CHAPTER 2

MODIFIED ADAPTIVE HUFFMAN COMPRESSION ALGORITHM

2.1 INTRODUCTION

Wireless Sensor Networks are popularly used to monitor a variety of physical environments. The network nodes gather data and process it to provide information in an appropriate manner as demanded by the primary application when required. Sensor Networks have some distinct features such as limited computation, distributed processing and degree of correlation. The gathered information is communicated to gateway nodes or sink nodes. Sensor nodes are designed with a limited battery power and they need to be unobtrusive. This mandates that they carry only a small battery, thus limiting the energy available for functioning and necessitating a low-power operation in order to conserve energy to prolong the useful life of the sensor node.

Francesco Marcelloni et al (2008), state that radio communication is often the principal area of energy consumption. The extension of sensor node lifetime is generally achieved by reducing transmission/receptions of data using data aggregation, compression techniques. The data compression algorithms are needed for these networks to reduce the amount of data communicated to the sink. The objective of the proposed work is to minimize the total number of bits required to be transmitted from the sensor node to reduce the energy consumed by the sensor node. The existence of temporal correlation in sensed data is considered as an advantage. A modified Huffman
Data compression algorithm suitable for WSN is proposed in this chapter. The proposed algorithm does not require the statistics of the sensed data though however encodes the difference of the current and the previous value of the sensed data. This algorithm provides a good compression ratio for both highly correlated and medially correlated sensor node data. The next section provides a brief review of literature of compression algorithms for wireless sensor network.

2.2 RELATED WORK

In wireless sensor networks, the power consumption problem has evinced a great deal of interest within the research community with more number of proposals being made for data compression. Spatial and temporal correlation is an important characteristic of sensor network. According to Akylidiz et al (2002), typical WSN applications require spatially dense sensor deployment in order to achieve satisfactory coverage. As a result, multiple sensors record information about a single event in the sensor field. Due to the high density in the network topology, spatially proximal sensors observations are highly correlated with decrease in inter node separation. Spatial correlation between sensor nodes can be modelled by computing cross correlation of sensor data. Cross Correlation is a standard method of estimating the degree to which two series are correlated. Consider two series \( x(i) \) and \( y(i) \) where \( i=0,1,2...N-1 \). The cross correlation \( r \) at delay \( d \) is defined as in Equation (2.1)

\[
r = \frac{\sum [(x(i) - mx) \ast (y(i - d) - my)]}{\sqrt{\sum (x(i) - mx)^2} \sqrt{\sum (x(i) - mx)^2}}
\]  

(2.1)

where \( mx \) and \( my \) are the means of the corresponding series.
As defined by Kusuma et al (2001), Sensor nodes sense a physical phenomenon and the nature of the phenomenon constitutes the temporal correlation between each consecutive event of the sensor node. The degree of correlation between consecutive sensor measurements may vary according to the temporal variation characteristics of the phenomenon. A high correlation is likely to indicate a periodicity in the signal for the corresponding time duration. The correlation is calculated between a series and a delayed version of itself.

The correlation coefficient at lag k of a series \( x_0, x_1, x_2, \ldots, x_{N-1} \) is normally given as in Equation (2.2)

\[
\text{Autocorr}(k) = \frac{\sum_{i=0}^{N-1} (x_i - mx)(x_{i+k} - mx)}{\sum_{i=0}^{N-1} (x_i - mx)^2}
\]  

(2.2)

where \( mx \) is the mean of the series.

The existence of above mentioned spatial and temporal correlations bring significant potential advantages for the development of efficient compression algorithms well suited for WSN paradigm.

Liu et al (2005) have proposed a piecewise linear approximation method to approximate the time series into a number of line segments. The main drawback is that the algorithm is not adaptive to changing correlation. Ngoc et al (2008) in their work have described an alternative algorithm for piecewise linear approximation which requires a shorter running time. Detailed analysis of the algorithm with different correlation characteristics is not considered.
Meng et al (2004) perform temporal suppression in which a node does not transmit a value if it has not changed since last reported. The base station in turn assumes any unreported values as unchanged data. This scheme is effective when values seldom change. On the other hand, when large scale change occurs in an area, all nodes must report to the base station, incurring a high cost.

Adler et al (2005) proposes pull-based data collection technique where the base station takes an active role in the process. The proposed algorithm achieves the information-theoretical lower bound on the number of bits sent by the sensor nodes, while at the same time overloading most of the compute-intensive work to the base station. However, the number of bits received by the sensor nodes may be very high. Sebastian Puthenpurayil et al (2007) have studied energy consumption tradeoffs associated with data compression, particularly in the context of lossless compression for acoustic signals. However this work does not consider the correlation of the sensed data. Ming Bo Lin et al (2006) have proposed a hardware architecture for lossless data compression and decompression algorithm. This work has not exploited the correlation characteristics of the data which is an important feature on sensor node.

Vasanth Iyer et al (2008) in their work have proposed a probability model to efficiently compress data at cluster heads. The compression algorithms are used at cluster heads to bring down the number of transmitted bits. This is suggested due to the complexity memory requirements of compression algorithms. But this algorithm requires all the data to be communicated to the cluster head where compression is performed.
2.3 PROPOSAL OF A MODIFIED ADAPTIVE HUFFMAN ALGORITHM

The key challenge in the design of data compression algorithms is that the algorithms should adapt to the changing correlation of the sensed data. Most of the data compression algorithms designed for sensor networks are based on in-network processing and are suitable when there is a high correlation in the sensed data. The basic idea of the proposed Modified Adaptive Huffman algorithm is its adaptation to the changing correlation in the sensor data by judiciously constructing the binary tree. Also, the number of levels in the binary tree is greatly reduced when compared with the traditional algorithms like Huffman Coding and Adaptive Huffman Coding. In the next section, the established concept of Huffman coding is discussed.

2.3.1 Huffman Coding Concept

There are two main categories of data compression namely the lossless data compression and lossy data compression. The lossless compression algorithms usually generate a statistical model of the data and map the data to bit strings based on the generated model. But in the lossy data compression, data is often transformed into a new space using appropriate transformation kernel functions. In the new space, the data information or signal energy is usually concentrated in a few coefficients. Hence, the compression can be achieved after quantization and entropy coding. The Huffman coding is an entropy encoding algorithm used for lossless data compression. The term refers to the use of a variable length code table for encoding a source symbol. The variable length code table has been derived in a particular way based on the estimated probability of occurrence for each possible value of source symbol. According to Khalid Sayood (2004), Huffman coding uses a specific method for choosing the representation for
each source symbol, resulting in a prefix code. Huffman coding requires the prior knowledge of source symbols.

The Huffman procedure is based on two observations regarding optimum prefix codes.

1. In an optimum code, symbols that occur more frequently, i.e., they have a higher probability of occurrence will have shorter codeword than symbols that occur less frequently.

2. In an optimum code, the two symbols that occur least frequently will have the longer length

Static Coding requires prior knowledge of the probabilities of the source sequence. From Khalid Sayood (2004), if this knowledge is not available, Huffman coding becomes a two pass procedure: the statistics are collected in the first pass and the source is encoded in the second pass. In order to convert this algorithm into a one pass procedure, Faller (1973) and Gallagher (1978) have independently developed Adaptive algorithms to construct the Huffman code based on the statistics of the symbols already encountered. These were later improved by Knuth (1985) and Vitter (1987).

Theoretically, to encode the \((k+1)\)th symbol using the statistics of the first \(k\) symbols, re-compute the code using Huffman coding procedure each time a symbol is transmitted. However, this would not be a very practical approach due to the large amount of computation involved. Hence the Adaptive Huffman coding procedures were framed.

The Huffman code can be described in terms of a Binary Tree. Consider the Figure 2.1.
The squares denote the external nodes or the leaves and correspond to the symbols in the source alphabet. The codeword for the symbol can be obtained by traversing the tree from the root to the leaf corresponding to the symbol where 0 corresponds to the left branch and 1 corresponds to the right branch. In order to describe how the adaptive Huffman code works, an extra parameter is added to the binary tree, i.e., the weight of each leaf. The weight of each external node is simply the number of times the symbol corresponding to the leaf has been encoded. The weight of each internal node is the sum of the weights of its offspring.

The Binary Tree should be balanced in order to satisfy the Sibling Property of the Binary Tree. The Sibling Property states that the weight of the left child should always be less than the weight of the right child.

In the Adaptive Huffman coding procedure, neither transmitter nor receiver knows anything about the statistics of the source sequence at the start of transmission. The tree, at both the transmitter and the receiver, consists of a single node that corresponds to all symbols Not Yet Transmitted (NYT) and has a weight of 0. As transmission progresses, nodes corresponding to symbols transmitted will be added to the tree and the tree is reconfigured using an update procedure. Before the beginning of transmission, a fixed n bit code depending whether the symbol is positive or negative is agreed upon between the transmitter and the receiver.
The actual code consists of two parts: the prefix corresponding to the code obtained by traversing the tree and the suffix corresponding to 4 or 5 bit binary representation corresponding to positive or negative data respectively.

When a symbol is encountered for the first time, the code for the NYT node is transmitted followed by the fixed code for the symbol. A node for the symbol is then created and the symbol is taken out of the NYT list. Both transmitter and receiver start with the same tree structure. The updating procedure used by both the transmitter and the receiver is identical. Therefore, the encoding and decoding processes remain synchronized.

The update procedure requires that the nodes be in a fixed order. This ordering is preserved by numbering the nodes. The largest node number is given to the root of the tree. The smallest number is assigned to the NYT node. The numbers from the NYT node to the root of the tree are assigned in increasing order from left to right and from lower level to upper level. The set of nodes with same weights make up a block.

The function of the update procedure is to preserve the sibling property. In order that the update procedures at the transmitter and at the receiver operate with the same information, the tree at the transmitter is updated after each symbol is encoded, and the tree at the receiver is updated after each symbol is decoded. The update procedure operates as follows:

After a symbol has been encoded or decoded, the control is returned to the parent to verify whether the weight of the left child has become greater than the weight of the right child. If yes, they are interchanged. Otherwise, the process is repeated until the control is transferred to the root node.
If the symbol to be encoded or decoded has occurred for the first time, a new external node assigned to the symbol and a new NYT node is appended to the tree. Both the new external node and the new NYT node are offsprings of the old NYT node. The weight of the new external node is incremented by 1. As the old NYT node is the parent of the new external node, the weight is incremented by 1 and then goes on to update all other nodes until the root of the tree is reached.

The Huffman decoding algorithm has to be essentially synchronized with the transmitter so as to decode the incoming data. The receiver also begins with the root node similar to that of the transmitter. The node parameters of the decoder are similar to that of the encoder.

If the node traversal leads to a node containing data, then the data can be directly decoded from the data available in the node of the tree. Then, the weight of the tree is incremented by one.

Using Adaptive Huffman algorithm, the probabilities were dynamically changed with the incoming data, through Binary tree construction. The Adaptive Huffman algorithm thus provides effective compression by just transmitting the node position in the tree without transmitting the entire code. Unlike static huffman algorithm the statistics of the sensor data need not be known for encoding the data.

2.3.2 Design of the Proposed Modified Adaptive Huffman Algorithm

In Static and Dynamic algorithms, the ultimate objective of compression was achieved with fixed and dynamic probabilities respectively. The main disadvantage of Static Huffman algorithm is that, it requires prior knowledge of the incoming source sequence. In Adaptive Huffman algorithm,
though the probabilities are assigned dynamically, due to the increased number of data available in the source sequence, the number of levels and hence the number of bits transmitted increases and is found to be effective only for very frequent data and data occurring at initial level of binary tree. Furthermore, the binary tree construction is based on the order of arrival of incoming data. Hence, both static and dynamic Huffman algorithms are observed to have some drawbacks.

The Modified Adaptive Huffman algorithm overcomes the disadvantages of both static and dynamic Huffman algorithm by combining the advantages of the two algorithms. Combined hybrid algorithm is proposed using the concept of “Grouping the data” into sets and the “Tree construction” for dynamic assignment of probabilities from the Adaptive Huffman algorithm.

Considering the advantages and disadvantages of both the algorithms, the third algorithm is framed which actually overcomes the limitations of both and ultimately increases the compression ratio.

2.3.3 Modified Adaptive Huffman Encoding Algorithm

Block diagram of the algorithm is given in Figure 2.2. In a sensor node, each measure $m_i$ acquired by a sensor is converted by an Analog to Digital Converter (ADC) to binary representation $r_i$ on R bits, where R is the resolution of the ADC. For each new acquisition $m_i$, the compression algorithm computes the difference $d_i = r_i - r_{i-1}$, which is an input to an entropy encoder. The main advantage of the proposed algorithm is that the assignment of the code to the incoming sensor values is dynamic in nature. Here, the Binary Tree is constructed with the incoming elements and the codes are framed by traversing the tree as in Adaptive Huffman algorithm. Elements are
grouped as nodes and codes are specifically assigned as in Static Huffman algorithm. At every update the weights at every level is checked and updated such that the higher weights will be occupying the initial stages of the tree. Each difference $d_i$ is represented as a bit sequence $b_i$ which is composed of the prefix $P_i$ and suffix $S_i$. The prefix is used to traverse the tree to identify the node and the suffix is used to represent the difference $d_i$.

![Figure 2.2 Modified Adaptive Huffman Compression](image)

**Figure 2.2 Modified Adaptive Huffman Compression**

**Table 2.1 Variable Length Code Table used in the algorithm**

<table>
<thead>
<tr>
<th>Number of bits used</th>
<th>Difference $d_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1, +1</td>
</tr>
<tr>
<td>2</td>
<td>-3,-2,+2,+3</td>
</tr>
<tr>
<td>3</td>
<td>-7,-6,...,-4,+4,+5,...,+7</td>
</tr>
<tr>
<td>4</td>
<td>-15,-14,...,-8,+8,...,+15</td>
</tr>
<tr>
<td>5</td>
<td>-31,-30,...,-16,+16,...,+31</td>
</tr>
</tbody>
</table>
Table 2.1 is used to encode the incoming data. \( d_i \) is the difference between the current value and the past value of the sensor. From Table 2.1 it can be seen that when the difference is -1 or +1, then the number of bit used for encoding is 1. That is, -1 is represented with bit 0 and +1 is represented as bit 1. Similarly 4 bits are used for encoding the difference values which are in the range -15, -14, ..., -8, +8, ..., +15. Table 1 can be easily extended or modified according to the maximum difference or user defined error threshold value. The binary tree is constructed with nodes based on the Table 2.1. The node of the binary tree in this algorithm is constructed with group of elements as shown in Table 2.1. Every node has a parameter weight \( w \) which gets incremented in the node every time the same difference is transmitted. Based on the weight parameter, the node which is having more weight is shifted to higher level in the tree. The encoding operation of the Huffman encoding block is given by the flowchart Figure 2.10. This enhances the performance in two ways.

1. It reduces the number of levels in the tree.

2. It brings the maximum possible elements to the top level of the tree.

This is further explained by considering the real time temperature data. The maximum difference in the data found to be ±1.5. The node organization in the Modified adaptive Huffman algorithm is shown in the Figure 2.3.
Figure 2.3 Node Organization in Modified Adaptive Huffman Algorithm
A sample source sequence of \([-0.2, 0.1, -0.1]\) is considered.

The transmitter begins with the root node corresponding to Not Yet Transmitted data as in Figure 2.4.

![Figure 2.4 Root Creation](image)

When the data -0.2 arrives, the tree is traversed in search of the node. Since no node is available in the tree, the code of the data -0.2 i.e., 101 is transmitted. This is followed by the insertion of the node containing the data as shown in Figure 2.5.

![Figure 2.5 Insertion of Node2](image)

![Figure 2.6 Insertion of Node1](image)
When the next data 0.1 arrives, the tree is traversed in search of the node which contains the data 0.1 in the binary tree. Since the node is not available, the code corresponding to NYT node (i.e., 0) is transmitted, followed by 1 corresponding to the position of insertion of node 1 and the code 1 corresponding to data 0.1. This is followed by the insertion of the node 1 in the tree as shown in Figure 2.6. Hence the transmitted code is 011.

![Figure 2.7 Traversing the tree](image)

Since the next data is -0.1. The tree is traversed in search of the node containing the data -0.1. Since node is already available in the tree, the weight of the node is compared with other nodes. Since its weight is greater than that of node 2, the tree is balanced in such a way that node having greater weight is shifted to right of root node and others are arranged in
according to their weight as shown in Figure 2.7. Coding consists of two part prefix and suffix. The prefix is a 1 denoting the position of node1 and suffix consisting of bit 0 denoting the value -0.1 in that node1. Figure 2.7 provides the pseudocode for encoding algorithm. The input to the program is the difference between the present and past values of sensor data and the node elements of the binary tree. A root node is created. If a node for incoming data is present, increment the weight of the node. If the node is not available then a node has to be inserted and update procedure has to be performed. The encoded data consists of prefix and suffix. The binary bit used for traversing the binary tree to reach the node is the prefix and index of the node is the suffix.

module modifiedhuffman_encode( diff , btree )
    // Creation of root in transmitter.
    createroot ( )
    // Traversal of tree in search of node.
    search ( diff , btree )
        if ( node present )
            prefix = traverse ( node )
            suffix = arrayindex ( diff )
            increment node->wt
        else
            prefix = traverse ( nyt )
            if( diff >= 0 )
                suffix = diff|4
            if( diff < 0 )
                suffix = diff|5
            nyt->lchild = createnyt( )
    // Insertion of node in the tree.
    nyt->rchild = createnode ( diff )
    nyt = nyt ->lchild
    code = <<prefix,suffix>>
    update ( btree )
    balance( btree )
endmodule

Figure 2.8 Pseudocode for Modified Adaptive Huffman Encoding Algorithm
module modifiedhuffman_decode( code[ ], btree )
    // Creation of root in the receiver.
    createroot( )
    // Traversal of tree in search of node.
    traverse ( btree )
    {
        if( code[ i ] = 0 )
            curr = curr -> lchild
        if( code[ i ] = 1 )
            curr = curr -> rchild
    }
    if( curr = node )
        read suffix
        data = data [ suffix ]
        increment node->wt
    // Encoding of data and insertion of node in the tree.
    if ( curr = nyt )
        read suffix
        if( suffix = 5bits)
            data = suffix|4
        if(suffix = 4 bits)
            data = suffix
        nyt->lchild = createnyt()
        nyt->rchild = createnode(diff)
        nyt = nyt->lchild
        update(btree)
        balance(btree)
endmodule

Figure 2.9 Pseudocode for Modified Adaptive Huffman Decoding Algorithm
Figure 2.8 and 2.9 gives the pseudocode for modified Huffman encoding and decoding algorithms. The node organization in the algorithm is given by

```c
// Node structure- parameter definition
struct node
{
    int wt, level, code, no;
    float data[16],
    struct node *lchild;
    struct node *rchild;
    struct node *parent;
};
```

Where variable data [16] corresponds to maximum size of data field, wt corresponds to the weight of the entire set of elements within the node, level corresponds to the depth of the tree at which the node is inserted, no corresponds to the number of data available in the node. The parameters lchild, rchild and parent correspond to left child, right child and parent respectively.

Both the transmitter and receiver begin with the root node corresponding to NYT. After the creation of root node, the left child will be NYT and the right child with the node containing the difference to be transmitted. Each time when the value is requested, it searches for the particular node having the value. If the value is not found, create node function is called which creates the node based on the difference. Each time the weight of the node changes, tree balancing should be done in order to make the node which has the higher probability of occurrence, be at the top of the tree.

2.3.4 Modified Adaptive Huffman Decoding Algorithm

The receiver also begins with the single root corresponding to all the data NYT. The node parameters are also defined in the same manner as
that of the transmitter. The data are grouped in the same manner as that of the transmitter to synchronize with the transmitter. As the code arrives, the tree is traversed. If the incoming code is has prefix bit as 1, the tree is traversed to the left child. If the code is 0, the tree is traversed to the right child. The code is checked until 1 occurs in order to split the prefix and suffix. Prefix gives information about the location of node in tree where as suffix give info about the difference value in that particular node. From the suffix value ‘v’ is found which denotes the node which contains the same value. Once the value ‘v’ is found, all the steps in encoder are followed in the decoder to build the tree in the same manner as has been built in the encoder. Figure 2.11 provides the flowchart for the decoding algorithm.

![Flowchart of Modified Adaptive Huffman Encoding Algorithm](image)

**Figure 2.10 Flowchart of Modified Adaptive Huffman Encoding Algorithm**
Figure 2.11  Flowchart for Modified Adaptive Huffman Decoding Algorithm

1. Start
2. Get the code
3. Go to the root node
4. Search for the corresponding node using the code
   - If prefix bit one, go to the right
   - If prefix bit is not one, go to the left
5. Is the node present?
   - Yes, use the remaining bits to get the output value
   - No, create the node
6. Increment the weight of the node and total weight of the tree
7. Traverse the tree
8. Stop
The proposed Modified Adaptive Huffman encoding algorithm, Figure 2.2, provides an effective compression algorithm by reducing the number of levels in the binary tree.

The implementation of this algorithm shows better compression ratio of nearly 60% for medially correlated data than using Marcelloni Huffman algorithm. Since the number of levels is restricted, this algorithm requires less computation than Adaptive Huffman which is an important challenge for wireless sensor nodes. The proposed algorithm is compared with the Marcelloni Huffman algorithm and Adaptive Huffman algorithm which is provided in the next section.

2.4 PERFORMANCE ANALYSIS OF MODIFIED ADAPTIVE HUFFMAN ALGORITHM

The objective of the proposed algorithm is to employ the In-network processing approach to improve the compression ratio in the implementation of Wireless Sensor networks using data compression algorithms. In assessing the goodness of the proposed algorithm in terms of compression ratio, the proposed algorithm is simulated and compared with the results of two well known compression algorithm found in literature.

The Marcelloni algorithm discussed in literature involves grouping of the sample space of the source set of symbols and assignment of fixed probabilities for each of the group. The prefix of the transmitted code thereby is predetermined and stored up as a lookup table and is referenced for each data group. The suffix is transmitted along with the prefix in order to maintain its integrity. This algorithm hence requires prior knowledge of the probabilities of the incoming source sequence.
The Adaptive Huffman algorithm involves dynamic assignment of probabilities of the incoming data through Binary tree construction. The weight of the node in the binary tree determines the probability of the data in the source sequence and is used to determine the code transmitted. Every time a new data is arrived, it is encoded and added to the binary tree. When the data appear at the subsequent times, the code transmitted corresponds to the location of the node in the binary tree. The tree is balanced in order to satisfy the sibling property. The Adaptive Huffman algorithm depends on the order of arrival of the elements of the incoming source sequence with also an increase in the depth of the binary tree.

The Modified Adaptive Huffman algorithm overcomes the drawback of the above two algorithms. The efficiency of the algorithm is computed based on the parameter compression ratio percentage which is given by Equation (2.3)

\[
\text{Compression ratio} = [1 - \{(\text{compressed no. of bits})/\text{original bits}\}] \times 100 \quad (2.3)
\]

An energy model for communication module developed by Alice Wang et al (2002) is used in the simulations. The radio module for energy dissipation is characterized into two types. The first is given \(E_{\text{elec}}(\text{J/b})\), the energy dissipated to run the transmit or receive electronics and the second is given by \(E_{\text{amp}}(\text{J/b/m}^2)\), the energy dissipated by the transmit power amplifier. To transmit a k bit packet through a distance, d, the energy dissipated is given in Equation (2.4).

\[
E_{\text{elec}}(k,d) = E_{\text{elec}} \cdot k + E_{\text{amp}} \cdot k \cdot d^2 \quad (2.4)
\]

For the parameters \(E_{\text{elec}} = 50\text{nJ/b}\), \(E_{\text{amp}} = 100\text{ pJ/b/m}^2\) and the radio module is capable of transmitting upto 1Mb/s at a range of up to 10 meters is used from Alice Wang et al(2002). Using Equation (2.4) the energy required
for transmission of $n$ packets is obtained. It is assumed that all samples have
to be transmitted to the data collector by using the lowest number of packets.
Also using the assumption of Mainwaring et al (2002), each packet can
contain at most 25 bytes of payload based on which the number of packets
required for uncompressed and compressed samples is obtained. Energy
required for transmission is shown in Figure 2.18. It is found that the
uncompressed samples required more data packets while the compressed
versions required lesser number of packets each thus allowing considerable
power saving.

The real time temperature of the squanna cook river, Nashua river
watershed was downloaded from the internet (www.mass.gov) has been used
for case analysis. Two sets of each 200 data were considered. One set
corresponds to Highly Correlated data and the other set corresponds to
medially correlated data. The plot of the temperature data is provided in
Figures 2.12 and in 2.14. Autocorrelation refers to the correlation of a time
series with its own past and future values. Confidence interval is used to
describe the amount of uncertainty associated with a sample statistic. The
autocorrelation graphs corresponding to the two sets of data, with the 95%
confidence limit consideration computed are given. From Figure 2.13, it could
be seen that the autocorrelation reduces gradually and reaches the 95%
confidence interval but the autocorrelation plot in Figure 2.15 shows a high
correlation.
Figure 2.12 Time Series Plot of Temperature (Medially Correlated)

Figure 2.13 Autocorrelation function for Medially Correlated Data
The performance of these data on the three compression algorithms was analyzed in terms of compression in number of bits transmitted, compression ratio system run time and energy required for transmission. Figure 2.16 shows the total number of bits required to be transmitted using the three algorithms. From Figure 2.16, it could be seen that the proposed algorithm is suitable for medially as well as highly correlated data as the binary tree is capable of adapting to changing correlation. Also the node construction in the binary tree helps to reduce the number of levels in the tree which in turn reduces the number of bits required for encoding.
Figure 2.16 Compression Analysis

Figure 2.17 shows the compression ratio analysis. The proposed modified adaptive Huffman compression algorithm provides a compression ratio of 55% for highly correlated data and nearly 40% compression for medially correlated data set.

Figure 2.17 Compression Ratio Analysis
From the Figure 2.16 to Figure 2.18, we establish the following observations

1. The Marcelloni Static Huffman algorithm requires previously determined probabilities of the sensed data.

2. The Adaptive Huffman algorithm works well for highly correlated data though not for medially correlated data due to the increased number of levels in the binary tree.

3. The Modified Adaptive Huffman algorithm is observed to provide an optimized and effective performance for both highly and medially correlated data and hence proposed here to be suitable for real time applications.

2.5 CONCLUSION

Data compression using Marcelloni, Dynamic Huffman and modified Huffman algorithms is implemented and the performance is
compared. Modified Huffman algorithm is found to be efficient than Marcelloni algorithm since this algorithm does not require apriori knowledge of sensor data. Its performance is also better than adaptive Huffman algorithm since the number of levels required for encoding is less. This algorithm is well suited for sensor network applications because of its simple encoding and decoding procedures. However the proposed algorithm does not reduce the number of transmissions. Hence an algorithm for reducing the number of transmissions combined with the proposed compression algorithm is considered in Chapter 3.