USER STATE TRACKING USING SMARTPHONES

by

Mehmet Sönercan

Sinan Dinçer

Submitted to the Department of Computer Engineering
in partial fulfilment of the requirements
for the degree of
Bachelor of Science
in
Computer Engineering

Boğaziçi University
June 2011
Contents

ABSTRACT........................................................................................................................................3

1. INTRODUCTION ..................................................................................................................................4

2. DATA COLLECTION ................................................................................................................................5
  2.1 DATA COLLECTION APPLICATION ...................................................................................................6
    2.1.1 Sensor Manager .............................................................................................................................7
    2.1.2 Location Manager ...........................................................................................................................7
    2.1.3 FileManager ..................................................................................................................................8
    2.1.3 Activity .........................................................................................................................................9
  2.2 DATA COLLECTION EXPERIMENTS .................................................................................................11

3. FEATURE EXTRACTION ........................................................................................................................13

4. ACTIVITY RECOGNITION ....................................................................................................................17
  4.1 ACTIVITY RECOGNITION APPLICATION .........................................................................................17

5. EXPERIMENTS .........................................................................................................................................21

6. CONCLUSION AND FUTURE WORK ....................................................................................................22

REFERENCES: .............................................................................................................................................23
ABSTRACT

Project Name   : User State Tracking Using Smart Phones

Project Team  : Mehmet Sönercan, Sinan Dinçer

Term          : 2010/11 II. Semester

Keywords      : Activity Recognition, Android, Accelerometer, Smartphone

Summary       : In this project we developed a system that uses phone-based sensors to perform activity recognition. A data collection system to track user activities from the available sensors on smart phones is implemented. We have evaluated the values coming from accelerometer. The data collection system will provide the necessary information to track user activities such as walking, sitting, standing, running and riding a bike. The system is implemented on Android smart phones and achieved an accuracy rate of %86.32.
1. INTRODUCTION

With high technology developing in recent years, smartphones have become the new generation of cell phones. Smartphones include many sensors such as cameras, microphones, light sensors, temperature sensors, acceleration sensors (accelerometers), GPS, orientation sensors, magnetic field sensors, pressure sensors, proximity sensors. The availability of these sensors created new exciting opportunities for sensor-based applications in the market.

Our project is a sensor-based application which is implemented for Android, which is open-source, easy to implement and expected to become dominant OS in the smart phones market. We examine accelerometer sensor to identify and track the physical activities that a user is performing such as walking, running, riding a bike, are sitting, standing, traveling on a vehicle. This task is called activity recognition. Activity recognition can be achieved by analyzing the data coming from the accelerometer and GPS since each different activity results in different accelerometer and GPS data series.

This project is important since the activity recognition model lets us to get useful information about the daily routines of millions of users just by having them carry their smartphones in their pockets. It can also be a base for many applications. For example, we can create daily activity report for a user which shows how active he is during the day and e-mail the report to the user. It can show if the user is doing enough exercise or not so that the user can be encouraged to do more exercise. Another example could be changing the phone settings during a particular activity. When a user is running, the incoming calls can be redirected to the voice mail service.

![Diagram](image)

Figure 1.1: Block diagram of the phases
2. DATA COLLECTION

We implemented our User State Tracking System on Android Platform. Android is an open source mobile operating system which is based on Linux Kernel. Its SDK is a plug-in that has been installed into Eclipse.

The Android software development kit (SDK) includes a comprehensive set of development tools. These include a debugger, libraries, a handset emulator, documentation, sample code, and tutorials.

Android SDK has a Java Framework and a powerful API support for the hardware embedded on smartphones. Figure 1.2 explains these layers:

![Android OS Layers](image)

Figure 1.2: Android OS Layers

The first phase of the project is to collect training data. For our purposes, we need to use data coming from acceleration, orientation sensor, also from location services (GPS, Network).

Accelerometer in smart phones is returning 3 current acceleration values in the units of \( \text{m/s}^2 \) along the \( x \), \( y \), and \( z \) axes.
In figure 1.3 mapping of directional acceleration axes in relation to the device at rest is depicted.

![Acceleration axes](image)

**Figure 1.3: Acceleration axes**

**X-axis (lateral):** Sideways acceleration (left to right) for which positive values represent the movements to the right whereas negative ones represent to the left.

**Y-axis (longitudinal):** Forward and backward acceleration for which positive values represent forward whereas negative values represent backward.

**Z-axis (vertical):** Upward or downward acceleration for which positive represents movements such as the device being lifted.

### 2.1 Data Collection Application

We have implemented an android application for collecting data from the available sensors in a smart phone. This application accesses the hardware components via hardware package provided by Android SDK. It simply works in the way of accessing sensor components and saving the collected values of these sensors in a csv file titled with the corresponding time and date of when the application is first started under the file *HumanTracing* in the SD card.

To access to the hardware components of the device such as accelerometer, orientation sensor, and GPS we need to use related Application Framework elements shown in the figure 1.2. To access to the location based services we need to create a *LocationManager* class object whereas for other sensors we need to create a *SensorManager* class object. To listen for the changes of values in the sensors and location service, Android SDK provides *SensorEventListener* and *LocationListener* interfaces, respectively.
Addition to these classes provided by Android SDK, we have implemented a *FileManager* class which makes dealing with file operations easier and usable. *LocationManager, SensorManager* and *FileManager* classes in the Application Framework layer are explained with their necessary fields and methods are explained in the following sections.

### 2.1.1 Sensor Manager

This class provides access the sensors embedded into a smart phone. This class can benefit when the application aims to access sensors such as orientation, proximity, accelerometer and temperature.

Only two of the functions related to this class is explained:

- public Sensor getDefaultSensor (int type)
  - Returns a Sensor class object given a type, such as the constant `SENSOR.TYPE_ACCELEROMETER`. These constant values are used while differentiating the types of the sensors in the same code snippet.

- public boolean registerListener (SensorEventListener listener, Sensor sensor, int rate)
  - Registers listener object to the given sensor with the rate sensor changes delivered at. During our data collection, we registered a sensor object to the accelerometer listener object with a sample rate at the rate of 24 samples/second.

### 2.1.2 Location Manager

This class provides access to the system location services. These services allow applications to obtain periodic updates of the device's geographical location, or to fire an application-specified Intent when the device enters the proximity of a given geographical location. In data collecting application, it provides periodic updates of the location.

The application uses Location Manager to get the location data updates of the user in data collection phase. The related code is presented as follows:

```java
criteria=new Criteria();
criteria.setAltitudeRequired(true);
criteria.setAccuracy(Criteria.ACCURACY_FINE);
criteria.setPowerRequirement(Criteria.POWER_LOW);
```
provider=locationManager.getBestProvider(criteria, true);
locationManager.requestLocationUpdates(provider,
MINIMUM_TIME_BETWEEN_UPDATES,MINIMUM_DISTANCE_CHANGE_FOR_UPDATES,
locListener);

To get the finest accuracy with the lowest power consumption we needed a criteria object. LocationManager object returns the best provider according to the given parameters of Criteria object.

Methods:

- public void requestLocationUpdates (String provider, long minTime, float minDistance, LocationListener listener)

  Registers the current activity to be notified periodically by the named provider. Periodically, the supplied LocationListener will be called with the current Location or with status updates. At least in minTime interval with minDistance change, LocationListener will be called.

2.1.3 FileManager

FileManager is a public class implemented to create a file object which provides a constructor with desired directory and filename, write to a file object and read from that file object. The FileManager class is modular, with sending an empty string as the header parameter; it is possible to create any kind of files.

Methods:

- public FileManager (String directory, String name, String header)

  The constructor gets three parameters, the directory, the filename and the header. It takes the directory of where the file is desired to be created. If this directory doesn’t exist, the constructor creates the necessary folders in the directory string. By using the name parameter, it creates the file with the desired filename under the desired directory, and if there is a sent header it writes this header string into the file then completes the construction of the file. A header parameter exists because the application creates csv files to store collected data. After now, the application can access this file by using only the object created.
• **public boolean** writeToFile(String toBeWritten)

  This function only takes one argument, the desired string to be written into the file object. In the function, it creates a FileWriter object and by using it, appends the string sent as the parameter at the end of the file. It returns true if the writing is performed, returns false if any exception occurred.

• **public boolean** isDirectory(String folderDirectory)

  This function checks if the given directory exists. It returns false if there is not such a directory.

• **public boolean** createDirectory(String folderDirectory)

  This function creates the directory of the given string. It returns true if the performing of creation is successful.

• **public boolean** fileExist(String fileDirectory)

  It checks if the given file directory is a file. If it is not, it returns false.

• **public boolean** createFile(String fileDirectory)

  This function creates the file with the given file directory. The fileDirectory here is the composition of folderDirectory and filename given to the constructor as parameters.

2.1.3 Activity

Activity class takes care of creating a window for the application in which the developer can place its UI with setContentView(View). We use activity lifecycle for designing applications user interface and getting some inputs from the user. Activity class has three important methods needs to be overridden. These methods are:

• **public void** onCreate()

• **public void** onResume()

• **public void** onStop()
The user interface for DATA COLLECTION application is quite simple. It consists of a dropdown menu, start button and a progress bar which is only active when the system is collecting and saving data.

The dropdown menu is a Spinner object which is provided by android.widget.Spinner class. Additionally there is a button which starts and stops the data collection event. This button is provided by android.widget.Button class. Lastly, we added a progress bar which is shown on active state only if the application is collecting and saving data. This progress bar is an object of android.widget.ProgressBar class.

![Figure 2.1: Data Collection UI](image)

Figure 2.1 shows the interface of the DATA COLLECTION application. When the application is started, the first picture is seen. The walking state is selected as default. The user can select any other state of activity from the list seen in the middle picture. After choosing an activity and pressing the start button, this button turns out a stop button; a progress bar appears and does not disappear until the stop button pressed. Additionally, the pop up messages are shown at every time the data saving starts or stops.

This application is developed for the purpose of collecting the output values of three sensors embedded in a smartphone using Android OS. Its output is a csv file with title of the date
and current time of when the application is started first, under the folder HumanTracking in SD card.

### 2.2 Data Collection Experiments

After Data Collection Application is developed, we started collecting data. There are two devices available to use in the development and data collection phases. These are Samsung Galaxy Tab and HTC G2 Touch. They both operate with Android OS, however while Galaxy Tab uses Android version 2.3, HTC uses version 1.5.

The data collection application is used with HTC and Samsung Galaxy Tab devices in data collection phase. The data collecting people are two male students who are in mid-twenties and have average height.

The output format of the Data Collection is a csv file titled with the current date and time of starting the application. The first row of this file is the header part.

The header of the output file is:

- **Time, Action, OrX, OrY, OrZ, AccX, AccY, AccZ, GpsAcc, GpsLat, GpsLong, GpsAlt, GpsSpeed**

  **Time:** Timestamp in the format of 29_Apr_2011_14_15_47  
  **Action:** The current activity selected from the list  
  **OrX, OrY, OrZ:** x, y, z values of orientation sensor  
  **AccX, AccY, AccZ:** x, y, z values of accelerometer  
  **GpsAcc:** Gps Accuracy of the GPS in meters  
  **GpsLat:** Gps Latitude value  
  **GpsLong:** Gps Longitude value  
  **GpsAlt:** Gps Altitude value  
  **GpsSpeed:** Current speed taken from GPS

We collect all the data on the field of daily routines. The smartphones are generally carried in the pockets since their dimensions allow. The case for Galaxy Tab is different, because it has larger dimensions compared to other devices. This fact cannot be underestimated. The article [1] questioning the carrying places of the mobile phones referenced as phone context problem concludes that the carrying place results in significant differences of the output values of sensors. We considered the application will be mostly used by the pocket-sized smartphones.
and it must be assumed that all data are collected when the device is carried in the pocket of pants. So, we collected all the training data with HTC smart phone.

In data sample collection, we collected data of all user states during daily/ordinary human activities. We collected the samples while the users were standing, sitting on a couch, walking on the streets, jogging on a park, travelling on mass transportation bus and an intercity bus.

![Figure 2.2 – A screenshot of the collected data](image)

In figure 2.2, the format of a collected data file can be seen. The data is written with the timestamp precision in seconds. At every second, application stores 24 samples of sensor values. That means we collect a sample every 41.66 milliseconds.

A data row consists of thirteen entries. We put the label action for the current activity of the user. It is directly copied from the selected user state from the dropdown menu on the user interface. The next three entries -OrX, OrY, OrZ- show the values of the orientation sensor at that time. AccX, AccY, AccZ entries are the values of accelerometer sensor. The last five entries are the values gathered from the GPS sensor.

We also get the current velocity from the GPS sensor. There are two ways of gathering the velocity of the user. First we can get this value from the getSpeed() function which is provided by the Location Manager class. Secondly, it is possible to calculate this value by dividing the horizontal distance between any two chosen points by the elapsed for the moving between these two points. We preferred the first one in this sampling to find out whether it gives reliable results for the selected activity.
3. FEATURE EXTRACTION

After usage of data collection application for each activity, it is seen that the accuracy of GPS may be too low because of some physical barriers (buildings, rainy weather). The accuracy of the position GPS returning can even take values such as 50 meters. This means the position returned can be within a circle of 50 meters centered at the current location. It’s also possible to use Android’s LocationManager class’ getSpeed function returning the speed of the device. However, in the data collection phase, it returns same values for sitting and running which can’t be helpful to differentiate the activities. Battery usage of GPS for activity recognition background service running always is another issue.

Orientation Sensor gives ideas about the device’s orientation (North, East, South, etc.). It has been noted that it’s not really helpful what direction a user is walking or running after some data collection and examination on the Excel.

After these factors are taken into account, GPS and orientation sensor usages are stopped to recognize different activities. On the other hand, the leg movements with different speeds cause accelerometer to give help during the recognition. So, it’s decided to focus only on the data from the accelerometer. To be able to recognize the activities continuously, a sliding window method is used to divide time series signals into smaller sequences. Window size is picked to be one second since it’s possible to get information for that particular activity in one second. A one second window overlapping with the next window for a half of a second is used. In a previous work\[2\] 50% overlapping is found out to be sufficient to classify activities.

For the recognition system, different features were calculated from one second windows to compress the information of interest in the data. The calculated features are shown in Table 3.1.
### Features within a window of the size 26 samples

<table>
<thead>
<tr>
<th>Description</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean of Accelerometer X-, Y-, Z-values</td>
<td>$\mu_x = \frac{1}{26} \sum_{i=1}^{26} x_i$, $\mu_y = \frac{1}{26} \sum_{i=1}^{26} y_i$, $\mu_z = \frac{1}{26} \sum_{i=1}^{26} z_i$</td>
</tr>
<tr>
<td>Standard Deviation of Accelerometer X-, Y-, Z-values</td>
<td>$s_x = \sqrt{\frac{\sum_{i=1}^{26}(x_i-\mu_x)^2}{25}}$, $s_y = \sqrt{\frac{\sum_{i=1}^{26}(y_i-\mu_y)^2}{25}}$, $s_z = \sqrt{\frac{\sum_{i=1}^{26}(z_i-\mu_z)^2}{25}}$</td>
</tr>
<tr>
<td>Minimum of each Accelerometer X, Y, Z values within the window</td>
<td>Minimum of the 26 window samples of X, Y, Z values of the accelerometer</td>
</tr>
<tr>
<td>Maximum of each Accelerometer X, Y, Z values within the window</td>
<td>Maximum of the 26 window samples of X, Y, Z values of the accelerometer</td>
</tr>
<tr>
<td>Mean of resultant accelerometer values, $\text{acc} = \sqrt{x^2 + y^2 + z^2}$</td>
<td>$\mu_{\text{acc}} = \frac{1}{26} \sum_{i=1}^{26} \text{acc}_i$</td>
</tr>
<tr>
<td>Standard deviation of resultant accelerometer values</td>
<td>$s_{\text{acc}} = \sqrt{\frac{\sum_{i=1}^{26}(\text{acc}<em>i-\mu</em>{\text{acc}})^2}{25}}$</td>
</tr>
<tr>
<td>Energy of resultant accelerometer values</td>
<td>$\text{Energy} = \frac{1}{26} \sum_{k=1}^{26}</td>
</tr>
<tr>
<td>Correlation coefficients of X-Y, X-Z, Y-Z</td>
<td>$r_{xy} = \frac{\sum_{i=1}^{26}(x_i-\mu_x)(y_i-\mu_y)}{25 s_x s_y}$, $r_{xz} = \frac{\sum_{i=1}^{26}(x_i-\mu_x)(z_i-\mu_z)}{25 s_x s_z}$, $r_{yz} = \frac{\sum_{i=1}^{26}(y_i-\mu_y)(z_i-\mu_z)}{25 s_y s_z}$</td>
</tr>
</tbody>
</table>

Table 3.1

Totally, 20 features were extracted. This extraction is done inside an MATLAB m-file. The CSV file created as the output of data collection application is the input of this m-file. In the code block of m-file, MATLAB’s built-in functions `mean`, `std`, `fft (calculates discrete Fourier Transform)`, `max`, `min`, `xcorr (calculates correlation coefficients of X-Y, X-Z, Y-Z)` are used to extract the features shown in the Table 3.1. Features for each window are extracted for every iteration of a loop. Each feature is kept as a vector. At the end of the extraction loop, these
vectors are combined to form a matrix. This matrix is given as an input to MATLAB’s built-in CSV file writing method `csvwrite`. This method creates a CSV file consisting of the features as columns. Then, it makes analyzing data easier. Figure 3.1 shows the format of this CSV file. Each column represents a feature in Figure 3.1 and the values in each row represent a window. The values of each row are calculated in each iteration of the loop in the m-file.

|   | A   | B     | C     | D     | E     | F     | G     | H     | I     | J     | K     | L     | M     | N     | O     | P     | Q     | R     | S     | T     |
|---|-----|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 1 | Max X | Min X | Avg X | Std X | Max Y | Min Y | Avg Y | Std Y | Max 2 | Min 2 | Avg 2 | Std 2 | Max Acc | Min Acc | Avg Acc | Std Acc | X | Y | Corr X | Y | | Corr Y | Corr Energy |
| 2 | 31.856 | -5.8884 | 1.0016 | 4.3372 | 0.59429 | -17.666 | -7.4488 | 3.6649 | 10.578 | -3.5499 | 3.5743 | 4.3004 | 19.591 | 3.0622 | 10.558 | 3.2638 | 0.21483 | 0.26305 | 0.31298 | 0.10173 |
| 7 | 19.532 | -14.601 | 4.0846 | 10.085 | 8.7715 | -19.613 | -7.0658 | 8.9298 | 12.299 | -10.079 | 0.01493 | 5.6419 | 29.206 | 2.521 | 14.954 | 7.2518 | -0.61568 | 0.037598 | 0.19794 | 116.31 |

Figure 3.1

In an input file with the feature information of all the activities’ mixed together, it was possible to get some insights. Figure 3.2 gives some insights. For instance, the average of the resultant acceleration values in each window for each activity takes values in the range of certain threshold values.

![Figure 3.2](image-url)
In Figure 3.2, regions 1, 3, 9 represent biking; 4, 6, 8 represent walking whereas 2, 5, 7 represent running. According to this figure, the range of the values for the resultant average acceleration data for walking is less than 13 and greater than 10.4 m/s\(^2\). The lower limit of these values for running is 13. The values of the resultant average acceleration data for sitting and sitting are shown in figure 3.3.

![Figure 3.3]

Figure 3.3

Figure 3.2 and Figure 3.3 show that standing average acceleration data values for standing may also be within the range values for walking. Another range intersection happens between sitting and biking because of their resultant average acceleration values. However, the standard deviations of the resultant acceleration data within a window for walking and standing, for biking and sitting differ which is represented in Figure 3.4.

![Figure 3.4]

Figure 3.4
In Figure 3.4, standard deviation of the resultant acceleration data within a window for *standing* is between 0.1 and 1 whereas it’s between 3 and 5 for *walking*. While this value is really small, less than 0.1 for *sitting*, it is greater than 1 for *biking*.

The figures 3.2, 3.3, 3.4 give some insights about the threshold values of standard deviation of the resultant acceleration data and average of the resultant acceleration data within a window.

Combined with the resultant average acceleration values, these standard deviation values may create a decision tree which is shown in the following section.

### 4. ACTIVITY RECOGNITION

After collecting data and extracting some features from these data by applying mathematical operations mentioned in feature extraction section, we developed a classification technique for activity recognition. The values coming from accelerometer sensor are processed and the result of these operations provided many threshold values for creating a decision tree and reaching conclusions in activity recognition phase. The activity recognition in this project is based on the values of accelerometer sensor. The values coming from this sensor are processed and the decision is made according to the extracted values of this process.

To get into more detail, a decision made for a specific time relies on the previous values of accelerometer sensor. The running average and standard deviation of the resultant of x, y and z values of the accelerometer sensor are the key factor in the determination of the activity performing at that time. At every performance of activity recognition, these two values, -average and standard deviation of resultant of accelerometer values- are computed for the window of a specific size. The specified window size in this project is 26 samples. The determination of the next activity recognition is achieved by making computation on the late thirteen values of resultants of accelerometer values and recent thirteen resultants. This means, 50% overlap of the windows is performed for every decision making process.

#### 4.1 Activity Recognition Application

An Android Application to detect users’ current state was developed after the feature extraction part had given successful output for detection of users’ current states. This new application consists of a simple user interface to start and stop the activity recognition service. A
service that runs in background to store current values coming from accelerometer sensor and performs decision of the current user activity is implemented in this application.

In this application, the data coming from the accelerometer sensor is dynamically stored in a buffer. This buffer stores the square root of the squares of the X, Y and Z axis values (which we previously referenced as resultant of the accelerometer output)

\[ \text{buffer}(\text{cursor}) = \sqrt{(x^2 + y^2 + z^2)} \]

These values are dynamically stored in an array named buffer [BUFFER_SIZE]. This array is implemented as a Ring Buffer, which stores the new coming data to the head of the array if array boundary is reached. In Figure 4.1.1 the area painted with green represents a window.

![Ring Buffer Structure](image)

Figure 4.1.1: Ring Buffer Structure

The new data of the accelerometer is stored to the proper location which is assigned by cursor variable. This cursor variable is incremented after storing a new value, when it reaches to the top value—which is BUFFER_SIZE- it becomes zero.

\[ \text{cursor} \leftarrow (\text{cursor} + 1) \mod \text{BUFFER}_{-\text{SIZE}} \]

There is a currentTime variable, which is incremented right after any data storing is performed. However, there is an upper bound for this variable, it is called as sensitivity.
`currentTime` can only be incremented up to sensitivity, then substantially becomes zero. When sensitivity is high, the delay between two subsequent outputs increases.

- `currentTime \leftarrow (currentTime + 1) \% sensitivity`

Every time `currentTime` becomes zero, the `findState()` function is called. This function includes the implementation of the decision making technique derived in Feature Extraction part. However, lots of noisy data is buffered right after the application is started; this function is not called at the first time it needs to be called. Otherwise, it would give incorrect outputs for this interval of time.

`findState()` function is the code part all performing of activity recognition is happening. First, it calculates the running average of the resultants stored in the buffer. It performs this calculation by taking the average of the values starting from the current index of buffer defined by `cursor` and the previously stored values old as the number of `WINDOW_SIZE` in the buffer.

Concurrently, it calculates the standard deviation of the values belonging to this window. These two values are used in determination of the users’ current activity. The thresholds values and intervals are extracted from the previous collected training set and used in the implementation of this application. The overall interval and threshold values are as stated in the decision tree in Figure 4.1.2.

![Decision Tree](image)

Figure 4.1.2: Decision Tree
Implementation of the decision tree in Figure 4.1.2 is stated as follows:

```java
private String findState()
{
    float avgACC=0, sumsquare=0;
    int dataNo=cursor; //cursor is the current global index of buffer
    String state="other";

    for(int i=0;i<WINDOW_SIZE;i++){
        avgACC+=buffer[dataNo]/WINDOW_SIZE;
        sumsquare+=buffer[dataNo]*buffer[dataNo];
        dataNo=(dataNo+BUFFER_SIZE-1)%BUFFER_SIZE;
        //data no is decremented one after every iteration up to window size
    }

    float std=(float)Math.sqrt((Math.abs(sumsquare- WINDOW_SIZE*(avgACC*avgACC))/(WINDOW_SIZE-1))); //standard deviation of the values in corresponding window is calculated

    if(avgACC>accValue1) //accValue1=13
        state="running";
    else{
        if(avgACC>=accValue2){ //accValue2=10.4
            if(std<stdValue1) //stdValue1=0.1
                state="standing";
            else
                state="walking";
        }
        else{
            if(std<stdValue2) //stdValue2=0.1
                state="sitting";
            else if(std>stdValue3) //stdValue3=1
                state="biking";
            else
                state="standing";
        }
    }
    return state;
}
```

The basic working principle of activity recognition application is collecting the accelerometer values and reaching a conclusion after processing them. The critical point in this application is the Sensor Service it provides. The activity recognition is provided by the code part –SensorService.java- which extends Service and implements SensorEventListener classes provided by Android SDK.
This service is started by using **Intent class**. This class provides a facility for performing late runtime binding between the codes in different applications. Its most significant use is in the launching of activities and services, where it can be thought of as the glue between them. It basically provides Sensor Service to run in background and ability to communicate with other services and activities if desired.

The output of the activity recognition is stored in a csv file titled with the date and time of the application is started under the folder **HumanTracking**. This output file includes the timestamp and the recognized activity for that specific time in every row. The system recognizes activities in every three seconds and stores this output in this file.

5. **EXPERIMENTS**

The experiments are conducted by 10 people with different clothes (shorts, jeans, sweat pants) to test the threshold values. Depending on their availability, the users ran the Android service application for 2-5 minutes for each activity. The values gathered from the sensors differ according to the type of the clothes. If skinny clothes are preferred, the average values and standard deviations are a little less than when comfort clothes are. We have collected the test values again calculated the average ratios of predicting the expected states and achieving wrong states. The confusion matrix of the experiments after classification phase is seen in Table 5.1. The numbers represent the amount of classification for each activity and their correctness ratio. Note that in the implementation outputs are given every 3 seconds. So, for walking activity, total value 775 means outputs for 775*3=2325seconds(~39 minutes). So, this means averagely a person walked for 3.9 minutes. **The overall accuracy of the application is %86.32.**

<table>
<thead>
<tr>
<th>Real State</th>
<th>Results</th>
<th>Recall</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Walking</td>
<td>Running</td>
<td>Biking</td>
</tr>
<tr>
<td>Walking</td>
<td>680</td>
<td>86</td>
<td>9</td>
</tr>
<tr>
<td>Running</td>
<td>50</td>
<td>336</td>
<td>18</td>
</tr>
<tr>
<td>Biking</td>
<td>36</td>
<td>0</td>
<td>245</td>
</tr>
<tr>
<td>Standing</td>
<td>19</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>Sitting</td>
<td>0</td>
<td>0</td>
<td>40</td>
</tr>
</tbody>
</table>

**Table 5.1: Experimental Results**
6. CONCLUSION AND FUTURE WORK

In the previous works[3,4] about the subject Activity Recognition, mostly wearable sensors and some other body sensors have been used which may be impractical in people’s daily lives since nobody would like to always carry sensors. However, people mostly carry their smart phones in their pockets and in our work we have conducted experiments with the assumption of keeping the smart phones in the jeans, pants, shorts pockets. Each smart phone has quite enough different sensors; temperature, pressure, light, magnetic field, acceleration, orientation, GPS. Taking advantage of these sensors, the human activities during a routine day may be recognized. In this project, we have taken advantage of accelerometer’s different values for each different activity. It’s really useful that the application runs as an Android Service in the background, so the user is not prevented from doing other tasks on his/her phone. In this service, by sensing the accelerometer 26 times each second, extracting the feature values and classifying the activity, the user is able to see how much percent each activity he/she does during the service runtime.

In our work, we have built creating threshold values on 1-second window signals overlapping by %50 characteristics giving the classification output every 3 seconds. In our work, we achieved accuracy rate of %86.32.

As future work, we are considering to prepare an application user interface which asks the users their heights, weights and age. Knowing a person at a specific age, height and weight, the average amount of calories he/she spends for a particular activity can be calculated. Then, our classification results can be used to calculate that person’s average amount of calories during that day. This generated result can be sent to users via e-mails which can trigger them to start a healthier life.
References:

