2008

Accelerometer based motion gestures for Mobile Devices

Neel Parikh
San Jose State University

Accelerometer based motion gestures for Mobile Devices

A Writing Project

Presented to

The Faculty of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements for Degree

Master of Science

By

Neel Parikh

Dec 2008

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Chris Pollett

Dr. Robert Chun

Dr. Mark Stamp

Abstract

Many smart phones today use tiny sensors called accelerometers to provide enhanced user interface controls. Accelerometers measure the linear acceleration in the x, y, z directions based on the movement of the phone. These sensors basically reduce the need of dedicated navigation and function keys on the mobile device. Accelerometer based mobile devices use this principle for creating applications like games, controlling the orientation of the display screen, etc.

The goal of this project is to extend the WebKit browser interface of Google's mobile development platform called 'Android' by creating accelerometer based motion features like shake feature, orientation of images, zoom in/out, scrolling, etc. For instance, the user can shake the phone in order to erase an entered text. Also, by rotating the phone clockwise or anti-clockwise, the orientation of the underlying images will change accordingly. While browsing a web page, the user could tilt the phone in left, right, top, bottom directions which will cause the web page to scroll accordingly. Also, by tilting the phone towards or away form the user, one can zoom in and zoom out on a web page.

Table of Contents

PHASE 1 OF THE PROJECT

PHASE 2 OF THE PROJECT

Index of figures

Zoom feature

Scroll feature

Testing media playback capabilities of Android

4. Introduction

The demand for consumer electronic equipments is growing rapidly. One such device is a mobile phone or more appropriately a ‘smart phone’ as we call it today. Many smart phone companies today make extensive use of sensors called accelerometers for developing mobile based applications. The prime purpose of an accelerometer is for measuring the acceleration and the force of gravity it experiences. Accelerometer based devices have been there for quite some time, but have recently gained popularity with the advent of ‘iPhone’ from Apple Inc.

The major smart phone makers like Nokia, Blackberry, Sony Ericsson, Palm etc today support accelerometer in their mobile devices. The latest player in this market is Google, Inc with their open source mobile development platform called ‘Android’. The capabilities of accelerometers are being exploited to develop full-fledged mobile application like games, to provide enhanced user interface controls and ease of navigation etc. Another good example is the ‘Wii gaming console’, which uses these sensors for achieving a realistic gaming experience.

4.1 Overview of Project

This project focuses on extending the WebKit browser interface of Google’s mobile development platform called ‘Android’ by creating accelerometer based motion features like shake feature, orientation of images, zoom in/out, scrolling, etc.

The applications for this platform have been developed using the ‘Android Software Development Kit’ version m5-rc14 released by Google, Inc. The testing and has been carried out on the Android emulator supported by the SDK which is integrated with the Eclipse Integrated Development Environment. The applications have been written

using the Java programming language and run on Dalvik Virtual Machine (DVM). It is a custom virtual machine designed by Google for embedded platforms like 'Android'. The project uses an Open Source Sensor Simulator in order to simulate real time accelerometer data as input to the Android Emulator. The project also uses various other Android related tools, the details of which are explained in the development tools section.

The challenge and excitement of working with Android platform can be attributed to the fact that it involved a learning curve; starting with a 'Hello world' and going on to develop full fledged accelerometer based motion features.

4.2 Organization of Report

This report gives a detailed description about the research work, testing and application development carried out on the Android Platform. Each deliverables involves experimenting with the Android SDK and developing accelerometer based applications mentioned in the project overview section 4.1. The entire project has been divided into two phases namely 'PHASE 1' and 'PHASE 2'. This report is organized in the form of well defined sections. Each section consists of an introduction with a clear objective, description of the simulation or event and the description of the code files.

Given below is the organization of various sections:

Section (5) talks about the Android platform in terms of system architecture and development tools for Android platform.

Section (6) describes the basic 'Hello World' application for Android which helped in understanding the creation, compilation, debugging etc of an application.

Section (7) describes the functioning of Sensor simulator and the automated input functionality which is required to send continuous input data from simulator to emulator.

Section (8) is related to a simulation even for testing the automated input functionality in a scenario of 'throwing the phone' and recording changing accelerometer values.

Section (9) explains the design methodology used for creating the accelerometer based features with the help of flow diagrams and description for the same.

Section (10) involves the implementation of the shake feature where an entered text can be erased by detecting shake motion of the phone.

Section (11) talks about implementation of the rotation feature where images can be rotated clockwise and anti-clockwise by changing the orientation of the phone.

Section (12) describes the Zoom feature extended with the WebKit browser interface, where a web page can be zoomed in and out based on tilting the phone.

Section (13) talks about implementation of the WebKit based Scrolling feature where a web page can be scrolled in up, down, left and right directions by tilting the phone accordingly.

Section (14) is an independent application to test the media capabilities of the Android platform.

Section (15) is dedicated to the challenges involved in the project.

Finally, the report concludes by summing up the accomplishments of the project and briefly talking about the future of Android followed by the references.

PHASE 1:

5. What is Android

Android is an Open Source mobile development platform developed by Google, Inc. It is a complete stack consisting of an Operating system, middleware and Applications. Android has been built on the Linux Kernel version 2.6. It uses a customized virtual machine which has been optimized to have minimum memory footprint and efficiently use hardware resources in a constrained mobile environment. The customized virtual machine in Android is called 'Dalvik Virtual machine' or DVM. The Software development Kit (SDK) released from Google provides the developers with the necessary tools for creating applications for this platform. The applications are created in java programming language. Section 5.1 describes in detail the underlying architecture of Android platform.

Some of the key features of this platform include Application framework, customized virtual machine, integrated browser, support for 2D and 3D graphic libraries, media support for various audio and video formats, Bluetooth, 3G, WiFi, hardware sensors like accelerometer, compass, etc. In addition to these, the platform also includes support for a development environment consisting of a device emulator, debugging tools, and plug-in to the Eclipse framework.

5.1 Android Architecture

Given below is the architectural diagram of Android and the key components within the Android system.

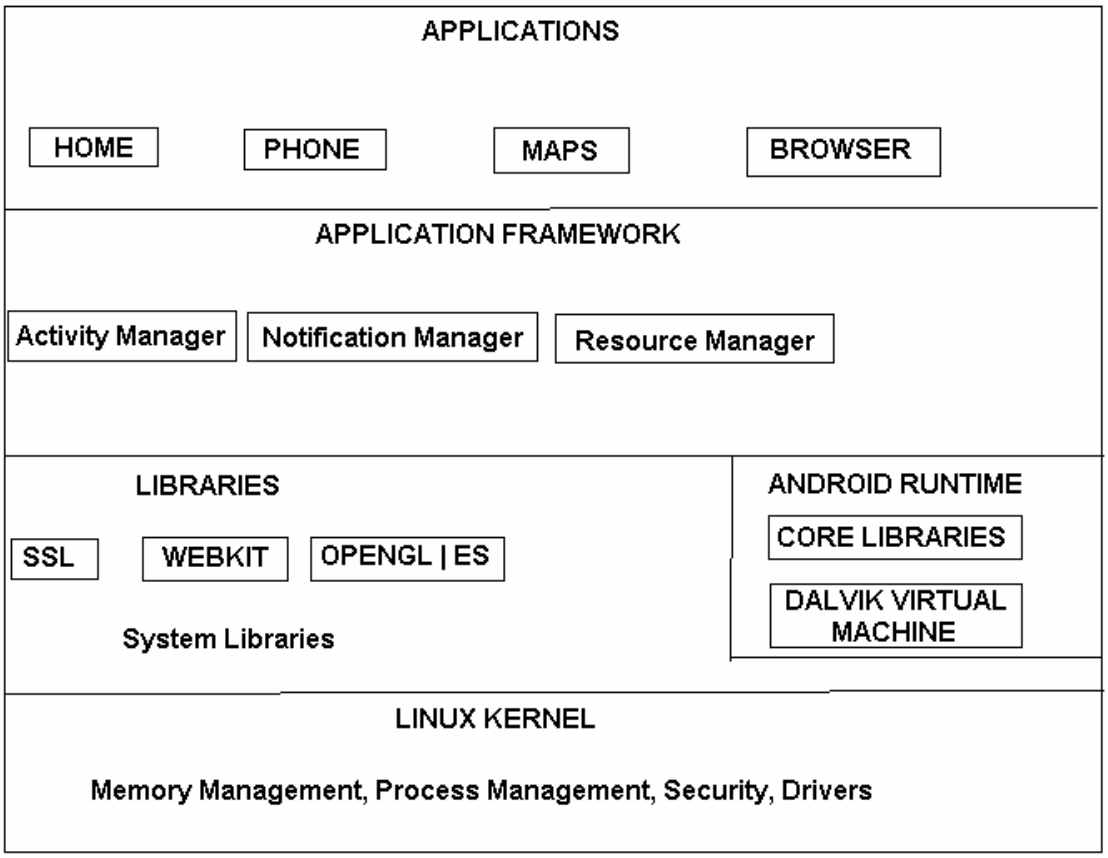

Fig 5.1: Android Architecture.

OS Layer: As seen in the above figure, Android used the Linux Kernel at the OS layer. The kernel is also responsible for memory management, process management, security and drivers. This layer provides the abstraction between the underlying hardware and the software system.

Middleware: Above the Linux kernel are set of C/C++ system libraries. Some of the important libraries include 'libc' which is the standard C system library. This has been

customized for supporting embedded Linux environment. There are media libraries which support popular audio, video and image formats. A 'SQLite' database engine for the applications, 'SGL' which is the 2D graphic engine, 'Webkit' which is the open source engine that powers the browser, etc.

Dalvik Virtual machine (DVM): Also, the middleware consists of the Android runtime which includes the customized Dalvik virtual machine (DVM). It executes files in 'dex' format which is dalvik executable format. DVM converts the classes compiled by java virtual machine into dex format with the help of a tool call 'dx'. The Linux Kernel handles threading, process management, memory management and other related issues for the Dalvik Virtual Machine. In addition to the DVM, the runtime also includes the Core Java Libraries which provides all the java functionality.

Application Framework: The application framework consists of:

Content Providers – It enables one application to share its data with another application. For example the details of an individual stored in the phone book application could be used by the email application.

Activity Manager - It is responsible for handling the life cycle of you Activity.

Notification Manager - It is responsible for displaying important alert messages.

Resource Manager - It is responsible making the resources like gif, png images or layout files used for the UI display available to your application.

Applications: The Android system includes some built-in core applications like the email client, browser, maps, calendar etc. These applications have been written using the Java programming language. Any user developed applications exist at this level and make use of the underlying functionalities of the core java libraries.

5.2 Android Development tools

This section gives a brief description of the tools used during the testing and development of various features for the project.

5.2.1 ADT Eclipse plugin – The Android Development Tools (ADT) is a plugin for working with the Eclipse framework. This plugin provides support for DDMS (Dalvik Debug and Monitoring Service) from within Eclipse. Overall, it makes the application easier and faster in terms of creation, debugging and running.

5.2.2 Android Emulator – The software development kit have a device emulator. It is a virtual emulator that runs on your computer and simulates the software and hardware functioning of the Android platform. A developer can easily develop and test applications without the need of an actual device. The emulator also has a console which allows you to log kernel output. One can also simulate incoming phone calls and messages (SMS) for testing purposes. Given below is a screenshot of the device emulator running in eclipse framework.

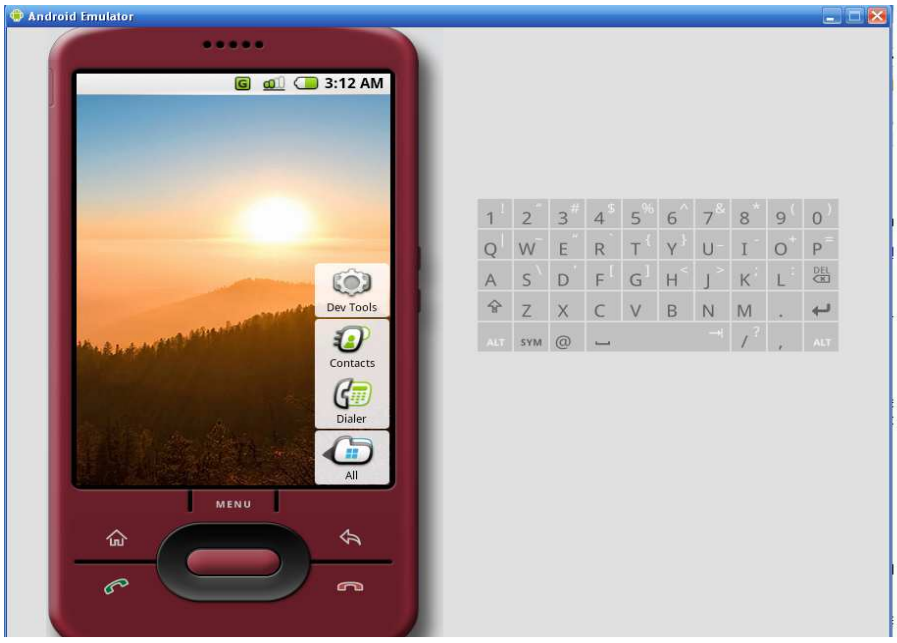

Fig 5.2: Android emulator running in Eclipse

5.2.3 DDMS monitoring service – It stands for Dalvik Debug and Monitoring service.

It is a debugging tool and enables a developer to perform various tasks such as screen capture of the emulator, list running threads, heap information, viewing debugging messages in logcat, list the running process, simulate incoming call and SMS, etc. Given below is a screenshot of the DDMS running in eclipse framework.

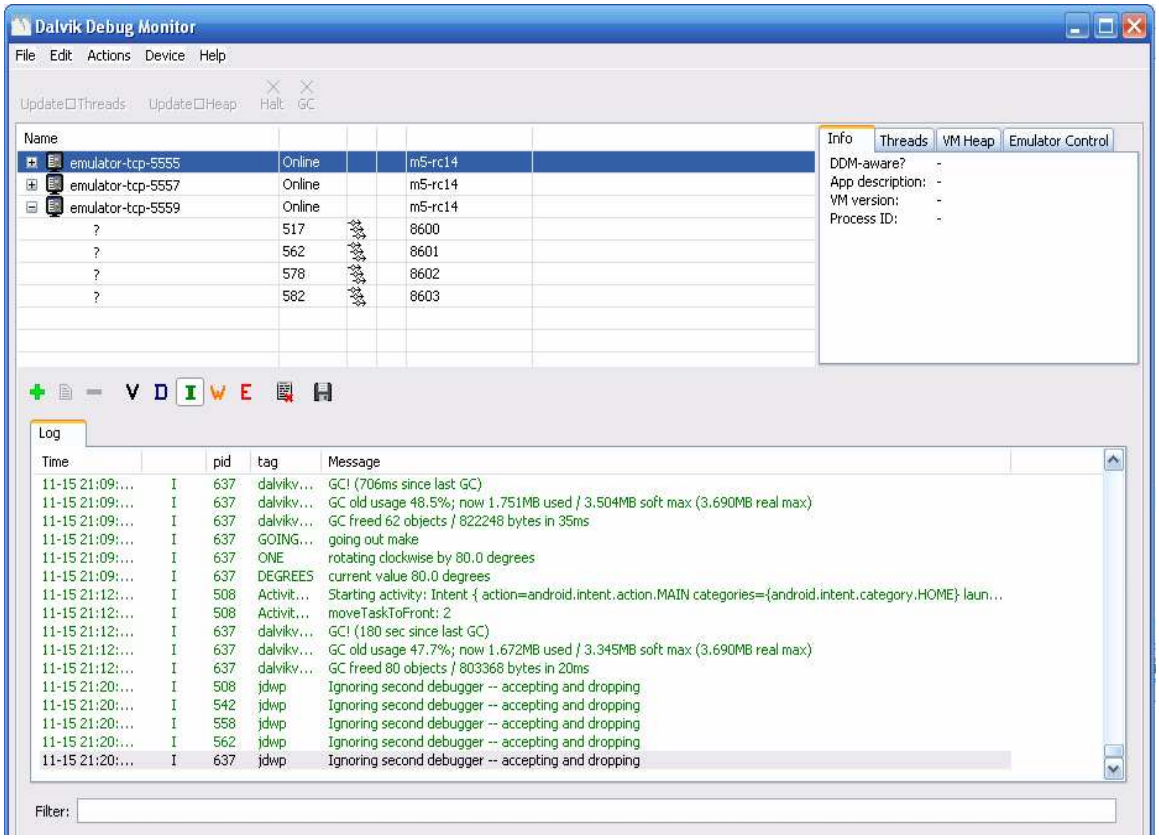

Fig 5.3: DDMS running in Eclipse

Some of the major features of DDMS are explained below which are very useful while developing and testing and Android application.

(i) Logcat: DDMS gives access to 'Logcat', a feature used for writing log messages. It helps in collecting and viewing system debug messages. One can also filter specific log messages view them using the 'logcat' command from the shell prompt. This feature is very helpful for checking the control flow of the code and debugging. One can insert log

messages in the code and monitor the logcat during the execution of the application to check for flow of control of the code. The following command is used for writing log messages to the logcat.

```
android.util.Log.i("DEGREES:",”current value ”+String.valueOf(degrees)+” degrees”);
```

Issuing the above command would create a ‘tag’ named Degrees in the log file with the associated ‘message’ containing the value of degrees. Given below is a screenshot of Logcat running in Eclipse framework.

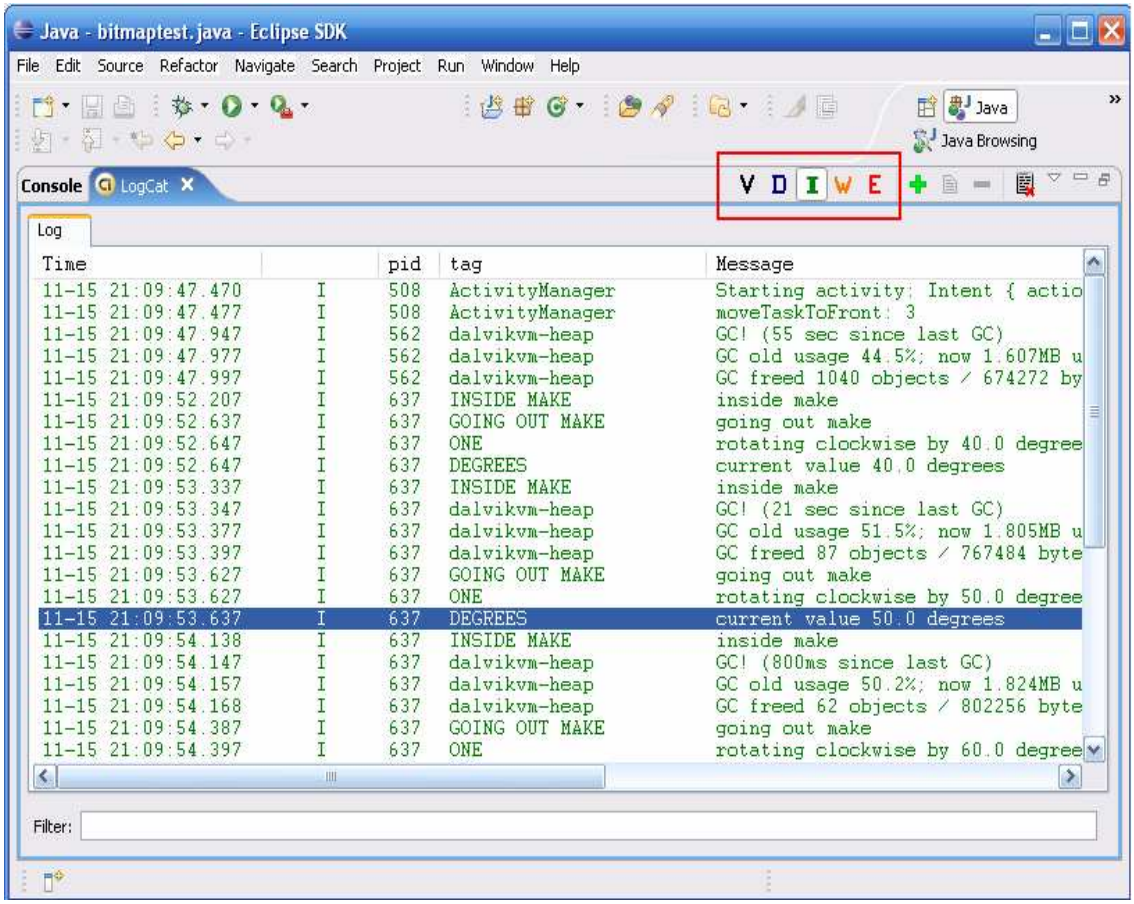

Fig 5.4: ‘Logcat’ running in Eclipse.

There are four different kinds of messages that are displayed by the logcat (highlighted by a red box in the above figure). They are ‘Verbose’, ‘Debug’, ‘Information’, ‘Warning’ and ‘Error’. Each of these messages is color coded for easily distinguishing one for another. The user inserted messages are displayed as ‘Information’ messages.

(ii) Processes: DDMS has options for viewing all the currently running processes in a graphical view. It is the equivalent of running the linux command ‘ps - x’. Given below is the screenshot of all the running processes as seen from DDMS.

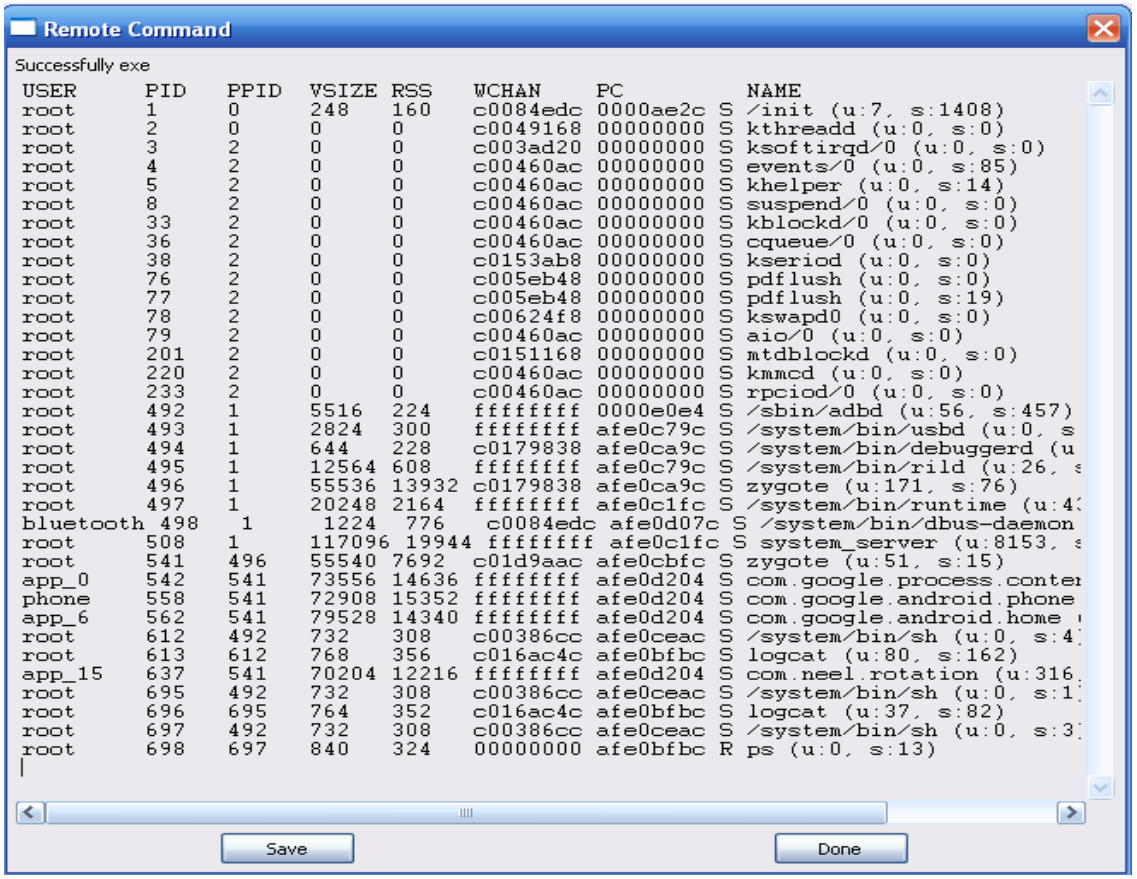

Fig 5.5 DDMS listing the currently running processes

(iii) File Explorer: The DDMS also has File Explorer view, which lets you see the currently installed application i.e. the .apk files installed on the emulator. It is the

equivalent of traversing the Linux file system and installing, deleting or listing application packages. Given below is the screenshot of the file explorer view.

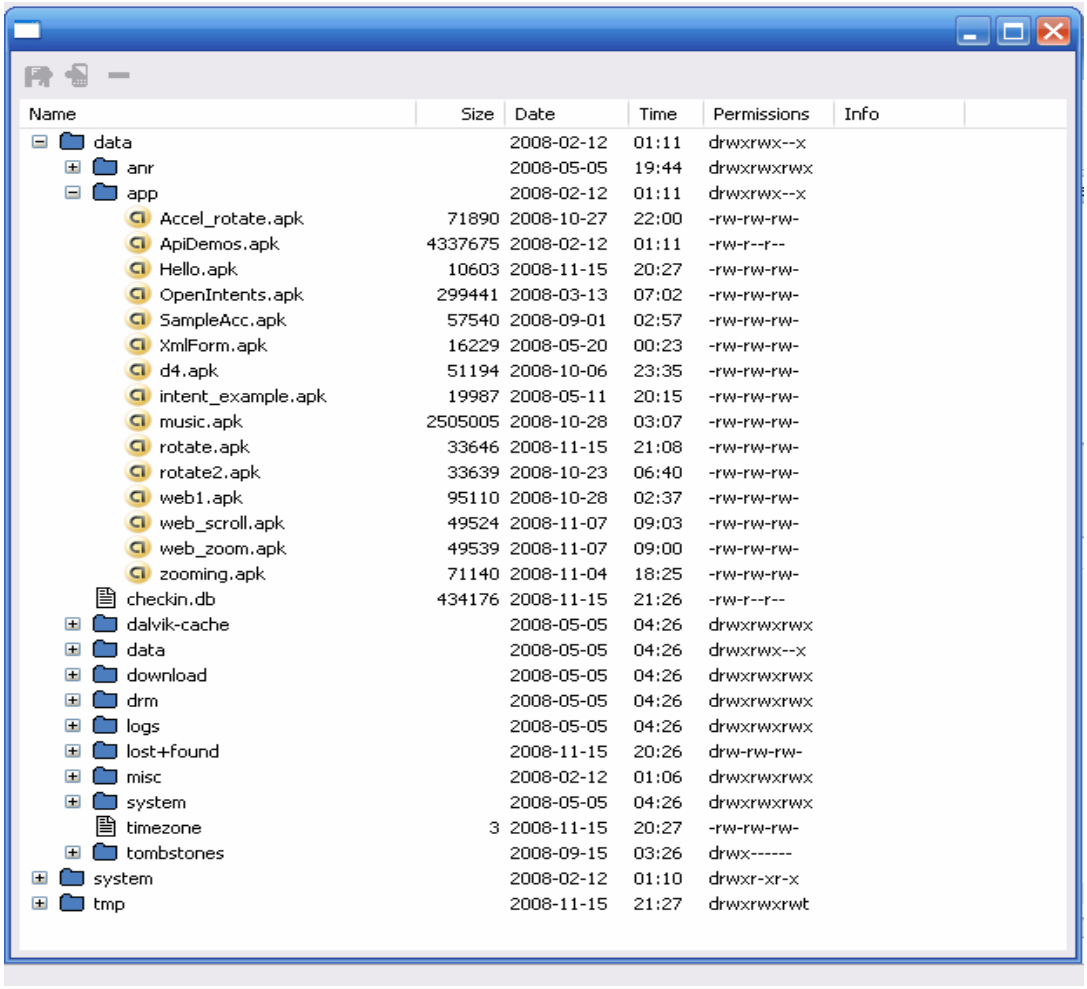

Fig 5.6: File explorer view running in DDMS

5.3.4 ADB – It stands for Android Debug Bridge. Incase when we are not using Eclipse framework, then ADB helps us to issue command from a DOS console (in windows). It provides all the GUI features of DDMS in a command line format. For instance, in android you can have various emulators running, each with their own applications. Some of the useful adb commands are explained below:

1. In order launch an emulator from the command line, traverse to the 'tools' directory of Android SDK and issue the following command:

```
C:\android-sdk_m5-rc14_windows\tools>emulator.exe
```

Each time this command is executed, a new emulator instance is created. One can use the adb commands in-order to query each of the emulator instances.

2. One of the basic commands which help you monitor the currently running emulator instances is listed below:

```
C:\android-sdk_m5-rc14_windows\tools>adb devices
```

Given below is the screenshot of issuing the above command from a command prompt in windows:

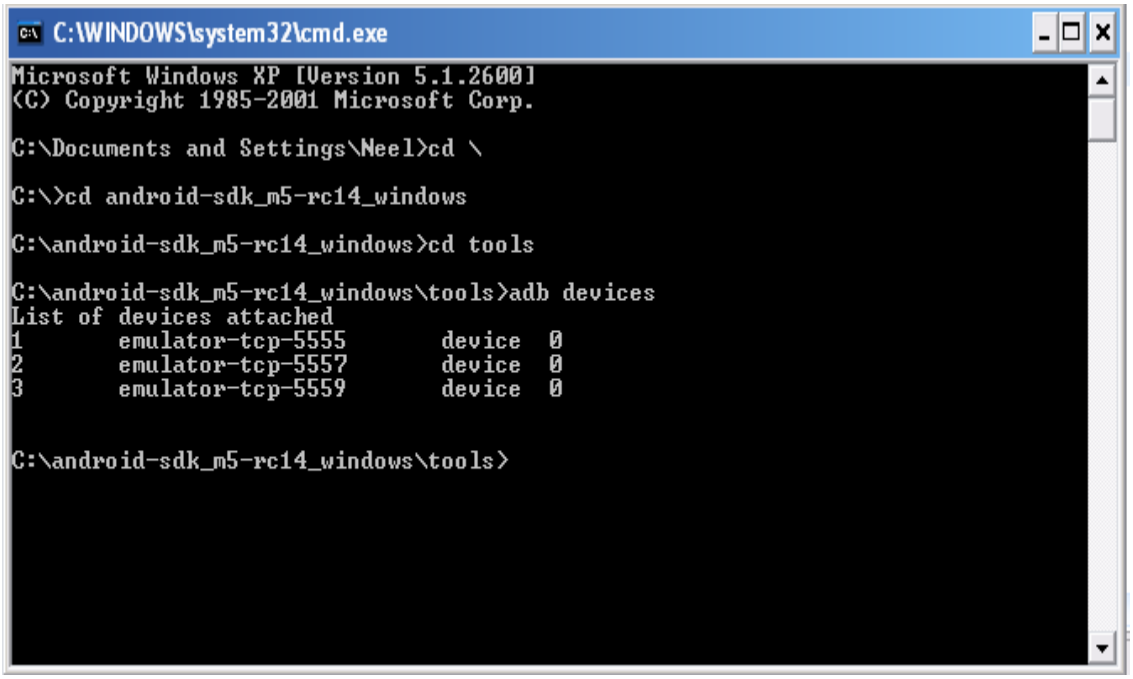

Fig 5.7: 'adb devices' command.

As seen from the above figure adb lists of all the emulator instances which are currently running at the time of execution of this command. It also displays following information about each of the emulator instances namely:

ID – It is a unique identifier assigned to each of the emulator instances running. The identifiers are assigned starting from 1,2,3,….. so on.

Serial number – It a unique string identifier associated with each of the emulator instances. It also has a unique port associated with each emulator incase we need to talk to a specific emulator.

State – This string identifies the state of the emulator. Here 'device' means that the emulator instance is linked to the adb server. The other states possible are 'offline' and 'bootloader'.

3. In-order to install an application to a specific emulator from command prompt we need to traverse to the 'tools' directory and issue the following command

```
C:\android-sdk_m5-rc14_windows\tools>adb –d 3 install C:\ android-sdk_m5-rc14_windows\tools\music.apk
```

The above command will install the application 'music.apk' on the emulator instance which has the identifier '3' associated with it. One can also use the string identifier instead if the numerical identifier as seen in the 'adb devices' command window. Given below is a screenshot of executing this command and the emulator screen with the 'music.apk' file installed.

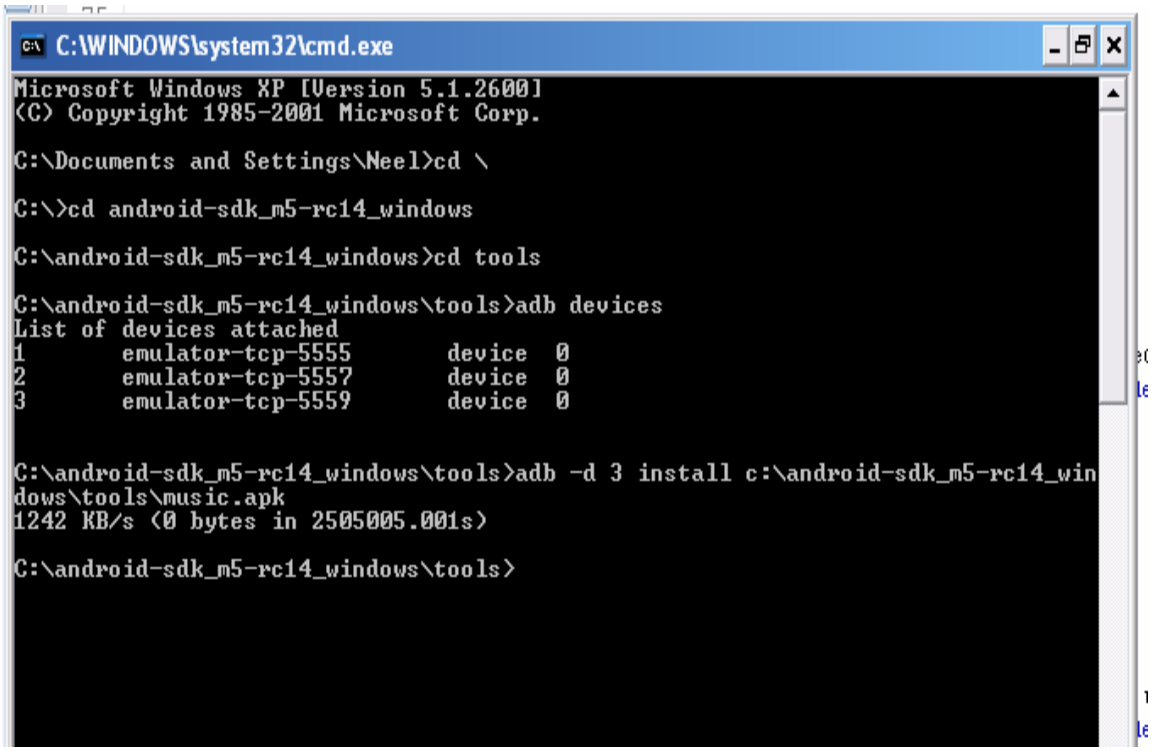

Fig 5.8: 'adb install' command.

This entire section gave an idea about the Android Platform and its architecture. It also described the development tools required for testing and debugging applications. The entire application development and testing of Accelerometer based motion features have been performed with the Android Development Tool (ADT) plug in for the Eclipse Integrated Development Environment.

6. Hello world in Android.

The Android platform was launched by Google in November 2007. We started with the project in January 2008 and hence the challenge and excitement rested in overcoming the initial learning curve which is associated with any new platform. Hence, it was evident to start with setting up the framework required for application development on the Android platform and begin with a 'Hello World' application.

This test application laid the basic ground work required to understand the process involved in creating, compiling, running and debugging an application on the Android platform.

The output of here would be an independent application which the user would see on the home screen of the emulator (Fig 6.1). Once the user clicks on this application, it would display a welcome message to the user (Fig 6.2).

Fig 6.1: Hello World application on home screen of emulator

Fig 6.2 Hello World running on the emulator

Given below is a brief description of the User Interface needed to create this application and the Android API's used.

Constructing the UI: The user interface for the Hello World program consists of a simple 'TextView' needed to display a message to the user. Given below is the code snippet for this for creating a 'TextView' and setting a user defined text to it.

```
TextView t = new TextView(this);

tv.setText("Hello people, welcome to the world of Android");

setContentView(tv);
```

The above snippet shows a 'TextView' object being instantiated. The 'setText' method is used to associate a string to the 'TextView'. Finally the text is displayed in the main view of the application using 'setContentView' method.

In this particular case, the user interface has been directly created in the source code. Android also provides an alternate approach to create user interfaces for an application by creating XML based layout files. Here, one can create all the UI elements like text boxes, buttons, etc in a separate XML layout file. Each of these UI elements would be assigned a unique id which can be referenced from the main source file. In Android, this XML layout file is called 'main.xml' by default. The XML layout file approach is very useful while developing large applications which involve a lot of UI elements. Having separate UI files helps in reducing the code clutter as the UI element creation is handled in the 'main.xml' file. Also, it makes it easy for a developer to make changes and enhancements to UI elements in a separate file.

The above section explained the process for developing 'Hello World' application on the Android platform. It also mentioned about the basic user interface creation in an application by either embedding the UI code within the main source code or having separate XML based layout files which could be referenced in the main source code.

After testing the 'Hello World' application, the next step was to get familiar with few more test applications involving event handling. These applications would help in understanding the event handling mechanism and interaction of UI elements. It would also provide an insight into sending and receiving of data between different UI screens within a main application. All these sample test applications created for understanding the even handling mechanism have not been included in this report.

7. Sensor Simulator

The hardware API's and packages in Android enable access to underlying platform sensors. The different types of sensors supported by the platform are accelerometers, Compass and Orientation. In order to work with these sensors API's in Android and to simulate, detect and test the movement of accelerometer on the Android platform, some form of motion simulator is required.

We have used an open source simulator called 'Sensor Simulator' (version 0.1.4) which has been shown in (Fig 7.1). This simulator has been developed by an open source project called OpenIntents [4]. This simulator provides a means to simulate movements for the underlying sensors supported by the Android platform, based on the movement of the mouse pointer on the simulator.

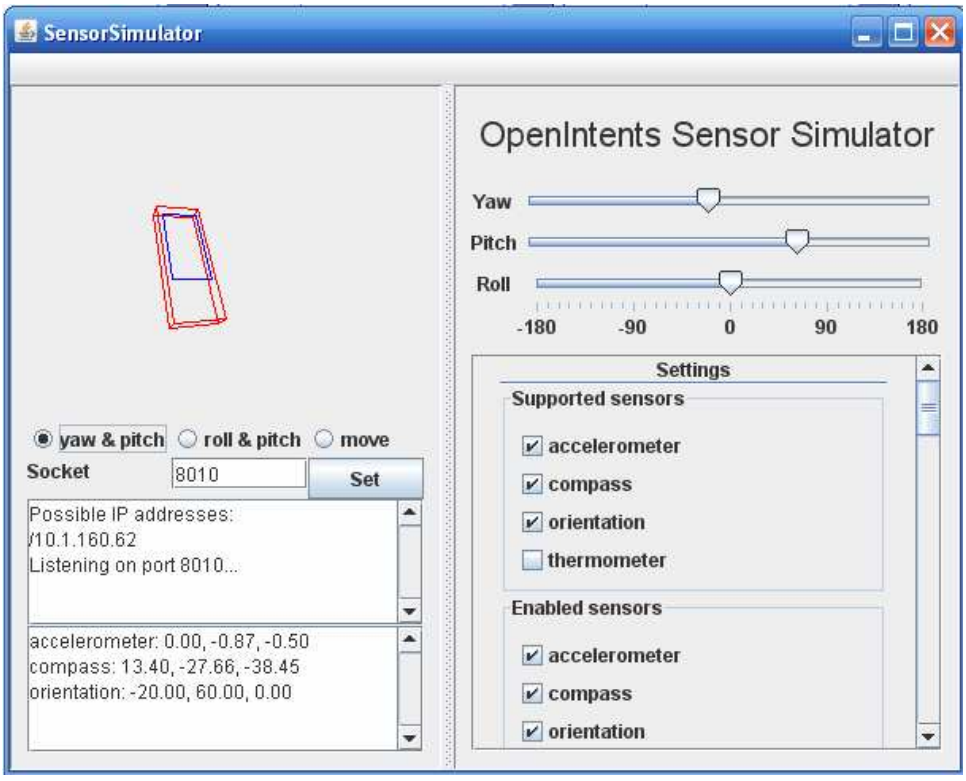

Fig 7.1: OpenIntents Sensor Simulator

This simulator will be used for simulating movement of the Android Emulator for testing accelerometer based features like shaking the phone. Also, the features such as orientation zoom in/out, and scrolling which have been developed by extending the WebKit browser interface of Android will use this simulator to send real-time input data while the application is running.

Hence, the simulator was very essential to the features being developed as it provided means to simulate real time movements of the emulator which could be used as input to the application running on the phone.

7.1 Connecting Sensor Simulator to Android Emulator.

In order to use this simulator, the next step in the project was to integrate the OpenIntents Sensor Simulator with Android Emulator. After integration, the accelerometer values on the simulator would be reflected on the emulator screen as shown in (Fig 7.3).

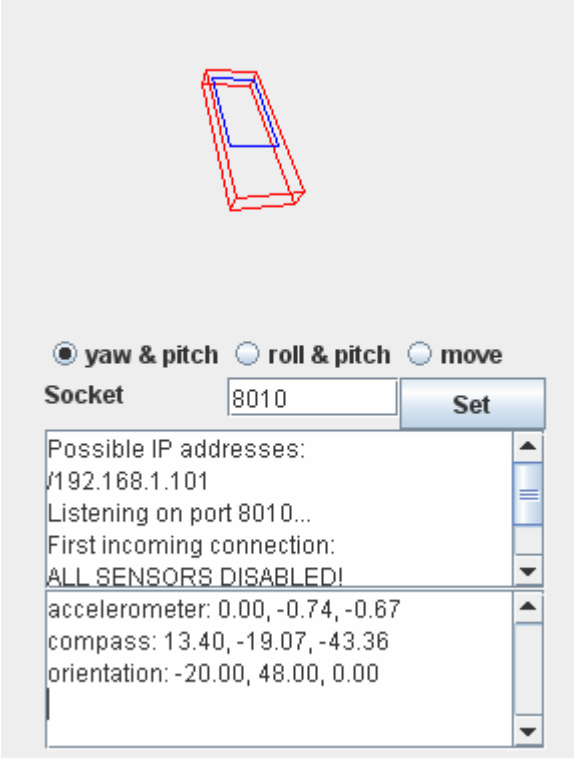

Fig 7.2: Accelerometer values generated by moving the Simulator phone

The above figure shows the OpenIntents Sensor Simulator. This is a standalone jar application. It lets you simulate sensor data with the mouse in real time. Given below are the steps required to connect the simulator to the Android emulator, which could then be used to simulate real-time accelerometer inputs to the all features developed in this project.

Connection: Procedure for connecting the Sensor Simulator to the Android emulator is as follows -

1. Download the openintents-binary-0.1.4.zip package.
2. Unzip it and then start the simulator by traversing tools > SensorSimulator.jar (Java standalone application as shown in Fig 7.1).
3. Install the OpenIntents.apk (application package) on the Android emulator from the command prompt using the adb install command as explained in section 5.4.4.
4. Launch the OpenIntents on the emulator and select the SensorSimulator from the options.
5. This pops up a UI where you can enter the IP address and socket number. This IP and socket number is shown in the Sensor Simulator (Fig 7.1).
6. Then go to the testing tab and select 'Connect' button on the emulator. During the first connection setup, all sensors are automatically disabled. This is due to the fact that we want the application to enable the sensors before reading the values.
7. Now you can see the sensor data (Fig 7.2) on the emulator screen with a small delay. The simulator data and the emulator data are in sync with each other.

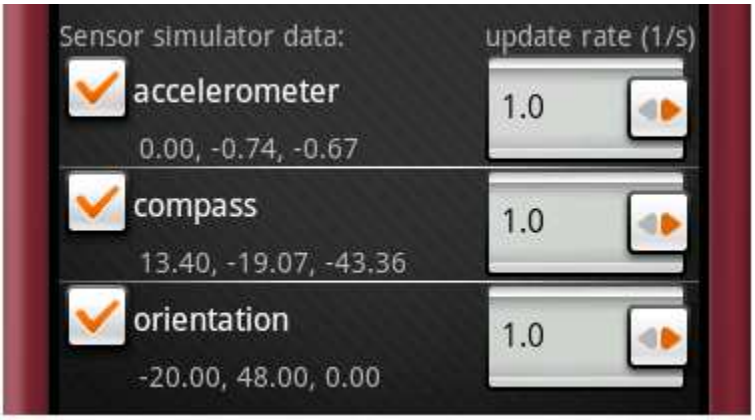

Fig 7.3 Emulator values in sync with the Sensor Simulator (Fig 7.1)

If you move the SensorSimulator phone with the mouse, the accelerometer values [x,y,z] will change in the Sensor Simulator and correspondingly change on the Android emulator. These values can then be used in any application on the emulator.

7.2 Addition of automated input functionality

The movement of simulator phone by mouse corresponds to subjecting the [x,y,z] values as show in (fig 7.4) to change. Hence, if one would manually change any of these fields, it would cause change in the motion of the simulator phone which would be detected by the accelerometer, causing [x,y,z] values to change.

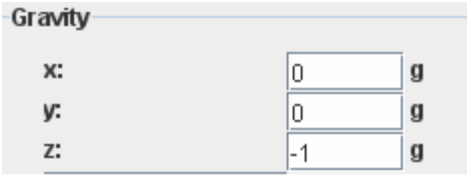

Fig 7.4: Gravity settings of Sensor Simulator

Changing the values of the text boxes of [x,y,z] caused the change in the motion of the simulator phone, which led to change in the sensor values of Accelerometer. The new generated value would get reflected on the emulator screen as explained in the connection procedure of section 7.2. However, this is a manual process and requires the user to change the values every time to generate a small change in the sensor values.

Hence, there was a need to come up with some automation which would continuously pass a set of values to the emulator in the background at a fixed interval, and the actual application running on the emulator would capture these incoming values and use it in the application.

So, to achieve this automation, we added the functionality of playing a set of input values to the [x,y,z] text boxes of the gravity fields in the standalone jar application. On selecting an input file and clicking the 'Play' button, a java thread would be invoked which would play in the background and read a value after a fixed interval and send it to the emulator. These values would be used by the application running on the emulator. This is the main mechanism involved in transmitting the values to the emulator and was a major step in-order to proceed and go on to create accelerometer based features which could be tested by a continuous set of input values. In other words, we had a means of simulating movements on the phone without the need for an actual physical device. The figure below shows the modifications made to the simulator Jar file. The additional 'Select input file', 'Play', 'Stop' buttons can be seen below.

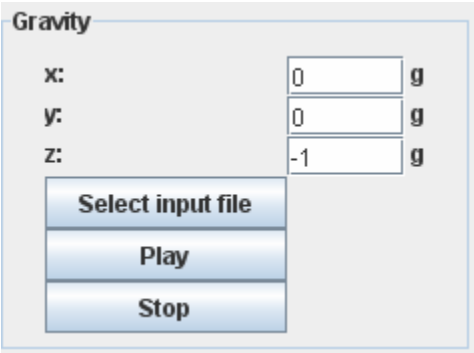

Fig 7.5: Additional functionality of playing the values from a file.

The challenge here was to dig through the source code of the OpenIntents project and add this functionality to the existing code. In doing so, I had to ensure that the overall working of the simulator is not affected and the new functionality behaves as expected. Given below is a small code snippet for this additional input file choosing functionality.

```
else if (action.equals("Select input file")) {
    fc = new JFileChooser();
/**
 * Pops up an "Open File" file chooser dialog
 */
    int returnVal = fc.showOpenDialog(SensorSimulator.this);

   if (returnVal == JFileChooser.APPROVE_OPTION) {

            file_name = fc.getSelectedFile().getPath();
            choose_file.setToolTipText(file_name);
     }
```

After this automated functionality was achieved, the next step was to test this automation by using it in an application which has been described in the next section with the help of a test scenario of 'throwing a phone'.

This entire section described the use of OpenIntents Sensor Simulator and its connection to the Android Emulator. It also explains about the automated input functionality required to send continuous real time input data, which would be used by accelerometer based features running on the emulator.

8. Simulation to test the automated input functionality.

This section describes the application used to simulate the scenario required for testing the automated input functionality of the simulator, as explained in the previous section. The idea was to test the automation functionality by means of recording the changing accelerometer values by simulating an event of 'throwing a phone'.

Simulated Scenario: Suppose a phone is lying on the table. If the phone starts falling down, then immediately an alert message should be generated informing the user that the phone is free falling and the changing accelerometer values should be recorded and displayed in the background (Fig 8.1). And finally when the phone strikes the ground another alert message pops up (Fig 8.2) and the accelerometer values come to rest.

Fig 8.1: The alert message when phone starts falling and accelerometer values being recorded.

Fig 8.2: The alert message when the phone strikes the ground.

Given below is the description of the event and the code files used in this simulation.

Description of the event: Initially we will assume that the phone is at rest on a surface. Hence the initial values of [x,y,z] will be [0,0,-1]. Here the 'x' and 'y' axis is 0, while the 'z' axis is -1 due to force of gravity. Then we will simulate an event that the phone is thrown and is falling down. As soon as the phone starts falling down, then immediately the accelerometer would start recording change in the x,y,z values. During the fall the [x,y,z] values will change rapidly to cause a change in the accelerometer values. An alert message would be generated informing the user that the phone is free falling and the changing accelerometer values would be recorded and displayed in the background (Fig 8.1). When the phone strikes the ground, the [x,y,z] values will come to rest and so will the accelerometer values. And finally when the phone strikes the ground another alert message pops up (Fig 8.2) and the accelerometer values slowly come to rest. The values generated during the entire simulation are also written to a log file to help in testing and debugging the code.

Description of code and files: The entire coding of this simulation event has been split up into four files.

Test.java: This file is the main class which is initially executed and which connects to Sensor Simulator. The functionality for capturing and displaying the [x,y,z] values is coded in this file. Given below is the snapshot of displaying a value on screen.

```
/**
 * Display values of [x,y,z] on the screen
 */
   if(j= =1){
         TextView temp1 = (TextView)findViewById(R.id.one);

temp1.setText("[x,y,z] = ["+ String.valueOf(String.format("%.2f",d1)) + ","

+String.valueOf(String.format("%.2f",d2))","

+String.valueOf(String.format("%.2f",d3))+"]");
               }
```

AccelerometerReader.java: This class is called from Test.java and it basically enables the sensors on the phone. It also tests for a particular sensor (Accelerometer in our case) and enables it. The accelerometer values corresponding to [x,y,z] are generated in this class and the result is passed to the Test.java class where they are displayed. Given below is the snapshot of the reading the values from accelerometer

```
int sensorValues =
Sensor.getNumSensorValues(Sensors.SENSOR_ACCELEROMETER);

float[] out = new float[sensorValues];

Sensors.readSensor(Sensors.SENSOR_ACCELEROMETER, out);

android.util.Log.i("FIVE","in readAccelerometer");

return out;
```

main.xml: This is the XML file which has the UI button "throw the phone", used to start the entire simulation.

main2.xml: It containts the UI text views to show the rapid changing values of accelerometer corresponding to [x,y,z] on screen.

For debugging purposes each value is written to a log file. Given below is a snapshot of the values being written to the log file.

```
05-13 04:55... I  676   Answer          7.348256587982178
05-13 04:55... I  676   Answer          -3.872981309890747
05-13 04:55... I  676   Answer          -5.291799545288086
05-13 04:55... I  676   IF              2nd value
05-13 04:55... I  676   FIVE            in readAccelerometer
05-13 04:55... I  676   FIVE            in readAccelerometer
05-13 04:55... I  676   Answer          6.919868469238281
05-13 04:55... I  676   Answer          6.200979232788086
05-13 04:55... I  676   Answer          5.259589195251465
05-13 04:55... I  676   FIVE            in readAccelerometer
05-13 04:55... I  676   FIVE            in readAccelerometer
05-13 04:55... I  676   Answer          6.919868469238281
05-13 04:55... I  676   Answer          6.200979232788086
05-13 04:55... I  676   Answer          5.259589195251465
05-13 04:55... I  676   FIVE            in readAccelerometer
05-13 04:55... I  676   FIVE            in readAccelerometer
05-13 04:55... I  676   Answer          0.0
05-13 04:55... I  676   Answer          -8.66025447845459
05-13 04:55... I  676   Answer          -5.0
```

Fig 8.3: Values being written to the log file when the code is executing.

After successfully testing this simulation we had a means of sending a continuous set of accelerometer based movements as input to the emulator. This laid the foundation to test the accelerometer based features like shake, zoom in/out, scrolling, image orientation, etc

This work was completed as part of Phase 1 of the project. It was followed by Phase 2 of the project which involved extending the Webkit browser interface by developing accelerometer based features. The testing and development done is ‘Phase 1’ was instrumental in setting up a strong foundation and knowledge for development on Android Platform.

PHASE 2:

9. Design Methodology:

Activity Lifecycle: Every accelerometer based feature is implemented as an Activity.

Given below is basic flow diagram for the lifecycle of an Activity in Android.

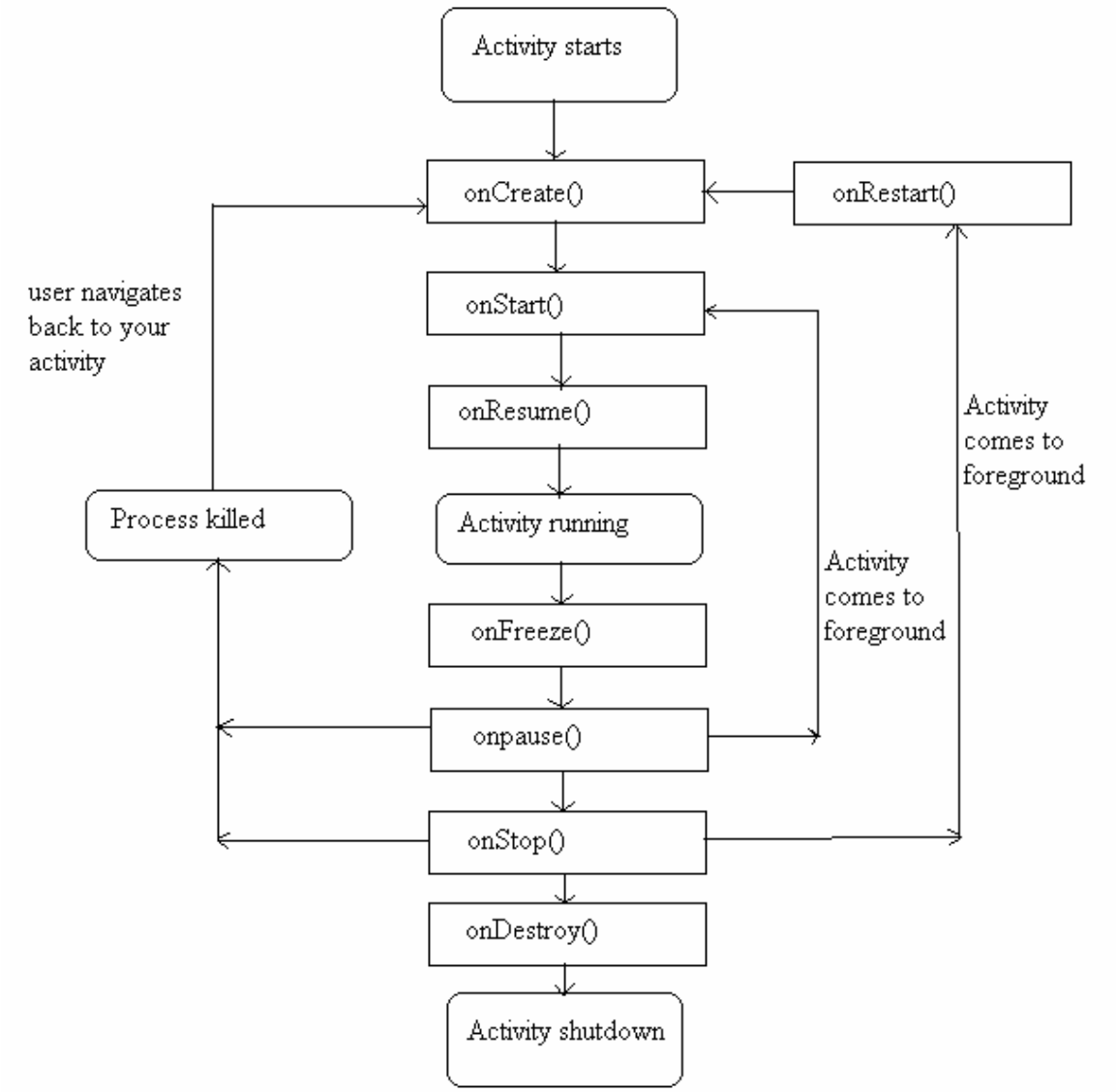

Fig 9.1 Activity lifecycle

The Phase 2 of the project involved developing a specific design strategy and flow in order to develop the accelerometer based features. In Android, 'Activity' class is

responsible for creating full screen window for the application in which the developer can set the user interface elements. A brief description of the flow of Activity is given below:

1. onCreate() : Every Android application has an 'onCreate' method. This method is called initially when the activity is first created. This is where all the initialization, association of XML layout files to UI elements, setting the content of the view takes place. It is followed by onStart() or onRestart().

2. onRestart() : Called after the activity has stopped. It calls onStart().

3. onStart() : It is called once the activity becomes visible. This method calls onResume().

4. onResume() : This is called when the activity starts interacting with the application. It is followed by call to onFreeze() or onPause().

5. onFreeze () : It saves the current state before calling onPause().

6. onPause () : This method is called when a previous activity is going to be restarted. In this state all he unsaved changes are saved. Once this state completes, then only the next activity can resume. It calls onResume() or onStop().

7. onStop () : When the current activity becomes invisible and is covered up by another activity, the onStop method is called. This is followed by onRestart() or onDestroy().

8. onDestroy () : It is called before the activity is destroyed

AndroidManifest file:

Every Android application will have an AndroidManifest.xml file. This is a required file in every application. Given below is the Androidmanifest file for the 'Hello World' program

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.neel">
    <application android:icon="@drawable/icon">
        <activity android:name=".Hello"
android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

As seen in the above Listing:

1. <manifest> tag: Every application will have a <manifest> tag which is the namespace declaration. This is the root node.

2. <application> tag: This tag includes all the components contained in the package. The activities and intent filters are defined in the application tags.

3. <Activity> tag: The user can interact with the application through activity. Every screen that is implemented is an activity and is defined here with additional <activity> tags. It defined the name of the activity which is '.Hello' in this case. One or more intent filters are declared within this tag.

4. <intent-filter> tag: An intent is the operation which is going to be performed.

It includes <action> tag which describes the name of the intent action. In this case it is android.intent.action.MAIN.

It also contains <category> tag which describes the name of the category which is going to be handled. Here it is 'android.intent.category.LAUNCHER'

9.1 Design pattern flow:

The design strategy and programming aspect involved for developing the accelerometer based features to extend the Webkit browser interface is described below. Every design feature namely: shake, image rotation, zomm in/out, scroll etc has two main files. One of these is a main 'Test.java' file from which the activity is launched and the other is an 'AccelerometerReader.java' file which handles the sensors on the Android platform.

Given below is the design pattern flow of the code functionality for these features. The flow diagram will be followed by the description of this design pattern.

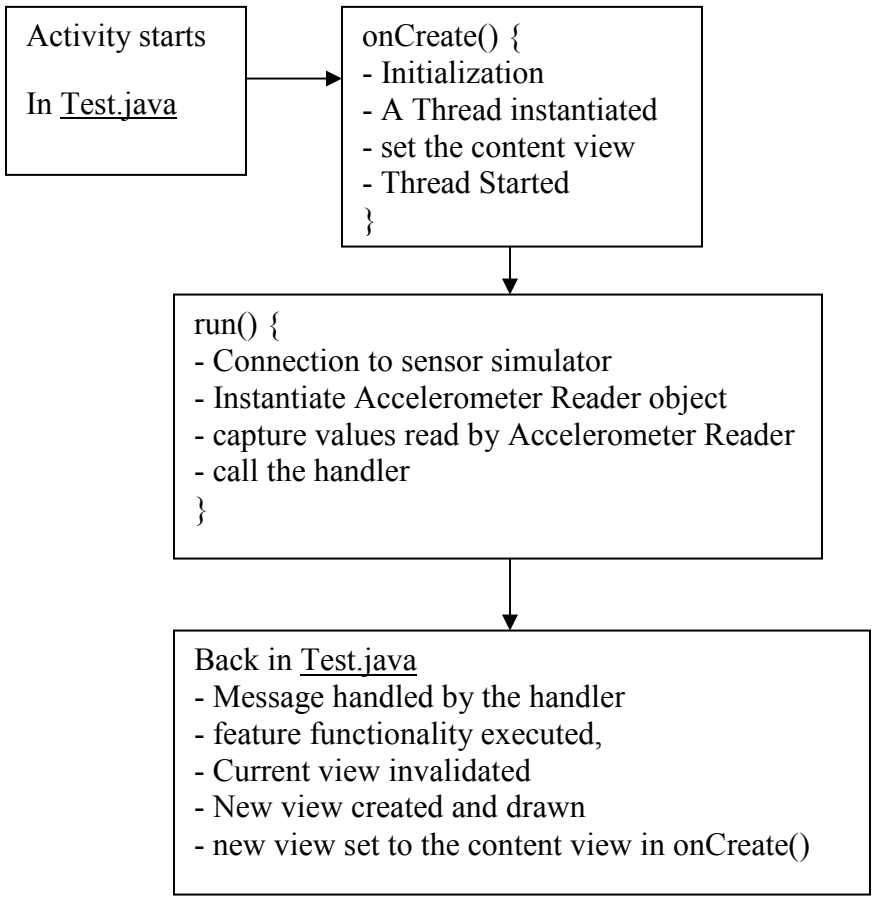

Fig 9.2 Flow diagram of main 'Test.java' class

9.1.1 Description of the flow diagram of Test.java class

Initially when the activity is launched, the onCreate method is called. All the initialization takes place. This includes referencing the UI elements from the XML layout files and referencing images from the resources folder. A new thread is instantiated which will handle the entire functionality of the feature. The content view of the Activity is set to the initial position. This could be the initial screen in case of zooming feature. For example the web page would be loaded and displayed here. In case of image rotation feature, the image would get displayed here. Finally, the thread is started.

Once the thread is started, it calls the 'run' method. Within the run method, the connection of Sensor Simulator to Android Emulator takes place in the background. An object of Accelerometer Reader class is instantiated. The functional details of the sensors are handled in this class and will be described in the following section. The sensor values i.e. accelerometer values in our case will be returned back to this thread by the Accelerometer Reader class. These values would be the continuous automated input values which would be playing in the background as explained in the section of automated input functionality. These values are captured in an on going loop. Each time a [x,y,z] value is received, a handler is invoked.

Handlers:

The mechanism of handlers is indispensable, since in Android only the main class i.e. the main UI thread which starts the activity has the control over the main content view of the phone. Handlers allow us to take control of the main content view from a non UI thread with the special functionality of handling calls between a UI and a non UI thread.

The call given to the handler is handled back in the main UI thread. Here the functionality pertaining to the respective feature is executed and the current content view is invalidated. The new view is draw based on the functionality of the feature and it is passed back to the content view in onCreate method. Thus, the new position on the screen is repainted.

The entire procedure repeats in a loop till the Sensor Simulator sends values to the Android Emulator. This explains the basic design flow for the features like shake, image orientation, zoom in/out, scroll etc.

Given below is design pattern flow of the code functionality of sensors used in developing the features.

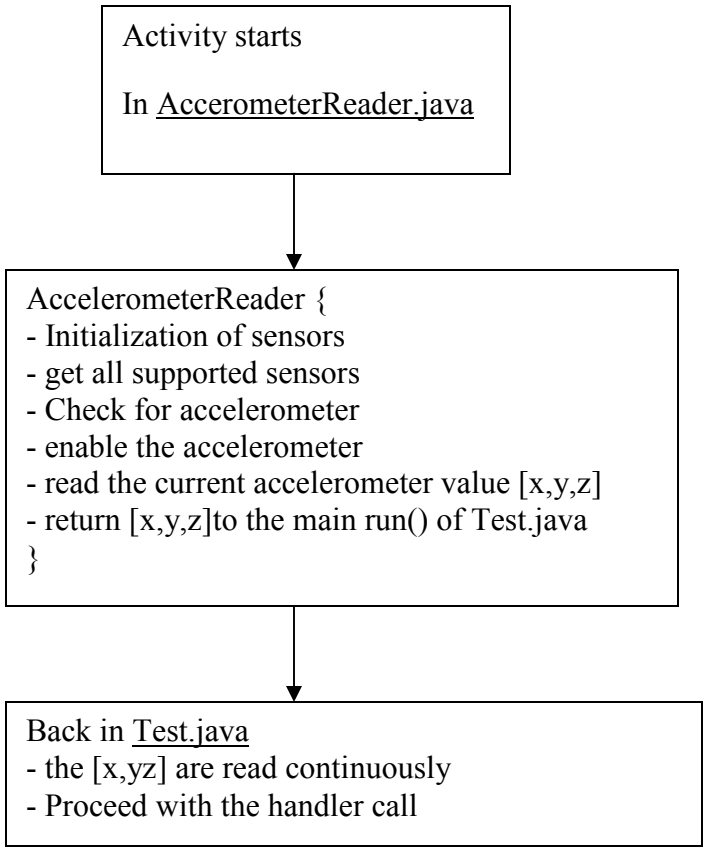

Fig 9.3 Flow diagram of 'AccelerometerReader.java' class

9.1.2 Description of the flow diagram of AccelerometerReader.java class

The ‘run’ method of Test.java class instantiates an ‘AccelerometerReader’ object. This calls the constructor of Accelerometer reader class. Here, the sensor related variables are initialized. This is followed by a call to get all the supported sensors on the Android platform. Specifically, a check is made for accelerometer. Once the accelerometer is detected from all the supported sensors, it is enabled. The accelerometer would then start recording the [x,y,z] values sent by the sensor simulator. These values are read by the Accelerometer reader class and are finally bundled together and returned to the calling method in the main Test.java class.

These values once returned to the main class are followed by the call to the handler and the flow proceeds as explained in the previous section.

This was the main design strategy used to develop accelerometer based motion features by extending the Webkit browser interface of Android platform. Once this design strategy was formulated, then the process of creating the features was started. The following sections will describe each of the accelerometer based features developed in the course of this project.

10. Shake feature.

After developing the design flow strategy, the next step was to get started with for developing accelerometer based features. The first feature was ‘shake’ detection.

The objective here was to implement the Shake feature based on movement of accelerometer. The advantage of this feature is that it eliminates the need for dedicated function keys like ‘clear’ and ‘back’ on the phone in this context. The test scenario here consists of the user entering some text as input (Fig 10.1) and on shaking the Sensor Simulator the Text Box entry would be cleared. (Fig 10.2).

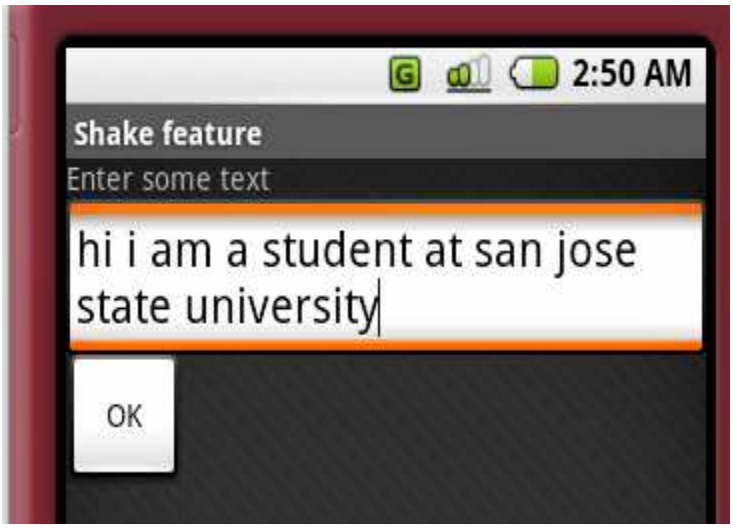

Fig 10.1: The UI screen with Text Box and connection button.

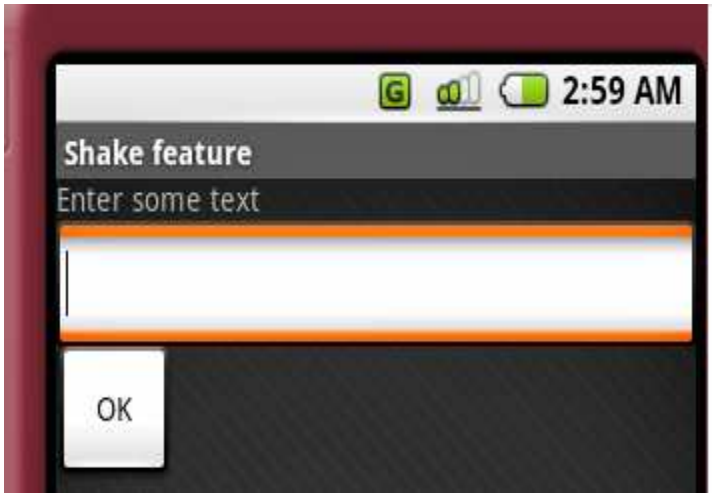

Fig 10.2: The cleared Text Box due to shake motion of Sensor Simulator

Description of the event: The basic principle of shake feature is that if a user wants to delete something that he previously entered then he can do so by shaking the phone. The implementation of this feature consisted of a Textbox where user can enter any data. The user makes a connection to the Sensor Simulator jar application. This is achieved by clicking the 'OK' button on the UI screen. When the user shakes the graphical image of the phone in the sensor simulator, (Fig 10.3) the data in the Textbox is cleared. When the phone is shaken the accelerometer values generated are tested against a threshold value. If this generated value is greater than the threshold then it is detected as a 'shake' motion. The values generated are written to a log file to help in debugging the code.

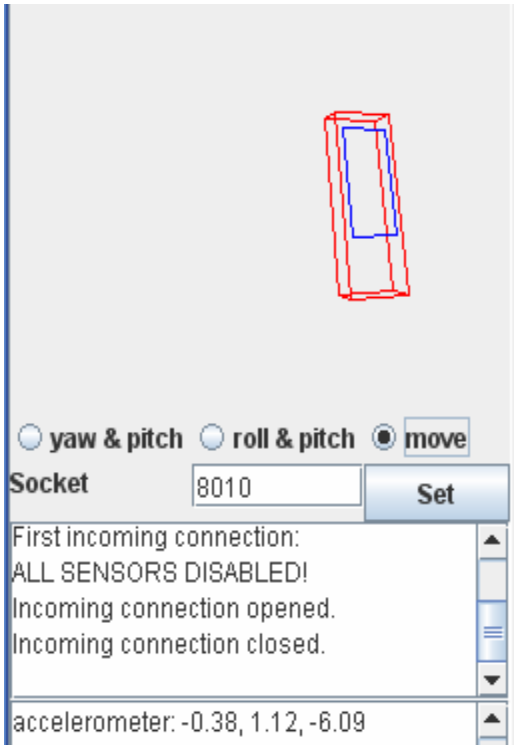

Fig 10.3: Shaking of Sensor Simulator phone

Description of code and files: The entire coding of this deliverable had been split up into three files.

Test.java: This is the main class from which the activity is launched. It connects to the Sensor Simulator jar application with the help of 'OK' UI button. The functionality for detecting the shake motion and capturing the [x,y,z] values during shaking of the phone is coded in this file. Every time the simulator sends a value to the emulator, the value is written to the log file. When the value exceeds the threshold, the shake motion is detected and it gets written to the log file in the form of an information message. Given below is the code snippet for writing to the log file.

```
android.util.Log.i("SHAKE","shake detected");
```

AccelerometerReader.java: This class gives access to the sensors supported by the Android platform. It detects the available sensors and enabled the accelerometer. This class is called from the main 'Test.java' class from a thread. The accelerometer values corresponding to [x,y,z] are generated in this class, bundled and the result is passed to the Test.java class where they are used to detect the shake motion. The values generated by shaking the phone have been written to the Log file as show in Fig 10.3.

main.xml: It is the UI file which shows the Text box and Sensor Simulator connection button. The code snippet for the two UI elements namely: text box and button is show below.

```
<EditText android:id="@+id/edittxt"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
```

```
android:paddingLeft="5px"
android:paddingRight="5px"
android:singleLine="false" />

<Button android:id="@+id/button"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="OK" />
```

As seen in he above listing, each element has an 'id' associated with it. This id is used to reference the element in the main UI file. Also we can set various display options and other parameters like the width, height, and font color etc. All these UI elements are placed in a Linear layout, which servers as a parent element.

Given below is a snapshot of the log file where the message of 'Shake detected' can be seen.

```
Hardware        Read line...
Hardware        Received: SensorSimulator
Hardware        Connected
THREE           connected to simulator
THREE           start initialization
FOUR            in constructor
Hardware        getSupportedSensors()
Hardware        Received: 3
Hardware        Received: accelerometer
Hardware        Received: compass
Hardware        Received: orientation
FIVE            enable sensor
SIX             in setEnableAccelerometer
Hardware2       enableSensor()
Hardware2       Send: accelerometer
Hardware2       Received: false
SEVEN           end setEnableAccelerometer
EIGHT           end constructor
NINE            completed initialization
FIVE            in readAccelerometer
FIVE            in readAccelerometer
Answer          -2.236241579055786
Answer          -1.5053974390029907
Answer          -1.8618059158325195
LEN is          11247038
SHAKE           shake detected
FIVE            in readAccelerometer
FIVE            in readAccelerometer
Answer          -0.2529144585132599
Answer          -0.8470873832702637
Answer          -0.6042823195457458
LEN is          1200177
```

Fig 10.4: The values being written to log file

11. Rotation feature.

The idea for rotation feature was based on the functionality of rotation supported on the Apple ‘iPhone’. The objective here was to implement the Rotation feature based on automated accelerometer inputs. The basic principle of rotation feature is that if the orientation of the phone is changed by a certain degrees, then the orientation of the application running on the phone must also get oriented and adjust itself accordingly.

Fig 11.1: The initial original position of image

Fig 11.2: The image rotated by 90 degrees clockwise

It was initially decided to have the rotation functionality work with keypad events. Here, every time the user clicked the ‘UP’ navigation key, the image would get rotated clockwise by 90 degrees. Similarly, on clicking the ‘DOWN’ navigation key, the image would rotate by 90 degrees in the anti-clockwise direction. Once this functionality was achieved, then it was migrated to work with accelerometer automated inputs.

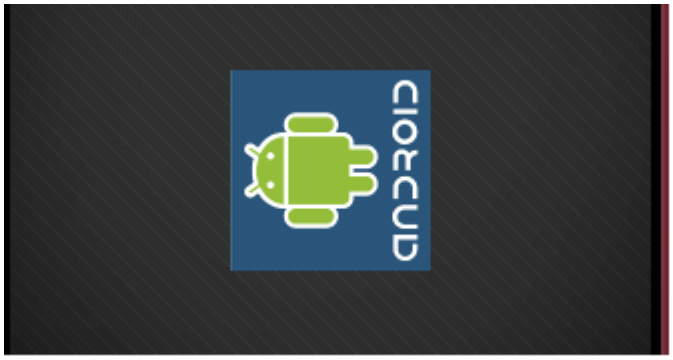

Fig 11.3: The image rotated by 90 degrees anti-clockwise

Description of the event: This deliverable has been simulated to work with images. Suppose the user is viewing images on his phone. If he rotates the device by certain degrees, then the images must also get rotated by that degrees and adjust the display accordingly. This feature has to respond to both clockwise and anti-clockwise rotations.

For demonstration purpose, this deliverable will rotate the images by 90 degrees in both clockwise and anti-clockwise directions. The implementation of this feature consisted of a loading an image as soon as the application is launched (Fig 11.1). A separate thread is instantiated which makes a connection to the Sensor Simulator jar application in the background. After an initial interval of few seconds, the Sensor simulator starts sending accelerometer inputs to the emulator. On each input the image must get rotated by 90 degrees in clockwise direction (Fig 11.2) and anti-clockwise (Fig 11.3) directions. There is a delay of few seconds as the simulator is assigns an IP address and a socket connection is made to the emulator. Each accelerometer input value and rotation is written to a log file to help in debugging the code.

Description of code and files: The entire coding of this deliverable had been split up into two files.

Test.java: This is the main class which connects to the Sensor Simulator jar application. When the application is launched, a Bitmap object is created which is used to create and load the image from a source. The connection is made to the Simulator in the background and a new thread is instantiated. This new thread is responsible for reading the accelerometer values from the AccelerometerReader.java file and passing it to the Test.java file. As soon as a new accelerometer value is fetched, a call is given to the 'make' function of the 'MakeImage' class. MakeImage is a subclass of Test class and is responsible for calculating the new position of image on the display screen.

A mechanism of Handler is used let the main UI thread talk to other activity classes. The handler sends a message to the 'handlemessage()' method which in turn calls the invalidate() method. This invalidate() method causes the current view on the display screen to be invalidated and gives a call to the onDraw() function. This onDraw is finally responsible to display the new rotated image on the screen.

Given below is the snippet of the onDraw() function which display the newly calculated position of the image on the display screen.

```
protected void onDraw(Canvas canvas) {
super.onDraw(canvas);

android.util.Log.i("INSIDE ON DRAW", "inside on draw");

bmd.setBounds((getWidth()/2)-scaling_factor, (getHeight()/2)-scaling_factor,
(getWidth()/2)+scaling_factor, (getHeight()/2)+scaling_factor);

int w = getWidth();
android.util.Log.i("WIDTH "+w," currently");
int h = getHeight();
```

```
android.util.Log.i("HEIGHT "+h," currently");

bmd.draw(canvas);
}
```

This 'onDraw' function is called when the handler invokes the invalidate method. The invalidate method further calls the onDraw and this new position of the image is calculated and set to the content view.

AccelerometerReader.java: This class is responsible for handling the sensor functionality. The values of the [x,y,z] are read and passed on to the Test.java class where they are used for rotating the image by giving a call to the make() function. The [x,y,z] values during each rotation are written to the Log file for debugging purposes.

Given below is a snapshot of the log file where the message of 'Shake detected' can be seen.

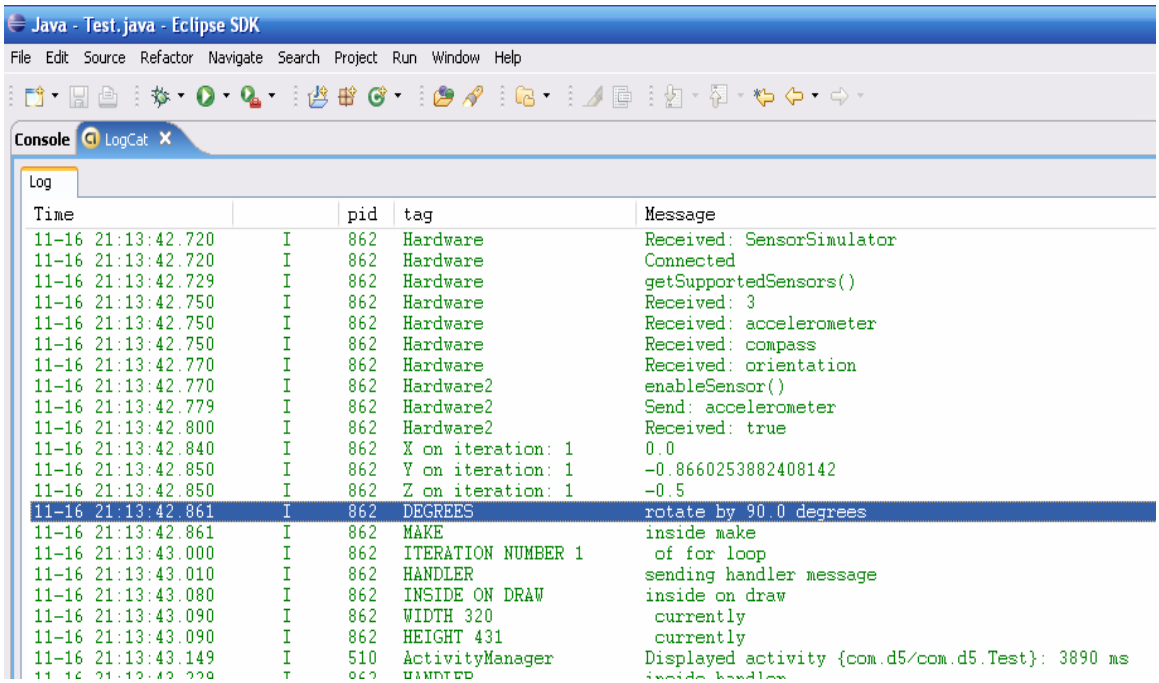

Fig 11.4: The [x,y,z] values being written to log file.

12. Zoom feature.

When a web page is loaded in a browser on a cell phone, it usually occupies the entire screen area. The underlying fonts are relatively small making it difficult to read easily. Hence, some form of zoom functionality is required. The goal of this feature was to extend the Webkit browser interface to support the zoom feature based on tilt detection of the phone. In the simulated scenario, when the device is tilted in forward direction, it will cause the web page to zoom in; whereas if the phone is tilted in the backward direction, the page would zoom out.

Fig 12.1: The initial web page when it is loaded in WebView.

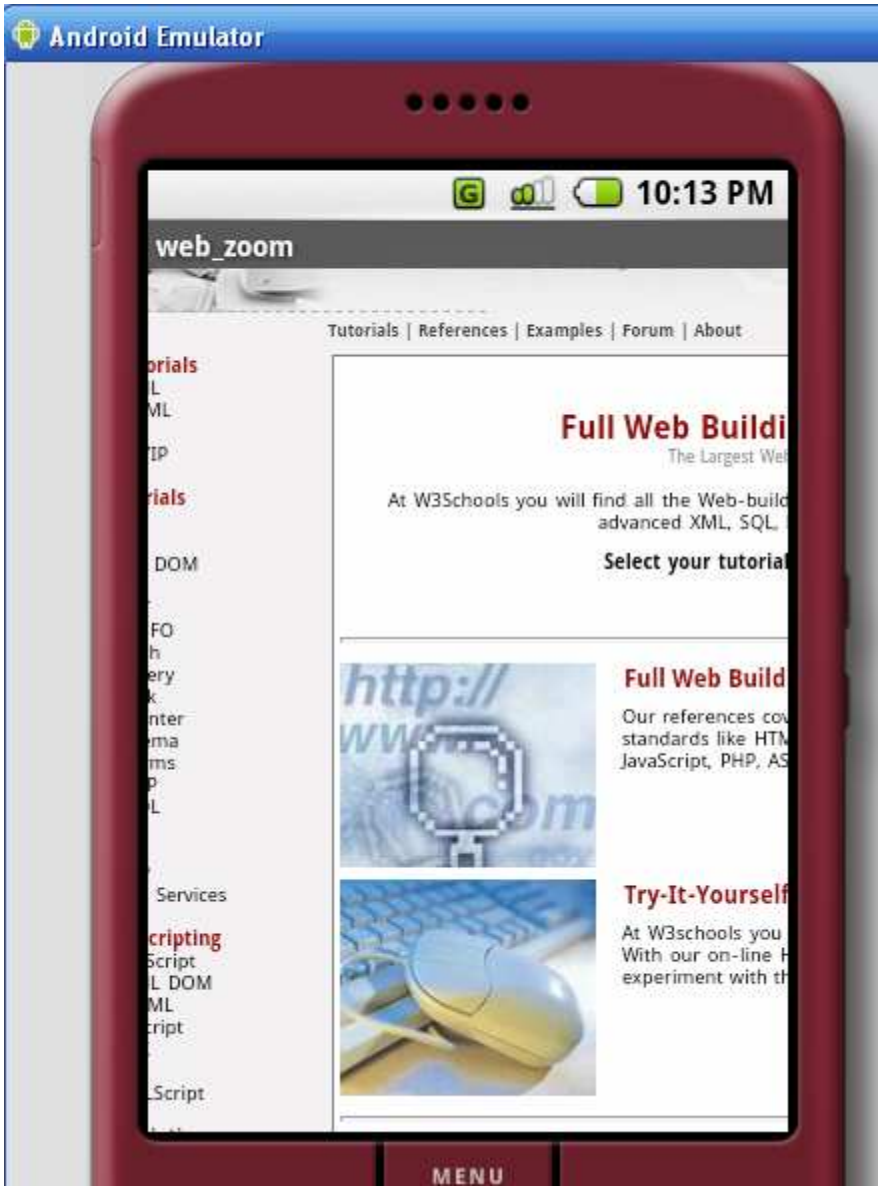

Fig 12.2: The web page zoomed in

Description of the event: The figure 11.2 shows the initial position of the web page when it is loaded. This feature does not use the existing browser which is built in Android. Instead, we have used the Webkit API's to instantiate our own WebView. This WebView can be thought to be a rudimentary mini browser created specially to test the Webkit API's and extend them to support zoom feature. Once the simulator connection is made and accelerometer inputs are detected, the web page starts zooming in. This can be

seen in the figure 11.2. This corresponds to the tilt of phone in forward direction. For the later half of the automated inputs the web page zooms out which corresponds to tilt of phone in forward direction. This is shown in the figure 11.3.

Fig 12.3: The web page zoomed out

Description of code and files: The entire coding of this deliverable had been split up into two files.

Web.java: This class starts the main activity. Here, we use the Webkit Api's to instantiate our own WebView and load a 'url' into the WebView. This is the main class which connects to the Sensor Simulator jar application. When the application is launched, a new WebView object is instantiated. A website url is passed to this object and it gets loaded into the webView. This initial position of the web page is set to the content view and it displayed on the screen. In the background, a connection is made to the Simulator and a new thread is instantiated. This new thread is responsible for reading the accelerometer values from the AccelerometerReader.java file and passing it to the Web.java file. As soon as a new accelerometer value is read, a call is given to the getZoomWidth() function which calculates the current zoom width of the entire WebView. Then a call is made to setZoomWidth() function which sets the new zoom width for the webView.

This functionality also makes use of handlers to pass the messages between the main UI and the non UI thread. The handler invalidates the current view on receiving the new accelerometer values and the onDraw() method calculates the new position of the zoomed page. This is set to the main content view and the new zoomed position is displayed.

AccelerometerReader.java: This class is called from web.java and it basically enables the sensors on the phone. The values of the [x,y,z] are read and passed on to the Web.java class. The [x,y,z] values during each zoom are written to the Log file for debugging purposes.

This is the basic functionality of the zoom feature. Also, the WebView which has been instantiated works successfully like a normal browser. The user can enter a url address, view different pages, navigate in different directions, etc.

13. Scroll feature.

While browsing a web page, a user normally has to use the ‘up’, ‘down’, ‘left’ and ‘right’ navigation keys in order to scroll the page. Also on touch screen phone, one is required to swipe the finger in the specific direction to scroll. The objective here was to implement the Scrolling feature based on accelerometer inputs. This feature has been integrated with the Webkit browser interface supported by the Android platform.

Fig 13.1: The web page being scrolled down.

Fig 13.2: The web page being scrolled towards the right.

Description of the event: As soon as a WebView is instantiated, the web page gets loaded in the WebView which is displayed to the user. Once, the automated accelerometer inputs are sent to the emulator, the web page starts scrolling initially in the downward direction, followed by scrolling towards the right. This would correspond to tilting the phone is downward direction followed by tilting towards right. Also, the scrolling is carried out for a scenario in which the user could tilt the phone in the

downward as well as right direction at a certain angle and upwards to the left direction. This corresponds to the case of tilting the phone down and towards right edge of screen and the last case where the device would be tilted towards top left edge of the screen.

Fig 13.3: The web page being scrolled towards top left side.

The above figure shows the web page being tilted towards the top left direction at a certain angle while the following figure shows the web page being tilted towards bottom right direction.

Fig 13.4: The web page being scrolled towards bottom right side.

Description of code and files: This feature is very much similar to the zoom feature in terms of the flow of code and execution of events. The entire coding of this deliverable had been split up into two files.

Web.java: This is the main class which instantiates a WebView and connects to the Sensor Simulator jar application. The coding functionality is similar to zoom feature.

Based on the tilt of the phone, the new scroll position is calculated by the handler. This new position is sent to the onDraw function which takes care of drawing the new scrolled view. Thus, the webView get updated every time the page is scrolled and it is set to the main content view of the display screen.

AccelerometerReader.java: This class is called from web.java and it basically enables the sensors on the phone. The underlying functionality of this class is similar to the zoom feature.

Thus, the four different features namely shake detection, image orientation, zoom in/out, and scroll have been designed and developed to enhance the Webkit browser interface. The design methodology and flow pattern served as the foundation for developing these features in a systematic manner.

All these four features were the software deliverables required for the completion of this project. Also, an independent application for testing the media capabilities of this platform was developed. This was an additional application which has been explained in the following section.

14. Testing media playback capabilities of Android.

Since Android was a new platform, we decided to test the media capabilities on this platform by implementing a Music player. This application was not targeted towards developing a full - fledged Media player, but aimed at testing the audio playback due to availability of additional time. This application is note related to accelerometer inputs.

Fig 14.1: The controls on the music application.

Description of the event: This feature tests the media capabilities of the Android platform. On the UI screen a user has four dedicated buttons namely 'Play', 'Pause', 'Stop' and 'Restart'. On clicking ‘Play’ the .mp3 file stores in the resources directory starts playing. Each of the independent buttons when clicked should respond accordingly in reference to the .mp3 file.

Description of code and files: The entire coding of this deliverable had been completed in a single java file.

Mediaplayer.java: This class makes use of the MediaPlayer API's of Android which help in playing audio and video files. The four buttons namely 'Play', 'Pause', 'Stop' and 'Restart' perform the event handling and respond accordingly.

15. Challenges involved in the project

This Android platform was launched in November 2007. We began working with the platform since January 2008. There were many obvious challenges and concerns associated with the project which are expected while working on any new platform

1. Initially, the main challenge was the learning curve along with the constraints of limited resources and reference material.

2. Throughout the course of this project I also had to overcome compatibility issues which aroused with every new release of the platform. The code was not completely backward compatible with the earlier version and hence required some modifications over the course of the project. Achieving proper compatibility between the Android platform, Eclipse framework and java development kit posed challenges during the development. Finally I stuck to the version m5_rc14 of the SDK.

3. Also, the OpenIntents sensor simulator required me to manually change the entries to simulate accelerometer movements. Hence, the solution was to automate the input functionality thereby allowing a continuous set of inputs which could be used for testing.

4. In order to enhance the sensor simulator with automated inputs, I had to dig through the source code of the OpenIntents project and make sure my modifications do not alter the actual working functionality of the simulator. This was time consuming process.

5. While designing the features like zoom in/out, image rotation, and scrolling, a mechanism of handlers was required in order to let a non UI thread talk to the main UI thread. This was very important mechanism without which helped in resolving a lot of debugging issues.

16. Conclusion

This project has developed and implemented accelerometer based features for the Android platform like scroll, image orientation, zoom in/out, scroll and has successfully extended the Webkit browser interface to support these features. These features portray a new and innovative way to exploit the capabilities of sensors on the Android platform. The advantage of these motion features is that they have helped in reducing the dependence on dedicated navigation keys on smart phones. This actually helps as the size of the display screen does not have to be compensated with adding specific function keys.

The features and experimental coding applications done during the last two semesters have greatly increased my understanding of the working of Android platform. There is no limit to the extent of applications which can be developed using Android platform. This platform might be very successful in future due to the fact that it is Open source and freely available. The Software development kit exactly simulates the Android system and hence the developers have a full environment to test and simulate Android applications.

Future Work:

In addition to accelerometers, the Android platform also supports sensors like Compass and Orientation. Hence, my future work would involve exploiting the capabilities of these sensors in my project. Also, I would like to port all these features on an actual Android device. This platform has already gained popularity with many of cell phone companies in the market. Given its flexibility and ease of use, there is a strong possibility that in future we could have a variety of embedded systems and electronic equipments running Android platform.

17. References

[1] [2007] Dennis Majoe, SQUEAK: A Mobile Multi Platform Phone and Networks Gesture Sensor, Proceedings of the 2007 IEEE 2nd International Conference on Pervasive Computing and Applications.

[2] [2007] Wook Bahn, A 16-bit Ultra-Thin Tri-axes Capacitive Micro accelerometer for Mobile Application, International Conference on Control, Automation and Systems.

[3] [2005] Eun-Seok Choi, Beatbox Music Phone: Gesture-based Interactive Mobile Phone using a Tri-axis Accelerometer, Industrial Technology, IEEE International Conference.

[4] [2005] Ian Okley. Tilt to Scroll: Evaluating a Motion Based Vibrotactile Mobile Interface, Proceedings of the First Joint Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems.

[5] [2003] I.J.Jang, Signal Processing of the Accelerometer for Gesture Awareness on Handheld Devices, Proceedings of the 2003 IEEE lntenational Workshop on Robot and Human Interactive Communication.

[6] Google's Android project.http://code.google.com/android/index.html

[7] Open Intents project. http://code.google.com/p/openintents/

[8] Android developer forums. http://www.anddev.org/

[9] Android Discussion Groups. http://code.google.com/android/groups.html