The EE3 Smart House

Exploring the application of machine learning in the context of home automation

Grubb, Sebastian
sg3510@ic.ac.uk

Murray, Michael
mm510@ic.ac.uk

Janssen, Loek
lfj10@ic.ac.uk

Zhao, Tony
tz1010@ic.ac.uk

Mason, Jonathan
jhm210@ic.ac.uk

Singleton, Matthew
ms8210@ic.ac.uk

van Beek, Jonathan
jv510@ic.ac.uk

Hepburn, Connel
ch3810@ic.ac.uk

June 22, 2013

Deptartment of Electrical and Electronic Engineering
Imperial College London
London SW7 2AZ

Executive Summary

With modern societies continued demand for energy, dwindling natural resources and rising prices, technologies that can curb wasteful expenditure are becoming ever more relevant. In 2009, 28% of the total energy produced in the UK was consumed by households, 65.7% of which was derived from heating alone. Therefore there is a need for a product which can improve the efficiency of household central heating systems, and as a by product save people money. Our solution is an automated system which not only adjusts the home environment to suit the needs of the user, but also minimises wasteful energy consumption. With a unique machine learning approach, the system we propose in this report aims to revolutionise the way in which home environments are controlled.

The scope of existing home automation systems is varied, some providing only remote control while others just data processing. Autohome is a fully integrated system using a unique machine learning approach, that not only automates thermostat settings, but also provides user interaction and remote appliance control. As a result Autohome offers a more comprehensive service that any of its competitors, presenting its users with greater functionality and potential energy savings.

The Autohome solution can be broken down into two key layers; hardware and software. The hardware layer consists of a network of sensors situated in the users home that collects data on the internal conditions. The software layer sits on a server and processes the collected data to infer thermostat predictions. User interaction with the system is achieved through a mobile app and website. This allows the user to manually adjust the thermostat, remotely control appliances and view feedback statistics on their financial and energy expenditure. As a byproduct of making the user more aware of their energy usage, Autohome should encourage users to become more energy conscious.

Home automation is an emerging market which is expected to grow 16.1% year-on-year worldwide. However, for example, currently only 0.1% of UK and 5% of US homes have a home automation system. Therefore there is huge untapped potential in this market, with as yet no cemented market leader. Considering this, and also combining modern society's concerns over fossil fuel emissions with the current trend in smart interconnected devices, the conditions are ideal for Autohome to enter the market.

Contents

Section 1

Introduction

Problem Identification and Market Feasibility

1.1 Problem Identification

Modern society's continued and growing need for energy, combined with dwindling fossil fuel reserves and environmental concerns around CO2 emissions, has created an environment in which technology that can limit the wasteful expenditure of energy can thrive. In 2009 UK households consumed 501TWh accounting for 28-32% of the UK's total energy expenditure [1]. Central heating undeniably plays its role here, and following the current trend of a 3.7 degree centigrade rise in the internal temperature of houses since 1970[2] ,and rising numbers of UK households its contribution is only set to increase. As a result there is a need for products that can improve the efficiency of a household heating system.

Current household heating systems are controlled manually via a thermostat, which is neither convenient for the user or sensitive to the actual environmental conditions. In today's busy world few people have the time or discipline to consistently alter their thermostat to fit their own dynamic schedule or the variable nature of the weather. This is not only wasteful in terms of energy but is also very expensive. Therefore there is a niche for a system that can learn and adapt to the chaotic nature of people's lives and the weather.

The proposed solution to this problem is to create a machine learning system that once trained by the user, intelligently checks whether the occupants of the house are in, out or asleep; and sets the thermostat setting in accordance with this and the external conditions. This system should therefore save energy and make the user experience more convenient while not compromising on the comfort of the user. As a by product of saving energy the system should also save the user money in the long term; an important consideration is that in the UK 2.6 million households are in fuel poverty, and hence spend at least 10% of their income on heating [3].

1.2 Market and Product Feasibility

Objects are increasingly gaining the ability to communicate and sense their environment, forming what is broadly defined as the Internet of Things. This ability to collect and transmit a vast amount of data in real-time gives rise to new insights, allowing for better day-to-day decisions to be made.

The home automation market forms a subset of the Internet of Things market, with both expected to grow worldwide in 2013 with a CAGR of 16.1%[4] and CAGR[1] of 33.2%[5] , respectively. Thus it is the ideal time to build a home automation product when so many enablers exist.

1.2.1 Product Feasibility

The first enabler of a home automation product is the reduction in cost of low power processing components. For example the processor used for AutoHome costs £6.39 per unit[6], with sensors often costing less than £1 which means that when assembled an affordable sub £150 build price is expected. The second enabler is the increased penetration of smartphones and wireless technology that allows the possibility of an elegant home automation implementation, along with increased consumer demand for such technological solutions. Thirdly is the rise of cloud computing which is the main enabler of AutoHome, as data collected in real-time cannot be as efficiently processed on the mbed as the dataset becomes larger (sensor data being sampled at 5 minute intervals leads to 8640 data-points per house per month) - this is especially true of the computationally intensive machine learning algorithms. Computations are instead done on remote servers(such as Amazon's EC2 compute solution used for AutoHome), which are often more powerful and manage computing loads more efficiently. Additionally current

[1]Compounded Annual Growth rate

server solutions offer competitive rates priced at $0.060 per hour and above, relatively affordable operating costs for computing power.

1.2.2 Market Feasibility

With 17.5 million owner occupied homes in the UK and 87.4 million owner occupied homes in the US the home automation market has a huge potential. However only 0.1% and 5% of homes, in the UK and US respectively, have such home automation systems. This poor market penetration is due to multiple factors - such as the current state of home automation products, with many of them only offering remote automation and not smart automation. The key difference being that remote automation requires constant human interaction to control the house's settings (though scheduling is possible it is not truly smart), whereas smart automation provides a frictionless experience, constantly adapting to the user's needs. A second reason is the high prices (over-£300) of currently available systems[7], which are also described as clunky and tiresome to install. Thirdly is the lack of incentives, as most systems merely act as an additional way to control the house - which amounts to paying a premium to install extra light switches. There is thus little value seen in the cost of such devices, leaving room for a category of affordable, frictionless and smart home automation products which give the consumer clear value - such as saving money and energy.

Additionally recent UK policy has mandated that all UK energy providers must retrofit a smart meter in homes by 2020[8], meaning there is an additional market involving teaming up with energy providers to provide such systems.

The main market segments are homeowners, where retrofitting this device will be necessary. The other major segment is that of new homes, where there is a focus on delivering modern and energy conscious habitats - meaning that a deeper integration is possible, leading to even more efficient system.

1.2.3 Competitors

Many companies exist in the home automation market, such as AMX, Control4, iControl Networks and Vivint. However most of these only offer remote automation, as explained earlier, though they cover more components of a house, such as window shutter, TV control, house music and personal CCTV. The table below gives a breakdown.

	AutoHome	**Control4**	**Vivint**	**Nest Thermostat**	**Opower**
Cost	£150-£200	<£650	Starts at £38/month	£161	N/A (B2B company)
Features	-Smart thermostat -Advanced house state detection -lighting and smart meter offered -focused on energy savings -easily installable	-Availability by modules (starter kit $1000) -Work across residential, commercial and hospitality industries -Offer in depth control including security solutions -Requires contractor for installation	-Offer Home Automation, Energy Management and Home Security solutions -Provide intuitive solutions -No real “smart” automation -Requires contractor for installation	-Smart temperature control -Strong brand in part due to being made by ex-Apple engineers -Focused on saving money -Does not require contractor for installation	-Very good at energy analytics -Sold via energy companies -Provides intelligent software for thermostats -Purely focused on energy
Comparison	N/A	AutoHome smarter, focusing more on energy saving features	Smarter than Control4 but not to the extent AutoHome offers, no monthly fee for AutoHome.	Possibly main competitor. Offers similar “intelligent” algorithms but does not use as much data to regress on as AutoHome.	Competes on the algorithm/machine learning side but offers no other home automation solutions.

Section 2

Top Level Design

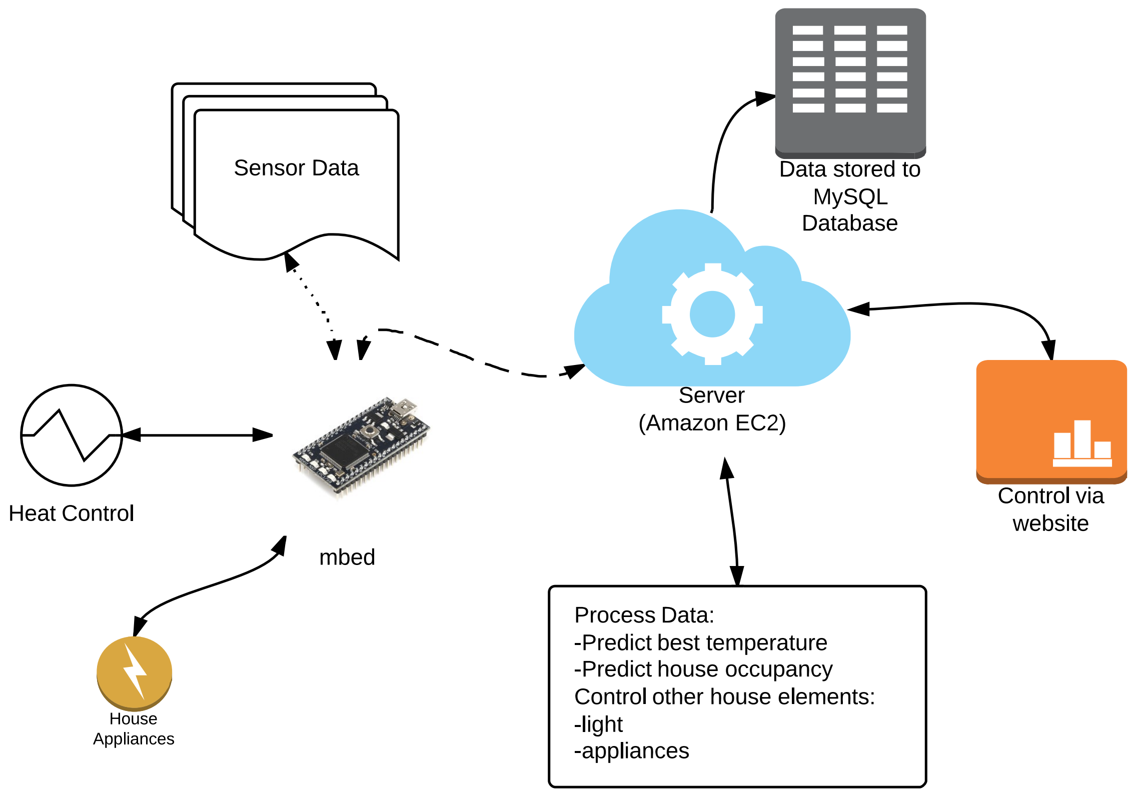


Figure 2.1: Overall System Design

Section 3

Modular Design

3.1 Hardware

For the first stage a prototype would be developed, which would include temperature and humidity measurement; an xbee module and a control interface; as well as the ability for the microprocessor to connect to the internet. To facilitate prototyping an 'mbed', prototyping board was used. This board is similar to a microcontroller breakout board but with many extra features, including a USB to FTDI connection for ease of programming, reset buttons and LEDs. This allowed for quick development and allowed basic prototyping to go ahead at an early stage. Each module was then developed in isolation using the mbed board, and then the final programme was written to combine the various elements together, after which the final link up of software and hardware was devised.

3.1.1 mbed

The 'mbed' prototyping board contains an LPC1768 processor which is a 32-bit ARM cortex-M3 microcontroller[24] and the board can be powered either through a USB connection or a 4.5-9.0V input voltage. Figure 3.1 shows the various pins that are available with the mbed board, including a regulated 3.3V supply voltage which is used to power several of the main components.

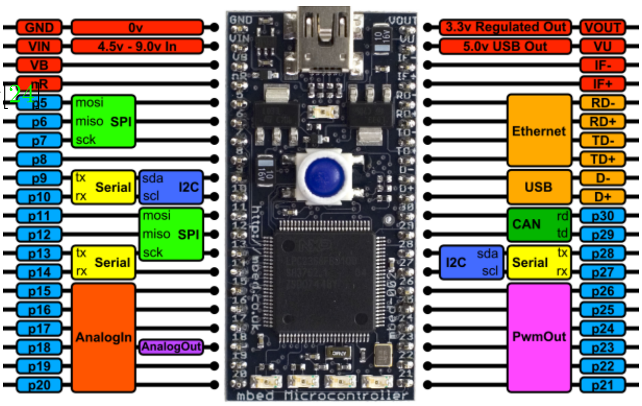


Figure 3.1: mbed NXP LPC1768 pinout diagram

3.1.2 Wireless Communications

As discussed in the top-level design, two wireless protocols would play pivotal roles in bringing the entire system together, with the standard wireless local area network (or Wi-Fi) to communicate with the internet via wireless routers which are now found within over 70% of UK households [25]. Then the xbee protocol was used to communicate with the various sensors and components around the house back to the microprocessor, due to both its low-power usage and the ability to easily create mesh networks. The mbed could then be used to pass the information and commands between the two networks (and hence our servers) as required.

Wi-Fi

As stated above there is a reasonable expectation that any home buying our smart thermostat will have wireless internet, both due to the target market and the high proportion of households with wireless routers. The roving networks Wifly RN-XV module was chosen due to its easy integration into the mbed system and the ability to easily call an ultra-low sleep mode during periods of decreased activity. One slight disadvantage was the inability of the module to join enterprise security networks at the current moment of time, with an updated firmware expected to make such connections possible within a year.

The Wifly module was then set up as shown in the overall circuit diagram (Appendix Section 8.1.1), using one of the 3 serial pair (RX and TX) connections available on the mbed. Simple ASCII commands using a serial console (in this case PuTTY) were used to experiment with the Wifly module, allowing demonstrable access to the internet which was tested using several standard ISP provided home routers.

Communicating with the server could be done via several methods, such as http and the new websockets. Both protocols were experimented, with websockets ultimately being chosen, this was due to the ability to have constant 2-way communication between the mbed and server, which could be set-up simply and easily. Using pre-existing libraries the correct SSID and WPA2-PSK code was placed in the Wifly, and then at the beginning of the programme the module is commanded to join the network (see appendix for code). One disadvantage that was found was that if the module failed to join a network over several resets (of the mbed) it would crash and have to be rebooted, either by power cycling or using command mode via PuTTY.

XBee

The XBee modules plays a paramount role in servicing communications between all the individual sensors and the MBED which is at the heart of everything. We chose the XBee as we believed that it provided the best features for our design. The XBee ZNET protocol can be used created a mesh network from all of the XBees. This is ideal for application in a household environment where consumers could add extra sensors which could possibly be placed a significant distance away from the central unit.

Figure 3.2 helps show how the mesh network operates. Our design incorporates one XBee co-coordinator on the central unit with all of the peripherals being configured as router XBees. The XBees which we used had the specifications listed below; the model which was used is the standard XBee not the XBee PRO.

Legend:
C - Coordinator
R - Router
E - End device

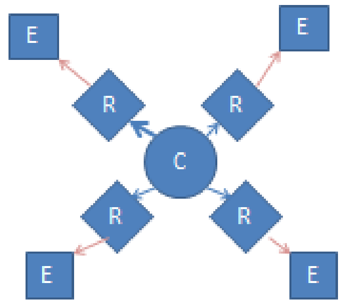


Figure 3.2: Diagram depicting the arrangement of the mesh network.

The XBee specifications allow a 40m indoor range which is more than enough to cover most households. On top of this is the fact that each XBee can use it's 40m range to form a mesh network allowing us to greatly increase the effective range of the network. The transmission power is far less than the WiFly equivalent. The power of the XBee is only 2mW even in boost mode vs. an average transmission power of 446mW of the lowest output power rating on the WiFly module in 802.11g mode.

There are two modes of communication to the XBee module, one is called transparent mode or AT mode and the other is the API (Application Programming Interface) mode. Our design is based upon the API mode as it allows greater flexibility and is significantly faster. This could be seen that when issuing commands to the XBee via AT, such as a change to a digital output pin level, there would be a noticeable amount of lag until transition occurred which was not found with API firmware. One other option which the API firmware provided the ability to easily address individual XBees in order to send them commands. This is critical as unique commands are associated with each peripheral type. The AT firmware requires that two registers be set with separate commands to change the destination address when sending to a specific XBee. This creates additional processing load for both the XBee and the central processing unit as well as reducing the speed at which commands can be sent to the XBees. API mode allows 64 bit addressing of XBees within an API frame, allowing a more efficient and convenient method of sending commands with the API framework.

The XBee uses a custom API framework which we created for the purposes of this project. This allows the MBED to construct basic API frames and send them out over its UART port. We have currently only implemented the frames which are required for our project to work though as it is a time consuming process to write and test the operation of a specific command. Currently the framework supports: Requests for the PAN ID, Changing the PAN ID, Sending data packets to other XBees, Requesting the value of all analogue and digital input pins, setting digital levels on the digital output pins.

Details of the framework,its use with different modules, the custom developed API and observed issues can be found in the appendix section 8.1.1.

Limitations and improvements

Had there been more time to debug and implement a more robust framework it would have been possible to improve it in a few ways. The first is to get a fully working UART interrupt such that everything can then be called on demand and not consume power when its not needed as well as providing more control on the data coming into the servers. This could be useful in order to throttle some of the connections if the server architecture wouldn't be able to handle the load. One more interesting thing would be to write custom firmware for the XBees. By doing this we could offload menial tasks from the MBED and put them onto the XBee. This could be useful if we wanted to convert the power calculations from their raw values to their actual values. This could then be offloaded from the MBED. This would help as the MBED uses a lot more power in its current form than one of the XBee modules. The power drawn has been measured at around 0.7W under normal operating conditions which is considerably higher than the XBee. By taking the load off the mbed we can activate and optimise a sleep routine for the MBED.

This can also be further advanced by changing the MBED with another model such as the one based on the Cortex M0+ from Freescale. This would allow even more power efficient operation as long as there was still enough processing power to run everything. The issue with the M0 is that it does have less connectivity options such as UARTs meaning that some external hardware might have to be used to compensate, thereby increasing the cost.

Lastly the MBED has its own RTOS[1]. By using an RTOS it would be possible to thread the web socket operations and the xbee operations in order to get more throughput from the microcontroller as well as massively reducing latency as well.This would come in useful with the power routine and its large wait cycle, allowing the

[1]Real Time Operating System

websocket to be serviced in the meantime. This would however increase complexity and thus would have taken more time to implement and debug.

3.1.3 Sensors

Temperature sensor

In order to measure temperature an 8-pin digital sensor was used that has a 2 wire serial interface with the microcontroller, using the I2C pins p28 and p27 upon the mbed board. This was the TMP75AIDR from Texas Instruments and for sensing the temperature, the chip itself is used with thermal paths running through the packaging. This is then converted internally to a digital temperature output with an accuracy of ± 0.5℃, which is then communicated to the microcontroller via I2C.

The I2C is a multimaster bus which allows easy digital communication with multiple devices if required, and hence the slave address must be set using 3 pin inputs which should match up to the chosen address in the software. Setting the 3 pins to ground gives the address 0x48 which we can see in the temperature configuration section of the code. Using libraries previously developed for mbed, a simple programme was devised (see appendix) which would only use a 9-bit resolution (set in the temperature configuration code) as a resolution of 0.0625℃(12-bit resolution) was deemed unnecessary. The chip itself was an SMD (surface mount device) and hence a small breakout was constructed to allow ease of use with breadboards.

Humidity sensor

Within the thermostat algorithm, internal humidity would play a role and so the low voltage HIH-5030 humidity sensor was acquired, also being an SMD model a small breakout board was required once again. The device has only 3 pins, a V_{cc} , ground, and analogue output which when measured can be used to determine the relative humidity (RH) of the atmosphere to ±3%.

Using the information supplied in the datasheet two equations were identified, one which could supply the sensor RH from the output voltage, and another to determine the true RH as the sensor RH is also affected by the temperature. Hence the temperature sensor is also required, the final equations within the mbed programme were then

$$SensorRH = \left(\tfrac{Voltage \times 5}{3.3} - 0.1515\right)/0.00636 \tag{3.1}$$

$$TrueRH = \frac{SensorRH}{1.0546 - (0.00216 \times Temperature)} \tag{3.2}$$

The final true relative humidity would then be calculated within the mbed, which would use an analogue-to-digital pin (ADC) to measure the output voltage of the sensor, the result can then be displayed and returned to the servers as required.

Wireless Power Monitoring

The ability to monitor the power consumption of a house in real time is useful in both making the user aware of the impact their habits have on their electricity bill and also to justify any saving the auto-home enables. This is by no means a new idea, and the British government, among many others, intend on a smart meter roll out scheme whereby every home in the UK will have a real time power monitor, or "smart meter", by 2020[9]. With this same power information, it is also possible to detect the appliances that are running in the house with non-intrusive load monitoring[10] (NILM), which can indicated unnecessary power usage when nobody is in the house, and switch these off to save power. With this considered, a power monitor is a useful part of the system, and was designed to make accurate measurements with minimum power usage and transmit this to the central unit.

Although measuring power seems a trivial exercise, at AC there are many ways of doing it. One way of making the measurement is calculating the power by summing many instantaneous voltage and current products over a period and divide by the number of samples. Although this is probably the most reliable and accurate method, it is also rather computationally intensive for a low powered 8-bit micro-controller. With the assumption that the voltage and current will have a reasonably consistent shape (sinusoidal), another simpler method is to measure the peak-peak voltage and current, and translate to RMS with a constant factor. However, this is quite a large assumption to make, and for many appliances such as switch mode laptop supplies, the current signal will be very different to that of a light bulb, but this will be less significant when looking at the total draw of a house giving an overestimation of the power usage, which isn't a bad thing.

With the peak-peak voltage/current measurements and a constant factor (for true sinusoids $\frac{\sqrt{2}}{2}$), the conversions can be made to RMS and the apparent power calculated from the product of the two. However, to be able to accurately calculate the dissipative real power, or characterise a load in terms of individual appliance signatures, the phase information must also be recorded. Again, for simplicity and ease of debugging, a comparator → XOR→ low-pass filter phase difference detector was used to give a third ADC input corresponding to the angle between the voltage and current. Again, by only a simple constant conversion factor, the angle and power factor can easily be calculated.

Motion Sensor

An important feature of the home automation project is the ability to detect whether there is anyone within the house. To do this we are using multiple passive infrared motion sensors, located within key locations, to monitor movement. This will provide one of the main sources of data for our learning algorithms. Our sensor itself is a long range infrared sensor with a detection range of up to 12m. With a detection angle of up to 102 degrees, this ensures that for the average sized house, all motion within a room will be detected (if placed in one corner). After an initial stabilisation time of roughly 30 seconds, the sensor will output a digital high when the detecting target is present and a digital low otherwise. This is fed straight to the XBee and then to the mbed for processing.

The method of detection for the sensor is via infrared radiation. This is perfectly suited for human detection from the radiation given out by their body temperature. It also rules out the potential of false triggering from wind blowing on open doors or curtains. However, this does not remove the problem of detection from pets. This is dealt with by the algorithm instead. The sensor consists of multiple polarized detection zones, each zone surrounded by 4 of opposite polarity. When the target moves across these detection zones, the combined polarity varies, hence movement is detected. However, a problem may occur if a target enters a positive and negative detection zone at the same time. In this case the signals will cancel each other and no detection notified. This will usually only occur near the maximum detection range, where the target is smaller in respect to the detection zone. Therefore it is not a significant problem as most rooms would not reach this distance.

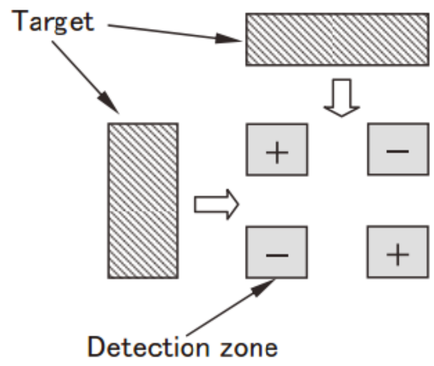
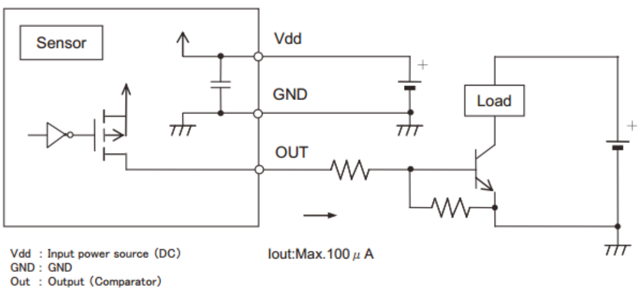


Figure 3.3: Space detection

Some other features of the sensor include a metallic can which encloses the sensing circuit. This increases the signal to noise ratio of the sensor. The result provides protection from the false detection of external electromagnetic fields caused by mobile phones and other electronic devices. The simplicity of the device along with the negligible power drawn when no detection is present means that very little total power is consumed during operation. The

Figure 3.4: Motion sensor circuit diagram



circuit is very simple and can be seen in figure 3.4. The load is replaced by the XBee and resistor values set to output current/voltage. The input power source is the standard 3.3V used for our XBee and mbed.

3.1.4 Other Hardware

Lighting

As part of the focus to improve user comfort and energy savings, another important feature of the home automation system is the capability to remotely control lights around the house. A large portion of a homeowner's electricity bill is contributed by lighting in the house. In the U.S, this figure is roughly 11 percent of a household's entire energy budget, and is significantly greater for commercial buildings [12]. Therefore only keeping the lighting on when necessary can provide substantial savings.

Our main feature is to keep track of all lighting in the house, allowing the user to see the states of the lights on either an app on their phone or via the website. The user can then change the state of the lights to on, off or auto accordingly. Notifications can also be set to alert the user if any light has been on for a certain length of time. The auto setting will access the motion sensors in the selected room to set those lights to be motion detected. People tend to have various preferences and opinions when it comes to motion detecting lights; therefore our system allows the user full control to choose what suits them best in each

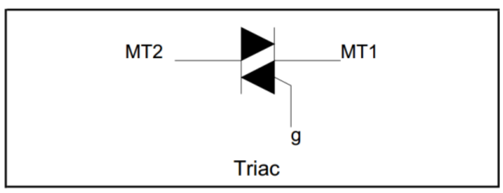
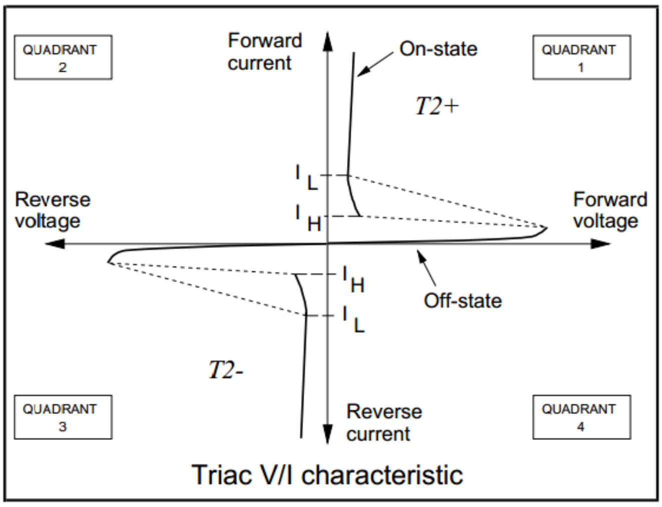


Figure 3.5: Triac diode diagram

room. For example, when in the living room watching TV, the auto setting may not be appropriate as the light will alternate between on/off due to the lack of movement. But the setting can be very convenient for a toilet, as it is used for shorter periods. All these features can ensure no lights are left switched on by accident.

Figure 3.6: Triac V/I characteristics



To remotely control the light switch, we use our mbed to send a signal via the zigbee wireless to an XBee module next to the switch. The XBee will then output a voltage as an input to a XOR gate, with the other being the manual switch located on the wall. Toggling either input will trigger an optoisolator, which will control the high power triac switch, turning it on/off. The triac is connected directly to the mains line of the light bulb, and will conduct only when triggered. The state of the light bulb will then be sent back via the XBee, updating the website and app.

The characteristics of a triac are ideal for our purpose of switching on/off the mains voltage. As shown in figure 3.5, it is made on two diodes with a gate connection leading from the side of MT1. The function can be comparable to a MOSFET, conducting in either direction when the gate requirements are met. However, unlike the MOSFET, the triac can conduct both AC and DC currents. This is vital as the mains voltage is always an AC. The gate of a triac is triggered by a current, rather than voltage for a MOSFET, and will remain conducting until the current drops below the latching current. For an alternating current, this means the triac will switch off twice every cycle. The V/I characteristics can be seen in figure 3.6.

To control the gate current of the triac, a zero-crossing optoisolator is used. The zero-crossing section ensures the gate of the triac is switched on when the current reaches 0. This is important as the phase of the current will be zero, so no noise is generated. Without it, switching could occur at peak voltages, which not only would produce EM noise, but also result in a large $\frac{dV}{dt}$ and potentially blowing the load. The optoisolator also keeps the mains voltage electrically isolated from the XBee circuit. This prevents the mains voltage blowing up the XBee in the case of a short circuit or if the triac blows. Isolation is achieved by simply using an LED to trigger a phototransistor. Hence, the input of the Xbee is attached to pins 1 and 2, and the triac to pins 4 and 6 (figure 3.7).

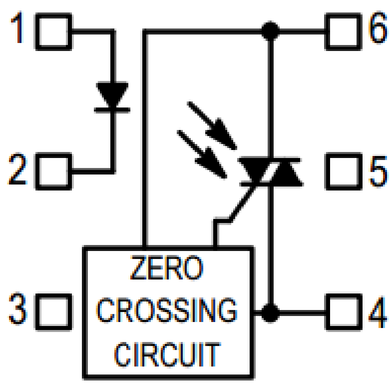


Figure 3.7: Triac module diagram

Not only does remotely controlled lighting reduce energy costs, it can also serve as a tool to provide home convenience for users. With a touch of the phone, no-longer do users need to feel around in the dark for the switch to use the toilet at night, or to get out of bed to switch lights off. Because the feature simply controls the mains supply, it can be easily applied to all electronic appliances. TVs, dishwashers and washing machines are just a few examples of common household appliances that are often left on standby. Switching these off when not in use can provide further cost cuts in the long run.

3.2 Software

Note: Software was developed in parallel of the hardware, thus no past data collection was available. Instead data made publicly available for the purpose of smart homes research used. The main sources used were the University of Massachusetts Amherst SMART* dataset [11] as well as Washington State's Tulum dataset [13].

3.2.1 Server Architecture and Technologies

The main server architecture works on a LAMP[2] server as this provides the system with a database (MySQL) to store all the collected sensor data, a means of processing this data (Python with the numerical NumPy and SciPy libraries) and easily controlling the system (HTTP server with PHP allowing delivery of interactive web-pages).

To enable the website to be able to access the data that is stored in the database, such as the predicted timetable or the past weather information, the server-side programming language of PHP was used. This is the language that is generally used for database access on websites, as it has good support for SQL databases and is easily used in conjunction with HTML. All PHP code is also executed on the server, before being returned to the end-user. This makes it useful as the code must include the database username and password, which would otherwise be passed to the user in the source code, giving anyone access to the database and the ability to alter information. PHP can also be used to create session variables on the server, which allows the creation of a basic login system. Again, as the code is executed server side, a redirect to the login page can be made if the user is not logged in, which will be executed before any of the html is loaded.

The only disadvantage of PHP being executed server side is that it cannot be used to write executable functions on the same page, which is needed when making changes to the timetable and a few other parts of the website. However using ajax from the jquery library, we can call other web pages without leaving the current page, and even pass variables to them which provides a solution for us.

PHP is also used for calling the weather API, to get the current weather conditions. The code gets the contents from a specially generated URL that contains the current weather conditions in JSON format. This can then be decoded, making each weather element easily accessible and these can then be stored into a database. This code is then be called every five minutes using a simple python script running constantly on the server.

Server Software

To control the system a constantly running program, monitoring, processing and relaying all the messages passed is needed. Thus a server-side program which could both send and receive messages to and from system modules was made.

To pass on messages the websocket technology was used. While a relatively new technology, websockets were chosen for their ideal use in real-time applications. This avoids using old 'hacks' such as using `HTTP POST` and `GET` requests[3] at irregular intervals to achieve a near-realtime connection. This is because websockets provide a full duplex connection with no need of polling[4] from the client. This would allow the message to turn off a light, or turn the temperature up, to be sent and received nearly instantly - thus being as useful if not more than a real switch.

The main server was written in Python, chosen for its ease of use and good integration with web technologies, using the tornado framework as code base. The server works by receiving messages in `JSON`[5] and passing them on to the appropriate program. For example if a user, via the website page, requests the current house temperature the server receives it and passes it on to the mbed module. Alternatively if the mbed sends the current sensor data to the server a separate script is called to add the values to a database and process this data to predict the most ideal temperature, for example. All the server code can be found in the appendix section 8.2.1.

[2]The LAMP acronym refers to the first letters of Linux, Apache(a HTTP server), MySQL (database solution), and PHP or Python, the main components to build most web servers.

[3]`HTTP POST` and `GET` requests are the traditional way in which webpages get accessed

[4]i.e. calling the server to ask for a reply

[5]`JSON` stands for JavaScript Object Notation and is a human readable standard allowing simple data structures to be represented. For example to encode the variable name='Alex' and age=26 the following message would be sent:`{"name":"Alex","age":26}`

Figure 3.8: Flow diagram of the server's general function

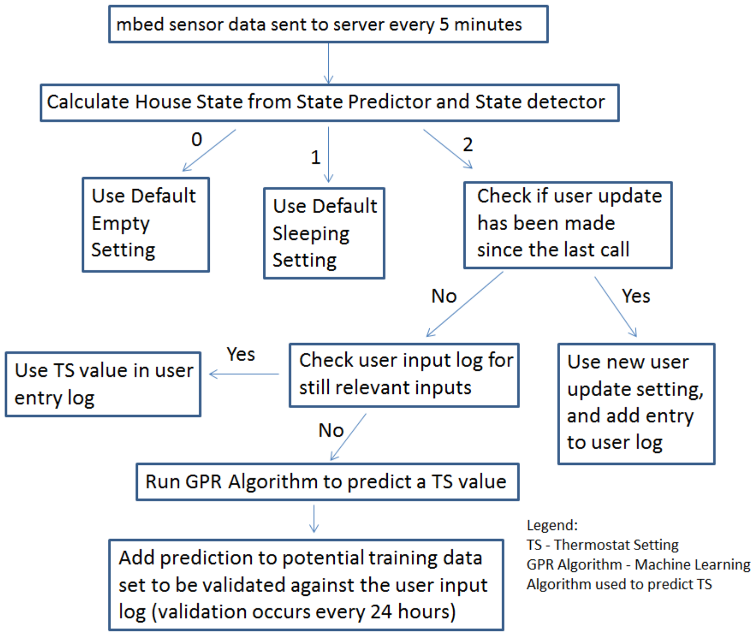


Figure 3.8 is an overview of how the server works when receiving the mbed data to then control the house's thermostat setting.

3.2.2 Hidden Markov Model for Occupation State Detection

3.2.3 Timetable Prediction

While a Hidden Markov Model provides a good way to determine the current state of the house another essential feature to ensure comfort and optimize energy usage is timetable prediction. Different methods were considered such as using the Hidden Markov Model to predict future states, taking an average state of every day of the week or making an ARMA/ARIMA model. The main criteria for choosing a method was an efficient one which could predict to a good degree of accuracy what will happen in about 6 hours in the future. Thus a method suggested by Microsoft Research [14] was implemented and adapted.

The idea behind the proposed solution is to look at n_p past states to predict the future n_f data-points by finding other similar instances. We thus measure similarity from taking the hamming distance of each time-series vector. Indeed each vector is in the form $\vec{t}_{vp} = [0, 1, 1, 1, 2, 2, \ldots, 2, 1, 0]$, with 0 signifying an away state, 1 a present state and 2 a sleeping state, meaning that taking the Hamming distance between two such vectors is possible. From this a determined number of similar days can be found to use their known future data-points to predict the current future. This is summarized in Algorithm 1.

Data: Time-series vector of past n_p states - t_{vp}
Result: Time-series vector of future n_f expected states - t_{vf}
take current time and give it an index n_n;
for *i index of all past days* **do**
  set t_{vd} equal to the $n_n - n_p$ to n_n data-points of day_i (i..e the same time range than t_{vp} but in the present);
  store in D_i the Hamming distance of t_{vd} and t_{vp};
end
get the indices of the 4 closest days in terms of Hamming Distance from D_i;
from these selected days get the n_n to $n_n + n_f$ data-points;
get the mode of these selected days and store in t_{vf};
return t_{vf}

Algorithm 1: Timetable State prediction

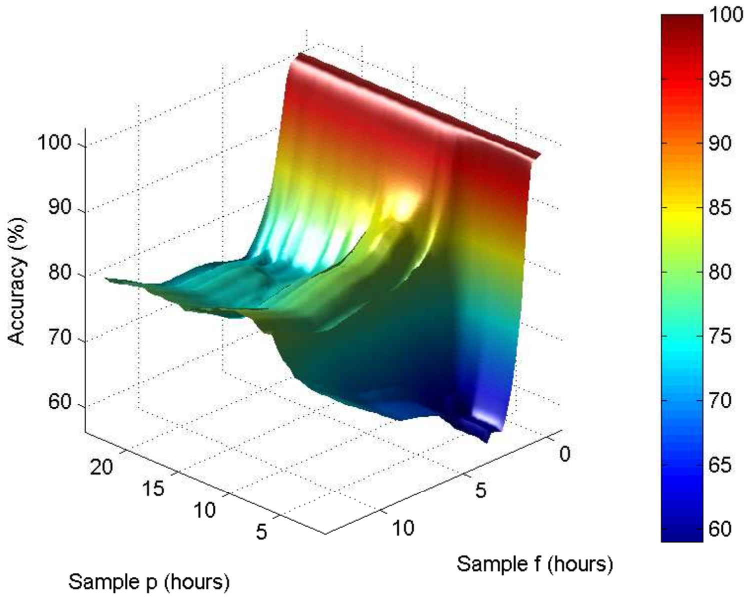
To find the optimal number of days to use to predict future expected states by taking their mode - i.e. should the closest x or y days be used - various prediction scenarios were run with the results presented in section 8.2.2. They show that, for the used dataset (3 months long), the optimal days to take the mode of was 4 as this consistently gave the highest average accuracy and one of the lowest standard deviations[6].

In deciding how long the vector $\vec{t}_{vp}$(i.e. the length of time to look back into the past to predict the future

[6]Over 3 months worth of data, about 80 days have their future states predicted and the accuracy of each of these instances is recorded, yielding an average and standard deviation for accuracy over different parameters

states) should be a MATLAB script testing for the accuracy of different settings[7]. In Figure 3.9 the results were plotted and suggest that taking the last 12 hours of house states gives us the most insight into the future - with the accuracy varying little with regards to the time looked into the future, except for very short amounts of time or 12 hours onwards which is because a house state is not likely to imminently change and that as we approach 24 hours we approach a day's length which is typically regular in nature.

Figure 3.9: Model of accuracy using time length past values (sample p) compared to the future hours predicted (sample f)



Data used: University of Massachusetts Amherst SMART dataset [11]*

3.2.4 Machine Learning for Thermostat Setting Prediction

A fundamental part of the functionality of the system is the ability to be able to predict the user's desired thermostat setting given the state of the house and external conditions. This can be achieved through a machine learning algorithm, which regresses through the training data provided by the user to predict a thermostat setting given external conditions not present in the training set. Two regression algorithms were developed, and in this section both are discussed.

Linear Regression

The initial implementation involved implementing a basic linear regression model to fit past data. This works by finding the parameters of the vector $\vec{\theta}$ which forms the prediction function $h(x,\theta) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x$ where $\vec{x}$ is the input data and n is the number of parameters. To fit the parameters we have a dataset of x's with it's corresponding values of y. From this a cost function $J(\theta) = \frac{1}{2m}\left(\sum_{i=1}^{m}\left(h(x_{(i)},\theta) - y_{(i)}\right)^2 + \lambda \sum_{i=1}^{n} \theta_i^2\right)$ is defined, where m is the number of data-points.Thus by choosing a θ which closely fits the input data we can see that $J(\theta)$ will be minimized. (The $\lambda \sum_{i=1}^{n} \theta_i^2$ term is there to ensure that θ's magnitude is not too large - avoiding the problem of the prediction function not being general enough for future, unknown inputs). Minimizing θ is a matter of performing a process called gradient descent which involves following the gradient direction. Figure 3.10 shows 3 arrows each representing the gradient at 3 different points. It can be seen from this that following the gradient leads to a less steep gradient until a minimum is reach - in which case the gradient will be zero[8].

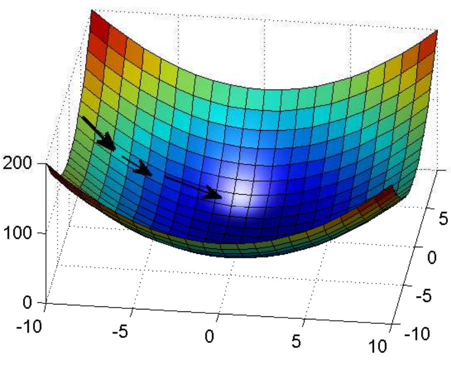


Figure 3.10: A convex cost function

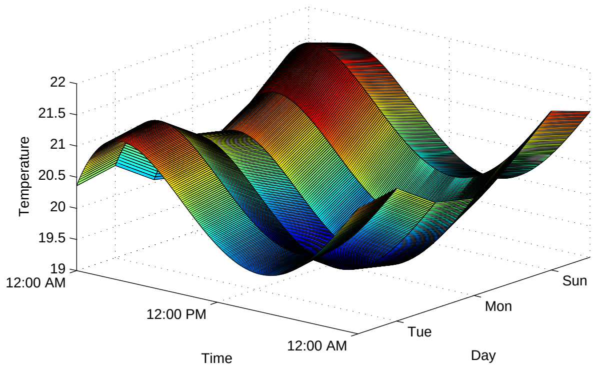
Using the SMART* dataset a θ was found to fit the past data whose features include inside & outside humidity,

[7]This was done by taking the states of each day in a 3 month long dataset separately, removing it from the data set, and trying to predict the future(know to us but unknown to the algorithm) states.

[8]This method also works on non-convex functions but will only return a local minima - however $J(\theta)$ is always convex for the case of linear regression

outside temperature,wind-speed, time of the day and day of the week with output inside temperature. From this Figure 3.11 was made, where all other features except time and day of the week were kept constant, to give an idea of the type of model created. It can be observed that this current model implies a trend of lower temperatures on weekdays with temperature consistently lower during sleeping hours - which implies that the model works in learning from past house activities.

Figure 3.11: Model of a house's temperature with time and day of the week as variables. Gradient descent was used to fit a 4^{th} order function.



Gaussian Process Regression

The issue with using 'non-Bayesian paradigm' algorithms, i.e. ones that typically assume no prior distribution or structure on a hypothesis, is that the regression curve is found (or trained) by minimizing the empirical error. However what is actually desired is to minimize the error on future predictions. This requires a Bayesian setting [15, 22].

Using a Bayesian setting a structure or prior is assumed, which, in the case of GPR, is a Multivariate Gaussian distribution [15, 16]. Parametric techniques, for example linear regression, restrict the class of functions which can fit the training data and then optimize parameters by minimizing the error between the target data set and the predicted values[15, 23]. GPR in contrast allows every possible function but gives each a prior probability, which can then be used to calculate a posterior probability once the training data has been taken into account. In this way then we can think of the GPR as filtering the functions in terms of their likelihood given the training data. As a result, while parametric techniques can generate curves that closely fit the data, only Bayesian techniques like GPR can quantify the expected error and therefore work to minimize it[15, 17, 18, 20].

Notes on Notation:

In this section, bold font indicates vectors and matrices while non-bold indicates scalars and functions. If a character denotes a function then the emboldened version of the same character represents a set or vector of elements taken by evaluating that function at certain points:

- Features vectors are denoted by $\boldsymbol{x}$.
- The training data is a set of feature points. In matrix form it is denoted by $\boldsymbol{X}$ and represents a collection of feature vectors. 'n' is the number of training examples in our training set.
- $f(\boldsymbol{x})$ (or,in suppressed argument form,f) is the hypothesis function we wish to infer. f^* is the prediction we want to make about the function $f(\boldsymbol{x})$ evaluated at a point $\boldsymbol{x}^*$ not present in the training set.
- $y(\boldsymbol{x})$ (in suppressed argument form as y) is the function of observed thermostat results, i.e. the underlying function we wish to infer plus a noise term: $y(\boldsymbol{x}) = f(\boldsymbol{x}) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma_n^2 \boldsymbol{I})$ is Independent Additive White Gaussian Noise (IAWGN) with $var(\epsilon) = \sigma_n^2$. The target data of the training set is denoted by y and is simply a vector of elements of $y(\boldsymbol{x})$ evaluated at different points.
- 'θ' Indicates the hyperparameters that are used to define the covariance function, in vector form this is denoted '$\boldsymbol{\theta}$'.

It is important to bear in mind that a vector or matrix is a way of expressing a set of data points. If these data points are random then this data set can be written as a random vector or matrix whose elements are random variables each with a probability distribution, which in the case of GPR is Gaussian. As a result data sets, made up of a collection of random variables each with a Gaussian distribution can be expressed as a multivariate Gaussian distribution.

Prior and Hyperparameters:

As previously discussed GPR is a non-parametric technique, which makes it a powerful tool as no assumptions need to be made on the hypothesis function directly, instead only on the probability distribution of the hypothesis function. The reason that GPR is a good choice in this case is because Gaussian distributions are continuous (important as the hypothesis function we wish to infer is continuous) and flexible. The prior for GPR is defined as:

$$f \sim GP(\mathbf{0}, k(\boldsymbol{x_i}, \boldsymbol{x_j})) \tag{3.3}$$

The above equation states that without any information the distribution of possible hypothesis functions is defined by a Gaussian Process with a mean function of 0 and a covariance function k. As a result to specify the Gaussian prior all that is required is to describe its covariance function (since the mean function can be set to 0 without any loss of generality) [15] . The kernel deployed in this particular application is the squared exponential, also called the Gaussian Kernel, with an added noise term [15]:

$$k(\boldsymbol{x_i}, \boldsymbol{x_j}) = \sigma_f^2 \exp\left(-\frac{1}{2}(\boldsymbol{x_i} - \boldsymbol{x_j})^T \boldsymbol{M}(\boldsymbol{x_i} - \boldsymbol{x_j})\right) + \sigma_n^2 \boldsymbol{\delta_{ij}} \tag{3.4}$$

$\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ are input feature vectors, σ_f^2 is the 'signal power' which represents the overall scale variation of the latent (not yet manifested) thermostat setting. σ_n^2 is the 'noise power' representing the noise variation or 'jitter' in the data [23]. Physically this is the manifestation of the users own error in their thermostat setting. For example, on a particularly cold day the user may be tempted to initially turn up their thermostat setting beyond what they actually find comfortable or need. δ_{ij} is the Kronecker product, meaning that the noise term is only added to the variance of a point[23]. This is due to the nature of the noise being IAWGN i.e. the noise on different samples is independent. Finally M is a diagonal matrix containing the length scales of each feature [15]:

$$\boldsymbol{M} = \begin{bmatrix} \frac{1}{l_1^2} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \frac{1}{l_n^2} \end{bmatrix} \tag{3.5}$$

Length scales represent the distance in a feature dimension that must be moved before the function value changes significantly. Large length scales imply irrelevant features while short length scales mean that the error in the prediction grows rapidly away from the data points [23]. Having individual length scales for each feature is more flexible than having just one general length scale as there will be variation in length scales from feature to feature.

The error squared exponential kernel was chosen as it provides a smooth (infinitely differentiable) hypothesis and is generally the most widely used and applicable kernel in machine learning.

Hyperparameter Optimization:

To achieve accurate predictions the hyperparameters need to be optimized to ensure the model best fits the training data. In this way the task of 'teaching' a Gaussian Process Regressor is therefore equivalent to optimizing the hyperparameters to fit the training data. The marginal likelihood is the 'model evidence' and by maximising the marginal likelihood the optimum hyperparameters can be found (See appendix for more information)[15, 18, 21, 22].

The probability of the target data 'y' given the training data, hyperparameters and assumed prior distribution is the called the likelihood [21, 22]:

$$P(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}, \boldsymbol{f}) \sim \mathcal{N}(\boldsymbol{f}, \sigma_n^2 \boldsymbol{I}) \tag{3.6}$$

This distribution makes intuitive sense considering that $y(\boldsymbol{x}) = f(\boldsymbol{x}) + \epsilon$ [15, 19] and that the any collection of samples drawn from a Gaussian Process naturally has a Multivariate Gaussian Distribution. The marginal likelihood, $P(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta})$, is found by taking the conditional probability of the likelihood given certain values of f, and then averaging this by integrating over all values of f. This is equivalent to taking the expectation:

$$P(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}) = E\{P(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}, f)\} = \int P(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}, f) P(f|\boldsymbol{X}, \boldsymbol{\theta}) \, df \tag{3.7}$$

$P(\boldsymbol{f}|\boldsymbol{X}, \boldsymbol{\theta}) \sim N(\mathbf{0}, \boldsymbol{K})$ is the probability distribution of the function we wish to infer without any target set. In this way, it is equivalent to simply drawing points from our prior distribution. We can see therefore that both the prior set and the likelihood set are both Gaussian. The product of two Gaussians is also Gaussian, integrating over f it can be proved that[15]:

$$\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{K} + \sigma_n^2 \boldsymbol{I}) \tag{3.8}$$

Referring to the equation for the PDF of a Gaussian Multivariable Distribution, and setting $\boldsymbol{C} = \boldsymbol{K} + \sigma_n^2 \boldsymbol{I}$ the PDF for the Marginal Likelihood (see Appendix) can be defined:

$$P(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) = 2\pi^{-\left(\frac{n}{2}\right)} |\boldsymbol{C}|^{-0.5} \exp\left(-\frac{1}{2} y^T \boldsymbol{C}^{-1} y\right) \tag{3.9}$$

In its current form it is not particularly conducive to the task of optimization. A common and successful technique for optimizing any parameters is to turn the problem into a minimization task, and find a global or acceptably good local minima using gradient descent. Taking the natural log of the equation:

$$\log\left(P\left(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta}\right)\right) = \ -\left(\frac{n}{2}\right)\log\left(2\pi\right)\ -\frac{1}{2}\log\left(|C|\right)\ -\frac{1}{2}y^{T}C^{-1}y \tag{3.10}$$

Rearranging and multiplying by negative one we end up with what is known as the 'Negative Log Marginal Likelihood' or NLML [21, 22]:

$$-\log\left(P\left(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta}\right)\right) = \frac{1}{2}y^{T}C^{-1}y + \frac{1}{2}\log\left(|C|\right) + \frac{n}{2}\log\left(2\pi\right) \tag{3.11}$$

Note that by minimizing the NLML we are maximising the Marginal Likelihood and therefore gradient descent techniques can be applied. First the hyperparameters are initialised randomly and then taking the partial differential of the NLML with respect to each hyperparameter the error curve can be traversed (error curve is the NLML as a function of the hyperparameters) to find a minima. Denoting the NLML as $L\left(\theta\right)$ [22]:

$$\frac{\partial L\left(\boldsymbol{\theta}\right)}{\partial\theta_i} = \frac{1}{2}tr\left(\boldsymbol{C}^{-1}\frac{\partial\boldsymbol{C}}{\partial\theta_i}\right) - \frac{1}{2}\boldsymbol{y}^{T}C^{-1}\frac{\partial\boldsymbol{C}}{\partial\theta_i}C^{-1}\boldsymbol{y} \tag{3.12}$$

A basic gradient descent function can be used to update the hyperparameters by subtracting a value proportional to the partial derivative of the NLML with the hyperparameter in question. Over the process of many iterations, and given an appropriately sized step size α (i.e. one that is not too large so as to diverge or too small so as to converge too slowly), the NLML will converge to a point of zero gradient. As we are subtracting this term this convergence limits towards a minima, not a saddle point or maxima:

$$\theta_i = \theta_i -\ \alpha\frac{\partial L\left(\theta\right)}{\partial\theta_i} \tag{3.13}$$

Unfortunately this problem is not convex (i.e. the error curve is not convex) and therefore there does not exist a global minima. Instead there is the potential for many minima and maxima depending on our selection of training data. As a result all that can be hoped for is that gradient descent gives us an acceptable local minimum.

The convergence point depends on what the initialisation of the hyperparameters; different initialisations lead to different starting points on the error curve and therefore potentially different local minima. The practical approach taken to solve this issue was to perform a grid search over an area of the error curve (that seemed most logical given our instincts on the data), perform gradient at points initialised in divisions of this area and then choose the area that seemed to give the lowest average NLML. The division with the lowest NLML can then be selected as the new area of consideration, and the process can be repeated till an appreciable initialisation range is found. This, and the computational complexity of all the algorithms discussed in this section, is discussed in more detail in the results section.

Intuition behind Gaussian Process Regression:

With a Gaussian prior the assumption is made that the space of possible functions that we can use as our hypothesis is normally distributed. By then observing data, i.e. getting a training set, the distribution can be conditioned on the training set. By then finding the mean or average of this distribution function that best, or most likely fits the user's desired thermostat setting given certain environmental conditions, can be found. Points in the training data set can be seen as anchors in this function distribution having zero variance [15, 17, 20, 23].

Considering a new input feature vector $\boldsymbol{x^*}$ and a desired prediction about $f(\boldsymbol{x}^*)$ (f^* for convenience) then adding this point to the training data results in a joint Gaussian distribution [15]:

$$\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{f^*} \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{0},\ \begin{bmatrix} \boldsymbol{K}+\sigma_n^2\boldsymbol{I} & \boldsymbol{K_*} \\ \boldsymbol{K_*^T} & \boldsymbol{K_{**}} \end{bmatrix}\right) \tag{3.14}$$

Here $\boldsymbol{K_*} = \begin{bmatrix} k\left(\boldsymbol{x_*},\boldsymbol{x_1}\right) \\ \vdots \\ k\left(\boldsymbol{x_*},\boldsymbol{x_n}\right) \end{bmatrix}$ denotes the covariance matrix (or in this case, as $\boldsymbol{x}^*$ is a single vector, a vector) of the input feature vector with each of the training examples. As already discussed the objective is to determine the best or most likely value of f^* given the training data. As a result what is of greatest relevance is the conditional probability of the predicted value given the training data. This is known as the 'posterior probability', and can be derived from Bayes Theorem:

$$P\left(f^*|\boldsymbol{X},\boldsymbol{y},\boldsymbol{\theta}\right) = \frac{P\left(\boldsymbol{f}|\boldsymbol{X},\boldsymbol{\theta}\right)P(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta},\boldsymbol{f})}{P\left(\boldsymbol{y}|\ \boldsymbol{X},\boldsymbol{\theta}\right)} \tag{3.15}$$

Referring to the set of definitions given above we can write this alternatively as [15]:

$$posterior = \frac{prior \times likelihood}{marginal\ likelihood} \tag{3.16}$$

The marginal likelihood is a 'normalizing constant', i.e. a constant as it is the Likelihood function marginalized over f. Therefore the posterior probability can be more simply written as[15, 21, 22]:

$$P\left(f^{*} \mid \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{\theta}\right) \propto P(\boldsymbol{f}|\boldsymbol{X}, \boldsymbol{\theta}) P(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}, \boldsymbol{f}) \tag{3.17}$$

As the prior and the likelihood function both follow a Gaussian distribution then the posterior probability distribution is also Gaussian (intuitively multiplying two exponentials together renders another exponential). Using some basic results found in Linear Algebra [19] this Gaussian distribution can be shown to have a mean and standard deviation of the following[15]:

$$P\left(f^{*}|\boldsymbol{y}, \boldsymbol{X}\right) \sim \mathcal{N}(\boldsymbol{K_{*}^{T}}\left(\boldsymbol{K}+\sigma_{n}^{2} \boldsymbol{I}\right)^{\mathbf{-1}} \boldsymbol{y}, \, \boldsymbol{K_{**}} - \boldsymbol{K_{*}^{T}}\left(\boldsymbol{K}+\sigma_{n}^{2} \boldsymbol{I}\right)^{\mathbf{-1}} \boldsymbol{K_{*}}) \tag{3.18}$$

The above equation describes the distribution of the possible hypothesis functions given the data available. The average or mean of this distribution gives the most likely regression function given the data; therefore the mean of this distribution is the best estimate of f^*:

$$\hat{f}^{*} = \boldsymbol{K_{*}^{T}}\left(\boldsymbol{K}+\sigma_{n}^{2} \boldsymbol{I}\right)^{\mathbf{-1}} \boldsymbol{y} \tag{3.19}$$

The confidence in the prediction is described by the variance $\boldsymbol{K_{**}} - \boldsymbol{K_{*}^{T}}\left(\boldsymbol{K}+\sigma_{n}^{2} \boldsymbol{I}\right)^{\mathbf{-1}} \boldsymbol{K_{*}}$, as a result GPR can calculate a best estimate of the desired thermostat setting given the training data and also gives a metric for the confidence the system has in the prediction.

3.3 User Interface

3.3.1 Website

The website was designed to be easy to use and minimalistic as to keep it as user friendly as possible. All text is placed inside lightened boxes, otherwise it became difficult to read against the background.

Trying to access any page other than the login page before logging in, will result in a redirect to the login page. Upon logging in, the user is forwarded to the main home page. The home page greets the user by their username, giving the site a more personal feel. This page also contains a toggle slider to turn the system off, for instance if they are going away for an extended period of time.

The control page gives direct control of the lights remotely via a simple toggle switch. The switch also updates in real time, so if the light is turned on or off manually, this will instantly be updated on the website. This means that you can check if you have left your light on when you are not at home and even turn it off if you have.

The statistics page primarily displays the the schedule for the house, showing the user when they are expected to be in, out or asleep. If the user disagrees with the prediction, they are also able to update the schedule themselves, with user corrections being displayed in a different colour.

At the bottom of the page, the user can view graphs of past external weather conditions, as well as past internal conditions, including the power usage of the house.

The thermostat page has the primary use of updating the current thermostat setting manually. Every time the user does this, the value is put into a table along with the most recent external and internal conditions, so that the algorithm that selects the temperature can learn from this.

The user can also change the maximum and minimum temperatures that the algorithm selects, as well as their preferred sleeping temperature.

3.3.2 Mobile App

Along with the website the decision was made to develop an application (or app) for smartphones, this was to be done for the android operating system and then would later be ported to the it's various competitors. The reasons for this were, that android was the most popular operating system for smartphones; the multitude of available examples and developer information, and most importantly the open (hence free) source nature of the operating system.

Using Eclipse combined with android SDK (software development kit) tools a basic app was built with a separate temperature and lights page which would then be used to change the thermostat setting and turn the lights on and off (with toggle buttons). Using an open source android websocket library the app is able to communicate with the server and adjust the various attributes of the system.

Section 4

System Analysis and Use

4.1 Savings Performance

Justifying the home automation system in terms of saving energy over the conventional thermostat is critical in justifying its usefulness. Firstly, the motivation to cut down the use of domestic heating is significant; housing accounted for over 28.3% of the UK's energy consumption in 2009[1] at 501 TWh. Of this, a staggering 65.7% is accounted for the heating alone. Naturally, cutting this will make a significant impact on total energy usage and the environment.

In order to estimate the energy the system can save, it is important to first investigate the usage from conventional thermostats. In recent years, a continual improvement in minimising heat loss is shown the graph below. In 2008, the total heat loss is at 253.7 W/K. This means that for every degree of temperature difference between inside and outside, over 250 Watts are needed to maintain the desired internal temperature.

Figure 4.1: Energy loss by house insulation

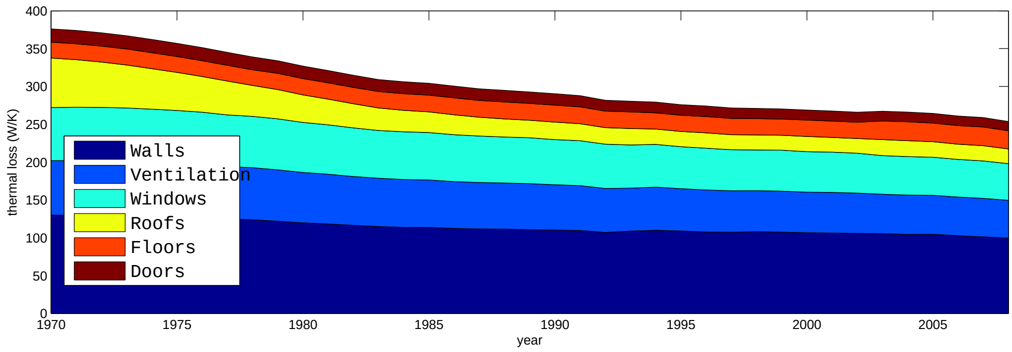
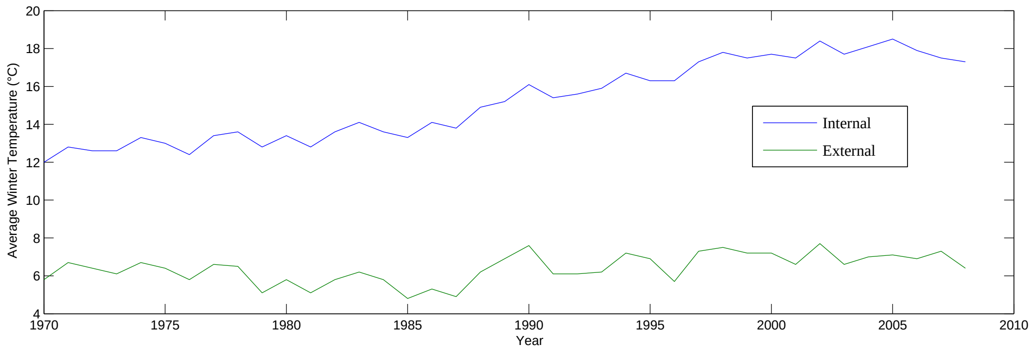


Figure 4.2: Year-on-year winter internal to external temperatures



From this data, an estimation of the power usage can be made from the average winter temperature difference between inside and out. Taking data from 2008, the average temperature difference was 10.9 °C. With a thermal loss of 253.7 W/K, this means a power of 2.77 kW is required to heat the average home. The weighted average fuel price is 6.02p/kWh: meaning it costs 16.7 pence for every hour the central heating is switched on. Assuming that the heating is used for 12 hours a day, this means a daily cost of £2. Considering that the weekly expenditure on lights, heating and power is £18.90, this seems to be a reasonable estimate.

Above are the approximate hourly savings the home automation system can provide the average UK household during winter months per hour. Although these appear small, if the heating was turned down for 7 hours a day by 2 °Cwhilst the occupants are sleeping, and 1 hour or so of unnecessary heating is reduced every day, then the daily

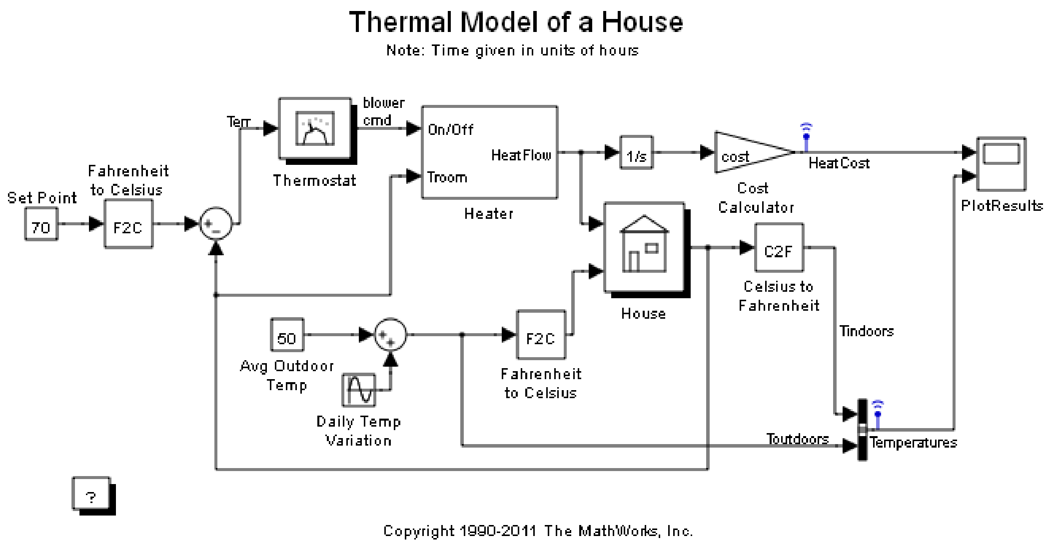
System prediction benefit	Approximate hourly saving(£)
Turning down temperature in sleep state	1.53 p/K
Turn off heating whilst house is unoccupied	16.7 p

saving the system would provide will be around £0.40. This is a reduction in 20%, which over the course of a year could be £130 for the average UK gas bill of £653 on direct debit.

Another way of estimating the amount the system could save is by thermally modelling a house and calculating the energy used for different thermostat settings: one set like a traditional thermostat, which maintains a temperature for 12 hours a day through the week and the entirety of Sunday; and one using simulated occupation data, which a machine learning algorithm could hopefully predict.

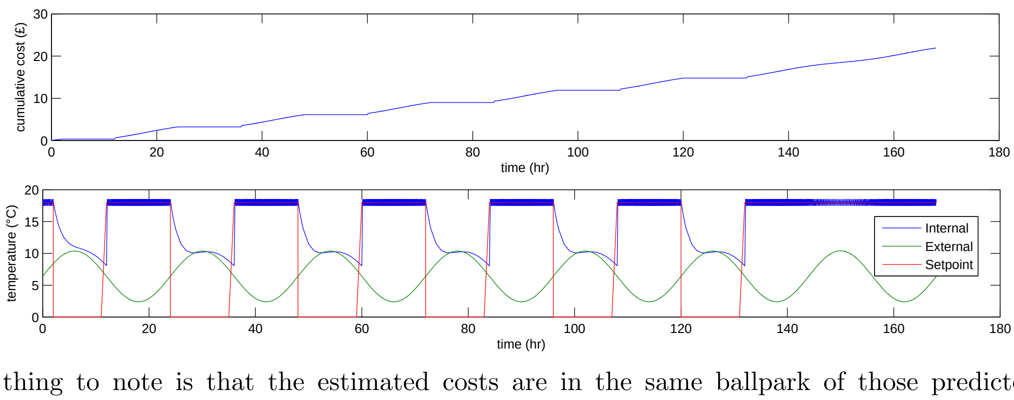
Fortunately, MATLAB has a Simulink model for a house heating system, so making the simulations was reasonably straightforward. The program was modified by firstly matching the program parameters to the thermal resistance, temperature and average p/kWh of a UK home as specified above. Additionally, the constant thermostat setting was replaced by a switchable signal generator giving either the AutoHome or conventional thermostat settings. The schematic for this model is shown in Figure 4.3.

Figure 4.3: The House Heating Simulink Model



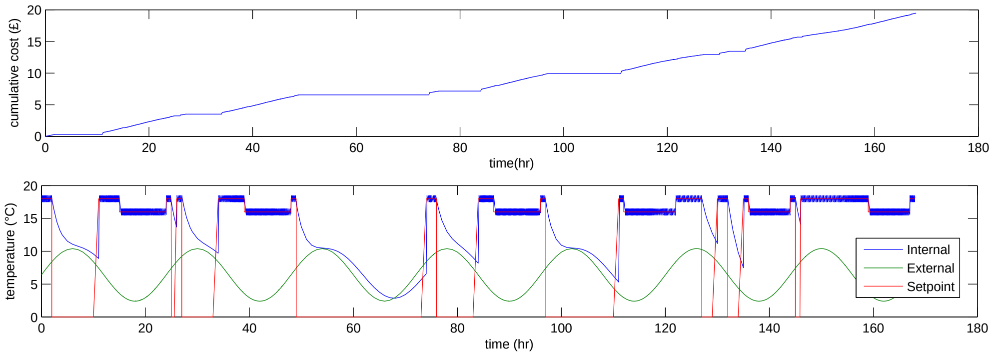
The model works by switching on/off a "blower", which heats 1 kg of air every second to 50 ℃when the internal temperature of the house deviates from the setpoint by more than 0.5 ℃. A feedback loop inside the house block takes the thermal losses from the temperature gradient into account. Integrating the thermal power going into the house gives the energy used, which the cost can be calculated from. A weeks worth of thermostat settings (168 hours) is fed into the system for the conventional and occupancy setpoints, and the results shown in the graphs below. The external temperature, shown as green in the lower plots, oscillates sinusoidally with an amplitude of 4 ℃ around the average winter temperature in 2008 of 6.4 ℃.

Figure 4.4: Simulation result for a week's worth of conventional thermostat program heating cost and temperatures



The first thing to note is that the estimated costs are in the same ballpark of those predicted by simple calculation earlier at £21.90 and £19.48 respectively, which represents a saving of almost £2.50. Although this a quite a modest saving, it is worth revisiting the other aim of the automation system, which is to increase the comfort and convenience of the house's heating. Clearly the occupancy of the house is non uniform unlike the setting of the conventional system, which means there will be times when the house is either unnecessarily heated or at an uncomfortable temperature.

Figure 4.5: Simulation result for a week's worth of predicted occupancy and corresponding thermostat setting cost and temperatures



4.2 Hardware Performance

Each module of the hardware was first developed separately and tested so in order tp fix any issues at their earliest stage. These were then all connected and the programmes combined to form a singular master programme which would control the whole mbed system.

First the digital temperature sensor was set up and programmed as discussed in section 3.1.3 after which initial tests found the temperature to be very low, however with the addition of pull-up resistors on the SCL and SDA pins the temperature was found to be as expected, this was then tested using an advanced handheld temperature sensor and was found to be accurate to ±0.5°C as expected.

With the temperature sensor now completed the humidity sensor could be utilised, as it requires the local temperature in order to calculate the correct humidity from the input voltage which is measured using an ADC port on the mbed. This was then tested by measuring the outside environment with the humidity returned from the weather API, this was found to return similar values though the humidity sensor itself was not particularly stable and varied by several % points. However this would be as expected as the datasheet tells us the ±3%.

With the local system complete, the wifly module was set up and connected to an access point setup using a laptop (as it cannot connect to enterprise enabled wifi). It was also tested using a standard home router which was supplied by the ISP company. After which a websocket connection was set up, this was generally successful but ran into a number of problems.

The first and major problem was that the websocket would close after an unknown period of time for unknown reasons. Furthermore the `is_connected` function failed to report when the websocket would disconnect, the general solution came from the common ping-pong idea. Where a ping was regularly sent from the mbed and the server would then return pong, if this return 'pong' was not detected by the mbed mode it would then re-join the websocket, however this was only partially successful and sometimes failed completely, and so a reset then reboot command was sent to the wifly module after which it would then reconnect quickly with no failures.

4.3 Software Performance

4.3.1 Timetable Prediction

Every three hours[1] the timetable prediction program(described in section 3.2.3) is called to predict what it expects the current day to look like, as well as what it expects (though with lower confidence) the other days of the week. This is shown in Figure 4.6 which shows that the user is allowed to override the estimated timetable for advanced energy savings - however as current systems work purely by such scheduling and are often seen as inefficient this use is discouraged.

This timetable is used if the current state is either `away` or `asleep` and is expected to be `inside` (which also implies awake) in the next 15 minutes[2]. When this is the case the house is heated up to the expected comfortable temperature (determined by the algorithm in section 3.2.4). It will sometimes be the case that the user does not come back when expected. In such a case a half-hour window will be given for the house state to become `inside` (i.e. detect someone is present) before ignoring this timetable and instead waiting for the house state to actually become `inside` to avoid energy waste (at the expense of compromising user comfort).

Testing the real world use of the predictor was a challenge as a lot of data would need to be collected - a feat not possible in the amount of time given. Instead the UMass SMART* dataset[11] was used to test this data and determine the accuracy (which also allowed for finding the best parameters) as seen in Figure 3.9. From this we can see that the expected accuracy is 85% which is a reasonably good performance which however requires a fallback

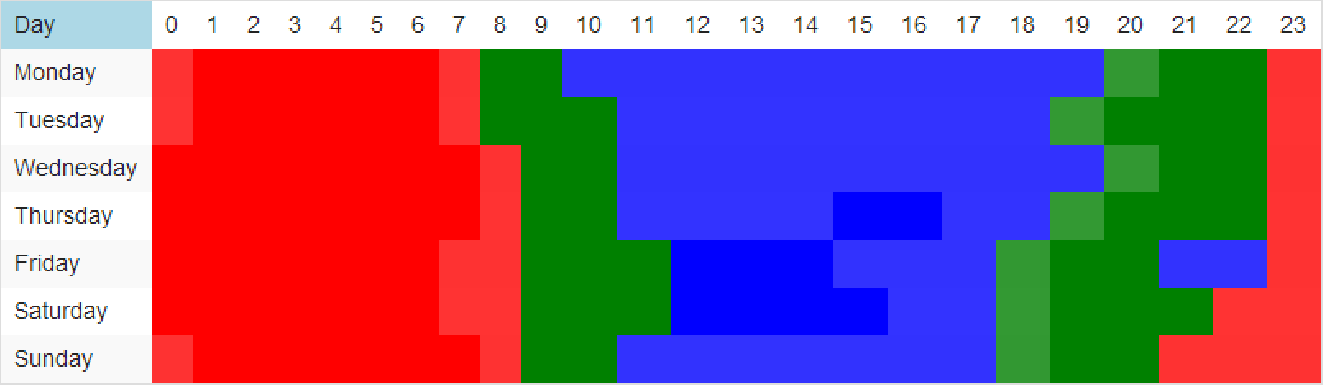
[1]This value can be changed but was chosen to be three hours as the algorithm works best by regressing on 6 hours of past data - however to accommodate for unexpected events and offer more granularity 3 hours was chosen as a good compromise

[2]The certainty - i.e. if the algorithm is 50% sure or 75% sure - can be changed but is set to 50 by default

& override in case of failure - which is described above.

Figure 4.6: Timetable presented on website

Legend:
Red: Asleep - Green: Inside - Blue: Away
Dark: Algorithm Predicted State - Light: User Set State



4.3.2 Website Usage

The website was primarily tested in college using Google Chrome. This is because the site is only accessible internally to college (although it can be connect to elsewhere using the VPN) and Chrome is the only browser on the college computers that is up to date. Using a recent browser is important to the website, as it has regular use of websockets; a protocol only introduced in HTML5, along with a few other features not supported in older browsers. Unfortunately the other browsers on campus are not up to date enough to support websockets.

The first test for each page was to simply check for errors in the console window and address any issues that arose. After this, any major change made to the website was heavily tested for bugs by a handful of people. Any changes that were needed were made, with this process being repeated again for the updated version. The websockets could be tested easily by checking the websocket server log for messages, with the results of anything returned being displayed directly on the website.

Ajax requests could be checked in one of two ways. Firstly it can be checked that they are actually called and receive a response by using the network console window. This lets you see detailed information about any request, such as what data is sent with the request and any data that is returned. Secondly, because the ajax requests all deal with database handling, it is possible to check the relevant tables using mysql directly.

It is possible to check that the weather API script is collecting data correctly by viewing the graphs on the statistics page.

Once the website was fully functional in Chrome, it was also tested in latest versions of Internet Explorer and Firefox. A few small modifications were needed to ensure cross browser compatibility which were easily made. The site was also tested on IE9 mobile, where it failed due to lack of support for websockets, but was otherwise fine and also on safari for iOS, where it worked fully. The site is not however optimised for mobile use, this is where a phone app would come in use.

4.4 System Practicality

The practicality of the system was determined by assessing the ease at which it could be installed into a house. It is important the the entire system can be easily retro fitted to a house. Otherwise, only the relatively small number of new builds could get the benefits the system has to offer, which would render it both economically invalid and removes any real potential to impact the national or worldwide energy consumption. For each of the hardware modules, important considerations are how the component will be installed, maintained, the intrusion caused to the users and the security of it.

The majority of UK households have a gas fired boiler, which is controlled by either bimetallic coil switch in mechanical thermostats or a relay in programmable flavours. For these, only four wires are needed: live, neutral and the two switch wires for the relay. Connecting the main mbed module to this can either be done directly or with a separate boiler relay unit included connected to the wireless ZigBee network. The thermostat then maintains the temperature at a given setpoint by switching on/off this relay. Whether the house has an oil, electric or gas boiler or a ground source heat pump, the thermostat can control a local temperature with ignorance. Installing such a system will be identical to any commercial programmable thermostat, which can be done by any plumber or even by the homeowner with relative ease.

Similarly, with the light switches, the electronics is simple enough to be integrated into a conventional light switch housing, and can be replaced easily. The power supply can be derived off the mains that it is switching.

Furthermore, since the lightswitch is effectively just a wirelessly controllable mains supply with a bulb attached to it, the design can be used in any domestic appliance and embedded into a traditional mains socket or integrated into the appliance itself.

As for the power meter, the same current clip design is used in many of the commercially available 'smart meters' that are designed for the user to simply install themselves due to the safety of not having to touch the mains at any point. Although not implemented on the prototype, a very simple and sensible way to power to unit is through the voltage sense transformer by rectifying, smoothing and applying to the regulation circuit.

Motion sensors need to be in critical places such as main living spaces such as kitchen, bathroom and lounges, but also in bedrooms to determine sleep state. Naturally, the idea of having a wirelessly transmitting motion sensor especially in a bedroom may seem like an invasion of privacy, in reality they are only a passive infrared receiver with a digital output giving a change in radiation due to motion. Therefore, no information apart from the activity of the house, which is what the system is trying to ascertain, can be extrapolated.

4.5 System Unit Cost

Note: A breakdown of costs can be found in the appendix section 8.3.

As can be seen in the table above the final cost of the prototype came to £212.34 (including breadboard) . While this is above our expected costs this does not take into account any benefits from economies of scale, plus the advantages from component switching when working on PCB (as opposed to breakout boards required for breadboard). Furthermore the mbed development board is the most expensive unit for our prototype, however when finally developed only the microprocessor (and some other debugging components) will be required. This would drop the cost of the microprocessor from £41.38 to just £5.51 (for a single unit, lower for multiples) and so already it can be seen that there is significant scope for reductions before the final design.

Section 5

Further Development

There are many ways in which this system could be enhanced, for example adding cameras or microphones to increase the detection rate of human presence. However many of these come at the cost of overcomplicating the device (and its installation) - for example the proposition of adding a camera not only complicates the installation and adds little benefits but also introduces privacy problems (many people would object to the installation of a camera in their home) - ultimately causing the product to be less attractive. Thus further developments must take this into account to genuinely improve the product and make it more attractive and useful rather than loading it with features.

5.1 LCD Screen

5.2 Modular Network Setup

One major improvement to our design would be the addition of a simple, ease of use module setup feature. This would be where different components could be purchased separately by the consumer and then with a simple few button touches would be connected into the network as a fully functional device, this could be done by using some of the abilities of Xbees which allows new Xbees to be easily included into a new network using an API framework.

5.3 Smart Door Lock

This improvement would involve offering a module which can be installed on the main door of the house and compliment a usual key lock by offering a wireless solution. The advantages of such a lock would be the ability to open the main door simply via a digital key adding more possibilities for key control. For example the keys could be given to guests temporarily or only work for the cleaning lady at certain times and deactivated remotely. Digital security would evidently be a high priority of such a device.

Additionally this type of device, while very new, is not innovative in itself as there are already two companies trying to enter this market - Lockitron and August Lock. The real interest in offering this as a module would be near 95-99% certainty in house occupancy[1]. Indeed by combining the house motion sensors with main door traffic flow (which can be determined whether someone has just entered the house or not). The state can be determined by a process similar to:

- **Key used to enter house**: Assume someone is entering house thus presence is inferred
- **Key not used**: Assume that someone is either leaving the house or guests are being let in. Differentiate by:
  No motion detected in house up to 5 minutes after event: Assume house empty (otherwise motion would be detected when moving inside house)
  Motion detected 5 minutes after event: Infer presence in house

While a Hidden Markov Model will still need to be relied upon to clearly differentiate states this method is expected to greatly increase house estimation accuracy while adding a useful module to the system.

[1] Assuming no pets which is another issue to deal with

Section 6

Conclusion

Section 7

References

[1] Jason Palmer, Ian Cooper *Great Britain's Housing Energy Fact File* https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/48195/3224-great-britains-housing-energy-fact-file-2011.pdf

[2] L.D.Shorrock, J.I.Utley *Domestic Energy Fact File 2008* http://www.bre.co.uk/filelibrary/pdf/rpts/fact_file_2008.pdf 2008.

[3] DOE *Annual Report on Fuel Poverty Statistics 2013* https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/199833/Fuel_Poverty_Report_2013_FINALv2.pdf 2013.

[4] MarketsandMarkets *Global Home Automation & Control Systems Market is expected to reach $35,627.83 million by 2016* http://www.marketsandmarkets.com/PressReleases/home-automation-control-systems.asp September 2011.

[5] MarketsandMarkets *Internet of Things (IoT) & Machine-To-Machine (M2M) Communication Market* http://www.marketsandmarkets.com/Market-Reports/internet-of-things-market-573.html September 2012.

[6] Farnell UK *NXP - LPC1768FBD100 - MCU, 32BIT, ARM CORTEX M3, 100LQFP* http://uk.farnell.com/nxp/lpc1768fbd100/mcu-32bit-arm-cortex-m3-100lqfp/dp/1718549MPKG June 2013.

[7] Mike Elgan *Home Tech: There's an Easier, Affordable Future for Home Automation* http://www.houzz.com/ideabooks/4307872/list/Home-Tech--There-s-an-Easier--Affordable-Future-for-Home-Automation September 2012.

[8] HM Government *Helping households to cut their energy bills* https://www.gov.uk/government/policies/helping-households-to-cut-their-energy-bills May 2013.

[9] Department of Energy and Climate Change *Smart Metering Implementation Programme Consultation document* https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/42953/6129-consultation-second-version-smets.pdf August 2012.

[10] Hart, G. W. *Residential energy monitoring and computerized surveillance via utility power flows* http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=31557 1989.

[11] Sean Barker, Aditya Mishra, David Irwin, Emmanuel Cecchet, and Prashant Shenoy *Smart*: An Open Data Set and Tools for Enabling Research in Sustainable Homes* http://lass.cs.umass.edu/papers/pdf/sustkdd-smart.pdf

[12] Department of Energy *Energy Efficiency and Renewable Energy* http://energy.gov/sites/prod/files/energy_savers.pdf 2011.

[13] Washington State University *WSU CASAS Datasets* http://ailab.wsu.edu/casas/datasets/index.html

[14] James Scott, A.J. Bernheim Brush, John Krumm, Brian Meyers, Mike Hazas, Steve Hodges, Nicolas Villar *PreHeat: Controlling Home Heating Using Occupancy Prediction* http://research.microsoft.com/en-us/um/people/jckrumm/publications%202011/preheat-ubicomp2011%20-%20distribute.pdf September 2011.

[15] Carl Edward Rasmussen and Christopher K. I. Williams *Gaussian Processes for Machine Learning* http://www.gaussianprocess.org/gpml/chapters/ 2006.

[16] Jonathan Taylor *Statistics 191: Introduction to Applied Statistics* http://www-stat.stanford.edu/~jtaylo/courses/stats191/notes/simple.pdf January 2009.

[17] Carl Edward Rasmussen, Hannes Nickisch *Documentation for GPML* http://www.gaussianprocess.org/gpml/code/matlab/doc/ January 2013.

[18] Manuel Blum, Martin Riedmiller *Optimization of Gaussian Process Hyperparameters using Rprop* http://ml.informatik.uni-freiburg.de/_media/publications/blumesann2013.pdf 2013.

[19] University of Manchester *Multivariate Normal Distribution* http://www.maths.manchester.ac.uk/~mkt/MT3732%20(MVA)/Notes/MVA_Section3.pdf 2010.

[20] Pedregosa et al. *Scikit-learn: Machine Learning in Python: Gaussian Processes* http://amueller.github.io/modules/gaussian_process.html 2013.

[21] Christopher K. I. Williams, Carl Edward Rasmussen *Gaussian Processes for Regression* http://mlg.eng.cam.ac.uk/pub/pdf/WilRas96.pdf 1996.

[22] Ed Snelson *Tutorial: Gaussian process models* http://mlg.eng.cam.ac.uk/tutorials/06/es.pdf October 2006.

[23] David P. Williams *Assessing Approximations for Gaussian Process Classification* http://people.ee.duke.edu/~lcarin/David1.27.06.pdf January 2006.

[24] ARM NXP *32-bit ARM Cortex-M3 microcontroller Rev. 9 — 10 August 2012 Product data sheet* http://www.nxp.com/documents/data_sheet/LPC1769_68_67_66_65_64_63.pdf August 2012.

[25] Strategy Analytics *A Quarter of Households Worldwide Now Have Wireless Home Networks* http://www.strategyanalytics.com/default.aspx?mod=pressreleaseviewer&a0=5193 April 2012.

Section 8

Appendix

8.1 Hardware

8.1.1 mbed

Main Control Code

```
#include "mbed.h"
#include "TMP175.h"
#include "WiflyInterface.h"
#include "HTTPClient.h"
#include "Websocket.h"
#include "SDFileSystem.h"
#include "picojson.h"
#include "header.h"
#include "xbee.h"
#include "MODSERIAL.h"
#define ATIClength 23 // 23 gives best results


AnalogIn ain(p18);          // Humidity pin
Serial wifi(p13,p14);       // Wifly module pins
DigitalOut rst1(p26);       // Temperature reset pin
DigitalOut WiFlyreset(p30);     //direct reset signal into WiFly for errorhandler
WiflyInterface wifly(p13, p14,p19,p20,"TP-LINK_804327", "C1804327", WPA); //wifly module set-up
SDFileSystem sd(p5, p6, p7, p8, "sd"); // SD card set up

//Tickers set for recurring functions
Ticker sendPower;
Ticker measureMotion;
Ticker lightstatus;
Ticker dataSend;

TMP175 mySensor(p28,p27);   // Temperature set up
Websocket wsoc("ws://ee-ug1.ee.ic.ac.uk:8888/ws"); //websocket address
picojson::value v;  //JSON parsing variable
XBEE myxbee;    //Xbee function
MODSERIAL pc(USBTX, USBRX); //Debugging using serial terminal
MODSERIAL xbee(p9,p10);     //Modserial allows increased buffer size for xbee
DigitalOut led(LED1);
DigitalOut flash(LED4);

//initialise variables
char response[ATIClength];
char orderedresponse[ATIClength];
char str[512];
char recv[512];
float humidity;
float Temp;
float thermostat = 15;
int pCurrent=0;
int pPhase=0;
int pVoltage=0;
int motionCount =0;
int bufferpos= 0;
int a=0;
//64 bit addresses of XBEEs
char MR0address[8]= {0x00,0x13,0xA2,0x00,0x40,0x8B,0x71,0x55};//motion sensor XBEE
char MR2address[8]= {0x00,0x13,0xA2,0x00,0x40,0x8B,0x70,0x79};//power monitoring XBEE
char MR3address[8]= {0x00,0x13,0xA2,0x00,0x40,0x8B,0x71,0x45};//Light control XBEE
//extern char MR1address[8];
extern char * test;
extern char * MR1address;//pointer to array which holds the currently selected address.
bool lightStatus=false;


void swapaddress(char address[])
{
/* used to swap xbee address as required for communication with the
```

```
    different xbees */

    MR1address=&address[0];

}

void readATICresponseISR(MODSERIAL_IRQ_INFO *q)
{

    MODSERIAL * serial= q->serial;


    if(serial->readable())
        response[bufferpos]=serial->rxGetLastChar();//read in char but dont delete from circular buffer

    bufferpos++;
    pc.printf("IRQ REPLAY %d IS %2X LONG\n",bufferpos, response[bufferpos]);
    if(bufferpos == ATIClength) {
        bufferpos = 0;
    }
    if(serial->rxBufferFull()==true) {
        serial->rxBufferFlush();//flush the buffer once it is full.
        if(serial->rxBufferEmpty()==true)
            pc.printf("BUFFER FLUSHED\n");
        bufferpos=0;
    }

}

void readATICresponseISR2()
{

    int pos;
    if(xbee.readable()) {
        response[bufferpos]=xbee.getc();//clear bytes from FIFO
    }

    bufferpos++;
    pc.printf(" %d IS %2X LONG\n",bufferpos, response[bufferpos]);

    if(bufferpos == ATIClength-1) {
        bufferpos = 0;
        pos=myxbee.lastpacketpos(response,ATIClength);


        pc.printf("packet starts here at byte %d pos\n the packet delimiter is %2X \n",pos+1,response[pos]);
        pc.printf("DIGITAL HEX VARS ARE %2X and %2X\n",response[ATIClength-4],response[ATIClength-3]);//extract the
     digital values from sent packet

    }

}

void lightCheck()
{
    //function to call xbee light check function and to send the status to the sever

    __disable_irq();
    myxbee.SENDLIGHTSTATUSREQ();
    //sprintf(str, "{\"type\":\"light_request_response\" , \"light_ID\":1 , \"status\": %d }", lightStatus);
    //wsoc.send(str);
    __enable_irq();
}

void pingpong()
{
/*  ping pong function used to detect whether websockets are still running
    and then reset and reboot wifly module then reconnect to websocket */


    __disable_irq();

    pc.printf("is connected %d\n",wifly.is_connected());
    wsoc.send("{\"type\":\"ping\"}");   //send ping tp sever
    wait_ms(100);
    //if server responds then websockets are working so return//
    if(wsoc.read(recv)) {
        pc.printf("%s\n",recv);
        //if(strcmp("{\"type\":\"pong\"}",recv) == 0)
        __enable_irq();
        return;
    }
    //otherwise reset and reboot wifly module
    WiFlyreset=0;
    pc.printf("RESETTING WIFLY\n");
    wait_ms(200);//make an active low pulse
    WiFlyreset=1;
    wait_ms(100);//give time to reset
    wifly.send("reboot",6); //send reboot cmd to wifly module
    wait_ms(200);
    wifly.init(); //Use DHCP
    wait(0.5);
    wifly.connect();  //connect to wifi
```

```
    wait(0.5);

    //websocket reconnect

    if(wsoc.connect() == 1) {
        pc.printf("This was a truimph!");
    } else {
        pc.printf("failed!");
        // pingpong();
    }
    __enable_irq();
    return;

}


int main()
{
    //set baud rate
    pc.baud(115200);
    xbee.baud(115200);

    //wifly reset at start
    __disable_irq();
    WiFlyreset=1;
    wait(2);
    WiFlyreset=0;
    wait(2);
    WiFlyreset=1;

    //set xbee addresses
    MR1address=&MR2address[0];


    pc.printf("Test Wifly!\r\n");


    mySensor.vSetConfigurationTMP175(SHUTDOWN_MODE_OFF|COMPARATOR_MODE|POLARITY_0|FAULT_QUEUE_6|RESOLUTION_9,0x48); //
      Temperature set-up
    wait_ms(400);
    wifly.reboot(); //reboot wifly
    wait(1.0);
    wifly.init(); //Use DHCP
    wait(1.0);
    wifly.connect();  //connect to wifi
    wait(1.0);
    wsoc.connect(); //connect to websocket
    __enable_irq();


    sendPower.attach(&myxbee,&XBEE::SENDPOWERREQ, 5);
    //ticker polling power sensor for data
    wait_ms(1000);

    measureMotion.attach(&myxbee,&XBEE::SENDMOTIONREQ,3.0);
    //ticker polling motion sensor
    wait_ms(1000);

    lightstatus.attach(&lightCheck,3);
    //ticker polling whether the light is on or off

    dataSend.attach(&senddata,17);
    //send power, humidity, temperature and motion data to server

    while (1) {


        //***************************Temp sensor*****************************************//

        Temp=mySensor; //get temperature in variable 'temp'

        //*************************Humidity Sensor***************************************//

        humidity = ain.read();
        humidity = (((humidity*5)/3.3)-0.1515)/0.00636;
        humidity = humidity/(1.0546-(0.00216*Temp));



        //**********************recieve section***************************************//

        recv[0] = '\0';         //set recieve string to 0
        wait_ms(1500);

        __disable_irq();
        if(wsoc.read(recv)) {   //if communication recieve pass data to recieve function

            pc.printf("%s\r\n", recv);
            recievedata();

        }
        else {
            //otherwise check websocket is still running
```

```
            pingpong();
        }
        __enable_irq();


        //end of continious while loop
    }

}
```

code/homeauto–main.cpp

```
#ifndef header_h
#define header_h
#include "Websocket.h"

#include "picojson.h"
#include "mbed.h"
#include "xbee.h"






extern Websocket wsoc; //websocket address
extern picojson::value v;

extern char str[512];
extern char recv[512];
extern float humidity;
extern float Temp;
extern float power;

extern float thermostat;
extern XBEE myxbee;

void lightsignal();
void senddata();
void thermoupdate();
void recievedata();
void powerconv();

#endif
```

code/homeauto–header.h

```
#include "header.h"
#define PI 3.1415926

extern void swapaddress(char address[]);

extern char MR0address[8];
extern char MR2address[8];
extern bool lightStatus;

float PCurrent;
float PVoltage;
float PPhase;
float power;


void senddata()
{
/* function sends temperature, humidity, power and motion data back to the server at regular intervals */
    __disable_irq();
    powerconv();    //function to work out correct power measurement from calibration data

    sprintf(str, " {\"type\":\"house_measurement\" , \"Temperature\":%f, \"Humidity\":%f, \"power\":%f, \"Motion\":%f }", Temp, humidity, power, motionCount); //place in string
    //data sent after stored in string
    wsoc.send(str);
    __enable_irq();
}

/*
float cosine_func(float x){

    x = 1- 0.5*x*x+ (1/24)*x*x*x*x - (1/720)*x*x*x*x*x*x;
    return x;
}
*/

void powerconv()
{
/* function uses previously collected power data to calibrate the sensor correctly
    first working out the voltage, current and phase and then real power
    using known power equations */

    PVoltage = 0.3535*pVoltage - 2.6009;
    PCurrent = 0.0022*pCurrent - 0.0065;
```

```
    PPhase = (PI/1024)*pPhase;
    power = PVoltage*PCurrent*cos(PPhase);
}


void lightsignal()
{
/* function to turn on and off the lights, using the JSON string parsing to
    find which light has been selected to turn off */

    int lightid = (int)v.get("light_ID").get<double>();
    //string parsing to get light id
    swapaddress(MR3address);
    //select correct xbee address

    /*switch function selects correct light and then using the "status"
    variable from the JSON string sets light to on or off */
    switch( lightid ) {
        case 1:
            if(v.get("status").get<double>() == 1) {
                //call xbee and turn light on
                myxbee.SENDLIGHTONREQMOD();

            } else {
                //call xbbe and turn light off
                myxbee.SENDLIGHTOFFREQMOD();
            }
            break;
        case 2:
            if(v.get("status").get<double>() == 1) {
                //call xbee and turn light on
            } else {
                //call xbbe and turn light off
            }
            break;
        default:
            break;
    }
}


void thermoupdate()
{
/* updates thermostat setting to correct variable from website */
    thermostat = v.get("setting").get<double>();
}


void recievedata()
{
/*  Function that is called when data is recieved from the websocket,
    the string is then copied to a pointer type and parsed (using
    JSON functions) to determine the type and then calls other
    functions depending on the type
*/

    char * json = (char*) malloc(strlen(recv)+1);
    strcpy(json, recv);

    string err = picojson::parse(v, json, json + strlen(json));
    //error function

    /*move the 'type' in JSON string into a string type which is then compared to
    expected types from the server, after which actions are taken accordingly */
    char  type[200] ;
    strcpy(type,v.get("type").get<string>().c_str());
    printf("%s",type);
    if( strcmp(type, "light_request") == 0) {
        //request whether light is on or off
        sprintf(str, "{\"type\":\"light_request_response\" , \"light_ID\":1 , \"status\": %d }", lightStatus);
        wsoc.send(str);
    } else if( strcmp(type, "light_control") == 0) {
        //turn on or off light
        lightsignal();
    } else if( strcmp(type, "thermostat_request") == 0) {
        //request temperature setting of thermostat, which is returned, alond with house temperature
        sprintf(str, "{\"type\":\"thermostat_request_response\" , \"temperature\":%f , \"setting\": %f }", Temp,
 thermostat);
        wsoc.send(str);
    } else if( strcmp(type, "thermostat_control") == 0) {
        //thermostat setting has been updated from website
        thermoupdate();
    }

    return;

}
```

code/homeauto–functions.cpp

XBee

API Framework

Figure 8.1: XBee and XBee PRO wireless specifications

Specification	XBee ZNet 2.5	XBee PRO ZNet 2.5
Performance		
Indoor/Urban Range	up to 133 ft. (40 m)	up to 300 ft. (100 m)
Outdoor RF line-of-sight Range	up to 400 ft. (120 m)	up to 1 mile (1.6 km)
Transmit Power Output	2mW (+3dBm), boost mode enabled 1.25mW (+1dBm), boost mode disabled	50mW (+17dBm) 10mW (+10 dBm) for International variant
RF Data Rate	250,000 bps	250,000 bps
Serial Interface Data Rate (software selectable)	1200 - 230400 bps (non-standard baud rates also supported)	1200 - 230400 bps (non-standard baud rates also supported)
Receiver Sensitivity	-96 dBm, boost mode enabled -95 dBm, boost mode disabled	-102 dBm

The API framework consisted of 3 main sections. First is a function specific frame creator which allows an explicitly specified frame to be transmitted to the UART on the XBee. These frame creator functions are able to calculate their own checksums and account for different addresses as well. This allowed the specific frame requests to be transmitted at request. Next a function called `readreply()` was created in order to capture the reply from the XBees and store it in an array for that function. In most cases the reply came almost immediately, thus there was no need to wait at all for the reply. There are different variants of readreply which show the reply bytes if needed. Last is the `readpacket()` function which decoded the reply depending on what it is a reply to. This was achieved by scanning the incoming data and picking out a packet from the reply then looking at the frame specifier type which allowed us to see what kind of response it was. Once this had been done, it was possible to then decode the data from those packets in each case. As mentioned earlier the analogue reading which is a remote AT IS command response has 2 frame ids returned depending on which function created it. Sensor Interfaces.

Our sensors use a variety of these features.

The Motion sensors use the analogue inputs. The API frame for measuring the value on the ADC inputs is sent in the `SENDSENSORREQ()` function. This function also reads the reply and decodes the appropriate bits to get the analogue value back. The ADC input only goes from 0 to 1.2V though so the levels it can read are different from the motion sensor output which is VDD-0.5 where VDD is 3.3V. Thus we used a potential divider to scale the voltage down around 0.8V for motion and 0.5V for no motion. This allows us to see when motion has been detected.

Light

In order to control the light switches with the XBee, we had to implement extra digital logic. This was controlled from two sides, where the physical light switch was one and the digital output from the XBee was the other. In order to service the web socket requests the digital IO functions for the XBee were called by using either `SENDLIGHTONREQ()` or `SENDLIGHTOFFREQ()`. These functions either pulled the pin high or low depending on which one was called. This gave the other input to the XOR gate which allows the triac circuit to turn the light on or off accordingly. The other job the XBee has is to measure the current output of the XOR gate in order to get the current status of the light. This is needed in order to update the graphic on the website relating to the relevant light toggle switch. In the end two ways were attempted to accomplish this.

The first was a polling based method which we ended up using; this method uses a similar analogue function as the motion sensor readings used. This was done in order to save a bit of time when writing the code, as the speed of detection was not an issue. When the function `SENDLIGHTSTATUS()` is called the XBee calls for an analogue reading which is a scaled down version of the digital level on the XOR gate. As well as giving an analogue reading a different frame ID is returned in order to differentiate between motion sensor readings and light switch readings. These are then loaded into global variables and passed via web socket. Our second method was to use UART interrupts along with the XBee pin change detection feature. The XBee can detect rising and falling edges on its monitored digital inputs. When an edge occurs the corresponding API frame is transmitted. The interrupts on the UART allow the MBED to listen for this reply which could come in at any time. Unfortunately the result turned out to be buggy as the modified serial library with a circular buffer had strange behaviour. getc() would not read the characters in the correct order, whilst using a function called `rxGetLastChar()` the operation worked fine until the buffer filled up. After flushing this buffer the UART only read in 5 or 6 bytes at a time. The standard serial library worked fine with the interrupts on the UART however not all of the XBee frames can be captured correctly as there is only a 16 byte buffer on the FIFO. Thus this idea was not used in the end. Since we chose to go down the polling route, there is a 3 second lag on how long it takes for the graphic on the webpage to update.

Power

The power sensor interfaced slightly differently to the other sensors. As there is a ATmega microcontroller on the sensor circuit it was decided that transferring the data as 16 bit short integers over UART would be most efficient. This was the case as the ADCs in the XBee are only 10 bit and would introduce more error into the measurements. Due to the serial interrupts not working as expected a workaround was devised, as it was not possible to just send a raw UART frame we leveraged the ATmega controller to help us. The power readings are taken on a regular poll using a standard timer in the MBED like every other function. When this timer activates the digital output on the power XBee is pulled high. This causes an interrupt to occur on the rising edge of the signal which causes the ATmega controller to explicitly send a new API frame containing raw data with the numbers to the co-ordinator XBee. A small wait happens in between these two functions in order to make sure that the circular buffer is filled with the UART data. After receiving and decoding the data, a request to set the XBee digital pin low is sent, and the sensor will be ready for another reading the next time `SENDPOWERREQ()` is called.

Issues with interrupts

At the beginning the ideal vision of how we would have handled XBee communication would have been to use ISRs to service requests as needed. For simplicities sake due to timing we decided that a polling based method would be more robust and just as viable. In order to use an interrupt based approach we would have to have had full interrupt capability on all of the UART ports so that ISRs could be activated upon the required JSON strings entering the system. Taking a poll based approach we used the timers on the Cortex-M3 in order to activate interrupts at specified time intervals. One problem with this was the fact that whilst one ISR was running, it could not be interrupted at all, so nested interrupts were not possible with this approach. If the ISR was interrupted again whilst sending out an API frame to the UART it would simply be discarded as invalid data. If an interrupt occured when the XBee received a frame, that data would then have been missed as well and the whole thing would have had to have been done again. Thus all interrupts were disabled globally for the duration of the XBee ISRs. This was done with the function calls `__disable_irq()` and `__enable_irq`. This ultimately limits the amount of processing that can occur on the CPU core as the ISRs are effectively blocking functions, and thus nothing else can run whilst they do. In the case of the power routine, there is a 200ms wait in there which squanders a lot of cycles however it has to be there in order to capture the response, thus in this instance an interruptable UART would have been superior. With polling comes inevitable latency between polls as well which can be several seconds. This is most noticeable when polling for the light context at which point the true context isn't updated for up to 3 seconds if the most recent poll has just passed. This causes a slight glitchyness on the web interface, however after the next poll it would correct itself.

Code

```
#include "mbed.h"
#include <vector>// alternate way of getting array length as standard wasnt working.
#include "MODSERIAL.h"
#include "xbee.h"


#define ATIDlength 50
#define ATISLENGTH 18
#define ATresponselength 9
#define MAXMESSAGELENGTH 100

DigitalOut myled(LED1);

extern MODSERIAL pc;
extern MODSERIAL xbee;


char ATIDplaceholder[ATIDlength];
//char BOSSaddress[8]= {0x00,0x13,0xA2,0x00,0x40,0x8B,0x71,0xDE};
//char MR1address[8]= {0x00,0x13,0xA2,0x00,0x40,0x8B,0x71,0x55};
char * MR1address;
char * test;
vector <char> comvec;
int hextoint(char MSB, char LSB)
{

    int total=0;

    total=(int)MSB*256+(int)LSB;

    return total;

}

void XBEE::swapaddress(char address[]){
//address pointed to becomes the one passed in argument.
MR1address=&address[0];

}

```

```
char * XBEE::findcurrentaddress(){

return MR1address;

}


char XBEE::checksum(char hexchar[])
{   //calculate checksume.
    int length;
    int calculatedchecksum;
    int total=0;

    length = (int)hexchar[1]+(int)hexchar[2];

    for(int i=3; i<3+length; i++) {//ignore delimiter and length bytes.
        total=total+(int)hexchar[i];
//pc.printf("%2x\n",hexchar[i]);
    }
    //pc.printf("%d\n",total);

    calculatedchecksum=0xFF-(total & 0xFF);
    //pc.printf("%2x\n",calculatedchecksum);


    return calculatedchecksum;
}


int XBEE::lastpacketpos(char packet[], int arraylength)//find last packet position in big reply string
{   //improvement -- detect all packet positions not just last one.

    int currentlastpos=0;

    //pc.printf("array length is...%df\n",arraylength);

    for(int i=0; i<arraylength; i++) { //static array, so valid method.

        if(packet[i]==0x7E)
            currentlastpos=i;
    }

    return currentlastpos;

}

void XBEE::readpacket(char packetstring[],int packetpos,int packetstringlength)
{

    if(packetstringlength != 0) {
        char snip[128];//makes a copy of current packet, a preparation for detecting multiple packets in returned string.
        enum packettype {ATcommandresponse,LocalATcommand,Modemstatus,Txreply,RemoteATcommandresponse,RemoteTxreply};
        enum packettype packetstatus;
        enum ATcommand {ATIS,ATID};//supported AT commands
        enum ATcommand command;
        int bytes =packetstring[packetpos+1]*256+packetstring[packetpos+2];
        //pc.printf("number of bytes in packet is %d\n",bytes);
        char status = packetstring[packetpos+3];//get frame specifier
        pc.printf("status is %2x\n",status);
        for(int j=0; j<bytes+2+1; j++) { //make a copy of current packet
            snip[j]=packetstring[packetpos+j];
        }

        char calculatedchecksum=checksum(snip);
        //pc.printf("calculated checksum is %2x\n",calculatedchecksum);
        //pc.printf("given checksum is %2x\n",packetstring[packetpos+bytes+2+1]);
        bool valid;


        if(calculatedchecksum==packetstring[packetpos+bytes+2+1]) {
            switch (status) {//assign meaningful specifiers

                case 0x88:
                    packetstatus=ATcommandresponse;

                    pc.printf("AT command response packet detected\n");
                    break;
                case 0x8A:
                    packetstatus=Modemstatus;
                    break;
                case 0x08:
                    packetstatus=LocalATcommand;
                    pc.printf("LOCAL AT command response packet detected\n");
                    break;
                case 0x8B:
                    packetstatus=Txreply;
                    pc.printf("TX response packet detected\n");
                    if(packetstring[packetpos+2+5]==0x00)
                        pc.printf("packet was sent sucessfully\n");
                    else
                        pc.printf("Something went wrong\n");
                    break;
```

```
                case 0x97:
                    packetstatus=RemoteATcommandresponse;

                    pc.printf("AT command equivalent is %c%c\n",packetstring[packetpos+2+2+8+2+1],packetstring[packetpos+2+2+8+2+2]);//2/2/8/2===length offset/cmd type and frameID/64bit address/16 bit address
                    char pt1,pt2;
                    pt1=packetstring[packetpos+2+2+8+2+1];
                    pt2=packetstring[packetpos+2+2+8+2+2];
                    if(pt1=='I' && pt2=='S') {
                        command=ATIS;
                    }

                    break;
                case 0x90:
                    int messagestartpos =0;
                    int datapos=0;
                    packetstatus=RemoteTxreply;
                    pc.printf("REMOTE TX PACKET DETECTED");
                    messagestartpos=packetpos+2+2+8+2+1;
                    pc.printf("char %c%c%c \n", packetstring[messagestartpos],packetstring[messagestartpos+1],packetstring[messagestartpos+2]);
                    if(packetstring[messagestartpos]=='p' && packetstring[messagestartpos+1]=='w' && packetstring[messagestartpos+2]=='r') {//condition for power and not just a standard message

                        datapos=messagestartpos+3;
                        pc.printf("VCP is %2x%2x%2x%2x%2x%2x\n",packetstring[datapos],packetstring[datapos+1],packetstring[datapos+2],packetstring[datapos+3],packetstring[datapos+4],packetstring[datapos+5]);
                        //get raw values from power uC.
                        pVoltage=(int)packetstring[datapos]*256+(int)packetstring[datapos+1];
                        pCurrent=(int)packetstring[datapos+2]*256+(int)packetstring[datapos+3];
                        pPhase=(int)packetstring[datapos+4]*256+(int)packetstring[datapos+5];

                    } else {
                    //IMPROVEMENT -- implment message printer
                        pc.printf("standard message\n");

                    }

                    break;


            }

            if(command==ATIS) {
                pc.printf("SENSOR READING REQUESTED\n");
                pc.printf("%2x%2x\n",packetstring[packetpos+2+bytes-1],packetstring[packetpos+2+bytes]);
                int num=(packetstring[packetpos+2+bytes-1]*256)+packetstring[packetpos+2+bytes];
                float voltage=0;
                voltage=(float)num/1023*1.2;
                pc.printf("rawis %d\n",num);
                pc.printf("voltage is %fV\n",voltage);
                if(voltage<0.72&&packetstring[packetpos+2+2]==0x01/*gives us Frame ID*/){
                    pc.printf("movement detected\n");
                    motionCount++;
                    }
                else if(voltage >=0.72 && packetstring[packetpos+2+2]==0x01){//no motion detected if voltage is 0.8V
                    pc.printf("NADA\n");
                    }
                 else if(voltage < 0.5 && packetstring[packetpos+2+2]==0x02){
                 pc.printf("Light is off\n");//if frame ID is 2 then it must have been a light packet.
                 lightStatus=false;


                 }
                 else if(voltage>= 0.5 && packetstring[packetpos+2+2]==0x02){
                 pc.printf("Light is on \n");
                 lightStatus=true;
                 }

        }

        else {
            pc.printf("the packet is messed up in some way\n");
        }

    }

}
}

char * XBEE::createmessagepacket(char message[],char address[])
{

    static char packet[MAXMESSAGELENGTH +10];
    short totallength=0;
short length = 0;
    short setuplength = 17;//number of bytes required for address+options etc.
    while(message[length]!='\0') {
        length++;
    }

    pc.printf("length of input message data is %hd\n",length);

```

```
    totallength=length+setuplength;
    pc.printf("number of bytes in this packet is%hd\n",totallength);

    packet[0]=0x7E;
    packet[1]=(0xFF00 & (totallength-3));//-3 for frame delimiters and len MSB/LSB
    packet[2]=(0x00FF & (totallength-3));
    packet[3]=0x10;
    packet[4]=0x01;
    packet[5]=address[0];
    packet[6]=address[1];
    packet[7]=address[2];
    packet[8]=address[3];
    packet[9]=address[4];
    packet[10]=address[5];
    packet[11]=address[6];
    packet[12]=address[7];
    packet[13]=0xFF;
    packet[14]=0xFE;
    packet[15]=0x00;
    packet[16]=0x00;
    for(int i=0; i<length; i++) {
        packet[setuplength+i]=message[i];
        pc.printf("%c wass added to message\n",packet[setuplength+i]);
    }
    packet[setuplength+length]=checksum(packet);

    for(int i=0; i<totallength+1; i++) { //+1 to get chesum output.

        pc.printf("created packet byte %d is %2x\n",i+1,packet[i]);

    }

    return packet;

}

void XBEE::readreply(char reply[], int &replylength)//if/for implementation, doesnt work well unless reading something
      very specific.
{
    replylength =0;
    if(xbee.readable()) {

        for(int i=0; i<18; i++) {
            reply[i]=xbee.getc();
//pc.printf("reply byte %d iz %2x\n",replylength,reply[i]);

            replylength++;
        }

    }

}

void XBEE::readreply2(char reply[], int &replylength)//shows bytes read in.
{
    replylength =0;
    while(xbee.readable()) {


        reply[replylength]=xbee.getc();
        pc.printf("reply byte %d iz %2x\n",replylength,reply[replylength]);

        replylength++;



    }


    pc.printf("Length of reply is %d\n",replylength);
}


void XBEE::readreply3(char reply[], int &replylength)//no bytes shown, so completes a lot quicker as printf is bad for
      interrupts.
{
    replylength =0;
    while(xbee.readable()) {


        reply[replylength]=xbee.getc();

        replylength++;



    }




}
```

```
char * XBEE::createsensorpacket(char address[])
{

    static char packet[ATISLENGTH];
    short totallength=18;//number of bytes required for address+options etc.



    //pc.printf("number of bytes in this packet is%hd\n",totallength);

    packet[0]=0x7E;
    packet[1]=(0xFF00 & (totallength-3));//-3 for frame delimiters and len MSB/LSB
    packet[2]=(0x00FF & (totallength-3));
    packet[3]=0x17;
    packet[4]=0x01;
    packet[5]=address[0];
    packet[6]=address[1];
    packet[7]=address[2];
    packet[8]=address[3];
    packet[9]=address[4];
    packet[10]=address[5];
    packet[11]=address[6];
    packet[12]=address[7];
    packet[13]=0xFF;
    packet[14]=0xFE;
    packet[15]=0x00;
    packet[16]=0x49;//I
    packet[17]=0x53;//S
    packet[totallength]=checksum(packet);



    return packet;

}


char * XBEE::createlightStatuspacket(char address[]) //could be altered to void if needed
{


    static char packet[ATISLENGTH];
    short totallength=18;//number of bytes required for address+options etc.



    //pc.printf("number of bytes in this packet is%hd\n",totallength);

    packet[0]=0x7E;
    packet[1]=(0xFF00 & (totallength-3));//-3 for frame delimiters and len MSB/LSB
    packet[2]=(0x00FF & (totallength-3));
    packet[3]=0x17;
    packet[4]=0x02;
    packet[5]=address[0];
    packet[6]=address[1];
    packet[7]=address[2];
    packet[8]=address[3];
    packet[9]=address[4];
    packet[10]=address[5];
    packet[11]=address[6];
    packet[12]=address[7];
    packet[13]=0xFF;
    packet[14]=0xFE;
    packet[15]=0x00;
    packet[16]=0x49;//I
    packet[17]=0x53;//S
    packet[totallength]=checksum(packet);



    return packet;

}


char * XBEE::createlightonpacket(char address[]) //could be altered to void if needed
{

    static char packet[ATISLENGTH];
    short totallength=19;//number of bytes required for address+options etc.



    //pc.printf("number of bytes in this packet is%hd\n",totallength);

    packet[0]=0x7E;
    packet[1]=(0xFF00 & (totallength-3));//-3 for frame delimiters and len MSB/LSB
    packet[2]=(0x00FF & (totallength-3));
    packet[3]=0x17;
    packet[4]=0x01;
```

```
    packet[5]=address[0];
    packet[6]=address[1];
    packet[7]=address[2];
    packet[8]=address[3];
    packet[9]=address[4];
    packet[10]=address[5];
    packet[11]=address[6];
    packet[12]=address[7];
    packet[13]=0xFF;
    packet[14]=0xFE;
    packet[15]=0x02;
    packet[16]=0x44;
    packet[17]=0x30;
    packet[18]=0x05;//make output high
    packet[totallength]=checksum(packet);



    return packet;

}

char * XBEE::createlightoffpacket(char address[]) //could be altered to void if needed
{

    static char packet[ATISLENGTH];
    short totallength=19;//number of bytes required for address+options etc.



   // pc.printf("number of bytes in this packet is%hd\n",totallength);

    packet[0]=0x7E;
    packet[1]=(0xFF00 & (totallength-3));//-3 for frame delimiters and len MSB/LSB
    packet[2]=(0x00FF & (totallength-3));
    packet[3]=0x17;
    packet[4]=0x01;
    packet[5]=address[0];
    packet[6]=address[1];
    packet[7]=address[2];
    packet[8]=address[3];
    packet[9]=address[4];
    packet[10]=address[5];
    packet[11]=address[6];
    packet[12]=address[7];
    packet[13]=0xFF;
    packet[14]=0xFE;
    packet[15]=0x02;
    packet[16]=0x44;
    packet[17]=0x30;
    packet[18]=0x04;//make output low
    packet[totallength]=checksum(packet);



    return packet;

}

void XBEE::readaddress(char address[]){


pc.printf("address is %2x%2x%2x%2x%2x%2x%2x%2x \n",address[0],address[1],address[2],address[3],address[4],address[5],address[6],address[7]);

}//printf current address pointer



void XBEE::SENDMESSAGE()
{

    int length=0;
    int replylength=0;
    char cstr[MAXMESSAGELENGTH];
    char reply[MAXMESSAGELENGTH];
    char * msgptr;
    pc.printf("entermessage\n");
    pc.scanf("%s",cstr);
    while(cstr[length]!='\0') {//find length of input string
        length++;
    }



    pc.printf("length of input is %d\n",length);

    length=length+18;

    msgptr=createmessagepacket(cstr,MR1address);
    pc.printf("message packet created\n");

    if(xbee.writeable()) {//send packet down UART
```

```
        for(int i=0; i<length; i++) {

            xbee.putc(*(msgptr+i));
            pc.printf("contents of message packet byte %d is %2x\n", i,*(msgptr+i));

        }
    }
    readreply2(reply,replylength);
    readpacket(reply,lastpacketpos(reply,replylength),replylength);

    for(int i=0; i<replylength; i++) {
        pc.printf("reply is %2x and %d\n",reply[i],replylength);
    }


}


void XBEE::SENDSENSORREQ()
{
    __disable_irq();
    int length=19;
    int replylength=0;
    char reply[MAXMESSAGELENGTH];
    char * msgptr;
    char * temp=findcurrentaddress();





    msgptr=createsensorpacket(MR1address);
    pc.printf("SENSOR packet created\n");


    if(xbee.writeable()) {

        for(int i=0; i<length; i++) {
            //pc.printf("contents of message packet byte %d is %2x\n", i,*(msgptr+i));

            xbee.putc(*(msgptr+i));

        }
    }
    readreply2(reply,replylength);
    readpacket(reply,lastpacketpos(reply,replylength),replylength);



        swapaddress(temp);
        __enable_irq();
}

void XBEE::SENDMOTIONREQ(){
__disable_irq();
char * temp=findcurrentaddress();
swapaddress(MR0address);
SENDSENSORREQ();
swapaddress(temp);
__enable_irq();
}
void XBEE::SENDLIGHTONREQ()
{


    readaddress(MR1address);
    int length=20;
    int replylength=0;
    char reply[MAXMESSAGELENGTH];
    char * msgptr;



    msgptr=createlightonpacket(MR1address);
    //pc.printf("message packet created\n");


    if(xbee.writeable()) {

        for(int i=0; i<length; i++) {
//pc.printf("contents of message packet byte %d is %2x\n", i,*(msgptr+i));

            xbee.putc(*(msgptr+i));

        }
    }

}

void XBEE::SENDLIGHTOFFREQ()
{

```

```


    int length=20;
    int replylength=0;
    char reply[MAXMESSAGELENGTH];
    char * msgptr;



    msgptr=createlightoffpacket(MR1address);
    pc.printf("message packet created\n");


    if(xbee.writeable()) {

        for(int i=0; i<length; i++) {
//pc.printf("contents of message packet byte %d is %2x\n", i,*(msgptr+i));

            xbee.putc(*(msgptr+i));

        }
    }
readreply2(reply,replylength);
readpacket(reply,lastpacketpos(reply,replylength),replylength);




}


void xbeeATIDCHANGE()
{

    char PANID[2];
    short hex;

    char hexchar[10]= {0x7E,0x00,0x06,0x08,0x01,0x49,0x44,0x0E,0xE4,0x79};

    pc.printf("what do you want the PANID to be (in hex)?\n");
    pc.scanf("%x", &hex);//hex input
    pc.printf("%hd\n",hex);
    pc.printf("%4x\n",hex);
    PANID[1]=hex/256;//shift left bvy 8 bits so that can fit in char(0xFF00 is too big).
    PANID[0]=hex & 0x00FF;
    hexchar[7]=PANID[1];
    hexchar[8]=PANID[0];
    hexchar[9]=checksum(hexchar);



    pc.printf("BEGIN");
    for(int i=0; i<10; i++) {

        pc.printf("%2x\n",hexchar[i]);
    }


    pc.printf("END");

    if(xbee.writeable()) {

        for(int i=0; i<10; i++) {
            xbee.putc(hexchar[i]);

        }

    }




}







void XBEE::xbeeATID()
{

    char hexchar[8] = {0x7E,0x00,0x04,0x08,0x01,0x49,0x44,0x69};
    char temp;
    int length=0;
    checksum(hexchar);
    if(xbee.writeable()) {

        for(int i=0; i<=7; i++) {
            xbee.putc(hexchar[i]);

        }

```

```
    }



    for(int i=0; i<ATIDlength; i++) {
        if(xbee.readable()) {
            temp=xbee.getc();
            ATIDplaceholder[i]=temp;
            comvec.push_back(temp);
            pc.printf("%2x\n",temp);


        }

    }

    pc.printf("done\n");
    while(ATIDplaceholder[length]!='\0') {
        length++;
    }
    pc.printf("stringlength size is%d\n",length);
    pc.printf("vector size is .... %d\n",comvec.size());
    pc.printf("last packet pos starts at ... %d\n",lastpacketpos(ATIDplaceholder,comvec.size()));
    readpacket(ATIDplaceholder,lastpacketpos(ATIDplaceholder,comvec.size()),comvec.size());
    comvec.clear();


}

char * XBEE::createlightonpacketMOD(char address[],requestType request) //could be altered to void if needed
{

    static char packet[ATISLENGTH];
    short totallength=19;//number of bytes required for address+options etc.



   // pc.printf("number of bytes in this packet is%hd\n",totallength);

    packet[0]=0x7E;
    packet[1]=(0xFF00 & (totallength-3));//-3 for frame delimiters and len MSB/LSB
    packet[2]=(0x00FF & (totallength-3));
    packet[3]=0x17;
    packet[4]=0x00;
    packet[5]=address[0];
    packet[6]=address[1];
    packet[7]=address[2];
    packet[8]=address[3];
    packet[9]=address[4];
    packet[10]=address[5];
    packet[11]=address[6];
    packet[12]=address[7];
    packet[13]=0xFF;
    packet[14]=0xFE;
    packet[15]=0x02;
    packet[16]=0x44;//D
    packet[17]=0x30;//0
    packet[18]=0x05;//make output high
    packet[totallength]=checksum(packet);

    if(request==light){
    lightStatus=true;
}
    return packet;

}

char * XBEE::createlightoffpacketMOD    (char address[],requestType request) //could be altered to void if needed
{

    static char packet[ATISLENGTH];
    short totallength=19;//number of bytes required for address+options etc.



    //pc.printf("number of bytes in this packet is%hd\n",totallength);

    packet[0]=0x7E;
    packet[1]=(0xFF00 & (totallength-3));//-3 for frame delimiters and len MSB/LSB
    packet[2]=(0x00FF & (totallength-3));
    packet[3]=0x17;
    packet[4]=0x00;
    packet[5]=address[0];
    packet[6]=address[1];
    packet[7]=address[2];
    packet[8]=address[3];
    packet[9]=address[4];
    packet[10]=address[5];
    packet[11]=address[6];
    packet[12]=address[7];
    packet[13]=0xFF;
    packet[14]=0xFE;
    packet[15]=0x02;
```

```
    packet[16]=0x44;//D
    packet[17]=0x30;//0
    packet[18]=0x04;//make output low
    packet[totallength]=checksum(packet);

    if(request==light){
    lightStatus=false;
}
    return packet;

}

void XBEE::SENDLIGHTONREQMOD()
{   __disable_irq();



    int length=20;
    int replylength=0;
    char reply[MAXMESSAGELENGTH];
    char * msgptr;



    msgptr=createlightonpacketMOD(MR1address,light);
    pc.printf("light on packet created\n");


    if(xbee.writeable()) {

        for(int i=0; i<length; i++) {
//pc.printf("contents of message packet byte %d is %2x\n", i,*(msgptr+i));

            xbee.putc(*(msgptr+i));

        }
    }



    __enable_irq();

}

void XBEE::SENDLIGHTOFFREQMOD()
{   __disable_irq();



    int length=20;
    int replylength=0;
    char reply[MAXMESSAGELENGTH];
    char * msgptr;



    msgptr=createlightoffpacketMOD(MR1address,light);
    pc.printf("light off packet created\n");


    if(xbee.writeable()) {

        for(int i=0; i<length; i++) {
//pc.printf("contents of message packet byte %d is %2x\n", i,*(msgptr+i));
            xbee.putc(*(msgptr+i));

        }
    }


 __enable_irq();
}

void XBEE::SENDPOWERONREQMOD()
{   __disable_irq();



    int length=20;
    int replylength=0;
    char reply[MAXMESSAGELENGTH];
    char * msgptr;



    msgptr=createlightonpacketMOD(MR1address,power);
    pc.printf("light on packet created\n");


    if(xbee.writeable()) {

        for(int i=0; i<length; i++) {
//pc.printf("contents of message packet byte %d is %2x\n", i,*(msgptr+i));
```

```
            xbee.putc(*(msgptr+i));

        }
    }


    __enable_irq();

}

void XBEE::SENDPOWEROFFREQMOD()
{   __disable_irq();



    int length=20;
    int replylength=0;
    char reply[MAXMESSAGELENGTH];
    char * msgptr;



    msgptr=createlightoffpacketMOD(MR1address,power);
    pc.printf("light off packet created\n");


    if(xbee.writeable()) {

        for(int i=0; i<length; i++) {
//pc.printf("contents of message packet byte %d is %2x\n", i,*(msgptr+i));
            xbee.putc(*(msgptr+i));

        }
    }




 __enable_irq();
}

void XBEE::SENDPOWERREQ()
{

__disable_irq();
    char * temp=findcurrentaddress();
      /* for(int i=0;i<8;i++){
    pc.printf("address is %2X\n",MR1address[i]);
    }*/
    swapaddress(MR2address);//point to power XBEE
    /*for(int i=0;i<8;i++){
    pc.printf("address iss %2X\n",MR1address[i]);
    }*/
    char reply[MAXMESSAGELENGTH];
    int replylength=0;

    SENDPOWERONREQMOD();//send a digital on request with no reply for interrupt on rising edge for powrr uC


    wait(0.2);//wait for serial buffer to fill.
//pc.printf("INCOMING MESSAGE\n");
    readreply3(reply,replylength);//see what the uC replies with when it sends a message back
    readpacket(reply,lastpacketpos(reply,replylength),replylength);//extract raw data values from API frame.


    SENDPOWEROFFREQMOD();//set the interrupt pin on the uC to low ready for the next call.
    swapaddress(temp);

__enable_irq();
}

void XBEE::SENDLIGHTSTATUSREQ(){
__disable_irq();

    int length=19;
    int replylength=0;
    char reply[MAXMESSAGELENGTH];
    char * msgptr;
    char * temp=findcurrentaddress();
    /*for(int i=0;i<8;i++){
    pc.printf("address is %2X\n",MR1address[i]);
    }*/
    swapaddress(MR3address);//edit for #def power XBEE
    /*for(int i=0;i<8;i++){
    pc.printf("address iss %2X\n",MR1address[i]);
    }*/




    msgptr=createlightStatuspacket(MR1address);
    pc.printf("LIGHT STATUS packet created\n");
```

```
    if(xbee.writeable()) {

        for(int i=0; i<length; i++) {
            //pc.printf("contents of message packet byte %d is %2x\n", i,*(msgptr+i));

            xbee.putc(*(msgptr+i));

        }
    }
    readreply2(reply,replylength);
    readpacket(reply,lastpacketpos(reply,replylength),replylength);



        swapaddress(temp);
        __enable_irq();

 }
```

code/homeauto–xbeeC.cpp

```
#ifndef XBEE_H
#define XBEE_H
#include "mbed.h"
#include <vector>
#include "MODSERIAL.h"

//external globals declared in main loop which allow measurements to be passed from this library to the main loop.

extern int pVoltage;
extern int pCurrent;
extern int pPhase;
extern int motionCount;

//addresses of the XBees

extern char MR0address[8];//motion
extern char MR2address[8];//power
extern char MR3address[8];//light control
extern bool lightStatus;//current light status

class XBEE{

char checksum(char hexchar[]);//calculates checksum of given packet
enum requestType{light,power};//enumeration for differentiating between light control analogue readings and power
     readings
void readpacket(char packetstring[],int packetpos,int packetstringlength);//decode packet information.
char * createmessagepacket(char message[],char address[]);//create API frame for sending ASCII messages
void readreply(char reply[], int &replylength);
void readreply2(char reply[], int &replylength);//read the reply with all bytes shown
void readreply3(char reply[],int &replylength);//read the reply with no prints (good for interrupts).
char * createsensorpacket(char address[]);
char * createlightStatuspacket(char address[]);
char * createlightonpacket(char address[]);
char * createlightoffpacket(char address[]);
char * createlightoffpacketMOD(char address[],requestType request);//MODIFIED packet so that there is no reply from
     the receiver XBEE so that it doesn't confuscate the ASCII string sent from power XBEE
char * createlightonpacketMOD(char address[],requestType request);
void readaddress(char address[]);
char * findcurrentaddress();//return pointer to array holding current address.







public:

void SENDSENSORREQ();//normal analogue reading on AD1
void SENDMOTIONREQ();//send packet and decode motion sensor reply
void SENDLIGHTONREQ();//turn light on using DO
void SENDLIGHTOFFREQ();//turn light off using
void SENDMESSAGE();
void xbeeATID();//find PAN ID
void xbeeATIDCHANGE();//change PAN ID
void SENDLIGHTONREQMOD();//no reply for power uC
void SENDLIGHTOFFREQMOD();//no reply for power uC
void SENDPOWERONREQMOD();//no reply for power uC
void SENDPOWEROFFREQMOD();//no reply for power uC
void SENDPOWERREQ();
void SENDLIGHTSTATUSREQ();//find status of light connected to XBee which is currently pointed to
void swapaddress(char address[]);
int lastpacketpos(char packet[], int arraylength);
```

```

};

#endif
```

code/homeauto–xbee.h

8.1.2 Wireless Power Monitoring

Power Diagram

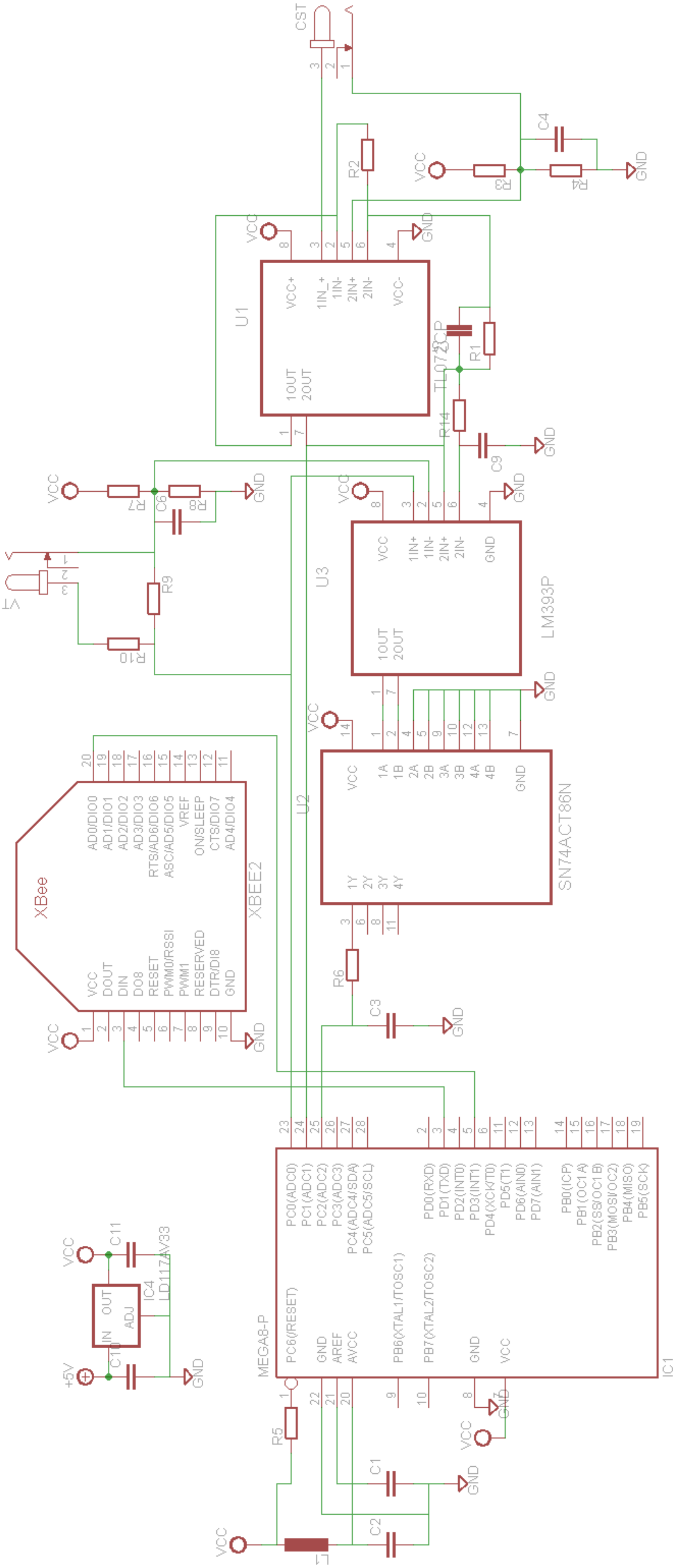


Figure 8.2: Power meter circuit diagram

8.2 Software

8.2.1 Server Code

```
#import required libraries
import tornado.httpserver
import tornado.websocket
import tornado.ioloop
import tornado.web
import time
import json
import requests
import random
import functions as mfunc
import system_primary_functions as syspf

#set user count to 0
users = 0
#custom function to test if a string is JSON
def json_test(string):
    try:
        json.loads(string)
        return 1
    except:
        return 0

#handle http
class MainHandler(tornado.web.RequestHandler):
    def get(self):
        self.write('{"type":"message","message":"Hello this is not the correct websocket address. Please add \ws."}')
#websocket handler
class WSHandler(tornado.websocket.WebSocketHandler):
  #keep track of all users currently connected
    clients = []
    def allow_draft76(self):
        # to use same draft as mbed library
        return True
  #on connection open
    def open(self):
    #add user to user list
        self.clients.append(self)
        print 'new connection'
    #update user count
        global users
        users = users + 1
        print 'users : %d' % users
    #send welcome message
        self.write_message('{"type":"message","message":"Hello World"}')
    #log the event
        with open("tornado_ws.log", "a") as myfile:
            localtime = time.asctime( time.localtime(time.time()) )
            myfile.write('%s:new connection : %d users \r\n' % (localtime, users))
     #on message receipt
    def on_message(self, message):
    #log the event
        print 'message received %s' % message
        with open("tornado_ws.log", "a") as myfile:
            localtime = time.asctime( time.localtime(time.time()) )
            if message != '{"type":"ping"}':
                myfile.write('%s:message received %s \r\n' % (localtime,message))
        #test if message is JSON and decide whether to process further
    if (json_test(message) == 1):
      print 'message is JSON'
      #extract data
            data = json.loads(message)
      #do different actions based on type of message
            if data['type'] == 'house_measurement':
                print "House measurement Data"
                self.write_message('{"type":"message","message":"Sent to Mike GPR"}')
                try:
          #pass on to rest of clients if temperature data received
                    response =  '{"type":"temperature_measurement","temperature":%d}' % (data['Temperature'])
                    print response
                    for client in self.clients:
                        client.write_message(response)
                except:
          print "No temp"
                try:
          #send data to the GPR script
          syspf.get_TS(data['Humidity'])
                except:
          print "GPR failed"
        #post data to database
                r = requests.post("http://ee-ug1.ee.ic.ac.uk/actual_web2/house_measurement.php", data=data)
                print r.text
      #forward all messages with appropriate type
            elif data['type'] == 'light_request':
                for client in self.clients:
                    client.write_message(message)
            elif data['type'] == 'light_request_response':
                for client in self.clients:
```

```python
                client.write_message(message)
        elif data['type'] == 'light_control':
            for client in self.clients:
                client.write_message(message)
        elif data['type'] == 'thermostat_control':
            for client in self.clients:
                client.write_message(message)
        elif data['type'] == 'ping':
    print 'pong'
    #reply pong to a ping only to the client who sent it
    response = '{"type":"pong"}'
    self.write_message(response)
        elif data['type'] == 'thermostat_set':
            for client in self.clients:
                client.write_message(message)
        elif data['type'] == 'thermostat_request':
            for client in self.clients:
                client.write_message(message)
        elif data['type'] == 'thermostat_request_response':
            for client in self.clients:
                client.write_message(message)
        else:
            print "Other data"
    #else:
    #    print 'message is not JSON'

    def on_close(self):
    #remove client from list
        self.clients.remove(self)
        print 'connection closed'
    #decrease user count
        global users
        users = users - 1
        print 'users : %d' % users
    #log it
        with open("tornado_ws.log", "a") as myfile:
            localtime = time.asctime( time.localtime(time.time()) )
            myfile.write('%s:connection closed : %d users \r\n' % (localtime,users))

application = tornado.web.Application([
    (r"/", MainHandler),
    (r'/ws', WSHandler),
])

#set up to run as process
if __name__ == "__main__":
    http_server = tornado.httpserver.HTTPServer(application)
  #listen to 8888
    application.listen(8888)
    tornado.ioloop.IOLoop.instance().start()
```

code/tornado_ws.py

8.2.2 Timetable Prediction

Choice of day

Testing how many days to take the mode

Obtained from: University of Massachusetts Amherst SMART* dataset

Acc Avg: Is the average accuracy as prediction algorithm is tested on a full 3 months of data

Std.Dev.: is the standard deviation in the average accuracy for each day

len: is the number of days of which the mode is taken to predict the future

s_p: number of past samples used to take the hamming distance of - for example 144 samples corresponds to 12 hours when the sampling is 5 minutes (the case in this data)

s_f: number of future samples taken to then check the accuracy against the know future data

Acc Avg:79.2% Std.Dev.:20% w/day len=2 s_p=144 s_f=144
Acc Avg:79.9% Std.Dev.:22% w/day len=3 s_p=144 s_f=144
Acc Avg:81.5% Std.Dev.:21% w/day len=4 s_p=144 s_f=144
Acc Avg:80.6% Std.Dev.:21% w/day len=5 s_p=144 s_f=144
Acc Avg:80.5% Std.Dev.:21% w/day len=6 s_p=144 s_f=144
Acc Avg:79.5% Std.Dev.:21% w/day len=7 s_p=144 s_f=144
Acc Avg:79.0% Std.Dev.:22% w/day len=8 s_p=144 s_f=144
Acc Avg:79.0% Std.Dev.:22% w/day len=9 s_p=144 s_f=144
Acc Avg:79.0% Std.Dev.:22% w/day len=10 s_p=144 s_f=144
Acc Avg:79.3% Std.Dev.:23% w/day len=11 s_p=144 s_f=144
Acc Avg:79.3% Std.Dev.:23% w/day len=12 s_p=144 s_f=144
Acc Avg:79.3% Std.Dev.:23% w/day len=13 s_p=144 s_f=144

```
Acc Avg:79.3% Std.Dev.:23% w/day len=14 s_p=144 s_f=144
Acc Avg:79.3% Std.Dev.:23% w/day len=15 s_p=144 s_f=144
Acc Avg:79.3% Std.Dev.:23% w/day len=16 s_p=144 s_f=144
Acc Avg:79.3% Std.Dev.:23% w/day len=17 s_p=144 s_f=144
Acc Avg:79.3% Std.Dev.:23% w/day len=18 s_p=144 s_f=144
Acc Avg:79.3% Std.Dev.:23% w/day len=19 s_p=144 s_f=144
Acc Avg:79.3% Std.Dev.:23% w/day len=20 s_p=144 s_f=144
Acc Avg:79.3% Std.Dev.:23% w/day len=21 s_p=144 s_f=144
Acc Avg:79.3% Std.Dev.:23% w/day len=22 s_p=144 s_f=144
Acc Avg:79.3% Std.Dev.:23% w/day len=23 s_p=144 s_f=144
Acc Avg:79.3% Std.Dev.:23% w/day len=24 s_p=144 s_f=144
Acc Avg:79.3% Std.Dev.:23% w/day len=25 s_p=144 s_f=144
Acc Avg:79.3% Std.Dev.:23% w/day len=26 s_p=144 s_f=144
Acc Avg:79.3% Std.Dev.:23% w/day len=27 s_p=144 s_f=144
Acc Avg:79.3% Std.Dev.:23% w/day len=28 s_p=144 s_f=144
Acc Avg:79.3% Std.Dev.:23% w/day len=29 s_p=144 s_f=144
Acc Avg:79.3% Std.Dev.:23% w/day len=30 s_p=144 s_f=144
```

```
Acc Avg:74.9% Std.Dev.:26% w/day len=2 s_p=144 s_f=72
Acc Avg:76.9% Std.Dev.:27% w/day len=3 s_p=144 s_f=72
Acc Avg:80.1% Std.Dev.:27% w/day len=4 s_p=144 s_f=72
Acc Avg:78.3% Std.Dev.:28% w/day len=5 s_p=144 s_f=72
Acc Avg:77.8% Std.Dev.:29% w/day len=6 s_p=144 s_f=72
Acc Avg:75.9% Std.Dev.:29% w/day len=7 s_p=144 s_f=72
Acc Avg:74.8% Std.Dev.:31% w/day len=8 s_p=144 s_f=72
Acc Avg:74.8% Std.Dev.:31% w/day len=9 s_p=144 s_f=72
Acc Avg:74.8% Std.Dev.:31% w/day len=10 s_p=144 s_f=72
Acc Avg:75.4% Std.Dev.:31% w/day len=11 s_p=144 s_f=72
Acc Avg:75.4% Std.Dev.:31% w/day len=12 s_p=144 s_f=72
Acc Avg:75.4% Std.Dev.:31% w/day len=13 s_p=144 s_f=72
Acc Avg:75.4% Std.Dev.:31% w/day len=14 s_p=144 s_f=72
Acc Avg:75.4% Std.Dev.:31% w/day len=15 s_p=144 s_f=72
Acc Avg:75.4% Std.Dev.:31% w/day len=16 s_p=144 s_f=72
Acc Avg:75.4% Std.Dev.:31% w/day len=17 s_p=144 s_f=72
Acc Avg:75.4% Std.Dev.:31% w/day len=18 s_p=144 s_f=72
Acc Avg:75.4% Std.Dev.:31% w/day len=19 s_p=144 s_f=72
Acc Avg:75.4% Std.Dev.:31% w/day len=20 s_p=144 s_f=72
Acc Avg:75.4% Std.Dev.:31% w/day len=21 s_p=144 s_f=72
Acc Avg:75.4% Std.Dev.:31% w/day len=22 s_p=144 s_f=72
Acc Avg:75.4% Std.Dev.:31% w/day len=23 s_p=144 s_f=72
Acc Avg:75.4% Std.Dev.:31% w/day len=24 s_p=144 s_f=72
Acc Avg:75.4% Std.Dev.:31% w/day len=25 s_p=144 s_f=72
Acc Avg:75.4% Std.Dev.:31% w/day len=26 s_p=144 s_f=72
Acc Avg:75.4% Std.Dev.:31% w/day len=27 s_p=144 s_f=72
Acc Avg:75.4% Std.Dev.:31% w/day len=28 s_p=144 s_f=72
Acc Avg:75.4% Std.Dev.:31% w/day len=29 s_p=144 s_f=72
Acc Avg:75.4% Std.Dev.:31% w/day len=30 s_p=144 s_f=72
```

```
Acc Avg:79.6% Std.Dev.:25% w/day len=2 s_p=144 s_f=36
Acc Avg:80.6% Std.Dev.:25% w/day len=3 s_p=144 s_f=36
Acc Avg:84.0% Std.Dev.:25% w/day len=4 s_p=144 s_f=36
Acc Avg:81.2% Std.Dev.:28% w/day len=5 s_p=144 s_f=36
Acc Avg:80.5% Std.Dev.:28% w/day len=6 s_p=144 s_f=36
Acc Avg:77.8% Std.Dev.:29% w/day len=7 s_p=144 s_f=36
Acc Avg:76.2% Std.Dev.:30% w/day len=8 s_p=144 s_f=36
Acc Avg:76.2% Std.Dev.:30% w/day len=9 s_p=144 s_f=36
Acc Avg:76.2% Std.Dev.:30% w/day len=10 s_p=144 s_f=36
Acc Avg:77.1% Std.Dev.:30% w/day len=11 s_p=144 s_f=36
Acc Avg:77.1% Std.Dev.:30% w/day len=12 s_p=144 s_f=36
Acc Avg:77.1% Std.Dev.:30% w/day len=13 s_p=144 s_f=36
```

```
Acc Avg:77.1% Std.Dev.:30% w/day len=14 s_p=144 s_f=36
Acc Avg:77.1% Std.Dev.:30% w/day len=15 s_p=144 s_f=36
Acc Avg:77.1% Std.Dev.:30% w/day len=16 s_p=144 s_f=36
Acc Avg:77.1% Std.Dev.:30% w/day len=17 s_p=144 s_f=36
Acc Avg:77.1% Std.Dev.:30% w/day len=18 s_p=144 s_f=36
Acc Avg:77.1% Std.Dev.:30% w/day len=19 s_p=144 s_f=36
Acc Avg:77.1% Std.Dev.:30% w/day len=20 s_p=144 s_f=36
Acc Avg:77.1% Std.Dev.:30% w/day len=21 s_p=144 s_f=36
Acc Avg:77.1% Std.Dev.:30% w/day len=22 s_p=144 s_f=36
Acc Avg:77.1% Std.Dev.:30% w/day len=23 s_p=144 s_f=36
Acc Avg:77.1% Std.Dev.:30% w/day len=24 s_p=144 s_f=36
Acc Avg:77.1% Std.Dev.:30% w/day len=25 s_p=144 s_f=36
Acc Avg:77.1% Std.Dev.:30% w/day len=26 s_p=144 s_f=36
Acc Avg:77.1% Std.Dev.:30% w/day len=27 s_p=144 s_f=36
Acc Avg:77.1% Std.Dev.:30% w/day len=28 s_p=144 s_f=36
Acc Avg:77.1% Std.Dev.:30% w/day len=29 s_p=144 s_f=36
Acc Avg:77.1% Std.Dev.:30% w/day len=30 s_p=144 s_f=36
```

Matlab Timetable Prediction Test Script

```matlab
clc;
clear all;
clear;
close all;
%load data
% load house_data
%{
load housestate.mat
clear t_mod
clear total_firings
clear p_state
%}
%%{
%setup variables
load usualday
t_range = dat;
house_state = num';
clear num;
clear dat;
%}
%load threemonths
% set all constants
day_samples = 96;%288
ratio = 288/day_samples; %ratio to 288 - used if non-5minute intervals
sample_p = 144/ratio; %half a day
sample_f = 144/ratio; %half a day

%% day prediction

%set time offset
time_o = 90/ratio; %equivalent to 7:30 AM

%store all data in a 3d array for optimisation
acc_3d = [1 1 1];

day_to_compare = 4; % days to perform a mode on
days = 2:length(house_state)/day_samples-1; % days to cycle through
for len=day_to_compare
    for sample_p = round(day_samples*(1:40)/40)
        for sample_f = round(day_samples*(1:40)/40)%[36 72 144]
            %set timing variable
            acc = 0;
            for day=days
                day_index = time_o+(day-1)*day_samples+1; %day with time offset
                %store data for future - i.e. to check for accuracy
                day_f = house_state(day_index:day_index+sample_f-1);
                %store data from past to compute Hamming distances
                day_p = house_state(day_index-sample_p:day_index-1);
                %prepare data on states - removing excess data
                states = house_state;
                states(day_index-sample_p:day_index+sample_f-1) = 3; %set to 3 as this is impossible state
                %loop through each day to calculate hamming distance of each
                %day relative to each other
                for i=1:length(states)/day_samples-1
        %                 disp(sprintf('Day: %d',i))
        %                 disp(datestr(t_range(i*day_samples-sample_p+time_o+1)))
        %                 disp(datestr(t_range(i*day_samples+time_o)))
                    comp = [day_p;states(i*day_samples-sample_p+time_o+1:i*day_samples+time_o)];
                    D(i) = pdist(comp,'hamming');
        %                 figure;
        %                 subplot(2,1,1), plot(states(i*day_samples-sample_p+time_o+1:i*day_samples+time_o))
        %                 subplot(2,1,2), plot(day_p)
        %                 title(sprintf('Day:%d Hamming distance:%.2f',i,D(i)))
```

```
                end
                %get minimum values of D with their indexes
                [sortedValues,sortIndex] = sort(D,'ascend');
                maxIndex = sortIndex(1:len); %get minimum hamming distance days
                %make sure not to go out of range
                if any(maxIndex == length(states)/day_samples-1)
                    index = find(maxIndex==length(states)/day_samples-1);
                    maxIndex(index) = sortIndex(len+1);
                end
                %days_hd stores all the days with the smallest hamming distance
                days_hd = ones(sample_f,len)';
                for i=maxIndex
        %                   disp(datestr(t_range(i*day_samples+time_o+1)))
        %                   disp(datestr(t_range(i*day_samples+time_o+sample_f)))
                    days_hd = [days_hd ;house_state(i*day_samples+time_o+1:i*day_samples+time_o+sample_f)];
                end
                %strip first value
                days_hd = days_hd(2:end,:)';
                days_hd = mode(days_hd');
                    days_hd(days_hd==2) = 1;
                    day_f(day_f==2) = 1;
                %calc accuracy
                acc = [acc 100*(1-pdist([days_hd; day_f],'hamming'))];
        %               figure;
        %               subplot(2,1,1), plot(days_hd)
        %               subplot(2,1,2), plot(day_f)
            end
            acc = acc(2:end);
            %print data
            fprintf('Acc Avg:%2.1f%% Std.Dev.:%2.f%% w/day len=%d s_p=%d s_f=%d\n',mean(acc),std(acc),len,sample_p,
    sample_f);
            acc_3d = [acc_3d;sample_p,sample_f,mean(acc)];
        end
    end
end
%store in variable to allow 3d plot
acc_3d = acc_3d(2:end,:);
```

general_prediction.m

Python code to predict timetable

```python
#import all used modules
import MySQLdb
import gc
import numpy as np
import scipy as sp
from scipy import spatial
from scipy import stats
from scipy import signal
import math
import sys
import time
from datetime import datetime, timedelta


#from past data predicts the expect day states
#in: day present states
#out: expected future states
def get_future_samples(day_p):
  #initialize distance array
  D = np.array([])
  #loop through each day finding the closest days in terms of hamming distance
  for i in range(1,days-2):
    #get array to compare
    comp =  np.array([day_p,house_state[i*day_samples-sample_p+time_o:i*day_samples+time_o]])[:,:]
    #store hamming distance in D
    D = np.append(D,sp.spatial.distance.pdist(comp,'hamming'))
  #sort by index
  sortIndex =  np.argsort(D)
  days_hd = []
  #get days to compare
  for i in sortIndex[0:day_to_compare]:
    days_hd.append(house_state[i*day_samples+time_o:i*day_samples+time_o+sample_f])
  #convert to array
  days_hd = np.array(days_hd)
  #get the mode
  days_hd = sp.stats.mode(days_hd)[0][0]
  return days_hd

#custom discrete decimating array function
def decimate_array(x, q):
  x_out = []
  if not isinstance(q, int):
    raise TypeError("q must be an integer")
  if len(x)%q != 0:
    raise NameError("Cannot decimate!")
  for i in range(0,len(x)/q):
    x_out.append(int(sp.stats.mode(x[i*q:(i+1)*q])[0][0]))
  return x_out
```

```
#custom array flattening function
def flatten_array(x):
  y = []
  for i in range(0,len(x)):
    y.append(x[i][0])
  return y


#define constants
day_samples = 288
ratio = 288/day_samples
#samples to regress on
sample_p = 144/ratio
#samples to look into the future of
sample_f = 288/ratio
day_to_compare = 4 # days to perform a mode on
#get minutes
minutes_sample = int(24*60/day_samples);




print "Welcome to Seb Grubb's (c) 2013 TimeTable filler & Predictor"
print "************************************************"
print '''
 .__                           __.
  \ `\~~---..---~~~~~~--.---~~| /
   `~-.   `                   .~         _____
       ~.                .--~~    .---~~~    /
        / .-.      .-.      |  <~~        __/
       |  |_|      |_|       \  \     .--/
      /-.      -       .-.    |  \_   \_
      \-'   -..-..-    `-'    |    \__  \_
       `.                     |     _/  _/
         ~-                .,-\   _/  _/
        /                 -~~~~\ /_  /_
       |               /   |    \  \_  \_
       |   /          /   /      | _/  _/
       |  |          |   /    . -|/  _/
       )__/           \_/    -~~~|  /
         \                      /  \\
          |           |        /_---/
          \    .______|      ./
          (   /        \    /
          |--|          /__/
'''
print "Legal notice: all incorrect future predictions are not \n\tthe responsibility of AutoHome"
#connect to sql and get all relevant data
db = MySQLdb.connect(host="127.0.0.1", port=3306, user="root", passwd="London123", db="Data")
cursor = db.cursor()
#get data
cursor.execute("SELECT State FROM H_State")
row = cursor.fetchall()
house_state = flatten_array(np.asarray(row))
#get data
cursor.execute("SELECT Timestamp FROM H_State")
date = np.asarray(cursor.fetchall())
#close SQL connection
db.close()
gc.collect()

#get date of last sample
date_object = datetime.fromtimestamp(int(date[len(date)-1][0])/1000.0)

#prepare variables needed
print "preparing past data",
days = int(math.ceil(len(house_state)/float(day_samples))) -1
print ".",
day_index = (days)*day_samples
print ".",
time_o = len(house_state)-day_index
print "."

#store past samples
print "processing data",
day_p = house_state[day_index-sample_p:day_index]
#get estimated future states
day_f = get_future_samples(day_p) #stores future predicted values
day_predict = house_state[day_index:day_index+time_o]#stores the values of a day to fill in.
print ".",
#convert to correct types
day_predict = np.asarray(day_predict)
day_f =  np.asarray(day_f).astype(int)
print ".",
#append to old data
day_predict = np.append(day_predict,day_f)
day_predict = day_predict[0:day_samples]
day_predict = decimate_array(day_predict,12)
print "."

#prepare to send to sql
#0 - away, 1 - in, 2 - asleep - my data
#0 - asleep, 1 - in , 2 -out
```

```
for i,v in enumerate(day_predict):
  if v==0:
    day_predict[i] = 2
  if v==2:
    day_predict[i] = 0
#send to mySQL
print "sending to mySQL",
#store in correctly named table
day_index_name = "Auto_%s" % ((date_object+timedelta(minutes=144*5)).strftime('%A'))
print ".",
db = MySQLdb.connect(host="127.0.0.1", port=3306, user="root", passwd="London123", db="House_states")
cursor = db.cursor()
print ".",
for i,v in enumerate(day_predict):
  command = "UPDATE `%s`\nSET state=%d\nWHERE Hour=%d;" % (day_index_name,v,i)
  cursor.execute(command)
print "."


#process other days in the week
day_list = range(0,(date_object+timedelta(minutes=144*5)).weekday())
day_list.extend(range((date_object+timedelta(minutes=144*5)).weekday()+1,7))
#loop through every last 4 days of the week and take the mode of each day
for d in day_list:
  iteration = 0
  day_states_data = []
  for i in range(days-1,-1,-1):
    if datetime.fromtimestamp(int(date[i*day_samples][0])/1000.0).weekday() == d:
      iteration+=1
      day_states_data.append(house_state[i*day_samples:(i+1)*day_samples])
    if iteration >= day_to_compare:
      break
  day_index_name = "Auto_%s" % (datetime.fromtimestamp(int(date[i*day_samples][0])/1000.0).strftime('%A'))
  day_states_data = decimate_array(np.asarray(sp.stats.mode(day_states_data)[0][0]).astype(int),12)
  #invert 2 and 0
  for i,v in enumerate(day_states_data):
    if v==0:
      day_states_data[i] = 2
    if v==2:
      day_states_data[i] = 0
  for i,v in enumerate(day_states_data):
    command = "UPDATE `%s`\nSET state=%d\nWHERE Hour=%d;" % (day_index_name,v,i)
    cursor.execute(command)
#close SQL connection
db.close()
gc.collect()


print'''                 __....___ ,  .
            _.-~ __...--~~  ~/\\
           / /  /           |  |
          | |  |             \\  \\
    __ _..---..-~\\            |  |
   |  ~  .-~-.    \\-.__       /  |
   /     \\.-~         .-~-._/   //
  |/-. <| __  .-\\     \\      \\_ //
  || o\\   \\/ /o  |    ~-.-~  \\/
 /  ~~         ~~              |
 \\__          ___--/   \\  _-~-  \\
  / ~~--.--~~     /     |/    __  |
 |/\\ \\          |_~|   /    \\|  |
 |\\/  \\__        /_-   /\\        |
 |_ __| |`~-.__|_ _ _/   \\ _ _ _/
 ' '  ' ' ''   ' '  '     ' ` `
'''
```

sql–ttfill.py

8.2.3 Gaussian Process Regression

General Machine Learning (ML) Terminology

Features are the input variables to the algorithm, i.e. if the algorithm is the function that outputs thermostat settings, then the features of the ML algorithm are the inputs or the feature space/domain. Features are inputted as vectors typically denoted by $\mathbf{x}$. Our choice of features was dictated by what made logical sense by the data available to us. The features selected are listed below:

1. External Temperature
2. External Humidity
3. Internal Humidity
4. Wind Speed
5. Wind Direction

(Internal temperature was omitted as this would provide unwanted feedback on the feedback from the control system, which the thermostat uses to regulate the internal temperature)

1. ***Training data set -*** is the set of feature vectors or 'training examples' in the feature space that the algorithm uses to fit its regression curve in order to predict the thermostat setting for different conditions.
2. ***Target data set -*** is the set of outputs (in this context thermostat settings) for each of the training data points.
3. ***Model Parameters -*** are the set of unknown coefficients of the model that is applied to the data. Linear Regression is an example of a parametric approach in which a model is chosen and then the parameters of the model are optimized to best fit the data.
4. ***Hyperparameters -*** are important in non- parametric models such as Gaussian Process Regression. Gaussian Process Regression has no direct parameters to find or optimize, but it does have parameters for the prior, i.e. the coefficients of the covariance function. These need to be optimized to fit the data in a similar way to model parameters in parametric regression.
5. ***Hypothesis -*** also known as the regression curve is the function that is inferred from the training data.

Terms and Definitions for GPR:

1. ***Random Variable*** – A random variable is a variable that can take on a set of possible values each with an associated probability.
2. ***Prior Probability*** –the probability distribution that one assigns to a quantity that expresses the uncertainty about the process before any data is observed. In GPR we assume a Gaussian distribution prior.
3. ***Likelihood*** – the probability of the system parameters given the data that is observed.
4. ***Marginal Likelihood*** –the likelihood function integrated over certain parameters. In this way the parameters have been 'marginalized'. It is also known as the 'model evidence' because integrating this over the function f gives the probability of the data given the chosen hyperparameters.
5. ***Posterior Probability*** –The conditional probability that is assigned once relevant data and information is taken into account.
6. ***Covariance*** - A measure of how much two random variables change with one another.
7. ***Stochastic Process*** – is a collection of random variables.
8. ***Covariance Function*** – For a stochastic process or random field, a covariance function describes the spatial covariance of points in the random field located at certain points in space. Covariance functions belong to a family of mapping functions called 'kernels'.
9. ***Multivariate Gaussian distribution*** –a generalization of the standard Gaussian or Normal distribution to multiple dimensions. It is described by a mean vector with a separate mean for each dimension, and a covariance matrix (a generalization of variance to multiple dimensions). It gives a similarity measure of each point with every other point in the distribution and is expressed as:

$$\beta \sim N(\mu,\ \mathbf{K})$$

β is a multi-dimensional random vector, μ is a mean vector defining the mean for each dimension**,** and $\mathbf{K}$ is the covariance matrix, detailing how much each random variable of $\mathbf{x}$ changes with another, defined as:

$$\mathbf{K} = \left[\begin{pmatrix} k\left(\mathbf{x_1},\ \mathbf{x_1}\right) & \cdots & k\left(\mathbf{x_1},\ \mathbf{x_n}\right) \\ \vdots & \ddots & \vdots \\ k\left(\mathbf{x_n},\ \mathbf{x_1}\right) & \cdots & k\left(\mathbf{x_n},\ \mathbf{x_n}\right) \end{pmatrix}\right]$$

The Probability Density Function (PDF) of a Multivariable Gaussian Distribution is:

$$pdf_\beta = P\left(\beta{=}\mathbf{a}\right) = (2\pi)^{-\left(\frac{n}{2}\right)}|\mathbf{K}|^{-0.5}\exp\left(-\frac{1}{2}(\mathbf{a}-\mu\ )^T\mathbf{K}^{-1}(\mathbf{a}-\mu)\right) \tag{8.1}$$

1. ***Gaussian Process*** - Is a stochastic process whose random variables, associated with each point in the random domain space, have a normal distribution. A Gaussian process can be thought of/visualised as an infinite dimensional Gaussian distribution. It is defined as:

$$f\left(\mathbf{x}\right) \sim GP(m(\mathbf{x}),\ k\left(\mathbf{x_i},\ \mathbf{x_j}\right)\ )$$

$m(\mathbf{x})$ is the mean function and '$k(\mathbf{x_i}, \mathbf{x_j})$' is the similarity or covariance function that measures the similarity between two random variables of the Gaussian process. For ease of computation, and without loss of generality we can set $m(x) = 0$ as summing an infinite array of possible functions averages to 0. An important property of a Gaussian Process is that it obeys the consistency requirement; i.e. inspection of a larger set of random variables does not change the distribution of a smaller set. Every finite collection of these random variables can be described with a multivariate Gaussian distribution.

Algorithms and their Complexity:

For this system to operate effectively three algorithms/functions are required; one to search the error curve to find a range of hyperparameter initialisations that find an acceptable minima, a second to train the GPR and a third to run the GPR and return a predicted thermostat setting. In this section *'m'* denotes the number of features and *'n'* the number of examples in the training set. Furthermore n>>m, n>> any loop counters

Algorithm 1: Hyperparameter Search

Description:

Called every time a significant change in the training data is detected, so fairly infrequently (e.g. the first time the training data reaches its maximum capacity, or when a significant amount of data is changed).

Pseudo code:

1. Import training data into $\mathbf{X}$ and cross validation data into $\mathbf{x}^*$
2. Define upper and lower bound (ub and lb) as the boundaries within which to initialise θ values. Initialise over some very large range.
3. Grid Search:

while (ub – lb)>1

$$\text{increment} = (\text{ub} - \text{lb})/\text{numb\_div}$$

lb_temp = lb

for j = 1->(numb_div – 1)

Initialise θ in range(lb_temp, ub)

Train the GPR

Run the cross validation data through 'GPR Run'

$$\text{AvError[j]} = \frac{1}{n}\sum_{i=1}^{n}\left\{\frac{(y_i - ypred_i) * 100}{y_i}\right\}$$

minIndex = index of minimum element in AvError[j]

ub = lb +minIndex*increment

lb = lb + (minIndex-1)*increment

1. Write upper bound and lower bound values to SQL database to be used by 'GPR train'

$$[Total\ Complexity \approx \log_{numb_{div}}(ub - lb) * numb_{div} * \left(O(n) + O\left(n^2\right) + O\left(n^3\right)\right) \approx O\left(n^3\right)]$$

Algorithm 2: GPR Train

Description:

Called every time the training data set is updated. Calculates the inverse covariance matrix (plus noise) and the optimized hyperparameters, which are then both written back to the SQL database to be used by the 'GPR run' algorithm. As an aside the complexity of the kernel $k(\mathbf{x_n}, \mathbf{x_n})$ is, considering only multiplications, divisions, subtractions and additions, $\left(m^2 + m + 1\right) \approx O(m^2)$

Pseudo code:

1. Import training data from SQL database into $\mathbf{X}$ and the target data into $\mathbf{y}$
2. Normalize $\mathbf{X}$: $\forall x_i\ in\ \mathbf{X}$, $x_i = \frac{(x_i - \mu)}{\sigma_n}$ (his ensures that all features are equally important and improves performance for gradient descent as each dimension has a similar range of relevant magnitudes)
3. Initialise hyperparameters, θ, by taking values from a uniform distribution with limits found by the hyper parameter search
4. Gradient Descent:

for j = 1->max_iters **or** until $L(\theta)$ not decreasing **or** $L(\theta) < desired\_error$

$$\mathbf{C} = \left[\begin{pmatrix} k(\mathbf{x_1},\ \mathbf{x_1}) + \sigma_n^2\delta_{\mathbf{i}j} & \cdots & k(\mathbf{x_1},\ \mathbf{x_n}) + \sigma_n^2\delta_{\mathbf{i}j} \\ \vdots & \ddots & \vdots \\ k(\mathbf{x_n},\ \mathbf{x_1}) + \sigma_n^2\delta_{\mathbf{i}j} & \cdots & k(\mathbf{x_n},\ \mathbf{x_n}) + \sigma_n^2\delta_{\mathbf{i}j} \end{pmatrix}\right] [n^2(2m^2+1) \approx O\left(n^2m^2\right)]$$

$$L(\theta) = \frac{1}{2}y^TC^{-1}y + \frac{1}{2}\log(|C|) + \left(\frac{n}{2}\right)\log(2\pi)\ [O(n^2) + O(n^3) \approx O(n^3)]$$

for i = 1-> numb_hyperparameters

$$\theta_i = \theta_i - \alpha\frac{\partial L(\theta)}{\partial \theta_i}[n^3 + 2n + 2n^2 \approx O(n^3)]$$

Return θ

1. Using optimized θ values calculate $\mathbf{C} = \left[\begin{pmatrix} k(\mathbf{x_1},\ \mathbf{x_1}) + \sigma_n^2\delta_{\mathbf{ij}} & \cdots & k(\mathbf{x_1},\ \mathbf{x_n}) + \sigma_n^2\delta_{\mathbf{ij}} \\ \vdots & \ddots & \vdots \\ k(\mathbf{x_n},\ \mathbf{x_1}) + \sigma_n^2\delta_{\mathbf{ij}} & \cdots & k(\mathbf{x_n},\ \mathbf{x_n}) + \sigma_n^2\delta_{\mathbf{ij}} \end{pmatrix}\right]$
2. Calculate $\mathbf{C^{-1}}$ $[O(n^3)]$
3. Write θ and $\mathbf{C^{-1}}$ to SQL database

$$[Total\ Complexity = 3O(n^3) + O\left(n^2m^2\right) \approx O(n^3)$$

Algorithm 3: GPR Run
Description:
Algorithm is called every time the embed requests a new thermostat setting. Returns the desired thermostat setting and the confidence with which the prediction is made.
Pseudo code:

1. Import training data into $\mathbf{X}$, target data into $\mathbf{y}$, new input feature vector $\mathbf{x}^*$, the hyperparameters θ and $\mathbf{C^{-1}}$
2. Calculate $\mathbf{K}_* = \begin{bmatrix} k(\mathbf{x}_*,\ \mathbf{x_1}) \\ \vdots \\ k(\mathbf{x}_*,\ \mathbf{x_n}) \end{bmatrix}$ $[n(m^2+m+1) \approx O(nm^2)]$
3. Calculate $\mathbf{K}_{**} = k(\mathbf{x}_*,\ \mathbf{x}_*)$ $[O(m^2)]$
4. Predicted Thermostat Setting: $TS = \mathbf{K}_*^{\mathbf{T}}(\mathbf{K} + \sigma_n^2\mathbf{I})^{-\mathbf{1}}\mathbf{y}$ $[2n^2 \approx O(n^2)]$
5. Confidence: $TS_{conf} = \mathbf{K}_{**} - \mathbf{K}_*^{\mathbf{T}}(\mathbf{K} + \sigma_n^2\mathbf{I})^{-\mathbf{1}}\mathbf{K}_*$ $[1 + 2n^2 \approx O(n^2)]$

$$[Total\ Complexity = O(nm^2) + 2O(n^2) + O(m^2) \approx O(n^2)]$$

Code

```
#!/usr/bin/python

from numpy import *
from math import *


#COVARIANCE FUNCTION
#Calculates the covariance, i.e. a measure of the similarity between two points
#using the error squared or Gaussian kernel
def cov_func(x1, x2, M, sig_pow):
        diff = x1 - x2
        temp = dot(diff.transpose(), M)
        k = sig_pow*exp(-0.5*dot(temp, diff))
  #print k
        return k




#CALCULATE COVARIANCE MATRIX K
#Uses the covariance function to calculate the covariance matrix of the multivariate gaussian
#distribution of the training data points.
def calc_K(X, theta):
```

```
  #Number of Input features
  m = len(X[0])
  #Number of data points
  n = len(X)

  #Initialise matrix M
  M = zeros((m,m))
  for i in range(m):
    M[i][i] = 1/(theta[i+2]*theta[i+2])

  #Initialise then calculate
  K = zeros((n,n))
  for i in range(n):
          for j in range(i, n):
            K[i][j] = cov_func(X[i].transpose(), X[j].transpose(), M, theta[0])
      K[j][i] = K[i][j]
  return K


#CALCULATE 'C'
#'C' is just the notation for the covariance matrix K plus a noise term that represents
#jitter in the training data
def calc_C(K, noise_pow):
  a = len(K)
  temp = noise_pow*eye(a)
  C = K + temp
  return C


#PARTIAL DIFFERENTIATE C WITH RESPECT TO HYPERPARAMETERS
#Funtion is necessary to carry out gradient descent. Returns a vector containing the
#evaluated partial differential of NLML with respect to each of the hyperparameters
def grad_C(theta, X, K, C):

  #a is the number of hyperparameters
  a = len(theta)
  #n is the number of data points in the training set, note that K and C are nxn
  n = len(C)
  #m is the number of data features
  m = len(X[0])



  #Initialise gradient of C
  partial_diff_C = zeros((a,n,n))

  #Find partial diff with respect to signal power
  partial_diff_C[0] = 2*theta[0]*K

  #Find partial diff with respect to the noise power
  partial_diff_C[1] = 2*theta[1]*eye(n)

  #Find partial diff with respect to the different scale lengths
  for k in range(2,a):
    for i in range(n):
                  for j in range(i, n):
        diff = (X[i][k-2] - X[j][k-2])**2
              l = theta[k]
        temp = (diff/(l**3))
                    partial_diff_C[k][i][j] = temp*K[i][j]
                          partial_diff_C[k][j][i] = partial_diff_C[k][i][j]#Note that C and K are symmetric, therefore only need to calculate half the elements

  return partial_diff_C



#CALCULATE THE GRADIENT OF THE NEGATIVE LOG MARGINAL LIKLIHOOD (NLML) WITH RESPECT TO ONE OF THE FEATURES
#Calculates the partial differential of the NLML with respect to a single hyper parameter
def calc_pdiff_nlml(C, y, pdiff_C):

  inv_C = linalg.inv(C)
  diff_L = 0.5*(trace(dot(inv_C, pdiff_C)) - dot(y.transpose(), dot(inv_C, dot(pdiff_C, dot(inv_C,y)))))
  return diff_L



#CALCULATE THE NEGATIVE LOG MARGINAL LIKLIHOOD (NLML)
#Calculates the Negative Log Marginal Likelihood
def calc_nlml(C, y):
  n = len(C)
  L = 0.5*(log(linalg.det(C)) + dot(y.transpose(),dot(linalg.inv(C), y)) + n*log(2*pi))
  return L


#GRADIENT DESCENT
#Carry out gradient descent to find a minima to optimize the hyper parameters with respect to
#minimising the NLML.
def gradient_descent(alpha, init_theta, X, y, error_level, max_iters):
  m = len(init_theta)
  L_hist = ones(max_iters+2)
  pdiff_L = zeros(m)
```

```
  theta = init_theta
  j = 0

  K = calc_K(X, theta)
        C = calc_C(K,theta[1])
        L_hist[j] = calc_nlml(C,y)

  successful = True
  while(L_hist[j] > error_level):
    j = j+1
    K = calc_K(X, theta)
    C = calc_C(K,theta[1])
    L_hist[j] = calc_nlml(C,y)
    pdiffCs = grad_C(theta, X, K, C)
    for k in range(m):
      p_diffL = calc_pdiff_nlml(C, y, pdiffCs[k])
      theta[k] = theta[k] - alpha*p_diffL


    if(j>max_iters):
      print("Failure to converge below desired error level after number of iterations =")
      print j
      print("Consider increasing value of alpha to decrease convergence time")
      successful = False
      break

    if(L_hist[j] > L_hist[j-1]):
      print("Failure to converge below required error level, nlml not decreasing with each iteration, local minima
     reached, consider decreasing alpha")
      print("Number of iterations =")
      print j
      successful = False
      break

  if(successful == True):
    print("Convergence below desired error level successful after number of iterations =")
    print j

  return (theta, successful, L_hist[j], L_hist, j)



#CALCULATE K_*
#Calculate the covariance of the input data point with each the points in the training set
def calc_K_star(x, X, theta):

  #Number of Input features
        m = len(X[0])
        #Number of data points
        n = len(X)
        #Initialise matrix M
        M = zeros((m,m))
        for i in range(m):
                M[i][i] = 1/(theta[i+2]*theta[i+2])
  K_star = zeros(n)

  for j in range(n):
    K_star[j] = cov_func(x.transpose(), X[j].transpose(), M, theta[0])
  return K_star

#CALCULATE EXPECTED THERMOSTAT SETTING
def calc_E_TS(inv_C, K_star, y):
  expected_TS = dot(K_star, dot(inv_C, y))
  return expected_TS

#NORMALIZE INPUT DATA BEFORE CARRYING OUT GRADIENT DESCENT
#Normalisation is necessary becuase otherwise differences in the order of magnitude between
#different features would result in a warped (and much slower) path of descent
def normalize_data(X):
  X = X.transpose()
  numfeatures = len(X)
  numdata = len(X[0])
  norm_X = zeros((numfeatures, numdata))

  for j in range(numfeatures):
    u = mean(X[j])
    u = u*ones(numdata)
    stan_dev = std(X[j])
    stan_dev
    norm_X[j] = (1/stan_dev)*(X[j] - u)
  return norm_X.transpose()

#NORMALIZE TRAINING AND INPUT DATA ACCORDING TO TRAINING DATA NORMALISATION
#Normalizes the training data and also the input data. Note the training data is normalized
#first and then the mean and std are taken and used to normalize the input. This ensures the same
#transformation occurs on the input data as on the training data and therfore that the relationship
#with the thermostat setting is the same.
def normalize_input(X, x):
  X = X.transpose()
        numfeatures = len(X)
        numdata = len(X[0])
        norm_X = zeros((numfeatures, numdata))
  norm_x = zeros(numfeatures)
```

```

        for j in range(numfeatures):
                u = mean(X[j])
                u_vec = u*ones(numdata)
                stan_dev = std(X[j])
                stan_dev
                norm_X[j] = (1/stan_dev)*(X[j] - u_vec)
    norm_x[j] = (1/stan_dev)*(x[j] - u)
        return norm_X.transpose(), norm_x


#Function used to find the index and value of an element in an
#array that is closest to the argument 'value'. Example application
#is finding closest time to the present time in a database.
def find_nearest(array,value):

  min_index = 0
  min_diff = array[min_index] - value
  diff = zeros(len(array))
  min_value = array[min_index]

  for j in range(len(array)):
    diff[j] = abs(array[j] - value)
    if(diff[j] <= min_diff):
      min_index = j
      min_diff = diff[min_index]
      min_value = array[j]

  return min_index, min_value, min_diff

#Calculates a 'similarity' metric for two vectors. An example application is in finding entries in the training data base whose
#external conditions best match a new training data point that needs to be swapped in.
def calc_similarity(a, b):
  if(len(a) != len(b)):
    print("Error: cannot calculate similarity between vectors as do not have the same number of elements")
  sum = 0

  for j in range(len(a)):
    sum = sum + abs(a[j] - b[j])

  return sum
```

code/gpr_functions.py

```
#!/usr/bin/python
import MySQLdb
import math
import random
import sqlite3
import time
from functions import *
from websocket import create_connection
from datetime import datetime
from scipy import *
from numpy import *



#GPR_RUN:
#Function reads in training data, trained hyperparameters and inverse C and uses them to calculate
#a thermostat estimate. This is then returned to be sent via websockets back to the embed module.
def active_TS(x):
  #print("active_TS function entered, calculating TS value using Gaussian Process Regression...")
  db = MySQLdb.connect("127.0.0.1","mike","ChickenKorma","GPR_Training_Data" )
  cur = db.cursor()
  #Read in trained hyperparameters to theta from SQL database
  cur.execute("SELECT * FROM trained_theta_values")
  row = cur.fetchall()
  numb_hp = len(row[0])
  theta = zeros(numb_hp)
  for j in range(numb_hp):
          theta[j] = row[0][j]
  #Read in inv_C from SQL database
  cur.execute("SELECT * FROM inv_C")
  rows = cur.fetchall()
  numrows = int(math.sqrt(len(rows)))
  inv_C  = ones((numrows, numrows))
  for j in range(numrows):
          for i in range(numrows):
                  inv_C[j][i] = rows[i][2]
  #Import the training data required in calculation
  cur.execute("SELECT * FROM training_data")
  rows = cur.fetchall()
  numrows = len(rows)
  numcols = len(rows[0])
  X = zeros((numrows, (numcols-1)))
  y = zeros((numrows, 1))
  for j in range((numrows)):
      y[j] = rows[j][0]
  for j in range(numrows):
    for i in range(1, numcols):
```

```
      X[j][(i-1)] = rows[j][i]

  #Normalize Data
  X, x = normalize_input(X, x)

  #Calculate K_star and then calculate expected TS value given input x
  K_star = calc_K_star(x, X, theta)
  m = len(X[0])
  M = zeros((m,m))
  for i in range(m):
    M[i][i] = 1/(theta[i+2]*theta[i+2])
  K_starstar = cov_func(x, x, M, theta[0])
  TS = calc_E_TS(inv_C, K_star, y)
  TS_conf = K_starstar - dot(K_star, dot(inv_C, K_star.transpose()))
  print "Predicted TS: %d, Confidence in prediction: %d" % (TS, TS_conf)
  return TS, TS_conf




#Function is necessary as for practical reasons because the training data set cannot be allowed to grow indefinitly. Instead
#new predicted values have to be swapped in for older training examples. Before they can be considered however new training
#data points have to be cross validated agains the user input log to make sure they were not over ridden. This function takes each
#entry in the user log in turn and removes any entry in potential data entry that is within a certain time the user entry. All potential
#training data points that pass this test are passed into temp to be inserted into the training data set by insert_td.
def validate_TD(sig_time):
  print("Validating potential new training data against user log...")
  db = MySQLdb.connect("127.0.0.1","mike","ChickenKorma","GPR_Training_Data")
  cur = db.cursor()

  clean_time = 86400

  #Import the potential training data
  cur.execute("SELECT * FROM potential_new_training_data")
  rows = cur.fetchall()
  if(len(rows) > 0):
    ptd = zeros((len(rows), len(rows[0])))
    for j in range(len(rows)):
      for i in range(len(rows[0])):
        ptd[j][i] = rows[j][i]

    #Import the user log
    cur.execute("SELECT * FROM user_TS_input_log")
    rows = cur.fetchall()
    user_log = zeros((len(rows), len(rows[0])))
    for j in range(len(rows)):
      for i in range(len(rows[0])):
        user_log[j][i] = rows[j][i]

    #PROCESS
    for j in range(len(ptd)):
      val_pass = True
      for i in range(len(user_log)):
        time_diff = (user_log[i][0] - ptd[j][6])
        print time_diff
        if(time_diff < 0):
          val_pass = True
        elif((time_diff > 0) & (time_diff<(sig_time*60))):
          val_pass = False
          break
        elif(time_diff > (sig_time*60)):
          val_pass = True
      print("Validation test result: ", val_pass)

      if(val_pass == True):
        print("Entry in potential new training data with following id passed: ", j)
        command = "INSERT INTO temp (thermostat_setting, external_temp, external_humidity, internal_humidity, wind_speed, wind_direction) VALUES ('%s', '%s', '%s', '%s', '%s', '%s')"%(ptd[j][0], ptd[j][1], ptd[j][2], ptd[j][3], ptd[j][4], ptd[j][5])
        print("Sending following command to database to add entry to table 'temp': ", command)
        cur.execute(command)

    print("Operation done, all entries that passed vaildation test moved to 'temp' for insertion into 'training data'.")
    cur.execute("TRUNCATE TABLE potential_new_training_data")
    db.close()

  print("Removing old data entries in user log")
  #Get current timestamp (ms)
  current_timestamp = int(time.time())
  time_bound = current_timestamp - clean_time
  command = "DELETE FROM user_TS_input_log WHERE time_stamp < '%d'" % int(time_bound)
  cur.execute(command)
  print("Validation process complete")


#Function is necessary as for practical reasons because the training data set cannot be allowed to grow indefinitly. Instead
#new predicted values have to be swapped in for older training examples. This function takes all the data points that
```

```
      pass
#the validation test and adds as many of them as it can to the training set till capacity is reached. After this new training examples
#are swapped in by removing the training points that are closest in the feature space.
def insert_new_td(desired_training_set_size):
  print("INSERTING NEW DATA INTO THE TRAINING DATA SET...")
  db = MySQLdb.connect("127.0.0.1","mike","ChickenKorma","GPR_Training_Data")
  cur = db.cursor()

  #Import the number of new entries
  cur.execute("SELECT COUNT(*) FROM temp")
  result=cur.fetchone()
  numb_new_entries = result[0]
  print("Number of new entries: ", numb_new_entries)

  #Import new data to be added to training data if it exists
  if(numb_new_entries > 0):
    cur.execute("SELECT * FROM temp")
    rows = cur.fetchall()
    temp = zeros((len(rows), len(rows[0])))
    for j in range(len(rows)):
      for i in range(len(rows[0])):
        temp[j][i] = rows[j][i]
    print("Data to be inserted: ", temp)

    #Import the current training data
    cur.execute("SELECT * FROM training_data")
    rows = cur.fetchall()
    numrows = len(rows)
    numcols = len(rows[0])
    X = zeros((numrows, numcols))
    for j in range((numrows)):
      for i in range(numcols):
        X[j][i] = rows[j][i]
    X_norm = normalize_data(X)#Normalize to ensure each feature has equal significance in the sum
    numb_training_points = len(X)


    if(numb_new_entries <= abs(desired_training_set_size - numb_training_points)):#Automatically insert all points if there is enough capacity in the training set.
      print("TRAINING SET NOT AT CAPACITY - INSERTING ALL NEW POINTS")
      for j in range(len(temp)):
        cur.execute("INSERT INTO training_data (thermostat_setting, external_temp, external_humidity, internal_humidity, wind_speed, wind_direction) VALUES (%s, %s, %s, %s, %s, %s)", (temp[j][0], temp[j][1], temp[j][2], temp[j][3], temp[j][4], temp[j][5]))

    else:
      print("TRAINING SET NEAR CAPACITY - REPLACING OBSELETE POINTS")#Automatically insert new points till capacity is reached
      numb_auto_insert = abs(desired_training_set_size - numb_training_points)
      print("Number to auto-insert into the training data set: ", numb_auto_insert)
      if(numb_auto_insert > 0):
        for j in range(numb_auto_insert):
          cur.execute("INSERT INTO training_data (thermostat_setting, external_temp, external_humidity, internal_humidity, wind_speed, wind_direction) VALUES (%s, %s, %s, %s, %s, %s)", (temp[j][0], temp[j][1], temp[j][2], temp[j][3], temp[j][4], temp[j][5]))

      for j in range(numb_auto_insert, len(temp)):#Swap in new training points for old ones that are closest to them in the feature space.
        similarities = zeros(len(X))
        for i in range(len(X_norm)):
          similarities[i] = calc_similarity(temp[j], X_norm[i])
        index_replace = similarities.argmax()
        command = "UPDATE ‘training_data‘ SET ‘thermostat_setting‘='%s', ‘external_temp‘='%s', ‘external_humidity‘='%s', ‘internal_humidity‘='%s', ‘wind_speed‘='%s', ‘wind_direction‘='%s' WHERE ‘thermostat_setting‘='%s' AND ‘external_temp‘='%s' AND ‘external_humidity‘='%s' AND ‘internal_humidity‘='%s' AND ‘wind_speed‘='%s' AND ‘wind_direction‘='%s';" % (temp[j][0], temp[j][1], temp[j][2], temp[j][3], temp[j][4], temp[j][5], X[index_replace][0], X[index_replace][1], X[index_replace][2], X[index_replace][3], X[index_replace][4], X[index_replace][5])
        cur.execute(command)
        print("Sent following command to sequel database: ", command)
    cur.execute("TRUNCATE TABLE temp")
  db.close()


#GPR TRAIN:
#Function trains the GPR by optimizing hyperparameters using gradient descent.
def train_GPR(alpha, max_iters, error_level, init_lower, init_upper, num_init):
  print("TRAINING GPR ALGORITHM")
  #READ IN TRAINING DATA
  # Open database connection
  db = MySQLdb.connect("127.0.0.1","mike","ChickenKorma","GPR_Training_Data" )
  # prepare a cursor object using cursor() method
  cur = db.cursor()
  #Read in training data from SQL database
  cur.execute("SELECT * FROM training_data")
  rows = cur.fetchall()
  # disconnect from server
  db.close()

  #PUT DATA INTO MATRICES
  #Find the number of rows (data points) and columns (Features
  numrows = len(rows)
  numcols = len(rows[0])
```

```
  #Process data into input feature data matrix X and actual outputs associated y
  X = zeros((numrows, (numcols-1)))
  y = zeros((numrows, 1))
  for j in range((numrows)):
      y[j] = rows[j][0]

  for j in range(numrows):
    for i in range(1, numcols):
      X[j][(i-1)] = rows[j][i]

  #NORMALIZE DATA
  X = normalize_data(X)

  #OPTIMISE AND DEFINE HYPERPARAMETERS USING GRADIENT DESCENT
  #Number of Input features
  m = len(X[0])
  #Number of data points
  n = len(y)

  theta_init = random.uniform(init_lower,init_upper,(m+2))
  theta, successful, nlml, nlml_history, exit_iters = gradient_descent(alpha, theta_init, X, y, error_level, max_iters
     )

  #Carry out gradient descenet multiple times as with different initialisations may get different local minima and
     therefore
  #worse or better performance. Select the set of hyperparameters that gives the lowest NLML.
  for j in range(num_init):
    print("j is: ", j)
    #Define and initialise theta, note is a row vector
    theta_init = random.uniform(init_lower,init_upper,(m+2))
    print("Initial Theta values: ", theta_init)
    #Use gradient descent to find optimized parameters
    theta_test, successful, nlml_test, nlml_history, exit_iters = gradient_descent(alpha, theta_init, X, y,
     error_level, max_iters)
    print("Initial nlml:", nlml_history[0])
    print("Final nlml: ", nlml)
    if(nlml_test < nlml):
      theta = theta_test
      nlml = nlml_test
  print("Theta optimized: ", theta)


  #CALCULATE MODEL VALUES
  #Calculate K with optimized hyperparameters
  K = calc_K(X, theta)
  #Calculate 'C'
  C = calc_C(K, theta[1])
  #Calculate inverse of C
  inv_C = linalg.inv(C)
  #OUTPUT INVERSE C AND THETA VALUES TO DATABASE TABLES
  # Open database connection
  db = MySQLdb.connect("127.0.0.1","mike","ChickenKorma","GPR_Training_Data" )
  cursor = db.cursor()
  #Empty table of obselete theta values
  cursor.execute("TRUNCATE TABLE trained_theta_values")
  #Insert trained theta values into the database
  cursor.execute("INSERT INTO trained_theta_values (signal_power, noise_power, external_temp, external_humidity,
     internal_humidity, wind_speed, wind_direction) VALUES (%s, %s, %s, %s, %s, %s, %s)", (theta[0], theta[1], theta
     [2], theta[3], theta[4], theta[5], theta[6]))
  #Empty table of obselete inv_C values
  cursor.execute("TRUNCATE TABLE inv_C")
  #Insert inv_C values into the database
  for j in range(n):
    for i in range(n):
      cursor.execute("INSERT INTO inv_C (row_number, column_number, value) VALUES (%s, %s, %s)", (j, i, inv_C[j][i]))
  db.close()
  print("GPR Algorithm trained")


#Function is called everytime a thermostat setting is requested.
#Description: Function reads in input data and recent user input tables. If the predicted state is occupied, and there
      has been a user input (i.e. a user input since the last call of this function) then
#the user input is used, otherwise former inputs from the user (found in the user log table) are used if they are
     within a certain  time range of the current timestamp. Else a TS value is generated by
#the GPR algorithm. If the state is 'empty' or 'asleep' default settings are adopted.
def get_TS(internal_humidity):
  #DEFINE VARIABLES USED IN PROGRAM
  #k is time in minutes that a user update holds for
  alpha = 0.0000001
  error_level = 1
  max_iters = 100
  numb_init = 10
  k = 60
  unoccupied_temp = 2
  desired_training_set_size = 500
  #Initialise TS
  TS = 0

  #GET HYPERPARAMETER INITIALISATION RANGE
  db = MySQLdb.connect("127.0.0.1","mike","ChickenKorma","algorithm_testing" )
  cur = db.cursor()
  cur.execute("SELECT * FROM hyperparameter_range")
  row = cur.fetchall()
```

```
  lb = row[0][0]
  ub = row[0][1]
  db.close()

  #GET DATA REQUIRED
  db = MySQLdb.connect("127.0.0.1","mike","ChickenKorma","Data" )
  cur = db.cursor()
  #Import user settings
  cur.execute("SELECT * FROM limits")
  row = cur.fetchall()
  max_temp = row[0][0]
  min_temp = row[0][1]
  sleeping_temp = row[0][2]
  print("IMPORTED USER SETTINGS: ")
  print("max_temp is: ", max_temp)
  print("min_temp is: ", min_temp)
  print("sleeping_temp is: ", sleeping_temp)

  #Import the input for this call
  x, predicted_state, current_timestamp = get_input(internal_humidity)

  print("DATA TO BE USED TO GENERATE TS IF NO USER INPUT")
  print("New input data entry: ",x)
  print("State: ",predicted_state)
  print("Timestamp: ", current_timestamp)

  print("ASCERTAINING IF THERE HAS BEEN A USER UPDATE...")
  #Ascertain if there has been a recent user update
  cur.execute("SELECT * FROM user_update")
  rows = cur.fetchall()
  numb_recent_updates = len(rows)
  print("Number of user updates in last k minutes: ", numb_recent_updates)

  if(numb_recent_updates > 0):
    new_user_update = True
    user_update = zeros((len(rows), len(rows[0])))
    for j in range(len(rows)):
      for i in range(len(rows[0])):
        user_update[j][i] = rows[j][i]
      print("user update data: ", user_update)

  else:
    new_user_update = False


  print("State of user update: ", new_user_update)
  db.close()

  #PROCESS
  #If the user has made a change since the last call then use this
  if(new_user_update == True):
    print("USER UPDATE DETECTED - USER INPUT PATH TAKEN")
    if(predicted_state == 0):
      print("State: Unoccupied, setting TS to unoccupied setting...")
      TS = unoccupied_temp
    elif(predicted_state == 1):
      print("State: Occupied, setting TS to user setting...")
      TS = user_update[(numb_recent_updates-1)][0]
    elif(predicted_state == 2):
      print("State: Asleep, setting TS to sleeping setting...")
      TS = sleeping_temp
    else:
      print("Error: 'predicted_state' value not valid:", predicted_state)


    if(TS > max_temp):
      TS = max_temp

    elif(TS < min_temp):
      TS = min_temp

    else:
      print("TS within boundry conditions")

    print("TS value selected",TS)

    #OUTPUT TS TO THE EMBED
    print("OUTPUT TO EMBED...")
    ws = create_connection("ws://ec2-54-214-164-65.us-west-2.compute.amazonaws.com:8888/ws")
    print("Sending Thermostat Setting...")
    command = '{"type": "thermostat_control" , "setting" : %d}' % int(TS)
    print("Output to embed: ", command)
    ws.send(command)
    print("Thermostat Setting Sent with value of: ", TS)
    ws.close()


    #HANDLE DATABASE
    db = MySQLdb.connect("127.0.0.1","mike","ChickenKorma","GPR_Training_Data" )
    cur = db.cursor()
    #Upload the user input to the temp file and retrain GPR algorithm
    cur.execute("INSERT INTO temp (thermostat_setting, external_temp, external_humidity, internal_humidity, wind_speed
     , wind_direction) VALUES (%s, %s, %s, %s, %s, %s)", (user_update[(numb_recent_updates-1)][0], user_update[(
```

```
numb_recent_updates-1)][1], user_update[(numb_recent_updates-1)][2], user_update[(numb_recent_updates-1)][3], user_update[(numb_recent_updates-1)][4], user_update[(numb_recent_updates-1)][5]))
    #Upload the user input to the user log
    command = "INSERT INTO user_TS_input_log (thermostat_setting, external_temp, external_humidity, internal_humidity, wind_speed, wind_direction, time_stamp) VALUES (%s, %s, %s, %s, %s, %s, %s)" % (user_update[(numb_recent_updates-1)][0], user_update[(numb_recent_updates-1)][1], user_update[(numb_recent_updates-1)][2], user_update[(numb_recent_updates-1)][3], user_update[(numb_recent_updates-1)][4], user_update[(numb_recent_updates-1)][5], user_update[(numb_recent_updates-1)][6])
    print("Uploading following entry into user log: ", command)
    cur.execute(command)
    #Retrain Algorithm and truncate 'temp' afterwards
    insert_new_td(desired_training_set_size)
    train_GPR(alpha, max_iters, error_level, lb, ub, numb_init)
    db.close()

    db = MySQLdb.connect("127.0.0.1","mike","ChickenKorma","Data" )
    cur = db.cursor()
    #Truncate user_data (user input)
    cur.execute("TRUNCATE TABLE user_update")
    #Truncate Data (input)
    cur.execute("TRUNCATE TABLE Data")


  #NO INPUT SINCE LAST CALL...
  else:
    print("USER UPDATE NOT DETECTED - CHECKING FOR FORMER STILL RELEVANT USER INPUTS...")

    #Import user log data
    db = MySQLdb.connect("127.0.0.1","mike","ChickenKorma","GPR_Training_Data" )
    cur = db.cursor()
    cur.execute("SELECT * FROM user_TS_input_log")
    rows = cur.fetchall()
    numb_user_entries = len(rows)
    user_time_stamps = zeros(numb_user_entries)

    #Load in the time stamps in the user log
    for j in range(numb_user_entries):
      user_time_stamps[j] = rows[j][0]

    print("Numer of entries in user log: ", numb_user_entries)
    print("User Log timestamps:", user_time_stamps)

    if(numb_user_entries > 0):
      #Check if any entries in the user log are relevant
      id, closest_user_timestamp, min_diff = find_nearest(user_time_stamps,current_timestamp)
      if(fabs(current_timestamp - closest_user_timestamp) < (k*60)):#Note timestamp is in milliseconds, k*60*10^3 ms in k minutes
        use_user_input = True

      else:
        use_user_input = False

      print("Current Time Stamp: ", current_timestamp)
      print("Closest time difference to current timestamp: ", min_diff)

    else:
      use_user_input = False

    print("State of use user log input: ", use_user_input)

    #Process
    if(predicted_state == 0):
      print("State: Unoccupied, setting TS to unoccupied setting")
      TS = unoccupied_temp
    elif(predicted_state == 1):
      if(use_user_input == True):
        print("Former user input still relevant, referring back to last user log entry...")
        TS = user_TS(current_timestamp)
      else:
        print("Former user entries no longer relevant, generating TS from GPR algorithm...")
        TS, TS_conf = active_TS(x)
    elif(predicted_state == 2):
      print("State: Asleep, setting TS to sleeping setting...")
      TS = sleeping_temp
    else:
      print("Error: 'predicted_state' value not valid:")
      print predicted_state

    #Bound the TS setting
    if(TS > max_temp):
      TS = max_temp

    elif(TS < min_temp):
      TS = min_temp
    else:
      print("TS within boundry conditions")

    #OUTPUT TS TO THE EMBED
    print("OUTPUT TO EMBED...")
    ws = create_connection("ws://ec2-54-214-164-65.us-west-2.compute.amazonaws.com:8888/ws")
    print("Sending Thermostat Setting...")
    command = '{"type": "thermostat_control" , "setting" : %d}' % int(TS)
    print("Output to embed: ", command)
```

```
    ws.send(command)
    print("Thermostat Setting Sent with value of: ", TS)
    ws.close()


    #HANDLE DATABASE UPDATES
    if((use_user_input == False) & (predicted_state == 1)):
      db = MySQLdb.connect("127.0.0.1","mike","ChickenKorma","GPR_Training_Data" )
      cur = db.cursor()
      #Upload input to potential new training data
      cur.execute("INSERT INTO potential_new_training_data (thermostat_setting, external_temp, external_humidity, internal_humidity, wind_speed, wind_direction, time_stamp) VALUES (%s, %s, %s, %s, %s, %s, %s)", (TS, x[0][0], x[0][1], x[0][2], x[0][3], x[0][4], current_timestamp))
      db.close()




#Function takes all the data in the potential_training_data table, runs a validation check against the user_TS_input_log and then outputs the data entries
#that pass into the 'temp' data table these are then inserted into the training data table and the GPR retrain algorithm is re-run. The temp and potential data table are then truncated
#and all entries older than a day in the user log are also removed. Note user inputs are automatically added to the training set as soon as they are detected. Called approximatly
#every 24 hours.
def retrain_with_updates():
  #GET HYPERPARAMETER INITIALISATION RANGE
  db = MySQLdb.connect("127.0.0.1","mike","ChickenKorma","algorithm_testing" )
  cur = db.cursor()
  cur.execute("SELECT * FROM hyperparameter_range")
  row = cur.fetchall()
  lb = row[0][0]
  ub = row[0][1]
  db.close()

  #PARAMETER DEFINITIONS
  alpha = 0.0000001
  error_level = 1
  max_iters = 100
  numb_init = 10
  sig_time = 60
  desired_training_set_size = 500

  #Check data against user log entries, if within a certain time distance then don't insert into the table 'temp'. Once complete truncates 'potential_training_data' and removes old entries in user log
  validate_TD(sig_time)

  #Insert data in 'temp' table into the 'training_data' table appropriatly. Once complete truncates 'temp'
  insert_new_td(desired_training_set_size)

  #Retrain Algorithm, i.e. obtain new theta values and correlation matrix
  train_GPR(alpha, max_iters, error_level, lb, ub, numb_init)



#HYPERPARAMETER SEARCH:
#Different initialisations of theta lead to different local minima and therefore different performance. This function searches the NLML space and initializes
#the hyperparameters in sub regions. It checks the performance of each region by comparing the predictions with the actual values from a test set (used to train
#the data) and a cross validation set (an independant set) and calculates the percentage error for both. The region with the best performance is then subdivided again
#until a certain initialisation range is reached. It then returns this initialisation range to the database to be used by all future 'train calls' to initialise the hyperparameters.#
#This function is simply put a range finder and gives an idea of where to look for the best minima, it is only called when there are very large changes in the data set.
def hyperparameter_search():
  #INITIALISE SYSTEM PARAMETERS
  alpha = 0.00001
  error_level = 1
  max_iters = 50
  num_init = 20

  #IMPORT INPUT DATA
  db = MySQLdb.connect("127.0.0.1","mike","ChickenKorma","algorithm_testing" )
  cur = db.cursor()

  #Read in training data
  cur.execute("SELECT * FROM training_data")
  rows = cur.fetchall()
  train_d = zeros((len(rows), len(rows[0])))
  for j in range(len(rows)):
    for i in range(len(rows[0])):
      train_d[j][i] = rows[j][i]

  #Read in test data
  cur.execute("SELECT * FROM test_data")
  rows = cur.fetchall()
  test_d = zeros((len(rows), (len(rows[0])-1)))
  TS_test = zeros((len(rows)))
  #print len(rows)
  for j in range(len(rows)):
    TS_test[j] = rows[j][0]
```

```
    for i in range(1,len(rows[0])):
      test_d[j][(i-1)] = rows[j][i]

  lower_bound = 0
  upper_bound = 100
  numb_divs = 10
  m = len(test_d[0])

  #Perform the grid search
  while((upper_bound - lower_bound) > 4):
    increment = (upper_bound - lower_bound)/numb_divs
    print "Considering range of initialisations %d to %d:" % (lower_bound, upper_bound)
    print "Increment is: %d" % (increment)
    lb_temp = lower_bound
    ub_temp = lb_temp + increment
    av_err = zeros((numb_divs))
    for k in range((numb_divs)):
      #TRAIN ALGORITHM:
      train_GPR(alpha, max_iters, error_level, lb_temp, ub_temp, num_init)
      #Initialise
      err = zeros((len(TS_test)))
      gpr_res = zeros((len(TS_test)))
      gpr_conf = zeros((len(TS_test)))
      for j in range(len(test_d)):
        gpr_res[j], gpr_conf[j] = active_TS(test_d[j])
        err[j] = 100*((abs(TS_test[j] - gpr_res[j]))/TS_test[j])
        av_err[k] = av_err[k] + err[j]

      av_err[k] = av_err[k]/(len(test_d))
      print "Error bound for division %d-%d is: %d" % (lb_temp, ub_temp, av_err[k])
      lb_temp = ub_temp
      ub_temp = ub_temp + increment

    ind_min_err = np.argmin(av_err)
    print "the index of the minimum division is: %d" % (ind_min_err)
    upper_bound = lower_bound + (ind_min_err*increment)
    lower_bound = upper_bound - increment
    print "Theta values found to be best initialised in this range are between %d - %d" % (lower_bound, upper_bound)


  print "Theta values found to be best initialised in this range are between %d - %d" % (lower_bound, upper_bound)
  print("Initialised with these Boundaries we have...")

  train_GPR(alpha, max_iters, error_level, lower_bound, upper_bound,num_init)
  err = zeros((len(TS_test)))
  gpr_res = zeros((len(TS_test)))
  gpr_conf = zeros((len(TS_test)))
  av_err = 0
  for j in range(len(test_d)):
    gpr_res[j], gpr_conf[j] = active_TS(test_d[j])
    err[j] = 100*((abs(TS_test[j] - gpr_res[j]))/TS_test[j])#Calculate error on each prediction
    av_err= av_err + err[j]
  av_err = av_err/len(test_d)#Calculate the % error in the test set

  min = np.argmin(err)
  max = np.argmax(err)

  print("TS values from test set", TS_test)
  print("TS values calculated", gpr_res)
  print("Variance of GPR values", gpr_conf)
  print("Percentage Error List:", err)
  print "Max percentage error in estimate: %d" % (err[max])
  print "Min percentage error in estimate: %d" % (err[min])
  print "Average Percentage Error in estimates: %d" % (av_err)

  cur.execute("TRUNCATE TABLE hyperparameter_range")
  cur.execute("INSERT INTO hyperparameter_range (lower_bound, upper_bound) VALUES (%s, %s)" % (lower_bound,
     upper_bound))






#With insufficient data to run the state detection algorithm this function is used to generate values
def estimate_current_state():
  x = random.randint(1,100)
  if x <= 10:
    return 0
  elif x <= 66:
    return 1
  else:
    return 2



#Function is run before 'get_TS' is called to compile an input vector
def get_input(internal_humidity):
  db = MySQLdb.connect("127.0.0.1","mike","ChickenKorma","Data" )
  cur = db.cursor()
  #Import enviromental data from Data2
  command = "SELECT Temperature, Humidity, Wind_speed, Wind_direction FROM Data2 WHERE Time_stamp=(select MAX(
     Time_stamp) from Data2)"
```

```
  print("Command sent to SQL database: ", command)
  cur.execute(command)
  temp = cur.fetchone()
  print temp
  #Get the state of the house:
  state = find_state()
  #Get current Time stamp
  current_timestamp = int(time.time())/1000
  #Format Oututs:
  x = zeros((5))
  x[0] = temp[0]
  x[1] = temp[1]
  x[2] = internal_humidity
  x[3] = temp[2]
  x[4] = temp[3]
  return x, state, current_timestamp




#Function uses a logic table to select a state by comparing the predicted and detected state. It then returns this
    state to
#the 'get_TS' function.
def find_state():
  localtime = time.localtime(time.time())
  print "Day is: %d" % (localtime[6])
  print "Hour is: %d" % (localtime[3])

  hour = localtime[3]

  db = MySQLdb.connect("127.0.0.1","mike","ChickenKorma","House_states" )
  cur = db.cursor()

  #First need find the correct day, the data is puleld of the same table that feeds the website.
  if(localtime[6] == 0):
    command = "SELECT state FROM Monday WHERE Hour =  '%d'" % (hour)
    print("Command sent to SQL database: ", command)
    cur.execute(command)
    temp = cur.fetchone()
    pred_state = int(temp[0])

  elif(localtime[6] == 1):
    command = "SELECT state FROM Tuesday WHERE Hour =  '%d'" % (hour)
    print("Command sent to SQL database: ", command)
    cur.execute(command)
    temp = cur.fetchone()
    pred_state = int(temp[0])

  elif(localtime[6] == 2):
    command = "SELECT state FROM Wednesday WHERE Hour =  '%d'" % (hour)
    print("Command sent to SQL database: ", command)
    cur.execute(command)
    temp = cur.fetchone()
    pred_state = temp[0]

  elif(localtime[6] == 3):
    command = "SELECT state FROM Thursday WHERE Hour =  '%d'" % (hour)
    print("Command sent to SQL database: ", command)
    cur.execute(command)
    temp = cur.fetchone()
    pred_state = temp[0]

  elif(localtime[6] == 4):
    command = "SELECT state FROM Friday WHERE Hour =  '%d'" % (hour)
    print("Command sent to SQL database: ", command)
    cur.execute(command)
    temp = cur.fetchone()
    pred_state = temp[0]

  elif(localtime[6] == 5):
    command = "SELECT state FROM Saturday WHERE Hour =  '%d'" % (hour)
    print("Command sent to SQL database: ", command)
    cur.execute(command)
    temp = cur.fetchone()
    pred_state = temp[0]

  elif(localtime[6] == 6):
    command = "SELECT state FROM Sunday WHERE Hour =  '%d'" % (hour)
    print("Command sent to SQL database: ", command)
    cur.execute(command)
    temp = cur.fetchone()
    pred_state = temp[0]
  else:
    print "Day not recognized, day = '%d'" % (localtime[6])
  db.close()

  rt_state = estimate_current_state()
  print type(rt_state)
  print type(pred_state)
#Logic table
  if(rt_state == 0 and (pred_state == 0 or pred_state == 3)):
    state = 0
  elif(rt_state == 0 and (pred_state == 1 or pred_state == 4)):
```

```
    state = 1
  elif(rt_state == 0 and (pred_state == 2 or pred_state == 5)):
    state = 0
  elif(rt_state == 1 and (pred_state == 0 or pred_state == 3)):
    state = 1
  elif(rt_state == 1 and (pred_state == 1 or pred_state == 4)):
    state = 1
  elif(rt_state == 1 and (pred_state == 2 or pred_state == 5)):
    state = 2
  elif(rt_state == 2 and (pred_state == 0 or pred_state == 3)):
    state = 2
  elif(rt_state == 2 and (pred_state == 1 or pred_state == 4)):
    state = 1
  elif(rt_state == 2 and (pred_state == 2 or pred_state == 5)):
    state = 2
  else:
    print "Real time and predicted state not recognised, rt_state = '%d', pred_state = '%d'" % (rt_state, pred_state)

  return state

random.seed()
```

code/gpr_system_primary_functions.py

8.3 Costs

Component	Price/unit	Quantity	Total price
Wifly module	£22.41	1	£22.41
Mbed	£41.38	1	£41.38
Motion sensor	£12.24	1	£12.24
Zigbee	£17.55	4	£70.20
Temperature Sensor	£1.21	1	£1.21
Breadboard	£3.15	5	£15.75
XBee to DIP Adapter	£2.50	4	£10.00
Humidity Sensor	£3.42	1	£3.42
Amega88 MCU	£2.66	1	£2.66
9V AC-AC adapter	£6.25	1	£6.25
2.1mm socket	£0.65	1	£0.65
Triac	£2.70	2	£5.40
Optocoupler	£1.07	1	£1.07
TL072 opamp	£0.68	2	£1.36
100W SPOT ES	£0.32	1	£0.32
Comparator	£0.17	4	£0.68
XOR gate	£0.26	2	£0.52
DC power adapter	£8.52	1	£8.52
Lead kettle	£2.20	1	£2.20
Voltage regulator	£0.60	1	£0.60
Small breadboard	£3.00	1	£3.00
Voltage regulator	£0.60	2	£1.20
2.1mm socket	£0.65	2	£1.30
		Total	£212.34