Activity Recognition using Sun SPOT

Rahul Goyal **Arvind Mahla** **Jagjeet Dhaliwal** **Ravee Malla**
2008CS50222 2011ANZ8418 2008CS50212 2008CS50224
{rahulgoyal34, arvindmahla, jagjeetdhaliwal, ravmalla}@gmail.com

Final Report, Wireless Networks (CSL838)
Supervised by: **Dr. Vinay Ribeiro**

8 May 2012

1 Introduction

Activity Recognition is a well studied problem. It has been attempted by employing video [3], environmental [5] or other wearable sensors which report physical quantities like acceleration, pressure, etc. We are using Sun SPOT, an accelerometer prototype by Sun that works on wireless Personal Area Network technology to communicate data. Such an approach is preferred over other approaches like video surveillance (which puts privacy at risk and is expensive) and environmental sensors which are inaccurate. Our objective is to place one or more motes at different places of the human body and use the data collected to determine the current activity being performed by the individual.

1.1 Motivation

Such an application can be used in various settings. For eg., this device can be worn by the elderly, and it can send data to the hospital/relatives. Any unusual activity recorded by the mote, like falling down or being unusually still can be immediately reported and looked into. It can also be used along with GPS to give more exact location, since we can track the movement of the subject over a period of time. It can be used by Sports/Dance Coaches to monitor the movement of body parts. It is also useful in wartime scenario to find injured soldiers who have restricted movements.

Organization. We begin by briefly describing the Sun SPOT motes, how they are configured, various ways by which they communicate data. We present a brief literature survey of the past work of Activity Recognition, in general and using Sun SPOT. We then describe the activities that were considered, and how we designed our experiments. Then we present our results and accuracy of Activity Recognition for various activities. Finally, we conclude by making some general remarks and suggesting further extensions of this project.

2 Sun SPOT

Sun SPOT is a wireless mote device developed by Sun Microsystems. It can be used to create a Wireless Sensor Network(WSN) based on the IEEE 802.15.4 standard (Zigbee) and can be programmed using Squawk Java VM. The basic Sun SPOT device architecture has the following layers.

- Sensor board
- Preprocessor board
- Battery

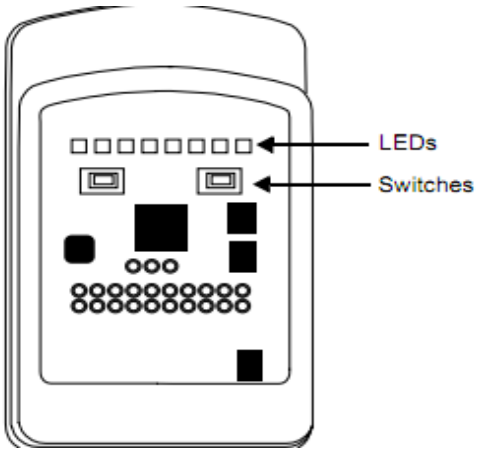

Figure 1: Front view of the wireless mote

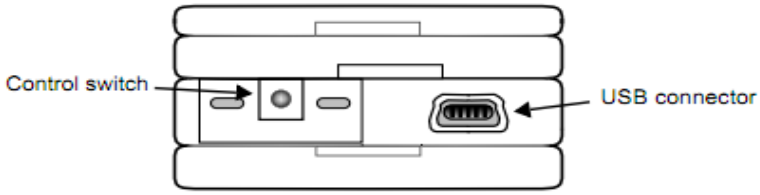

Figure 2: Top view of the wireless mote

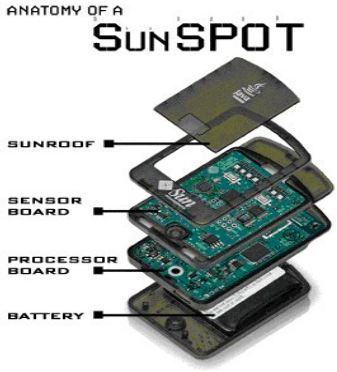

Figure 3: Anatomy of a Sun SPOT

It has various features like

- Three-axis accelerometer
- Temperature & Light sensor
- Radio Transmitter based on Zigbee protocol
- 8 multi-color LEDs

It has found various applications in the fields of Surveillance, Art & Toys, Robotics, etc.

2.1 Protocol implemented

The protocol implemented by Sun SPOT device is the IEEE 802.15.4 Zigbee standard. This protocol is aimed towards a low cost, low power, short range communication across multiple clients and a single basestation. It can support sustained speeds of 250 Kbits/sec over a range of 10 meters. For handling packet collision, it can support CSMA/CA with guaranteed time slots for all the clients. Since it is a WPAN network, it can operate in 3 fixed unlicensed bands viz. 868.0-868.6 MHz (Europe), 902-928 MHz (North America) & 2400-2483.5 MHz (worldwide).

2.2 How we used the device

The Sun SPOT[1] device was issued from the Wireless Lab. The box contains 1 base station, 2 wireless motes and documentation. The motes are low power, light-weight accelerometers which report in real-time, the acceleration along 3 dimensions, the tilt, the temperature, etc. Each of the wireless motes consists a set of LEDs, some buttons and USB connectivity to load the program. The program is burnt onto these devices and then they operate without any supervision. The base station acts as the mediator between the motes and our analyzer program. It implements & manages the wireless protocol used by the motes for communication. The wireless mote reports accelerations in units of g^2, hence the noted readings are scaled by this factor. The mote has a range of about 10-15 meters and can measure acceleration of magnitudes upto $6g$.

2.3 Application Architecture

Our Activity Recognition happens by running 2 applications in parallel, one running on the host which collects the data across all the motes, and the other running on the mote which computes the current acceleration along the 3 axis and sends it to the host.

2.4 Sample Application

We began by trying out certain sample applications provided by Sun on the devices. These helped us get a hold of the API that we can use to get the necessary acceleration values of the motes. One application uses the current orientation of the Sun SPOT to simulate effect of gravity on a ball represented as one of 8 LEDs (the ball moves along these 8 LEDs). Another application prints a welcome message on the LEDs with the speed of message controlled by the acceleration values.

2.5 Calibration

There is a need to calibrate the motes so as to maintain consistency in taking the readings. For this we followed the procedure described in the manual[3]. Fig. 4 & 5 show the schematic diagrams of the mote which show the axis convention adopted and the methodology for computing the tilt. Before calibration, even when the mote was stationary, we found some fluctuating non-zero values. When the device is perfectly calibrated, we find that the only acceleration is along the Z-direction because of acceleration due to gravity. The values are multiplied by g

2.6 Deployment on the mote

There are 2 ways to deploy a program on a given mote.

1. Connect the mote directly to the host machine and load the code on the mote
2. Connect the mote wirelessly to the base station which is connected to the host. The program is then loaded onto the mote

We tried with both methods, but we have decided to use the second method since it is easier to deploy. The mote runs a Sqwauwk Java program as an OS and the host machine runs on a J2SE JVM.

2.7 3 ways to run the protocol

We implemented the Zigbee protocol to communicate the acceleration values to the host in the following 3 ways

[1] www.sunspotworld.com
[2] acceleration due to gravity
[3] http://www.sunspotworld.com/docs/AppNotes/AccelerometerAppNote.pdf

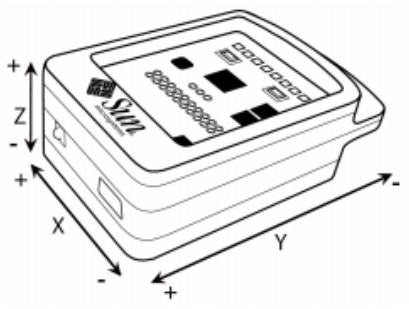

Figure 4: Axis convention for the mote

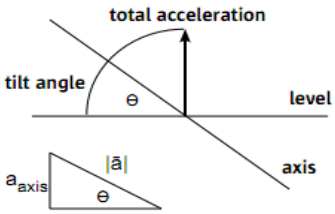

Figure 5: Method of computing tilt values

- Datagram communication protocol where the mote sent Datagram packets at regular intervals. This was not very reliable, since multiple motes would collide at the host and not retransmit their lost packets. Also, the address of the mote was not saved by the host, and it was repeatedly determined by examining the packet itself
- Handshaking before every connection request. This is similar to HTTP where the mote requesting for a persistent connection from the basestation once at the start, and there on bandwidth was reserved for the mote. This method allows an on-demand bandwidth allocation to the motes. So more and more motes can join the Wireless Sensor network. But there was an overhead of initial handshaking
- Hardcoding the connections for each of the motes

We are currently using the third mechanism which allows us to skip the stage of handshaking.

3 Past Work

Activity Recognition in general is a well studied problem in the context of images & video [1, 3]. These technologies have their own disadvantages. For eg. in the context of video, there are issues of privacy. Also, a rich set of training data is required for reasonable accuracy in predicting the activity and even then it is subject dependent. Recently, with the explosion of smart phones and wireless devices, efforts have been made to integrate these devices seamlessly into users daily routines[9, 6, 7]. The basic idea in the latter approach is to measure physical variable like acceleration, temperature, etc. in real-time, and predict the activity being performed.

We found a PhD dissertation [4] to be very useful as our reference. It looks at many aspects of activity recognition using wireless sensors, and we plan to follow it closely. We have also reviewed other work[1], which helped us answer questions like the polling frequency of acceleration data logging (which is kept around 20-30 Hz), location of placing sensors, etc. It is widely believed that subject independent activity recognition is very difficult. Also, in a practical scenario the activies are much more unconstrained, hence these are much harder to classify. It is also very hard to acquire clean training set. Activities are commonly classified as

1. Static: Lying, Sitting, Standing
2. Transitions: Lying $\rightarrow$ Standing, Standing $\rightarrow$ Lying, Lying $\rightarrow$ Sitting, Walking $\rightarrow$ Standing, etc
3. Dynamic: Walking, Running, Hopping

We have found that once acceleration data from various subjects performing different activities has been collected, two kinds of classification approaches can be applied. One can model the logged acceleration values as a time series and apply basic signal processing [2] to extract features of characteristic to each activity. The other approach is to apply higher level Machine Learning ideas like Bayesian and Hidden Markov Models [8, 10, 6].

Figure 6: Location of the 2 wireless sensor motes on the body

4 Methodology 1

4.1 Data Collection

For data collection, we decided to log the data readings from 2 wireless sensors, one placed above the elbow, while the other just above the knee cap as shown in the figure. However, in our later experiments we decided to place the mote inside trouser pockets as it was more convenient. The activities (comprising of several static as well as dynamic activities) that we have decided to have trained models are walking, running, hopping, lying down, sitting and standing. So the data collection involves collecting clean data sets for all the above activities. We placed the wireless motes on two people on positions specified in the figure and made them do each of the above activities at least three times to create a large enough data set to analyze. Each reading was recorded over a time span of 18-20 seconds at a sample rate of 5 Hz to give us a large enough window to get a clean data set. A time window this large was chosen because the first and the last 5 seconds of the readings had to be ignored owing to the observation that the wireless motes took some time before they actually started sending information to the base station after they had been activated and also the pattern at the beginning or near the end of any activity was unreliable. This gave us about 8-10 second long clean data logs from which we had to analyze and train our models for those particular activities.

The values recorded for each experimented were the calibrated acceleration values reported from each of the motes, in the 3 dimensions. Using these readings, we experimented with the representation of this data, that would work best for this set. The issue was that since each input sample is a time evolving set of values, a suitable representation is difficult. If we take a time average of the whole sample, it might lead to significant errors in detection, since we ignore the rich time variation pattern of each of the activities. If we took all the time sample values as independent dimensions, this would lead to the data set being computationally infeasible and incorrect results due to the so called *curse of dimensionality*[4].

Hence, in our representation, we show the each of the input samples as point the space $\mathbb{R}^9$. Each of these points, represent an activity done by an individual. Using these, we developed the class conditional distributions of the activities independent of the person doing that activity. Finally, we make the prediction using MLE method.

We have decided to log the data readings from 2 wireless sensors, though we are yet to decide whether we will need the readings from both of them. One is placed above the elbow, while the other above the knee cap. The positions are shown in Fig. 6.

Our approach is divided into 2 phases, the training phase and the classification phase. These phases are represented as block diagrams in Fig. 7 & 8. Initially, we will be collecting labelled training data. We will ask various subjects to perform fixed activities like standing up, walking, running, etc while wearing the motes, and

[4]http://en.wikipedia.org/wiki/Curse_of_dimensionality

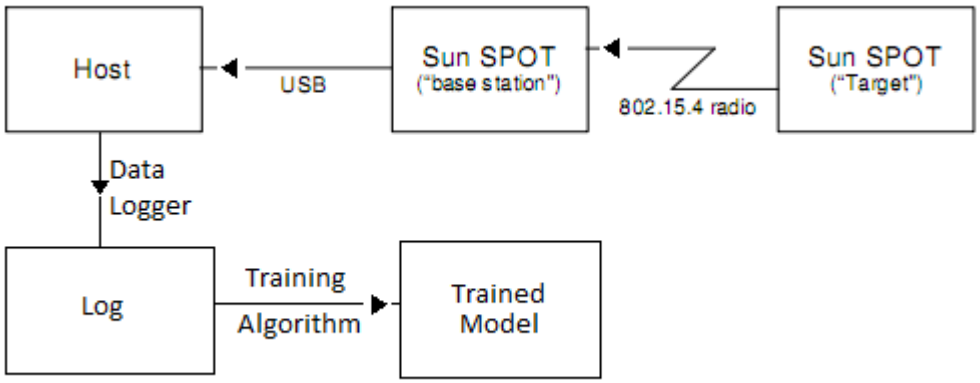

Figure 7: Training Phase of Algorithm

record the readings obtained. These will be logged under the suitable activity class. We will then use these logs offline, to build a rich trained model. This model may even be represented as a set of fuzzy/logic rules or certain statistical measures which can later be used for classification.

For the second phase, we implement a real-time classifier, which looks at the acceleration reading of the mote(s) for the past 6 second window, and predict the activity being performed at that instant. Note, that this could be a composite activity like standing up, walking and then running. Using the trained model, and a set of learned rules, we are able to classify the current pattern as an activity.

The process of building a training model involves two fundamental steps.

1. Modelling the input data in some representation, assigning some distribution to it
2. Defining a particular predictor strategy which will be used to chose one class over another while testing

For the first step, we have taken the representation of our input set as a point in $\mathbb{R}^9$ space as defined above. Additionally, we will assume that each of the activities (classes) generate points in this space independently using a gaussian distribution. The gaussian distribution in d-dimensions is given as

$$\ln P(\mathbf{x}) = -\pi d - \frac{|\mathbf{\Sigma}|}{2} - \frac{(\mathbf{x}-\mu)'\mathbf{\Sigma^{-1}}(\mathbf{x}-\mu)}{2} \tag{1}$$

where Σ represents the cross-correlation matrix between the dimensions, μ represents the mean. By MLE theory, we get that for the given test data points, the most accurate predictor class-conditional distributions are when we set the variance and mean as the following.

$$\mu = \frac{\Sigma_i x_i}{N} \tag{2}$$

$$\Sigma_{ij} = \mathbb{E}[x_i x_j] - \mathbb{E}[x_i]\mathbb{E}[x_j] \tag{3}$$

Hence, using this thoery, we calculated the appropriate means and variances for each of the classes, i.e. walking, running, lying, hopping & standing. We have taken the readings from the arm to be more appropriate. Having calculated the CCD (Class-Conditional Distributions) for each of the classes, we can use the Bayesian rule shown below to classify a new activity in the following way. Given a test sample point $\mathbf{x}$, we will evaluate the probability $P(\omega|\mathbf{x})$ where ω is a class (either of standing, lying, running, hopping, etc) and then choose that ω for which $P(\omega|\mathbf{x})$ is the highest.

$$P(\mathbf{x}|\mathbf{y}) \propto P(\mathbf{y}|\mathbf{x})P(\mathbf{x}) \tag{4}$$

The following are the plots of acceleration variation for various activities that were logged using a Python code attached in appendix. Using these plots, we made our observations and rules to predict activities. Consider

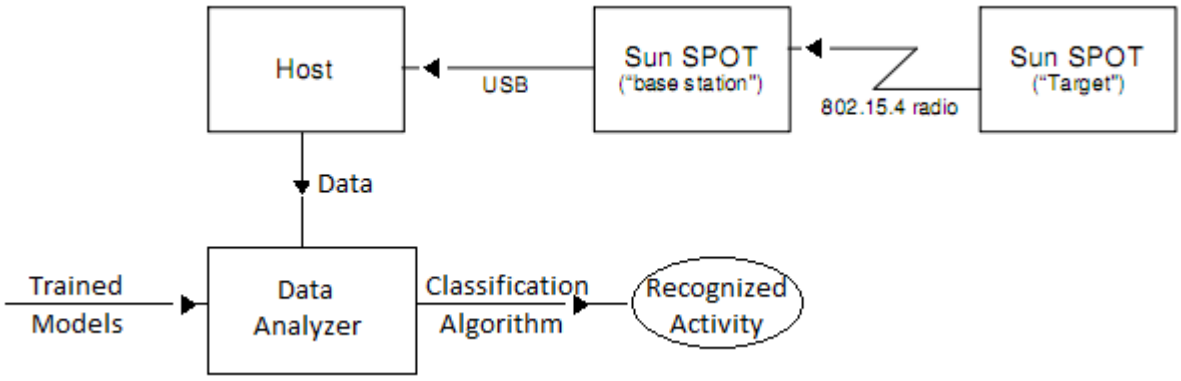

Figure 8: Training Phase of Algorithm

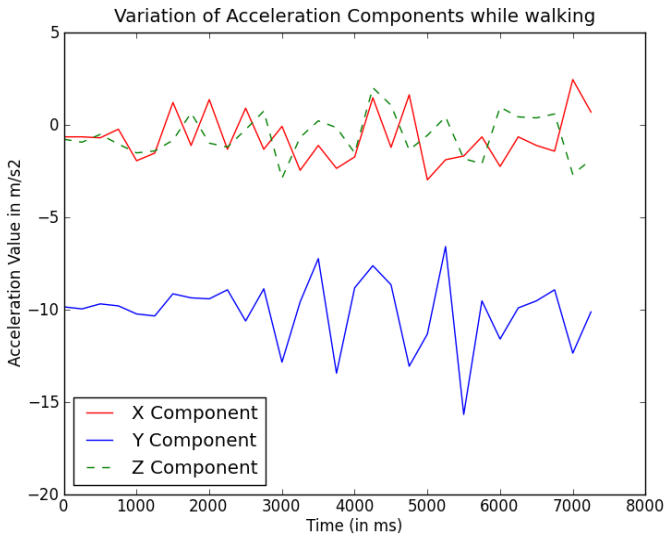

Figure 9: Acceleration Plots for Walking from Mote above Elbow

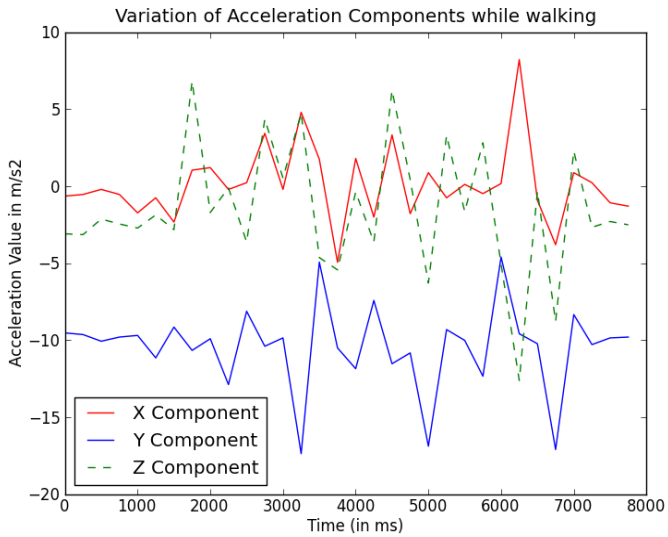

Figure 10: Acceleration Plots for Walking from Mote above Knee

fig. 8,9,10,11. If we compare running and walking activities, we can clearly see that for plot in fig. 8 & 10, the component of acceleration in X direction shows drastic change in case of running as compared to walking. This is due to the fact that while running the lateral movement of our arms is much more in running than in walking. Also, since the movement is fast while running, the magnitude of acceleration in Y direction is also more. In case of hopping/jumping activity, we can clearly see that there is a large margnitude of unidirectional acceleration in Y direction because due up-down motion.

Consider fig. 14 & 15 which are plots for the transition sitting to standing. We find that for the mote above the elbow, the initial and final acceleration states are identical which is as expected since the hand comes to the same position at the end of the transition. However, for the mote above the knee, acceleration due to gravity was earlier along the Z axis and in the final state the orientation changes. Such a scenario will be fairly universal across subjects, and can be used to predict such activities easily. Hence in case mote 1 reports no change while mote 2 changes, we can conclude that the above transition has occured. A similar rule can also be applied for reverse transition of standing to sitting. For the transition lying to standing, we can note that the acceleration states of both the motes will change. We can develop similar rules for all other activities.

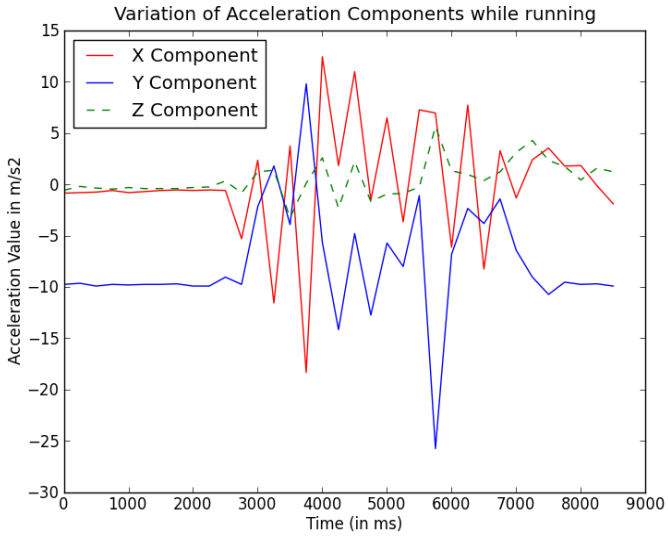

Figure 11: Acceleration Plots for Running from Mote above Elbow

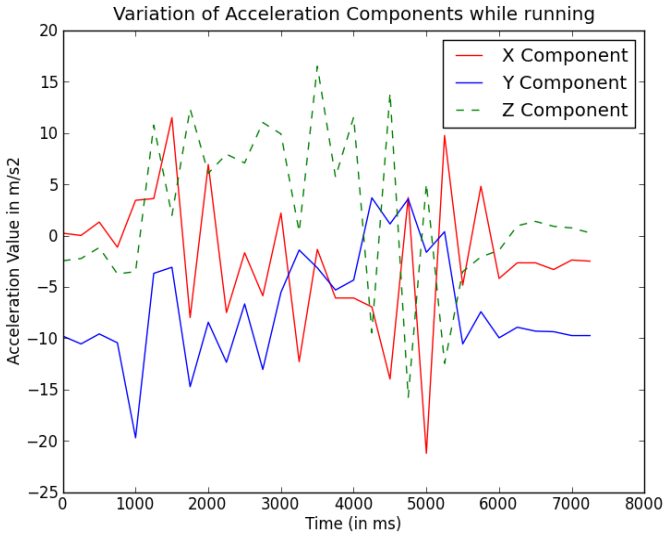

Figure 12: Acceleration Plots for Running from Mote above Knee

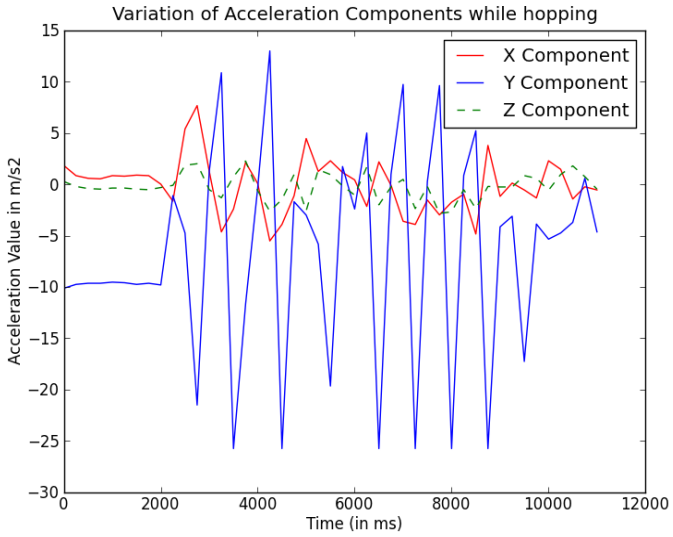

Figure 13: Acceleration Plots for Hopping from Mote above Elbow

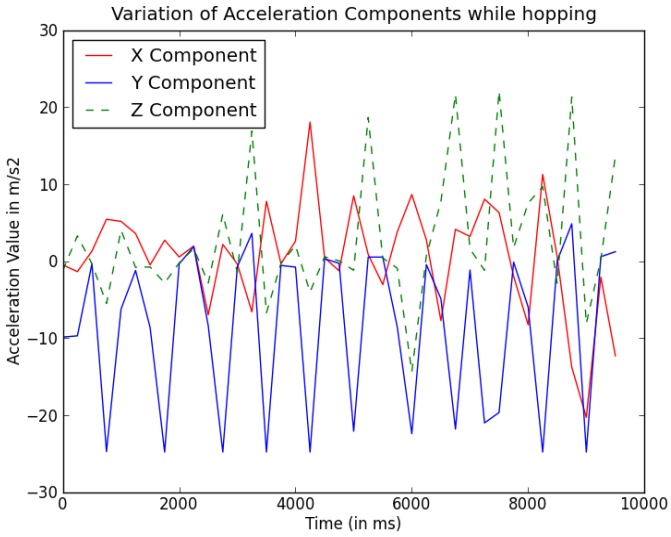

Figure 14: Acceleration Plots for Hopping from Mote above Knee

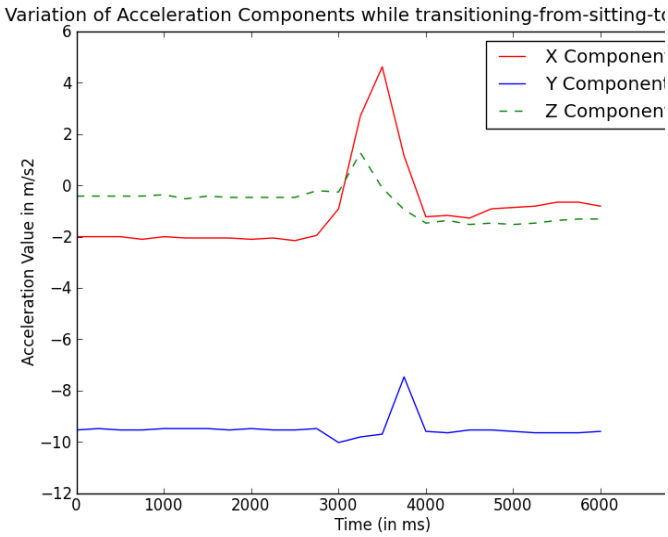

Figure 15: Acceleration Plots in transitioning from standing to sitting using Mote above Elbow

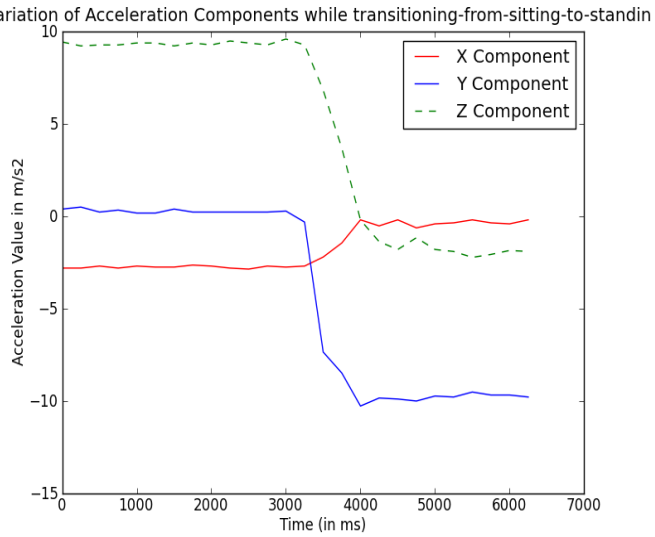

Figure 16: Acceleration Plots in transitioning from standing to sitting using Mote above Knee

5 Methodology 2

We found that for our case, the sophisticated Bayesian learning we explained did not give good results. This was because of lack of training data and data being temporal in nature. Hence we resorted to a simpler approach of observing the following along the 3 directions of acceleration

- Standard deviation
- Range (max - min)

We applied control flow rules on these values and have got very good results (almost 100% accuracy). We noticed that these values are characteristic of the activity performed, and hence their use was justified. The code attached below implements the ideas mentioned above to have activity prediction.

6 Experimental Set Up

Finally, we have been able to interface with multiple motes in parallel. We plot the real time acceleration values for all motes and show the average value in the past window. On the interface, we show the activity predicted based on the data collected in the last 6 seconds. Upto 4 people can start performing an activity, and the prediction algorithm displays the predicted activity.

7 Results

On our interface, we show the past variation of accleration values along the 3 independent directions in time. We have used the JFreeChart[5] java library to plot these values. One can also see the current activity being performed based on the activity observed in the past 10 seconds. The main drawback of our approach is the need to keep the mote in an upright position and that during a transition between 2 activities, we have not predicted the correct activity. The programming was done in Java using the Netbeans 7 IDE which has the

[5]http://jfreechart.org

preloaded Sun SPOT libraries in particular the ANT binaries to deploy the program. The program running on the mote is written on Java and it reports the data values via the radio link to a program running on the host written in Java. The data values were logged and analyzed. Some analysis work done in Python and Matlab. Some screenshots of the interface are attached below showing the acceleration variations and the activities predicted. The actual readings which were characteristic of the activity are circled in black.

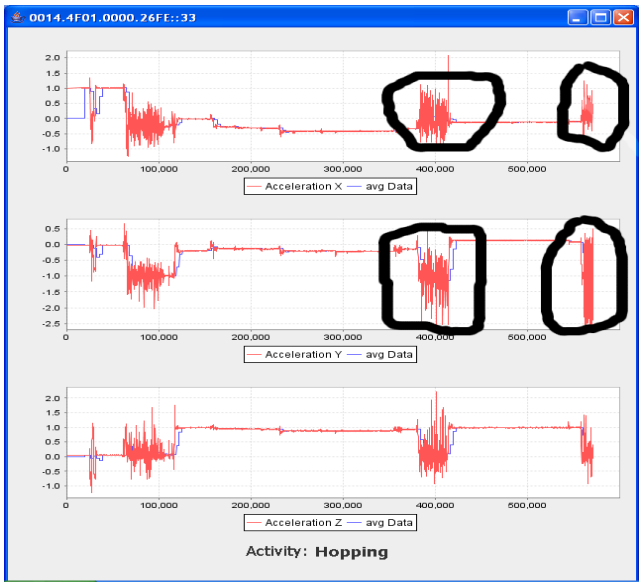

Figure 17: Interface showing acceleration variation for hopping activity

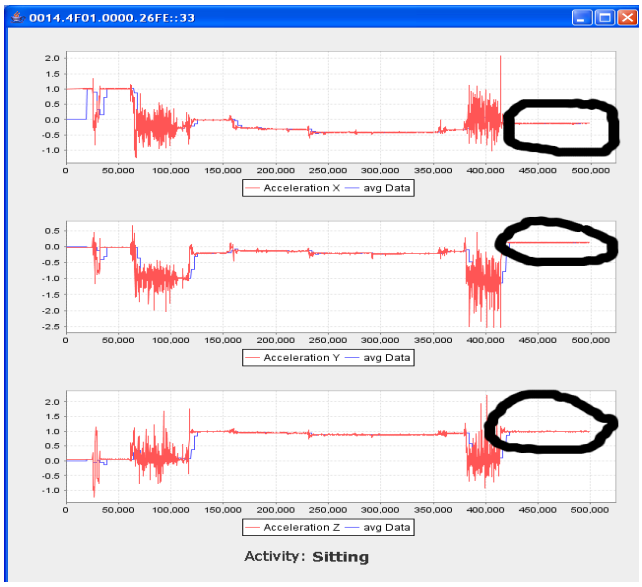

Figure 18: Interface showing acceleration variation for sitting activity

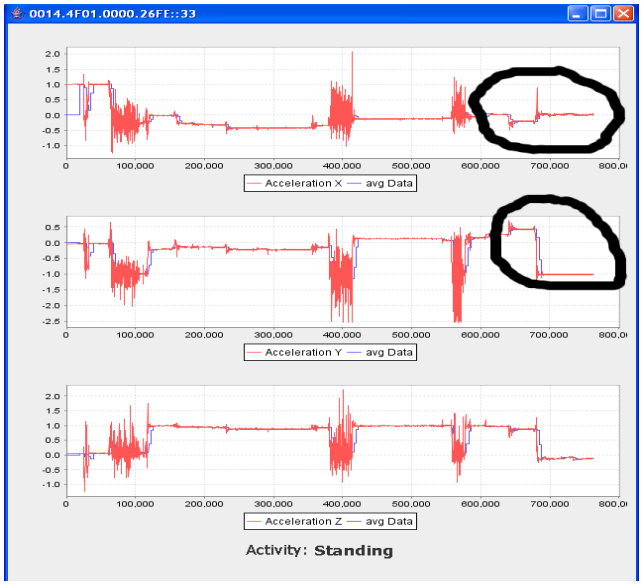

Figure 19: Interface showing acceleration variation for standing activity

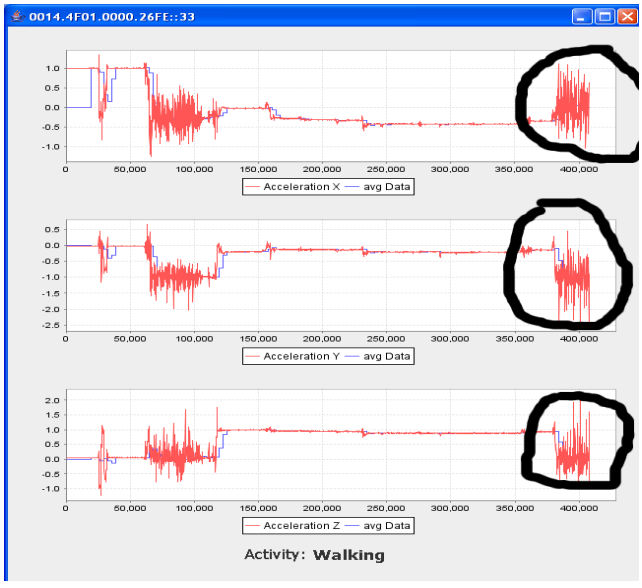

Figure 20: Interface showing acceleration variation for walking activity

8 Conclusions

We were able to successfully interface with 4 motes in parallel and are able to predict the activity performed by the 4 people wearing each of these motes respectively. The activities considered were Standing, Sitting, Running, Walking & Hopping. The rules considered can be extended to include any other activity.

References

[1] J.K. Aggarwal and M.S. Ryoo. Human activity analysis: A review. *ACM Comput. Surv.*, 43:16:1–16:43, April 2011.

[2] Pierluigi Casale, Oriol Pujol, and Petia Radeva. Human activity recognition from accelerometer data using a wearable device. In *Proceedings of the 5th Iberian conference on Pattern recognition and image analysis*, IbPRIA'11, pages 289–296, Berlin, Heidelberg, 2011. Springer-Verlag.

[3] Anthony Hoogs and A. G. Amitha Perera. Video activity recognition in the real world. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 3*, pages 1551–1554. AAAI Press, 2008.

[4] Adil Mehmood Khan. *Human Activity Recognition Using A Single Tri-axial Accelerometer*. PhD thesis, Kyung Hee University, Seoul, Korea, 2011.

[5] Asad Masood Khattak, A.M Khan, Young-Koo Lee, and Lee Sungyoung. Analyzing association rule mining and clustering on sales day data with xlminer and weka. *International Journal of Database Theory and Application*, June 2010.

[6] Agha Muhammad, Niklas Klein, Kristof Van Laerhoven, and Klaus David. A feature set evaluation for activity recognition with body-worn inertial sensors. In *Workshop on Interactive Human Behavior Analysis in Open or Public Spaces (InterHub) 2011*, Amsterdam, 11/2011 2011. Springer Verlag, Springer Verlag.

[7] Martha E. Pollack, Laura Brown, Dirk Colbry, Colleen E. McCarthy, Cheryl Orosz, Bart Peintner, Sailesh Ramakrishnan, and Ioannis Tsamardinos. Autominder: An intelligent cognitive orthotic system for people with memory impairment, 2003.

[8] Shinichi Takeuchi, Satoshi Tamura, and Satoru Hayamizu. Accelerometry-based classification of human activities using markov modeling. *Computational Intelligence and Neuroscience*, 2011:10, 2011.

[9] X Yang, A Dinh, and L Chen. *A wearable real-time fall detector based on Naive Bayes classifier*, page 14. IEEE, 2010.

[10] Xiuxin Yang, Anh Dinh, and Li Chen. A wearable real-time fall detector based on naive bayes classifier. In *Electrical and Computer Engineering (CCECE), 2010 23rd Canadian Conference on*, pages 1 –4, may 2010.

9 Appendix: Code

9.1 Java code: Data Collection, GUI interface & Prediction

We present below the set of Java files to collect the data from the motes, and predict the activity performed based on certain rules.

```
1
2
3
4
5
```

```
 6 package org.sunspotworld.demo;
 7
 8 import java.io.DataInputStream;
 9 import java.io.DataOutputStream;
10 import java.io.IOException;
11
12 import javax.microedition.io.Connector;
13
14 import com.sun.spot.io.j2me.radiostream.RadioOutputStream;
15 import com.sun.spot.io.j2me.radiostream.RadiostreamConnection;
16 import com.sun.spot.peripheral.NoRouteException;
17 import com.sun.spot.peripheral.TimeoutException;
18 import com.sun.spot.sensorboard.EDemoBoard;
19 import com.sun.spot.sensorboard.peripheral.IAccelerometer3D;
20
21 public class DataInputOutputStreamConnection {
22
23   private final int TIMEOUT = 3000;
24
25   private boolean connected = false;
26   private RadiostreamConnection conn = null;
27   private RadiostreamConnection connROS = null;
28   private DataInputStream dis = null;
29   private DataOutputStream dos = null;
30   private RadioOutputStream ros = null;
31
32
33         IAccelerometer3D accelerometer;
34         double accelerationX;
35         double accelerationY;
36         double accelerationZ;
37         double tiltX;
38         double tiltY;
39         double tiltZ;
40
41   public DataInputOutputStreamConnection() {
42   }
43
44   public void connect(String address,String ouraddress)
45         {
46                   if(ouraddress.equals("0014.4F01.0000.26FE"))
47                      PORT = 33;
48                   else if(ouraddress.equals("0014.4F01.0000.2AE6"))
49                      PORT = 43;
50                   else if(ouraddress.equals("0014.4F01.0000.2B44"))
51                      PORT = 53;
52                   else if(ouraddress.equals("0014.4F01.0000.27AA"))
53                      PORT = 63;
54                   else
55                       System.out.println("Error");
56
57
58
59                 System.out.println("Port = " + PORT);
60                 System.out.println("Address = " + address);
61                 connected = false;
62     try {
63                         System.out.println("Making connecetion open call");
64       conn.setTimeout(TIMEOUT);
65     } catch (Exception e1) {
66                       System.out.println("Error in connect()...1");
67       e1.printStackTrace();
```

```
68        conn = null;
69      }
70
71
72      String tmp = null;
73
74      try {
75        dis = (DataInputStream)conn.openDataInputStream();
76        dos = (DataOutputStream)conn.openDataOutputStream();
77      } catch (IOException e) {
78                      System.out.println("Error in connect()...2");
79        e.printStackTrace();
80      }
81
82      while(!connected) {
83        try {
84          dos.writeUTF("GO");
85                                      System.out.println("GO send to basestation");
86          dos.flush();
87        } catch (IOException e1) {
88          e1.printStackTrace();
89          System.out.println("Exception on writeUTF");
90
91
92        }
93
94        try {
95          tmp = dis.readUTF();
96                                      System.out.println("Received = " + tmp);
97        } catch (TimeoutException e) {
98                                  e.printStackTrace();
99          System.out.println("Timeout... other end is not responding");
100       } catch (IOException e1) {
101         e1.printStackTrace();
102         System.out.println("Exception on readUTF");
103       }
104
105       if(tmp != null) {
106         if(tmp.equalsIgnoreCase("GOT Message")) {
107           connected = true;
108           System.out.println("Connected to " + address);
109         } else {
110           connected = false;
111           System.out.println("NOT connected to " + address);
112         }
113       }
114     }
115   }
116
117   public void closeConnection() {
118     if(connected == false)
119       return;
120     connected = false;
121     try {
122       dos.close();
123       dis.close();
124       conn.close();
125     } catch (IOException e) {
126       e.printStackTrace();
127     }
128   }
129
```

```
130     public void send() throws IOException, NoRouteException
131     {
132         accelerometer = EDemoBoard.getInstance().getAccelerometer();
133         accelerationX = accelerometer.getAccelX();
134         accelerationY = accelerometer.getAccelY();
135         accelerationZ = accelerometer.getAccelZ();
136         tiltX = accelerometer.getTiltX();
137         tiltY = accelerometer.getTiltY();
138         tiltZ = accelerometer.getTiltZ();
139         dos.writeUTF((int)System.currentTimeMillis() + ":" + accelerationX + ":" +
                accelerationY +":" + accelerationZ + ":" + tiltX + ":" + tiltY + ":" + tiltZ);
140         dos.flush();
141     }
142
143     public String receive() {
144       String recv = null;
145       try {
146       recv = dis.readUTF();
147     } catch (IOException e) {
148       e.printStackTrace();
149       recv = "nothing received...";
150     }
151     return recv;
152     }
153
154     public void startSendingThread() {
155         Runnable r = new Runnable()
156         {
157             public void run()
158             {
159               conn.setTimeout(500);
160             while(connected)
161                         {
162               try {
163                                                 System.out.println("Sending ");
164             send();
165           } catch (NoRouteException e1) {
166                                                 System.out.println("Error in startSendingThread
                                                        111");
167             e1.printStackTrace();
168           } catch (IOException e1) {
169                                                 System.out.println("Error in startSendingThread
                                                        222");
170             e1.printStackTrace();
171           }
172                                           catch (Exception ex)
173                                           {
174                                               System.out.println("Error in startSendingThread
                                                        333");
175                                               ex.printStackTrace();
176                                           }
177
178               try {
179                 Thread.sleep(100);
180               } catch (InterruptedException e) {
181                 e.printStackTrace();
182               }
183             }
184           };
185         };
186         (new Thread(r)).start();
187     }
```

```
188 }
```

Listing 1: codes/tDataInputOutputStreamConnection.java

```
 1 package org.sunspotworld.demo;
 2
 3
 4 public class DisplayNew extends javax.swing.JFrame
 5 {
 6
 7     XYAreaNew g1,g2,g3;
 8
 9
10     public DisplayNew()
11     {
12
13         initComponents();
14
15         this.setSize(650,820);
16         g1=new XYAreaNew();
17         this.jPanel1.add(g1.Create_Chart(this.jPanel1.getWidth(), this.jPanel1.getHeight())
               );
18
19         g2=new XYAreaNew();
20         this.jPanel2.add(g2.Create_Chart(this.jPanel2.getWidth(), this.jPanel2.getHeight())
               );
21
22         g3=new XYAreaNew();
23         this.jPanel3.add(g3.Create_Chart(this.jPanel3.getWidth(), this.jPanel3.getHeight())
               );
24
25         g1.addDynamicDataSeries("Acceleration X");
26         g2.addDynamicDataSeries("Acceleration Y");
27         g3.addDynamicDataSeries("Acceleration Z");
28
29     }
30
31     private void initComponents() {
32
33         jPanel1 = new javax.swing.JPanel();
34         jPanel3 = new javax.swing.JPanel();
35         jPanel2 = new javax.swing.JPanel();
36         jLabel1 = new javax.swing.JLabel();
37         jLabel2 = new javax.swing.JLabel();
38
39         setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
40         getContentPane().setLayout(null);
41
42         jPanel1.setBorder(javax.swing.BorderFactory.createLineBorder(new java.awt.Color(0,
               0, 0)));
43         jPanel1.setLayout(null);
44         getContentPane().add(jPanel1);
45         jPanel1.setBounds(25, 25, 600, 200);
46
47         jPanel3.setBorder(javax.swing.BorderFactory.createLineBorder(new java.awt.Color(0,
               0, 0)));
48         jPanel3.setLayout(null);
49         getContentPane().add(jPanel3);
50         jPanel3.setBounds(25, 475, 600, 200);
51
52         jPanel2.setBorder(javax.swing.BorderFactory.createLineBorder(new java.awt.Color(0,
               0, 0)));
53         jPanel2.setLayout(null);
```

```
54          getContentPane().add(jPanel2);
55          jPanel2.setBounds(25, 250, 600, 200);
56
57          jLabel1.setText("Activity:");
58          getContentPane().add(jLabel1);
59          jLabel1.setBounds(240, 690, 70, 20);
60
61          jLabel2.setText("IDLE");
62          getContentPane().add(jLabel2);
63          jLabel2.setBounds(310, 690, 220, 22);
64
65          pack();
66
67      public static void main(String args[]) {
68          java.awt.EventQueue.invokeLater(new Runnable() {
69              public void run() {
70                  new DisplayNew().setVisible(true);
71              }
72          });
73      }
74
75      private javax.swing.JLabel jLabel1;
76      private javax.swing.JLabel jLabel2;
77      private javax.swing.JPanel jPanel1;
78      private javax.swing.JPanel jPanel2;
79      private javax.swing.JPanel jPanel3;
80
81
82      public void setLabel(String newText)
83      {
84          jLabel2.setText(newText);
85      }
86  }
```

Listing 2: codes/tDisplayNew.java

```
1  package org.sunspotworld.demo;
2
3  import com.sun.spot.io.j2me.radiogram.RadiogramConnection;
4  import com.sun.spot.peripheral.TimeoutException;
5  import java.io.IOException;
6  import javax.microedition.io.Connector;
7  import javax.microedition.io.Datagram;
8
9  public class HandShakeConnection {
10
11     private final int TIMEOUT = 2000;
12     private final int PORT = 100;
13
14     private boolean connected = false;
15     private RadiogramConnection conn = null;
16     private Datagram dg = null;
17     String ownAddress = null;
18
19     HandShakeConnection(String ownAddress1)
20     {
21         ownAddress = ownAddress1;
22     }
23
24     public void connect(String address)
25     {
26         connected = false;
27         try
```

```
28          {
29              System.out.println("Making connection open call  gram");
30
31              conn.setTimeout(TIMEOUT);
32          }
33          catch (Exception e1)
34          {
35              System.out.println("Error in connect()...1");
36              e1.printStackTrace();
37              conn = null;
38          }
39
40
41          String tmp = null;
42
43          try
44          {
45              dg = conn.newDatagram(conn.getMaximumLength());
46          }
47          catch (IOException e)
48          {
49              System.out.println("Error in connect()...2");
50                  e.printStackTrace();
51          }
52
53          while(!connected)
54          {
55              try
56              {
57                  dg.writeUTF(ownAddress);
58                  conn.send(dg);
59                  conn.receive(dg);
60                  String response = dg.readUTF();
61                  System.out.println ("Received: " + response);
62                  if(response.equals("Received"))
63                      connected = true;
64              }
65              catch (IOException e1)
66              {
67                      e1.printStackTrace();
68                      System.out.println("Exception on writeUTF  111");
69              }
70              catch (Exception e1)
71              {
72                      e1.printStackTrace();
73                      System.out.println("Exception on writeUTF   222 ");
74              }
75          }
76      }
77
78      public void closeConnection()
79      {
80          if(connected == false)
81              return;
82          connected = true;
83          try
84          {
85              conn.close();
86          }
87          catch (IOException e)
88          {
89              e.printStackTrace();
```

```
90          }
91      }
92  }
```

Listing 3: codes/tHandShakeConnection.java

```
 1  package org.sunspotworld.demo;
 2
 3  import com.sun.spot.io.j2me.radiostream.RadiostreamConnection;
 4  import com.sun.spot.util.Queue;
 5  import java.io.DataInputStream;
 6  import java.io.DataOutputStream;
 7  import java.util.Vector;
 8
 9  public class Spot {
10
11
12      Spot()
13      {
14          startTime = 0;
15          dataX_One = new Vector();
16          dataY_One = new Vector();
17          dataZ_One = new Vector();
18          dataX_Two = new Vector();
19          dataY_Two = new Vector();
20          dataZ_Two = new Vector();
21          averageX = 0;
22          averageY = 0;
23          averageZ = 0;
24          firstRecord = true;
25      }
26      public String address;
27      public int port;
28      public RadiostreamConnection conn;
29
30      public DataInputStream dis ;
31      public DataOutputStream dos ;
32      boolean active;
33
34      public double averageX;
35      public double averageY;
36      public double averageZ;
37      public double maxX;
38      public double minX;
39      public double maxY;
40      public double miny;
41      public double maxZ;
42      public double minZ;
43
44      Queue queue;
45
46
47      Vector dataX_One;
48      Vector dataY_One;
49      Vector dataZ_One;
50
51      Vector dataX_Two;
52      Vector dataY_Two;
53      Vector dataZ_Two;
54
55      double startTime;
56      double currentTime;
57
```

```
58
59
60      boolean firstRecord;
61
62      public int noOfMaxima;
63      public int noOfMinima;
64
65
66
67
68
69 }
```

Listing 4: codes/tSpot.java

```
 1 package org.sunspotworld.demo;
 2
 3 import com.sun.spot.peripheral.NoRouteException;
 4 import com.sun.spot.peripheral.Spot;
 5 import com.sun.spot.sensorboard.EDemoBoard;
 6 import com.sun.spot.sensorboard.peripheral.IAccelerometer3D;
 7 import com.sun.spot.sensorboard.peripheral.ITriColorLED;
 8 import com.sun.spot.peripheral.radio.IRadioPolicyManager;
 9 import com.sun.spot.io.j2me.radiostream.*;
10 import com.sun.spot.io.j2me.radiogram.*;
11 import com.sun.spot.peripheral.ChannelBusyException;
12 import com.sun.spot.util.*;
13
14 import java.io.*;
15 import javax.microedition.io.*;
16 import javax.microedition.midlet.MIDlet;
17 import javax.microedition.midlet.MIDletStateChangeException;
18
19 public class SunSpotApplication extends MIDlet {
20
21     ITriColorLED [] leds;
22     IAccelerometer3D accelerometer;
23     double accelerationX;
24     double accelerationY;
25     double accelerationZ;
26     double tiltX;
27     double tiltY;
28     double tiltZ;
29
30
31
32
33     private final String REMOTE_SPOT_ADDRESS = "0014.4F01.0000.298A";
34     private DataInputOutputStreamConnection rConnection = null;
35     private HandShakeConnection handShakeConn = null;
36
37
38
39     protected void startApp() throws MIDletStateChangeException
40     {
41         IEEEAddress ourAddr = new IEEEAddress(Spot.getInstance().getRadioPolicyManager().
               getIEEEAddress());
42         leds = EDemoBoard.getInstance().getLEDs();
43
44
45
46
47
```

```
48
49          String recv = null;
50          int iRecv = 0;
51          rConnection = new DataInputOutputStreamConnection();
52
53          System.out.println("I'm about to connect to the host application!");
54
55          rConnection.connect(REMOTE_SPOT_ADDRESS, ourAddr.toString());
56
57          System.out.println("I'm connected");
58
59      rConnection.startSendingThread();
60
61      while(true)
62                  {
63        recv = rConnection.receive();
64        System.out.println(recv);
65
66        System.out.println(iRecv);
67
68        try {
69          Thread.sleep(100);
70        } catch (InterruptedException e) {
71          e.printStackTrace();
72        }
73      }
74      }
75
76
77
78      protected void pauseApp() {
79      }
80
81      protected void destroyApp(boolean unconditional) throws MIDletStateChangeException {
82      }
83  }
```

Listing 5: codes/tSunSpotApplication.java

```
1  package org.sunspotworld.demo;
2
3
4  import com.sun.squawk.io.BufferedReader;
5  import com.sun.squawk.io.BufferedWriter;
6  import java.lang.*;
7  import java.io.*;
8  import java.util.Calendar;
9  import javax.microedition.io.*;
10 import javax.microedition.midlet.*;
11 import com.sun.spot.peripheral.*;
12 import com.sun.spot.peripheral.radio.*;
13 import com.sun.spot.io.j2me.radio.*;
14 import com.sun.spot.io.j2me.radiogram.*;
15 import com.sun.spot.io.j2me.radiostream.RadiostreamConnection;
16 import com.sun.spot.util.IEEEAddress;
17 import com.sun.spot.util.Utils;
18 import com.sun.squawk.util.StringTokenizer;
19 import javax.microedition.rms.RecordStore;
20 import javax.microedition.rms.RecordStoreException;
21 import java.util.Date;
22 import java.util.Timer;
23 import java.util.TimerTask;
24 import java.util.Vector;
```

```
25 public class SunSpotHostApplication
26 {
27     Spot spot;
28     DisplayNew DisplayNew;
29
30
31     public SunSpotHostApplication(String address, int port)
32     {
33         spot=new Spot();
34         spot.address = address;
35         spot.port = port;
36         DisplayNew=new DisplayNew();
37         DisplayNew.setVisible(true);
38     }
39
40     public void populateSpotDetails(int i)
41     {
42         if(i==0)
43         {
44             spot.address = "0014.4F01.0000.26FE";
45             spot.port = 33;
46         }
47         else
48         {
49             spot.address = "0014.4F01.0000.2AE6";
50             spot.port = 43;
51         }
52
53         DisplayNew.setTitle(spot.address +"::"+ spot.port);
54     }
55
56
57
58     public void runSimulation() throws IOException
59     {
60
61         System.out.println("");
62         System.out.println("");
63         System.out.println("");
64         System.out.println("");
65         System.out.println("");
66         System.out.println("");
67         DisplayNew.setTitle(spot.address +"::"+ spot.port);
68
69                 spot.address + ":" + spot.port);
70
71         spot.conn.setTimeout(2000);
72
73         spot.dis = spot.conn.openDataInputStream();
74         spot.dos = spot.conn.openDataOutputStream();
75
76
77         Timer timer=new Timer();
78         timer.schedule(new TimerTask()
79         {
80             public void run()
81             {
82                 try
83                 {
84                     String question = null;
85                     question = spot.dis.readUTF();
86                     if(question.length()>10)
```

```
87                      {
88                          StringTokenizer str=new StringTokenizer(question,":");
89
90                          double time=Double.parseDouble(str.nextToken());
91                          double ax=Double.parseDouble(str.nextToken());
92                          double ay=Double.parseDouble(str.nextToken());
93                          double az=Double.parseDouble(str.nextToken());
94
95                          if(spot.firstRecord)
96                          {
97                              System.out.println("First record");
98                              spot.startTime = time;
99                              spot.firstRecord = false;
100                         }
101
102                         spot.currentTime = time;
103
104                         double timeDifference = spot.currentTime - spot.startTime;
105
106
107                         spot.dataX_Two.addElement(Double.toString(ax));
108                         spot.dataY_Two.addElement(Double.toString(ay));
109                         spot.dataZ_Two.addElement(Double.toString(az));
110
111                         if((timeDifference > 3000))
112                         {
113                             System.out.println("timeDifference = " + timeDifference);
114                             spot.firstRecord = true;
115                             if(spot.dataX_One.size()>0)
116                             {
117                                 spot.averageX = calculateAverage(spot.dataX_One,spot.
                                    dataX_Two);
118                                 spot.averageY = calculateAverage(spot.dataY_One,spot.
                                    dataY_Two);
119                                 spot.averageZ = calculateAverage(spot.dataZ_One,spot.
                                    dataZ_Two);
120                                 double deviationScore=0;
121                                 double rangeScore=0;
122                                 double deviation=deviation(spot.dataX_One,spot.dataX_Two);
123                                 deviationScore += deviation;
124                                 double range=range(spot.dataX_One,spot.dataX_Two);
125                                 rangeScore += range;
126                                 System.out.println("deviation= "+deviation);
127                                 System.out.println("range= "+range);
128                                 BufferedWriter out1 = null;
129                                 FileWriter fileWriter = null;
130                                 File file = new File("E:\\ActivityData\\"+spot.address+".
                                    txt");
131                                 fileWriter = new FileWriter(file,true);
132                                 out1 = new BufferedWriter(fileWriter);
133                                 out1.write("spot="+spot.address+"\n");
134                                 out1.write("X:deviation="+deviation +"\n");
135                                 out1.write("X:range="+range+"\n");
136                                 deviation=deviation(spot.dataY_One,spot.dataY_Two);
137                                 deviationScore += deviation;
138                                 range=range(spot.dataY_One,spot.dataY_Two);
139                                 rangeScore += range;
140                                 out1.write("Y:deviation="+deviation +"\n");
141                                 out1.write("Y:range="+range +"\n");
142
143                                 deviation=deviation(spot.dataZ_One,spot.dataZ_Two);
144                                 deviationScore += deviation;
```

```
145                             range=deviation(spot.dataZ_One,spot.dataZ_Two);
146                             rangeScore += range;
147                             out1.write("Z:deviation="+deviation +"\n");
148                             out1.write("Z:range="+range +"\n\n\n\n\n");
149
150                             out1.close();
151
152                             String Activity[];
153                             Activity = new String[3];
154                             Activity[0] = "WALK";
155                             Activity[1] = "HOP";
156                             Activity[2] = "RUN";
157
158                             double score=rangeScore+deviationScore;
159
160                             double max_s=2.5;
161                             double min_w=1.5;
162                             double max_w = 6;
163                             double min_h = 4.6;
164                             double max_h = 11;
165                             double min_r=11;
166
167                             System.out.println("in x="+spot.averageX);
168                             System.out.println("in y="+spot.averageY);
169                             System.out.println("in z="+spot.averageZ);
170
171                             String activity="";
172                             String stat="Sitting";
173                             if (Math.abs(spot.averageY)>.6)
174                                 stat="Standing";
175
176                             if (score<min_w)
177                             {
178                                 activity=stat;
179                             }
180                             else if (score>min_w && score<max_s )
181                            {
182                                if (Math.abs(spot.averageZ)>.06)
183                                    activity="Walking";
184                                else
185                                  activity=(stat);
186                             }
187                             else if (score>max_s && score<min_h)
188                             {
189                                 activity="Walking";
190                             }
191                            else if (score<max_w && score>min_h)
192                            {
193                                if (deviation(spot.dataY_One,spot.dataY_Two)>.65)
194                                    activity="Hopping";
195                                else
196                                    activity = "Walking";
197                            }
198
199                            else if (score < min_r && score>max_w)
200                            {
201                                activity="Hopping";
202                            }
203                            else if (score>min_r)
204                            {
205                             activity=("Running");
206                           }
```

```
207
208                                 System.out.println("Activity = " +activity);
209                                 DisplayNew.setLabel(activity);
210
211                             }
212
213
214                             spot.dataX_One.removeAllElements();
215                             spot.dataY_One.removeAllElements();
216                             spot.dataZ_One.removeAllElements();
217
218                             for(int i=0;i<spot.dataX_Two.size();i++)
219                             {
220                                 spot.dataX_One.addElement(spot.dataX_Two.elementAt(i));
221                                 spot.dataY_One.addElement(spot.dataY_Two.elementAt(i));
222                                 spot.dataZ_One.addElement(spot.dataZ_Two.elementAt(i));
223                             }
224                             spot.dataX_Two.removeAllElements();
225                             spot.dataY_Two.removeAllElements();
226                             spot.dataZ_Two.removeAllElements();
227                         }
228
229                         DisplayNew.g1.updateDynamicData(time, ax, spot.averageX);
230                         DisplayNew.g2.updateDynamicData(time, ay, spot.averageY);
231                         DisplayNew.g3.updateDynamicData(time, az, spot.averageZ);
232                     }
233
234
235                     spot.dos.writeUTF("GOT Message");
236
237
238                     spot.dos.flush();
239                 }
240
241                 catch (NoRouteException e)
242                 {
243                     System.out.println ("No route to spot");
244                     e.printStackTrace();
245                 }
246                 catch(IOException ex)
247                 {
248                     System.out.println("Nothing to read from " + spot.address);
249                     ex.printStackTrace();
250                 }
251             }
252         },100,100);
253
254
255
256
257
258
259     }
260
261 public double calculateAverage(Vector one, Vector two)
262 {
263     double average = 0;
264     double sum = 0;
265     for(int i=0;i<one.size();i++)
266     {
267         double x = Double.parseDouble((String)one.elementAt(i));
268         sum = sum + x;
```

```
269     }
270      for(int i=0;i<two.size();i++)
271     {
272         double x = Double.parseDouble((String)two.elementAt(i));
273         sum = sum + x;
274 
275     }
276 
277     for(int i=0;i<one.size();i++)
278     {
279         double x = Double.parseDouble((String)one.elementAt(i));
280         sum = sum + x*x;
281     }
282     for(int i=0;i<two.size();i++)
283     {
284         double x = Double.parseDouble((String)two.elementAt(i));
285         sum = sum + x*x;
286 
287     average = sum/((double)(one.size() + two.size()));
288 
289     return average;
290 }
291 
292 public double range(Vector one, Vector two)
293 {
294     double max = -222;
295     double min = 222;
296     for(int i=0;i<one.size();i++)
297     {
298         double x = Double.parseDouble((String)one.elementAt(i));
299         if (x>max)
300             max=x;
301         if (x<min)
302             min=x;
303     }
304      for(int i=0;i<two.size();i++)
305     {
306         double x = Double.parseDouble((String)two.elementAt(i));
307         if (x<min)
308             min=x;
309 
310           if (x>max)
311             max=x;
312     }
313      return max-min;
314 
315     for(int i=0;i<one.size();i++)
316     {
317         double x = Double.parseDouble((String)one.elementAt(i));
318         sum = sum + x*x;
319     }
320     for(int i=0;i<two.size();i++)
321     {
322         double x = Double.parseDouble((String)two.elementAt(i));
323         sum = sum + x*x;
324 
325 
326 }
327 
328 public double calculatesquareAverage(Vector one, Vector two)
329 {
330     double average = 0;
```

```
331     double sum = 0;
332     for(int i=0;i<one.size();i++)
333     {
334         double x = Double.parseDouble((String)one.elementAt(i));
335         sum = sum + x*x;
336     }
337      for(int i=0;i<two.size();i++)
338     {
339         double x = Double.parseDouble((String)two.elementAt(i));
340         sum = sum + x*x;
341
342     }
343
344     average = sum/((double)(one.size() + two.size()));
345
346     return average;
347 }
348
349 public int calculatepeaks(Vector one, Vector two)
350 {
351     double average = calculateAverage(one,two);
352     int no_peaks=0;
353
354     for(int i=0;i<one.size();i++)
355     {
356         double x = Double.parseDouble((String)one.elementAt(i));
357         if (x>average)
358             no_peaks++;
359     }
360      for(int i=0;i<two.size();i++)
361     {
362         double x = Double.parseDouble((String)two.elementAt(i));
363         if (x>average)
364             no_peaks++;
365
366     }
367
368
369     return no_peaks;
370 }
371
372
373 public double deviation(Vector one, Vector two)
374 {
375     double dev=0;
376     dev=calculatesquareAverage(one,two)-calculateAverage(one,two)*calculateAverage(one,two)
        ;
377     return dev;
378 }
379
380     public static void main(String[] args) throws Exception
381     {
382
383         Vector sunSpotHostApps = new Vector();
384
385
386         SunSpotHostApplication app1 = new SunSpotHostApplication("0014.4F01.0000.26FE",33);
387        SunSpotHostApplication app2 = new SunSpotHostApplication("0014.4F01.0000.2AE6",43);
388         SunSpotHostApplication app3 = new SunSpotHostApplication("0014.4F01.0000.2B44",53);
389        SunSpotHostApplication app4 = new SunSpotHostApplication("0014.4F01.0000.27AA",63);
390
391        sunSpotHostApps.addElement(app1);
```

```
392
393         for(int i=0;i<sunSpotHostApps.size();i++)
394         {
395
396             try
397             {
398                 ((SunSpotHostApplication)sunSpotHostApps.elementAt(i)).runSimulation();
399             }
400             catch(Exception ex)
401             {
402                 ex.printStackTrace();
403                 System.out.println("Error 111");
404             }
405         }
406     }
407 }
```

Listing 6: codes/tSunSpotHostApplication.java

```
 1 package org.sunspotworld.demo;
 2
 3
 4
 5
 6
 7
 8
 9
10
11 import java.awt.Color;
12
13 import java.util.Vector;
14 import org.jfree.chart.ChartPanel;
15 import org.jfree.chart.JFreeChart;
16 import org.jfree.chart.axis.NumberAxis;
17 import org.jfree.chart.plot.XYPlot;
18 import org.jfree.chart.renderer.xy.StandardXYItemRenderer;
19 import org.jfree.chart.renderer.xy.XYDotRenderer;
20 import org.jfree.chart.renderer.xy.XYItemRenderer;
21 import org.jfree.data.xy.XYDataset;
22 import org.jfree.data.xy.XYSeries;
23 import org.jfree.data.xy.XYSeriesCollection;
24
25 public class XYAreaNew {
26
27
28     NumberAxis rangeAxis1;
29     NumberAxis domainAxis1;
30     XYPlot plot;
31     XYSeriesCollection xy_series_collection;
32     public XYSeries[] grph_series;
33     public XYSeries dynamicData;
34     XYDataset xy_dataset;
35     JFreeChart chart;
36     XYDotRenderer XYDotRend;
37     XYItemRenderer StdXYRend;
38     public boolean first=true;
39     double initialTime=0;
40     XYSeries avgData;
41     double avgValue;
42
43
44     public XYAreaNew() {
```

```
45          rangeAxis1 = new NumberAxis();
46          domainAxis1 = new NumberAxis();
47          xy_series_collection = new XYSeriesCollection();
48          StdXYRend = new StandardXYItemRenderer();
49          XYDotRend=new XYDotRenderer();
50          XYDotRend.setDotHeight(2);
51          XYDotRend.setDotWidth(2);
52          plot = new XYPlot(xy_series_collection, domainAxis1, rangeAxis1, StdXYRend);
53      }
54      public void createDataSeries(int nSeires, String seriesTitle[]){
55          grph_series=new XYSeries[nSeires];
56          for(int i=0;i<nSeires;i++)
57          {
58              grph_series[i]=new XYSeries(seriesTitle[i],false);
59              xy_series_collection.addSeries(grph_series[i]);
60          }
61
62      }
63      public void addDynamicDataSeries(String seriesName){
64          dynamicData=new XYSeries(seriesName,false);
65          avgData=new XYSeries("avg Data",false);
66          xy_series_collection.addSeries(dynamicData);
67          xy_series_collection.addSeries(avgData);
68      }
69      public void updateDynamicData(double xValue, double yValue, double average)
70      {
71
72          if(first){
73              first=false;
74              initialTime=xValue;
75          }
76          dynamicData.add(xValue-initialTime,yValue);
77          avgValue=average;
78          if(dynamicData.getItemCount()>0){
79              avgData.add(xValue-initialTime, avgValue);
80          }
81      }
82
83      public void setGraphTitle(String title){
84              chart.setTitle(title);
85         }
86      public void resetGraph(){
87         xy_series_collection.removeAllSeries();
88
89     }
90      public void addGraphSeries(XYSeries graphData){
91
92          chart.getXYPlot().setRenderer(StdXYRend);
93          xy_series_collection.addSeries(graphData);
94
95      }
96      public void setGraphRenderer(boolean dotOrStdXY){
97           if (dotOrStdXY==true){
98               chart.getXYPlot().setRenderer(StdXYRend);
99           }else{
100              chart.getXYPlot().setRenderer(XYDotRend);
101          }
102
103
104      }
105     public void setXYLabels(String xlabel, String ylabel){
106        chart.getXYPlot().getDomainAxis().setLabel(xlabel);
```

```
107         chart.getXYPlot().getRangeAxis().setLabel(ylabel);
108
109     }
110      public void setGraphYAxisRange(double min_y, double max_y){
111          plot.getRangeAxis().setAutoRange(false);
112          plot.getRangeAxis().setRange(min_y, max_y);
113          plot.getRangeAxis().setAutoRangeMinimumSize(0.001);
114      }
115      public void setGraphXAxixRange(double min_x, double max_x){
116          plot.getDomainAxis().setAutoRange(false);
117          plot.getDomainAxis().setRange(min_x, max_x);
118          plot.getDomainAxis().setAutoRangeMinimumSize(0.010);
119      }
120      public void setGraphBackGroundColor(Color clr){
121          plot.setBackgroundPaint(clr);
122      }
123      public ChartPanel Create_Chart(int width, int height) {
124          chart = new JFreeChart("", JFreeChart.DEFAULT_TITLE_FONT, plot, true);
125          ChartPanel chart_pnl = new ChartPanel(chart);
126          chart_pnl.setSize(width, height);
127          chart_pnl.setBackground(new Color(255, 0, 0));
128          return chart_pnl;
129      }
130      void Update_DataSet(int no_of_series, String[] series_title, int no_of_rows, double[][]
             radar_data) {
131          try {
132
133
134              for (int s = 1; s <= no_of_series; s++) {
135                  for (int r = 1; r <= no_of_rows; r++) {
136                      grph_series[s].add(radar_data[r][0], radar_data[r][s]);
137                  }
138              }
139
140              xy_dataset = xy_series_collection;
141          }
142
143          catch (Exception e) {
144              System.out.println(e.getMessage());
145          }
146      }
147      void Update_DateSet1(double x, double y1, double y2, double y3, double y4) {
148          grph_series[1].add(x, y1);
149          grph_series[2].add(x, y2);
150          grph_series[3].add(x, y3);
151          grph_series[4].add(x, y4);
152      }
153
154 }
```

Listing 7: codes/tXYAreaNew.java

9.2 Training Analysis

Some additional code was written in Python/Matlab which is skipped for the sake of brevity.