ANALYSIS AND OPTIMIZATION OF SENSOR PLACEMENT IN SMART ENVIRONMENTS

HUSNU DURAN

SUPERVISED BY DR. NIROSHINIE FERNANDO

TABLE OF CONTENTS

ABSTRACT

This thesis focuses on the automation and optimization of sensor placement for indoor locations with a deterministic algorithm. The main goal is to propose an algorithm that requires minimal information about sensor and environment specifications and creates optimal sensor placement solutions. In addition, an application is developed that uses the algorithm, lists all possible solutions and visualizes every solution to facilitate user to choose the best solution. The coverage percentage, required number of sensors and sensor efficiency rates are calculated and analysed to understand their relationships and impacts on sensor placement. The results indicate that the proposed solution has many opportunities compared to other methods previously used in the same area such as low cost, polynomial response time, visualization of the sensor placement solution and requiring less information for calculation.

1. INTRODUCTION

Nowadays, smart environments have been attracting the attention of people, which causes a dramatic increase in the adaption and use of smart technologies. Due to the increasing interest in applications such as environmental monitoring, navigation, object recognition and security, sensor networks have become a significant area of study. Sensors are defined as devices that can sense a specific physical phenomenon, process data and communicate information to central unit (Chepuri & Leus, 2015). Geographically distributed sensors provide data to be used in statistical inference problems such as target localization and field (sound, heat, etc.) estimation. Despite the fact that sensors can be used to sense various environmental phenomena such as light, motion and moisture, there are common challenges for sensor systems. According to Becker, Guerra-Filho and Makedon (2009), independently from the type of the sensor, the variables that have highest impact on performance of sensor networks are spatial location of the sensors, topology and the number of sensors. In order to increase the performance of the sensor networks, especially for motion and PIR sensors, these variables should be adjusted by using appropriate optimization tools.

Due to the physical space, energy vs. efficiency reasons and economic constraints, the number of sensors that can be used is generally limited. This restriction in the available number of sensors affects the accuracy of the estimation negatively. Furthermore, different positioning of the sensors leads to generate different values of the accuracy and performance of the sensor networks. From a different perspective on the sensor networks, Cowan and Kovesi (1988) state that the labour required for the design of the sensor networks is a significant factor that causes the system to have a higher cost in real life applications. In practical, the placement configurations for each smart home depends on the house layout and hence different for each home. Manually calculating the optimal placement plan for each house is, therefore labour intensive and also error prone. All these problems created a need for an autonomous system to make sensor placement decisions

that minimizes the number of sensors while optimizing the coverage of the sensors and the performance of the system.

The objective of this thesis is to address the issue of sensor placement by creating a software application that calculates and visualizes optimized sensor placement solutions according to user specified inputs about sensor and room properties. Development of this application involves several steps, which are algorithm development, application development and visualization of the sensor placement solutions. The contributions of this thesis can be identified as;

- Development of a simple and fast algorithm that calculates the possible positions of the sensors, required number of sensors, coverage percentage and sensor efficiency. It is a deterministic algorithm that performs instant calculation through mathematical equations and can be used as a foundation for further development of advanced sensor placement solutions.
- Creation of a software that automatically calculates the coverage rates and creates sensor placement solutions that can be visualized in a form that is simple to understand and use for setting up any sensor network.

The thesis is organized in several sections. Next section contains a literature review that analyses the previous studies in the field of sensor placement, identifies the gaps in literature and explores current methodologies and theories. After the literature review, there is the methodology section, which includes the methodologies that were used for the algorithm and application development. In the results and discussion section, the results of the proposed solution were presented and analysed. After that, future research directions are pointed out. Finally, the thesis is concluded and the source code of the application can be found in the appendices section.

2. LITERATURE REVIEW

The thesis mainly focuses on the sensor placement problem and proposing a solution that requires to design an algorithm that generates automated and optimized sensor placement solutions. Therefore, the literature review focuses on specific topics. First, the background of the sensor placement issue was investigated in order to discover the origin of the problem and cover the most important aspect for a better solution. Then coverage problem was examined since it is the most significant element in placement solutions due to its impact on the performance of the sensor systems. Lastly, previous algorithms used in the area were analysed to build an initial point for the algorithm development and identify the flaws to create an improved solution that can cover them.

2.1 BACKGROUND OF SENSOR PLACEMENT ISSUE

Recently, with the emerging developments in sensor technologies, sensor placement has become a popular area of study. Several works have been applied in various fields that are based on sensor placement methodologies such as transportation, agriculture and whole building health (Thomas, Crandall & Cook, 2016). The main purpose of these systems is monitoring the behaviour and routine activities, which is possible by the use of different sensors.

Key challenge of sensor resource management is determining a sensor field architecture, which is capable of optimizing the cost, providing high coverage and being resilient to sensor failures (Dhillon, Chakrabarty & Iyengar, 2002). Although different issues, such as data fusion, localization and placement, are addressed during the deployment of sensor networks to overcome this challenge, the common goal of the studies conducted in this field is finding optimal orientation and position for the sensors within the specified environment. The optimization of sensor placement is a significant aspect since it creates huge impacts on the

performance of the sensor networks through the coverage (Akbarzadeh et al, 2014). The significance of the sensor placement has led us to create our project in order to fulfil the need in the area of sensor networks, enhance current methods and produce a solution for real life applications.

2.2 COVERAGE PROBLEM IN SENSOR PLACEMENT

Coverage can be evaluated as a representation of the performance because the main purpose of sensor usage is sensing a phenomena and the overall quality of the sensor network are measured by the coverage. Therefore, in our project, sensor placement will be performed by considering the specifications of the sensors such as range and detection angle. In most studies, it is assumed that the sensor is able to sense a circular area with a known radius (Huang & Tseng, 2005). This assumption does not provide accurate results for different kinds of sensors, such as cameras and ultrasonic sensors. Depending on the type of the sensor the sensing area may differ from 140 to 360 degrees, which requires more information than only the radius. Using the specifications of the sensors such as range and sensitivity will increase the accuracy and create more realistic results for using in real life applications.

Computational geometry is also a field that addresses to the sensor placement issue. The coverage of sensors can be estimated by using Delaunay triangulation and Voronoi diagrams (Argany, Mostafavi and Karimipour, 2010). According to Voronoi diagrams, each sensor covers its Voronoi cell and in order to eliminate the coverage holes, the Voronoi vertices of each cell should be covered by the range of the sensor. Similar to their approach, in this project, computational geometry is used to calculate the coverage of the placed sensors to be able to assess the efficiency and performance of the solutions. The method of how computational geometry is applied to the project is explained in the methodology section in detail.

2.3 ALGORITHM SELECTION FOR SENSOR PLACEMENT

Looking at the studies on sensor placement, the optimization is performed by two different approaches, which are heuristic based and exact methods. Some researches considered the sensor placement problem similar to maximum coverage problem (Agarwal, Ezra & Gangjugunte, 2009; Johnson & Bar-Noy, 2011) by formulating the problem and using a greedy algorithm to find the optimal solutions within boundaries. Also, linear programming (Horster & Lienhart, 2006) and binary integer programming (Zhao, Cheung & Nguyen, 2008) have been preferred but these algorithms assume that the target environment is two dimensional and sensors have binary coverage and these assumptions create shortcomings.

Additionally, meta- heuristic algorithms, such as genetic algorithm (Jourdan & de Weck, 2004), evolution strategies (Akbarzadeh et al, 2010), simulated annealing (Chiu & Lin, 2004) and swarm optimization (Wang, Wang & Ma, 2007), have been used to solve sensor placement problem. However, the meta-heuristic algorithms require multiple evaluations of possible solutions, therefore high processors are necessary and the computational cost is higher.

Thomas, Crandall and Cook (2016) conducted a study on sensor placement algorithms including five different algorithms, which are human intuition based, Monte Carlo based, two dimensional uniform placement (Grid), hill climbing and genetic algorithm. The results indicate that the layouts generated for sensor placement are more efficient when genetic algorithm is used rather than hill climbing. Also, it is proved that genetic algorithm is more accurate than other algorithms included in their study. Compared to the genetic algorithm, hill climbing is computationally more expensive and more time consuming. It is observed that hill climbing algorithm takes weeks to retrieve results where genetic algorithm takes only hours. Considering the purpose of our project, both approaches are not suitable because the software program that will be created should provide faster solution for sensor placement problem. The application is planned to be used by the people who wants to deploy sensors to their houses and spending hours for the results is not appealing. Therefore, the response time of the program is a prioritized element that was highly considered during the development of the algorithm.

According to Sabrina and Hafid (2012), the methods used for sensor placement problem can be separated into two; model based and not model based methods. Not model based approaches are generally based on genetic algorithms, simulated annealing or neuronal approaches, which are analysed previously. In the model based approach, the system is represented in a form of graphic, which requires a lot of information about the system in order to have a more reliable model. Sabrina and Hafid (2012) also investigated structural analysis method, which is a type of model based approach in sensor placement problem. In this approach, the model is formed by set of equations and contains information about the variables included in the equations. Since, this method only uses information about the constraints of variables, it facilitates to identify most of the properties of the system without requiring the values of parameters. Similar to their approach, our project uses graph theory to include all of the possible combinations of sensor placement solutions. The use of computational techniques increases the efficiency in order to minimize the complexity and create solutions in polynomial response time.

David, Idasiak and Kratz (2007) developed a system called CAPTHOM to find human presence in an environment. The optimum locations for the sensors are determined by a software program and a simulator. Two different models are used which are for modelling the sensors that will be placed and modelling the environment. The coverage of the sensors and the number of sensors required for maximum performance are the focus points of their application similar to our project.

3. METHODOLOGY

The methodologies used during the development of this research can be separated into three parts. First part is the literature review, which was presented above. Second part is algorithm development. Since the algorithm that decides the placement of the sensors is the key component of this research, previously designed algorithms were examined in the literature review and another algorithm capable of sensor placement with additional benefits was created for sensor placement solution. Last part is the application development.

3.1 ALGORITHM DEVELOPMENT

Since the discovery of the sensor placement problem, there have been many studies that focus on this issue. Many algorithms have been created or used on the sensor placement problem, which were investigated in the literature review. The proposed algorithm is inspired by Cookie Cutter algorithm. Cookie Cutter is mainly used to locate the sensors in a form that all of them touch each other and there is not any overlap.

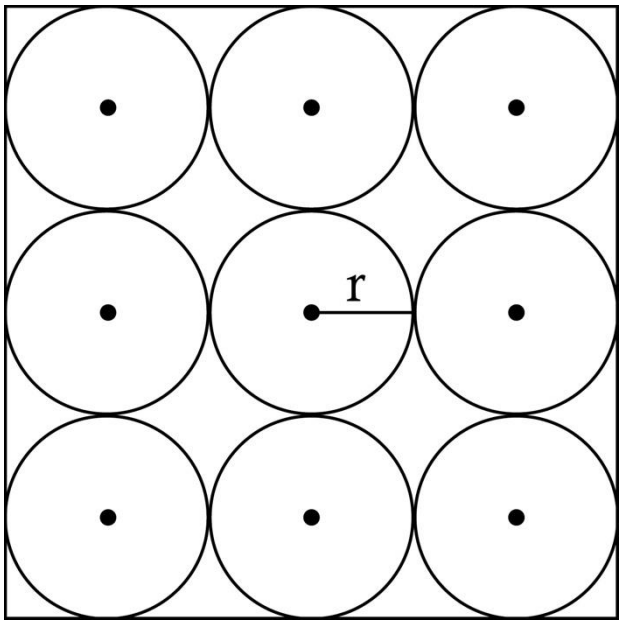

Figure 1. Cookie Cutter

Figure 1 represents the sensor placement solution calculated by using Cookie Cutter algorithm. The sensors are presented by using dots and the circles show their range, which means the area that they are able to sense. As it can be seen from the figure, in this placement there are many gaps between the sensors, which may cause the sensor system to fail in some situations. The presence of the gaps reduces the coverage of the system, which is not a desired property to have less coverage. According to Cooper et al (2003), the coverage ratio (C) is calculated by the formula below;

$$C = \frac{Area\ Effectively\ Covered}{Total\ Area}$$

The starting point of the proposed algorithm is based on Cookie Cutter. In order to reduce the gaps occurred in Cookie Cutter, the distance between the two sensors is reduced gradually. In Cookie Cutter, this distance is always 2*r (r= sensor range). So, in the first sensor placement calculation, the distance is 2r but in each of the following placements the distance is reduced by 0,145*r until it is equal to √2*r. The reason of reducing until √2r is, according to the geometrical calculations, after that point the sensors become too close that almost half of their sensing range overlaps, which is not suitable because it is important to use less number of sensors while increasing the coverage.

Figure 2 shows sample sensor placement that is calculated by the proposed algorithm. As it can be observed, there is no gaps between the sensors. The only small gaps are on the two sides of the room that can be neglected but also the required number of sensors is increased to 16, which was 9 according to Cookie Cutter.

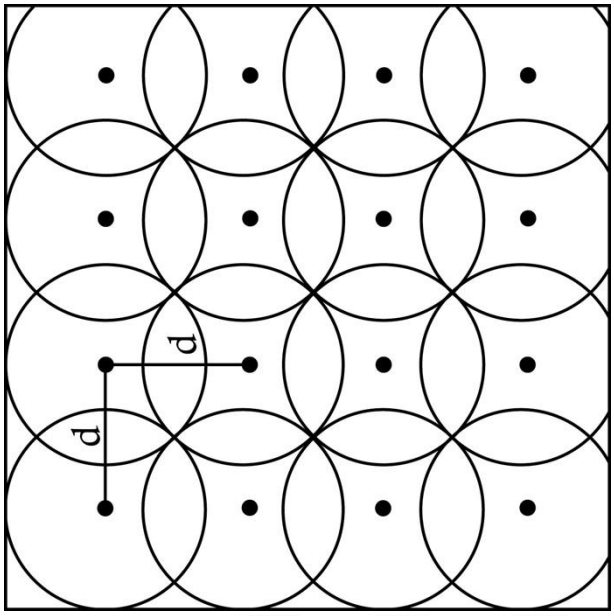

Figure 2. The proposed algorithm

Since coverage and number of sensors are two key factors for the optimization of sensor placement solutions, these values should be calculated for each placement in order to evaluate their performances and compare the placements at the decision making process. Sensors are added to the system one by one through the algorithm according to the distance, therefore the number of sensors are known by the algorithm. However, the calculation of the coverage is performed by using a formula derived from the coverage formula of Cookie Cutter algorithm. Since there are overlaps in the proposed algorithm different from the Cookie Cutter, computational geometry is used for the calculations. All formulas used for the coverage calculation are explained below;

$$Coverage\ Percentage = \frac{Sensor\ Area - Intersect\ Area - Outage\ Area}{Room\ Area} * 100$$

Sensor Area is the area that covered by the sensors and calculated by;

$$Sensor\ Area = \#\ of\ Sensors * \ Area\ Covered\ by\ One\ Sensor$$

$$Area\ Covered\ by\ One\ Sensor = \ \pi * r^2$$

r = sensor range

Intersect Area is the area that two sensors are intersected and calculated by;

$$Intersect\ Area = 2 * [\left(\pi * r^2 * \frac{\alpha}{360} \right) - \left(r^2 * \frac{sin\alpha}{2} \right)]$$

$$sin\alpha = \frac{\sqrt{r^2 - \left(\frac{d}{2}\right)^2}}{r} * \frac{d}{r}$$

d = distance between two sensors

Outage Area is the area that overflows from the area of the room. It occurs when the sensors that are placed close to the sides of the room and some part of them senses outside of the room.

$$Outage = \left(\pi * r^2 * \frac{\theta}{360} \right) - \left(r^2 * \frac{sin\theta}{2} \right)$$

Lastly, the area of the room is calculated by multiplying the length and the width values entered into the room properties section.

Besides the coverage, the algorithm also calculates sensor efficiency. It is the value that shows how efficiently the sensors are used. This variable is to measure if the placement is logical or most of the sensors are used unnecessarily because it is important to use as much of the coverage of one sensor as possible. Higher sensor efficiency means that there are less overlaps and/or less outages.

$$Sensor\ Efficiency = \frac{Coverage}{Expected\ Coverage}$$

Expected Coverage is the maximum value of the area that can be covered by the used number of sensors. It is calculated by the following formula;

$$Expected\ Coverage = \#\ of\ Sensors * \pi * r^2$$

3.2 APPLICATION DEVELOPMENT

Last part of the methodologies used in the research is the development of the application. Java was used as the programming language of the application within NetBeans environment. The purpose of developing the application is transforming the sensor placement solution into a form that is usable by everyone who intend to setup a sensor network. The program asks for user to input the specifications of the sensor and the properties of the room and creates sensor placement solutions for the entered parameters. Below figure belongs to the home screen of the application.

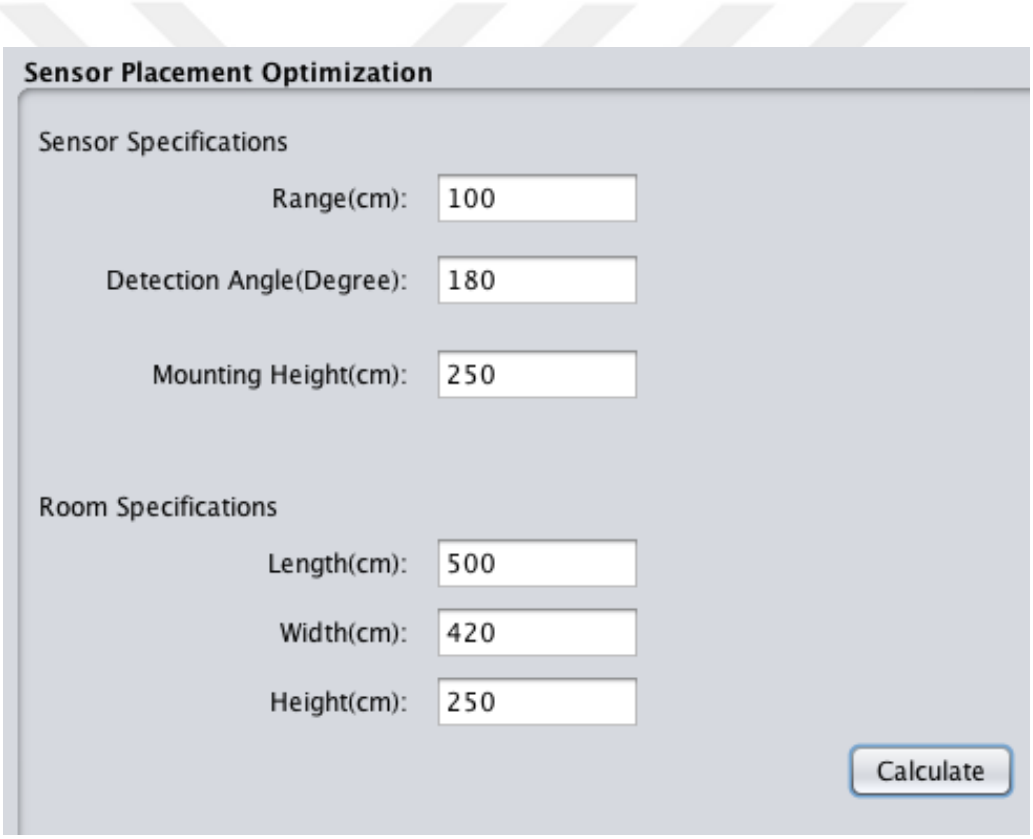

Figure 3. Home Screen of the Sensor Placement Application

When the user enters the values and clicks to the calculate button, a table filled with possible sensor placement options is shown. The table includes several options for user to choose for visualization. For each option, the values of the required number of sensors, coverage percentage and the sensor efficiency is shown in order to improve the decision making process of the user. Below figure is a sample of the mentioned page.

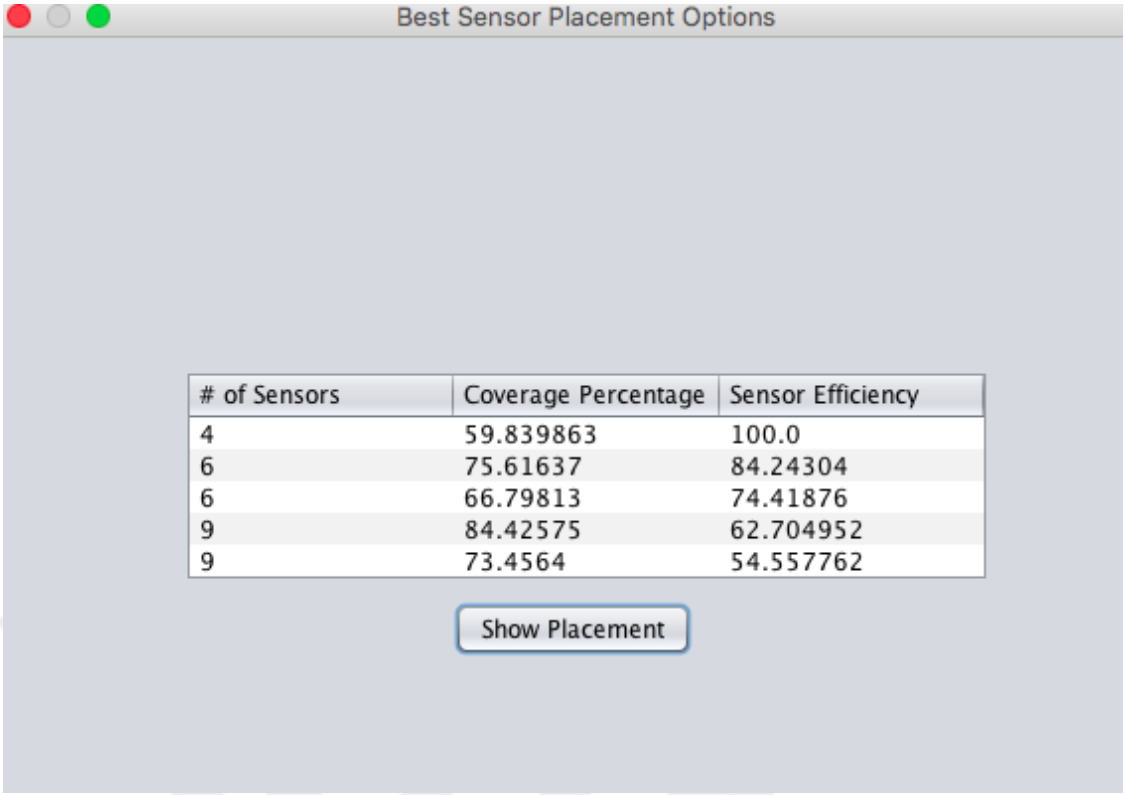

Figure 4. Second page of the application includes the sensor placement options table

According to the table in Figure 4, there are five possible placement options that are calculated by the algorithm. The user selects by clicking any option and clicks to the “Show Placement” button in order to see the visualization of the placement. Figure 5 is an example of sensor placement visualization.

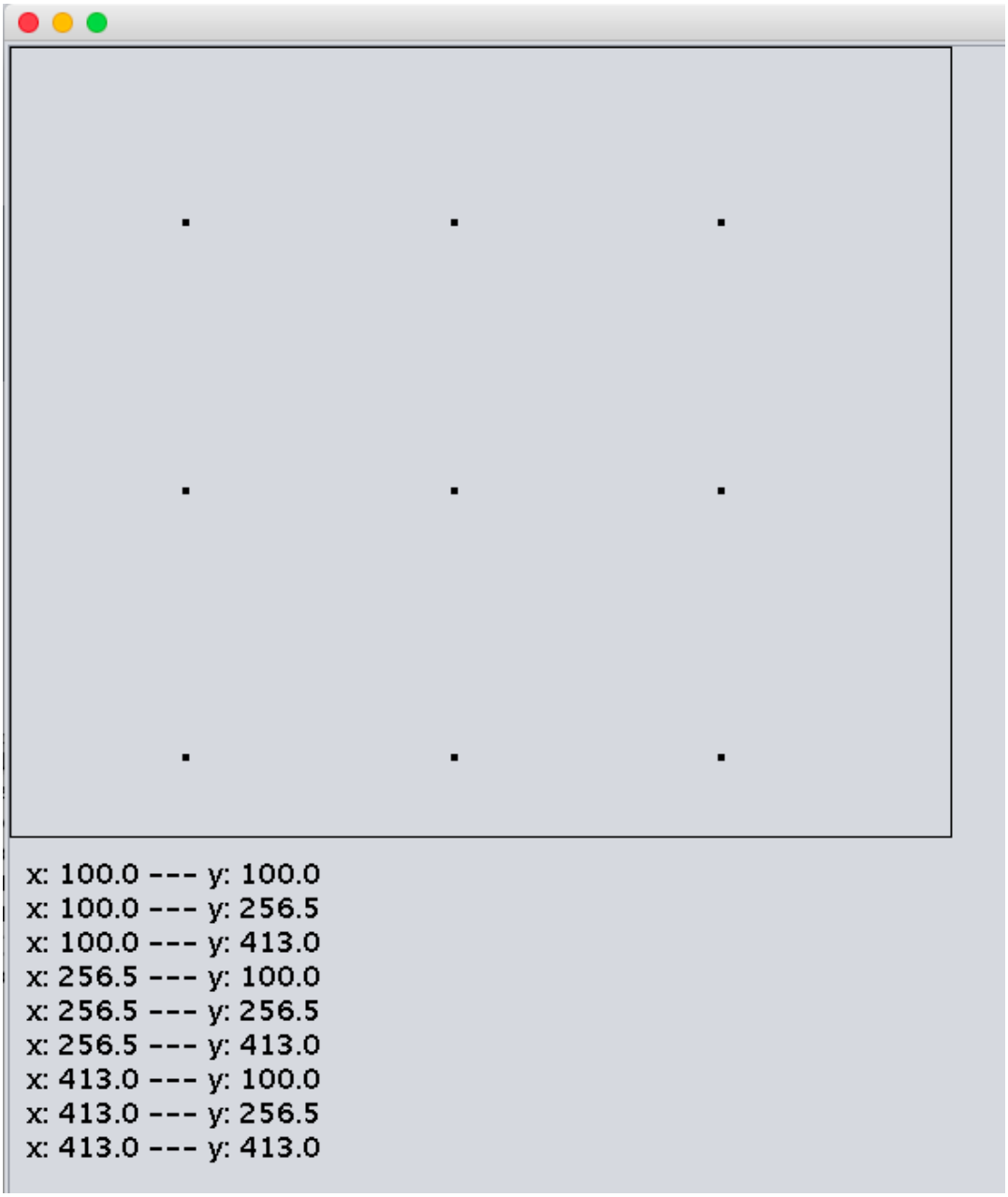

Figure 5. Visualization of sensor placement

The required sensors are represented by dots within the room, which is shown as rectangle. The x and y values are for the pinpoint locations of the sensors. The left side of the rectangle is considered as the y axis and the top side represents the x axis. For instance, first sensor will be located in (100,100), which means it will be 100 cm far from the left side and 100 cm below of the top side of the rectangle.

4. RESULTS & DISCUSSION

In order to evaluate the results of the sensor placement solution, several test runs have been used. In each run, the algorithm adjusts the distance between two sensors and calculates different sensor placement solutions. For each solution, the user inputs remain same but the change in the distance (d) creates differences in the required number of sensors, coverage percentage and sensor efficiency, therefore the placement solutions differ for each d value.

The parameters listed below are the fixed variables for each test;

- Sensor range (r)
- Sensor detection angle
- Sensor mounting height
- Room length
- Room width
- Room height

The variable that creates differences between the sensor placement solutions and is therefore a significant constraint for the accuracy and performance of the solutions is d. The distance is reduced by 0,145 r, which is a coefficient calculated according to the positions of circles (sensor sensing ranges). It is possible to include more sensor placement solutions by decreasing this coefficient, however there will not be huge differences between the produced solutions since it already has a really small value. The possible placement solutions started to be calculated when d = 2r, until it is reduced to $\sqrt{2}r$.

The below figure shows the results of test number 1 for analysis. The parameters entered for this test are;

- r= 100cm, detection angle= 180^{o}, mounting height= 250cm
- room length= 500cm, room width= 420cm, room height= 250cm

Also the distance values for each solution are;

- d1 = 2r = 200
- d2 = 2r – 0,145r = 200 – 14,5 = 185,5
- d3 = 2r – 0,145r*2 = 200 – 29 = 171
- d4 = 2r – 0,145r*3 = 200 – 43,5 = 156,5

- d5 = 2r – 0,145r*4 = 200 – 58 = 142

# of Sensors	Coverage Percentage	Sensor Efficiency
4	59.839863	100.0
6	75.61637	84.24304
6	66.79813	74.41876
9	84.42575	62.704952
9	73.4564	54.557762

Figure 6. Results of test number 1

According to the results of the test number 1, there are five possible sensor placement solutions. One option uses 4 sensors and creates a solution with almost 60% coverage rate and 100% sensor efficiency. This solution is calculated by considering the distance as 2*r, which means this placement is Cookie Cutter placement and the sensors do not overlap. Therefore, there are many gaps in between, which is the reason of 60% coverage. However, since there is not any overlaps, all the sensing areas of the sensors are used effectively inside of the room, which results in having 100% efficiency. Additionally, it can be observed that there are two different solutions with the same number of sensors, which is 6. It is due to the entered parameters and means that 6 sensors can be placed in two different ways. If the coverage percentages and sensor efficiencies are compared, there are major changes due to the difference in the distance values. When the number of sensors is equal, increasing the distance reduces the coverage percentage and sensor efficiency. It can be said that for the equal number of sensors, it is better to use higher distance for better performance. It is due to the fact that less distance causes more overlap and therefore less coverage percentage. Also more overlap means that the sensors are covering the same space unnecessarily and it causes sensor efficiency to be less. Last two solutions have the highest number of sensors, which normally increases the coverage percentage. However, from the fact that their sensor efficiencies are around 55% and 63%, it can be claimed that there are many overlaps or outages which causes the solutions to become unfeasible.

As a results, one may choose option two or four from the possible sensor placement solutions above according to their priorities. The highest coverage is observed in fourth option but since the number of sensors is high, its cost is also higher than the solution

number two. Also, the sensor efficiency of fourth solution is a lot less than solution two. Hence, the decision should be made by the user depending on whether they want higher coverage or lower cost with higher efficiency.

The relationship between the number of sensors and the distance between two sensors are visualized in the below figure. It seems like there is an inverse ratio between them since when the distance is reduced, the number of sensors are increased. However, the decrease in the distance is regular since it is adjusted by the algorithm by the same ratio. Hence, for the same value of the number of sensors there can be differences in the distance, which is caused due to the really small changes in the distance, which may have major effect on coverage.

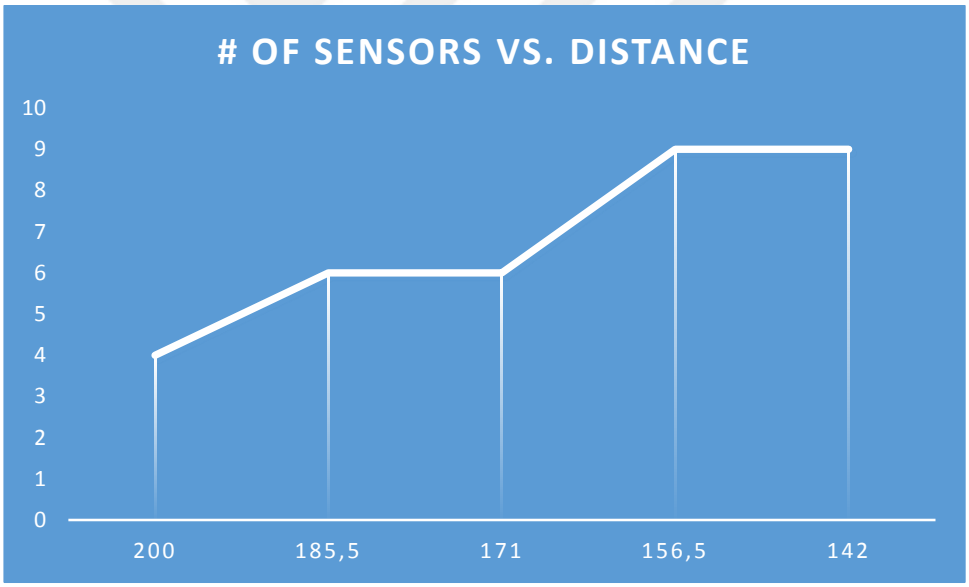

Figure 7. # of sensors and distance relationship

Figure 8 shows the relationship between sensor efficiency and the distance between sensors. It is obvious that they are proportional. The decrease in the distance causes the sensor efficiency to reduce. This is because when the distance is reduced, there are more overlaps between the sensor ranges since their centroids become closer. This situation unsurprisingly causes the sensor efficiency to reduce.

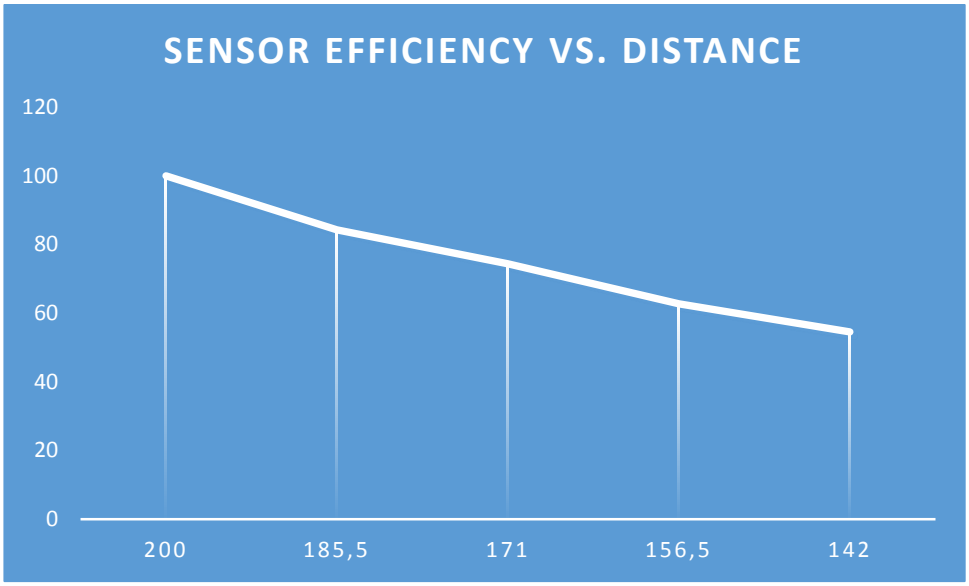

Figure 8. Sensor Efficiency and Distance relationship

Figure 9 below shows the relationship between the number of sensors and coverage percentage. Looking at the graph, one may claim that there is not a specific relationship between them because the number of sensors does not necessarily have the same effect on the coverage or vice versa due to multiple solutions for the same number of sensors. However, looking at the solutions with different number of sensors only, it can be said that increase in the number of sensors increases the coverage. Even if there are many overlaps and outages, adding extra sensor to the placement will naturally increase the coverage due to its range. However, it is not desired to use high amounts of sensors because it increases the cost.

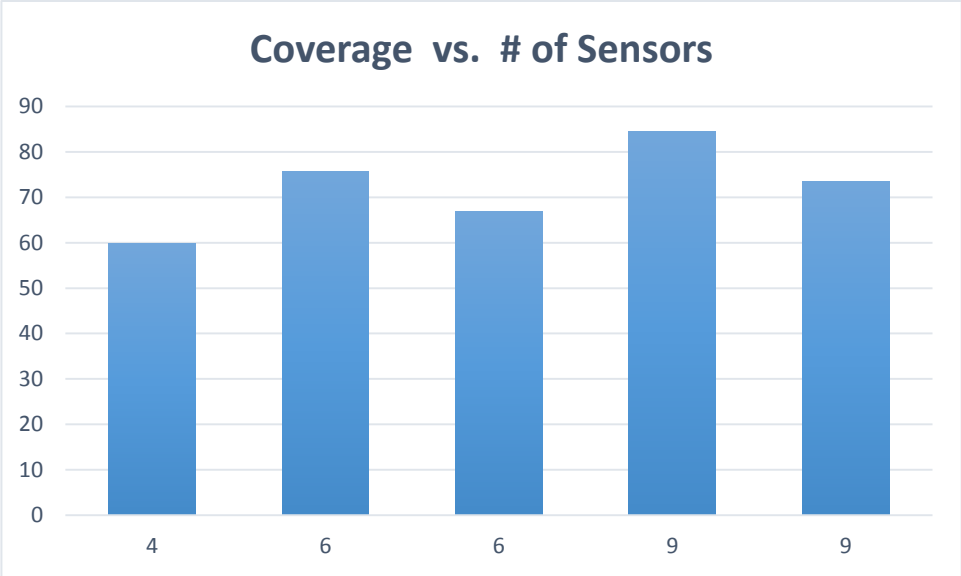

Figure 9. # of Sensors and Coverage relationship

As a result, there are multiple variables that have impact on the establishment of the optimum sensor placement solution. The calculation is therefore complex and require detailed analysis. Selecting the solution with the minimum number of sensors does not necessarily the best action as well as selecting the option with the highest coverage. It is important to evaluate both of them together to obtain better performance. It is also significant to consider the sensor efficiency in order to eliminate the possibility of using sensors redundant, which creates extra cost.

4.1 ADVANTAGES OF THE PROPOSED SENSOR PLACEMENT SOLUTION

There are several advantages of the proposed sensor placement optimization system. These advantages could be listed as;

- The algorithm, therefore the application, uses minimal information about both the environment and sensor specifications in order to create possible optimized sensor placement solutions.
- The application is suitable for any type of user without requiring advanced background knowledge about algorithms or smart technologies. It

- The visually presented sensor placement solutions are in a form that is simple and understandable, which is significant in terms of being user friendly and practical.
- The algorithm minimizes the required number of sensors, which is one of the key objectives of the project in order to optimize sensor placement. Also, the coverage and efficiency values are maximized for higher performance.
- In terms of response time, the algorithm runs a lot faster than other algorithms used in the previous sensor placement studies such as genetic algorithm and hill climbing. It takes usually more than two hours for these kinds of meta heuristic algorithms to create solutions for sensor placement problems where with the proposed algorithm, it is possible to get instant solutions.
- In some cases, the algorithm creates solutions with the same number of sensors but different coverage values. It allows user to see multiple placement solutions with the same amount of sensors, which facilitates to use the sensors in a more efficient way to acquire higher coverage rates.

4.1 LIMITATIONS & ASSUMPTIONS OF THE RESEARCH

In order to minimize the complexity and improve the operability, several assumptions made during the development of the research. These assumptions can be listed as;

- The physical shape of the rooms is assumed to be in rectangular shape. This is a simple assumption to eliminate the complexity of dealing with assorted room structures. For the rooms with different shapes than rectangle, the maximum length and width distances can be entered to retrieve solutions, which will probably be appropriate with maybe small changes.
- The presence of any furniture or columns that can block the sensors or render some of the sensors unnecessary, was not considered in the algorithm. In the case of

having such an obstacle in the room, the solution for the empty room should be examined by thinking the positions of the obstacles and see if the sensors are blocked.

Besides, there are some limitations of the research that should be mentioned;

- The project uses a deterministic algorithm, which probably yields less efficient results than using meta-heuristic algorithms as mentioned in the literature review before. Meta-heuristic algorithms such as genetic algorithm and hill climbing have been used in sensor placement problems in the previous studies and they generate a lot more possible solutions, which increase the probability of having a more efficient solution.
- The sensor placement is based on only one type of sensor with the same range. In some cases, it could be better to use sensors with different ranges in order to eliminate or minimize outages, which also reduced the cost.
- The algorithm does not consider about the possible fixtures in the room, such as doors and windows. These fixtures may have severe impacts on the optimization. For example, covering the door should be prioritized to know when someone enters or exits.

5. FUTURE RECOMMENDATIONS

This research can be counted as a first step toward creating an advanced automated and optimized solution for sensor placement problem and can be enhanced by further development. The basic perspective of this research on the sensor placement optimization will probably inspire future researchers for creating more improved methodologies and algorithms. It is recommended to consider additional variables in future studies in order to create more realistic solutions. For example, the algorithm needs to be modified for rooms that do not have rectangular shapes. Another point is considering the obstacles and fixtures. It is significant to add these components to this research, because even with a high coverage rate, important parts of the rooms can be vulnerable to system failures. For instance, talking about the motion sensors, if the sensors do not cover the door area, the lights will not

automatically turn on when someone enters the room. Another aspect could be the use of different type of sensors in the same placement problem. If sensors with different ranges are combined, there will probably be less outages and overlaps, which will yield better performance. Moreover, sensor range creates differences in the price of the sensors, which may lead to setup cheaper placement solutions.

In terms of sensor placement algorithms, instead of using a deterministic method, machine learning techniques could be used for better solutions. Machine learning methods will allow the system to analyse all the possible solutions by considering multiple attributes and constraints at the same time. Using sample datasets for training and testing will facilitate to select the features that have the highest impacts on the placement and therefore enhance the accuracy of the solutions. Additionally, meta-heuristic algorithms can be used since it is known that they generate more efficient solutions. However, future studies could focus on reducing the response time of the meta-heuristic algorithms because they take at least couple hours and even some of them take days to create a solution, which is not feasible.

6. CONCLUSION

Smart environments have been appearing in various parts of human life such as smart buildings, smart cars and smart offices. Smart environment can be defined as physical worlds that interwoven with displays, actuators and sensors that are connected through continuous networks. From the definition, it is obvious that sensors are significant for smart environments. Therefore, analysing the problems occurred in sensor networks will facilitate in everyday actions of people. As one of the most prioritized problem of sensor networks is optimization of sensor placement, this thesis is developed to create a solution for this problem. In order to address this issue a software program was created, which uses minimal information that can be provided by any type of user without requiring background knowledge in algorithms or smart technologies, about the environment and sensor properties and provide an easily understandable visual solution for sensor placement.

The application revealed important information about the relationships between different variables that have high impacts on sensor placement such as coverage percentage, required number of sensors and sensor efficiency. It is explored that sensor placement is a complex problem that should be further investigated in order to reach the best optimization method. This research is a major step in the area of sensor placement and it is likely that it will facilitate to the future studies.

7. APPENDICES

Source Code of the Sensor Placement Optimization Application

```
package Form;

import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;

/**
 *
 * @author HUSNU
 */
public class Home extends javax.swing.JFrame {

    /**
     * Creates new form Home
     */
    public Home() {
        initComponents();
    }
```

```
    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jPanel1 = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jLabel4 = new javax.swing.JLabel();
        jLabel5 = new javax.swing.JLabel();
        jLabel6 = new javax.swing.JLabel();
        jLabel7 = new javax.swing.JLabel();
        jLabel8 = new javax.swing.JLabel();
        jLabel9 = new javax.swing.JLabel();
        jTextField1 = new javax.swing.JTextField();
        jTextField3 = new javax.swing.JTextField();
        jTextField4 = new javax.swing.JTextField();
        jTextField5 = new javax.swing.JTextField();
        jTextField6 = new javax.swing.JTextField();
        jTextField7 = new javax.swing.JTextField();
        jButton1 = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

        jPanel1.setBorder(javax.swing.BorderFactory.createTitledBorder("Sensor Placement
Optimization"));

        jLabel1.setText("Sensor Specifications");

        jLabel2.setText("Range(cm):");

        jLabel4.setText("Detection Angle(Degree):");

        jLabel5.setText("Mounting Height(cm):");

        jLabel6.setText("Room Specifications");

        jLabel7.setText("Length(cm):");

        jLabel8.setText("Width(cm):");

        jLabel9.setText("Height(cm):");
```

```
        jTextField3.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jTextField3ActionPerformed(evt);
            }
        });

        jTextField6.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jTextField6ActionPerformed(evt);
            }
        });

        jButton1.setText("Calculate");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
                    .addComponent(jLabel1)
                    .addComponent(jLabel6))
                .addGap(0, 0, Short.MAX_VALUE))
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(32, 32, 32)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILIN
G)
                    .addComponent(jLabel9)
                    .addComponent(jLabel8)
                    .addComponent(jLabel7)
                    .addComponent(jLabel5)
                    .addComponent(jLabel4)
                    .addComponent(jLabel2))
                .addGap(12, 12, 12)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILIN
G)
                    .addComponent(jTextField4, javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, 99, Short.MAX_VALUE)
```

```
                .addComponent(jTextField5, javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(jTextField7, javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(jTextField6, javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(jTextField1)
                .addComponent(jTextField3))
            .addGap(207, 207, 207))
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel1Layout.createSequentialGroup()
            .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addComponent(jButton1)
            .addGap(29, 29, 29))
    );
    jPanel1Layout.setVerticalGroup(
       jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
       .addGroup(jPanel1Layout.createSequentialGroup()
          .addContainerGap()
          .addComponent(jLabel1)
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELIN
E)
             .addComponent(jLabel2)
             .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELIN
E)
             .addComponent(jLabel4)
             .addComponent(jTextField3, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
          .addGap(18, 18, 18)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELIN
E)
             .addComponent(jLabel5)
             .addComponent(jTextField4, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
          .addGap(42, 42, 42)
          .addComponent(jLabel6)
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELIN
E)
             .addComponent(jLabel7)
             .addComponent(jTextField5, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
```

```
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELIN
E)
                .addComponent(jLabel8)
                .addComponent(jTextField6, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELIN
E)
                .addComponent(jLabel9)
                .addComponent(jTextField7, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jButton1)
            .addContainerGap(58, Short.MAX_VALUE))
        );

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addGap(0, 12, Short.MAX_VALUE)
                .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGap(23, 23, 23)
                .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addContainerGap())
        );

        pack();
    }// </editor-fold>

    private void jTextField3ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void jTextField6ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
```

```
    }


    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        float range, sense, angle, mouheight, length, width, height;
        range = Float.parseFloat(jTextField1.getText());
        angle = Float.parseFloat(jTextField3.getText());
        mouheight = Float.parseFloat(jTextField4.getText());
        length = Float.parseFloat(jTextField5.getText());
        width = Float.parseFloat(jTextField6.getText());
        height = Float.parseFloat(jTextField7.getText());
        JDialog mydialog = new JDialog();
        mydialog.setSize(600,500);
        mydialog.setTitle("Best Sensor Placement Options");
        //mydialog.setModalityType(Dialog.ModalityType.APPLICATION_MODAL); // prevent
user from doing something else


        //for the modified model

        float[] len2 = new float[100];
        float[] wid2 = new float[100];
        len2[0] = range;
        wid2[0] = range;
        float[] coverage2 = new float[5];
        int[] numofsensors2 = new int[5];
        float[] distance = new float[5];
        float[] efficiency = new float[5];

        for(int x = 0; x<5; x++)
        {

        distance[x] = (float) (range * 2 - 0.145 * range * x);

        int xnumsen2 = 0 ;
        int ynumsen2 = 0 ;


        for(int i= 0; i<10;i++){
            xnumsen2 = i+1;
            if(len2[i] + distance[x]  < length){
                len2[i+1] = len2[i] + distance[x];
                //xnumsen = i+2;
            }
            else
                break;
```

```
    }

    for(int i= 0; i<10;i++){
       ynumsen2 = i+1;
       if(wid2[i] + distance[x] < width){
          wid2[i+1] = wid2[i] + distance[x];
          //ynumsen = i+2;
       }
       else
          break;
    }

     numofsensors2[x] = xnumsen2 * ynumsen2 ;

    for (int i = 0 ; i<6; i++){
       System.out.println(len2[i] + "-----" + wid2[i] +"------"+xnumsen2 + "----" +ynumsen2);
    }




    // if ynumsen & xnumsen >= 1
    float ylast = (ynumsen2 - 1) * distance[x] + range;
    float xlast = (xnumsen2 - 1) * distance[x] + range;
    float ydif= width - ylast;
    float xdif= length - xlast;

    float intsinalfa = 2 * ((float)Math.sqrt((float)Math.pow(range, 2) -
(float)Math.pow(distance[x]/2, 2))/range) * ((distance[x]/2)/range);
    float intalfa = (float)Math.toDegrees((float)Math.asin(intsinalfa));
    float intersect = 0;
    if(distance[x]<2*range){
    intersect = 2 * ((float)Math.PI * (float)Math.pow(range, 2) * intalfa / 360) -
((float)Math.pow((range), 2) * intsinalfa / 2);
    }
//      System.out.println(intersect);
    int numofint= ((xnumsen2-1) * ynumsen2) + (xnumsen2 * (ynumsen2-1));

    float ysinalfa = 2 * ((float)Math.sqrt((float)Math.pow(range, 2) - (float)Math.pow(ydif,
2))/range) * (ydif/range);
    float xsinalfa = 2 * (((float)Math.sqrt((float)Math.pow(range, 2) - (float)Math.pow(xdif,
2)))/range) * (xdif/range);
    float yalfa = (float)Math.toDegrees((float)Math.asin(ysinalfa));
    float xalfa = (float)Math.toDegrees((float)Math.asin(xsinalfa));
    float outx2 = ((float)Math.PI * (float)Math.pow(range, 2) * xalfa / 360) -
((float)Math.pow((range), 2) * xsinalfa / 2);
```

```
      float outy2 = ((float)Math.PI * (float)Math.pow(range, 2) * yalfa / 360) -
((float)Math.pow((range), 2) * ysinalfa / 2);

      if(ylast+range<=width && xlast+range<=length){
      coverage2[x] = ( (numofsensors2[x] * (float)Math.PI * (float)Math.pow(range, 2) ) -
(intersect * numofint) ) / (length * width) *100;
      }else if(ylast+range<=width){
      coverage2[x] = ( (numofsensors2[x] * (float)Math.PI * (float)Math.pow(range, 2) ) -
(ynumsen2 * outx2) - (intersect * numofint)) / (length * width) *100;
      }else if(xlast+range<=length){
      coverage2[x] = ( (numofsensors2[x] * (float)Math.PI * (float)Math.pow(range, 2) ) -
(xnumsen2 * outy2) - (intersect * numofint)) / (length * width) *100;
      }else {
      coverage2[x] = ( (numofsensors2[x] * (float)Math.PI * (float)Math.pow(range, 2) ) -
((xnumsen2) * outy2) - ((ynumsen2) * outx2) - (intersect * (numofint))) / (length * width)
*100;
      }
      efficiency[x] = (coverage2[x]*length*width)/ (numofsensors2[x] * (float)Math.PI *
(float)Math.pow(range, 2));
//       System.out.println(outy2+" "+yalfa+" "+ysinalfa +" " +ydif + " "+ylast);
      }
      int ROWS = 5;
      Object[][] data = { { numofsensors2[0], coverage2[0], efficiency[0]},
                  { numofsensors2[1], coverage2[1], efficiency[1]} ,
                  { numofsensors2[2], coverage2[2], efficiency[2]},
                  { numofsensors2[3], coverage2[3], efficiency[3]} ,
                  { numofsensors2[4], coverage2[4], efficiency[4]} };

      Object[] columnNames = { "# of Sensors", "Coverage Percentage", "Sensor Efficiency " };
      JTable table = null;
      table = new JTable(data, columnNames) {
         @Override
         public Dimension getPreferredScrollableViewportSize() {
            Dimension d = getPreferredSize();
            int n = getRowHeight();
            return new Dimension(400, (n * ROWS));
         }
      };

      final StringBuffer selectedrow = new StringBuffer();

      table.addMouseListener( new MouseAdapter()

{

   @Override
   public void mousePressed(MouseEvent e)
```

```
    {

        JTable source = (JTable)e.getSource();
        int row = source.rowAtPoint( e.getPoint() );
        int column = source.columnAtPoint( e.getPoint() );

        selectedrow.delete(0, selectedrow.length());
        selectedrow.append(row);
        System.out.println(selectedrow);


        if (! source.isRowSelected(row))
           source.changeSelection(row, column, false, false);



    }
});


//    table.setBackground(Color.blue);
    table.setRowSelectionAllowed(true);
    table.setEnabled(false);

    JPanel jPanel = new JPanel();
    jPanel.setLayout(new GridBagLayout());
    JScrollPane sp = new JScrollPane(table);
    jPanel.add(sp);

    JButton button;
    button = new JButton("Show Placement");

    GridBagConstraints c = new GridBagConstraints();
    c.anchor = GridBagConstraints.ABOVE_BASELINE_TRAILING; //bottom of space
    c.insets = new Insets(10,0,0,150);
    c.gridwidth = 20;   //2 columns wide
    c.gridy = 3;       //third row
    jPanel.add(button, c);

//    mydialog.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    mydialog.setContentPane(jPanel);
    mydialog.setVisible(true);


    button.addActionListener( new ActionListener()
{
```

```
 class MyCanvas extends JComponent {

  public void paint(Graphics g) {

//     int selectedRow = table.getSelectedRow();

    g.drawRect (0,0,(int)(length*1.1), (int)(width*1.1));

        float[] distance = new float[5];
        int xnumsen2 = 0 ;
        int ynumsen2 = 0 ;
        int selrow = 0;
        String sel = selectedrow.toString();

        for(int i= 0; i <5; i++){
           if(sel.equals("0") ){
              selrow = 0;
              selectedrow.delete(0, selectedrow.length());
           }
           else if(sel.equals("1") ){
              selrow = 1;
              selectedrow.delete(0, selectedrow.length());
           }
           else if(sel.equals("2") ){
              selrow = 2;
              selectedrow.delete(0, selectedrow.length());
           }
           else if(sel.equals("3") ){
              selrow = 3;
              selectedrow.delete(0, selectedrow.length());
           }
           else{
              selrow = 4;
              selectedrow.delete(0, selectedrow.length());
           }
        }
//       System.out.println(selrow);

        for(int x = 0; x<5; x++)
        {

        distance[x] = (float) (range * 2 - 0.145 * range * x);

        for(int i= 0; i<10;i++){
           xnumsen2 = i+1;
           if(len2[i] + distance[selrow]  < length){
```

```
            len2[i+1] = len2[i] + distance[selrow];
            //xnumsen = i+2;
          }
          else
            break;
       }

       for(int i= 0; i<10;i++){
          ynumsen2 = i+1;
          if(wid2[i] + distance[selrow] < width){
            wid2[i+1] = wid2[i] + distance[selrow];
            //ynumsen = i+2;
          }
          else
            break;
       }


       }

    int counter = (int)width + 50;

    for(int i =0; i<xnumsen2 ; i++){
       for(int j=0; j<ynumsen2;j++){

       g.fillOval((int)Math.ceil(len2[i]),(int)Math.ceil(wid2[j]),5,5);
//       g.fillOval(35,110,10,10);
//       g.fillOval(35,181,10,10);
//       g.fillOval(121,35,10,10);
//       g.fillOval(121,110,10,10);
//       g.fillOval(121,185,10,10);
         g.setFont(new Font("default", Font.BOLD, 16));
         g.drawString("x: " + len2[i]+" --- y: " + wid2[j], 10, counter+20 );
         counter += 20;
//       System.out.println((int)Math.ceil(len[i])+"    " + (int)Math.ceil(wid[j]));
       }
    }

  }
}
    public void actionPerformed(ActionEvent e)
    {
       JFrame window = new JFrame();
//    window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

       window.setBounds(100, 100, 800, 800);
//    window.getContentPane().add(scp);
```

```
        window.getContentPane().add(new JScrollPane(new MyCanvas()));
        window.setVisible(true);

    }

});


    }



    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and
feel.
         * For details see
http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(Home.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(Home.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(Home.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
```

```
java.util.logging.Logger.getLogger(Home.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        }
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Home().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JTextField jTextField3;
    private javax.swing.JTextField jTextField4;
    private javax.swing.JTextField jTextField5;
    private javax.swing.JTextField jTextField6;
    private javax.swing.JTextField jTextField7;
    // End of variables declaration
}
```

REFERENCES

Agarwal, P, Ezra, E & Ganjugunte, S 2009, 'Efficient sensor placement for surveillance problems', *In Proceedings of the 5th IEEE International Conference on Distributed Computing in Sensor Systems,* Marina Del Rey, CA, USA, pp. 301–314.

Akbarzadeh, V, Ko, AHR, Gagné, C & Parizeau, M 2010, 'Topography-aware sensor deployment optimization with CMA-ES', *In International Conference on Parallel Problem Solving from Nature*, pp. 141-150.

Akbarzadeh, V, Lévesque, JC, Gagné, C & Parizeau, M 2014, 'Efficient Sensor Placement Optimization Using Gradient Descent and Probabilistic Coverage', *Sensors*, vol. 14, no. 8, pp. 15525-15552.

Argany, M, Mostafavi, MA & Karimipour, F 2010, 'Voronoi-based approaches for geosensor networks coverage determination and optimisation: A survey', *In Proceedings of the IEEE 2010 International Symposium on Voronoi Diagrams in Science and Engineering (ISVD)*, Quebec, Canada, pp. 115–123.

Becker, E, Guerra-Filho, G & Makedon, F 2009, 'Automatic sensor placement in a 3D volume', *In Proceedings of the 2nd International Conference on Pervasive Technologies Related to Assistive Environments*, p. 36, ACM.

Chepuri, SP & Leus, G 2015, 'Continuous sensor placement', *IEEE Signal Processing Letters*, vol. *22*, no. 5, pp. 544-548.

Chiu, PL & Lin, FY 2004, 'A simulated annealing algorithm to support the sensor placement for target location', *In Electrical and Computer Engineering, 2004. Canadian Conference*, vol. 2, pp. 867-870, IEEE.

Cooper, DC, Frost, JR & Robe, RQ 2003, 'Compatibility of Land SAR Procedures with Search Theory', Potomac Management Group, ALEXANDRIA VA.

Cowan, CK & Kovesi, PD 1988, 'Automatic sensor placement from vision task requirements', *IEEE Transactions on Pattern Analysis and machine intelligence*, vol. 10, no. 3, pp. 407-416.

David, P, Idasiak, V & Kratz, F 2007, 'A sensor placement approach for the monitoring of indoor scenes', *In European Conference on Smart Sensing and Context*, pp. 110-125.

Dhillon, SS, Chakrabarty, K & Iyengar, SS 2002, 'Sensor placement for grid coverage under imprecise detections'. *In Information Fusion, Proceedings of the Fifth International Conference*, vol. 2, pp. 1581-1587, IEEE.

Horster, E & Lienhart, R 2006, 'Approximating optimal visual sensor placement', *In Proceedings of the 2006 IEEE International Conference on Multimedia and Expo*, Toronto, ON, Canada, pp. 1257–1260.

Huang, CF & Tseng, YC 2005, 'The coverage problem in a wireless sensor network', *Mobile Networks and Applications*, vol. 10, no. 4, pp. 519-528.

Johnson, M & Bar-Noy, A 2011, 'Pan and scan: Configuring cameras for coverage', *In Proceedings of the 2011 Proceedings IEEE International Conference on Computer Communications (INFOCOM)*, Shanghai, China, pp. 1071–1079.

Jourdan, DB & de Weck, OL 2004, 'Layout optimization for a wireless sensor network using a multi-objective genetic algorithm', *In Vehicular technology conference,* vol. 5, pp. 2466-2470, IEEE.

Sabrina, A & Hafid, H 2012, 'Sensor Placement Optimization for FDI: Graph Tripartite Approach', *International Journal of Systems Control*, vol. 3, no. 1.

Thomas, BL, Crandall, AS & Cook, DJ 2016, 'A Genetic Algorithm approach to motion sensor placement in smart environments', *Journal of Reliable Intelligent Environments*, vol. *2*, no. 1, pp. 3-16.

Wang, X, Wang, S & Ma, JJ 2007, 'An improved co-evolutionary particle swarm optimization for wireless sensor networks with dynamic deployment', *Sensors*, vol. 7, no. 3, pp. 354-370.

Zhao, J, Cheung, SC & Nguyen, T 2008, 'Optimal camera network configurations for visual tagging', *IEEE Journal of Selected Topics in Signal Processing*, vol. 2, no. 4, pp. 464-479.