
520         #        # "-vcodec", "copy",
521         #        # name + '_visual.mp4',
522         #        # ]
523
524         #subprocess.call(args)
525
526         #os.remove(visfile)
527
528
529     if hcap:
530         self.capture_to_file(buff, hz, name)
531
532     return (hz, buff)
533
534 def capture_synced(self, seconds, name, hz=2):
535     cap_method = getattr(self._tcam, "capture", None)
536     if not callable(cap_method):
537         raise "Provided tcam class must support the capture method"
538
539     if self._camera is None:
540         raise "No picamera object provided, cannot proceed"
541
542     camera = self._camera
543     camera.resolution = (1920, 1080)
544
545     # TODO: Currently produces black images. Need to fix.
546     # Wait for analog gain to settle on a higher value than 1
547     #while camera.analog_gain <= 1 or camera.digital_gain <= 1:
548     #    time.sleep(1)
549
550     # Now fix the values
551     #camera.shutter_speed = camera.exposure_speed
552     #camera.exposure_mode = 'off'
553     #g = camera.awb_gains
554     #camera.awb_mode = 'off'
555     #camera.awb_gains = g
556
557     import datetime, threading, time
558
559     dir_name = name
560     frames = seconds * hz
561

```

```

562     buff = []
563     imgbuff = [io.BytesIO() for _ in range(frames + 1)]
564     fps_avg = []
565     lag_avg = []
566
567     try:
568         os.mkdir(dir_name)
569     except OSError:
570         pass
571
572     def trigger(next_call, i):
573         if i % (hz * 3) == 0:
574             print('{} / {} seconds'.format(i/hz, seconds))
575
576             t1_start = time.time()
577             camera.capture(imgbuff[i], 'jpeg', use_video_port=True)
578             t1_t2 = time.time()
579             buff.append(self._tcam.capture())
580             t2_stop = time.time()
581
582             sec = t2_stop - t1_start
583             fps_avg.append(sec)
584             lag_avg.append(t2_stop - t1_t2)
585
586             if sec > (1.0/float(hz)):
587                 print('Cannot keep up with frame rate!')
588
589             if frames == i:
590                 return
591
592             th = threading.Timer( next_call - time.time(), trigger,
593                                 args=[next_call+(1.0/float(hz)), i + 1] )
594             th.start()
595             th.join()
596
597     trigger(time.time(), 0)
598
599     print('Average time for frame capture = {}
600         ↳ seconds'.format(sum(fps_avg)/len(fps_avg)))
601     print('Average lag between camera and thermal capture = {}
602         ↳ seconds'.format(sum(lag_avg)/len(lag_avg)))
603
604     self.capture_to_file(buff, hz, os.path.join(dir_name, 'output'))
605
606     for i, b in enumerate(imgbuff):
607         img_name = os.path.join(dir_name, 'video-{:09d}.jpg'.format(i))
608         with open(img_name, 'wb') as f:
609             f.write(b.getvalue())

```



```
609     return (hz, buff)
```

B.1.2 pxdisplay.py

```
1  from __future__ import division
2  from __future__ import print_function
3
4  from multiprocessing import Process, Queue
5  import colorsys
6  import time
7
8  def millis_diff(a, b):
9      diff = b - a
10     return (diff.days * 24 * 60 * 60 + diff.seconds) * 1000 + diff.microseconds
11         ↪ / 1000.0
12
13 def temp_to_rgb(temp, tmin, tmax):
14     OLD_MIN = tmin
15     OLD_MAX = tmax
16
17     if temp < OLD_MIN:
18         temp = OLD_MIN
19
20     if temp > OLD_MAX:
21         temp = OLD_MAX
22
23     v = (temp - OLD_MIN) / (OLD_MAX - OLD_MIN)
24
25     rgb = colorsys.hsv_to_rgb((1-v), 1, v * 0.5)
26
27     return tuple(int(c * 255) for c in rgb)
28
29 def create(q=None, limit=0, width=100, tmin=15, tmax=45, caption="Display"):
30     if q is None:
31         q = Queue()
32
33     p = Process(target=_display_process, args=(q, caption, tmin, tmax, limit,
34         ↪ width))
35     p.daemon = True
36     p.start()
37
38     return (q, p)
39
40 def _display_process(q, caption, tmin, tmax, limit, pxwidth):
41     import pygame
42     pygame.init()
43     pygame.display.set_caption(caption)
```

```

43 size = (16 * pxwidth, 4 * pxwidth)
44 screen = pygame.display.set_mode(size)
45
46 background = pygame.Surface(screen.get_size())
47 background = background.convert_alpha()
48
49 font = pygame.font.Font(None, 36)
50
51 while True:
52     for event in pygame.event.get():
53         if event.type == pygame.QUIT:
54             pygame.quit()
55             return
56
57     # Keep the event loop running so the windows don't freeze without data
58     try:
59         qg = q.get(True, 0.3)
60     except:
61         continue
62
63     px = qg['ir']
64
65     #lag = q.qsize()
66     #if lag > 0:
67     # print("WARNING: Dropped " + str(lag) + " frames")
68
69     for i, row in enumerate(px):
70         for j, v in enumerate(row):
71             rgb = temp_to_rgb(v, tmin, tmax)
72
73             x = i*pxwidth
74             y = j*pxwidth
75
76             screen.fill(rgb, (y, x, pxwidth, pxwidth))
77
78     if 'text' in qg:
79         background.fill((0, 0, 0, 0))
80         text = font.render(qg['text'], 1, (255,255,255))
81         background.blit(text, (0,0))
82
83     # Blit everything to the screen
84     screen.blit(background, (0, 0))
85
86     pygame.display.flip()
87
88     if limit != 0:
89         time.sleep(1.0/float(limit))

```

B.1.3 features.py

```
1  from __future__ import division
2  from __future__ import print_function
3
4  import threading
5  import pxdisplay
6  import time
7  import math
8  import copy
9  import networkx as nx
10 import itertools
11 import collections
12 #import matplotlib.pyplot as plt
13
14 def tuple_to_list(l):
15     new = []
16
17     for r in l:
18         new.append(list(r))
19
20     return new
21
22 def min_temps(l, n):
23     flat = []
24     for i, r in enumerate(l):
25         for j, v in enumerate(r):
26             flat.append(((i,j), v))
27     flat.sort(key=lambda x: x[1])
28
29     ret = [x[0] for x in flat]
30     return ret[:n]
31
32
33 def init_arr(val=None):
34     return [[val for x in range(16)] for x in range(4)]
35
36 class Features(object):
37     _q = None
38     _thread = None
39
40     _background = None
41     _means = None
42     _stds = None
43     _stds_post = None
44     _active = None
45
46     _num_active = None
47     _connected_graph = None
```

```

48     _num_connected = None
49     _size_connected = None
50
51     _lock = None
52
53     _rows = None
54     _columns = None
55
56     motion_weight = None
57     nomotion_weight = None
58
59     motion_window = None
60
61     hz = None
62
63     display = None
64
65     def __init__(self, q, hz, motion_window=10, motion_weight=0.1,
66         ↪ nomotion_weight=0.01, display=True, rows=4, columns=16):
67         self._q = q
68         self.hz = hz
69         self.motion_weight = motion_weight
70         self.nomotion_weight = nomotion_weight
71         self.display = display
72         self.motion_window = motion_window
73
74         self._active = []
75
76         self._rows = rows
77         self._columns = columns
78
79         self._thread = threading.Thread(group=None, target=self._monitor_thread)
80         self._thread.daemon = True
81
82         self._lock = threading.Lock()
83
84         self._thread.start()
85
86     def get_background(self):
87         self._lock.acquire()
88         background = copy.deepcopy(self._background)
89         self._lock.release()
90         return background
91
92     def get_means(self):
93         self._lock.acquire()
94         means = copy.deepcopy(self._means)
95         self._lock.release()
96         return means

```

```

96
97     def get_stds(self):
98         self._lock.acquire()
99         stds = copy.deepcopy(self._stds_post)
100        self._lock.release()
101        return stds
102
103     def get_active(self):
104         self._lock.acquire()
105         active = copy.deepcopy(self._active)
106         self._lock.release()
107         return active
108
109     def get_features(self):
110         self._lock.acquire()
111         num_active = self._num_active
112         num_connected = self._num_connected
113         size_connected = self._size_connected
114         self._lock.release()
115         return (num_active, num_connected, size_connected)
116
117     def _monitor_thread(self):
118         bdisp = None
119         ddisp = None
120
121         freq = self.hz * self.motion_window
122         mwin = collections.deque([False] * freq)
123
124         n = 1
125         while True:
126             if self.display and bdisp is None:
127                 bdisp, _ = pxdisplay.create(caption="Background", width=80)
128                 ddisp, _ = pxdisplay.create(caption="Deviation", width=80)
129
130                 fdata = self._q.get()
131                 frame = fdata['ir']
132
133                 mwin.popleft()
134                 mwin.append(fdata['movement'])
135                 motion = any(mwin)
136
137                 self._lock.acquire()
138
139                 self._active = []
140
141                 g = nx.Graph()
142
143                 if n == 1:
144                     self._background = tuple_to_list(frame)

```

```

145     self._means = tuple_to_list(frame)
146     self._stds = init_arr(0)
147     self._stds_post = init_arr()
148 else:
149     weight = self.nomotion_weight
150     use_frame = frame
151
152     # Not currently working
153     #if motion:
154     #     indeces = min_temps(frame, 5)
155     #     scalepx = []
156     #
157     #     for i, j in indeces:
158     #         scalepx.append(self._background[i][j] / frame[i][j])
159     #
160     #     scale = sum(scalepx) / len(scalepx)
161     #     scaled_bg = [[x * scale for x in r] for r in frame]
162     #
163     #     weight = self.motion_weight
164     #     use_frame = scaled_bg
165
166     for i in range(self._rows):
167         for j in range(self._columns):
168             prev = self._background[i][j]
169             cur = use_frame[i][j]
170
171             cur_mean = self._means[i][j]
172             cur_std = self._stds[i][j]
173
174             if not motion: # TODO: temp fix
175                 self._background[i][j] = weight * cur + (1 - weight) * prev
176
177             # maybe exclude these from motion calculations?
178             # n doesn't change when in motion, so it'll cause all sort of
179             ↳ corrupted results, as they use n?
180             self._means[i][j] = cur_mean + (cur - cur_mean) / n
181             self._stds[i][j] = cur_std + (cur - cur_mean) * (cur -
182                 ↳ self._means[i][j])
183             self._stds_post[i][j] = math.sqrt(self._stds[i][j] / (n-1))
184
185             if (cur - self._background[i][j]) > (3 * self._stds_post[i][j]):
186                 self._active.append((i,j))
187
188                 g.add_node((i,j))
189
190                 x = [(-1, -1), (-1, 0), (-1, 1), (0, -1)] # Nodes that have
191                 ↳ already been computed as active
192                 for ix, jx in x:
193                     if (i+ix, j+jx) in self._active:

```

```

191         g.add_edge((i,j), (i+ix,j+jx))
192
193     active = self._active
194
195     self._num_active = len(self._active)
196
197     components = list(nx.connected_components(g))
198
199     self._connected_graph = g
200     self._num_connected = nx.number_connected_components(g)
201     self._size_connected = max(len(component) for component in components)
202     ↪ if len(components) > 0 else None
203
204     self._lock.release()
205
206     if self.display:
207         bdisp.put({'ir': self._background})
208
209         if n >= 2:
210             std = {'ir': init_arr(0)}
211
212             for i, j in active:
213                 std['ir'][i][j] = frame[i][j]
214
215             ddisp.put(std)
216
217             #print(n)
218             #if n > 30:
219             #    nx.draw(g)
220             #    plt.show()
221
222         if not motion:
223             n += 1

```

B.2 Arduino Sketch

```

1  /**
2   * MLX90260 Arduino Interface
3   * Based on code from http://forum.arduino.cc/index.php/topic,126244.0.html
4   */
5  //#define __ASSERT_USE_STDERR
6
7  //#include <assert.h>
8  #include <math.h>
9  #include <Wire.h>
10 #include <EEPROM.h>

```

```

11 #include "SimpleTimer.h" // http://playground.arduino.cc/Code/SimpleTimer
12
13 // Configurable options
14 const int POR_CHECK_FREQ = 2000; // Time in milliseconds to check if MLX
    ↳ reset has occurred
15 const int PIR_INTERRUPT_PIN = 0; // D2 on the Arduino Uno
16
17 // Configuration constants
18 #define PIXEL_LINES 4
19 #define PIXEL_COLUMNS 16
20 #define BYTES_PER_PIXEL 2
21 #define EEPROM_SIZE 255
22 #define NUM_PIXELS (PIXEL_LINES * PIXEL_COLUMNS)
23
24 // EEPROM helpers
25 #define E_READ(X) (EEPROM_DATA[X])
26 #define E_WRITE(X, Y) (EEPROM_DATA[X] = (Y))
27
28 // Bit fiddling helpers
29 #define BYTES2INT(H, L) ( ((H) << 8) + (L) )
30 #define UBYTES2INT(H, L) ( ((unsigned int)(H) << 8) + (unsigned int)(L) )
31 #define BYTE2INT(B) ( ((int)(B) > 127) ? ((int)(B) - 256) : (int)(B) )
32 #define E_BYTES2INT(H, L) ( BYTES2INT(E_READ(H), E_READ(L)) )
33 #define E_UBytes2INT(H, L) ( UBYTES2INT(E_READ(H), E_READ(L)) )
34 #define E_BYTE2INT(X) ( BYTE2INT(E_READ(X)) )
35
36 // I2C addresses
37 #define ADDR_EEPROM 0x50
38 #define ADDR_SENSOR 0x60
39
40 // I2C commands
41 #define CMD_SENSOR_READ 0x02
42 #define CMD_SENSOR_WRITE_CONF 0x03
43 #define CMD_SENSOR_WRITE_TRIM 0x04
44
45 // Addresses in the sensor RAM (see Table 9 in spec)
46 #define SENSOR_PTAT 0x90
47 #define SENSOR_CPIX 0x91
48 #define SENSOR_CONFIG 0x92
49
50 // Addresses in the EEPROM (see Tables 5 & 7 in spec)
51 #define EEPROM_A_I_00 0x00 // A_i(0,0) IR pixel individual offset
    ↳ coefficient (ends at 0x3F)
52 #define EEPROM_B_I_00 0x40 // B_i(0,0) IR pixel individual offset
    ↳ coefficient (ends at 0x7F)
53 #define EEPROM_DELTA_ALPHA_00 0x80 // Delta-alpha(0,0) IR pixel individual
    ↳ offset coefficient (ends at 0xBF)
54 #define EEPROM_A_CP 0xD4 // Compensation pixel individual offset
    ↳ coefficients

```



```

55 #define EEPROM_B_CP          0xD5 // Individual Ta dependence (slope) of
    ↳ the compensation pixel offset
56 #define EEPROM_ALPHA_CP_L    0xD6 // Sensitivity coefficient of the
    ↳ compensation pixel (low)
57 #define EEPROM_ALPHA_CP_H    0xD7 // Sensitivity coefficient of the
    ↳ compensation pixel (high)
58 #define EEPROM_TGC           0xD8 // Thermal gradient coefficient
59 #define EEPROM_B_I_SCALE     0xD9 // Scaling coefficient for slope of IR
    ↳ pixels offset
60 #define EEPROM_V_TH_L        0xDA // V_TH0 of absolute temperature sensor
    ↳ (low)
61 #define EEPROM_V_TH_H        0xDB // V_TH0 of absolute temperature sensor
    ↳ (high)
62 #define EEPROM_K_T1_L        0xDC // K_T1 of absolute temperature sensor
    ↳ (low)
63 #define EEPROM_K_T1_H        0xDD // K_T1 of absolute temperature sensor
    ↳ (high)
64 #define EEPROM_K_T2_L        0xDE // K_T2 of absolute temperature sensor
    ↳ (low)
65 #define EEPROM_K_T2_H        0xDF // K_T2 of absolute temperature sensor
    ↳ (high)
66 #define EEPROM_ALPHA_O_L     0xE0 // Common sensitivity coefficient of IR
    ↳ pixels (low)
67 #define EEPROM_ALPHA_O_H     0xE1 // Common sensitivity coefficient of IR
    ↳ pixels (high)
68 #define EEPROM_ALPHA_O_SCALE 0xE2 // Scaling coefficient for common
    ↳ sensitivity
69 #define EEPROM_DELTA_ALPHA_SCALE 0xE3 // Scaling coefficient for individual
    ↳ sensitivity
70 #define EEPROM_EPSILON_L     0xE4 // Emissivity (low)
71 #define EEPROM_EPSILON_H     0xE5 // Emissivity (high)
72 #define EEPROM_TRIMMING_VAL  0xF7 // Oscillator trimming value
73
74 // Config flag locations
75 #define CFG_TA      8
76 #define CFG_IR      9
77 #define CFG_POR     10
78
79 // Arduino EEPROM addresses
80 #define AEEP_FREQ_ADDR 0x00
81
82 // Global variables
83 unsigned int PTAT; // Proportional to absolute temperature value
84 int CPIX; // Compensation pixel
85
86 int IRDATA[NUM_PIXELS]; // Infrared raw data
87 byte EEPROM_DATA[EEPROM_SIZE]; // EEPROM dump
88

```

```

89 float ta; // Absolute chip temperature / ambient chip
   ↪ temperature (degrees celsius)
90 float emissivity; // Emissivity compensation
91 float k_t1; // K_T1 of absolute temperature sensor
92 float k_t2; // K_T2 of absolute temperature sensor
93 float da0_scale; // Scaling coefficient for individual
   ↪ sensitivity
94 float alpha_const; // Common sensitivity coefficient of IR pixels
   ↪ and scaling coefficient for common sensitivity
95
96 int v_th; // V_THO of absolute temperature sensor
97 int a_cp; // Compensation pixel individual offset
   ↪ coefficients
98 int b_cp; // Individual Ta dependence (slope) of the
   ↪ compensation pixel offset
99 int tgc; // Thermal gradient coefficient
100 int b_i_scale; // Scaling coefficient for slope of IR pixels
   ↪ offset
101
102 float alpha_ij[NUM_PIXELS]; // Individual pixel sensitivity coefficient
103 int a_ij[NUM_PIXELS]; // Individual pixel offset
104 int b_ij[NUM_PIXELS]; // Individual pixel offset slope coefficient
105
106 char hdbuf[2]; // Hex printing buffer
107 int res; // Error code storage
108
109 float temp[NUM_PIXELS]; // Final calculated temperature values in
   ↪ degrees celsius
110
111 SimpleTimer timer; // Allows timed callbacks for temp functions
112
113 void(* reset_arduino_now) (void) = 0; // Creates function to reset Arduino
114
115 // Stores references to the 3 timers used in the program
116 int ir_timer;
117 int ta_timer;
118 int por_timer;
119
120 // Stores refresh frequency, read out of the EEPROM
121 short REFRESH_FREQ;
122
123 volatile bool pir_motion_detected = false;
124
125 /*
126 // Send assertion failures over serial
127 void __assert(const char *__func, const char *__file, int __lineno, const char
   ↪ *__sexp) {
128     // transmit diagnostic informations through serial link.
129     Serial.println(__func);

```

```

130     Serial.println(__file);
131     Serial.println(__lineno, DEC);
132     Serial.println(__sexp);
133     Serial.flush();
134     // abort program execution.
135     abort();
136 }*/
137
138 void reset_arduino() {
139     Serial.flush();
140     reset_arduino_now();
141 }
142
143 // Basic assertion failure function
144 void assert(boolean a) {
145     if (!a) Serial.println("ASSFAIL");
146 }
147
148 // Takes byte value and will output 2 character hex representation on serial
149 void print_hex(byte b) {
150     hpbuf[0] = (b >> 4) + 0x30;
151     if (hpbuf[0] > 0x39) hpbuf[0] +=7;
152
153     hpbuf[1] = (b & 0x0f) + 0x30;
154     if (hpbuf[1] > 0x39) hpbuf[1] +=7;
155
156     Serial.print(hpbuf);
157 }
158
159 // Will read memory from the given sensor address and convert it into an
    ⇐ integer
160 int _sensor_read_int(byte read_addr) {
161     Wire.beginTransmission(ADDR_SENSOR);
162     Wire.write(CMD_SENSOR_READ);
163     Wire.write(read_addr);
164     Wire.write(0x00); // address step (0)
165     Wire.write(0x01); // number of reads (1)
166     res = Wire.endTransmission(false); // we must use the repeated start here
167     if (res != 0) return -1;
168
169     Wire.requestFrom(ADDR_SENSOR, 2); // technically the 1 read takes up 2 bytes
170
171     int LSB, MSB;
172     int i = 0;
173     while( Wire.available() ) {
174         i++;
175
176         if (i > 2) {
177             return -1; // Returned more bytes than it should have

```

```

178     }
179
180     LSB = Wire.read();
181     MSB = Wire.read();
182 }
183
184 return UBYTES2INT(MSB, LSB); // rearrange int to account for endian
    ↪ difference (TODO: check)
185 }
186
187 // Will read a configuration flag bit specified by flag_loc from the sensor
    ↪ config
188 bool _sensor_read_config_flag(int flag_loc) {
189     int cur_cfg = _sensor_read_int(SENSOR_CONFIG);
190     return (bool)(cur_cfg & ( 1 << flag_loc )) >> flag_loc;
191 }
192
193 // Reads Proportional To Absolute Temperature (PTAT) value
194 int sensor_read_ptat() {
195     return _sensor_read_int(SENSOR_PTAT);
196 }
197
198 // Reads compensation pixel
199 int sensor_read_cpix() {
200     return _sensor_read_int(SENSOR_CPIX);
201 }
202
203 // Reads POR flag
204 bool sensor_read_por() {
205     return _sensor_read_config_flag(CFG_POR); // POR is 10th bit
206 }
207
208 // Read Ta measurement flag
209 bool sensor_read_ta_measure() {
210     return _sensor_read_config_flag(CFG_TA);
211 }
212
213 // Read IR measurement flag
214 bool sensor_read_ir_measure() {
215     return _sensor_read_config_flag(CFG_IR);
216 }
217
218 // Reads all raw IR data from sensor into IRDATA variable
219 boolean sensor_read_irdata() {
220     int i = 0;
221
222     // Due to wire library buffer limitations, we can only read up to 32 bytes
    ↪ at a time

```

```

223 // Thus, the request has been split into multiple different requests to get
224 // ↳ the full 128 values
225 // Each pixel value takes up two bytes (???) thus NUM_PIXELS * 2
226 for (int line = 0; line < PIXEL_LINES; line++) {
227     Wire.beginTransaction(ADDR_SENSOR);
228     Wire.write(CMD_SENSOR_READ);
229     Wire.write(line);
230     Wire.write(0x04);
231     Wire.write(0x10);
232     res = Wire.endTransmission(false); // use repeated start to get answer
233
234     if (res != 0) return false;
235
236     Wire.requestFrom(ADDR_SENSOR, PIXEL_COLUMNS * BYTES_PER_PIXEL);
237
238     byte PIX_LSB, PIX_MSB;
239
240     for(int j = 0; j < PIXEL_COLUMNS; j++) {
241         if (!Wire.available()) return false;
242
243         // We read two bytes
244         PIX_LSB = Wire.read();
245         PIX_MSB = Wire.read();
246
247         IRDATA[i] = BYTES2INT(PIX_MSB, PIX_LSB);
248         i++;
249     }
250 }
251
252 return true;
253 }
254
255 // Will send a command and the provided most significant and least significant
256 // ↳ bit
257 // with the appropriate check bit added
258 // Returns the Wire success/error code
259 boolean _sensor_write_check(byte cmd, byte check, byte lsb, byte msb) {
260     Wire.beginTransaction(ADDR_SENSOR);
261     Wire.write(cmd); // Send the command
262     Wire.write(lsb - check); // Send the least significant byte check
263     Wire.write(lsb); // Send the least significant byte
264     Wire.write(msb - check); // Send the most significant byte check
265     Wire.write(msb); // Send the most significant byte
266     return Wire.endTransmission() == 0;
267 }
268
269 // See datasheet: 9.4.2 Write configuration register command
270 // See datasheet: 8.2.2.1 Configuration register (0x92)
271 // Check byte is 0x55 in this instance

```

```

270 boolean sensor_write_conf() {
271     byte cfg_MSB = B01110100;
272     //          //////////
273     //          //////////*--- Ta measurement running (read only)
274     //          //////////*---- IR measurement running (read only)
275     //          //////////*----- POR flag cleared
276     //          //////////*----- I2C FM+ mode enabled
277     //          //**----- Ta refresh rate (2 byte code, 2Hz hardcoded)
278     //          /*----- ADC high reference
279     //          *----- NA
280
281     byte cfg_LSB = B00001110;
282     //          //////////
283     //          ////***** 4 byte IR refresh rate (4 byte code, 1Hz
284     //          ↪ default)
285     //          //**----- NA
286     //          /*----- Continuous measurement mode
287     //          *----- Normal operation mode
288
289     switch(REFRESH_FREQ) {
290     case 0: // 0.5Hz
291         cfg_LSB = B00001111;
292         break;
293     case 2:
294         cfg_LSB = B00001101;
295         break;
296     case 4:
297         cfg_LSB = B00001100;
298         break;
299     case 8:
300         cfg_LSB = B00001011;
301         break;
302     case 16:
303         cfg_LSB = B00001010;
304         break;
305     case 32:
306         cfg_LSB = B00001001;
307         break;
308     case 64:
309         cfg_LSB = B00001000;
310         break;
311     case 128:
312         cfg_LSB = B00000111;
313         break;
314     case 256:
315         cfg_LSB = B00000110;
316         break;
317     case 512:
318         cfg_LSB = B00000000; // modes 5 to 0 are all 512Hz

```

```

318     break;
319 }
320
321 return _sensor_write_check(CMD_SENSOR_WRITE_CONF, 0x55, cfg_LSB, cfg_MSB);
322 }
323
324 // See datasheet: 9.4.3 Write trimming command
325 // Check byte is 0xAA in this instance
326 boolean sensor_write_trim() {
327     return _sensor_write_check(CMD_SENSOR_WRITE_TRIM, 0xAA,
328         ↪ E_READ(EEPROM_TRIMMING_VAL), 0x00);
329 }
330
331 // Reads EEPROM memory into global variable
332 boolean eeprom_read_all() {
333     int i = 0;
334     // Due to wire library buffer limitations, we can only read up to 32 bytes
335     ↪ at a time
336     // Thus, the request has been split into 4 different requests to get the
337     ↪ full 128 values
338     for(int j = 0; j < EEPROM_SIZE; j = j + 32) {
339         Wire.beginTransaction(ADDR_EEPROM);
340         Wire.write( byte(j) );
341         res = Wire.endTransmission();
342
343         if (res != 0) return false;
344
345         Wire.requestFrom(ADDR_EEPROM, 32);
346
347         i = j;
348         while( Wire.available() ) { // slave may send less than requested
349             byte b = Wire.read(); // receive a byte as character
350             E_WRITE(i, b);
351             i++;
352         }
353     }
354
355     if (i < EEPROM_SIZE) { // If we didn't get the whole EEPROM
356         return false;
357     }
358
359     return true;
360 }
361
362 // Writes various calculation values from EEPROM into global variables
363 void calculate_init() {
364     v_th = E_BYTES2INT(EEPROM_V_TH_H, EEPROM_V_TH_L);
365     k_t1 = E_BYTES2INT(EEPROM_K_T1_H, EEPROM_K_T1_L) / 1024.0;
366     k_t2 = E_BYTES2INT(EEPROM_K_T2_H, EEPROM_K_T2_L) / 1048576.0;

```

```

364
365     a_cp = E_BYTE2INT(EEPROM_A_CP);
366     b_cp = E_BYTE2INT(EEPROM_B_CP);
367     tgc  = E_BYTE2INT(EEPROM_TGC);
368
369     b_i_scale = E_READ(EEPROM_B_I_SCALE);
370
371     emissivity = E_UBYTES2INT(EEPROM_EPSILON_H, EEPROM_EPSILON_L) / 32768.0;
372
373     da0_scale = pow(2, -E_READ(EEPROM_DELTA_ALPHA_SCALE));
374     alpha_const = (float)E_UBYTES2INT(EEPROM_ALPHA_O_H, EEPROM_ALPHA_O_L) *
        ↪ pow(2, -E_READ(EEPROM_ALPHA_O_SCALE));
375
376     for (int i = 0; i < NUM_PIXELS; i++){
377         float alpha_var = (float)E_READ(EEPROM_DELTA_ALPHA_OO + i) * da0_scale;
378         alpha_ij[i] = (alpha_const + alpha_var);
379
380         a_ij[i] = E_BYTE2INT(EEPROM_A_I_OO + i);
381         b_ij[i] = E_BYTE2INT(EEPROM_B_I_OO + i);
382     }
383 }
384
385 // Calculates the absolute chip temperature from the proportional to absolute
        ↪ temperature (PTAT)
386 float calculate_ta() {
387     float ptat = (float)sensor_read_ptat();
388     assert(ptat != -1);
389     return (-k_t1 +
390         sqrt(
391             square(k_t1) -
392             ( 4 * k_t2 * (v_th-ptat) )
393         )
394     ) / (2*k_t2) + 25;
395 }
396
397 // Calculates the final temperature value for each pixel and stores it in temp
        ↪ array
398 void calculate_temp() {
399     float v_cp_off_comp = (float) CPIX - (a_cp + (b_cp/pow(2, b_i_scale)) * (ta
        ↪ - 25));
400
401     for (int i = 0; i < NUM_PIXELS; i++){
402         float alpha_ij_v = alpha_ij[i];
403         int a_ij_v = a_ij[i];
404         int b_ij_v = b_ij[i];
405
406         float v_ir_tgc_comp = IRDATA[i] - (a_ij_v + (float)(b_ij_v/pow(2,
        ↪ b_i_scale)) * (ta - 25)) - (((float)tgc/32)*v_cp_off_comp);
407         float v_ir_comp = v_ir_tgc_comp / emissivity;

```



```

408     temp[i] = sqrt(sqrt((v_ir_comp/alpha_ij_v) + pow((ta + 273.15),4))) -
        ↪ 273.15;
409 }
410
411 }
412
413 // Prints all of EEPROM as hex
414 void print_eeprom() {
415     Serial.print("EEPROM ");
416     for(int i = 0; i < EEPROM_SIZE; i++) {
417         print_hex(E_READ(i));
418     }
419     Serial.println();
420 }
421
422 // Prints a serial "packet" containing IR data
423 void print_packet(unsigned long cur_time) {
424     Serial.print("START ");
425     Serial.println(cur_time);
426
427     Serial.print("MOVEMENT ");
428     Serial.println(pir_motion_detected);
429
430     for(int i = 0; i < NUM_PIXELS; i++) {
431         Serial.print(temp[i]);
432
433         if ((i+1) % PIXEL_COLUMNS == 0) {
434             Serial.println();
435         } else {
436             Serial.print("\t");
437         }
438     }
439
440     Serial.print("STOP ");
441     Serial.println(millis());
442     Serial.flush();
443 }
444
445 // Prints info about driver, build and configuration
446 void print_info() {
447     Serial.println("INFO START");
448     Serial.println("DRIVER MLX90620");
449
450     Serial.print("BUILD ");
451     Serial.print(__DATE__);
452     Serial.print(" ");
453     Serial.println(__TIME__);
454
455     Serial.print("IRHZ ");

```

```

456     Serial.println(REFRESH_FREQ);
457     Serial.println("INFO STOP");
458 }
459
460 // Runs functions necessary to initialize the temperature sensor
461 void initialize() {
462     assert(eeprom_read_all());
463     assert(sensor_write_trim());
464     assert(sensor_write_conf());
465
466     calculate_init();
467
468     ta_loop();
469 }
470
471 // Calculates absolute temperature
472 void ta_loop() {
473     ta = calculate_ta();
474 }
475
476 // Checks if the sensor as been reset, and if so, re-runs the initialize
477 ↳ functions
478 void por_loop() {
479     if (!sensor_read_por()) { // there has been a reset
480         initialize();
481     }
482 }
483
484 // Runs functions necessary to compute and output the temperature data
485 void ir_loop() {
486     unsigned long cur_time = millis();
487
488     assert(sensor_read_irdata());
489
490     CPIX = sensor_read_cpix();
491     assert(CPIX != -1);
492
493     calculate_temp();
494
495     print_packet(cur_time);
496
497     pir_motion_detected = false;
498 }
499
500 // Configures timers to poll IR and other data periodically
501 void activate_timers() {
502     float hz = REFRESH_FREQ;
503
504     if (REFRESH_FREQ == 0) {

```

```

504     hz = 0.5;
505 }
506
507 // Calculate how many milliseconds each timer should run for
508 // based upon the configured refresh rate of the IR data and
509 // absolute temperature data
510 long irlen = (1/hz) * 1000;
511 long talen = (1/2.0) * 1000;
512
513 if (talen < irlen) {
514     talen = irlen;
515 }
516
517 ir_timer = timer.setInterval(irlen, ir_loop);
518 ta_timer = timer.setInterval(talen, ta_loop);
519 por_timer = timer.setInterval(POR_CHECK_FREQ, por_loop);
520
521 attachInterrupt(PIR_INTERRUPT_PIN, pir_motion, RISING);
522 }
523
524 // Disables timers to poll IR and other data periodically
525 void deactivate_timers() {
526     timer.disable(ir_timer);
527     timer.deleteTimer(ir_timer);
528
529     timer.disable(ta_timer);
530     timer.deleteTimer(ta_timer);
531
532     timer.disable(por_timer);
533     timer.deleteTimer(por_timer);
534
535     detachInterrupt(PIR_INTERRUPT_PIN);
536 }
537
538 void pir_motion() {
539     pir_motion_detected = true;
540 }
541
542 void read_freq() {
543     byte rd = EEPROM.read(0);
544
545     if (rd > 9) {
546         rd = 0;
547         EEPROM.write(AEEP_FREQ_ADDR, 0);
548     }
549
550     switch(rd) {
551     case 1:
552         REFRESH_FREQ = 1;

```

```

553     break;
554 case 2:
555     REFRESH_FREQ = 2;
556     break;
557 case 3:
558     REFRESH_FREQ = 4;
559     break;
560 case 4:
561     REFRESH_FREQ = 8;
562     break;
563 case 5:
564     REFRESH_FREQ = 16;
565     break;
566 case 6:
567     REFRESH_FREQ = 32;
568     break;
569 case 7:
570     REFRESH_FREQ = 64;
571     break;
572 case 8:
573     REFRESH_FREQ = 128;
574     break;
575 case 9:
576     REFRESH_FREQ = 256;
577     break;
578 case 10:
579     REFRESH_FREQ = 512;
580     break;
581
582 default:
583 case 0:
584     REFRESH_FREQ = 0;
585     break;
586 }
587 }
588
589 void write_freq(int freq) {
590     byte wt;
591
592     switch(freq) {
593     case 1:
594         wt = 1;
595         break;
596     case 2:
597         wt = 2;
598         break;
599     case 4:
600         wt = 3;
601         break;

```

```

602     case 8:
603         wt = 4;
604         break;
605     case 16:
606         wt = 5;
607         break;
608     case 32:
609         wt = 6;
610         break;
611     case 64:
612         wt = 7;
613         break;
614     case 128:
615         wt = 8;
616         break;
617     case 256:
618         wt = 9;
619         break;
620     case 512: // writing 512 to the config doesn't work for some reason
621         wt = 10;
622         break;
623
624     default:
625     case 0:
626         wt = 0;
627         break;
628 }
629
630 EEPROM.write(AEEP_FREQ_ADDR, wt);
631 }
632
633 // Configure libraries and sensors at startup
634 void setup() {
635     pinMode(2, INPUT);
636
637     Wire.begin();
638     Serial.begin(115200);
639
640     Serial.println();
641     Serial.print("INIT ");
642     Serial.println(millis());
643
644     read_freq();
645     print_info();
646     initialize();
647
648     Serial.print("ACTIVE ");
649     Serial.println(millis());
650     Serial.flush();

```

```

651 }
652
653 char manualLoop = 0;
654
655 // Triggered when serial data is sent to Arduino. Used to trigger basic
    ↪ actions.
656 void serialEvent() {
657     while (Serial.available()) {
658         char in = (char)Serial.read();
659         if (in == '\r' || in == '\n') continue;
660
661         switch (in) {
662             case 'R':
663             case 'r':
664                 reset_arduino();
665                 break;
666
667             case 'I':
668             case 'i':
669                 print_info();
670                 break;
671
672             case 'T':
673             case 't':
674                 activate_timers();
675                 break;
676
677             case 'O':
678             case 'o':
679                 deactivate_timers();
680                 break;
681
682             case 'P':
683             case 'p':
684                 if (manualLoop == 16) { // Run ta_loop every 16 manual iterations
685                     ta_loop();
686                     manualLoop = 0;
687                 }
688
689                 ir_loop();
690
691                 manualLoop++;
692                 break;
693
694             case 'f':
695             case 'F':
696                 write_freq(Serial.parseInt());
697                 reset_arduino();
698                 break;

```

```
699
700     default:
701         Serial.println("UNKNOWN COMMAND");
702     }
703 }
704 }
705
706 void loop() {
707     timer.run();
708 }
```

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] AUSTRALIAN BUREAU OF STATISTICS. Household water and energy use, Victoria: Heating and cooling. Tech. Rep. 4602.2, October 2011. Retrieved October 6, 2014 from <http://www.abs.gov.au/ausstats/abs@.nsf/0/85424ADCCF6E5AE9CA257A670013AF89>.
- [6] BALAJI, B., XU, J., NWOKAFOR, A., GUPTA, R., AND AGARWAL, Y. Sentinel: occupancy based HVAC actuation using existing WiFi infrastructure within commercial buildings. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems* (2013), ACM, p. 17.
- [7] 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.
- [8] CHAN, M., CAMPO, E., ESTÈVE, D., AND FOURNIOLS, J.-Y. Smart homes - current features and future perspectives. *Maturitas* 64, 2 (2009), 90–97.
- [9] ERICKSON, V. L., ACHLEITNER, S., AND CERPA, A. E. POEM: Power-efficient occupancy-based energy management system. In *Proceedings of the 12th international conference on Information processing in sensor networks* (2013), ACM, pp. 203–216.

- [10] FISK, W. J., FAULKNER, D., AND SULLIVAN, D. P. Accuracy of CO2 sensors in commercial buildings: a pilot study. Tech. Rep. LBNL-61862, Lawrence Berkeley National Laboratory, 2006. Retrieved October 6, 2014 from http://eaei.lbl.gov/sites/all/files/LBNL-61862_0.pdf.
- [11] 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.
- [12] 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.
- [13] GUPTA, M., INTILLE, S. S., AND LARSON, K. Adding gps-control to traditional thermostats: An exploration of potential energy savings and design challenges. In *Pervasive Computing*. Springer, 2009, pp. 95–114.
- [14] HAILEMARIAM, E., GOLDSTEIN, R., ATTAR, R., AND KHAN, A. Real-time occupancy detection using decision trees with multiple sensor types. In *Proceedings of the 2011 Symposium on Simulation for Architecture and Urban Design* (2011), Society for Computer Simulation International, pp. 141–148.
- [15] HNAT, T. W., GRIFFITHS, E., DAWSON, R., AND WHITEHOUSE, K. Doorjamb: unobtrusive room-level tracking of people in homes using doorway sensors. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems* (2012), ACM, pp. 309–322.
- [16] KLEIMINGER, W., BECKEL, C., DEY, A., AND SANTINI, S. Inferring household occupancy patterns from unlabelled sensor data. Tech. Rep. 795, ETH Zurich, 2013. Retrieved October 6, 2014 from http://eaei.lbl.gov/sites/all/files/LBNL-61862_0.pdf.
- [17] KLEIMINGER, W., BECKEL, C., STAAKE, T., AND SANTINI, S. Occupancy detection from electricity consumption data. In *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings* (2013), ACM, pp. 1–8.
- [18] 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.

- [19] LI, N., CALIS, G., AND BECERIK-GERBER, B. Measuring and monitoring occupancy with an RFID based system for demand-driven HVAC operations. *Automation in construction* 24 (2012), 89–99.
- [20] 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.
- [21] 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.
- [22] SERRANO-CUERDA, J., CASTILLO, J. C., SOKOLOVA, M. V., AND FERNÁNDEZ-CABALLERO, A. Efficient people counting from indoor overhead video camera. In *Trends in Practical Applications of Agents and Multiagent Systems*. Springer, 2013, pp. 129–137.
- [23] SHELBY, Z., AND BORMANN, C. *6LoWPAN: The wireless embedded Internet*, vol. 43. John Wiley & Sons, 2011.
- [24] TEIXEIRA, T., DUBLON, G., AND SAVVIDES, A. A survey of human-sensing: Methods for detecting presence, count, location, track, and identity. Tech. rep., Embedded Networks and Applications Lab (ENALAB), Yale University, 2010. Retrieved October 6, 2014 from http://www.eng.yale.edu/enalab/publications/human_sensing_enalabWIP.pdf.
- [25] 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>.