Contents

Chapter 1

Introduction

1.1 Motivation

The European Union (EU) has pledged to cut the consumption of primary energy by 20%, by the year 2020. It is estimated that buildings consume 40% of the energy produced[1]. This has resulted in an increase in the demand to reduce the energy consumption of buildings. To reduce the consumption of energy, building automation systems (BAS) are being widely employed. BAS are computer-based systems that help to manage, control and monitor building technical services (HVAC, lighting etc.) and the energy consumption of devices used by the building. BAS deploy a huge amount of sensors, which provide inputs to perform efficient control of various services like HVAC and lighting. BAS brings with it various benefits, at the same time, offers numerous challenges too. For BAS, sensor measurements alone are not sufficient to understand the condition of the facilities, unless combined with additional contextual information. One of the important meta-data required is that of the physical location of the sensor as discussed in [1]. As the size and distribution of the deployed sensors are high, it is highly cumbersome and error prone to manually maintain the meta-data about the sensor placement. Apart from being error prone, the manual configuration has to be repeated every time there is a change in the meta-data. Change in meta-data can be due to various reasons, such as a change in the office setup, replacement and/or relocation of sensors. All these factors result in inaccurate information about the location of the sensors. Without the accurate information of the location of the sensors, interpreting the data collected from the sensor is difficult and also can be misleading. This could result in the decrease in the effectiveness of the deployed BAS systems. Hence there is a need to develop methods to accurately determine the location of the sensors within the building.

[1]according to value published at https://ec.europa.eu/energy/en/topics/energy-efficiency/buildings

1.2 Problem Statement

Several studies have been carried out to infer the sensor location from sensor data. Most of the methods developed so far have identified ways to cluster the sensors that are located within the same room; however the methods do not identify these rooms within the building [2]. Therefore the research question that is being tackled in this work is:

How to Automatically determine the sensor location utilizing the data from the sensors and the information about the grid (coordinates of the vertices constituting the grid)?

1.3 Contribution of the thesis

This work makes the following contribution:

- Using the energy of the sensor data stream as a feature to distinguish neighboring sensors from non neighboring sensors for binary occupancy sensor data.
- Reducing the problem of determination of sensor locations on the grid to a problem of graph matching.

1.4 Outline of thesis

Rest of the thesis is organized as follows: in Chapter 2 we give a brief overview of related work. In chapter 3 we present the method that has been developed. In chapter 4 we describe our testbed setup. Next, in chapter 5 we present the results obtained by applying the method that we have developed on actual sensor data obtained from 2 different testbeds. In the end in chapter 6 we conclude the work done and discuss future work.

Chapter 2

Literature Review

There has been an extensive body of research that aims at obtaining the location information of the sensors in WSN's. All the approach that has been taken till now has been categorized by Wang et al. in [3] to fall into one of the following categories:

- Proximity based localization
- Range based localization
- Angle based localization

In proximity based localization, WSN is represented as a graph $G(V, E)$. Location of the subset of the nodes, $H \subset G$ are known. The goal is to estimate the location of the remaining $V - H$ nodes relative to the position of the H nodes. Range based localization makes use of ranging techniques using RSSI, time of arrival and time difference of arrival of the signals. The distance is computed using the ranging technique. Computing the locations of the nodes using distance information is non trivial. Range based approach may or may not need anchor nodes. With anchor based approach Multi-lateration technique is used to find the location of the non anchor nodes. Without anchor nodes, multi dimensional scaling (MDS) is adopted for localization. Angle based localization uses the information of angle of arrival of the signals to determine the location of the sensors. To determine the angle, antenna array or multiple receivers on the node are required.

Apart from the techniques mentioned above, recently there have been few works which look at the data measured by the sensors to obtain the information about the location of the sensor. As our approach towards determining the sensor location uses sensor data as input, in this chapter we provide a brief overview of the works which take the same approach.

Various approaches have been taken to automate the process of determining the sensors location using the data from the sensors in buildings using different data analytics and signal processing tools. Hong et al.[4] apply empirical mode decomposition to 15 sensors in 5 rooms to cluster sensors which belong to the same room by analyzing the correlation coefficients of the intrinsic mode functions. They characterize the correlation coefficient distribution of sensors that are located in the same room and different rooms. They were able to show that there exists a correlation boundary analogous to the physical boundary which can be discovered empirically. In [2] Akinci et al. propose a feature: energy content in HVAC delivered air, which can be derived from HVAC system sensors which could lead to the identification of the space in which the sensors are located . They combine sensor measurements and building characteristics(floor area) for the identification of the space in which the sensors are present. In [5] Koc et al. propose a method to automatically identify the zone temperature and discharge temperature sensors that are closest to each other by using a statistical method on the collected raw data. They explore whether linear correlation or a statistical dependency measure(distance correlation) are better suited to infer spatial relationship between the sensors. They carry out their analysis on three different testbeds. They also investigate the effects of distance between sensors and measurement periods on the matching results. In the end, the authors conclude that linear correlation coefficient provides better matching results compared to distance correlation. The authors also conclude that as the distance between the sensors increase, the data size needed to infer spatial relationships also increase.

Lu et al.[6] describe a method to generate representative floor plans for a house. Their method clusters sensors to a room and assigns connectivity based on the simultaneous firing of the sensors placed on the door and window jambs. The algorithm gives a small set of possible maps from which the user has to choose the right map. Method requires special placement of the sensors . The authors were able to calculate the floor map of 3 out of the 4 houses they evaluated. Ellis et al.[7] proposed an algorithm to compute the room connectivity using PIR and light sensor data. They compute room connectivity based on the artificial light spill over between rooms; occupancy detection due to movements between rooms; and fusion of the two. They calculate the transition matrix for the light sensor and occupancy sensor. Fuse both the data together to compute the connectivity graph. Here the authors have considered a situation where there is only one PIR and light sensor per room. Müller et al. define sensor topology as a graph with directed and weighted edges. All pairs of consecutive sensors triggers are interpreted as a user walking from the former to the latter sensing region indicated in the sensor graph by an edge from the former to the latter. Every time a consecutive edge triggers are observed the weight of the edge between the sensors is incremented. They define a method to filter out erroneous edges. In [9] Marinakis et al. obtain the sensor network topology using Monte Carlo Expectation Maximization. They assign activity to people present in the space to obtain the graph topology. The algorithm requires, number of people present in the space as the input. They demonstrate the effectiveness of the algorithm using various simulated data.

Chapter 3

Methodology

This chapter explains the method developed to locate the sensors in a sensor grid, using the information about the locations of the vertices of the grid and data obtained from the sensors. Consider a $m \times n$ grid as shown in the figure 3.1. There are $N = (m \times n)!$ ways in which the sensors can be uniquely placed on the grid. Out of these N ways we have to identify the actual locations of the sensors in the grid.

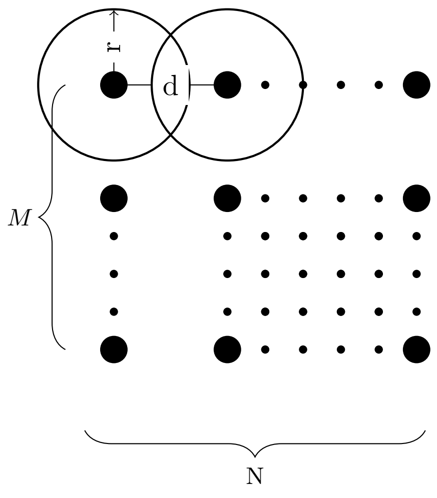


Figure 3.1: A $M \times N$ regular grid having a sensor with field of view r and distance between the neighboring nodes being d.

Before we explain the method we introduce the definitions of the terms that we use in our thesis.

Definition 3.0.1. Neighboring sensors: Two sensors are said to be neighbors if they have overlapping field of view.

Definition 3.0.2. Grid Adjacency Matrix: We represent the sensor grid in the form of an adjacency matrix. Two vertices of the grid i and j are adjacent if the distance between them ($d_{i,j}$) is less than twice the radius (r) of the sensors field of view.

$$GAM_{i,j} = \begin{cases} 1, & \text{if } d_{i,j} < \text{ 2r } \forall\, i \neq j \\ 0, & \text{otherwise} \end{cases}$$

$d_{i,j}$ is the euclidean distance between the vertex i(x_i,y_i) and j(x_j,y_j).

$$d_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

From the definitions 3.0.2 and 3.0.1 we can say that neighboring sensors will always be placed on neighboring vertices. As can be seen in the figure 3.1 there will be overlap in the field of view between two sensors if the distance (d) between them is less than 2r.

3.1 Feature

As neighboring sensors have an overlapping field of view, they observe the same events and thus are highly correlated. On the raw signals of the PIR data stream we use a sliding window with 50% overlap between the consecutive windows and compute the energy feature using the equation 3.1. 50% overlapping window has been chosen as it has been shown to be effective for feature extraction previously[10].

$$E_s = \sum_{n=0}^{k} |x(n)|^2 \tag{3.1}$$

For a PIR sensors whose output is binary, computing energy reduces to counting the number of triggers observed within the window.

With the newly computed energy data stream, we compute the cross correlation between all the sensors using equation 3.2

$$r(x,y) = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \overline{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \overline{Y})^2}} \tag{3.2}$$

X and Y are data streams.
$\overline{X}$ and $\overline{Y}$ is the mean value of X and Y respectively.
n is the number of samples.

3.2 Grid Correlation Sum

Using the cross correlation values between the sensors, we define correlation matrix R between the n sensor nodes as shown in the equation 3.3 .

$$R = \begin{bmatrix} r(1,1) & r(1,2) & \dots & r(1,n) \\ r(2,1) & r(2,2) & \dots & r(2,n) \\ \vdots & \vdots & \ddots & \vdots \\ r(n,1) & r(n,2) & \dots & r(n,n) \end{bmatrix} \tag{3.3}$$

$r(\alpha, \beta)$ represents correlation value between the sensor α and β.

Using *correlation matrix* (R) and *grid adjacency matrix* (GAM) we define a quantity called *Grid Correlation Sum (GCS)* as given in the equation 3.4. GCS represents the sum of correlation value of the sensor pair residing on neighboring vertices of the grid. As the matrices are symmetrical along the diagonals we only consider the elements of the upper triangle of the matrices excluding the diagonal elements.

$$GCS = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} R(\phi(i), \phi(j)) \times GAM(i,j) \tag{3.4}$$

i, j represent i^{th} and j^{th} vertex on the grid.
ϕ is a mapping functions which gives the sensor present at vertex i.
$R(\alpha, \beta)$: correlation coefficient between the sensor α and β.
GAM: Adjacency matrix of the grid as per definition 3.0.2.

We use GCS to identify the correct arrangement out of all the N possible arrangement. If we compute GCS for all the possible arrangements, the arrangement with the maximum GCS will represent the actual arrangement of the sensors on the grid. If two non-neighboring sensors are kept on neighboring vertices or vice versa then the correlation value between those two sensors will be low and thus decreasing the GCS. To illustrate this consider a sensor grid as shown in the figure 3.2. It consists of 3×3 grid. GCS for the the grid is :

$GCS_1 =$ r(1,2)+ r(1,4)+...**r(2,3)**+...**r(5,6)**+**r(6,9)**..+r(8,9)

Now consider the arrangement shown in figure 3.3, where the position of the sensors 3 and 6 are interchanged

$GCS_2 =$ r(1,2)+r(1,4)+...+**r(2,6)**+...+**r(3,5)**+**r(3,9)**...+r(8,9)

GCS_1 will be greater than GCS_2 as r(2,3) $>$ r(2,6) ,r(5,6)>r(3,5) and r(6,9) $>$ r(3,9) as sensors 2 and 6 , 3 and 5, 3 and 9 are non-neighboring

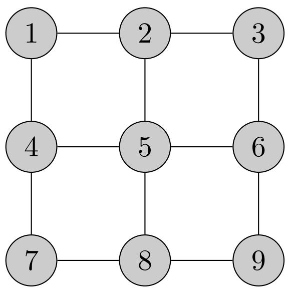


Figure 3.2: Correct arrangement

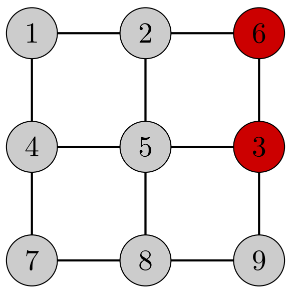


Figure 3.3: Incorrect arrangement

sensor nodes placed on neighboring vertices. Hence we can say that GCS will be maximum for a mapping which maps sensors onto the grid accurately. With this condition, a straight forward way to determine the sensor location will be to carry out a brute force search where GCS is computed for all the N possible mappings. The brute force search can be carried out only when the number of sensors is low, as the number of sensors increases the number of possible mappings also increase drastically, which makes it impossible to carry out a brute force search. Hence there is a need to reduce the search space.

3.3 Maximum spanning Tree

Having established a criterion to determine the right arrangement of sensors on the grid from possible N arrangements. The next step is to reduce the search space. For which we build a graph $G(V, E, W)$ such that $V = \{S_1, S_2, S_3 ... S_n\}$, $E = \{(S_1, S_2), (S_1, S_3) .... (S_{n-1}, S_n)\}$, $W = R$ and compute the Maximum Spanning Tree (MST) for it. The generated MST consists of all the sensor nodes connected to at least one other node by an edge. The MST chooses an edge for every sensor such that the weight of the edge outgoing from the sensor α to sensor β is the maximum among all the edges starting from sensor α, in graph G. As the weights represent correlation values it implies that the maximum spanning tree connects a node to another with which it has the maximum correlation. As the correlation between the neighboring nodes is maximum we can say that MST connects the neighboring sensors.

To compute MST we make use of Prim's algorithm[11]. Prim's algorithm is originally used to compute the minimum spanning tree. To compute the maximum spanning tree the weights of the graph G are negated and with these weights, minimum spanning tree is computed using Prim's algorithm. The minimum spanning tree thus obtained represents the maximum spanning tree for the original graph G.

As the sensors are placed on the grid and there exists an edge between the neighboring vertices of the grid. The MST over R represents one of the spanning trees for the grid. As illustrated in the figure 3.4.

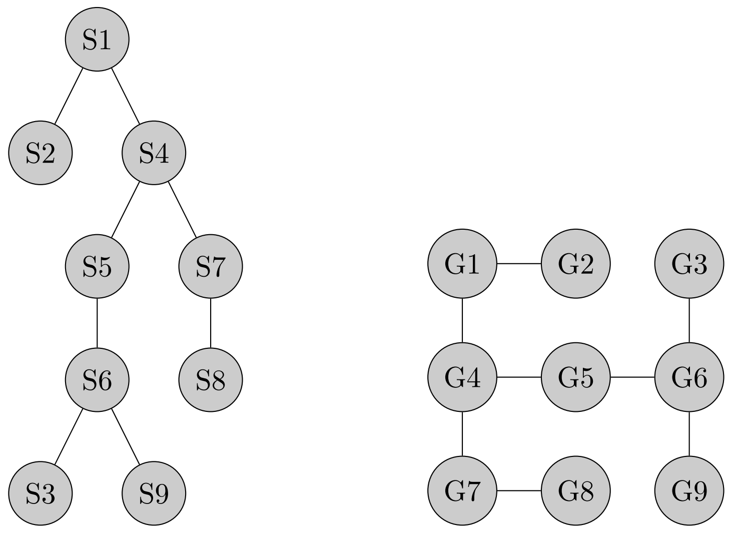


Figure 3.4: A maximum spanning tree incident on the grid

It is possible that for a given grid there may be several spanning trees with the same structure. Figure 3.5 shows few of the spanning tree of the grid similar to the MST obtained for the correlation matrix.

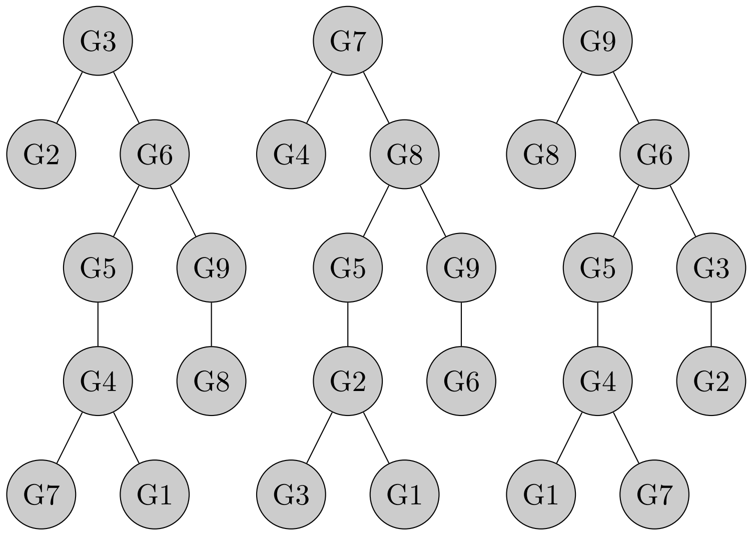


Figure 3.5: Various spanning tree for the grid with similar structure of the MST of the correlation matrix.

If we compute all the possible spanning trees for the grid which maintains the same structure as that of the maximum spanning tree for the correlation matrix and map each vertex on the MST to the corresponding spanning tree of the grid as shown in the figure 3.6, we obtain several possible mappings of the sensors to grid location. Among the possible mappings, we have to choose the correct arrangement. To choose the true arrangement of the sensor on the grid we make use of GCS. The number of arrangements that are obtained from this method is lesser than the N possible arrangements that had to be evaluated for brute force technique.

The main reason for the decrease in the possible mapping is because when we are computing various mappings between the sensors maximum spanning tree and spanning tree for the grid, adjacency has to be maintained. Which implies that when sensor S_1 gets assigned to grid vertex G_1, while assigning sensor S_2 which is a neighbor of S_1 to the grid vertices, care has to be taken that the assigned grid node has to be a neighbor of the vertex G_1.

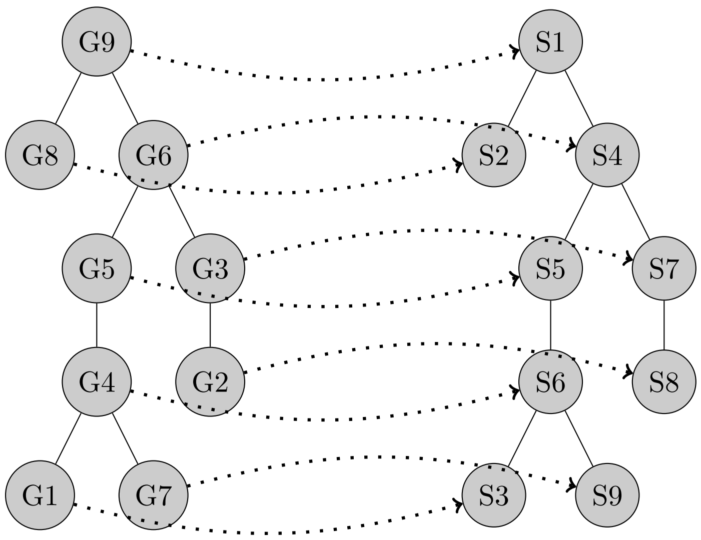


Figure 3.6: Mapping between the grid spanning tree and the MST for the correlation matrix

3.4 Graph Matching

In the previous section, we obtain a maximum spanning tree from R and saw how this represents a spanning tree for the grid. We also observe that there can be multiple spanning trees whose structure is similar to the MST. In this section we describe a method to obtain all the possible spanning trees that have the same structure as the MST obtained from the correlation matrix of the sensors. The task of checking if a pattern graph H is contained in a base graph G and obtaining all the mappings from G to H is a standard problem of graph matching.

A graph matching process between two graphs $G = (V_G, E_G)$ and H = (V_H, E_H) consists of determining a mapping M which associates nodes of the graph G to H and vice versa. Different constraints can be imposed onto M which results in different mapping types: monomorphism, isomorphism, graph-subgraph isomorphism are the most popular ones [12]. Our problem falls under the category of graph monomorphism .

Two graphs $G = (V_G, E_G)$ and H =(V_H, E_H) are *monomorphic* if and only if there exists an injective (node) mapping $\phi(V_G) \rightarrow V_H$: for which $\forall v, w \in V_G : (v, w) \in E_G \Rightarrow (\phi((v), \phi(w)) \in E_H$. [13].

Monomorphism is often confused with subgraph isomorphism. Monomorphism is a weaker kind of subgraph isomorphism.

Two graphs $G = (V_G, E_G)$ and H =(V_H, E_H) are *sub graph isomorphic* if and only if there exists an injective (node) mapping $\phi(V_G) \rightarrow V_H$: for which $\forall$ v,w $\in V_G : (v, w) \in E_G \Leftrightarrow (\phi((v), \phi(w)) \in E_H$. [13].

The relationship between edges of the graphs are equivalence for subgraph isomorphism, and for Monomorphism relationship is an implication.

A problem of finding all the Monomorphisms of the pattern graph into the base graph is defined as Monomorphism problem. Graph Monomorphism is a NP-Complete problem [14].

3.4.1 Related Work

Graph matching is widely used in pattern recognition and image processing applications. Graph matching also finds its application in the field of biomedical and biological applications. Most of the algorithms that are developed for purpose of graph matching uses tree search of some form with backtracking. The key principle is that there will be a partial mapping which will be initially empty and the mappings gets expanded iteratively by adding to it new pairs of matched node. The matched nodes are chosen based on the conditions determined by the matching type.

The first prominent and one of the most widely used algorithm in the area of graph matching was proposed by Ullmann [15]. Ullmann algorithm addresses the problem of graph isomorphism, sub graph isomorphism , graph monomorphism. Ullmann algorithm is based on depth-first search with backtracking. To prune unfruitful matches the author proposes a refinement procedure, which employs conditions based on the knowledge of the adjacent nodes and the degree of the nodes that are being matched. Ghahraman et al. in [16] propose a graph monomorphism algorithm. The authors formulate the graph monomorphism problem as a minimum weight clique problem of weighted nets . The major drawback of the algorithm is that the algorithm uses a $N^2 \times N^2$ matrix to represent netgraph obtained from the cartesian product of the nodes of two graphs under investigation. N representing the number of nodes of the largest graph. This leads to large memory consumption, making the algorithm viable for only small graphs. A

```
Procedure Match(s)
  INPUT: an intermediate state s; the initial state $s_0$ has $M(s_0) = \emptyset$
  OUTPUT: the mappings between two graphs
  Match(s)
  IF M(s) covers all the nodes of $G_2$ THEN
    OUTPUT M(s)
  ELSE
    Compute the set P(s) of the pairs of candidates for inclusion in M(s)
    FOREACH (n,m) $\in$ P(s)
      IF F(s,n,m) THEN
        Compute the state s' obtained by adding (n,m) to M(s)
        CALL Match(s')
      ENDIF
    ENDFOREACH
    Restore data structure
  ENDIF
```

Figure 3.7: Pseudo code for VF2 algorithm[18]

more recent algorithm for graph isomorphism, subgraph isomorphism, and monomorphism was proposed by Cordella et al. in [17], popular as VF algorithm. The authors define a heuristic that is based on the analysis of sets of nodes adjacent to the ones already considered in the partial mapping. The authors further improved the algorithm in 2001 in their paper [18]. The authors proposed a modification to the algorithm and called it VF2. The authors introduce a method to restore data structure, which enabled them to reduce the memory consumption from $O(N^2)$ to $O(N)$ where N is the number of nodes in the graph, thus enabling the algorithm to work with large graphs. In our work, we use the VF2 algorithm to obtain the mappings between the spanning tree and grid graph.

3.4.2 VF2

In this section, we give a brief description of VF2 algorithm.

The VF2 algorithm is a graph matching algorithm to solve graph isomorphism, monomorphism, and subgraph isomorphism problem. VF2 uses depth-first search method to iterate through all the nodes and recursive backtracking technique to check for all the possible mappings. A process of matching a base graph G to a pattern graph H consists of determining a mapping M which associate nodes of the base graph (G) with nodes of the pattern graph (H) and vice versa, with some constraints. Mapping is expressed as a set of pairs of node (n, m) with $n \in G$ and $m \in H$. In VF2 algorithm the process of finding the mapping function is described by a State Space Representation (SSR). Each state s of the matching process can be associated with a partial mapping solution $M(s)$, which contains only a subset of M. A transition from current state (s) to the next state (s') represents the addition of a mapping (n,m) to the state s.

VF2 algorithm introduces a set of rules which helps to prune the number of possible SSR that needs to be checked before obtaining a valid mapping. Figure 3.7 gives a high-level description of the VF2 algorithm. There are 3 important functionalities in the algorithm:

- generation of possible mappings (P(s)) .
- checking of the validity of the mapping (F(s,n,m)).
- Restore data structure.

3.4.3 Computation of candidate pair set $P(s)$

This section explains the method to compute the candidate pair set $P(S)$ for an undirected graph G and H. For every intermediate state s the algorithm computes $P(s)$, a set of possible mapping pairs. For each pair p belonging to $P(s)$ the feasibility of its addition $F(s, n, m)$ is checked : if the check is successful the next state $s' = s \cup p$ and the whole process recursively applies to s'.

To compute $P(S)$ set $T_1(s)$ and $T_2(s)$ are defined for Graph G and H respectively. T_1 and T_2 are the set of nodes in G and H, which are neighbors of the set of nodes included in the partial mapping state $M(s)$ but are not included in $M(s)$. Set $P(s)$ will be made of all the node pairs (n, m) with n being a node in T_1 with the smallest label and m being a node in T_2 . If T_1 and T_2 sets are empty then the set $P(s)$ be calculated by using the sets of nodes not contained in either $G(s)$, $H(s)$.

3.4.4 Feasibility Rules

Feasibility Rules are used to check the consistency of the partial solution s' obtained by adding nodes n,m and prune the search tree. The functionality of the rules is as explained below.

In the algorithm, those candidates (n, m) are pruned if n is not connected to already matched nodes in base graph G (i.e nodes of G included in $M(s)$). Subsequently, the pruning step also removes those node-pairs (n, m) in which m is not connected to the matched nodes in the pattern graph H. These pruning rules assume that the query and data graph are connected.

The algorithm also compares the number of neighboring nodes of each n and m that are connected to nodes in $M(s)$ but are not included in $M(s)$. The number of such nodes in the base graph must be greater than or equal to the number of such nodes in the pattern graph.

Finally, the number of neighboring nodes of each of n and m that are not directly connected to nodes in $M(s)$ are compared. The number of such nodes in the base graph must be greater than or equal to the number of such nodes in the pattern graph.

3.4.5 Restore data structure

In VF2 algorithm all the vectors that are used to store data have the following property: If an element is assigned a value in a state s, it will remain assigned in all states descending from s. This property, together with the depth-first strategy of the search is used to avoid the need to store a different copy of the vectors for each state. When the algorithm backtracks, it restores the previous value of the vectors. This was the improvement introduced in the VF2 algorithm. This is illustrated in the figure 3.8.Consider every block to hold the mappings between the base graph and the pattern graph. As can be seen from the figure as one goes down in the tree, a new variable is created for every level, but with restore data structure the same variable is used at every level. When the algorithm has finished traversing all the levels of the search tree, the algorithm without restore data structure will have N variables, where as one with restore data structure will have only one variable created. Thus reducing the space complexity of VF2 algorithm to $O(N)$ from $O(N^2)$, which was the case in the VF algorithm.

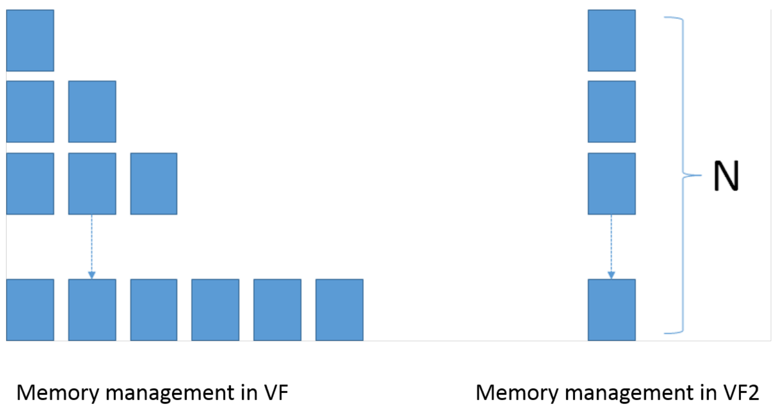


Figure 3.8: Memory cells occupied at each state without and with restore data

3.4.6 Computation Complexity

Computational complexity of the VF2 depends on two factors:

- The time required to calculate the feasibility of every SSR's , computation of the node pair mappings.
- Number of SSR visited.

According to the authors, the former has a time complexity of $O(N)$. For the latter the authors consider a best case; where only one of the potential mappings are feasible which makes the computation complexity $O(N)$ and thus making the overall complexity $O(N^2)$. In the worst case, the algorithm has to explore all the states. This can happen when every node is connected to every other node (base graph is a complete graph) and the graph exhibits strong symmetry. The authors show that the complexity in such a condition is $O(N!)$ and thus making the total complexity $O(N!N)$. In our case, we show the order of complexity for an 8-neighborhood grid, we consider such an arrangement as it represents high connectivity that is possible for a sensor grid. The worst case for such a grid will arise when at every stage of the search tree the algorithm has to explore every possible branch. Since it's an 8-neighborhood grid all the nodes except the ones at the edges of the grid will have 8 neighbors. As one of the neighboring nodes would be the parent node the node at stage i will have 7^i possibilities to explore and stage N there will be 7^n possibilities to explore as illustrated in the figure 3.9. The total number of states to explore will be given by the sum:

$1 + 7 + 7^2 + ... + 7^i + ...7^n$

This represents a geometric progression with a= 1, r =7.

Sum of a geometric progression is given by:

$$S_n = \frac{a(r^n - 1)}{r - 1}$$

$$S_n = \frac{7^{n+1} - 1}{7 - 1}$$

When n is large

$$S_n \approx 7^N$$

Thus computational complexity is $O(7^N N)$

3.5 Determination of sensor placement

After we obtain the various mappings. We need to decide which out of the various mappings gives the actual placement of the sensors on the grid. To

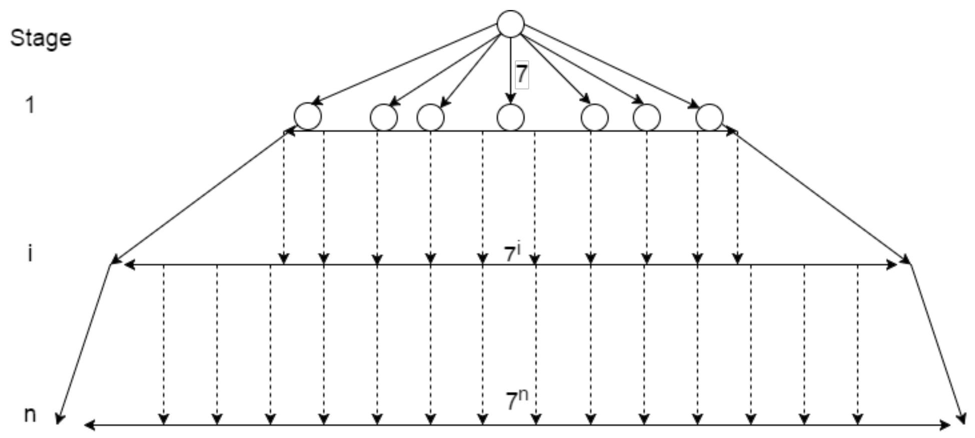


Figure 3.9: All the visited states in case of a 8 neighborhood grid with n nodes.

identify the correct mapping we compute *GCS* as explained in section 3.2. We calculate the grid correlation sum for all the mappings obtained and the mappings which give the maximum sum will be the actual placement of the sensors on the grid.

3.5.1 Limitations

Our method can identify the location of the sensors upto rotational symmetry. *GCS* remains the same for a sensor arrangement which is rotationally symmetrically to the original sensor arrangements as the adjacency between the sensor nodes is maintained, illustrated in the two arrangements that are shown in the figure 3.10 and 3.11. Therefore when we pick the mappings which have the highest *GCS* we get multiple mappings. These mappings include the actual arrangement and arrangements which are rotationally symmetrical to the actual arrangement.

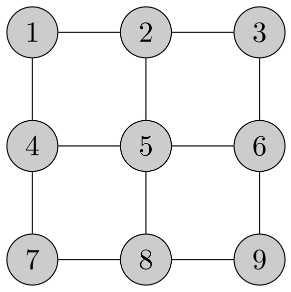


Figure 3.10: Actual arrangement

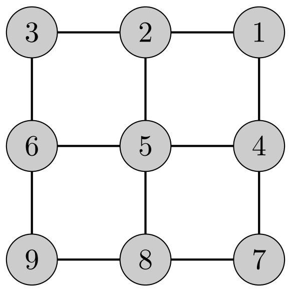


Figure 3.11: Sensors arrangement rotationally symmetric to the actual arrangement

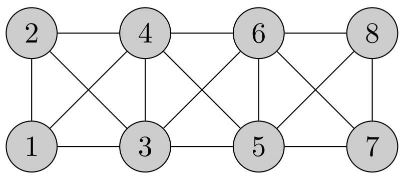


Figure 3.12: Actual arrangement of 2×4 grid.

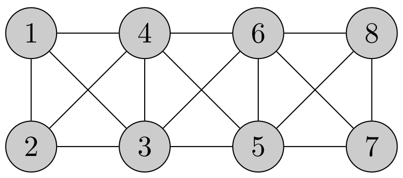


Figure 3.13: Incorrect arrangement of 2×4 grid with the same GCS values.

3.5.2 Special case when the grid is $n \times 2$ or $2 \times n$

Consider a sensor $2 \times n$ sensor grid with diagonally opposite nodes connected as shown in the figure 3.12. These grid structure posses a special case of symmetry. As shown in the figure 3.13 nodes 1 and 2 are interchanged; it is as if the whole structure was rotated with edge (1,2) fixed. The diagonally opposite edges (1,3) in figure 3.12 becomes non diagonal edges in the figure 3.13. To overcome this error we use a *Weighted Grid Adjacency Matrix (WGAM)* instead of a binary grid adjacency matrix. We compute ($WGAM$) first by taking the reciprocal of the distance between the nodes which has an edge between them in the GAM and then normalizing it by multiplying with the least distance (d_{min}) that exists between two nodes in the entire grid. All the members of the $WGAM$ value lies between 0 and 1. Value close to 1 signifying that the 2 vertices are close to each other and value close to 0 signifying that the 2 vertices are far away from each other.

$$WGAM_{i,j} = \begin{cases} \dfrac{d_{min}}{d_{i,j}}, & \text{if } GAM_{i,j} = 1 \\ 0, & \text{otherwise} \end{cases}$$

We use $WGAM$ instead of GAM to compute GCS in equation 3.4. The reasoning behind the use of $WGAM$ matrix instead of the GAM is that we hypothesize that the correlation values for the sensor which are closer are higher than the one compared to sensors which are further apart. By assigning weights to the edges in this manner, more importance is given to the edges between the sensors that are close to the each other. Doing so the sensors which are close by contribute more to the GCS . Now if we consider arrangement as shown in the figure 3.12 and 3.13 though their GCS remain the same, when computed using GAM, GCS won't remain the same when GAM is substituted by $WGAM$. The contribution of the correlation between the sensors placed on the nondiagonal edges is more compared to the contribution made by the diagonal edges to GCS. Therefore if we place the non-diagonally located sensors on the diagonals of the grid, GCS decreases. Thus by using a *Weighted Grid Adjacency Matrix* we are able to obtain the mapping of the sensors onto the grid upto rotational symmetry.

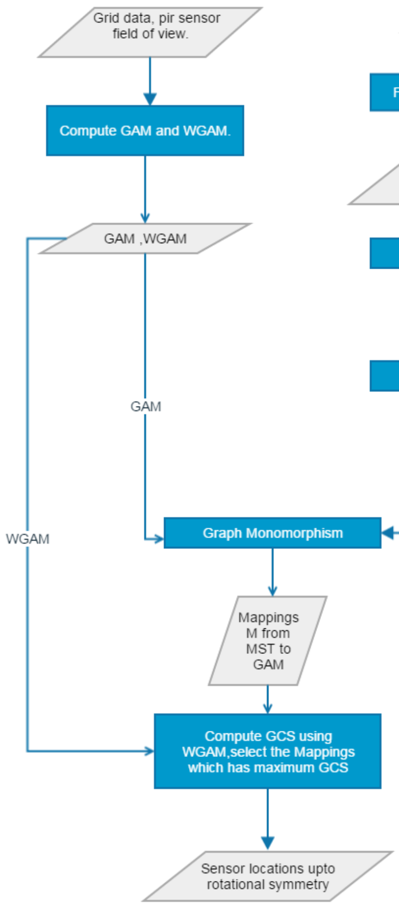

Grid data, pir sensor field of view.
Sensor data
Feature Extraction (energy)
Compute GAM and WGAM.
Feature(energy) data stream
GAM ,WGAM
Compute R
R
Compute MST
GAM
MST
WGAM
Graph Monomorphism
Mappings M from MST to GAM
Compute GCS using WGAM,select the Mappings which has maximum GCS
Sensor locations upto rotational symmetry

Figure 3.14: Flowchart summarizing the developed method.

3.6 Summary

In this chapter, we describe a method to locate the sensor on the grid making use of sensor data and grid vertices information up-to rotational symmetry. Figure 3.14 gives the summary of steps involved in the form of a flowchart.

Chapter 4

Test Bed

We setup our testbed in a $8m \times 6m$ office space. Sensors are mounted on the ceiling which is at a height of 3m. The sensors are placed 2.5m apart in the horizontal direction and 2.2m apart in the vertical direction. We use PIR sensor, EKMB1101112 from Panasonic. The output of the sensor is binary, with 1 indicating occupancy and 0 if the region is unoccupied. We use the sensors with eZ430-RF2500 toolkit which consists of MSP430 micro-controller and CC2500 multi-channel RF transceiver for wireless communication. The testbed uses simpliciTI a TI proprietary low-power RF network protocol. To avoid collision CSMA/CA is employed. We mount the 8 sensors on the ceiling of the room as shown in the figure 4.1 with the sensors highlighted in red. A schematic representation of the lab is as shown in the figure 4.2. All the 8 nodes communicate directly to a centrally located access point which is connected to a laptop, which stores the data. To decrease the packet loss we enable acknowledgment. An acknowledgment (ack) is sent from the AP if the packets are received. If no ack is received by a node, the node retransmits its data a maximum of 6 times until an ack is received from the AP. If no ack is received even after retransmission the packet is dropped and the node continues carrying on its function. From the figure 4.3 and table we can see that the there was a decrease in the packet loss per node after ack was enabled.

PIR sensors are sampled every 100ms. All the nodes have a local clock running on them. With 1 clock tick corresponding to 1ms. All the local clocks are synchronized to one global clock. Since the data is binary we combine 32 samples and transmit the packet every 3.2s. Data packet is an array of 8 bit. The packet structure is as shown in the figure 4.4

The transmitted packet is an array of 1 byte. Consisting of 14 bytes of payload. The payload structure is as shown in the figure 4.4

- Mac Id - unique identifier for each node.
- Packet number- Keeps track of the packets that are being transmitted

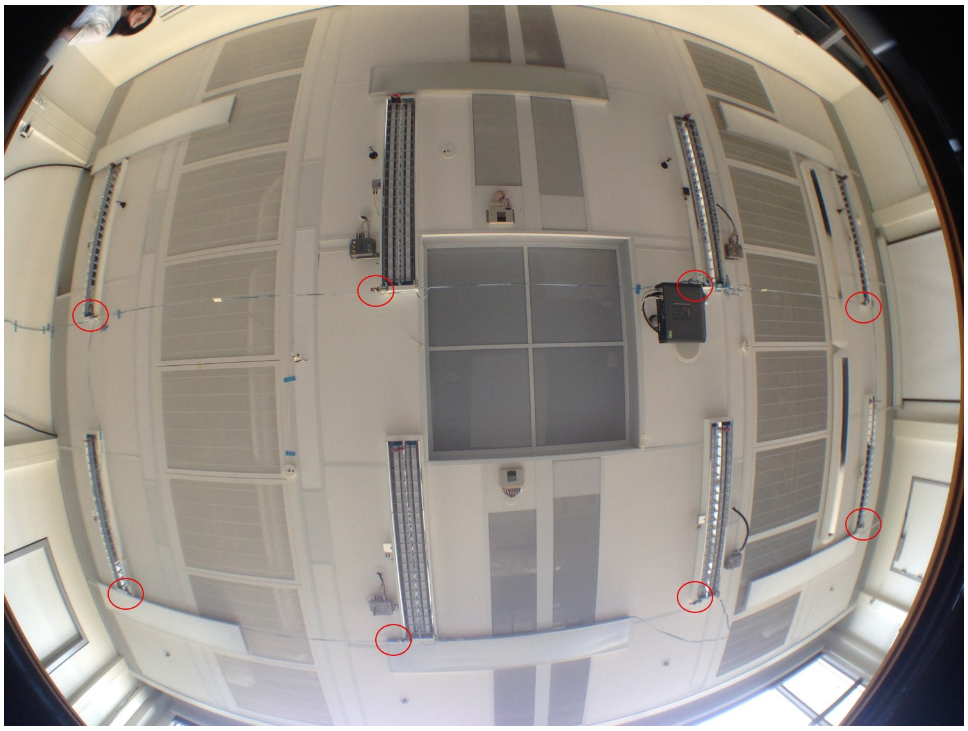


Figure 4.1: Photo of the lab setup with the sensors highlighted in red

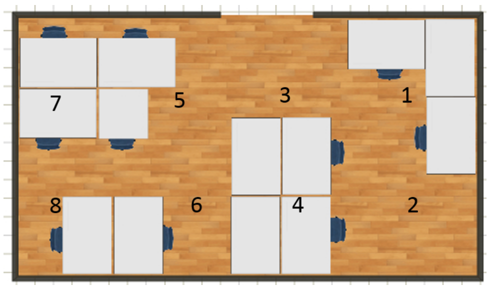
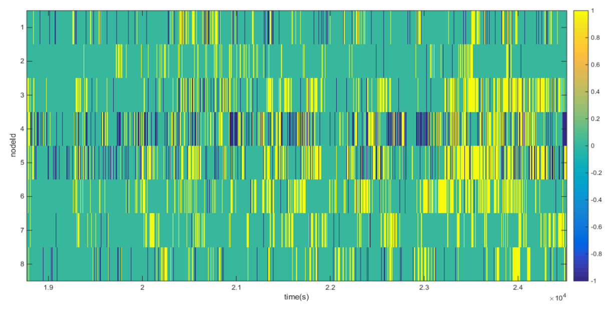


Figure 4.2: Room layout of the testbed.

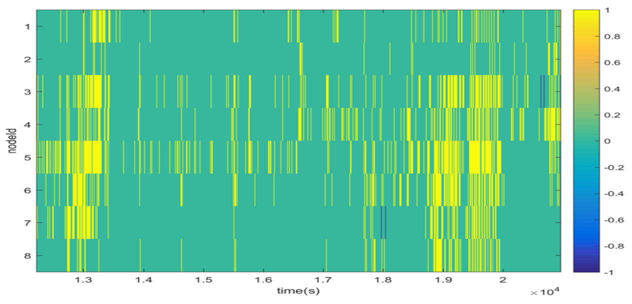
from a node. It helps to identify the lost packets as well as repeated packets.

- Start time - 4 byte array corresponding to the time of first PIR sample of the PIR sensor in the packet.
- End time - 4 byte array corresponding to the time of last sample of the PIR sensor in the packet.
- Data - 32 samples of the PIR sensor.

Data was collected for a span of 2 months. During the course of the



(a) Data before ack was enabled



(b) Data after ack was enabled

Figure 4.3: Data from the test bed. - 1 indicates the lost data , 1 indicates occupancy, 0 indicates unoccupied space

Node	Packet loss Without ack(%)	Packet loss With ack(%)
1	7.66	0
2	1.01	0
3	5.65	0.32
4	23.67	0
5	14.05	0
6	1.48	0
7	0.96	2.20
8	4.54	0

Table 4.1: Packet loss without and with ack for all the sensor nodes

0	1	2	3	4	5	6	7	8	9	10	11	12	13
Mac id	packet number	start time				end time				data			

Figure 4.4: Packet Structure

evaluation, the occupants were notified of data gathering, but were never instructed to behave in any particular way or was there any constrained applied on the activity or the number of people in the room.

Chapter 5

Results

In order to verify the proposed method we test it with data obtained from two different testbeds. First we apply our method to the data collected from the test bed which is described in chapter 4 . Next we make use of the data released by the WSU smart home project[1] [19]. We use C++ implementation of the VF2 algorithm released by the authors available at [20] for finding the mappings between the grid graph and the MST of correlation Matrix (R).

5.1 Error Metrics

To evaluate the results we define an error metrics as follows:

$$error = \frac{\text{Number of misplaced sensors}}{\text{Number of sensors}} \tag{5.1}$$

5.2 Data Length

An important aspect to know in a data driven method is, how much of data is required in order to get accurate results? To answer this question, the entire dataset of length T is divided into blocks of time interval t as shown in the figure 5.1. We iteratively increment t until a value beyond which error observed for all the blocks of data of length t is zero. This is done because there can be parts of the data which might give pseudo high correlation value between two non neighboring nodes or vice versa due to various reasons. As the data length increases, the effect of the data influencing the incorrect correlation values between the sensors diminishes.

[1]http://ailab.wsu.edu/casas/datasets/

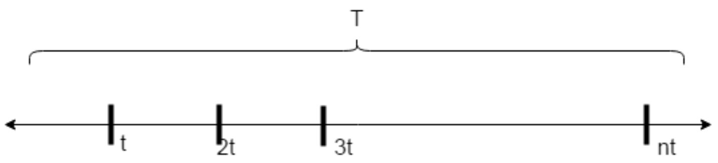


Figure 5.1: Data divided into blocks of period t

5.3 HTC34 testbed

As described earlier, the testbed consists of 8 sensors arranged as a 2×4 grid. The field of view of the sensors used in the testbed has a radius of 2.6m. Using the coordinates of the vertex of the grid and the field of view of the sensors we can determine the neighboring vertices for every vertex of the grid. All the vertices of the gird with field of view marked around them are as shown in the figure 5.2. The co-ordinates of the vertices, calculation of the field of view for the sensor and the computation of GAM is explained in detail in appendix A.

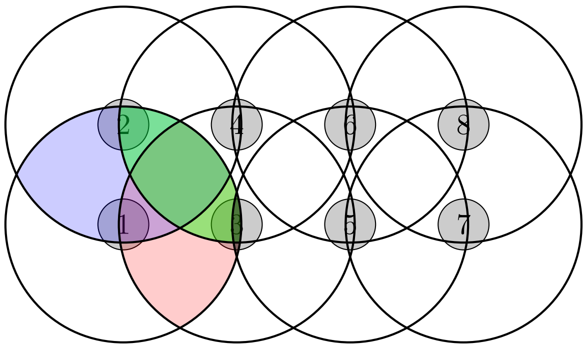


Figure 5.2: Field of view of the sensors on the vertex with the overlapping field of view for sensor 1 highlighted in color

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

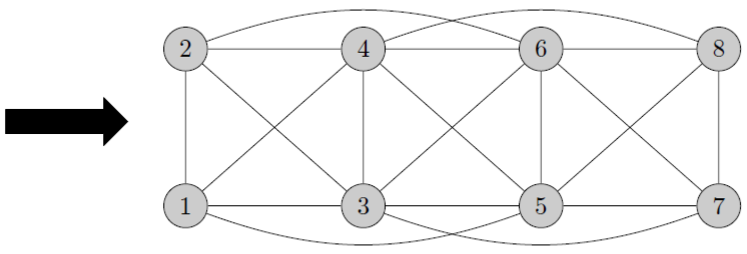
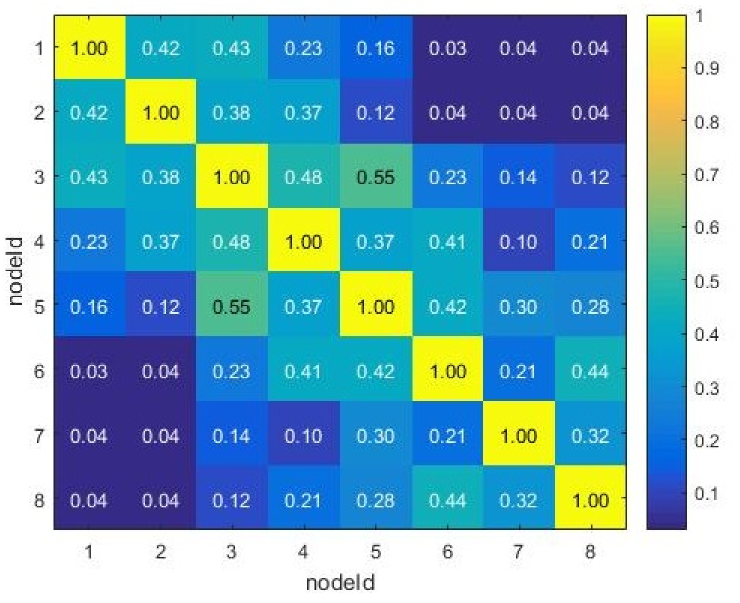


Figure 5.3: Grid adjacency matrix for the grid and its corresponding adjacency graph

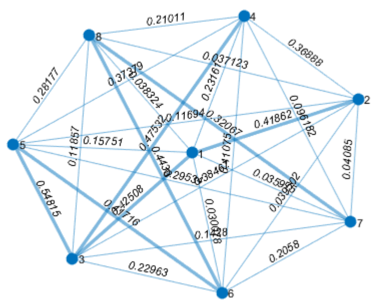
5.3.1 Results

The Grid adjacency matrix for the grid and corresponding grid graph is as shown in figure 5.3. Computed correlation matrix (R), its corresponding graph $G(V, E, W)$ and MST for one of the dataset are as shown in figure 5.4. As can be seen from figure 5.4a the correlation values between the neighboring sensors are higher than the correlation values between non-neighboring sensors. In the MST we can observe that every sensor is connected to at least one of its neighboring nodes.

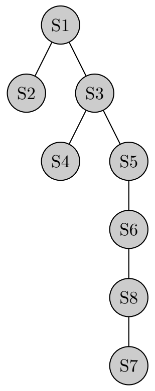


nodeId	1	2	3	4	5	6	7	8
1	1.00	0.42	0.43	0.23	0.16	0.03	0.04	0.04
2	0.42	1.00	0.38	0.37	0.12	0.04	0.04	0.04
3	0.43	0.38	1.00	0.48	0.55	0.23	0.14	0.12
4	0.23	0.37	0.48	1.00	0.37	0.41	0.10	0.21
5	0.16	0.12	0.55	0.37	1.00	0.42	0.30	0.28
6	0.03	0.04	0.23	0.41	0.42	1.00	0.21	0.44
7	0.04	0.04	0.14	0.10	0.30	0.21	1.00	0.32
8	0.04	0.04	0.12	0.21	0.28	0.44	0.32	1.00

(a) R



(b) Graph $G(V, E, W)$ for R, with MST highlighted



(c) Maximum spanning tree of R

Figure 5.4: Correlation Matrix R and it's corresponding graph G and the maximum spanning tree derived from it

Using the method described in section 5.2, we determine that the minimum data length required to compute the sensors location accurately to be 6hrs. Hence we divide our data into blocks of 6hrs length, from which we choose 10 blocks randomly and apply our method to evaluate its performance. 10 different datasets of 6hrs each are used to evaluate the performance of our method. For all the mappings in M we compute the GCS using $WGAM$. The mappings which gave the maximum GCS are as shown in the figure 5.6. These mappings corresponds to the actual arrangements of the sensors on the grid and its rotationally symmetrical arrangements. We also evaluate the datasets using brute force search method and we obtain the same set of mappings as the solution that is obtained from our method. The results for both brute force and our method is presented in table 5.1. As can be seen from table we obtain 0% error for all the datasets. Further table 5.1 also provides details about the number of states visited (including all the intermediate states), number of mappings between the MST and grid, the number of mappings obtained as the solution from the developed method. Figure 5.5 shows the comparison between the search space for the brute force method and our method. The number of states visited and the number of mappings is lesser than number of mappings which has to be checked in case of brute force search.

	Graph Matching method				Brute Force		
dataset	Number of states visited	Mappings	Solution Mapping	Error	Mappings	Solution Mapping	Error
1	11888	2744	4	0	40320	4	0
2	12360	2744	4	0	40320	4	0
3	12144	2664	4	0	40320	4	0
4	11984	2752	4	0	40320	4	0
5	11736	2592	4	0	40320	4	0
6	11888	2744	4	0	40320	4	0
7	11776	2736	4	0	40320	4	0
8	11368	2664	4	0	40320	4	0
9	11752	2656	4	0	40320	4	0
10	11368	2664	4	0	40320	4	0

Table 5.1: Results obtained for HTC34 testbed.

5.4 WSU Tokyo testbed

To further validate the effectiveness of our method, we use the data from WSU smart workplace testbed[21], a publicly available data set. The testbed called Tokyo is located in a lab, where students perform their normal work routine. Data was collected over a period of 4 months. The testbed consists

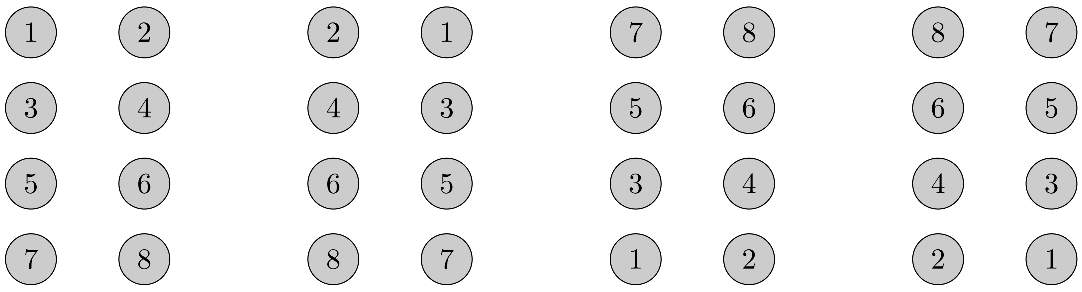


Figure 5.6: Arrangements obtained as solution for the HTC34 testbed

of 43[2] motion sensors, placed as shown in the figure 5.7. The sensors field of view is $1.2m \times 1.2m$. In appendix B we provide a detailed explanation about how we obtain the co-ordinates and the grid adjacency matrix for the testbed. The data from the WSU testbed is represented in the form of a four tuple as shown in table 5.2.

5.4.1 Pre processing

Plot of the dataset for a period of 12hrs is as shown in the figure 5.8. As can be inferred from the figure the dataset mainly consists of parts which are largely inactive. There should be certain amount of activity happening in the space to observe correlation values between the sensors. Also if no activity is observed in the space it will result in all the sensors having the same values(0) and unfairly influencing the correlation values between the sensors, as correlation value is a measure of similarity between two datasets. For these reasons we filter out the parts of the data where we do not observe any trigger for a sensor for a time period of more than 3hrs. The resulting data is sampled at 100ms and is used to verify our method to find the location of the sensors.

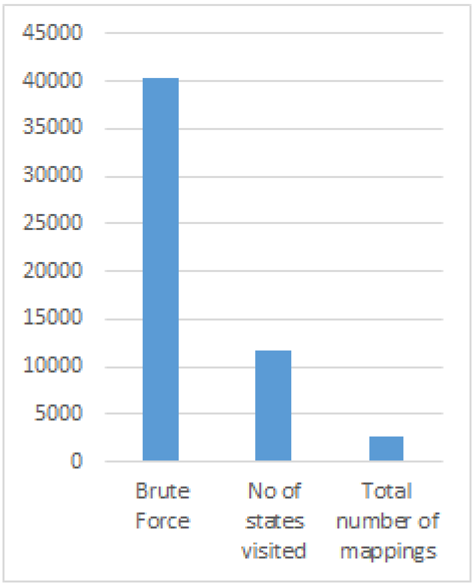


Figure 5.5: Comparing the search space for brute force and our method

[2]Sensor 38 has been ignored as no triggers were found for it throughout the dataset

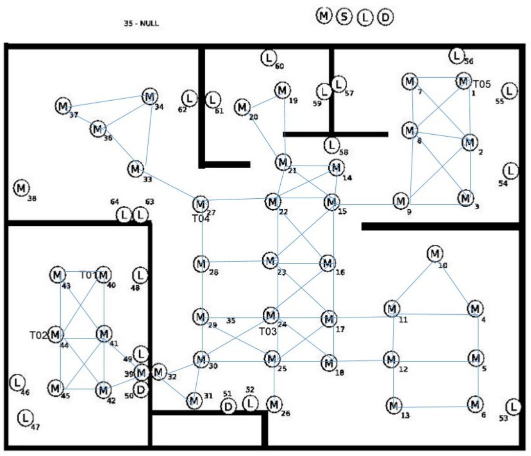


Figure 5.7: Floor plan for WSU CASAS office testbed, named Tokyo with adjacency marked.

Date	Time	sensor	state
2008-01-02	08:02:19.58542	M39	ON
2008-01-02	08:02:19.884461	M41	OFF
2008-01-02	08:02:20.297468	M32	ON
2008-01-02	08:02:20.297468	M32	ON
2008-01-02	08:02:20.689425	M31	ON
2008-01-02	08:02:21.957307	M44	OFF
2008-01-02	08:02:22.517275	M39	OFF

Table 5.2: Data structure of the WSU data.

5.4.2 Sub layout

Before we use the complete layout of the grid, we carry out our analysis on a 4×3 sub-layout which includes sensors 27, 22, 15, 28, 23, 16, 29, 24, 17, 30, 25, 18. This grid was chosen as it is mentioned in [21] that the region covered by these sensors is more active compared to other regions in the testbed. After processing, the data reduces to a span of 207hrs.

5.4.3 Data Length

By using the method described in section 5.2 we determine that 85hrs of processed data is required to obtain the location of the sensors accurately. Figure 5.9 gives an account of error for every block of data of length t, over the entire length of the data, with t varying from 10 to 100 hours. As it can be seen from the figure, for $t > 85$ the error observed for the blocks of data is 0. Hence to carry out our analysis we use data of length 85hrs. Required length of the data in case of WSU data set is much larger

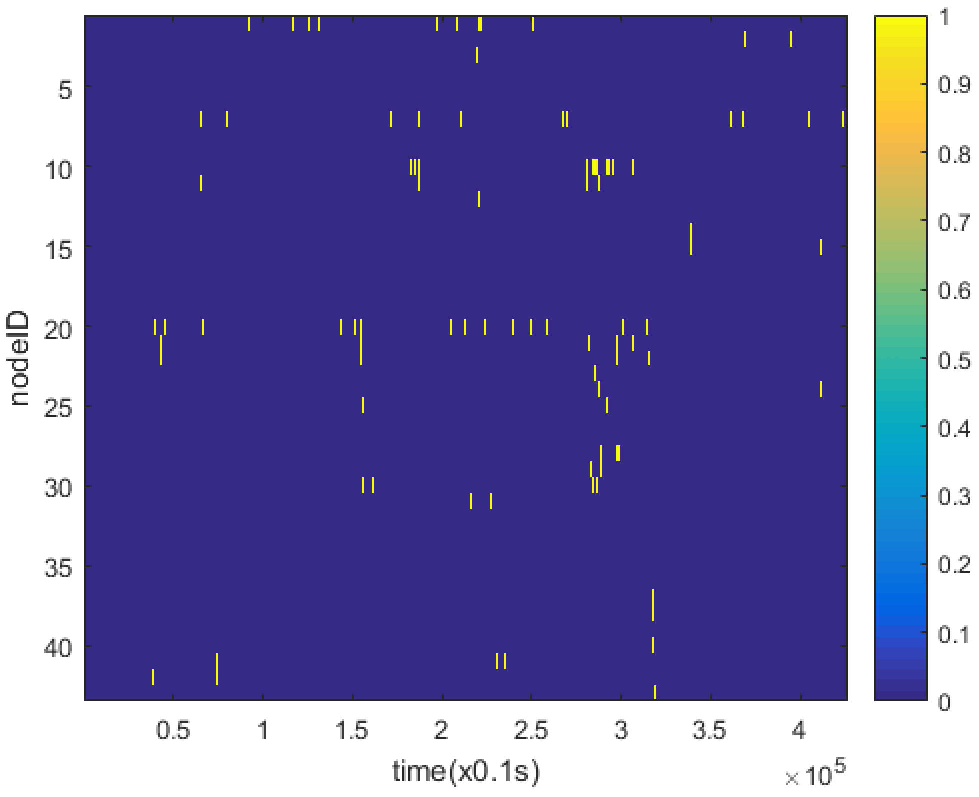


Figure 5.8: Occupancy sensors data with 1 indicating occupancy and 0 indicating non occupancy.

compared to that of HTC34 testbed, the main reason can be because of the overlapping area in HTC34 testbed is much higher than the overlapping area in Tokyo testbed, and the usage patterns are also different for both the testbed. HTC34 testbed is more active compared to Tokyo testbed.

5.4.4 Results

Data is split in to two datasets of 85hrs each. The grid graph is as shown in the figure 5.7. The obtained MST for both the datasets are as shown in figure 5.10. The result obtained from the method is presented in the table 5.3. Both brute force and our method gives the same locations of the sensors as the result, but the number of arrangements that had to be checked in case of brute-force is 479001600 whereas with our method the number of mappings that had to be checked were 2120 and 1811 for dataset 1 and 2 respectively. A brute force search for the 12 sensor arrangement required approximately 27hrs to compute the results. Whereas our method needed on an average 10 seconds to compute the results. As can be seen from the

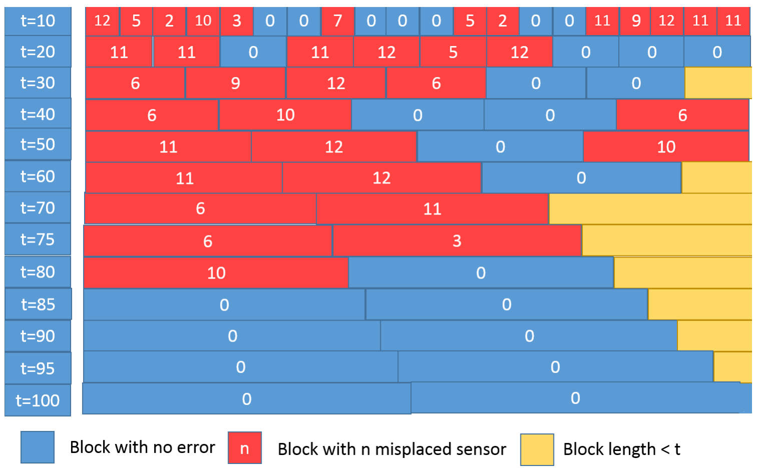


Figure 5.9: Error per block of length t for various values of t across the 200 hours of data for the 4 × 3 layout.

	Graph matching method				Brute force		
Dataset	Number of states visited	Mappings	Solution Mappings	Error	Mappings	Solution Mappings	Error
1	52664	2120	1	0	479001600	1	0
2	48909	1811	1	0	479001600	1	0

Table 5.3: Results obtained for WSU Tokyo testbed for 4 × 3 grid

table 5.1 and 5.3 the number of mappings obtained is larger in case of sensor topology of HTC34 testbed compared to a 4×3 sub-layout topology of Tokyo testbed, even though the number of sensors is more in case of the latter. This is because HTC34 arrangement exhibits rotational symmetry. Due to which for every valid mapping between the MST and grid, the mappings which are rotationally symmetric will also be valid. Thus increasing the number of mappings.

5.4.5 Complete Layout of Tokyo testbed

After validating the 4 × 3 sub layout, we apply our method to the complete layout of Tokyo testbed.

5.4.6 Data length

After applying the processing step considering all the sensors, we are left with only 27hrs of data. The main reason for the drastic fall in the data length is that sensor 1 and 4 are highly inactive and very less triggers are observed as a result majority of the data had to be filtered out. We perform

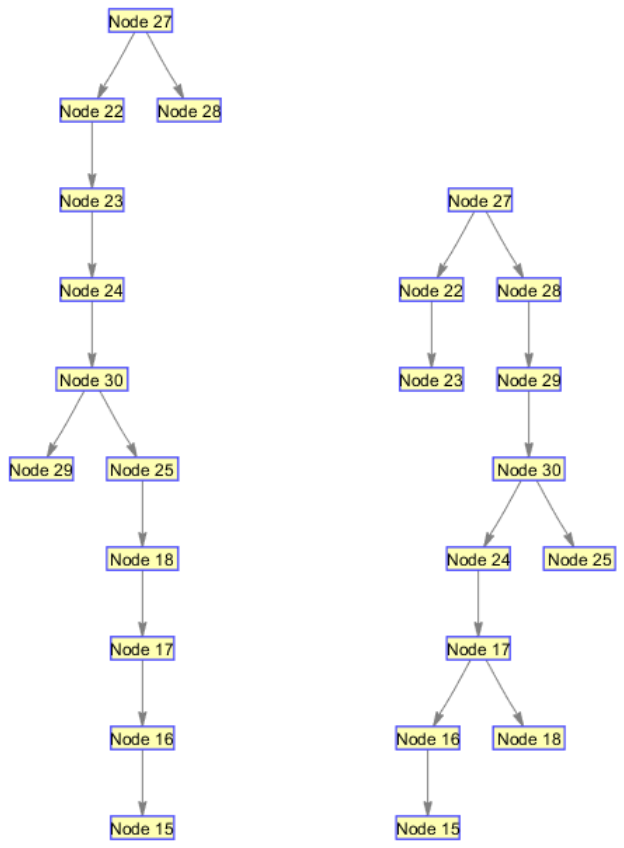


Figure 5.10: MST for the 4×3 layout

our analysis with the processed data.

5.4.7 Results

We use the obtained data to compute the location of the sensors. The MST for the complete layout is as shown in figure 5.11. We obtain two different mappings. Nodes 20 and 19 show rotational symmetry with respect to node 21 so in one arrangement nodes 19 and 20 are mapped to their respective locations in the other mapping nodes 20 and 19 locations are interchanged. As the sensors are equi-distant from node 21, the locations cannot be uniquely identified using a $WGAM$. The results are as shown in the table 5.4. We observe an error of 6.9% as location of three sensors are incorrectly mapped. Brute force search for 43 sensors was not computed as the search space is of the order of 10^{52}, even if the computer is able to compute 10^{12} combinations per second it will take 7e+33 years to finish evaluation. While our proposed method was able to compute the sensor location within 414s $\approx$ 6mins.

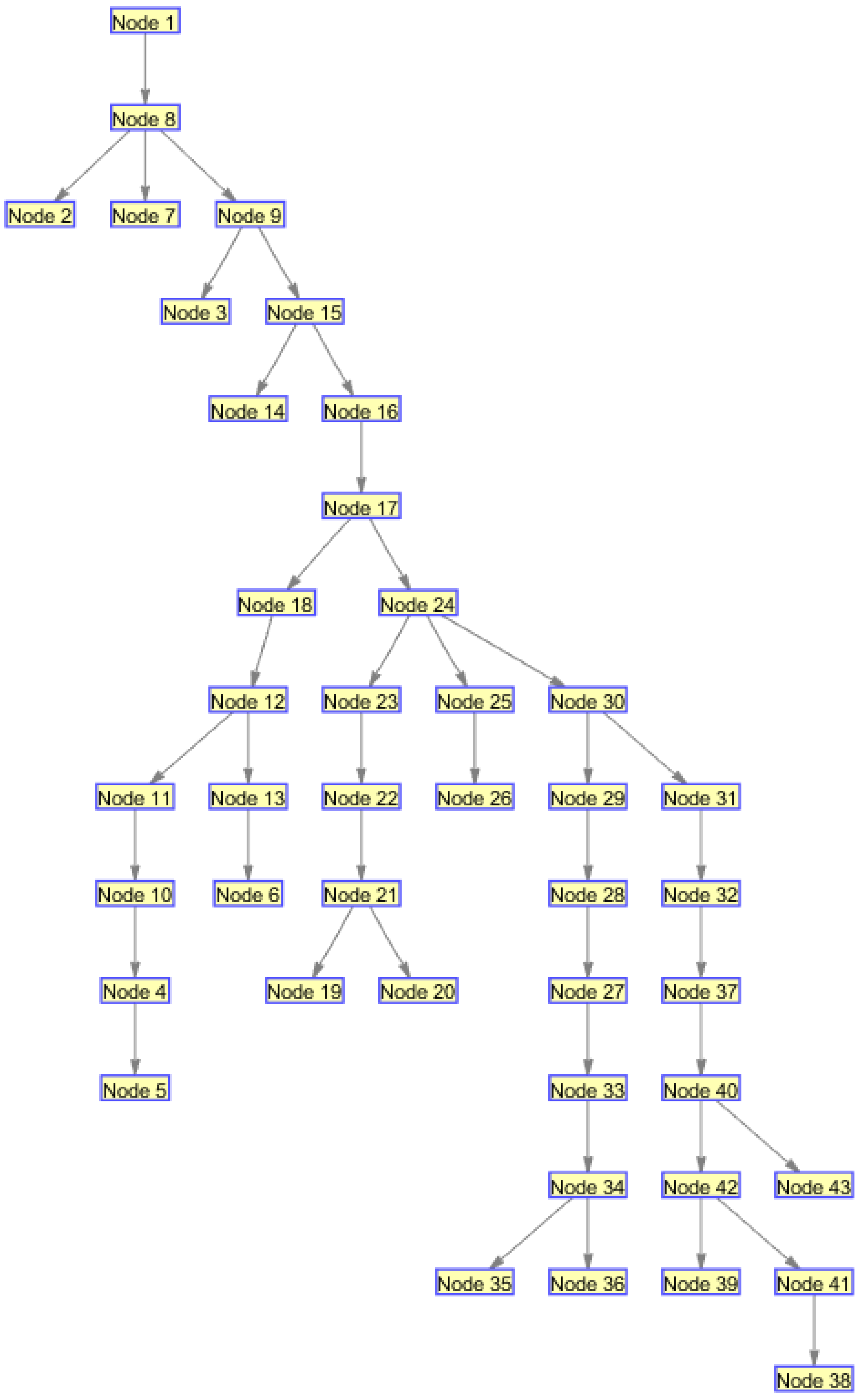
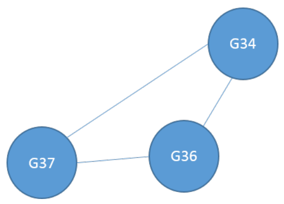


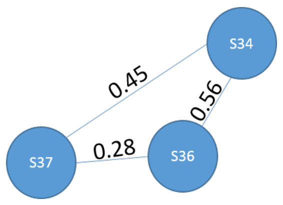
Figure 5.11: MST for the complete layout

Dataset	Number of states visited	Solution Mappings	Error(%)
1	70656	2	6.9

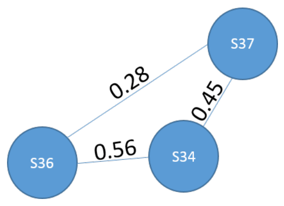
Table 5.4: Results for the complete layout of WSU testbed



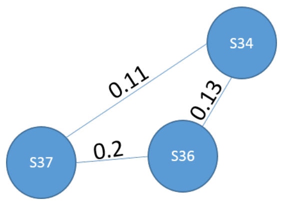
(a) Location of vertices 34,36 and 37 with neighboring vertices marked.



(b) Correlation value between the sensors with actual arrangement



(c) Arrangement computed by our method



(d) Correlation values with data-set of long duration

Figure 5.12: Arrangement of sensors and there correlation coefficient

5.4.8 Understanding the errors

We observe error in the mappings of sensors S34, S36 and S37. Senors S34, S36 and S37 are mapped to the grid locations G36, G37 and G34 respectively. The correlation values between the sensors are as shown in the figure 5.12b and the positions of the sensors obtained from our method is as shown in the figure 5.12c. Since we are computing GCS using $WGAM$ higher correlation values are placed on the edges which are closer to each other. Among sensors S34, S36 and S37, the maximum correlation is observed between the sensors S34 and S36. The edge with the maximum weight (vertices which are close) is between vertices G36 and G37. Therefore sensors S34 and S36 are to be placed on vertices G36 and G37 and the remaining sensor S37 will be placed on G34. The next step would be to identify which sensor should be placed on which vertex out of the two. This is decided by looking at the correlation values with sensor S37 placed on vertex G34, weight of edge (G36, G34) is greater than the weight of edge (G34,G37). Therefore sensors with higher correlation with S37 will be placed on vertex G36, which in this case would be S34. Thus we get the arrangement computed in figure 5.12c.

Testbed	Brute force Mappings	Graph matching mappings	Error(%)
HTC34	40320	2696	0
Tokyo (sublayout)	479001600	1966	0
Tokyo(complete layout)	6.04×10^{52}	70656	6.9

Table 5.5: Search space comparison of brute force and our method

One of the reason for such distribution of correlation can be due to the short duration of the sensor data available. Therefore to verify, we consider only these 3 sensors and apply the preprocessing step for the parent dataset. We get resultant data of around 97hrs. Using this data we compute the correlation values. The values obtained are as shown in the figure 5.12d. For which when the GCS is calculated using $WGAM$, the maximum value of GCS corresponds to the actual arrangement as shown in figure 5.12d. Confirming that the error was caused due to the shortage of data.

5.5 Summary

In this chapter we present the results obtained for the proposed method. Table 5.5 summarizes the results for HTC34 and the Tokyo testbed. As can be seen the proposed method is able to determine the locations of the sensors with no error for HTC34 and sub layout of Tokyo testbed. For the complete layout of the testbed we obtain an error with respect to the location of three sensors, on further investigation we see that the main reason for the error is due to the shortage of the data. From our analysis we can conclude that the accuracy of the results mainly depends on the data length and the level of activity happening in the space under consideration. We also see that the amount of data required for analysis becomes more if the overlapping regions between the sensors is less.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

We present a new data driven method to identify the location of the sensors within a facility. We first identify a feature for the occupancy sensor data and calculate the correlation coefficient for the obtained feature data stream between the sensors. From the correlation values, we build correlation matrix (R) which aids in identifying the neighboring sensor nodes. By computing the MST for R we could identify at least one of the neighbors per sensor node. With the help of the coordinates of the vertices of the grid location, we model the grid as a graph and using the grid graph and the MST we reduce the problem of locating the sensors on the grid to a well-known problem of graph matching. We use the VF2 algorithm, a well known graph matching algorithm to map the sensor nodes to their locations on the grid. We validate the method developed with data from two different testbeds. We are able to identify the sensor locations with 0% error for HTC testbed and 4×3 sub-layout grid for Tokyo testbed. For the entire layout of the Tokyo testbed, we were able to identify the location of all the 43 sensors except 3.

6.2 Future Work

The results are encouraging, going forward we would like to address the following issues in the future:

- Determine the effect of overlapping region on the performance of the algorithm: As could be seen while evaluating the performance of our results with the Tokyo testbed, we could notice that the length of data required to obtain accurate results varied. One of the main factors could be the extent of overlap between the sensors. In the future, we would like to investigate the effect of overlapping region on the performance of our algorithm.

- We are only able to identify the location of the sensors up to rotational symmetry. To overcome rotational symmetry and obtain the exact location of the sensors, extra information about the space is required. For our testbed, we can make use of the location of the door and observe the sensors which are triggered last before a large period of inactivity,this basically represents the last person leaving the room. The sensors which observe the last triggers can be placed on the side of the door and thus eliminating rotational symmetry in one direction. But this method cannot be generalized to all testbed. Therefore we would like to investigate methods which will be effective to eliminate the problem caused due to rotational symmetry.

- Apply our approach to work with other sensors: In our work, we determine the location of the sensors using the correlation matrix obtained from the occupancy sensor data. In the future, we would like to extend our method to other sensors. The modification that would have to be done to our approach would be to identify feature for the sensor data stream for which we can compute correlation values for the sensors. If we are able to obtain a correlation matrix such that the correlation values are high for neighboring sensors and low for non neighboring sensors, we could use our method to obtain the location of the sensors.

- We have developed this method under the assumption that the grid is a connected graph, i.e every sensor has at least one neighboring sensor. In practice, it may not be the case that the grid will always be a connected graph, therefore we would like to expand our method such that it will be able to locate the sensors even when the grid is not a connected graph.

Bibliography

[1] X. Liu and B. Akinci. Requirements and evaluation of standards for integration of sensor data with building information models. *Computing in Civil Engineering*, 2009:10, 2009.

[2] B. Akinci, M. Berges, and A. G. Rivera. *Exploratory Study Towards Streamlining the Identification of Sensor Locations Within a Facility*, chapter 226, pages 1820–1827. doi: 10.1061/9780784413616.226. URL http://ascelibrary.org/doi/abs/10.1061/9780784413616.226.

[3] J. Wang, R. K. Ghosh, and S. K. Das. A survey on sensor localization. *Journal of Control Theory and Applications*, 8(1):2–11, 2010.

[4] D. Hong, J. Ortiz, K. Whitehouse, and D. Culler. Towards automatic spatial verification of sensor placement in buildings. In *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings*, BuildSys'13, pages 13:1–13:8, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2431-1. doi: 10.1145/2528282.2528302. URL http://doi.acm.org/10.1145/2528282.2528302.

[5] M. Koc, B. Akinci, and M. Bergés. Comparison of linear correlation and a statistical dependency measure for inferring spatial relation of temperature sensors in buildings. In *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*, BuildSys '14, pages 152–155, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3144-9. doi: 10.1145/2674061.2674075. URL http://doi.acm.org/10.1145/2674061.2674075.

[6] J. Lu, Y. T. Shams, and K. Whitehouse. Smart blueprints: How simple sensors can collaboratively map out their own locations in the home. *ACM Trans. Sen. Netw.*, 11(1):19:1–19:23, Aug. 2014. ISSN 1550-4859. doi: 10.1145/2629441. URL http://doi.acm.org/10.1145/2629441.

[7] C. Ellis, J. Scott, I. Constandache, and M. Hazas. Creating a room connectivity graph of a building from per-room sensor units. In *Proceedings of the Fourth ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, pages 177–183. ACM, 2012.

[8] S. Müller, A. Helmer, E.-E. Steen, M. Frenken, and A. Hein. Automated clustering of home sensor networks to functional re-gions for the deduction of presence information for medical applica-tions. *Wohnen–Pflege–Teilhabe–Besser leben durch Technik* , 2014.

[9] D. Marinakis, G. Dudek, and D. J. Fleet. Learning sensor network topology through monte carlo expectation maximization. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 4581–4587. IEEE, 2005.

[10] L. Bao and S. S. Intille. Activity recognition from user-annotated acceleration data. In *International Conference on Pervasive Computing*, pages 1–17. Springer, 2004.

[11] R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36(6):1389–1401, 1957. ISSN 1538-7305. doi: 10.1002/j.1538-7305.1957.tb01515.x. URL `http://dx.doi.org/10.1002/j.1538-7305.1957.tb01515.x`.

[12] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. Performance evaluation of the vf graph matching algorithm. In *Image Analysis and Processing, 1999. Proceedings. International Conference on*, pages 1172–1177. IEEE, 1999.

[13] J. Singler. Graph isomorphism implementation in leda 5.1. Technical report, Technical Report. Résumé, 2005.

[14] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979. ISBN 0716710447.

[15] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23 (1):31–42, Jan. 1976. ISSN 0004-5411. doi: 10.1145/321921.321925. URL `http://doi.acm.org/10.1145/321921.321925`.

[16] D. E. Ghahraman, A. K. C. Wong, and T. Au. Graph optimal monomorphism algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 10(4):181–188, April 1980. ISSN 0018-9472. doi: 10.1109/TSMC.1980.4308468.

[17] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. Fast graph matching for detecting cad image components. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 2, pages 1034–1037 vol.2, 2000. doi: 10.1109/ICPR.2000.906251.

[18] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. An improved algorithm for matching large graphs.

[19] D. J. Cook and M. Schmitter-Edgecombe. Assessing the quality of activities in a smart environment. *Methods of information in medicine*, 48(5):480, 2009.

[20] M. L. R. G. of the University of Salerno. VF Library. http://mivia.unisa.it/datasets/graph-database/vflib/, 2008. [Online; accessed 22-July-2016].

[21] D. J. Cook, A. Crandall, G. Singla, and B. Thomas. Detection of social interaction in smart spaces. *Cybernetics and Systems: An International Journal*, 41(2):90–104, 2010.

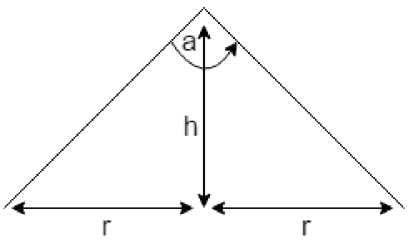
[22] A. Crandall and D. Cook. Tracking systems for multiple smart home residents. *Human Behavior Recognition Technologies: Intelligent Applications for Monitoring and Security*, pages 111–129, 2011.

Appendix A

HTC Testbed

A.1 Location information of the Grid

The coordinates of the vertices are as given in the table A.1. We use EKMB1101112 PIR sensors from Panasonic, with detection angle of 82°. PIR nodes are placed on the ceiling which is at a height of 3m. From this information we can compute the radius of the field of view of the PIR sensors as follows:



$h = 3m, a = 82°$
$r = h \cdot tan(\frac{a}{2})$
$r = 2.6m$

The distances between all the sensor nodes are represented in a matrix

vetrex	x	y
1	0	0
2	0	2.2
3	2.5	0
4	2.5	2.2
5	5	0
6	5	2.2
7	7.5	0
8	7.5	2.2

Table A.1: Coordinates of the vertices of the sensor network

vertices	1	2	3	4	5	6	7	8
1	0.00	2.00	2.50	3.20	5.00	5.39	7.50	7.76
2	2.00	0.00	3.20	2.50	5.39	5.00	7.76	7.50
3	2.50	3.20	0.00	2.00	2.50	3.20	5.00	5.39
4	3.20	2.50	2.00	0.00	3.20	2.50	5.39	5.00
5	5.00	5.39	2.50	3.20	0.00	2.00	2.50	3.20
6	5.39	5.00	3.20	2.50	2.00	0.00	3.20	2.50
7	7.50	7.76	5.00	5.39	2.50	3.20	0.00	2.00
8	7.76	7.50	5.39	5.00	3.20	2.50	2.00	0.00

Table A.2: Euclidean distances between vertices

vertices	1	2	3	4	5	6	7	8
1	0	1	1	1	1	0	0	0
2	1	0	1	1	0	1	0	0
3	1	1	0	1	1	1	1	0
4	1	1	1	0	1	1	0	1
5	1	0	1	1	0	1	1	1
6	0	1	1	1	1	0	1	1
7	0	0	1	0	1	1	0	1
8	0	0	0	1	1	1	1	0

Table A.3: Adjacency matrix

form and is as given in the table A.2 . The radius of the PIR sensor is 2.6m thus any 2 sensors located less than 5.2m(2r) apart are considered neighboring sensors. The adjacecy matrix is obtained as shown in the table A.3

Appendix B

Tokyo Testbed

The coordinates of the vertices in Tokyo testbed are not explicitly specified. The information available are the room layout map as shown in figure 5.7, pir sensor field of view given as $1.2m \times 1.2m$ and the average distance between the sensors as 1.2m in [22]. Assuming that the given layout is to scale and the distance between sensors 27 and 22 as 1.2m, the coordinates of all the sensors are computed. We start by placing the origin(0,0) at vertex 30 and measure the distance of every vertex from the x and y axis which gives the coordinates of all vertices on the grid. The coordinates thus obtained are as shown in the table B.1. No triggers was found for sensor 38 in the dataset, hence sensor 38 is not considered.

Adjacency between two vertices is determined if the square of dimension 1.2×1.2 drawn around the vertices have any overlap. If there is any overlap then the vertices are considered neighbors.

vertices	x	y	vertices	x	y	vertices	x	y	vertices	x	y
1	4.40	5.60	12	3.60	0.00	23	1.20	2.00	34	-0.60	5.00
2	4.40	4.00	13	3.60	-1.20	24	1.20	0.80	36	-1.40	4.40
3	4.40	3.20	14	2.40	3.60	25	1.20	0.00	37	-1.80	4.80
4	4.80	1.20	15	2.40	3.20	26	1.20	-1.20	39	-1.40	-0.20
5	4.80	0.00	16	2.40	2.00	27	0.00	3.20	40	-2.20	1.88
6	4.80	-1.20	17	2.40	0.80	28	0.00	2.00	41	-2.20	0.68
7	3.40	5.60	18	2.40	0.00	29	0.00	0.80	42	-2.20	-0.52
8	3.40	4.40	19	1.20	4.80	30	0.00	0.00	43	-3.40	1.88
9	3.40	3.20	20	0.60	4.64	31	0.00	-1.20	44	-3.40	0.68
10	3.90	2.00	21	1.20	3.60	32	-1.20	-0.40	45	-3.40	-0.52
11	3.60	1.20	22	1.20	3.20	33	-0.60	3.80			

Table B.1: Coordinates obtained for the vertices