Prototype of a wireless monitoring system for a gas turbine engine

Goldsmith, D. , Gaura, E. , Brusey, J. , Shuttleworth, J. and Poole, N.

COGENT
computing

Prototype of a Wireless Monitoring System for a Gas Turbine Engine

Daniel Goldsmith, Elena Gaura, James Brusey, James Shuttleworth, and Nigel Poole

{goldsmid, j.brusey, e.gaura, j.shuttleworth,esx040}@coventry.ac.uk

A critical aspect of modern condition-based maintenance (CBM) systems is the provision of detailed, accurate and reliable sensing for the part, or subsystem under observation. Existing thermocouple based flow sensing systems for gas turbine engines deal effectively with the multipoint measurement of the high temperatures involved, however, due to the need for low added weight, they only provide averaged temperature data over a single heavy duty cable. The conflicting measurement system requirements of low weight and detailed, high rate, robust, multipoint measurement can be mitigated through the use of wireless instrumentation. This technical report describes a prototype system that explores the use of multiple wirelessly networked sensors to deliver detailed spatial-temporal flow temperature information to enable CBM and enhance informational output from engine testing.

This work forms the initial real-world wireless sensor network deployment case study within of the one author's PhD program (Dan Goldsmith) and was also contributed to by the supervisory team, which includes: Dr James Brusey, Dr James Shuttleworth, and Dr Elena Gaura.

Technical report number COGENT.005
Cogent Computing Applied Research Centre

Coventry University

1 Introduction

1.1 Problem Statement

Measurement of temperature in a gas turbine engine is critical to its control and the assessment of its health and performance. Currently, gas temperature is measured predominantly by thermocouples installed at a number of sites within the engine. For example, in the exhaust region of the engine, the temperature is measured at different circumferential (and often radial) positions via an array of thermocouples connected through harness cabling. Transmission of individual thermocouple data to the central control unit would require many individual cables and so, due to weight restrictions, measurements are averaged before transmission over a single heavy duty cable to the central control unit. Not only does this preclude the determination of a detailed picture of the engine gas temperature, which could indicate potential engine problems, but also prevents the diagnosis of individual sensor faults, de-calibration, and sensor drift. A wireless instrumentation system could substantially increase the complexity of the data that could be sent to the engine control unit and hence enable more sophisticated engine health monitoring. Replacing cables with wireless transmissions will reduce the monitoring system weight and, given the availability of detailed temperature profiles for engine control, lead to improved fuel efficiency and reduced carbon emissions. On-line statistical analysis of data from such a wireless system could also permit a clearer understanding of engine/aircraft health. The system proposed here will allow condition-based maintenance, whereby maintenance can be scheduled according to actual wear and usage rather than at fixed intervals, thereby reducing through-life costs. In addition, a wireless system could allow for the sensors in the network to communicate their “health metrics” with each other, in turn allowing faults and drift to be identified and possibly corrected for in the engine control systems. This would give much greater confidence in the accuracy of the measured temperature and could, potentially, allow the engine to run with less safety margin and, therefore, more efficiently (with similar benefits on fuel consumption and emissions).

However, embedding wireless technology into an aerospace or industrial gas turbine will have some significant challenges to overcome, particularly for aero-engines, which require a high degree of safety assurance and certification. With regard to temperature measurement, for example, the temperatures outside the casing of the engine can reach in excess of 250°C, precluding the use of most conventional silicon-based electronic systems. Moreover, maintaining the integrity of an RF signal transmission in an environment that is largely composed of metal whilst not interfering with (or having interference from) other electronic equipment will present major hurdles. Powering the sensors also presents a significant challenge as battery power is not appropriate, hence some means of energy harvesting will be required. However, if these hurdles can be overcome, the benefits to engine management will be significant and could also pave the way for use with other types of engine sensors such as vibration sensors, tip clearance and speed sensors. With a view of establishing a proof of concept for the application at hand, an end-to-end instrumentation system prototype has been designed, implemented and evaluated. The system consists primarily of a wireless sensor network with five nodes, 24 sensors, and a back-end system for receiving, storing, analysing, and visualising collected data. The end-to-end system has been successfully deployed on a cold jet-pipe section and detailed testing carried out. The project, within which the work described here represents the first step is funded as a EPSRC CASE Studentship through the Integrated products Manufacturing KTN and it conducted in collaboration with Vibro-Meter UK Ltd and TRW

Conekt Ltd (suppliers and developers of engine temperature sensors and harsh environments electronics respectively).

1.2 Prototype Goals

Building a proof of concept demonstrator of an end-to-end health monitoring wireless networked system has been motivated by several factors, in two distinct categories:

A) User led, application specific goals The demonstrator's goals were to allow initial feasibility and wireless technologies suitability analyses for the real-life problem at hand. The study has been commissioned by the industrial partners in the research, TRW Conekt and Vibro-Meter UK. The primary aim here was to build a table-top technology demonstrator which could ease the identification of further research, implementation and deployment issues. Solutions to these issues would lead to the development of an engine-deployable health monitoring wireless instrument for gas turbines. Secondly, the demonstrator, set as an end-to-end system, was seen as an enabler of system requirements gathering for the instrument design and development. Given the novelty of wireless networked sensing technologies, it was found that defining system requirements starting from a fully working (albeit not deployable in uncontrolled environments) end-to-end system demonstrator will shorten the instrument design cycle, by capturing the essential instrument features either in contrast with implemented functionality or through similarity with such functionality. Indeed, within the design cycle of the demonstrator, such a requirements gathering phase has been run twice – firstly as a "wish list" from the application specialists (the two industrial partners) and, secondly, mid-way through the development by observing and discussing the implemented functionality of a very basic, end-to-end demonstrator, precursor of the system described in this report. The sharpness of focus on the requirements at this stage was indeed observed, by the design team, to be higher.

B) Researchers led, software engineering goals This motivational category is somewhat richer and forms a key element in the research carried by the authors towards an unified architectural framework for the development of WSNs for monitoring applications. The architectural framework aims to facilitate practical high-level deployment, maintenance and development of WSNs. The research methodology adopted involves a succession of system design/implementation / deployment cycles with the requirement that, at each stage, both a clear set of design requirements and testing plans exist, gradually allowing the core functionality of WSN for monitoring applications to be encapsulated into an architectural framework. This research methodology should allow the scope of the framework to be modified to take into account new advances in the fast moving field of WSN research. Moreover, by implementing a series of "real-world" WSN monitoring applications the results from each stage of experimentation can be compared to those in the literature, allowing evaluation of the framework.

Considering the above, the specification of intent and the evaluation of the demonstrator would be as follows:

- the demonstrator needs to be a fully-functional end-to-end system, developed using best software

engineering practise and fully exploiting prior, published research in wireless sensor networks system design;

- at least one novel service has to be incorporated in the end-to-end system, beyond the “sense and send” state of the art; information visualisation and fault handling and management have been chosen to be integrated as novel services;
- the system should seamlessly integrate debugging and performance analysis tools at several levels, within all functional components;
- the system should allow “plug and play” of several functional components; this needs to be ensured both at core functional components level (to allow for refined components to be tried out without redesign) and “additional functionality” level (for example, a “calibration” component might be plugged in and out, depending on the use of the system; similarly, various “visualisation” modes should be plug-able);
- the design should allow clear assessment of the value of the novel middleware developed which glues the functional components together and also allow encapsulation of core functionality ;
- three ways evaluation of the system design versus application requirements and versus system performance should enable identification of some design patterns to be of further use in research towards structuring functional requirements for an architectural framework for monitoring applications;
- the system should enable organic growth both in terms of its networking component and in terms of refinement of application requirements; for example, the system should have a scalable architecture with opportunity to add or reduce node numbers, sensors and sensor types; the design should allow for smooth adaptation to functional specification changes (such as sensor sample rate, bandwidth availability, multi-hop networking, communication protocols changes);
- the tools developed and integrated should enable assessment of how the demonstrator addresses (or should be modified to better address) the traditional WSN concerns of overhead, size, and energy. The relative benefits of performing in-network information extraction should be able to be evaluated also;
- the demonstrator should provide insight into whether a fixed API would enable applications to be developed without low-level hardware access and, if low-level access is required, how can it be provided whilst still retaining the advantages of an architectural framework.

1.3 Systems Requirements

The instrumentation system presented here is designed to sit within a larger conceptual flow, represented in Figure 1 as a closed, human-in-the-loop system with several possible routes through, as described below.

Temperature is sensed at a number of circumferential and radial locations (“sense”) within the gas turbine engine. Raw sensor data is noisy and in some cases, sensors may be faulty. Modelling using a Kalman

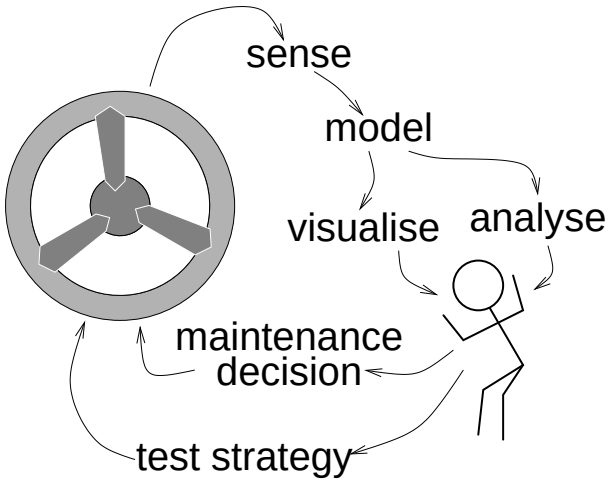


Figure 1: Conceptual flow for prototype gas turbine engine monitoring system

filter (“model”) smooths the data, making use of assumptions about the rate of change of temperature. An interpolation model is also used here to derive a continuous field function that fits the sensed data. From the model, real-time visualisation (“visualise”) is performed, allowing temperature events to be identified as they occur. Furthermore, analysis of data over a time period (“analyse”) can also be performed off-line, post event.

These two information flows allow the human expert to either derive a “maintenance decision” (referring to the instrumentation itself - component or sensor is faulty and must be replaced- or, to the engine/component being monitored), or to devise a “test strategy” (such as modifying the engine actuation during testing, in some way to try to even out the heat distribution). The set of steps and actions can be seen as part of a control loop, feeding back changes to improve or maintain the engine. Further control loops such as the above could be designed for the situation where the instrumentation is permanently fitted onto the engine and in-flight engine control and actuation decisions are generated by the integrated instrument. It is indeed here, i.e. the addition of actuation to sensing and the integration of the wireless instrumentation into closed loop systems, that the hope for future killer applications of the WSN technologies lay and also, in the authors’ opinion, how their contribution to increased safety, environmental control and active monitoring would be maximised to benefit society.

In order to enable the control loop described in Figure 1, several end-to-end system requirements have been established for the instrumentation prototype described here. The instrument should enable:

Requirement	Reason	Possible approaches
Multi-point and multi-modal sensing	Sensing at multiple points allows a detailed picture of the phenomena to be built. Multi-modal sensing (such as temperature and vibration) should allow inter-modal correlations to be better understood and thus yield more reliable high level information.	Multiple sensors (say, temperature and sound), possibly supported by multiple wireless nodes. Sound can be used as a place holder for vibration monitoring given the similar characteristics of the two signals in this context.
Communication of sensed data wirelessly to a base-station	This requirement is central to this work: communication back to a base-station should be over a wireless link, reliably over a short distance.	Most popular choices for this type of application are Bluetooth and ZigBee although WiFi might also be used.
Field mapping of sensed parameter in real-time	Without some form of field mapping, it is difficult to understand or interpret the multitude and variety of sensed data. This requirement avoids the “data overload” at user end, problem often encountered in WSN applications.	Various function approximation approaches, such as linear interpolation and radial basis functions, may be used. *Important note:* conceptually this is a worthy design requirement, however, its fulfilment from a fusion/signal processing viewpoint would need research into appropriate, specific interpolation methods (possibly model based) which can credibly represent in space and time the phenomena and the system under observation, from sparse data points.
Selection of sensors for history/time-series type display	For better interpretation of localised phenomena, individual sensor time-series display is required. Previous experience has shown that displaying a single sensor at a time is more intuitive than showing all and providing a legend, when a particular aspect of the phenomena needs to be analysed in detail.	Time series graphs could be produced in a separate window, following sensor(s) selection by the user, in real-time.

Requirement	Reason	Possible approaches
Data storage	Essential given the proposed use of the system for maintenance and engine testing, enabling post analysis upon experimentation and testing.	A possible, user friendly approach, is to use a MySQL database.
Post event data retrieval	Tools are needed to be able to extract previously logged data to allow for post analysis, system validation, calibration, etc.	A SQL query approach could be appropriate.
Cross-validation of the mapping produced at point and over specified intervals of time	It is necessary to identify the accuracy of the function approximation used to produce a field map so that it can be used with some degree of confidence.	A possible approach is to gather data using standard sensors and then use leave-one-out and leave-two-out cross-validation methods. Supporting supplementary mobile probes should also be considered in order to enable validation of interpolation accuracy between data points.
System extensibility (ease of addition of sensors and nodes)	If sensing or communication devices break, it is important to be able to replace them quickly. Also, due to the prototype nature of the system, it is important to be able to add in or remove sensors or nodes easily. The design should be scalable in essence and allow for easy user end expansion of network size.	Configuration files could be used to control which specific nodes are used in a given monitoring task. Nodes may be able to automatically detect which sensors are available for data acquisition. Upon evaluating the prototype, guidance should be offered as to the overall capacity of the network in terms of bandwidth considering the RF stack used.

Requirement	Reason	Possible approaches
Integration of calibration tools	Considering the type of sensors used (likely in the lowest MEMS price bracket), it is expected that there will be a need for frequent calibration and compensation of the thermal effects and drift in sensors, given the temperature ranges the system is aimed at.	Calibration slope and intercept for each sensor should be stored in a configuration file. An approach to calibration is to take ground truth values versus sensed values and estimate a line of best fit. The procedure should be automated and system integrated as far as possible.
Networked system debugging and testing	Networking, wireless configuration and sensors issues may cause unreliability. It is important to be able to identify such sources of unreliability and correct them on-the-flight, without system redeployment.	Several third party tools exist for displaying transmitted packets, assessing latency and packet loss, and so forth.
Fault isolation and management	Faulty or missing sensors should be clearly identified and old (stale) data should not be processed towards information extraction or displayed.	Use of NTP to support time synchronisation between nodes will allow identification of data that has been delayed in arrival. Both faulty nodes and faulty sensors can be displayed as such shortly after detection. Transient faults should be allowed for with clear indication at user end that nodes or sensors are back on line.
Easy adaptation of sensor sampling rates to fit data in the sensed phenomena	Changing sampling rates is important to allow for a balance between power consumption and responsiveness / time resolution of the data.	Manual alteration of sampling rates via start up configuration is the most basic approach but an adaptive approach that responds to data needs or battery depletion is also possible.
Power via either batteries or mains power supply	Given current power usage, long running tests will require mains supply but batteries are also important to show how the prototype might be used in real time deployments.	An approach is to use external DC transformers that provide the correct voltage to substitute for batteries.

Requirement	Reason	Possible approaches
Easy distribution of modified code into the network	During the prototype phase, it is important to be able to quickly update code.	Tools for automatically distributing code to all nodes are required to do this well. Given the deployment considerations, this procedure should not require dismantling of the prototype system from the jet pipe.

Further requirements identified during a meeting with experts from Rolls Royce are shown below:

Requirement	Reason	Possible approaches
Tolerance to EMI (Electro Magnetic Interference)	EMI is likely to be a big challenge to any wireless system responding to applications such as the one in hand.	Careful antenna placement and / or design may avoid interference due to EMI.
Support modularity in engine design.	This requirement is particularly relevant to engine test rigs where the time taken to assemble or disassemble engines, which are largely modular, is limited by wiring of sensors.	Wireless approaches will support this well if other factors can be overcome.
Fault diagnosis support and residual life estimation	Identifying when and where to perform maintenance operations will be aided by this requirement.	Development of a full blown end-to-end fault management system to include the energy related fault forecasting could be developed. This is a complex research question on its own. Presently, an extension of the basic fault management system in view for the prototype can be considered to cater for some aspects of this design requirement.

Requirement	Reason	Possible approaches
Tolerance to high temperatures	Temperatures around engine casing are not well understood (and may be quite high) and conventional electronics can not be placed in the areas where sensors would be needed.	Both research into deployment opportunities offered by the engine coupled with harsh environment electronics design and a mixture of wired and wireless transmissions are possibilities.
Minimal EMI produced	As with any in-flight system, the instrument should not produce interference that is likely to interfere with navigation or other critical systems.	This requirement was not researched so far.
Secure from malicious or accidental interference	Without some form of security mechanism, wireless transmission might be intercepted and / or fake transmissions generated.	Security protocols suitable for constrained WSN systems are currently an active area of research. An approach is to treat this issue as resolvable with a software off-the-shelf component to be integrated into the nodes support software.

2 Systems Design

The engine monitoring system developed here is not intended to be just a proof-of-concept prototype, but, more importantly, a framework for developing middleware, tools, interfaces and persistent storage components, integrated into a end-to-end system that can be used throughout the lifetime of a larger in-flight engine instrumentation development project. Moreover, from the software development perspective, solutions for a whole class of monitoring applications should be able to be retrofitted onto the designs produced here. Hence, while certain elements of the design are “placeholders” (such as the microphones in lieu of vibration sensors, or the low-cost temperature sensors in lieu of dedicated high temperature thermocouples), the software framework in which they are to function needs not be altered when they are replaced. This should remain true for higher level features of the design such as the fault management functions for example. It is hence important to note that a view of “organic growth” has been kept at the forefront of the design process and informed many of the design decisions.

Since most of the system components are intended to have a longer lifetime than the current prototype described here, many design decisions need careful thought as they will dictate structure, capability and extensibility of this system and other systems for some time. Section 2.1 summarises the key design decisions.

2.1 Design Decisions

In-network or centralised processing In their simplest and most common form to date, implemented wireless sensing systems follow a “sense and send” philosophy and hence use centralised processing entirely. Each node in a centralised system transmits sensed data back to a single point, the sink node. The sink can either be an identical sensing node that has the extra task of passing data to a point outside the network, or a “special” node, such as a desktop computer, laptop, or, in a limited number of applications, a hand-held device. The raw data is only processed once it has left the network and been successfully stored.

In-network processing, a concept well founded theoretically, allows nodes themselves to process the data in some way before transmission. This approach allows for a more timely processing of data, close to the source (filtering, for example) but is more complex to design and deploy on constrained nodes. The benefits of most forms of in-network processing are however well documented and mostly have to do with minimising the energy needed to transmit the data or information from the network to the sink[7].

For the prototype here it was decided that the nodes would need to be able to perform at least some minimal processing of data and benefit from some level of distribution of computation. By filtering the data, for example, at each node, this functional overhead is distributed. Additionally, since the interface to the network is simplified, writing new applications that use the network becomes simpler. Further, by having a framework that caters for some level of processing on the node, later developments towards more sophisticated tasks distribution schemes (for data processing, information extraction and complex querying) are easier to incorporate. An example of a more complex processing in view is aggregating data from multiple, node wired 3D vibration sensors and converting it to a unit vector - the weight of the transmitted data packages will be reduced by an order of magnitude at least. Further, should the design

evolve towards incorporating one of the many network self-organisation algorithms found in the literature, providing neighbouring nodes with access to higher-level information/meta-data at the individual node level would be very important.

From an hardware implementation viewpoint, the in-network processing feature can be easily accommodated by the powerful platform used in this work, as documented in the Implementation Section of this document.

Sensor/node ratio. The classic view of a wireless sensor network contains processing and communication nodes with sensors attached directly to them. The sensors usually sample the observed phenomena at the position of the node. The hardware platform used here, through an in-house sensing expansion board provide the opportunity to add many sensors and sensor types to each node, up to a bus limit of 128. Whilst in deployments over large areas it is not feasible to reduce node numbers and run wires to multiple sensors, in a system such as this, however, deployed in a small area, it is possible and advantageous to do so: Firstly, the communication load is reduced through the use of a mixed wired-wireless system, in terms of actual data sent over the wireless link. Packets become more efficient as the data/overhead ratio increases, and also, in terms of network saturation, fewer nodes have fewer collisions in a CSMA scheme or less wait between slices in a TDMA scheme. Optimising the instrumentation system from energy consumption viewpoint is essential in further stages of the project as the nodes are ultimately aimed to become self-powered (few powering options are viable at the high temperatures exhibited in a jet engine). Hence, since the biggest drain on the battery life of a node is communication, the argument for fewer nodes for a given number of sensors is strong. Secondly, in this particular system, there is an additional user pressure to minimise weight and take advantage of the deployment conditions (relatively small, confined space on the jet pipe) and the availability of appropriate cabling for this deployment environment.

As an outcome, the final decision is to use four processing and communication nodes, each with four temperature sensors and one microphone. The sensors are arranged in a pattern dictated by engine design and the experience of engine designers, while the node placement is based on simplicity, with each node mounted outside of the pipe, at equal distances.

Active/passive data collection. Although there are a number of ways to structure the sending and receiving of data within a network, they are almost always sufficiently described in the abstract as either a polling, publish/subscribe or passive collection method.

In a polling system, the receiver is responsible for requesting data. In a WSN this would be the base-station or sink node. The base station would poll, possibly indirectly, the sensing node. The sensing node would then respond with the required data.

A publish/subscribe model works by each node publishing the availability of a “stream” of data. Other nodes, typically the base-station, then “subscribe” to the stream and from then on will be sent data without having to make further requests. This reduces overheads somewhat, since the requests are sent just once, and works well for continuous monitoring systems where there is a requirement for the base station to receive all data. Rule-based modifications to this system are possible and allow customisation to selective data flows.

For a system in which all data from all nodes is always required, the overheads of polling puts that model in it’s worst-case for efficient communication. In the publish/subscribe model, the act of subscription

becomes an unnecessary formality, since all streams will always be subscribed. In this situation, a passive collection of data by the base station might be recommended. Each node sends data to the base-station without polling or subscription. In the application presented in this report, this is the most suitable method and is the one that has been implemented.

The other methods have not been ruled out for future work and every effort has been made to ensure that decisions made now will not preclude their inclusion at a later date.

Communications The hardware platform available (a combination of the Gumstix platform and in-house expansion board) offers a number of possible modes for communication: Bluetooth, Zigbee, WiFi and wired Ethernet.

The simplest of these to use in developmental work are WiFi and wired Ethernet as they are the most common methods of networking and, as such, are well understood. Neither is a good fit in a system such as this one - while WiFi meets the requirement of wirelessness, it does not perform well in terms of power consumption.

ZigBee, a standard developed for wireless personal area networks (WPANs), has been put forward as a strong communications technology candidate for wireless sensor networks. It is low-power, cheap and capable of automatically creating multi-hop networks of many nodes. However, the prescriptive nature of the standard makes it difficult to implement many of the leading edge efficient routing algorithms, develop and integrated middleware and so on, while the automatic network discovery and prescribed routing are not useful features for the system under design here.

Bluetooth is less flexible than Zigbee in terms of numbers of devices able to be accommodated in a network (limited to just seven connections to base). It also has a shorter range (approx. 10m indoors) and, according to the literature, higher power consumption. And yet, its point-to-point nature makes it fairly attractive for small networks. Also, authors' previous comparative experiments have shown very little difference between Bluetooth and ZigBee in terms of energy consumption and range.

Considering the above, Bluetooth is, presently, the most appropriate mode of communication for this application. However, there is also an identified need for ability to perform prototype monitoring and logging into the running instrumentation system during testing and prototype development without affecting the normal functionality and performance of the wireless network. Hence, a secondary mode of communication has also been deployed, to avoid adding demand to the network that is not related to the application or final system, or causing network performance degradation by interference. The wired Ethernet connection will allow such tasks to be performed in a strictly out-of-band fashion.

Since the system itself is expected to outlive the application, however, the other communication modes should not be ignored. In environments with access to a power source, WiFi supports faster, larger, networks, while, if large networks are required with low power consumption and no specific network topology requirement, ZigBee would be suitable.

Consequently, a combination of Bluetooth and Ethernet has been implemented whilst work advanced towards developing implementations of the other communications modes using the same (API). To date, in the current system, it is possible to swap between WiFi and Bluetooth with nothing more than a configuration change.

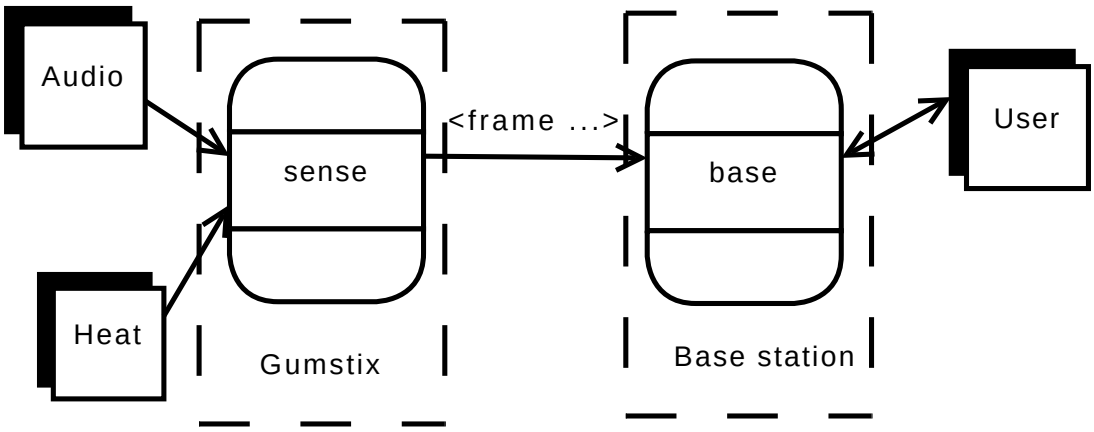


Figure 2: System overview

Time Synchronisation Time synchronisation is a important service in this prototype, The maintenance of a network time where the clock on the basestation and nodes are synchronised allows comparing the timestamps on the data samples to the system time at the base station, to provide an effective (if somewhat simple) error and fault handling mechanism. Whilst having a global network time is essential, synchronising the clock of the base station and to that of the real world is also important, allowing the sampled data to be stored in a logical way within the persistent storage database, and facilitating the retrieval of historical experiment data according to logged experiment date and time rather than some arbitrary value. It follows that the time synchronisation problem needs to be addressed in two stages: first, the basestation is synchronised with the real world, then the nodes within the network are synchronised with the basestation.

In the work here, the use of Ethernet emulation over Bluetooth has allowed Network Time Protocol (NTP) to be used for clock synchronisation. NTP is the most widely used protocol for synchronising computer clocks to Coordinated Universal Time (UTC) and generally provides an accuracy within 1ms within LAN conditions. This level of clock accuracy within the prototype system is greater than that achieved in current WSN systems and more than adequate for the error handling and data logging purpose. The accuracy of the node clocks as compared to the base station is also important, particularly for evaluating and specifying the network performance in terms of the end-to-end system latency. Whilst NTP can provide the individual nodes synchronisation as well as the base synchronisation, other time stamping methods should be considered for the nodes data, such as for example stamping on arrival at base procedures.

Data Packet Design Data is formatted for transmission using XML - whilst not ideal for use within a WSN deployment due to the large packet size overhead associated with the markup language, within the prototype system it provides a simple, easy to understand and standardised syntax for formatting data.

Whilst the prototype system gathers two types of sensed data (temperature and sound), the similarities in data format allows one general encoding schema to be used, with different attributes for temperature and sound modes. This flexibility inherent in XML will allow the system to be extended without requiring a large change in encoding for different sensing modes. Figure 3 shows the XML schema for the prototype

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
<attribute name="version" type="float"></attribute>
<complexType name="frameType">
  <sequence>
    <element name="tstamp" type="float" maxOccurs="1" minOccurs="1">
    </element>
    <element name="sample" type="tns:sampleType" maxOccurs="unbounded" minOccurs="1">
    </element>
  </sequence>
  </complexType>

  <complexType name="sampleType">
    <attribute name="id" type="int"></attribute>
<attribute name="cov" type="float"></attribute>
</complexType>
      <attribute name="mode" type="string">
</attribute>
</schema>
```

Figure 3: XML Schema

application.

Debugging and evaluation tools The immaturity of WSN hardware and general lab-limited state-of-the-art in the field of deployable WSNs means that standardisation is lacking and so is detailed design documentation for existing working systems. Most research efforts in the area of real-life WSNs have involved bottom-up developments towards working systems and little of the work is documented in such a way that it is transferable to the design and implementation of few applications. With particular reference to the hardware support used in the WSN application here, given the combination of sensing, processing and communication functions mapped onto the platform, there can be many causes to hardware failures which in turn affect software development strategies. To this end, a suite of hardware and software debugging and evaluation tools are necessary, to allow hardware failures to be diagnosed, and the developed software to be tested and functionally assessed.

1. Hardware debugging tools The hardware debugging tools are required to provide information about the sensors and communication modules. In the prototype here, the choice of physical and software connectivity between the individual sensors and the nodes provides the first level of hardware debugging at no extra cost: the I2C interface software used to gather temperature data allows the nodes to determine sensor connectivity and recognise invalid data values, hence allowing the user to understand which sensors are functioning correctly.

Debugging the communication stack can generally be performed using stack-relevant debugging tools. Given that the design choice here is the use of Bluetooth stack, its *Bluez* utilities, *hcidump* and *pand* can provide a high amount of low level information on the current state of the communication module. WiFi communication stacks also provide the same information via kernel level tools. These are the tools drawn upon in the prototype.

2. System performance evaluation tools In contrast to the "real-time" nature of the hardware debugging tools. Many of the software evaluation tools will make use of the data gathered during experimental runs, using the data storage facility to retrieve and post process data and information. Several dedicated tools need to be considered:

a) To assess the *accuracy of the interpolation* algorithms a "leave one out" cross validation approach has been decided on.

b) Given that the aim of the end-to-end system is to provide real-time spatio-temporal representations of the phenomena, *latency* (time elapsed between a data sample is acquired and the time it is delivered to the visualisation component) is an important issue. Latency information is also a valuable tool in diagnosing the network as a whole and tuning a variety of parameters, such as sampling rate for example for best performance. A way of measuring latency is through the use of timestamps, both when the data sample is gathered from the hardware sensor, and when the processed information is passed to the visualisation system by the basestation. This is the approach implemented here.

Other *network performance measurements* can be made using a *"packet sniffing"* tool such as *hcidump* or *wireshark*, which allow the user to see details of all packets transmitted within the network. The use of such tools is common practise when studying the effectiveness of wired networks, and have reached a high level of maturity providing a wealth of information on network statistics. These tools have been integrated in the prototype here, with a view towards ease-of-use.

Database design decisions Data storage is an essential requirement of the end-to-end system to be developed. A database approach has been chosen. The database is expected to hold a large amount of data, given that engine monitoring experiment runs involve gathering approximately 64 temperature samples/second and 4 sound samples/second for prolonged periods of time (days/weeks). This will hence require an efficient database design. After normalisation, the database consists of four tables as shown in figure 4.

event Holds details of interesting events within experiments. Each experimental run begins with a "start" event, which logs the time stamp and details of the particular experiment. The design of the table is such that a user can "tag" events (such as the start of a heating phase, for example) within experiments, with entries within this table.

sensorConf Holds details of nodes' configuration. For each sensor involved in an experiment a *sensorConf* entry is generated, allowing the user to understand the exact setting and configuration of the sensors during the particular experiment.

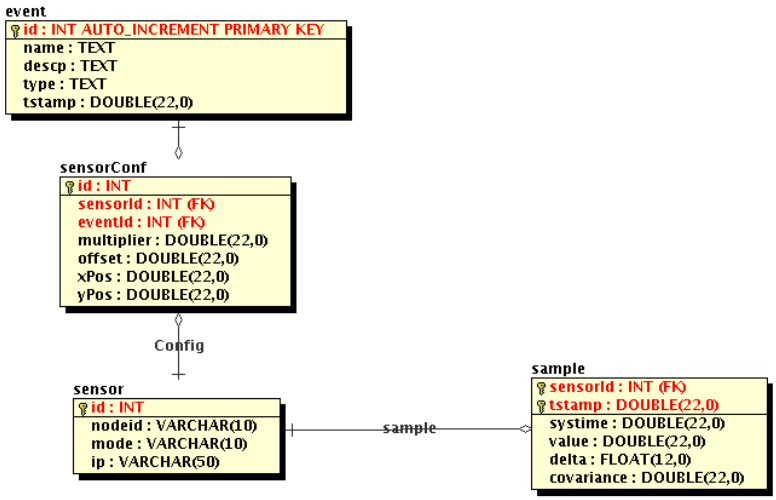


Figure 4: Database Entity-Relationship diagram

Sensor Holds details of individual sensors; each sensor is given an id based on its individual id and Gumstix id. This will allow the exact pairing of sensor-Gumstix to be traced for each experiment run (this is also helpful debugging information).

Sample Contains details of sound and temperature samples received by the system.

2.2 Abstract design

Figure 2 shows a high-level abstract view of the system. External phenomena are sensed in two modes: Audio and Heat. (As mentioned before, the Audio component is presently a place-holder for the investigation of vibration monitoring. The decision to use microphones at this stage comes from the similarity of vibration and audio data in terms of format, frequency profiles and data rate.) The sensors are attached to wireless processing nodes that communicate with the base-station. Within the *sense* component, mode dependent sub-components deal with temperature and sound, while a generic filtering system applies filtering as required. (Details of the filtering options accommodated as plug-ins in the architecture are given separately).

The base-station, which is running on an ordinary desktop/laptop computer, is responsible for storing data, displaying the visualisation and interacting with the user. The "*sense*" and "*base*" components are shown in Figures 5 and 6 and detailed below.

2.3 The sense component

The breakdown of the "Sense" component are given in Figure 5

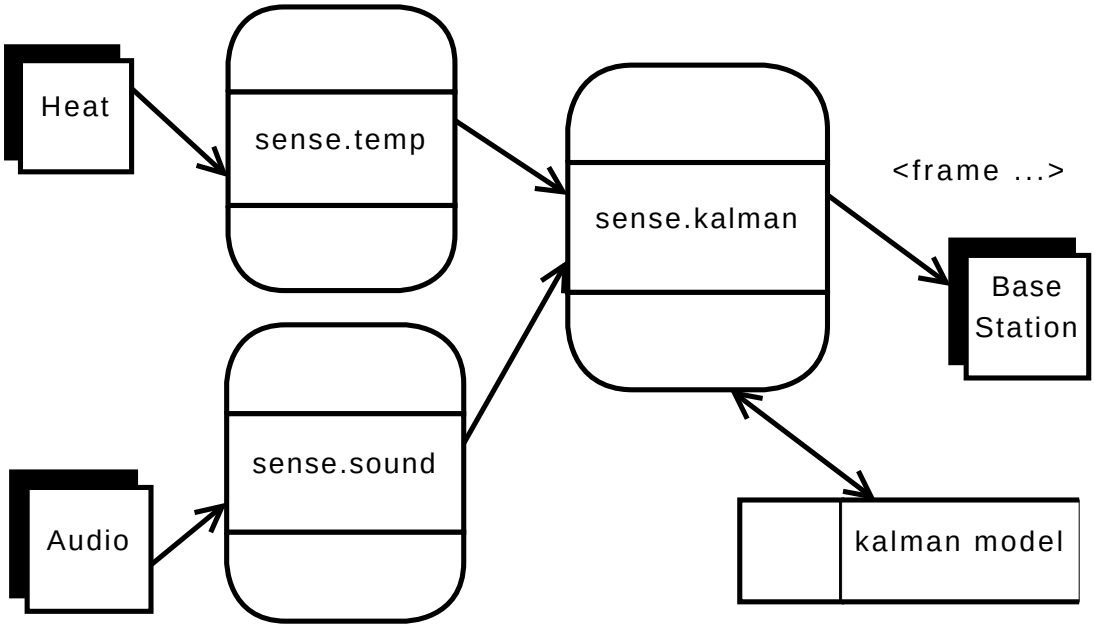


Figure 5: Breakdown of the “Sense” component

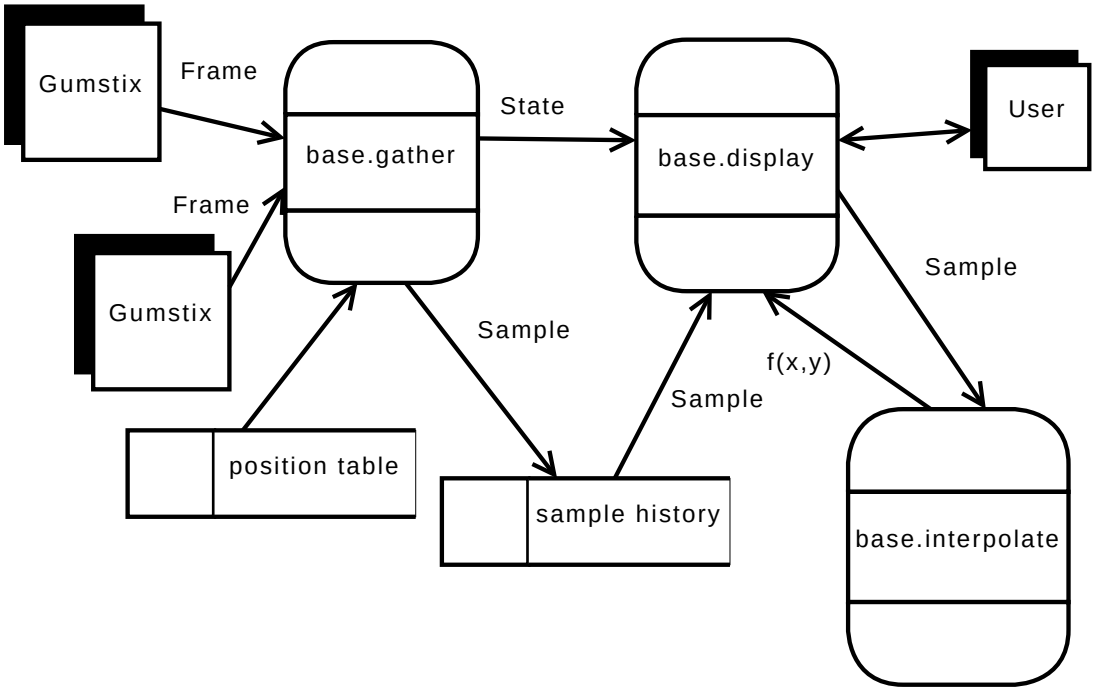


Figure 6: Breakdown of the “Base” component

The *sense.temp* module makes use of an in-house written I^2C library to query each temperature sensor in turn.

Timestamps are also gathered at this point. The use of NTP is important here in ensuring that the timestamps correspond to the actual time. Through trial and error, it was found that a transmission cycle time of about 0.25 seconds maximised the data rate without causing congestion or loading the processor too heavily.

The *sense.sound* module uses the platform's on-board AC'97 processor and periodically extracts signal peak levels over short periods of time, currently the sampling period for sound data is 1 second. .

The *sense.kalman* module performs filtering on the data. The initial approach for filtering was to use a simple Kalman filter that took no account of the proximity of nearby sensors or their current reading but rather assumed that the ground truth sensor value was constant. The Kalman "model" is on a per sensor basis and consists of the current estimate and the covariance matrix. It was found that the assumption of constant value was not valid for sound, which tends to vary rapidly and a pass-through filter was used instead for this sensing modality.

A later requirement added to the system specification was to allow the user to visualise the rate of change of temperature. To support this, a new Kalman filter was developed based on the assumption of a constant rate of change of temperatures. The advantage of this approach is that both the absolute value and the rate of change are smoothed.

Note that the component based design and the care exercised at integration stage allows for the three filtering options above to be easily swapped.

2.4 The base component

The breakdown of the "Base" component is shown in Figure 6.

The *base.gather* module is responsible for receiving frames and breaking them up into individual samples. Each frame is a recording of the environment at a given instant and can contain multiple samples – two volume measurements, for example, for the audio sensing mode. The *base.gather* module tracks the current state of the sensor value and reports this, on request, to *base.display*. The timestamps of original transmission are used to determine if the value is "*stale*" (hence identifying a faulty sensor or group of sensors/node). The *base.gather* module also logs all received sensor samples to a remote MySQL database. This module deals with both *calibration* of sensor values (by applying a linear transform) and identifying the position of the nodes physically and on-screen, within the visualiser. Again, the *calibration* component is in the form of a plug-in and can be readily changed to apply suitable transforms to the data, drawn from sensor calibration and drift correction procedures.

In *base.interpolate*, three interpolation methods have been developed and fully integrated into the end-to-end system:

- cinterpol - based on a radial basis function (RBF) applied to provided sensor points
- QSHEP2D - based on a series of quadratic curve fits

- Nearest neighbour – estimating the value for any point as being the same as its nearest neighbour.

The user-interface allows the user to select which interpolation algorithm is to be used. Furthermore, a number of false-colour schemes have been developed: red-scale (values between black and red); jet-map (provided by TRW); thermograph. The thermograph false-colour scheme has been selected for implementation in the current prototype but, could easily be replaced by a more appropriate scale, according to the user-requirements without interference with the rest of the system.

The maximum and minimum values of the interpolated result at every given instant in time are used to scale values before being mapped through the false-colour look-up table, allowing for an adaptive, easy-to-interpret scale to be obtained.

The interpolated region shape is a ring (doughnut), corresponding directly to the deployment arrangements for the sensor network as described in the Introduction section. Specifically, it is a large circle with a smaller circle cut out from the centre, representing the hollow region of the engine exhaust (lumen). A number of options for region topology exist. The method used here is to consider the distance between two points to be the Euclidean distance.

The *base.display* module displays interpolation results and time series history of a selected sensor. The *base.display* module performs the following main functions:

- periodically (default of 0.25 seconds, but user adjustable) requests gathering of new samples;
- initiating the logging of data to the database;
- interpolation of current data;
- displays interpolation results and history of selected node;
- allows configuration, including: whether to display sound or temperature; selection of which sensor to provide history for; block size (to improve or reduce resolution) setting; desired frame rate (adjusting block size automatically to meet set frame rate) setting.

In summary, the software support for the system here performs, at node, sensed data acquisition and time synchronisation, Kalman filtering of values and gradients and data forwarding via Bluetooth to the base station. Base station software receives and stored temperature, temperature gradients and sound samples and interpolates one of these, as per user request, in real time to obtain an estimated map of the desired parameter. Stale sensor values (possibly due to sensor faults or power loss) are also identified and excluded from the real-time map. Historical data is maintained and can be queried by selecting any sensor from the display. Whole test runs can be post analysed using the integrated database storage coupled with a multifunctional user interface.

2.5 Communication subsystem

Both the base and sense component make use of a communication subsystem, developed as part of a middleware framework for WSN (under development by one of the authors here as part of their PhD). The

following section describes this work, and discusses it in context of the prototype system.

2.5.1 Background

As the availability of low cost, low power computing hardware increases, one of the common goals of WSN research is to achieve mainstream adoption of WSN technologies. However, the inherent complexity of WSN systems, combined with the wide range of existing hardware platforms, sensor technologies, communication technologies and design methodologies means that specialist knowledge is required to develop, test and deploy WSN applications. This provides a significant roadblock to the wider adoption of WSN technology, as it is difficult for non specialist users to develop, deploy and maintain WSN applications.

The majority of research effort in the WSN community is “protocol centric”. That is, the focus of the research is in developing protocols for storage, transmission or processing with as little resource use as possible, but without the context of a real application. Whilst this work has been of great benefit in terms of theory, protocols and techniques developed, the diversity of hardware and software solutions mean there is little standardisation of design and algorithmic approaches, with most applications implemented with a tight coupling between the application and hardware stack. Such tightly coupled approaches have created two impediments to development, which in turn increase the difficulties associated with developing applications:

- Lack of interoperability between individual components on different systems
- Lack of a common framework for new developers to build applications

The communications modules used in the prototype have been designed as part of a middleware API (Application Programming Interface) for supporting WSN development and deployment, addressing the issues above by abstracting away the difficulty associated with programming common WSN tasks. As communication tasks are common to all WSN applications, providing a cross platform API for performing these tasks is a logical starting point for middleware development. Presently, the communications modules provide a transparent, cross platform, service for multiple transmission media such as TCP/IP (WiFi / Ethernet) or Bluetooth Networks. (Support for Zigbee networks is currently under development.)

2.5.2 Design Principles

Layered Model Abstraction based middleware frameworks typically use layers of abstraction to hide implementation details of a common task, providing a cleaner work flow, allowing functionality to be compartmentalised, and enabling a component based code design. The communications modules developed as part of our framework use a three layer stack as shown in Figure 7.

- The Abstraction Layer provides the user with a functional API to access the common functionality. Within the communications system this API is common to all hardware platforms and communica-

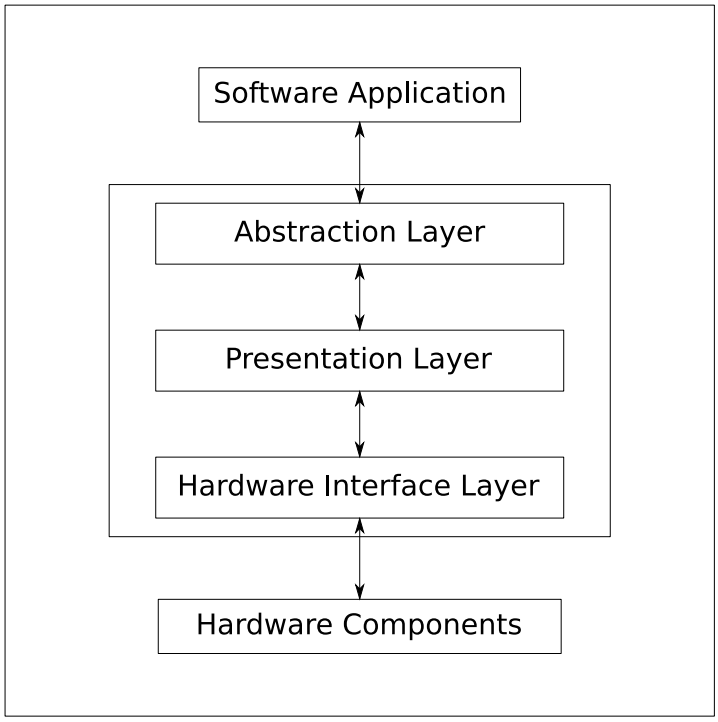


Figure 7: Networking modules abstraction stack

tion media, calling upon functionality contained within the presentation layer to facilitate communications tasks.

- The Presentation Layer is responsible for formatting data for passing between the Abstraction and Hardware interface layers. This ensures that each layer receives the data it is expecting, regardless of the Hardware Interface layer in use.
- The Hardware Interface Layer is responsible for interacting with the hardware devices. This performs hardware specific, low level communication tasks, such as sending and receiving data via the radio chip.

Cross media design The design of the communications modules facilitates cross media communications. Differences in communication media such as Bluetooth and Zigbee are abstracted away by the Presentation and Hardware Interface layers. With the Application layer providing one API for all communication types, it allows applications to migrate between communications methods without requiring a large amount of code re-factoring.

This allows the developed code to be executed on any hardware platform with communications modules developed (currently Linux x86/arm, Windows, Symbian) without modification. Different communications protocols such as UDP, TCP and Bluetooth RFCOM and L2CAP are also interchangeable without major modification. However, due to differences in the node addressing scheme used such as TCP's IPv4 and

Bluetooth hardware addressing, some modification of the nodes addressing scheme may be required if switching between communications media.

An illustration of cross transmission media design is given in 3.3.3.

Cross platform functionality. Much like the cross media functionality, another key objective of the middleware framework is cross platform functionality, allowing code developed on one platform to be executed on other hardware without extensive modification. This offers a significant advantage, as sensing applications can be developed and tested on “resource rich” desktop computers, utilising the wide range of development tools available for these platforms, and giving the opportunity to address many common implementation errors. Once the underlying code is shown to function correctly in this environment it can then be deployed on the intended platform, where testing can concentrate on issues raised by using constrained resources.

The choice of an interpreted language such as Python for the framework implementation reduces some of the difficulties associated with cross platform design. Python was intended from conception to be cross-platform and implementations for platforms such as Windows, Mac and Linux are available. More importantly recent increases in hardware capability has meant that a new generation of Linux based embedded devices are becoming common in WSN development. Devices such as Gumstix, iPAQ, and Stargate nodes are capable of running a full Linux kernel and compilation tool chain, and have the Python interpreter available.

As with the frameworks cross media capabilities design choices such as layers of abstraction and object orientation make cross platform design a relatively trivial task for the user, as applications developed using the communications framework can be ported to these devices without source code modifications. However, the cross platform capability requires hardware specific versions of the code base used in the Presentation and Hardware interface layers to be developed for each platform, taking account of the specifications of the intended hardware. Currently implementations of the communications framework are available for Windows and Linux devices, and also for the ARM Linux used on the Gumstix, with a implementation for Symbian based series 60 Nokia mobile phones also under development.

Object oriented design. The communications modules take a object orientated design, with super classes defining general functionality subclassed into hardware specific implementations. Keeping to the pre and post conditions in the superclass function definitions ensures that hardware specific implementations perform in the same way. Hence, applications can be written with code reuse in mind, as any applications developed using the communications subsystem can expect tasks to function in the same way. The development of hardware specific modules is also simplified, as a template for class functionality is provided via the superclass.

Figure 8 shows a class diagram for the Link classes, these classes function at the Presentation and Hardware Interface layers, allowing the communications modules to interact with different networking hardware. The links.socketLink superclass contains the generic functionality required by the communications subsystems, subclasses such as btlinks.l2capLink provide hardware and protocol specific implementation of communication methods (in this case allowing communication via the Bluetooth native L2CAP protocols).

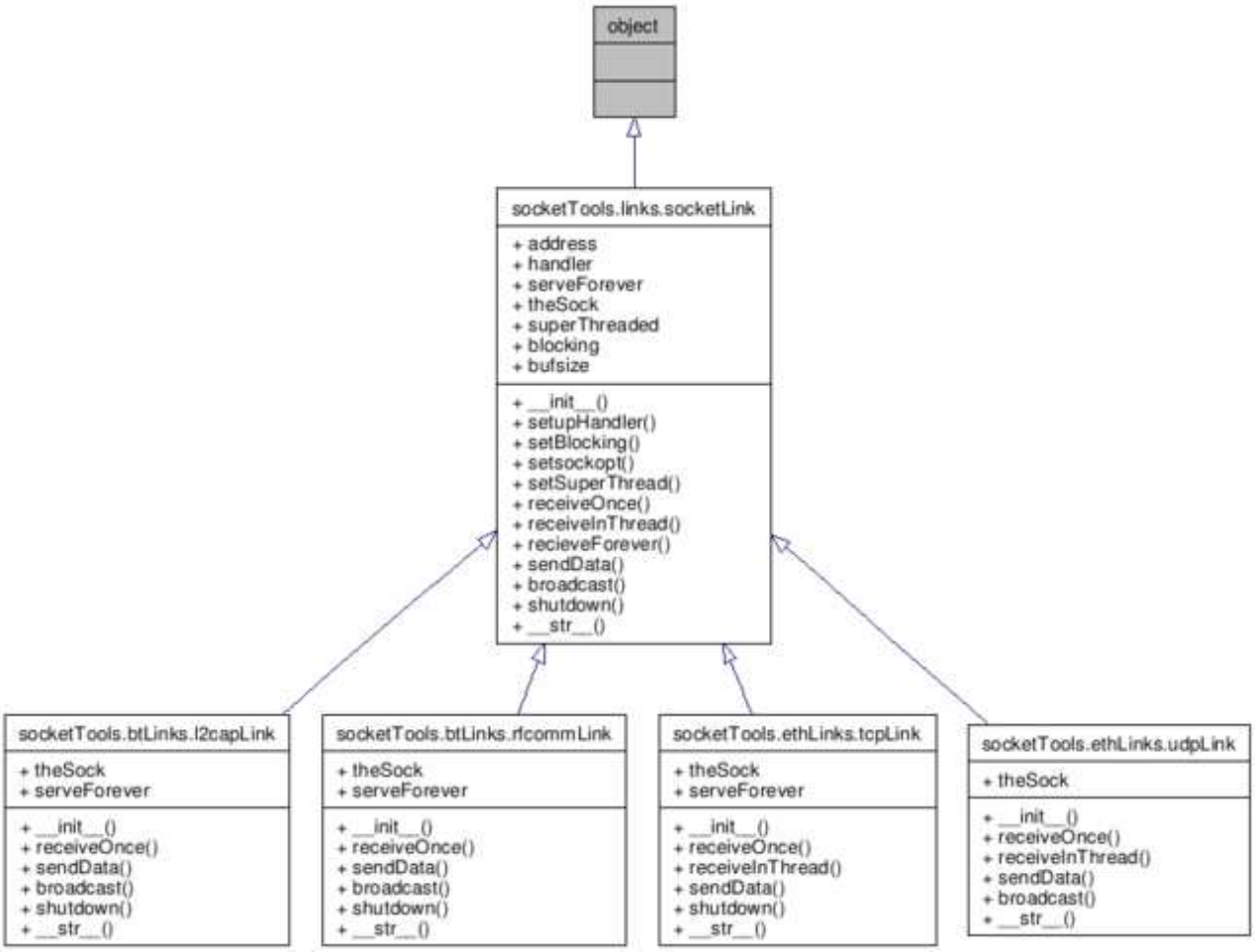


Figure 8: Communication modules class structure

Plug-and-play code reuse Following the object oriented paradigm presents a greater opportunity for code reuse, as methods using the communication modules are able to be transferred between applications, communications media and hardware platforms without requiring refactoring. For example, if a routing algorithm is developed, the framework should allow the same code base to be used in any other application using the framework without the need for extensive modification. Allowing plug-and-play code reuse is a major consideration of the middleware framework, aimed at reducing the complexity of developing code by offering a library of pre-developed modules.

3 Implementation

3.1 System level data flow and hardware support

3.1.1 Hardware architecture

The hardware architecture of the implemented system is given in Figure 9.

The prototype system is comprised of 4 data gathering nodes. (Note that a fifth node is added to the system for evaluation purposes as described earlier in the report). These collect sound and temperature data from the attached sensors, filter and calculate “delta” values then forward the processed data to the basestation via a Bluetooth connection.

The basestation processes the incoming data, attaching to each data packet information such as node position from the configuration file. Following this the calibration coefficients are also applied resulting in calibrated data being used further in the chain. At this stage the calibrated data is checked for errors (stale data is excluded), before being passed to the interpolation engine. The interpolated field is then displayed on the user interface. Concurrently the calibrated data is forwarded to the external database for storage and post processing.

3.1.2 Node hardware

The Gumstix platform was chosen as the base for the sensing nodes. The Gumstix Verdex board includes an Intel XScale PXA270 400MHz processor, 16MB of flash memory, 64MB of RAM, a Bluetooth controller and antenna, and 60-pin Hirose, 120-pin MOLEX and 24 pin flex ribbon connectors for expansion boards. The in-house custom expansion board provides Zigbee communications via a MaxBee chip, I^2C bus support, and audio processing, and is discussed fully in Section 3.1.3. Temperature sensing is achieved via the Analog Digital ADT75A chip, which performs sampling and conversion internally, before delivering the sensed temperature values via an I^2C bus. The ADT75A provides temperature resolution of 0.0625°C via a 12 bit ADC, and is rated for operation between -55°C and +125°C.

During operation, the software running on the node consumes approximately 6MB of RAM. The operating system consumes a further 15MB, leaving around 43MB free for further application software that might be developed in the future. Secondary storage requirements are very low, and typically each node has 7MB free out of a total 16MB available. When in full operation the application uses between 11-14% of the available processing capability.

Networking using Bluetooth, Ethernet or WiFi is offered here as a system service, and can be used by multiple applications at any one time, just as with a desktop operating system. (Zigbee is currently limited to a single user. That is, more than one application can communicate using Zigbee but they must all use the same UID. Only a Python API has been developed for Zigbee at the moment. When it has been fully evaluated, the single user limitations will be overcome if Zigbee is considered a suitable choice.)

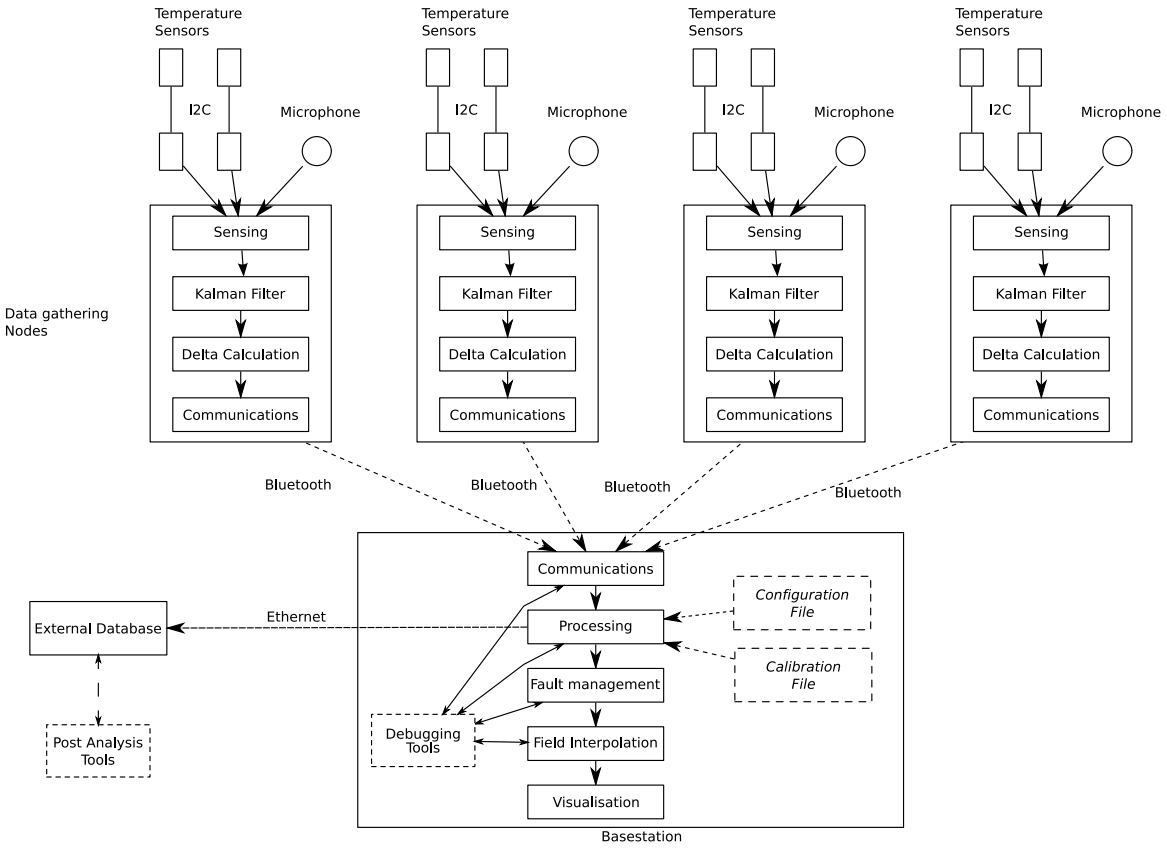


Figure 9: Prototype hardware architecture

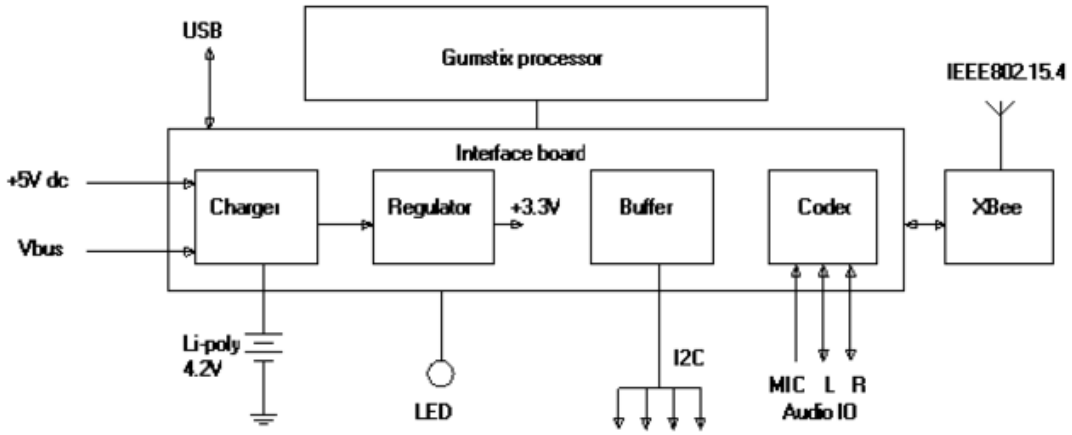


Figure 10: Cogent interface board

3.1.3 Interface board design

To deploy the Gumstix [1] processor module in this application with a compact footprint and reasonable power budget, a special purpose interface board was designed as shown in Figure 10. Connecting to the Gumstix via its 60-pin Hirose connector, the main functions of the 100x30mm, double-sided surface mount board are to provide essential interconnection features and a flexible power scheme.

External power is sourced from either a 5V DC jack or a 5V USB bus, where a 500mA capable device is required to provide adequate power. The external power supplies a Maxim 1551[8] battery charger which will select between the DC and USB source to either charge a connected battery or provide direct power to the system if no battery is present. The battery is a single cell Lithium-polymer 1000mAh device. Power to the interface electronics and processor board is supplied at 3.3 V through a Maxim 8887 [9] low drop out regulator incorporating overload protection and thermal limiting. The power budget depends on precise operating conditions but an unloaded interface typically consumes between 10 - 20mA, rising to between 200 - 300mA with a processor and typical interface configuration. The power consumption of the Gumstix processor is particularly peaky during its boot cycle which can see consumption rise to over 400mA for short periods.

Interface capabilities provided by the design include: USB master/slave port over a mini-B connector; 4-way I2C buffered bus for connecting a variety of sensors to suit the application; Line-level stereo audio input/output and microphone input provided by the Phillips UCB1400[6] codec connecting to the Gumstix XScale processor via AC97; Asynchronous serial connection to an on-board Maxtream XBee [2] module for IEEE802.15.4 radio; Programmable LED indicator for status monitoring.

The most complicated aspect of the board design was determining the correct GPIO signals to use for the serial connection and monitor LED across a few different versions of the Gumstix module given the multi-functional nature of the XScale processor IO. In future it is planned to increase the battery charger power capability to allow faster battery charge while also powering a processor.

3.2 Signal processing: Calibration and filtering options

3.2.1 Sensor Calibration

To calibrate the temperature sensors, a process similar to that used in industry was followed. Placing the sensors in a water bath and recording the ground truth temperature of the bath with a reference thermometer. Recordings of the temperature values reported by the sensors were gathered and compared to this reference temperature value to discover the sensor offset.

The offset is calculated using the following process:

It is assumed that the water bath cools according to Newtons law of cooling.

$$T(t) = T_{\text{env}} + (T(0) - T_{\text{env}})e^{-rt}$$

where $T(t)$ is the temperature at time t, and $T(0)$ is the initial temperature at time 0.

Therefore, treating $T(0)$ and r as unknown

$$\ln(T(t) - T_{\text{env}}) = \ln(T(0) - T_{\text{env}}) - rt$$

Assuming T_{env}= 24(°C) this is an equation of the form

$$y = mx + c$$

where

$$\begin{aligned} x &= t \\ y &= \ln(T(t) - T_{\text{env}}) \\ m &= -r \\ c &= \ln(T(0) - T_{\text{env}}) \end{aligned}$$

So a line of best fit for t versus $\ln(T(t) - T_{\text{env}})$ gives an estimate of the slope and offset, mx and c respectively.

This line allows us to estimate temperature for any time t based on a small set of readings. We estimate a line of best fit between $T_{\text{ref}}(e)$and the sensor temperature $T_{\text{S}}(t)$ to identify sensor response. It is assumed that actual response is linear and without hysteresis. The slope was formed to be close to 1 and so was assumed to be so.

The calculated calibration values are recorded in a configuration file on the basestation. This is read upon systems start up, and the reported temperature values from the sensing nodes, adjusted using the calibration values calculated using the above process.

3.2.2 Kalman filter and "delta" calculation

The generic framework defines a node level filter f. In the gas turbine engine demonstrator, the filter was comprised of several processing steps: First, outliers were removed; second, sensor values were calibrated according to sensor-specific calibration coefficients; third a Kalman filter was used to reduce sensor noise.

Outlier removal is necessary because the sensors occasionally produce extreme values, possibly due to I2C bus communication errors. This is not dealt with by the Kalman filter, which assumes that noise is distributed normally with a zero mean. Although the digital thermal sensors provide degrees Celsius as output and are factory calibrated, sensor readings differed slightly from the actual temperature. Sensor response tends to be quite linear with a roughly unitary slope. Therefore, calibration was restricted to adding a sensor-specific offset. This calibration coefficient was obtained by placing sensors in a stirred water bath and comparing with a calibrated mercury thermometer.

The third stage of filter processing is to use a Kalman filter. There are several reasons for incorporating Kalman filtering [5, 15]: 1) to recover some resolution in the temperature measurement that is lost through A/D conversion; 2) to reduce sensor and measurement noise; and 3) to fuse multiple redundant sensor readings into a single estimate. Temperature in the jet pipe over time is clearly a non-linear function. However, since there are so many factors (both measurable and unmeasurable) affecting it, and since it tends to change relatively slowly, the linear assumption implicit in a Kalman filter is a good compromise. Two possible state models for the filter were considered for this work: one that assumes that the temperature does not change (and thus any change is noise), and one that assumes that the rate of change of temperature is constant (and thus any change in the rate of change is noise). In the present system, the latter was used, as it provides smoothed estimates of both temperature and rate of change of temperature.

Given a single location temperature τ_ℓ, a constant temperature rate model is comprised of the state space for the location $\mathbf{x}_\ell = (\tau_\ell, \dot{\tau}_\ell)^T$, a transition model $\mathbf{F} = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix}$, and some model noise $\mathbf{w}$, such that at time k,

$$\mathbf{x}_k = \mathbf{F}\mathbf{x}_{k-1} + \mathbf{w}_k$$

The vector of sensor measurements for a probe $\mathbf{z}_k$ is given by

$$\mathbf{z}_k = \mathbf{H}\mathbf{x}_k + \mathbf{v}_k$$

where $\mathbf{H} = (1,0)$ is the sensor model (corresponding to a single, calibrated temperature sensor) and $\mathbf{v}$ is noise affecting the sensor. Sensor measurement noise is distributed normally with a zero mean and covariance $\mathbf{R}$. It is assumed that sensor noise is uniform across sensors and uncorrelated between sensors and thus $\mathbf{R} = \sigma_z^2$, where σ_z^2 is the variance due to measurement noise. This variance was estimated for the sensors used by measuring a series of values for two sensors at room temperature and taking the individual variance as half the variance of the difference. This estimate is valid if the sensor noise for the two sensors can be considered to be independent. The model noise $\mathbf{w}$ is also distributed normally with a zero mean and covariance $\mathbf{Q}$. In terms of the model, the noise $\mathbf{w}_k$ corresponds to the change in temperature and temperature rate due to the temperature "acceleration" a_k, which is

$$\mathbf{w}_k = \mathbf{G}a_k$$

where $\mathbf{G} = (\Delta t/2, \Delta t)^T$. It is thus possible to derive an expression for the covariance $\mathbf{Q}$ in terms of Δt and the variance in the acceleration rate σ_a^2. The acceleration rate variance provides a convenient tuning parameter to allow for more or less rapid variations in temperature.

A key challenge is to support the Kalman filter with minimal computational cost. In the prototype system, Python was used with matrix manipulation done via Numeric, and although this is a relatively inefficient approach, it reduced coding time dramatically and was still able to run in a real-time mode on the platform of choice for this prototype.

The result of filtering is to produce a state / management vector of the form $\mathbf{x}_{\ell,t} = (\tau, \dot{\tau}, \nu, p_{1,1}, p_{1,2}, p_{2,1}, p_{2,2})^T$ where τ is the temperature, $\dot{\tau}$ is the temperature rate, ν is the sound level, and p is the 2×2 estimate covariance matrix. Note that the sound level ν is not processed by the Kalman filter. Also, in the implemented system, sensor locations for sound and temperature were always distinct and thus the state vector contained one or the other but not both.

No event triggers were used in the gas turbine engine demonstrator, nor any priority ordering. Event detection could be used to substantially reduce traffic when the system is relatively stable. For example, an event might be "temperature has increased by 2°C." Note that event detection can make use of knowledge of the state evolution model m, and only transmit when the model error would be too large. Under this view, an event might be "based on the last transmitted state, the base station estimate error will exceed 0.5°C." In a sense, the event predicate must make a judgement about the information value of the state vector to the user.

3.3 Communications

Much of the networking and message passing in the prototype is controlled by the communication modules discussed in Section 2.5, however it was still necessary to implement communications functionality specific to the jet pipe application on top of this framework.

3.3.1 TCP emulation over Bluetooth

Whilst the communications subsystem offers native Bluetooth communications, Ethernet emulation over Bluetooth via PAND was used to facilitate networking. Whilst this may not be optimal from a communications viewpoint as the Ethernet emulation does inflate packet size, experimentation has shown this overhead to be minimal. Emulation offered many benefits in the implementation, allowing TCP based networking tools NTP for time synchronisation and packet sniffing to be used. If future work requires native Bluetooth communications, a time synchronisation module would need to be implemented.

The network topology used in the prototype is based on Bluetooth Piconets, with each sensing device connected as a slave to the basestation. The network is initiated using a start up script, that starts the networking daemon on each sensing device. The basestation can then form a network using PAND commands run as a setup script on the basestation.

3.3.2 Communications methodology

The communications engine was configured to use UDP sockets, providing best effort, connectionless packet delivery. UDP offered a sensible choice for the underlying communications protocol, as although it is possible for packets to be dropped due to errors, the high data rates used in the prototype mean the effect of this is minimised. The low overhead of UDP compared to TCP based communications means there is sufficient bandwidth available for the data rate used.

Sensing nodes passed filtered data directly to the communications engine, to be sent in real time. This was in keeping with the passive nature of data collection used by the basestation, allowing the nodes to offer a stream of data that can be queried at any time by the basestation. Whilst data collection forwards sensed data to the sink without a publish / subscribe or polling based methodology, a method of collecting data from the communications subsystem was still required. As the bulk of the processing on the basestation was performed by the interpolation algorithms, it was beneficial to retrieve data from the network when each interpolation cycle was complete, rather than deal with updated temperature data during interpolation. Hence, a polling approach was used on the basestation to retrieve data from the communications subsystem. Incoming data was stored in the network buffer, the communications system then checked for new data after each interpolation cycle, updating the values used in interpolation with the latest data from the sensors.

3.3.3 Implementation Examples

Whilst UDP communications were originally implemented, it was possible to switch between communications protocols without extensively modifying the code. To illustrate the “minimum effort” approach the communications modules offer, the UDP based communication was migrated to Bluetooth native L2CAP. However, as discussed above this method of communication meant that time synchronisation was unavailable, affecting the error handling system. Bluetooth native communications are only shown here as an example.

Figures 11-12 show the source code required for the two different communications protocols. Note that for clarity the examples are without the optional address look up table, as this would obscure the problems presented by the different between the addressing schemes used.

Whilst the two communication protocols provide similar connectionless best-effort delivery, they make use of very different hardware and communication methodologies. However, the software implementation differs in only two ways. The first difference is the modification of:

```
link = ethLinks.udpLink(....)
to
link = btLinks.l2capLink(....)
```

this informs the SocketSet which network protocol to use, and is a simple substitution for the desired communications protocol.

The second alteration of

```
link = ethLinks.udpLink("192.168.0.1",8080)
to
link = btLinks.l2capLink("00:00:00:00:00:01",0x1001)
```

And

```
sockSet.sendData(data,("192.168.0.2",8080)
to.
sockSet.sendData(data,("00:00:00:00:00:01",0x1001)
```

illustrates the requirement to modify the addressing scheme to use the hardware based addressing scheme of Bluetooth rather than UDPs IP address based scheme:

```
import socketSet

import ethLinks

def onRecv(address,data):
    """Simple callback method that is processed when we get data"""
    print "%s sent %s" %(address,data)

def setupSockets(self):
    """Setup the socketset"""
    #Listen on someAddress port 8080
    link = ethLinks.udpLink(("192.168.0.1",8080))
    #And a socketset to use this link
    sockSet = socketSet.SocketSet(link,theHandler = onRecv)
    #Use callbacks and serve forever
    sockSet.serveForever()

def sendData(self,data):
    """Send data via the socketset"""
    #Send to another Address port 8080
    sockSet.sendData(data,("192.168.0.2",8080)
```

Figure 11: UDP communications code

```
import socketSet

import btLinks

def onRecv(address,data):
    """Simple callback method that is processed when we get data"""
    print "%s sent %s" %(address,data)

def setupSockets(self):
    """Setup the socketset"""
    #Listen on address 00:00:00:00:00:00 port 0x1001
    link = btLinks.l2capLink(("00:00:00:00:00:00",0x1001))
    #And a socketset to use this link
    sockSet = socketSet.SocketSet(link,theHandler = onRecv)
    #Use callbacks and serve forever
    sockSet.serveForever()

def sendData(self,data):
    """Send data via the socketset"""
    #Send to 00:00:00:00:00:01 port 0x1001
    sockSet.sendData(data,("00:00:00:00:00:01",0x1001))
```

Figure 12: L2CAP communications code

3.4 Interpolation Methods

The generation of field-like visualisations is essentially a problem of interpolation from sparse and irregular points. Given a set of known data points representing the nodes' perception of a given measurable parameter of the phenomenon, what is the most likely complete and continuous map of that parameter? That is, with only a small number of points within the area for which a measurement can be taken, the problem is one of interpolating between them to produce a complete visualisation.

In the field of computer graphics, this problem is known as an unorganised points problem, or a cloud of points problem. Since we assume that the position of the points in xy is known, the third parameter can be thought of as height and surface reconstruction algorithms can be applied.

Simple algorithms use the point cloud as vertices in the reconstructed surface. These are not difficult to calculate, but can be inefficient if the point cloud is not evenly distributed, or is dense in areas of little geometric variation. Since the interpolation is linear, it does not usually produce a good approximation of the propagation of the phenomena.

Approximation, or iterative fitting algorithms define a new surface that is iteratively shaped to fit the point cloud. Although approximation algorithms can be more complex, the positions of vertices are not bound to the positions of points from the cloud. For applications in WSNs, this means that we can define a mesh density different to the number of sensor nodes, and produce a mesh that makes more efficient use of the vertices. Self organising maps are one of the algorithms that can be used for surface reconstruction [16]. This method uses a fixed number of vertices that move towards the known data.

Note that surface reconstruction on typical non-overlapping terrains is equivalent to sparse-data interpolation. This kind of geometric parameter interpolation has been shown to work well for reconstructing underlying geography when the entire network has been queried [12]. However, it does not extend well to variable surfaces or overlapping local mapping, since it requires a complete data set to define the surface.

3.4.1 Radial basis function

The simple inverse distance algorithm, used in WSN applications before [14], is defined as:

$$f(P) = \begin{cases} \dfrac{\sum_{i=1}^{N} d_i^{-u} z_i}{\sum_{i=1}^{N} d_i^{-u}} & \text{if } d \neq 0 \text{ for all } D_i \\ \\ z_i & \text{if } d_i = 0 \text{ for some } D_i \end{cases}$$

Where, P is the point at which the interpolated value is required, d_i is the distance from P to the point numbered i in the N known points and z_i is the known value at point i. The exponent, u, is used to control the smoothness of the interpolation. High values lead to sharp edges between regions while low values lead to soft edges.

Simply: given a point, P, the function evaluates to the interpolated value at that point.

Looking more closely at the function, we find two parts. The second simply states that if there exists a known value at the given point, the function returns that value. It does not interpolate when there is a known value.

The first part, $\frac{\sum_{i=1}^{N} d_i^{-u} z_i}{\sum_{i=1}^{N} d_i^{-u}}$, is used when there is no known value at the given point. In this case, in the top of the fraction each known point is multiplied by a value calculated to be inverse to its distance to P. In other words, the further from P a known point is, the smaller the value it is multiplied by becomes. The bottom part of the fraction is simply the sum of all of those weights. The division, then, results in a value ranging between the smallest and highest known values.

Although much of our development work has been done in Python, because of its flexibility and the ease of porting software to the nodes after developing on a desktop, the more processor intensive tasks, like interpolation, have been implemented in C. The C implementation is then made available using a Python wrapper.

The RBF interpolation was first implemented as a function that takes a range of values representing known points and the coordinates of the point for which an interpolation is required and returns the result. This improved the speed of calculation greatly over our first Python prototype, but it was found that passing a large number of values in Pythonic form to retrieve each value (up to 480,000 times per frame) took up more time than calculating the value. So, instead, a linked list was developed in C and then wrapped in Python to produce a data structure that was fairly intuitive while in python, since it appears as an object with methods for adding, etc., but was also natively understood by the code written in C. Now when the interpolation function is called, it is simply given the list object containing the known points. The C function actually receives a pointer to the beginning of the list and so there is no copying or translation to be done.

Even with this speed improvement, in an unpredictable environment like a desktop computer, there is no guarantee of how much load is currently on the processor. It is not always possible to achieve a target frame rate of around 12FPS when interpolating each point. To compensate for this, the implementation recognises when the frame rate has fallen below the target and adjusts the quality of the visualisation. The reduction in quality is given by interpolating fewer points, but doing so in a regular grid so that the returned values can be drawn as boxes. The overall result is that the visualiser appears to move to a lower resolution as demand increases.

3.4.2 Shepard interpolation

The implementation used is QSHEP2D[11] a modified quadratic Shepard interpolation, that produces the function Q(X,Y) such that Q interpolates a set of N data values, The FORTRAN QSHEP2D implementation [10] is used, with the original FORTRAN code wrapped to allow access by the python application.

3.4.3 Nearest-neighbour interpolation

Is a simple method of interpolation, that applies to the field points the value of the nearest known data point. The nearest neighbour interpolation algorithm implemented performs a simple linear search of all known data points, calculating the Euclidean distance between these and the unknown value points and

```
Input: Point to Interpolate
closestPoint = None;
foreach Known dataPoint do
    calculate Distance from queryPoint;
    if Distance < closestPoint.distance then
        closestPoint = dataPoint;
    end
end
queryPoint.value = closestPoint.value;
```

Figure 13: Nearest neighbour pseudo-code

selects the value of the closest point. Figure 13 gives pseudo-code for the nearest neighbour interpolation

3.5 The visualisation component

A snapshot of the visualiser is shown in Figure 14. The main portion of the screen is a live display of the interpolated values received from the network. The top left-hand corner shows the current sensing mode being viewed and the algorithm used to produce it. In Figure 14 the snapshot shows temperature values interpolated using the radial basis function.

Since one of the goals of this work is the construction of a re-usable framework for WSN application development, the interface has been designed to be as flexible as possible. That is, apart from the strings containing identifiers for sense modality and their human-readable counterpart, the visualiser contains nothing that precludes it from being used to display any interpolation of scattered-point data. In fact, this generality was exploited to add to the visualisation component the ability to show relative changes in the sensing data displayed rather than absolute values, at the users request. (This mode of visualisation was used to report the “delta” values.)

Further, the graph facility of the visualiser, shown in Figure 15, has no details peculiar to this project. Using remote method invocation (RMI), the graph runs in a separate interpreter and is given all details at run-time, including the X and Y ranges, labels and, of course, data. This facility allows the user to select individual sensors for the time-series validation.

As well as providing an interface that can be used for multiple sense modalities, The authors here were keen to provide an interface suite that can be used for a variety of applications. At the moment, the layout of the visualiser is fixed. However, each component has been developed to require only the provision of a reference to screen-estate to function. In other words, each visual component is unaware of the others. This is the first step towards a UI library for WSNs. Next steps include the addition of isopleth generation and topology visualisation.

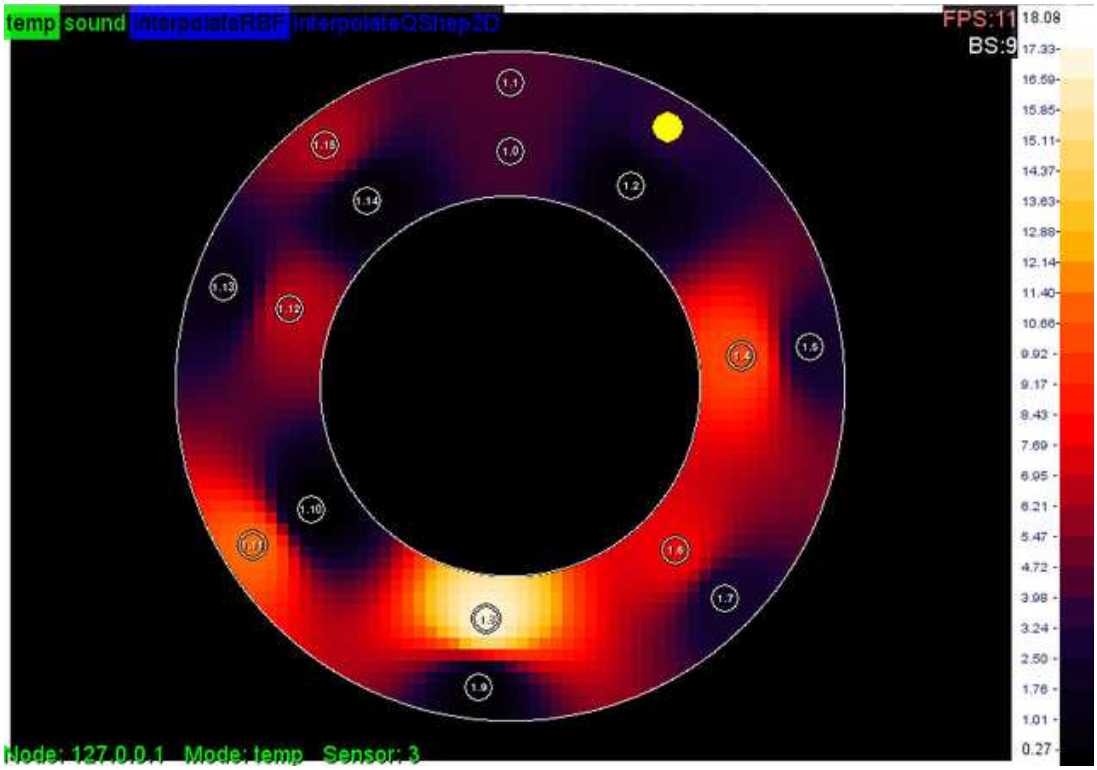


Figure 14: Snapshot of the visualiser

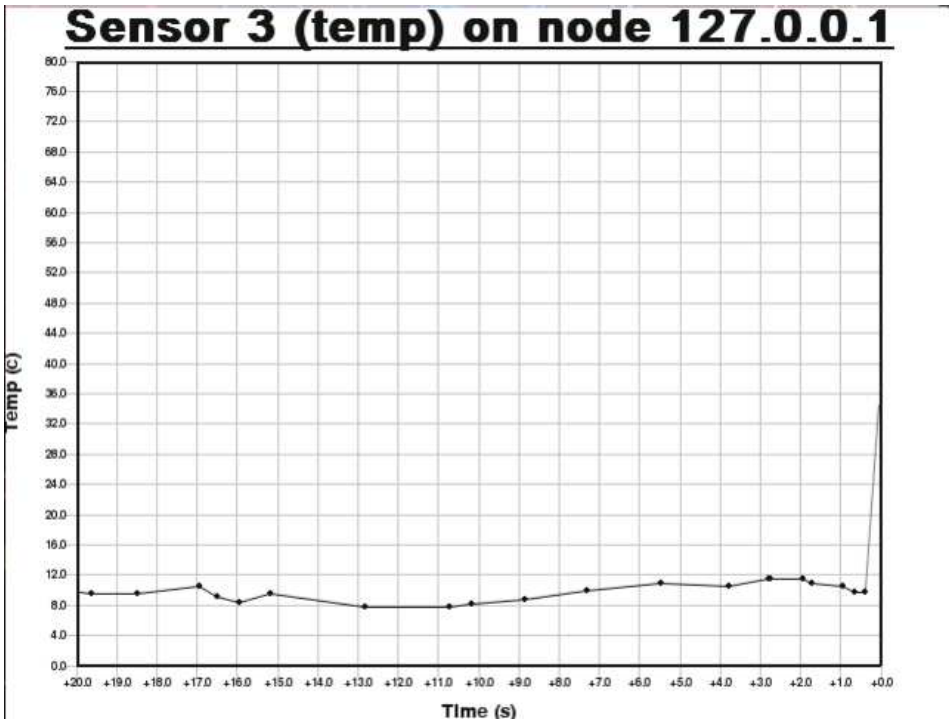


Figure 15: Historical data graph component of the visualiser

Figure 16: Base station and instrumented gas turbine engine

4 Evaluation

Preliminary evaluation of the system as a whole and in parts has been performed. Over a period of 30 days, in excess of 6 million samples have been logged in the database and made available for post-analysis. Experimental results and observations, and evaluation conclusions against the set demonstrator goals are detailed below.

4.1 Experimental setup

The deployment environment for the prototype instrumentation consists of a jet pipe, with the sensors mounted in a radial pattern within the pipe. Sensors are wired in sets to five microprocessing and communication nodes mounted on the outside of the jet pipe (Figure 16). Sensor values are wirelessly transmitted to a base station (a laptop), which provides a visualisation of the sensed data or its rate of change over the monitored surface area, and, on demand, a history of individual sensor values. As well as temperature sensors, a smaller number of microphones are connected to the nodes to demonstrate the system's capability for multi-modal sensing and the ability of the prototype to function with high bandwidth sensors. All-in-all, the wireless gas turbine engine monitoring system consists of 20 temperature sensors and 4 microphones; 5 processing nodes based on Gumstix Connex 400xm-bt, with custom-built expansion board to provide audio and I^2C connectivity and a base station with visualisation software and a non-volatile database for data storage. Four of the processing nodes are part of the fixed experimental set-up whilst the 5th is a so called "*probe*" node used for evaluation. Temperature sensors are arranged in a similar manner to Vibro-Meter's existing thermocouple harness shown in Figure 17.

The basic annulus configured with sensors and Gumstix microprocessors is shown in Figure 18. The

Figure 17: Annulus thermocouple harness (picture supplied by Vibro-Meter UK)

annulus came with holes that were intended for thermocouple sensors. Dowelling was placed into these holes to support the I^2C sensors at two insertion depths: one roughly on the rim and one halfway between the rim and the central cone. The I^2C sensors were connected to the Gumstix processing and communication nodes situated on the outside of the annulus, with 4 temperature sensors per Gumstix, 2 sensors on each I^2C data cable. The microphones were mounted on the Gumstix. The experimentation took place in the Cogent Computing lab, a room of approximately 4m x 8m with natural ventilation through windows and doors. Two types of heat actuation tools have been used: a hair drier and several stage lights.

Some of the experiments were carried out with the annulus in the open environment as shown in figure 18, whilst some of the experiments were slightly more controlled in terms of convection by covering the top of the annulus with clingfilm as shown in figure 19.

Figure 18: Annulus with I^2C sensors and Gumstix attached

Figure 19: Reduced convection experiment

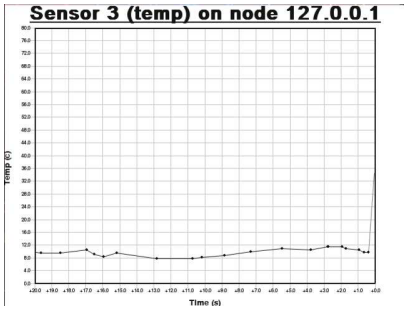


Figure 20: Graph module output

4.2 Fulfilment of the application specific goals

4.2.1 Functionality based on User Requirements

The prototype has been developed in a two stage process - first, basic functionality was implemented according to user requirements; a demonstration of the functionality was given and allowed for focused user feedback. Based on this, addition of functional features followed, in order for the design to better achieve the perceived goals of the prototype. The following three features are notable from within the functional enhancements:

Provision of "delta" field visualisation The original prototype design displayed field representations of current temperature values. Whilst this allowed a view of the overall state of the sensed phenomena, the end-user claimed that a more effective way of interpreting the data acquired by the prototype is through observation of temporal temperature variations. This feature was catered for through the implementation of a "delta" mode. This involved adding another level of in network processing, at the sensing nodes that measured the current rate of change of the sensed temperature, allowing changes to the values to be communicated to the base station and displayed.

Ability to request historical information on temperature readings. Time-series based graphing functionality has been developed to allow the user to display, on request, sensor historical values. Figure 20 gives an example of the information displayed by the graphing module. Similarly to the "delta" mode, this allows the user to observe the changes in sensed data values, creating a better picture of the state of the system being monitored in case of interesting events.

Tools and features to allow assessment and comparison of different interpolation methods To assess the usefulness of the interpolation options offered by the system, a method of validating the field representations produced by the interpolation algorithms was required. To this end the "leave one out" cross validation approach was developed as discussed elsewhere in the report. Moreover, a "probe node" has been added to the system of fixed sensors, holding four temperature sensing devices, able to be

placed at an location within the field of interest, in order to validate interpolated data accuracy between the fixed sensing points.

4.3 Fulfilment of software engineering goals

4.3.1 End to end system performance

Several issue have been considered when evaluating the end to end systems performance, with particular attention given to system robustness. This was evaluated through:

Repeated start-stop tests Within a period of four hours the system has been started 5 times and found to function correctly most times. During one such test required the NTP server to be reset as the time forwarded to the Gumstix was incorrect. Using a protocol such as NTP that is dependent on a external network connection has the potential to cause failures in time synchronisation if the system is deployed outside of laboratory conditions. To this end, the use of a time synchronisation protocol such as [13] would allow the network time to be synchronised to that of the base station, providing a global system time based on the basestations current clock, regardless of the correctness of this compared to world time servers.

Further tests involved the system being dismantled from the pipe, used on a table top, re-arranged on the pipe and tested. No loss of functionality has been observed.

Continuous functional tests The system has been left to gather data continuously for a period of 7 days without node failure, or incurring any data loss. Repeated functional tests have led to very rare node failures, with the worst case being that of one node failing after 20 hours over a 24 hours test; this was due to the Bluetooth connection between the node and basestation failing, although the reason for this was unclear.

Given that in the current system design the Bluetooth network is created manually each time a test or demonstration is needed, this has allowed modification to the network parameters such as node connection/topology, or master slave role swapping between devices for testing purposes. In further prototypes, it will be possible to have the Bluetooth connections created during the devices boot cycle, in which case one can create a persistent Bluetooth connection, where the instrument monitors the Bluetooth link and attempts to reconnect if the link is broken.

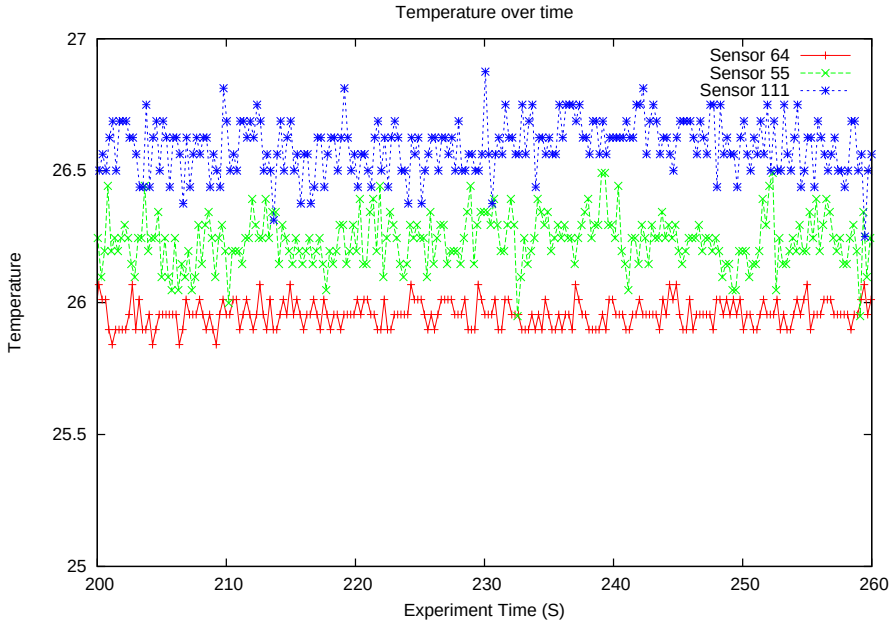
In the current prototype, various problems with the Bluetooth connection were diagnosed using the utilities provided by the Bluetooth stack; these include hcidump that can output a report of all packet information between paired Bluetooth devices.

4.3.2 Sensing performance

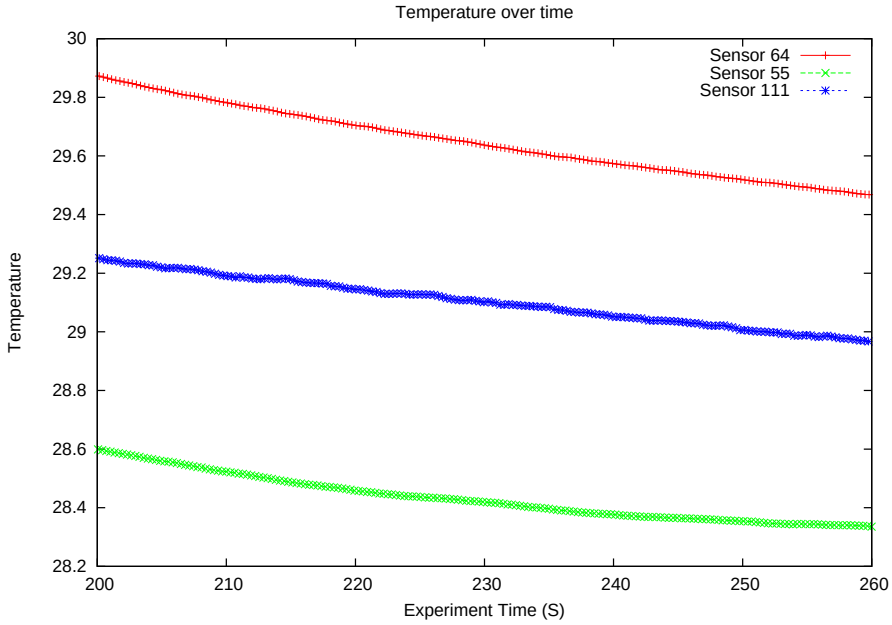
As with any measurement system, sensor calibration is an important issue. The system reported here has been built with low cost integrated temperature sensors that would not be suitable for industrial de-

ployment where reliably accurate temperature measurement is required. Repeated experimentation with the ADT75A sensors showed that in near uniform environments their response varied greatly. Both offset and slope errors in factory calibration are significant.

Use of a Kalman filter has helped to remove noise from sensor data. Figure 21 compares filtered and non filtered temperature data from three sensors in the system over a 60 second period in a even un-stimulated indoor environment. Figure 22 shows filtered temperature data from 16 sensors (grouped by associated Gumstix) over a 16 hour overnight period. The room environment was, during the data collection, even over the sensors set. However the sensor reading show variations of up to 3.6°C on the maximum sensed temperature and of 2.5°C on the minimum sensed temperature. Sensor sensitivity variations have also been observed.



(a) Unfiltered Sensor Data



(b) Filtered Sensor Data

Figure 21: Comparison of unfiltered and filtered uncalibrated Sensor Data

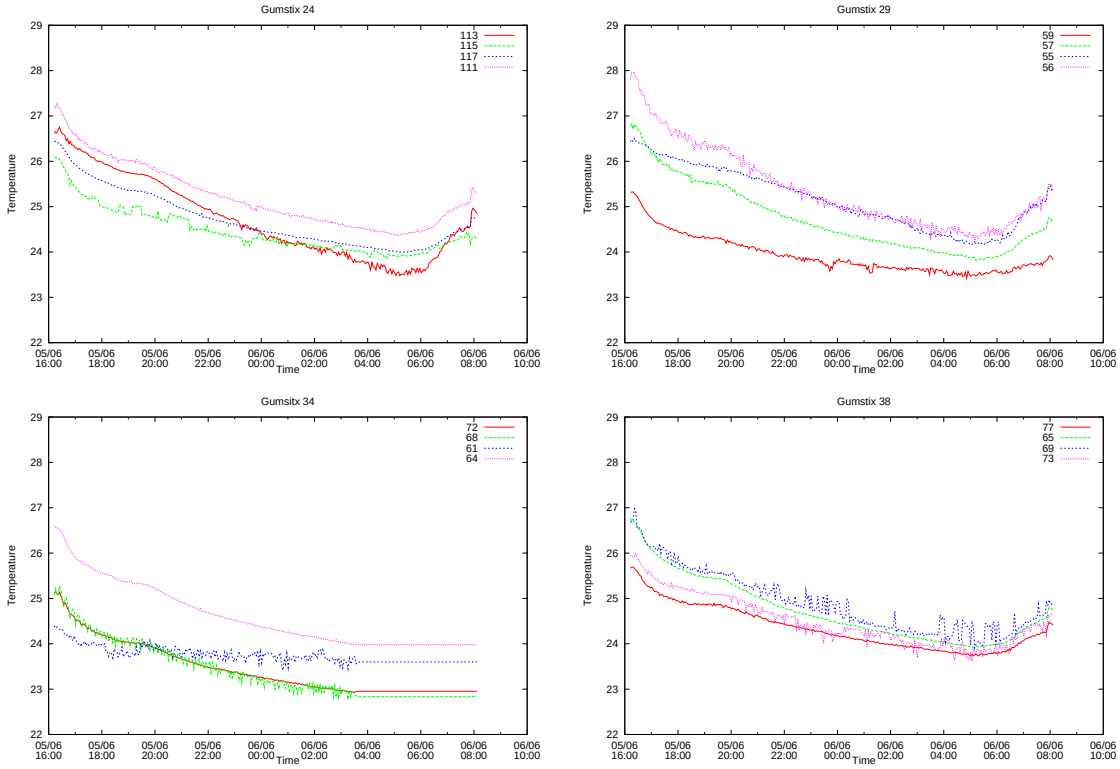
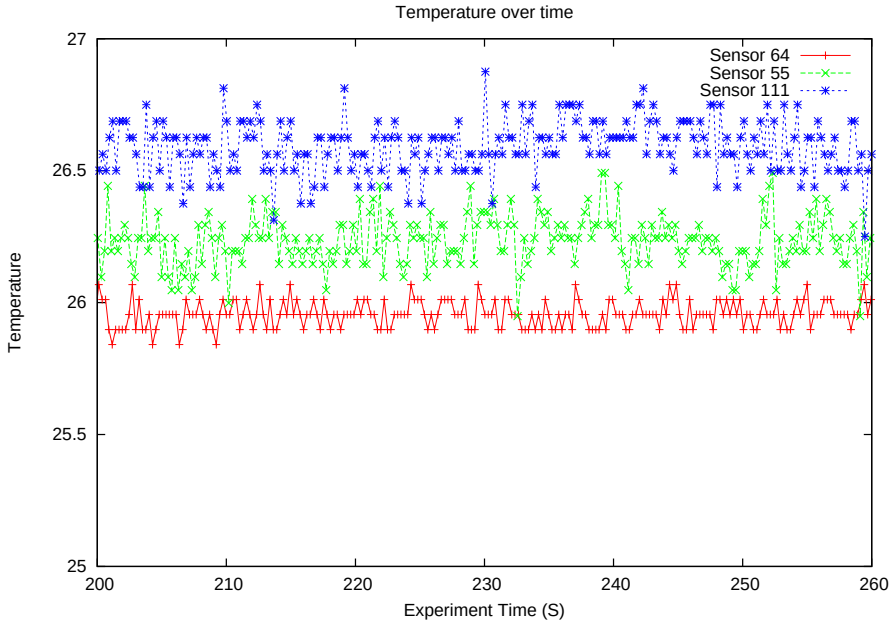


Figure 22: Filtered temperature data over a 16 hour overnight period

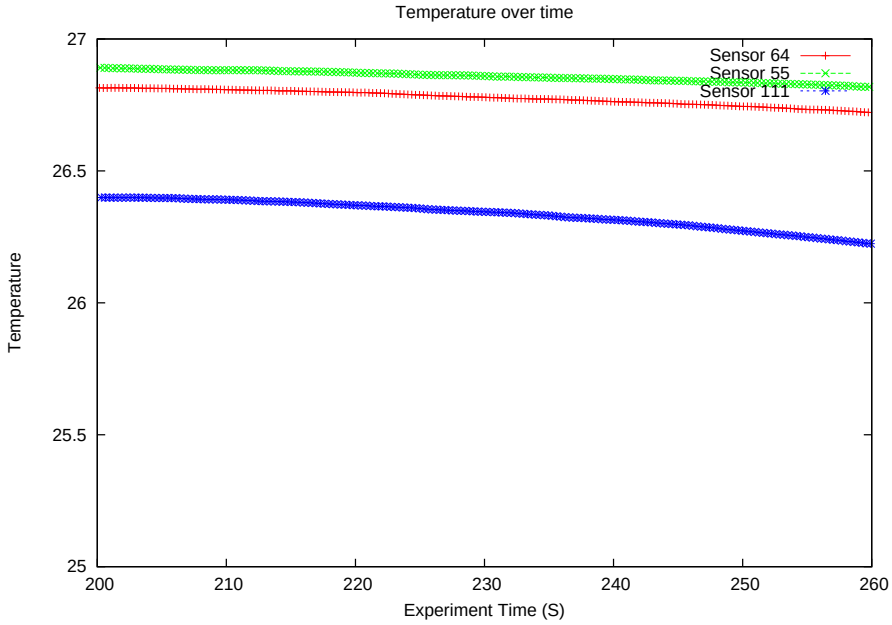
Whilst the effects of sensor to sensor and sensor to ground truth variation can be alleviated through additional calibration (using the integrated calibration facility) experimentation has shown that the temperature response of the ADT75A tends to drift over time and an adjustment to the calibration is needed every so often. The calibration errors in sensor values prevented, initially, a clear evaluation of the absolute temperatures visualisation and interpolation procedures. However, the “rate-of-change” (“delta mode”) visualisation method removes some of the problems to do with lack of calibration as it inherently ignores offsets between sensors and is hence explored and evaluated more fully below.

Lengthy experimentation has revealed a design fault in the sensor boards, leading to high instability of the sensed data. 100nF capacitors were fitted on the temperature sensor boards which lead to the performance of the sensors increasing as shown in figure 23. The capacitor smooths the power supplied to the node, and removes fluctuations in the data supplied to the end user, increasing the accuracy reliability of the gathered data. Future work will assess the usefulness of filtering the sensed data with the sensor-capacitor pairs, as the increase in reliability of raw data values compared to that gathered by nodes without capacitors, may make this smoothing step unnecessary. Analysis of the end-to-end data processing and transmission time 4.3.3 indicate that the time overhead associated with this filtering step is approximately 0.18s. Whilst at current sampling rates of 4 samples a second the impact of this on overall performance is minor. If further experimentation with capacitors proves that there is a significant reduction in sensor noise there is scope for increasing the processing performance of the system without adversely affecting accuracy by removing the Kalman system.

Given that in the future the project aims to use high quality thermocouples with well specified properties and well understood behaviour in the next prototype, it is expected that, from a viewpoint of sensed data accuracy, the system will undergo further development and evaluation.



(a) Unfiltered Sensor Data



(b) unfiltered data with capacitor

Figure 23: Uncalibrated sensor data without and with on board capacitor

Bluetooth 2.1 + EDR Features and Specification.

Maximum Permitted Power	2.5mW (4dBm)
Range	10m
Signalling Rate	3Mbits/s
Transmission Speed	2.1Mbits/s

Table 3: Class 2.1 Bluetooth device specification

4.3.3 Network performance

In a system designed to monitor and provide information on sensed phenomena, the timeliness of the displayed data is important, old data can lead to the incorrect information being displayed, which in turn affects any decisions made based upon the information offered by the system. To this end the wireless network performance has been analysed in terms of latency, data throughput, bandwidth and communications range.

A selection of network level tools are used for this, including the hcidump Bluetooth debugging tool provided within the Linux Bluez Bluetooth stack. This provides packet level information on all Bluetooth devices paired with the basestation. The logs supplied by the hcidump tool were analysed with the wireshark analysis tool.

Device Bluetooth Specification The Bluetooth module used to provide network communications is a Infineon singlestone PBA 31308 [3], which follows the Bluetooth 2.0 +EDR specification. Table 3 shows the specification for Class 2 Bluetooth devices.

Maximum transmission range The maximum transmission range has been found to be 11m indoors after which the transmission signal was lost. This is consistent with the quoted range for class 2 Bluetooth devices. Class one Bluetooth devices have a quoted range of 100m, hence if further transmission range was required, upgrading the Bluetooth device would allow an increase in transmission distance. The systems design is such that no software changes are required when a change of device class is made.

Latency The latency performance of the network has been measured in several circumstances, designed to give a overview of systems performance under various conditions.

The following conditions have been selected to provide an cross section of circumstances the application will potentially operate under:

- Normal operational conditions
- Maximum sampling rate applied

- Transmission of unfiltered data
- Noisy Bluetooth environment

Each experiment followed the same approach: sampling 10 minutes of data under stable unstimulated conditions. Timestamps were taken when the system requested data from the sensors, and when the data was passed to the visualiser by the basestation, giving an indication of performance of the entire sampling process (gathering - filtering - transmission - visualisation). The data recorded in the database was then analysed using tools to generate latency information. Packet information was also recorded using the *hcidump* Bluetooth packet sniffing tool.

Operation under normal conditions This experiment was intended to provided reference latency figures, offering a baseline for systems performance, and allowing comparison against the latency figures gathered from the other experiments. Table 4 shows the latency figures for this experiment. Figure 24 shows a graph of the latency values over the course of the experiment, after the initial start up phase.

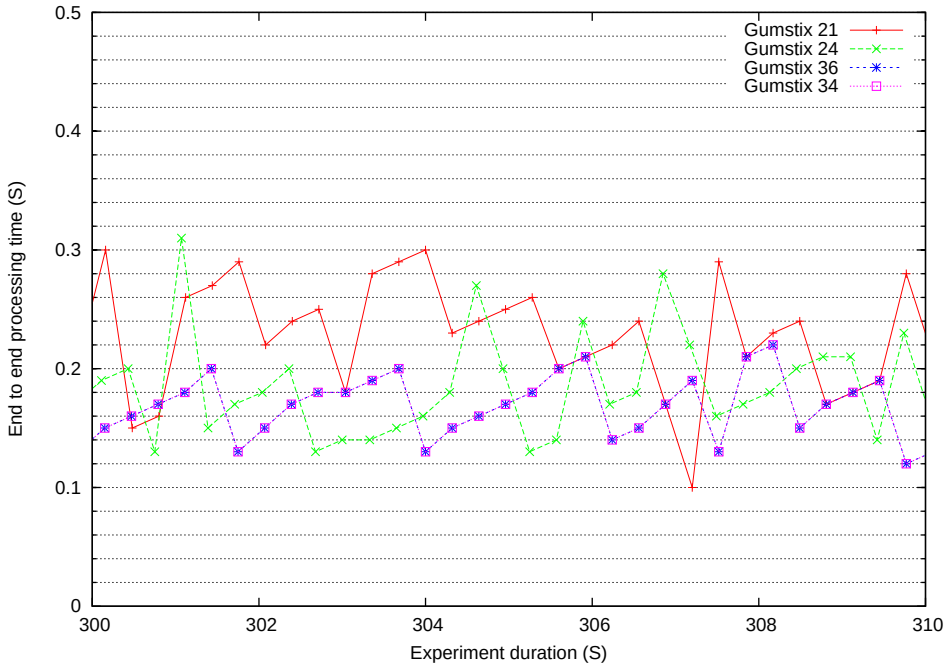
Latency	Value in seconds
Minimum	0.03
Maximum	8.48
Mean	0.568
Median	0.270
Lower Quartile	0.210
Upper Quartile	0.380

Table 4: Latency figures for operation under normal conditions

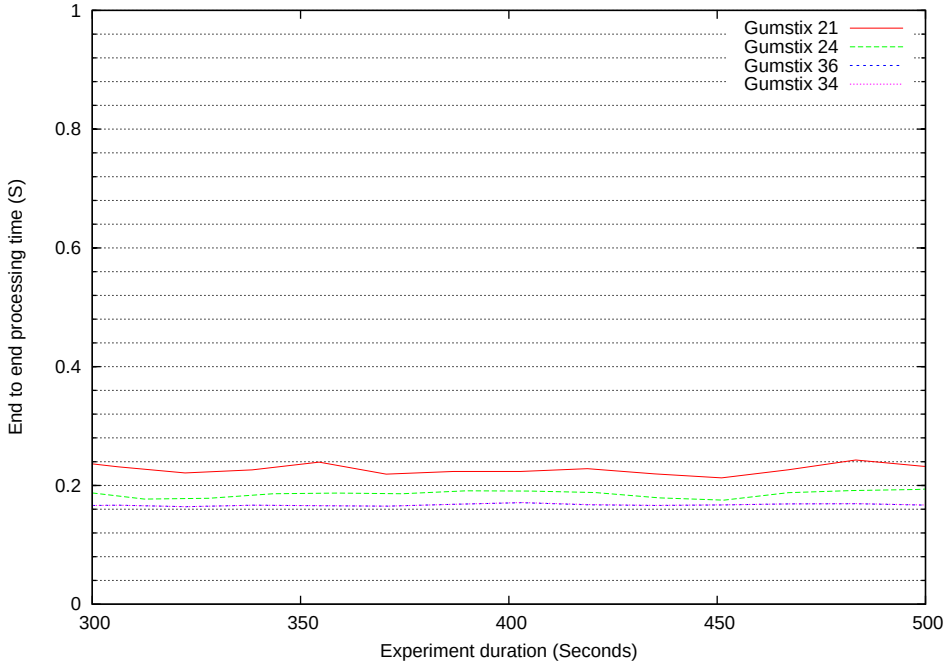
The Maximum latency value of 8.48s corresponds to the systems start up process, where the clock is synchronised with the basestation takes place. During this period the temperature sensing nodes attached to the device are also discovered, requiring a poll of all possible I^2C device addresses. This can cause the bus to timeout where no device is connected. After this process is completed the devices known to be connected are used to gather temperature data and the latency reduces considerably. Whilst the mean end-to-end processing time of 0.56s is high, the figure is skewed by the high latency values associated with systems start up, therefore the median value, of 0.270s gives a more accurate measure of the systems performance. This average end to end transmission time is acceptable in this application, given that in current engine monitoring applications temperature data is sought on average once every second.

Transmission of unfiltered data The filtering of data to “smooth” the samples forwarded to the visualiser requires a significant amount of processing on the sensing nodes. This experiment turned off the filtering process on the sensor nodes, to provide a baseline for the network performance, where no node processing takes place. Table 5 shows the latency figures over the course of the experiment.

As expected, removing the processor intensive filtering operation performed on the nodes improved the end to end processing time of the system. Again the maximum end to end transmission time is high

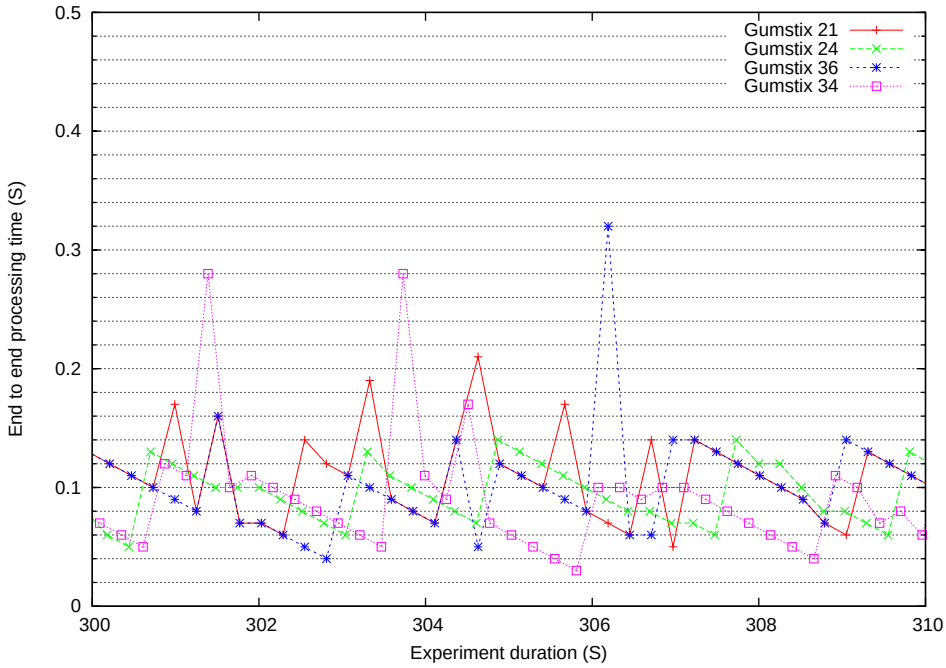


(a) Latency over 10 seconds

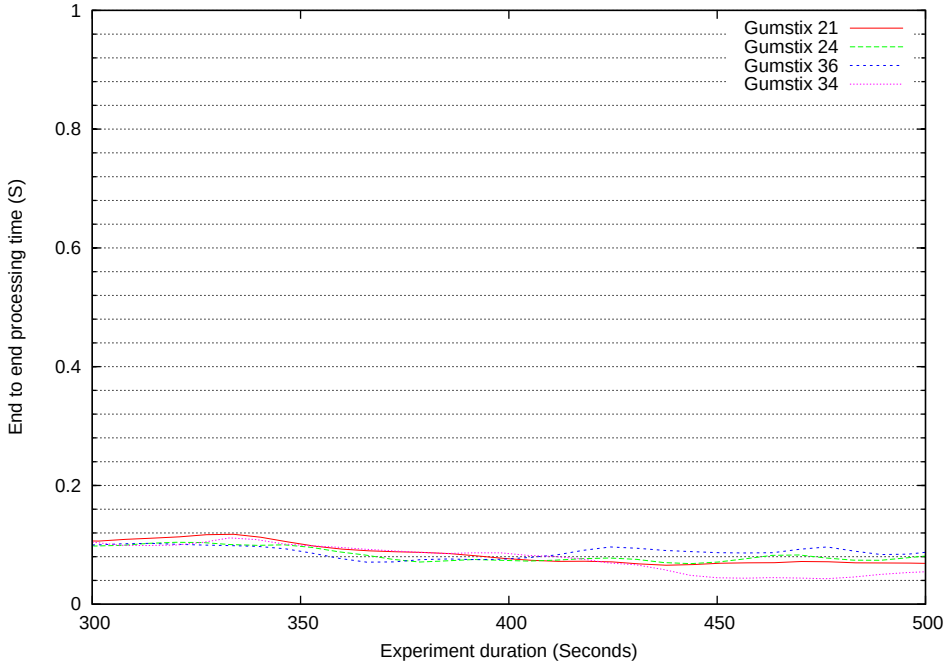


(b) Latency over expermet duration

Figure 24: Latency graphs of operation under normal conditions



(a) Latency over 10 seconds



(b) Latency over expermet duration

Figure 25: Latency graphs with unfiltered data

Latency	Value in seconds
Minimum	0.01
Maximum	5.1
Mean	0.100629
Median	0.09
Lower Quartile	0.06
Upper Quartile	0.11

Table 5: Latency figures when transmitting unfiltered data

due to the start up process. It can be noted that the difference between the Median times in the tables above indicates a 0.18s overhead per sample to filter the data. Should unfiltered data be transmitted, the filtering operation needs to be performed before the visualisation/interpolation step. The unfiltered data processing latency gives an idea of the baseline performance of the Bluetooth radios used in this work, as only the data gathering and transmission time is taken into account, rather than the entire gathering-filtering-transmission process.

Figure 25 provides some latency graphs over the experiment, after system start-up phase has finished.

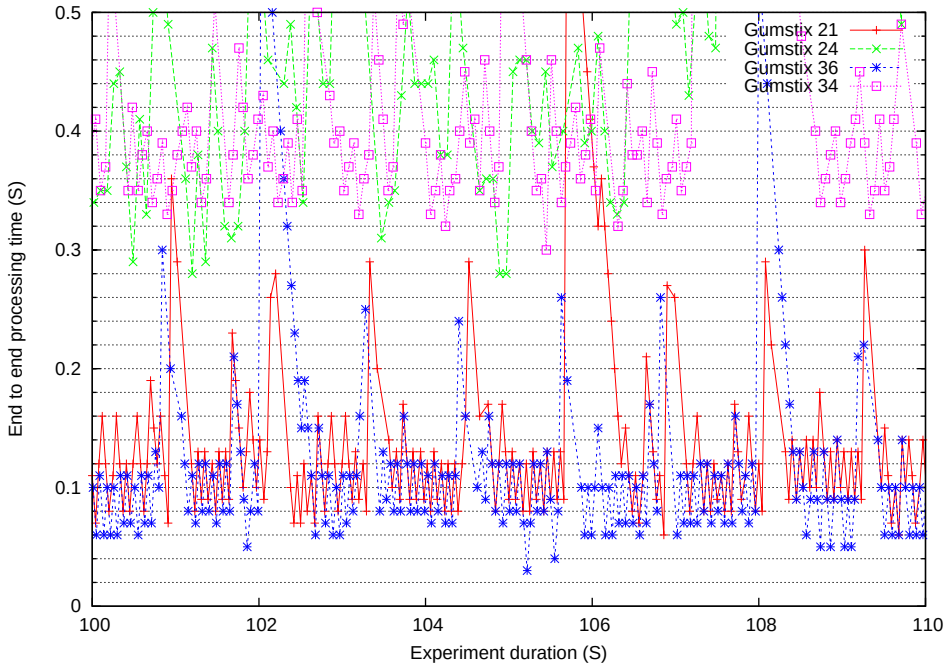
Operation using maximum sampling rate. Here the system was run at the maximum operational rate, i.e. with data sampled and forwarded at the fastest rate possible. This will provide an indication of the systems performance at the hardware's operational and processing limits. Table 6 gives the latency information gathered during the course of the experiment. Figure 26 provides some latency graphs over the experiment, after system start-up phase has finished.

Latency	Value in seconds
Minimum	0.03
Maximum	15.20
Mean	0.502
Median	0.21
Lower Quartile	0.12
Upper Quartile	0.33

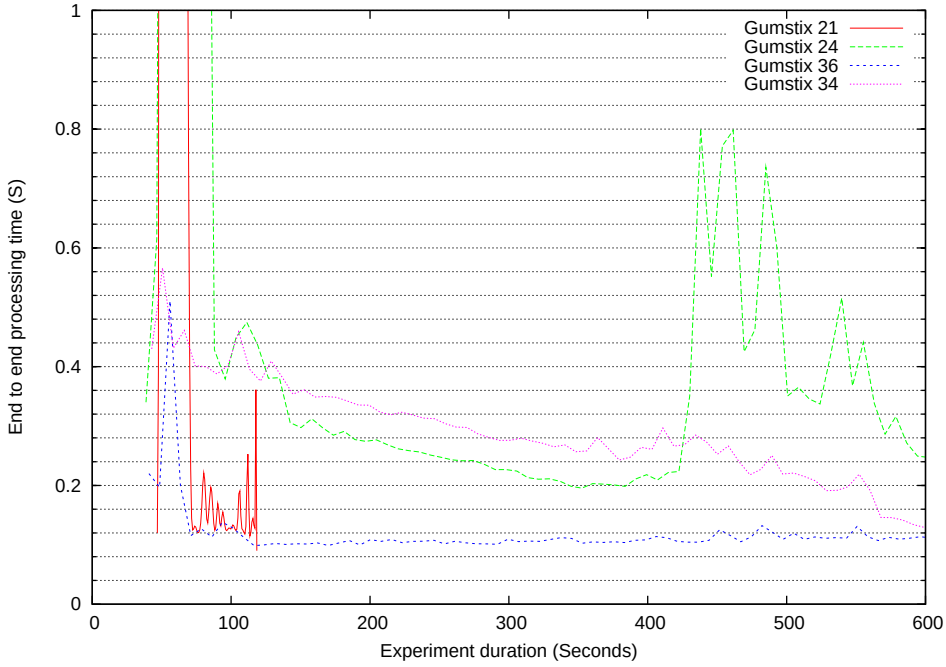
Table 6: Latency figures at maximum sampling rate

Statistically the end to end processing times here are similar to those generated using the standard experiment, although the system is functioning at its maximum sampling rate. Each node is sending a temperature data sample on average every 0.05 seconds rather than the standard setups 0.25 of seconds.

Operation within a noisy Bluetooth environment Although Bluetooth performs automatic channel hopping as an effort to maximise throughput, it is expected that a noisy Bluetooth environment will result in lower throughput within the system. This experiment gathered latency figures for operation within such a environment, and took part in two stages.



(a) Latency over 10 seconds



(b) Latency over expermet duration

Figure 26: Latency graphs for operation with maximum sampling rates

Latency	Value in seconds
Minimum	0.09
Maximum	15.38
Mean	0.544
Median	0.260
Lower Quartile	0.200
Upper Quartile	0.380

Table 7: Latency figures for operation with background noise

Latency	Value in seconds
Minimum	0.1
Maximum	10.1
Mean	0.479
Median	0.3
Lower Quartile	0.23
Upper Quartile	0.4

Table 8: Latency figures for operation within noisy network conditions

The first stage of experimentation was a repeat of the first baseline latency experiment performed, though a large amount of background Bluetooth noise was added by creating a separate network of seven Bluetooth nodes. These nodes then transmitted ICMP (Internet Control Message Protocol) pings to provide a known level of background noise. Table 7 shows the latency statistics for this experiment.

The second stage of experimentation added an extra 3 nodes to the applications piconet, with 1024byte ICMP pings from these nodes every 0.2 seconds. This added a level of extra traffic to the sensing network and provides an indication of the systems performance under these conditions. Table 8 gives the latency information during this experiment.

Experimentation within a noisy Bluetooth environment shows little difference between the prototypes operation in both clean and noisy environments, suggesting that the native Bluetooth channel hopping algorithm is effective at combating network congestion due to background noise.

XML performance The use of XML to transmit data also increased the transmission overhead as opposed to a raw data stream. Tables 9 and 10 show the network overhead resulting from the use of XML to package sound and temperature data for transmission during an experimental run.

```
<frame version="1" mode="temp">
<tstamp>1214398901.21</tstamp>
<sample id="4" cov="1214398901.22">26.6875</sample>
<sample id="5" cov="1214398901.22">26.875</sample>
<sample id="6" cov="1214398901.22">27.3125</sample>
<sample id="7" cov="1214398901.22">25.9375</sample>
</frame>
```

	Total Bytes	Data Bytes	XML Bytes	Overhead (%)
Average	278	103	175	62.94

Table 9: Sample XML packet and XML overhead for temperature data

```
<frame version="1" mode="sound">
<tstamp>1214398899.67</tstamp>
<sample id="0" cov="1214398900.83">95</sample>
</frame>
```

	Total Bytes	Data Bytes	XML Bytes	Overhead (%)
Average	120	35	85	70.83

Table 10: Sample XML packet and XML overhead for sound data

Given the current data rates, it means that there is a large overhead caused by the current XML scheme. This gives scope to improve the network performance by optimising the schema used. However, obviously, if sample rates are to be increased or multiple sound sensors are to be connected to one node, the percentual overhead will decrease. For the system usage foreseen here, Table 11 gives an example of one optimised scheme for the sound data given above. Although optimising the XML scheme offers a small improvement in efficiency, any future improvement would require a different approach to formatting data for transmission. However as discussed below optimising the XML schema is currently unnecessary, as the current throughput is only a small percentage of the total available bandwidth.

Network Overhead Given that the network was formed using Ethernet emulation over Bluetooth using the Bluetooth PAN daemon, a level of overhead was expected in the data transmitted, as each packet was formatted for transmission using UDP packets. Data on latency and timing is gathered from the database, as timestamps are logged both when the data is gathered from the sensors and processed on the basestation.

The overhead of using TCP emulation over Bluetooth has been analysed using logs provided using the network analysis tools described previously in the report. Table 12 shows the overhead associated for a typical data packet sent during the systems operation whilst transmitting data using various network

```
<f v="1" m="t">
<t>1214398901.21</t>
<s id="4" c="1214398901.22">26.6875</s>
<s id="5" c="1214398901.22">26.875</s>
<s id="6" c="1214398901.22">27.3125</s>
<s id="7" c="1214398901.22">25.9375</s>
</f>
```

	Total Bytes	Data Bytes	XML Bytes	Overhead (%)
Average	190	103	87	54.2%

Table 11: Sample XML packet and XML overhead for temperature data

	Header Size (Bytes)	Payload (Bytes)	Total Packet size (Bytes)	Protocol Overhead (%)
UDP	42	274	316	13.29%
Bluetooth HCI frame	40	274	314	12.73%
Bluetooth L2Cap Packet	31	274	305	10.16%

Table 12: Network Transmission overhead using TCP emulation over Bluetooth

communication protocols. The packet header overhead for the communication protocols is shown, along with the total payload and percentage of packet taken by the protocol header. The table shows little difference in requirements for each protocol, with the lightweight Bluetooth native L2CAP having the least overhead requirements.

The use of Bluetooth TCP emulation also involves a small amount of network overhead used in flow control and routing. Table 13shows the figures gathered during a typical sampling run.

The figures in Table 13 show that even with an unoptimised XML schema, and the overhead associated with the TCP/ IP suite of protocols, the application can support a high level of expansion before the throughput exceeds the available bandwidth. The current sampling rate of 4Hz on 18 sensors, requires only 2% of the total available bandwidth. This suggests that the system could be extended use approximately 1050 temperature nodes sampling at the same frequency without impacting performance.

Total no of Packets transmitted	10430
Overall Experiment Time	526s
Avg Packets/s	19.825
Avg Packet Size	271.108b
Bytes transmitted	2827657
Avg. bytes/s	5374.804
Avg. MBit/s	0.043

Table 13: Network statistics during a typical data gathering run

Table	Record Size	Total Records	Table Size
event	62 bytes	280	17.1Kb
sample	64 bytes	9,605,406	582.8Mb
sensor	56 bytes	75	4.1Kb
sensorConf	85 bytes	8,221	681.3Kb

Table 14: Database Record size with records stored during 280 experiments, total xxx hours of monitoring

4.3.4 Power management

The systems design allows the network to be powered using either mains or battery power, although no power management is implemented in the software. Within the prototype system the power consumption is less of a concern than the correctness of the data gathered and processed. To date, no assessment of power consumption has been made.

4.3.5 Persistent Storage and Database

Storing the gathered data in a remote database provides the opportunity of preforming post analysis on gathered data. Several tools have been implemented to this effect, acting on the database.

The first of these tools provides a 'replay' function, which allows a user to view previous experimental runs in the visualiser, similarly to the real-time play. This allows interesting phenomena to be framed in time and post-analysed, using the same visualisation tool as that available for real-time analysis.

The information stored in the database also allows system performance analysis to be performed as detailed above with regard to latency, etc. Gathered data analysis, including the "leave one out" cross-validation approach used to evaluate the usefulness of the interpolation algorithms is also enabled.

Normalisation of the database tables has given an efficient data storage mechanism, with data gathered from 280 experiments (over 9million separate records) stored within 580Mb. Table 14 shows the record size, for each table in the database and the overall table size at the present time.

4.3.6 Distribution of source code updates.

Updates to the code used on the sensing nodes can be distributed using a script that takes the latest version of the application code and forwards it to each node in the application. This automated source code distribution allows modification to the sensing nodes code base to be distributed from the basestation. This method of distribution is suitable for small single hop networks, but a more efficient method of source code distribution and modification may be required for a larger multihop network. Whilst current source code updates are "offline" with application code updated whilst the system is not operational, experimentation has taken place with "real-time" source code updates, allowing the application code to be modified whilst the system is operation. It is envisaged that this method of updating will be integrated into the system shortly.

4.3.7 Extending the system

The modular design of the software allows nodes to be interchanged in case of hardware failure. The prototype was originally designed to use four sensing nodes to gather data on the sensed phenomenon although new nodes or temperature sensors can be introduced to the system to increase the resolution of the field data. On systems start-up the application polls the I^2C bus to discover which temperature sensors are attached. (Polling addresses without a sensor attached causes a timeout on the bus.) These sensor Id's are recorded and used to gather data and network related information from the attached sensors in a timely fashion. This design allows sensors to be swapped between nodes and new sensors to be added (or excluded) without modification of the source code, providing a simple method of both maintaining and extending the system.

To evaluate the above capabilities and also in order to aid evaluation of the visualisation and information extraction components, the system has been extended beyond the 4 nodes, with a "probe" node, equipped in the same way as the nodes used to gather data during normal operation. This node can be deployed at any place within the jet pipe cross section. The probe node has been used throughout the instrument evaluation to provide temperature data values between the fixed sensors, providing extra information for the cross validation process.

Consequently, currently there are 5 nodes in the network, configured into a single Bluetooth piconet. Given that a piconet can support up to 8 active network devices, the system is extensible up to this limit with the only changes required to the software support being pairing the devices with the basestation in the system start-up script. If further nodes were to be added beyond the 8 nodes, the network would become more complex as Bluetooth scatternets would be required to be built, allowing multiple piconets to be connected, with one node acting as a gateway between piconets. The network performance would suffer here, as multihop routing is necessary to transmit data between the devices in two separate piconets. Depending on the design of the Bluetooth network, the end to end latency of the system would increase due to the time taken to process and forward messages. Such a scatternet of 12 Bluetooth nodes has been developed and deployed and is documented and evaluated elsewhere.

4.3.8 Fault isolation

Given that the instrument itself is aimed at engine health monitoring, it is of out most importance that instrument faults are detected, isolated and managed, leading to increased data confidence. Whilst designing and implementing a full blown fault management system for a wireless network is a large research effort in itself, a placeholder for such a component has been catered for n this design and a simple fault isolation and management component was deployed to cater for nodes irrecoverable faults and sensing devices faults (presence at bus level and data freshness). To this end each data sample is given a time stamp when it is gathered. This time is taken from the global system clock synchronised with the basestation using NTP. Upon receiving a data sample, the basestation software updates global sample list with the most recent sample received from each node. If the most recent sample received is over 30 seconds old, the data from this node is marked as stale and ignored in any interpolation and visualisation routines. Although this method provides no fault diagnostics or error correction removing erroneous data from the display reduces the potential to misinterpret the data displayed. Cross validation of the visualiser as dis-

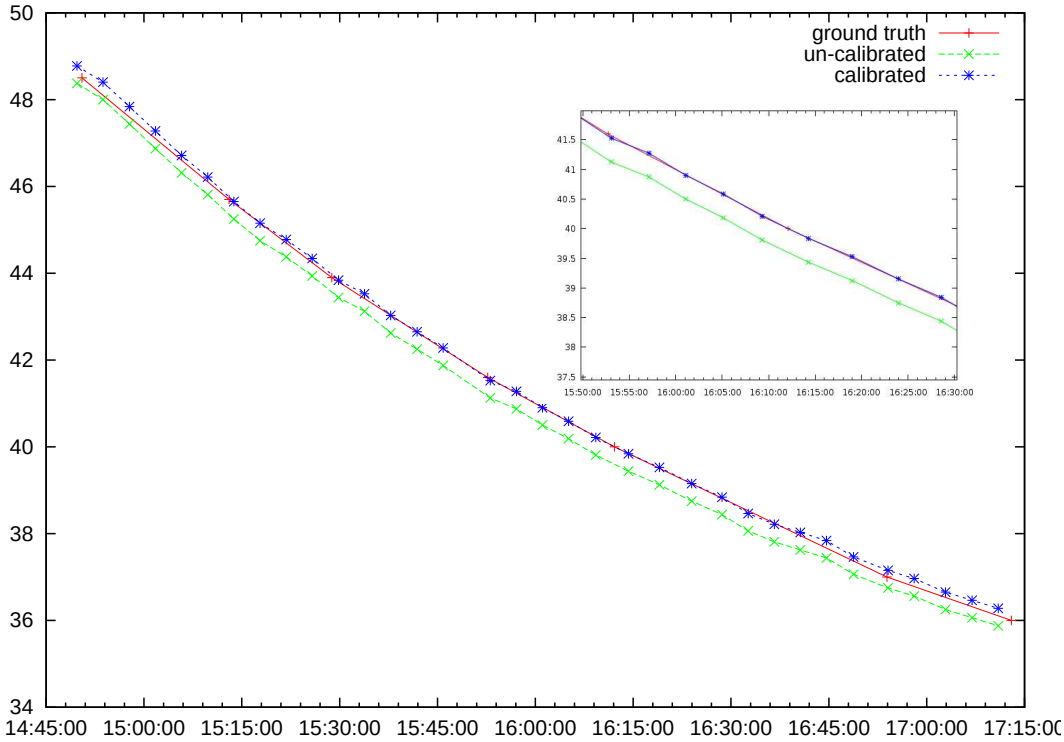


Figure 27: Calibrated against uncalibrated temperature readings

cussed below, has shown that the interpolation algorithms used can provide a high level of confidence in the data displayed in areas with no physically sensed data.

4.3.9 Sensor Calibration

It is important to provide accurate data for the visualisation and interpolation processes, and increase the confidence in the processed data. The ADT75A temperature sensor nodes were put subjected to calibration, to better fit the reported values to the ground truth temperature. The calibration followed a similar process to that used in industry [4], where the sensors are immersed in a water bath and recordings of the reported values taken at intervals. These readings are compared to a ground truth temperature reading taken using a reference thermometer, to provide a offset and multiplier for each sensor. During our calibration run the water bath temperature ranged between 50°C and room temperature at approximately 25°C. Calibration coefficients were then calculated through a curve fitting process and loaded to the calibration component of the prototype which then applies them automatically to the sensed data. Figure 27 shows some sensed data before and after calibration.

4.3.10 Visualisation performance

Generating field representations of the sensed / sampled phenomena is a useful means to enable event detection and isolation, and flow characterisation in the application at hand. It is important, from a information utility viewpoint however, to be able to assess / measure the accuracy of such field representations, produced through interpolation from sparse data. The evaluation of this facility, as a system component has been preformed in two ways.

- Starting from a hypothesis that sensors do fail, within a deployed system, it is interesting to asses the effect of such failures on the global, field view of the sensed phenomena for the data-to-information chain supported by the prototype here.
- Should all the sensors in the system be functional, it is important to evaluate the accuracy of predictions at points between the sensor sample points, leading to the field representation of the sensed phenomena.

With this in mind, the aforementioned “leave-one-out” cross-validation tool has been developed. The tool allows the user to retrieve historical data from the database for off-line processing and evaluation. Upon user selection of a logged test and selection of a point in time within the test, the cross-validation tool excludes the data for each sensor in turn (simulating a sensor failure) and attempts to predict it from the remaining points. The difference between the actual and predicted values provides an error value and hence an indication of the interpolation quality on the absence of sensed data at a measurement point. The root-mean-square (RMS) of the errors between the actual and predicted values over the duration of individual tests also provides a good indication of the level of error expected to be encountered in the interpolated representation if a sensor has failed. An example of interpolated versus actual sensor values for both temperature and “delta” mode during a cross validation run are given in Figures 28-29, for one of the sensor in the system (sensor 117). Figure 30 represents a snapshot of the cross validation tool interface.

In this instance, the experiment consisted of two heating and stabilisation cycles, providing a indication of the interpolation performance between 20-40°C. The heating was performed using two high powered stage lamps, placed to provide heating without “hotspots” on individual sensors. Figures 31-33 show the sensor layout during the experiment and the heating arrangement.

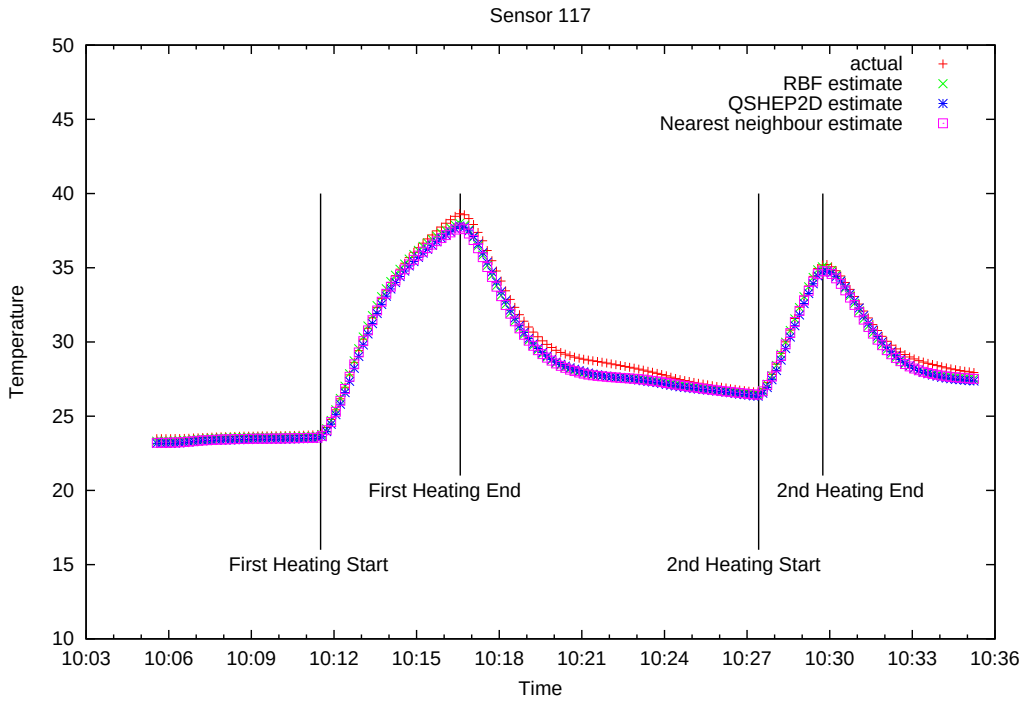


Figure 28: Interpolated temperature values against real temperature data (temperature in °C)

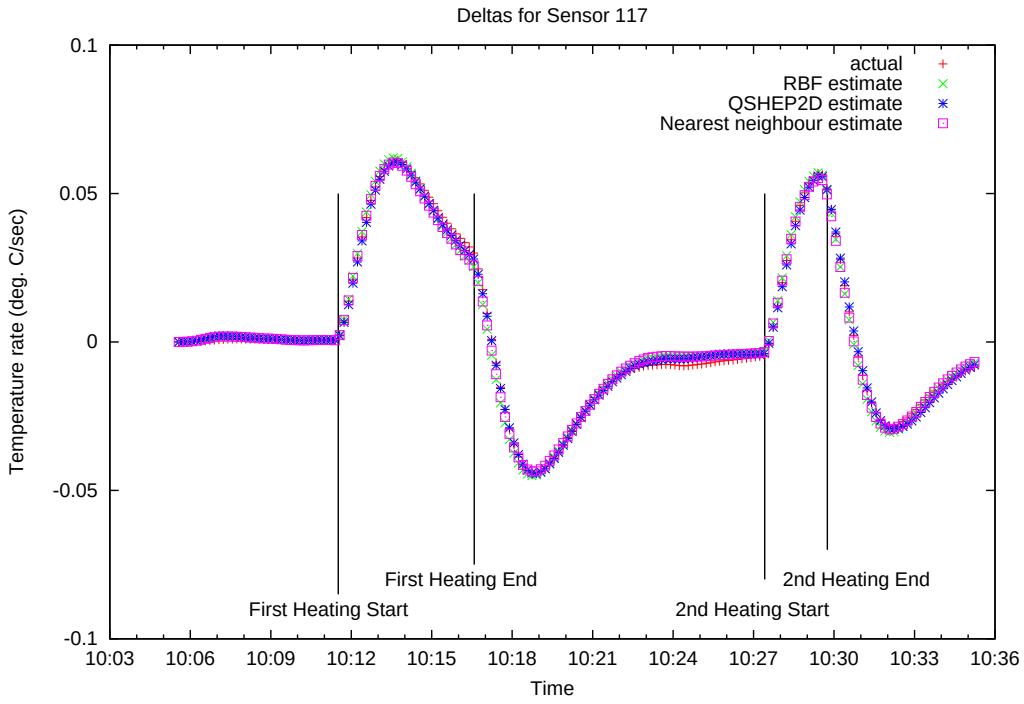


Figure 29: Interpolated delta values against real data (temperature in °C).

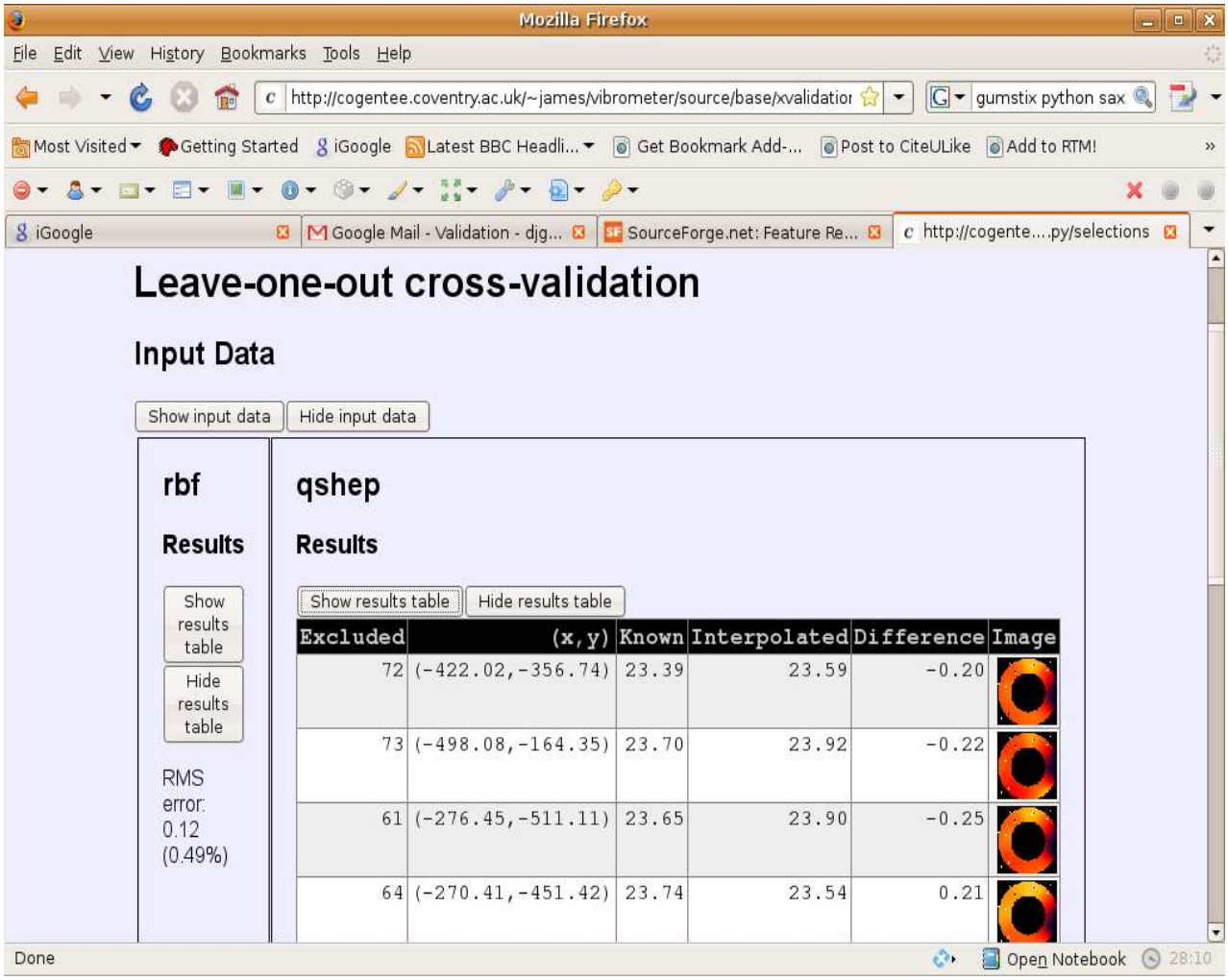


Figure 30: Web based cross validation

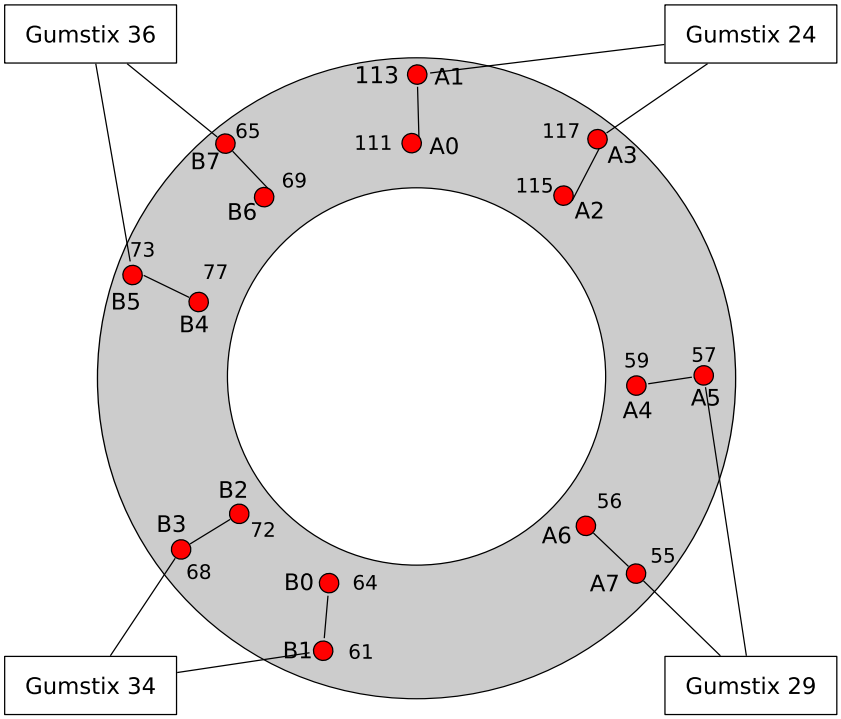
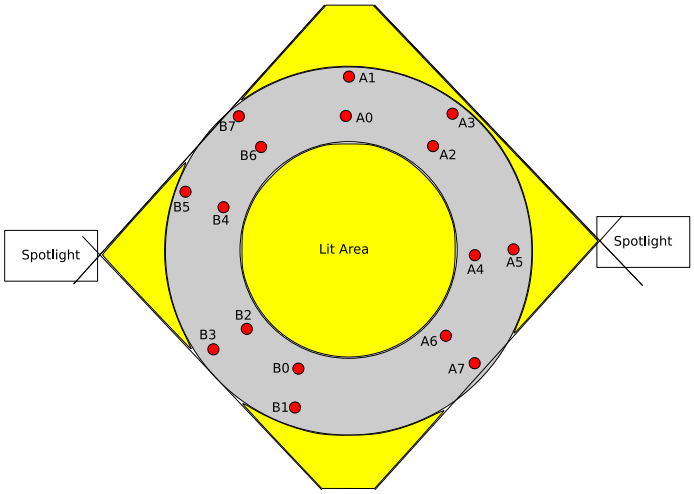


Figure 31: Sensor layout on the jet pipe. Note there are two temperature sensing devices on each I^2C line (sensors 65-69,sensors 73-77, etc.)

(a) heating overview

Figure 32: Lighting arrangement during cross validation



(a) Lamp positions

Figure 33: Sensor Layout and lamps position during cross validation.

Event Id	Description	Time (UTC + 0:00)	Time stamp
279	Experiment Start	11:03:57	1217903637
	Stabilisation		
280	Start 1st heating with lamps	11:08:11	1217930891
281	Stop 1st Heating with lamps	11:16:33	1217931393
	Stabilisation		
282	Start 2nd Heating	11:27:20	1217932040
283	Stop 2nd Heating	11:29:22	1217932162
284	Experiment End	11:35:21	1217932521

Table 15: Experiment Schedule

Each heating and stabilisation cycle lasted for approximately 10 minutes with 2 minutes of heating followed by a 8 minute stabilisation period. The timing associated with this particular test is given in Table 23.

For the experiment duration Table 16 shows the error values between actual and interpolated temperatures for all sensors and all samples. To be particularly noted are the extremely low errors for the delta mode, and also the similar performance of the RBF based and nearest-neighbour based interpolations.

Interpolation method	Mode	MSE
RBF	Absolute sensed	2.06664110461
RBF	Delta	5.74338546652e-05
Nearest Neighbour	Absolute sensed	2.23771360624
Nearest Neighbour	Delta	6.15025572264e-05
QShep2D	Absolute sensed	4.34863757561
QShep2D	Delta	0.000142017859048

Table 16: Cross validation RMS values over whole experiment duration

Screen shots have also been taken at key points during the experiment to illustrate the data displayed on the User Interface, as follows:

End of stabilisation before first heating (11:08:11)

Screen shots and values were taken prior the commencing the first heating cycle. The temperature and "delta" values for all the sensors at that point in time can be found in Table 18

		Interpolated estimate		Error (between measured and interpolated values) (°C)	
Excluded sensor Id	Measured Temperature (°C)	RBF (°C)	QSHEP (°C)	RBF (°C)	QSHEP (°C)
72	23.39	23.4	23.59	-0.01	-0.2
73	23.7	23.8	23.92	-0.1	-0.22
61	23.65	23.73	23.9	-0.08	-0.25
64	23.74	23.64	23.54	0.1	0.2
65	23.7	23.74	23.5	-0.04	0.2
115	23.45	23.72	23.68	-0.27	-0.23
68	23.36	23.41	23.11	-0.05	0.25
69	23.71	23.75	23.95	-0.04	-0.24
117	23.62	23.54	23.4	0.08	0.22
111	24.17	24.05	23.91	0.12	0.26
113	24.23	24.06	24.48	0.17	-0.25
77	23.82	23.7	23.63	0.12	0.19
RBF RMS error over all cross validated sensors:				0.12 (0.49%)	
QSHEP RMS error over all cross validated sensors:					0.23 (0.94%)

Table 18: Temperature and Interpolation values at 11:08:11

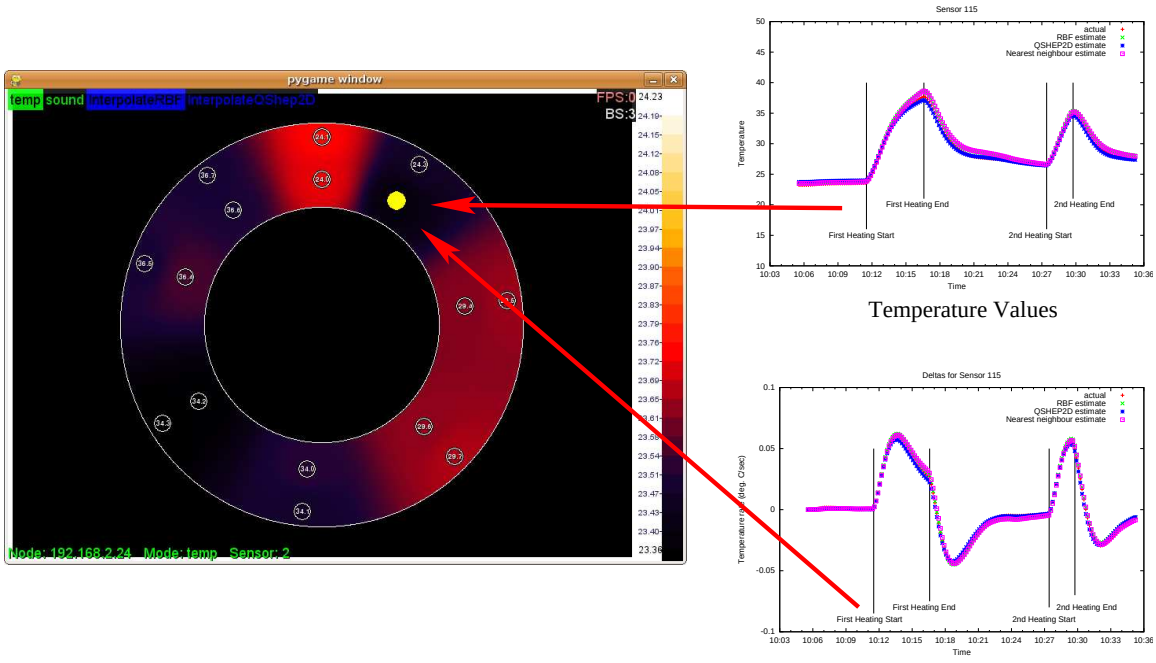


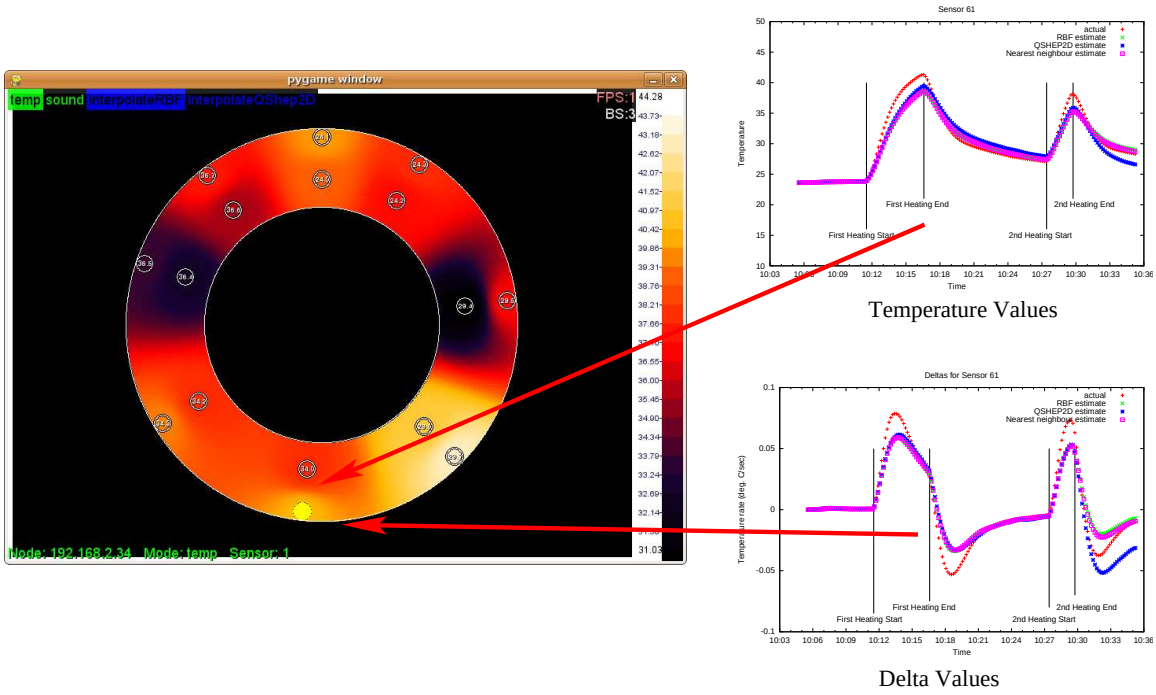
Figure 34: Screenshot and interpolation detail at 11:08:11

End of 1st Heating (11:16:11)

Screen shots were taken at the end of the first period of heating. The temperature and “delta” values for all the sensors at that point in time can be found in Table 19.

		Interpolated estimate		Error (between measured and interpolated values) (°C)	
Excluded sensor Id	Measured Temperature (°C)	RBF (°C)	QSHEP (°C)	RBF (°C)	QSHEP (°C)
72	38.66	40.02	37.29	1.37	-1.36
73	34.18	33.72	31.86	2.32	0.46
61	41.34	38.61	42.01	-0.67	2.73
64	38.62	41.15	38.32	0.3	-2.53
65	38.17	36.23	37.22	0.95	1.94
115	37.71	38.86	38.73	-1.02	-1.15
68	40.43	38.53	42.08	-1.65	1.9
69	36.13	37.74	36.54	-0.41	-1.61
117	38.64	38	37.8	0.84	0.64
111	39.96	39.92	38.3	1.66	0.04
113	40.86	39.51	43.17	-2.31	1.35
77	33.01	34.92	34.39	-1.38	-1.91
RBF RMS error over all cross validated sensors:				1.66 (4.07%)	
QSHEP RMS error over all cross validated sensors:					1.39 (3.44%)

Table 19: Temperature and Interpolation values at 11:16:33



Temperature Values

Delta Values

Figure 35: Screen shot and interpolation values at 11:16

Prior to commencing 2nd heating (11:27:20)

Screen shots were taken approximately half way into the stabilisation period between the two heating cycles. The temperature and “delta” values for all the sensors at that point in time can be found in Table 20.

		Interpolated estimate		Error (between measured and interpolated values) (°C)	
Excluded sensor Id	Measured Temperature (°C)	RBF (°C)	QSHEP (°C)	RBF (°C)	QSHEP (°C)
72	26.8	26.91	26.98	-0.18	-0.11
73	26.77	26.71	26.8	-0.03	0.06
61	27.08	27.36	27.79	-0.71	-0.28
64	27.38	27.06	26.87	0.51	0.32
65	26.94	26.74	26.94	0	0.2
115	26.44	26.61	26.63	-0.19	-0.17
68	26.91	26.81	26.56	0.35	0.1
69	26.76	26.86	26.68	0.08	-0.1
117	26.63	26.46	26.35	0.28	0.17
111	26.47	26.56	26.45	0.02	-0.09
113	26.53	26.51	26.67	-0.14	0.02
77	26.69	26.78	26.72	-0.03	-0.09
RBF RMS error over all cross validated sensors:				0.17 (0.60%)	
QSHEP RMS error over all cross validated sensors:					0.30 (1.03%)

Table 20: Values at 11:27:20

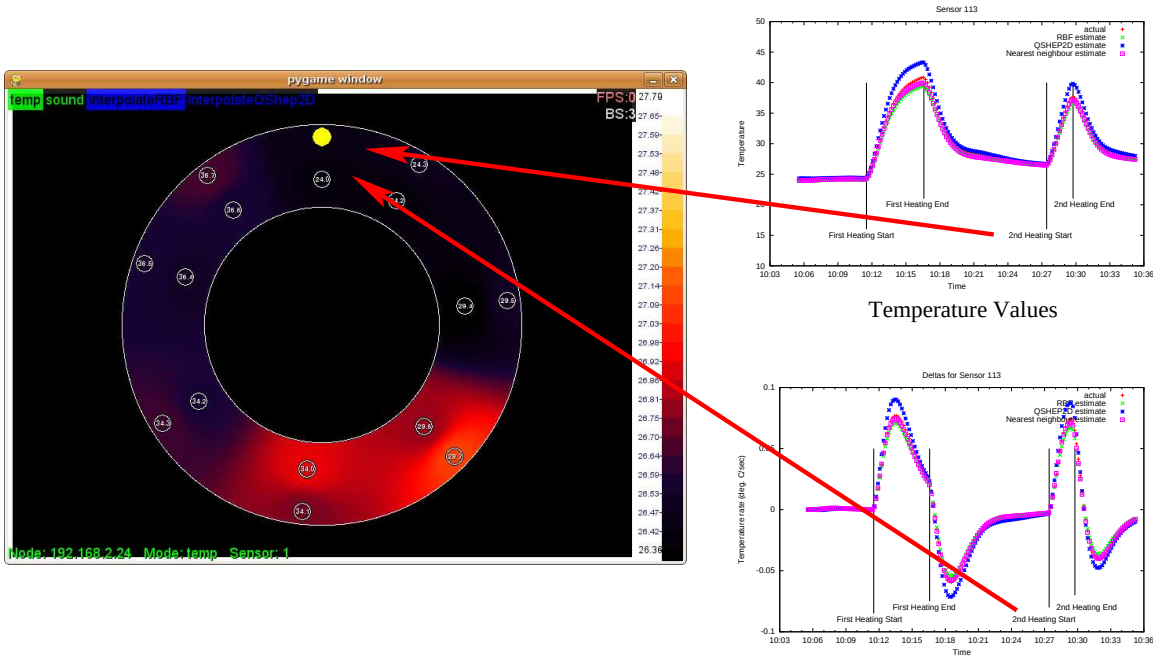


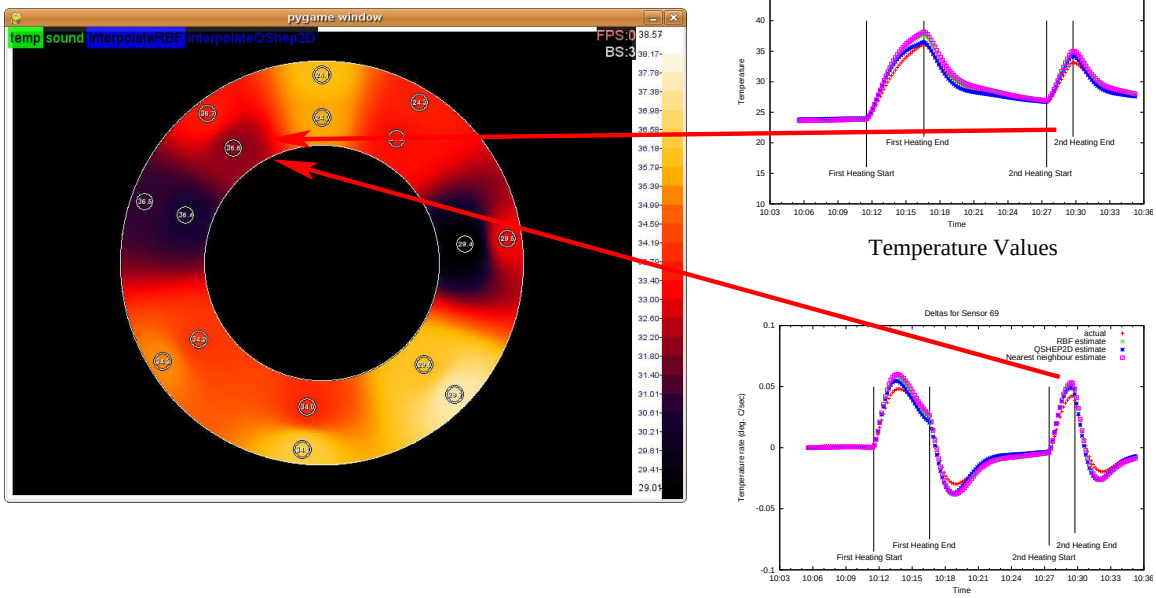
Figure 36: Screenshot and interpolation values at 11:27

End of 2nd heating (11:29:22)

Screen shots were taken at the end of the second heating period. The temperature and “delta” values for all the sensors at that point in time can be found in Table 22.

		Interpolated estimate		Error (between measured and interpolated values) (°C)	
Excluded sensor Id	Measured Temperature (°C)	RBF (°C)	QSHEP (°C)	RBF (°C)	QSHEP (°C)
72	34.78	35.7	33.74	1.04	-0.92
73	31.23	31.31	30.55	0.68	-0.08
61	37.28	34.52	36.77	0.51	2.76
64	34.52	37.11	35	-0.48	-2.59
65	34.21	32.63	32.31	1.9	1.58
115	33.97	34.65	34.79	-0.82	-0.68
68	35.99	34.68	37.46	-1.47	1.31
69	32.45	34.01	33.6	-1.15	-1.56
117	34.32	34.24	33.82	0.5	0.08
111	36.35	35.87	34.61	1.74	0.48
113	36.72	35.85	38.84	-2.12	0.87
77	30.86	31.78	31.17	-0.31	-0.92
RBF RMS error over all cross validated sensors:				1.42 (3.86%)	
QSHEP RMS error over all cross validated sensors:					1.21 (3.32%)

Table 21: Values at 11:29:22



Temperature Values

Delta Values

Table 22: Values at 11:29:22

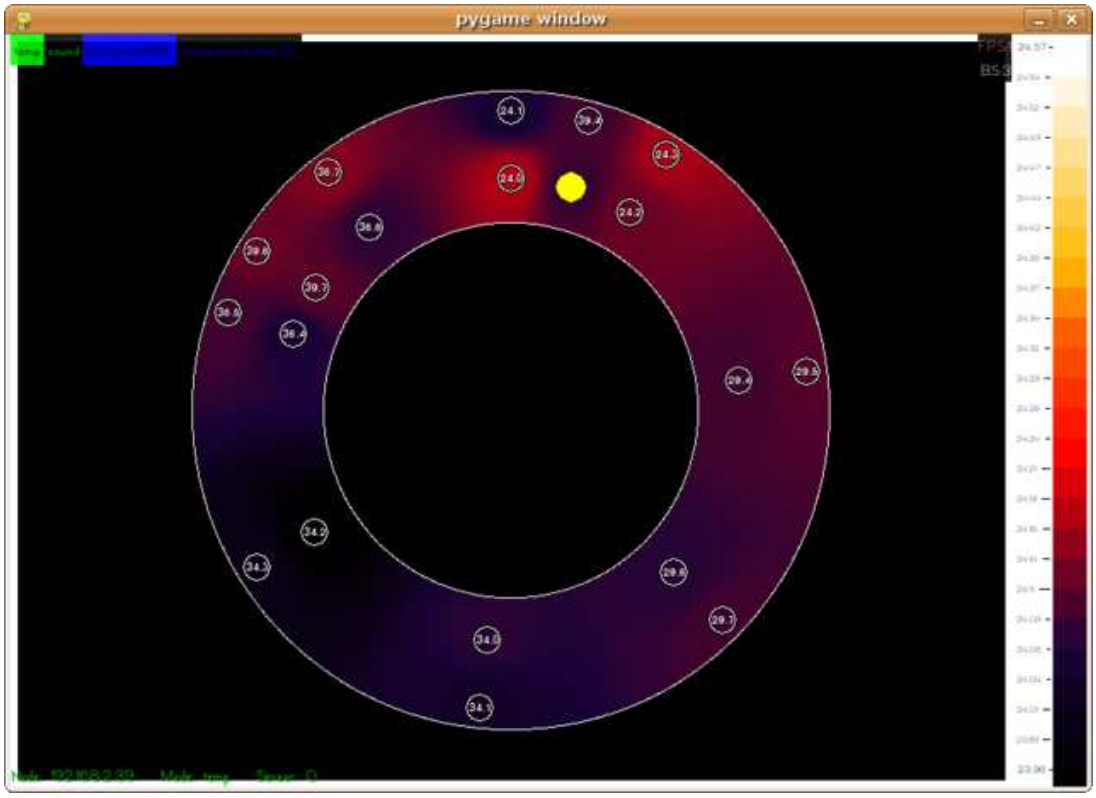


Figure 37: User Interface with extra probe sensors.

Probe based validation

The “leave-one-out” cross-validation procedure explained above, however, does not provide a clear measure of the difference *between* real and predicted values between the sensed data points should all the sensors in the system be functional. In order to assess the quality of the interpolation between points, a fifth node was added to the network, geared with 4 temperature sensors, and used as a “probe” node. This node and its sensors have no fixed location. Instead, the sensors are used to take measurements at arbitrary points in various test runs. Combined with the cross-validation tool, the probe node provides a method to determine the accuracy of predicted (interpolated) temperature values between sensing points. The experiment reported here consisted yet again of two heating and stabilisation cycles performed as per Section 4.3.10. The same layout as shown in Figure 31 was used. Figure 37 shows the user interface updated with the positions of the probe sensors.

Each heating and stabilisation cycle lasted for approximately 10 minutes with 2 minutes of heating followed by a 8 minute stabilisation period. The timing associated with this particular test is given in Table 24.

For the experiment duration Table 25 shows the error values between actual and interpolated temperatures for all sensors and all samples.

Whilst the examples above have focused on the interpolated predictions for all sensors, the tables below

Event Id	Description	Time (UTC + 0:00)	Time stamp
279	Experiment Start	11:03:57	1217903637
	Stabilisation		
280	Start 1st heating with lamps	11:08:11	1217930891
281	Stop 1st Heating with lamps	11:16:33	1217931393
	Stabilisation		
282	Start 2nd Heating	11:27:20	1217932040
283	Stop 2nd Heating	11:29:22	1217932162
284	Experiment End	11:35:21	1217932521

Table 23: Experiment Schedule

Event Id	Description	Time (UTC + 0:00)	Time stamp
279	Experiment Start	15:36	1221233932
	Stabilisation		
280	Start 1st heating with lamps	15:44	1221234248
281	Stop 1st Heating with lamps	15:46	1221234368
	Stabilisation		
282	Start 2nd Heating	15:54	1221234852
283	Stop 2nd Heating	15:56	1221234968
284	Experiment End	16:05	1221235739

Table 24: Experiment Schedule

focus on the RBF interpolated values for the four probe sensors. Screen shots have also been taken at key points during the experiment to illustrate the data displayed on the User Interface, as follows:

Interpolation method	Mode	MSE
RBF	Absolute sensed	0.570255985441
RBF	Delta	2.73757913681e-05
Nearest Neighbour	Absolute sensed	0.82868825714
Nearest Neighbour	Delta	4.00779552564e-05
QShep2D	Absolute sensed	0.996146376516
QShep2D	Delta	5.18516559429e-05

Table 25: Cross validation RMS values over whole experiment duration

End of stabilisation before first heating (15:44)

Screen shots and values were taken prior the commencing the first heating cycle. The temperature and "delta" values for the probe sensors at that point in time can be found in Table 38.

		Interpolated estimate		Error (between measured and interpolated values) (°C)	
Excluded sensor Id	Measured Temperature (°C)	RBF (°C)	QSHEP (°C)	RBF (°C)	QSHEP (°C)
56	24.32	24.33	24.5	-0.01	-0.18
57	24.45	24.46	24.41	-0.01	0.04
59	24.46	24.46	24.39	0	0.07
61	24.2	24.35	24.68	-0.15	-0.48
64	24.35	24.22	24.32	0.13	0.03
65	24.53	24.71	24.59	-0.18	-0.06
68	24.11	24.08	22.87	0.03	1.24
69	24.36	24.99	25.64	-0.63	-1.28
72	24.02	24.18	24.28	-0.16	-0.26
73	24.37	25.02	24.5	-0.65	-0.13
77	24.38	25.39	25.64	-1.01	-1.26
111	24.74	24.64	24.62	0.1	0.12
113	24.36	24.77	24.73	-0.41	-0.37
115	24.49	24.73	24.8	-0.24	-0.31
117	24.65	24.64	24.72	0.01	-0.07
125	24.79	24.67	24.69	0.12	0.1
127	25.96	24.6	24.58	1.36	1.38
129	25.4	24.93	25.34	0.47	0.06
130	24.9	24.63	24.57	0.27	0.33
RBF MSE error over all cross validated sensors:				0.22	
QSHEP MSE error over all cross validated sensors:					0.37
RBF MSE error over probe sensors:				0.54	
QSHEP MSE error over probe sensors:					0.51

Table 26: Temperature and Interpolation values at 15:46

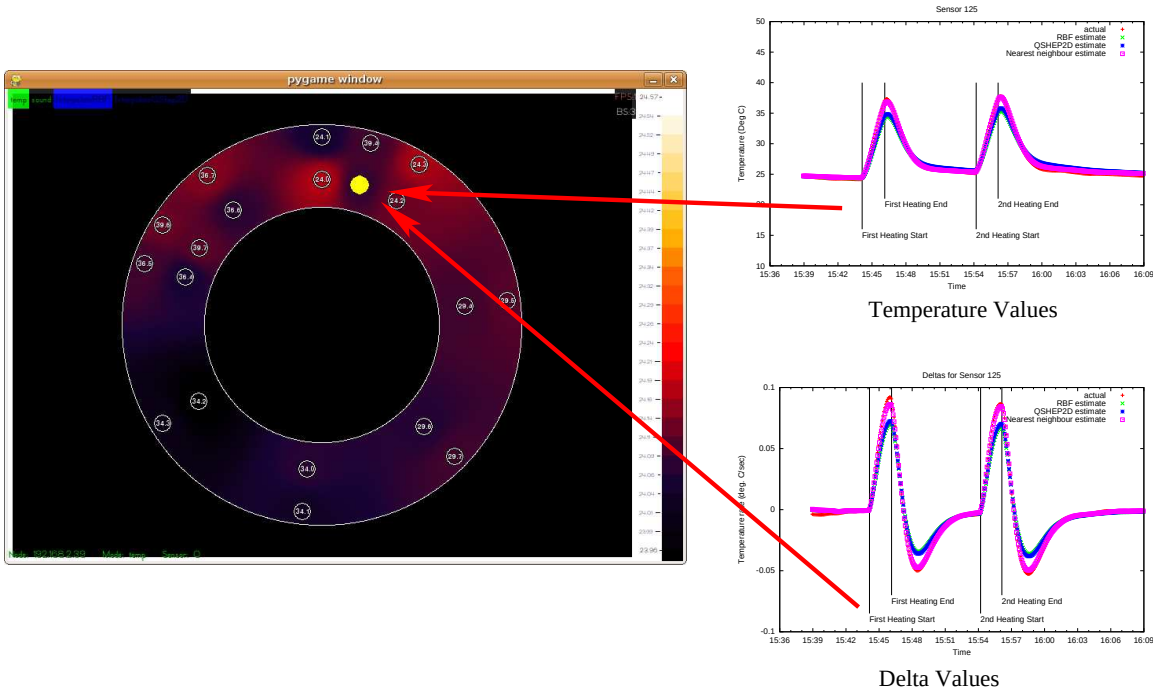


Figure 38: Screenshot and interpolation detail at 15:44

End of first heating period (15:46)

Screen shots and values were taken at the end of the first heating cycle. The temperature and “delta” values for the probe sensors at that point in time can be found in Table 27.

		Interpolated estimate		Error (between measured and interpolated values) (°C)	
Excluded sensor Id	Measured Temperature (°C)	RBF (°C)	QSHEP (°C)	RBF (°C)	QSHEP (°C)
55	32.96	32.87	35.95	0.09	-2.99
56	32.85	32.95	31.12	-0.1	1.73
57	32.52	31.26	28.4	1.26	4.12
59	31.03	32.66	32.26	-1.63	-1.23
61	34.84	32.45	32.59	2.39	2.25
64	32.33	34.72	33.86	-2.39	-1.53
65	33.77	31.56	29.88	2.21	3.89
68	36.63	34.25	37.47	2.38	-0.84
69	29.83	33.9	34.1	-4.07	-4.27
72	34.31	36.26	33.93	-1.95	0.38
73	34.57	34.07	36.39	0.5	-1.82
111	36.64	35.07	36.08	1.57	0.56
113	33.78	35.37	35.33	-1.59	-1.55
115	32.4	34.98	36.54	-2.58	-4.14
117	32.99	33.57	29.41	-0.58	3.58
125	37.12	34.2	34.59	2.92	2.53
127	33.37	33.29	30.47	0.08	2.9
129	34.55	33.67	34.16	0.88	0.39
130	33.59	34.97	35	-1.38	-1.41
RBF MSE error over all cross validated sensors:				3.69	
QSHEP MSE error over all cross validated sensors:					6.58
RBF MSE error over probe sensors:				2.8	
QSHEP MSE error over probe sensors:					4.24

Table 27: Temperature and Interpolation values at 15:46

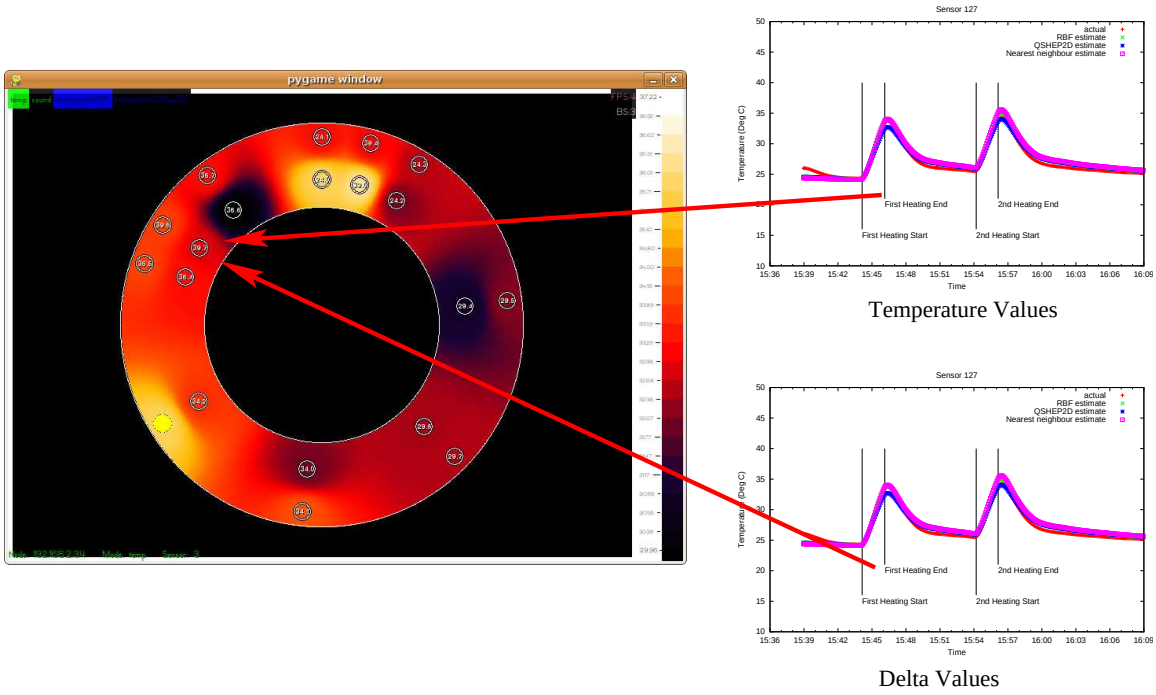


Figure 39: Screenshot and interpolation detail at 15:46

Start of 2nd heating period (15:54)

Screen shots and values were taken at the start of the second heating cycle. The temperature and “delta” values for the probe sensors at that point in time can be found in Table 28.

		Interpolated estimate		Error (between measured and interpolated values) (°C)	
Excluded sensor Id	Measured Temperature (°C)	RBF (°C)	QSHEP (°C)	RBF (°C)	QSHEP (°C)
56	25.98	25.98	25.88	0	0.1
57	26.02	25.93	26.05	0.09	-0.03
59	25.95	25.99	26.04	-0.04	-0.09
61	26.06	26.15	25.92	-0.09	0.14
64	26.17	26.05	26.36	0.12	-0.19
65	26.13	25.77	25.95	0.36	0.18
68	26.12	25.95	25.6	0.17	0.52
69	25.89	25.84	26.12	0.05	-0.23
72	25.95	26.1	25.57	-0.15	0.38
73	26.13	25.61	26.03	0.52	0.1
111	25.49	25.29	25.19	0.2	0.3
113	24.88	25.61	25.16	-0.73	-0.28
115	25.8	25.54	25.91	0.26	-0.11
117	25.76	25.71	25.46	0.05	0.3
125	25.29	25.64	26.47	-0.35	-1.18
127	25.55	25.81	25.7	-0.26	-0.15
129	25.58	25.87	25.94	-0.29	-0.36
130	25.85	25.38	26.16	0.47	-0.31
130	25.85	25.38	25.06	0.47	0.79
RBF MSE error over all cross validated sensors:				0.09	
QSHEP MSE error over all cross validated sensors:					0.24
RBF MSE error over probe sensors:				0.15	
QSHEP MSE error over probe sensors:					0.22

Table 28: Temperature and Interpolation values at 15:54

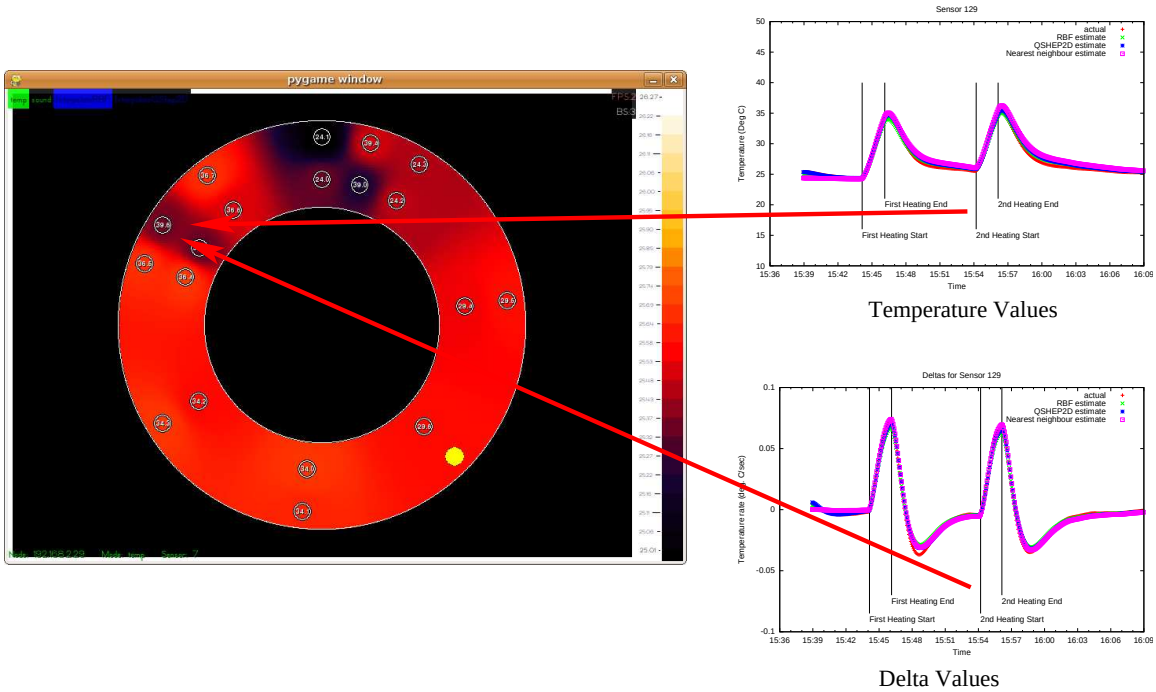


Figure 40: Screenshot and interpolation detail at 15:54

End of 2nd heating period (15:56)

Screen shots and values were taken at the end of the second heating cycle. Temperature and “delta” values for the probe sensors at that point in time can be found in Table 29.

		Interpolated estimate		Error (between measured and interpolated values) (°C)	
Excluded sensor Id	Measured Temperature (°C)	RBF (°C)	QSHEP (°C)	RBF (°C)	QSHEP (°C)
56	33.94	34.04	36.89	-0.1	-2.95
57	33.61	32.39	32.26	1.22	1.35
59	32.18	33.72	29.74	-1.54	2.44
61	36.01	33.63	33.3	2.38	2.71
64	33.52	35.88	33.82	-2.36	-0.3
65	34.87	32.52	34.93	2.35	-0.06
68	37.86	35.38	30.68	2.48	7.18
69	31.07	34.73	38.97	-3.66	-7.9
72	35.45	37.46	34.72	-2.01	0.73
73	35.71	34.55	34.99	1.16	0.72
111	37.3	35.51	36.18	1.79	1.12
113	33.99	36.1	36.38	-2.11	-2.39
115	33.41	35.63	36.36	-2.22	-2.95
117	33.9	34.43	37.04	-0.53	-3.14
125	37.45	35.02	30.98	2.43	6.47
127	33.91	34.09	35.47	-0.18	-1.56
129	34.89	34.57	31.69	0.32	3.2
130	34.53	35.51	35.26	-0.98	-0.73
130	34.53	35.51	35.25	-0.98	-0.72
RBF MSE error over all cross validated sensors:				3.45	
QSHEP MSE error over all cross validated sensors:					5.48
RBF MSE error over probe sensors:				0.51	
QSHEP MSE error over probe sensors:					3.43

Table 29: Temperature and Interpolation values at 15:56

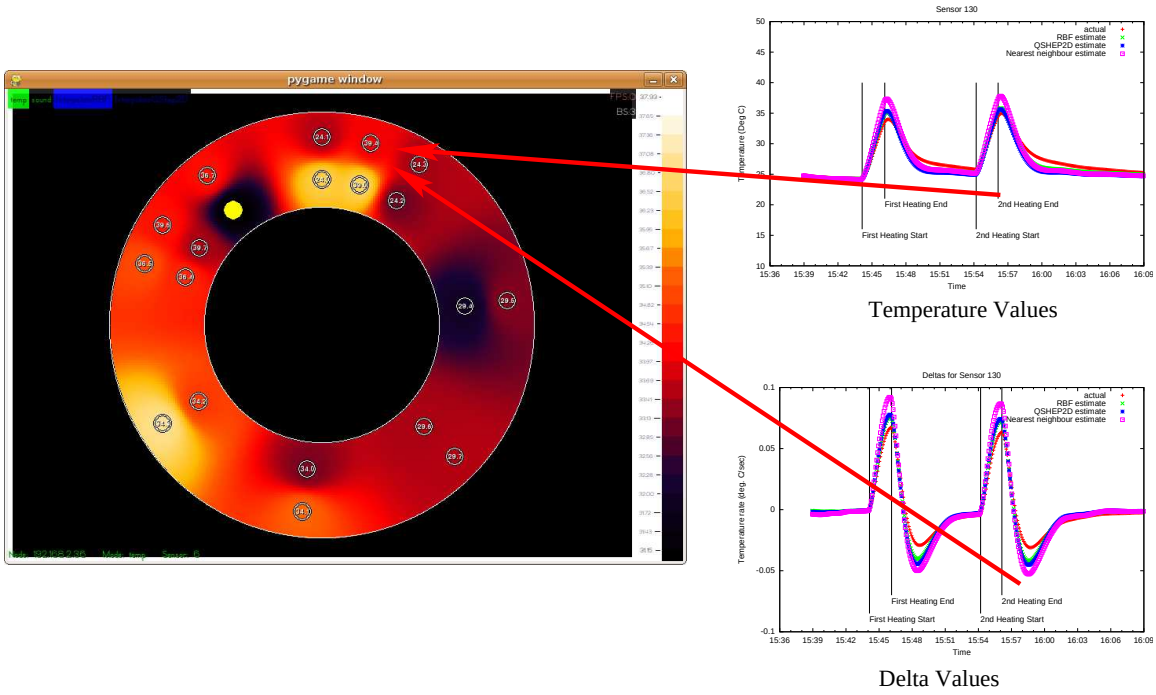


Figure 41: Screenshot and interpolation detail at 15:56

Conclusions

The visualiser has proven to be robust, able to display data for both sound and temperature, using several different interpolation functions. Extending the user interface to display “delta” information enabled a greater understanding of the state of the system being monitored, by displaying rate of change in temperature rather than absolute values.

Wit respect to the interpolation accuracy, the experimental results produced encouraging results, with the RBF interpolation producing lower levels of error for the chosen experimental environment, than either Nearest Neighbour or the Quadratic Shepard methods. The experimental results obtained using the “probe” node are especially encouraging, with the extra sensing points greatly increasing the accuracy of the estimated field values. However, a close examination of the estimated field values at specific points in time highlights some issues with all the interpolation functions implemented. When subjected to a high level of stimulation, as experienced during the heating cycles, all interpolation methods produce higher levels of error than in unstimulated conditions. Given the non uniform heating of the sensors, the differences in temperature between the sensors are exaggerated by the interpolation function. Interpolation also relies on a flat field of stimulation for accuracy. The non-uniform field represents a noisy environment, which the interpolation methods did not cope well with (therefore, there are high error levels at the end of each heating cycle).

The interpolation function used is a placeholder for future work. The airflow within a jet pipe is complex, and Computational Fluid Dynamics (CFD) coupled with industry experience is required to accurately model the flow.

5 Concluding Remarks

A fully integrated end-to-end system prototype has been designed, implemented and evaluated, which builds upon and advances research in the area of WSNs for monitoring applications. The work on this project built was aimed at:

- rapid production of a fully functional prototype instrument for monitoring and visualising field temperature profiles in jet engines
- through the experience gained, enable a generic design framework to be produced for a larger class of WSN based monitoring applications.

The system proposed has a high innovative value, potentially allowing detailed in-flight monitoring of temperatures within a gas turbine engine, with extension to a wide range of potential aircraft monitoring applications. A key potential benefit of the instrument is weight reduction through replacing cabling with wireless transmission. Additional benefits are in the area of providing better exploitation of available sensor data through computer visualisation of the data and autonomous identification of sensor faults. The work described here is building on existing test-bed demonstrators and visualisation systems at the Cogent Computing Applied Research Centre. The application domain specialism is covered by Vibro-Meter UK through Dr. Mark Langley whilst the electronics for harsh environments issues are brought forth with proposed solutions by TRW Conekt through Roger Hazelden.

References

[1] Gumstix motherboard io [online] http:/docwiki.gumstix.org/, 11th September 2008. accessed 11th September 2008.

[2] MaxStream Inc. *XBee Series 2 OEM RF Modules*. Lindon, UT, 2007.

[3] infineon. http://www.infineon.com/cms/en/product/channel.html?channel=ff80808112ab68 accessed 16/06/2008.

[4] B.G. Lipták. *Instrument Engineers' Handbook*. CRC Press, 2002.

[5] Peter S. Maybeck. *Stochastic models, estimation, and control*, volume 141 of *Mathematics in Science and Engineering*. 1979.

[6] Phillips Electronics N.V. *UCB1400 Audio codec with touch screen controller and power management monitor*. Eindhoven, 2002.

[7] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Commun. ACM*, 43(5):51–58, 2000.

[8] Maxim Intergrated Products. *SOT23 Dual-input USB/AC Adapter 1-cell Li+ Battery Charger*. Sunnyvale, CA, 2003.

[9] Maxim Intergrated Products. *Low-drop out, 300mA Linear Regulators in SOT23*. Sunnyvale, CA, 2006.

[10] Robert J. Renka. Algorithm 660: Qshep2d: Quadratic shepard method for bivariate interpolation of scattered data. *ACM Trans. Math. Softw.*, 14(2):149–150, 1988.

[11] Robert J. Renka. Multivariate interpolation of large sets of scattered data. *ACM Trans. Math. Softw.*, 14(2):139–148, 1988.

[12] J. K. Shuttleworth, E. I. Gaura, and R. M. Newman. Surface reconstruction: Hardware requirements of a som implementation. In *Proceedings of the ACM Workshop on Real-World Wireless Sensor Networks (REALWSN'06)*, pages 95–96, June 2006. ACM ISBN: 1-59593-431-6.

[13] Daniela Tulone. A resource–efficient time estimation for wireless sensor networks. In *DIALM-POMC '04: Proceedings of the 2004 joint workshop on Foundations of mobile computing*, pages 52–59, New York, NY, USA, 2004. ACM.

[14] R. Tynan, G.M.P. OHare, D. Marsh, and D. OKane. Interpolation for wireless sensor network power management. In *Proceedings of the International Workshop on Wireless and Sensor Networks (WSNET-05)*. IEEE Press, June 2005.

[15] Greg Welch and Gary Bishop. An introduction to the Kalman filter. Technical report, University of North Carolina at Chapel Hill, 1995.

[16] Yizhou Yu. Surface reconstruction from unorganized points using self-organizing neural networks. In *Proceedings of IEEE Visualization*, pages 61–64, 1999.

Cogent Computing Applied Research Centre
Coventry University
Priory Street, Coventry CV1 5FB
www.cogentcomputing.org