ADDING INTELLIGENCE TO A FLOOR BASED ARRAY PERSONNEL DETECTOR

A Thesis
presented to
the Faculty of the Graduate School
at the University of Missouri-Columbia

In Partial Fulfillment
Of the Requirements for the Degree
Master of Science

by
FADI MUHEIDAT

Dr. Harry W. Tyer, Thesis Supervisor
DECEMBER 2015
The undersigned, appointed by the dean of the Graduate School, have examined the thesis entitled

ADDING INTELLIGENCE TO A FLOOR BASED ARRAY PERSONNEL DETECTOR

Presented by Fadi Muheidat,

A candidate for the degree of master of science

And hereby certify that, in their opinion, it is worthy of acceptance.

______________________________________________________________
Dr. Harry Tyrer

______________________________________________________________
Dr. Justin Legarsky

______________________________________________________________
Dr. Mihail Popescu
DEDICATION

I would like to dedicate this thesis to the love of my life, *Tahani*, and my kids: *Kinda* and *Yaman*. Their love, unlimited patience, support and understanding have lightened up my spirit to finish this thesis.
ACKNOWLEDGEMENTS

I would like to express my appreciation to everyone who helped me complete this project in any aspects. Firstly, I would like to express my sincere gratitude and special thanks to Professor Harry Tyrer for the continuous support, guidance, motivation, and for his patience. His guidance helped me in every aspect of my schoolwork, research and teaching. His immense knowledge and expertise enriched my work and supported me in writing my thesis.

Besides my advisor, I would like to thank the rest of my thesis committee: Dr. Mihail Popescu, and Dr. Justin Legarsky for their acceptance to serve in my thesis committee and for the insightful comments, and suggestions to explore other perspectives in my research. Also, I would like to thank my entire lab colleagues, who helped me conduct and prepare for the experiments, and shared ideas and stimulating discussions. Special thanks to my friend Kittisak Sajjapongse for his great intellectual discussions and motivations.

Last but not the least; I would like to give my heartfelt appreciation to my wife, Tahani and kids, Kinda and Yaman and my parents, father Ali, and mother Laila, and my in-laws, Suleiman, and Samira and my brothers and sisters for their continual support, encouragement and love.
# TABLE OF CONTENTS

ACKNOWLEDGEMENTS .................................................................................. ii

LIST OF FIGURES .................................................................................................. v

LIST OF TABLES ....................................................................................................... vii

ABSTRACT ........................................................................................................ viii

1  Chapter 1: Introduction ...................................................................................... 1

2  Chapter 2: Background .................................................................................... 3
   2.1 Rationale ........................................................................................................ 3
   2.2 System overview .......................................................................................... 8
   2.3 WEKA classifiers .......................................................................................... 10
       2.3.1 Naïve Bayes .................................................................................... 10
       2.3.2 Logistic function ............................................................................. 10
       2.3.3 Multilayer Perceptron (MLP) ......................................................... 11
       2.3.4 J48 Decision Trees ......................................................................... 11
   2.4 Performance of fall detection ..................................................................... 12

3  Chapter 3: Methods .......................................................................................... 14
   3.1 The Manufacturer provided board for fast computing .......................... 16
   3.2 Data acquisition and processing ............................................................... 16
       3.2.1 Data storage and display .............................................................. 18
       3.2.2 Notifications ................................................................................ 19
   3.3 Fall detection ............................................................................................... 20
       3.3.1 Data collection experiments .......................................................... 20
       3.3.2 Data Analysis and algorithms ....................................................... 22
       3.3.3 Preparing the data ......................................................................... 24
       3.3.4 WEKA classifiers ....................................................................... 27

4  Chapter 4: Results ............................................................................................ 29
   4.1 Database and webserver capabilities of the new processor .................. 30
       4.1.1 Notification .................................................................................. 32
   4.2 Tripping Falling Backward ....................................................................... 33
   4.3 Fall Detection Algorithms ....................................................................... 39
   4.4 Computational Intelligence ..................................................................... 41
       4.4.1 Training set, Supplied Test set option ........................................ 42
       4.4.2 Cross-validation 10 Folds option ............................................... 45
       4.4.3 Percentage Split (66%) option ..................................................... 47

5  Chapter 5: Discussion ....................................................................................... 50
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Jetson TK1</td>
<td>50</td>
</tr>
<tr>
<td>5.2</td>
<td>Data Collection and fall detection</td>
<td>51</td>
</tr>
<tr>
<td>5.3</td>
<td>Computational Intelligence and Weka framework</td>
<td>52</td>
</tr>
<tr>
<td>6</td>
<td>Chapter 6: Conclusion</td>
<td>55</td>
</tr>
<tr>
<td>7</td>
<td>References</td>
<td>56</td>
</tr>
<tr>
<td>8</td>
<td>Appendices</td>
<td>59</td>
</tr>
<tr>
<td>8.1</td>
<td>Appendix A: Source code</td>
<td>59</td>
</tr>
<tr>
<td>8.2</td>
<td>Appendix B: CD-ROM contents: experimental data and code</td>
<td>111</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 2-1 Estimation of the number of fall detection studies.</td>
<td>6</td>
</tr>
<tr>
<td>Figure 2-2 Previous System Setup</td>
<td>7</td>
</tr>
<tr>
<td>Figure 2-3 The new board and software components</td>
<td>8</td>
</tr>
<tr>
<td>Figure 3-1 The new system overview</td>
<td>17</td>
</tr>
<tr>
<td>Figure 3-2 The database view of the stored active sensors data</td>
<td>19</td>
</tr>
<tr>
<td>Figure 3-3 Fall patterns performed by volunteers</td>
<td>21</td>
</tr>
<tr>
<td>Figure 3-4 Three views of experiment recording</td>
<td>22</td>
</tr>
<tr>
<td>Figure 3-5 Sample dataset file for Weka framework</td>
<td>25</td>
</tr>
<tr>
<td>Figure 3-6 Weka Explorer GUI preprocessing stage –attribute analysis</td>
<td>26</td>
</tr>
<tr>
<td>Figure 3-7 Weka Explorer GUI preprocessing stage –view all attributes</td>
<td>27</td>
</tr>
<tr>
<td>Figure 4-1 The full system developed</td>
<td>29</td>
</tr>
<tr>
<td>Figure 4-2 Sample frames for the pattern “Tripping Falling Backward”</td>
<td>31</td>
</tr>
<tr>
<td>Figure 4-3 The database view of the same set of frames shown in Figure 4-2</td>
<td>32</td>
</tr>
<tr>
<td>Figure 4-4 Live view of the carpet data display</td>
<td>32</td>
</tr>
<tr>
<td>Figure 4-5 Two notifications methods to alert for occurrence of event</td>
<td>33</td>
</tr>
<tr>
<td>Figure 4-6 The count of the active sensors in each frame</td>
<td>34</td>
</tr>
<tr>
<td>Figure 4-7 The count of the active sensors in a window size (WS) Fall only</td>
<td>35</td>
</tr>
<tr>
<td>Figure 4-8 The count of the active sensors in a window size (WS) Walk only</td>
<td>36</td>
</tr>
<tr>
<td>Figure 4-9 The size of the connected sub region and the frame at which the fall was detected for different window sizes WS=1, 3, 8</td>
<td>37</td>
</tr>
<tr>
<td>Figure 4-10 The size of the connected component of the active sensors in a single frame (WS=1)</td>
<td>38</td>
</tr>
<tr>
<td>Figure 4-11 The size the connected component of the active sensors in a window of frames (WS&gt;1)</td>
<td>39</td>
</tr>
</tbody>
</table>
Figure 4-12 The number of falls occurred and detected using 5 algorithms for the 4 falling patterns.................................................................41

Figure 4-13 The J48 pruned tree. (Train-Test option).................................................................45

Figure 4-14 The J48 pruned tree. (Cross-validation option)..................................................47

Figure 4-15 The J48 pruned tree (66% split)........................................................................49
LIST OF TABLES

Table 2-1 List of Fall detection systems .................................................................4
Table 2-2 The confusion matrix for our classifiers .................................................13
Table 3-1 Comparison of set of low power and cheap boards...............................15
Table 4-1 Count of detected falls for 5 algorithms. There were 14 fall patterns, a
sensitivity produced for the same of the detected falls, and a false negative rate
(FNR) for patterns detected as no falls where falls occurred.................................40
Table 4-2 Classifier performance of the training set ..............................................43
Table 4-3 Classifier performance of the testing set .................................................44
Table 4-4 Classifier performance of the 10-Fold cross validation............................46
Table 4-5 Classifier performance of the 66% percentage split ...............................48
ABSTRACT

A high-risk problem for seniors is falls. Therefore, there is value in detecting falls. We enhanced smart carpet, which is a floor based personnel detector system, to detect falls using a faster but low cost processor. Our new hardware front end reads from 128 sensors (the sensors output a voltage due to a person walking or falling on the carpet). The processor is Jetson TK1 [1], which provides more computing power than before. We generated a dataset with volunteers who walked and fell to test our algorithms. Data Obtained allowed examining data frames read from the data acquisition system. We used different algorithms and techniques, and varied the windows size of number of frames \((WS \geq 1)\) and threshold \((TH)\). We found that at \((WS = 8)\), and threshold \((TH = 8)\) using connected component labeling algorithm (CCL) produced a fall sensitivity of 87.9%. We then used the dataset obtained from applying a set of fall detection algorithms and the video recorded for the fall patterns experiments to train a set of classifiers using multiple test options using the Weka framework. We found that the widow size \((WS = 8)\) at a threshold \((TH = 8)\) using connected component algorithm generated attribute contributed to the fall sensitivity. Other algorithms attributes did not contribute significantly to the detection of the fall. More work needs to be done to embed the software into a single integrated package.
CHAPTER 1: INTRODUCTION

According to the World Health Organization-WHO, a fall is defined, as “inadvertently coming to rest on the ground, floor or other lower level, excluding intentional change in position to rest in furniture, wall or other objects” [2]. It is known that a senior who falls is at risk for serious injury and after necessary spiraling down eventually to death [6]. Researchers concern is to develop new technology or enhance existing one to detect falls and reduce the consequences of a fall.

Many fall detection systems have been developed. Our lab is focusing on state of the art floor based personnel detector system we call the smart carpet. The system uses a signal-scavenging technique wherein a sensor made from a conductive material picks up stray 60 Hz noise to detect presence of the person. It has sensors installed under the carpet, and the electronics can send sensor activation data, which is modified to produce notifications to cell phones or email through the Internet when a fall or any set of events occur. Our laboratory developed a low cost system to detect human falls on the floor [3, 4, 5]. To make the smart carpet “smarter”, we used new low cost and powerful hardware, the JetsonTK1 board. This hardware added computing power to develop and assess different falling detection algorithms. We added a computational intelligence (CI) component and the ability to run CI on the data generated form the data acquisition system. We measured the sensitivity of the system and other metrics of intelligence for detection of falls and produced a smarter fall detection system.

One limitation of the previously developed systems is the need for external hardware and remote computers to display the activity on the carpet. Our hardware computing resources helped develop web application (using MySQL database and
Apache server) to display the occurrence of events on the carpet in a quasi-real-time. To notify the caregivers instantly of an occurrence of a fall we send SMS or email.
2 CHAPTER 2: BACKGROUND

2.1 Rationale

Falls and fall-related injuries are major incidents, especially for elderly people, placing them at risk for serious injury, functional decline, and health care services utilization. Approximately 28-35% of people aged 65 and over fall each year increasing to 32-42% for those over 70 years of age [2]. Severe fall injuries can also lead to deaths [6]. Several studies have shown that better outcomes are correlated with rapid initiation of medical intervention immediately after a fall [7]. The duration of hospital stay due to falls varies; however, it may be long taking on average 22 days in the USA [2]. The economic impact of falls is critical to family, community, and society. The average cost of hospitalization for fall related injury for people 65 year and older is US$ 17,483 in the USA [2]. So technologies for helping elderly live independently, with quick fall detection, and prompt response on emergency, reduces the healthcare costs, and provides improved caregiver access to elderly.

There has been much research interest in fall detection and alert systems. Regardless what system exists, they all focus to facilitate daily living of elders. Typically, fall detection systems have a generic functionality: when an elderly person falls, data measurement (pressure sensors, noise based sensors, image sensors, audio signals sensors...etc.) send data to the detection system, which will run fall detection algorithm(s), and triggers the alerting system to notify the caregivers or authorized personnel to provide rapid assistance. All fall detection systems have a common objective; to distinguish a fall from activities of daily living, which tends not to be an easy problem to solve. Sensors and technologies have been developed, including smart
carpet (noise scavenging) based sensors, pressure mats, images, acceleration signals...etc. are summarized in Table 2-1. The table shows that the fall detection systems can be classified into two main categories: non-computer vision-based methods and computer vision-based methods. Some require the person to wear special devices like FitBit, bracelets, neck collars, and so on. The wearable devices that are not bound by place, but have their own disadvantages such as must be worn concerns for battery drain, and data collection and activities reporting require internet or cellular services. In contrast, context-aware devices are used where the sensors are deployed; examples are smart carpet, pressure mats, Kinect devices and others.

**Table 2-1 List of Fall detection systems**

<table>
<thead>
<tr>
<th>System /Sensor</th>
<th>*Basis</th>
<th>Features</th>
<th>*Algorithms</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smart Carpet</td>
<td>NCV</td>
<td>Noise scavenging system, count of number of active sensors.</td>
<td>Connected component labeling.</td>
<td>[3,4,5,8]</td>
</tr>
<tr>
<td>Microsoft Kinect camera</td>
<td>CV</td>
<td>Vertical velocity and acceleration, Occlusion adjusted change in the number of elements on the x-y plane after projection of points below knee height, Frame-to-frame vertical velocity, Silhouette, curvature scale space, Deformation on the joint structure.</td>
<td>Decision Trees, event segmentation, SVM, (FFNN), KNN.</td>
<td>[9,10,11]</td>
</tr>
<tr>
<td>Doppler radar and motion sensors</td>
<td>NCV</td>
<td>Reduces false alarm due to other people or visitors) to the resident, Mel-frequency cepstral coefficients</td>
<td>Fusion between a doppler radar system and a motion sensor network. KNN, SVM, Bayes</td>
<td>[12,13,14]</td>
</tr>
<tr>
<td>Microphone array (acoustic-FADE)</td>
<td>NCV</td>
<td>Circular microphone array that captures the sounds. Height threshold estimation</td>
<td>Beamforming, NN classifier</td>
<td>[15,16]</td>
</tr>
<tr>
<td>Floor vibration sensor</td>
<td>NCV</td>
<td>Raw vibration signal, floor mounted</td>
<td>N/A</td>
<td>[17]</td>
</tr>
<tr>
<td>Wearables, and smart phones</td>
<td>NCV</td>
<td>Accelerator, gyroscopes, microphones, pressure sensors, GPS</td>
<td>Sensor position</td>
<td>[18-21]</td>
</tr>
</tbody>
</table>

Researchers developed different systems to detect falls maintaining the privacy of the person. In [9] a two-stage system is used for fall detection. The first stage of the system characterizes the vertical state of a segmented 3-D object for individual frames, and then identifies on ground events through temporal segmentation of the vertical state time series of tracked 3-D objects. The second stage of the system utilizes an ensemble of decision trees and features extracted from an on ground event to compute the confidence that a fall preceded it. One limitation in this work is that far distance from the Kinect makes segmentation difficult and decreased precision of the pixel depth estimates. Therefore, there is a need for improved Kinect and multiple devices to cover all areas.

[15, 16] A system consisting of a circular microphone array detects falls by capturing the sound, generated during falls. It correctly locates the source of falling sound in attempt to reduce false alarms, which also makes the system robust under noisy situations. In [10] Microsoft Kinect presents a depth-based human fall detection technique, utilizing both shape-based and learning-based classifiers. In [17] a fall detection system based on floor vibration using a piezoelectric sensor was designed.

Several studies [18-21] described the use of wearable devices as fall detection systems focusing on the optimal placements of the sensor on the human body. Some placed the sensors on the head, waist, thigh, trunk, chest, and ankle. Researchers showed interest in using smartphones as fall detection system; since most of the wearable sensors are embedded in smartphones, and it has good cost effectiveness, usability and online time (availability and accessibility) [22].

Our lab uses context-aware type, non-computer vision based fall detection system [8]. The motion on the carpet activates a set of sensors that output a voltage signal that is
amplified, decoded, and then translated into a frame of data read by the fall detection hardware, such as the Jetson TK1 board we are using. The software component executes a set of computations, which determines the occurrence of a fall or not. Once a fall is detected, the caregivers are notified by email or SMS. Figure 2-1(below) shows the increase of the context-aware systems compared to wearables and smartphone based system [23].

Figure 2-1 Estimation of the number of fall detection studies. Detection techniques were classified into three categories. The line graph (associated with the right axis) represents the estimated absolute number of studies published in the three categories from 2005 to 2012. The bar graph (associated with the left axis) shows the estimated percentage of studies published every year in relation to the total number of existing studies for each category (e.g., 43.3% of the existing smartphone-based studies were published in 2012) [23]

A full System was developed to detect falls [5, 8]. As shown in Figure 2-2 (below), the system includes the sensor data acquisition, manipulating, data reading, storage, display and communication. The system operates by recording the person’s movement and storing the floor sensor data. Then using a sheevaplug, a low cost, but
slow, computer for data storage and communication. The system needed another desktop to run a display program utilizing client server communication.

The system has two major issues; the extra hardware, remote computer, that uses client server communication protocol (TCP) that has delays of 2 to 3 frames to show the steps performed on the carpet on the remote computer. Another issue was the processing power of the Sheevaplug that limits utilizing more sophisticated, hence, useful algorithms and programs to enhance the speed and the usability of the system.

Figure 2-3 (below) shows this new hardware and software component. The new computer, Jetson TK1, has a state of the art Kepler GPU with 192 CUDA cores, with (4 plus 1) quad core 2.3 GHz processors and 2 GB RAM. The computer produced by NVIDIA featuring a Tegra K1 SoC (System on Chip) has an embedded Linux
development platform. Equipped with preinstalled Ubuntu OS. It allows software development languages and frameworks like C, C++, Java, and CUDA [1]. The computing resources make it best candidate to enhance and improve the performance of the smart carpet technology. We have previous work provided a detailed description of the system setup, data acquisition system, data handling, and storage [5].

![New System: Jetson TK1](image)

**Figure 2-3 The new board and software components**
The new board and software components to replace the old system shown in Figure 2-2

This work proposes a new data storage method, new hardware, and added computational intelligence to the operation of the system to understand and learn the system, and detect falls.

### 2.2 System overview

The system consists of sensor array made into four segments A, B, C, and D. Each segment is connected to data acquisition system [8], which is connected to our computer Jetson TK1 through serial to USB connector. The data acquisition system scans the carpet and provides the serial data in HEX format. Each frame is one scan over the 128 sensors where a frame consists of data, segment labels, start (S) and end (E) markers. Once received by the Jetson TK1 board, a software program written in Java parses the frame data and store the data in different formats: raw, Comma Separated Value (CSV)
of active sensors, matrix, sum matrix, and relational database using MySQL database management system. The latter format helped us show live and history data on demand.

The new hardware computing power allowed us to build web application to display the activity on the smart carpet in quasi real-time by configuring the Jetson TK1 board as database (MySQL) and web (Apache) server. Compared to the previous work [5], where the Sheevaplug communicated with the remote computer using client server configuration with some delay, this was a substantial improvement.

We processed the data to detect a fall and send notification accordingly. We were able to send notifications via E-mail, and Short Messaging Service (SMS) free of charge. Other tools of notifications are possible like paid SMS, pager, and phone calls [31].

Previous work [5] used an image processing based algorithm, the connected component labeling, to detect a fall. It used the matrix representing a single frame read from the data acquisition system. We adopted the same algorithm and applied it on a window size of number of frames (WS) rather than single frame. We will abbreviate window size of number of frames by “window size”. Convex hull area using quick hull technique [24] is another algorithm we developed and tested to detect falls. A heuristics algorithm we also developed and tested. It is based on the count of neighboring active sensors in each frame read. All these algorithms use a preset threshold (TH). The threshold was determined by experiment using as a reference the video of fall patterns performed by the 10 volunteers, who helped us generate the data.

We then built a training dataset using both the video recorded and the fall detection algorithms to train and test the performance a set of computational intelligence classifiers; Naïve Bayes [25], Multilayer Perceptron [26], Decision Tree (J48 aka
C4.5)[27] and Logistic Function [28]. We used the Weka framework [29], which is a collection of machine learning algorithms for data mining tasks. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization.

2.3 WEKA classifiers

We used Weka to analyze and evaluate different computational intelligence algorithms or models to help detect a fall. In the following sections, we will briefly describe the operation of the selected algorithms: Naïve Bayes, Logistic, Multilayer perceptron, and decision tree J48 (also known as C4.5 classifier).

2.3.1 Naïve Bayes

Naïve Bayes methods are a set of supervised learning algorithms based on applying Bayes’ theorem (Bays rule of conditional probability) with the “naïve” assumption of independence between every pair of features. It uses Maximum A Posteriori (MAP) estimation to estimate the probability distribution, the prior and the likelihood probabilities. Combining both the prior and the likelihood probabilities to form a posterior probability produces the final classification [25].

2.3.2 Logistic function

In linear regression, classes expressed as a linear combination of the attributes, with predetermined weights. It then predicts the data based on a threshold. Of interest is the difference between actual and predicted values. The weights are chosen to minimize
the sum of the squares of these differences over all training data. In logistic regression, the original classes replaced by a target variable and resulting values are no longer constrained to a binary interval, but can lie anywhere from negative infinity and positive infinity [28].

2.3.3 Multilayer Perceptron (MLP)

A multilayer perceptron is a feedforward artificial neural network model that maps sets of input data onto a set of appropriate output. It is a modification of the standard linear perceptron in that it uses three or more layers of nodes with nonlinear activation functions (sigmoid for example), and it can distinguish data that is not linearly separable, or separable by a hyper-plane. Learning occurs in the perceptron by changing connection weights after each piece of data is processed, based on the amount of error in the output compared to the expected result. This is an example of supervised learning, and is carried out through backpropagation, a generalization of the least mean squares algorithm in the linear perceptron [26].

2.3.4 J48 Decision Trees

The J48 decision trees algorithm is a Weka implementation of decision tree learner. A decision tree is a predictive machine-learning model that decides the target value of a new data instance based on various attribute values of the available data. The internal nodes of a decision tree denote the different attributes; the branches between the nodes tell us the possible values that these attributes can have in the data instances, while the terminal nodes tell us the classification. In order to classify a new data instance, it
first needs to create a decision tree based on the attribute values of the available training data. Therefore, whenever it encounters a set of training instances it identifies the attribute that discriminates the various instances most clearly with highest gain. Among the possible values of this feature, if there is any value for which the data instances falling within its category have the same value for the target variable, then we terminate that branch and assign to it the target value that we have obtained. For the other cases, we then look for another attribute that gives us the highest information gain. Hence, we continue in this manner until we either get a clear decision of what combination of attributes gives us a particular target value or we run out of attributes. In the event that we run out of attributes, or if we cannot get an unambiguous result from the available information, we assign this branch a target value that the majority of the items under this branch possess [27].

2.4 Performance of fall detection

We conducted the experiments to determine the performance of the fall detection algorithms and classifiers. We found that the confusion matrix of all methods and algorithms represents a base for the performance analysis. The confusion matrix is more commonly named contingency table. In our case we have two classes (fall, no fall), and therefore a 2x2 confusion matrix, the matrix could be arbitrarily large. The number of correctly classified instances is the sum of diagonals in the matrix; all others are incorrectly classified. Table 2-2 shows a two –class confusion matrix. The relation between Predicted and Actual classes gives four outcomes. The true positives (TP) and true negatives (TN) are correct classifications. A false positive (FP) is when the outcome
is incorrectly predicted as fall (or positive) when it is actually no fall (negative). A false negative (FN) is when the outcome is incorrectly predicted as negative when it is actually positive. The true positive rate is TP divided by the total number of positives, which is TP + FN; the false positive rate is FP divided by the total number of negatives, which is FP + TN. The overall success rate is the number of correct classifications divided by the total number of classifications: \((TP + TN) / (TP + TN + FP + FN)\).

Table 2-2 The confusion matrix for our classifiers

<table>
<thead>
<tr>
<th>Actual Class</th>
<th>Predicted Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes: Fall</td>
<td>Yes: Fall</td>
</tr>
<tr>
<td>No: No Fall</td>
<td>False Positive (FP)</td>
</tr>
</tbody>
</table>

Another two measure we can determine from the confusion matrix is the Specificity and Sensitivity. Sensitivity refers to the proportion of patterns with actual patterns with a fall that has been classified as falls (TP). Specificity refers to the proportion of actual patterns with no fall that has been classified as no fall, which is 1 – FP.

The J48 decision trees classifier outputs a decision tree or set of rules. The tree employs a case's attribute values to map it to a leaf designating one of the classes. Every leaf of the tree is followed by a cryptic \((n)\) or \((n/m)\). The value of \(n\) is the number of instances in the data that are mapped to this leaf, and \(m\) (if it appears) is the number of them that are classified incorrectly by the leaf.
3 CHAPTER 3: METHODS

Previous work [3, 4, 5, 8] produced a system to detect motion of personnel with sensors on a floor surface developed and extensive smart carpet technology. Improving hardware, software and communication protocols resulted. However, in order to have a reliable and full-featured system, which may detect falls, and more robustly monitor elderly and analyze the extensive long-term data produced we needed a more powerful hardware. Table 3-1 shows specifications of some low power and low cost boards. Comparing the prices, we found that the Raspberry Pi is the lowest cost with good specifications, but a single processor (700 MHz clock), and low memory. Future work sees a need to run concurrent or multithreaded programs. In considering the value of multiprocessing offer same effect and focusing on low cost, we found that the Jetson TK1 board is the candidate with the needed functionalities.
### Table 3-1 Comparison of set of low power and cheap boards

<table>
<thead>
<tr>
<th>Board</th>
<th>Sheevaplug</th>
<th>Raspberr y Pi V1</th>
<th>Raspberry Pi V2</th>
<th>Jetson Tk1</th>
<th>TS-7250 V2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td>1.2GHz Marvell, single core</td>
<td>700MHz ARM, single core</td>
<td>900MHz quad-core ARM Cortex-A7</td>
<td>quad-core 2.3GHz ARM Cortex-A15 CPU</td>
<td>1 GHz ARM Marvell PXA168</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>512 MB</td>
<td>512 MB</td>
<td>2 GB</td>
<td>512 MB</td>
<td></td>
</tr>
<tr>
<td><strong>Storage</strong></td>
<td>512 MB Flash</td>
<td>SD, MMC multiple sizes</td>
<td>MicroSD</td>
<td>16GB fast eMMC, with SD multiple sizes</td>
<td>2 GB SLC eMMC Flash Storage</td>
</tr>
<tr>
<td><strong>peripherals</strong></td>
<td>JTAG through USB Ethernet</td>
<td>HDMI, GPIO, I2C, SPI, Ethernet, USB</td>
<td>HDMI, GPIO, I2C, SPI, Ethernet, USB</td>
<td>HDMI, GPIO, USB, RS232, I2c SPI, JTAG, Ethernet</td>
<td>RS232, USB, 2 Ethernet ports</td>
</tr>
<tr>
<td><strong>Price</strong></td>
<td>99$</td>
<td>25$</td>
<td>3$</td>
<td>192$</td>
<td>165$ order of 100 pcs</td>
</tr>
<tr>
<td><strong>Power</strong></td>
<td>100-240VAC/50-60Hz Max. 20W</td>
<td>5 V via Micro-USB, 600 mA (3.0 W)[3]</td>
<td>5 V via Micro-USB, 800 mA[58](4.0 W)[59]</td>
<td>a 12V DC barrel power jack</td>
<td>5VDC or 8-28VDC power input</td>
</tr>
<tr>
<td><strong>Size</strong></td>
<td>110mm (L) x 69.5mm (W) x 48.5 mm (H)</td>
<td>(3.370 in x 2.224 in)</td>
<td>(3.370 in x 2.224 in)</td>
<td>5&quot; x 5&quot; (127mm x 127mm)</td>
<td>96.5mm x 114.3mm</td>
</tr>
<tr>
<td><strong>Other</strong></td>
<td></td>
<td></td>
<td>Kepler GPU with 192 CUDA cores</td>
<td>8 or 17 K Lut FPGA, A/D convertor</td>
<td></td>
</tr>
</tbody>
</table>
3.1 The Manufacturer provided board for fast computing

We used Jetson TK1 computer with an embedded Linux development platform produced by NVIDIA featuring a Tegra K1 SoC (System on Chip) (CPU+GPU+ISP in a single chip), purchased for $192 USD. Jetson TK1 comes pre-installed with Linux4Tegra OS (Operating System) (basically Ubuntu 14.04 with pre-configured drivers). Processing occurs with quad-core 2.3GHz ARM Cortex-A15 CPU and the Kepler “GK20a” GPU (Graphics Processing Unit) with 192 CUDA cores. The Cortex-A15 is a battery-saving shadow-core. The board has 2 GB 933 MHz memory with 16 GB fast eMMC storage. It requires 12V DC power. As with the well-known Raspberry Pi, the Jetson TK1 development board has some PC-oriented features such as SATA, mini-PCIe and a fan reduce heat to allow continuous operation under heavy workloads. The Board is 5” x 5” with typical power consumption between 1 to 5 Watts [1].

3.2 Data acquisition and processing

We used the smart carpet technology from previous work [3,4,8]. Figure 3-1 (below) shows the system setup and required software component. Sensors of the carpet were scanned and the data is obtained by the Jetson TK1 board as a series of frames of length 19 bytes at a speed of 19.2Kbps.
We convert the data to different formats for its further analysis, storage and display. The details of conversion and storage techniques in three different formats: raw data with timestamp, matrix and CSV (Comma separated Value) were discussed in [5]. Our work proposes enhancements in both storage and display of carpet data. We use MySQL database management system to store the carpet active sensor data, and Apache webserver to help produce the web application to display the carpet data on browser. Besides the database, we used the two storage formats consists of frames of sensor data time-stamped as read by the data acquisition board. The matrix shows each frame of sensor data such that “0” means sensor not activated and “1” the sensor is activated. The form of the matrix follows that of the carpet layout on the floor, which provides...
positional information. We had to provide computing resources for database storage, carpet display and notification. The latter is to notify authorized people of a potentially dangerous occurrence.

3.2.1 Data storage and display

We developed web application to display activity in the browser to replace previous work [5]. Using database management system, MySQL, we were able to save active sensor data with their corresponding locations on the carpet into the database, and then query the data and display in web browser using the Apache HTTP server. MySQL database was used on the Jetson TK1, which allowed access to the carpet data any time. This allows us to display the occurrence of steps on the carpet instantly, and as desired. MySQL allows us to query the data any time so we can view the past events.

Figure 3-1 shows the stored sensors data in the database. We store the coordinates (x,y) and the location of the active sensors, the date and time of the insertion into the database, which usually requires negligible time. The database table structure follows that of Figure 3-2 with the following fields, a field for unique id, Xcord, Ycord fields represent the location of the active sensors, and the Location field shows the numerical position of the sensor. The DateTime represents the time of insertion into the database. Our experiments shows that this matches to a certain degree the time the board received the data frames from the data acquisition system. Hence, allowing for almost instant display of the occurrence of the steps on the web browser. Space was also provided for a de-identified designation of the volunteer subjects and experimental fall patterns used.
The database view of the stored active sensors data

The id coordinates, location, and time that the data frame was received by the board after receiving active sensor. We do also have a way to designate the specific experiments and fall type. Subjects’ names were de-identified and placeholder was stored for experimental purposes.

### 3.2.2 Notifications

In order to alert the authorized party for any possible fall occurrence, many ways of notifications were developed and tested. Visual and live display of the person activity as shows in Figure 4-4 in section 4.1. Another approach is to send email to set of preconfigured personnel. Also, by sending e-mail to cell phone numbers as SMS. All those are free of charge. However, we can have use a commercial or non-free notification tools like paging, paid SMS and phone calls.
3.3 Fall detection

To assess the new system’s ability to detect falls, we collected data from 10 adult volunteers. We then built a training dataset to train and test the performance a set of computational intelligence algorithms; Bayes Network, Multilayer Perceptron, Decision Tree (J48 aka C4.5) and Logistic function. The following sections describe data collection, analysis, building training and testing data and the performance of each algorithm.

3.3.1 Data collection experiments

10 volunteers performed 14 falling patterns described by 4 categories: falling from standing, falling form tripping, falling from sitting on chair, and falling from slumping form a chair. Figure 3-3 (below) shows the detailed fall patterns performed. First row describes the Standing to falling patterns (4 patterns). Second row describes the Tripping to falling patterns (4 patterns). The third row describes the Sitting to falling patterns (3 patterns) and the last row describes the slumping to falling patterns (3 patterns).

Figure 3-4 (below) shows the views used to record the video of all fall patterns. We recorded the video in two views for the fall pattern and one for the occurrence of the steps on the developed web application showing the active sensors. We used this data to study and develop fall detection algorithms. The collected data will be reused for future work and analysis. The outputs of the experiments are the video files, database files for
all active sensor data, and the raw data files from the frames obtained from the data acquisition system.

**Figure 3-3 Fall patterns performed by volunteers**
First row describes the standing to falling patterns (4 patterns). Second row describes the tripping to falling patterns (4 patterns). The third row describes the sitting to falling patterns (3 patterns) and the last row describes the slumping to falling patterns (3 patterns). Each performed by 10 volunteers
The three views we recorded while performing the experiments. The top left view: the front and left side view of the person performing the fall pattern. The top right view: the back and right side of the person. The front view and back view alternate.

### 3.3.2 Data Analysis and algorithms

The data from the 10 volunteers consisted of 14 fall patterns replicated by each volunteer so that we had 10 instances of each fall patterns. We used this raw data obtained from the data acquisition system, converted into a matrix and used both data in different fall detection algorithms.

An important strategy is to use the connected component-labeling algorithm to build training data for classifiers. We developed a new approach using a window of number of frames of different sizes (WS). The work done by [5] was focused on a single frame. We first counted the number of active sensors per frame and used different window sizes. Then, we used the same approach to find the connected component within that set of window. The algorithm declares a fall occurs when the size of largest contiguous sub region is found to be greater than a predefined threshold (TH), where threshold was found experimentally. We developed and studied three main fall detection algorithms: Connected Component Labeling (CCL) with different window sizes, the
Convex Hull area also with different window size of number of frames, and a heuristic algorithm based on active sensors count and their neighborhood characteristics.

In connected component labeling as described in [5], we applied the same procedure for both single frames and window size encompassing variable number of frames. In our work, we modified the algorithm to account for the active on the sides of the carpet, which [5] ignored.

In convex hull area algorithm, we used a windows size of number of frames to form an array list of active sensors, and then apply the quick convex hull algorithm [24]. We found the points forming the convex hull (polygon) for the set of active sensors on the carpet. We calculated the area of the polygon according the shoelace algorithm [30]. To detect a fall, we run the algorithm for different window sizes and thresholds.

In heuristic algorithm, we converted the array of sensors data into a matrix format. Then we counted the number of active sensors that are contiguous in a row or column, and if that count meets or exceeds some threshold (TH), we mark it as a fall. We evaluated and compared the performance of all the algorithms above.

Our goal is to apply computational intelligence to a system for personnel monitoring. We used a set of classifiers from WEKA framework [29] and trained them by a training data generated from the algorithms discussed above. Issues include how we generated and prepared the data, what classifiers we used and what metric we measured from each classifier.
3.3.3 Preparing the data

We used 8 patterns (Walk-Fall) of the 14 patterns performed by the 10 volunteers to generate the training and testing data. For each subject we identified walk, fall, stand, and walk again. We used the portion of walk and fall for this analysis. We used both the active sensors count and the connected component size, as well as different windows size of number of frames (WS=1, 8) with threshold values (TH=4, 7) respectively. In addition to these criteria, we used the video recorded for each pattern to label the training and testing data. Figure 3-5 shows portion of the generated training dataset. The file starts with the dataset’s name indicated by the @relation tag, the attribute information using @attribute shows the attributes or features we used to predict a fall: SF_Count: Single frame (WS=1) active sensors count. SF_CCL: Single frame (WS=1) connected component size. WS8_Count: multiple frames (WS=8) active sensors count, and WS8_CCL: multiple frames (WS=8) connected component size. The last entry is the class: yes (Fall), and no (No Fall). A similar file structure used for the testing dataset. Weka explorer can accept different data files and formats. We used the native data storage ARFF format. The @data line shows the list of the instances with commas separating the attribute values.
The beginning portion training data file used to train the classifier in WEKA framework.

Attributes or features:
- @ATTRIBUTE SF_Count  NUMERIC
- @ATTRIBUTE SF_CCL   NUMERIC
- @ATTRIBUTE WS8_Count NUMERIC
- @ATTRIBUTE WS8_CCL  NUMERIC
- @ATTRIBUTE class {yes, no}

@data:
- denotes the start of the training or testing data

We used the Weka explorer to preprocess the dataset file and analyze it. Figure 3-6 (below) shows the explorer GUI interface. This interface tells us more about the dataset: It has 1153 instances (110 of class yes, and 1043 of class no) and 5 attributes; the attributes are SF_Count, SF_CCL, WS8_Count, WS8_CCL and Class (yes: fall, no: no fall). The WS8_CCL is selected, for example, and has no missing data, 22 distinct values (the number of dissimilar values contained for the selected attribute) and 3 unique
values (the number and percentage of instances having a value for this attribute that no other instances have in the data). It has continuous values from 0 to 25 with mean of 2.65. A histogram shows how often each of the two values of the class (yes-blue, no-red) occurs for each value of the WS8_CCL attribute. Similarly, we can see the details of all data attributes.

![Weka Explorer GUI preprocessing stage – attribute analysis](image)

**Figure 3-6 Weka Explorer GUI preprocessing stage – attribute analysis**
It shows all attributes, and their characteristics: nominal or numeric, mean, average, distinct and unique numbers. It shows number of training instances and their classes.

Figure 3-7 (below) shows a visualization of all attributes in Weka explorer GUI preprocessing stage in one view. The x-axis represents the range of values each attribute can take, and the y-axis represents the total occurrences per attribute value. It shows how often each of the two values of the class (yes-blue, no-red) occurs for each attribute.
3.3.4 WEKA classifiers

We used Weka to analyze and evaluate different computational intelligence algorithms or models to help detect a fall. Weka supports the following classifiers: Naïve Bayes, Logistic, Multilayer perceptron, and decision tree J48 (also known as C4.5 classifier). We used the generated datasets (training and test) for the falling patterns to train on the selected classifier. We measured the accuracy of each classifier by calculating a 2x2-confusion matrix or (contingency table) that describes the performance of the classifiers. Sensitivity and Specify were calculated and studied. We visually viewed the set of classification rules used by generating the decision tree from the J48 decision trees classifier.
We used three different testing options: Train and test, 10-Fold Cross-validation, and Percentage split. We evaluated the accuracy of the classifier to the dataset provided based on the contiguous table provided. We decide what feature(s) or attribute(s) contribute most in the classifier decision for fall detection. Most classifiers should run within reasonable time compared to the size of the dataset. However, classifiers with testing option that requires less computing resources are preferred. For example, 10-Fold cross-validation needs more time compared to the others; it breaks the data into 10 sets of size n/10 (n: number of the data set vectors or instances), and then trains on 9 datasets and tests on 1, repeats 10 times and takes a mean accuracy. The next options, train and test option, we left 10% of the dataset for testing and used the 90% for training. However, in percentage split, the training and testing portion can be automated and randomly processed without biasing; this makes for better classification. We compared the performance classifiers for each of these three testing options.
CHAPTER 4: RESULTS

Continuing the efforts done on building a data acquisition system front-end to produce data, and the efforts to take the data and perform further computational analysis with it. Here we discuss the results of the work we have done putting the system together. Figure 4-1 shows the system as it has evolved, wherein we show the electronics front end and we show the addition of the Jetson TK1 board, and the software work we have done, particularly in providing computational intelligence to the system. As before, we retained the ability to provide signaling and indeed even improved a little bit on that. But, importantly the work here reflects the use of formal computational intelligence techniques to identify the activity of the personnel on the floor.

Figure 4-1 The full system developed
All the pieces of the system: data acquisition system, Jetson TK1 board, software components, and results.(Notifications, quasi real-time display, and intelligence classifiers)
Speed is important in smart carpet technology to provide rapid data evaluation, display and notification of the results. In particular, high computing speed is essential for evaluating large amounts of data for purpose of making a timely decision. Smart carpet technology has been enhanced by the use of Jetson TK1 NVIDIA GPU (4+1) core processor.

The new computer Jetson TK1 provides more processing power than the small processors like the raspberry pi used heretofore, allowing sophisticated algorithms and programs to run concurrently. As before reading data from the data acquisition system takes place within one full scan of all the sensors. The new faster system will process data within a shorter time frame to provide for quasi-real-time processing. Previous work was to develop algorithms to detect falls rapidly [5]. This chapter improves on that with a more formal computational intelligence performance. We demonstrate the ability of the new board to read, evaluate and display data, and to detect falls using four 4 different algorithms. We discuss limitations in the next chapter. We require the faster processor to achieve quasi-real-time performance.

4.1 **Database and webserver capabilities of the new processor**

We configured the board as web and database server. This allows us to use web-based resources for data acquisition purposes we produced a display program that can be accessed locally and remotely. Using the board as database and web server made display almost instantly available, unlike the raspberry pi: the moment a step occurs it shows up on the browser.
Figure 4-2 (below) shows sample of the sensor data (frames) obtained from the data acquisition system by a volunteer performing the “Tripping Falling Backward” pattern. The figure shows seven frames, each time-stamped with the time read by our board. Figure 4-3 (below) shows the data after we parse the frames. These frames give rise to an array showing only active sensors data (corresponds to binary “1”). So that the second from the bottom frame in Figure 4-2 (below), timestamped by (15:41:19.61) compared to the database row with id number 81371 in Figure 4-3 (below). This shows that sensor in row 2, column 8 was activated. This is then displayed on the browser instantly. Display of data can be delayed but not made faster.

Figure 4-4 shows the date and time stamps of the active sensors data for the “Tripping Falling Backward” pattern. Contrasting sensors data acquisition and displaying it (in milliseconds) to the video recorded for those experiments supports the apparent instantaneous display. In normal operation, we insert the data into the database, from which we generate the display. This results in a loss of time synchrony. However, a substantial benefit is the ability to recall the data when desired. Other work in the lab is data mining, displaying sensors data in the cloud, and developing a feature-rich portal.

Figure 4-2 Sample frames for the pattern “Tripping Falling Backward”
Only one active sensor in this set of frames. Sensor 66 corresponds to C2 according to the frame SA00000000B00000000C20000000D0000000E.
Figure 4-3 The database view of the same set of frames shown in Figure 4-2. Only one record inserted corresponds to the active sensor 66 or located at x=2, y=8 in the Cartesian coordinates (See Figure 4-4 below)

Carpet Live data

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2015-05-15 15:41:17</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2015-05-15 15:41:17</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2015-05-15 15:41:16</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2015-05-15 15:41:15</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2015-05-15 15:41:15</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2015-05-15 15:41:15</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2015-05-15 15:41:15</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2015-05-15 15:41:15</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2015-05-15 15:41:15</td>
</tr>
</tbody>
</table>

Figure 4-4 Live view of the carpet data display
Only active sensor data is displayed to show the steps of fall behavior. This view has the date and time of the inserted data into database. Each rectangle represents a sensor.

4.1.1 Notification

Fall detection assumes a responsibility to notify an authorized party that a fall has occurred. Figure 4-5 shows the emergency notification text received on smartphone as SMS and email to report fall occurrence. The message and levels are configurable.
Phone, pager and commercial SMS can be used to send alert and notification. We have confirmed that message receipt is nearly instantaneous. This confirms earlier work in our lab [5, 31]. The average time for email notification was 2.8 seconds, with similar times for with SMS messaging.

![Figure 4-5 Two notifications methods to alert for occurrence of event](image)

A fall message sent by (a) e-mail and (b) SMS. The message format is customizable.

### 4.2 Tripping Falling Backward

To assess the new system ability to detect falls, we collected data from 10 adult volunteers. A volunteer to walk, trip and fall backwards on to the carpet. Figure 4-6 shows the count of the active sensors in each frame obtained for the falling pattern “Tripping Falling Backward” for a period of 101 frames (14 seconds). As shown in the graph, at the beginning of the walk the count is zero, no activation of sensors. The moment the volunteer stepped on the carpet the sensors activated and were counted in
each frame. Counts increase or decrease, depending on sensor activation, rate of scan relative to the walk, and noise. So low levels of count occur until a fall occurs causing a sharp increase in the number of active sensors noticed as seen in frames (71-76). Ideal scenario for a fall is a jump in the count of active sensors and then a sharp decrease to zero (Frames 92-100). We see that if the fallen walker stays stationary on the ground the counts are all zero.

Figure 4-6 The count of the active sensors in each frame.
Data read from the data acquisition system for the falling pattern “Tripping Falling Backward”, for the period of 101 frames (14 seconds). Note the active sensor count for no walk, walk, and fall.

Figure 4-7 shows the count of the active sensors during a fall for three window sizes. The number of frames read for the same data defines each window. Specifically this came from the pattern “Tripping Falling Backward”. See Figure 4-7 for the fall period of 35 frames (5 seconds) for different window sizes (WS=1, 3, 8 frames) and window step size =1. From the figure, the fall produces a large accumulated count, 23.
For the same fall pattern, we look only at the walk, 8, and with window size 8, as shown in Figure 4-8.

In contrast, Figure 4-8 is taken from the walk part of the “Tripping Falling Backward” pattern. Again we plot the counts detected for 3 window sizes (WS=1, 3, 8). Compare the fall WS=8, (23) with the walk WS= 8, (8), and as expected the walk is substantially lower.

**Figure 4-7 The count of the active sensors in a window size (WS) Fall only**
The data read from the data acquisition system for the falling pattern “Tripping Falling Backward” for the period of 35 frames (5 seconds). It shows only the for three different window sizes (WS= 1, WS= 3, WS= 8) with window step 1.
Figure 4-8 The count of the active sensors in a window size (WS) Walk only
The data read from the data for the falling pattern “Tripping Falling Backward”. The walk estimated to be a period of 67 frames (9 seconds). The data was accumulated over 3 different window sizes (WS=1, WS=3, WS=8) and window step 1.

We ran the collected data for the fall pattern “Tripping Falling Backwards” on the computer and detected the fall using the size of the connected sub region using the connected component-labeling (CCL) algorithm. Figure 4-9 shows three runs using different window sizes (WS=1, 3, 8). The connected sub-region is calculated at the last frame of the window. For example, at windows size of 8, the CCL algorithm is run at frames 8, 16, 24 …etc. Figure 4-10 and Figure 4-11 show both the size of the sub-region, and the frame where the fall is detected. Results were verified by reviewing the video recorded and data collected.
Another metric we observed to detect a fall is the size of the connected component (CC) counted during a fall. Figure 4-10 shows the connected component sizes for Single Frame window, and Figure 4-11 shows multiple frames (WS=3, 8) for the falling pattern “Tripping Falling Backward”. In this case we only observed the connected component at the windows step size equals to the window size. Different window sizes detect the fall at different frames, since we locate the connected sub region at the last frame of the window after accumulating active sensors. The larger the window size, the
bigger the connected component size there is. This helped us build the training data used later (3.3.3 Preparing the data)

Figure 4-10 The size of the connected component of the active sensors in a single frame (WS=1)
The data read from the falling pattern “Tripping Falling Backward” for the period of 101 frames (14 seconds). It shows no walk, walk, and fall.
We measured the performance of the fall detection algorithms by counting the number of fall patterns that were detected as falls and provided the sensitivity of the detected falls on all patterns done by 10 volunteers. This is shown in Table 4-1, which also shows the window size and the threshold (TH) used for the decision. The shaded column in Table 4-1 shows the result of using the CCL-Multiple frames algorithm. At Window size (WS= 8, and TH= 8) we obtained a sensitivity or True Positive Rate (TPR) of 87.9% compared to lower window sizes. Also, it outperformed all other algorithms. Figure 4-12 shows graphical view of the results detailed in Table 4-1. We believe there is a room for each algorithm for improvement, so we decided to use Computational Intelligence to get better detection and system analysis. Section 4.4 Computational Intelligences shows detailed results.
Table 4-1 Count of detected falls for 5 algorithms. There were 14 fall patterns, a sensitivity produced for the same of the detected falls, and a false negative rate (FNR) for patterns detected as no falls where falls occurred

<table>
<thead>
<tr>
<th>Fall Patterns</th>
<th>CCL- Single Frame (WS=1, TH=4)</th>
<th>CCL- Multiple Frames (WS=3, TH=7)</th>
<th>CCL- Multiple Frames (WS=8, TH=8)</th>
<th>HULL Area- (WS=3, TH=7)</th>
<th>Heuristic (WS=1, TH=4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Standing Falling Forward</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>2. Standing Falling Backward</td>
<td>9</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>3. Standing Falling Right Side</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>4. Standing Falling Left Side</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>5. Tripping Falling Forward</td>
<td>6</td>
<td>6</td>
<td>10</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>6. Tripping Falling Backward</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>7. Tripping Falling Right Side</td>
<td>9</td>
<td>7</td>
<td>10</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8. Tripping Falling Left Side</td>
<td>8</td>
<td>7</td>
<td>10</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>9. Sitting Falling Forward</td>
<td>9</td>
<td>5</td>
<td>9</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>10. Sitting Falling Right Side</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>11. Sitting Falling Left Side</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>12. Slumping Falling Forward</td>
<td>5</td>
<td>3</td>
<td>6</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>13. Slumping Falling Right Side</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>14. Slumping Falling Left Side</td>
<td>10</td>
<td>6</td>
<td>9</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>Sensitivity (TPR)</td>
<td>80.7%</td>
<td>70%</td>
<td>87.9%</td>
<td>75.7%</td>
<td>75.7%</td>
</tr>
<tr>
<td>FNR</td>
<td>19.3%</td>
<td>30%</td>
<td>12.1%</td>
<td>24.3%</td>
<td>24.3%</td>
</tr>
</tbody>
</table>

*WS is the window size. TH is the threshold above which we say a fall occurs*
Figure 4-12 The number of falls occurred and detected using 5 algorithms for the 4 falling patterns.
CCL with WS=8, TH=8 shows best performance. It detects most occurred falls as falls with 87.9% sensitivity

4.4 Computational Intelligence

We used the data obtained from the previous sections to build the training and testing data for the classifiers described in Chapter 3. Training data, which we used in previous analysis we calculate four attributes; active sensors count and connected region size each in both single frame (WS=1, TH=4), and multiple frames (WS=8, TH=7).

We measured the performance of different classifiers using three test options:

- Use training set, supplied test set, section 4.4.1
- Cross-validation 10 folds, section 4.4.2
- Percentage split 66%, section 4.4.3

Next, we compare these classifications and test options with the results obtained in Table 4-1.
4.4.1 Training set, Supplied Test set option

We ran the Testing set for the four classifiers and then re–evaluated the models or classifiers on the testing set. Table 4-2 shows the performance of the classifiers when using training set. Table 4-3 shows the performance of the classifier after re-evaluated on testing set. Both tables have the same data that can mainly be obtained from the last row, the confusion matrix. All correct classifications are on the top-left (fall classified as fall) to bottom-right (No fall classified as no fall) diagonal. Everything off that diagonal is an incorrect classification of some sort.

Figure 4-13 shows the J48 pruned tree. The figure shows that the size of the connected component of windows of frames is the key attribute in deciding on fall detection. The size of the connected component for single frame and the sum of active sensors in a window of frames are of the same importance. And the active sensors counts for a single frame show no importance.
Table 4-2 Classifier performance of the training set

<table>
<thead>
<tr>
<th>Training set:</th>
<th>Naive Bayes Classifier</th>
<th>Logistic Regression Classifier</th>
<th>Multilayer Perceptron Classifier</th>
<th>J48 Classifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes: 100</td>
<td>992 / 1057</td>
<td>1025 / 1057</td>
<td>1031 / 1057</td>
<td>1029 / 1057</td>
</tr>
<tr>
<td>No: 957</td>
<td>6.1495 %</td>
<td>6.03 %</td>
<td>2.46%</td>
<td>2.65%</td>
</tr>
<tr>
<td>Total: 1057</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Correctly Classified Instances</th>
<th>Logistic Regression Classifier</th>
<th>Multilayer Perceptron Classifier</th>
<th>J48 Classifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes: 0.89</td>
<td>Yes: 0.77</td>
<td>Yes: 0.81</td>
<td>Yes: 0.79</td>
</tr>
<tr>
<td>No: 0.944</td>
<td>No: 0.991</td>
<td>No: 0.993</td>
<td>No: 0.993</td>
</tr>
<tr>
<td>False Positive Rate</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yes: 0.056</td>
<td>Yes: 0.009</td>
<td>Yes: 0.007</td>
<td>Yes: 0.007</td>
</tr>
<tr>
<td>No: 0.11</td>
<td>No: 0.23</td>
<td>No: 0.19</td>
<td>No: 0.21</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Confusion Matrix</th>
<th>Classified as:</th>
<th>Classified as:</th>
<th>Classified as:</th>
<th>Classified as:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fall No Fall</td>
<td>Fall No Fall</td>
<td>Fall No Fall</td>
<td>Fall No Fall</td>
</tr>
<tr>
<td></td>
<td>89 11</td>
<td>77 23</td>
<td>81 19</td>
<td>79 21</td>
</tr>
<tr>
<td></td>
<td>54 903</td>
<td>9 948</td>
<td>7 950</td>
<td>7 950</td>
</tr>
<tr>
<td>Testing set:</td>
<td>Naive Bayes Classifier</td>
<td>Logistic Regression Classifier</td>
<td>Multilayer Perceptron Classifier</td>
<td>J48 Classifier</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------</td>
<td>--------------------------------</td>
<td>---------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Fall: 10</td>
<td>Correctly Classified Instances</td>
<td>79 82.2917 %</td>
<td>90 / 96 93.75%</td>
<td>90 / 96 93.75%</td>
</tr>
<tr>
<td>No Fall: 86</td>
<td>Incorrectly Classified Instances</td>
<td>17 17.7083 %</td>
<td>6 / 96 6.25%</td>
<td>6 / 96 6.25%</td>
</tr>
<tr>
<td>Total: 96</td>
<td>True Positive Rate</td>
<td>Yes: 0.80 No: 0.826</td>
<td>Yes: 0.7 No: 0.965</td>
<td>Yes: 0.7 No: 0.965</td>
</tr>
<tr>
<td></td>
<td>False Positive Rate</td>
<td>Yes: 0.174 No: 0.20</td>
<td>Yes: 0.035 No: 0.30</td>
<td>Yes: 0.035 No: 0.30</td>
</tr>
<tr>
<td>Confusion Matrix</td>
<td>Classified as:</td>
<td>Classified as:</td>
<td>Classified as:</td>
<td>Classified as:</td>
</tr>
<tr>
<td></td>
<td>Fall</td>
<td>No Fall</td>
<td>Fall</td>
<td>No Fall</td>
</tr>
<tr>
<td>Fall</td>
<td>8</td>
<td>2</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>No Fall</td>
<td>15</td>
<td>71</td>
<td>3</td>
<td>83</td>
</tr>
</tbody>
</table>
4.4.2 Cross-validation 10 Folds option

We divided the dataset into 10 parts, and we train on the 9 parts, and test with the 10th part alone, and average the 10 results. And each part in the dataset is used once for testing and 9 times for training. Table 4-4 shows again the results for the classifiers and we note that for three of the classifiers with the 10-fold cross validation increases above 96% for correct classification and drop below 4% for incorrect classification of the same three classifiers. Regardless in all instances Table 4-4 shows improved results: higher
percentage classification and lower in incorrect classification compared to the first of the three options, the train-test shown in Table 4-3.

Figure 4-14 shows the J48 pruned tree. The figure shows that the size of the connected component of windows of number of frames (WS=8) is the key attribute in deciding on fall detection. Furthermore, the second level is the count of active sensors, and they too play a discriminant role. In contrast, Figure 4-13, the count of active sensors for a single frame does not contribute to the classification.

Table 4-4 Classifier performance of the 10-Fold cross validation

<table>
<thead>
<tr>
<th>All data set</th>
<th>Naive Bayes Classifier</th>
<th>Logistic Regression Classifier</th>
<th>Multilayer Perceptron Classifier</th>
<th>J48 Classifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fall: 110</td>
<td>1074 / 1153 93.15%</td>
<td>1112 / 1153 96.44 %</td>
<td>1109 / 1153 96.18 %</td>
<td>1108 / 1153 96.10%</td>
</tr>
<tr>
<td>No Fall: 1043</td>
<td>79 / 1153 6.85%</td>
<td>41 / 1153 3.56%</td>
<td>44 / 1153 3.82%</td>
<td>45 / 1153 3.90%</td>
</tr>
<tr>
<td>Correctly Classified Instances</td>
<td>110 / 1153 9.40%</td>
<td>96 / 1153 8.26%</td>
<td>97 / 1153 8.28%</td>
<td>98 / 1153 8.29%</td>
</tr>
<tr>
<td>Incorrectly Classified Instances</td>
<td>93 / 1043 8.92%</td>
<td>101 / 1043 9.73%</td>
<td>101 / 1043 9.74%</td>
<td>101 / 1043 9.75%</td>
</tr>
<tr>
<td>True Positive Rate</td>
<td>Yes: 0.845</td>
<td>Yes: 0.745</td>
<td>Yes: 0.809</td>
<td>Yes: 0.791</td>
</tr>
<tr>
<td>False Positive Rate</td>
<td>Yes: 0.059</td>
<td>Yes: 0.012</td>
<td>Yes: 0.022</td>
<td>Yes: 0.021</td>
</tr>
<tr>
<td>Confusion Matrix</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.4.3 Percentage Split (66%) option

The dataset is randomly split into training (66%), and testing (34%), we then run the classifiers on the training set and then re-evaluate on the testing set. Table 4-5 shows the performance of the four classifiers. And there is a slight improvement in the performance of each classifier: the correctly classified instances are slightly increased and the incorrectly classified instances are slightly decreased.

In Figure 4-15 the J48 pruned tree again shows that the size of the connected component provided the primary classification. Interestingly, for the second level of
classification attributes we use either single frame active sensors count or again the connected component size.

Table 4-5 Classifier performance of the 66% percentage split

<table>
<thead>
<tr>
<th>All data set</th>
<th>Naive Bayes Classifier</th>
<th>Logistic Regression Classifier</th>
<th>Multilayer Perceptron Classifier</th>
<th>J48 Classifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fall: 110</td>
<td>371 / 392</td>
<td>380 / 392</td>
<td>380 / 392</td>
<td>380 / 392</td>
</tr>
<tr>
<td>No Fall: 1043</td>
<td>94.64 %</td>
<td>96.94%</td>
<td>96.94%</td>
<td>96.94%</td>
</tr>
<tr>
<td>66% Train,</td>
<td>Correctly Classified</td>
<td>94.64 %</td>
<td>96.94%</td>
<td>96.94%</td>
</tr>
<tr>
<td>34%: 392 Test</td>
<td>Instances</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Fall: 38,</td>
<td>Incorrectly Classified</td>
<td>21</td>
<td>12 / 392</td>
<td>12 / 392</td>
</tr>
<tr>
<td>No Fall: 354)</td>
<td>Instances</td>
<td>5.36 %</td>
<td>3.06%</td>
<td>3.06%</td>
</tr>
<tr>
<td></td>
<td>True Positive Rate</td>
<td>Yes: 0.842</td>
<td>Yes: 0.737</td>
<td>Yes: 0.737</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No: 0.958</td>
<td>No: 0.994</td>
<td>No: 0.994</td>
</tr>
<tr>
<td></td>
<td>False Positive Rate</td>
<td>Yes: 0.042</td>
<td>Yes: 0.006</td>
<td>Yes: 0.006</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No: 0.158</td>
<td>No: 0.263</td>
<td>No: 0.263</td>
</tr>
<tr>
<td></td>
<td>Confusion Matrix</td>
<td>Classified as:</td>
<td>Classified as:</td>
<td>Classified as:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fall</td>
<td>No Fall</td>
<td>Fall</td>
</tr>
<tr>
<td>Fall: 110</td>
<td>32</td>
<td>6</td>
<td>28</td>
<td>10</td>
</tr>
<tr>
<td>No Fall: 1043</td>
<td>15</td>
<td>339</td>
<td>2</td>
<td>352</td>
</tr>
</tbody>
</table>
Figure 4-15 The J48 pruned tree (66% split)
The size of the connected component of windows of frames is the key attribute in deciding on fall detection. The count of active sensors for single frame has more effect compared to train-test options. The sum of active sensors in a window of frames is of the least importance.
5 CHAPTER 5: DISCUSSION

We have integrated and demonstrated the workability of our fast computer board, the Jetson TK1, with the smart carpet hardware data acquisition system. We showed how we integrated and used the hardware to generate sensor signals, fall detection algorithms that best use the sensor data, and their computational intelligence components, and finally the use of Weka classifiers. Importantly we note that more work is required to embed the functionalities of the Weka classifier into an integrated system that indeed makes the carpet “Smart”.

5.1 Jetson TK1

We interfaced the board with advance commuting resources to the to the existing smart carpet data acquisition system. The speed of the sensor data reading is bounded by the acquisition system hardware. We believe that this is largely a solvable hardware problem. So our focus was to enhance data processing of personnel data using complicated algorithms, the new display system benefited from the power of the hardware - by running a web and database servers to compensate for the slow hardware in previous work [3, 4, 5, 8, 31]. We also produced a notification of occurrence of events to alert caregivers by email or SMS.

We found that storing the active sensors data into the database saved space, and added a value of developing a quasi-real-time display application using the MySQL database and Apache server engines topped by web application development tools; PHP, HTML and JavaScript. We were able to achieve this using the Jetson TK1 board, an advantage over the old slow system [5, 31], and the limitation imposed by using external
display hardware to view the carpet activities. We have produced an apparently instant display of the sensor activity on the carpet compared to the previous work. The old display did provide a sense of time watching people walking on the carpet. Now we see the walk in seeming real time, but when we recall the stored data the display is so fast we see saved frames at once, losing the sense of time of activity.

We were able to view the carpet activities on the display monitor connected to the Jetson Board, and remotely using any web browser within the same network. It is also possible to view and access the data from any geographical location with Internet access with the condition that the board is connected to public network with static or public IP address. Configuring the board to be publicly accessed adds the burden of security. Our lab is working on a web portal (cloud solution) to have centralized access to the data.

5.2 Data Collection and fall detection

We collected data from 10 adult volunteers (9 males and 1 female) performing 14 fall patterns as shown in Figure 3-3. We videotaped the experiments to help verify the performance of the fall detection algorithms. We used the time and the frames over which the fall happened. We developed and tested different algorithms to detect the occurrence of a fall on the carpet: the algorithms for connected component labeling, convex hull area and heuristics were used. We measured the performance of the algorithms in detecting the fall by calculating the sensitivity (True Positive Rate-TPR), and False Negative Rates (FNR). We tweaked each algorithm by modifying parameters: connected component size,
window size of number of frames, and threshold, to better detect the fall. We used the videotaped files to cross check the accuracy of the algorithms.

We measured the performance of the fall detection algorithms by counting the number of fall patterns that were detected as falls and produced the sensitivity for each algorithm. Table 4-1 Count of detected falls for 5 algorithms. There were 14 fall patterns, a sensitivity produced for the same of the detected falls, and a false negative rate (FNR) for patterns detected as no falls where falls occurred shows the result of using the CCL-Multiple frames algorithm. At Window size (WS= 8, and TH= 8) we obtained a sensitivity of 87.9% compared to lower window sizes of number of frames. Also, it outperformed all other algorithms. The data acquisition system scans the smart carpet and sends data to the Jetson board on average every 140 milliseconds. The fall occurrence takes longer than a second that is roughly 6-8 frames time length. We found that running the algorithms on a windows size of number of frames leads to better results. Still the single frame fall detection with low threshold (TH=4) works fine with sensitivity of 80.7%, however, very sensitive to noise. At any instant there is noise in the carpet can cause the size of the connected component of single frame to reach the threshold, hence, increase false positive rates. Organizing a single frame matrix to include matrices (frames) from a window-size-of-number-of-frames reduced the storage size by the totality of windows size (WS). Rather than storing a matrix for each frame we have a matrix for each WS of frames.

5.3 Computational Intelligence and Weka framework
The intuitive results described in section 5.2 are likely not going to provide robust results. We believe there is room for improvement by using more formal computational intelligence techniques. The Jetson TK1 board helped run the concurrent algorithms, used for the data in sections 4.2 and discussion in 5.2, some of which generated dataset training and testing datasets for the classifiers. To look at different classifiers, we used the Weka framework. Here we could analyze and measure the performance of different classifiers that best fit our dataset, such as Naïve Bayes, logistic, multilayer perceptron, and J48 decision tree (C4.5 algorithm).

We used different test options to train the classifiers and then evaluate the performance by running testing data not used in the training phase. Among the different classifiers with different testing options we found that the Naïve Bayes algorithm for all testing options has the worst performance compared to other classifiers. Naïve Bayes shows higher false positives rates in all testing patterns. All other classifiers have relatively similar performance (96.94% with the 66% percentage split option). Yet, the implementation cost of each will be a decision to consider once implementing on standalone system with available processing resources.

We found that the connected component for a window size of number frames (WS) dominates all other attributes: running that attribute on the same classifiers using the percentage split option yields performance of 96.939% accuracy. This experiment proves to us that we can use computational intelligence to detect occurrence of falls and further assess fall prediction and risk analysis. We still need to develop the proper classifiers and extract or modify Weka framework to be used on the Jetson board. We
also consider adding more features or attributes like the convex hull area, and centroid calculations to achieve fall prediction and detection.

Examining this data gives us the hope to be able to classify the expected activity on the floor; pets walking, kids walking, stroller rolling and so on. Focusing on elders, primarily, adult seniors is to classify the type of walk and gait. In addition, to be able to retain these data years or decades and make them available when necessary. We want to be able to handle single or multiple adults dwelling on the carpet. We are sure that we can detect a person walking on the carpet. We hope to identify the places of risk in the apartment. We believe the smart carpet can provide an indication of the presence of activity that could be risky like falling in bathtub. The smart carpet will not tell if the person falls or not, but will detect that a person being in the bathroom for longer time.

We are to study the issues of the privacy and security of the collected data, how should keep this data for log times storage. The security concerns and that need to be met. We need to build a standalone piece of software that could support smart carpet detection and identification algorithms to improve the independence of elderly to extend the caregivers circle and provide information for clinical use.
6 CHAPTER 6: CONCLUSION

We demonstrated that our board, software and algorithms work successfully; it reads the sensor data from the data acquisition system, converts it to a format suitable for storage and running algorithms. The software performs the calculations, monitors the activity on the smart carpet and then alerts caregivers for occurrence of events. We used MySQL database and Apache server and web development tools to develop a display for stored active sensors data files on the web browser. We used the google e-mail engine free service to send alerts and notifications via SMS and e-mails.

We monitored the activity of volunteers walking then falling on the carpet, focusing on the fall detection. We analyzed the sensor data generated by the data acquisition system and developed different fall detection algorithms: connected component labeling for both single frame data and multiple frames of data, convex hull area and Heuristic algorithm based on the single frame count with spatial locality. The connected component for window size of number of frames (WS) was the best algorithm with sensitivity of 87.9%.

We used Weka framework for computational intelligence techniques. We evaluated the performance of four classifiers: Naïve Bayes, logistic function, multilayer perceptron and J48 decision tree. We used different test options to train the classifiers and then evaluate the performance by using separate testing data. We used three testing options: Train and test, 10-Fold cross validation and percentage split (66% train and 34% test). We measured the performance of each testing options and output a decision tree for the J48 algorithm. The best feature was again the size of the connected component with WS=8, with classification accuracy of 96.94%.
7 REFERENCES


[16] M. Popescu, Y. Li, M. Skubic, and M. Rantz, "An acoustic fall detector system that uses sound height information to reduce the false alarm rate," in Engineering 62


[27] Decision Tree: [http://www.d.umn.edu/~padhy005/Chapter5.html](http://www.d.umn.edu/~padhy005/Chapter5.html)


8 APPENDICES

8.1 Appendix A: Source code

/
* MyServer.java */

/ * To change this license header, choose License Headers in Project Properties. 
* To change this template file, choose Tools | Templates 
* and open the template in the editor. 
*/
package myserver;

/**
 * @author fadimuheidat
 */
import gnu.io.ComPortIdentifier;
import gnu.io.PortInUseException;
import gnu.io.SerialPort;
import gnu.io.SerialPortEvent;
import gnu.io.SerialPortEventListener;
import gnu.io.UnsupportedCommOperationException;
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.Socket;
import java.net.ServerSocket;
import java.sql.Timestamp;
import java.util.Enumeration;
import java.util.TooManyListenersException;
public class MyServer implements SerialPortEventListener {

    SerialPort sp;
    private InputStream istream;
    private static final String PORT_NAMES[] = {"/dev/ttyUSB0", "/dev/sda1", 
"COM17");
    private final String displaydata = "";

    }
private final boolean data_started = false;
private final boolean data_ended = false;
private long rawDataType;
private long matrixDataType;
private long csvDataType;
private long socket_time;
private Timestamp timestamp;
String email = null;
static int port = 15376;
static ServerSocket server;
static Socket clientSocket;
private BufferedReader br;
private PrintStream pstream;
private final boolean connected = true;
int current_level = 0;
boolean level1NotificationSent = false;
boolean level2NotificationSent = false;
boolean level3NotificationSent = false;

// Method to Identify the port number of Sheevaplug
@SuppressWarnings("rawtypes")
private void SheevaConnect() {
    CommPortIdentifier portId = null;
    Enumeration portList = CommPortIdentifier.getPortIdentifiers();

    while (portList.hasMoreElements()) {
        CommPortIdentifier currentPortId = (CommPortIdentifier) portList.nextElement();

        for (String portName : PORT_NAMES) {
            if (currentPortId.getName().equals(portName)) {
                portId = currentPortId;
                break;
            }
        }
    }

    if (portId == null) {
        System.out.println("Could not find the port.");
    }

    try {

sp = (SerialPort) portId.open("Elder Care", 2004);
sp.setSerialPortParams(19200, SerialPort.DATABITS_8, SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);
sp.addEventListener(this);
sp.notifyOnDataAvailable(true);

istream = sp.getInputStream();

} catch (PortInUseException | UnsupportedCommOperationException | TooManyListenersException | IOException e) {
    System.out.println(e);
}

public synchronized void close() {
    if (sp != null) {
        sp.removeEventListener();
        sp.close();
    }
}

// Accessing Serial Data using serialEvent method
@Override
public void serialEvent(SerialPortEvent serialEvent) {
    switch (serialEvent.getEventType()) {
    case SerialPortEvent.BI:
    case SerialPortEvent.CD:
    case SerialPortEvent.CTS:
    case SerialPortEvent.DSR:
    case SerialPortEvent.FE:
    case SerialPortEvent.OE:
    case SerialPortEvent.PE:
    case SerialPortEvent.RI:
        System.out.println("carrier not detected");
        break;
    case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
        // No data Available
        System.out.println("no data");
        break;

    case SerialPortEvent.DATAAVAILABLE:
        byte[] readBuffer = new byte[8];
        int nbytes = 0;

try {
    while (istream.available() > 0) {

        nbytes = istream.read(readBuffer);
        String carpetdata = new String(readBuffer);
        System.out.println(carpetdata);
    }
}

} catch (IOException e) {
    System.out.println(e);
}
break;
}

public static void sleep(int time) {
    try {
        Thread.sleep(time);
    } catch (Exception e) {
    }
}

@SuppressWarnings("empty-statement")
public static void main(String args[]) throws IOException, Exception {

    DataManipulate dmobj = new DataManipulate();

    String carpetdata = " ";
    String fall = null;
    String personName = null;
    String currentDirectory1 = System.getProperty("user.dir");
    File folder = new File(currentDirectory1 + "/run/");
    File[] listOfFiles = folder.listFiles();

    for (int i = 0; i < listOfFiles.length; i++) {
        FrameIdGlobal.FID=0;
        if (listOfFiles[i].isFile() && listOfFiles[i].getName().endsWith(".txt")) {
            System.out.println(listOfFiles[i].getName());
        } else if (listOfFiles[i].isDirectory()) {
            System.out.println("Directory " + listOfFiles[i].getName());
        }
        personName = "Person";
        FileInputStream fstream = new FileInputStream(listOfFiles[i]);
        // or using Scanner
        DataInputStream in = new DataInputStream(fstream);
BufferedReader br = new BufferedReader(new InputStreamReader(in));
String strLine;
//Read File Line By Line
while ((strLine = br.readLine()) != null) {
    // split string and call your function
    // System.out.println(strLine);
    String[] tokens = strLine.split(",");
    String timestamp1 = tokens[tokens.length - 1];
    //System.out.println(timestamp); 
    carpetdata = tokens[0];
    //String timestamp1 = tokens[2];
    // carpetdata = Arrays.toString(tokens).replaceAll("[\[\]\]]", "").replaceAll(",", "\n");
    // System.out.println(strLine);
    
    // double frameTime1 = System.nanoTime();
    dmobj.manipulateCarpetData(carpetdata, timestamp1, 38, fall, personName);
    // double frameTime2 = System.nanoTime();
    // double moving = (double)(frameTime2 - frameTime1)/1000000;
    // System.out.println(moving);
    //sleep(130);
}
/* DataManipulate.java*/
/*
* To change this license header, choose License Headers in Project Properties.
* To change this template file, choose Tools | Templates
* and open the template in the editor.
*/
package myserver;
/**
 * @author fadimuhediat
 */
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Timestamp;
import java.text.DateFormatSymbols;
import java.util.ArrayList;
import java.util.GregorianCalendar;
import java.util.Calendar;
import au.com.bytecode.opencsv.CSVWriter;
import java.awt.Point;
import java.util.Stack;
import java.util.logging.Level;
import java.util.logging.Logger;

public class DataManipulate {

    private final int NAIVETHRESHOLD = 10;

    private final boolean BOUNDARY = true;  //first pass fo CC needs to chekc the boundary
    private final boolean CCL = false; // Connected Component method
    private final int CCL_THRESHOLD = 7;
    int WS = 8; //on average 8 frames/second
    private final boolean ASCOUNT = false; // Active Sensors & in a row or Column Count method
    private final int ASCOUNT_THRESHOLD = 4;

    private final boolean GADRI_MOD = false;
    private final int GADRI_THRESHOLD = 4;

    private final boolean TRAIN = true;
    private final boolean HULL = false;
    private final int HULL_THRESHOLD = 7;

}
private final boolean CENTROID = false; // Moving Centroid (Velocity)

private final boolean DEBUG = false;

String timestamp_modified;
private String displaydata = "";
private boolean data_started;
private boolean data_ended = false;
private long rawDataTime;
private long matrixDataTime;
private long csvDataTime;
private Timestamp timestamp;
String email = null;

private BufferedReader br;
//private PrintStream pstream;
private final boolean connected = true;
int current_level = 0;
boolean level1NotificationSent = false;
boolean level2NotificationSent = false;
boolean level3NotificationSent = false;
private final int previous_frame = 0;
//private final float[][] oldCentroid= {0,0};
int[][] oldMatrix = new int[12][12];
Point PreviousCentroid = new Point(0, 0);
Point newC = new Point(0, 0);
ArrayList<Integer> ActiveSensorsCount = new ArrayList<>();
ArrayList<String> ActiveSensorsTimes = new ArrayList<>();
ArrayList<Integer> ActiveSensorsDiff = new ArrayList<>();
ArrayList<Integer> MovingSum = new ArrayList<>();
ArrayList<int[]>> listOfFrames = new ArrayList<>();// place holder to add more frames

//int [][] AccumulatedMatrix= new int [12][12]; //store the accumulated frames
int sensors_activated = 0;
int FrameId = 0;
int sum1 = 0, sum = 0;

public DataManipulate() {
   //this.data_started = false;
}

public void manipulateCarpetData(String carpetdata, String timestamp1, int nbytes, String fall, String personName) throws IOException, InterruptedException {
   timestamp_modified = timestamp1.split(" ")[1];
   //System.out.println(timestamp_modified);
}
for (int n = 0; n < nbytes; n++) {
    char currentdata = (char) carpetdata.charAt(n); // cast to char
    if (!data_started) { // Here data_started boolean value= false and
        data_started = false;
        if (displaydata.endsWith("SA")) {
            data_started = true;
            displaydata = "SA";
        } else {
            displaydata = displaydata + currentdata;
        }
    }
    if (data_started) {
        displaydata = displaydata + currentdata;
        if (data_started && !data_ended && displaydata.endsWith("E")) {
            data_started = false;
            data_ended = true;
            // System.out.println("data -->"+ displaydata+" "+displaydata.length());
        }
    }
}
if (data_started) {
    displaydata = displaydata + currentdata;
    if (data_started && !data_ended && displaydata.endsWith("E")) {
        data_started = false;
        data_ended = true;
        // System.out.println("data -->"+ displaydata+" "+displaydata.length());
    }
}
Calendar cal = Calendar.getInstance();
long timeinms = cal.getTimeInMillis();
rawDataTime = System.nanoTime(); // System.nanoTime() gives you a
nanosecond-precise time, relative to some arbitrary point.
timestamp = new Timestamp(timeinms);

// storeCarpetData(displaydata, timestamp.toString(), fall, personName);//
store Raw frames
//storeCarpetData(displaydata, timestamp_modified, fall, personName);//
store Raw frames
    // Analyze the data
    processCarpetData(displaydata, fall, personName);
    // pstream.println(displaydata+"","+timestamp);
    // pstream.flush();
    displaydata = "";
} else /* System.out.println("string not matching
"+Display_data); data_started=false;
data_ended=true;
Display_data="";*/ if (displaydata.length() > 38) {
    System.out.println("String length greater than 38 " + displaydata.length());
}
private void storeCarpetData(String rawdata, String timestamp1, String fall, String personName) {

    //System.out.println(timestamp1);
    BufferedWriter writer = null;
    if (!rawdata.equals("SABCDE")) {

        // This signifies an empty frame of raw data which is the literal output of the board with out any sensors activated!
        try {
            writer = new BufferedWriter(new FileWriter(getTimeAMPM("_raw", ".txt", fall, personName), true));
            writer.write(rawdata + "," + timestamp1);
            writer.newLine();
        } catch (FileNotFoundException e) {
            System.out.println(e);
        } catch (IOException e) {
            System.out.println(e);
        } finally {
            try {
                writer.close();
            } catch (IOException e) {
                System.out.println(e);
            }
        }
    }
}

private File getTimeAMPM(String FileName, String Extension, String fall, String personName) {
    Calendar cal = new GregorianCalendar();
    int month = cal.get(Calendar.MONTH);
    String monthName = DateFormatSymbols.getInstance().getMonths()[month];
    int year = cal.get(Calendar.YEAR);
    int date = cal.get(Calendar.DATE);
    int time = cal.get(Calendar.HOUR_OF_DAY);
    String currentDirectory = System.getProperty("user.dir");
    // String fileName = "/Users/fadimuheidat/Documents/java-work/MyServer/" + fall + "+personName+"_"+monthName + "_" + date + "_" + year + "_" + time + "00" + FileName + Extension;
    //String fileName = currentDirectory +"/"+ personName+"_"+fall+"_"+monthName + "_" + date + "_" + year + "_" + time + "00" + FileName + Extension;

}
String fileName = currentDirectory + "/" + personName + "_" + fall + "_" +
monthName + "_" + date + "_" + year + "_" + time + "00" + FileName + Extension;

return new File(fileName);
}

private void processCarpetData(String rawdata, String fall, String personName)
throws InterruptedException {

FrameId++;
FrameIdGlobal.FID++;

int count = 0;
int Aindex = 1;
int Bindex = 10;
int Cindex = 19;
int Dindex = 28;

int[] finalData = new int[128]; // (total 144 actual 128 ) holds the data from the
frame received on serial
if (!rawdata.equals("SABCDE")) {
    int n = 0; // number of bytes in the frame
    for (int k = 1; k < rawdata.length(); k++) {
        if (((rawdata.charAt(k) == 'A') && (k == 1)) || ((rawdata.charAt(k) == 'B')
&& (k == 10)) || ((rawdata.charAt(k) == 'C') && (k == 19)) || ((rawdata.charAt(k) == 'D')
&& (k == 28))) {
            continue;
        }
        if (rawdata.charAt(k) == 'E') {
            k = 128;
            // System.out.println("k: "+k);
            break;
        }
    }
    // Convert to Binary
    String str1, str2;
    int j = Integer.parseInt(Character.toString(rawdata.charAt(k)), 16);
    str2 = String.format("%04d", Integer.parseInt(Integer.toBinaryString(j)));

    // build finaldata array
    for (int i = 0; i < 4; i++) {
        if (n * 4 + i > 127) {
            break;
        }
        switch (str2.charAt(i)) {
        
        }
    }

    // Convert to Binary
    String str1, str2;
    int j = Integer.parseInt(Character.toString(rawdata.charAt(k)), 16);
    str2 = String.format("%04d", Integer.parseInt(Integer.toBinaryString(j)));

    // build finaldata array
    for (int i = 0; i < 4; i++) {
        if (n * 4 + i > 127) {
            break;
        }
        switch (str2.charAt(i)) {
        
        }
    }
}
case '1':
    finalData[n * 4 + i] = 1;
    count++;
    break;

case '0':
    finalData[n * 4 + i] = 0;
    break;
default:
    break;
}
}
//System.out.println("n:"+n);
    n++;
}
} else {
    System.out.print("empty Frame");
}

// Insert to DB
databaseReady(finalData);

if (ASCOUNT) {
    int[][] matrix1 = ChangeToMatrix(finalData);
    //int [][]AccumulatedMatrixWindow= addtoWindow(finalData);
    //Save Matrix to file
    storeMatrixData(matrix1, fall, personName);
    measureCount(matrix1);
    //measureCount(AccumulatedMatrixWindow);
}

if (TRAIN) {
    int maxGroup = 0;
    double hullArea=0;

    int[][] matrix1 = ChangeToMatrix(finalData);
    // save to file
    //storeMatrixData(matrix1, "Single_Frame", "Matrix");

    int[][] weightedMatrix = computeMatrixData(matrix1);
    //storeMatrixData(weightedMatrix, "Single_Frame", "Weighted");
    // maxGroup=GCC(weightedMatrix);

    if (GCC2(weightedMatrix)[1] >= GCC2(weightedMatrix)[2]) {

maxGroup = GCC2(weightedMatrix)[1];
} else {
    maxGroup = GCC2(weightedMatrix)[2];
}
// System.out.println("MaxGroup:"+maxGroup);

int sum2 = addToBuffer(count);

int[][] AccumulatedMatrixWindow = addtoWindow(finalData);

// save to file
// storeMatrixData(AccumulatedMatrixWindow, "WS3", "Accumulated");

int[][] computedMatrix = computeMatrixData(AccumulatedMatrixWindow);
//storeMatrixData(computedMatrix, "WS3", "Weighted");
if (sum2 > 0) {
    ConnectedComponent ccobj = new ConnectedComponent();
    int CC[] = ccobj.connectedComponet(computedMatrix);
    // Check the group size of the connected component
    if (CC[1] >= CC[2]) {
        sensors_activated = CC[1];
    } else {
        sensors_activated = CC[2];
    }
} else {
    sensors_activated = 0;
}

ConvexHull chobj = new ConvexHull();
// ArrayList<Point> p = chobj.getList(matrix);
ArrayList<Point> p = chobj.getList(AccumulatedMatrixWindow);
ArrayList<Point> hullpoints = chobj.quickHull(p);
if (hullpoints.size() >= 3) {

    // System.out.println("ConvexHull Points: ");
    hullpoints.stream().forEach((p1) -> {
        // System.out.println( p1.x + ", " + p1.y + ", ");
        // System.out.println(10 * p1.y + ", " + 10 * p1.x);
    });

    hullArea = chobj.HullArea((ArrayList<Point>) hullpoints);
    if (hullArea >= HULL_THRESHOLD) {

}
System.out.println(hullpoints.size() + " ** " + hullArea + " ---> Fall
Detected by  HULL @ " + timestamp_mod);)
} else
hullArea=0;

System.out.println((count) + "," + maxGroup + "," + sum2 + ",," + 
sensors_activated + "," + 
hullArea+"," + " ---->" + timestamp_mod);
//if(FrameId%WS==0){
/* System.out.println("FId: "+(FrameId)+" SF_count: "+(count)+ " CCL_SF:
"+ maxGroup + 
" WS3_Count:"+sum2+ " CCI_"+WS+"_F: "+sensors_activated
+" --"+"timestamp_mod);/*

 // }else{
 // System.out.println("Frame Number: "+(FrameId)+" count: "+(count)+ " CCI_Single_Frame: "+ maxGroup );
 // }
if (sensors_activated > CCL_THRESHOLD) {
 System.out.println("n "+ sensors_activated + " ---> Fall Detected using
CCL+WS @ " + timestamp_mod + "n");
sensors_activated = 0;
}

if (maxGroup >= GADRI_THRESHOLD) {
 System.out.println("n "+ maxGroup + " ---> Fall Detected using
Single Frame @ " + timestamp_mod + "n");
maxGroup = 0;
}

if (FrameIdGlobal.FID % WS == 0) {
 System.out.println("-------------------------------------------------------------------
----------------");
}

}

if (GADRI_MOD) {

int maxGroup = 0;
int[][] matrix1 = ChangeToMatrix(finalData);
// save to file
storeMatrixData(matrix1, fall, "Matrix");
int[][] weightedMatrix = computeMatrixData(matrix1);
storeMatrixData(weightedMatrix, fall, "Weighted");
// maxGroup=GCC(weightedMatrix);

if (GCC2(weightedMatrix)[1] >= GCC2(weightedMatrix)[2]) {
    maxGroup = GCC2(weightedMatrix)[1];
} else {
    maxGroup = GCC2(weightedMatrix)[2];
}
// System.out.println("MaxGroup:"+maxGroup);

if (maxGroup >= GADRI_THRESHOLD) {
    System.out.println(maxGroup + " --- Fall Detected using GADRI's @ " +
timestamp_modified);
    maxGroup = 0;
}

// CCL with Windows size (WS) and Speed =1 (default)
if (CCL) {

    // build accumulated matrix based on Windows Size (WS)
    int[][] AccumulatedMatrixWindow = addtoWindow(finalData);

    // save to file
    // storeMatrixData(AccumulatedMatrixWindow,fall, "Accumulated");
    // addToBuffer(count);
    // long frameTime1 = System.nanoTime();
    int sum2 = addToBuffer(count);
    // if(sum>=0)
    //   MovingSum.add(sum2);
    // long frameTime2 = System.nanoTime();
    //long moving = (frameTime2 - frameTime1) / 1000;
    //System.out.println(moving);
    // System.out.println("Accumluated sum:"+MovingSum);
    // if(count>0)
    // System.out.println("count:"+(count));
    // System.out.println("Frame Number:"+(FrameId));

    //ActiveSensorsCount.add(count);
    // if (ActiveSensorsCount.size()>=WS)
    // detectFall1(ActiveSensorsCount);
    //ActiveSensorsCount.remove(WS);
    //ActiveSensorsDiff.add(count);
// ActiveSensorsDiff.add(ActiveSensorsCount.get(FrameId-2)-
   ActiveSensorsCount.get(FrameId-1));
//ActiveSensorsTimes.add(timestamp_modified.substring(11));
// System.out.println(ActiveSensorsCount.get(FrameId-1));
//System.out.println("Active Sensors:"+ActiveSensorsCount);
//System.out.println(ActiveSensorsTimes);
// System.out.println("Difference:  "+ActiveSensorsDiff);
// Take 1D array of all sensors and converto Matrix that represents the Carpet

// Save Matrix to file
// storeMatrixData(AccumulatedMatrixWindow,fall, personName);
//count number of time the sensor being activated
//sumMatrixData(AccumulatedMatrixWindow, fall, personName);
// First Pass Connected Component , Weighted Matrix , weight of 2's or 1's
// System.out.println("-- First Pass Matrix of 2s or 1s --");
int[][] computedMatrix = computeMatrixData(AccumulatedMatrixWindow);

//Store this Matrix for debug
//storeMatrixData(computedMatrix,fall, "Weighted");
// storeMatrixData(AccumulatedMatrix,fall, "Accumulated");
//CCL= false;
// Call Connected Component
// System.out.println("--Connected Component--");
ConnectedComponent cobj = new ConnectedComponent();
int CC[] = cobj.connectedComponet(computedMatrix);
// Check the group size of the connected component
if (CC[1] >= CC[2]) {
    sensors_activated = CC[1];
} else {
    sensors_activated = CC[2];
}

if (FrameId % WS == 0) {
    System.out.println("CC: " + sensors_activated + " @ Frame: " + FrameId);
}
//System.out.println(FrameId+ ", "+ sensors_activated + ", "+sum2);
//sensors_activated = 0;
if (sensors_activated >= CCL_THRESHOLD) {
    System.out.println("CC: " + sensors_activated + " Fall Detected @ Frame: " + FrameId);+" @ "+timestamp_modified);
    sensors_activated = 0;
}
}
if (HULL) {

    // System.out.println("--Convex Hull--");
    // Call convexHull
    // int [][] matrix = ChangeToMatrix(finalData); // per frame HULL
    int [][] AccumulatedMatrixWindow = addtoWindow(finalData);
    storeMatrixData(AccumulatedMatrixWindow, fall, "Accumulated");
    sumMatrixData(AccumulatedMatrixWindow, fall, personName);

    ConvexHull chobj = new ConvexHull();
    // ArrayList<Point> p = chobj.getList(matrix);
    ArrayList<Point> p = chobj.getList(AccumulatedMatrixWindow);
    ArrayList<Point> hullpoints = chobj.quickHull(p);
    if (hullpoints.size() >= 3) {

        System.out.println("Convex Hull Points: ");
        hullpoints.stream().forEach((p1) -> {
            // System.out.println( p1.x + ", " + p1.y + "," );
            System.out.println(10 * p1.y + ", " + 10 * p1.x);
        });

        double p2 = chobj.HullArea((ArrayList<Point>) hullpoints);
        if (p2 >= HULL_THRESHOLD) {
            System.out.println(hullpoints.size() + " ** " + p2 + " ---> Fall Detected by HULL @ " + timestamp_modified);
        }
    }
}

if (CENTROID) {

    // Calling Centroid
    int [][] matrix = ChangeToMatrix(finalData); // per frame CENTROID
    // int [][] AccumulatedMatrixWindow= addtoWindow(finalData);
    Centroid cobj = new Centroid();
    Point currentCentroid = cobj.GetCentroid(matrix);
    if (!(currentCentroid.getX() == 0 && currentCentroid.getY() == 0)) {
        // System.out.println("--Centroid--");
        System.out.println("FrameId:" + FrameId + "*" + "Active Sensors:" + count + " Centroid: (" + currentCentroid.getX() + "+", " + currentCentroid.getY() + "+") @: " + timestamp_modified);
    }

    // newC = new Point(Math.abs((int)(currentCentroid.getX()-
    PreviousCentroid.getX())), Math.abs((int)(currentCentroid.getY()-
    PreviousCentroid.getY())));
}
//PreviousCentroid = currentCentroid;
//System.out.println("New Location :"+newC.getX()+","+newC.getY());

    }
    }

public int rtCircShift(int bits, int k) {
    return (bits >>> k) | (bits << (Integer.SIZE - k));
}

private void sumMatrixData(int[][] matrix, String fall, String personName) {
    for (int r = 0; r < matrix.length; r++) {
        for (int c = 0; c < matrix.length; c++) {
            oldMatrix[r][c] = oldMatrix[r][c] + matrix[r][c];
        }
        //storeMatrixDataSum(oldMatrix, fall, personName);
    }

    // To DO : modify the DB query to include the coordinates
    private void databaseReady(int[] finalData) throws InterruptedException {
        MySQLAccess dao = new MySQLAccess();
        StoreFirebase fb = new StoreFirebase();

        int[][] datamatrix = new int[12][12];
        int i = 0;

        int k = 0, m = 0; // counter for number of bytes
        // System.out.println("--- Segment A ---");
        for (int c = 0; c < 12; c++) {
            for (int r = 0; r < 8; r++) {
                if ((r == 5 && c == 10) || (r == 0 && c == 11)) {
                    datamatrix[r][c] = 0;
                    i++;
                } else {
                    datamatrix[r][c] = finalData[i++];
                }
            }
            if (datamatrix[r][c] == 1) {
                System.out.println(datamatrix[r][c] + "(" + r + "," + c + ") -->
                                          " + (i - 1));
                // fb.save(i-1);
                // Thread.sleep(200);
                m++;
            }
        }
    }
try {
    dao.insertdata(Integer.toString(r), Integer.toString(c), Integer.toString((i - 1)), Integer.toString(0), Integer.toString(0));
    //Thread.sleep(100);
} catch (Exception ex) {
    Logger.getLogger(DataManipulate.class.getName()).log(Level.SEVERE, null, ex);
}

//System.out.println("--- Segment D ---");
for (int r = 11; r > 7; r--) {
    for (int c = 0; c < 8; c++) {
        datamatrix[r][c] = finalData[i++];
        if (datamatrix[r][c] == 1) {
            System.out.println(datamatrix[r][c] + "(" + r + "," + c + ") --> " + (i - 1));
            m++;
            // fb.save((i-1));
            // Thread.sleep(200);
            try {
                dao.insertdata(Integer.toString(r), Integer.toString(c), Integer.toString((i - 1)), Integer.toString(0), Integer.toString(0));
                //Thread.sleep(100);
            } catch (Exception ex) {
                Logger.getLogger(DataManipulate.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }
}
// System.out.println("m="+m);

/* accept array of data and convert into Matrix 12X12 with empty portion with no sensors*/
private int[][] ChangeToMatrix(int[] finalData) {
    int[][] datamatrix = new int[12][12];
    int i = 0;
for (int c = 0; c < datamatrix.length; c++) {
    for (int r = 0; r < 8; r++) {
        datamatrix[r][c] = finalData[i++];
        //sumMatrix[r][c] = sumMatrix[r][c] + datamatrix[r][c];
        // if (datamatrix[r][c] == 1)
        // AccumulatedMatrix[r][c] = 1;
        // System.out.println(datamatrix[r][c] + "(" + r + "," + c + ")");
    }
}

for (int r = 0; r < 8; r++) {
    for (int c = 0; c < 8; c++) {
        datamatrix[r][c] = finalData[i++];
        //sumMatrix[r][c] = sumMatrix[r][c] + datamatrix[r][c];
        //if (datamatrix[r][c] == 1)
        // AccumulatedMatrix[r][c] = 1;
        // System.out.println(datamatrix[r][c] + "(" + r + "," + c + ")");
    }
}

return datamatrix;

private void storeMatrixData(int[][] matrix, String fall, String personName) {
    //System.out.println(timestamp.toString());
    BufferedWriter bw = null;
    try {
        bw = new BufferedWriter(new FileWriter(getTimeAMPM("_matrix", ".txt", fall, personName), true));
        bw.write("--Matrix Data--");
        bw.write(timestamp_modified);
        bw.newLine();
        for (int r = 0; r < matrix.length; r++) {
            String row = "";
            for (int c = 0; c < matrix[r].length; c++) {
                row = row + " " + matrix[r][c];
            }
            bw.write(row);
            bw.newLine();
        }
    } catch (FileNotFoundException e) {
        System.out.println(e);
    } catch (IOException e) {
        System.out.println(e);
    } finally {
}
try {
    bw.close();
} catch (IOException e) {
    System.out.println(e);
}
}

private void storeMatrixDataSum(int[][] matrix, String fall, String personName) {
    BufferedWriter bw = null;
    try {
        bw = new BufferedWriter(new FileWriter(getTimeAMPM("_matrixSum", ".txt", fall, personName), true));
        bw.write("-- Matrix Data--");
        bw.write(timestamp_modified);
        bw.newLine();
        for (int r = 0; r < matrix.length; r++) {
            String row = "";
            for (int c = 0; c < matrix[r].length; c++) {
                row = row + "t" + matrix[r][c];
            }
            bw.write(row);
            bw.newLine();
        }
    } catch (FileNotFoundException e) {
        System.out.println(e);
    } catch (IOException e) {
        System.out.println(e);
    } finally {
        try {
            bw.close();
        } catch (IOException e) {
            System.out.println(e);
        }
    }
}

//@SuppressWarnings("rawtypes")
@SuppressWarnings("unchecked")
private void convertToCSV(int[][] matrix, File file) {
    @SuppressWarnings("rawtypes")
    ArrayList valueCoordinates = new ArrayList();

    //@SuppressWarnings("rawtypes")
    @SuppressWarnings("unchecked")
    private void convertToCSV(int[][][] matrix, File file) {
        @SuppressWarnings("rawtypes")
        ArrayList valueCoordinates = new ArrayList();

    } //@SuppressWarnings("rawtypes")

    public static void main(String[] args) {
        // Code for main method
    }
}
for (int r = 0; r < matrix.length; r++) {
    for (int c = 0; c < matrix[r].length; c++) {
        if (matrix[r][c] == 1) {
            valueCoordinates.add(matrix[r][c] + "(" + r + "," + c + ")");
        }
    }
}
//System.out.println(valueCoordinates);
valueCoordinates.trimToSize();

Object[] finalData = valueCoordinates.toArray();

writeIntoCSV(finalData, matrix, file);
}

private void writeIntoCSV(Object[] finalData, int[][] matrix, File file) {
    String[] finalString = new String[finalData.length + 1];
    CSVWriter csvwriter = null;
    PrintWriter pw = null;
    synchronized (pw) {
        try {
            pw = new PrintWriter(new FileWriter(file, true));
            csvwriter = new CSVWriter(pw);
        } catch (IOException e) {
            System.out.println(e);
        }
        for (int i = 0; i < finalData.length; i++) {
            finalString[i] = "" + finalData[i];
        }
        finalString[finalData.length] = timestamp_modified;
        if (finalString != null && finalString.length > 1) {
            csvwriter.writeNext(finalString);
            try {
                csvwriter.close();
            } catch (IOException e) {
                System.out.println(e);
            }
        }
    }
    // first pass for connected component
    int[][] computeMatrixDataold(int[][] matrix) {
int[][] changed_matrix = new int[12][12];
System.arraycopy(matrix[0], 0, changed_matrix[0], 0, matrix.length);// copies one line /row

// redo this code
//System.out.print(Arrays.toString(matrix[0]));
for (int c = 0; c < matrix.length; c++) {
    changed_matrix[c][0] = matrix[c][0];
    //System.out.print((matrix[c][0])+"\t");
    //System.out.print(("c"+matrix[c][1])+"\t");
}
for (int r = 1; r < matrix.length - 1; r++) {
    for (int c = 1; c < matrix.length - 1; c++) {
        //System.out.print(matrix[r][c]);
        if (matrix[r][c] == 1) {
            //System.out.println(matrix[r][c]);
            changed_matrix[r][c] = getMax(matrix[r][c - 1], matrix[r][c + 1], matrix[r - 1][c], matrix[r + 1][c], matrix[r - 1][c - 1], matrix[r + 1][c + 1], matrix[r + 1][c + 1]) + 1;
            //System.out.println(matrix[r][c]+ " r:"+r+" c:"+c);
        } else {
            changed_matrix[r][c] = 0;
        }
    }
}
return changed_matrix;

// returns weighted matrix
int[][] computeMatrixData(int[][] matrix) {
    int[][] changed_matrix = new int[12][12];
    for (int r = 0; r < matrix.length; r++) {
        changed_matrix[0][r] = matrix[0][r];
    }
    for (int c = 0; c < matrix.length; c++) {
        changed_matrix[c][0] = matrix[c][0];
    }
    if (BOUNDARY) {
        for (int r = 0; r < 1; r++) {
            for (int c = 0; c < matrix.length - 1; c++) {
                if (matrix[r][c] == 1) {
                    if (r == 0 && c == 0) {
                        changed_matrix[r][c] = getMax1(matrix[r][c + 1], matrix[r][c + 1],
                                                         matrix[r + 1][c], matrix[r + 1][c + 1], matrix[r + 1][c + 1]) + 1;
                    }
                }
            }
        }
    }
    return changed_matrix;
}
```java
for (int c = 0; c < 1; c++) {
    for (int r = 0; r < matrix.length - 1; r++) {
        if (matrix[r][c] == 1) {
            changed_matrix[r][c] = getMax1(matrix[r][c - 1], matrix[r][c + 1],
                                           matrix[r + 1][c], matrix[r + 1][c + 1], matrix[r + 1][c - 1]) + 1;
        }
    }
}
// start from the second raw and column since the first one will be part of the neighbors list
for (int r = 1; r < matrix.length - 1; r++) {
    for (int c = 1; c < matrix.length - 1; c++) {
        if (matrix[r][c] == 1) {
            changed_matrix[r][c] = getMax(matrix[r][c - 1], matrix[r][c + 1], matrix[r - 1][c], matrix[r + 1][c], matrix[r - 1][c - 1], matrix[r + 1][c + 1], matrix[r - 1][c + 1],
                                           matrix[r + 1][c - 1]) + 1;
            // System.out.println(matrix[r][c]+ " r:"+r+" c:"+c);
        } else {//matrix[r+1][c-1] replaced matrix[r+1][c+1])
            changed_matrix[r][c] = 0;
        }
    }
}
return changed_matrix;

private int getMax(int a, int b, int c, int d, int e, int f, int g, int h) {
    int max = a;
    if (max < b) {
        max = b;
    }
    if (max < c) {
        max = c;
    }
if (max < d) {
    max = d;
}
if (max < e) {
    max = e;
}
if (max < f) {
    max = f;
}
if (max < g) {
    max = g;
}
if (max < h) {
    max = h;
}
return max;

private int getMinimum(int i, int j, int k) {
    int min = i;
    if (min > j) {
        min = j;
    }
    if (min > k) {
        min = k;
    }
    return min;
}

private int getLevel(int[] finalData) {
    // TODO Auto-generated method stub
    // compute level
    // 1: Entrance, 2: Exit, 3: others
    int level = 0;
    for (int k = 0; k < finalData.length; k++) {
        if (finalData[k] == 1) {
            System.out.println(k + " -" + finalData[k]);
            if (k == 0 || k == 1 || k == 35) {
                System.out.println("Entrance");
                level = 1;
            }
            if (k == 3 || k == 7) {
                System.out.println("RestRoom");
                level = 3;
            }
        }
    }
}
if (k == 32 || k == 33) {
    //System.out.println("Kitchen");
    level = 4;
} 
if (k == 15 || k == 14 || k == 19 || k == 18) {
    //System.out.println("Bedroom");
    level = 5;
} 
if ((k >= 40 && k <= 42) || (k >= 44 && k <= 46) || (k >= 48 && k <= 50) || 
(k >= 52 && k <= 54)) {
    // System.out.println("Living Room");
    level = 6;
} 
if (k == 28 || k == 29 || k == 63 || k == 62) {
    // System.out.println("Exit");
    level = 2;
}
}
return level;

// This method will give more alerts for Falls rather than single Fall
private void measureCount(int[][] matrix) {
    int sum = 0;
    boolean flag = false;

    for (int r = 0; r < matrix.length; r++) {
        for (int c = 0; c < matrix[r].length; c++) {
            if (matrix[r][c] == 1) {
                if (c < matrix[r].length - 3) {
                    if (matrix[r][c + 1] == 1 && matrix[r][c + 2] == 1) {
                        flag = true;
                    }
                }
                sum = sum + matrix[r][c];
                // System.out.println("sum in r: "+sum + " r:"+r+" c:"+c);
            }
        }
    }
    for (int c = 0; c < matrix[0].length; c++) {
        for (int r = 0; r < matrix.length; r++) {
            if (matrix[r][c] == 1) {
                if (r < matrix.length - 3) {

            }
if (matrix[r + 1][c] == 1 && matrix[r + 2][c] == 1) {
    flag = true;
}

//System.out.println(sum + "-"+flag);
if (sum >= ASCOUNT_THRESHOLD && flag) {
    System.out.println(sum + " --> Fall Detected using COUNT @ " +
    timestamp_modified);

    sum = 0;
    flag = false;
}

private void detectFall1(ArrayList<Integer> ActiveSensorsCount) {
    System.out.println("Active Sensors:" + ActiveSensorsCount);
}

private int getMax1(int a, int b, int c, int d, int e) {
    int max = a;
    if (max < b) {
        max = b;
    }
    if (max < c) {
        max = c;
    }
    if (max < d) {
        max = d;
    }
    if (max < e) {
        max = e;
    }
    return max;
}

private int addToBuffer(int FrameId) {
    //Speed of Window is 1 Frame/unit of time Speed
    // int Speed =1; // The windows moves 2 Frames /unit time
    ActiveSensorsCount.add(FrameId);
int listSize = ActiveSensorsCount.size();
//System.out.println("size:"+listSize+"Before:"+ActiveSensorsCount);
if (listSize >= (WS)) {
    // ActiveSensorsCount.remove(listSize-WS-1);
    sum = ActiveSensorsCount.stream().mapToInt(Integer::intValue).sum();
    // System.out.println(sum);
    //System.out.println("After:"+ActiveSensorsCount+
    "size:"+ActiveSensorsCount.size());
    ActiveSensorsCount.subList(0, Speed).clear();
} else {
    sum = -1;
}
//sum1=sum;
return sum;

@SuppressWarnings("empty-statement")
private int[][] addtoWindow(int[] finalData) {
    int[][] AccumulatedMatrix = new int[12][12]; //store the accumulated frames
    //Speed of Window is 1 Frame/unit of time SPeed
    //int Speed =1; // The windows moves 2 Frames /unit time
    listOfFrames.add(finalData);
    if (DEBUG) {
        for (int i = 0; i < listOfFrames.size(); i++) {
            for (int j = 0; j < listOfFrames.get(i).length; j++) {
                System.out.print(listOfFrames.get(i)[j]);
            }
            System.out.println();
        }
    }
    int listSize = listOfFrames.size();
    // System.out.println("size:"+listSize);
    if (listSize >= (WS)) {
        for (int i = 0; i < listOfFrames.size(); i++) {
            int k = 0;
            for (int c = 0; c < 12; c++) {
                for (int r = 0; r < 8; r++) {
                    //System.out.print(listOfFrames.get(i)[k++]);
                    if (listOfFrames.get(i)[k++] == 1) {
                    }
                }
            }
        }
    }
}
AccumulatedMatrix[r][c] = 1;
//System.out.println(AccumulatedMatrix[r][c] + "(" + r + "," + c + ")");
}
}

for (int r = 11; r > 7; r--) {
  for (int c = 0; c < 8; c++) {
    // System.out.print(listOfFrames.get(i)[k++]);
    if (listOfFrames.get(i)[k++] == 1) {
      AccumulatedMatrix[r][c] = 1;
      // System.out.println(AccumulatedMatrix[r][c] + "(" + r + "," + c + ")");
    }
  }
}
// System.out.println();

listOfFrames.subList(0, Speed).clear();
return AccumulatedMatrix;

public int GCC(int[][] matrix) {
  int nrow = matrix.length;
  int ncol = matrix[0].length;
  int lab = 1;
  int[] pos;
  Stack<int[]> stack = new Stack<int[]>();
  int[][] label = new int[nrow][ncol];
  int groupSize = 0, maxGroup = 0;
  for (int r = 1; r < nrow - 1; r++) {
    for (int c = 1; c < ncol - 1; c++) {
      if (matrix[r][c] <= 1) {
        continue;
      }
      if (label[r][c] > 0) {
        continue;
      }
      stack.push(new int[]{r, c});
      label[r][c] = lab;
      if (groupSize > maxGroup) {
maxGroup = groupSize;
}
groupSize = 0;
//System.out.println("New group");
while (!stack.isEmpty()) {
    pos = (int[]) stack.pop();
    int i = pos[0];
    int j = pos[1];
    //System.out.println("i " + i + " j " + j + " groupsize:" + groupSize + " maxGroup:" + maxGroup);
    groupSize++;
    if (matrix[i - 1][j - 1] > 1 && label[i - 1][j - 1] == 0) {
        stack.push(new int[]{i - 1, j - 1});
        label[i - 1][j - 1] = lab;
    }
    if (matrix[i - 1][j] > 1 && label[i - 1][j] == 0) {
        stack.push(new int[]{i - 1, j});
        label[i - 1][j] = lab;
    }
    if (matrix[i - 1][j + 1] > 1 && label[i - 1][j + 1] == 0) {
        stack.push(new int[]{i - 1, j + 1});
        label[i - 1][j + 1] = lab;
    }
    if (matrix[i][j - 1] > 1 && label[i][j - 1] == 0) {
        stack.push(new int[]{i, j - 1});
        label[i][j - 1] = lab;
    }
    if (matrix[i][j + 1] > 1 && label[i][j + 1] == 0) {
        stack.push(new int[]{i, j + 1});
        label[i][j + 1] = lab;
    }
    if (matrix[i + 1][j - 1] > 1 && label[i + 1][j - 1] == 0) {
        stack.push(new int[]{i + 1, j - 1});
        label[i + 1][j - 1] = lab;
    }
    if (matrix[i + 1][j] > 1 && label[i + 1][j] == 0) {
        stack.push(new int[]{i + 1, j});
        label[i + 1][j] = lab;
    }
    if (matrix[i + 1][j + 1] > 1 && label[i + 1][j + 1] == 0) {
        stack.push(new int[]{i + 1, j + 1});
        label[i + 1][j + 1] = lab;
    }
}
lab++;
int GCC2(int[][] matrix) {
    // TODO Auto-generated method stub
    // problems are teh boundary pixels/sensors===> Fixed
    int outputArray[] = new int[3];
    int nrow = matrix.length;
    int ncol = matrix[0].length;
    int lab = 1;
    int[] pos;
    Stack<int[]> stack = new Stack<>();
    int[][] label = new int[nrow][ncol];
    int groupSize = 0, maxGroup = 0;
    for (int r = 1; r < nrow - 1; r++) {
        for (int c = 1; c < ncol - 1; c++) {
            if (matrix[r][c] <= 1) {
                continue;
            }
            if (label[r][c] > 0) {
                continue;
            }
            /* encountered unlabeled foreground pixel at position r, c */
            /* push the position on the stack and assign label */
            stack.push(new int[]{r, c});
            label[r][c] = lab;
            if (groupSize > maxGroup) {
                maxGroup = groupSize;
            }
            groupSize = 0;
        }
    }
    // System.out.println("New group");
    while (!stack.isEmpty()) {
pos = stack.pop();
int i = pos[0];
int j = pos[1];
groupSize++;

//System.out.println("i "+i+" j "+j+" lab="+lab+" GroupSize:"+groupSize+" maxGroup:"+maxGroup);

//To DO

/* Add logic if the 2s are boundary*/
//pivot found , check neighbors
if (i == 0 && j <= 11) {
    if (i == 0 && j == 0) {
        if (matrix[i][j + 1] > 1 && label[i][j + 1] == 0) {
            stack.push(new int[]{i, j + 1});
            label[i][j + 1] = lab;
        }
    } else {
        if (matrix[i][j - 1] > 1 && label[i][j - 1] == 0) {
            stack.push(new int[]{i, j - 1});
            label[i][j - 1] = lab;
        } else {
            if (matrix[i + 1][j] > 1 && label[i + 1][j] == 0) {
                stack.push(new int[]{i + 1, j});
                label[i + 1][j] = lab;
            } else {
                if (matrix[i + 1][j - 1] > 1 && label[i + 1][j - 1] == 0) {
                    stack.push(new int[]{i + 1, j - 1});
                    label[i + 1][j - 1] = lab;
                } else {
                    if (matrix[i + 1][j + 1] > 1 && label[i + 1][j + 1] == 0) {
                        stack.push(new int[]{i + 1, j + 1});
                    }
                }
            }
        }
    }
} else {
    if (matrix[i][j + 1] > 1 && label[i][j + 1] == 0) {
        stack.push(new int[]{i, j + 1});
        label[i][j + 1] = lab;
    } else {
        if (matrix[i + 1][j] > 1 && label[i + 1][j] == 0) {
            stack.push(new int[]{i + 1, j});
            label[i + 1][j] = lab;
        } else {
            if (matrix[i + 1][j - 1] > 1 && label[i + 1][j - 1] == 0) {
                stack.push(new int[]{i + 1, j - 1});
                label[i + 1][j - 1] = lab;
            } else {
                if (matrix[i + 1][j + 1] > 1 && label[i + 1][j + 1] == 0) {
                    stack.push(new int[]{i + 1, j + 1});
                }
            }
        }
    }
}
\[
\text{label}[i + 1][j + 1] = \text{lab};
\]

\[
\}
\]

} else if (j == 0 && i < 11) {

if (matrix[i - 1][j] > 1 && label[i - 1][j] == 0) {
    stack.push(new int[]{i - 1, j});
    label[i - 1][j] = lab;
}

if (matrix[i - 1][j + 1] > 1 && label[i - 1][j + 1] == 0) {
    stack.push(new int[]{i - 1, j + 1});
    label[i - 1][j + 1] = lab;
}

if (matrix[i][j + 1] > 1 && label[i][j + 1] == 0) {
    stack.push(new int[]{i, j + 1});
    label[i][j + 1] = lab;
}

if (matrix[i + 1][j] > 1 && label[i + 1][j] == 0) {
    stack.push(new int[]{i + 1, j});
    label[i + 1][j] = lab;
}

if (matrix[i + 1][j + 1] > 1 && label[i + 1][j + 1] == 0) {
    stack.push(new int[]{i + 1, j + 1});
    label[i + 1][j + 1] = lab;
}

} else {

if (matrix[i - 1][j - 1] > 1 && label[i - 1][j - 1] == 0) {
    stack.push(new int[]{i - 1, j - 1});
    label[i - 1][j - 1] = lab;
}

if (matrix[i - 1][j] > 1 && label[i - 1][j] == 0) {
    stack.push(new int[]{i - 1, j});
    label[i - 1][j] = lab;
}

if (matrix[i - 1][j + 1] > 1 && label[i - 1][j + 1] == 0) {
    stack.push(new int[]{i - 1, j + 1});
    label[i - 1][j + 1] = lab;
}

} if (matrix[i][j - 1] > 1 && label[i][j - 1] == 0) {
    stack.push(new int[]{i, j - 1});
}
label[i][j - 1] = lab;
}
if (matrix[i][j + 1] > 1 && label[i][j + 1] == 0) {
    stack.push(new int[]{i, j + 1});
    label[i][j + 1] = lab;
}
if (matrix[i + 1][j - 1] > 1 && label[i + 1][j - 1] == 0) {
    stack.push(new int[]{i + 1, j - 1});
    label[i + 1][j - 1] = lab;
}
if (matrix[i + 1][j] > 1 && label[i + 1][j] == 0) {
    stack.push(new int[]{i + 1, j});
    label[i + 1][j] = lab;
}
if (matrix[i + 1][j + 1] > 1 && label[i + 1][j + 1] == 0) {
    stack.push(new int[]{i + 1, j + 1});
    label[i + 1][j + 1] = lab;
}
}
lab++;
}
outputArray[0] = lab - 1;
outputArray[1] = groupSize;
outputArray[2] = maxGroup;
//System.out.println("Max Group: " +maxGroup+" "+Lab +(lab-1)+" "+GS:
"+groupSize);
return outputArray;
}

currentIndex:

/* ConnectedComponent*/

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package myserver;

91
import java.util.Stack;

/**
 * @author fadimuheidat
 */
public class ConnectedComponent {

    public ConnectedComponent() {
        
        //System.out.println("CC obj Created !!");
    }

    int[] connectedComponet(int[][] matrix) {
        
        // TODO Auto-generated method stub
        // problems are teh boundary pixels/sensors
        
        int outputArray[] = new int[3];
        int nrow = matrix.length;
        int ncol = matrix[0].length;
        int lab = 1;
        int[] pos;
        Stack<int[]> stack = new Stack<>();
        int[][] label = new int[nrow][ncol];
        int groupSize = 0, maxGroup = 0;
        for (int r = 1; r < nrow - 1; r++) {
            for (int c = 1; c < ncol - 1; c++) {
                if (matrix[r][c] <= 1) {
                    continue;
                }
                if (label[r][c] > 0) {
                    continue;
                }
                /* encountered unlabeled foreground pixel at position r, c */
                /* push the position on the stack and assign label */
                stack.push(new int[] {r, c});
                label[r][c] = lab;
                if (groupSize > maxGroup) {
                    maxGroup = groupSize;
                }
                groupSize = 0;

                // System.out.println("New group");
                while (!stack.isEmpty()) {
        }
pos = stack.pop();
int i = pos[0];
int j = pos[1];
groupSize++;
//System.out.println("i "+i+" j "+j+" lab="+lab+" GroupSize:"+groupSize+" maxGroup:"+maxGroup);

//To DO
/* Add logic if the 2s are boundary*/
//pivot found , check neighbors
if (i == 0 && j <= 11) {
    if (i == 0 && j == 0) {
        if (matrix[i][j + 1] > 1 && label[i][j + 1] == 0) {
            stack.push(new int[]{i, j + 1});
            label[i][j + 1] = lab;
        }
        if (matrix[i + 1][j] > 1 && label[i + 1][j] == 0) {
            stack.push(new int[]{i + 1, j});
            label[i + 1][j] = lab;
        }
        if (matrix[i + 1][j + 1] > 1 && label[i + 1][j + 1] == 0) {
            stack.push(new int[]{i + 1, j + 1});
            label[i + 1][j + 1] = lab;
        }
    }
    else {
        if (matrix[i][j - 1] > 1 && label[i][j - 1] == 0) {
            stack.push(new int[]{i, j - 1});
            label[i][j - 1] = lab;
        }
        if (matrix[i][j + 1] > 1 && label[i][j + 1] == 0) {
            stack.push(new int[]{i, j + 1});
            label[i][j + 1] = lab;
        }
        if (matrix[i + 1][j - 1] > 1 && label[i + 1][j - 1] == 0) {
            stack.push(new int[]{i + 1, j - 1});
            label[i + 1][j - 1] = lab;
        }
        if (matrix[i + 1][j] > 1 && label[i + 1][j] == 0) {
            stack.push(new int[]{i + 1, j});
            label[i + 1][j] = lab;
        }
        if (matrix[i + 1][j + 1] > 1 && label[i + 1][j + 1] == 0) {
            stack.push(new int[]{i + 1, j + 1});
        }
    }
}
label[i + 1][j + 1] = lab;
}
}
}
}
}
}
if (j == 0 && i < 11) {
    if (matrix[i - 1][j] > 1 && label[i - 1][j] == 0) {
        stack.push(new int[]{i - 1, j});
        label[i - 1][j] = lab;
    }
    if (matrix[i - 1][j + 1] > 1 && label[i - 1][j + 1] == 0) {
        stack.push(new int[]{i - 1, j + 1});
        label[i - 1][j + 1] = lab;
    }
    if (matrix[i][j + 1] > 1 && label[i][j + 1] == 0) {
        stack.push(new int[]{i, j + 1});
        label[i][j + 1] = lab;
    }
    if (matrix[i + 1][j] > 1 && label[i + 1][j] == 0) {
        stack.push(new int[]{i + 1, j});
        label[i + 1][j] = lab;
    }
    if (matrix[i + 1][j + 1] > 1 && label[i + 1][j + 1] == 0) {
        stack.push(new int[]{i + 1, j + 1});
        label[i + 1][j + 1] = lab;
    }
}
}
}
else {
    if (matrix[i - 1][j - 1] > 1 && label[i - 1][j - 1] == 0) {
        stack.push(new int[]{i - 1, j - 1});
        label[i - 1][j - 1] = lab;
    }
    if (matrix[i - 1][j] > 1 && label[i - 1][j] == 0) {
        stack.push(new int[]{i - 1, j});
        label[i - 1][j] = lab;
    }
    if (matrix[i - 1][j + 1] > 1 && label[i - 1][j + 1] == 0) {
        stack.push(new int[]{i - 1, j + 1});
        label[i - 1][j + 1] = lab;
    }
    if (matrix[i][j - 1] > 1 && label[i][j - 1] == 0) {
        stack.push(new int[]{i, j - 1});
        label[i][j - 1] = lab;
    }
    if (matrix[i][j + 1] > 1 && label[i][j + 1] == 0) {
        stack.push(new int[]{i, j + 1});
        label[i][j + 1] = lab;
    }
    if (matrix[i + 1][j - 1] > 1 && label[i + 1][j - 1] == 0) {
        stack.push(new int[]{i + 1, j - 1});
        label[i + 1][j - 1] = lab;
    }
    if (matrix[i + 1][j] > 1 && label[i + 1][j] == 0) {
        stack.push(new int[]{i + 1, j});
        label[i + 1][j] = lab;
    }
    if (matrix[i + 1][j + 1] > 1 && label[i + 1][j + 1] == 0) {
        stack.push(new int[]{i + 1, j + 1});
        label[i + 1][j + 1] = lab;
    }
}
}
if (matrix[i][j + 1] > 1 && label[i][j + 1] == 0) {
    stack.push(new int[]{i, j + 1});
    label[i][j + 1] = lab;
}
if (matrix[i + 1][j - 1] > 1 && label[i + 1][j - 1] == 0) {
    stack.push(new int[]{i + 1, j - 1});
    label[i + 1][j - 1] = lab;
}
if (matrix[i + 1][j] > 1 && label[i + 1][j] == 0) {
    stack.push(new int[]{i + 1, j});
    label[i + 1][j] = lab;
}
if (matrix[i + 1][j + 1] > 1 && label[i + 1][j + 1] == 0) {
    stack.push(new int[]{i + 1, j + 1});
    label[i + 1][j + 1] = lab;
}
lab++;
}
outputArray[0] = lab - 1;
outputArray[1] = groupSize;
outputArray[2] = maxGroup;
//System.out.println("Max Group: " +maxGroup+" "+"Lab "+(lab-1)+" "+"GS: "+groupSize);
return outputArray;
}
public class FrameIdGlobal {

    public static int FID=0;

}

package myserver;

import java.util.Stack;

public class GCC {

    public void GCC() {
    }

    public int GCC(int[][] matrix) {
        // TODO Auto-generated method stub
        int nrow = matrix.length;
        int ncol = matrix[0].length;
        int lab = 1;
        int[] pos;
        Stack<int[]> stack = new Stack<>();
        int[][] label = new int[nrow][ncol];
        int groupSize = 0, maxGroup = 0;
        for (int r = 1; r < nrow - 1; r++) {
            for (int c = 1; c < ncol - 1; c++) {
                if (matrix[r][c] <= 1) {
                    continue;
                }
                if (label[r][c] > 0) {
                    continue;
                }
                }
/* encountered unlabeled foreground pixel at position r, c */
/* push the position on the stack and assign label */
stack.push(new int[]{r, c});
label[r][c] = lab;
if (groupSize > maxGroup) {
    maxGroup = groupSize;
}

groupBySize = 0;
System.out.println("New group");
while (!stack.isEmpty()) {
    pos = stack.pop();
    int i = pos[0];
    int j = pos[1];
    //System.out.println("i "+i+" j "+j);
    groupSize++;
    if (matrix[i - 1][j - 1] > 1 && label[i - 1][j - 1] == 0) {
        stack.push(new int[]{i - 1, j - 1});
        label[i - 1][j - 1] = lab;
    }
    if (matrix[i - 1][j] > 1 && label[i - 1][j] == 0) {
        stack.push(new int[]{i - 1, j});
        label[i - 1][j] = lab;
    }
    if (matrix[i - 1][j + 1] > 1 && label[i - 1][j + 1] == 0) {
        stack.push(new int[]{i - 1, j + 1});
        label[i - 1][j + 1] = lab;
    }
    if (matrix[i][j - 1] > 1 && label[i][j - 1] == 0) {
        stack.push(new int[]{i, j - 1});
        label[i][j - 1] = lab;
    }
    if (matrix[i][j + 1] > 1 && label[i][j + 1] == 0) {
        stack.push(new int[]{i, j + 1});
        label[i][j + 1] = lab;
    }
    if (matrix[i + 1][j - 1] > 1 && label[i + 1][j - 1] == 0) {
        stack.push(new int[]{i + 1, j - 1});
        label[i + 1][j - 1] = lab;
    }
    if (matrix[i + 1][j] > 1 && label[i + 1][j] == 0) {
        stack.push(new int[]{i + 1, j});
        label[i + 1][j] = lab;
    }
    if (matrix[i + 1][j + 1] > 1 && label[i + 1][j + 1] == 0) {
        stack.push(new int[]{i + 1, j + 1});
        label[i + 1][j + 1] = lab;
    }
}
stack.push(new int[]{i + 1, j + 1});
label[i + 1][j + 1] = lab;
}
}
lab++;
}
System.out.println("Max Group: " + maxGroup);
for (int[] computedMatrix1 : label) {
    String row = "";
    for (int c = 0; c < computedMatrix1.length; c++) {
        row = row + " " + computedMatrix1[c];
    }
    System.out.print(computedMatrix1[c]);
    System.out.println(" ");
}
return maxGroup;
}

/* Centroid.java*/

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package myserver;

import java.awt.Point;
import java.util.ArrayList;
import java.util.List;

/**
 * @author fadimuheidat
 */

98
public class Centroid {

    List<Point> lst = new ArrayList<>();
    // List<Point> lst = new ArrayList<>();

    public Centroid() {
        // System.out.println("Centroid Object Created");
    }

    @SuppressWarnings("empty-statement")
    public Point GetCentroid(int[][] matrix) {
        for (int r = 0; r < matrix.length; r++) {
            for (int c = 0; c < matrix.length; c++) {
                if (matrix[r][c] == 1) {
                    lst.add(new Point(r, c));
                    // System.out.println("r:" + r + " c:" + c);
                }
            }
            // call Calculate Centroid
            Point c = new Point(0, 0);
            if (lst.size() >= 2) {
                c = calculateCentroid(lst);
                // System.out.println(c.getX()+","+c.getY());
                // newC = new Point((int)(c.getX()-PreviousCentroid.getX()), (int)(c.getY()-PreviousCentroid.getY()));
                // PreviousCentroid = c;
                // System.out.println(newC.getX()+","+newC.getY());
            }
        }
        return c;
    }

    @SuppressWarnings("empty-statement")
    private Point calculateCentroid(List<Point> lst) {
        int Cx = 0;
        int Cy = 0;
        int numPoints = lst.size();
        Point centroid;
        for (Point lst1 : lst) {
            // pointsString += "x=" + lst1.x + "y=" + lst1.y;
        }
    }
}

// Example usage:
public class Main {
    public static void main(String[] args) {
        int[][] matrix = {{0, 0, 1, 0, 0}, {0, 1, 1, 1, 0}, {0, 1, 0, 1, 0}, {0, 0, 0, 0, 0}};
        Centroid centroid = new Centroid();
        Point centroidPoint = centroid.GetCentroid(matrix);
        System.out.println("Centroid Point: x=" + centroidPoint.getX() + ", y=" + centroidPoint.getY());
    }
}
```java
    Cx += lst1.x;
    Cy += lst1.y;
}
// System.out.println(lst.size());
centroid = new Point(Cx / numpoints, Cy / numpoints);
return centroid;
}

/* ConvexHull.java*/

/*
* To change this license header, choose License Headers in Project Properties.
* To change this template file, choose Tools | Templates
* and open the template in the editor.
*/
package myserver;
import java.awt.Point;
import java.util.ArrayList;
import java.util.List;
/**
* @author fadimuheidat
*/
public class ConvexHull {
    List<Point> lst = new ArrayList<>();
    /**
     *
     * @param matrix
     * @return
     */
    public ArrayList<Point> getList(int[][] matrix) {
        for (int r = 0; r < matrix.length; r++) {
            for (int c = 0; c < matrix.length; c++) {
                if (matrix[r][c] == 1) {
                    lst.add(new Point(r, c));
                }
            }
        }
        return lst;
    }
}
```
// System.out.println("r:"+r+" c:"+c);
}
}

return (ArrayList<Point>) lst;
}

public ArrayList<Point> quickHull(ArrayList<Point> points) {
    ArrayList<Point> convexHull = new ArrayList<Point>();
    if (points.size() < 3) {
        return (ArrayList) points.clone();
    }

    int minPoint = -1, maxPoint = -1;
    int minX = Integer.MAX_VALUE;
    int maxX = Integer.MIN_VALUE;
    for (int i = 0; i < points.size(); i++) {
        if (points.get(i).x < minX) {
            minX = points.get(i).x;
            minPoint = i;
        }
        if (points.get(i).x > maxX) {
            maxX = points.get(i).x;
            maxPoint = i;
        }
    }
    Point A = points.get(minPoint);
    Point B = points.get(maxPoint);
    convexHull.add(A);
    convexHull.add(B);
    points.remove(A);
    points.remove(B);

    ArrayList<Point> leftSet = new ArrayList<>();
    ArrayList<Point> rightSet = new ArrayList<>();
    for (int i = 0; i < points.size(); i++) {
        Point p = points.get(i);
        if (pointLocation(A, B, p) == -1) {
            leftSet.add(p);
        } else if (pointLocation(A, B, p) == 1) {
            rightSet.add(p);
        }
    }
    hullSet(A, B, rightSet, convexHull);
hullSet(B, A, leftSet, convexHull);

/* System.out.println("inside convex Hull ");
convexHull.stream().forEach((p1) -> {
    System.out.println("(" + p1.x + ", " + p1.y + ")");
}); */
return convexHull;

public int distance(Point A, Point B, Point C) {
    int ABx = B.x - A.x;
    int ABy = B.y - A.y;
    int num = ABx * (A.y - C.y) - ABy * (A.x - C.x);
    if (num < 0) {
        num = -num;
    }
    return num;
}

public void hullSet(Point A, Point B, ArrayList<Point> set, ArrayList<Point> hull) {
    int insertPosition = hull.indexOf(B);
    if (set.isEmpty()) {
        return;
    }
    if (set.size() == 1) {
        Point p = set.get(0);
        set.remove(p);
        hull.add(insertPosition, p);
        return;
    }
    int dist = Integer.MIN_VALUE;
    int furthestPoint = -1;
    for (int i = 0; i < set.size(); i++) {
        Point p = set.get(i);
        int distance = distance(A, B, p);
        if (distance > dist) {
            dist = distance;
            furthestPoint = i;
        }
    }
    Point P = set.get(furthestPoint);
    set.remove(furthestPoint);
    hull.add(insertPosition, P);
// Determine who's to the left of AP
ArrayList<Point> leftSetAP = new ArrayList<Point>();
for (int i = 0; i < set.size(); i++) {
    Point M = set.get(i);
    if (pointLocation(A, P, M) == 1) {
        leftSetAP.add(M);
    }
}

// Determine who's to the left of PB
ArrayList<Point> leftSetPB = new ArrayList<>();
for (int i = 0; i < set.size(); i++) {
    Point M = set.get(i);
    if (pointLocation(P, B, M) == 1) {
        leftSetPB.add(M);
    }
}
hullSet(A, P, leftSetAP, hull);
hullSet(P, B, leftSetPB, hull);

}

public int pointLocation(Point A, Point B, Point P) {
    int cp1 = (B.x - A.x) * (P.y - A.y) - (B.y - A.y) * (P.x - A.x);
    if (cp1 > 0) {
        return 1;
    } else if (cp1 == 0) {
        return 0;
    } else {
        return -1;
    }
}

double HullArea(ArrayList<Point> polyPoints) {
    int i, j, n = polyPoints.size();
    //System.out.println("n:"+n);
    double area = 0;
    if (n < 3) {
        return area;
    } else {
        for (i = 0; i < n - 1; i++) {
            //j = (i + 1) % n;
            //System.out.println(i+ " "+j);
            area += (polyPoints.get(i).getX() * polyPoints.get(i + 1).getY()) -
                    polyPoints.get(i + 1).getX() * polyPoints.get(i).getY());
    }
}
} //System.out.println(i);
    area += (polyPoints.get(i).getX() * polyPoints.get(0).getY() -
            polyPoints.get(0).getX() * polyPoints.get(i).getY());
    area /= 2.0;
    }
    return (Math.abs(area));

    }

    }/* Notification.java*/

    /*
    * To change this license header, choose License Headers in Project Properties.
    * To change this template file, choose Tools | Templates
    * and open the template in the editor.
    */
    package myserver;

    /**
     *
     * @author fadimuheidat
     */
    public class Notification {

        boolean level1NotificationSent = false;
        boolean level2NotificationSent = false;
        boolean level3NotificationSent = false;

        /* public static void main(String[] args) {
        // portList = CommPortIdentifier.getPortIdentifiers();
            Notification nobj = new Notification();
        for(int i=0; i<3;i++)
            nobj.Notify(i+1);

        /* while (portList.hasMoreElements()) {
            portId = (CommPortIdentifier) portList.nextElement();
            if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
                if (portId.getName().equals("/dev/ttyUSB0")) {
        //                if (portId.getName().equals("/dev/term/a")) {
                    SimpleRead reader = new SimpleRead();
            }
            }
        }

        } //System.out.println(i);

    }
public Notification() {
    // System.out.println("No Alert yet");
}

class Notification {
    public void Notify(int levelnumber) {
        // TODO Auto-generated method stub
        if (levelnumber == 1) {
            if (!level1NotificationSent) {
                sendMobileNotification("Level 1 Alert");
                //sendPagerNotification("Level 1 Alert");
                System.out.println("Level 1 Notification sent");
                level1NotificationSent = true;
                level2NotificationSent = false;
                level3NotificationSent = false;
            }
        }
        if (levelnumber == 2) {
            if (!level2NotificationSent) {
                sendMobileNotification("Level 2 Alert");
                //sendPagerNotification("Level 2 Alert");
                System.out.println("Level 2 Notification sent");
                level1NotificationSent = false;
                level2NotificationSent = true;
                level3NotificationSent = false;
            }
        }
        if (levelnumber == 3) {
            if (!level3NotificationSent) {
                sendMobileNotification("Level 3 Alert");
                //sendPagerNotification("Level 3 Alert");
                System.out.println("Level 3 Notification sent");
                level1NotificationSent = false;
                level2NotificationSent = false;
                level3NotificationSent = true;
            }
        }
    }

    public void sendMobileNotification(String msg) {
        System.out.println(msg);
    }
}

105
public void sendFallNotification() {
    System.out.println("this is a Fall!");
}

/* MySQLAccess.java*/
/**
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
/**
 * @author fadimuheidat
 */
package myserver;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Date;

public class MySQLAccess {

    private Connection connect = null;
    private Statement statement = null;
    private PreparedStatement preparedStatement = null;
    private ResultSet resultSet = null;

    public void insertdata(String xcord, String ycord, String i, String PName, String FallName) throws Exception {
        //System.out.println("here");
        try {

            // This will load the MySQL driver, each DB has its own driver
            Class.forName("com.mysql.jdbc.Driver");
            // Setup the connection with the DB
            connect = DriverManager

.getConnection("jdbc:mysql://localhost/smart_carpet?"
+ "user=root&password=root");
//System.out.println("connected successfully!");

// Statements allow to issue SQL queries to the database
statement = connect.createStatement();

// PreparedStatements can use variables and are more efficient
// "myuser, webpage, datum, summery, COMMENTS from feedback.comments"
// Parameters start with 1
preparedStatement.setString(1, xcord);
preparedStatement.setString(2, ycord);
preparedStatement.setString(3, Integer.toString(i);
// java.sql.Date date = getCurrentJavaSqlDate();
java.util.Date dt = new java.util.Date();
java.text.SimpleDateFormat sdf;
sdf = new java.text.SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS");
String currentTime = sdf.format(dt);
//System.out.println(currentTime);
preparedStatement.setString(4, currentTime);
preparedStatement.setString(5, "0");
preparedStatement.setString(6, "0");
preparedStatement.setString(7, "0");
preparedStatement.setString(8, "PName");
preparedStatement.setString(9, "FallName");

preparedStatement.executeUpdate();

} catch (ClassNotFoundException | SQLException e) { throw e; }
} finally {
    close();
}

// You need to close the resultSet
private void close() {
    try {
        if (resultSet != null) {
            resultSet.close();
        }
    }
}
if (statement != null) {
    statement.close();
}

if (connect != null) {
    connect.close();
}
} catch (Exception e) {
    
}

public java.sql.Date getCurrentJavaSqlDate() {
  java.util.Date today = new java.util.Date();
  return new java.sql.Date(today.getTime());
}

public java.sql.Date getCurrentJavaSqlDate() {
  java.util.Date today = new java.util.Date();
  return new java.sql.Date(today.getTime());
}

package myserver;
import com.firebase.client.Firebase;
import com.firebase.client.FirebaseError;

/**
 * @author fadimuheidat
 */
public class StoreFirebase {
  public void save(int index) throws InterruptedException {
    Firebase ref = new Firebase("https://omu9k3h1q20.firebaseio-demo.com/smartcarpet");
    Firebase hopperRef = ref.child("gracehop");
  }
}
ref.child("Location")./*push().*/setValue(index, new Firebase.CompletionListener()
{
    //hopperRef.push().setValue("I'm fadi....", new
Firebase.CompletionListener() {
    @Override
    public void onComplete(FirebaseError firebaseError, Firebase firebase) {
        if (firebaseError != null) {
            System.out.println("Data could not be saved. "+
firebaseError.getMessage());
        } else {
            //System.out.println("Data saved successfully.");
        }
    }
});
}

/* SendMailTLS.java*/

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package myserver;

/**
 *
 * @author fadimuheidat
 */
import java.util.Properties;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

public class SendMailTLS {
    //public static void main(String[] args) {

public SendMailTLS() {

    System.out.println("send email object");
}

public void sendEmail(String msg) {
    final String username = "xyz@gmail.com";
    final String password = "xxxxxxxxxxxxx";

    Properties props = new Properties();
    props.put("mail.smtp.auth", "true");
    props.put("mail.smtp.starttls.enable", "true");
    props.put("mail.smtp.host", "smtp.gmail.com");
    props.put("mail.smtp.port", "587");

    Session session = Session.getInstance(props, new javax.mail.Authenticator() {
        protected PasswordAuthentication getPasswordAuthentication() {
            return new PasswordAuthentication(username, password);
        }
    });

    try {
        Message message = new MimeMessage(session);
        message.setFrom(new InternetAddress("xyz@gmail.com"));
        message.setRecipients(Message.RecipientType.TO, InternetAddress.parse("abc@gmail.com"));
        message.setSubject("Fall Detected ");
        message.setText(msg);
        /* message.setText("Dear Healthcare Admin,\n                   \n                   Somebody is falling!"); */

        Transport.send(message);

        System.out.println("Done");
    } catch (MessagingException e) {
        throw new RuntimeException(e);
    }
}
}
8.2 Appendix B: CD-ROM contents: experimental data and code.

Attached with the thesis