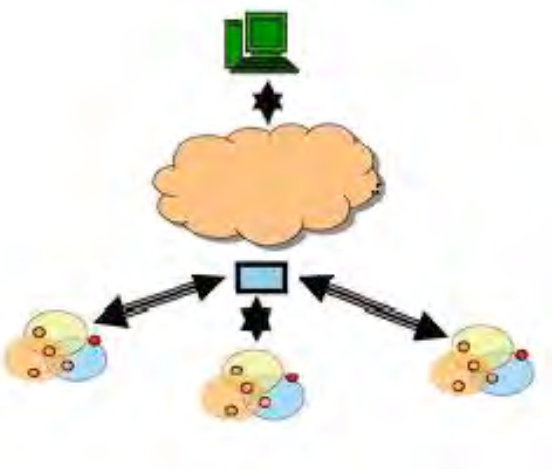


Outlier/Event Detection Techniques in Wireless Sensor Networks

Senior Year Project
Design and Simulation Report

by
Kamran Ali 13100174
Syed Bilal Ali 13100028
Muhammad Asad Lodhi 13100175
Ovais bin Usman 13100026

Advisor
Dr. Ijaz Haider Naqvi
[ijaznaqvi@lums.edu.pk]

Reader

December 14, 2012
Department of Electrical Engineering
Syed Babar Ali School of Science and Engineering
Lahore University of Management Sciences, Pakistan

Contents

1 Introduction **2**

2 Unsupervised Outlier Detection Using Aggregation Tree[2] **2**
2.1 Introduction 2
2.1.1 Preliminary 2
2.2 Algorithm 3
2.3 Experiments and Results 3
2.3.1 Local Outlier Detection 3
2.3.2 Global Outlier Detection 3
2.4 Complexity and Communication Overhead . . 3
2.5 Conclusion 3

3 Fixed Width Clustering **4**
3.1 Algorithm 4
3.2 Experiments and Results 4
3.3 Complexity and Communication Overhead . . 4
3.4 Conclusion 4

4 *K-Means* Clustering approach **4**
4.1 Algorithm 4
4.2 Experimental Results and Complexity 4
4.3 Conclusion 5

5 Clustering Ellipsoids **5**
5.1 Introduction 5
5.2 Preliminary 5
5.3 Algorithms 5
5.3.1 Local Detection 5
5.3.2 Global Detection: Positive Root Eigen Value (PRE) Technique 6
5.3.3 Detection for Streaming Data 6
5.3.4 Event Detection 6
5.4 Experimental Results 6
5.5 Complexity 6
5.6 Conclusion 6

6 Online Methods **7**

7 Incremental elliptical boundary estimation[7] **8**
7.1 Forgetting factor 8
7.1.1 Problem 8
7.2 The Effective N approach 8

8 A Simpler Incremental method for calculating the elliptical boundary **10**
8.1 Event Detection 10
8.1.1 Temporal correlation: 10
8.1.2 Spatial Correlation: 11
8.2 Event Identification 11
8.2.1 Taking one attribute and excluding the rest 11
8.2.2 Excluding one attribute and taking the rest 11

9 Hardware Overview[8] **11**

10 Progress **13**

11 Updated Timeline **13**

12 Upcoming Challenges **13**

1 Introduction

A Wireless Sensor Network (WSN) is a collection of low power and small memory nodes, also called *Motes*. Each node consists of sensors for measuring different attributes of the environment like Temperature, Humidity, etc., a microprocessor/microcontroller and wireless transceiver. So, each node has capabilities of both processing and communicating a stream of data. A wide variety of applications of WSNs includes those relating to personal, industrial, business, and military domains, such as environmental and habitat monitoring, object and inventory tracking, health and medical monitoring, battlefield observation, industrial safety and control, to name but a few. Applications also include Gunshot Detection, Monitoring Volcanic Eruptions, Glacier Monitoring, Forest Fire Detection, Neuromotor Disease Assessment, Urban Scale Monitoring, Communications in Smart Grids and Electrical Power Systems, and so on. In many of these applications, real-time data mining of sensor data to promptly make intelligent decisions is essential[1]. And to achieve this we must have a great degree of accuracy which demands reliable sensor data.

But the data measured and collected by WSNs is often unreliable. The quality of data set may be affected by noise and error, missing values, duplicated data, or inconsistent data. The low cost and low quality sensor nodes have stringent resource constraints such as energy (battery power), memory, computational capacity, and communication bandwidth[1]. The limited resource and capability make the data generated by sensor nodes unreliable and inaccurate. Moreover there can be malfunctioning due to harshness in the environment or malicious attacks. Due to this reason, actual events are likely to go undetected. Thus to ensure the reliability and accuracy of sensor data before the decision-making process, different outlier detection techniques are applied on WSNs data. Research has been done in this field and several different techniques have been developed. This report evaluates and compares four methods for outlier detection. One is *Distance based* and others are *Clustering based*.

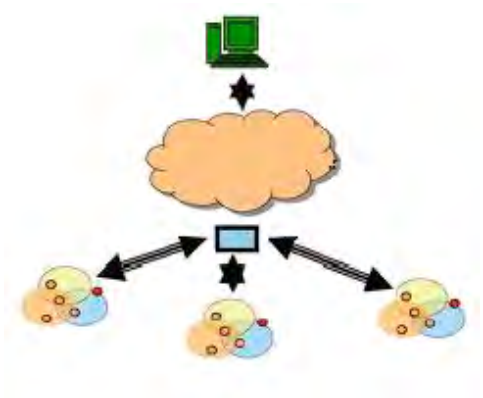


Figure 1: A typical Wireless Sensor Network

2 Unsupervised Outlier Detection Using Aggregation Tree[2]

2.1 Introduction

"*A WSN consists of hundreds or thousands of small, resource limited sensor nodes except one or more powerful nodes gathering the information of others. The powerful node is called the* sink *or* the base station *(BS). Main design goal of this method is to minimize communication overhead of the low powered sensor nodes. The approach is suitable to answer both snapshot and continuous queries. The basic idea is to build all the nodes in a structure of aggregation tree. Then let every node in the tree transmit some useful data to its parent after gathering all the messages from its children. The sink uses the information from its children to calculate global top n outliers, and floods these outliers for verification. If any node finds that it has two kinds of data which may modify the global result, it will send them to its parent in an appropriate time interval. All these processes are done level-by-level in the tree. Through a few such commit-disseminate-verify processes, all the nodes will agree on the global result calculated by the sink.*"[2]

2.1.1 Preliminary

Deviation Rank of data *x*, we can use the distance to *kth* nearest neighbors to measure its deviation rank. For a data x in dataset *S*, *R(x, S)* is used to denote the deviation rank of *x* in *S*. Here *R* is a deviation rank function. For different measurement criteria, *R(x, S)* equals different values. We suppose in all cases, *R* satisfies that for given $S1 \subseteq S2, R(x, S1) \geq R(x, S2)$. Here are a few more definitions :

Support Set 1 *Given a set $S_0 \subseteq S$ and a deviation rank function R, S_0 is called the support set of $x \in S$ over S if R(x, S_0) = R(x, S).*

This means that the support set S_0 of a data point $x \in$ S contains all the distinct data points in S which give the same rank for *x*. According to definition and implementation of the deviation rank in this paper, there can be one and only one distinct point in S_0 which would be the k^{th} neighbor of data point *x*.

Candidate Set 2 *For set S_i, its modifier set for global outliers consists of the data whose distance to a global outlier is less than the global outliers deviation rank.*

This simply means that Modifier set will contain all the distinct points at any node *i* whose rank, when calculated with respect to the Global outlier, comes out to be less than the rank of the Global outlier (which has been calculated at the sink) itself. Modifier set thus contains the data points which are no longer '*candidates*' for being Global outliers.

Modifier Set 3 *For set S_i, its candidate set for global outliers consists of the data which may be candidates of correct global outliers, i.e., the data whose local deviation rank is larger than a global outliers.*

The Candidate set will contain all the distinct points at any node *i* whose '*local*' rank comes out to be greater than the rank of the Global outlier itself. Candidate set thus contains the data points which are possible '*candidates*' for being Global outliers.

2.2 Algorithm

For a node N_i in the tree, S_i is used to denote the sample set of S_i in current sliding window. The knowledge of N_i is defined as $\bar{S_i} = S_i \cup \{\text{data received from } N_i\text{'s children}\}$. And dataset HS_i is used to record the data N_i has sent to its parent. Every node must transmit the most useful dataset for outlier detection to its parent to reduce communication overhead. For node N_i, it sends its local outliers A_i = A [S_i] and their smallest support set SA_i = [$\bar{S_i}|A_i$] to its parent after incorporating all the data from its children into its S_i. Meanwhile, N_i sets $HS_i = A_i \cup SA_i$. At the end of this process, the sink calculates the global top *n* outliers for the first time which may be not right. This is done because though the outliers of parent (the top *n* outliers in the dataset of the sub-tree rooted at the parent) may not be outliers of its children, they come from the union of these childrenfs outliers with high probability, especially when the data from different nodes is nearly the same.

To disseminate, the sink floods the unverified global outliers and their deviation rank. After receiving that, every node verifies and commits again. For node N_i, it calculates its modifier set M(S_i), sets $Info_i^1$ = M(S_i)\$HS_i$, and waits for enough time to receive messages from all its children. After incorporating all the data from its children into its S_i, N_i calculates its candidate set C(S_i), and sets $Info_i^2$ = C(S_i)$\cup$[S_i|C(S_i)]\$HS_i$[1] . If there is remaining data in $Info_i^1$ and $Info_i^2$, N_i sends ($Info_i^1$, $Info_i^2$) to parent node and adds them into HS_i. The reason we append the smallest support set of C(S_i) i.e. [S_i|C(S_i)] to $Info_i^2$ is that upper node can re-calculate deviation rank of the data in C(S_i) exactly after incorporating $Info_i^2$ into a larger data set[2]. After gathering all the committed information from its children, the sink recalculates global top n outliers. If the result is different from last one (either the outliers or their deviation rank), the sink will disseminate again until the result does not change.

2.3 Experiments and Results

We applied both the local and global detection part of this method on the *Temperature and Humidity* data obtained from neighborhood of a lake in Australia[6].

2.3.1 Local Outlier Detection

We applied the *Deviation Rank* function with different values of *n* and *k* for $Node_1$. *Figure 1* shows the results with False Positive Rates and Detection Rates.

2.3.2 Global Outlier Detection

We used data available for *3* nodes and set *one of them as sink and other two as the parent of the sink*. We applied the complete *Aggregation Tree algorithm* with different values of *n* and *k*. The communication between the sink and nodes carried on until there was nothing 'new' to pass to the sink i.e. when

[1] \ sign indicates *set difference* e.g. (*A* - *B*) = *A* \ *B*

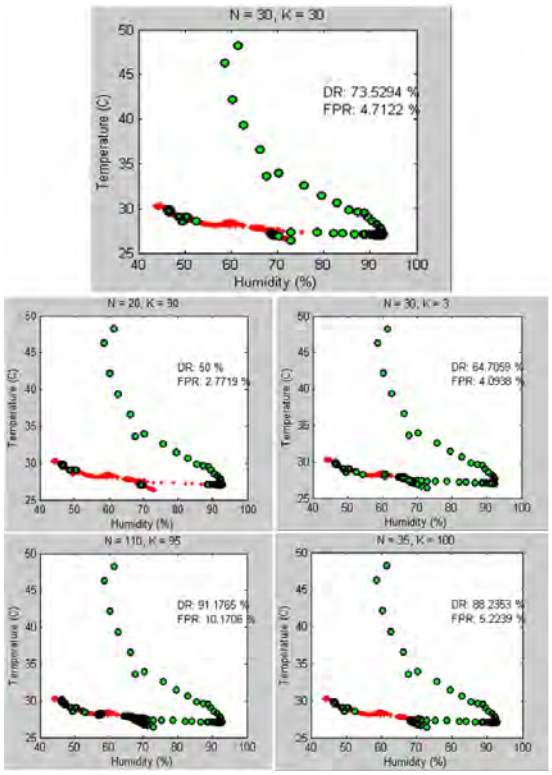


Figure 2: DK Method, Local Outlier Detection Results

$Info_i^1$ and $Info_i^2$ become empty for each node and when the new Global Outliers calculated by the sink are the same as of the previous iteration. *Figure 2* shows the results with False Positive Rates(FPRs) and Detection Rates(DRs).

2.4 Complexity and Communication Overhead

The above *3* node aggregation tree structure gave us very quick result; in just *2* to *3* iterations. So, judging from the *Communication Cycles* it took to the completion of the algorithm, we can appreciate the fact that the *overhead* of this Global Outlier Detection method is very less as compared to some other methods which Zhang discusses in his paper[2].

This method is mathematically simple. Calculation of Deviation Rank is of the order **O(n)**, *n* being the data points in a specific time interval. All other operations are linear and thus the overall complexity is of **O(n)**.

2.5 Conclusion

Figure 1 shows the results after running the DK algorithm on N_1 of data set. For local outlier detection *two* dimensions (Temperature and Humidity) are considered and outliers in both of them are plotted. In the detection of the outliers, we haven't taken any correlation between the humidity and temperature measurements (attribute correlation) into account. We simply found top N outliers in both humidity and temperature and then plotted them. *N = Top Outliers to be detected (User Specified) K = kth neighbor with respect to which the rank is calculated (User Specified).*

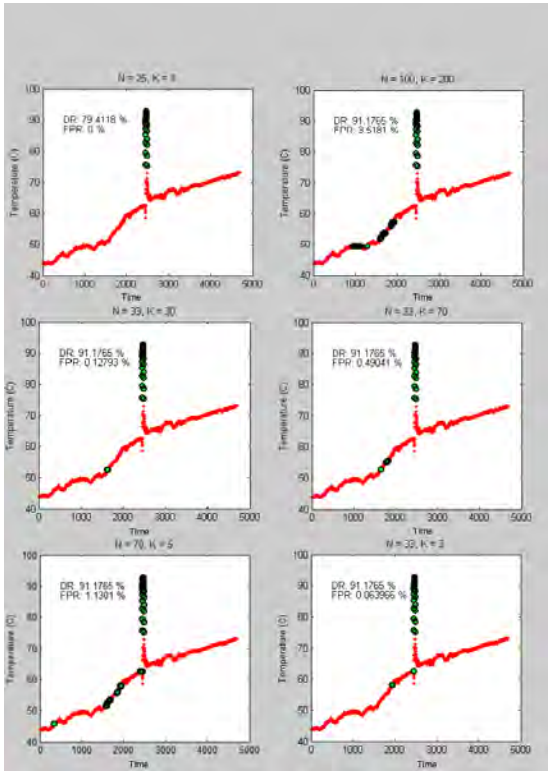


Figure 3: DK Method, Global Outlier Detection Results

Although the mathematical complexity and communication overhead are less and show less dependence on user given parameters , but the final results are too much dependent on the user specified values of N and K. Moreover, to get appropriate results, the selection of N and K can vary for different types of data.Due to this dependence we have to compromise on *False Positive Rate* in order to get a high *Detection Rate*.

3 Fixed Width Clustering

3.1 Algorithm

In this method a set of *k clusters* of fixed radius (width) *w* was created. *k* and *w* are the user specified parameters. First of all a data point from the data set is chosen and is designated as the centroid of the first cluster with radius *w*. Afterwards, *Euclidean* distance is calculated for each data point from the centroid of the current cluster. If the distance of a point is less than the given parameter *w*, that data point is added to that cluster and the cluster centroid is updated to the mean of the points assigned to it. If the distance is more than *w*, then if limit *k* is not reached a new cluster is formed with that data point as the centroid or that data point is declared as an outlier. This method thus creates a set of disjoint clusters in the feature space of our data with fixed width *w*.

3.2 Experiments and Results

3.3 Complexity and Communication Overhead

This method was for local outlier detection so no analysis on communication overhead was done. However, as all the operations applied are linear the mathematical complexity is of order **O(n)**, *n* being number of data points at a node within some time interval.

3.4 Conclusion

Results were reasonably good as compared to the Aggregation Tree approach but still the algorithm was dependent on the choice of radius *w* of the clusters and the number of clusters allowed.

4 *K-Means* Clustering approach

4.1 Algorithm

In clustering by K-Means approach[3], we first pick *k* number of points from the data set and declare them as the cluster *centroids*. Then we measure the distance of each point in our data set from these initially nominated points. We assign a point to the cluster from whose centroid it is least distant. After all points have been assigned to their particular clusters, we calculate the mean of the points assigned to each cluster and update the centroid to the value of the mean calculated. Then we repeat the part where we assigned the points to a specific cluster and after that the part where we update the mean until the distance of each point from its cluster centroid is minimized. After the clustering is complete, the outliers are nominated on the basis of a parameter *r* which in fact defines a circle of radius *(*r) around the cluster centroids. Any data point that is beyond the circular boundary of its cluster is then designated as an *outlier*.

4.2 Experimental Results and Complexity

Following were the results obtained by replicating this algorithm in [3] and implementing it on the data set mentioned in [3] (*Abalone*). *K = 4* clusters were used. We tested our algorithm on the data set[6] and found that it produced better results as compared to the distance based approaches regarding both outlier detection and computational complexity. Order of complexity was still of the order ***O(n)***.

For the abalone data we performed clustering in two ways. In the first one we applied the clustering algorithm on all the eight dimensions of the data and then projected the eight dimensional outliers in two dimensions. In the second one we applied the clustering algorithm on any two dimensions of the data to get the results.

As far as the sensor node data from the lakes is concerned, only two of its parameters were taken into account: humidity and temperature. The clustering algorithm applied on these nodes produced very good results. Although this clustering algorithm produced agreeable results yet it was dependent on two user-defined parameters: number of clusters “k” and radius of the circular boundary “r”. So, we moved on to a more efficient clustering technique: clustering by means of ellipsoidal boundaries. To be precise, K-means is of order *O(IcpN)*, in which *I* is the number of iterations. If we want this method to be independent

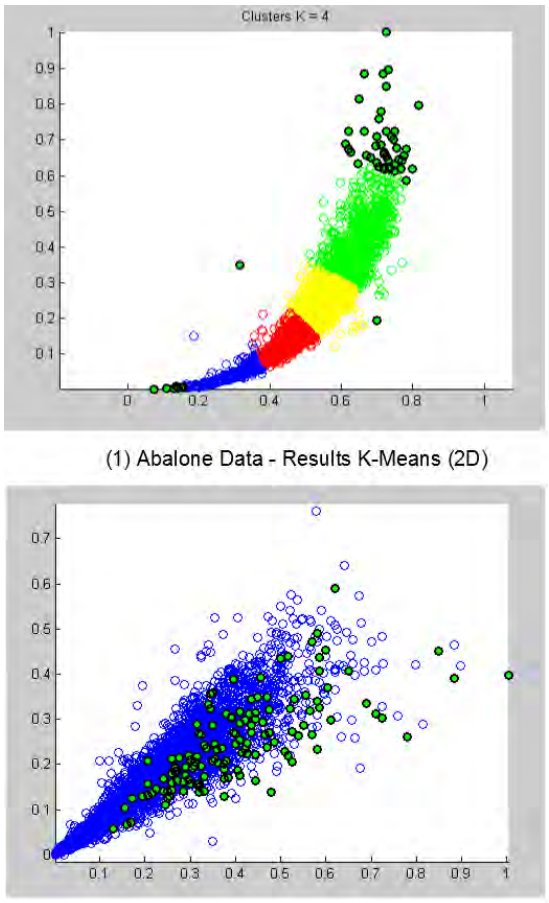


Figure 4: *K-Means* Clustering results

of *k* then to determine the number of clusters, K-means needs to perform extra processing, and the PRE plot[4] would impose O(N^3) computational cost which is very high. *Figure 3* shows the results after applying the algorithm on Abalone data set as used in[3].

4.3 Conclusion

The clustering algorithm applied on these nodes produced very good results. Although this clustering algorithm produced agreeable results yet it was dependent on two user-defined parameters: number of clusters "k" and radius of the circular boundary "r". So, we moved on to a more efficient clustering technique: clustering by means of ellipsoidal boundaries.

5 Clustering Ellipsoids

5.1 Introduction

In this new approach, which has been described in[4], each sensor reports a *hyperellipsoid* that characterizes the locally observed distribution of measurements at that sensor. These hyperellipsoids are then reported to a central base station, which clusters these local hyperellipsoids so that these resulting clusters characterise the underlying modes of the distribution of measurements in the network as a whole. This set of hyperellipsoidal clusters can be used by each sensor node as the basis for anomaly detection. Both *local and global* outlier detection techniques have been discussed in this paper.

5.2 Preliminary

General form of an elliptical boundary is given by

$$ell(a, A;\ t) = \{x \in \Re^p \mid (x-a)^T A(x-a) = t^2\} \quad (1)$$

where a is the center of the ellipsoid and t is its effective radius.

In statistical terms, the Mahalanobis for a multivariate observation x from a set of measurements with mean m and covariance V can be formulated as follows:

$$\|x-m\|_{V^{-1}} = \sqrt{(x-a)^T V^{-1}(x-m)} \quad (2)$$

By combining equations (1) and (2), we can see that any observation whose Mahalanobis distance from the sample mean is t will reside on a hyperellipsoidal boundary induced by *V with effective radius t*, i.e.,

$$B(m, V^{-1};\ t) = \{x \in \Re^p \mid\mid\mid x-m \parallel^2_{V^{-1}}\} = t^2 \quad (3)$$

To differentiate between normal and anomalous measurements, we choose t so that the hyperellipsoid covers most of the data distribution. If we assume that the data distribution at each sensor is normal, then a hyperellipsoid with t=3 standard deviations from the mean would cover *99%* of the data.

We use the observed sample mean m_j and covariance V_j of the measurements at node N_j to define an elliptical boundary for normal measurements, that is three standard deviations from the observed mean, i.e.,

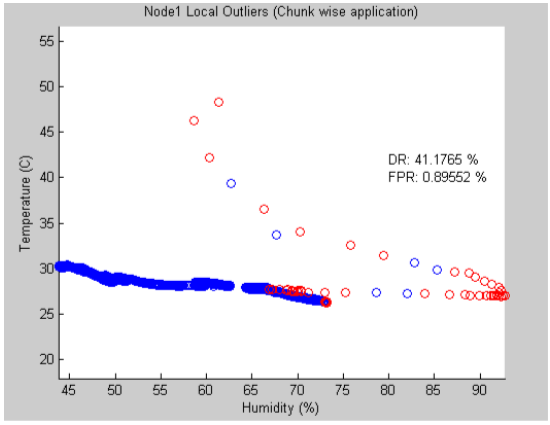
$$NP_{X_j} = \{x_k \in X_j \mid x_k \in ell(m_j, V_j^{-1};\ t \leq 3)\} \quad (4)$$

A data point which falls outside this boundary is considered to be *locally* anomalous.

5.3 Algorithms

5.3.1 Local Detection

This technique tries to fit a *ellipsoidal distribution* to the data set. This is done in several steps, first of which is to determine the mean of all the points in the data set. This mean value is set as the center of the ellipsoid that will be fit through the data set. Then this mean is subtracted from all the data points. Next, the covariance matrix of the data set is calculated. Afterwards *B* is calculated as mentioned in equation (3). *B* is a vector that contains a Mahalanobis distance corresponding to all the data points in the data set. If the value in vector *B* corresponding to a data point is more in magnitude than t^2, then it is declared as an outlier. Here *t* is the standard deviation of a distribution. From the papers we learnt that setting *t* to *3* is a appropriate measure of outlier-ness.



(1)

Figure 5: Detection for Streaming Data in chunks

5.3.2 *Global Detection: Positive Root Eigen Value (PRE) Technique*

This technique, basics of which are provided in[4][5], is applied for global outlier detection (after the local algorithm finishes running on each node) to minimize the number of distributions which call fully characterize all the data. And in order to do that we need to define a parameter (a similarity or distance measure between a pair of ellipsoids) to compare and cluster ellipsoids. Let m_1 and m_2 be the centers of ellipsoids E_1 and E_2, respectively. The similarity function S(E_1,E_2) is then defined as

$$S(E_1, E_2) = e^{-\|m_1 - m_2\|} \tag{5}$$

For *N* nodes we have $\binom{N}{2}$ combinations and thus $\binom{N}{2}$ similarity matrices[5]. *Eigen values* are calculated for each of these matrices and then acccording to the PRE Technique, ellipsoidal distributions of '*large*' eigen valued nodes are picked. And these distributions collectively cover approximately 99% of the data. The information/parameters of these **Global Ellipses** is then sent to all the nodes which then calculate their *Global Outliers* respectively.

5.3.3 *Detection for Streaming Data*

Data that we get in real time is continuously streamed. So, to simulate that continuous streaming, we divided our data from each node into equal chunks and applied the ellipsoidal clustering on each chunk. Each new chunk's ellipse was merged with the previous ellipse so as to get one ellipse at each node which has all chunks' ellipses merged in it. Results are shown in figure 5.

5.3.4 *Event Detection*

After the ellipsoidal clustering was applied on the chunks and the outliers were found we extended this technique to Event Detection. The outliers of one attribute of a node were cross correlated with outliers of other attributes of the same node and with the outliers of the same attribute of every other node. If any outliers were correlated enough (*e.g.* correlation co-efficient was more than 0.80), they were declared as an *Event*.

We took the outliers of one node after every (lets say) 10 seconds (we divided the whole data that we had into 70 seconds) and then found its cross correlation with outliers of all other nodes at the same time (after 10 seconds had passed like for the 1st node). This is just the *cross* correlation in which we find the correlation between two vectors with some lag introduced (zero lag means corresponding elements are being compared and other lag values mean that 1st whole vector is being compared with the second vector with some of the elements of second vector at either ends omitted[2]). This way we get to know the correlation with the outliers that occurred in previous time as well (as we do it every 10 seconds). Thus by doing this in **chunks** we have a more appropriate check on events rather than comparing each stream (data for every second) of node data.

5.4 Experimental Results

At first we applied the ellipsoidal clustering method to the whole data[6] at once and got the outliers. Afterwards to get global outliers, we merged the four ellipses from the nodes using PRE technique as mentioned in[5] and made two global ellipses each of which was made by merging two ellipses that were closer to each other. Then we again calculated the outliers with two global ellipses.

Below are the plots of ellipsoids and outliers for the respective nodes and the *Global* ellipsoids and outliers after application of PRE Technique(Merged ellipses plots contain data from all nodes, red and purple color points are all global outliers). *Figure 4* represents the result of both Global and Local Outlier Detection applied on $Node_1$ with DRs and FPRs. *Figure 5* represents the result of Local detection applied on all nodes one by one and of Global detection applied on whole data at once (done at the sink node).

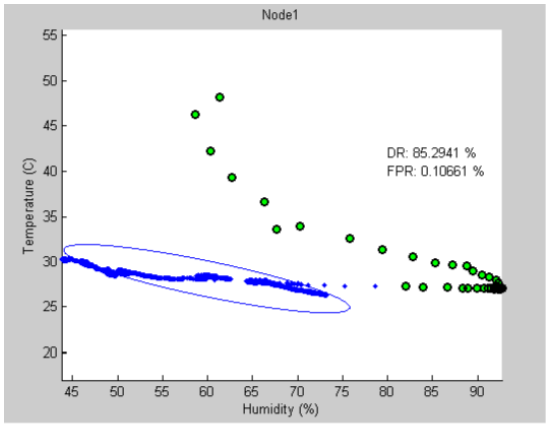
5.5 Complexity

To evaluate the computational cost of clustering ellipsoids, each sensor node N_j with n_j number of data points each having *p* attributes, needs to compute the covariance and mean, which can be done in $O(n_j p^2)$. So this can be done in linear time with respect to the number of data points. In the context of sensor networks the number of feature (dimensions) are usually very low, and the p^2 term, which relates to finding ellipsoidal boundary, is also very small. The most computationally intensive part is the PRE plot computation at the base station, which is of order $O(m^3)$, in which *m* is the number of sensors. Note that m is substantially smaller than the number of data points ($m \ll N$). So overall the algorithm is of order *O(N)* with respect to the number of data points. In the centralized approach no processing takes place in each node, and the computational complexity for K-means is of order *O(IcpN)*, in which *I* is the number of iterations. To determine the number of clusters, K-means needs to perform extra processing, and the PRE plot would impose $O(N^3)$ computational cost which is very high.

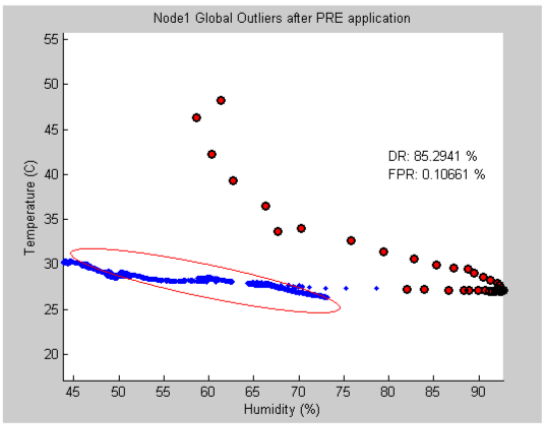
5.6 Conclusion

Although this method is computationally more complex than other, but its accuracy doesn't depend on any user specified pa-

[2]Matlab function **xcorr()** property



(1)



(2)

Figure 6: (1) Local Outlier Detection Result, (2) Global Outlier Detection (PRE) Result

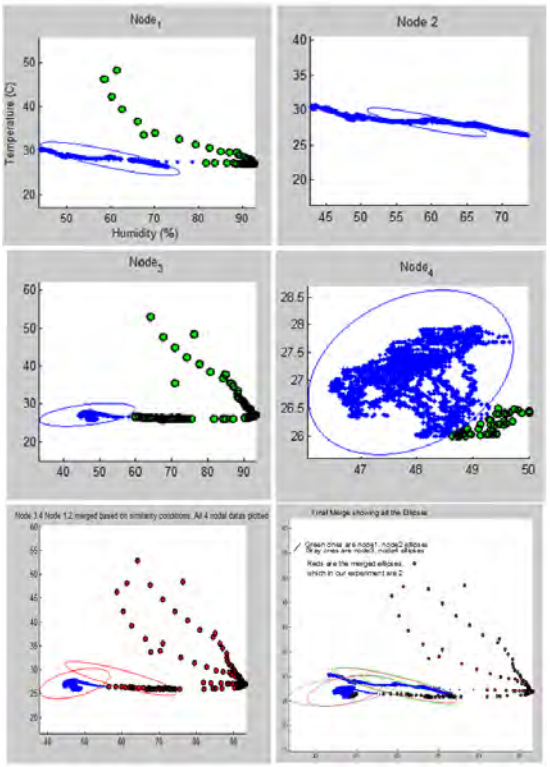


Figure 7: Global Outlier Detection: Positive Root Eigen Value Technique Results

Techniques	Attribute	Local	Global	Event Detection	Outlier Degree/Score	Parameters Dependence
Aggregation Tree	Univariate	Yes	Yes	No	Yes	High (N, K)
Fixed width Clustering	Bivariate	Yes	No	No	No (Can be added)	High (w, k)
K-Means Clustering	Multivariate	Yes	No	No	No (Can be added)	Medium (r, k)
Ellipsoidal Clustering	Multivariate	Yes	Yes	Yes (Our extension)	No (Can be added)	Negligible

Table 1: Comparison of the Outlier Detection Techniques

rameter. It gave us perfectly agreeable results for all the nodal data. *Table 1* shows a comparison between all the methods of outlier detection discussed in this paper. As evident from the table, the *Ellipsoidal Clustering* technique is the one we should choose because

- Supports multivariate data and thus allows multiple attributes of the environment to be analysed
- Negligible dependence on the user defined parameter t
- Both a *local* and *global* detection technique
- Extended for event detection using *Temporal Correlations*

6 Online Methods

The clustering technique that mentioned above isn't the optimal approach for our nodes. In that technique we took large chunks of data and made clusters to estimate the boundary of the ellipses and make decisions about the outliers. But, the problems with this approach are:

- What if the **memory** capacities of our nodes are low? We won't be able to store larger chunks of data for processing. And the chunk size is essential for the clustering algorithm.

- Secondly, what if the computational abilities of our nodes are low? To compute the parameters for a large chunk of data requires ample computational powers which might not be available in our nodes.

- Thirdly, we haven't taken into account any temporal changes in our data. So, our algorithm will not be available to track the temporal changes in the data.

So there is a need to design techniques for online data streams which use temporal and spatial correlations to not only *detect* but *identify* events as well.

7 Incremental elliptical boundary estimation[7]

After identifying the issue we implemented our first *incremental* elliptical boundary estimation technique described by Masud et al[7]. It offers two improvements: an iterative formula for the estimation of ellipsoidal boundary and a forgetting factor in the iterative method for tracking the changes in the data.

The batch technique calculated the boundaries over a training period and required to keep the measurements in the memory for the whole duration of the training period. Also, all the measurements were processed in a batch mode. Our approach in this method goes as follows:

- Initially we wait for some points to accumulate from the stream. Once we have k points, we calculate the initial *mean* and the *covariance* by the following relations:

$$m_k = \frac{1}{k}\sum_{j=1}^{k} x_j$$

$$S_k = \frac{1}{k-1}\sum_{j=1}^{k}(x_j - m_k)(x_j - m_k)^T$$

- Then the first hyper-ellipsoidal boundary is calculated as follows:

$$e_k(m_k, S_k^{-1};\ t) = [x \in R_p \mid (x_j - m_k)^T S_k^{-1}(x_j - m_k) \leq t^2]$$

This makes a *hyper-ellipsoid* of radius t centered at m_k with covariance matrix S_k and the outliers are detected as:

$$(x_j - m_k)^T S_k^{-1}(x_j - m_k) > t^2$$

- We need some initial points to start our calculations because if we start our calculations with lesser number of points it results in a singular covariance matrix.

- Here comes the difference from the batch approach. If we were doing batch technique, we would have waited for another chunk to accumulate and then proceeded with our calculations. But here we can update the parameters of the ellipsoidal boundary by the following formula:

$$m_{k+1} = m_k + ((x_{k+1} - m_k))/(k+1)$$

$$S_{k+1} = \frac{(k-1)S_k}{k} + \frac{(x_{k+1} - m_k)(x_{k+1} - m_k)^T}{(k+1)}$$

- So, now we only need to store one measurement, mean, covariance matrix and outliers.

7.1 Forgetting factor

As we mentioned, the batch technique did not take into consideration the temporal changes in the data. To enable the iterative algorithm to track the temporal changes in the monitored environment, we introduce a new term called the **forgetting factor** (λ) in the algorithm. In this approach, we incorporate the forgetting factor in the update equations of mean and covariance as follows:

$$m_{k+1,\lambda} = \lambda m_{k,\lambda} + (1-\lambda)x_{k+1}$$

$$S_{k,\lambda} = \frac{1}{k-1}\sum_{j=1}^{k}\{(x_j - m_{k\lambda})(x_j - m_{k\lambda})^T\}$$

$$S_{k+1,\lambda} = \frac{\lambda(k-1)S_{k\lambda}}{k} + \frac{\lambda^2(x_k + 1 - m_{k\lambda})(x_k + 1 - m_{k\lambda})^T}{k}$$

The value of forgetting factor is between 0 and 1. What the forgetting factor does is that it assigns a weight of λ^j to the measurement from j samples ago. The forgetting factor method increases the significance of the current measurement compared to previous measurements.

7.1.1 Problem

We have a problem with incorporating forgetting factor in the update equations. Let us have a look again at the update equation of the covariance

$$S_{k+1,\lambda} = \frac{\lambda(k-1)S_{k\lambda}}{k} + \frac{\lambda^2(x_k + 1 - m_{k\lambda})(x_k + 1 - m_{k\lambda})^T}{k}$$

$$S^{-1}_{k+1,\lambda} = \frac{kS^{-1}_{k\lambda}}{\lambda(k-1)} + [I - \frac{(x_k + 1 - m_{k\lambda})(x_k + 1 - m_{k\lambda})^T S_{k\lambda}}{k - \frac{1}{\lambda} + (x_k + 1 - m_{k\lambda})^T S_{k\lambda}(x_k + 1 - m$$

For very large values of k, the term in brackets in the inverse equation approaches I and thus the characteristic matrix (S^{-1}) updates approaches $\frac{inv(S_k\lambda)}{\lambda}$. So as k grows, the effect of newer readings grows small and the update of characteristic matrix is controlled by $\frac{inv(S_k\lambda)}{\lambda}$.

7.2 The Effective N approach

To deal with the issue of large k for tracking, we simply use a constant n_{eff} instead of k in the inverse equation when $k \geq n_{eff}$. The idea is that after $k \geq n_{eff}$ weights assigned to data

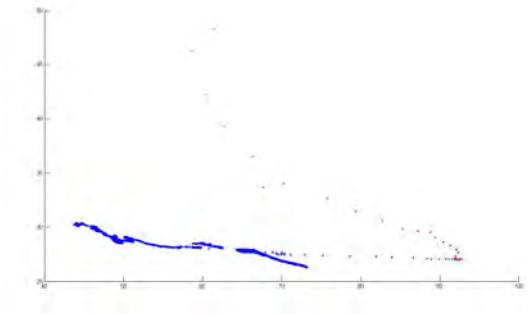


Figure 8: Outliers in node 1 with forgetting factor

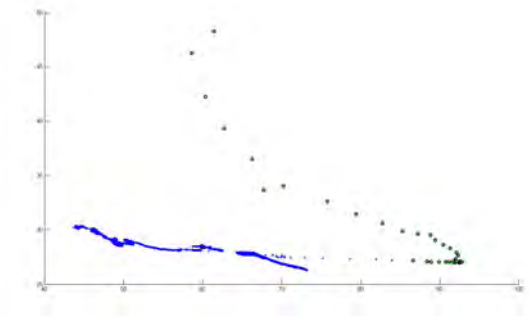


Figure 9: Outliers in node 1 with neff

samples approach zero, i.e. $\lambda^k \approx 0$, which means that the corresponding samples have been (almost) forgotten completely. The suggestion is to use $n_{eff} = 3\tau$, where $\tau = 1/(1-\lambda)$. τ is known as the **memory horizon** of the iterative algorithm with an exponential forgetting factor λ.

$$S_{k+1,\lambda} = \frac{\lambda(n_{eff}-1)S_{k\lambda}}{n_{eff}} + \frac{\lambda^2(x_k+1-m_{k\lambda})(x_k+1-m_{k\lambda})^T}{n_{eff}}$$

outliers in node 1 with neff

So, for $k \geq n_{eff}$ we have the update equations as follows:

The results from the simulations that we did implementing different versions of the iterative algorithms with their *detection* rates and *false positive* rates are shown in Figures

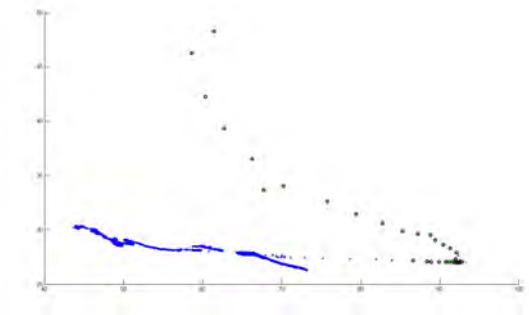


Figure 10: Outliers in node 1 without forgetting factor

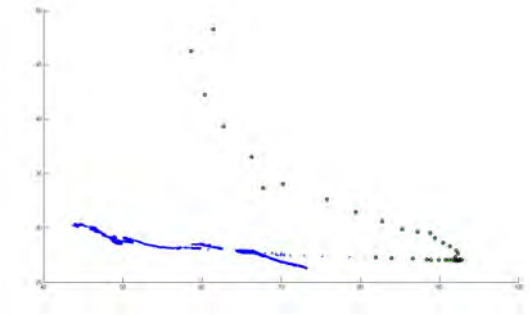


Figure 11: Outliers in node 1 without iterative

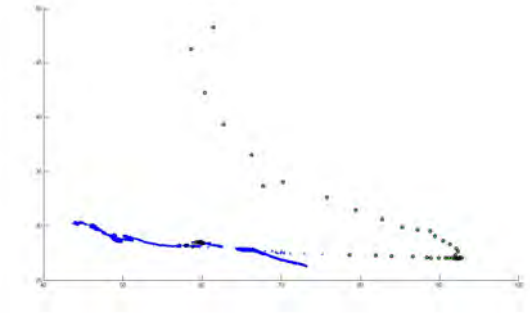


Figure 12: Outliers in node 1 without neff

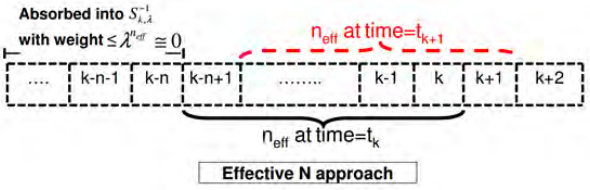


Figure 13: Effective N approach

8 A Simpler Incremental method for calculating the elliptical boundary

Equation for *Mahalanobis Distance* of a Point *X* takes three arguments: the coordinates of *X*, The mean *M* and the covariance matrix *S*. It is not practical to store all the points of a streaming data and update *M* and *S*. We therefore need to find an Iterative for incrementing both *M* and *S*.

Means can simply be incremented by adding the new data point to the previous mean, weighted by the number of points and dividing the sum by the total number of points as given in the equation:

$$M_{k+1} = \frac{1}{k+1}(M_k \times k + X_{k+1})$$

Incrementing the covariance matrix directly from the previous covariance matrix is not as simple, but if we can find the covariance matrix 'S' in terms of means we can use the iterative equation above to increment the covariance and hence the elliptical boundary. Bellow it is shown how, by replacing S by its definition we can write the equation for the *Mahalonobis* distance entirely in terms of means:

Mahalonobis distance is given by the equation:

$$\sqrt{(X_{1\times n} - M_{1\times n})^T S^{-1}_{n\times n}(X_{1\times n} - M_{1\times n})} = t$$

Replacing S with:

$$S_{n\times n}(t) = M_{X^2}(t) - (M_X(t))^T(M_X(t))$$

Where:

$$M_{X^2} = E[X(t)^T X(t)]$$

We get the following equation:

$$\sqrt{(X - M_X)^T_{1\times n}(M_{X^2} - M_X^T M_X)^{-1}_{n\times n}(X - M_X)_{1\times n}} = t$$

Now we can write down the iterative algorithm by incrementing the means by the equation mentioned above:

$$M_{k+1} = \frac{1}{k+1}(M_k \times k + X_{k+1})$$

$$M_{X^2,k+1} = \frac{1}{k+1}(M_{X^2,k} \times k + X^T_{k+1}X_{k+1})$$

$$B_{k+1} = \sqrt{(X - M_X)^T_{1\times n}(M_{X^2} - M_X^T M_X)^{-1}_{n\times n}(X - M_X)_{1\times n}}$$

Now, if $(B > t^2)$, then, X_{k+1} = *outlier*. Fig.14 shows how the elliptical boundary follows the trend in data when the iterative algorithm is applied, Fig.16 shows the results of the algorithm on actual experimental data.

8.1 Event Detection

When outliers occurring are correlated in both time and space they correspond to an *event*.

8.1.1 Temporal correlation:

In our algorithm for event detection each node maintains an *event array*. Incoming outliers are stored in the event array. The size of the event array increases with each incoming outlier. The

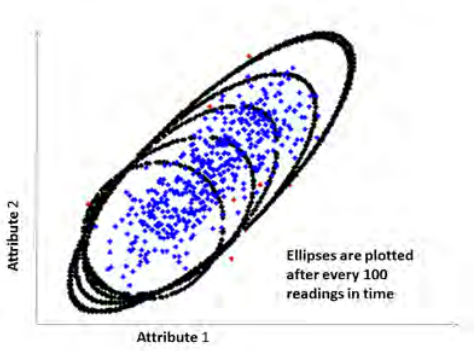


Figure 14: Elliptical boundary trend

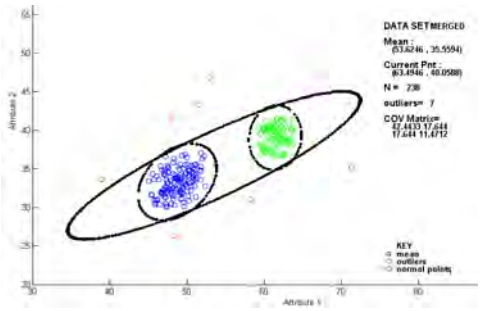


Figure 15: Merging of two Ellipsoids

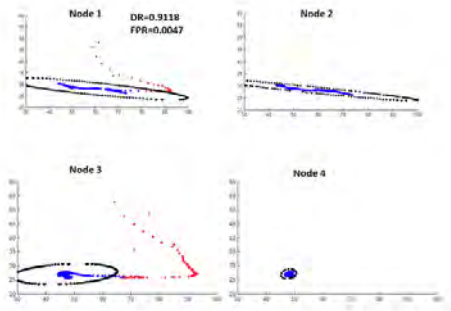


Figure 16: Results

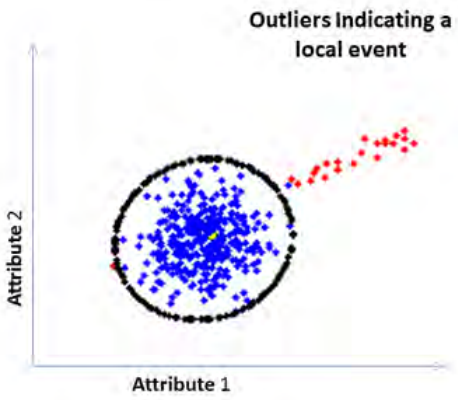


Figure 17: Event Detection

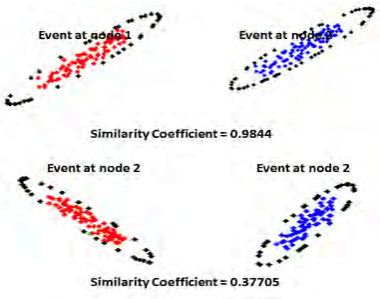


Figure 18: Bhattacharia Coefficients between event clusters of two nodes

event array is emptied if no outlier is detected for a certain time period. This way the size of the array will only increase if the outliers are not significantly spaced in time. If the size of the array exceeds a certain size a local event is declared at the node. Fig.17 below shows what a *local* event looks like when plotted for a two attribute data.

8.1.2 Spatial Correlation:

Once a local event is detected at a node we check if similar events occurred at surrounding nodes. The similarity can be found by calculating the Bhattacharya similarity coefficient between the Event Arrays of the nodes. The equation for *Bhattacharya coefficient* takes in the means and covariance of two clusters as arguments and returns their similarity on a scale from 0 to 1. This is the equation for the *Bhattacharya coefficient*:

$$S = e^{(-\frac{1}{8}(m1-m2)(\frac{v1-v2}{2})^{-1}(m1-m2))} \times \sqrt{\frac{\sqrt{det(v1)det(v2)}}{det(\frac{v1-v2}{2})}}$$

Fig.18 shows examples of *Bhattacharya* coefficients between event clusters of two nodes: If the similarity coefficient between events at surrounding nodes is high then the event is spatially correlated. An Event is only declared if it is correlated in both time and space. If it is only correlated in time and not in space i.e. it does not occur at the surrounding nodes, it is declared a sensor fault.

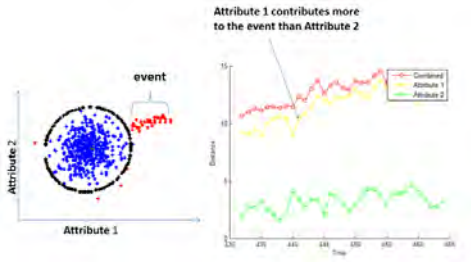


Figure 19: Event Identification

Techniques	Memory	Correlation	Local	Global		Parameter Dependence	Online Streaming
Aggregation	High	-	Yes	Yes		High (N, K)	Batch
K-means	High	-	Yes	No		Medium (r, k)	No
Ellipsoidal	Medium	-	Yes	Yes		Negligible	Batch
Ellipsoidal (Incremental)	Lowest	Temporal + Spatial	Yes	Yes		Negligible	Yes, Point by Point

Figure 20: Comparisons of implemented techniques

8.2 Event Identification

Once an event is detected, we need find which attribute contributed most to the event.

8.2.1 Taking one attribute and excluding the rest

- Take the readings of one Attribute and ignore the others
- Calculate the distance of outliers based on only that attribute
- The larger the distance the more it contributes to the event.

8.2.2 Excluding one attribute and taking the rest

- Find the distance of outliers based on all attributes
- Remove one attribute at a time and find the distance based on the rest
- Find the change in distance by ignoring a attribute
- The attribute whose removal causes the largest change in distance contributes most the event.

Fig.19 shows the contributions (distances) of the outliers of the event over the duration of the event. It can be seen that Attribute 1 contributes more to the event than Attribute 2 as confirmed by the plotted results on the left of Fig.19.

9 Hardware Overview[8]

We are going to use Synapse RF266PC1 RF Module. Its a 2.4 Ghz transceiver and can be easily configured for wireless sensor

Figure 21: Synapse RF266PC1 - 2.4GHz (chip antenna)

Figure 22: Synapse RF266PC1 - 2.4GHz (chip antenna)

nodes. It has built in *Python virtual Manchine* and its built in *SNAP* network protocol. It can be programmed *over the air* and different analogue and digital sensors can be interfaced easily by writing python script and uploading it to the node.Figure 21 shows the node. As you can see in figure 22, the node is very small in size. It is also *pin-compatible* with sockets already designed in for Digi Internationals XBee and XBee-PRO RF modules. This module supports data rates from 250kbps up to 2Mbps and distances of up to 3 miles because of a high gain 22 dBm transmit amp; a built in receive amp provides a -107 dBm receive sensitivity! These small, low-powered, 2.4 GHz transmitter-receiver modules are modest in power consumption as low as 0.37 μA. *Portal* 25 - a software provided with the nodes which uses SNAPpy (a subset of python) to program nodes over the air - will be used to link the base station with the rest of the network.

- 15 GPIO, and up to 4 10-bit A/D inputs
- One UART port for control or transparent data
- Low power modes: 0.37 A with internal timer running

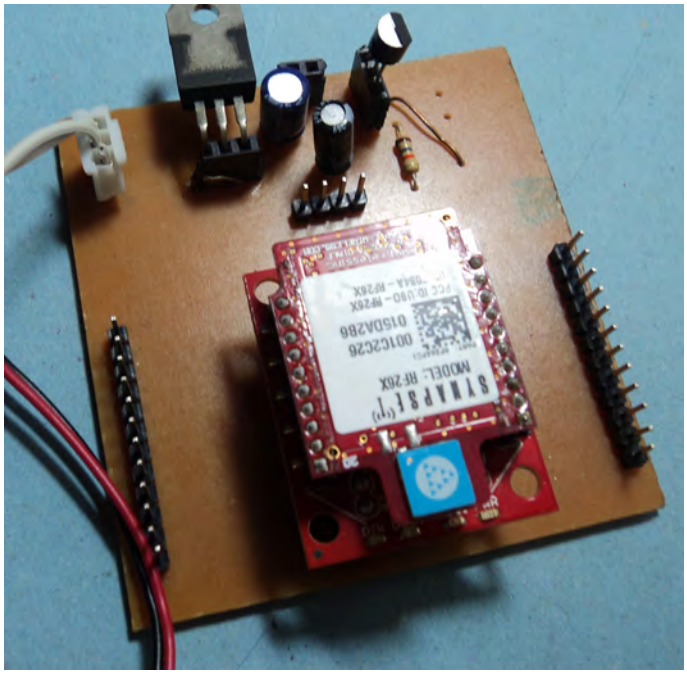


Figure 23: Designed PCB with LM35 Temperature Sensing circuit

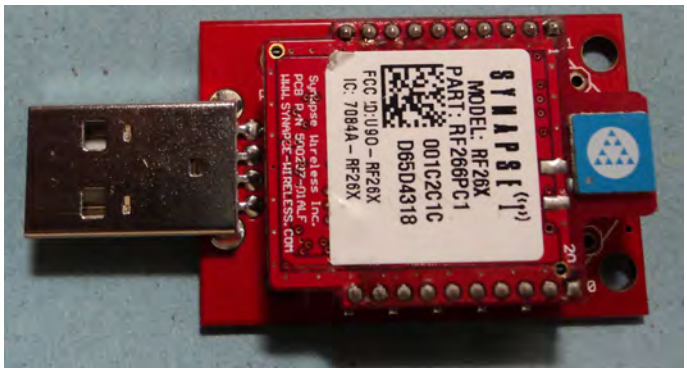


Figure 24: USB programmer

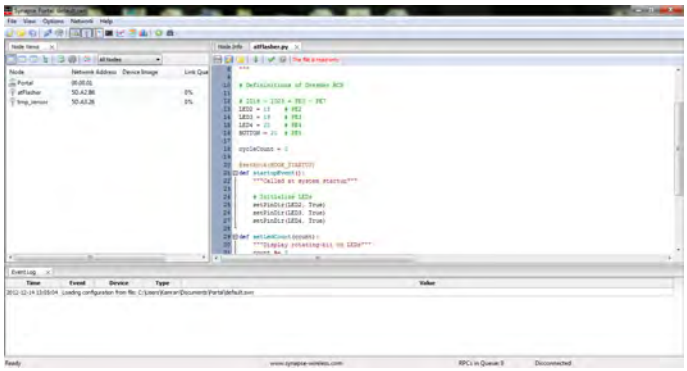


Figure 25: Portal Software

- 128k flash, 56k free for over-the-air uploaded user apps
- Up to 4000 feet, LoS at 250kbps with on-board chip antenna
- Socket-able or solder-able
- SNAP Cloud Compatible
- 250 kbps to 2 Mbps Data Rate
- 2.4 GHz RF Frequency
- Spread Spectrum (DSSS) technology
- Receive Amplifier (10 dBm)
- Transmit amplifier (20 dBm)
- Chip antenna

10 Progress

So far we have been able to effectively detect an outlier and establish it as a false alarm or an actual and event. Once an event is detected we have successfully identified the attribute which has caused the event. We have designed PCBs for the mounting of Synapse RF266PC1 RF modules and started programming the nodes using Portal software which uses SNAPpy (a subset of python) to program nodes over the air. We have also started working on the hardware implementation of the algorithms for event and outlier detection.

11 Updated Timeline

So far we have followed the time line we provided in our proposal. The new tentative time line is :

Winter break:
1)Getting to further know the hardware.
2)Research on event localization will also be done

Spring semester:
1)Implemeting algorithms on event localization.
2)Hardware implemetaion of event/outlier detection and identification using WSNs.
3)Designing a Python based GUI software which will give a user full command over the nodes in the network and continuously provide the user with different graphs, etc. the outliers and events being detected at different areas of the WSN.
3)Writing a research paper based on our sproj.

12 Upcoming Challenges

- Implementation of the above mentioned algorithms keeping in mind the communication overhead, limited memory and the computation and consumption power of the motes.
- Calibration of WSNs.
- Increasing the accuracy of outlier detection and localization which needs research and development of more efficient techniques on our part.

References

[1] Yang Zhang, Nirvana Meratnia, and Paul Havinga: *Outlier Detection Techniques for Wireless Sensor Networks: A Survey*

[2] Kejia Zhang, Shengfei Shi, Hong Gao, and Jianzhong Li: *Unsupervised Outlier Detection in Sensor Networks Using Aggregation Tree*

[3] Manzoor Elahi, Kun Li, Wasif Nisar, Xinjie Lv and Hongan Wang:Anomaly Detection by Clustering Ellipsoids in Wireless Sensor Networks *Effcient Clustering-Based Outlier Detection Algorithm for Dynamic Data Stream*

[4] Masud Moshtaghi, Sutharshan Rajasegarar, Christopher Leckie, Shanika Karunasekera: *Anomaly Detection by Clustering Ellipsoids in Wireless Sensor Networks*

[5] Shafagh Fallah, David Tritchler and Joseph Beyene: *Estimating Number of Clusters Based on a General Similarity Matrix with Application to Microarray Data*

[6] S. Suthaharan, M. Alzahrani, S. Rajasegarar, C. Leckie, and M. Palaniswami: *Labelled data collection for anomaly detection in wireless sensor networks, in Intelligent Sensors* Sensor Networks and Information Processing (ISSNIP), 2010 Sixth International Conference on, pp. 269 274, dec. 2010.

[7] Masud Moshtaghi et al: *Incremental Elliptical Boundary Estimation for Anomaly Detection in Wireless Sensor Networks* NICTA Victoria Research Laboratories.

[8] Sparkfun Electronics: *https://www.sparkfun.com/products/11279*

[9] : N.Shahid, S.B.Ali, K.Ali, M.A.Lodhi, O.B.Usman, I.H.Naqvi: *Clustering based Event Detection for Wireless Sensor Networks*