Design of a virtual sensor using machine learning imputation techniques in a wireless sensor network.

Final Report

M. Matusowsky
11093804

Submitted as partial fulfilment of the requirements of Project EPR402
in the Department of Electrical, Electronic and Computer Engineering
University of Pretoria

November 2018

Study leader: Mr D. Ramotsoela

Copy 1 of 4

Part 1. Preamble

This report describes the work that I did in designing a virtual sensor for a wireless sensor network using a machine learning technique.

Project proposal and technical documentation
This main report contains a copy of the approved Project Proposal (Part 3 of the report) and technical documentation (Part 5 of the report). The latter provides details of the schematics, stripboard layout, and software code. I have included this appendix on a CD or DVD that accompanies this report.

Project history
This project does not build on any projects that were completed in previous years. Stripboard design software as well as the circuit error checking software was provided by Fritzing.org, an opensource initiative. The stripboard sensor circuit was designed and soldered by me. One of the equations I used was adapted from Steinhart [1]. I used the Tkinter python library to implement the graphical user interface as well as the NumPy python library for statistical analysis in Python. All other work in this report was my own and completed from first principles.

Language editing
This document has been language edited by a knowledgeable person. By submitting this document in its present form, I declare that this is the written material that I wish to be examined on.

My language editor was Mr Joseph Henry Mervitz

_____ _____
Language editor signature *Date*

Declaration

I, Michael Matusowsky understand what plagiarism is and have carefully studied the plagiarism policy of the University. I hereby declare that all the work described in this report is my own, except where explicitly indicated otherwise. Although I may have discussed the design and investigation with my study leader, fellow students or consulted various books, articles or the internet, the design/investigative work is my own. I have mastered the design and I have made all the required calculations in my lab book (and/or they are reflected in this report) to authenticate this. I am not presenting a complete solution of someone else.

Wherever I have used information from other sources, I have given credit by proper and complete referencing of the source material so that it can be clearly discerned what is my own work and what was quoted from other sources. I acknowledge that failure to comply with the instructions regarding referencing will regarded as plagiarism. If there is any doubt about the authenticity of my work, I am willing to attend an oral ancillary examination/evaluation about the work.

I certify that the Project Proposal appearing as the Introduction section of the report is a verbatim copy of the approved Project Proposal.

_____ _____

M. Matusowsky Date

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

TDD	Test-driven development
PIC32	Peripheral interface controller 32-bit
MCU	Microcontroller Unit
TCP/IP	Transmission control protocol/Internet protocol
PC	Personal computer
USART	Universal Synchronous Asynchronous Receiver Transmitter
ADC	Analog-to-digital converter
IoT	Internet of Things
WSN	Wireless Sensor Network
USB	Universal Serial Bus
KNN	K-Nearest Neighbour
MLP	Multiple Layer Perceptron
FFNN	Feed-forward neural network
SOM	Self-Organising Map
VS	Virtual Sensor
GA	Genetic Algorithm
kB	kilo Byte
PnP	Plug-and-Play

Part 2. Summary

This report describes work carried out on the design of a virtual sensor using machine learning in a wireless sensor network with the objective of using the virtual sensor to impute sensor values of a single sensor node, if a sensor node in the network fails, using the other sensor nodes that are still actively transmitting.

What has been done

A literature survey was completed on modern wireless communication systems as well as different machine learning implementations. The hardware and software for the sensor nodes and server was then designed and implemented from first principles. communication system was then designed from first principles. At the core of the hardware system is three PIC32 (Peripheral interface controller 32-bit) microcontroller units (MCU) with an ESP8266 WiFi module that includes a transmission control protocol/internet protocol (TCP/IP) framework, and all additional hardware was designed and implemented. A Python program was developed to simulate the system, as well as C code for the PIC32 processor. The system was implemented and tested throughout the development phase using a test-driven development (TDD) process. Four WiFi modules were used in a star network topology whereby one WiFi module acted as the TCP/IP server on the desktop personal computer (PC) and the other three modules acted as TCP/IP clients on the sensor nodes. The neural network for each sensor node was implemented on the server as well as on the corresponding sensor node's MCU. The main result from one of the sensor nodes is displayed in figure 1.

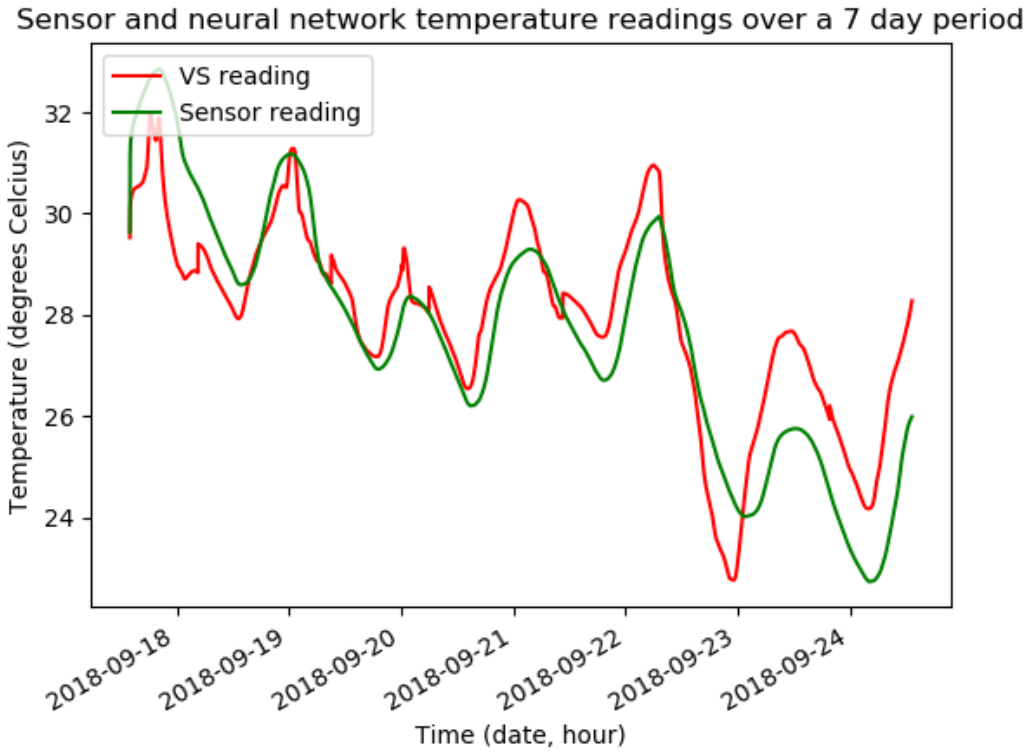

Figure 1.
Neural network and sensor readings over a 7-day period.

What has been achieved

Successful imputation using machine learning was achieved in a wireless sensor network that was deployed in multiple locations. The accuracy of the imputation depended highly on the amount of training data available where it was discovered that a minimum of 5 days of data was required before a sensor node would be adequately trained.

Findings

It was found, while analysing the weights of the neural networks, that the time input in the neural network was much more important than the sensor inputs with regards to imputing temperature data. Another important discovery was that due to glitches within the hardware, the sensed value by the thermistor would sometimes return a ground value and thus a filtering algorithm had to be implemented to account for the glitches as displayed in figure 2.

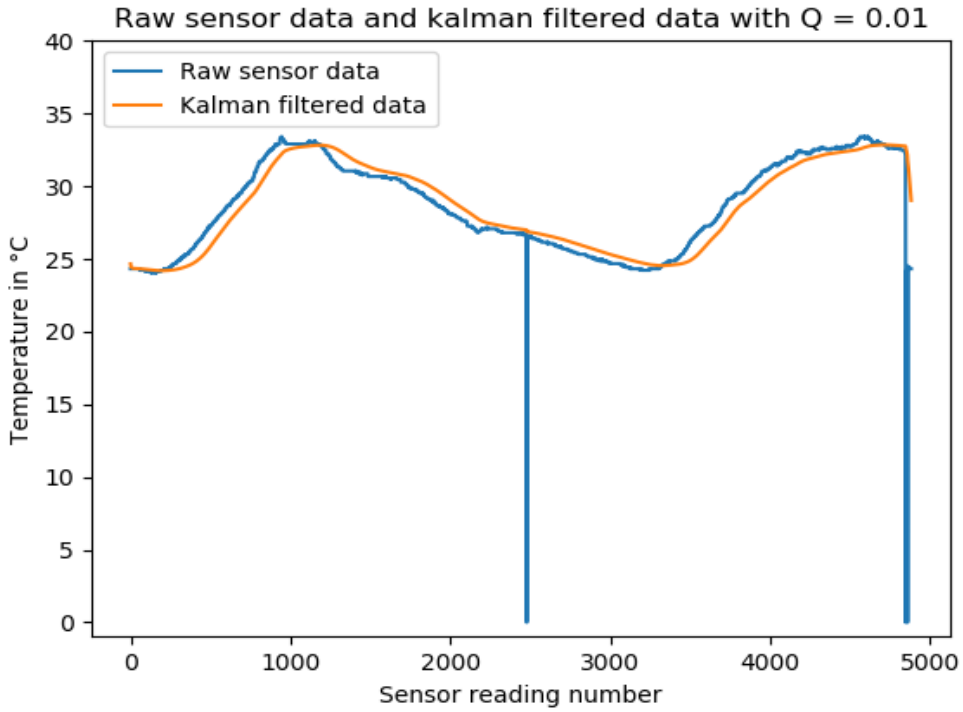

Figure 2.
Scalar Kalman filter with Q = 0.01

Contribution

New software that had to be mastered to complete this project was Fritzing. Fritzing is an opensource PCB/stripboard/circuit designing software that undergraduate students would not usually be aware of and has not been covered in any undergraduate module. The Fritzing software was used specifically for the stripboard module to plan, test and implement the circuitry on a stripboard. Further software that needed to be learned was the MPLAB Harmony Framework for PIC32 MCU's which was extremely useful in configuring the right clock speed as well as the Universal Synchronous/Asynchronous Receiver/Transmitter (USART) and analog-to-digital converter (ADC) modules.

Due to the limited amount of program memory in the PIC32 and the requirement for string handling, some functions usually provided by the standard library for C had to be re-written from first principles due to the overhead in program memory introduced

when using functions such as sprint() which would allow data to be converted from integers and floats to strings and then written to the USART buffer.

A server was developed using Python on a PC connected by USB with a WiFi module. The server would act as the communication center to periodically request and transmit readings from the sensor nodes as well as determine if a sensor node is offline. The code for the serial communication was implemented by the student with the use of the pyserial library.

Code for the neural network, including the training algorithms, was developed by the student without reliance on any existing libraries in python. A framework for training neural networks with a minimum of 3 inputs, theoretically unlimited hidden nodes and layers, and a maximum of 1 output was developed from first principles. A friend in a postgraduate course helped provide some clarity on the chosen training method that is used as it was complex and completely new to the student since no undergraduate module covers evolutionary models for training in neural networks. The study leader provided no assistance with regards to the training but did provide some advice regarding the output layer of the neural network as the student originally wanted to have the output be a binarized output layer with multiple output nodes. The study leader advised to use a single output node in the output layer.

There was a strong reliance on an existing python library called csv to deal with reading and writing from the local database.

Part 3. Project identification: approved Project Proposal

1. Problem Statement

In this section the problem that is being addressed by the project proposal is described in terms of the motivation, context, technical challenges and limitations of the proposed project.

Motivation. Sensors are devices used in everyday objects such as mobile phones, touch-sensitive elevators and self-driving cars. There are innumerable amounts of applications where sensors are used to gather data about the environment and then used in other electronics to monitor or react to the conditions being sensed such as the lighting in a room being adjusted based on the required illumination level. A problem arises in wireless sensor networks (WSN) where, due to the nature of the wireless communication, data can be lost or corrupted during the transmission phase due to external factors such as solar radiation corrupting data from satellites or congestion in a network causing packet loss when transmitting data to other devices. Furthermore, the cost of implementing or needing to replace many physical sensors in a network can become prohibitively expensive.

This project will look at designing and implementing a wireless sensor network that will make use of data imputation methods and machine learning to realise virtual sensors that can completely replace physical sensor nodes and give accurate substituted data in place of nodes with failed sensor modules.

Context. The integrity of received information is an important issue in the modern age. Sensors play a pivotal role in electronic devices of all shapes, sizes and function and especially more so in wireless sensor networks where data loss is an expected occurrence [1]. Data imputation and virtual sensors are tools that allow a system to counter-act the effect of this data loss by accurately substituting values that real sensors would most likely return [2]. This then allows the system to still make use of incomplete data rather than completely discarding affected data entries.

The main function of this project will be to ensure the robustness of a wireless sensor network by making sure that damaged nodes in a wireless sensor network can be replaced by virtual sensors using imputation techniques that make use of machine learning algorithms which will allow the system to remain robust in terms of the provided sensor data even when a node malfunctions or is removed from the network thus allowing the system to continue running in real-time while gathering data that closely resembles the affected sensor nodes would-be data.

K-Nearest Neighbours (KNN, lazy learning), multi-layered perceptrons (MLP, supervised learning) and self-organizing maps (SOM, unsupervised learning) are three popular machine learning methods that have been used to great effect in data sets that do not deal with time-series analysis [3,4,5,6] where KNN, MLP and SOM outperform traditional imputations techniques , such as hot-swapping, to significant degrees on multiple data sets including but not limited to a breast cancer detection data set, a seed classifying data set and sonar imaging data set.

Technical Challenges. The technical challenges for this project are: (i) Designing and implementing an imputation technique using machine learning algorithms on individual sensor nodes that can act as a virtual sensor. (ii) The algorithm should be robust to ensure the integrity of the data that is being substituted by the virtual sensors. (iii) The algorithm should be efficient enough so that congestion does not occur due to computations taking place which in itself would then cause loss of more data.

Limitations. The availability of bandwidth in the network will limit how many readings can be transferred and received per time interval between all nodes and the server. The second limitation is the processing speed and power of the processing unit at each sensor node. Another limitation is the cost of developing the product thus cost-effective hardware is a necessity as well as putting a financial limit on the number of nodes that can be physically implemented for this project.

2. Project requirements

The main aim of the project will be to create a wireless sensor network that makes use of the machine learning imputation techniques to ensure the integrity and robustness of the data that is transmitted and received over the network.

2.1 Mission requirements of the product

The system will need to fulfil the following requirements.

- The system must use machine learning algorithms to implement virtual sensors in the wireless sensor network.
- Communication between nodes in the network must be done using wireless communication.
- The algorithm implemented must be efficient enough so as not to introduce computational congestion in the system.
- The system must detect a malfunctioning node and replace it with a virtual sensor.
- The virtual sensor must be able to replicate data accurately as if they were real sensors.
- The data read by all the sensors must be stored in a database.
- The virtual sensors must be implemented on every corresponding node as well as having a copy of every trained virtual sensor on the server.
- Each sensor node should consist of a power unit, a sensor module and a processing unit.

2.2 Student tasks

The following tasks will need to be completed.

- An investigation of the current machine learning imputation techniques in literature must be done.
- An investigation of where and how virtual sensors have been implemented must be done.
- An investigation of wireless communication interfaces must be done to decide on the best communication interface for the product must be done.
- An investigation for the causes of lost data in wireless sensor networks must be done.
- The imputation algorithms must be designed and implemented with a focus on robustness.
- The algorithms must be trained and tested in MATLAB or Python using data that has been collected from the environment being sensed by the hardware.
- The control algorithms on the processing units for each sensor node must be implemented.
- The circuitry for the sensor nodes must be implemented on stripboard.
- The software and GUI for the server must be designed and implemented on a PC.

3. Functional analysis

This section contains information on the subsystems and each function unit of the system separated into the hardware and software components.

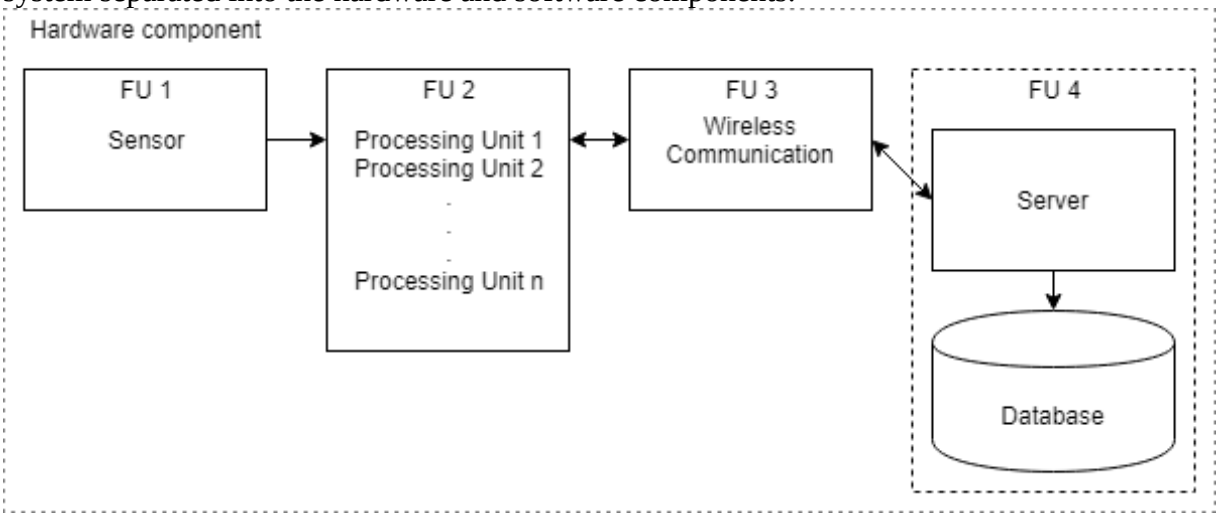

Figure 1.
Overview of the hardware component

The hardware component, shown in Figure 1, will consist of a sensor module (FU 1) which will sense the characteristics of the environment and be attached to a processing unit (FU 2). These two functional units cover a single sensor node. The processing unit will, at regular intervals, take environmental readings from the attached sensor(s) and package the data for transmission via wireless communication (FU 3) with the other sensor nodes as well as a central server (FU 4) which will contain the database that will store historical data for later use in data modelling and then training the virtual sensors of each sensor node as well as general storage for historical data of the network.

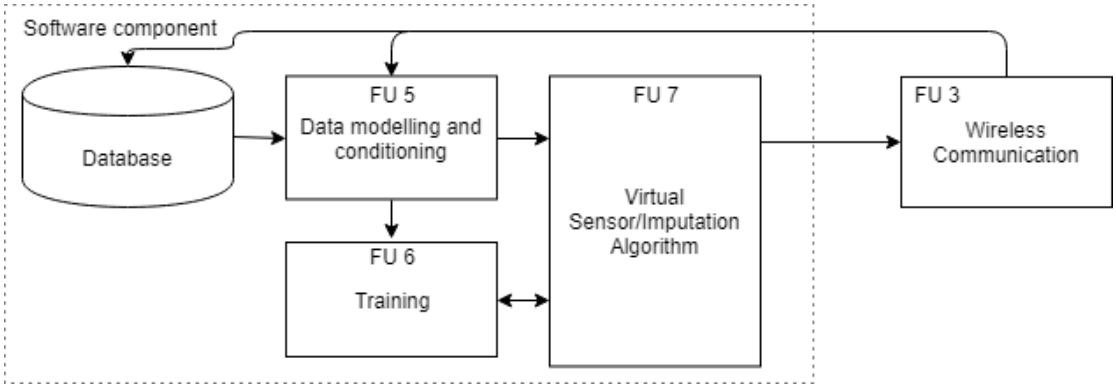

Figure 2.
Overview of the server software component.

The server software component, shown in Figure 2, will consist of the database and all the software of the server of the system.

When the system is functioning in a manner that all the sensor nodes are active with functioning physical sensor nodes, the server will simply store the sensor readings received from the sensor nodes through a wireless communication link (FU 3) in the local database. This data will then be modelled (FU 5) so that I may be used as offline training (FU 6) data for the machine learning algorithms that will be deployed as the virtual sensors on both the server as well as the processing unit of the sensor nodes.

When the system is functioning in a manner that a sensor node in the network has been taken offline, the incoming sensor data to the server from the remaining sensor nodes will be stored in the database and modelled (FU 5) to be used as an input for the virtual sensor (FU 7) contained on the server that will be activated once it is determined that a sensor node has stopped communicating with the server.

When the system is functioning in a manner that a sensor node is online (i.e. it is communicating with the server) but does not have a functioning sensor module attached, the server will continue to receive readings from the remaining sensor nodes that do have a functioning sensor module attached. These readings will be stored in the database and forwarded to the sensor node without a functioning sensor module to be used as the input for the virtual sensor contained on the sensor module.

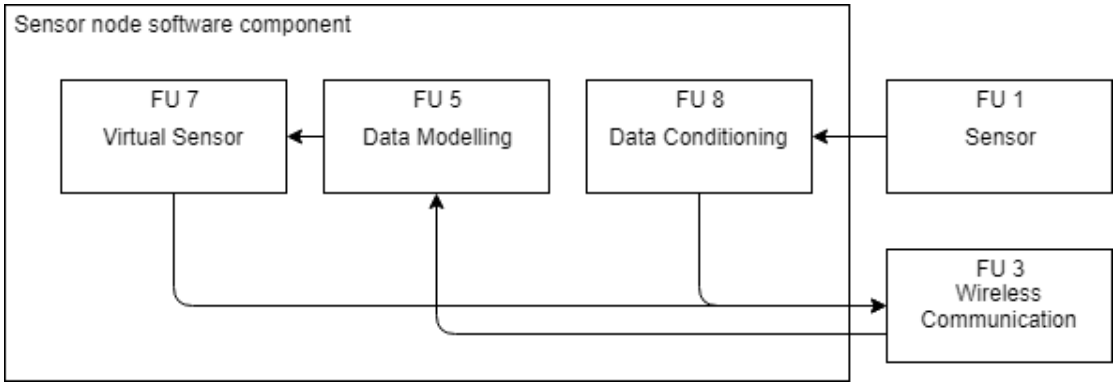

Figure 3.
Overview of the sensor node software component.

The sensor node software component, shown in Figure 3, will consist of two modes of operation.

The first and default mode of operation will consist of the sensor node taking environmental readings using the attached sensor module (FU 1) and using the digital filtering algorithms to reduce noise from the read data (FU 8). This data is then uploaded to the server using a wireless communication link (FU 3) to be stored in the server's database.

The second mode of operation will consist of the sensor node imputing environmental readings by activating the virtual sensor (FU 7), which itself will use the machine learning imputation method, if the node detects that no sensor module is attached to the processing unit. Readings from the other sensors in the network will be passed to the sensor node by the server through the wireless communication link (FU 3) and then modeled so that the data can be used as input data for the virtual sensor. The imputed sensor data from the virtual sensor is then uploaded to the server for storage in the database.

One of the potential methods of the virtual sensor involves the usage of a SOM, an unsupervised neural network, where the neurons are organised in a connected 2-D grid as shown in figure 5 and the correlation in data is learned using the steps in Figure 4.

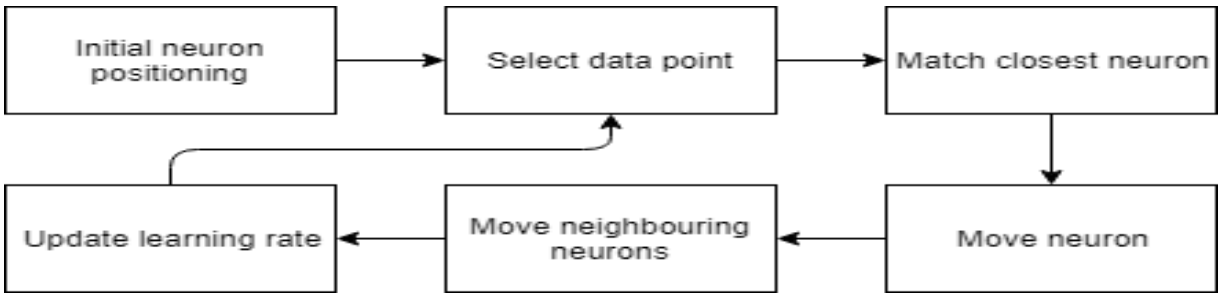

Figure 4.
The steps of a self-organising map.

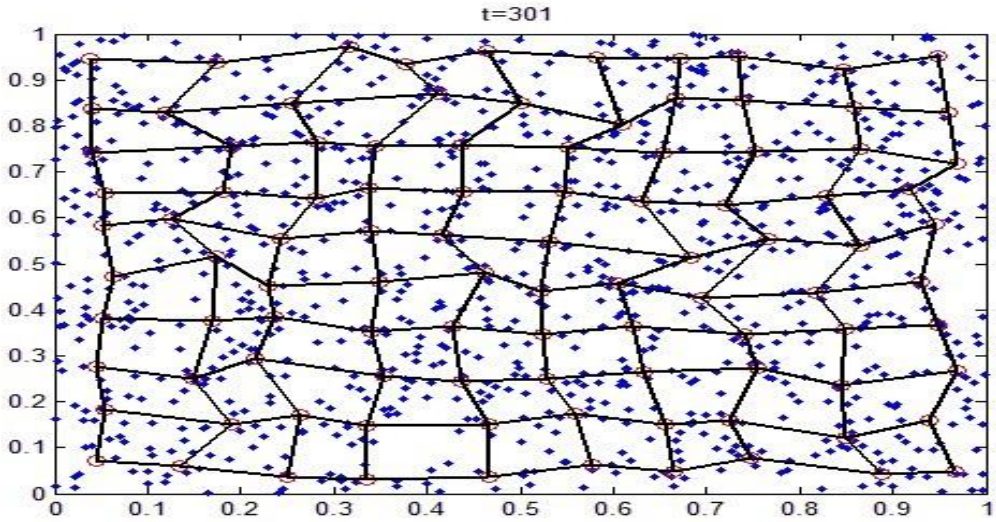

Figure 5.
A self-organising map neural network after 301 iterations.
The SOM initially randomises the positions of all the neurons. A data point is then selected randomly to be used as the reference point to match the neuron that is the closest in proximity to the point. The neuron is then moved closer to the data point. The neighbouring neurons in a certain radius of that neuron are then also moved closer based on how far they were positioned from the data point, with further neighbouring neurons moving less. The moving rate and the radius for the chosen neuron, also known as the learning rate, is then decreased on each iteration. These steps are repeated until the positions of all the neurons has stabilised in a way that significant neuron movement does not occur as seen in Figure 5.

4. Target specifications

This section contains the main specifications of the proposed project.

4.1 Mission-critical system specifications

The mission critical systems specifications are given in table 1 below.

SPECIFICATION (in measurable terms)	**ORIGIN** **(**or motivation of this specification)	**VERIFICATION** (how will you confirm that your system complies with the specification?)
The virtual sensor using the MLP neural network imputation method should impute values, at minimum, within 30% accuracy of the actual observed values for 90% of the data.	[2] and [3] both show that the imputed values from MLP neural network methods are within 30% of the actual observed values on average and within 15% in an absolute best-case scenario.	The imputed values from the virtual sensors will be compared to the physical sensor readings.
The virtual sensor using the MLP neural network imputation method should not stray further than 1.5 degrees Centigrade standard deviation from the mean values that would be sensed by a physical sensor.	A standard deviation greater than 1 would indicate that the virtual sensor has been improperly trained and will not be of any practical use. [7] suggests a standard deviation of not higher than 1.5 degrees Centigrade would provide a good indication of a well-trained system.	Each virtual sensor, both on the server and on the node, will be tested by turning nodes ability to physically sense the environment as well as completely turning off a sensor node to ensure that the server can activate the corresponding virtual sensor. If the virtual sensor can impute data that would be sensed by the physical sensor within the specified standard deviation from the mean, it would be deemed functioning correctly.
Imputation of a single value should take no longer than 7 seconds when using the MLP neural network algorithm	Imputation of a single value takes up to 7 seconds in [1] when using the MLP neural network imputation	A timer will be setup between the time that the virtual sensor has been given the input and the time that the virtual

while running on the server once the physical sensor data from the functioning sensor nodes has been input into the virtual sensor.	technique and this is due to the computations being done by the processor. To prevent congestion due to server processes taking place, some time must be given for these computations to take place before new values can be imputed.	sensor gives an output. This will determine the time it takes to impute a value on the server for the given algorithm.

Table 1.

Mission-critical system specifications

4.2 Field conditions

REQUIREMENT	**SPECIFICATION** (in measurable terms)
The sensor nodes must be directly connected to the server.	The connection will need to work over WiFi or Bluetooth
The server must run on a computer capable of processing large streams of data for training purposes.	Any modern computer with a CPU speed of at least 2GHz.

Table 2.
Field conditions

5. Deliverables

5.1 Technical deliverables

Table 3 shows the technical deliverables that will be required to complete the project.

DELIVERABLES	DESIGNED AND IMPLEMENTED BY STUDENT	OFF-THE-SHELF
Atmospheric (barometric or temperature or humidity) sensor modules for the processing units.		X
16-bit or 32-bit microcontroller for the sensor nodes.		X
Wireless communication modules.		X
Power supply for the sensor nodes.		X
Sensor module interface with the microcontroller.	X	
Wireless communication module interface with the microcontroller.	X	
Desktop PC for the server.		X
Desktop monitor.		X
Software libraries for accessing the database.	X	
Processing unit sensor control software.	X	
Processing unit communication protocol.	X	
Database on the server to store the data received from the physical sensors.	X	
Server communication protocol.	X	
Imputation algorithm using MLP neural network technique in MATLAB or Python for simulation.	X	
Imputation algorithm using MLP neural	X	

network technique on the sensor node processing units.		
Imputation algorithm on the server.	X	
Data modelling software on the server.	X	
Data modelling software on the sensor nodes.	X	
Graphical user interface for the PC application.	X	

Table 3.
Deliverables

5.2 Demonstration at the examination

1. All the software will be preloaded onto the sensor nodes and the server before the demonstration begins.
2. The physical system will then be demonstrated by switching on all the sensor nodes and the server and activating the system.
3. The graphical user interface on the PC will be used to show the current outputs of the sensor nodes in the network.
4. A software command will be submitted to one of the sensor nodes from the PC to activate the virtual sensor using the MLP neural network algorithm to impute data.
5. The sensor node will acknowledge this command and return the imputed value as well as the sensor reading to compare it values. This will confirm the virtual sensor is functioning correctly using the MLP neural network imputation technique.
6. A software command will then be entered on the server to activate a virtual sensor using the MLP neural network imputation technique on the server that will impute values for the chosen sensor node. The values will again be compared to the values sensed by the sensor node. This will confirm that the virtual sensor on the server is functioning correctly using the MLP neural network imputation technique.

6. References

[1] Y. Li and L. Parker, "Nearest neighbour imputation using spatial-temporal correlations in wireless sensor networks," *Information Fusion*, vol. 15, pp. 64-79, 2014.
[2] J. Jerez, I. Molina, P. Garcia-Laencina, E. Alba, N. Ribelles, M. Martin, and L. Franco, "Missing data imputation using statistical and machine learning methods in a real breast cancer problem," *Artifical Intelligence in Medicine*, vol. 50, no. 2, pp. 105-115, 2010.
[3] P. Garcia-Laencina, J. Sancho-Gomez, A. Figueras-Vidal, and M. Verleysen, "K nearest neighbours with mutual information for simultaneous classification and missing data imputation," *Neurocomputing*, vol. 72, no. 7-9, pp. 1483-1493, 2009.
[4] L. Li, J. Zhang, F. Yang, and B. Ran, "Robust and flexible strategy for missing data imputation in intelligent transportation system", *Intelligent Transport System*, vol. 12, no. 2, pp. 151-157.
[5] K. K. Nalanda, "Using fuzzy c means and multi layer perceptron for data imputation: Simple v/s complex dataset," in *3rd International Conference on Recent Advances in Information Technology*. IEEE, March 2016.
[6] Y. Li and L. Parker," Spatial-temporal imputation technique for classification with missing data in a wireless sensor network," in *Proceedings of IEEE International Conference on Intelligent Robots and Systems*. IEEE, 2008.
[7] A. Wang, Y. Chen, N. An, J. Yang, L. Li, and L. Jiang, "Microarray missing value imputation: A regularized local learning method," in *IEEE/ACM Transactions on Computational Biology and Bioinformatics*. IEEE, 2018.

Part 4. Main report

1. Literature study

1.1 BACKGROUND AND CONTEXT OF THE PROBLEM

With the widespread proliferation of sensors in all different types of environments a new use case has come up in recent times called the Internet of Things (IoT) where sensor devices are interconnected to form wireless sensor networks (WSN). WSN have applications in many different fields including but not limited to the mining and aeronautic industry.

Different WSN may use different technologies to achieve the required goals, be it networks that may require low-power usage (LoRa), networks that need to communicate over vast distances at a low cost (Sigfox) or networks that need fast and reliable communication over shorter distances (WiFi).

Two problems arise in all these WSN that may cause loss of important data. The first problem deals with the noise level in the environment. If the noise level is high, then it may result in a degraded signal strength and thus performance in the WSN degrades to a point where data packets are lost (packet loss) or corrupted packets being transmitted across the network. These lost or corrupted packets may then cause errors further down the WSN pipeline if the application requires corrective action. The second problem deals with dead sensor nodes in the network. Dead sensor nodes are sensor nodes in a network that have failed or stopped responding and thus the data from these nodes is no longer being collected.

Much of the research based on virtual sensors (VS) has focused on using different machine learning approaches in representing the lost data or in creating new data by finding the relationship between different nodes in a network.

VS systems implementing machine learning can be segmented into three main different types of learning namely lazy learning, supervised learning and unsupervised learning.

1.1.1 Lazy learning systems

In lazy learning systems, the artificial intelligence does not require an explicit learning phase but uses an existing database to find the closest matching neighbours to the incoming inputs with *k*-Nearest Neighbours (KNN) algorithm [1], visualised in figure 1, being the one of the most widely used methods.

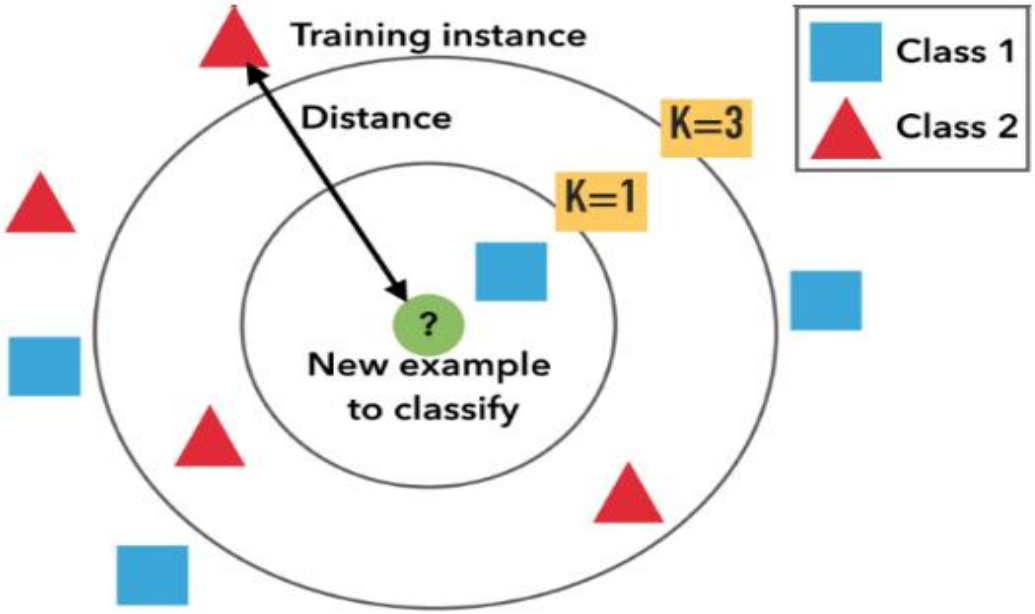

Figure 1.
Visualisation of the KNN algorithm.

The advantage of these systems is that no time is required to train the artificial intelligence as the generalisation of the training data does not occur until a query is made to the system. One such system using the KNN algorithm in combination with a decision tree, as discussed in [2], was found to be very effective in smartphones that need to detect head movements for a virtual reality device with accuracy only going as low as 90% when using a combination of accelerator, gyroscope and magnetometer sensor readings; however, the data set that is used must be carefully selected beforehand or else the classification accuracy will not have the optimal result. This requirement of acquiring a suitable database beforehand is not feasible in the real-world environment where environmental noise must be accounted for as well in the data set.

1.1.2 Supervised learning systems

In supervised learning systems, unlike the lazy learning systems, the artificial intelligence requires a training phase to generalise data so that queries can be made to the system. Learning is supervised in the manner that training targets are provided to the artificial intelligence for the desired model and parameters are adjusted accordingly over many training epoch cycles until a desirable model emerges. The most famous of these supervised learning systems is the artificial neural network (ANN). There are differing types of ANN namely feed-forward neural networks (FFNN), recurrent neural networks (RNN) and convolutional neural networks (CNN) being just a few, with each having their own field of applications in artificial intelligence. Multiple layered perceptron (MLP), figure 2 [3], are a form of FFNN which have a minimum of three layers namely the input layer, the hidden layer of which there may be more than one and finally the output layer. In the case of MLPs there are various differing training methods in the literature that are covered later in this section.

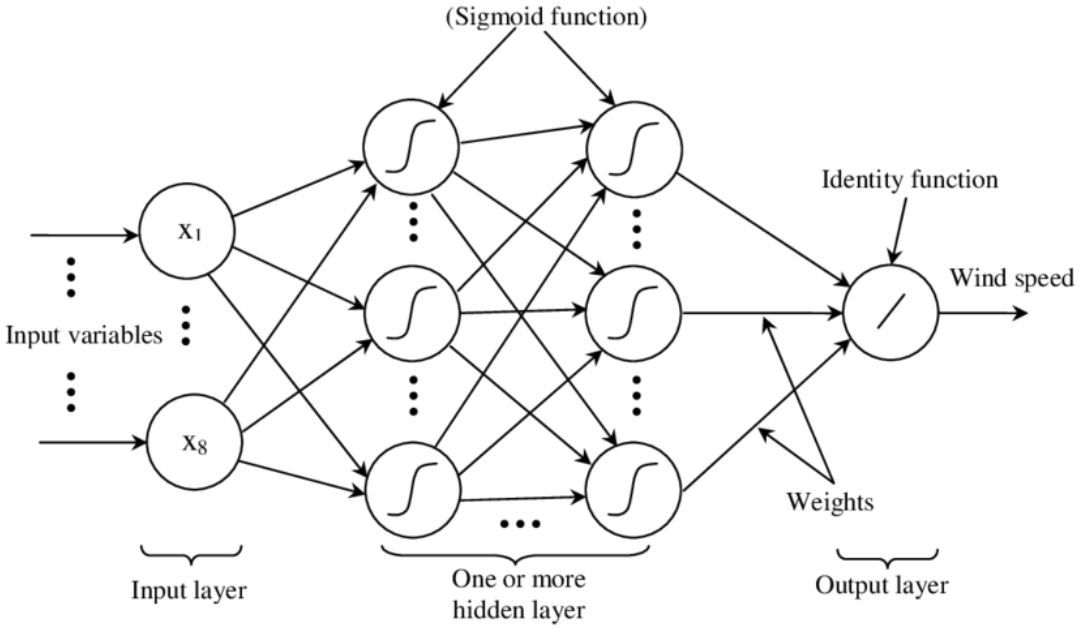

Figure 2.
Structure of a MLP with multiple inputs and hidden layers and a single output node.

As shown by [3], MLPs are universal function approximators which allow them to create mathematical models through regression analysis as well as being useful in the field of classification problems which is another case of regression. VSs can be implemented using MLPs as was done in [4] where a virtual Infrared Radiation (IR) sensor based on a conventional visual (RGB) sensor is used to estimate thermal IR images for terrain classification. Mean Squared Error is used as the loss function during training and is defined as

$$\mathcal{L}_{MSE} = \frac{1}{|S| \times C} \sum_{i \epsilon S} \sum_{j=1}^{C} \left(a_{ij} - b_{ij}\right)^2 \quad (1)$$

where C is the number of channels, $\boldsymbol{a_{ij}}$ and $\boldsymbol{b_{ij}}$ are thermal values at each pixel (i, j) of a ground truth thermal image $\boldsymbol{a}$ and an output thermal image $\boldsymbol{b}$**.** Training was done using genetic algorithms instead of the widely used backpropagation algorithm. The results of the work in [4] using the MLP method were promising where the MSE and standard deviation of the temperature difference was 1.8 degrees Celsius and 3.7 degree Celsius respectively.

1.1.3 Unsupervised learning systems

In unsupervised learning systems the artificial intelligence requires a training phase but, unlike supervised learning systems, there are no corrections made in terms of requirements in terms of the training targets i.e. no backpropagation or similar algorithm is used to correct the error of the artificial intelligence. The Kohonen map, figure 3, described in [5], otherwise known as a SOM is a widely used type of FFNN with an input layer and an output layer each which has multiple nodes associated with

the layer. The SOM works by first being initiated to a state with random weights and then training occurs by forward propagating a dataset over several epochs.

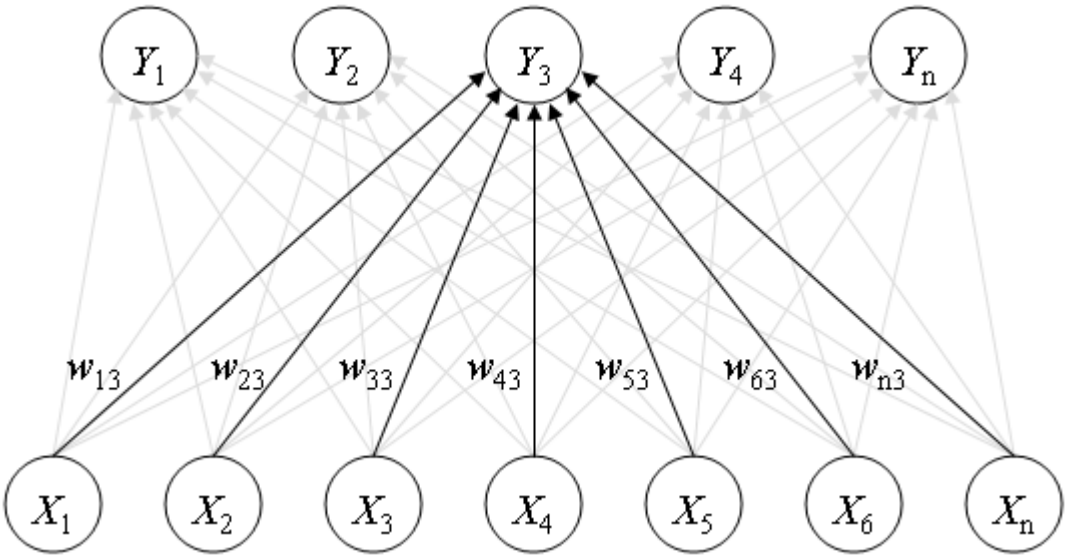

Figure 3.
A Kohonen Map with n inputs and outputs, fully connected from the input layer to the output layer.

With each forward propagation the nodes within a certain range of the desired output value move closer to the target output where the closer nodes move more than the further nodes. The amount that the nodes may move towards these target values decreases with each epoch by an amount set beforehand. The goal of the SOM is to organise the neural network outputs in such a manner that the target data is mapped accurately. This requires the output layer have enough nodes to cover as many data points as possible which may not be immediately clear during experimentation and would require iterating several times before an optimal number of nodes is found for the relationship between input data and output target data. A SOM is used in [6] for the localisation of sensor nodes in a WSN. Each sensor node is represented by an output node in the SOM and is initially trained based on the received signal strength from the nodes to then triangulate a precise location for the node. It was found that the number of nodes heavily influenced the accuracy of the data and that an increase in output nodes leads to less accurate localisation in some cases for certain sensor nodes. This VS proved to be less accurate than the triangulation method with which it was tested against.

1.1.4 Supervised learning algorithms

Within the supervised learning systems there are many different methods used in training these systems. The most widely used training method makes use of the backpropagation algorithm but in recent genetic algorithms (GA) have found their uses in training MLPs.

1.1.4.a Backpropagation algorithm

The backpropagation algorithm [7] calculates the output error in an MLP then assigns fault to each weight in the MLP after the initial forward pass of inputs. There are various supplementing algorithms used, the most basic being simple gradient descent and more recent algorithms using the Levenberg-Marquardt [8, 9] method to calculate a gradient. The only requirement for using the backpropagation algorithm is that the activation function of a neuron in the MLP must be differentiable. The main issue with the backpropagation algorithm is that the weights in the MLP may be trained in a way that an optimum solution may be found for the desired input-output relationship, but it may not be the most optimal due to the algorithm regressing to a local optimum instead of a global optimum. Another issue is that the backpropagation algorithm does not perform well if plateaus are present during the training phase. Thus, the performance of an MLP trained using backpropagation is affected by the initial randomised weights of the MLP before training is performed and so more research is being done in other methods of training MLPs especially with the increase in computational power.

1.1.4.b Genetic algorithm

GA work on the idea of natural selection where only the fittest in a population will pass on their genes. This is known as neuro-evolution. There are many different versions of the GA where different properties are used to determine genes and phenomes that would be crossed over to generate new populations from the fittest sample size. In MLPs, GAs generally use the weights as the genes but there are other implementations that combine the weights as well as the number of nodes and hidden layers [10]. The way a GA is used in training neural networks is that a population of neural networks are initialised with random weights. The dataset is then forward propagated through the network with target outputs and an error is calculated based on some pre-determined loss function, the simplest of which is based on the Euclidean distance. The fittest candidates in the population are then selected based on the calculated error and are known as the parent nodes. Through various differing methods of selection, parent nodes are paired and their genes crossed over to form new child nodes, some of which may be mutated using some random process to avoid stagnation in the population, until the population reaches the original population value and the process restarts. The advantage of this system is that, due to the randomly initiated state of the first population, there is less likelihood that the global optimum solution would not be found using neuro-evolution. The disadvantage is that the computational power required is much greater than for backpropagation; however, in [11] it was found that training FFNN using GAs shows that this method outperformed the backpropagation algorithm both in the final evaluation and in the time to train as well as never having an increase in error as the number of training iterations increased seen in figure 4.

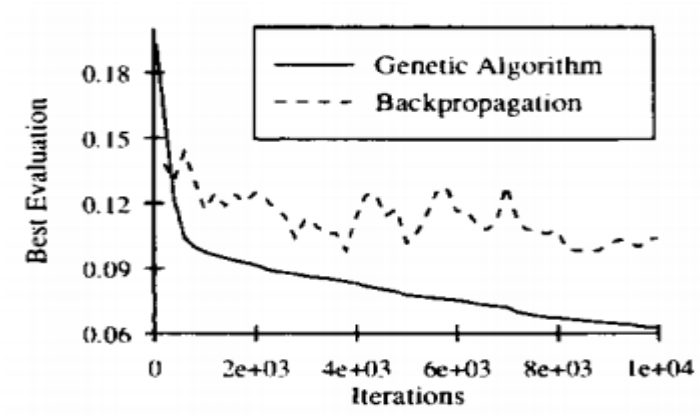

Figure 4.
Training error of GA and backpropagation

1.2 APPLICATION OF THE LITERATURE

This project is largely concerned with the identification of the relationship between sensor nodes in a sensor network using data that has been collected by the sensor network hence many of the above algorithms are applicable to the proposed scenario. The fact that multiple sensor nodes will be deployed to generate training data results in an interesting case where each sensor node will require its own virtual sensor. It is assumed that a user of this system will have collected the required training data for each sensor node prior to activating a virtual sensor. The algorithms were carefully considered with their advantages and disadvantages kept in mind and the most suitable algorithm was chosen for the problem based on modern trends in machine learning.

Machine learning is by far the most integral part of the project since the relationship that must be modelled by the system between all the sensor nodes in the network is incredibly important to the functioning of the virtual sensors. GA was chosen due to experimental evidence showing that the training method converges much faster than the backpropagation algorithm which would lessen the amount of training time required as well as not falling into the potential trap of local optimums which may require a reinitialization of the training due to stagnant results. GAs have multiple different approaches that can be used when finding the fittest population and then repopulating so plenty of experimentation may be done to find the most optimal MLP topology during training whereas the backpropagation would need to be iterated through using a grid search.

A star topology was chosen for the sensor network that would make use of WiFi communication technology where the sensor nodes would communicate with a central server. Low power usage is of no concern in this project and the project is concerned with sensor networks where distances from each node are not too far apart from each other (<50m) which would mean that expensive technology such as ZigBee and LoRa are not required.

The database of the system will be stored locally by the server and used as training and test data for the VS once enough data has been gathered. A separate database will collect the VS outputs for comparison to real sensed data once training is complete.

2. Approach

The solutions to the engineering problems provided by this project were carefully studied and approached in a manner that enough consideration was given to alternative designs with respect to each design's trade-offs. All the available tools were utilised to aid in the design and implementation of the project as well as an engineering approach to solve the problem. Below each entity in the functional analysis as in the project proposal (**Part 3)**, **Section 3** is described systematically.

2.1 TEMPERATURE RECORDING, FILTERING AND CONVERSION

The temperature that must be recorded by each sensor node is one of the most important steps in the system as without this recording mechanism no data can be acquired to train the VSs that accompany the system. Since the ambient temperature where the sensor nodes would be deployed is not expected to change too quickly under normal conditions, a sampling rate of 1 sample per 30 second interval was chosen. The sensor node was designed from first principles instead of using an off-the-shelf solution as cost was taken into consideration. No scaling circuit was designed as the modified Steinhart-Hart equation [12] was deemed a better software solution than using hardware to deal with converting voltages to degrees Celsius. It was found during testing that a filtering algorithm was required for the incoming values as a small bit of noise was experienced in the form of temperature anomalies due to hot air pockets moving through the buildings as well as hardware glitches that resulted in ground values being returned on very rare occasions. A mean filter, median filter and a scalar Kalman filter (SKF) was designed and implemented in software on the server to deal with these anomalies and it was found that the SKF performed much better in terms of dealing with the anomalies than the other two methods as well as being computationally simple and was chosen as the final filtering algorithm.

Temperature is recorded when the server sends a query to a sensor node requesting a temperature sensor reading. The sensor node analog-to-digital converter (ADC) converts the voltage values to a 10-bit digital value and packaged in a byte array and sent to the server. This procedure is performed two more times for the other sensor nodes and the values are then stored in a CSV file along with a timestamp which forms the database.

2.2 VIRTUAL SENSOR ALGORITHM

For the virtual sensor algorithm, only the MLP methods were considered while SOM and KNN were discounted due to the lack of ability for the SOM algorithm or KNN algorithm to perform or otherwise find a regression model. An algorithm was implemented to iterate through various topologies based on number of hidden nodes as well as hidden layers to determine an optimal middle ground where an acceptable accuracy was reached by the MLP while taking training time into consideration.

Iterations took the average final error over 20 tries per topology and the optimum topology was then used for the final MLP. Further reading is suggested in **Part3, section 3.7**. All algorithms were designed, implemented and tested in Python.

2.3 VIRTUAL SENSOR TRAINING

The database for training the VS consists of a minimum of 5 days' worth of data which equates to roughly 14400 collected sensor readings. This data is split up into a training set and a test set of roughly 60% training data and 40% test data. Before training can begin, the data is first normalised to values between 0.0 and 1.0 by finding the maximum and minimum in the data then scaling it down by using the scaling equation defined as

$$\frac{x_i - x_{min}}{x_{max} - x_{min}} \quad (1)$$

Where x_i is the current value being scaled, x_{min} is the lowest value in the dataset and x_{max} is the largest value in the dataset. An initial population of randomly initialised neural networks is instanced with the above normalisation kept in mind.

2.3.1 Forward propagation

Forward propagation in the MLP consists of giving the appropriate input nodes the correct input values and then propagating these values forward through the MLP through a series of nodes and layers where each node sums all incoming inputs then activates by using an activation function defined in this project by the Softsign function as

$$y = \frac{x}{1+|x|} \quad (2)$$

until ultimately arriving at the output node with the predicted value. The Softsign function was chosen due to its faster computation speed as compared to the sigmoid activation function and tanh activation function.

2.3.2 Loss function

The loss function for the neural networks was adapted from the literature, the loss function described by [11] was of specific interest as they are low cost computationally and give enough information regarding the error percent of the neural network. The error was calculated by adding the error of each prediction and dividing by the total predictions made during each epoch.

2.3.3 Fittest population selection and mating

Following on from the error calculation, the fittest neural networks are selected for breeding. Parents are then randomly selected, irrespective of which parent is the most fit in the fittest population, for breeding. Weights are randomly selected across both parent's weight arrays to complete a child neural network's weight array with a small chance to completely randomise a single weight in the array to act as a mutator until

the original population size is reached. This is repeated over a specified number of epochs until a candidate neural network emerges for a specific VS.

2.4 VIRTUAL SENSOR IMPLEMENTATION

The VSs are deployed on both the server and the sensor nodes, applying the weights that have been found through the training phase. As such, the readings from the VS on the sensor node should not differ much or at all from the VS deployed on the servers. The system will identify if a sensor node is not communicating with the server and use the appropriate VS to take over sensing operations until the sensor node is able to reconnect to the network. Otherwise the sensor node will send both VS data and actual sensed data back to the server.

3. Design and implementation

The design approach adopted is a top-down design making use of the functional decomposed blocks of the functional block diagram detailed in figure 1, figure 2 and figure 3 in the project proposal (**Part 3**). Each major functional block is discussed in terms of the design, simulation, performance, evaluation and optimisation where applicable.

3.1 SENSOR

The sensor circuit is implemented using a thermistor to realise a temperature sensor. From the PIC32MX220F032B (PIC32) microcontroller datasheet (**Part 5**), the maximum input voltage to the ADC is 3.3V. Taking this into consideration, an NTC thermistor with R25°C = 1000Ω was chosen as the sensing component and a 1000Ω resistor used for the voltage divider bias resistance. This ensures that the room temperature would always be half the maximum input voltage as seen by the ADC terminal of the microcontroller. The circuit schematic is shown in figure 5.

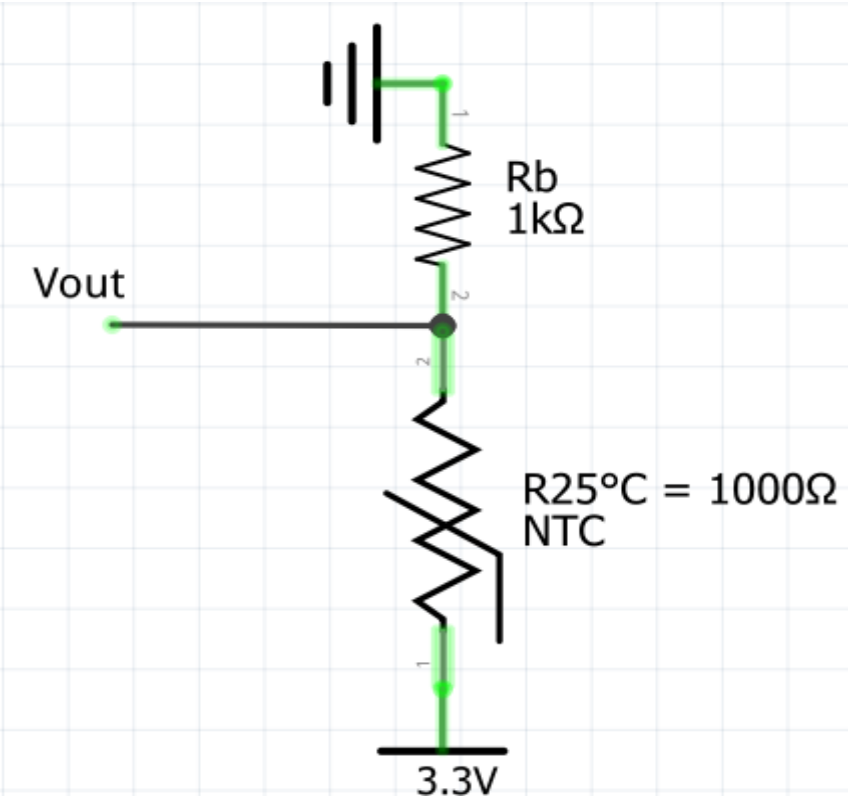

Figure 5.
Voltage divider for the sensor node.

The conversion equation from voltage to an unsigned integer is defined as

$$\frac{Resolution\ of\ the\ ADC}{System\ Voltage} = \frac{ADC\ Reading}{Analog\ Voltage\ Measured} \quad (3)$$

Where the resolution of the ADC is taken from the PIC32 datasheet and is 1023, the system voltage is 3.3V, the ADC Reading and Analog Voltage Measured are the desired values. From the TTC05 thermistor datasheet (**Part 5**), the lowest possible resistance is 0.031Ω for a temperature of 130°C and the highest possible resistance is 10556Ω for a temperature of -30°C. Using the equation for a voltage divider output defined as

$$V_{out} = \frac{V_{in}R_t}{R_t+R_b} \quad (4)$$

where the bias resistance, R_b, is 1000Ω and V_{in} is 3.3V, the lowest and highest expected values by the ADC in terms of voltage are:

- Lowest voltage when temperature is high with $R_t = 0.031\Omega$: 0.001V
- Highest voltage when temperature is low with $R_t = 10556\Omega$: 3.014V

This indicates that the sensor would not make use of the full scale of the ADC by not utilising the highest 9.13% possible but is still deemed acceptable without needing to implement a scaling circuit.

Based on the calculated values above, the highest and lowest converted values to be expected as well as the room temperature converted value are shown below in table 1 by re-arranging equation 3.1 as

$$\frac{(Resolution\ of\ the\ ADC)(Analog\ Voltage\ Measured)}{System\ Voltage} = ADC\ Reading \quad (5)$$

Temperature	Input voltage	ADC Reading
130°C	3.014V	935
25°C	1.65V	512
-30°C	0.001V	1

Table 1.
ADC Readings based on input voltage from temperature sensor.

Two voltage divider circuits were implemented with the outputs connected to different analog input pins on the microcontroller meaning two temperature sensors were implemented per sensor node. This was done for redundancy since components may have slightly different tolerances than specified in the datasheet.

In [4] a weight analysis was conducted on the trained neural networks and it was found that all weights connected to the atmospheric pressure and atmospheric humidity sensors were dead weights (close to 0). In other words, the neural network training determined the inputs had no effect on the output. Considering this, it was decided to not implement these atmospheric sensors.

3.2 MICROCONTROLLER

Due to the low power usage of this microcontroller (100mA on average when running at 80MHz), having software support in the form of the Harmony framework as well as being a well-stocked microcontroller solution available locally in South Africa the PIC32 was chosen as the processing unit for the sensor node. The pins that were chosen to be used are indicated in figure 6 and their function detailed in table 2. All power pins

were attached with a 10uF tantalum capacitor to ground as was suggested in the PIC32 datasheet to protect against current surges.

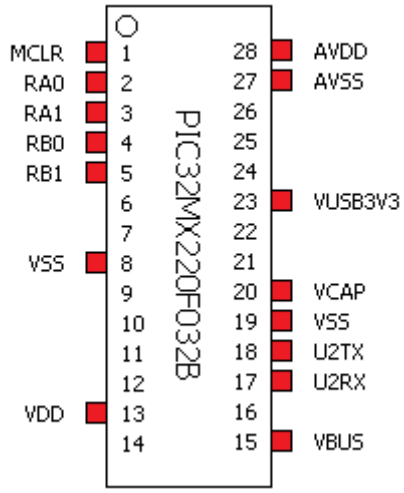

Figure 6.
PIC32 pins used.

Pin number	Function
1	Required to program PIC32
2	Analog Sensor Input 1
3	Analog Sensor Input 2
4	Required to program PIC32
5	Required to program PIC32
8	PIC32 Voltage Ground Supply
13	PIC32 Voltage Power Supply (3.3V)
15	PIC32 Voltage Power Supply (3.3V)
17	USART Receive pin
18	USART Transmit pin
19	PIC32 Voltage Ground Supply
20	Voltage Capacitor 10uF
23	PIC32 Power Pin
27	ADC Voltage Reference (3.3V)
28	ADC Voltage Ground

Table 2.
Pin description.

3.3 WIFI MODULE

For the wireless communication there was a choice between using Bluetooth technology or WiFi. WiFi was chosen as TCP/IP was the preferred method of message transmission in the network as well as allowing for a relatively simple implementation of a star network topology. The ESP8266 WiFi module was chosen due to the low power options available on the ESP8266 microcontroller as well as for the ability to communicate through USART. Four ESP8266 WiFi modules were implemented in the project, one to serve as the server's communication device and the other three to be

used on each sensor node. The firmware version installed on each module is 1.5.1 and is available in **Part 5** as well as the instructions and hardware setup required to flash the firmware.

3.3.1 Server module

The module used to act as the server's communication interface requires communication through USB to transmit all received data to the server on the desktop PC. A serial-to-TTL device was required. Power to the module is provided through the USB port to both the serial-to-TTL device as well as the ESP8266 module which has a maximum current draw of 350mA on startup and 150mA on average while transmitting or receiving data. It was found that the desktop PCs USB ports were able to supply the required power and no external power source was required. The baud rate of the module was set to 115200bps as this was the maximum baud rate possible from the desktop PCs communication link which translated to 14.4kB/s maximum transfer rate, well above what is required to transmit and receive data without losing any packets.

The following commands in table 3 must be sent to the ESP8266 module through a terminal interface, such as Termite or PuTTY, to set it to server mode.

Command	**Description**
AT+CWMODE=2	Set the module to station and access point mode.
AT+CWSAP= "ESP Access Point 1"	Set the SSID of the module to be accessed via WiFi
AT+CIPMUX=1	Set the module into multiple access mode so that multiple devices (maximum 8) can connect to the server
AT+CIPSERVER=1,333	Start the TCP/IP server on port 333 and begin accepting TCP client connections.

Table 3.
ESP8266 server mode setup commands.

3.3.2 Client module

The modules used to act as the client's communication interface requires communication through USART to transmit all received data to the microcontroller as well as transmit data over WiFi when data is received through USART from the microcontroller. 10uF tantalum capacitors were placed between power and ground due to the high current draw at startup. The baud rate of the module was set to 115200bps as this was the maximum baud rate possible from the microcontroller's communication link, well above what is required to transmit and receive data without losing any packets.

The following commands in table 4 must be sent to the ESP8266 module through the USART interface by the microcontroller to set it to client mode prior to attempting to connect to the server.

Command	Description
AT+CWMODE=1	Set the module to access point mode.
AT+CWJAP= “ESP Access Point 1”	Set the module to automatically seek out the server’s specified SSID and connect to it if found.

Table 4.
ESP8266 client mode setup commands.

3.4 FINAL SENSOR NODE CIRCUIT

The circuits were implemented on a stripboard although PCB was available at a greater cost. Due to USB power supplies having a standard output voltage of 5V, a 3.3V voltage regulator was used to bring down the voltage as both the microcontroller and WiFi module have a maximum voltage tolerance of 3.6V. The schematic is shown in figure 7 and the stripboard design, done in Fritzing, in figure 8. There is a single power line and ground line shared between all components in the circuitry.

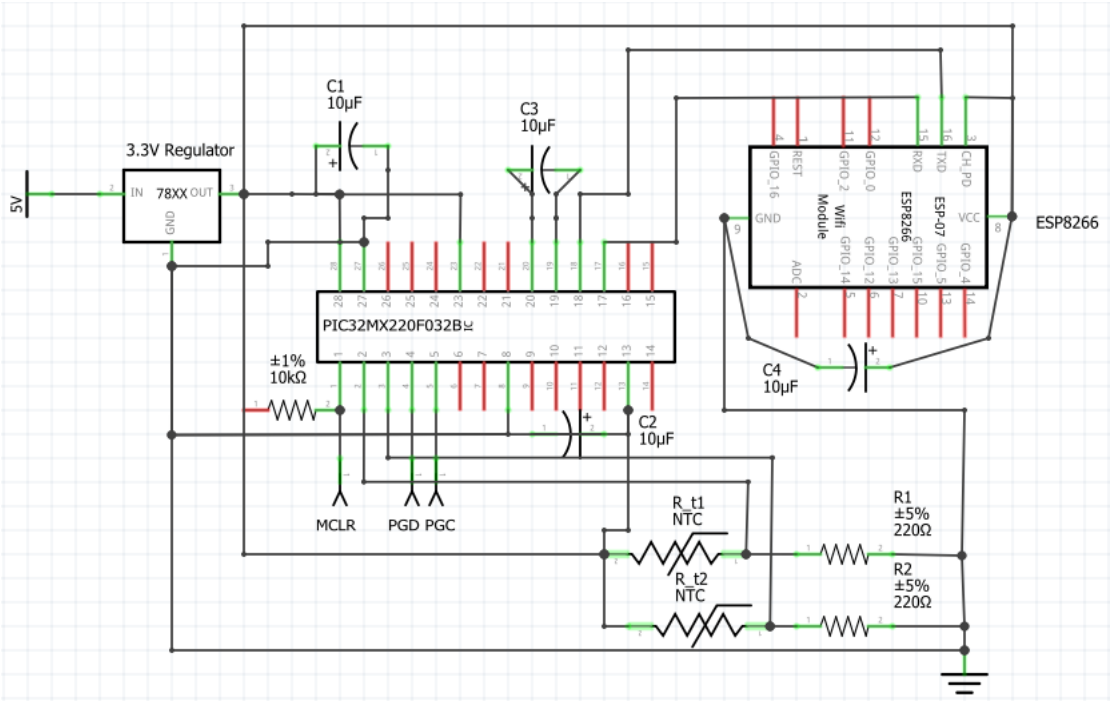

Figure 7.
Full circuit schematic of the sensor node.

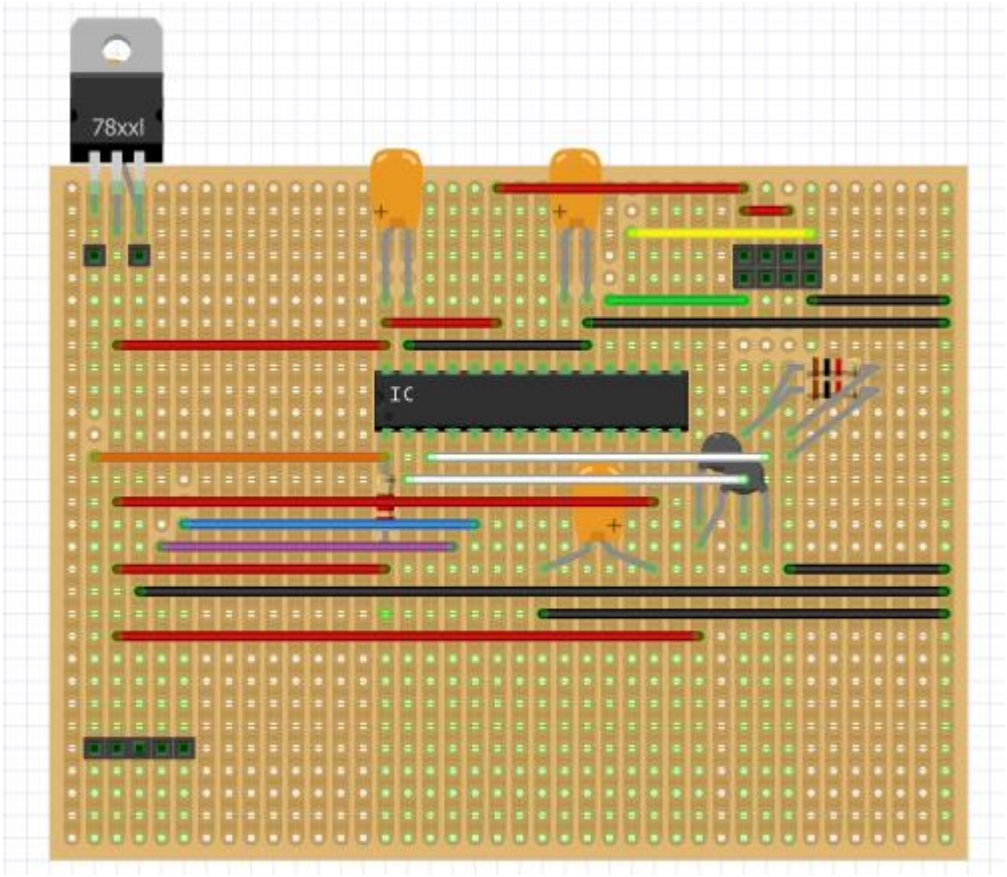

Figure 8.
Stripboard design of the sensor node done in Fritzing

Due to the Fritzing software not including an ESP8266 WiFi module it must be noted that the 8 pin holders in figure 8 in the top right section that are in a 4x4 configuration of the stripboard were reserved for the pins of the ESP8266 WiFi module. The five pins in the bottom left are reserved for the PICKIT3 programming device that is required to program the PIC32.

3.5 MICROCONTROLLER SOFTWARE

The Harmony framework was used to initialize the clock speed and peripherals to the required states to perform specific operations, specifically USART communication and PLL clock required for the ADC. The ADC was configured manually using information provided by the datasheet. A workflow engine was considered but due to the unpredictability of peripheral interrupts interrupting the program flow, a state machine was implemented to ensure no important functions would be interrupted during important functions. The design of the state machine is shown in figure 9.

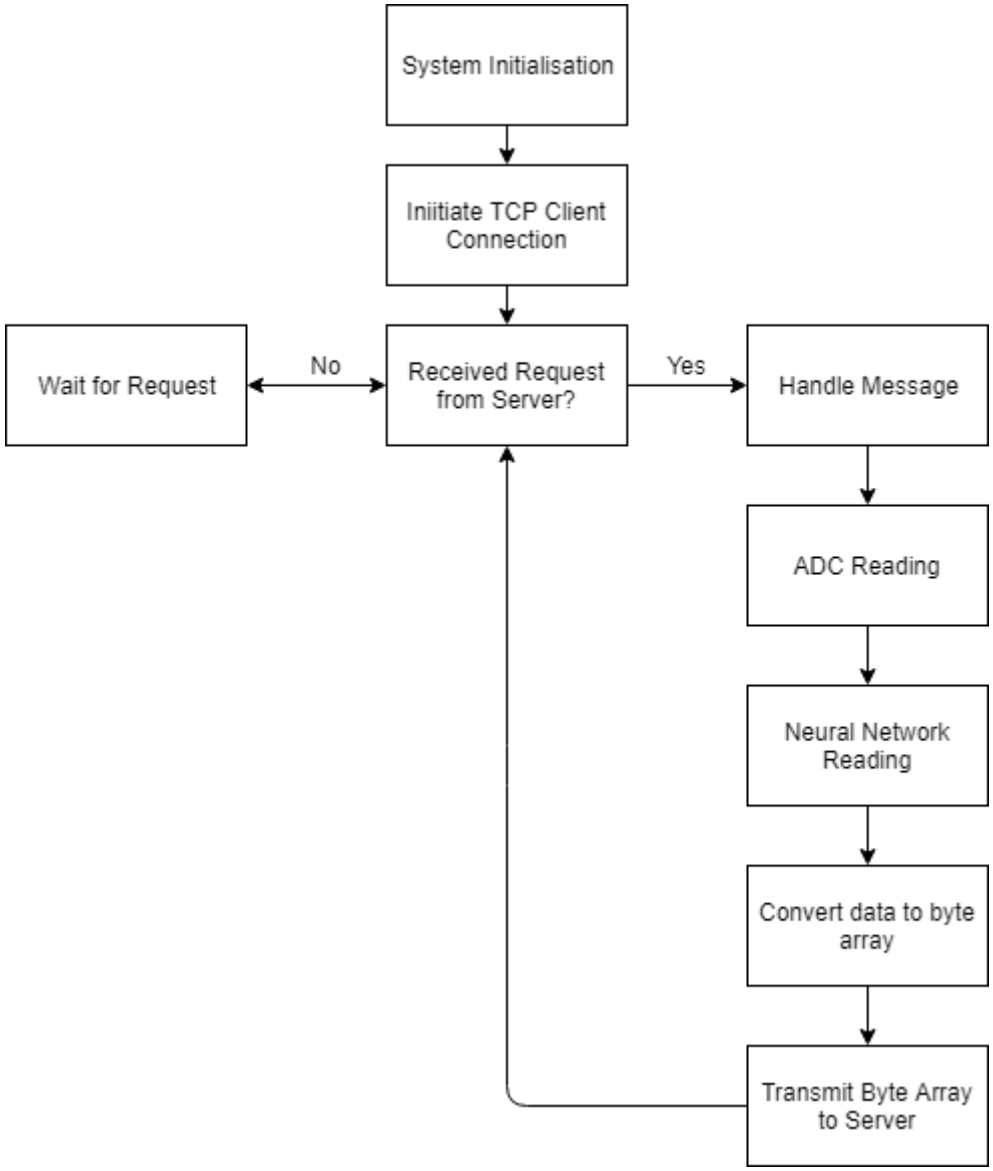

Figure 9.
Microcontroller State Machine.

3.5.1 System Initialisation

The microcontrollers initial state involves initializing all data structures that were used as well as the system clocks and pin configurations. The Harmony framework was used to set the pin configuration of the system shown in table 5.

Pin number	Function
2	Analog Input 0
3	Analog Input 1
18	USART Transmit 0
19	USART Receive 1

Table 5.
PIC32 pin configuration.

The following settings were used in setting up the USART communication with the PIC32 microcontroller:

- Static driver implementation
- Interrupt Mode
- Byte model support
- Baud rate: 115200 (required for communication with ESP8266 WiFi module)
- Interrupt priority level: 1
- Operation mode: normal USART operation mode
- No handshake protocols
- Line control: 8 data bits, no parity bit, 1 stop bit (8-N-1)

The following settings were used in setting up the ADC for sensor data acquisition:

- Pin2 and Pin3 set to analog
- Pin2 and Pin3 set to input
- Internal counter set to 16
- Set voltage reference to Pin27 and Pin28
- Set acquisition time to 3.841us (based on PIC32 datasheet suggestion)
- Enable ADC

3.5.2 Initial TCP Client Connection

The sensor node must start an initial client connection to the server. The commands in table 6 are sent from the microcontroller through USART to the ESP8266 WiFi module to initiate the TCP client connection. Due to the ESP8266 WiFi module having a start-up time, the microcontroller waits before sending the commands at startup as seen in figure 10. There is a mandatory 100ms waiting time between each command so that the ESP8266 WiFi module can process and execute the commands.

Command	**Function**
AT+CWMODE=1	Set the WiFi module to access point mode.
AT+CIPMODE=0	Set the WiFi module to TCP connection mode.
AT+CIPSTART=”TCP”,”192.168.4.1”,333	Initiate a TCP connection to a specified IP Address and port number.

Table 6.
Initial commands required to initiate a TCP client connection.

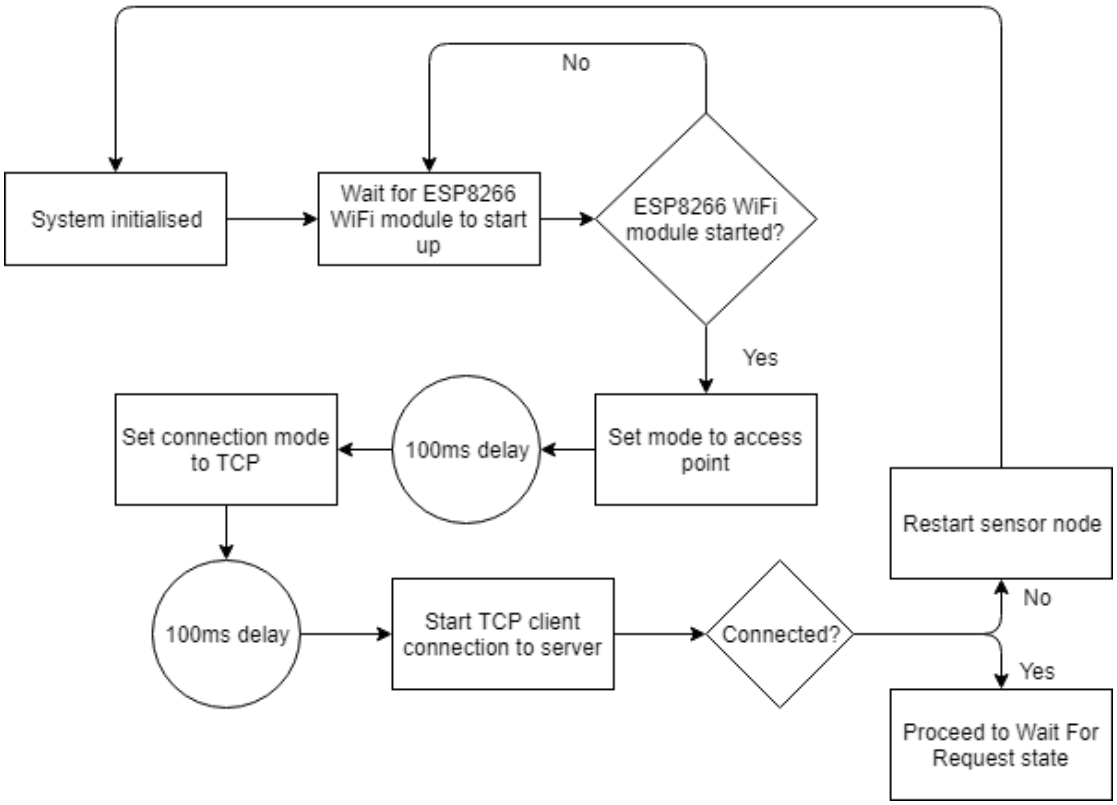

Figure 10.
Design of the initial connection state.

3.5.3 Message handling

When a message is received through WiFi it is communicated to the microcontroller through USART. The bytes received must be handled and assigned to the relevant data structures to be used in the rest of the program. An interrupt handler was used to deal with incoming bytes. A byte message will consist of the following attributes:

- Header byte [1 byte]
- Message size [1 byte]
- Data bytes [6 bytes]
- Tail byte [1 byte]

The data bytes consist of the raw sensor data that the server receives from the other sensor nodes in the network as well as the seconds that have passed since midnight in byte format. The design of the message handling is shown in figure 11.

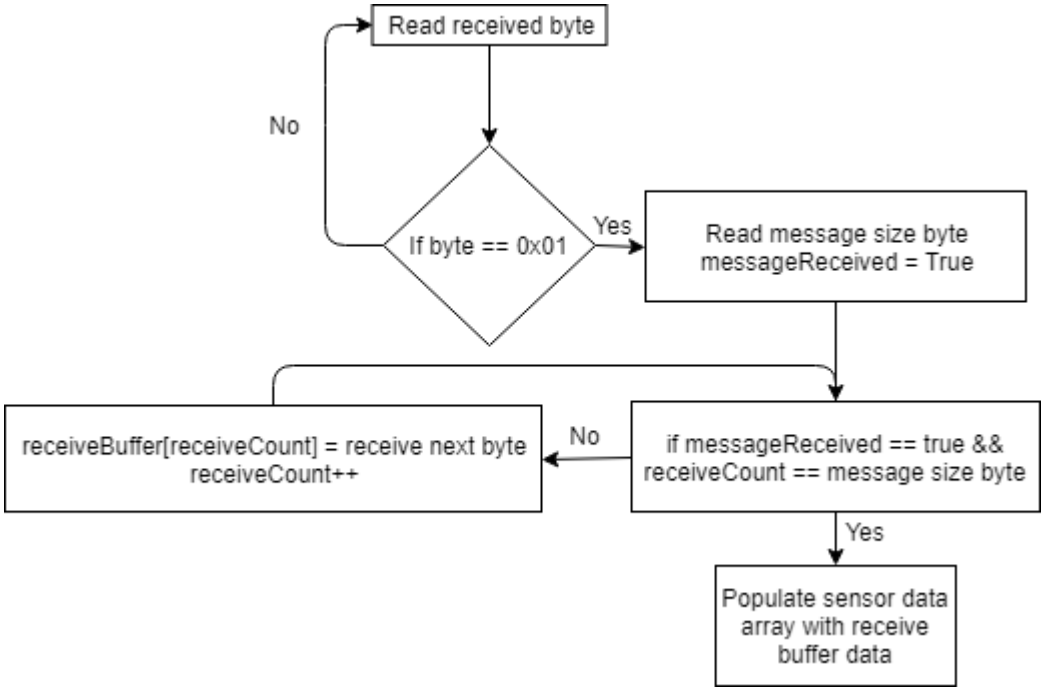

Figure 11.
Read data from USART buffer.

3.5.4 ADC and neural network reading

After the data has been received from the server the system changes state to transmission mode. The block diagram in figure 12 describes the general process before the transmission of the data to the server occurs.

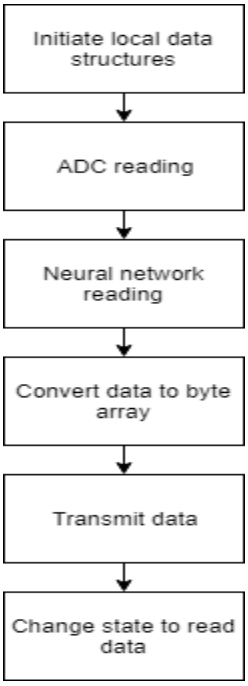

Figure 12.
PIC32 transmit process.

3.5.3.a. ADC reading

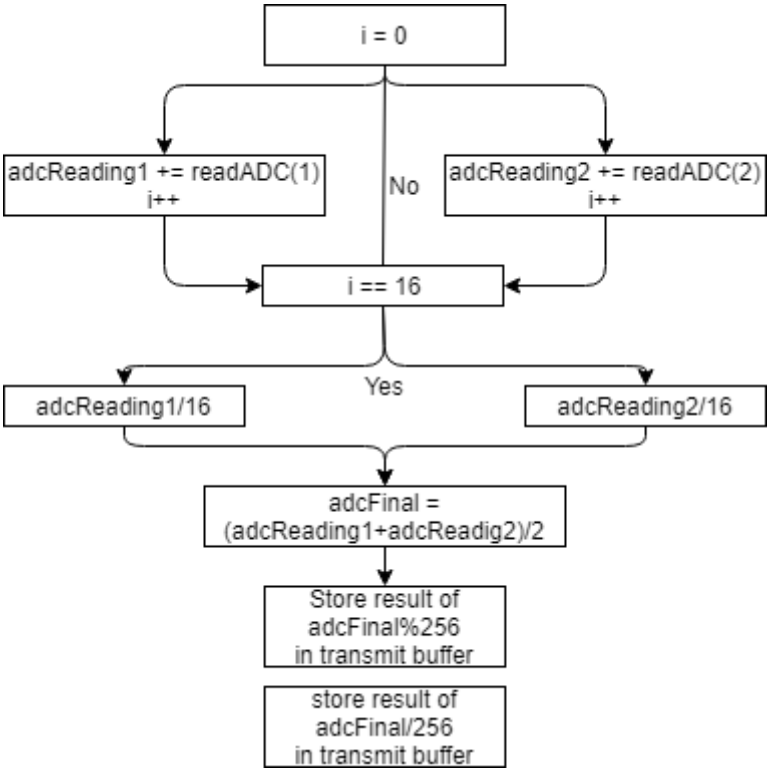

Figure 13.
ADC reading and averaging.

From figure 13 above it can be seen that two ADCs were used. The average reading over 16 cycles was used from both temperature sensors and then summed together and divided by two to get a final average reading. This was done to minimise the effect that small differences in the thermistor element could have due to the manufacturing process. The result, due to each byte in the transmit array only being able to handle integer values up to a maximum size of 256, is split into a low and high byte by first taking the modulus 256 of the original value and storing it as the low byte value then dividing the value by 256 and storing the result in a high byte.

3.5.4.b Neural network reading

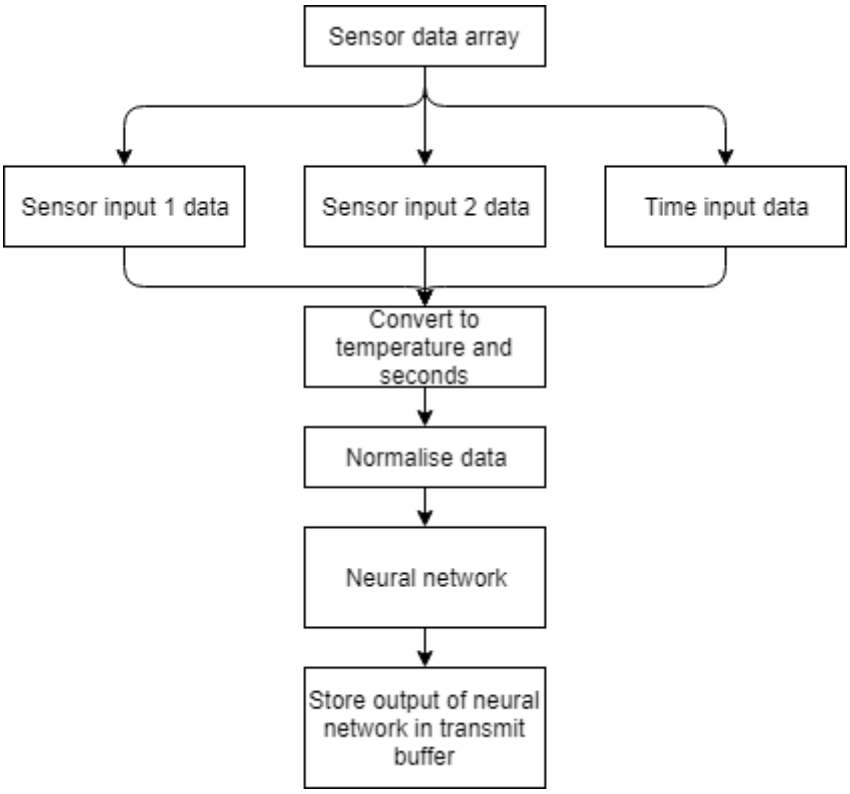

Figure 14.
Neural network reading.

Figure 14 above, shows the steps associated with extracting the sensor and time data received from the server then converting it to a format that can be normalised and used as input to the neural network. Conversion involves the usage of the Steinhart-Hart equation as

$$\frac{1}{T} = a + bln|R| + c[ln|R|]^3 \quad (6)$$

where:

- T is the temperature in Kelvin
- a, b and c are the Steinhart coefficients
- R is the resistance of the thermistor divided by the bias resistor $\frac{R_t}{R_0}$

By setting $c = 0$ and taking making $a = \frac{1}{T_0}$, where T_0 is the room temperature and setting $b = \frac{1}{B}$, where B is a Steinhart parameter found in the thermistor datasheet (**Part 5**) and is equal to 3800, the equation simply becomes

$$\frac{1}{T} = \frac{1}{T_0} + \frac{1}{3800} ln \left|\frac{R_t}{R_0}\right| \quad (7)$$

This is a less computationally complex adaptation of the Steinhart-Hart equation since only one logarithmic function needs to be calculated with minimal loss in accuracy. After the conversion is complete, the values are normalised using equation (2.1) to be used as inputs into the neural network. Once the neural network forward propagates the input values and produces an output, the output is stored in the transmit buffer.

3.5.5 Conversion to byte array and transmission

During experimentation it was found that the built-in C library function that could convert integers to character strings so that the USART could transmit the values introduced a significant overhead (12% program memory was added for every call to the function) to the program memory. A solution was found, figure 15, that reduced the overhead significantly and allowed the accurate transmission of the required characters by adding the '0' character to each integer. By hardcoding an integer-to-character conversion, the byte array to be transmitted was populated with the sensor data and neural network data then transmitted to the ESP8266 WiFi module after the send command was given. Transmitting messages is a two step process where the first step requires a message length be specified and the second step is transmitting the message to be sent. The commands required are listed in table 7.

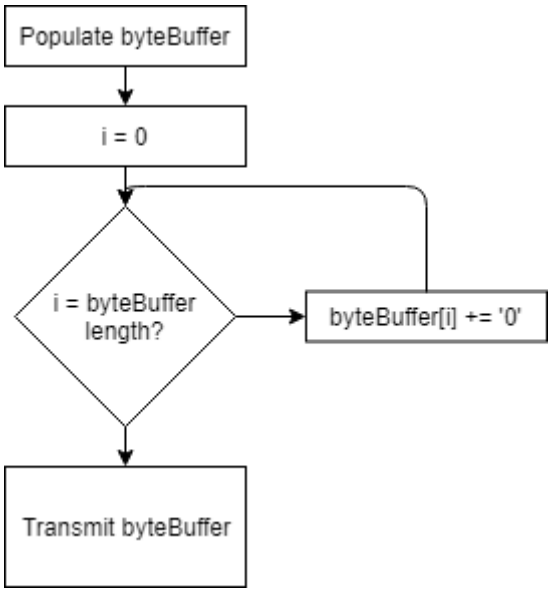

Figure 15.
Manual conversion of integers to character bytes.

Command	Function
AT+CIPSEND=<length of message>	Specifies the length of the message to be sent over WiFi to the server.
<message>	The message to be sent after CIPSEND command has been transmitted.

Table 7.
Required steps to send a message to the server from the microcontroller.

3.6 SERVER SOFTWARE

The software programmed in Python for the server contains all the data structures, GUI and algorithms regarding controlling the flow of the system as a whole. The VSs for all the sensor nodes in the network are contained in separate states as well as a WiFi re-identification system for the sensor nodes in cases where a sensor node re-establishes communication after being disconnected from the network. The architectural design of the server system is shown in the block diagram of figure 16.

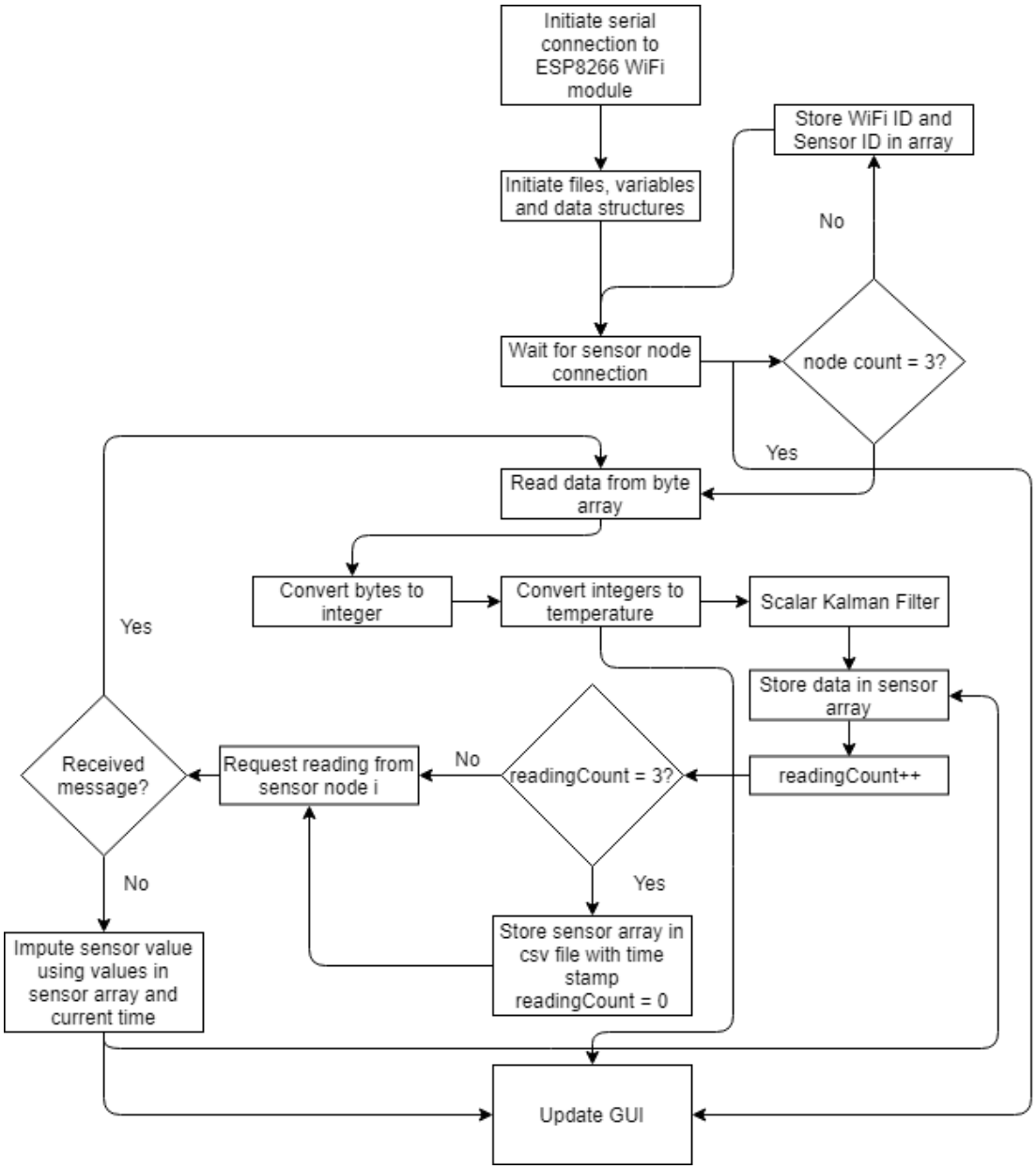

Figure 16.
Architecture design of the server.

3.6.1 Server initialisation

The pyserial library is used to communicate with the serial COM ports available through USB communication on the desktop PC. This is a requirement to transmit data through the USART communication channel to the connected ESP8266 WiFi module. The following settings were used to establish a working connection:

- Port: COM3
- Baud rate: 115200
- No parity bit
- No stop bits
- Byte size: 8 bits
- Timeout: 3 seconds

When the serial connection is established, the server is started using the commands listed in table 8. All variables and data structures, discussed in section **3.9 Data design**, are then created and initialised. The file to store the collected data is also created if it has not already been created. The block diagram of these steps is shown in figure 17.

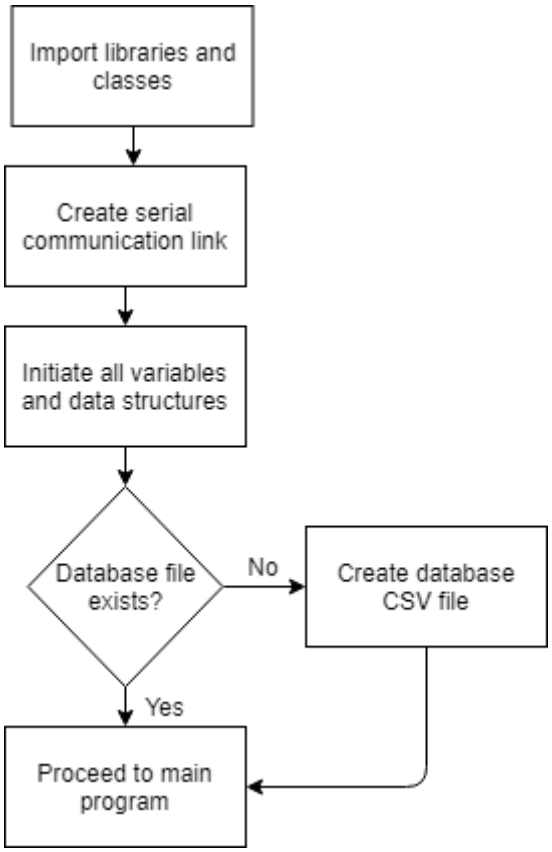

Figure 17.
Server initialisation procedure.

Command	Function
AT+CIPMUX=1	Allow multiple incoming connections (maximum 8)
AT+CIPSERVER=1,333	Start the server on port 333.

Table 8.
Required commands to start the server.

3.6.2 WiFi re-identification system

Each sensor node contains information that identifies itself with the server. The server ESP8266 WiFi module also automatically assigns a TCP client number to each sensor node that expires and is re-assigned if a sensor node disconnects from the network. A WiFi identification system was implemented to ensure that no sensor node would be mistaken for a different sensor node due to duplicate TCP client numbers that have been previously assigned but have not yet expired after a sensor node disconnects. If no changes are required the program proceeds as before. The block diagram of this function is shown in figure 18.

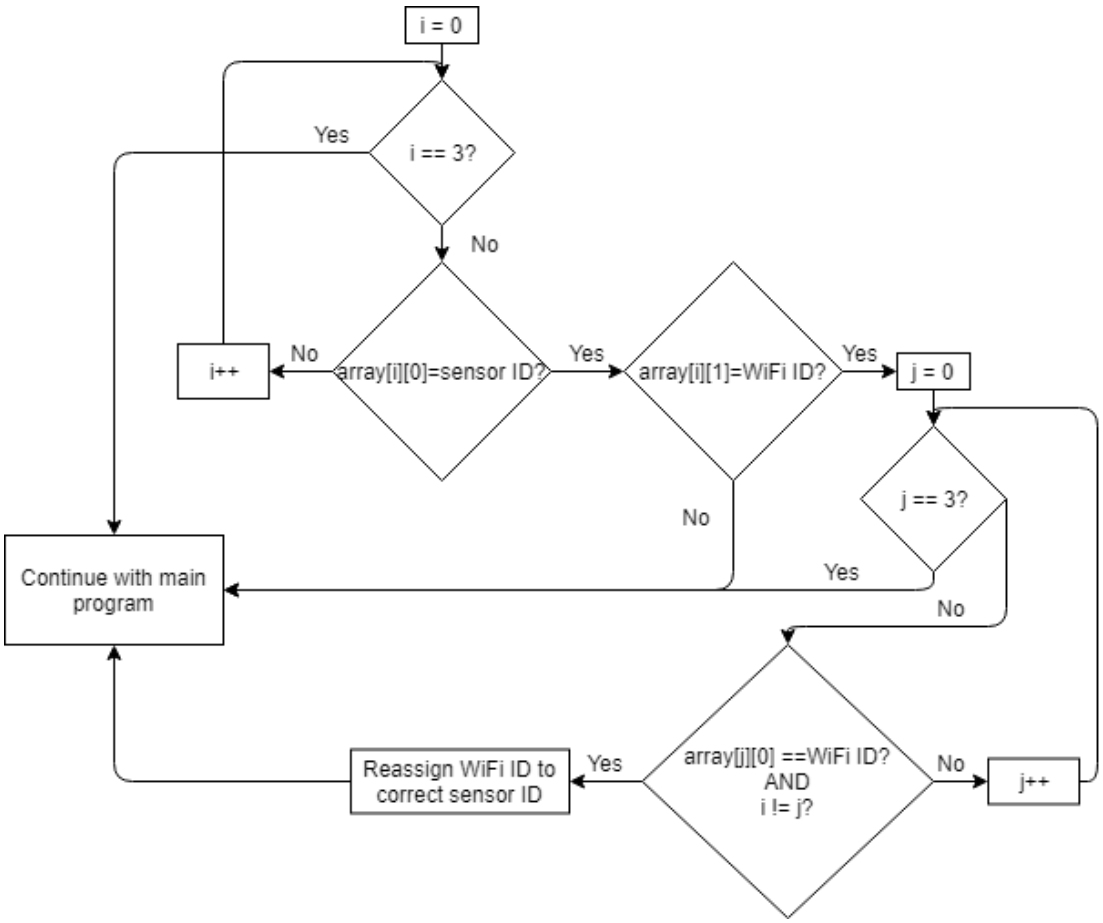

Figure 18.
Wifi re-identification procedure.

3.6.3 Sensor data filter

During experimentation it was found that there is a rare glitch in the system that occurs when a sensor node reconnects to the network where ground values are received instead of the expected sensor readings. A SKF was designed and implemented to filter out the glitches as well as to smooth out the received readings. The noise analysis of the SKF was done using gathered data and scaling up the noise covariance. Through observation from the simulations, the covariance noise value was determined and chosen to be 0.1 as minimal loss of data was experienced and the glitches were successfully filtered out. The flow diagram of how the SKF functions is shown in figure 19. The scalar Kalman filter consists of the following equations:

$$\text{Prediction: } S\left[\frac{n}{n-1}\right] = aS\left[\frac{n-1}{n-1}\right] \quad (8)$$

$$\text{Prediction error: } M\left[\frac{n}{n-1}\right] = \sigma_u^2 + a^2 M\left[\frac{n-1}{n-1}\right] \quad (9)$$

$$\text{Kalman gain: } K[n] = \frac{M\left[\frac{n}{n-1}\right]}{\sigma_w^2 + M\left[\frac{n}{n-1}\right]} \quad (10)$$

$$\text{Correction: } S\left[\frac{n}{n}\right] = S\left[\frac{n}{n-1}\right] + K[n]\left(x[n] - S\left[\frac{n}{n-1}\right]\right) \quad (11)$$

$$\text{Update error: } M\left[\frac{n}{n}\right] = (1 - K[n])M\left[\frac{n}{n-1}\right] \quad (12)$$

Where:

- $x[n]$ is the current sensor reading
- S[n] is the prediction
- M[n] is the error associated with the prediction
- K[n] is the Kalman gain associated with the prediction
- σ_u is the covariance noise in the system

The initial system state is assumed to be a room temperature of 25°C.

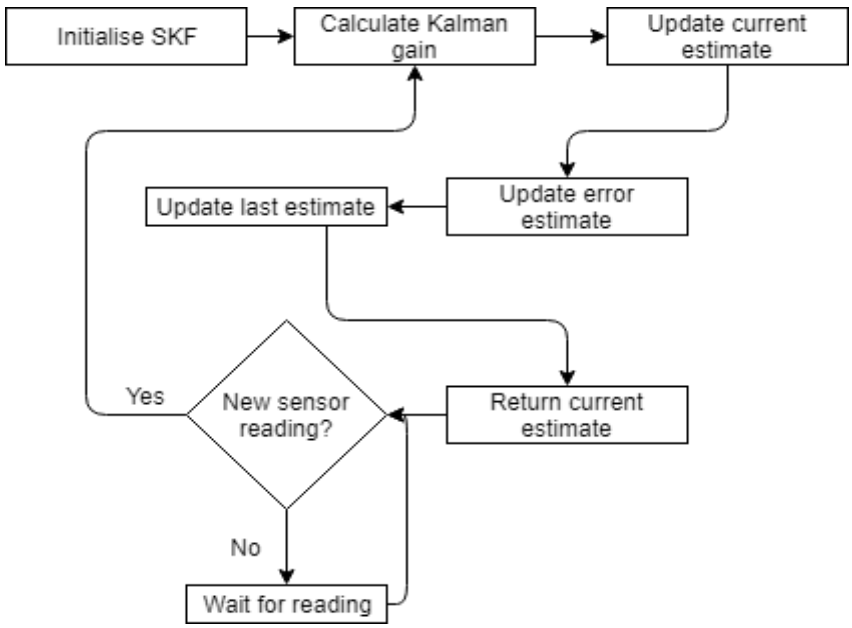

Figure 19.
Scalar Kalman filter function.

Simulations

The simulations to determine the covariance noise, or Q, are shown in figure 20 to figure 23. Two glitches are visible in the plotted data.

- Aim: The aim of this experiment is to demonstrate the performance of the SKF with different Q values.
- Data size: 5000 individual sensor readings
- Kalman filter parameters: R = 25, Q = 0.001, 0.01, 0.1, 1.0
- Platform: Python.

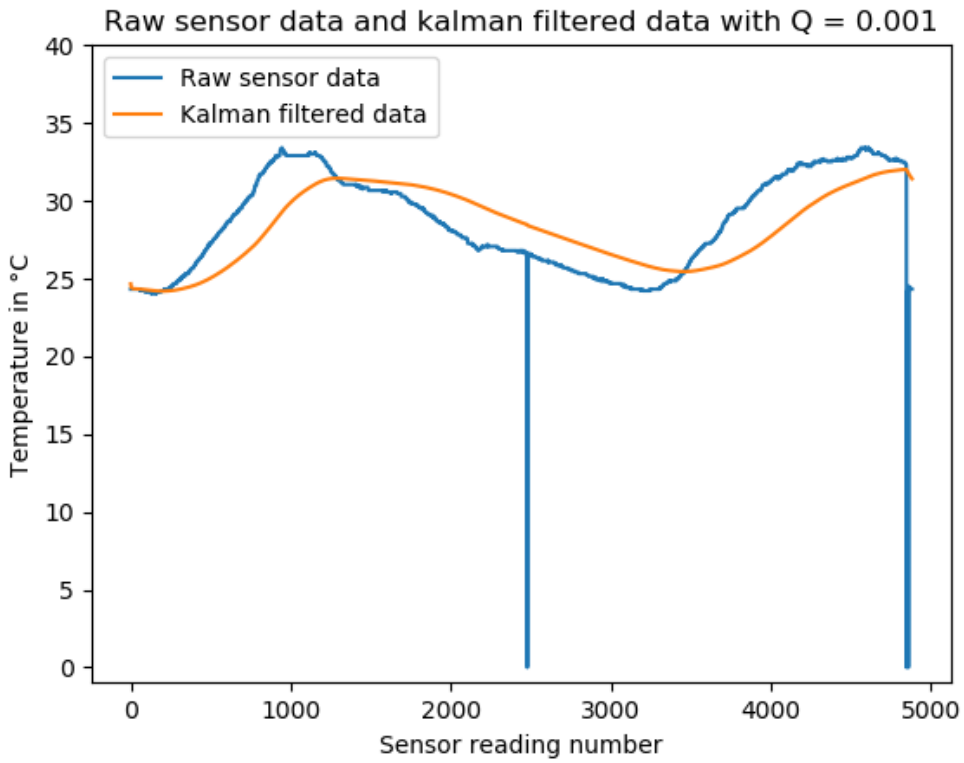

Figure 20.
Scalar Kalman filter with Q = 0.001.

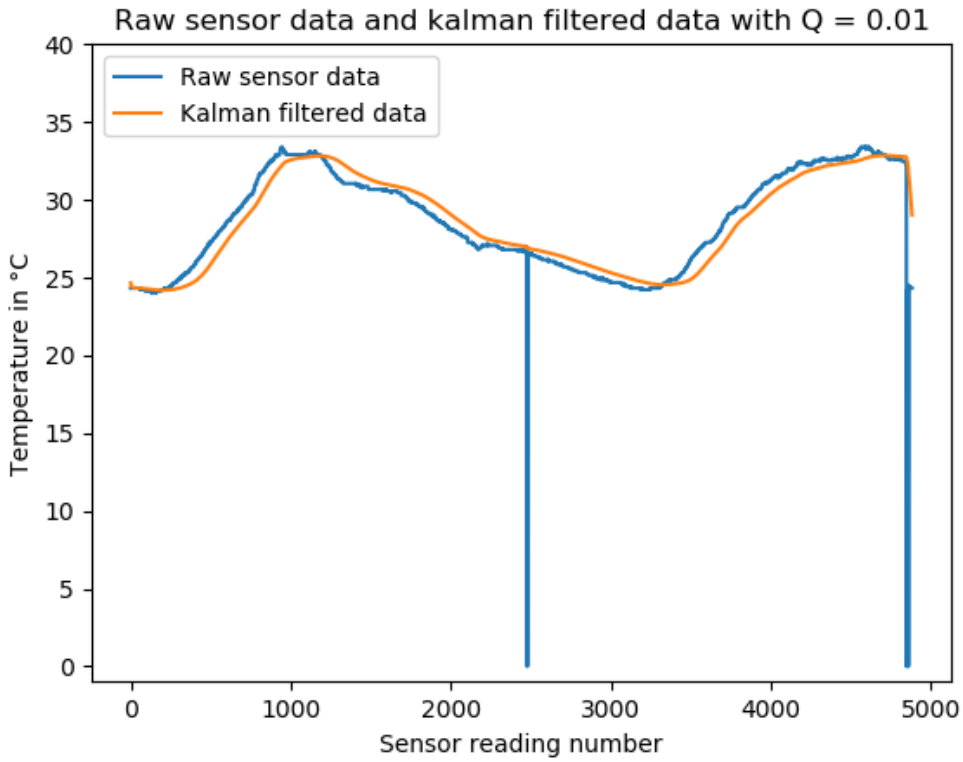

Figure 21.
Scalar Kalman filter with Q = 0.01

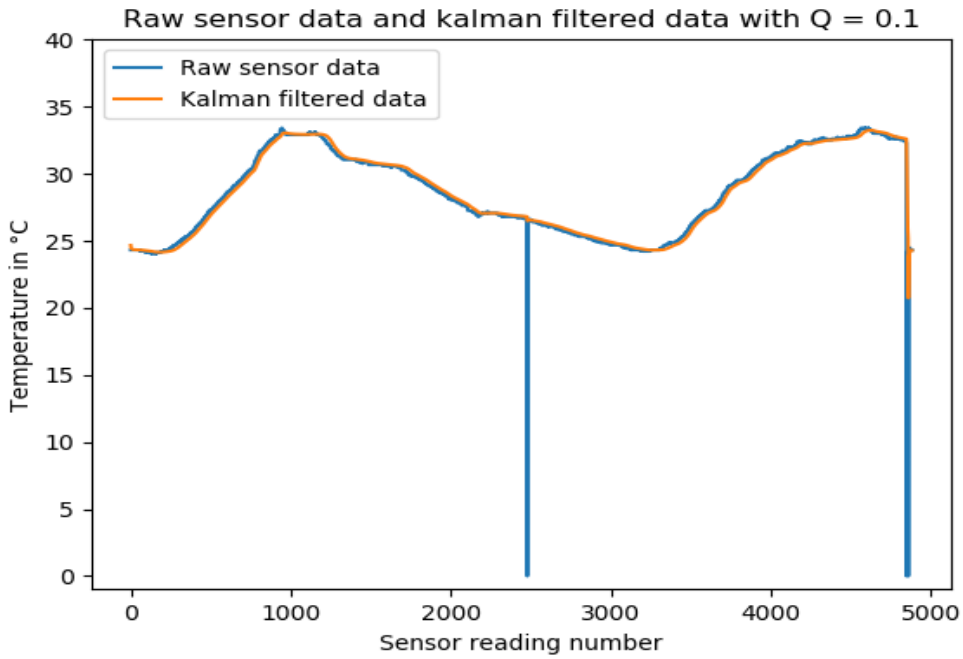

Figure 22.
Scalar Kalman filter with Q = 0.1

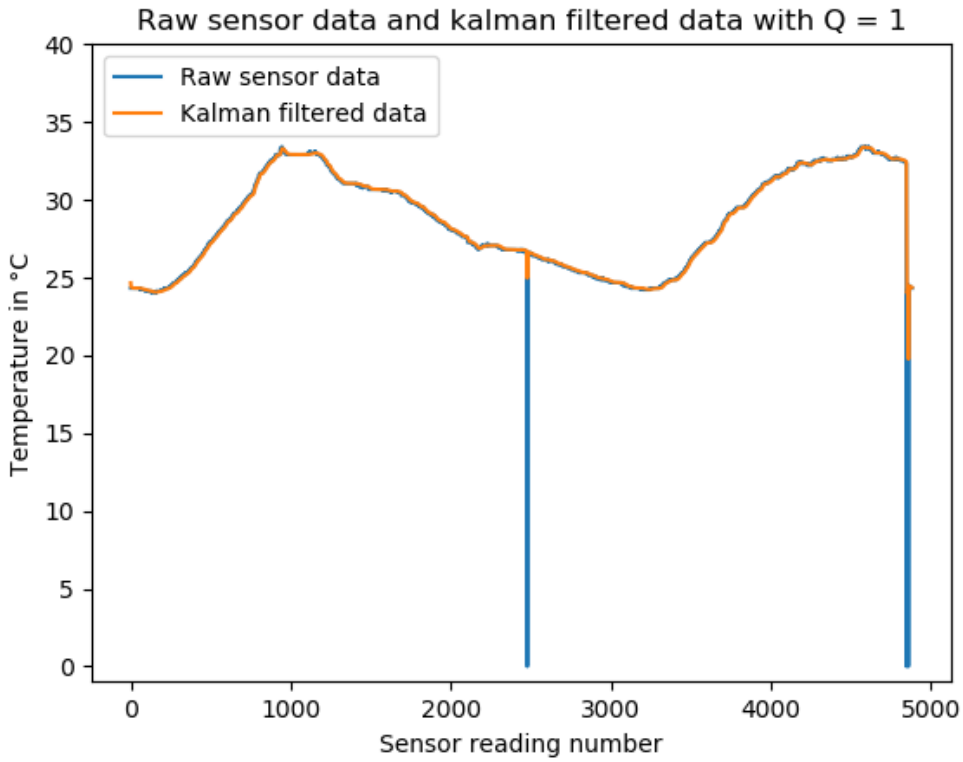

Figure 23.
Scalar Kalman filter with Q = 1.0

The covariance error that is decided upon is a Q value of 0.01 as the overall shape of the data does not vary as much from the origin as with the Q value of 0.001 while being able to filter out the glitches more effectively than with a Q value of 0.1 while smoothing the incoming data. A Q value of 1.0 was deemed not effective in filtering out the glitches.

3.7 IMPUTATION WITH NEURAL NETWORK

The MLP neural network is the core feature required to implement the VSs in the WSN. The design and implementation of the chosen training algorithm, neural network as well as the simulations to choose an optimal MLP topology in terms of training time and accuracy are described in this section.

3.7.1 Architectural design

The MLP class design is illustrated below in figure 24. Each layer, including all hidden layers, is constructed individually until the output layer is reached. Once the neural network is constructed with random weights applied, it is returned to the program that called the neural network constructor. It takes two parameters to determine the hidden topology of the neural network which will always have 3 input nodes and 1 output node:

- The number of hidden nodes

- The number of hidden layers

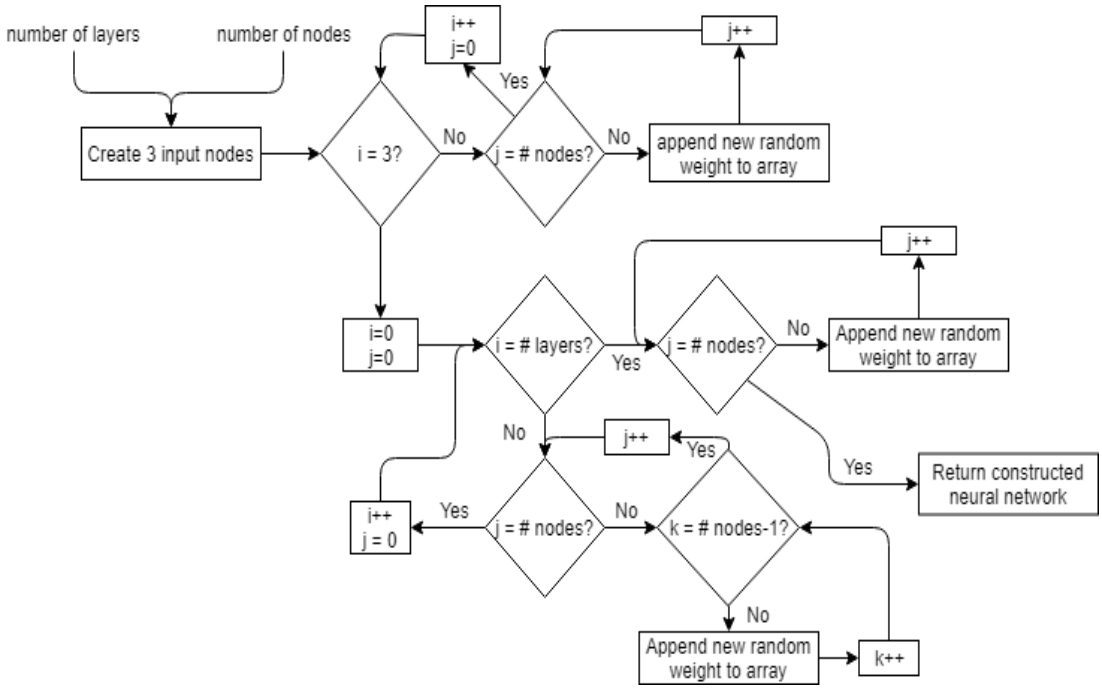

Figure 24. **Neural network construction block diagram.**

3.7.2 Simulation

The training loss versus the number of hidden nodes in the hidden layer over 20 iterations to find the optimal topology was simulated and plotted in figure 25.

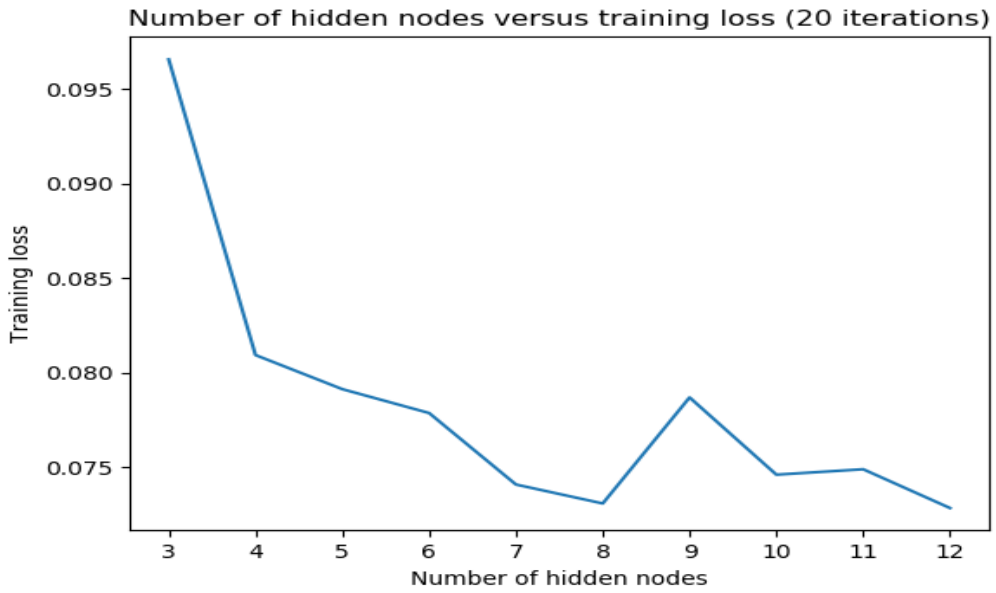

Figure 25.
Training loss against the number of hidden nodes after 20 epochs.

3.7.3 Mathematical description

A topology of 3 input nodes, 11 hidden nodes plus bias node and one output node was chosen after simulations, figure 25, showed that this topology would train the fastest while not overfitting the data while not being too computationally expensive.

The output can be described mathematically as the sum and products of the inputs, weights and activation functions. The mathematical model is described as:

For each node in the input layer:

$$a_{1,i} = \frac{x_{1,i}}{1+|x_{1,i}|} \tag{13}$$

where:

- $a_{1,i}$ is the resulting output after an input has been put through the softmax activation function.
- $x_{1,i}$ is the normalised input to the input node.

For each node in the hidden layer:

$$x_{2,i} = \sum_{i=1}^{N}\sum_{j=1}^{3} w_{i,j}a_{1,i} \tag{14}$$

And

$$a_{2,i} = \frac{x_{2,i}}{1+|x_{2,i}|} \tag{15}$$

where:

- $a_{2,i}$ is the resulting output after the inputs have been summed and multiplied with the weights then put through the softmax activation function.
- $x_{2,i}$ is the result of the sums of the inputs multiplied by the weights.
- $w_{i,j}$ is the weight associated with the input.
- $a_{2,i}$ is the resulting sum.
- N is the number of hidden nodes but not including the bias node if it is the first hidden layer.

For the output node in the output layer a linear activation is used and thus:

$$a_3 = \sum_{i=1}^{N} w_i a_{2,i} \tag{16}$$

where:

- a_3 is the resulting output after the inputs have been multiplied with the weights and summed together. N is the number of hidden nodes in the previous layer.
- w_i is the weight associated with the input.
- N is the number of nodes in the previous layer including the bias node.

In addition to the above, the input layer and every hidden layer has a bias node included that is not connected to the previous layer. This is implemented to increase the flexibility of the model to fit the data and to allow the network to fit data if in the unlikely scenario all the input features are equal to 0. The output from the bias node will always be equal to 1.0.

3.7.4 Training design

The genetic algorithm designed and implemented for neuro-evolution of the MLP is described in figure 26. There are four main steps in the genetic algorithm, namely the forward propagation, prediction, error calculation, parent selection and finally repopulation which are described in more detail in this section.

Figure 26.
Genetic algorithm training block diagram.

The input parameters of the training algorithm are

- The number of MLPs to initiate in the population
- The number of hidden nodes per hidden layer in each MLP
- The number of hidden layers per MLP
- The number of training epochs
- The sensor dataset that must be used

The dataset used for training the MLP is previously collected sensor data from the implemented WSN. The dataset is randomly split into 60% training data with targets and 40% test data with associated targets using the algorithm described in the block diagram of figure 27.

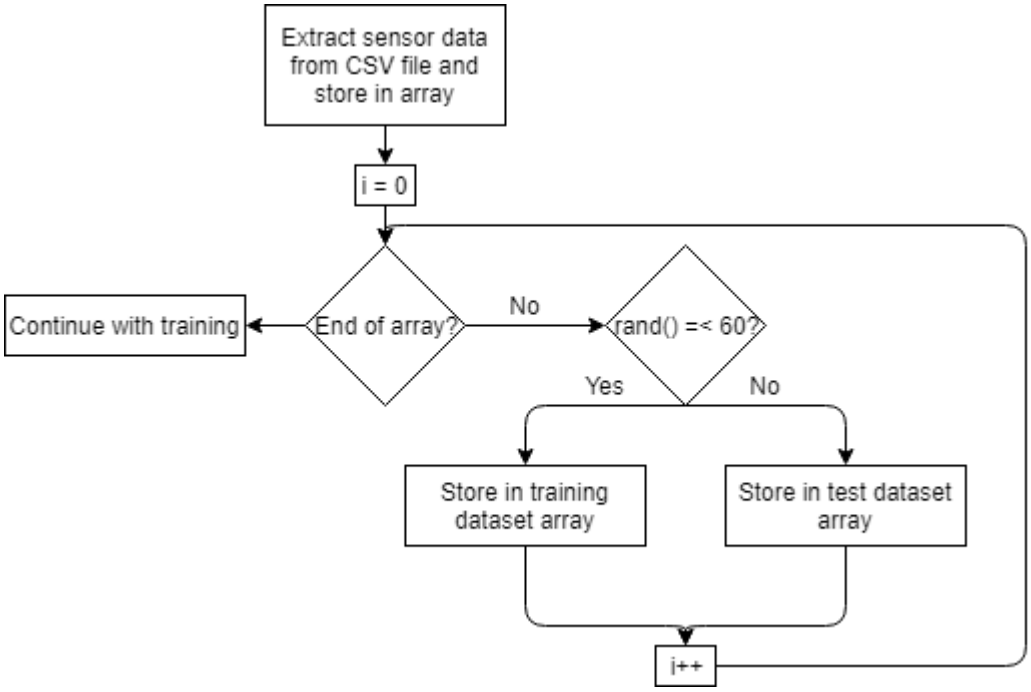

Figure 27.
Splitting data into training and test data.

3.7.4.a Prediction

The prediction of a value involves an initial forward propagation described by figure 29 and equations 13-16.

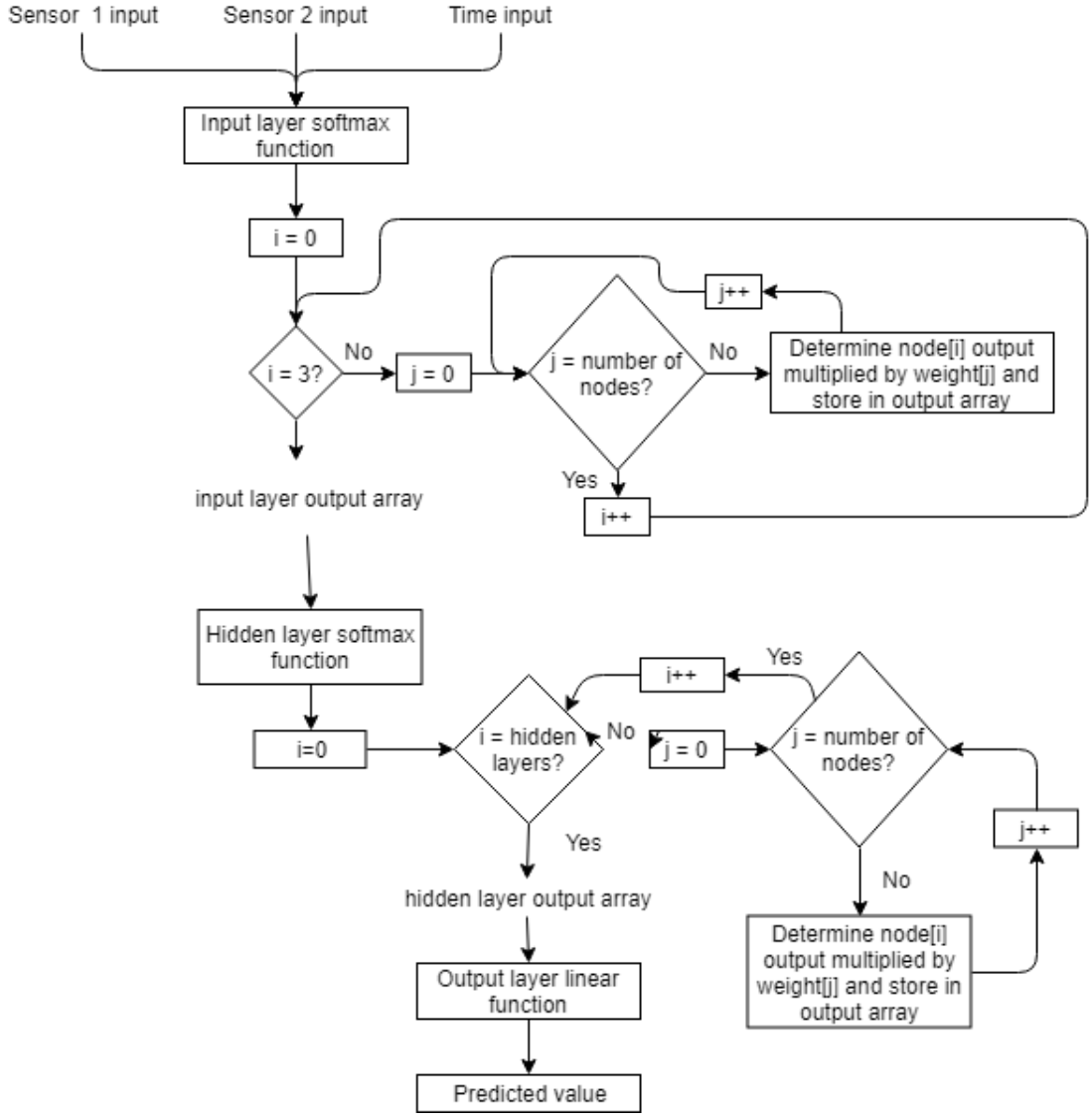

Figure 28.
Prediction block diagram.

Each prediction is based on three input parameters which are the other two sensor node inputs in the WSN and the time that those data points were collected during the day. The prediction value aims to emulate the value that the sensor it is associated with in the WSN would sense. The block diagram describing the process is shown in figure 28.

3.7.4.b Forward propagation

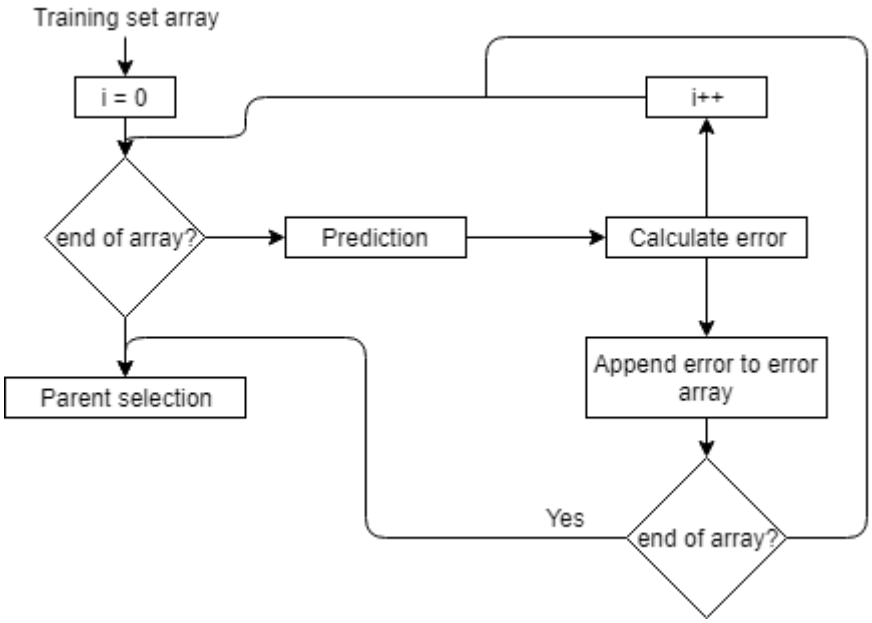

Figure 29.
Forward propagation block diagram.

3.7.4.c Error calculation

The error of each MLP is calculated using the mean absolute error (MAE) as

$$MAE = \frac{\sum_{i=1}^{N} |p_i - t_i|}{N} \quad (17)$$

where:

- MAE is the mean absolute error of an individual MLP
- N is the amount of training points
- p_i is the predicted value by the MLP
- t_i is the target value associated with the inputs to the MLP

MSE was an alternative error function that was considered. However, the MAE is robust to outlier values as it does not make use of the square as in the MSE. Furthermore, outlier values are not expected in the training dataset.

3.7.4.d Selection

The selection of parent MLPs is done immediately after the last MLP in the population has had its MAE calculated after forward propagation of the entire training dataset is complete. The top 10% of the population is chosen to breed new MLPs. The block diagram description of the selection process is shown in figure 30.

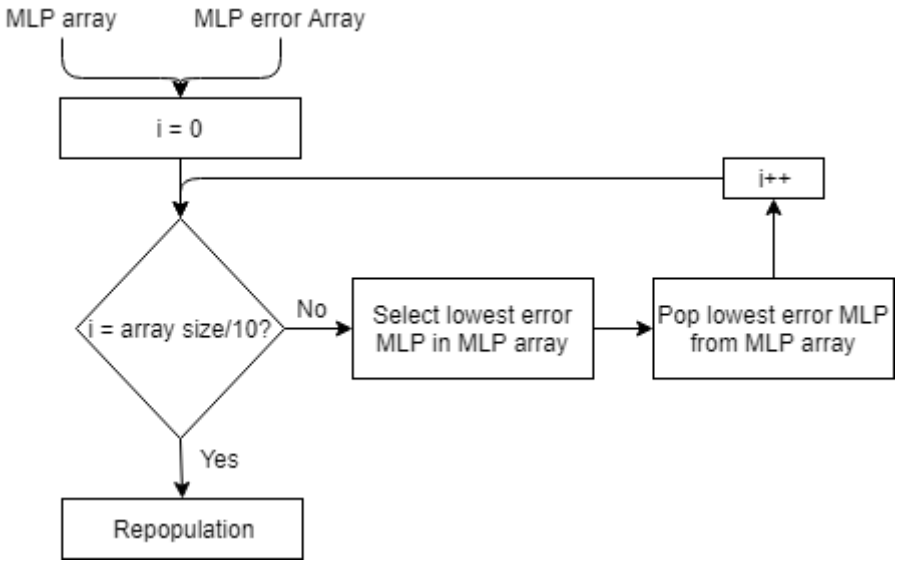

Figure 30.
Parent selection block diagram.

3.7.4.e Repopulation

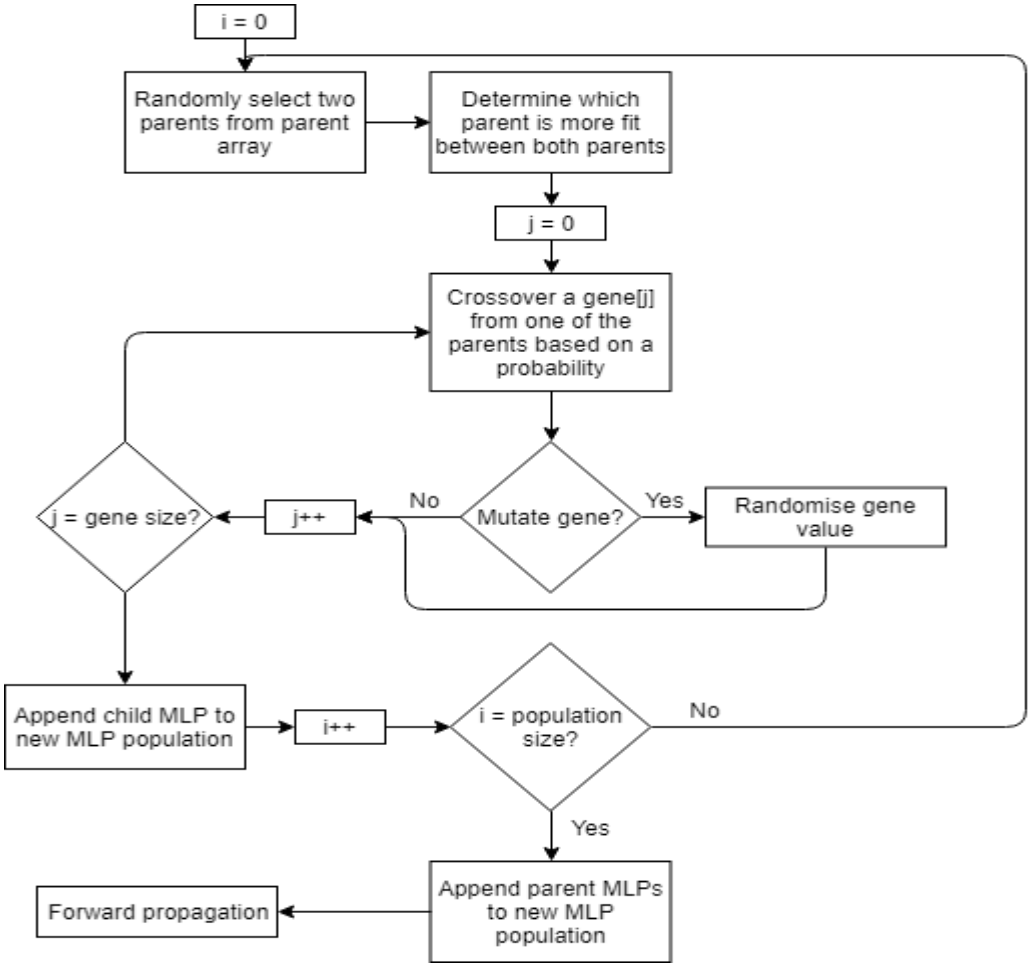

Figure 31.
Repopulation using most fit parents block diagram.

With the top 10% parent MLPs selected, breeding of child MLPs commences. For each child, two parents are randomly selected from the parent pool. The probability that a gene is copied over to the child MLP is calculated using the error associated with both parents using the probability equation

$$P(A) = 1 - \frac{MAE_1}{MAE_1 + MAE_2} \tag{18}$$

where MSE_1 is the error associated with the first parent, MSE_2 is the error associated with the second parent and $P(A)$ is the probability of the first parent copying over a gene from its gene array to the child MLP gene array. Accordingly, the probability for the second parent to copy over a gene is

$$P(B) = 1 - P(A) \tag{19}$$

which allows the fitter parent a higher likelihood to copy over weights that are more desirable.

To avoid stagnation in the population as the training epochs increase, every gene, when copied, has a small probability (<=2%), to be completely randomised instead of copied over from the parent MLPs. This ensures that there is always some diversity in the genetic pool of the MLP population.

Once the old population size has been reached by repopulating the new population with child MLPs, the process is repeated beginning with the forward propagation step until the specified epoch number has been reached. The block diagram describing the process is shown in figure 31.

3.7.4.f Training error simulations

The training loss was simulated over 500 epochs for each sensor node, figure 32-34.

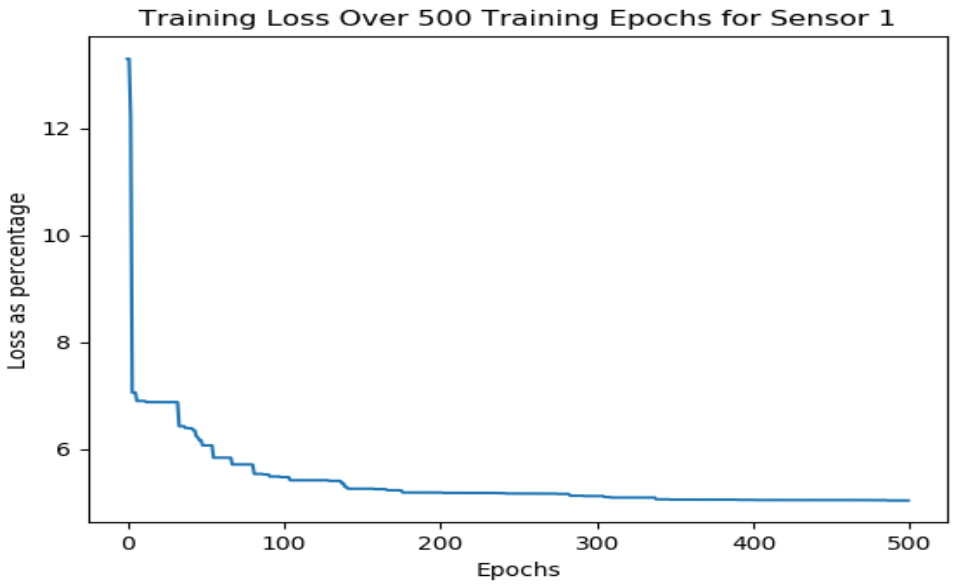

Figure 32.
Training loss of the MLP over 500 epochs for sensor 1.

Figure 33.
Training loss of the MLP over 500 epochs for sensor 2.

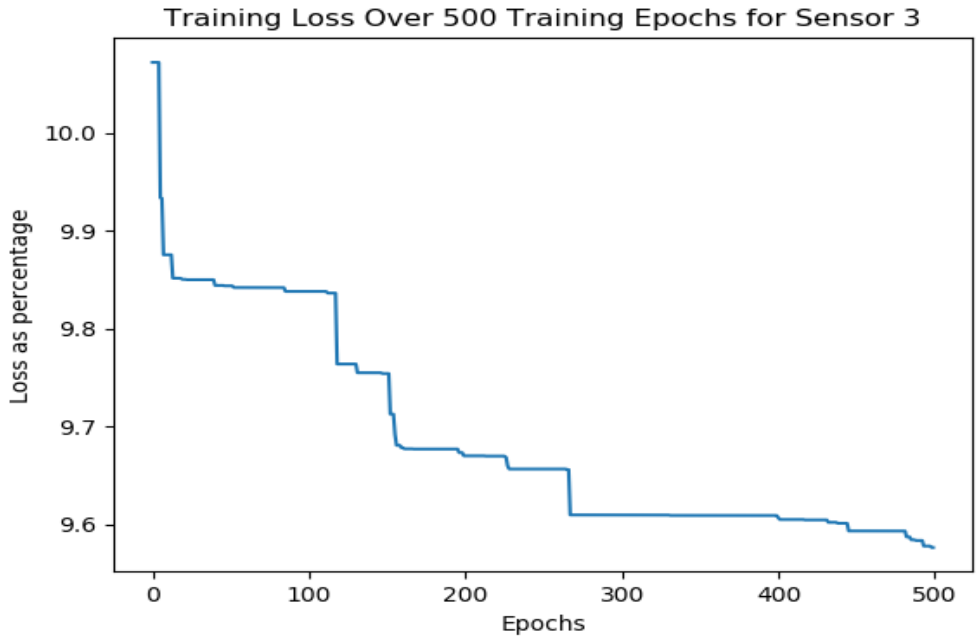

Figure 34.
Training loss of the MLP over 500 epochs for sensor 3.

3.8 DATA DESIGN

3.8.1 Microcontroller data internal software data structure

One dimensional data of type int, char and float are passed internally between components. i.e. the WriteString() method outputs each character to the USART interface which is used by the ESP8266 WiFi module as serial input.

3.8.2 Microcontroller global data structure

The global data structures comprise of structs and one-dimensional char arrays i.e. the data associated with the sensors and program control structure is stored in a struct called app.data and stores multiple data types including but not limited to int, float and char as well as multiple one-dimensional arrays of type int, float and char.

3.8.3 Microcontroller temporary data structure

One-dimensional float arrays are used as temporary data structures to store the outputs of the neural network through each layer as well as to keep track of the current weight being multiplied with the outputs. Two integers are used as iterators to process the neural network.

3.8.4 Database description

A database is used to store all the acquired sensor readings locally on a desktop PCs storage. The items stored are the date, enabling iteration with a for-loop to access more specific data gathered for a specific day. The sensor data and time stamps are stored in a CSV file whereby each entry corresponds to the time that the sensor readings were stored. i.e. an entry at timestamp 12:04:32 corresponds with sensor values that were taken at this time. Below in figure 35 a graphical representation of the database is provided.

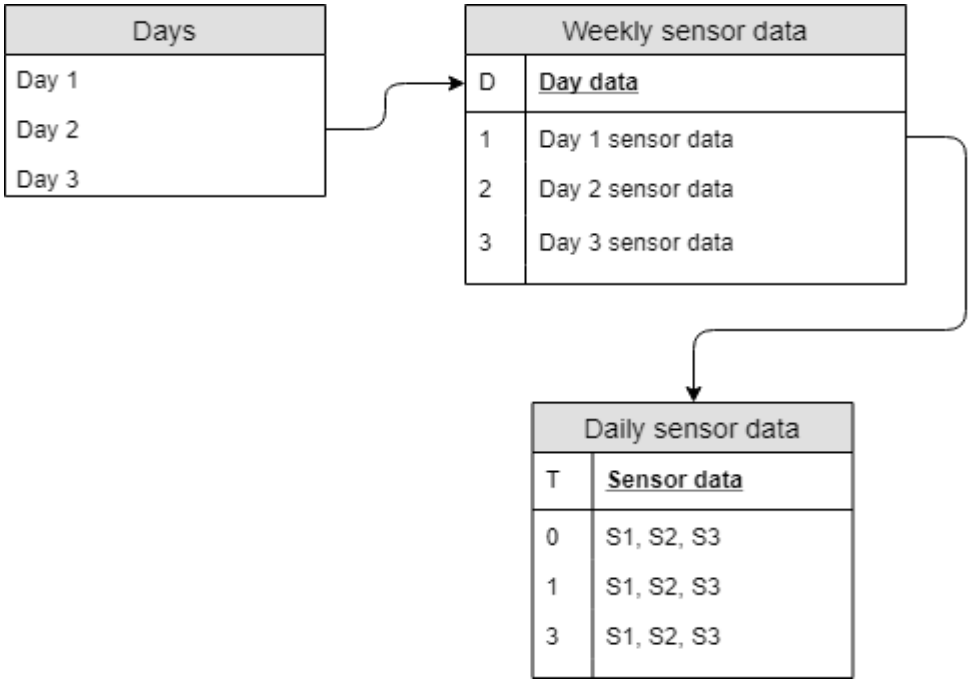

Figure 35.
Database description block diagram.

The database is comprised of all collected data and is overwritten on a weekly basis to store the most relevant data for weekly retraining of the neural network which is done to counteract the effect of changing weather patterns throughout the year.

3.9 ARCHITECTURAL AND COMPONENT-LEVEL DESIGN

3.9.1 Program structure

The program structure is modular and consists of inputs, outputs, an indexed data structure, pipelines, state machines, procedural structures, algorithmic structures with a combination of sequential and iterative conditional structures.

The VS project's architecture consists of processing pipelines in the case of the server and state machines in the case of the microcontroller software. In some cases the pipelines may be connected to one another. Ultimately, each pipeline is connected directly or indirectly to the GUI for display purposes. The pipelines used are listed below.

- Reading request pipeline: Consists of the server sending a query, with data from the other sensor nodes, to an individual sensor node.
- Data handling pipeline: Consists of the server handling the incoming data from a sensor node by applying the values received to internal data structures for further computation.
- Conversion pipeline: Consists of the server converting the raw sensor data to usable data:
- VS pipeline: Consists of the VS on the server when a sensor node is not responding to requests.
- Storage pipeline: Storing the received readings in a database

The state machine states that are used in the microcontroller are listed below.

- Initialisation state: Consists of initialising all data structures and initial connection to the server.
- Reading state: Consists of taking ADC and neural network readings.
- Transmitting state: Consists of transmitting the acquired data back to the server.
- Waiting state: Consists of waiting for a new query from the server.

3.9.2 Architectural diagram

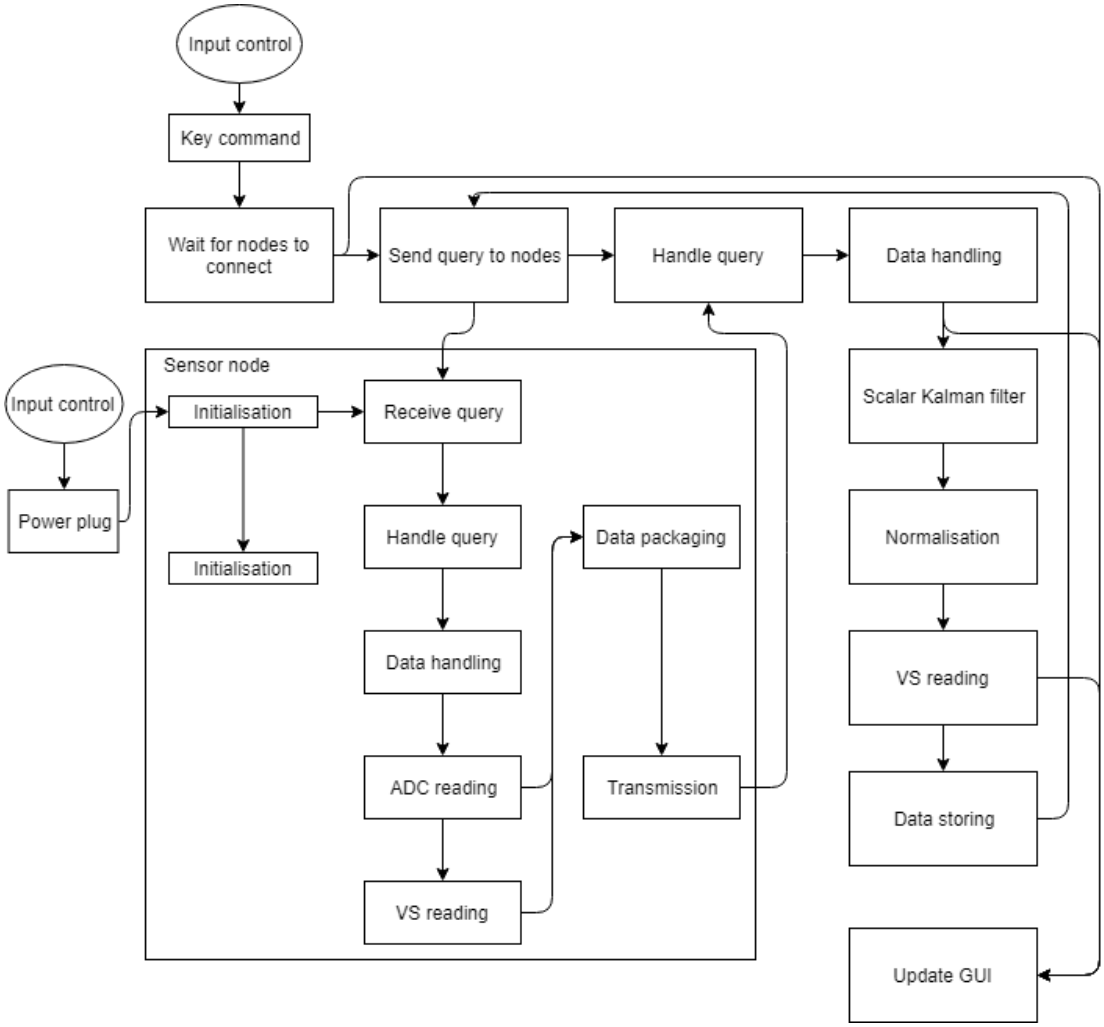

Figure 36.
Architecture block diagram.

The software architecture block diagram is depicted in figure 36. It is structured into input, output, control and process blocks. Some functional blocks are interconnected.

3.9.2.a. User interface – Control inputs

The user interface comprises of no control inputs. The single input control required by the user is to simply start the program and to plug in the sensor node to a power source. Once this task is completed, the system is automated to assume full control of the program.

3.9.2.b. Control and process functions

Each sensor node is individually queried for a sensor reading as well as given the readings from the other nodes. The data is handled and conditioned for the VS as well as the readings from the ADC. The data is then packaged for transmission and transmitted back to the server. The server handles the data and applies the incoming data to the relevant data structures. The ADC reading from the sensor node is filtered used the SKF then normalised. If a sensor node was disconnected, a virtual sensor on the server imputes the relevant sensors value. The data is stored in the update. Throughout these processes, the GUI is updated to reflect the changes occurring in the system.

3.10 DESCRIPTION OF COMPONENTS

3.10.1 Class relationship diagram

3.10.1.a. Microcontroller relationship diagram

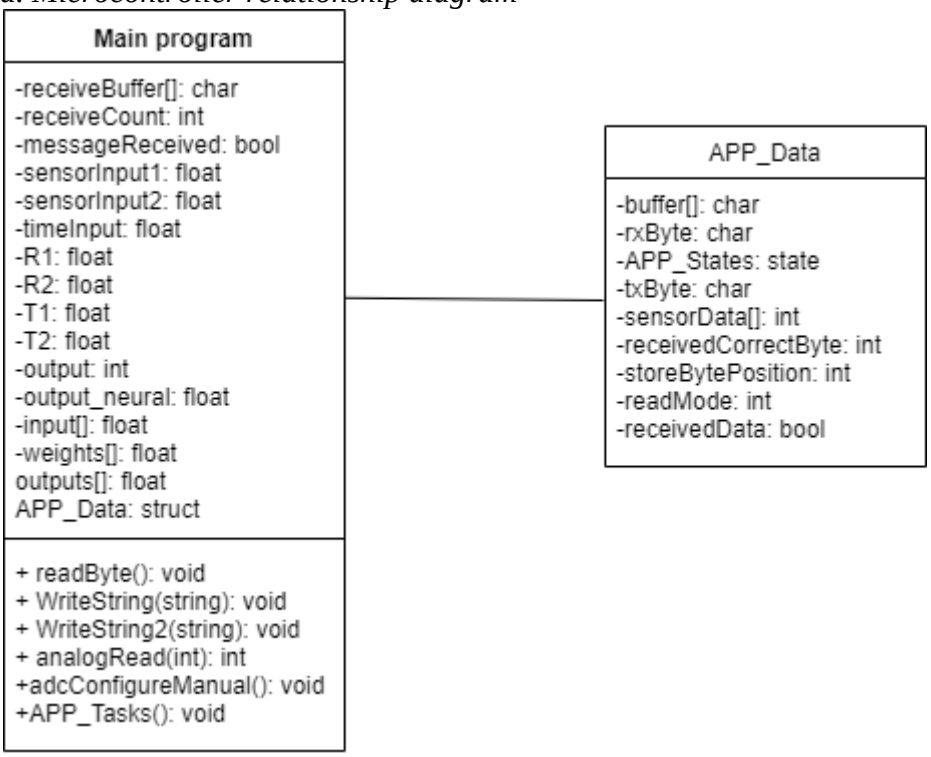

Figure 37.
Microcontroller class relationship diagram.

3.10.1.b. Server relationship diagram

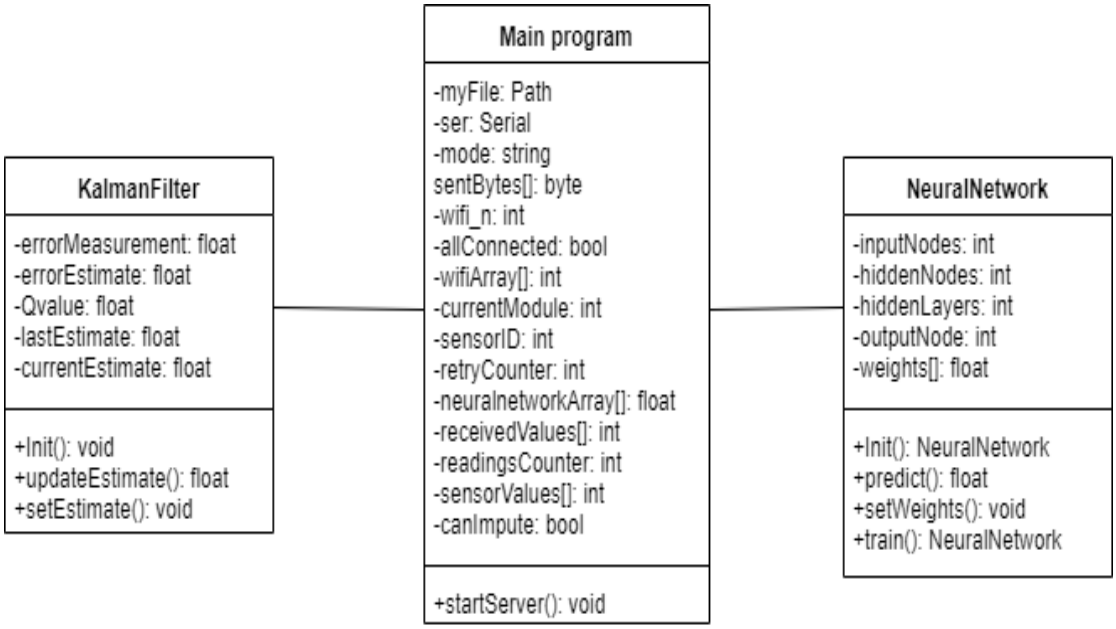

Figure 38.
Server class relationship diagram.

3.10.2 Description for class main program of microcontroller.

Class name: Main program	Method description
	A method to initialise, create and make available for use peripherals and internal data structures such as the neural network, USART communication and ADC.
	Processing Narrative (PSPEC)
	The object is responsible for keeping the program flow in order by ensuring certain actions can only be executed in the applicable state machine.
	Program Description Language

	int main (0){
	//Initialisation of state machine
	//Initialisation of variables and data structures
	//Initial connection to server
	//Handling receive interrupts on USART
	//Handling incoming data from USART
	//Reading from ADC
	//Executing neural network
	//Packaging data for transmission
	//Transmission
	//Changing state
	}

	Restrictions and/or limitations
	The main program is self-contained and has no practical restrictions.
	Local data structures
	The following local data structures are used: • One-dimensional array of floats • One-dimensional array of chars • One-dimensional array of ints • C struct
	Performance issues
	Performance is limited due to the available processing power and lack of multi-threading available for parallel processing in the microcontroller.

Table 9.
Description for main program of microcontroller part one.

Class name: Main program	**Design constraints**
	The class is constrained by the limitation of having only one instance per sensor node. The design is constrained by a single query from the server that must
	Interface description
	Main program interfaces with the following components: • readByte() • WriteString() • WriteString2() • analogRead() adcConfigureManual()

Table 9.
Description for main program of microcontroller part two.

3.10.3 Description for class APP_Data

Class name: APP_Data	Method description
	A method to keep track of the state of the state machine and various parameters of the internal data structures.
	Processing narrative (PSPEC)
	The object is responsible for holding the values for: • Received sensor data from the server • The current state of the state machine
	Program description language

	typedef enum{
	//list of states
	} APP_States
	
	typedef struct{
	//variables
	} APP_DATA

	Restrictions and or limitations
	The class can only keep track of states and data structures that have been assigned at initialisation. The class has no functions associated with it which means manual configuration of the internal variables is required.
	Local data structures
	The following local data structures are used: • One-dimensional array of ints
	Performance issues
	The microcontroller needs to access the APP_Data struct before accessing further variables which very slightly increases the amount of processing required.
	Design constraints
	Any class that needs to access the variables within the struct needs to have a copy instantiated in the header file.
	Interface description
	APP_Data does not interface.

Table 10.
Description for APP_Data.

Class name: Main program	**Method description**
	A method to initialise, create and make available for use objects such as the VS, SKF, controlling program flow with regards to connected sensor nodes and to identify if a sensor node is not connected to the network.
	Processing narrative (PSPEC)
	When a user starts the program a prompt asks for the sensor nodes to be turned on. The class is responsible for transmitting data throughout the WSN as well as storing all received data. The class is also responsible for identifying if a sensor node in the network has failed so that a VS can take over operations for that sensor node.
	Program description language

	int main(){
	//import required libraries
	//initialise serial connection
	//initialise data structures and variables
	//wait for incoming connections
	//Transmit requests to sensor nodes
	//Receive data from sensor nodes
	//Data handling
	//Filtering
	//Storing
	//Identifying if a sensor node is disconnected
	//Activating VS on server.
	//Display data to GUI
	}

	Restrictions and/or limitations
	The serial COM port must be specified beforehand.

Table 11.
Description of main program on server part one.

Class name: Main program	Restrictions and/or limitations
	The program can only run with a minimum of 2 sensor nodes connected and maximum 3 sensor nodes connected to the network.
	Local data structures
	The following local data structures are used: • One-dimensional array of ints • One-dimensional array of floats • Python lists
	Performance issues
	Due to the sequential nature of the program flow, the application can not make use of parallel processing in the server's multicore CPU.
	Design constraints
	If the COM port is already in use, the program can not run. The class is limited in that only one sensor nodes data can be handled at any time.
	Interface description
	Main program interfaces with the following components: • KalmanFilter() • NeuralNetwork()

Table 11.
Description of main program on server part two.

3.10.4 Description for class Kalman filter

Class name: KalmanFilter	Method description
	A method to filter acquired sensor values.
	Processing narrative (PSPEC)
	The object is responsible for calling the following functions: • updateEstimate()
	Program description language

	Class KalmanFilter{
	//updateEstimate updates the current estimate
	}

Table 12.
Description of the Kalman filter class part one.

Class name: KalmanFilter	Restrictions and/or limitations
	The Q value and mean initial value need to be found prior to initialising the class which may be time consuming. The Kalman filter can only be used for temperature estimates.
	Local data structures
	Individual variables are used.
	Performance issues
	No significant performance issues were recognised with the class.
	Design constraints
	The class is constrained by only handling one estimate per value provided. The initial estimate needs to be processed before the next estimate can be processed.
	Interface description
	Main program()

Table 12.
Description of the Kalman filter class part two.

3.10.5 Description for class Neural Network

Class name: NeuralNetwork	Method description
	A method to impute values using a neural network given three inputs.
	Processing narrative (PSPEC)
	The object is responsible for calling the following functions: • Initalisation() • setWeights() • predict() • train()
	Program description language

	Class NeuralNetwork{
	//Initialisation method initialises the network
	//setWeights method sets the weights of the neural network
	//Predict method returns a prediction given three inputs
	//Train method begins the training process given the correct parameters.
	}

	Restrictions and/or limitations
	A maximum time of 7 seconds is given for each imputation.

Table 13. Description of the neural network class part one.

Class name: NeuralNetwork	Local data structures
	One-dimensional array of type float are used Variables of type integer and float are used.
	Performance issues
	Due to the sequential implementation, parallel processing can not be used for this class in both prediction and training.
	Design constraints
	The class is constrained by requiring three normalised inputs. The weights must be found using the training method before implementing in the main system.
	Interface description
	Main program()

Table 13.
Description of the neural network class part two.

3.11 SOFTWARE INTERFACE DESCRIPTION

The user interface has been designed around the display of a desktop monitor. Several labels were placed in rows and columns to inform the user of the data that is displayed on the screen.

3.12 USER INTERFACE DESIGN

The user interface will consist of rows and columns for each different sensor node as well as each neural network. A user will intuitively know that the values displayed correspond to individual sensor nodes.

3.12.1 Objects

The user interface consists of a single screen displayed in figure39.

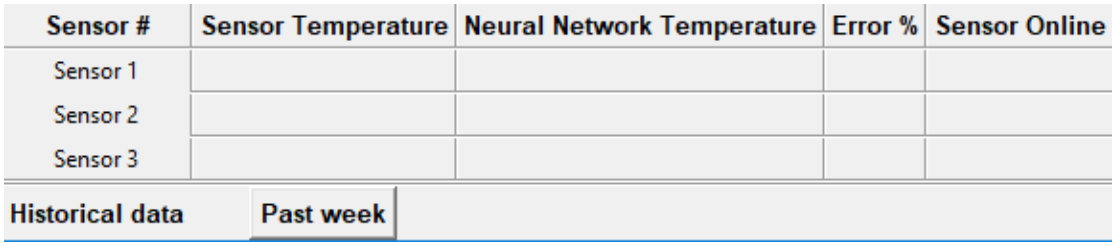

Figure 39.
Server user interface.

3.12.2 Components available

The Python Tkinter library provides a wide variety of GUI components available for use. The following components are used:

- Activity: This refers to the environment housing all the UI components laid out on the screen.
- Label: A UI component providing the user with descriptions.
- Button: A UI component providing the user with something to click
- Icon: A UI component that gives other UI components meaning

3.12.3 Interface design rules

The following two elements as described in [13] for the UI design have been considered:

- Cognition
- Utility

UI design element	Application	Relevance
Cognition	The layout of the components in the GUI and use thereof are initiative and consistent.	Promotes ease of use and a fast learning curve.
Utility	The plot of the temperature is a visual representation of the collected data.	The visual representation of the collected data has more meaning than values that simply update over time. It provides the user with feedback of the collected data.

Table 14.
Description of the user interface.

3.13 USE CASES

There are two use cases for the system which are defined in table15 and table 16.

Use case ID:	UC_1		
Use Case Name	Acquiring training data		
Created by:	M. Matusowsky	**Last updated by:**	N/A
Date created:	04-09-2017	**Last revision date:**	N/A
Actors	The user		

Table 15
Use case one part one.

Trigger/Entry condition:	• The software must be loaded on each sensor node • The software must be available on the desktop PC acting as the server • When the user starts the server application they should be prompted to turn on the sensor nodes
Normal flow:	• The user makes use of the GUI to view the incoming data • The data that is collected is stored locally in a CSV file
Alternative flow:	• The user must terminate the current operation and turn off all sensor nodes before beginning a new sensor reading session.
Exit conditions:	• The user can terminate the program by pressing the close button of the window.

Table 15
Use case one part two.

Use case ID:	UC_ 2		
Use Case Name	Virtual sensor active		
Created by:	M. Matusowsky	**Last updated by:**	N/A
Date created:	09-09-2017	**Last revision date:**	N/A
Actors	The user		
Trigger/Entry condition:	• The software must be loaded on each sensor node • The software must be available on the desktop PC Acting as the server • When the user starts the server application they should be prompted to turn on the sensor nodes		
Normal flow:	• The user makes sue of the GUI to view the incoming data • The system determines if a sensor node is disconnected and uses the virtual sensor to impute values if it is not connected. • The data is stored locally in a CSV file		
Alternative flow:	• The user must terminate the current operation and turn off all sensor nodes before beginning a new sensor reading session		
Exit conditions:	• The user must terminate the program by pressing the close button of the window		

Table 16.
Use case two.

3.14 RESTRICTIONS, LIMITATIONS AND CONSTRAINTS

This section describes the issues that have an impact on the design and implementation of the software.

Due to the sequential implementation of the neural network, the training cannot make use of multiple cores to make use of parallel processing to train the neural network faster. The performance of the system however is not impacted once the training is complete.

The user is required to manually turn a sensor node off and on again if it fails during the test data acquisition phase otherwise the system will hang up. Due to the implementation of the TCP connection, each sensor node has a timeout which means if one sensor node fails and is not restarted within a certain timeframe then all the sensor nodes will need to be shut down and reconnected.

Queries to each sensor node can not occur faster than once every five seconds as the current spikes that occur during transmission cannot be sustained by the USB power outlet provided by the desktop PC. Shorter times result in the server ESP8266 WiFi module entering a frozen state, requiring a cold restart of the system.

The software implementation is largely sequential in nature in that a sensor reading must be acquired first before filtering of the data or imputation using a VS can occur. This is an iterative process for each sensor node. This is a limitation in terms of parallel processing techniques.

3.15 DESIGN SUMMARY

This section summarises the project tasks and how they were implemented.

Task	Implementation	Task completion
Investigate suitable microcontrollers for the sensor node	The PIC32MX220F023B was selected as the processing unit of the sensor nodes.	completed
Investigate suitable WiFi modules in terms of cost-effectiveness	The ESP8266 WiFi module was selected as a very cheap WiFi communication device that could communicate through USART as well.	completed
Design of the microcontrollers state machine control paradigm	Implemented using the Harmony framework in MPLAB	completed
Design of a stripboard layout for the sensor node	Designed and implemented using the Fritzing opensource software from first priinciples.	completed
Design of a temperature sensor	Designed and implemented from first principles.	completed
Design of an MLP to be used as a VS	Designed and implemented in Python	completed.
Design of the MLP training method using genetic algorithms	Implemented in Python	completed
Optimisation of the MLP topology	An optimal topology was obtained for the MLP through simulations	completed
Design of the SKF filtering method	Implemented in python	completed
Optimisation of the SKF filtering method.	Optimal values for the SKF was obtained for the covariance noise coefficient	completed

Table 17.
Design summary part one.

Task	Implementation	Task completion
Performance evaluation for MLP	The performance of the MLP was measured in terms of the accuracy, time to run as well as statistical analysis	completed
Design of system architecture for the server	Implemented in Python	completed
Database design	Implemented Python	completed
Software design	Implemented in Python	completed
Design of user interface	Implemented in Python	completed

Table 17.
Design summary part two.

4. Results

4.1 Summary of results achieved

Description of requirement or specification (intended outcome)	Actual outcome	Actual outcome
Mission requirements of the product		
The system must use machine learning algorithms to implement virtual sensors in the wireless sensor network	The system was trained using a MLP machine learning algorithm.	Section 4.3.3c
Communication between nodes in the network must be done using wireless communication	WiFi communication was implemented and the sensor nodes communicated using the WiFi communication channels.	Section 4.3.3a
The algorithms implemented must be efficient enough so as not to introduce computational congestion into the system	No computational congestion was introduced into the system.	Section 4.2.3.
The virtual sensor smust be able to replicate data accurately as if they were real sensors	The virtual sensors were able to replicate the values satisfactorily.	Section 4.2.1 Section 4.2.2
The data read by all the sensors must be stored in a database	The data was stored in CSV files that were used as the database	Section 4.3.3.e
The virtual sensors must be implemented on every corresponding node as well as having a copy of every trained virtual sensor on the server	The virtual sensors were implemented on the server and on all three sensor nodes in the WSN	Section 4.2.1 Section 4.2.2 Section 4.2.3
Each sensor node should consist of a power unit, a sensor module and a processing unite	Every node was implemented with a USB charger, a temperature sensor designed with a thermistor and a PIC32.	Section 4.3.3.f
Field conditions		
The sensor nodes must be directly connected to the server.	The sensor nodes were connected directly to the server using a WiFi communication channel.	Section 4.3.2a
The server must run on a computer capable of processing large streams of data	The server was implemented on a sextuple core CPU	Section 4.2.1
Specifications		
Accuracy should be within 30% for 90% of observed values	The accuracy was within 30% for 100% of all observed values	Section 4.2.1
The standard deviation should not be more than 1.5 °C	The highest standard deviation was 1.229	Section 4.2.2

Table 18
Summary of results achieved part one.

Description of requirement or specification (intended outcome)	Actual outcome	Actual outcome
Specifications		
Imputation of a single value should take no longer than 7 seconds when using the MLP neural network algorithm while running on the sensor node or server	Imputation never took longer than 0.588 seconds.	Section 4.3.3
Deliverables		
Atmospheric (barometric or temperature or humidity) sensor modules for the processing units	Temperature sensors were designed and implemented by the student.	Section 4.3.3.b
Sensor module interface with the microcontroller	The temperature sensors could interface with the ADC of the microcontroller and was developed and implemented by the student.	Section 4.3.3.b
Wireless communication module interface with the microcontroller	The microcontroller could communicate with the ESP8266 WiFi modules using USART communication designed and implemented by the student	Section 4.3.3.a
Processing unit sensor control software	Designed and implemented by the student	Section 4.3.3
Processing unit communication protocol	Designed and implemented by the student	Section 4.3.3.a
Server communication protocol	Designed and implemented by the student	Section 4.3.3.a
Database on the server to store the data received on the sensor nodes	Designed and implemented by the student	Section 4.3.3.e
Imputation algorithm using MLP neural network technique in Python	Designed and implemented by student	Section 4.2.1 Section 4.2.2 Section 4.2.3
Data modelling software on the server	Designed and implemented by student	Section 3.6.3
GUI for the PC application	Designed and implemented by the student	Section 3.12.1

Table 18
Summary of results achieved part two.

4.2 QUALIFICATION TESTS

4.2.1 Qualification test 1: testing of the virtual sensor accuracy

4.2.1.a. Qualification test 1

Objectives of test

The objective of the virtual sensor accuracy test is to verify that the accuracy specifications set out in the mission-critical system specifications in **Part 3, section 4** was achieved.

Equipment used

The accuracy analysis was performed within the Spyder (Python 3.6.0) simulation environment using 14000 VS and sensor readings acquired during a five-day period.

Experimental parameters and set-up

Two NumPy array were populated with the VS readings and sensor readings respectively collected from a small residential complex apartment building and a large residential building (floor plans available in **Appendix 8.1 Floor Plans**). The accuracy was then calculated for each member in the arrays using equation

$$acc\% = \left(1 - \frac{|VS_i - sensor_i|}{sensor_i}\right) x100 \quad (20)$$

where VS_i is the virtual sensor reading at position i, $sensor_i$ is the sensor reading at position i and $acc\%$ is the accuracy as a percentage. The sensor readings are considered the ground-truth when acquiring the accuracy percentage for the VS.

The NumPy statistics library was imported to perform statistical analysis on the data.

Experimental protocol

The sensor and VS readings were iterated through and the accuracy calculated for each reading in the corresponding position of both arrays. The minimum, maximum and average accuracy were then calculated using the NumPy statistics library. The amount of readings that fall out of the accuracy specification, if any, were added together and divided by the total amount of readings to calculate the percentage of values that fall out of the specification.

4.2.1.b. Results and observations

Measurements

Small building measurements

Virtual sensor #	Minimum accuracy %	Maximum accuracy %	Average accuracy %	% of values within specification
Virtual sensor 1	78.968465 %	99.998470 %	94.032483 %	100.000000 %
Virtual sensor 2	92.472035 %	99.999959 %	97.075886 %	100.000000 %
Virtual sensor 3	86.522910 %	99.998250 %	95.154216 %	100.000000 %

Table 19.
Accuracy results of virtual sensors deployed in a small residential building

Big building measurements

Virtual sensor #	Minimum accuracy %	Maximum accuracy %	Average accuracy %	% of values within specification
Virtual sensor 1	85.523886 %	99.999673%	94.702047 %	100.000000 %
Virtual sensor 2	98.345442 %	99.999982%	96.548743 %	100.000000 %
Virtual sensor 3	87.905764 %	99.999920 %	96.281312 %	100.000000 %

Table 20.
Accuracy results of virtual sensors deployed in a large residential building.

Description of results

The minimum, maximum and average accuracy expressed in percentages were recorded for each virtual sensor using both acquired datasets. The overall averages of the small residential building virtual sensors were 94.03%, 97.08% and 95.15%. The overall averages of the large residential building virtual sensors were 94.70%, 96.55% and 96.28% All the values measured were within the specification. The worst-case accuracy experienced was by virtual sensor 1 in both cases with 78.97% and 85.52%. The best-case scenario resulted in a near 100% accuracy for all three sensors in both test cases. The best performing virtual sensor was virtual sensor 2 in both test cases. All the accuracy results were recorded in table 19 and table 20 respectively.

4.2.2 Qualification test 2: testing of the standard deviation for the virtual sensors

4.2.2.a. Qualification test 2

Objectives of test

The objectives of the standard deviation test are to verify that the standard deviation specification associated with the virtual sensor imputation method has been achieved.

Equipment used

The standard deviation calculation is performed within the Spyder (Python 3.6.0) simulation environment.

Experimental parameters and set-up

Two NumPy arrays were populated with the acquired datasets of the sensor readings and VS readings. There are 14000 VS and sensor readings in the dataset. The sensor readings are considered ground -truth for calculating the error of the VS.

Experimental protocol

The absolute value of the difference between the VS reading and the sensor reading is calculated and stored as the error in a NumPy array. Once the array is populated with each error associated with a reading, the standard deviation of the error is calculated using the NumPy library. The plot of the sensor readings as well as the VS readings is also constructed for visualisation purposes.

4.2.2.b. Results and observations

Measurements

Small building measurements

Virtual sensor #	**Standard deviation**
Virtual sensor 1	1.226344 °C
Virtual sensor 2	0.518652 °C
Virtual sensor 3	0.721729 °C

Table 21.
Standard deviation results of virtual sensors deployed in a small residential building.

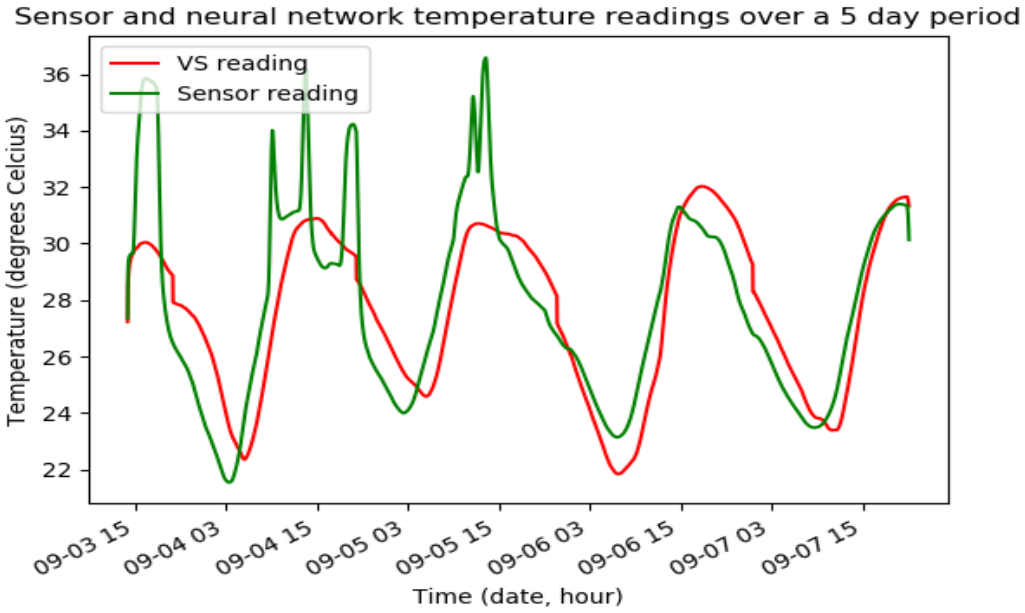

Figure 40.
Sensor 1 and VS 1 readings over a 5-day period in a small residential building.

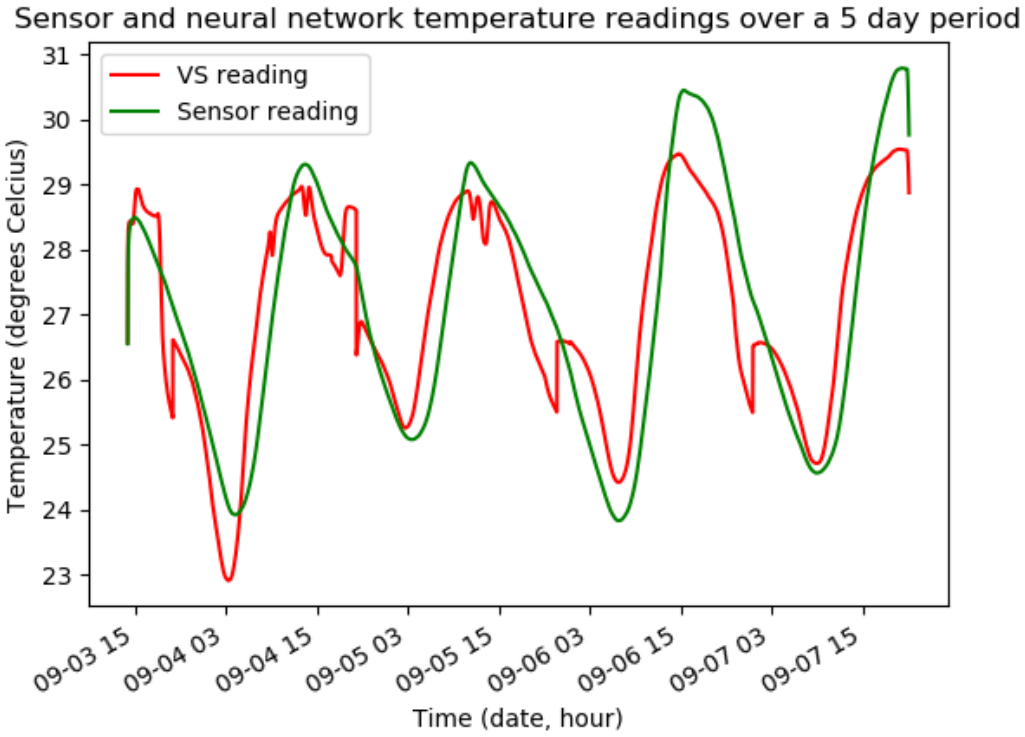

Figure 41.
Sensor 2 and VS 2 readings over a 5-day period in a small residential building.

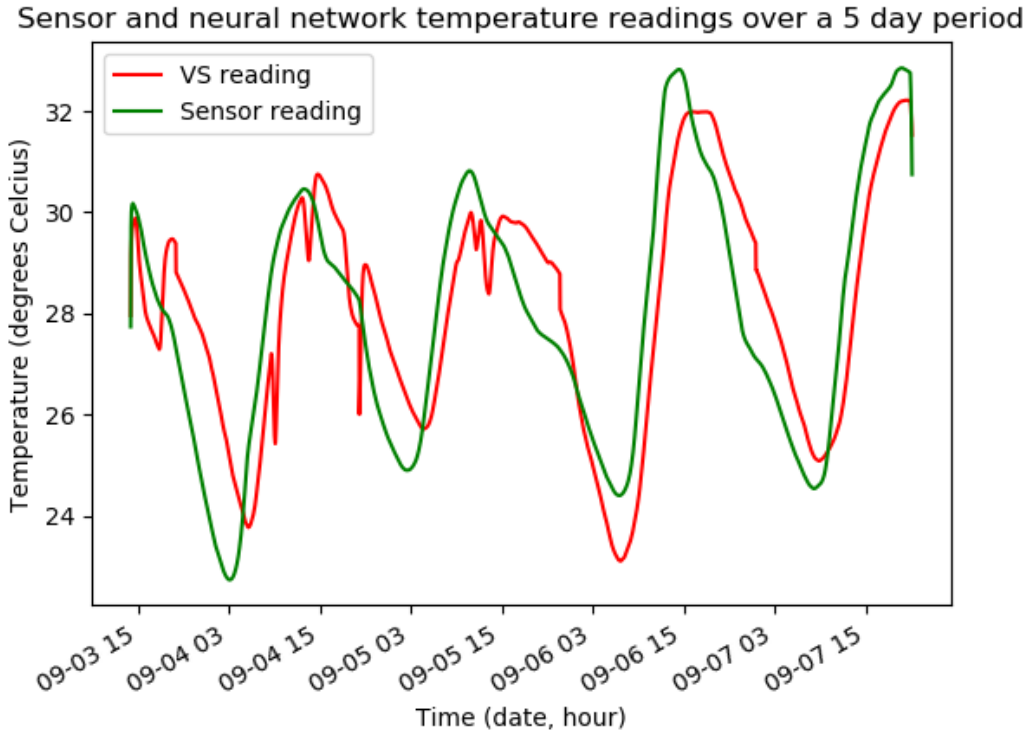

Figure 42.
Sensor 3 and VS 3 readings over a 5-day period in a small residential building.

Big building measurements

Virtual sensor #	**Standard deviation**
Virtual sensor 1	0.964535 °C
Virtual sensor 2	0.784815 °C
Virtual sensor 3	0.692576 °C

Table 22.
Standard deviation results of virtual sensors deployed in a large residential building.

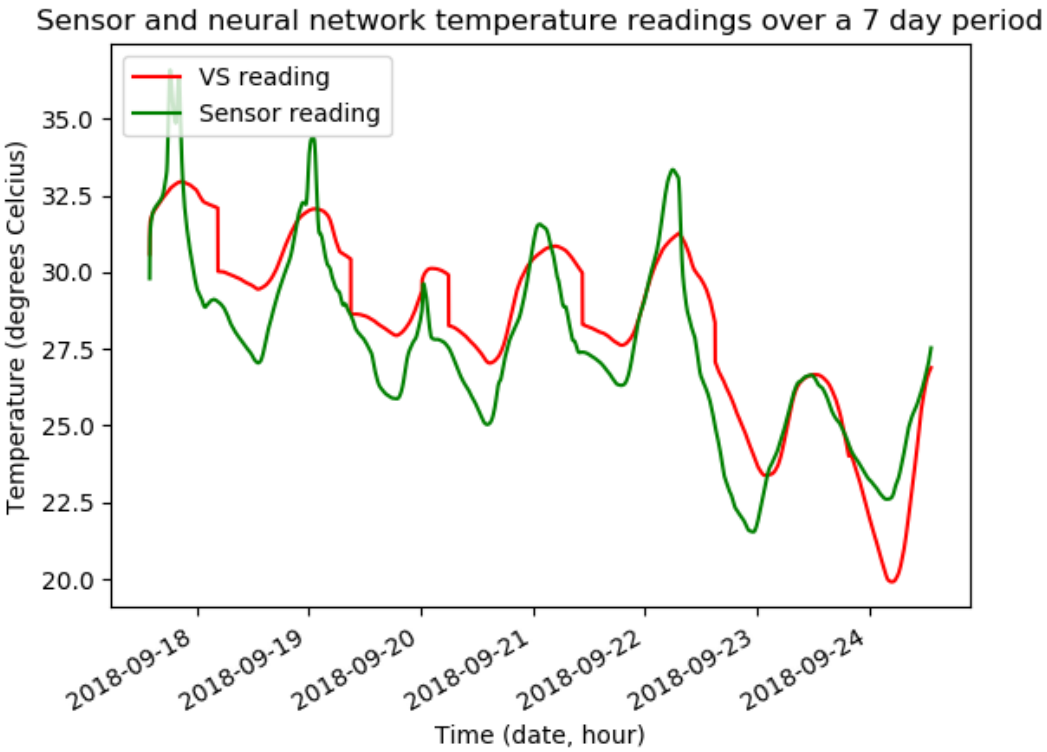

Figure 43.
Sensor 1 and VS 1 readings over a 7-day period in a large residential building.

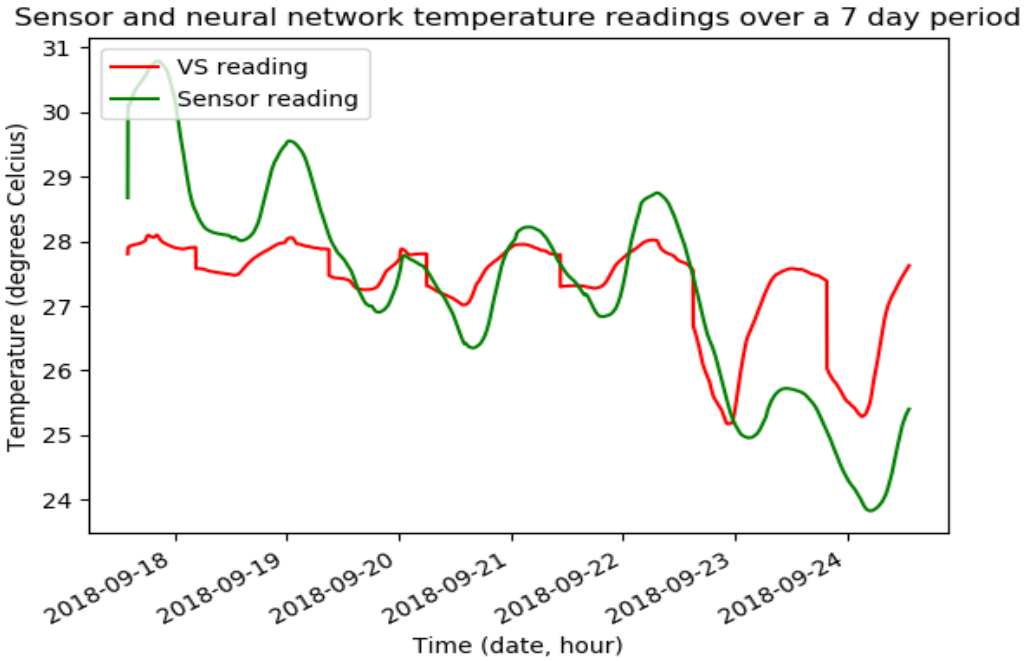

Figure 44.
Sensor 2 and VS 2 readings over a 7-day period in a large residential building.

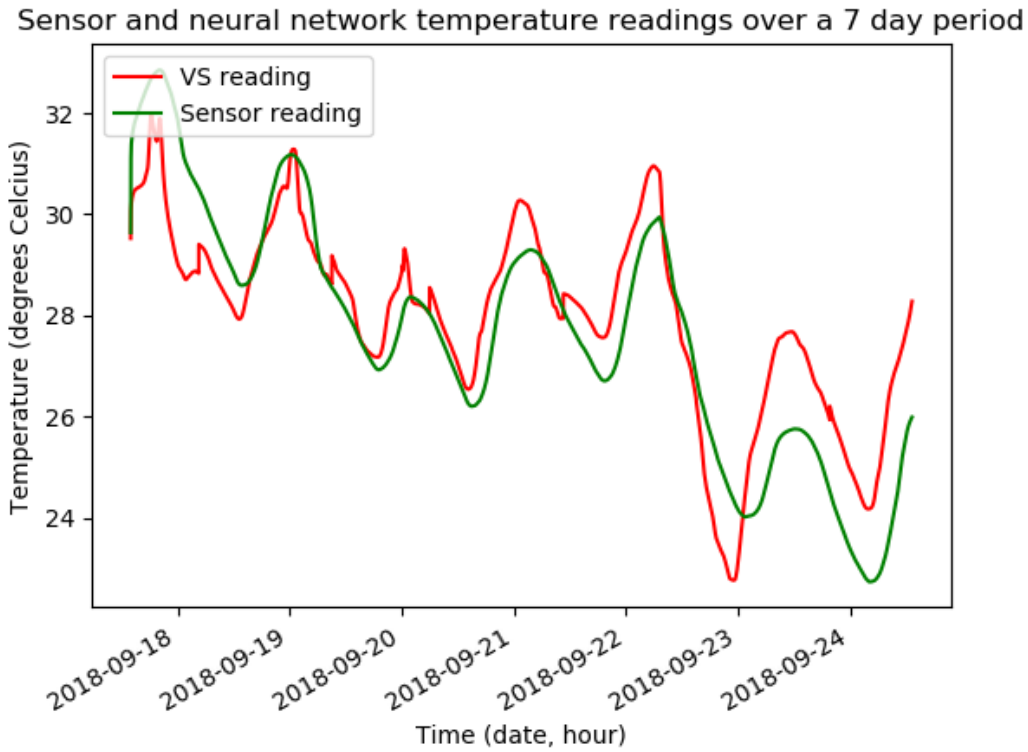

Figure 45.
Sensor 3 and VS 3 readings over a 7-day period in a large residential building.

Description of results

The standard deviation °C is recorded in table 21 and table 22 respectively for the small residential building and the large residential building. The 5-day plots of the VS and sensor readings for the small residential building are displayed in figure 41, figure 42 and figure 43. The 7-day plots of the VS and sensor readings for the large residential building are displayed in figure 44, figure 45 and figure 46.

4.2.3 Qualification test 3: testing of the time to impute values using the MLP

4.2.3.a. Qualification test 3

Objectives of test

The objective of the time to impute value test is to verify that the specification associated with the amount of time that the imputation algorithm has been achieved.

Equipment used

The developed sensor nodes were connected to a PICKIT 3 which in turn was connected to the desktop PC running the server. A timer was setup in the MPLAB simulation environment for the neural network on the PIC32 microcontroller on the sensor node

and a separate timer was setup in the Spyder (Python 3.6.0) simulation environment for the neural networks on the server.

Experimental parameters and setup

A timer was setup in the mentioned simulation environments. The system has three sensor nodes and a desktop server as part of the WSN. Thereafter the system is started, and the time measurements stored in an array for analysis.

Experimental protocol

The system is activated to store the amount of time it takes to impute data both on the server and through the sensor nodes. The time data is then averaged for each individual sensor to get a mean-time-to-impute as well as the virtual sensors on the server. The minimum time and maximum time to impute is also extracted from the data.

4.2.3.b. Results and observations

Measurements

Sensor #	**Minimum time**	**Maximum time**	**Average time**
Sensor 1	0.382 s	0.588 s	0.413 s
Sensor 2	0.391 s	0.547 s	0.412 s
Sensor 3	0.378 s	0.55 s	0.412 s

Table 23.
Virtual sensor time to impute value on a sensor node.

Virtual sensor #	**Minimum time**	**Maximum time**	**Average time**
Virtual sensor 1	9.322 ms	9.366 ms	9.379 ms
Virtual sensor 2	9.343 ms	9.392 ms	9.379 ms
Virtual sensor 3	9.327 ms	9.374 ms	9.379 ms

Table 24.
Virtual sensor time to impute value on the server.

Description of results

The time to impute in seconds (s) for each virtual sensor on each sensor node as well as each virtual sensor on the server were recorded and averaged. The minimum time to impute as well as the maximum time to impute were recorded as well in table 23 for the sensor nodes and table 24 for the server respectively.

4.3 COMPONENT TEST PLANS AND PROCEDURES

4.3.1 Component test strategy overview

This section provides an overview of the testing strategy used for each identified component.

4.3.2 Component test procedure

4.3.2.a. Component: WiFi communication

Test case number	1
Features to be tested	WiFi connectivity in the WSN
Testing approach	The sensor nodes were configured with the correct connection commands. The server was configured with the correct start-up commands. Each sensor node was then turned on to see if the node would connect to the available WiFi ssid and transmit a short message.
Pass/fail criteria	Pass if sensor node was assigned a WiFi ID and a message was received by the server. Fail if no connection was established between a sensor node and the server
Testing environment	Python and real-world environment in a building.

Table 25.
WiFi communication test component.

4.3.2.b. Component: ADC sensor measuring

Test case number	2
Features to be tested	Measuring ability of the ADC
Testing approach	The sensor was connected to the ADC port of the PIC32. A test program was then downloaded to the microcontroller that would periodically attempt to take an ADC reading.
Pass/fail criteria	Pass if an ADC reading was taken and it was similar to the ambient environment. Fail if no ADC reading was taken by the microcontroller.
Testing environment	MPLAB

Table 26.
ADC measuring component test.

4.3.2.c. Component: VS accuracy test

Test case number	3
Features to be tested	Imputation accuracy
Testing approach	The collected dataset was extracted into corresponding arrays. The accuracy was then determined using the equation for accuracy for each sensor node and virtual sensor. Statistical analysis was then done using the NumPy library.
Pass/fail criteria	Pass if the accuracy is met or exceeded. Fail if the accuracy does not meet the specification
Testing environment	Spyder (Python 3.6.0)

Table 27.
VS accuracy component test.

4.3.2.d. Component: Training method to train the VS

Test case number	4
Features to be tested	Genetic algorithm training method
Testing approach	The virtual sensors will be trained using a genetic algorithm. A specified number of epochs is given, and the error calculated on every epoch.
Pass/fail criteria	Pass if the training error generally decreases as the epochs increase. Fail if the algorithm does not decrease the error but stochastically randomises the neural networks on every epoch.
Testing environment	Spyder (Python 3.6.0)

Table 28.
Genetic algorithm training method component test

4.3.2.e. Component: Data acquisition and storing

Test case number	5
Features to be tested	The sensor and VS readings must be stored in a database
Testing approach	The system will be run for a few days. The database should be populated over time by the sensor readings and VS readings.
Pass/fail criteria	Pass if the database is populated with data that is not corrupted. Fail if the database does not populate or is corrupted.
Testing environment	Spyder (Python 3.6.0)

Table 29.
Database acquisition and storing component test

4.3.2.f. Component: Sensor power using USB charger

Test case number	6
Features to be tested	The sensor node must be able to be powered using a USB charger plugged into a power source.
Testing approach	The USB connecting wire will be plugged into a USB charger. The charger is then plugged into a wall socket.
Pass/fail criteria	Pass if the system turns on. Fail if the system does not power up.
Testing environment	Real-world environment in a building.

Table 30.
Sensor power using a USB charger component test.

5. Discussion

This section contains a discussion and interpretation of the results in the previous section

5.1 INTERPRETATION OF THE RESULTS

The success of any virtual sensor system using machine learning at the core of the design is dependent on the successful design and implementation of the chosen method of machine learning as well as the quality of the training data that is used to train the virtual sensors. This was evident in the results obtained for the overall virtual sensor network and its associated subsystems. During the literature study and design, the design considerations associated with the machine learning algorithms and data filtering techniques were considered. The design was informed by the inferences drawn from the literature study namely the chosen MLP algorithm widely used in time-series problems as well as the emerging field of neuroevolution to train the MLP using genetic algorithms. The specific choice of sensors informed the overall testing environment.

When the results for the accuracy are compared with regards to the size of a building, table 19 and table 20, it was found that a virtual sensor network deployed in a large residential building very marginally outperforms a small residential building where the overall average accuracy of the three virtual sensors were within specification and of very similar accuracy. It was further found that the system was able to impute values with a 100% success rate for all values within 30% of the filtered sensed values. This is consistent with the findings in literature.

The choice of using genetic algorithms to train the MLP proved to be well suited in all three virtual sensor cases as evidenced by the standard deviation measurements in table 21 and table 22 respectively and visualised by the plots given in figures 39-45 where it was seen that the virtual sensors accurately depicted the changes in temperatures over several days.

It was found that with the increase in computational power in the modern day that the studied literature was outdated in terms of the amount of time that should be given to impute sensor values using an MLP where the 40 MHz processing speed of the PIC32 was able to impute values in under a second with the longest imputation lasting 0.588 seconds as compared to the literature's reported maximum time of 7 seconds on a desktop CPU whereas the desktop CPU used in this project was able to impute values on average in 9.379 ms for all three sensors as reported in table 23 and table 24.

The approach to implement the wireless communication in the WSN using WiFi communication was a positive design choice as many IoT applications in literature make use of the TCP/IP stack which works well over the WiFi 2.4 GHz spectrum and allows easier management of clients that connect to the network. It was found that due to the power spikes that occur when transmitting data from the server that a query could only be sent every 5 seconds or WiFi module connected to the server would crash and require a cold restart which is due to the oscillations in current draw. The USB port on

the PC was not able to sustain a high current draw for the ESP8266 WiFi if the current was required on a relatively frequent basis.

In conclusion, all the mission-critical specifications were met. In hindsight, the critical specifications for the time to impute as well as the accuracy should have been more optimistic given the advancements made in computational power and the topic of machine learning but the original specifications were based on the literature that was available for virtual sensor networks deployed for atmospheric sensing.

5.2 ASPECTS TO BE IMPROVED

Following the current design of the genetic algorithm training method, the lack of parallel processing in the implementation means that the full scope of the computational power available was not being utilised which affects the speed with which a virtual sensor can be trained.

With regards to the server ESP8266 WiFi module a separate power supply could be added to increase the amount of queries that can be made to sensor nodes throughout the network although due to the temperature sensor implementation and the slow rate of change in temperature it is not a necessary improvement.

The MLP algorithm can be improved by optimising the forward propagation and activation functions which can further improve the imputation speed especially on the microcontrollers.

5.3 STRONG POINTS

The use of a genetic algorithm to train the virtual sensors ensured that the most optimal solution was found for each sensor node with the training data provided. The genetic algorithm was able to converge on an optimal solution in a relatively small amount of time despite no parallelism being implemented into the training algorithm.

The good performance of the virtual sensors is attributed to the training method chosen as backpropagation would not have guaranteed an optimal solution and may have exhibited an oscillatory nature during training whereas the genetic algorithm error only decreased every epoch.

5.4 FAILURE MODES OF THE DESIGN

The product has two fail mode that were identified during testing. If the sensor nodes in the network take too long to establish a TCP connection with the server, the sensor nodes already connected to the server will time out. The second failure mode occurs if two sensor nodes are disconnected during run time as the system is designed to only handle one sensor node failing in the network since the input for a virtual sensor requires to physical sensor node readings and a time reading.

5.5 DESIGN ERGONOMICS

The ergonomics design is detailed in table 14. No user input is required other than starting the system which enhances the users experience by not needing any user input to activate a virtual sensor in the system as it is able to determine on its own if a sensor node needs to be replicated by a virtual sensor. The display is too the point and a user should be able to tell which values belong to a sensor node and the values that belong to a virtual sensor from the displayed labels. The sensor nodes themselves were designed to simply be Plug-and-Play (PnP) so that a user would only need to place the nodes in the desired positions and plug them into a power source with a two-point plug adapter with no additional technical requirements from the user other than ensuring that the sensor nodes are all connected before any time out during the connection phase.

5.6 HEALTH AND SAFETY ASPECTS OF THE DESIGN

All the sensor nodes are enclosed in plastic boxes where a user's hands are safe from the electrical components. The power cable is an insulated USB cable with no exposed wires that could cause an electric shock risk to the user.

Since the system works in the WiFi frequency range, a user is exposed to low GHz electromagnetic frequencies (EMF) where research into the harmful effects that may be caused by EMFs, such as headaches and tumours, is still being conducted to determine if there is a definite link between the two despite the widespread use of the 2.4 GHz frequency used for WiFi.

5.7 SOCIAL AND LEGAL IMPACT OF THE DESIGN

The product is designed for English users but may be extended to other localisations. The system can be used in public spaces to automatically regulate room temperatures using the developed virtual sensors instead of control systems currently used in room temperature regulators.

Open source software was used to develop the system which may have to go through the Open Source Initiative's license review process if the product is to be commercialised.

5.8 ENVIRONMENTAL IMPACT AND BENEFITS OF THE DESIGN

The system makes use of power supplied by the electrical grid which consists of power generated through nuclear and coal powered plants however the power used by the system is very minimal such that the carbon footprint is miniscule. Furthermore, the system can be implemented as a cost saving measure in sensor networks where the cost of upkeeping sensor nodes is prohibitively expensive or as a control environment where

electricity can be saved using a smart virtual sensor network i.e. controlling air conditioning machines using virtual sensors.

6. Conclusion

6.1 SUMMARY OF THE WORK

A virtual sensor using machine learning was developed for use in a wireless sensor network. The virtual sensor met the mission-critical requirements of the project which were designed and implemented from first-principles.

6.2 SUMMARY OF THE OBSERVATIONS AND FINDINGS

The project was designed and implemented using an MLP neural network machine learning algorithm which proved to be quite effective in imputing values to an accurate degree once trained with the collected training data. The use of genetic algorithms to train the MLP, a method not as widely used as the backpropagation algorithm, was an important design decision as it was guaranteed that the most optimal MLP would be generated from the training in a minimal time.

The implementation of the SKF proved to be effective in filtering out unwanted noisy values such as the ground glitches.

The system was able to successfully identify and replace a sensor node with a virtual sensor that was disconnected from the network. The virtual sensor imputed values well within the margin of error allowed by the mission-critical specifications when compared to the sensor values. The system was also able to turn off the server's virtual sensor when the sensor node in question reconnected to the network.

6.3 SUGGESTIONS FOR FUTURE WORK

Referring to the above report, the positive results obtained for each virtual sensor can inform further implementations. Using different sensors, different information can be imputed once relationships have been established between sensors i.e. rotation speed of a vehicle's wheels may inform the expected heat experienced by the tires and so a virtual sensor instead of a temperature sensor can be used to save on costs in smart vehicles.

It can also be suggested to use multiple machine learning algorithms in combination to get possibly even more accurate virtual sensor results i.e. using the backpropagation training algorithm on an MLP that was found to be the most optimal by the GA.

Finally, the use of a different wireless communication technology could be used for WSNs working over much larger distances than in this project.

7. References

[1] N. S. Altman, “An introduction to kernel and nearest-neighbour nonparametric regression”, *The American Statistician,* vol. 46, no. 3, pp. 175-185, 1992.

[2] M. S. Astriani, G. P. Kusuma, Y. Heryadi, E. Abdurachman, “Smartphone sensors selection using decision tree and KNN to detect movements in Virtual Reality Application”, in *2017 International Conference on Applied Computer and Communication Technologies (ComCom),* IEEE, 2017.

[3] G. Cybenko, “Approximation by superpositions of sigmoidal function”, *Mathematics of Control, Signals, and Systems,* vol. 2, no. 4, pp. 303-314, 1989.

[4] Y. Iwashita, A. Stoica, K. Nakashima, R. Kurazume, J. Torresen, “Virtual Sensors Determined Through Machine Learning”, in *2018 World Automation Congress (WAC),* IEEE, 2018.

[5] T. Kohonen, “The self-organizing map”, *Proc. IEEE,* vol. 78, no. 9, pp. 1464-1480, 1990.

[6] Y. Takizawa, “Node localizaiton for sensor networks using Self-Organizing Maps”, in *2011 IEEE Topical Conference on Wireless Sensors and Sensor Networks,* IEEE, 2011.

[7] P. J. Werbos, “Backpropagation: What it does and how to do it”, *Proc. IEEE,* vol. 78, no. 10, pp. 1550-1560, IEEE, 1990.

[8] K. Levenberg, “A Method for the Solution of Certain Non-Linear Problems in Least Squares”, *Quarterly of Applied Mathematics,* vol. 2, pp. 164-168, 1944.

[9] D. Marquardt, “An Algorithm for Least-Squares Estimation of Nonlinear Parameters”, *SIAM Journal on Applied Mathematics,* vol. 11, no. 2, pp. 431-441, 1963.

[10] M. Harvey, “Let’s evolve a neural network with a genetic algorithm-code included”, 2017. [Online]. Available: https://blog.coast.ai/lets-evolve-a-neural-network-with-a-genetic-algorithm-code-included-8809bece164. [Accessed 2018- Aug -17].

[11] D. J. Montana, L. Lawrence, “Training feedforward neural networks using genetic algorithms”, *Proc. of the 11th international joint conference on Artificial intelligence,* pp.762-767, IJCAI, 1989.

[12] J. S. Steinhart, S. R. Hart, “Calibration curves for thermistors”, *Deep-Sea Research and Oceanographic Abstracts,* vol. 15, no. 4, pp. 497-503, 1968.

[13] K. R. Fowler, K. Fowler, *Electronic instrument design: architecting for the life cycle*. Oxford University Press on Demand, 1996.

8. Appendix

8.1 Small residential building floor plan

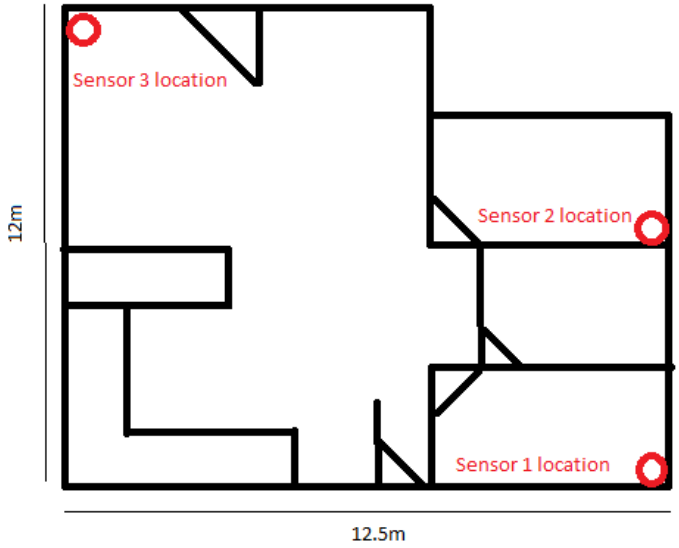

Figure 47
Floor plan of a small residential building.

8.2 Large residential building floor plan

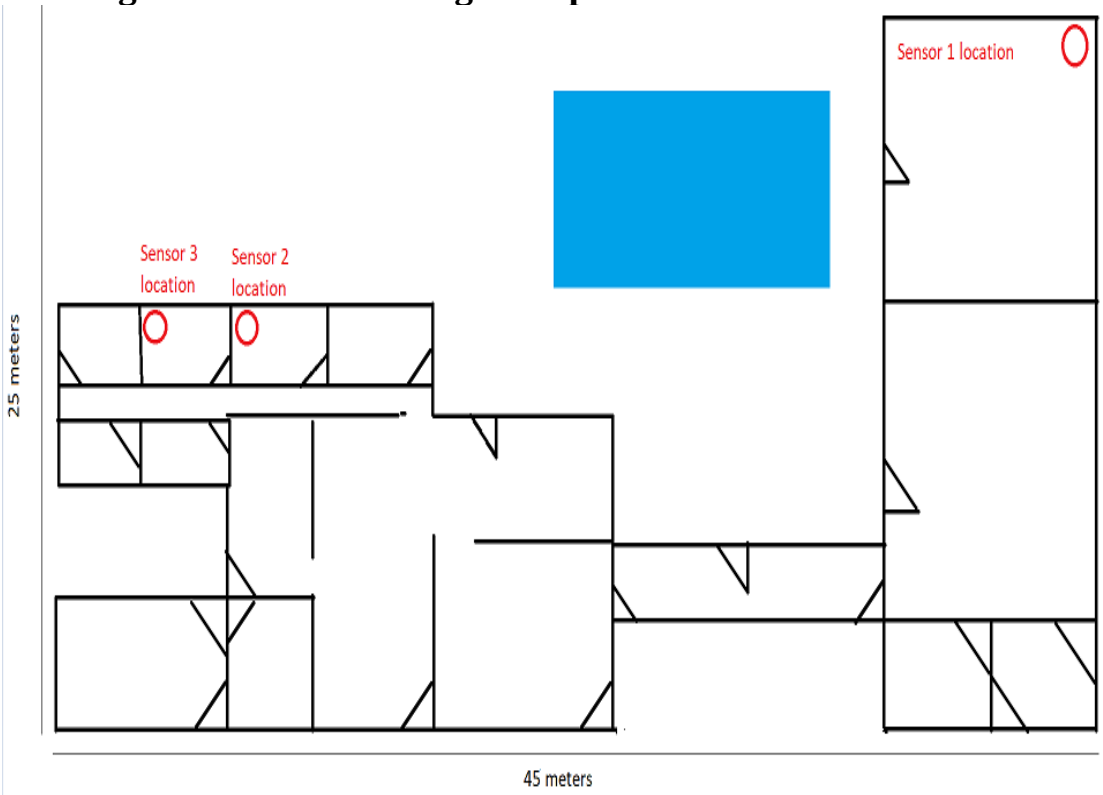

Figure 48
Floor plan of a large residential building.

Part 5. Technical documentation

This main report is supplemented with technical documentation. This provides more detail on the software that was used in the experiments, including program listings, a user guide and circuit designs. This section appears on the electronic medium that accompanies this report.

The CD (or DVD, or flash disk) is organized into the directories listed below.

Main report
Part 5: Technical documentation
Software
References
Datasheets
Author
Datasets/Raw
Datasets/Final